National Semiconductor

400071

Microprocessor Databook

- *Series 32000*
- *NSC800 Family*

A Corporate Dedication to Quality and Reliability

National Semiconductor is an industry leader in the manufacture of high quality, high reliability integrated circuits. We have been the leading proponent of driving down IC defects and extending product lifetimes. From raw material through product design, manufacturing and shipping, our quality and reliability is second to none.

We are proud of our success . . . it sets a standard for others to achieve. Yet, our quest for perfection is ongoing so that you, our customer, can continue to rely on National Semiconductor Corporation to produce high quality products for your design systems.

Charles E. Sporck
President, Chief Executive Officer
National Semiconductor Corporation

Wir fühlen uns zu Qualität und Zuverlässigkeit verpflichtet

National Semiconductor Corporation ist führend bei der Herstellung von integrierten Schaltungen hoher Qualität und hoher Zuverlässigkeit. National Semiconductor war schon immer Vorreiter, wenn es galt, die Zahl von IC Ausfällen zu verringern und die Lebensdauern von Produkten zu verbessern. Vom Rohmaterial über Entwurf und Herstellung bis zur Auslieferung, die Qualität und die Zuverlässigkeit der Produkte von National Semiconductor sind unübertroffen.

Wir sind stolz auf unseren Erfolg, der Standards setzt, die für andere erstrebenswert sind. Auch ihre Ansprüche steigen ständig. Sie als unser Kunde können sich auch weiterhin auf National Semiconductor verlassen.

La Qualité et La Fiabilité:

Une Vocation Commune Chez National Semiconductor Corporation

National Semiconductor Corporation est un des leaders industriels qui fabrique des circuits intégrés d'une très grande qualité et d'une fiabilité exceptionelle. National a été le premier à vouloir faire chuter le nombre de circuits intégrés défectueux et a augmenter la durée de vie des produits. Depuis les matières premières, en passant par la conception du produit sa fabrication et son expédition, partout la qualité et la fiabilité chez National sont sans équivalents.

Nous sommes fiers de notre succès et le standard ainsi défini devrait devenir l'objectif à atteindre par les autres sociétés. Et nous continuons à vouloir faire progresser notre recherche de la perfection; il en résulte que vous, qui êtes notre client, pouvez toujours faire confiance à National Semiconductor Corporation, en produisant des systèmes d'une très grande qualité standard.

Un Impegno Societario di Qualità e Affidabilità

National Semiconductor Corporation è un'industria al vertice nella costruzione di circuiti integrati di altà qualità ed affidabilità. National è stata il principale promotore per l'abbattimento della difettosità dei circuiti integrati e per l'allungamento della vita dei prodotti. Dal materiale grezzo attraverso tutte le fasi di progettazione, costruzione e spedizione, la qualità e affidabilità National non è seconda a nessuno.

Noi siamo orgogliosi del nostro successo che fissa per gli altri un traguardo da raggiungere. Il nostro desiderio di perfezione è d'altra parte illimitato e pertanto tu, nostro cliente, puoi continuare ad affidarti a National Semiconductor Corporation per la produzione dei tuoi sistemi con elevati livelli di qualità.

Charles E. Sporck
President, Chief Executive Officer
National Semiconductor Corporation

MICROPROCESSOR
DATABOOK

- Series 32000
- NSC800

1989 Edition

TRADEMARKS

Following is the most current list of National Semiconductor Corporation's trademarks and registered trademarks.

AbuseableTM
AnadigTM
ANS-R-TRANTM
APPSTM
ASPECTTM
Auto-Chem DeflasherTM
BCPTM
BI-FETTM
BI-FET IITM
BI-LINETM
BIPLANTM
BLCTM
BLXTM
Brite-LiteTM
BTLTM
CheckTrackTM
CIMTM
CIMBUSTM
CLASICTM
Clock✓ChekTM
COMBOTM
COMBO ITM
COMBO IITM
COPSTM microcontrollers
Datachecker®
DENSPAKTM
DIBTM
Digitalker®
DISCERNTM
DISTILLTM
DNR®
DPVMTM
ELSTARTM
Embedded System Processor™
E-Z-LINKTM
FACTTM
FAIRCADTM
FairtechTM
FAST®
5-Star ServiceTM
GENIXTM
GNXTM
HAMRTM
HandiScanTM
HEX 3000TM
HPCTM
I3L®
ICMTM
INFOCHEXTM
Integral ISETM
IntelisplayTM
ISETM
ISE/06TM
ISE/08TM
ISE/16TM
ISE32TM
ISOPLANARTM
ISOPLANAR-ZTM
KeyScanTM
LMCMOSTM
M2CMOSTM
MacrobusTM
MacrocomponentTM
MAXI-ROM®
Meat✓ChekTM
MenuMasterTM
Microbus™ data bus
MICRO-DACTM
µtalkerTM
MicrotalkerTM
MICROWIRETM
MICROWIRE/PLUSTM
MOLETM
MSTTM
Naked-8TM
National®
National Semiconductor®
National Semiconductor Corp.®
NAX 800TM
Nitride PlusTM
Nitride Plus OxideTM
NMLTM
NOBUSTM
NSC800TM
NSCISETM
NSX-16TM
NS-XC-16TM
NTERCOMTM
NURAMTM
OXISSTM
P2CMOSTM
PC MasterTM
Perfect WatchTM
Pharma✓ChekTM
PLANTM
PLANARTM
Plus-2TM
PolycraftTM
POSilinkTM
POSitalkerTM
Power + ControlTM
POWERplanarTM
QUAD3000TM
QUIKLOOKTM
RATTM
RTX16TM
SABRTM
Script✓ChekTM
SCXTM
SERIES/800TM
Series 900TM
Series 3000TM
Series 32000®
Shelf✓ChekTM
SofChekTM
SPIRETM
Staggered RefreshTM
STARTM
StarlinkTM
STARPLEXTM
Super-BlockTM
SuperChipTM
SuperScriptTM
SYS32TM
TapePak®
TDSTM
TeleGateTM
The National Anthem®
Time✓ChekTM
TINATM
TLCTM
TrapezoidalTM
TRI-CODETM
TRI-POLYTM
TRI-SAFETM
TRI-STATE®
TURBOTRANSCEIVERTM
VIPTM
VR32TM
WATCHDOGTM
XMOSTM
XPUTM
Z STARTM
883B/RETSTM
883S/RETSTM

IBM®, PC®, and AT® are registered trademarks of International Business Machines, Inc.
MULTIBUS® is a registered trademark of Intel Corporation.
Sun-3® Workstation is a registered trademark of Sun Microsystems, Inc.
UNIX® and DWB® are registered trademarks of AT&T.
Z80® is a registered trademark of Zilog Corporation.
CCS-Page™ is a trademark of Control-C Software, Inc.
CP/M™ is a trademark of Digital Research Corporation.
Documenter's Workbench™ is a trademark of AT&T.
Model 19™ is a trademark of DATA I/O Corporation.
Opus5™ is a trademark of Opus Systems.
PAL® and PALASM™ are trademarks of and are used under license from Monolithic Memories, Inc.
SunOS™ is a trademark of Sun Microsystems.
VAX™, VMS™, DEC™, PDP-11™, RSX-11™ and ULTRIX™ are trademarks of Digital Equipment Corporation.
VisiCalc™ is a trademark of Visi Corporation.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation 2900 Semiconductor Drive, P.O. Box 58090, Santa Clara, California 95052-8090 (408) 721-5000
TWX (910) 339-9240

National Semiconductor

Product Status Definitions

Definition of Terms

Data Sheet Identification	Product Status	Definition
Advance Information	Formative or In Design	This data sheet contains the design specifications for product development. Specifications may change in any manner without notice.
Preliminary	First Production	This data sheet contains preliminary data, and supplementary data will be published at a later date. National Semiconductor Corporation reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
No Identification Noted	Full Production	This data sheet contains final specifications. National Semiconductor Corporation reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

National Semiconductor Corporation reserves the right to make changes without further notice to any products herein to improve reliability, function or design. National does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

Table of Contents

Table of Contents (Continued)

Alpha-Numeric Index

Alpha-Numeric Index (Continued)

Section 1

Series 32000 Overview

Section 1 Contents

Introduction

Series 32000 offers the most complete solution to your 32-bit microprocessor needs via CPUs, slave processors, system peripherals, evaluation/development tools and software.

We at National Semiconductor firmly believe that it takes a total family of microprocessors to effectively meet the needs of a system designer.

This Series 32000 Databook presents technical descriptions of Series 32000 8-, 16- and 32-bit microprocessors, slave processors, peripherals, software and development tools. It is designed to be updated frequently so that our customers can have the latest technical information on the Series 32000.

Series 32000 leads the way in state-of-the-art microprocessor designs because of its advanced architecture, which includes:

- 32-Bit Architecture
- Demand Paged Virtual Memory
- Fast Floating-Point Capability
- High-Level Language Support
- Symmetrical Architecture

When we at National Semiconductor began the design of the Series 32000 microprocessor family, we decided to take a radical departure from popular trends in architectural design that dated back more than a decade. We chose to take the time to design it properly.

Working from the top down, we analyzed the issues and anticipated the computing needs of the 80's and 90's. The result is an advanced and efficient family of microprocessor hardware and software products.

Clearly, software productivity has become a major issue in computer-related product development. In microprocessor-based systems this issue centers around the capability of the microprocessor to maximize the utility of software relative to shorter development cycles, improved software reliability and extended software life cycles.

In short, the degree to which the microprocessor can maximize software utility directly affects the cost of a product, its reliability, and time to market. It also affects future software modification for product enhancement or rapid advances in hardware technology.

Our approach has been to define an architecture addressing these software issues most effectively. Series 32000 combines 32-bit performance with efficient management of large address space. It facilitates high-level language program development and efficient instruction execution. Floating-point is integrated into the architecture.

This combination gives the user large system computing power at two orders of magnitude less cost.

But we didn't stop there. Advanced architecture isn't enough. Our top-down approach includes the hardware, software, and development support products necessary for your design. The evaluation board, in-system emulator, software development tools, including a VAX-11 cross-software package, and third party software are also available now for your evaluation and development.

The Series 32000 is a solid foundation from which National Semiconductor can build solutions for your future designs while satisfying your needs today.

For further information please contact your local sales office.

National Semiconductor

Key Features of Series 32000®

Some of the features that set the Series 32000 family apart as the best choice for 32-bit designs are as follows:

FAMILY OF MICROPROCESSORS

Series 32000 is more than just a single chip set, it is a family of chip sets. By mixing and matching Series 32000 CPUs with compatible slave processors and support chips, a system designer has an unprecedented degree of flexibility in matching price/performance to the end product.

CLEANEST 32-BIT ARCHITECTURE

Series 32000 was designed around a 32-bit architecture from the beginning. It has a fully symmetrical instruction set so that all addressing modes and all data types can be operated on by all instructions. This makes it easy to learn the architecture, easy to program in assembly language, and easy to write code-efficient, high-level language compilers.

APPLICATION-SPECIFIC SLAVE PROCESSORS

Series 32000 architecture allows users to design their own application-specific slave processors to interface with the existing chip set. These processors can be used to increase the overall system performance by accelerating customized CPU instructions that would otherwise be implemented in software. At the same time, software compatibility is maintained, i.e., it is always possible to substitute lower-cost software modules in place of the slave processor.

FLOATING-POINT SUPPORT

The Series 32000 offers a complete set of floating-point solutions. This includes the NS32081 Floating-Point Unit, the NS32381 Floating-Point Unit and the NS32580 Floating-Point Controller. The NS32081 provides high-speed arithmetic computation with high precision and accuracy at low cost. The NS32381 provides low power consumption and even greater performance than the NS32081 while maintaining high-precision and accuracy.

The NS32580 is a floating-point controller that provides a direct interface between the Weitek WTL 3164 Floating-Point Data Path and the NS32532 CPU. This two chip combination, NS32580/WTL3164, provides optimum performance for speed critical floating-point applications.

HIGH-LEVEL LANGUAGE SUPPORT

Series 32000 has special features that support high-level languages, thus improving software productivity and reducing development costs. For example, there are special instructions that help the compiler deal with structured data types such as Arrays, Strings, Records, and Stacks.

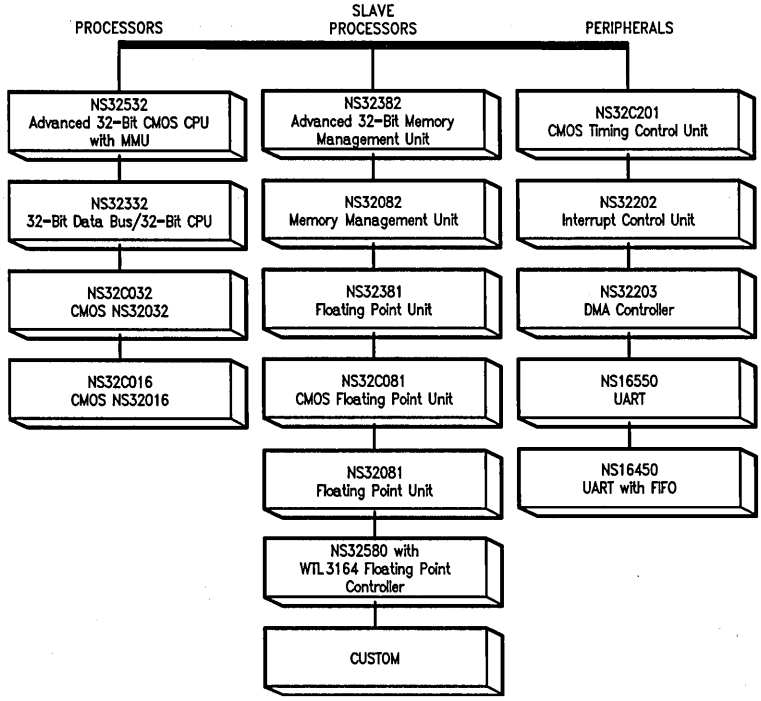
National Semiconductor

Series 32000® Component Descriptions

Device	Description	Bus Width			Process	Package Type
		Internal	External			
			Address	Data		
CENTRAL PROCESSING UNITS (CPU's)						
NS32532	High-Performance 32-Bit Microprocessor	32	32	32	M^2CMOS	175-pin PGA
NS32332	32-Bit Advanced Microprocessor	32	32	32	XMOS (NMOS)	84-pin PGA
NS32C032	High-Performance Microprocessor	32	24	32	CMOS	68-pin LCC Leadless Chip Carrier
NS32C016	High-Performance Microprocessor	32	24	16	CMOS	48-pin DIP Dual-In-Line Package
SLAVE PROCESSORS						
NS32382	Memory Management Unit	32	32	32	XMOS (NMOS)	PGA
NS32082	Memory Management Unit	32	24	16	XMOS (NMOS)	48-pin DIP Package
NS32081	Floating-Point Unit	64	—	16	XMOS	24-pin DIP Dual-In-Line Package
NS32381	Floating-Point Unit	64	—	16	CMOS	68-pin PGA
NS32580	Floating-Point Controller	64	—	16 or 32	CMOS	172-pin PGA
PERIPHERALS						
NS32C201	CMOS Timing Control Unit	—	—	—	CMOS	24-pin DIP Dual-In-Line Package
NS32202	Interrupt Control Unit	32	—	16	XMOS (NMOS)	40-pin DIP Dual-In-Line Package
NS32203	Direct Memory Access Controller	—	—	16	XMOS (NMOS)	48-pin DIP Dual-In-Line Package

National Semiconductor

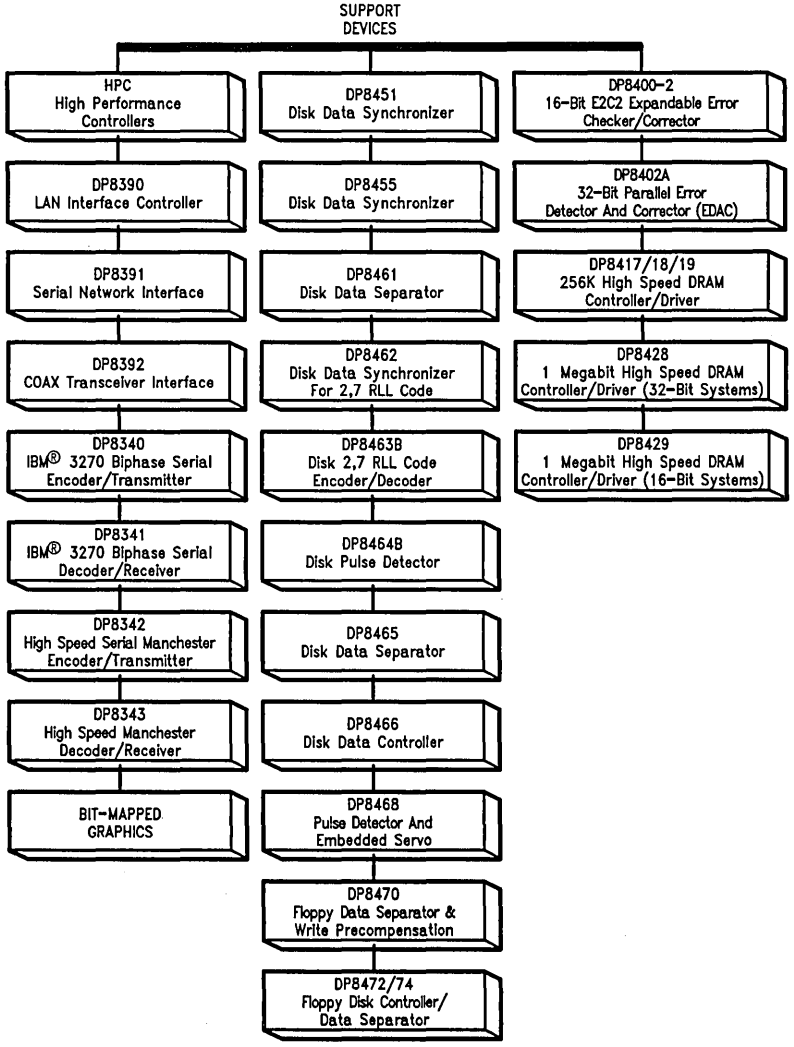
Hardware Chart



TL/XX/0084-1

National Semiconductor

Support Devices Chart



TL/XX/0166-1

1

National Semiconductor

Military/Aerospace Programs from National Semiconductor

This section is intended to provide a brief overview of military Series 32000 products available from National Semiconductor.

-883

Although originally intended to establish uniform test methods and procedures, MIL-STD-883 has also become the general specification for non-JAN military product. Revision C of this document defines minimum requirements for a device to be marked and advertised as 883-compliant. Included are design and construction criteria, documentation controls, electrical and mechanical screening requirements, and quality control procedures. Details can be found in paragraph 1.2.1 of MIL-STD-883.

National offers both 883 Class B and 883 Class S product. The screening requirements for both classes of product are outlined in Table I.

As with DESC specifications, a manufacturer is allowed to use his standard electrical tests provided that all critical parameters are tested. Also, the electrical test parameters, test conditions, test limits, and test temperatures must be clearly documented. At National Semiconductor, this information is available via our RETS (Reliability Electrical Test Specification) program. The RETS document is a complete description of the electrical tests performed and is controlled by our QA department. Individual copies are available upon request.

-MIL

Some of National's older products are not completely compliant with MIL-STD-883, but are still required for use in military systems. These devices are screened to the same stringent requirements as 883 product but are marked "-Mil".

-MSP

National's Military Screening Program (MSP) was developed to make screened versions of advanced products such as gate arrays and microprocessors available more quickly than is possible for JAN and 883 devices. Through this program, screened product is made available for prototypes and brassboards prior to or during the JAN or 883 qualification activities. MSP products receive the 100% screening of Table I, but are not subjected to group C and D quality conformance testing. Other criteria such as electrical testing and temperature range will vary depending upon individual device status and capability.

TABLE I. 100% Screening Requirements

Screen	Class S		Class B	
	Method	Reqmt	Method	Reqmt
1. Wafer Lot Acceptance	5007	All Lots		
2. Nondestructive Bond Pull	2023	100%		
3. Internal Visual (Note 1)	2010, Condition A	100%	2010, Condition B	100%
4. Stabilization Bake	1008, Condition C, Min, 24 Hrs. Min	100%	1008, Condition C, Min, 24 Hrs. Min	100%
5. Temp. Cycling (Note 2)	1010, Condition C	100%	1010, Condition C	100%
6. Constant Acceleration	2001, Condition E (Min) Y_1 Orientation Only	100%	2001, Condition E (Min) Y_1 Orientation Only	100%
7. Visual Inspection (Note 3)		100%		100%
8. Particle Impact Noise Detection (PIND)	2020, Condition A (Note 4)	100%		
9. Serialization	(Note 5)	100%		
10. Interim (Pre-Burn-In) Electrical Parameters	Per Applicable Device Specification (Note 13)	100%	Per Applicable Device Specification (Note 6)	
11. Burn-In Test	1015 240 Hrs. at 125°C Min (Cond. F Not Allowed)	100%	1015, 160 Hrs. at 125°C Min	100%

TABLE I. 100% Screening Requirements (Continued)

Screen	Class S		Class B	
	Method	Reqmt	Method	Reqmt
12. Interim (Post-Burn-In) Electrical Parameters	Per Applicable Device Specification (Note 13)	100%		
13. Reverse Bias Burn-In (Note 7)	1015; Test Condition A, C, 72 Hrs. at 150°C Min (Cond. F Not Allowed)	100%		
14. Interim (Post-Burn-In) Electrical Parameters	Per Applicable Device Specification (Note 13)	100%	Per Applicable Device Specification	100%
15. PDA Calculation	5% Parametric (Note 14) 3% Functional — 25°C	All Lots	5% Parametric (Note 14)	All Lots
16. Final Electrical Test a) Static Tests 1) 25°C (Subgroup 1, Table I, 5005) 2) Max & Min Rated Operating Temp (Subgroups 2, 3, Table I, 5005) b) Dynamic Tests & Switching Tests, 25°C (Subgroups 4, 9, Table I, 5005) c) Functional Test, 25°C (Subgroup 7, Table I, 5005)	Per Applicable Device Specification	 100% 100% 100% 100%	Per Applicable Device Specification	 100% 100% 100% 100%
17. Seal Fine, Gross	1014	100% (Note 8)	1014	100% (Note 9)
18. Radiographic (Note 10)	2012 Two Views	100%		
19. Qualification or Quality Conformance Inspection Test Sample Selection	(Note 11)	Samp.	(Note 11)	Samp.
20. External Visual (Note 12)	2009	100%		100%

Note 1: Unless otherwise specified, at the manufacturer's option, test samples for Group B, bond strength (Method 5005) may be randomly selected prior to or following internal visual (Method 5004), prior to sealing provided all other specification requirements are satisfied (e.g. bond strength requirements shall apply to each inspection lot, bond failures shall be counted even if the bond would have failed internal visual).

Note 2: For Class B devices, this test may be replaced with thermal shock method 1011, test condition A, minimum.

Note 3: At the manufacturer's option, visual inspection for castastrophic failures may be conducted after each of the thermal/mechanical screens, after the sequence or after seal test. Catastrophic failures are defined as missing leads, broken packages or lids off.

Note 4: The PIND test may be performed in any sequence after step 9 and prior to step 16. See MIL-M-38510, paragraph 4.6.3.

Note 5: Class S devices shall be serialized prior to interim electrical parameter measurements.

Note 6: When specified, all devices shall be tested for those parameters requiring delta calculations.

Note 7: Reverse bias burn-in is a requirement only when specified in the applicable device specification. The order of performing burn-in and reverse bias burn-in may be inverted.

Note 8: For Class S devices, the seal test may be performed in any sequence between step 16 and step 19, but it shall be performed after all shearing and forming operations on the terminals.

Note 9: For Class B devices, the fine and gross seal tests shall be performed separate or together in any sequence and order between step 6 and step 20 except that they shall be performed after all shearing and forming operations on the terminals. When 100% seal screen cannot be performed after shearing and forming (e.g. flatpacks and chip carriers) the seal screen shall be done 100% prior to those operations and a sample test (LTPD = 5) shall be performed on each inspection lot following these operations. If the sample fails, 100% rescreening shall be required.

Note 10: The radiographic screen may be performed in any sequence after step 9.

Note 11: Samples shall be selected for testing in accordance with the specific device class and lot requirements of Method 5005.

Note 12: External visual shall be performed on the lot any time after step 19 and prior to shipment.

Note 13: Read and Record when post burn-in data measurements are specified.

Note 14: PDA shall apply to all static, dynamic, functional and switching measurements at either 25°C or maximum rated operating temperature.

Section 2

CPU—Central Processing Units

Section 2 Contents

National Semiconductor

NS32532-20/NS32532-25/NS32532-30 High-Performance 32-Bit Microprocessor

General Description

The NS32532 is a high-performance 32-bit microprocessor in the Series 32000® family. It is software compatible with the previous microprocessors in the family but with a greatly enhanced internal implementation.

The high-performance specifications are the result of a four-stage instruction pipeline, on-chip instruction and data caches, on-chip memory management unit and a significantly increased clock frequency. In addition, the system interface provides optimal support for applications spanning a wide range, from low-cost, real-time controllers to highly sophisticated, general purpose multiprocessor systems.

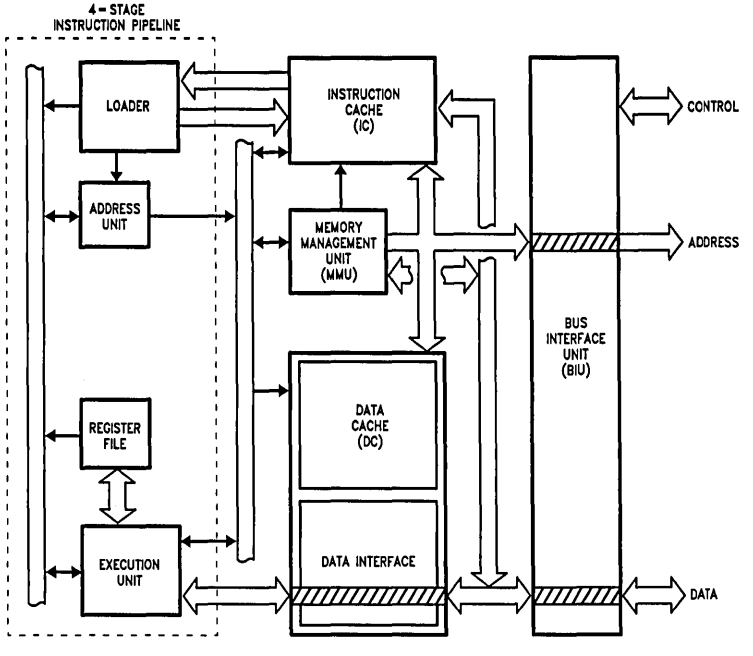
The NS32532 integrates more than 370,000 transistors fabricated in a 1.25 μm double-metal CMOS technology. The advanced technology and mainframe-like design of the device enable it to achieve more than 10 times the throughput of the NS32032 in typical applications.

In addition to generally improved performance, the NS32532 offers much faster interrupt service and task switching for real-time applications.

Features

- Software compatible with the Series 32000 family
- 32-bit architecture and implementation
- 4-GByte uniform addressing space
- On-chip memory management unit with 64-entry translation look-aside buffer
- 4-Stage instruction pipeline
- 512-Byte on-chip instruction cache
- 1024-Byte on-chip data cache
- High-performance bus
  - — Separate 32-bit address and data lines
  - — Burst mode memory accessing
  - — Dynamic bus sizing
- Extensive multiprocessing support
- Floating-point support via the NS32381 or NS32580
- 1.25 μm double-metal CMOS technology
- 175-pin PGA package

Block Diagram



TL/EE/9354-1

FIGURE 1

Table of Contents

Table of Contents (Continued)

2

List of Illustrations

List of Illustrations (Continued)

List of Tables

1.0 Product Introduction

The NS32532 is an extremely sophisticated microprocessor in the Series 32000 family with a full 32-bit architecture and implementation optimized for high-performance applications.

By employing a number of mainframe-like features, the device can deliver 15 MIPS peaks performance with no wait states at a frequency of 30 MHz.

The NS32532 is fully software compatible will all the other Series 32000 CPUs. The architectural features of the Series 32000 family and particularly the NS32532 CPU, are described briefly below.

Powerful Addressing Modes. Nine addressing modes available to all instructions are included to access data structures efficiently.

Data Types. The architecture provides for numerous data types, such as byte, word, doubleword, and BCD, which may be arranged into a wide variety of data structures.

Symmetric Instruction Set. While avoiding special case instructions that compilers can't use, the Series 32000 architecture incorporates powerful instructions for control operations, such as array indexing and external procedure calls, which save considerable space and time for compiled code.

Memory-to-Memory Operations. The Series 32000 CPUs represent two-address machines. This means that each operand can be referenced by any one of the addressing modes provided.

This powerful memory-to-memory architecture permits memory locations to be treated as registers for all usefull operations. This is important for temporary operands as well as for context switching.

Memory Management. The NS32532 on-chip memory management unit provides advanced operating system support functions, including dynamic address translation, virtual memory management, and memory protection.

Large, Uniform Addressing. The NS32532 has 32-bit address pointers that can address up to 4 gigabytes without requiring any segmentation; this addressing scheme provides flexible memory management without added-on expense.

Modular Software Support. Any software package for the Series 32000 family can be developed independent of all other packages, without regard to individual addressing. In addition, ROM code is totally relocatable and easy to access, which allows a significant reduction in hardware and software costs.

Software Processor Concept. The Series 32000 architecture allows future expansions of the instruction set that can be executed by special slave processors, acting as extensions to the CPU. This concept of slave processors is unique to the Series 32000 family. It allows software compatibility even for future components because the slave hardware is transparent to the software. With future advances in semiconductor technology, the slaves can be physically integrated on the CPU chip itself.

To summarize, the architectural features cited above provide three primary performance advantages and characteristics:

- High-level language support
- Easy future growth path
- Application flexibility

2.0 Architectural Description

2.1 REGISTER SET

The NS32532 CPU has 28 internal registers grouped according to functions as follows: 8 general purpose, 7 address, 1 processor status, 1 configuration, 7 memory management and 4 debug. All registers are 32 bits wide except for the module and processor status, which are each 16 bits wide. *Figure 2-1* shows the NS32532 internal registers.

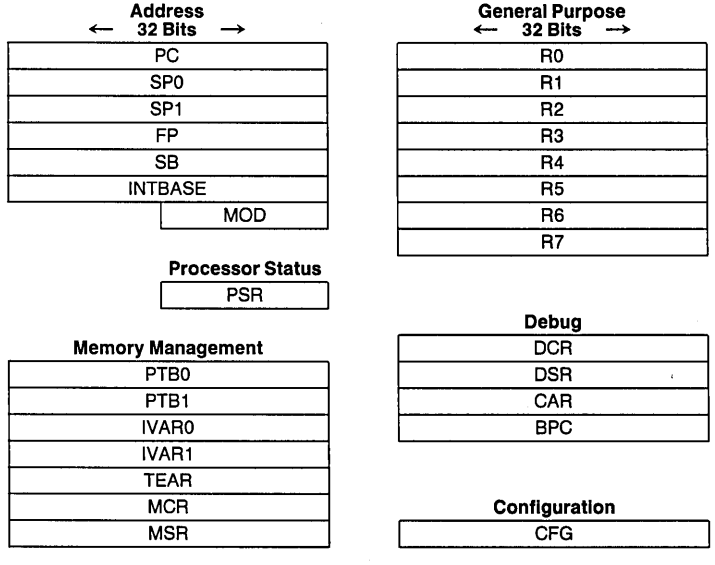


FIGURE 2-1. NS32532 Internal Registers

2.0 Architectural Description (Continued)

2.1.1 General Purpose Registers

There are eight registers (R0–R7) used for satisfying the high speed general storage requirements, such as holding temporary variables and addresses. The general purpose registers are free for any use by the programmer. They are 32 bits in length. If a general purpose register is specified for an operand that is eight or 16 bits long, only the low part of the register is used; the high part is not referenced or modified.

2.1.2 Address Registers

The seven address registers are used by the processor to implement specific address functions. A description of them follows.

PC—Program Counter. The PC register is a pointer to the first byte of the instruction currently being executed. The PC is used to reference memory in the program section.

SP0, SP1—Stack Pointers. The SP0 register points to the lowest address of the last item stored on the INTERRUPT STACK. This stack is normally used only by the operating system. It is used primarily for storing temporary data, and holding return information for operating system subroutines and interrupt and trap service routines. The SP1 register points to the lowest address of the last item stored on the USER STACK. This stack is used by normal user programs to hold temporary data and subroutine return information.

When a reference is made to the selected Stack Pointer (see PSR S-bit), the terms 'SP Register' or 'SP' are used. SP refers to either SP0 or SP1, depending on the setting of the S bit in the PSR register. If the S bit in the PSR is 0, SP refers to SP0. If the S bit in the PSR is 1 then SP refers to SP1.

The NS32532 also allows the SP1 register to be directly loaded and stored using privileged forms of the LPRi and SPRi instructions, regardless of the setting of the PSR S-bit. When SP1 is accessed in this manner, it is referred to as 'USP Register' or simply 'USP'.

Stacks in the Series 32000 family grow downward in memory. A Push operation pre-decrements the Stack Pointer by the operand length. A Pop operation post-increments the Stack Pointer by the operand length.

FP—Frame Pointer. The FP register is used by a procedure to access parameters and local variables on the stack. The FP register is set up on procedure entry with the ENTER instruction and restored on procedure termination with the EXIT instruction.

The frame pointer holds the address in memory occupied by the old contents of the frame pointer.

SB—Static Base. The SB register points to the global variables of a software module. This register is used to support relocatable global variables for software modules. The SB register holds the lowest address in memory occupied by the global variables of a module.

INTBASE—Interrupt Base. The INTBASE register holds the address of the dispatch table for interrupts and traps (Section 3.2.1).

MOD—Module. The MOD register holds the address of the module descriptor of the currently executing software module. The MOD register is 16 bits long, therefore the module table must be contained within the first 64 kbytes of memory.

2.1.3 Processor Status Register

The Processor Status Register (PSR) holds status information for the microprocessor.

The PSR is sixteen bits long, divided into two eight-bit halves. The low order eight bits are accessible to all programs, but the high order eight bits are accessible only to programs executing in Supervisor Mode.

C The C bit indicates that a carry or borrow occurred after an addition or subtraction instruction. It can be used with the ADDC and SUBC instructions to perform multiple-precision integer arithmetic calculations. It may have a setting of 0 (no carry or borrow) or 1 (carry or borrow).

T The T bit causes program tracing. If this bit is set to 1, a TRC trap is executed after every instruction (Section 3.3.1).

L The L bit is altered by comparison instructions. In a comparison instruction the L bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as unsigned integers. Otherwise, it is set to "0". In Floating-Point comparisons, this bit is always cleared.

V The V-bit enables generation of a trap (OVF) when an integer arithmetic operation overflows.

F The F bit is a general condition flag, which is altered by many instructions (e.g., integer arithmetic instructions use it to indicate overflow).

Z The Z bit is altered by comparison instructions. In a comparison instruction the Z bit is set to "1" if the second operand is equal to the first operand; otherwise it is set to "0".

N The N bit is altered by comparison instructions. In a comparison instruction the N bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as signed integers. Otherwise, it is set to "0".

U If the U bit is "1" no privileged instructions may be executed. If the U bit is "0" then all instructions may be executed. When U = 0 the processor is said to be in Supervisor Mode; when U = 1 the processor is said to

15							8	7							0
				I	P	S	U	N	Z	F	V		L	T	C

FIGURE 2-2. Processor Status Register (PSR)

2.0 Architectural Description (Continued)

be in User Mode. A User Mode program is restricted from executing certain instructions and accessing certain registers which could interfere with the operating system. For example, a User Mode program is prevented from changing the setting of the flag used to indicate its own privilege mode. A Supervisor Mode program is assumed to be a trusted part of the operating system, hence it has no such restrictions.

S The S bit specifies whether the SP0 register or SP1 register is used as the Stack Pointer. The bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use the SP0 register) or 1 (use the SP1 register).

P The P bit prevents a TRC trap from occuring more than once for an instruction (Section 3.3.1). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I If I = 1, then all interrupts will be accepted. If I = 0, only the NMI interrupt is accepted. Trap enables are not affected by this bit.

2.1.4 Configuration Register

The Configuration Register (CFG) is 32 bits wide, of which ten bits are implemented. The implemented bits enable various operating modes for the CPU, including vectoring of interrupts, execution of slave instructions, and control of the on-chip caches. In the NS32332 bits 4 through 7 of the CFG register selected between the 16-bit and 32-bit slave protocols and between 512-byte and 4-Kbyte page sizes. The NS32532 supports only the 32-bit slave protocol and 4-Kbyte page size: consequently these bits are forced to 1.

When the CFG register is loaded using the LPRi instruction, bits 14 through 31 should be set to 0. Bits 4 through 7 are ignored during loading, and are always returned as 1's when CFG is stored via the SPRi instruction. When the SETCFG instruction is executed, the contents of the CFG register bits 0 through 3 are loaded from the instruction's short field, bits 4 through 7 are ignored and bits 8 through 13 are forced to 0.

The format of the CFG register is shown in *Figure 2-3*. The various control bits are described below.

I Interrupt vectoring. This bit controls whether maskable interrupts are handled in nonvectored (I=0) or vectored (I=1) mode. Refer to Section 3.2.3 for more information.

F Floating-point instruction set. This bit indicates whether a floating-point unit (FPU) is present to execute floating-point instructions. If this bit is 0 when the CPU executes a floating-point instruction, a Trap (UND) occurs. If this bit is 1, then the CPU transfers the instruction and any necessary operands to the FPU using the slave-processor protocol described in Section 3.1.4.1.

M Memory management instruction set. This bit enables the execution of memory management instructions. If this bit is 0 when the CPU executes an LMR, SMR, RDVAL, or WRVAL instruction, a Trap (UND) occurs. If this bit is 1, the CPU executes LMR, SMR, RDVAL, and WRVAL instructions using the on-chip MMU.

C Custom instruction set. This bit indicates whether a custom slave processor is present to execute custom instructions. If this bit is 0 when the CPU executes a custom instruction, a Trap (UND) occurs. If this bit is 1, the CPU transfers the instruction and any necessary operands to the custom slave processor using the slave-processor protocol described in Section 3.1.4.1.

DE Direct-Exception mode enable. This bit enables the Direct-Exception mode for processing exceptions. When this mode is selected, the CPU response time to interrupts and other exceptions is significantly improved. Refer to Section 3.2.1 for more information.

DC Data Cache enable. This bit enables the on-chip Data Cache to be accessed for data reads and writes. Refer to Section 3.4.2 for more information.

LDC Lock Data Cache. This bit controls whether the contents of the on-chip Data Cache are locked to fixed memory locations (LDC=1), or updated when a data read is missing from the cache (LDC=0).

IC Instruction Cache enable. This bit enables the on-chip Instruction Cache to be accessed for instruction fetches. Refer to Section 3.4.1 for more information.

LIC Lock Instruction Cache. This bit controls whether the contents of the on-chip Instruction Cache are locked to fixed memory locations (LIC=1), or updated when an instruction fetch is missing from the cache (LIC=0).

PF Pipelined Floating-point execution. This bit indicates whether the floating-point unit uses the pipelined slave protocol. When PF is 1 the pipelined protocol is selected. PF is ignored if the F bit is 0. Refer to Section 3.1.4.2 for more information.

31 14	13 8						7 0							
Reserved	PF	LIC	IC	LDC	DC	DE	1	1	1	1	C	M	F	I

FIGURE 2-3. Configuration Register (CFG) Bits 13 to 31 are Reserved; Bits 4 to 7 are Forced to 1.

2.0 Architectural Description (Continued)

2.1.5 Memory Management Registers

The NS32532 provides 7 registers to support memory management functions. They are accessed by means of the LMR and SMR instructions. All of them can be read and written except IVAR0 and IVAR1 that are write-only. A description of the memory management registers is given in the following sections.

PTB0, PTB1—Page Table Base Pointers. The PTBn registers hold the physical addresses of the level-1 page tables used in address translation. The least significant 12 bits are permanently zero, so that each register always points to a 4-Kbyte boundary in memory.

When either PTB0 or PTB1 is loaded by executing an LMR instruction, the MMU automatically invalidates all entries in the TLB that had been translated using the old value in the selected PTBn register.

The format of the PTBn registers is shown in *Figure 2-4*.

31 12	11 0
Base Address	000000000000

FIGURE 2-4. Page Table Base Registers (PTBn)

IVAR0, IVAR1—Invalidate Virtual Address. The Invalidate Virtual Address registers are write-only registers. When a virtual address is written to IVAR0 or IVAR1 using the LMR instruction, the translation for that virtual address is purged, if present, from the TLB. This must be done whenever a Page Table Entry has been changed in memory, since the TLB might otherwise contain an incorrect translation value.

Another technique for purging TLB entries is to load a PTBn register. Turning off translation (clearing the MCR TU and/or TS bits) does not purge any entries from the TLB.

TEAR—Translation Exception Address Register. The TEAR register is loaded by the on-chip MMU when a translation exception occurs. It contains the 32-bit virtual address that caused the translation exception.

TEAR is not updated if a page fault is detected while prefetching an instruction that is not executed because the previous instruction caused a trap.

MCR—Memory Management Control. The MCR register controls the operation of the MMU. Only four bits are implemented. Bits 4 to 31 are reserved for future use and must be loaded with zeroes.

When MCR is read as a 32-bit word, bits 4 to 31 are returned as zeroes. The format of MCR is shown in *Figure 2-5*. Details on the control bits are given below.

TU Translate User. While this bit is 1, address translation is enabled for User-Mode memory references. While this bit is 0, address translations is disabled for User-Mode memory references.

TS Translate Supervisor. While this bit is 1, address translation is enabled for Supervisor Mode memory references. While this bit is 0, address translation is disabled for Supervisor-Mode memory references.

DS Dual Space. While this bit is 1, then PTB1 contains the level-1 page table base address of all addresses specified in User-Mode, and PTB0 contains the level-1 page table base address of all addresses specified in Supervisor Mode. While this bit is 0, then PTB0 contains the level-1 page table base address of all addresses specified in both User and Supervisor Modes.

AO Access Level Override. When this bit is set to 1, User-Mode accesses are given Supervisor Mode privilege.

31 4	3			0
Reserved	AO	DS	TS	TU

FIGURE 2-5. Memory Management Control Register (MCR)

MSR—Memory Management Status. The MSR register provides status information related to the occurrence of a translation exception. Only eight bits are implemented. Bits 8 to 31 are ignored when MSR is loaded and are returned as zeroes when it is read as a 32-bit word. MSR is only updated by the MMU when a protection violation or page fault is detected while translating an address for a reference required to execute an instruction. It is not updated if a page fault is detected during either an operand or an instruction prefetch, if the data being prefetched is not needed due to a change in the instruction execution sequence. The format of MSR is shown in *Figure 2-6*. Details on the function of each bit are given below.

TEX Translation Exception. This two-bit field specifies the cause of the current address translation exception. (Trap(ABT)). Combinations appearing in this field are summarized below.

00 No Translation Exception

01 First Level PTE Invalid

10 Second Level PTE Invalid

11 Protection Violation

During address translation, if a protection violation and an invalid PTE are detected at the same time, the TEX field is set to indicate a protection violation.

DDT Data Direction. This bit indicates the direction of the transfer that the CPU was attempting when the translation exception occurred.

DDT = 0 => Read Cycle

DDT = 1 => Write Cycle

UST User/Supervisor. This bit indicates whether the Translation Exception was caused by a User-Mode or Supervisor Mode reference. If UST is 1, then the exception was caused by a User-Mode reference; otherwise it was caused by a Supervisor Mode reference.

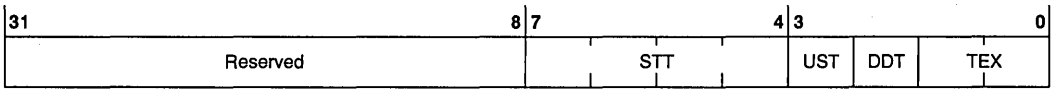


FIGURE 2-6. Memory Management Status Register (MSR)

STT CPU Status. This four bit field is set on an address translation exception according to the following encodings.

1000 Sequential Instruction Fetch

1001 Non-Sequential Instruction Fetch

1010 Data Transfer

1011 Read Read-Modify-Write Operand

1100 Read for Effective Address

If a reference for an Interrupt-Acknowledge or End-of-Interrupt bus cycle (either Master of Cascaded) causes a Translation Exception, then the value of the STT-field is undefined.

2.1.6 Debug Registers

The NS32532 contains 4 registers dedicated for debugging functions.

These registers are accessed using privileged forms of the LPRi and SPRi instructions.

DCR—Debug Condition Register. The DCR Register enables detection of debug conditions. The format of the DCR is shown in *Figure 2-7;* the various bits are described below. A debug condition is enabled when the related bit is set to 1.

CBE0 Compare Byte Enable 0; when set, BYTE0 of an aligned double-word is included in the address comparison

CBE1 Compare Byte Enable 1; when set, BYTE1 of an aligned double-word is included in the address comparison

CBE2 Compare Byte Enable 2; when set, BYTE2 of an aligned double-word is included in the address comparison

CBE3 Compare Byte Enable 3; when set, BYTE3 of an aligned double-word is included in the address comparison

VNP Compare virtual address (VNP = 1) or physical address (VNP = 0)

CWR Address-compare enable for write references

CRD Address-compare enable for read references

CAE Address-compare enable

TR Enable Trap (DBG) when a debug condition is detected

PCE PC-match enable

UD Enable debug conditions in User-Mode

SD Enable debug conditions in Supervisor Mode

DEN Enable debug conditions

The following 2 bits control testing features that can be used during initial system debugging. These features are unique to the NS32532 implementation of the Series 32000 architecture; as such, they may not be supported in future implementations. For normal operation these 2 bits should be set to 0.

SI Single-Instruction mode enable. This bit, when set to 1, inhibits the overlapping of instruction's execution.

BCP Branch Condition Prediction disable. When this bit is 1, the branch prediction mechanism is disabled. See Section 3.1.3.1.

DSR—Debug Status Register. The DSR Register indicates debug conditions that have been detected. When the CPU detects an enabled debug condition, it sets the corresponding bit (BC, BEX, BCA) in the DSR to 1. When an address-compare condition is detected, then the RD-bit is loaded to indicate whether a read or write reference was performed. Software must clear all the bits in the DSR when appropriate. The format of the DSR is shown in *Figure 2-8;* the various fields are described below.

RD Indicates whether the last address-compare condition was for a read (RD = 1) or write (RD = 0) reference

BPC PC-match condition detected

BEX External condition detected

BCA Address-compare condition detected

Note 1: The content of the DSR register is not defined if a debug condition was detected on a floating-point instruction in pipelined mode and a trap was generated by a previous floating-point instruction.

Note 2: If an address compare is detected on a read and a write for the same instruction then the RD-bit will remain clear.

CAR—Compare Address Register. The CAR Register contains the address that is compared to operand reference addresses to detect an address-compare condition. The address must be double-word aligned; that is, the two least-significant bits must be 0. The CAR is 32 bits wide.

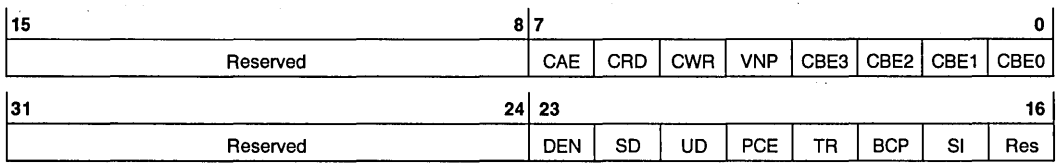


FIGURE 2-7. Debug Condition Register (DCR)

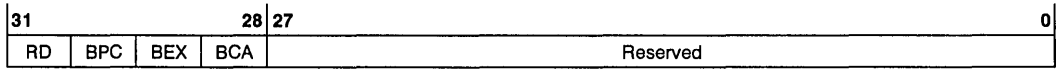


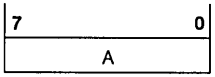
FIGURE 2-8. Debug Status Register (DSR)

2.0 Architectural Description (Continued)

BPC—Breakpoint Program Counter. The BPC Register contains the address that is compared with the PC contents to detect a PC-match condition. The BPC Register is 32 bits wide.

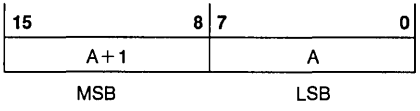
2.2 MEMORY ORGANIZATION

The NS32532 implements full 32-bit virtual addresses. This allows the CPU to access up to 4 Gbytes of virtual memory. The memory is a uniform linear address space. Memory locations are numbered sequentially starting at zero and ending at $2^{32}-1$. The number specifying a memory location is called an address. The contents of each memory location is a byte consisting of eight bits. Unless otherwise noted, diagrams in this document show data stored in memory with the lowest address on the right and the highest address on the left. Also, when data is shown vertically, the lowest address is at the top of a diagram and the highest address at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number zero, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.



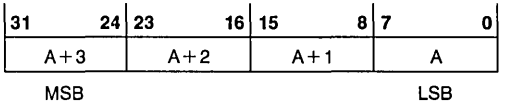
Byte at Address A

Two contiguous bytes are called a word. Except where noted, the least significant byte of a word is stored at the lower address, and the most significant byte of the word is stored at the next higher address. In memory, the address of a word is the address of its least significant byte, and a word may start at any address.



Word at Address A

Two contiguous words are called a double-word. Except where noted, the least significant word of a double-word is stored at the lowest address and the most significant word of the double-word is stored at the address two higher. In memory, the address of a double-word is the address of its least significant byte, and a double-word may start at any address.



Double-Word at Address A

Although memory is addressed as bytes, it is actually organized as double-words. Note that access time to a word or a double-word depends upon its address, e.g. double-words that are aligned to start at addresses that are multiples of four will be accessed more quickly than those not so aligned. This also applies to words that cross a double-word boundary.

2.2.1 Address Mapping

Figure 2-9 shows the NS32532 address mapping.

The NS32532 supports the use of memory-mapped peripheral devices and coprocessors. Such memory-mapped devices can be located at arbitrary locations in the address space except for the upper 8 Mbytes of virtual memory (addresses between FF800000 (hex) and FFFFFFFF (hex), inclusive), which are reserved by National Semiconductor Corporation. Nevertheless, it is recommended that high-performance peripheral devices and coprocessors be located in a specific 8 Mbyte region of virtual memory (addresses between FF000000 (hex) and FF7FFFFF (hex), inclusive), that is dedicated for memory-mapped I/O. This is because the NS32532 detects references to the dedicated locations and serializes reads and writes. See Section 3.1.3.3. When making I/O references to addresses outside the dedicated region, external hardware must indicate to the NS32532 that special handling is required.

In this case a small performance degradation will also result. Refer to Section 3.1.3.2 for more information on memory-mapped I/O.

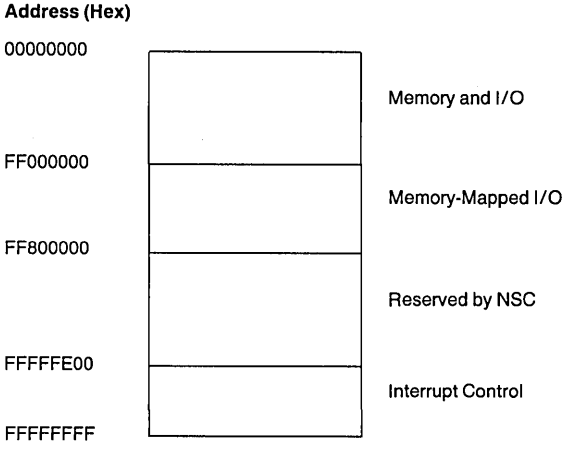


FIGURE 2-9. NS32532 Address Mapping

2.0 Architectural Description (Continued)

2.3 MODULAR SOFTWARE SUPPORT

The NS32532 provides special support for software modules and modular programs.

Each module in a NS32532 software environment consists of three components:

1. Program Code Segment.

   This segment contains the module's code and constant data.

2. Static Data Segment.

   Used to store variables and data that may be accessed by all procedures within the module.

3. Link Table.

   This component contains two types of entries: Absolute Addresses and Procedure Descriptors.

   An Absolute Address is used in the external addressing mode, in conjunction with a displacement and the current MOD Register contents to compute the effective address of an external variable belonging to another module.

   The Procedure Descriptor is used in the call external procedure (CXP) instruction to compute the address of an external procedure.

Normally, the linker program specifies the locations of the three components. The Static Data and Link Table typically reside in RAM; the code component can be either in RAM or in ROM. The three components can be mapped into non-contiguous locations in memory, and each can be independently relocated. Since the Link Table contains the absolute addresses of external variables, the linker need not assign absolute memory addresses for these in the module itself; they may be assigned at load time.

To handle the transfer of control from one module to another, the NS32532 uses a module table in memory and two registers in the CPU.

The Module Table is located within the first 64 kbytes of virtual memory. This table contains a Module Descriptor (also called a Module Table Entry) for each module in the address space of the program. A Module Descriptor has four 32-bit entries corresponding to each component of a module:

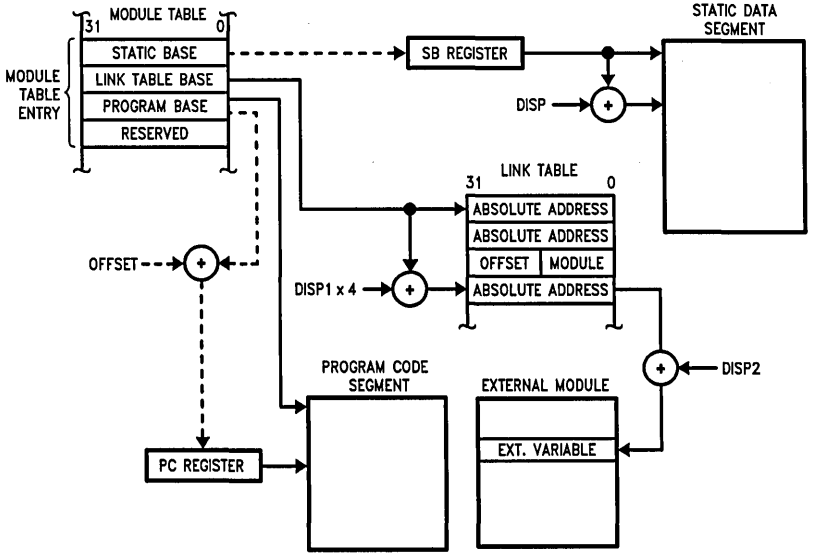
- The Static Base entry contains the address of the beginning of the module's static data segment.
- The Link Table Base points to the beginning of the module's Link Table.
- The Program Base is the address of the beginning of the code and constant data for the module.
- A fourth entry is currently unused but reserved.

The MOD Register in the CPU contains the address of the Module Descriptor for the currently executing module.

The Static Base Register (SB) contains a copy of the Static Base entry in the Module Descriptor of the currently executing module, i.e., it points to the beginning of the current module's static data area.

This register is implemented in the CPU for efficiency purposes. By having a copy of the static base entry or chip, the CPU can avoid reading it from memory each time a data item in the static data segment is accessed.

In an NS32532 software environment modules need not be linked together prior to loading. As modules are loaded, a linking loader simply updates the Module Table and fills the Link Table entries with the appropriate values. No modification of a module's code is required. Thus, modules may be stored in read-only memory and may be added to a system independently of each other, without regard to their individual addressing. *Figure 2-10* shows a typical NS32532 run-time environment.



TL/EE/9354-2

Note: Dashed lines indicate information copied to registers during transfer of control between modules.

FIGURE 2-10. NS32532 Run-Time Environment

2.0 Architectural Description (Continued)

2.4 MEMORY MANAGEMENT

The Memory Mangement Unit of the NS32532 provides support for demand-paged virtual memory. The MMU translates 32-bit virtual addresses into 32-bit physical addresses. The page size is 4096 bytes.

The mapping from virtual to physical addresses is defined by means of sets of tables in physical memory. These tables are found by the MMU using one of its two Page Table Base registers: PTB0 or PTB1. Which register is used depends on the currently selected address space. See Section 2.4.2.

Translation efficiency is improved by means of an on-chip 64-entry translation look-aside buffer (TLB). Refer to Section 3.4.4 for details.

If the MMU detects a protection violation or page fault while translating an address for a reference required to execute an instruction, a translation exception (Trap (ABT)) will result.

2.4.1 Page Tables Structure

The page tables are arranged in a two-level structure, as shown in *Figure 2-11*. Each of the MMU's PTBn registers may point to a Level-1 page table. Each entry of the Level-1 page table may in turn point to a Level-2 page table. Each Level-2 page table entry contains translation information for one page of the virtual space.

The Level-1 page table must remain in physical memory while the PTBn register contains its address and translation is enabled. Level-2 Page Tables need not reside in physical memory permanently, but may be swapped into physical memory on demand as is done with the pages of the virtual space.

The Level-1 Page Table contains 1024 32-bit Page Table Entries (PTE's) and therefore occupies 4 Kbytes. Each entry of the Level-1 Page Table contains a field used to construct the physical base address of a Level-2 Page Table. This field is a 20-bit PFN field, providing bits 12–31 of the physical address. The remaining bits (0–11) are assumed zero, placing a Level-2 Page Table always on a 4-Kbyte (page) boundary.

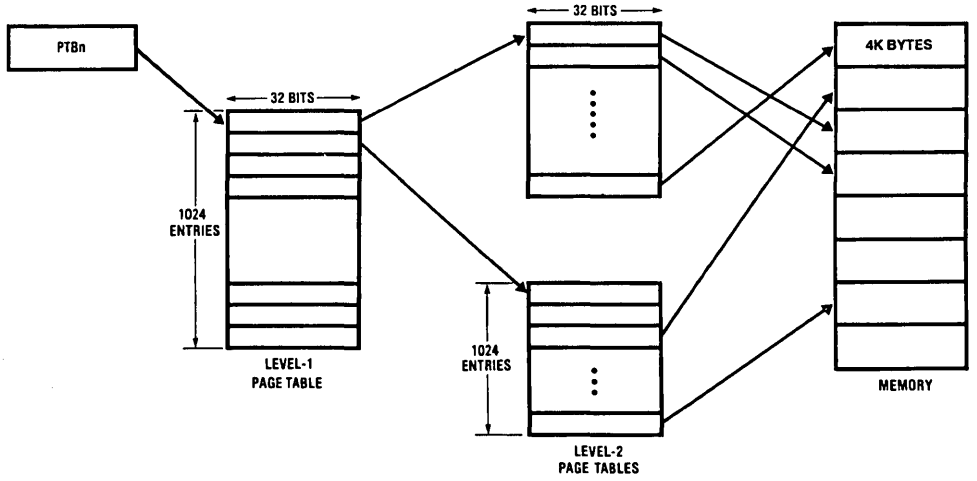
Level-2 Page Tables contain 1024 32-bit Page Table entries, and so occupy 4 Kbytes (1 page). Each Level-2 Page Table Entry points to a final 4-Kbyte physical page frame. In other words, its PFN provides the Page Frame Number portion (bits 12–31) of the translated address (*Figure 2-13*). The OFFSET field of the translated address is taken directly from the corresponding field of the virtual address.

2.4.2 Virtual Address Spaces

When the Dual Space option is selected for address translation in the MCR (Section 2.1.5) the on-chip MMU uses two maps: one for translating addresses presented to it in Supervisor Mode and another for User Mode addresses. Each map is referenced by the MMU using one of the two Page Table Base registers: PTB0 or PTB1. The MMU determines the map to be used by applying the following rules.

1) While the CPU is in Supervisor Mode (U/$\overline{S}$ pin = 0), the CPU is said to be generating virtual addresses belonging to Address Space 0, and the MMU uses the PTB0 register as its reference for looking up translations from memory.
2) While the CPU is in User Mode (U/$\overline{S}$ pin = 1), and the MCR DS bit is set to enable Dual Space translation, the CPU is said to be generating virtual addresses belonging to Address Space 1, and the MMU uses the PTB1 register to look up translations.
3) If Dual Space translation is not selected in the MCR, there is no Adress Space 1, and all virtual addresses generated in both Supervisor and User modes are considered by the MMU to be in Address Space 0. The privilege level of the CPU is used then only for access level checking.

Note: When the CPU executes a Dual-Space Move instruction (MOVUSi or MOVSUi), it temporarily enters User Mode by switching the state of the U/$\overline{S}$ pin. Accesses made by the CPU during this time are treated by the MMU as User-Mode accesses for both mapping and access level checking. It is possible, however, to force the MMU to assume Supervisor Mode privilege on such accesses by setting the Access Override (AO) bit in the MCR (Section 2.1.5).



TL/EE/9354-3

FIGURE 2-11. Two-Level Page Tables

2.0 Architectural Description (Continued)

2.4.3 Page Table Entry Formats

Figure 2-12 shows the formats of Level-1 and Level-2 Page Table Entries (PTE's).

The bits are defined as follows:

V Valid. The V bit is set and cleared only by software.

V = 1 => The PTE is valid and may be used for translation by the MMU.

V = 0 => The PTE does not represent a valid translation. Any attempt to use this PTE to translate and address will cause the MMU to generate an Abort trap.

PL Protection Level. This two-bit field establishes the types of accesses permitted for the page in both User Mode and Supervisor Mode, as shown in Table 2-1.

The PL field is modified only by software. In a Level-1 PTE, it limits the maximum access level allowed for all pages mapped through that PTE.

TABLE 2-1. Access Protection Levels

Mode	U/$\overline{S}$	Protection Level Bits (PL)			
		00	01	10	11
User	1	no access	no access	read only	full access
Supervisor	0	read only	full access	full access	full access

NU Not Used. These bits are reserved by National for future enhancements. Their values should be set to zero.

CI Cache Inhibit. This bit appears only in Level-2 PTE's. It is used to specify non-cacheable pages.

R Referenced. This is a status bit, set by the MMU and cleared by the operating system, that indicates whether the page mapped by this PTE has been referenced within a period of time determined by the operating system. It is intended to assist in implementing memory allocation strategies. In a Level-1 PTE, the R bit indicates only that the Level-2 Page Table has been referenced for a translation, without necessarily implying that the translation was successful. In a Level-2 PTE, it indicates that the page mapped by the PTE has been sucessfully referenced.

R = 1 => The page has been referenced since the R bit was last cleared.

R = 0 => The page has not been referenced since the R bit was last cleared.

M Modified. This is a status bit, set by the MMU whenever a write cycle is successfully performed to the page mapped by this PTE. It is initialized to zero by the operating system when the page is brought into physical memory.

M = 1 => The page has been modified since it was last brought into physical memory.

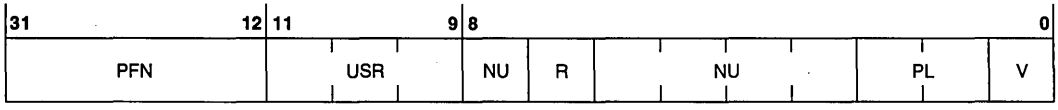
M = 0 => The page has not been modified since it was last brought into physical memory.

In Level-1 Page Table Entries, this bit position is undefined, and is unaltered.

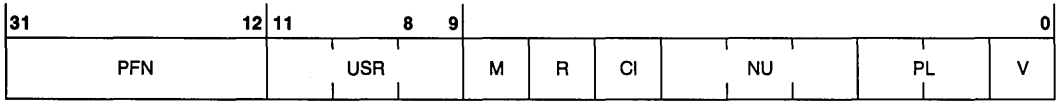
USR User bits. These bits are ignored by the MMU and their values are not changed.

They can be used by the user software.

PFN Page Frame Number. This 20-bit field provides bits 12–31 of the physical address. See *Figure 2-13*.



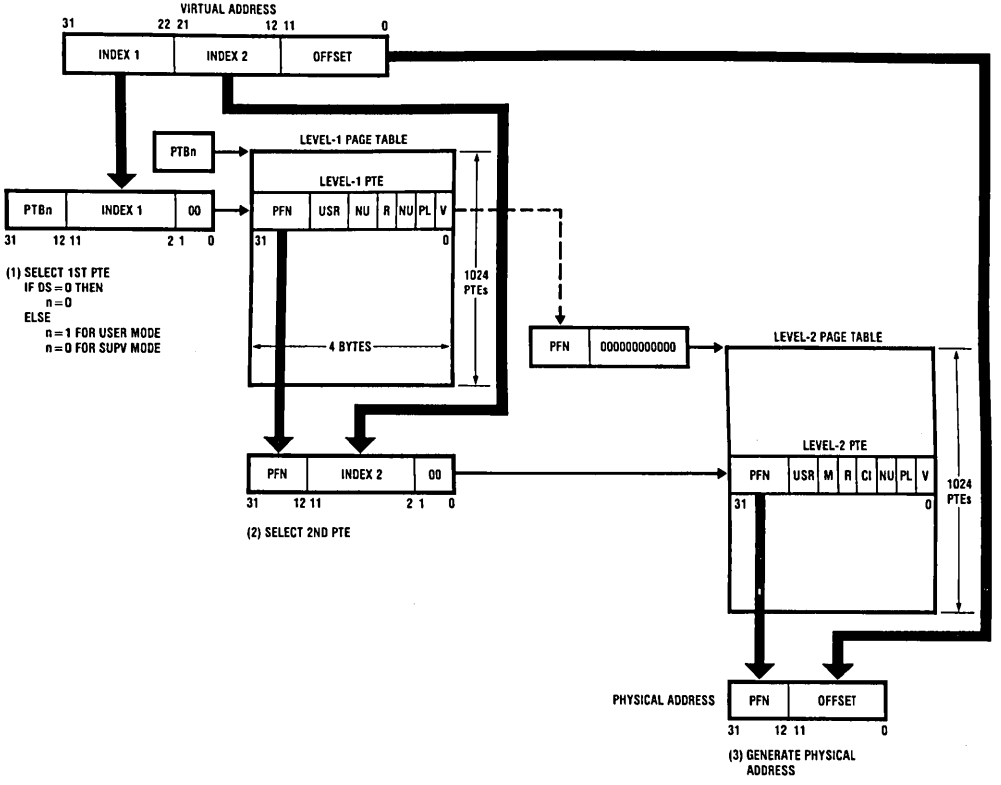
First Level PTE



Second Level PTE

FIGURE 2-12. Page Table Entries (PTE's)

2.0 Architectural Description (Continued)



TL/EE/9354–4

FIGURE 2-13. Virtual to Physical Address Translation

2.4.4 Physical Address Generation

When a virtual address is presented to the MMU and the translation information is not in the TLB, the MMU performs a page table lookup in order to generate the physical address.

The Page Table structure is traversed by the MMU using fields taken from the virtual address. This sequence is diagrammed in *Figure 2-13.*

Bits 12–31 of the virtual address hold the 20-bit Page Number, which in the course of the translation is replaced with the 20-bit Page Frame Number of the physical address. The virtual Page Number field is further divided into two fields, INDEX 1 and INDEX 2.

Bits 0–11 constitute the OFFSET field, which identifies a byte's position within the accessed page. Since the byte position within a page does not change with translation, this value is not used, and is simply echoed by the MMU as bits 0–11 of the final physical address.

The 10-bit INDEX 1 field of the virtual address is used as an index into the Level-1 Page Table, selecting one of its 1024 entries. The address of the entry is computed by adding INDEX 1 (scaled by 4) to the contents of the current Page Table Base register. The PFN field of that entry gives the base address of the selected Level-2 Page Table.

The INDEX 2 field of the virtual address (10 bits) is used as the index into the Level-2 Page Table, by adding it (scaled by 4) to the base address taken from the Level-1 Page Table Entry. The PFN field of the selected entry provides the entire Page Frame Number of the translated address.

The offset field of the virtual address is then appended to this frame number to generate the final physical address.

2.4.5. Address Translation Algorithm

The MMU either translates the 32-bit virtual address to a 32-bit physical address or generates an abort trap to report a translation error. The algorithm used by the MMU to perform the translation is compatible with that of the NS32382. Refer to Appendix C for differences between the two MMUs.

In the description that follows, the symbol 'U' takes the value 1 for a User-Mode memory reference. A reference is a User-Mode reference in the following cases:

1. The reference is performed while executing in User-Mode.
2. The reference is for the source operand of a MOVUS instruction.
3. The reference is for the destination operand of a MOVSU instruction.

The following notations are used in the algorithm.

- A||B → A concatenated with B
- A.B → B is a field inside register A
- (A) → object pointed to by address A
- (A).B → B field of the object pointed to by address A

2.0 Architectural Description (Continued)

Each access is associated with one of two Address Spaces (AS), defined as follows:

AS = U AND MCR.DS

If AS = 1, Page Table Base Register 1 (PTB1) is used to select the first-level page table. If AS = 0, PTB0 is used to select the first-level page table.

The access-level is a 2-bit value used to specify the privilege level of an access. It is determined as follows:

- BIT1 = U AND (NOT(MCR.A0))
- BIT0 = 1 for write, or read with 'RMW' status
  0 otherwise

START TRANSLATION:

If (U = 0 AND MCR.TS = 0 OR U = 1 AND MCR.TU = 0) **then**

/* address translation disabled */

(physical address ← virtual address; CIOUT pin = 0);

/* Note: CIOUT = 0 in all MMU generated accesses */

else BEGIN /* (see also *Figure 2-13*) */

1. Select PTB:
- **If** (MCR.DS = 1 AND U = 1) **then**
— PTB = PTB1,
— AS = 1;
- **else** (PTB = PTB0, AS = 0);

2. Fetch first level PTE:
- PTE Pointer = PTB.BASE ADDRESS||INDEX1||00;
- PTE ← (PTE Pointer); /* Fetch PTE1 */
- Effective PL ← PTE.PL

3. Validate First Level PTE:
- **If** (PTE.PL < access level) **then**
- /* Protection Exception */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 11,
— terminate translation;
- **If** (PTE.V = 0) **then**
- /* PTE1 Invalid */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 01,
— terminate translation;
- **If** (PTE.R = 0) **then**
— Write a Byte (PTE Pointer) .R = 1;
- Effective PL ← PTE.PL

4. Fetch second level PTE:
- PTE Pointer = PTE.PFN||INDEX2||00;
- PTE ← (PTE Pointer); /* Fetch PTE2 */
- **If** (PTE.PL < effective PL) **then**
— Effective PL ← PTE.PL;

5. Validate Second Level PTE:
- **If** (PTE.PL < access level) **then**
- /* Protection Exception */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 11,
— terminate translation;
- **If** (PTE.V = 0) **then**
- /* PTE2 Invalid */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 10,
— terminate translation;
- **If** ((read AND NOT interlocked) AND PTE.R = 0) **then** Read-Modify-Write a double-word interlocked (PTE Pointer).R = 1;
- **If** ((write OR interlocked read) AND (PTE.R = 0 OR PTE.M = 0) **then** Read-Modify-Write a double-word interlocked (PTE Pointer).R = 1, (PTE Pointer).M = 1;

6. Generate Physical address:
- physical address ← PTE.PFN||OFFSET
- CIOUT pin ← PTE.CI

7. Update Translation Buffer:
- Select entry for replacement;
- TLB. Virtual Page Number ← INDEX1|| INDEX2;
- TLB.AS ← AS;
- TLB. Physical Frame Number ← PTE.PFN
- TLB.PL ← Effective PL
- TLB.CI ← PTE.CI
- TLB.M ← (PTE Pointer) .M
- Enable entry

END

Note 1: The TEAR and MSR are only updated when a Trap (ABT) occurs. It is possible that the MMU detects a page fault or protection violation on a reference for an instruction that is not executed, for example on a prefetch. In that event, Trap (ABT) does not occur, and the TEAR and MSR are not updated.

Note 2: If the MMU is translating a virtual address to check protection while executing a RDVAL or WRVAL instruction, then Trap (ABT) occurs only if the level-1 PTE is invalid and the access is permitted by the PL-field. These instructions will not generate an abort if the F bit value can be determined from Level-1 PTE.

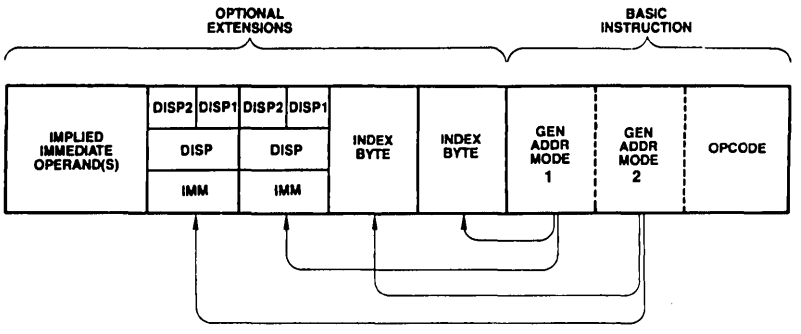
2.5 INSTRUCTION SET

2.5.1 General Instruction Format

Figure 2-14 shows the general format of a Series 32000 instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing Mode ("Gen") fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

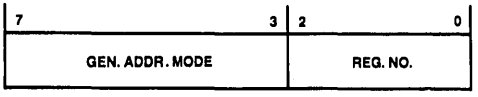
Index Bytes appear when either or both Gen fields specify Scaled Index. In this case, the Gen field specifies only the Scale Factor (1, 2, 4 or 8), and the Index Byte specifies which General Purpose Register to use as the index, and which addressing mode calculation to perform before indexing. See *Figure 2-15.*

2.0 Architectural Description (Continued)



TL/EE/9354-5

FIGURE 2-14. General Instruction Format



TL/EE/9354-6

FIGURE 2-15. Index Byte Format

Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Disp/Imm field may contain one or two displacements, or one immediate value. The size of a Displacement field is encoded with the top bits of that field, as shown in *Figure 2-16*, with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the Opcode field. Both Displacement and Immediate fields are stored most significant byte first. Note that this is different from the memory representation of data (Section 2.2).

Some instructions require additional, 'implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition (Section 2.5.3).

2.5.2 Addressing Modes

The CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."

Addressing modes are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode, within the instruction that acts upon that variable. Extraneous data movement is therefore minimized.

Addressing Modes fall into nine basic types:

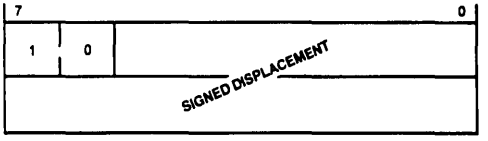
Register: The operand is available in one of the eight General Purpose Registers. In certain Slave Processor instructions, an auxiliary set of eight registers may be referenced instead.

Register Relative: A General Purpose Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.

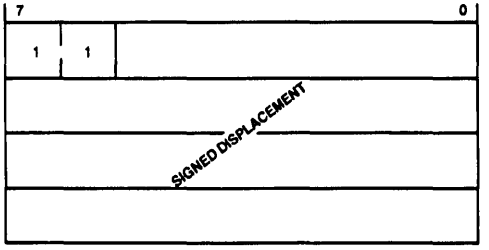
Memory Space: Identical to Register Relative above, except that the register used is one of the dedicated registers PC, SP, SB or FP. These registers point to data areas generally needed by high-level languages.

Byte Displacement: Range −64 to +63

Word Displacement: Range −8192 to +8191



Double Word Displacement:
Range $-(2^{29} - 2^{24})$ to $+ (2^{29} - 1)$*



TL/EE/9354-7

FIGURE 2-16. Displacement Encodings

***Note:** The pattern "11100000" for the most significant byte of the displacement is reserved by National for future enhancements. Therefore, it should never be used by the user program. This causes the lower limit of the displacement range to be $-(2^{29}-2^{24})$ instead of -2^{29}.

2.0 Architectural Description (Continued)

Memory Relative: A pointer variable is found within the memory space pointed to by the SP, SB or FP register. A displacement is added to that pointer to generate the Effective Address of the operand.

Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.

Absolute: The address of the operand is specified by a displacement field in the instruction.

External: A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.

Top of Stack: The currently-selected Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.

Scaled Index: Although encoded as an addressing mode, Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any General Purpose Register by 1, 2, 4 or 8 and adding it into the total, yielding the final Effective Address of the operand.

Table 2-2 is a brief summary of the addressing modes. For a complete description of their actions, see the Instruction Set Reference Manual.

2.5.3 Instruction Set Summary

Table 2-3 presents a brief description of the NS32532 instruction set. The Format column refers to the Instruction Format tables (Appendix A). The Instruction column gives the instruction as coded in assembly language, and the Description column provides a short description of the function provided by that instruction. Further details of the exact operations performed by each instruction may be found in the Instruction Set Reference Manual.

Notations:

i = Integer length suffix: B = Byte
W = Word
D = Double Word

f = Floating Point length suffix: F = Standard Floating
L = Long Floating

gen = General operand. Any addressing mode can be specified.

short = A 4-bit value encoded within the Basic Instruction (see Appendix A for encodings).

imm = Implied immediate operand. An 8-bit value appended after any addressing extensions.

disp = Displacement (addressing constant): 8, 16 or 32 bits. All three lengths legal.

reg = Any General Purpose Register: R0–R7.

areg = Any Processor Register: Address, Debug, Status, Configuration.

mreg = Any Memory Management Register.

creg = A Custom Slave Processor Register (Implementation Dependent).

cond = Any condition code, encoded as a 4-bit field within the Basic Instruction (see Appendix A for encodings).

2.0 Architectural Description (Continued)

TABLE 2-2. NS32532 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0, F0, L0	None: Operand is in the specified register.
00001	Register 1	R1, F1, L1	
00010	Register 2	R2, F2, L2	
00011	Register 3	R3, F3, L3	
00100	Register 4	R4, F4, L4	
00101	Register 5	R5, F5, L5	
00110	Register 6	R6, F6, L6	
00111	Register 7	R7, F7, L7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Relative			
10000	Frame memory relative	disp2(disp1(FP))	Disp2 + Pointer; Pointer found at address Disp1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1(SP))	
10010	Static memory relative	disp2(disp1(SB))	
Reserved			
10011	(Reserved for Future Use)		
Immediate			
10100	Immediate	value	None. Operand is input from instruction queue.
Absolute			
10101	Absolute	@disp	Disp.
External			
10110	External	EXT(disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top of Stack			
10111	Top of stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register; "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	*+disp	
Scaled Index			
11100	Index, bytes	mode[Rn:B]	EA (mode) + Rn.
11101	Index, words	mode[Rn:W]	EA (mode) + 2 × Rn.
11110	Index, double words	mode[Rn:D]	EA (mode) + 4 × Rn.
11111	Index, quad words	mode[Rn:Q]	EA (mode) + 8 × Rn. "Mode' and 'n' are contained within the Index Byte. EA (mode) denotes the effective address generated using mode.

2.0 Architectural Description (Continued)

TABLE 2-3. NS32532 Instruction Set Summary

MOVES

Format	Operation	Operands	Description
4	MOVi	gen,gen	Move a value.
2	MOVQi	short,gen	Extend and move a signed 4-bit constant.
7	MOVMi	gen,gen,disp	Move Multiple: disp bytes (1 to 16).
7	MOVZBW	gen,gen	Move with zero extension.
7	MOVZiD	gen,gen	Move with zero extension.
7	MOVXBW	gen,gen	Move with sign extension.
7	MOVXiD	gen,gen	Move with sign extension.
4	ADDR	gen,gen	Move Effective Address.

INTEGER ARITHMETIC

Format	Operation	Operands	Description
4	ADDI	gen,gen	Add.
2	ADDQi	short,gen	Add signed 4-bit constant.
4	ADDCi	gen,gen	Add with carry.
4	SUBi	gen,gen	Subtract.
4	SUBCi	gen,gen	Subtract with carry (borrow).
6	NEGi	gen,gen	Negate (2's complement).
6	ABSi	gen,gen	Take absolute value.
7	MULi	gen,gen	Multiply.
7	QUOi	gen,gen	Divide, rounding toward zero.
7	REMi	gen,gen	Remainder from QUO.
7	DIVi	gen,gen	Divide, rounding down.
7	MODi	gen,gen	Remainder from DIV (Modulus).
7	MEIi	gen,gen	Multiply to Extended Integer.
7	DEIi	gen,gen	Divide Extended Integer.

PACKED DECIMAL (BCD) ARITHMETIC

Format	Operation	Operands	Description
6	ADDPi	gen,gen	Add Packed.
6	SUBPi	gen,gen	Subtract Packed.

INTEGER COMPARISON

Format	Operation	Operands	Description
4	CMPi	gen,gen	Compare.
2	CMPQi	short,gen	Compare to signed 4-bit constant.
7	CMPMi	gen,gen,disp	Compare Multiple: disp bytes (1 to 16).

LOGICAL AND BOOLEAN

Format	Operation	Operands	Description
4	ANDi	gen,gen	Logical AND.
4	ORi	gen,gen	Logical OR.
4	BICi	gen,gen	Clear selected bits.
4	XORi	gen,gen	Logical Exclusive OR.
6	COMi	gen,gen	Complement all bits.
6	NOTi	gen,gen	Boolean complement: LSB only.
2	Scondi	gen	Save condition code (cond) as a Boolean variable of size i.

SHIFTS

Format	Operation	Operands	Description
6	LSHi	gen,gen	Logical Shift, left or right.
6	ASHi	gen,gen	Arithmetic Shift, left or right.
6	ROTi	gen,gen	Rotate, left or right.

2.0 Architectural Description (Continued)

TABLE 2-3. NS32532 Instruction Set Summary (Continued)

BITS

Format	Operation	Operands	Description
4	TBITi	gen,gen	Test bit.
6	SBITi	gen,gen	Test and set bit.
6	SBITIi	gen,gen	Test and set bit, interlocked.
6	CBITi	gen,gen	Test and clear bit.
6	CBITIi	gen,gen	Test and clear bit, interlocked.
6	IBITi	gen,gen	Test and invert bit.
8	FFSi	gen,gen	Find first set bit.

BIT FIELDS

Bit fields are values in memory that are not aligned to byte boundaries. Examples are PACKED arrays and records used in Pascal. "Extract" instructions read and align a bit field. "Insert" instructions write a bit field from an aligned source.

Format	Operation	Operands	Description
8	EXTi	reg,gen,gen,disp	Extract bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented).
7	EXTSi	gen,gen,imm,imm	Extract bit field (short form).
7	INSSi	gen,gen,imm,imm	Insert bit field (short form).
8	CVTP	reg,gen,gen	Convert to Bit Field Pointer.

ARRAYS

Format	Operation	Operands	Description
8	CHECKi	reg,gen,gen	Index bounds check.
8	INDEXi	reg,gen,gen	Recursive indexing step for multiple-dimensional arrays.

STRINGS

String instructions assign specific functions to the General Purpose Registers:

R4 - Comparison Value
R3 - Translation Table Pointer
R2 - String 2 Pointer
R1 - String 1 Pointer
R0 - Limit Count

Options on all string instructions are:

B (Backward): Decrement string pointers after each step rather than incrementing.

U (Until match): End instruction if String 1 entry matches R4.

W (While match): End instruction if String 1 entry does not match R4.

All string instructions end when R0 decrements to zero.

Format	Operation	Operands	Description
5	MOVSi	options	Move String 1 to String 2.
	MOVST	options	Move string, translating bytes.
5	CMPSi	options	Compare String 1 to String 2.
	CMPST	options	Compare translating, String 1 bytes.
5	SKPSi	options	Skip over String 1 entries.
	SKPST	options	Skip, translating bytes for Until/While.

2.0 Architectural Description (Continued)

TABLE 2-3. NS32532 Instruction Set Summary (Continued)

JUMPS AND LINKAGE

Format	Operation	Operands	Description
3	JUMP	gen	Jump.
0	BR	disp	Branch (PC Relative).
0	Bcond	disp	Conditional branch.
3	CASEi	gen	Multiway branch.
2	ACBi	short,gen,disp	Add 4-bit constant and branch if non-zero.
3	JSR	gen	Jump to subroutine.
1	BSR	disp	Branch to subroutine.
1	CXP	disp	Call external procedure.
3	CXPD	gen	Call external procedure using descriptor.
1	SVC		Supervisor Call.
1	FLAG		Flag Trap.
1	BPT		Breakpoint Trap.
1	ENTER	[reg list],disp	Save registers and allocate stack frame (Enter Procedure).
1	EXIT	[reg list]	Restore registers and reclaim stack frame (Exit Procedure).
1	RET	disp	Return from subroutine.
1	RXP	disp	Return from external procedure call.
1	RETT	disp	Return from trap. (Privileged)
1	RETI		Return from interrupt. (Privileged)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save General Purpose Registers.
1	RESTORE	[reg list]	Restore General Purpose Registers.
2	LPRi	areg,gen	Load Processor Register. (Privileged if PSR, INTBASE, USP, CFG or Debug Registers).
2	SPRi	areg,gen	Store Processor Register. (Privileged if PSR, INTBASE, USP, CFG or Debug Registers).
3	ADJSPi	gen	Adjust Stack Pointer.
3	BISPSRi	gen	Set selected bits in PSR. (Privileged if not Byte length)
3	BICPSRi	gen	Clear selected bits in PSR. (Privileged if not Byte length)
5	SETCFG	[option list]	Set Configuration Register. (Privileged)

FLOATING POINT

Format	Operation	Operands	Description
11	MOVf	gen,gen	Move a Floating Point value.
9	MOVLF	gen,gen	Move and shorten a Long value to Standard.
9	MOVFL	gen,gen	Move and lengthen a Standard value to Long.
9	MOVif	gen,gen	Convert any integer to Standard or Long Floating.
9	ROUNDfi	gen,gen	Convert to integer by rounding.
9	TRUNCfi	gen,gen	Convert to integer by truncating, toward zero.
9	FLOORfi	gen,gen	Convert to largest integer less than or equal to value.
11	ADDf	gen,gen	Add.
11	SUBf	gen,gen	Subtract.
11	MULf	gen,gen	Multiply.
11	DIVf	gen,gen	Divide.
11	CMPf	gen,gen	Compare.
11	NEGf	gen,gen	Negate.
11	ABSf	gen,gen	Take absolute value.
12	POLYf	gen,gen	Polynomial Step.
12	DOTf	gen,gen	Dot Product.
12	SCALBf	gen,gen	Binary Scale.
12	LOGBf	gen,gen	Binary Log.
12	SQRTf	gen,gen	Square Root
12	MACf	gen,gen	Multiply and Accumulate
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

2.0 Architectural Description (Continued)

TABLE 2-3. NS32532 Instruction Set Summary (Continued)

MEMORY MANAGEMENT

Format	Operation	Operands	Description
14	LMR	mreg,gen	Load Memory Management Register. (Privileged)
14	SMR	mreg,gen	Store Memory Management Register. (Privileged)
14	RDVAL	gen	Validate address for reading. (Privileged)
14	WRVAL	gen	Validate address for writing. (Privileged)
8	MOVSUi	gen,gen	Move a value from Supervisor Space to User Space. (Privileged)
8	MOVUSi	gen,gen	Move a value from User Space to Supervisor Space. (Privileged)

MISCELLANEOUS

Format	Operation	Operands	Description
1	NOP		No Operation.
1	WAIT		Wait for interrupt.
1	DIA		Diagnose. Single-byte "Branch to Self" for hardware breakpointing. Not for use in programming.
14	CINV	options,gen	Cache Invalidate. (Privileged)

CUSTOM SLAVE

Format	Operation	Operands	Description
15.5	CCAL0c	gen,gen	Custom Calculate.
15.5	CCAL1c	gen,gen	
15.5	CCAL2c	gen,gen	
15.5	CCAL3c	gen,gen	
15.5	CMOV0c	gen,gen	Custom Move.
15.5	CMOV1c	gen,gen	
15.5	CMOV2c	gen,gen	
15.5	CMOV3c	gen,gen	
15.5	CCMP0c	gen,gen	Custom Compare.
15.5	CCMP1c	gen,gen	
15.1	CCV0ci	gen,gen	Custom Convert.
15.1	CCV1ci	gen,gen	
15.1	CCV2ci	gen,gen	
15.1	CCV3ic	gen,gen	
15.1	CCV4DQ	gen,gen	
15.1	CCV5QD	gen,gen	
15.1	LCSR	gen	Load Custom Status Register.
15.1	SCSR	gen	Store Custom Status Register.
15.0	LCR	creg,gen	Load Custom Register. (Privileged)
15.0	SCR	creg,gen	Store Custom Register. (Privileged)

3.0 Functional Description

This chapter provides details on the functional characteristics of the NS32532 microprocessor.

The chapter is divided into five main sections:

Instruction Execution, Exception Processing, Debugging, On-Chip Caches and System Interface.

3.1 INSTRUCTION EXECUTION

To execute an instruction, the NS32532 performs the following operations:

- Fetch the instruction
- Read source operands, if any (1)
- Calculate results
- Write result operands, if any
- Modify flags, if necessary
- Update the program counter

Under most circumstances, the CPU can be conceived to execute instructions by completing the operations above in strict sequence for one instruction and then beginning the sequence of operations for the next instruction. However, due to the internal instruction pipelining, as well as the occurrence of exceptions, the sequence of operations performed during the execution of an instruction may be altered. Furthermore, exceptions also break the sequentiality of the instructions executed by the CPU.

Details on the effects of the internal pipelining, as well as the occurrence of exceptions on the instruction execution, are provided in the following sections.

Note: 1 In this and following sections, memory locations read by the CPU to calculate effective addresses for Memory-Relative and External addressing modes are considered like source operands, even if the effective address is being calculated for an operand with access class of write.

3.1.1 Operating States

The CPU has five operating states regarding the execution of instructions and the processing of exceptions: Reset, Executing Instructions, Processing An Exception, Waiting-For-An-Interrupt, and Halted. The various states and transitions between them are shown in *Figure 3-1*.

Whenever the $\overline{RST}$ signal is asserted, the CPU enters the reset state. The CPU remains in the reset state until the $\overline{RST}$ signal is driven inactive, at which time it enters the Executing-Instructions state. In the Reset state the contents of certain registers are initialized. Refer to Section 3.5.3 for details.

In the Executing-Instructions state, the CPU executes instructions. It will exit this state when an exception is recognized or a WAIT instruction is encountered. At which time it enters the Processing-An-Exception state or the Waiting-For-An-Interrupt state respectively.

While in the Processing-An-Exception state, the CPU saves the PC, PSR and MOD register contents on the stack and reads the new PC and module linkage information to begin execution of the exception service procedure (see note).

Following the completion of all data references required to process an exception, the CPU enters the Executing-Instructions state.

In the Waiting-For-An-Interrupt state, the CPU is idle. A special status identifying this state is presented on the system interface (Section 3.5). When an interrupt or a debug condition is detected, the CPU enters the Processing-An-Exception state.

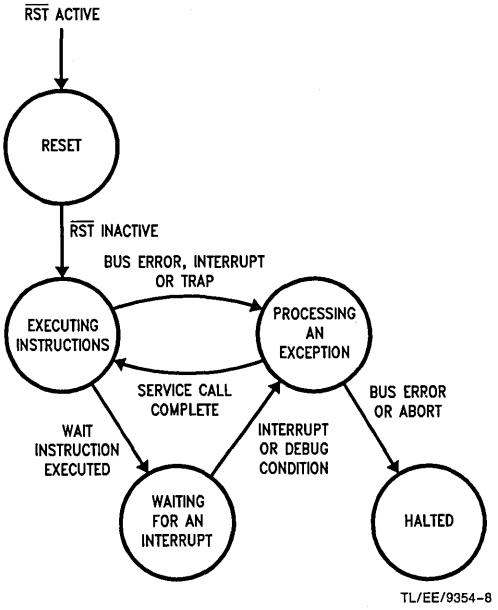


FIGURE 3-1. Operating States

The CPU enters the Halted state when a bus error or abort is detected while the CPU is processing an exception, thereby preventing the transfer of control to an appropriate exception service procedure. The CPU remains in the Halted state until reset occurs. A special status identifying this state is presented on the system interface.

Note: When the Direct-Exception mode is enabled, the CPU does not save the MOD Register contents nor does it read the module linkage information for the exception service procedure. Refer to Section 3.2 for details.

3.1.2 Instruction Endings

The NS32532 checks for exceptions at various points while executing instructions. Certain exceptions, like interrupts, are in most cases recognized between instructions. Other exceptions, like Divide-By-Zero Trap, are recognized during execution of an instruction. When an exception is recognized during execution of an instruction, the instruction ends in one of four possible ways: completed, suspended, terminated, or partially completed. Each type of exception causes a particular ending, as specified in Section 3.2.

3.1.2.1 Completed Instructions

When an exception is recognized after an instruction is completed, the CPU has performed all of the operations for that instruction and for all other instructions executed since the last exception occurred. Result operands have been written, flags have been modified, and the PC saved on the Interrupt Stack contains the address of the next instruction to execute. The exception service procedure can, at its conclusion, execute the RETT instruction (or the RETI instruction for vectored interrupts), and the CPU will begin executing the instruction following the completed instruction.

3.0 Functional Description (Continued)

3.1.2.2 Suspended Instructions

An instruction is suspended when one of several trap conditions or a restartable bus error is detected during execution of the instruction. A suspended instruction has not been completed, but all other instructions executed since the last exception occurred have been completed. Result operands and flags due to be affected by the instruction may have been modified, but only modifications that allow the instruction to be executed again and completed can occur. For certain exceptions (Trap (ABT), Trap (UND), Trap (ILL), and bus errors) the CPU clears the P-flag in the PSR before saving the copy that is pushed on the Interrupt Stack. The PC saved on the Interrupt Stack contains the address of the suspended instruction.

For example, the RESTORE instruction pops up to 8 general-purpose registers from the stack. If an invalid page table entry is detected on one of the references to the stack, then the instruction is suspended. The general-purpose registers due to be loaded by the instruction may have been modified, but the stack pointer still holds the same value that it did when the instruction began.

To complete a suspended instruction, the exception service procedure takes either of two actions:

1. The service procedure can simulate the suspended instruction's execution. After calculating and writing the instruction's results, the flags in the PSR copy saved on the Interrupt Stack should be modified, and the PC saved on the Interrupt Stack should be updated to point to the next instruction to execute. The service procedure can then execute the RETT instruction, and the CPU begins executing the instruction following the suspended instruction. This is the action taken when floating-point instructions are simulated by software in systems without a hardware floating-point unit.
2. The suspended instruction can be executed again after the service procedure has eliminated the trap condition that caused the instruction to be suspended. The service procedure should execute the RETT instruction at its conclusion; then the CPU begins executing the suspended instruction again. This is the action taken by a debugger when it encounters a BPT instruction that was temporarily placed in another instruction's location in order to set a breakpoint.

Note 1: Although the NS32532 allows a suspended instruction to be executed again and completed, the CPU may have read a source operand for the instruction from a memory-mapped peripheral port before the exception was recognized. In such a case, the characteristics of the peripheral device may prevent correct reexecution of the instruction.

Note 2: It may be necessary for the exception service procedure to alter the P-flag in the PSR copy saved on the Interrupt Stack: If the exception service procedure simulates the suspended instruction and the P-flag was cleared by the CPU before saving the PSR copy, then the saved T-flag must be copied to the saved P-flag (like the floating-point instruction simulation described above). Or if the exception service procedure executes the suspended instruction again and the P-flag was not cleared by the CPU before saving the PSR copy, then the saved P-flag must be cleared (like the breakpoint trap described above). Otherwise, no alteration to the saved P-flag is necessary.

3.1.2.3 Terminated Instructions

An instruction being executed is terminated when reset or a nonrestartable bus error occurs. Any result operands and flags due to be affected by the instruction are undefined, as are the contents of the Stack Pointers. The result operands of other instructions executed since the last serializing operation may not have been written to memory. A terminated instruction cannot be completed.

3.1.2.4 Partially Completed Instructions

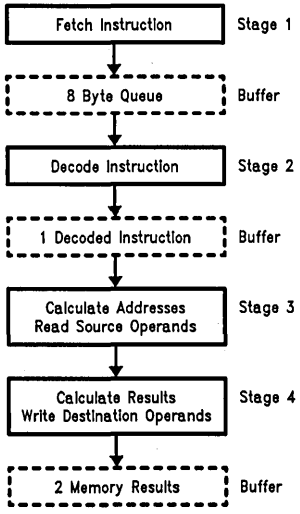
When a restartable bus error, interrupt, abort, or debug condition is recognized during execution of a string instruction, the instruction is said to be partially completed. A partially completed instruction has not completed, but all other instructions executed since the last exception occurred have been completed. Result operands and flags due to be affected by the instruction may have been modified, but the values stored in the string pointers and other general-purpose registers used during the instruction's execution allow the instruction to be executed again and completed.

The CPU clears the P-flag in the PSR before saving the copy that is pushed on the Interrupt Stack. The PC saved on the Interrupt Stack contains the address of the partially completed instruction. The exception service procedure can, at its conclusion, simply execute the RETT instruction (or the RETI instruction for vectored interrupts), and the CPU will resume executing the partially completed instruction.

3.1.3 Instruction Pipeline

The NS32532 executes instructions in a heavily pipelined fashion. This allows a significant performance enhancement since the operations of several instructions are performed simultaneously rather than in a strictly sequential manner.

The CPU provides a four-stage internal instruction pipeline. As shown in *Figure 3-2*, a write buffer, that can hold up to two operands, is also provided to allow write operations to be performed off-line.



TL/EE/9354-9

FIGURE 3-2. NS32532 Internal Instruction Pipeline

Due to the pipelining, operations like fetching one instruction, reading the source operands of a second instruction, calculating the results of a third instruction and storing the results of a fourth instruction, can all occur in parallel.

3.0 Functional Description (Continued)

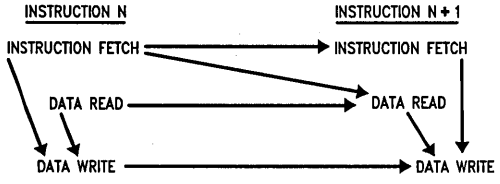
The order of memory references performed by the CPU may also differ from that related to a strictly sequential instruction execution. In fact, when an instruction is being executed, some of the source operands may be read from memory before the instruction is completely fetched. For example, the CPU may read the first source operand for an instruction before it has fetched a displacement used in calculating the address of the second source operand. The CPU, however, always completes fetching an instruction and reading its source operands before writing its results. When more than one source operand must be read from memory to execute an instruction, the operands may be read in any order. Similarly, when more than one result operand is written to memory to execute an instruction, the operands may be written in any order.

An instruction is fetched only after all previous instructions have been completely fetched. However, the CPU may begin fetching an instruction before all of the source operands have been read and results written for previous instructions.

The source operands for an instruction are read only after all previous instructions have been fetched and their source operands read. A source operand for an instruction may be read before all results of previous instructions have been written, except when the source operand's value depends on a result not yet written. The CPU compares the physical address and length of a source operand with those of any results not yet written, and delays reading the source operand until after writing all results on which the source operand depends. Also, the CPU ensures that the interlocked read and write references to execute an SBITIi or CBITIi instruction occur after writing all results of previous instructions and before reading any source operands for subsequent instructions.

The result operands for an instruction are written after all results of previous instructions have been written.

The description above is summarized in *Figure 3-3*, which shows the precedence of memory references for two consecutive instructions.



TL/EE/9354–10

FIGURE 3-3. Memory References for Consecutive Instructions
(An arrow from one reference to another indicates that the first reference always precedes the second.)

Another consequence of overlapping the operations for several instructions, is that the CPU may fetch an instruction and read its source operands, even though the instruction is not executed (e.g., due to the occurrence of an exception). In such a case, the MMU may update the R-bit in Page Table Entries used in referring to the fetched instruction and its source operands.

Special care is needed in the handling of memory-mapped I/O devices. The CPU provides special mechanisms to ensure that the references to these devices are always performed in the order implied by the program. Refer to Section 3.1.3.2 for details.

It is also to be noted that the CPU does not check for dependencies between the fetching of an instruction and the writing of previous instructions' results. Therefore, special care is required when executing self-modifying code.

3.1.3.1 Branch Prediction

One problem inherent to all pipelined machines is what is called "Pipeline Breakage".

This occurs every time the sequentiality of the instructions is broken, due to the execution of certain instructions or the occurrence of exceptions.

The result of a pipeline breakage is a performance degradation, due to the fact that a certain portion of the pipeline must be flushed and new data must be brought in.

The NS32532 provides a special mechanism, called branch prediction, that helps minimize this performance penalty.

When a conditional branch instruction is decoded in the early stages of the pipeline, a prediction on the execution of the instruction is performed.

More precisely, the prediction mechanism predicts backward branches as taken and forward branches as not taken, except for the branch instructions BLE and BNE that are always predicted as taken.

Thus, the resulting probability of correct prediction is fairly high, especially for branch instructions placed at the end of loops.

The sequence of operations performed by the loader and execution units in the CPU is given below:

- Loader detects branches and calculates destination addresses
- Loader uses branch opcode and direction to select between sequential and non-sequential streams
- Loader saves address for alternate stream
- Execution unit resolves branch decision

Due to the branch predicition, some special care is required when writing self-modifying code. Refer to the appropriate section in Appendix B for more information on this subject.

3.1.3.2 Memory-Mapped I/O

The characteristics of certain peripheral devices and the overlapping of instruction execution in the pipeline of the NS32532 require that special handling be applied to memory-mapped I/O references. I/O references differ from memory references in two significant ways, imposing the following requirements:

1. Reading from a peripheral port can alter the value read on the next reference to the same port or another port in the same device. (A characteristic called here "destructive-reading".) Serial communication controllers and FIFO buffers commonly operate in this manner. As explained in "Instruction Pipeline" above, the NS32532 can read the source operands for one instruction while the previous instruction is executing. Because the previous instruction may cause a trap, an interrupt may be recognized, or the flow of control may be otherwise altered, it is a requirement that destructive-reading of source operands before the execution of an instruction be avoided.

3.0 Functional Description (Continued)

2. Writing to a peripheral port can alter the value read from another port of the same device. (A characteristic called here "side-effects of writing"). For example, before reading the counter's value from the NS32202 Interrupt Control Unit it is first necessary to freeze the value by writing to another control register.

However, as mentioned above, the NS32532 can read the source operands for one instruction before writing the results of previous instructions unless the addresses indicate a dependency between the read and write references. Consequently, it is a requirement that read and write references to peripheral that exhibit side-effects of writing must occur in the order dictated by the instructions.

The NS32532 supports 2 methods for handling memory-mapped I/O. The first method is more general; it satisfies both requirements listed above and places no restriction on the location of memory-mapped peripheral devices. The second method satisfies only the requirement for side effects of writing, and it restricts the location of memory-mapped I/O devices, but it is more efficient for devices that do not have destructive-read ports.

The first method for handling memory-mapped I/O uses two signals: $\overline{\text{IOINH}}$ and $\overline{\text{IODEC}}$. When the NS32532 generates a read bus cycle, it asserts the output signal $\overline{\text{IOINH}}$ if either of the I/O requirements listed above is not satisfied. That is, $\overline{\text{IOINH}}$ is asserted during a read bus cycle when (1) the read reference is for an instruction that may not be executed or (2) the read reference occurs while a write reference is pending for a previous instruction. When the read reference is to a peripheral device that implements ports with destructive-reading or side-effects of writing, the input signal $\overline{\text{IODEC}}$ must be asserted; in addition, the device must not be selected if $\overline{\text{IOINH}}$ is active. When the CPU detects that the $\overline{\text{IODEC}}$ input signal is active while the $\overline{\text{IOINH}}$ output signal is also active, it discards the data read during the bus cycle and serializes instruction execution. See the next section for details on serializing operations. The CPU then generates the read bus cycle again, this time satisfying the requirements for I/O and driving $\overline{\text{IOINH}}$ inactive.

The second method for handling memory-mapped I/O uses a dedicated region of virtual memory. The NS32532 treats all references to the memory range from address FF000000 to address FFFFFFFF inclusive in a special manner.

While a write to a location in this range is pending, reads from locations in the same range are delayed. However, reads from locations with addresses lower than FF000000 may occur. Similarly, reads from locations in the above range may occur while writes to locations outside of the range are pending.

It is to be noted that the CPU may assert $\overline{\text{IOINH}}$ even when the reference is within the dedicated region. Refer to Section 3.5.8 for more information on the handling of I/O devices.

3.1.3.3 Serializing Operations

After executing certain instructions or processing an exception, the CPU serializes instruction execution. Serializing instruction execution means that the CPU completes writing all previous instructions' results to memory, then begins fetching and executing the next instruction.

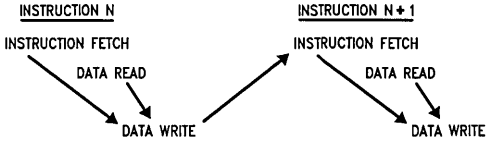
For example, when a new value is loaded into the PSR by executing an LPRW instruction, the pipeline is flushed and a serializing operation takes place. This is necessary since the privilege level might have changed and the instructions following the LPRW instruction must be fetched again with the new privilege level and possibly with a different MMU mapping. See Section 2.4.2.

The CPU serializes instruction execution after executing one of the following instructions: BICPSRW, BISPSRW, BPT, CINV, DIA, FLAG (trap taken), LMR, LPR (CFG, INTBASE, PSR, UPSR, DCR, BPC, DSR, and CAR only), RETT, RETI, and SVC. *Figure 3-4* shows the memory references after serialization.

Note 1: LPRB UPSR can be executed in User Mode to serialize instruction execution.

Note 2: After an instruction that writes a result to memory is executed, the updating of the result's memory location may be delayed until the next serializing operation.

Note 3: When reset or a nonrestartable bus error exception occurs, the CPU discards any results that have not yet been written to memory.



TL/EE/9354-11

FIGURE 3-4. Memory References after Serialization

3.1.4 Slave Processor Instructions

The NS32532 recognizes two groups of instructions being executable by external slave processors:

- Floating Point Instructions
- Custom Slave Instructions

Each Slave Instruction Set is enabled by a bit in the Configuration Register (Section 2.1.4). Any Slave Instruction which does not have its corresponding Configuration Register bit set will trap as undefined, without any Slave Processor communication attempted by the CPU. This allows software simulation of a non-existent Slave Processor.

Note that the Memory Management Instructions, like Floating Point and Custom Slave Instructions, have to be enabled through an appropriate bit in the configuration register in order to be executable.

However, they are not considered here as Slave Instructions, since the NS32532 integrates the MMU on-chip and the execution of them does not follow the protocol of the Slave Instructions.

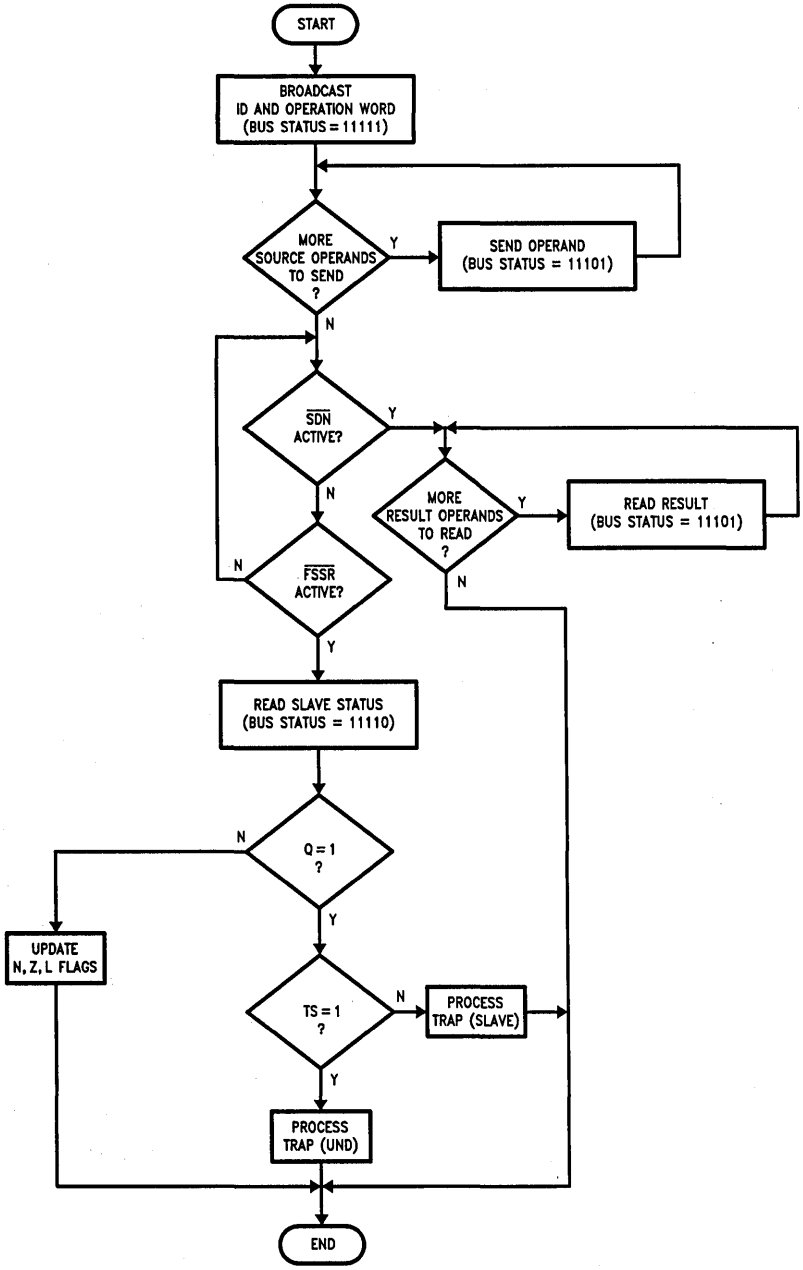
3.1.4.1 Regular Slave Instruction Protocol

Slave Processor instructions have a three-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word. The ID Byte has three functions:

1) It identifies the instruction as being a Slave Processor instruction.
2) It specifies which Slave Processor will execute it.
3) It determines the format of the following Operation Word of the instruction.

Upon receiving a Slave Processor instruction, the CPU initiates the sequence outlined in *Figure 3-5*. While applying Status code 11111 (Broadcast ID Section 3.5.4.1), the CPU transfers the ID Byte on bits D24–D31, the operation

3.0 Functional Description (Continued)



TL/EE/9354-12

FIGURE 3-5. Regular Slave Instruction Protocol: CPU Actions

3.0 Functional Description (Continued)

31			0
ID BYTE	OPCODE (LOW)	OPCODE (HIGH)	XXXXXXXX

FIGURE 3-6. ID and Operation Word

31	15		7							0
ZERO	TS	ZERO	N	Z	0	0	0	L	0	Q

FIGURE 3-7. Slave Processor Status Word

word on bits D8–D23 in a swapped order of bytes and a non-used byte XXXXXXXX (X = don't care) on bits D0–D7 (*Figure 3-6*).

All slave processors observe the bus cycle and inspect the identification code. The slave selected by the identification code continues with the protocol; other slaves wait for the next slave instruction to be broadcast.

After transferring the slave instruction, the CPU sends to the slave any source operands that are located in memory or the General-Purpose registers. The CPU then waits for the slave to assert $\overline{SDN}$ or $\overline{FSSR}$. While the CPU is waiting, it can perform bus cycles to fetch instructions and read source operands for instructions that follow the slave instruction being executed. If there are no bus cycles to perform, the CPU is idle with a special Status indicating that it is waiting for a slave processor. After the slave asserts $\overline{SDN}$ or $\overline{FSSR}$, the CPU follows one of the two sequences described below.

If the slave asserts $\overline{SDN}$, then the CPU checks whether the instruction stores any results to memory or the General-Purpose registers. The CPU reads any such results from the slave by means of 1 or 2 bus cycles and updates the destination.

If the slave asserts $\overline{FSSR}$, then the NS32532 reads a 32-bit status word from the slave. The CPU checks bit 0 in the slave's status word to determine whether to update the PSR flags or to process an exception. *Figure 3-7* shows the format of the slave's status word.

If the Q bit in the status word is 0, the CPU updates the N, Z and L flags in the PSR.

If the Q bit in the status word is set to 1, the CPU processes either a Trap (UND) if TS is 1 or a Trap (SLAVE) if TS is 0.

Note 1: Only the floating-point and custom compare instructions are allowed to return a value of 0 for the Q bit when the $\overline{FSSR}$ signal is activated. All other instructions must always set the Q bit to 1 (to signal a Trap), when activating $\overline{FSSR}$.

Note 2: While executing an LMR or CINV instruction, the CPU displays the operation code and source operand using slave processor write bus cycles, as described in the protocol above. Nevertheless, the CPU does not wait for $\overline{SDN}$ or $\overline{FSSR}$ to be asserted while executing these instructions. This information can be used to monitor the contents of the on-chip TLB, Instruction Cache, and Data Cache.

Note 3: The slave processor must be ready to accept new slave instruction at any time, even while the slave is executing another instruction or waiting for the CPU to read results. For example, the CPU may terminate an instruction being executed by a slave because a non-restartable bus error is detected while the MMU is updating a Page Table Entry for an instruction being prefetched.

Note 4: If a slave instruction stores a result to memory, the CPU checks whether Trap (ABT) would occur on the store operation before reading the result from the slave. For quad-word destination operands, the CPU checks that both double-words of the destination can be stored without an abort before reading either double-word of the result from the slave.

3.1.4.2 Pipelined Slave Instruction Protocol

In order to increase performance of floating-point instructions while maintaining full software compatibility with the Series 32000 architecture, the NS32532 incorporates a pipelined floating-point protocol. This protocol is designed to operate in conjunction with the NS32580 FPC, or any other floating-point slave which conforms to the protocol and the Series 32000 architecture. The protocol is enabled by the PF bit in the CFG register.

The basic methods of transferring data and control information between the CPU and the FPC, are the same as in the regular slave protocol.

However, in pipelined mode, the CPU may send a new floating-point instruction to the FPC before the previous instruction has been completed.

Although the CPU can advance as many as four floating-point instructions before receiving a completion pulse on $\overline{SDN}$ for the first instruction, full exception recovery is assured. This is accomplished through a FIFO mechanism which maintains the addresses of all the floating-point instructions sent to the FPC for execution.

Pipelined execution can occur only for instructions which do not require a result to be read from the FPC.

In cases where a result is to be read back, the CPU will wait for instruction completion before issuing the next instruction. Floating-point instructions can be divided into two groups, depending on the amount of pipelining permitted.

Group A. Fully-Pipelined Instructions

Instructions in this group can be sent to the FPC before previous group A instructions are completed. No instruction completion indication from the FPC is required in order to continue to another group A or group B instruction.

Group A contains floating-point instructions satisfying all of the following conditions.

1. The destination operand is in a floating-point register.
2. The source operand is not of type TOS or IMM.
3. The instruction format is either 11 or 12.

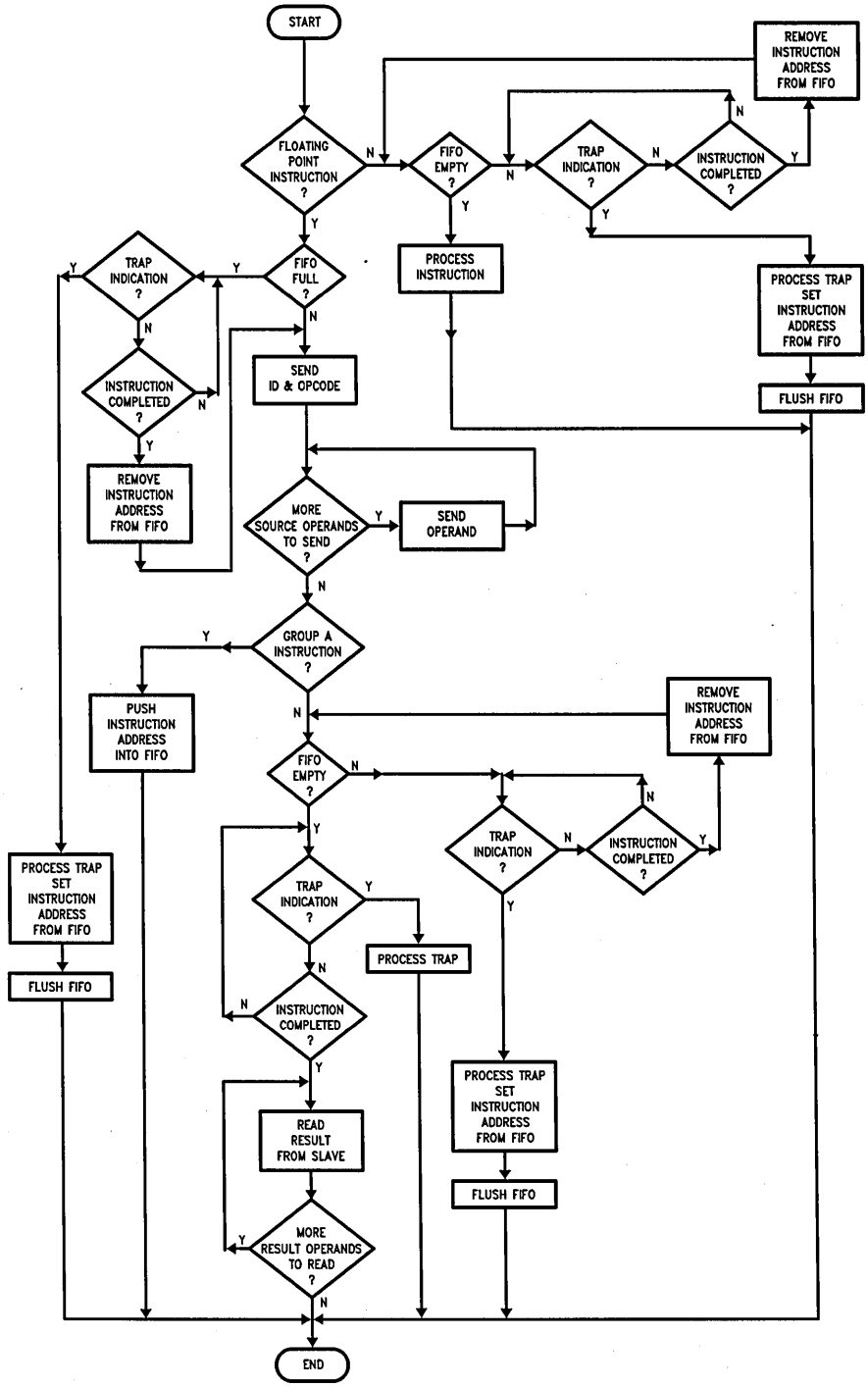
Group B. Half-Pipelined Instructions

Group B instructions can begin execution before previous group A instructions are completed. However, they cannot complete before the FPC signals completion of all the previous floating-point instructions.

Group B contains floating-point instructions satisfying at least one of the following conditions.

1. The destination operand is either in memory or in a CPU register (this includes the CMPf instruction which modifies the PSR register).
2. The source operand is of type TOS or IMM.
3. The instruction format is 9.

3.0 Functional Description (Continued)



TL/EE/9354–73

FIGURE 3-8. Instruction Flow in Pipelined Floating-Point Mode

3.0 Functional Description (Continued)

Note: Non-floating-point instructions cannot be pipelined. They can begin execution only after all other instructions have been completed. The CPU cannot proceed to other instructions before their execution is completed.

3.1.4.3 Instruction Flow and Exceptions

When operating in pipelined mode, the CPU will push the address of group A instructions into a five-entry FIFO after the ID, opcode and source operands have been sent to the FPC. The address will be pushed into the FIFO only if no exception is detected during the transfer of the source operands needed for the execution of the instruction.

Group A instructions are only stalled when the FIFO is full, in which case the CPU will wait before sending the next instruction. Group B instructions can begin execution while some entries are still in the FIFO, but cannot complete before the FIFO is empty (i.e., before all previous instructions are completed). Non-floating-point instructions cannot begin execution until the FIFO is empty. When a normal completion indication is received, the instruction address at the bottom of the FIFO is dropped. If a trap indication is received and the FIFO is not empty, the instruction address at the bottom of the FIFO is copied to the PC register and the floating-point exception is serviced. The remaining entries in the FIFO are discarded.

A floating-point exception may be received and serviced at any time after the CPU has sent the ID and opcode for the first instruction and until the FPC has signalled completion for the last instruction.

Other exceptions may occur while the FIFO is not empty. This may be the case when an interrupt is received or a translation exception is detected in the access of an operand needed for the execution of the next floating-point instruction. These exceptions will be processed as soon as the FIFO becomes empty, and after any floating-point exception has been acknowledged.

In the event of a non-restartable bus error, the acknowledge will occur immediately. The CPU will flush the internal FIFO and will reset the FPC by performing a dummy read of the slave status word. This operation is performed for both the regular and pipelined floating-point protocol and regardless of whether any floating-point instruction is pending in the FPC instruction queue.

The CPU may cancel the last instruction sent to the FPC by sending another ID and opcode, before the last source operand for that instruction has been sent. *Figure 3-8* shows the instruction flow in pipelined floating-point mode.

3.1.4.4 Floating Point Instructions

Table 3-1 gives the protocols followed for each Floating Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Appendix A.

The Operand class columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see Instruction Set Reference Manual).

The Operand Issued columns show the sizes of the operands issued to the Floating Point Unit by the CPU. "D" indicates a 32-bit Double Word. "i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). "f" indicates that the instruction specifies a Floating Point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR-Bits-Affected column indicates which PSR bits, if any, are updated from the Slave Processor Status Word *(Figure 3-7)*.

Any operand indicated as being of type "f" will not cause a transfer if the Register addressing mode is specified. This is because the Floating Point Registers are physically on the Floating Point Unit and are therefore available without CPU assistance.

3.1.4.5 Custom Slave Instructions

Provided in the NS32532 is the capability of communicating with a user-defined, "Custom" Slave Processor. The instruction set provided for a Custom Slave Processor defines the instruction formats, the operand classes and the communication protocol. Left to the user are the interpretations of the Op Code fields, the programming model of the Custom Slave and the actual types of data transferred. The protocol specifies only the size of an operand, not its data type.

Table 3-2 lists the relevant information for the Custom Slave instruction set. The designation "c" is used to represent an operand which can be a 32-bit ("D") or 64-bit ("Q") quantity in any format; the size is determined by the suffix on the mnemonic. Similarly, an "i" indicates an integer size (Byte, Word, Double Word) selected by the corresponding mnemonic suffix.

Any operand indicated as being of type "c" will not cause a transfer if the register addressing mode is specified. It is assumed in this case that the slave processor is already holding the operand internally.

For the instruction encodings, see Appendix A.

3.2 EXCEPTION PROCESSING

Exceptions are special events that alter the sequence of instruction execution. The CPU recognizes three basic types of exceptions: interrupts, traps and bus errors.

An interrupt occurs in response to an event signalled by activating the $\overline{\text{NMI}}$ or $\overline{\text{INT}}$ input signals. Interrupts are typically requested by peripheral devices that require the CPU's attention.

Traps occur as a result either of exceptional conditions (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., supervisor call instruction).

A bus error exception occurs when the $\overline{\text{BER}}$ signal is activated during an instruction fetch or data transfer required by the CPU to execute an instruction.

When an exception is recognized, the CPU saves the PC, PSR and optionally the MOD register contents on the interrupt stack and then it transfers control to an exception service procedure.

Details on the operations performed in the various cases by the CPU to enter and exit the exception service procedure are given in the following sections.

It is to be noted that the reset operation is not treated here as an exception. Even though, like any exception, it alters the instruction execution sequence.

The reason being that the CPU handles reset in a significantly different way than it does for exceptions.

Refer to Section 3.5.3 for details on the reset operation.

3.0 Functional Description (Continued)

TABLE 3-1. Floating Point Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op.2	none
SUBf	read.f	rmw.f	f	f	f to Op.2	none
MULf	read.f	rmw.f	f	f	f to Op.2	none
DIVf	read.f	rmw.f	f	f	f to Op.2	none
MOVf	read.f	write.f	f	N/A	f to Op.2	none
ABSf	read.f	write.f	f	N/A	f to Op.2	none
NEGf	read.f	write.f	f	N/A	f to Op.2	none
CMPf	read.f	read.f	f	f	N/A	N, Z, L
FLOORfi	read.f	write.i	f	N/A	i to Op.2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op.2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op.2	none
MOVFL	read.F	write.L	F	N/A	L to Op.2	none
MOVLF	read.L	write.F	L	N/A	F to Op.2	none
MOVif	read.i	write.f	i	N/A	f to Op.2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op.2	none
POLYf	read.f	read.f	f	f	f to F0	none
DOTf	read.f	read.f	f	f	f to F0	none
SCALBf	read.f	rmw.f	f	f	f to Op.2	none
LOGBf	read.f	write.f	f	N/A	f to Op.2	none
SQRTf	read.f	write.f	f	N/A	f to Op.2	none
MACf	read.f	read.f	f	f	f to F1	none

TABLE 3-2. Custom Slave Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
CCAL0c	read.c	rmw.c	c	c	c to Op.2	none
CCAL1c	read.c	rmw.c	c	c	c to Op.2	none
CCAL2c	read.c	rmw.c	c	c	c to Op.2	none
CCAL3c	read.c	rmw.c	c	c	c to Op.2	none
CMOV0c	read.c	write.c	c	N/A	c to Op.2	none
CMOV1c	read.c	write.c	c	N/A	c to Op.2	none
CMOV2c	read.c	write.c	c	N/A	c to Op.2	none
CMOV3c	read.c	write.c	c	N/A	c to Op.2	none
CCMP0c	read.c	read.c	c	c	N/A	N,Z,L
CCMP1c	read.c	read.c	c	c	N/A	N,Z,L
CCV0ci	read.c	write.i	c	N/A	i to Op.2	none
CCV1ci	read.c	write.i	c	N/A	i to Op.2	none
CCV2ci	read.c	write.i	c	N/A	i to Op.2	none
CCV3ic	read.i	write.c	i	N/A	c to Op.2	none
CCV4DQ	read.D	write.Q	D	N/A	Q to Op.2	none
CCV5QD	read.Q	write.D	Q	N/A	D to Op.2	none
LCSR	read.D	N/A	D	N/A	N/A	none
SCSR	N/A	write.D	N/A	N/A	D to Op.2	none
LCR*	read.D	N/A	D	N/A	N/A	none
SCR*	write.D	N/A	N/A	N/A	D to Op.1	none

Note:
D = Double Word
i = Integer size (B,W,D) specified in mnemonic.
c = Custom size (D:32 bits or Q:64 bits) specified in mnemonic.
* = Privileged instruction: will trap if CPU is in User Mode.
N/A = Not Applicable to this Instruction.

3.0 Functional Description (Continued)

3.2.1 Exception Acknowledge Sequence

When an exception is recognized, the CPU goes through three major steps:

1) Adjustment of Registers. Depending on the source of the exception, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR) and the currently-selected Stack Pointer (SP). A copy of the PSR is made, and the PSR is then set to reflect Supervisor Mode and selection of the Interrupt Stack. Trap (TRC) and Trap (OVF) are always disabled. Maskable interrupts are also disabled if the exception is caused by an interrupt, Trap (DBG), Trap (ABT) or bus error.
2) Vector Acquisition. A vector is either obtained from the data bus or is supplied internally by default.
3) Service Call. The CPU performs one of two sequences common to all exceptions to complete the acknowledge process and enter the appropriate service procedure. The selection between the two sequences depends on whether the Direct-Exception mode is disabled or enabled.

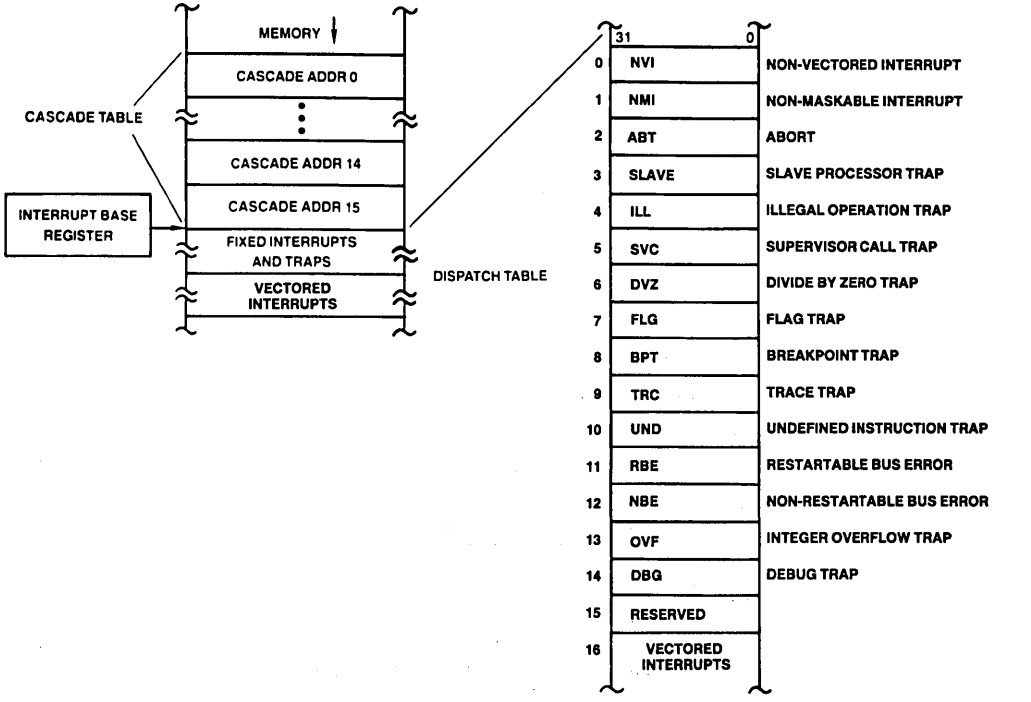
Direct-Exception Mode Disabled

The Direct-Exception mode is disabled while the DE bit in the CFG register is 0 (Section 2.1.4). In this case the CPU first pushes the saved PSR copy along with the contents of the MOD and PC registers on the interrupt stack. Then it reads the double-word entry from the Interrupt Dispatch table at address 'INTBASE + vector × 4'. See *Figures 3-9* and *3-10*. The CPU uses this entry to call the exception service procedure, interpreting the entry as an external procedure descriptor.

A new module number is loaded into the MOD register from the least-significant word of the descriptor, and the static-base pointer for the new module is read from memory and loaded into the SB register. Then the program-base pointer for the new module is read from memory and added to the most-significant word of the module descriptor, which is interpreted as an unsigned value. Finally, the result is loaded into the PC register.

Direct-Exception Mode Enabled

The Direct-Exception mode is enabled when the DE bit in the CFG register is set to 1. In this case the CPU first pushes the saved PSR copy along with the contents of the PC register on the Interrupt Stack. The word stored on the Interrupt Stack between the saved PSR and PC register is reserved for future use; its contents are undefined. The CPU then reads the double-word entry from the Interrupt Dispatch Table at address 'INTBASE + vector × 4'. The CPU uses this entry to call the exception service procedure, interpreting the entry as an absolute address that is simply loaded into the PC register. *Figure 3-11* provides a pictorial of the acknowledge sequence. It is to be noted that while the



TL/EE/9354–13

FIGURE 3-9. Interrupt Dispatch Table

3.0 Functional Description (Continued)

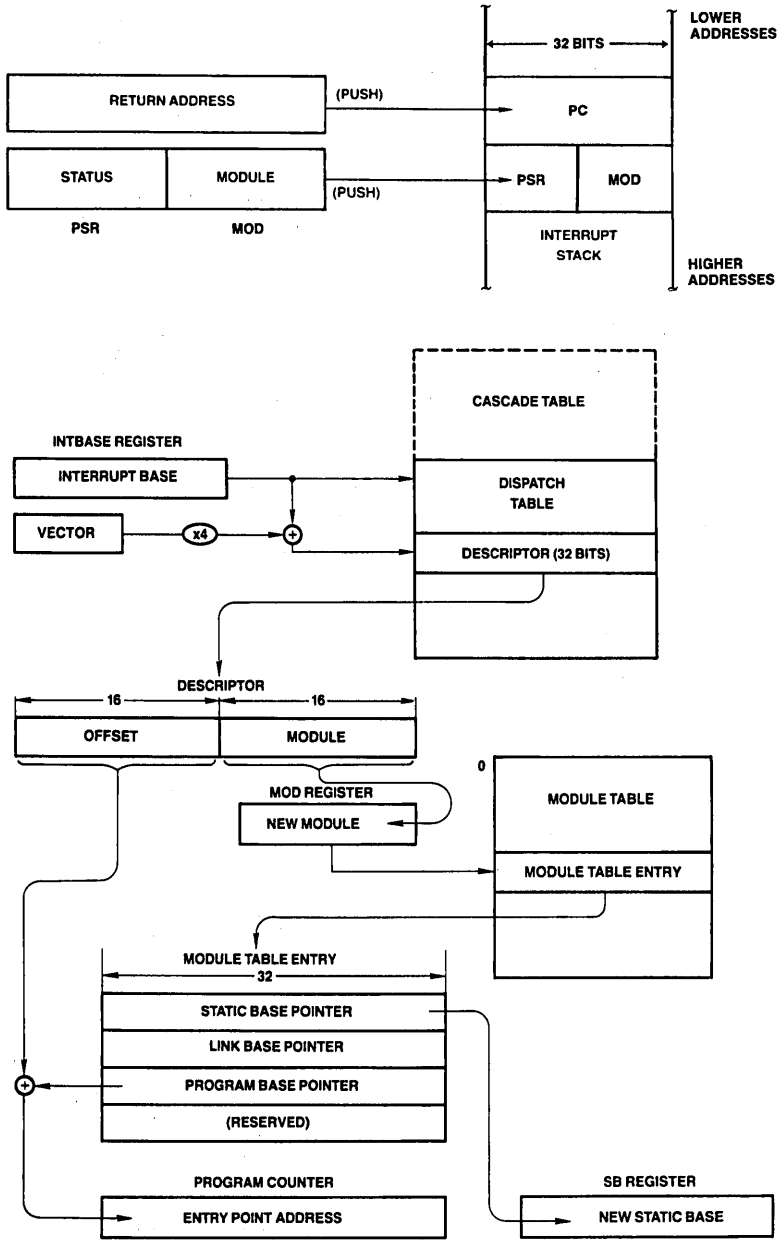


FIGURE 3-10. Exception Acknowledge Sequence. Direct-Exception Mode Disabled.

3.0 Functional Description (Continued)

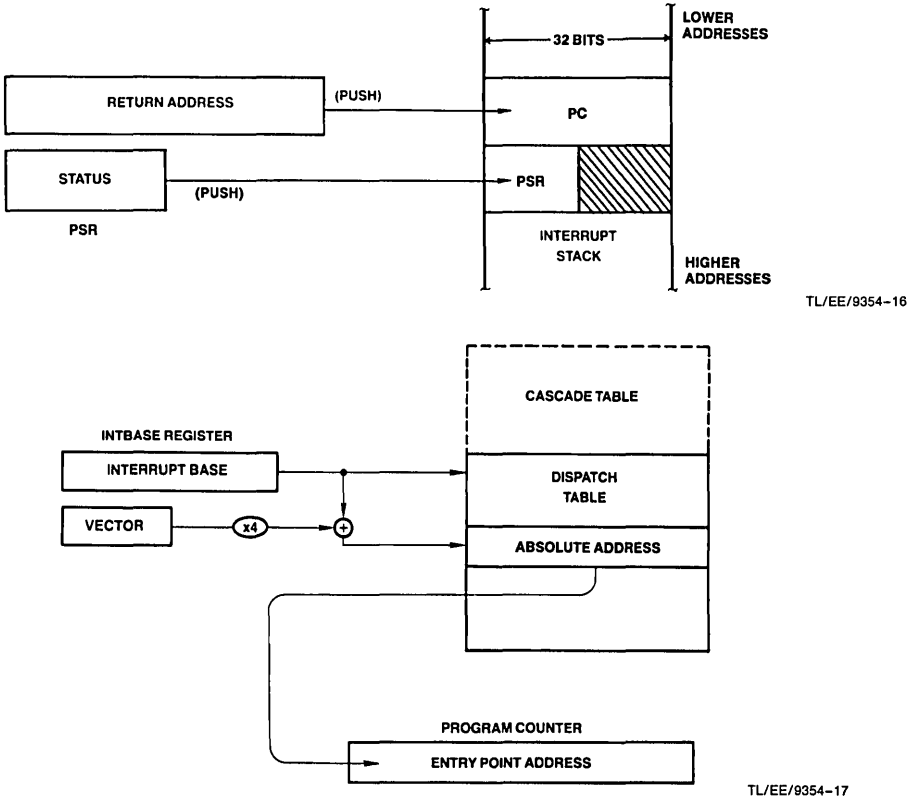


FIGURE 3-11. Exception Acknowledge Sequence. Direct-Exception Mode Enabled.

direct-exception mode is enabled, the CPU can respond more quickly to interrupts and other exceptions because fewer memory references are required to process an exception. The MOD and SB registers, however, are not initialized before the CPU transfers control to the service procedure. Consequently, the service procedure is restricted from executing any instructions, such as CXP, that use the contents of the MOD or SB registers in effective address calculations.

3.2.2 Returning from an Exception Service Procedure

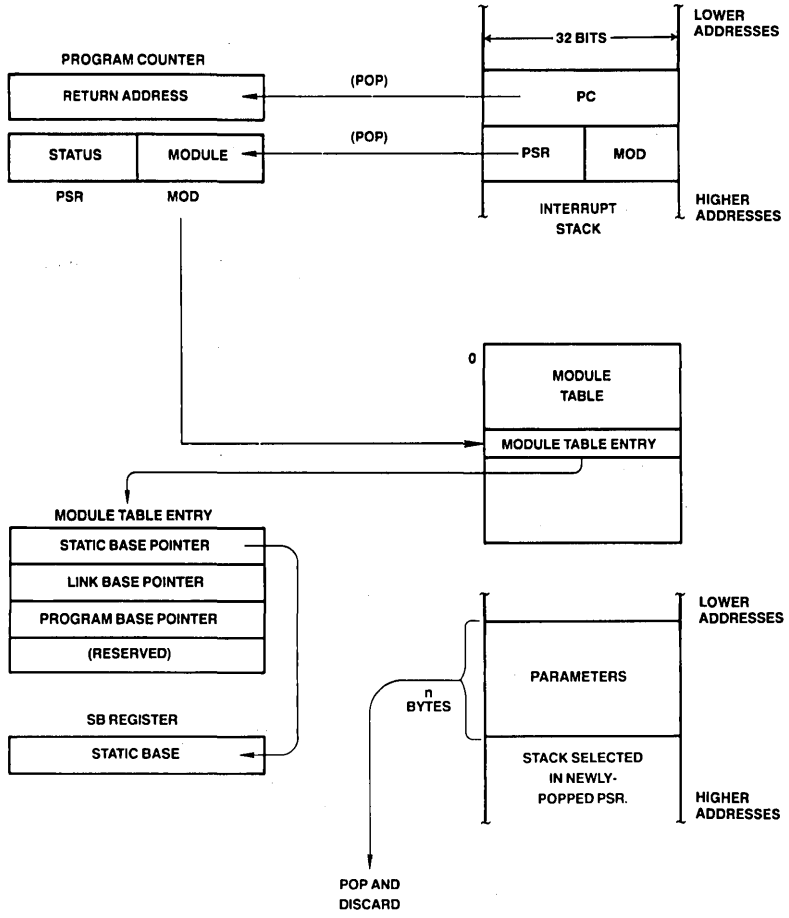
To return control to an interrupted program, one of two instructions can be used: RETT (Return from Trap) and RETI (Return from Interrupt).

RETT is used to return from any trap, non-maskable interrupt or bus error service procedure. Since some traps are often used deliberately as a call mechanism for supervisor mode procedures, RETT can also adjust the Stack Pointer (SP) to discard a specified number of bytes from the original stack as surplus parameter space.

RETI is used to return from a maskable interrupt service procedure. A difference of RETT, RETI also informs any external interrupt control units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not discard parameters from the stack.

Both of the above instructions always restore the Program Counter (PC) and the Processor Status Register from the interrupt stack. If the Direct-Exception mode is disabled, they also restore the MOD and SB register contents. *Figures 3-12* and *3-13* show the RETT and RETI instruction flows when the Direct-Exception mode is disabled.

3.0 Functional Description (Continued)



TL/EE/9354-18

FIGURE 3-12. Return from Trap (RETT n) Instruction Flow. Direct-Exception Mode Disabled.

3.2.3 Maskable Interrupts

The $\overline{INT}$ pin is a level-sensitive input. A continuous low level is allowed for generating multiple interrupt requests. The input is maskable, and is therefore enabled to generate interrupt requests only while the Processor Status Register I bit is set. The I bit is automatically cleared during service of an $\overline{INT}$, $\overline{NMI}$, Trap (DBG), Trap (ABT) or Bus Error request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

The $\overline{INT}$ pin may be configured via the SETCFG instruction as either Non-Vectored (CFG Register bit I = 0) or Vectored (bit I = 1).

3.2.3.1 Non-Vectored Mode

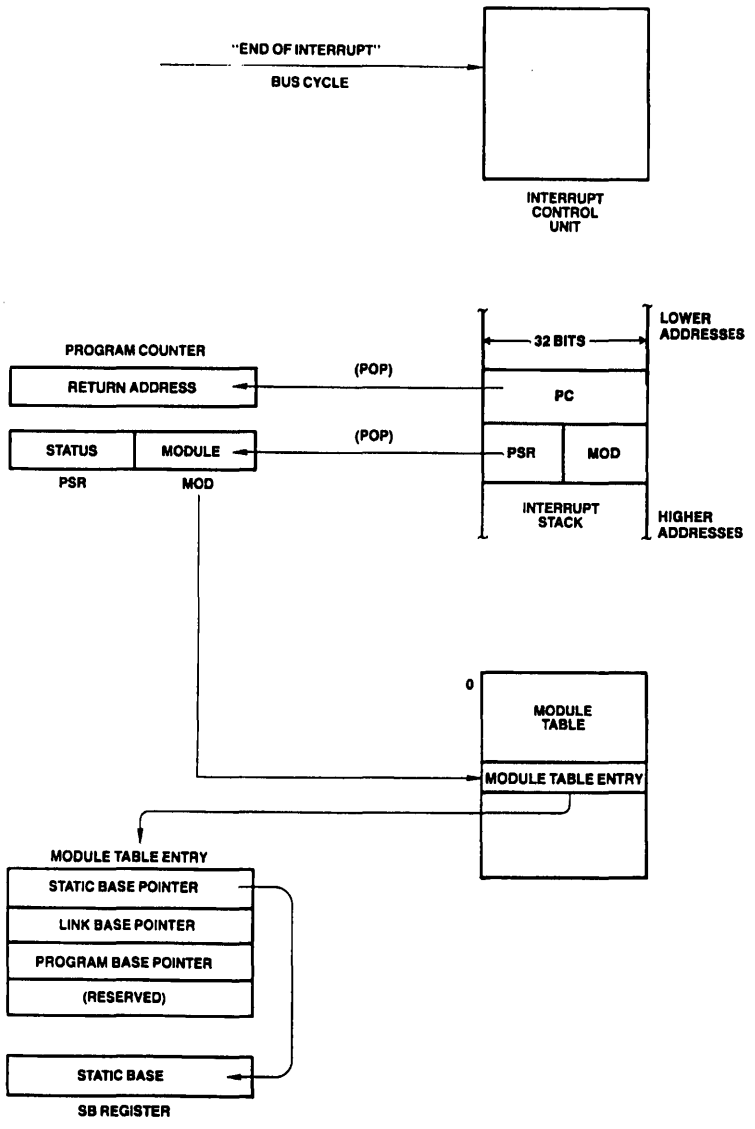
In the Non-Vectored mode, an interrupt request on the $\overline{INT}$ pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.

3.2.3.2 Vectored Mode: Non-Cascaded Case

In the Vectored mode, the CPU uses an Interrupt Control Unit (ICU) to prioritize many interrupt requests. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an "Interrupt Acknowledge, Master" bus cycle (Section 3.5.4.6) reading a vector value from the low-order byte of the Data Bus. This vector is then used as an index into the Dispatch Table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) instruction, which performs an End of Interrupt bus cycle, informing the ICU that it may re-prioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it needs also to inform a Cascaded ICU (see below).

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range of 0 through 127; that is, they must be positive numbers in eight bits. By providing

3.0 Functional Description (Continued)



TL/EE/9354-19

FIGURE 3-13. Return from Interrupt (RETI) Instruction Flow. Direct-Exception Mode Disabled.

3.0 Functional Description (Continued)

a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU (see below).

3.2.3.3 Vectored Mode: Cascaded Case

In order to allow more levels of interrupt, provision is made in the CPU to transparently support cascading. Note that the Interrupt output from a Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin. Refer to the ICU data sheet for details.

In a system which uses cascading, two tasks must be performed upon initialization:

1) For each Cascaded ICU in the system, the Master ICU must be informed of the line number on which it receives the cascaded requests.
2) A Cascade Table must be established in memory. The Cascade Table is located in a NEGATIVE direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

Figure 3-9 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0 to 15) and subtract 16 from it, giving an index in the range -16 to -1. Multiply this value by 4, and add the resulting negative number to the contents of the INTBASE Register. The 32-bit entry at this address must be set to the address of the Hardware Vector Register of the Cascaded ICU. This is referred to as the "Cascade Address."

Upon receipt of an interrupt request from a Cascaded ICU, the Master ICU interrupts the CPU and provides the negative Cascade Table index instead of a (positive) vector number. The CPU, seeing the negative value, uses it as an index into the Cascade Table and reads the Cascade Address from the referenced entry. Applying this address, the CPU performs an "Interrupt Acknowledge, Cascaded" bus cycle, reading the final vector value. This vector is interpreted by the CPU as an unsigned byte, and can therefore be in the range of 0 through 255.

In returning from a Cascaded interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any Maskable Interrupt. The CPU performs an "End of Interrupt, Master" bus cycle, whereupon the Master ICU again provides the negative Cascade Table index. The CPU, seeing a negative value, uses it to find the corresponding Cascade Address from the Cascade Table. Applying this address, it performs an "End of Interrupt, Cascaded" bus cycle, informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the interrupt mask register of the interrupt controller.

However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the $\overline{INT}$ line before the ICU deasserted it. This could cause the ICU to provide an invalid vector. To avoid this problem the above operation should be performed with the CPU interrupt disabled.

3.2.4 Non-Maskable Interrupt

The Non-Maskable Interrupt is triggered whenever a falling edge is detected on the $\overline{NMI}$ pin. The CPU performs an "Interrupt Acknowledge, Master" bus cycle (Section 3.5.4.6) when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is $FFFFFF00_{16}$. The vector value used for the Non-Maskable Interrupt is taken as 1, regardless of the value read from the bus.

The service procedure returns from the Non-Maskable Interrupt using the Return from Trap (RETT) instruction. No special bus cycles occur on return.

3.2.5 Traps

Traps are processing exceptions that are generated as direct results of the execution of an instruction.

The return address saved on the stack by any trap except Trap (TRC) and Trap (DBG) is the address of the first bye of the instruction during which the trap occurred.

When a trap is recognized, maskable interrupts are not disabled except for the case of Trap (ABT) and Trap (DBG).

There are 11 trap conditions recognized by the NS32532 as described below.

Trap (ABT): An abort trap occurs when an invalid page table entry or a protection level violation is detected for any of the memory references required to execute an instruction.

Trap (SLAVE): An exceptional condition was detected by the Floating Point Unit or another Slave Processor during the execution of a Slave Instruction. This trap is requested via the Status Word returned as part of the Slave Processor Protocol (Section 3.1.4.1).

Trap (ILL): Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR bit U = 1).

Trap (SVC): The Supervisor Call (SVC) instruction was executed.

Trap (DVZ): An attempt was made to divide an integer by zero. (The FPU trap is used for Floating Point division by zero.)

Trap (FLG): The FLAG instruction detected a "1" in the PSR F bit.

Trap (BPT): The Breakpoint (BPT) instruction was executed.

Trap (TRC): The instruction just completed is being traced. Refer to Section 3.3.1 for details.

Trap (UND): An Undefined-Instruction trap occurs when an attempt to execute an instruction is made and one or more of the following conditions is detected:

1. The instruction is undefined. Refer to Appendix A for a description of the codes that the CPU recognizes to be undefined.
2. The instruction is a floating point instruction and the F-bit in the CFG register is 0.
3. The instruction is a custom slave instruction and the C-bit in the CFG register is 0.
4. The instruction is a memory-management instruction and the M-bit in the CFG register is 0.
5. An LMR or SMR instruction is executed while the U-flag in the PSR is 0 and the most significant bit of the instruction's short field is 0.
6. The reserved general adressing mode encoding (10011) is used.
7. Immediate addressing mode is used for an operand that has access class different from read.

3.0 Functional Description (Continued)

8. Scaled Indexing is used and the basemode is also Scaled Indexing.
9. The instruction is a floating-point or custom slave instruction that the FPU or custom slave detects to be undefined. Refer to Section 3.1.4.1 for more information.

Trap (OVF): An Integer-Overflow trap occurs when the V-bit in the PSR register is set to 1 and an Integer-Overflow condition is detected during the execution of an instruction. An Integer-Overflow condition is detected in the following cases:

1. The F-flag is 1 following execution of an ADDi, ADDQi, ADDCi, SUBi, SUBCi, NEGi, ABSi, or CHECKi instruction.
2. The product resulting from a MULi instruction cannot be represented exactly in the destination operand's location.
3. The quotient resulting from a DEIi, DIVi, or QUOi instruction cannot be represented exactly in the destination operand's location.
4. The result of an ASHi instruction cannot be represented exactly in the destination operand's location.
5. The sum of the 'INC' value and the 'INDEX' operand for an ACBi instruction cannot be represented exactly in the index operand's location.

Trap (DBG): A debug trap occurs when one or more of the conditions selected by the settings of the bits in the DCR register is detected. This trap can also be requested by activating the input signal $\overline{DBG}$. Refer to Section 3.3.2 for more information.

Note 1: Following execution of the WAIT instruction, then a Trap (DBG) can be pending for a PC-match condition. In such an event, the Trap (DBG) is processed immediately.

Note 2: If an attempt is made to execute a memory-management instruction while in User-Mode and the M-bit in the CFG register is 0, then Trap (UND) occurs.

Note 3: If an attempt is made to execute a privileged custom instruction while in User-Mode and the C-bit in the CFG register is 0, then Trap (UND) occurs.

Note 4: While operating in User-Mode, if an attempt is made to execute a privileged instruction with an undefined use of a general addressing mode (either the reserved encoding is used or else scaled-index or immediate modes are incorrectly used), the Trap (UND) occurs.

Note 5: If an undefined instruction or illegal operation is detected, then no data references are performed for the instruction.

Note 6: For certain instructions that are relatively long to execute, such as DEID, the CPU checks for pending interrupts during execution of the instruction. In order to reduce interrupt latency, the NS2532 can suspend executing the instruction and process the interrupt. Refer to Section B.5 in Appendix B for more information about recognizing interrupts in this manner.

3.2.6 Bus Errors

A bus error exception occurs when the $\overline{BER}$ signal is asserted in response to an instruction fetch or data transfer that is required to execute an instruction.

Two types of bus errors are recognized: Restartable and Non-Restartable. Restartable bus errors are recognized during read bus cycles, except for MMU read cycles (from Page Tables) needed to translate the address of a result being stored into memory. All other bus errors are non-restartable.

The CPU responds to restartable bus errors by suspending the instruction that it was executing. When a non-restartable bus error is detected, the CPU responds immediately and the instruction being executed is terminated. See Section 3.1.2.3.

The PC value saved on the stack is undefined.

The NS32532 does not respond to bus errors indicated for instructions that are not executed. For example, no bus error exception occurs in response to asserting the $\overline{BER}$ signal during a bus cycle to prefetch an instruction that is not executed because the previous instruction caused a trap.

An exception to this rule occurs if the bus error is detected during an MMU write cycle to update the R-bit in a page table entry.

In this case the CPU recognizes the bus error and considers it as non-restartable even though the bus cycle that caused it belongs to a non-executed instruction.

If a bus error is detected during a data transfer required for the processing of another exception or during the ICU read cycle of a RETI instruction, then the CPU considers it as a fatal bus error and enters the 'HALTED' state.

Note 1: If the address and control signals associated with the last bus cycle that caused a bus error are latched by external hardware, then the information they provide can be used by the service procedure for restartable bus errors to analyze and resolve the exception recognized by the CPU. This can be accomplished because upon detecting a restartable bus error, the NS32532 stops making memory references for subsequent instructions until it determines whether the instruction that caused the bus error is executed and the exception is processed.

Note 2: When a non-restartable bus error is recognized, the service procedure must execute the CINV and LMR instructions to invalidate the on-chip caches and TLB. This is necessary to maintain coherence between them and external memory.

3.2.7 Priority Among Exceptions

The CPU checks for specific exceptions at various points while executing an instruction. It is possible that several exceptions occur simultaneously. In that event, the CPU responds to the exception with highest priority.

Figure 3-14 shows an exception processing flowchart. A non-restartable bus error is assigned highest priority and is serviced immediately regardless of the execution state of the CPU.

Before executing an instruction, the CPU checks for pending Trap (DBG), interrupts, and Trap (TRC), in that order. If a Trap (DBG) is pending, then the CPU processes that exception, otherwise the CPU checks for pending interrupts. At this point, the CPU responds to any pending interrupt requests; nonmaskable interrupts are recongized with higher priority than maskable interrupts. If no interrupts are pending, then the CPU checks the P-flag in the PSR to determine whether a Trap (TRC) is pending. If the P-flag is 1, a Trap (TRC) is processed. If no Trap (DBG), interrupt or Trap (TRC) is pending, the CPU begins executing the instruction.

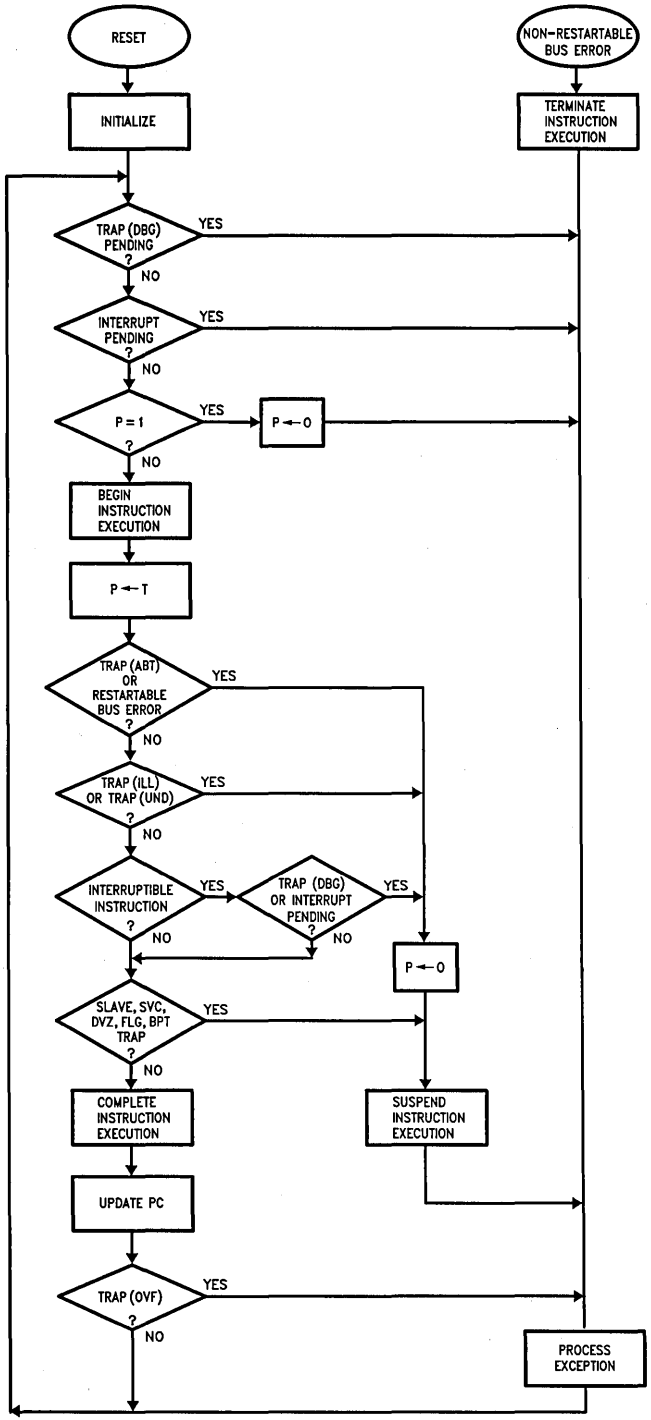
While executing an instruction, the CPU may recognize up to four exceptions:

1. trap (ABT)
2. restartable bus error
3. trap (DBG) or interrupt, if the instruction is interruptible
4. one of 7 mutually exclusive traps: SLAVE, ILL, SVC, DVZ, FLG, BPT, UND

Trap (ABT) and restartable bus error have equal priority; the CPU responds to the first one detected.

If no exception is detected while the instruction is executing, then the instruction is completed and the PC is updated to point to the next instruction. If a Trap (OVF) is detected, then it is processed at this time.

3.0 Functional Description (Continued)



TL/EE/9354–20

FIGURE 3-14. Exception Processing Flowchart

3.0 Functional Description (Continued)

While executing the instruction, the CPU checks for enabled debug conditions. If an enabled debug condition is met, a Trap (DBG) is held pending until after the instruction is completed (see Note 3). If another exception is detected before the instruction is completed, the pending Trap (DBG) is removed and the DSR register is not updated.

Note 1: Trap (DBG) can be detected simultaneously with Trap (OVF). In this event, the Trap (OVF) is processed before the Trap (DBG).

Note 2: An address-compare debug condition can be detected while processing a bus error, interrupt, or trap. In this event, the Trap (DBG) is held pending until after the CPU has processed the first exception.

Note 3: Between operations of a string instruction, the CPU responds to pending operand address compare and external debug conditions as well as interrupts. If a PC-match debug condition is detected while executing a string instruction, then Trap (DBG) is held pending until the instruction has completed.

3.2.8 Exception Acknowledge Sequences: Detailed Flow

For purposes of the following detailed discussion of exception acknowledge sequences, a single sequence called "service" is defined in *Figure 3-15*.

Upon detecting any interrupt request, trap or bus error condition, the CPU first performs a sequence dependent upon the type of exception. This sequence will include saving a copy of the Processor Status Register and establishing a vector and a return address. The CPU then performs the service sequence.

3.2.8.1 Maskable/Non-Maskable Interrupt Sequence

This sequence is performed by the CPU when the $\overline{NMI}$ pin receives a falling edge, or the $\overline{INT}$ pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or, in the case of an interruptible instruction (e.g., string instruction), at the next interruptible point during its execution.

1. If an interruptible instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P bit.
   b. Set "Return Address" to the address of the first byte of the interrupted instruction.

   Otherwise, set "Return Address" to the address of the next instruction.
2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits T, V, U, S, P and I.
3. If the interrupt is Non-Maskable:
   a. Read a byte from address $FFFFFF00_{16}$, applying Status Code 00100 (Interrupt Acknowledge, Master). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.
4. If the interrupt is Non-Vectored:
   a. Read a byte from address $FFFFFE00_{16}$, applying Status Code 00100 (Interrupt Acknowledge, Master). Discard the byte read.
   b. Set "Vector" to 0.
   c. Go to Step 8.
5. Here the interrupt is Vectored. Read "Byte" from address $FFFFFE00_{16}$, applying Status Code 00100 (Interrupt Acknowledge, Master).
6. If "Byte" ≥ 0, then set "Vector" to "Byte" and go to Step 8.
7. If "Byte" is in the range −16 through −1, then the interrupt source is Cascaded. (More negative values are reserved for future use.) Perform the following:
   a. Read the 32-bit Cascade Address from memory. The address is calculated as INTBASE + 4* Byte.
   b. Read "Vector," applying the Cascade Address just read and Status Code 00101 (Interrupt Acknowledge, Cascaded).
8. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.2 Abort/Restartable Bus Error Sequence

1. Suspend instruction and restore the currently selected Stack Pointer to its original contents at the beginning of the instruction.
2. Clear the PSR P bit.
3. Copy the PSR into a temmporary register, then clear PSR bits T, V, U, S and I.
4. Set "Vector" to the value corresponding to the exception type:

   Abort: Vector = 2

   Restartable Bus Error: Vector = 11
5. Set "Return Address" to the address of the first byte of the suspended instruction.
6. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.3 SLAVE/ILL/SVC/DVZ/FLG/BPT/UND Trap Sequence

1. Restore the currently selected Stack Pointer and the Processor Status Register to their original values at the start of the trapped instruction.
2. Set "Vector" to the value corresponding to the trap type.

   SLAVE: Vector = 3.
   ILL: Vector = 4.
   SVC: Vector = 5.
   DVZ: Vector = 6.
   FLG: Vector = 7.
   BPT: Vector = 8.
   UND: Vector = 10.
3. If Trap (ILL) or Trap (UND)
   a. Clear the Processor Status Register P bit.
4. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits T, V, U, S and P.
5. Set "Return Address" to the address of the first byte of the trapped instruction.
6. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.4 Trace Trap Sequence

1. In the Processor Status Register (PSR), clear the P bit.
2. Copy the PSR into a temporary register, then clear PSR bits T, V, U and S.
3. Set "Vector" to 9.
4. Set "Return Address" to the address of the next instruction.
5. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.5 Integer-Overflow Trap Sequence

1. Copy the PSR into a temporary register, then clear PSR bits T, V, U, S and P.
2. Set "Vector" to 13.
3. Set "Return Address" to the address of the next instruction.

3.0 Functional Description (Continued)

4. Perform Service (Vector, Return Address), *Figure 3-15.*

3.2.8.6 Debug Trap Sequence

A debug condition can be recognized either at the next instruction boundary or, in the case of an interruptible instruction, at the next interruptible point during its execution.

1. If PC-match condition, then go to Step 3.
2. If a String instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P bit.
   b. Set "Return Address" to the address of the first byte of the instruction.
   c. Go to Step 4.
3. Set "Return Address" to the address of the next instruction.
4. Set "Vector" to 14.
5. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits T, V, U, S, P and I.
6. Perform Service (Vector, Return Address), *Figure 3-15.*

Note: In case of PC-match or address-compare on write, the Trap (DBG) may occur before the instruction is executed.

3.2.8.7 Non-Restartable Bus Error Sequence

1. Set "Vector" to 12.
2. Set "Return Address" to "Undefined".
3. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits T, V, U, S, P and I.
4. Perform a dummy read of the Slave Status Word to reset the Slave Processor.
5. Perform Service (Vector, Return Address), *Figure 3-15.*

3.3 DEBUGGING SUPPORT

The NS32532 provides serveral features to assist in program debugging.

Besides the Breakpoint (BPT) instruction that can be used to generate soft breaks, the CPU also provides instruction tracing as well as debug trap (or hardware breakpoints) capabilities. Details on these features are provided in the following sub-sections.

3.3.1 Instruction Tracing

Instruction tracing is a very useful feature that can be used during debugging to single-step through selected portions of a program. Tracing is enabled by setting the T-bit in the PSR Register. When enabled, the CPU generates a Trace Trap (TRC) after the execution of each instruction.

At the beginning of each instruction, the T bit is copied into the PSR P (Trace "Pending") bit. If the P bit is set at the end of an instruction, then the Trace Trap is activated. If any other trap or interrupt request is made during a traced instruction, its entire service procedure is allowed to complete before the Trace Trap occurs. Each interrupt and trap sequence handles the P bit for proper tracing, guaranteeing only one Trace Trap per instruction, and guaranteeing that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

Due to the fact that some instructions can clear the T and P bits in the PSR, in some cases a Trace Trap may not occur at the end of the instruction. This happens when one of the privileged instructions BICPSRW or LPRW PSR is executed.

TABLE 3-3. Summary of Exception Processing

Exception	Instruction Ending	Cleared Before Saving PSR	Cleared After Saving PSR
Restartable Bus Error	Suspended	P	TVUSI
Nonrestartable Bus Error	Terminated	Undefined	TVUS
Interrupt	Before Instruction	None/P*	TVUSPI
ABT	Suspended	P	TVUSI
ILL, UND	Suspended	P	TVUS
SLAVE, SVC, DVZ, FLG, BPT	Suspended	None	TVUSP
OVF	Completed	None	TVUSP
TRC	Before Instruction	P	TVUS
DBG	Before Instruction	None/P*	TVUSPI

***Note:** The P bit of the saved PSR is cleared in case the exception is acknowledged before the instruction is completed (e.g., interrupted string instruction). This is to avoid a mid-instruction trace trap upon return from the Exception Service Routine.

Service (Vector, Return Address):

1) Push the PSR copy onto the Interrupt Stack as a 16-bit value.

2) If Direct-Exception mode is selected, then go to step 4.

3) Push MOD Register into the Interrupt Stack as a 16-bit value.

4) Read 32-bit Interrupt Dispatch Table (IDT) entry at address 'INTBASE + vector × 4'.

5) If Direct-Exception mode is selected, then go to Step 10.

6) Move the L.S. word of the IDT entry (Module Field) into the MOD register.

7) Read the Program Base pointer from memory address 'MOD + 8', and add to it the M.S. word of the IDT entry (Offset Field), placing the result in the Program Counter.

8) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.

9) Go to Step 11.

10) Place IDT entry in the Program Counter.

11) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.

12) Serialize: Non-sequentially fetch first instruction of Exception Service Routine.

Note: Some of the Memory Accesses indicated in the service sequence may be performed in an order different from the one shown.

FIGURE 3-15. Service Sequence

3.0 Functional Description (Continued)

In other cases, it is still possible to guarantee that a Trace Trap occurs at the end of the instruction, provided that special care is taken before returning from the Trace Trap Service Procedure. In case a BICPSRB instruction has been executed, the service procedure should make sure that the T bit in the PSR copy saved on the Interrupt Stack is set before executing the RETT instruction to return to the program begin traced. If the RETT or RETI instructions have to be traced, the Trace Trap Service Procedure should set the P and T bits in the PSR copy on the Interrupt Stack that is going to be restored in the execution of such instructions.

Note: If instruction tracing is enabled while the WAIT instruction is executed, the Trap (TRC) occurs after the next interrupt, when the interrupt service procedure has returned.

3.3.2 Debug Trap Capability

The CPU recognizes three different conditions to generate a Debug Trap:

1) Address Compare

2) PC Match

3) External

These conditions can be enabled and monitored through the CPU Debug Registers.

An address-compare condition is detected when certain memory locations are either read or written. The double-word address used for the comparison is specified in the CAR Register. The address-compare condition can be separately enabled for each of the bytes in the specified double-word, under control of the CBE bits of the DCR Register. The VNP bit in the DCR controls whether virtual or physical addresses are compared. The CRD and CWR bits in the DCR separately enable the address compare condition for read and write references; the CAE bit in the DCR can be used to disable the compare-address condition independently from the other control bits. The CPU examines the address compare condition for all data reads and writes, reads of memory locations for effective address calculations, Interrupt-Acknowledge and End-of-Interrupt bus cycles, and memory references for exception processing. An address-compare condition is not detected for MMU references to Page Table Entries.

The PC-match condition is detected when the address of the instruction equals the value specified in the BPC register. The PC-match condition is enabled by the PCE bit in the DCR.

Detection of address-compare and PC-match conditions is enabled for User and Supervisor Modes by the UD and SD bits in the DCR. The DEN-bit can be used to disable detection of these two conditions independently from the other control bits.

An external condition is recognized whenever the $\overline{\text{DBG}}$ signal is activated.

When the CPU detects an address-compare or PC-match condition while executing an instruction or processing an exception, then Trap (DBG) occurs if the TR bit in the DCR is 1. When an external debug condition is detected, Trap (DBG) occurs regardless of the TR bit. The cause of the Trap (DBG) is indicated in the DSR Register.

When an address-compare or PC-match condition is detected while executing an instruction, the CPU asserts the $\overline{\text{BP}}$ signal at the beginning of the next instruction, synchronously with $\overline{\text{PFS}}$. If the instruction is not completed because a higher priority trap (i.e., ABORT) is detected, the $\overline{\text{BP}}$ signal may or may not be asserted.

Note 1: The assertion of $\overline{\text{BP}}$ is not affected by the setting of the TR bit in the DCR register.

Note 2: While executing the MOVUS and MOVSU instructions, the compare-address condition is enabled for the User space memory reference under control of the UD-bit in the DCR.

Note 3: When the LPRi instruction is executed to load a new value into the BPC, CAR or DCR, it is undefined whether the address-compare and PC-match conditions, in effect while executing the instruction, are detected under control of the old or new contents of the loaded register. Therefore, any LPRi instruction that alters the control of the address-compare or PC-match conditions should use register or immediate addressing mode for the source operand.

3.4 ON-CHIP CACHES

The NS32532 provides three on-chip caches: the Instruction Cache (IC), the Data Cache (DC) and the Translation Look-aside Buffer (TLB).

The first two are used to hold the contents of frequently used memory locations, while the TLB holds address-translation information.

The IC and DC can be individually enabled by setting appropriate bits in the CFG Register (See Section 2.1.4); the TLB is automatically enabled when address-translation is enabled.

The CPU also provides a locking feature that allows the contents of the IC and DC to be locked to specific memory locations. This is accomplished by setting the LIC and LDC bits in the CFG register.

Cache locking can be successfully used in real-time applications to guarantee fast access to critical instruction and data areas.

Details on the organization and function of each of the caches are provided in the following sections.

Note: The size and organization of the on-chip caches may change in future Series 32000 microprocessors. This however, will not affect software compatibility.

3.4.1 Instruction Cache (IC)

The basic structure of the instruction cache (IC) is shown in *Figure 3-16*.

The IC stores 512 bytes of code in a direct-mapped organization with 32 sets. Direct-mapped means that each set contains only one block, thus each memory location can be loaded into the IC in only one place.

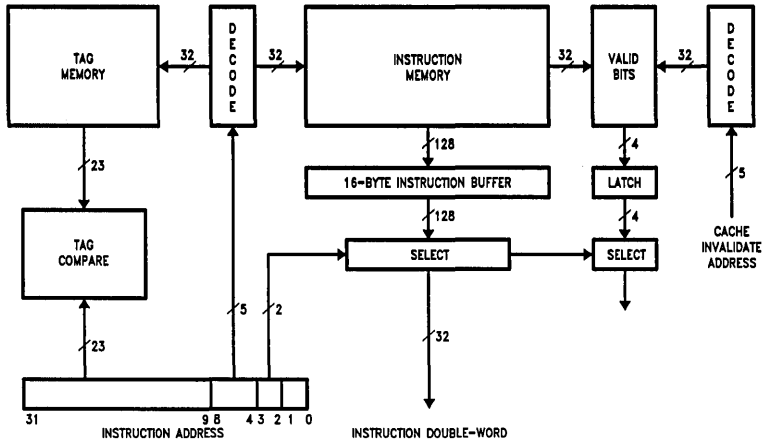
Each block contains a 23-bit tag, which holds the most-significant bits of the physical address for the locations stored in the block, along with 4 double-words and 4 validity bits (one for each double-word).

A 4-double-word instruction buffer is also provided, which is loaded either from a selected cache block or from external memory. Instructions are read from this buffer by the loader unit and transferred to an 8-byte instruction queue.

The IC may or may not be enabled to cache an instruction being fetched by the CPU. It is enabled when the IC bit in the CFG Register is set to 1 and either the address translation is disabled or the CI bit in the Level-2 PTE used to translate the virtual address of the instruction is set to 0.

If the IC is disabled, the CPU bypasses it during the instruction fetch and its contents are not affected. The instruction is read directly from external memory into the instruction buffer.

3.0 Functional Description (Continued)



TL/EE/9354-21

FIGURE 3-16. Instruction Cache Structure

When the IC is enabled, the instruction address bits 4 to 8 are used to select the IC set where the instruction may be stored. The tag corresponding to the single block in the set is compared with the 23 most-significant bits of the instruction's physical address. The 4 double-words in this block are loaded into the instruction buffer and the 4 validity bits are also retrieved. Bits 2 and 3 of the instruction's physical address select one of these double-words and the associated validity bit.

If the tag matches and the selected double-word is valid, a cache 'hit' occurs and the double-word is directly transferred to the instruction queue for decoding; otherwise a cache 'miss' will result.

In the latter case, if the cache is not locked, the CPU will take the following actions.

First, if the tag of the selected block does not match, the tag is loaded with the 23 most-significant bits of the instruction address and all the validity bits are cleared. Then, the instruction is read from external memory into the instruction buffer.

If the CIIN input signal is not active during the fetching of the missing instruction, then the IC is updated and the instruction double-words fetched from memory are stored into it with the validity bits set.

If the cache is locked, its contents are not affected, as the CPU reads the missing instruction from external memory.

Whenever the CPU accesses external memory, whether or not the IC is enabled, it always fetches instruction double-words in a non-wrap-around fashion. Refer to Sections 3.5.4.3 and 3.5.6 for more information.

The contents of the instruction cache can be invalidated by software through the CINV instruction or by hardware through the appropriate cache invalidation input signals. Clearing the IC bit in the CFG Register also invalidates the instruction cache. Refer to Sections 3.5.10 and C.3 for details.

Note: If the IC is enabled for a certain instruction and a 'miss' occurs due to a tag mismatch, the CPU will clear all the validity bits of the selected tag before fetching the instruction from external memory. If the CIIN input signal is activated during the fetching of that instruction, the validity bits are not set and the IC is not updated.

3.4.2 Data Cache (DC)

The Data Cache (DC) stores 1,024 bytes of data in a two-way set associative organization as shown in *Figure 3-17*.

Each of the 32 sets has 2 cache blocks. Each block contains a 23-bit tag, which holds the most-significant bits of the physical address for the locations stored in the block, along with 4 double-words and 4 validity bits (one for each double-word).

The DC is enabled for a data read when all of the following conditions are satisfied.

- The DC bit in the CFG Register is set to 1.
- Either the address translation is disabled or the CI bit in the Level-2 PTE used to translate the virtual address of the data reference is set to 0.
- The reference is not an interlocked read resulting from executing a CBITI or SBITI instruction.

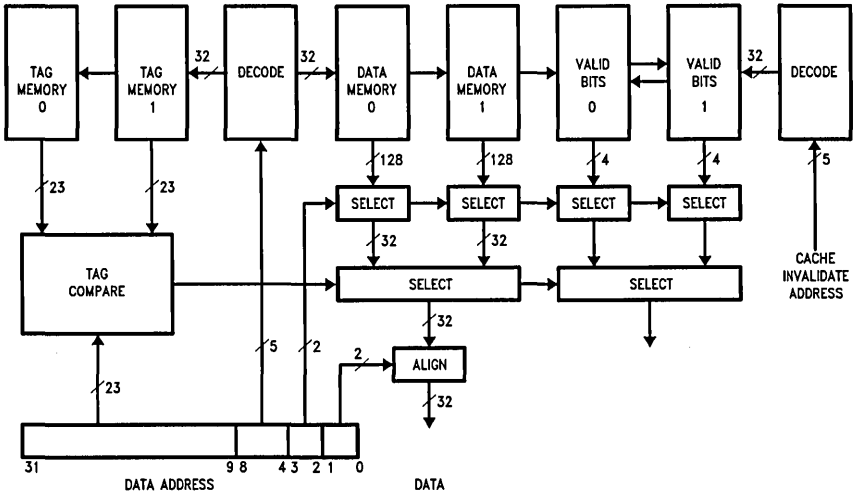
If the DC is disabled, the CPU bypasses it during the data read and its contents are not affected. The data is read directly from external memory. The DC is also bypassed for MMU reads from Page Table entries during address translation and for Interrupt-Acknowledge and End-of-Interrupt bus cycles.

When the DC is enabled for a data read, the address bits 4 to 8 are used to select the DC set where the data may be stored.

The tags corresponding to the two blocks in the set are compared to the 23 most-significant bits of the physical address. Bits 2 and 3 of the address select one double-word in each block and the associated validity bit.

If one of the tag matches and the selected double-word in the corresponding block is valid, a cache 'hit' occurs and the data is used to execute the instruction; otherwise a cache 'miss' will result. In the latter case, if the cache is not locked, the CPU will take the following actions.

3.0 Functional Description (Continued)



TL/EE/9354-22

FIGURE 3-17. Data Cache Structure

First, if the tag of either block in the set matches the data address, that block is selected for updating. Otherwise, if neither tag matches, then the least recently used block is selected; its tag is loaded with the 23 most-significant bits of the data address, and all the validity bits are cleared.

Then, the data is read from external memory; up to 4 double-word bits are read into the cache in a wrap-around fashion. Refer to Sections 3.5.4.3 and 3.5.6 for more information.

If the CIIN and $\overline{\text{IODEC}}$ input signals are both inactive during the bus cycles performed to read the missing data, then the DC is updated, as each double-word is read from memory, and the corresponding validity bit is set. If the cache is locked, its contents are not affected, as the CPU reads the missing data from external memory.

The DC is enabled for a data write whenever the DC bit in the CFG Register is set to 1, including interlocked writes resulting from executing the CBITI and SBITI instructions, and MMU writes to Page Table entries during address translation.

The DC does not use write allocation. This means that, during a write, if a cache 'hit' occurs, the DC is updated, otherwise it is unaffected. The data is always written through to external memory.

The contents of the data cache can be invalidated by software through the CINV instruction or by hardware through the appropriate cache invalidation input signals. Clearing the DC bit in the CFG Register also invalidates the data cache. Refer to Sections 3.5.10 and C.3 for details.

Note: If the DC is enabled for a certain data reference and a "miss" occurs due to tag mismatch, the CPU will clear all the validity bits for the least recently used tag before reading the data from external memory. If either CIIN or $\overline{\text{IODEC}}$ are activated during the data read bus cycles, the validity bits are not set and the DC is not updated.

3.4.3 Cache Coherence Support

The NS32532 provides several mechanisms for maintaining coherence between the on-chip caches and external memory. In software, the use of caches can be inhibited for individual pages using the CI-bit in the level-2 Page Table Entries. The CINV instruction can be executed to invalidate entriely the Instruction Cache and/or Data Cache; the CINV instruction can also be executed to invalidate a single 16-byte block in either or both caches.

In hardware, the use of the caches can be inhibited for individual locations using the CIIN input signal. A cache invalidation request can cause the entire Instruction Cache and/or Data Cache to be invalidated; a cache invalidation request can also cause invalidation of a single set in either or both caches. Refer to Section 3.5.7 for more information.

An external "Bus Watcher" circuit can also be used to help maintain cache coherence. The Bus Watcher observes the CPU's bus cycles to maintain a copy of the on-chip cache tags while also monitoring writes to main memory by DMA controllers and other microprocessors in the system. When the Bus Watcher detects that a location in one of the on-chip caches has been modified in main memory, it issues an invalidation request to the CPU. The CPU provides the necessary information on the system interface to help maintain an external copy of the on-chip tags.

The status codes differentiate between instruction fetches and data reads.

The set, affected during the bus access (if CIOUT is low), as well as the tag can be determined from the address bits A4 through A8 and A9 through A31 respectively.

During a data read the CPU also indicates, by means of the CASEC signal, which block in the set is being updated.

Whenever a CINV instruction is executed, the operation code and operand appear on the system interface using slave processor bus cycles. Thus, invalidations of the on-chip caches by software can be monitored externally.

Note, however, that the software is responsible for communicating to the external circuitry the values of the cache enable and lock bits in the CFG Register, since the CPU does not generate any special cycle (e.g., Slave Cycle) when the CFG Register is loaded.

3.0 Functional Description (Continued)

3.4.4 Translation Look-aside Buffer (TLB)

The Translation Look-aside Buffer is an on-chip fully associative memory. It provides direct virtual to physical mapping for 64 pages, thus minimizing the time needed to perform the address translation.

The efficiency of the on-chip MMU is greatly increased by the TLB, which bypasses the much longer Page Table lookup in over 99% of the accesses made by the CPU.

Entries in the TLB are allocated and replaced automatically; the operating system is not involved. The TLB entries cannot be read or written by software; however, they can be purged from it under program control.

Figure 3-18 shows a model of the TLB. Information is placed into the TLB whenever a Page Table lookup is performed. If the retrieved mapping is valid (V = 1 in both levels of the Page Tables), and the access attempted is permitted by the protection level, an entry of the TLB is loaded from the information retrieved from memory.

The on-chip MMU places the Virtual Page Number (VPN) and the Address Space qualifier (AS) into the tag portion of the TLB entry.

The value portion of the entry is loaded from the Page Tables as follows:

- The PFN field (20 bits) as well as the CI and M bits are loaded from the Level-2 Page Table Entry (PTE2).
- The PL field (2 bits) is loaded to reflect the most restrictive of the protection levels imposed by the PL fields of the Level-1 and Level-2 Page Table Entries (PTE1 and PTE2).

Not shown in the figure is an additional bit associated with each TLB entry which indicates whether the entry is valid.

Address translation can be either enabled or disabled for a memory reference. If translation is disabled, then the TLB is bypassed and the physical address is identical to the virtual address.

When translation is enabled and a virtual address needs to be translated, the high-order 20 bits (VPN) and the Address Space qualifier are compared associatively to the corresponding fields in all entries of the TLB.

For a read reference, if the tag portion of a valid TLB entry, completely matches the input values, then the value portion of the entry is used to complete the address translation and protection checking.

For a write reference, if a valid entry with a matching tag is present in the TLB, then the M bit is examined. If the M bit is 1, the value portion of the entry is used to complete the address translation and protection checking. If the M bit is 0, the entry is invalidated.

In either case, if a protection level violation is detected, a translation exception (Trap (ABT)) is generated. When no matching entry is found or a matching entry is invalidated because the M bit is 0 in a write reference, a Page Table lookup is performed. The virtual address is translated according to the algorithm given in Section 2.4.5 and the translation information is loaded into the TLB.

The recipient entry is selected by an on-chip circuit that implements a First-In-First-Out (FIFO) algorithm.

Note that for a translation to be loaded into the TLB it is necessary that the Level-1 and Level-2 Page Table Entries be valid (V bit = 1). Also, it is guaranteed that in the process of loading a TLB entry (during a Page Table lookup) the Level-1 and Level-2 R bits will be set in memory if they were not already set. For these reasons, there is no need to replicate either the V bit or the R bit in the TLB entries.

Whenever a Page Table Entry in memory is altered by software, it is necessary to purge any matching entry from the TLB, otherwise the corresponding addresses would be translated according to obsolete information. TLB entries may be selectively purged by writing a virtual address to one of the IVARn registers using the LMR instruction. The TLB entry (if any) that matches that virtual address is then purged, and its space is made available for another translation. Purging is also performed whenever an address space is remapped by altering the contents of the PTB0 or PTB1 register. When this is done, all the TLB entries corresponding to the address space mapped by that register are purged. Turning translation on or off (via the MCR TU and TS bits) does not affect the contents of the TLB.

It is possible to maintain an external copy of the valid contents of the on-chip TLB by observing the CPU's system interface during the replacement and invalidation of TLB entries. Whenever the CPU replaces a TLB entry, the page tables are accessed in external memory using bus cycles with a special Status. Because a FIFO replacement algorithm is used, it is possible to determine which entry is being replaced by using a 6-bit counter that is incremented whenever a Level-1 PTE is accessed. The contents of the new entry can be found as follows:

- VPN appears on A2 through A11 during the PTE1 and PTE2 accesses. The most-significant 10 bits appear during the PTE1 access, and the least-significant 10 bits appear during the PTE2 access.
- AS can be determined from the U/$\overline{S}$ signal during the PTE1 access.
- PFN, M and CI can be determined from the PTE2 value read on the Data Bus. PL can be determined from the most restrictive of the PTE1 and PTE2 values read on the Data Bus.

Whenever a LMR instruction is executed, the operation code and operand appear on the system interface using slave processor bus cycles. Thus, the information is available externally to determine the translation modes controlled by the MCR and to identify that a TLB entry has been invalidated.

When the PTB0 register is loaded by executing the 'LMR PTB0 src' instruction, the internal FIFO pointer is also reset to point to the first TLB entry.

Note that the contents of the TLB maintained externally include copies of all valid entries in the on-chip TLB, but the external copy may include some entries that are invalid in the on-chip TLB. For example, when the TLB is searched for a write reference and a matching entry is found with the M bit clear, then the on-chip entry is invalidated and a miss is processed. It is not possible to detect externally that the old matching entry on-chip has been invalidated.

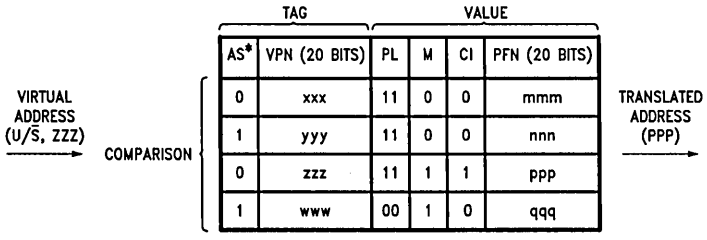
3.5 SYSTEM INTERFACE

This section provides general information on the NS32532 interface to the external world. Descriptions of the CPU requirements as well as the various bus characteristics are provided here. Details on other device characteristics including timing are given in Chapter 4.

3.5.1 Power and Grounding

The NS32532 requires a single 5-volt power supply, applied on 21 pins. The logic voltage pins (VCCL1 to VCCL6) supply

3.0 Functional Description (Continued)



*AS represents the virtual address space qualifier.

FIGURE 3-18. TLB Model

the power to the on-chip logic. The buffer voltage pins (VCCB1 to VCCB14) supply the power to the output drivers of the chip. The bus clock power pin (VCCCLK) is the power supply for the on-chip clock drivers. All the voltage pins should be connected together by a power (VCC) plane on the printed circuit board.

The NS32532 grounding connections are made on 20 pins. The logic ground pins (GNDL1 to GNDL6) are the ground pins for the on-chip logic. The buffer ground pins (GNDB1 to GNDB13) are the ground pins for the output drivers of the chip. The bus clock ground pin (GNDCLK) is the ground connection for the on-chip clock drivers. All the ground pins should be connected together by a ground plane on the printed circuit board.

Both power and ground connections are shown in *Figure 3-19*.

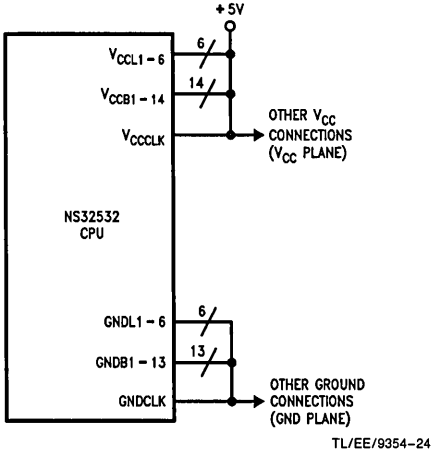


FIGURE 3-19. Power and Ground Connections

3.5.2 Clocking

The NS32532 requires a single-phase input clock signal (CLK) with frequency twice the CPU's operating frequency.

This clock signal is internally divided by two to generate two non-overlapping phases PHI1 and PHI2. One single-phase clock signal BCLK in phase with PHI1 and its complement $\overline{\text{BCLK}}$, are also generated and output by the CPU for timing reference.

Following power-on, the phase relationship between BCLK and CLK is undefined. Nevertheless, in some systems it may be necessary to synchronize the CPU bus timing to an external reference. The $\overline{\text{SYNC}}$ input signal can be used to initialize the phase relationship between CLK and BCLK. $\overline{\text{SYNC}}$ can also be used to stretch BCLK (Low) while CLK is toggling.

$\overline{\text{SYNC}}$ is sampled on each rising edge of CLK. As shown in *Figure 3-20*, whenever $\overline{\text{SYNC}}$ is sampled low, BCLK stops toggling and stays low. On the first rising edge that $\overline{\text{SYNC}}$ is sampled high, BCLK is driven high and then toggles on each subsequent rising edge of CLK.

Every rising edge of BCLK defines a transition in the timing state ("T-State") of the CPU.

One T-State represents the execution of one microinstruction within the CPU and/or one step of an external bus transfer.

Note: The CPU requirement on the maximum period of BCLK must be satisfied when $\overline{\text{SYNC}}$ is asserted at times other than reset.

3.5.3 Resetting

The $\overline{\text{RST}}$ input pin is used to reset the NS32532. The CPU samples $\overline{\text{RST}}$ synchronously on the rising edge of BCLK. Whenever a low level is detected, the CPU responds immediately. Any instruction being executed is terminated; any results that have not yet been written to memory are discarded; and any pending bus errors, interrupts, and traps are eliminated. The internal latches for the edge-sensitive $\overline{\text{NMI}}$ and $\overline{\text{DBG}}$ signals are cleared.

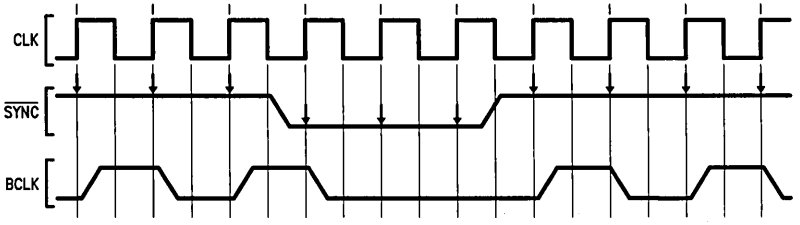


FIGURE 3-20. Bus Clock Synchronization

3.0 Functional Description (Continued)

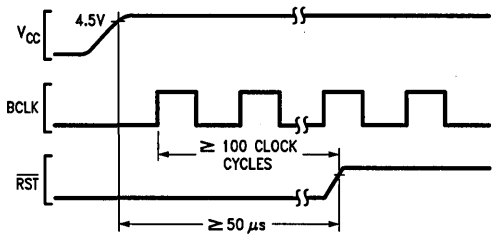
The CPU stores the PC contents in the R0 Register and the PSR contents in the least-significant word of R1, leaving the most-significant word undefined. The PC is then cleared to 0 and so are all the implemented bits in the PSR, MSR, MCR and CFG registers. The DEN-bit in the DCR Register is also cleared to 0. After reset, the remaining implemented bits in DCR and the contents of all other registers are undefined. The CPU begins executing the instruction at Address 0.

On application of power, $\overline{RST}$ must be held low for at least 50 μs after V_{CC} is stable. This is to ensure that all on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must also remain active for not less than 64 BCLK cycles. See *Figures 3-21* and *3-22*.

While in the Reset state, the CPU drives the signals $\overline{ADS}$, $\overline{BE0}$–3, $\overline{BMT}$, $\overline{CONF}$ and $\overline{HLDA}$ inactive. The data bus is floated and the state of all other output signals is undefined.

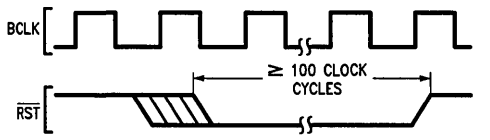
Note 1: If $\overline{HOLD}$ is active at the time $\overline{RST}$ is deasserted, the CPU acknowledges $\overline{HOLD}$ before performing any bus cycle.

Note 2: If $\overline{SYNC}$ is asserted while the CPU is being reset, then BCLK does not toggle. Consequently, $\overline{SYNC}$ must be high for at least 128 CLK cycles while $\overline{RST}$ is low.



TL/EE/9354–26

FIGURE 3-21. Power-On Reset Requirements



TL/EE/9354–27

FIGURE 3-22. General Reset Timing

3.5.4 Bus Cycles

The NS32532 CPU will perform bus cycles for one of the following reasons:

1. To fetch instructions from memory.
2. To write or read data to or from memory or peripheral devices. Peripheral input and output are memory mapped in the Series 32000 family.
3. To read and update Page Table Entries in memory to perform memory management functions.
4. To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
5. To transfer information to or from a Slave Processor.

In terms of bus timing, cases 1 through 4 above are identical. For timing specifications, see Section 4. The only external difference between them is the 5-bit code placed on the Bus Status pins (ST0–ST4). Slave Processor cycles differ in that separate control signals are applied (Section 3.5.4.7).

3.5.4.1 Bus Status

The CPU presents five bits of Bus Status information on pins ST0–ST4. The various combinations on these pins indicate why the CPU is performing a bus cycle, or, if it is idle on the bus, then why is it idle.

The Bus Status pins are interpreted as a five-bit value, with ST0 the least significant bit. Their values decode as follows:

00000 The bus is idle because the CPU does not yet need to access the bus.

00001 The bus is idle because the CPU is waiting for an interrupt following execution of the WAIT instruction.

00010 The bus is idle because the CPU has halted after detecting an abort or bus error while processing an exception.

00011 The bus is idle because the CPU is waiting for a Slave Processor to complete executing an instruction.

00100 Interrupt Acknowledge, Master.

The CPU is reading an interrupt vector to acknowledge an interrupt request.

00101 Interrupt Acknowledge, Cascaded.

The CPU is reading an interrupt vector to acknowledge a maskable interrupt request from a Cascaded Interrupt Control Unit.

00110 End of Interrupt, Master.

The CPU is performing a read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction at the completion of an interrupt's service procedure.

00111 End of Interrupt, Cascaded.

The CPU is performing a read cycle from a Cascaded Interrupt Control Unit to indicate that it is executing a Return from Interrupt (RETI) instruction at the completion of an interrupt's service procedure.

01000 Sequential Instruction Fetch.

The CPU is fetching the next double-word in sequence from the instruction stream.

01001 Non-Sequential Instruction Fetch.

The CPU is fetching the first double-word of a new sequence of instruction. This will occur as a result of any JUMP or BRANCH, any exception, or after the execution of certain instructions.

01010 Data Transfer.

The CPU is reading or writing an operand for an instruction, or it is referring to memory while processing an exception.

01011 Read RMW Class Operand.

The CPU is reading an operand with access class of read-modify-write.

01100 Read for Effective Address Calculation.

The CPU is reading a pointer from memory in order to calculate an effective address for Memory Relative or External addressing modes.

01101 Access PTE1 by MMU.

The CPU is reading or writing a Level-1 Page Table Entry while the on-chip MMU is translating virtual address.

3.0 Functional Description (Continued)

01110 Access PTE2 by MMU.

The CPU is reading or writing a Level-2 Page Table Entry while the on-chip MMU is translating a virtual address.

11101 Transfer Slave Processor Operand.

The CPU is transferring an operand to or from a Slave Processor.

11110 Read Slave Processor Status.

The CPU is reading a status word from a slave processor after the slave processor has activated the $\overline{FSSR}$ signal.

11111 Broadcast Slave Processor ID + OPCODE.

The CPU is initiating the execution of a Slave Instruction by transferring the first 3 bytes of the instruction, which specify the Slave Processor identification and operation.

3.5.4.2 Basic Read and Write Cycles

The sequence of events occurring during a basic CPU access to either memory or peripheral device is shown in *Figure 3-23* for a read cycle, and *Figure 3-24* for a write cycle.

The cases shown assume that the selected memory or peripheral device is capable of communicating with the CPU at full speed. If not, then cycle extension may be requested through the $\overline{RDY}$ line. See Section 3.5.4.4.

A full speed bus cycle is performed in two cycles of the BCLK clock, labeled T1 and T2. For both read and write bus cycles the CPU asserts $\overline{ADS}$ during the first half of T1 indicating the beginning of the bus cycle. From the beginning of T1 until the completion of the bus cycle the CPU drives the Address Bus and other relevant control signals as indicated in the timing diagrams. For cacheable data read cycles the CPU also drives the CASEC signal to indicate the block in the DC set where the data will be stored. If the bus cycle is not cancelled (e.g., state T2 is entered in the next clock cycle), the confirm signal ($\overline{CONF}$) is asserted in the middle of T1. Note that due to a bus cycle cancellation, the $\overline{BMT}$ signal may be asserted at the beginning of T1, and then deasserted before the time in which it is guaranteed valid (see Section 4.4.2).

A confirmed bus cycle is completed at the end of T2, unless a cycle extension is requested. Following state T2 is either state T1 of the next bus cycle, or an idle T-state, if the CPU has no bus cycle to perform.

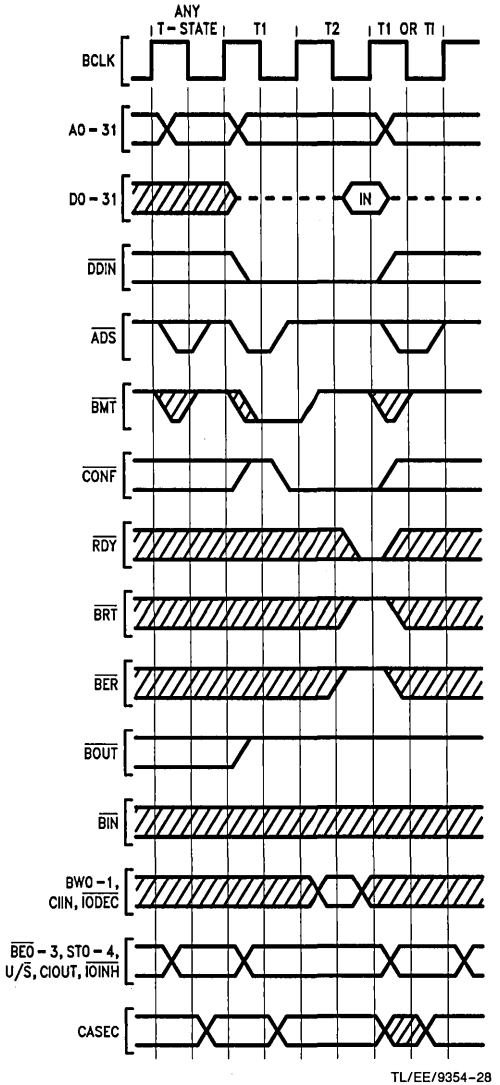
In case of a read cycle the CPU samples the data bus at the end of state T2.

If a bus exception is detected, the data is ignored.

For write bus cycles, valid data is output from the middle of T1 until the end of the cycle. When a write bus cycle is immediately followed by another write cycle, the CPU keeps driving the bus with the data related to the previous cycle until the middle of state T1 of the second bus cycle.

The CPU always inserts an idle state before a write cycle when the write immediately follows a confirmed read cycle.

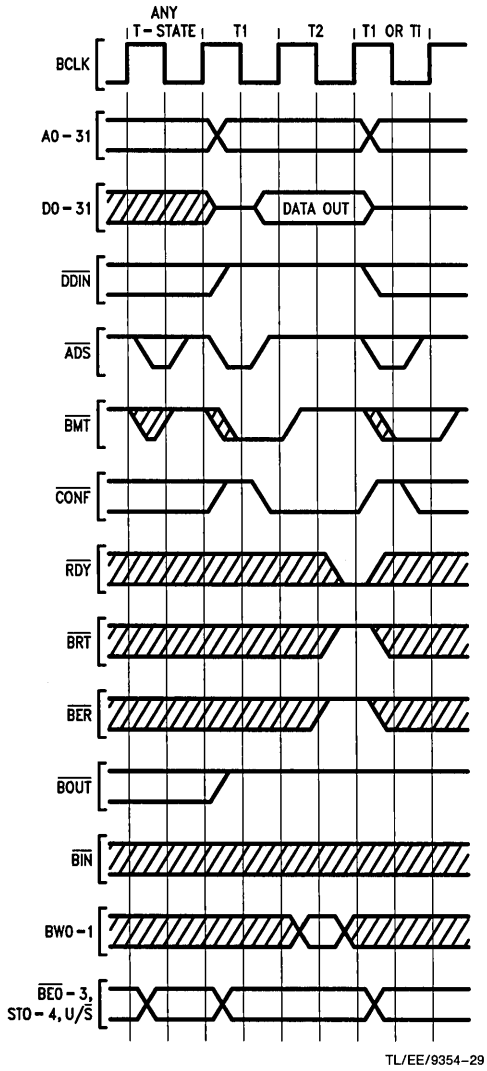
Note: The CPU can initiate a bus cycle with a T1-state and then cancel the cycle, such as when a TLB miss or a Cache hit occurs. In such a case, the $\overline{CONF}$ signal remains High and the $\overline{BMT}$ signal is driven High; the T1-state is followed by another T1-state or an idle T-state.



TL/EE/9354-28

FIGURE 3-23. Basic Read Cycle

3.0 Functional Description (Continued)



TL/EE/9354-29

FIGURE 3-24. Write Cycle

3.5.4.3 Burst Cycles

The NS32532 is capable of performing burst cycles in order to increase the bus transfer rate. Burst is only available in instruction fetch cycles and data read cycle from 32-bit wide memories. Burst is not supported in operand write cycles or slave cycles.

The sequence of events for burst cycles is shown in *Figure 3-25*. The case shown assumes that the selected memory is capable of communicating with the CPU at full speed. If not, then cycle extension can be requested through the $\overline{RDY}$ line. See Section 3.5.4.4.

A Burst cycle is composed of two parts. The first part is a regular cycle (opening cycle), in which the CPU outputs the new status and asserts all the other relevant control signals. In addition, the Burst Out Signal ($\overline{BOUT}$) is activated by the CPU indicating that the CPU can perform Burst cycles. If the selected memory allows Burst cycles, it will notify the CPU by activating the burst in signal ($\overline{BIN}$). $\overline{BIN}$ is sampled by the CPU in the middle of T2 on the falling edge of BCLK. If the memory does not allow burst ($\overline{BIN}$ high), the cycle will terminate at the end of T2 and $\overline{BOUT}$ will go inactive immediately. If the memory allows burst ($\overline{BIN}$ low), and the CPU has not deasserted $\overline{BOUT}$, the second part of the Burst cycle will be performed and $\overline{BOUT}$ will remain active until termination of the Burst.

The second part consists of up to 3 nibbles, labeled T2B. In each of them a data item is read by the CPU. For each nibble in the burst sequence the CPU forces the 2 least-significant bits of the address to 0 and increments address bits 2 and 3 to select the next double-word; all the byte enable signals ($\overline{BE}$0–3) are activated.

As shown in *Figures 3-25* and *4-8* (in Section 4), the CPU samples $\overline{RDY}$ at the end of each nibble and extends the access time for the burst transfer if $\overline{RDY}$ is inactive.

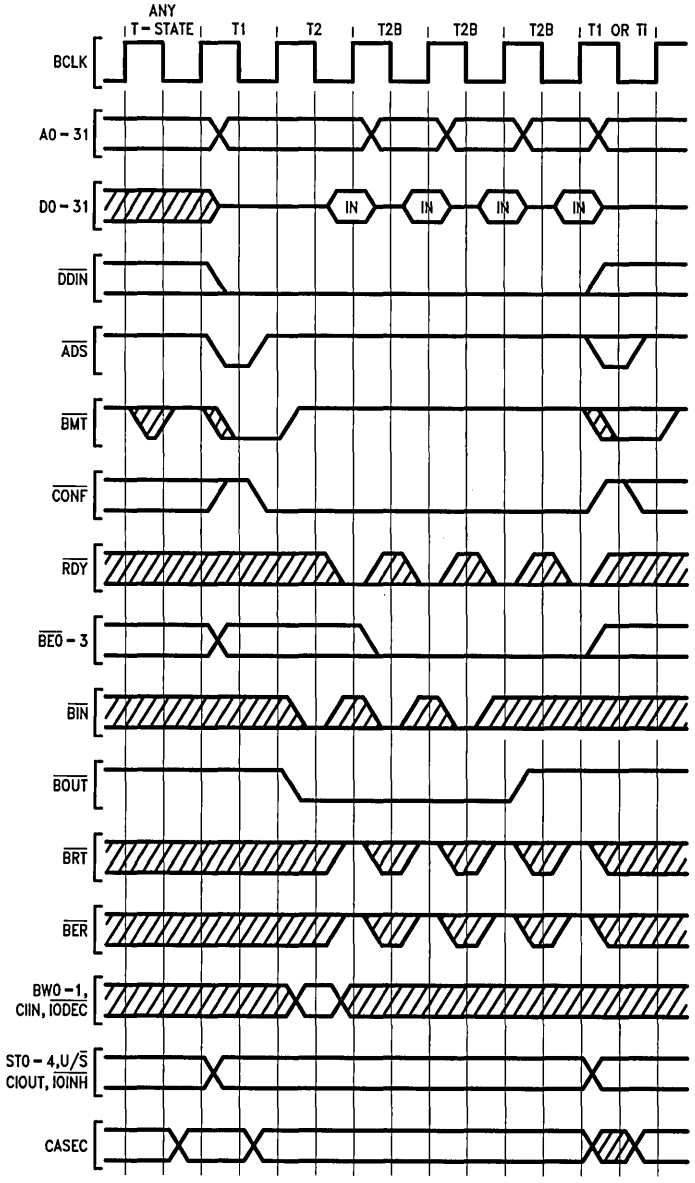
The CPU initiates burst read cycles in the following cases.

1. An instruction must be fetched (Status = 01000 or 01001), and the instruction address does not fall within the last double-word in an aligned 16-byte block (e.g., address bits 2 and 3 are not both equal to 1).
2. A data item must be read (Status = 01010, 01011 or 01100), and all of the following conditions are met.
   - The data cache is enabled and not locked. (DC = 1 and LDC = 0 in the CFG register.)
   - The addressed page is cacheable as indicated in the Level-2 Page Table Entry.
   - The bus cycle is not an interlocked data access performed while executing a CBITI or SBITI instruction.

The Burst sequence will be terminated when one of the following events occurs.

1. The last instruction double-word in an aligned 16-byte block has been fetched.
2. The CPU detects that the instructions being prefetched are no longer needed due to an alteration of the flow of control. This happens, for example, when a Branch instruction is executed or an exception occurs.
3. 4 double-words of data have been read by the CPU. The double-words are transferred within an aligned 16-byte block in a wrap-around order. For example, if a source operand is located at address 104, then the burst read cycle transfers the double-words at 104, 108, 112, and 100, in that order.

3.0 Functional Description (Continued)



TL/EE/9354-30

FIGURE 3-25. Burst Read Cycles

3.0 Functional Description (Continued)

4. The $\overline{BIN}$ signal is deasserted.
5. $\overline{BRT}$ is asserted to signal a bus retry.
6. $\overline{IODEC}$ is asserted or the BW0–1 signals indicate a bus width other than 32-bits. The CPU samples these signals during state T2 of the opening cycle. During T2B-states BW0–1 are ignored and $\overline{IODEC}$ must be kept HIGH.

The CPU uses only the values of the above signals sampled during the last state of the transfer when the cycle is extended. See Section 3.5.4.4.

Note: A burst sequence is not stopped by the assertion of either $\overline{BER}$ or CIIN. See Note 3 in Section 3.5.5.

3.5.4.4 Cycle Extension

To allow sufficient access time for any speed of memory or peripheral device, the NS32532 provides for extension of a bus cycle. Any type of bus cycle except a slave processor cycle can be extended.

A bus cycle can be extended by causing state T2 for a normal cycle or state T2B for a Burst cycle to be repeated.

At the end of each T2 or T2B state, on the rising edge of BCLK, the $\overline{RDY}$ line is sampled by the CPU. If $\overline{RDY}$ is active, then the transfer cycle will be completed. If $\overline{RDY}$ is inactive, then the bus cycle is extended by repeating the T-state for another clock cycle. These additional T-states inserted by the CPU in this manner are called 'WAIT' states.

During a transfer the CPU samples the input control signals $\overline{BIN}$, $\overline{BER}$, $\overline{BRT}$, BW0–1, CIIN and $\overline{IODEC}$.

When wait states are inserted, only the values of these signals sampled during the last wait state are significant.

Figures 3-26 and *4-8* (in Section 4) illustrate both a normal read cycle and a Burst cycle with wait states added through the $\overline{RDY}$ pin.

Note: If $\overline{RST}$ is asserted during a bus cycle, then the cycle is terminated without regard of $\overline{RDY}$.

3.5.4.5 Interlocked Bus Cycles

The NS32532 supports indivisible read-modify-write transactions by asserting the $\overline{ILO}$ signal during consecutive read and write operations. See *Figure 4-7* in Section 4.

Interlocked transactions are always preceded and followed by one or more idle T-states.

The $\overline{ILO}$ signal is asserted in the middle of the idle T-state preceding state T1 of the read operation, and is deasserted in the middle of one of the idle T-states following completion of the write operation, including any retried bus cycles.

No other bus operations (e.g., instruction fetches) will occur while an interlocked transaction is taking place.

Interlocked transactions are required in multiprocessor systems to handle shared resources. The CPU uses them to reference data while executing a CBITIi or SBITIi instruction, during which a single byte of data is read and written. They are also used when the on-chip MMU is updating a Level-2 Page Table Entry during a Page Table Lookup.

In this case a double-word is read and written. If the Level-2 Page Tables are located in a memory area whose width is other than 32 bits, multiple interlocked reads followed by multiple interlocked writes will result. The $\overline{ILO}$ signal is always released for one or more clock cycles in the middle of two consecutive interlocked transactions.

Note 1: If a bus error is detected during an interlocked read cycle, the subsequent interlocked write cycle will not be performed, and $\overline{ILO}$ is deasserted before the next bus cycle begins.

Note 2: The CPU may assert $\overline{ILO}$ before a read cycle that is cancelled (for example, due to a TLB miss). In such a case, the CPU deasserts $\overline{ILO}$ before performing any additional bus cycles.

3.5.4.6 Interrupt Control Cycles

The CPU generates Interrupt-Acknowledge bus cycles in response to non-maskable interrupt and enabled maskable interrupt requests.

The CPU also generates one or two End-of-Interrupt bus cycles during execution of the Return-from-Interrupt (RETI) instruction.

The timing for the interrupt control cycles is the same as for the basic memory read cycle shown in *Figure 3-23*; only the status presented on pins ST0–4 is different. These cycles are single-byte read cycles, and they always bypass the data cache.

Table 3-4 shows the interrupt control sequences associated with each interrupt and with the return from its service procedure.

3.5.4.7 Slave Processor Bus Cycles

The NS32532 performs bus cycles to transfer information to or from slave processors while executing floating-point or custom-slave instructions.

The CPU uses slave write bus cycles to broadcast the identification and operation codes of a slave instruction as well as to transfer operands from memory or general purpose registers to a slave.

Figure 3-27 shows the timing for a slave write bus cycle. The CPU asserts $\overline{SPC}$ during T1; the status is valid during T1 and T2. The operation code or operand is output on the data bus from the middle of T1 until the end of T2.

The CPU uses a slave read bus cycle to transfer a result operand from a slave to either memory or a general purpose register. A slave read cycle is also used to read a status word when the $\overline{FSSR}$ signal is asserted. *Figure 3-28* shows the timing for a slave read bus cycle.

During T1 and T2 the CPU drives the status lines and asserts $\overline{SPC}$. The data from the slave is sampled at the end of T2.

The CPU will never perform another slave cycle immediately following a slave read cycle. In fact, the T-state following state T2 of a slave read cycle is either an idle T-state or the T1 state of a memory cycle.

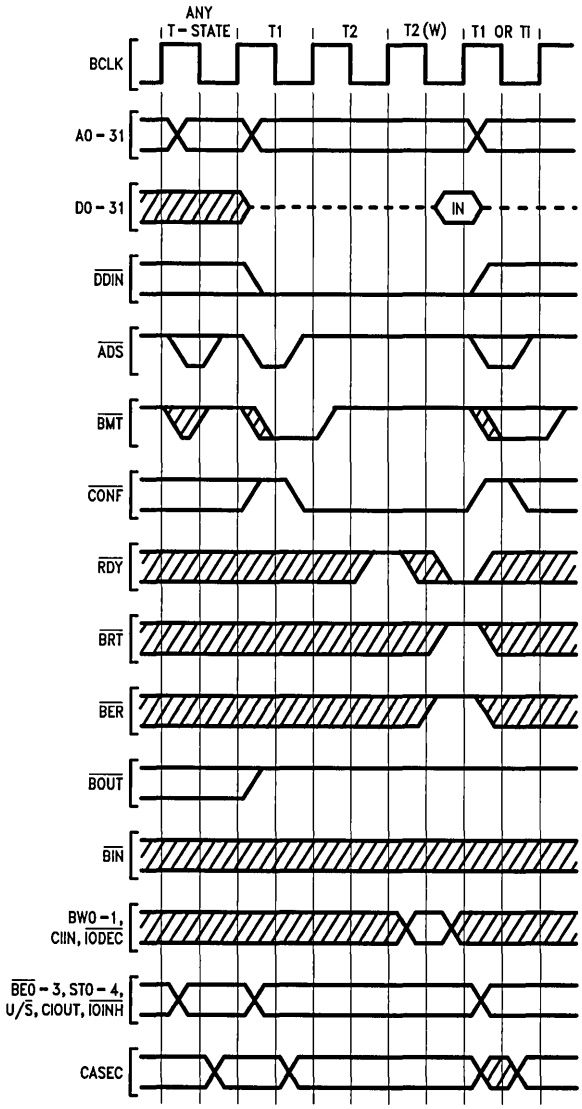
Slave processor data transfers are always 32 bits wide. If the operand is a single byte, then it is transferred on D0 through D7. If it is a word, then it is transferred on D0 through D15.

When two operands are transferred, operand 1 is transferred before operand 2. For double-precision operands, the least-significant double-word is transferred before the most-significant double-word.

During a slave bus cycle the output signals $\overline{BE}$0–3 are undefined while the input signals BW0–1 and $\overline{RDY}$ are ignored.

$\overline{BER}$ and $\overline{BRT}$ must be kept high.

3.0 Functional Description (Continued)



TL/EE/9354-31

3-26. Cycle Extension of a Basic Read Cycle

3.0 Functional Description (Continued)

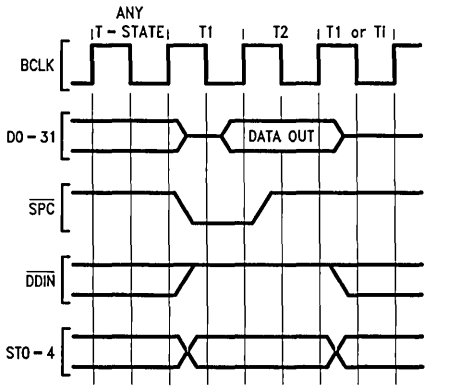
TABLE 3-4. Interrupt Sequences

Cycle	Status	Address	$\overline{DDIN}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Data Bus Byte 3	Byte 2	Byte 1	Byte 0
A. Non-Maskable Interrupt Control Sequences											
Interrupt Acknowledge											
1	00100	$FFFFFF00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
None: Performed through Return from Trap (RETT) instruction.											
B. Non-Vectored Interrupt Control Sequences											
Interrupt Acknowledge											
1	00100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
1	00110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	X
C. Vectored Interrupt Sequences: Non-Cascaded											
Interrupt Acknowledge											
1	00100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Range: 0–127
Interrupt Return											
1	00110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Same as in Previous Int. Ack. Cycle
D. Vectored Interrupt Sequences: Cascaded											
Interrupt Acknowledge											
1	00100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: range −16 to −1
(The CPU here uses the Cascade Index to find the Cascade Address)											
2	001101	Cascade Address	0	See Note				Vector, range 16–255; on appropriate byte of data bus.			
Interrupt Return											
1	00110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: Same as in previous Int. Ack. Cycle
(The CPU here uses the Cascade Index to find the Cascade Address)											
2	00111	Cascade Address	0	See Note				X	X	X	X

X = Don't Care

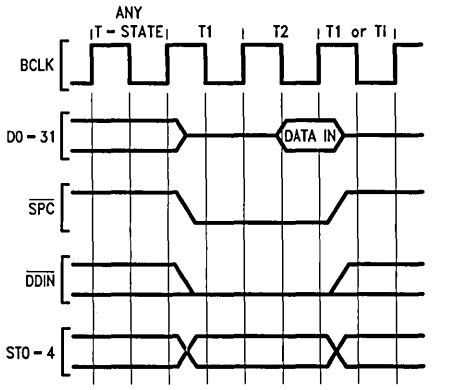
Note: $\overline{BE0}$–$\overline{BE3}$ signals will be activated according to the cascaded ICU address

3.0 Functional Description (Continued)



TL/EE/9354-32

FIGURE 3-27. Slave Processor Write Cycle



TL/EE/9354-33

FIGURE 3-28. Slave Processor Read Cycle

3.5.5 Bus Exceptions

The NS32532 has the capability of handling errors occurring during the execution of a bus cycle. These errors can be either correctable or incorrectable, and the CPU can be notified of their occurrence through the input signals $\overline{BRT}$ and/or $\overline{BER}$.

Bus Retry

If a bus error can be corrected, the CPU may be requested to repeat the erroneous bus cycle. The request is done by asserting the $\overline{BRT}$ signal. $\overline{BRT}$ is sampled at the end of state T2 or T2B.

When the CPU detects that $\overline{BRT}$ is active, it completes the bus cycle normally, but ignores the data read in case of a read cycle, and maintains a copy of the data to be written in case of a write cycle. Then, after a delay of two clock cycles, it will start executing the bus cycle again.

If the transfer cycle is multiple (e.g., for non-aligned data), only the problematic part will be repeated.

For instance, if a non-aligned double-word is being transferred and the second half of the transfer fails, only the second part will be repeated.

The same applies for a retry during a burst sequence. The repeated cycle will begin where the read operation failed (rather than the first address of the burst) and will finish the original burst.

Figures 3-29 and *4-10* (in Section 4) show the $\overline{BRT}$ timing for a basic access cycle and for burst cycles respectively.

The CPU always waits for $\overline{BRT}$ to be HIGH before repeating the bus cycle. While $\overline{BRT}$ is LOW, the CPU places all the output signals shown in *Figure 4-11* in a TRI-STATE® condition.

Bus Error

If a bus error is incorrectable the CPU may be requested to interrupt the current process and branch to an appropriate procedure to handle the error. The request is performed by activating the $\overline{BER}$ signal. $\overline{BER}$ is sampled by the CPU at the end of state T2 or T2B on the rising edge of BCLK.

When $\overline{BER}$ is sampled active, the CPU completes the bus cycle normally. If a bus error occurs during a bus cycle for a reference required to execute an instruction, then a bus error exception is recognized. However, if an error occurs during an acknowledge cycle of another exception or during the ICU read cycle of a RETI instruction, the CPU interprets the event as a fatal bus error and enters the 'halted' state.

In this state the CPU floats its address and data buses and places a special status code on the ST0–4 lines. The CPU can exit this condition only through a hardware reset. Refer to Section 3.2.6 for more details on bus error.

Note 1: If the erroneous bus cycle is extended by means of wait states, then the CPU uses the values of $\overline{BRT}$ and/or $\overline{BER}$ sampled during the last wait state.

Note 2: If the CPU samples both $\overline{BRT}$ and $\overline{BER}$ active, $\overline{BRT}$ has higher priority. The bus error indication is ignored, and the bus cycle is repeated.

Note 3: If $\overline{BER}$ is asserted during a bus cycle of a multi-cycle data transfer, the CPU completes the entire transfer normally, but the data will be ignored. The CPU also ignores any subsequent assertion of $\overline{BER}$ during the same data transfer.

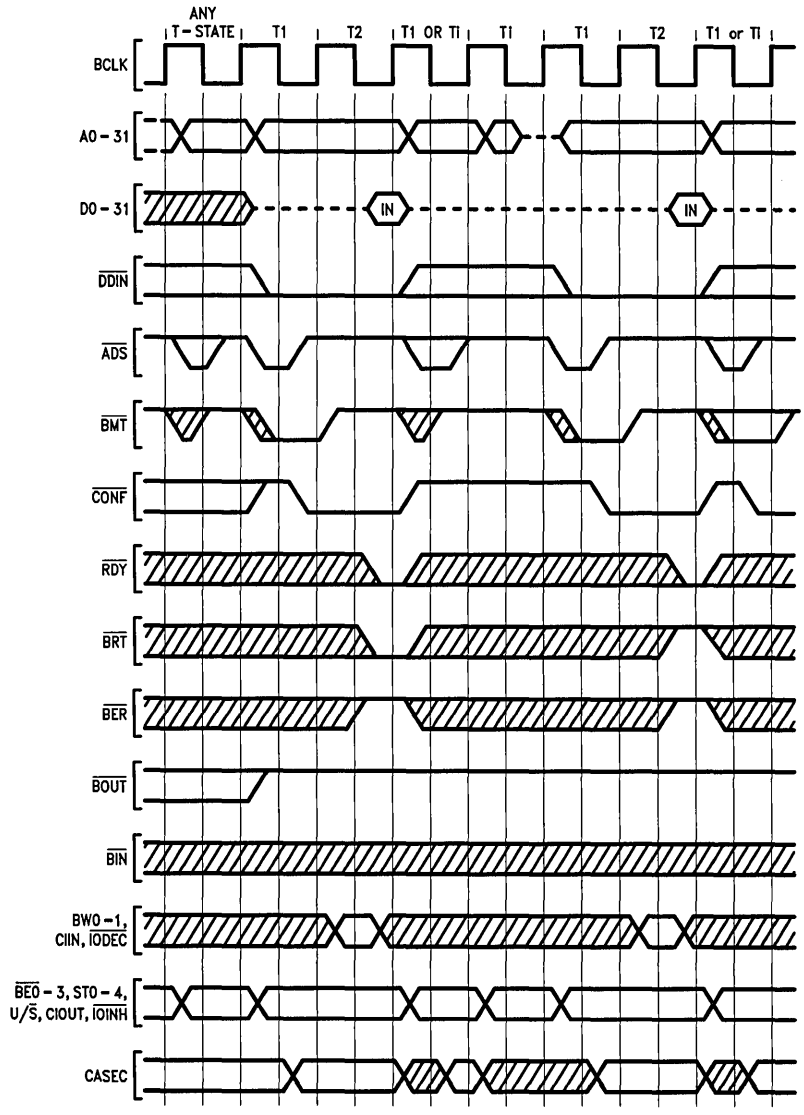
Note 4: Neither $\overline{BRT}$ nor $\overline{BER}$ should be asserted during the T2 state of a slave processor bus cycle.

3.5.6 Dynamic Bus Configuration

The NS32532 is tuned to operate with 32-bit wide memory and peripheral devices. The bus also supports 8-bit and 16-bit data widths, but at reduced efficiency. The CPU can switch from one bus width to another dynamically; the only restriction is that the bus width cannot change for locations within an aligned 16-byte block.

The CPU determines the bus width in effect for a bus cycle by using the values of the BW0 and BW1 signals sampled during the last T2 state. Values of BW0 and BW1 sampled before the last T2 state or during T2B states are ignored. Whenever a bus width other than 32-bit is detected by the CPU, two idle states are inserted before the next bus cycle is initiated. These idle states are only inserted once during an operand access, even if more than two bus cycles are needed to complete the access.

3.0 Functional Description (Continued)



TL/EE/9354-34

FIGURE 3-29. Bus Retry During a Basic Read Cycle

3.0 Functional Description (Continued)

The various combinations for BW0 and BW1 are shown below.

BW1	BW0	
0	0	Reserved
0	1	8-Bit Bus
1	0	16-Bit Bus
1	1	32-Bit Bus

The bus width must always be 32 bits during slave cycles. An important feature of the NS32532 is that it does not impose any restrictions on the data alignment, regardless of the bus width.

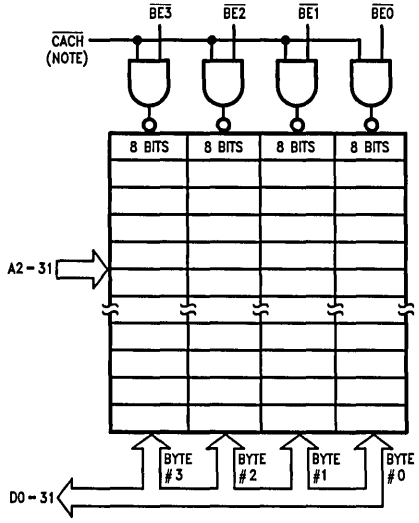
Bus accesses are performed in double-word units. Accesses of data operands that cross double-word boundaries are decomposed into two or more aligned double-word accesses.

The CPU provides four byte enable signals ($\overline{BE}0-3$) which facilitate individual byte accessing on either a 32-bit or a 16-bit bus.

Figures 3-30 and *3-31* show the basic interfaces for 32-bit and 16-bit memories. An 8-bit memory interface (not shown) is even simpler since it does not use any of the $\overline{BE}0-3$ signals and its single bank is always enabled whenever the memory is selected. Each byte location in this case is selected by address bits A0–31.

The NS32532 does not keep track of the bus width used in previous instruction fetches or data accesses. At the beginning of every memory transaction, the CPU always assumes that the bus is 32-bit wide and the $\overline{BE}0-3$ signals are activated accordingly.

The $\overline{BOUT}$ signal is also asserted during instruction fetches or data reads if the conditions for bursting are satisfied. If the bus is other than 32-bit wide, the $\overline{BIN}$ signal is ignored and $\overline{BOUT}$ is deasserted at the beginning of the T state following T2, since burst cycles are not allowed for 8-bit or 16-bit buses.



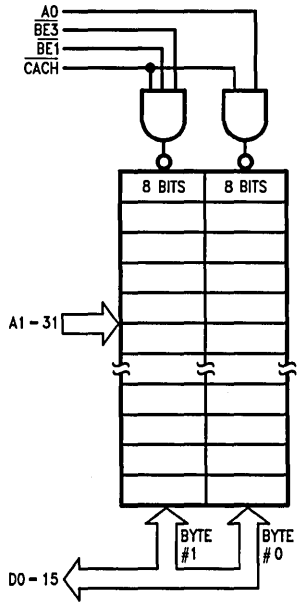
TL/EE/9354-35

FIGURE 3-30. Basic Interface for 32-Bit Memories

Note: The $\overline{CACH}$ signal must be asserted during cacheable read accesses.

The following subsections provide detailed descriptions of the access sequences performed in the various cases.

Note: Although the NS32532 ignores the $\overline{BIN}$ signal for 8-bit and 16-bit bus widths, it is recommended that $\overline{BIN}$ be asserted only if the system supports burst transfers. This is to ensure compatibility with future versions of the CPU that might support burst transfers for 8-bit and 16-bit buses.



TL/EE/9354-36

FIGURE 3-31. Basic Interface for 16-Bit Memories

3.5.6.1 Instruction Fetch Sequences

The CPU performs two types of instruction fetch cycles: sequential and non-sequential. These can be distinguished from each other by the differing status combinations on pins ST0–4. For non-sequential instruction fetches the CPU presents on the address bus the exact byte address of the first instruction in the instruction stream that is about to begin; for sequential instruction fetches, the address of the next aligned instruction double-word is presented on the address bus. The CPU always activates all byte enable signals ($\overline{BE}0-3$) for both sequential and non-sequential fetches. $\overline{BOUT}$ is also asserted during T2 if the addressed double-word is not the last in an aligned 16-byte block. Tables 3-5 to 3-7 show the fetch sequence for the various bus widths.

32-Bit Bus Width

The CPU reads the entire double-word present on the data bus into its internal instruction buffer.

If $\overline{BOUT}$ and $\overline{BIN}$ are both active, the CPU reads up to 3 consecutive double-words using burst cycles. Burst cycles are used for instruction fetches regardless of whether the accesses are cacheable.

3.0 Functional Description (Continued)

Example: JUMP @5

- The CPU performs a fetch cycle at address 5 with $\overline{BE0}$–3 all active.
- Two burst cycles are then performed and addresses 8 and 12 are output while $\overline{BE0}$–3 are kept active.

16-Bit Bus Width

The word on the least-significant half of the data bus is read by the CPU. This is either the even or the odd word within the required instruction double-word, as determined by address bit 1.

The CPU then complements address bit 1, clears address bit 0 and initiates a bus cycle to read the other word, while keeping all the $\overline{BE0}$–3 signals active.

These two words are then assembled into a double-word and transferred into the instruction buffer.

In case of a non-sequential fetch, if the access is not cacheable and the instruction address selects the odd word within the instruction double-word, the even word is not fetched.

Example JUMP @6

- A fetch cycle is performed at address 6 with $\overline{BE0}$–3 all active.
- The word at address 4 is then fetched if the access is cacheable.

8-Bit Bus Width

The instruction byte on the bus lines D0–7 is fetched. The CPU performs three consecutive cycles to read the remaining bytes within the required double-word, while keeping $\overline{BE0}$–3 all active. The 4 bytes are then assembled into a double-word and transferred into the instruction buffer. For a non-sequential fetch, if the access is not cacheable, the CPU will only read the upper bytes within the instruction double-word starting with the byte at the instruction address.

Example: JUMP @7

- The CPU performs a fetch cycle at address 7 with $\overline{BE0}$–3 all active.
- Bytes at addresses 4, 5 and 6 are then fetched consecutively if the access is cacheable.

TABLE 3-5. Cacheable/Non-Cacheable Instruction Fetches from a 32-Bit Bus

1. In a burst access four bytes are fetched with the L.S. bits of the address set to 00.
2. A 'C' on the data bus refers to cacheable fetches and indicates that the byte is placed in the instruction cache. An 'I' refers to non-cacheable fetches and indicates that the byte is ignored.

Number of Bytes	Address LSB	Bytes to be Fetched				Address Bus	$\overline{BE0}$–3	Data Bus			
1	11	B0	—	—	—	A	LLLL	B0	C/I	C/I	C/I
2	10	B1	B0	—	—	A	LLLL	B1	B0	C/I	C/I
3	01	B2	B1	B0	—	A	LLLL	B2	B1	B0	C/I
4	00	B3	B2	B1	B0	A	LLLL	B3	B2	B1	B0

TABLE 3-6. Cacheable/Non-Cacheable Instruction Fetches from a 16-Bit Bus

1. A bus access marked with '*' in the 'Address Bus' column is performed only if the fetch is cacheable.

Number of Bytes	Address LSB	Bytes to be Fetched				Address Bus	$\overline{BE0}$–3	Data Bus			
1	11	B0	—	—	—	A *A – 3	LLLL LLLL	— —	— —	B0 C	C/I C
2	10	B1	B0	—	—	A *A – 2	LLLL LLLL	— —	— —	B1 C	B0 C
3	01	B2	B1	B0	—	A A + 1	LLLL LLLL	— —	— —	B0 B2	C/I B1
4	00	B3	B2	B1	B0	A A + 2	LLLL LLLL	— —	— —	B1 B3	B0 B2

3.0 Functional Description (Continued)

TABLE 3-7. Cacheable/Non-Cacheable Instruction Fetches from an 8-Bit Bus

Number of Bytes	Address LSB	Bytes to be Fetched				Address Bus	$\overline{BE}$0–3	Data Bus			
1	11	B0	—	—	—	A	LLLL	—	—	—	B0
						*A − 3	LLLL	—	—	—	C
						*A − 2	LLLL	—	—	—	C
						*A − 1	LLLL	—	—	—	C
2	10	B1	B0	—	—	A	LLLL	—	—	—	B0
						A + 1	LLLL	—	—	—	B1
						*A − 2	LLLL	—	—	—	C
						*A − 1	LLLL	—	—	—	C
3	01	B2	B1	B0	—	A	LLLL	—	—	—	B0
						A + 1	LLLL	—	—	—	B1
						A + 2	LLLL	—	—	—	B2
						*A − 1	LLLL	—	—	—	C
4	00	B3	B2	B1	B0	A	LLLL	—	—	—	B0
						A + 1	LLLL	—	—	—	B1
						A + 2	LLLL	—	—	—	B2
						A + 3	LLLL	—	—	—	B3

3.5.6.2 Data Read Sequences

The CPU starts a data read access by placing the exact address of the operand on the address bus. The byte enable lines are activated to select only the bytes required by the instruction being executed. This prevents spurious accesses to peripheral devices that might be sensitive to read accesses, such as those which exhibit the characteristic of destructive reading. If the on-chip data cache is internally enabled for the read access, the $\overline{BOUT}$ signal is asserted at the beginning of state T2. $\overline{BOUT}$ will be deasserted if the data cache is externally inhibited (through CIIN or $\overline{IODEC}$), or the bus width is other than 32 bits. During cacheable accesses the CPU always reads all the bytes in the double-word, whether or not they are needed to execute the instruction, and stores them into the data cache. The external memory, in this case, must place the data on the bus regardless of the state of the byte enable signals.

If the data cache is either internally or externally inhibited during the access, the CPU ignores the bytes not selected by the $\overline{BE}$0–3 signals. Data read sequences for the various bus widths are shown in tables 3-8 to 3-10.

32-Bit Bus Width

The entire double-word present on the bus is read by the CPU. If the access is cacheable and the memory allows burst accesses, the CPU reads up to 3 additional double-words within the aligned 16-byte block containing the first byte of the operand. These burst accesses are performed in a wrap-around fashion within the 16-byte block.

Example: MOVW @5, R0

- The CPU reads a double-word at address 5 while keeping $\overline{BE}$1 and $\overline{BE}$2 active.
- If the access is not-cacheable, $\overline{BOUT}$ is deasserted and the data bytes 0 and 3 are ignored.
- If the access is cacheable, the CPU performs burst cycles with $\overline{BE}$0–3 all active, to read the double-words at addresses 8, 12, and 0.

16-Bit Bus Width

The word on the least-significant half of the data bus is read by the CPU. The CPU can then perform another access cycle with address bit 1 complemented and address bit 0 cleared to read the other word within the addressed double-word.

If the access is cacheable, the entire double-word is read and stored into the cache.

If the access is not cacheable, the CPU ignores the bytes in the double-word not selected by $\overline{BE}$0–3. In this case, the second access cycle is not performed, unless selected bytes are contained in the second word.

Example: MOVB @5, R0

- The CPU reads a word at address 5 while keeping $\overline{BE}$1 active.
- If the access is not cacheable, the CPU ignores byte 0.
- If the access is cacheable, the CPU performs another access cycle, with $\overline{BE}$0–3 all active, to read the word at address 6.

8-Bit Bus Width

The data byte on the bus lines D0–7 is read by the CPU. The CPU can then perform up to 3 access cycles to read the remaining bytes in the double-word.

If the access is cacheable, the entire double-word is read and stored into the cache.

If the access is not cacheable, the CPU will only perform those access cycles needed to read the selected bytes.

Example: MOVW @5, R0

- The CPU reads the byte at address 5 while keeping $\overline{BE}$1 and $\overline{BE}$2 active.
- If the access is not cacheable, the CPU activates $\overline{BE}$2 and reads the byte at address 6.
- If the access is cacheable, the CPU performs three bus cycles with $\overline{BE}$0–3 all active, to read the bytes at addresses 6, 7 and 4.

3.0 Functional Description (Continued)

TABLE 3-8. Cacheable/Non-Cacheable Data Reads from a 32-Bit Bus

1. In a burst access four bytes are read with the L.S. bits of the address set to 00.
2. A 'C' on the data bus refers to cacheable reads and indicates that the byte is placed in the data cache. An 'I' refers to non-cacheable reads and indicates that the byte is ignored.

Number of Bytes	Address LSB	Bytes to be Read				Address Bus	$\overline{BE}$0–3	Data Bus			
1	00	—	—	—	B0	A	HHHL	C/I	C/I	C/I	B0
1	01	—	—	B0	—	A	HHLH	C/I	C/I	B0	C/I
1	10	—	B0	—	—	A	HLHH	C/I	B0	C/I	C/I
1	11	BO	—	—	—	A	LHHH	B0	C/I	C/I	C/I
2	00	—	—	B1	B0	A	HHLL	C/I	C/I	B1	B0
2	01	—	B1	B0	—	A	HLLH	C/I	B1	B0	C/I
2	10	B1	B0	—	—	A	LLHH	B1	B0	C/I	C/I
3	00	—	B2	B1	B0	A	HLLL	C/I	B2	B1	B0
3	01	B2	B1	B0	—	A	LLLH	B2	B1	B0	C/I
4	00	B3	B2	B1	B0	A	LLLL	B3	B2	B1	B0

TABLE 3-9. Cacheable/Non-Cacheable Data Reads from a 16-Bit Bus

1. A bus access marked with '*' in the 'Address Bus' column is performed only if the read is cacheable.

Number of Bytes	Address LSB	Data to be Read				Address Bus	$\overline{BE}$0–3		Data Bus			
							Cach.	Non Cach.				
1	00	—	—	—	B0	A *A + 2	HHHL LLLL	HHHL	— —	— —	C/I C	B0 C
1	01	—	—	B0	—	A *A + 1	HHLH LLLL	HHLH	— —	— —	B0 C	C/I C
1	10	—	B0	—	—	A *A − 2	HLHH LLLL	HLHH	— —	— —	C/I C	B0 C
1	11	B0	—	—	—	A *A − 3	LHHH LLLL	LHHH	— —	— —	B0 C	C/I C
2	00	—	—	B1	B0	A *A + 2	HHLL LLLL	HHLL	— —	— —	B1 C	B0 C
2	01	—	B1	B0	—	A A + 1	HLLH LLLL	HLLH HLHH	— —	— —	B0 C/I	C/I B1
2	10	B1	B0	—	—	A *A − 2	LLHH LLLL	LLHH	— —	— —	B1 C	B0 C
3	00	—	B2	B1	B0	A A + 2	HLLL LLLL	HLLL HLHH	— —	— —	B1 C/I	B0 B2
3	01	B2	B1	B0	—	A A + 1	LLLH LLLL	LLLH LLHH	— —	— —	B0 B2	C/I B1
4	00	B3	B2	B1	B0	A A + 2	LLLL LLLL	LLLL LLHH	— —	— —	B1 B3	B0 B2

3.0 Functional Description (Continued)

TABLE 3-10. Cacheable/Non-Cacheable Data Reads from an 8-Bit Bus D8–12

Number of Bytes	Address LSB	Data to be Read				Address Bus	$\overline{BE}$0–3		Data Bus			
							Cach.	Non Cach.				
1	00	—	—	—	B0	A	HHHL	HHHL	—	—	—	B0
						*A + 1	LLLL		—	—	—	C
						*A + 2	LLLL		—	—	—	C
						*A + 3	LLLL		—	—	—	C
1	01	—	—	B0	—	A	HHLH	HHLH	—	—	—	B0
						*A + 1	LLLL		—	—	—	C
						*A + 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
1	10	—	B0	—	—	A	HLHH	HLHH	—	—	—	B0
						*A + 1	LLLL		—	—	—	C
						*A − 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
1	11	B0	—	—	—	A	LHHH	LHHH	—	—	—	B0
						*A − 3	LLLL		—	—	—	C
						*A − 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
2	00	—	—	B1	B0	A	HHLL	HHLL	—	—	—	B0
						A + 1	LLLL	HHLH	—	—	—	B1
						*A + 2	LLLL		—	—	—	C
						*A + 3	LLLL		—	—	—	C
2	01	—	B1	B0	—	A	HLLH	HLLH	—	—	—	B0
						A + 1	LLLL	HLHH	—	—	—	B1
						*A + 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
2	10	B1	B0	—	—	A	LLHH	LLHH	—	—	—	B0
						A + 1	LLLL	LHHH	—	—	—	B1
						*A − 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
3	00	—	B2	B1	B0	A	HLLL	HLLL	—	—	—	B0
						A + 1	LLLL	HLLH	—	—	—	B1
						A + 2	LLLL	HLHH	—	—	—	B2
						*A + 3	LLLL		—	—	—	C
3	01	B2	B1	B0	—	A	LLLH	LLLH	—	—	—	B0
						A + 1	LLLL	LLHH	—	—	—	B1
						A + 2	LLLL	LHHH	—	—	—	B2
						*A − 1	LLLL		—	—	—	C
4	00	B3	B2	B1	B0	A	LLLL	LLLL	—	—	—	B0
						A + 1	LLLL	LLLH	—	—	—	B1
						A + 2	LLLL	LLHH	—	—	—	B2
						A + 3	LLLL	LHHH	—	—	—	B3

3.5.6.3 Data Write Sequences

In a write access the CPU outputs the operand address and asserts only the byte enable lines needed to select the specific bytes to be written.

In addition, the CPU duplicates the data to be written on the appropriate bytes of the data bus in order to handle 8-bit and 16-bit buses.

The various access sequences as well as the duplication of data are summarized in tables 3-11 to 3-13.

32-Bit Bus Width

The CPU performs only one access cycle to write the selected bytes within the addressed double-word.

Example: MOVB R0, @6

- The CPU duplicates byte 2 of the data bus into byte 0 and performs a write cycle at address 6 with $\overline{BE}$2 active.

16-Bit Bus Width

Up to two access cycles are needed to complete the write operation.

3.0 Functional Description (Continued)

Example: MOVW R0, @5

- The CPU duplicates byte 1 of the data bus into byte 0 and performs a write cycle at address 5 with $\overline{BE1}$ and $\overline{BE2}$ active.
- A write at address 6 is then performed with $\overline{BE2}$ active and the original byte 2 of the data bus placed on byte 0.

8-Bit Bus Width

Up to 4 access cycles are needed in this case to complete the write operation.

Example: MOVB R0, @7

- The CPU duplicates byte 3 of the data bus into bytes 0 and 1, and then performs a write cycle at address 7 with $\overline{BE3}$ active.

3.5.7 Bus Access Control

The NS32532 has the capability of relinquishing its control of the bus upon request from a DMA device or another CPU. This capability is implemented with the $\overline{HOLD}$ and $\overline{HLDA}$ signals. By asserting $\overline{HOLD}$, an external device requests access to the bus. On receipt of $\overline{HLDA}$ from the CPU, the device may perform bus cycles, as the CPU at this point has placed all the output signals shown in *Figure 3-32* into the TRI-STATE condition.

To return control of the bus to the CPU, the external device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

The CPU samples $\overline{HOLD}$ in the middle of each T-state on the falling edge of BCLK. If $\overline{HOLD}$ is asserted when the bus is idle between access sequences, then the bus is granted immediately (see *Figure 3-31*). If $\overline{HOLD}$ is asserted during an access sequence, then the bus is granted immediately after the access sequence, including any retried bus cycles, has completed (see *Figure 4-13*). Note that an access sequence can be composed of several bus cycles if the bus width is 8 or 16 bits.

TABLE 3-11. Data Writes to a 32-Bit Bus

1. Bytes on the data bus marked with '•' are undefined.

Number of Bytes	Address LSB	Data to be Written				Address Bus	$\overline{BE}$0–3	Data Bus			
1	00	—	—	—	B0	A	HHHL	•	•	•	B0
1	01	—	—	B0	—	A	HHLH	•	•	B0	B0
1	10	—	B0	—	—	A	HLHH	•	B0	•	B0
1	11	B0	—	—	—	A	LHHH	B0	•	B0	B0
2	00	—	—	B1	B0	A	HHLL	•	•	B1	B0
2	01	—	B1	B0	—	A	HLLH	•	B1	B0	B0
2	10	B1	B0	—	—	A	LLHH	B1	B0	B1	B0
3	00	—	B2	B1	B0	A	HLLL	•	B2	B1	B0
3	01	B2	B1	B0	—	A	LLLH	B2	B1	B0	B0
4	00	B3	B2	B1	B0	A	LLLL	B3	B2	B1	B0

TABLE 3-12. Data Writes to a 16-Bit Bus

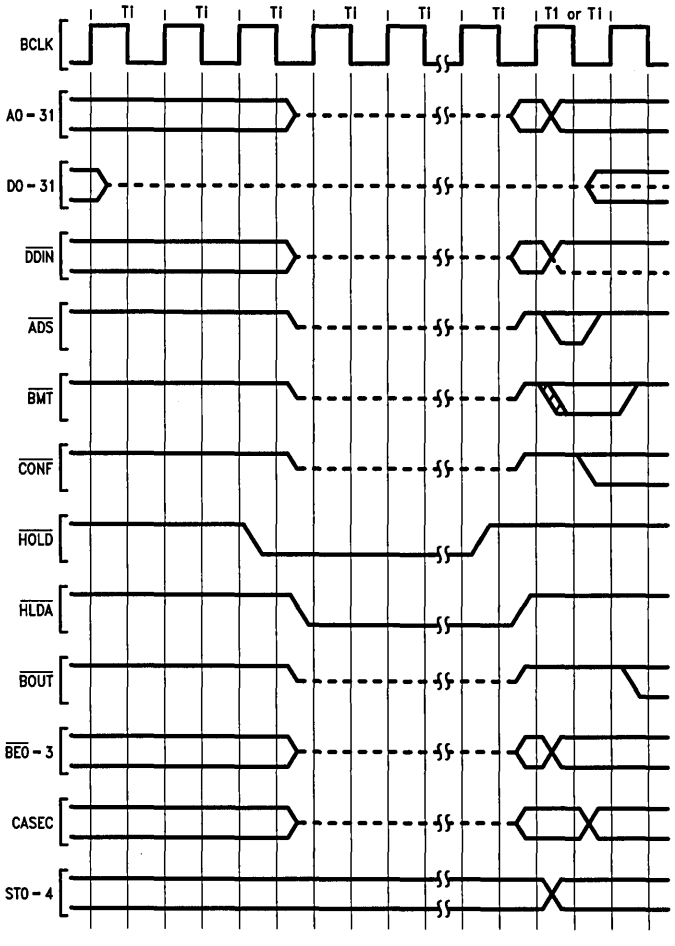
Number of Bytes	Address LSB	Data to be Written				Address Bus	$\overline{BE}$0–3	Data Bus			
1	00	—	—	—	B0	A	HHHL	•	•	•	B0
1	01	—	—	B0	—	A	HHLH	•	•	B0	B0
1	10	—	B0	—	—	A	HLHH	•	B0	•	B0
1	11	B0	—	—	—	A	LHHH	B0	•	B0	B0
2	00	—	—	B1	B0	A	HHLL	•	•	B1	B0
2	01	—	B1	B0	—	A A + 1	HLLH HLHH	• •	B1 •	B0 •	B0 B1
2	10	B1	B0	—	—	A	LLHH	B1	B0	B1	B0
3	00	—	B2	B1	B0	A A + 2	HLLL HLHH	• •	B2 •	B1 •	B0 B2
3	01	B2	B1	B0	—	A A + 1	LLLH LLHH	B2 •	B1 •	B0 B2	B0 B1
4	00	B3	B2	B1	B0	A A + 2	LLLL LLHH	B3 •	B2 •	B1 B3	B0 B2

3.0 Functional Description (Continued)

TABLE 3-13. Data Writes to an 8-Bit Bus

Number of Bytes	Address LSB	Data to be Written				Address Bus	$\overline{BE}$0–3	Data Bus			
1	00	—	—	—	B0	A	HHHL	•	•	•	B0
1	01	—	—	B0	—	A	HHLH	•	•	B0	B0
1	10	—	B0	—	—	A	HLHH	•	B0	•	B0
1	11	B0	—	—	—	A	LHHH	B0	•	B0	B0
2	00	—	—	B1	B0	A A + 1	HHLL HHLH	• •	• •	B1 •	B0 B1
2	01	—	B1	B0	—	A A + 1	HLLH HLHH	• •	B1 •	B0 •	B0 B1
2	10	B1	B0	—	—	A A + 1	LLHH LHHH	B1 •	B0 •	B1 •	B0 B1
3	00	—	B2	B1	B0	A A + 1 A + 2	HLLL HLLH HLHH	• • •	B2 • •	B1 • •	B0 B1 B2
3	01	B2	B1	B0	—	A A + 1 A + 2	LLLH LLHH LHHH	B2 • •	B1 • •	B0 • •	B0 B1 B2
4	00	B3	B2	B1	B0	A A + 1 A + 2 A + 3	LLLL LLLH LLHH LHHH	B3 • • •	B2 • • •	B1 • • •	B0 B1 B2 B3

3.0 Functional Description (Continued)



TL/EE/9354-37

FIGURE 3-32. Hold Acknowledge. (Bus Initially Idle.)

Note: The status indicates 'IDLE' while the bus is granted. If the cause of the IDLE changes (e.g., CPU starts waiting for an interrupt), the status also changes.

The CPU will never grant the bus between interlocked read and write bus cycles.

Note: If an external device requires a very short latency to get control of the bus, the bus retry signal ($\overline{BRT}$) can be used instead of hold. See Section 3.5.5.

3.5.8 Interfacing Memory-Mapped I/O Devices

In Section 3.1.3.2 it was mentioned that some special precautions are needed when interfacing I/O devices to the NS32532 due to its internal pipelined implementation. Two special signals are provided for this purpose: $\overline{IOINH}$ and $\overline{IODEC}$. The CPU asserts $\overline{IOINH}$ during a read bus cycle to indicate that the bus cycle should be ignored if an I/O device is selected. The system responds by asserting $\overline{IODEC}$ to indicate to the CPU that an I/O device has been selected. $\overline{IODEC}$ is sampled by the CPU in the middle of state T2. If the cycle is extended, then the CPU uses the $\overline{IODEC}$ value sampled during the last wait state. If a bus error or a bus retry occurs, the sampled $\overline{IODEC}$ value is ignored. $\overline{IODEC}$ must be kept high during burst transfer cycles.

When $\overline{IODEC}$ is active during a bus cycle for which $\overline{IOINH}$ is asserted, the CPU discards the data and applies the special handling required for I/O devices. *Figure 3-33* shows a possible implementation of an I/O device interface where the address mapping of the I/O devices is fixed.

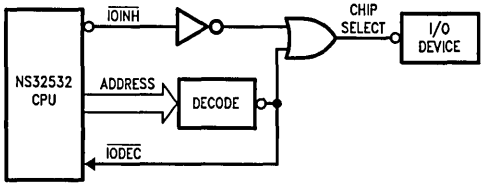
In an open system configuration, $\overline{IODEC}$ could be generated by the decoding logic of each I/O device subsystem.

When the on-chip MMU is enabled, the CIOUT signal could also be used for this purpose, since I/O devices are located in noncacheable areas. In this case however, a small performance degradation could result, due to the fact that the special I/O handling is also applied on references to noncacheable program and/or data areas.

Note 1: When $\overline{IODEC}$ is active in response to a read bus cycle, the CPU treats the reference as noncacheable.

Note 2: $\overline{IOINH}$ is kept inactive during write cycles.

3.0 Functional Description (Continued)



TL/EE/9354-38

FIGURE 3-33. Typical I/O Device Interface

3.5.9 Interrupt and Debug Trap Requests

Three signals are provided by the CPU to externally request interrupts and/or a debug trap. $\overline{INT}$ and $\overline{NMI}$ are for maskable and non-maskable interrupts respectively. $\overline{DBG}$ is used for requesting an external debug trap.

The CPU samples $\overline{INT}$ and $\overline{NMI}$ on every other rising edge of BCLK, starting with the second rising edge of BCLK after $\overline{RST}$ goes high.

$\overline{NMI}$ is edge-sensitive; a high-to-low transition on it is detected by the CPU and stored in an internal latch, so that there is no need to keep it asserted until it is acknowledged.

$\overline{INT}$ is level-sensitive and, as such, once asserted, it must be kept asserted until it is acknowledged.

The $\overline{DBG}$ signal, like $\overline{NMI}$, is edge-sensitive; it differs from $\overline{NMI}$ in that the CPU samples it on each rising edge of BCLK. $\overline{DBG}$ can be asserted asynchronously to the CPU clock, but it should be at least 1.5 clock cycles wide in order to be recognized.

If $\overline{DBG}$ meets the specified setup and hold times, it will be recognized on the rising edge of BCLK deterministically.

Refer to *Figures 4-19* and *4-20* for more details on the timing of the above signals.

Note: If the $\overline{NMI}$ signal is pulsed to request a non-maskable interrupt, it may be necessary to keep it asserted for a minimum of two clock cycles to guarantee its detection, unless extra logic ensures that the pulse occurs around the BCLK sampling edge.

3.5.10 Cache Invalidation Requests

The contents of the on-chip Instruction and Data Caches can be invalidated by external requests from the system. It is possible to invalidate a single set or all sets in the Instruction Cache, Data Cache or both. The input signals $\overline{INVIC}$ and $\overline{INVDC}$ request invalidation of the Instruction Cache and Data Cache respectively. The input signal $\overline{INVSET}$ indicates whether the invalidation applies to a single set (16 bytes for the Instruction Cache and 32 bytes for the Data Cache) or to the entire cache. When only a single set is invalidated, the set number is specified on CIA0–CIA6.

$\overline{INVIC}$, $\overline{INVDC}$, $\overline{INVSET}$ and CIA0–CIA6 are all sampled synchronously by the CPU on the rising edge of BCLK. The CPU can respond to cache invalidation requests at a rate of one per BCLK cycle.

As shown in *Figures 3-16* and *3-17*, the validity bits of the on-chip caches are dual-ported. One port is used for accessing and updating the caches, while the other port is used independently for invalidation requests. Consequently, invalidation of the on-chip caches occurs with no interference to on-going cache accesses or bus cycles.

A cache invalidation request can occur during a read bus cycle for a location affected by the invalidation. In such a case, the data will be invalid in the cache if the invalidation request occurs after the T2- or T2B-state of the bus cycle.

Note: In the case of the Data Cache, the cache location will also be invalidated if the invalidation occurs during T2 or T2B of the read cycle.

Refer to *Figure 4-18* in Section 4 for timing details.

3.5.11 Internal Status

The NS32532 provides information on the system interface concerning its internal activity.

The U/$\overline{S}$ signal indicates the Address Space for a memory reference (See Section 2.4.2).

Note that U/$\overline{S}$ does not necessarily reflect the value of the U bit in the PSR register. For example, U/$\overline{S}$ is high during the memory access used to store the destination operand of a MOVSU instruction.

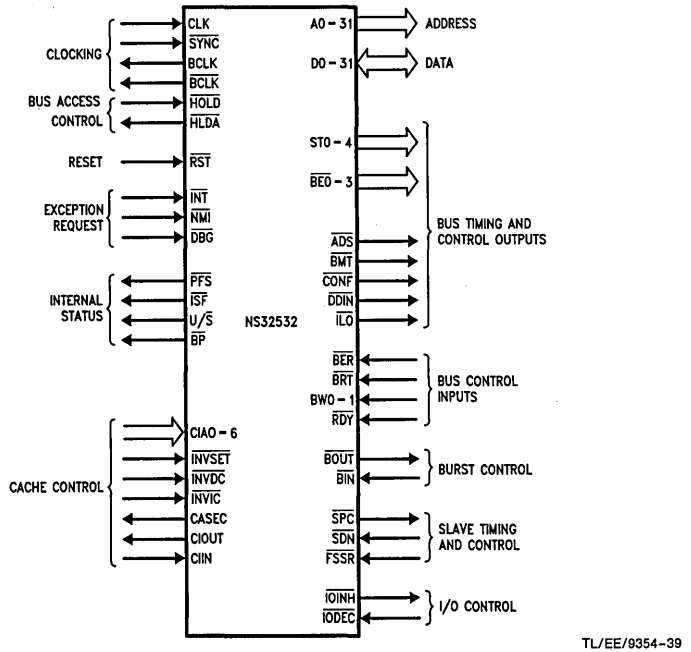
The $\overline{PFS}$ signal is asserted for one BCLK cycle when the CPU begins executing a new instruction. The $\overline{ISF}$ signal is driven High along with $\overline{PFS}$ if the new instruction does not follow the previous instruction in sequence. More specifically, $\overline{ISF}$ is High along with $\overline{PFS}$ after processing an exception or after executing one of the following instructions: ACB (branch taken), Bcond (branch taken), BR, BSR, CASE, CXP, CXPD, DIA, JSR, JUMP, RET, RETT, RETI, and RXP.

The $\overline{BP}$ signal is asserted for one BCLK cycle when an address-compare or PC-match condition is detected. If the $\overline{BP}$ signal is asserted one BCLK cycle after $\overline{PFS}$, it indicates that an address-compare debug condition has been detected. If $\overline{BP}$ is asserted at any other time, it indicates that a PC-Match debug condition has been detected.

While executing an LMR or CINV instruction, the CPU displays the operation code and source operand using slave processor write bus cycles. This information can be used to monitor the contents of the on-chip TLB, Instruction Cache and Data Cache.

During idle bus cycles, the signals ST0–ST4 indicate whether the CPU is waiting for an interrupt, waiting for a Slave Processor to complete executing an instruction or halted.

4.0 Device Specifications



TL/EE/9354-39

FIGURE 4-1. NS32532 Interface Signals

4.1 NS32532 PIN DESCRIPTIONS

Descriptions of the NS32532 pins are given in the following sections.

Included are also references to portions of the functional description, Section 3.

Figure 4-1 shows the NS32532 interface signals grouped according to related functions.

Note: An asterisk next to the signal name indicates a TRI-STATE condition for that signal when $\overline{HOLD}$ is acknowledged or during an extended retry.

4.1.1 Supplies

VCCL1-6 **Logic Power.**
+5V positive supplies for on-chip logic.

VCCB1-14 **Buffers Power.**
+5V positive supplies for on-chip output buffers.

VCCCLK **Bus Clock Power.**
+5V positive supply for on-chip clock drivers.

GNDL1-6 **Logic Ground.**
Ground references for on-chip logic.

GNDB1-13 **Buffers Ground.**
Ground references for on-chip output buffers.

GNDCLK **Bus Clock Ground.**
Ground reference for on-chip clock drivers.

4.1.2 Input Signals

CLK **Clock.**
Input Clock used to derive all CPU Timing.

$\overline{SYNC}$ **Synchronize.**
When $\overline{SYNC}$ is active, BCLK will stop toggling. This signal can be used to synchronize two or more CPUs (Section 3.5.2).

$\overline{HOLD}$ **Hold Request.**
When active, causes the CPU to release the bus for DMA or multiprocessing purposes (Section 3.5.7).

Note:
If the $\overline{HOLD}$ signal is generated asynchronously, its set up and hold times may be violated. In this case it is recommended to synchronize it with the falling edge of BCLK to minimize the possibility of metastable states.

The CPU provides only one synchronization stage to minimize the $\overline{HLDA}$ latency. This is to avoid speed degradations in cases of heavy $\overline{HOLD}$ activity (i.e. DMA controller cycles interleaved with CPU cycles).

$\overline{RST}$ **Reset.**
When $\overline{RST}$ is active, the CPU is initialized to a known state (Section 3.5.3).

$\overline{INT}$ **Interrupt.**
A low level on this signal requests a maskable interrupt (Section 3.5.9).

$\overline{NMI}$ **Nonmaskable Interrupt.**
A High-to-Low transition of this signal requests a nonmaskable interrupt (Section 3.5.9).

4.0 Device Specifications (Continued)

$\overline{DBG}$ Debug Trap Request.

A High-to-Low transition of this signal requests a debug trap (Section 3.5.9).

CIA0–6 Cache Invalidation Address Bus.

Bits 0 through 4 specify the set address to invalidate in the on-chip caches. CIA0 is the least significant. Bits 5 and 6 are reserved (Section 3.5.10).

$\overline{INVSET}$ Invalidate Set.

When Low, only a set in the on-chip cache(s) is invalidated; when High, the entire cache(s) is (are) invalidated.

$\overline{INVDC}$ Invalidate Data Cache.

When Low, the Data Cache contents are invalidated. $\overline{INVSET}$ determines whether a single set or the entire Data Cache is invalidated.

$\overline{INVIC}$ Invalidate Instruction Cache.

When Low, the Instruction Cache contents are invalidated. $\overline{INVSET}$ determines whether a single set or the entire Instruction Cache is invalidated.

CIIN Cache Inhibit In.

When active, indicates that the location referenced in the current bus cycle is not cacheable. CIIN must not change within an aligned 16-byte block.

$\overline{IODEC}$ I/O Decode.

Indicates to the CPU that a peripheral device is addressed by the current bus cycle. The value of $\overline{IODEC}$ must not change within an aligned 16-byte block (Section 3.5.8).

$\overline{FSSR}$ Force Slave Status Read.

When asserted, indicates that the slave status word should be read by the CPU (Section 3.1.4.1). An external 10 kΩ resistor should be connected between $\overline{FSSR}$ and V_{CC}.

$\overline{SDN}$ Slave Done.

Used by a slave processor to signal the completion of a slave instruction (Section 3.1.4.1). An external 10 kΩ resistor should be connected between $\overline{SDN}$ and V_{CC}.

$\overline{BIN}$ Burst In.

When active, indicates to the CPU that the memory supports burst cycles (Section 3.5.4.3).

$\overline{RDY}$ Ready.

While this signal is not active, the CPU extends the current bus cycle to support a slow memory or peripheral device.

BW0–1 Bus Width.

These lines define the bus width (8, 16 or 32 bits) for each data transfer; BW0 is the least significant bit. The bus width must not change within an aligned 16-byte block—encodings are:

00—Reserved

01—8 Bits

10—16 Bits

11—32 Bits

$\overline{BRT}$ Bus Retry.

When active, the CPU will reexecute the last bus cycle (Section 3.5.5).

$\overline{BER}$ Bus Error.

When active, indicates that an error occurred during a bus cycle. It is treated by the CPU as the highest priority exception after reset.

4.1.3 Output Signals

BCLK Bus Clock.

Output clock for bus timing (Section 3.5.2).

$\overline{BCLK}$ Bus Clock Inverse.

Inverted output clock.

$\overline{HLDA}$ Hold Acknowledge.

Activated by the CPU in response to the $\overline{HOLD}$ input to indicate that the CPU has released the bus.

$\overline{PFS}$ Program Flow Status.

A pulse on this signal indicates the beginning of execution for each instruction (Section 3.5.11).

$\overline{ISF}$ Internal Sequential Fetch.

Indicates along with $\overline{PFS}$ that the instruction beginning execution is sequential ($\overline{ISF}$ Low) or non-sequential ($\overline{ISF}$ High).

U/$\overline{S}$ User/Supervisor.

User or supervisor mode status.

$\overline{BP}$ Break Point.

This signal is activated when the CPU detects a PC or operand-address match debug condition (Section 3.3.2).

CASEC *Cache Section.

For cacheable data read bus cycles indicates the Section of the on-chip Data Cache where the data will be placed; undefined for other bus cycles. This signal can be used for external monitoring of the data cache contents.

CIOUT Cache Inhibit Out.

This signal reflects the state of the CI bit in the second level page table entry (PTE). It is used to specify non-cacheable pages. It is held low while address translation is disabled and for MMU references to page table entries.

$\overline{IOINH}$ I/O Inhibit.

Indicates that the current bus cycle should be ignored if a peripheral device is addressed.

$\overline{SPC}$ Slave Processor Control.

Data strobe for slave processor transfers.

$\overline{BOUT}$ *Burst Out.

When active, indicates that the CPU is requesting to perform burst cycles.

$\overline{ILO}$ Interlocked Operation.

When active, indicates that interlocked cycles are being performed (Section 3.5.4.5).

4.0 Device Specifications (Continued)

$\overline{DDIN}$ ***Data Direction.**

Indicates the direction of a data transfer. It is low for reads and high for writes.

$\overline{CONF}$ ***Confirm Bus Cycle.**

When active, indicates that a bus cycle initiated by $\overline{ADS}$ is valid; that is, the bus cycle has not been cancelled (Section 3.5.4.2).

$\overline{BMT}$ ***Begin Memory Transaction.**

When Stable Low indicates that the current bus cycle is valid; that is, the bus cycle has not been cancelled (Section 3.5.4.2).

$\overline{ADS}$ ***Address Strobe.**

When active, indicates that a bus cycle has begun and a valid address is on the address bus.

$\overline{BE}$0-3 ***Byte Enables.**

Used to selectively enable data transfers on bytes 0–3 of the data bus.

ST0-4 **Status.**

Bus cycle status code; ST0 is the least significant. Encodings are:

00000—Idle: CPU Inactive on Bus.

00001—Idle: WAIT Instruction.

00010—Idle: Halted.

00011—Idle: The bus is idle while the slave processor is executing an instruction.

00100—Interrupt Acknowledge, Master.

00101—Interrupt Acknowledge, Cascaded.

00110—End of Interrupt, Master.

00111—End of Interrupt, Cascaded.

01000—Sequential Instruction Fetch.

01001—Non-Sequential Instruction Fetch.

01010—Data Transfer.

01011—Read Read-Modify-Write Operand.

01100—Read for Effective Address.

01101—Access PTE1 by MMU.

01110—Access PTE2 by MMU.

01111 • • • 11100 } Reserved.

11101—Transfer Slave Operand.

11110—Read Slave Status Word.

11111—Broadcast Slave ID.

A0-31 ***Address Bus.**

Used by the CPU to output a 32-bit address at the beginning of a bus cycle. A0 is the least significant.

4.1.4 Input/Output Signals

D0-31 ***Data Bus.**

Used by the CPU to input or output data during a read or write cycle respectively.

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Case Temperature Under Bias	0°C to +95°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	4 W

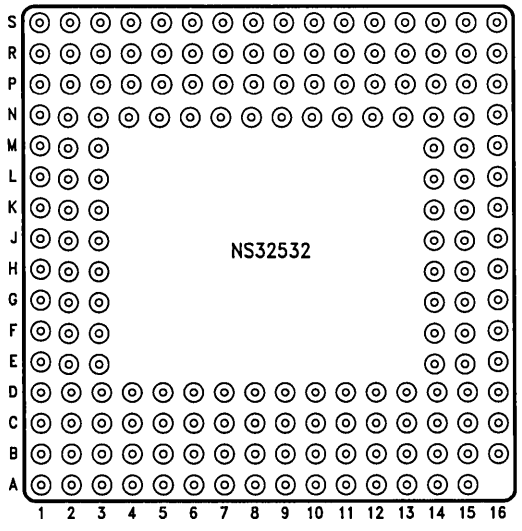
Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_{CASE} = 0° to +95°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		V_{CC} + 0.5	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low Level Output Voltage					
	A0-11, D0-31, $\overline{DDIN}$	I_{OL} = 4 mA			0.4	V
	$\overline{CONF}$, $\overline{BMT}$	I_{OL} = 6 mA			0.4	V
	BCLK, $\overline{BCLK}$	I_{OL} = 16 mA			0.4	V
	All Other Outputs	I_{OL} = 2 mA			0.4	V
I_L	Input Load Current	$0 \le V_{IN} \le V_{CC}$	−20		20	μA
I_L	Leakage Current (Output and I/O pins in TRI-STATE/Input Mode)	$0.4 \le V_{IN} \le V_{CC}$	−20		20	μA
C_{IN}	CLK Input Capacitance			10		pF
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C, V_{CC} = 5V		650 @ 30 MHz 550 @ 25 MHz 450 @ 20 MHz	800 @ 30 MHz 675 @ 25 MHz 575 @ 20 MHz	mA

4.0 Device Specifications (Continued)

Connection Diagram



TL/EE/9354-40

Bottom View

FIGURE 4-2. 175-Pin PGA Package

NS32532 Pinout Descriptions

Desc	Pin	Desc	Pin	Desc	Pin
Reserved	A1	D26	B16	GNDB13	D14
Reserved	A2	Reserved	C1	VCCB14	D15
Reserved	A3	Reserved	C2	D23	D16
$\overline{BP}$	A4	VCCL2	C3	$\overline{IOINH}$	E1
$\overline{ISF}$	A5	Reserved	C4	$\overline{ILO}$	E2
$\overline{RST}$	A6	$\overline{PFS}$	C5	GNDB3	E3
$\overline{NMI}$	A7	$\overline{SDN}$	C6	D24	E14
GNDB1	A8	Reserved	C7	D22	E15
Reserved	A9	$\overline{BCLK}$	C8	D20	E16
VCCB2	A10	VCCCLK	C9	A30	F1
$\overline{INVIC}$	A11	$\overline{SYNC}$	C10	$\overline{CASEC}$	F2
Reserved (1)	A12	CIA0	C11	Reserved	F3
CIA1	A13	CIA6	C12	D21	F14
CIA4	A14	VCCL6	C13	D19	F15
VCCB1	A15	D29	C14	D18	F16
Reserved	B1	D27	C15	A29	G1
VCCB4	B2	D25	C16	A31	G2
Reserved	B3	U/$\overline{S}$	D1	VCCB5	G3
Reserved	B4	Reserved	D2	GNDB12	G14
VCCB3	B5	Reserved	D3	D17	G15
$\overline{FSSR}$	B6	GNDL3	D4	D16	G16
$\overline{INT}$	B7	GNDB2	D5	A27	H1
VCCL1	B8	$\overline{DBG}$	D6	A28	H2
GNDL2	B9	Reserved	D7	GNDB4	H3
$\overline{INVSET}$	B10	BCLK	D8	VCCB13	H14
$\overline{INVDC}$	B11	GNDCLK	D9	D15	H15
CIA3	B12	CLK	D10	D14	H16
CIA5	B13	CIA2	D11	A26	J1
D30	B14	D31	D12	A25	J2
D28	B15	GNDL1	D13	A24	J3

Desc	Pin	Desc	Pin	Desc	Pin
GNDL6	J14	GNDL5	N9	A0	R6
VCCL5	J15	$\overline{CONF}$	N10	VCCB9	R7
D13	J16	$\overline{RDY}$	N11	CIOUT	R8
VCCB6	K1	$\overline{HOLD}$	N12	$\overline{SPC}$	R9
A23	K2	VCCB11	N13	$\overline{BE3}$	R10
GNDL4	K3	GNDB10	N14	VCCB10	R11
GNDB11	K14	D4	N15	$\overline{ADS}$	R12
D11	K15	D6	N16	BW1	R13
D12	K16	A16	P1	$\overline{BER}$	R14
A22	L1	VCCB7	P2	CIIN	R15
A21	L2	GNDB6	P3	D2	R16
VCCL3	L3	A10	P4	A13	S1
D8	L14	A6	P5	A8	S2
D9	L15	A2	P6	A5	S3
D10	L16	ST3	P7	A3	S4
A20	M1	GNDB8	P8	A1	S5
GNDB5	M2	VCCL4	P9	ST2	S6
A17	M3	$\overline{BE1}$	P10	ST1	S7
D5	M14	GNDB9	P11	ST0	S8
D7	M15	BW0	P12	$\overline{BOUT}$	S9
VCCB12	M16	$\overline{BIN}$	P13	$\overline{DDIN}$	S10
A19	N1	Reserved	P14	$\overline{BE2}$	S11
A18	N2	D0	P15	$\overline{BE0}$	S12
A14	N3	D3	P16	$\overline{BMT}$	S13
A11	N4	A15	R1	$\overline{BRT}$	S14
VCCB8	N5	A12	R2	$\overline{IODEC}$	S15
GNDB7	N6	A9	R3	D1	S16
ST4	N7	A7	R4		
$\overline{HLDA}$	N8	A4	R5		

Note 1: This pin should be grounded.
All other reserved pins should be left open.

4.0 Device Specifications (Continued)

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 0.8V or 2.0V on all the signals as illustrated in *Figures 4-3* and *4-4*, unless specifically stated otherwise.

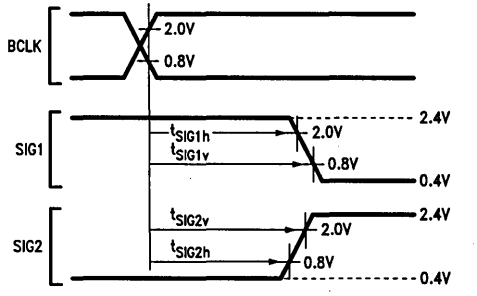


FIGURE 4-3. Output Signals Specification Standard

ABBREVIATIONS:

L.E.—leading edge R.E.—rising edge

T.E.—training edge F.E.—falling edge

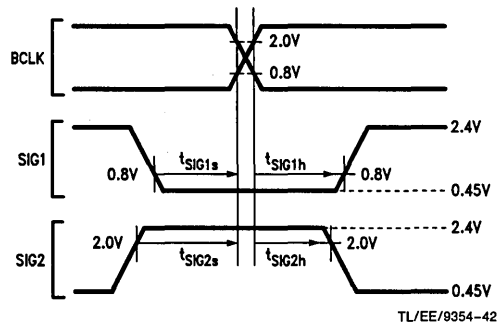


FIGURE 4-4. Input Signals Specification Standard

4.0 Device Specifications (Continued)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32532-20, NS32532-25, NS32532-30

- Maximum times assume capacitive loading of 100 pF on the clock signals and 50 pF on all the other signals. A minimum capacitance load of 50 pF on BCLK and $\overline{BCLK}$ is also assumed.

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{BCp}	4-24	Bus Clock Period	R.E., BCLK to Next R.E., BCLK	50	100	40	100	33.3	100	ns
t_{BCh}	4-24	BCLK High Time	At 2.0V on BCLK (Both Edges)	20		16		13		
t_{BCl}	4-24	BCLK Low Time	At 0.8V on BCLK (Both Edges)	20		16		13		
t_{BCr}(1)	4-24	BCLK Rise Time	0.8V to 2.0V on R.E., BCLK		5		4		3	ns
t_{BCf}(1)	4-24	BCLK Fall Time	2.0V to 0.8V on F.E., BCLK		5		4		3	ns
t_{NBCh}	4-24	$\overline{BCLK}$ High Time	At 2.0V on $\overline{BCLK}$ (Both Edges)	20		16		13		
t_{NBCl}	4-24	$\overline{BCLK}$ Low Time	At 0.8V on $\overline{BCLK}$ (Both Edges)	20		16		13		
t_{NBCr}(1)	4-24	$\overline{BCLK}$ Rise Time	0.8V to 2.0V on R.E., $\overline{BCLK}$		5		4		3	ns
t_{NBCf}(1)	4-24	$\overline{BCLK}$ Fall Time	2.0V to 0.8V on F.E., $\overline{BCLK}$		5		4		3	ns
t_{CBCdr}	4-24	CLK to BCLK R.E. Delay	2.0V on R.E., CLK to 2.0V on R.E., BCLK		20		17		15	ns
t_{CBCdf}	4-24	CLK to BCLK F.E. Delay	2.0V on R.E., CLK to 0.8V on F.E., BCLK		20		17		15	ns
t_{CNBCdr}	4-24	CLK to $\overline{BCLK}$ R.E. Delay	2.0V on R.E., CLK to 0.8V on R.E., $\overline{BCLK}$		20		17		15	ns
t_{CNBCdf}	4-24	CLK to $\overline{BCLK}$ F.E. Delay	2.0V on R.E., CLK to 0.8V on F.E., $\overline{BCLK}$		20		17		15	ns
$t_{BCNBCrf}$	4-24	Bus Clocks Skew	2.0V on R.E., BCLK to 0.8V on F.E., $\overline{BCLK}$	−2	+2	−2	+2	−1	+1	ns
$t_{BCNBCfr}$	4-24	Bus Clocks Skew	0.8V on F.E., BCLK to 2.0V on R.E., $\overline{BCLK}$	−2	+2	−2	+2	−1	+1	ns
t_{Av}	4-5, 4-6	Address Bits 0-31 Valid	After R.E., BCLK T1		11		9		8	ns
t_{Ah}	4-5, 4-6	Address Bits 0-31 Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{Af}	4-11, 4-12	Address Bits 0-31 Floating	After F.E., BCLK Ti		21		17		13	ns
t_{Anf}	4-11, 4-12	Address Bits 0-31 Not Floating	After F.E., BCLK Ti	0		0		0		ns

Note 1: Guaranteed by characterization. Due to tester conditions this parameter is not 100% tested.

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32532-20, NS32532-25, NS32532-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{ABv}	4–8	Address Bits A2, A3 Valid (Burst Cycle)	After R.E., BCLK T2B		11		9		8	ns
t_{ABh}	4–8	Address Bits A2, A3 Hold (Burst Cycle)	After R.E., BCLK T2B	0		0		0		ns
t_{DOv}	4–6, 4–15	Data Out Valid	After R.E., BCLK T1		0.5 t_{BCp} + 13 ns		0.5 t_{BCp} + 12 ns		0.5 t_{BCp} +11 ns	ns
t_{DOh}	4–6, 4–15	Data Out Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{DOspc}	4–15	Data Out Setup (Slave Write)	Before $\overline{SPC}$ T.E.	12		10		8		ns
t_{DOf}	4–7	Data Bus Floating	After R.E., BCLK T1 or Ti		21		17		13	ns
t_{DOnf}	4–7	Data Bus Not Floating	After F.E., BCLK T1	0		0		0		ns
t_{BMTv}	4–5, 4–7	$\overline{BMT}$ Signal Valid	After R.E., BCLK T1		30		25		21	ns
t_{BMTh}	4–5, 4–7	$\overline{BMT}$ Signal Hold	After R.E., BCLK T2	0		0		0		ns
t_{BMTf}	4–11, 4–12	$\overline{BMT}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
t_{BMThf}	4–11, 4–12	$\overline{BMT}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{CONFa}	4–5, 4–8	$\overline{CONF}$ Signal Active	After R.E., BCLK T1	0.5 t_{BCp}	0.5 t_{BCp} + 11	0.5 t_{BCp}	0.5 t_{BCp} + 9	0.5 t_{BCp}	0.5 t_{BCp} + 8	ns
t_{CONFia}	4–5, 4–8	$\overline{CONF}$ Signal Inactive	After R.E., BCLK T1 or Ti		11		9		8	ns
t_{CONFf}	4–11, 4–12	$\overline{CONF}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
t_{CONFnf}	4–11, 4–12	$\overline{CONF}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{ADSa}	4–5, 4–8	$\overline{ADS}$ Signal Active	After R.E., BCLK T1		11		9		8	ns
t_{ADSia}	4–5, 4–8	$\overline{ADS}$ Signal Inactive	After F.E., BCLK T1		11		9		8	ns
t_{ADSw}	4–6	$\overline{ADS}$ Pulse Width	At 0.8V (Both Edges)	15		12		10		ns
t_{ADSf}	4–11, 4–12	$\overline{ADS}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
t_{ADSnf}	4–11, 4–12	$\overline{ADS}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{BEv}	4–6, 4–8	$\overline{BE}_n$ Signals Valid	After R.E., BCLK T1		11		9		8	ns
t_{BEh}	4–6, 4–8	$\overline{BE}_n$ Signals Hold	After R.E., BCLK T1, Ti or T2B	0		0		0		ns
t_{BEf}	4–11, 4-12	$\overline{BE}_n$ Signals Floating	After F.E., BCLK Ti		21		17		13	ns
t_{BEnf}	4–11, 4–12	$\overline{BE}_n$ Signals Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{DDINv}	4–5, 4–6	$\overline{DDIN}$ Signal Valid	After R.E., BCLK T1		11		9		8	ns
t_{DDINh}	4–5, 4–6	$\overline{DDIN}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{DDINf}	4–11, 4–12	$\overline{DDIN}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
t_{DDINnf}	4–11, 4–12	$\overline{DDIN}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{SPCa}	4–14, 4–15	$\overline{SPC}$ Signal Active	After R.E., BCLK T1		19		15		12	ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32532-20, NS32532-25, NS32532-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
$t_{SPC_{ia}}$	4-14, 4-15	$\overline{SPC}$ Signal Inactive	After R.E., BCLK Ti, T1 or T2		19		15		12	ns
t_{DDSPC}(1)	4-14	$\overline{DDIN}$ Valid to $\overline{SPC}$ Active	Before $\overline{SPC}$ L.E.	0		0		0		ns
t_{HLDA_a}	4-12, 4-13	$\overline{HLDA}$ Signal Active	After F.E., BCLK Ti		15		11		10	ns
$t_{HLDA_{ia}}$	4-12	$\overline{HLDA}$ Signal Inactive	After F.E., BCLK Ti		15		11		10	ns
t_{ST_v}	4-5, 4-14	Status (ST0-4) Valid	After R.E., BCLK T1		11		9		8	ns
t_{ST_h}	4-5, 4-14	Status (ST0-4) Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{BOUT_a}	4-8, 4-9	$\overline{BOUT}$ Signal Active	After R.E., BCLK T2		15		12		11	ns
$t_{BOUT_{ia}}$	4-8, 4-9	$\overline{BOUT}$ Signal Inactive	After R.E., BCLK Last T2B, T1 or Ti		15		12		11	ns
t_{BOUT_f}	4-11, 4-12	$\overline{BOUT}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{BOUT_{nf}}$	4-11, 4-12	$\overline{BOUT}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{ILO_a}	4-7	Interlock Signal Active	After F.E., BCLK Ti		11		9		8	ns
$t_{ILO_{ia}}$	4-7	Interlock Signal Inactive	After F.E., BCLK Ti		11		9		8	ns
t_{PFS_a}	4-21	$\overline{PFS}$ Signal Active	After F.E., BCLK		15		11		10	ns
$t_{PFS_{ia}}$	4-21	$\overline{PFS}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{ISF_a}	4-22	$\overline{ISF}$ Signal Active	After F.E., BCLK		15		11		10	ns
$t_{ISF_{ia}}$	4-22	$\overline{ISF}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{BP_a}	4-23	$\overline{BP}$ Signal Active	After F.E., BCLK		15		11		10	ns
$t_{BP_{ia}}$	4-23	$\overline{BP}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{US_v}	4-5	U/$\overline{S}$ Signal Valid	After R.E., BCLK T1		11		9		8	ns
t_{US_h}	4-5	U/$\overline{S}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{CAS_v}	4-5	$\overline{CASEC}$ Signal Valid	After F.E., BCLK T1		15		11		10	ns
t_{CAS_h}	4-5	$\overline{CASEC}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{CAS_f}	4-11, 4-12	$\overline{CASEC}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{CAS_{nf}}$	4-11, 4-12	$\overline{CASEC}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{CIO_v}	4-5	CIOUT Signal Valid	After R.E., BCLK T1		15		11		10	ns
t_{CIO_h}	4-5	CIOUT Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{IOI_v}	4-5	$\overline{IOINH}$ Signal Valid	After R.E., BCLK T1		15		11		10	ns
t_{IOI_h}	4-5	$\overline{IOINH}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32532-20, NS32532-25, NS32532-30

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{Cp}	4–24	Input Clock Period	R.E., CLK to Next R.E., CLK	25	50	20	50	16.6	50	ns
t_{Ch}	4–24	CLK High Time	At 2.0V on CLK (Both Edges)	0.5 t_{Cp} −5 ns		0.5 t_{Cp} −5 ns		0.5 t_{Cp} −4 ns		
t_{Cl}	4–24	CLK Low Time	At 0.8V on CLK (Both Edges)	0.5 t_{Cp} −5 ns		0.5 t_{Cp} −5 ns		0.5 t_{Cp} −4 ns		
t_{Cr} (1)	4–24	CLK Rise Time	0.8V to 2.0V on R.E., CLK		5		4		3	ns
t_{Cf} (1)	4–24	CLK Fall Time	2.0V to 0.8V on F.E., CLK		5		4		3	ns
t_{DIs}	4–5, 4–14	Data In Setup	Before R.E., BCLK T1 or Ti	12		10		8		ns
t_{DIh}	4–5, 4–14	Data In Hold	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{RDYs}	4–5	$\overline{RDY}$ Setup Time	Before R.E., BCLK T2(W), T1 or Ti	19		15		12		ns
t_{RDYh}	4–5	$\overline{RDY}$ Hold Time	Ater R.E., BCLK T2(W), T1 or Ti	1		1		1		ns
t_{BWs}	4–5	BW0–1 Setup Time	Before F.E., BCLK T2 or T2(W)	19		15		12		ns
t_{BWh}	4–5	BW0–1 Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{HOLDs}	4–12, 4–13	$\overline{HOLD}$ Setup Time	Before F.E., BCLK	19		15		12		ns
t_{HOLDh}	4–12	$\overline{HOLD}$ Hold Time	After F.E., BCLK	1		1		1		ns
t_{BINs}	4–8	$\overline{BIN}$ Setup Time	Before F.E., BCLK T2 or T2(W)	18		14		11		ns
t_{BINh}	4–8	$\overline{BIN}$ Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{BERs}	4–6, 4–8	$\overline{BER}$ Setup Time	Before R.E., BCLK T1 or Ti	19		15		12		ns
t_{BERh}	4–6, 4–8	$\overline{BER}$ Hold Time	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{BRTs}	4–6, 4–8	$\overline{BRT}$ Setup Time	Before R.E., BCLK T1 or Ti	19		15		12		ns
t_{BRTh}	4–6, 4–8	$\overline{BRT}$ Hold Time	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{IODs}	4–5	$\overline{IODEC}$ Setup Time	Before F.E., BCLK T2 or T2(W)	18		14		11		ns
t_{IODh}	4–5	$\overline{IODEC}$ Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{PWR} (1)	4–26	Power Stable to R.E. of $\overline{RST}$	After VCC Reaches 4.5V	50		40		30		μs
t_{RSTs}	4–27	$\overline{RST}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{RSTw}	4–27	$\overline{RST}$ Pulse Width	At 0.8V (Both Edges)	64		64		64		t_{BCp}

Note 1: Guaranteed by characterization. Due to tester conditions this parameter is not 100% tested.

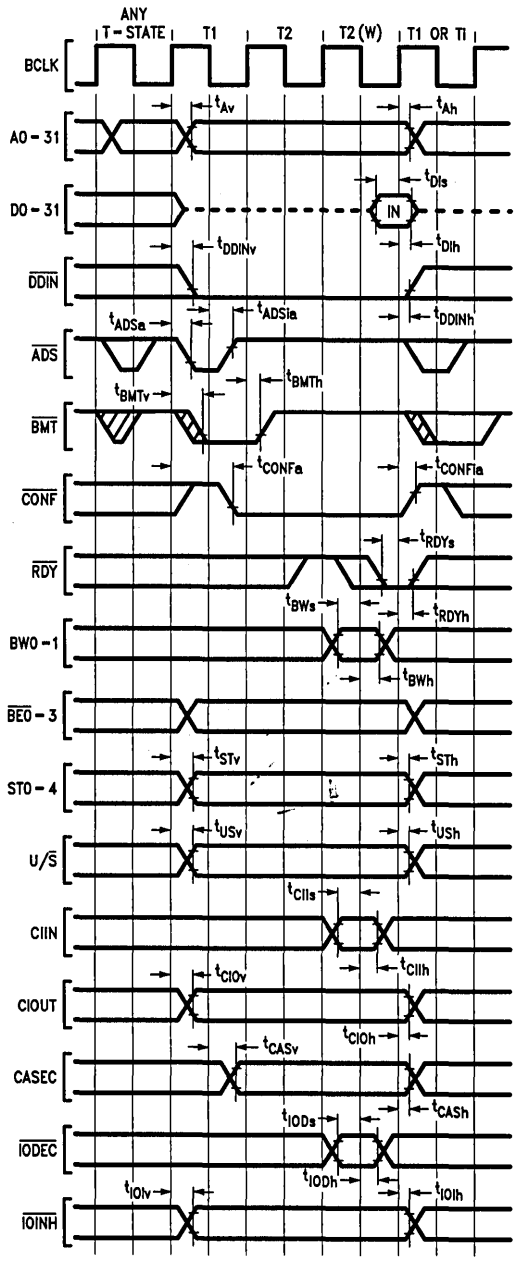
4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32532-20, NS32532-25, NS32532-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{CII_S}	4–5	$\overline{CIIN}$ Setup Time	Before F.E., BCLK T2	19		15		12		ns
t_{CII_H}	4–5	$\overline{CIIN}$ Hold Time	After F.E., BCLK T2	1		1		1		ns
t_{INT_S}	4–19	$\overline{INT}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{INT_H}	4–19	$\overline{INT}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{NMI_S}	4–19	$\overline{NMI}$ Setup Time	Before R.E., BCLK	18		15		14		ns
t_{NMI_H}	4–19	$\overline{NMI}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{SD_S}	4–16	$\overline{SDN}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{SD_H}	4–16	$\overline{SDN}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{FSSR_S}	4–17	$\overline{FSSR}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{FSSR_H}	4–17	$\overline{FSSR}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{SYNC_S}	4–25	$\overline{SYNC}$ Setup Time	Before R.E., CLK	10		8		7		ns
t_{SYNC_H}	4–25	$\overline{SYNC}$ Hold Time	After R.E., CLK	1		1		1		ns
t_{CIA_S}	4–18	CIA0–6 Setup Time	Before R.E., BCLK	12		10		9		ns
t_{CIA_H}	4–18	CIA0–6 Hold Time	After R.E., BCLK	1		1		1		ns
t_{INVS_S}	4–18	$\overline{INVSET}$ Setup Time	Before R.E., BCLK	12		11		9		ns
t_{INVS_H}	4–18	$\overline{INVSET}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{INVI_S}	4–18	$\overline{INVIC}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{INVI_H}	4–18	$\overline{INVIC}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{INVD_S}	4–18	$\overline{INVDC}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{INVD_H}	4–18	$\overline{INVDC}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{DBG_S}	4–20	$\overline{DBG}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{DBG_H}	4–20	$\overline{DBG}$ Hold Time	After R.E., BCLK	1		1		1		ns

4.0 Device Specifications (Continued)

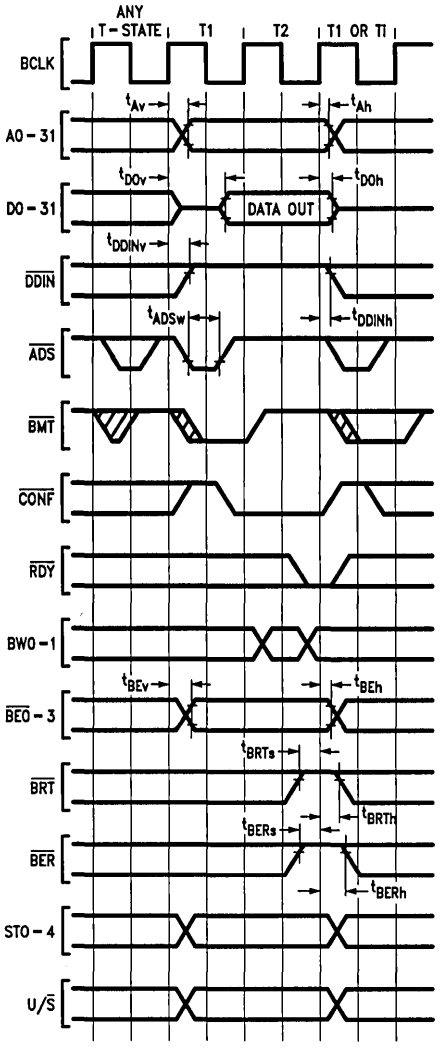
4.4.3 Timing Diagrams



TL/EE/9354-43

FIGURE 4-5. Basic Read Cycle Timing

4.0 Device Specifications (Continued)

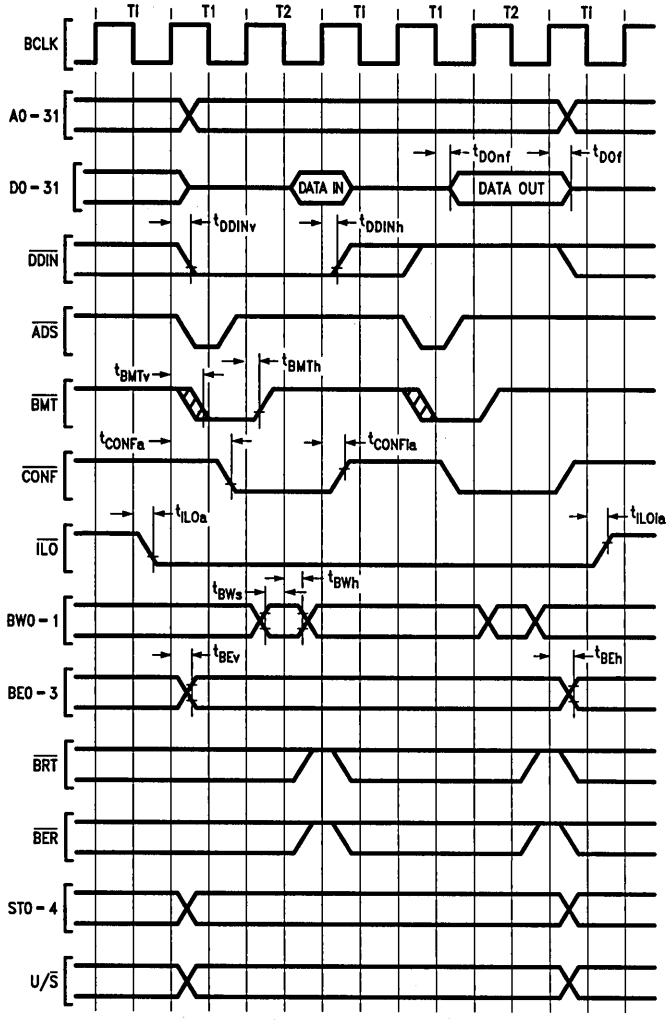


TL/EE/9354-44

Note: An Idle State is always inserted before a Write Cycle when the Write immediately follows a confirmed Read Cycle.

FIGURE 4-6. Write Cycle Timing

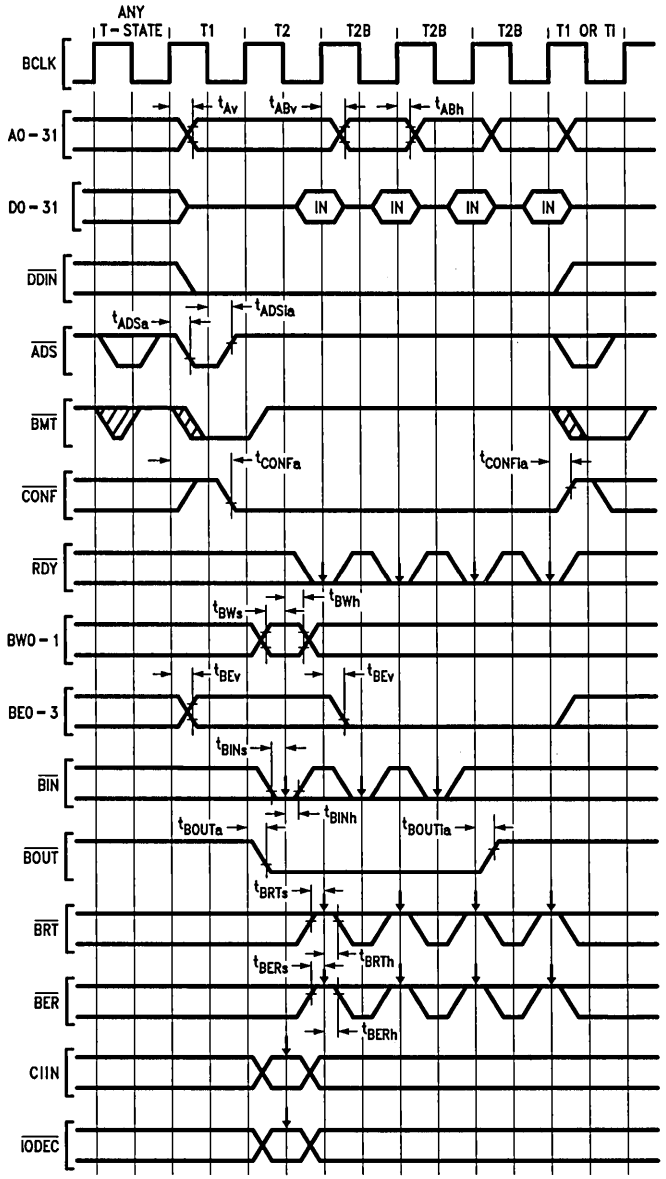
4.0 Device Specifications (Continued)



TL/EE/9354-45

FIGURE 4-7. Interlocked Read and Write Cycles

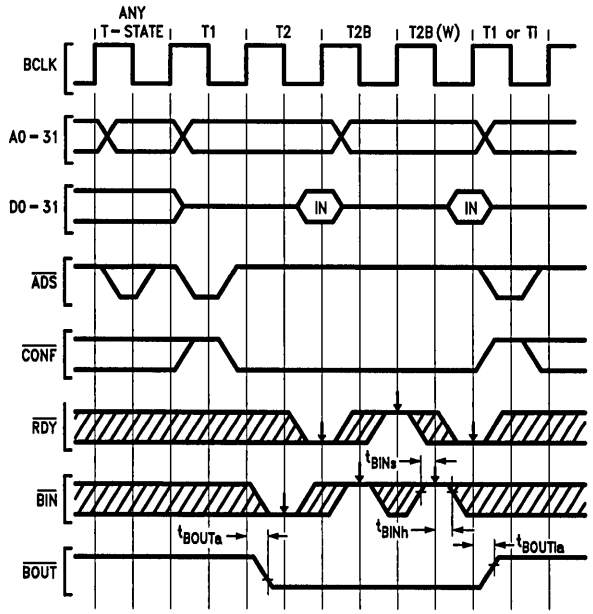
4.0 Device Specifications (Continued)



TL/EE/9354-46

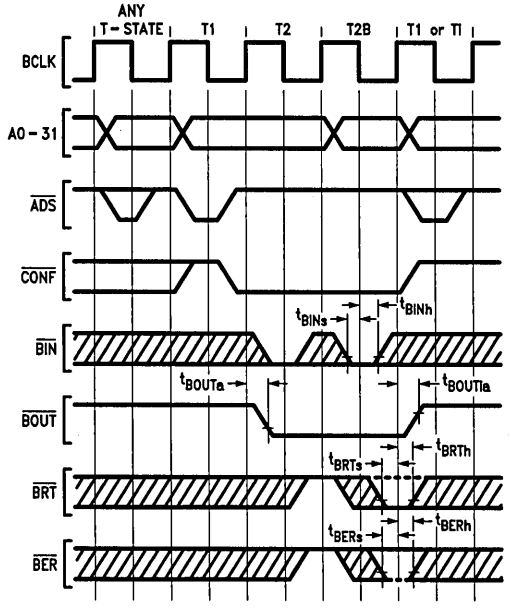
FIGURE 4-8. Burst Read Cycles

4.0 Device Specifications (Continued)



TL/EE/9354-47

FIGURE 4-9. External Termination of Burst Cycles

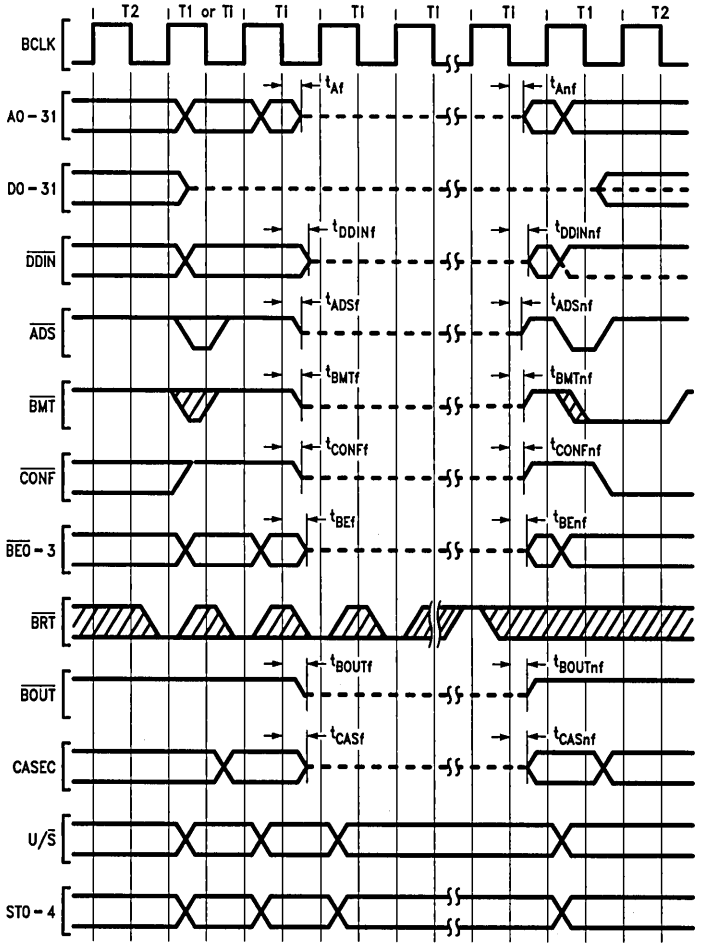


TL/EE/9354-48

FIGURE 4-10. Bus Error or Retry During Burst Cycles

Note: Two idle state are always inserted by the CPU following the assertion of $\overline{BRT}$.

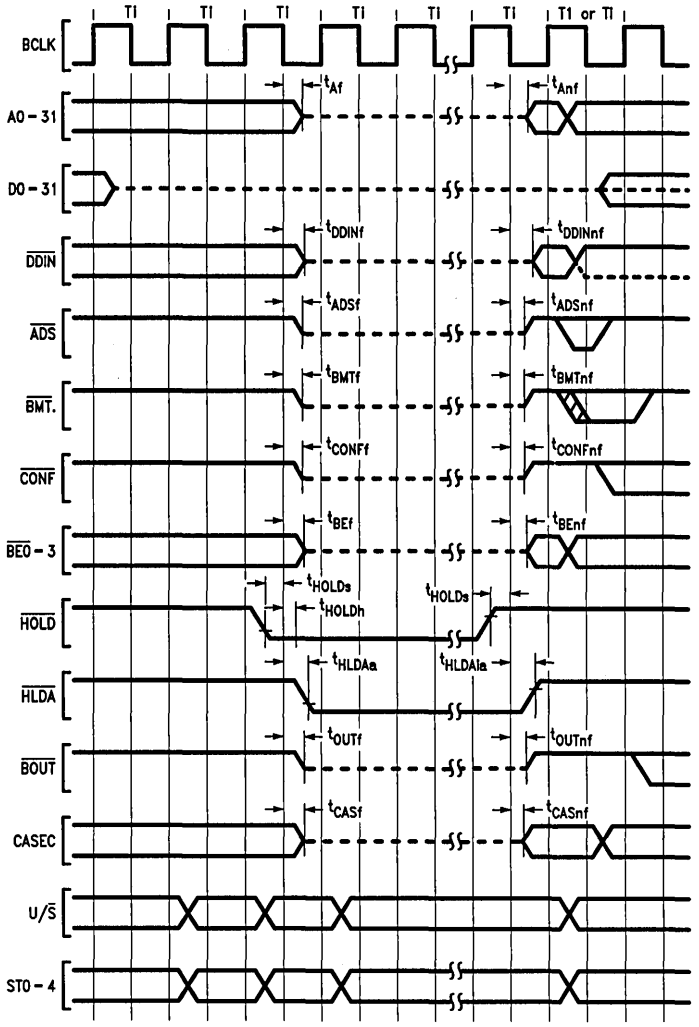
4.0 Device Specifications (Continued)



TL/EE/9354-49

FIGURE 4-11. Extended Retry Timing

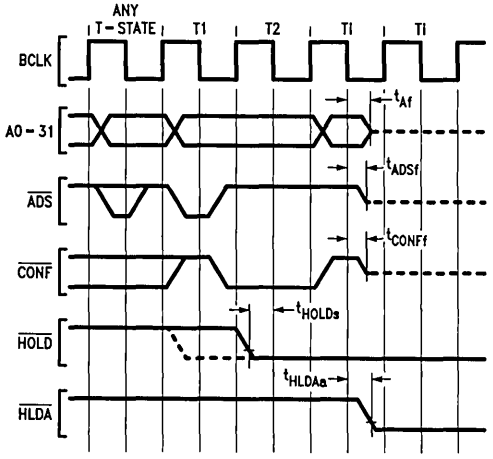
4.0 Device Specifications (Continued)



TL/EE/9354-50

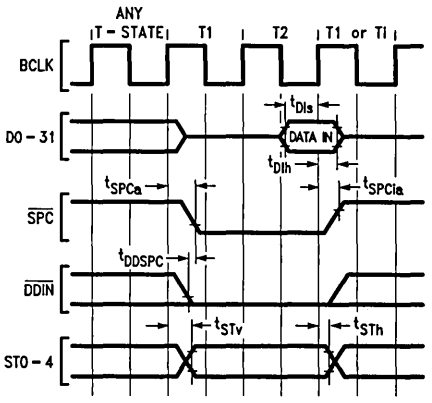
FIGURE 4-12. Hold Timing (Bus Initially Idle)

4.0 Device Specifications (Continued)



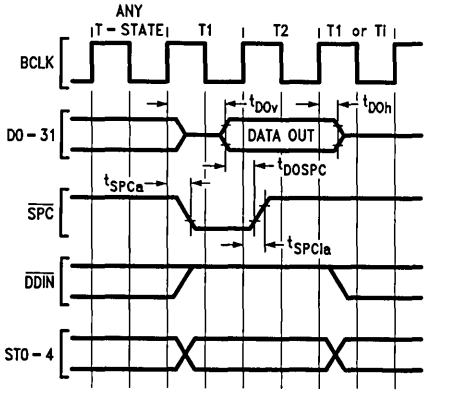
TL/EE/9354-51

FIGURE 4-13. $\overline{\text{HOLD}}$ Acknowledge Timing (Bus Initially Not Idle)



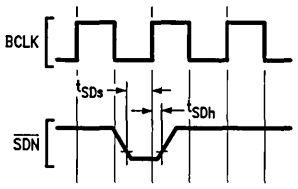
TL/EE/9354-52

FIGURE 4-14. Slave Processor Read Timing



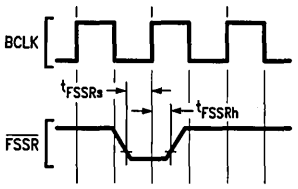
TL/EE/9354-53

FIGURE 4-15. Slave Processor Write Timing



TL/EE/9354-54

FIGURE 4-16. Slave Processor Done

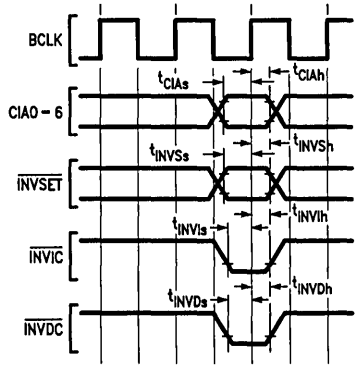


TL/EE/9354-55

FIGURE 4-17. $\overline{\text{FSSR}}$ Signal Timing

2

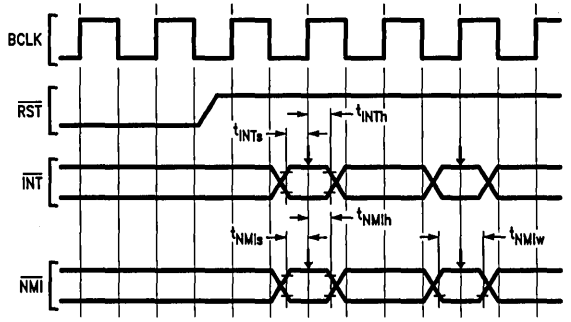
4.0 Device Specifications (Continued)



TL/EE/9354-56

FIGURE 4-18. Cache Invalidation Request

Note 1: CIA0–6 and $\overline{\text{INVSET}}$ are only relevant when $\overline{\text{INVIC}}$ and/or $\overline{\text{INVDC}}$ are asserted.

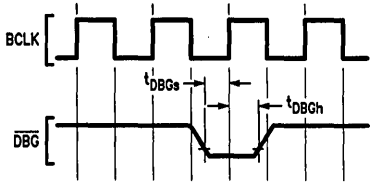


TL/EE/9354-57

FIGURE 4-19. $\overline{\text{INT}}$ and $\overline{\text{NMI}}$ Signals Sampling

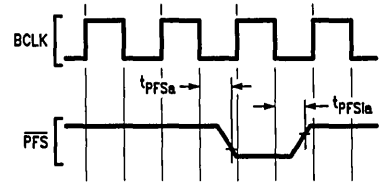
Note 1: $\overline{\text{INT}}$ and $\overline{\text{NMI}}$ are sampled on every other rising edge of BCLK, starting with the second rising edge of BCLK after $\overline{\text{RST}}$ goes high.

Note 2: $\overline{\text{INT}}$ is level sensitive, and once asserted, it should not be deasserted until it is acknowledged.



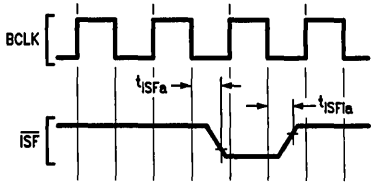
TL/EE/9354-58

FIGURE 4-20. Debug Trap Request



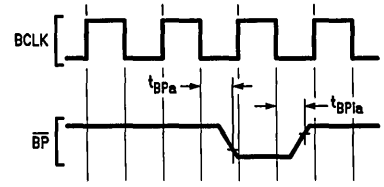
TL/EE/9354-59

FIGURE 4-21. $\overline{\text{PFS}}$ Signal Timing



TL/EE/9354-60

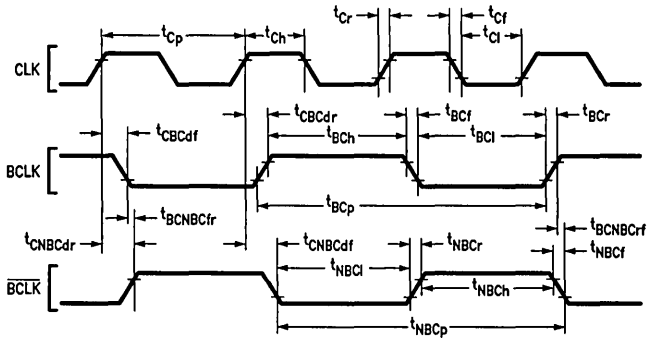
FIGURE 4-22. $\overline{\text{ISF}}$ Signal Timing



TL/EE/9354-61

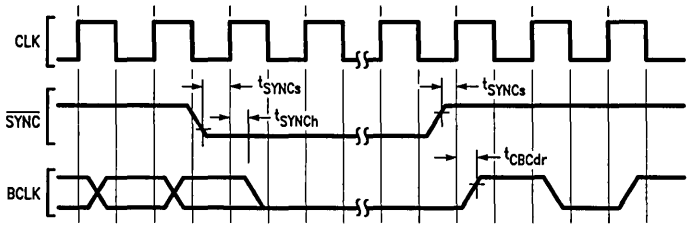
FIGURE 4-23. Break Point Signal Timing

4.0 Device Specifications (Continued)



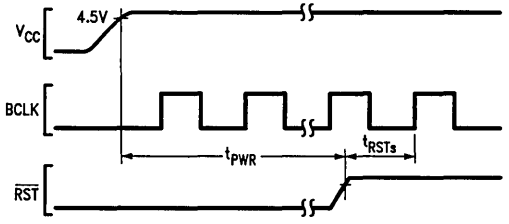
TL/EE/9354-62

FIGURE 4-24. Clock Waveforms



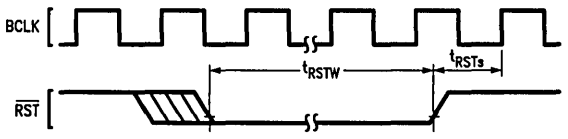
TL/EE/9354-63

FIGURE 4-25. Bus Clock Synchronization



TL/EE/9354-64

FIGURE 4-26. Power-On Reset



TL/EE/9354-65

FIGURE 4-27. Non-Power-On Reset

2

Appendix A: Instruction Formats

NOTATIONS:

i = Integer Type Field
B = 00 (Byte)
W = 01 (Word)
D = 11 (Double Word)

f = Floating Point Type Field
F = 1 (Std. Floating: 32 bits)
L = 0 (Long Floating: 64 bits)

c = Custom Type Field
D = 1 (Double Word)
Q = 0 (Quad Word)

op = Operation Code
Valid encodings shown with each format.

gen, gen 1, gen 2 = General Addressing Mode Field
See Section 2.2 for encodings.

reg = General Purpose Register Number

cond = Condition Code Field
0000 = EQual: Z = 1
0001 = Not Equal: Z = 0
0010 = Carry Set: C = 1
0011 = Carry Clear: C = 0
0100 = HIgher: L = 1
0101 = Lower or Same: L = 0
0110 = Greater Than: N = 1
0111 = Less or Equal: N = 0
1000 = Flag Set: F = 1
1001 = Flag Clear: F = 0
1010 = LOwer: L = 0 and Z = 0
1011 = Higher or Same: L = 1 or Z = 1
1100 = Less Than: N = 0 and Z = 0
1101 = Greater or Equal: N = 1 or Z = 1
1110 = (Unconditionally True)
1111 = (Unconditionally False)

short = Short Immediate value. May contain:
quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
cond: Condition Code (above), in Scond.
areg: CPU Dedicated Register, in LPR, SPR.
0000 = UPSR
0001 = DCR
0010 = BPC
0011 = DSR
0100 = CAR
0101–0111 = (Reserved)
1000 = FP
1001 = SP
1010 = SB
1011 = USP
1100 = CFG
1101 = PSR
1110 = INTBASE
1111 = MOD

Options: in String Instructions

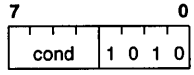
U/W	B	T

T = Translated
B = Backward
U/W = 00: None
01: While Match
11: Until Match

Configuration bits, in SETCFG Instruction:

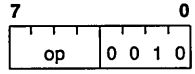
1	1	1	1	C	M	F	I

mreg: MMU Register number, in LMR, SMR.
0000 = } Trap (UND)
•
•
•
0111 = }
1000 = Reserved
1001 = MCR
1010 = MSR
1011 = TEAR
1100 = PTB0
1101 = PTB1
1110 = IVAR0
1111 = IVAR1



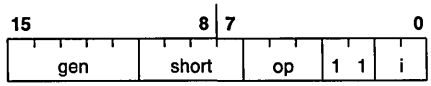
Format 0

Bcond (BR)



Format 1

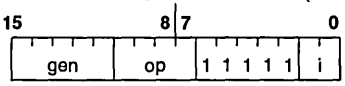
BSR	-0000	ENTER	-1000
RET	-0001	EXIT	-1001
CXP	-0010	NOP	-1010
RXP	-0011	WAIT	-1011
RETT	-0100	DIA	-1100
RETI	-0101	FLAG	-1101
SAVE	-0110	SVC	-1110
RESTORE	-0111	BPT	-1111



Format 2

ADDQ	-000	ACB	-100
CMPQ	-001	MOVQ	-101
SPR	-010	LPR	-110
Scond	-011		

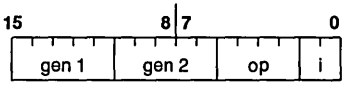
Appendix A: Instruction Formats (Continued)



Format 3

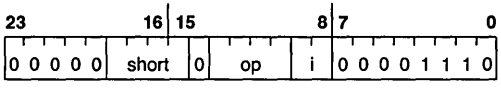
CXPD	-0000	ADJSP	-1010
BICPSR	-0010	JSR	-1100
JUMP	-0100	CASE	-1110
BISPSR	-0110		

Trap (UND) on XXX1, 1000



Format 4

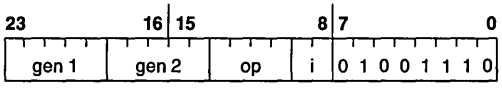
ADD	-0000	SUB	-1000
CMP	-0001	ADDR	-1001
BIC	-0010	AND	-1010
ADDC	-0100	SUBC	-1100
MOV	-0101	TBIT	-1101
OR	-0110	XOR	-1110



Format 5

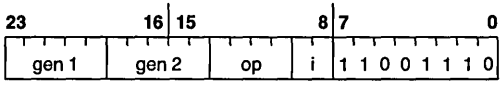
MOVS	-0000	SETCFG	-0010
CMPS	-0001	SKPS	-0011

Trap (UND) on 1XXX, 01XX



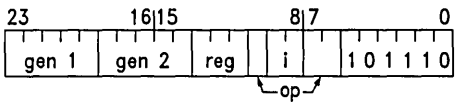
Format 6

ROT	-0000	NEG	-1000
ASH	-0001	NOT	-1001
CBIT	-0010	Trap (UND)	-1010
CBITI	-0011	SUBP	-1011
Trap (UND)	-0100	ABS	-1100
LSH	-0101	COM	-1101
SBIT	-0110	IBIT	-1110
SBITI	-0111	ADDP	-1111



Format 7

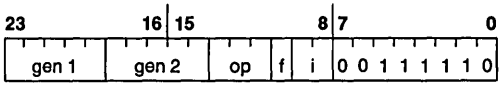
MOVM	-0000	MUL	-1000
CMPM	-0001	MEI	-1001
INSS	-0010	Trap (UND)	-1010
EXTS	-0011	DEI	-1011
MOVXBW	-0100	QUO	-1100
MOVZBW	-0101	REM	-1101
MOVZiD	-0110	MOD	-1110
MOVXiD	-0111	DIV	-1111



TL/EE/9354-66

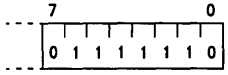
Format 8

EXT	-0 00	INDEX	-1 00
CVTP	-0 01	FFS	-1 01
INS	-0 10		
CHECK	-0 11		
MOVSU	-110, reg = 001		
MOVUS	-110, reg = 011		



Format 9

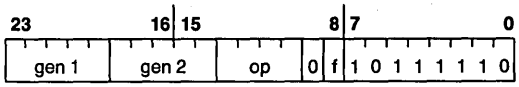
MOVif	-000	ROUND	-100
LFSR	-001	TRUNC	-101
MOVLF	-010	SFSR	-110
MOVFL	-011	FLOOR	-111



TL/EE/9354-67

Format 10

Trap (UND) Always

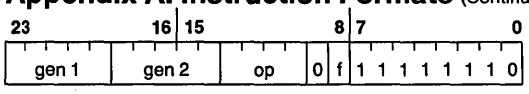


Format 11

ADDf	-0000	DIVf	-1000
MOVf	-0001	Note 1	-1001
CMPf	-0010	Note 3	-1010
Note 3	-0011	Note 1	-1011
SUBf	-0100	MULf	-1100
NEGf	-0101	ABSf	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111

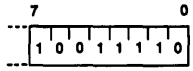
2

Appendix A: Instruction Formats (Continued)



Format 12

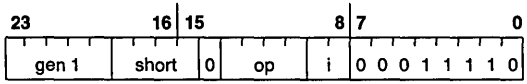
Note 2	-0000	Note 2	-1000
SQRTf	-0001	Note 1	-1001
POLYf	-0010	MACf	-1010
DOTf	-0011	Note 1	-1011
SCALBf	-0100	Note 2	-1100
LOGBf	-0101	Note 1	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111



TL/EE/9354-68

Format 13

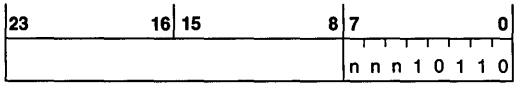
Trap (UND) Always



Format 14

RDVAL	-0000	LMR	-0010
WRVAL	-0001	SMR	-0011
		CINV	-1001

Trap (UND) on 01XX, 1000, 101X, 11XX

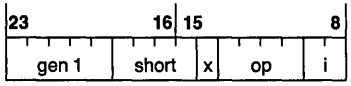


Format 15

(Custom Slave)

nnn **Operation Word Format**

000

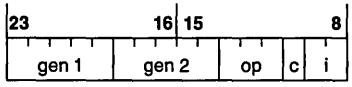


Format 15.0

LCR	-0010
SCR	-0011

Trap (UND) on all others

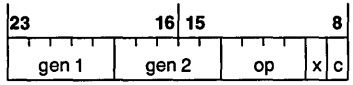
001



Format 15.1

CCV3	-000	CCV2	-100
LCSR	-001	CCV1	-101
CCV5	-010	SCSR	-110
CCV4	-011	CCV0	-111

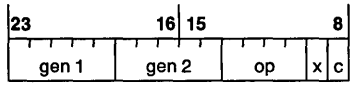
101



Format 15.5

CCAL0	-0000	CCAL3	-1000
CMOV0	-0001	CMOV3	-1001
CCMP0	-0010	Note 3	-1010
CCMP1	-0011	Note 1	-1011
CCAL1	-0100	CCAL2	-1100
CMOV2	-0101	CMOV1	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111

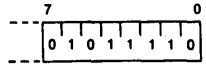
111



Format 15.7

Note 2	-0000	Note 2	-1000
Note 1	-0001	Note 1	-1001
Note 3	-0010	Note 3	-1010
Note 3	-0011	Note 1	-1011
Note 2	-0100	Note 2	-1100
Note 1	-0101	Note 1	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111

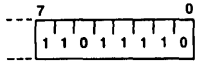
If nnn = 010, 011, 100, 110 then Trap (UND) Always.



TL/EE/9354-69

Format 16

Trap (UND) Always



TL/EE/9354-70

Format 17

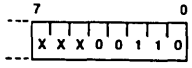
Trap (UND) Always

TL/EE/9354-71

Appendix A: Instruction Formats (Continued)

Format 18

Trap (UND) Always

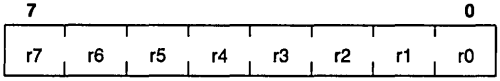


TL/EE/9354–72

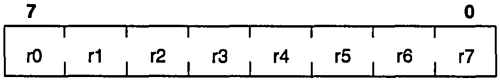
Format 19

Trap (UND) Always

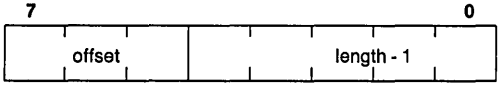
Implied Immediate Encodings:



Register Mark, Appended to SAVE, ENTER



Register Mark, Appended to RESTORE, EXIT



Offset/Length Modifier Appended to INSS, EXTS

Note 1: Opcode not defined; CPU treats like MOV_f or $CMOV_c$. First operand has access class of read; second operand has access class of write; f or c field selects 32- or 64-bit data.

Note 2: Opcode not defined; CPU treats like ADD_f or $CCAL_c$. First operand has access class of read;, second operand has access class of read-modify-write; f or c field selects 32- or 64-bit data.

Note 3: Opcode not defined; CPU treats like CMP_f or $CCMP_c$. First operand has access class of read;, second operand has access class of read; f or c field selects 32- or 64-bit data.

Appendix B. Compatibility Issues

The NS32532 is compatible with the Series 32000 architecture implemented by the NS32032, NS32332, and previous microprocessors in the family. Compatibility means that within certain limited constraints, programs that execute on one of the earlier Series 32000 microprocessors will produce identical results when executed on the NS32532. Compatibility applies to privileged operating systems programs, as well as to non-privileged applications programs. This appendix explains both the restrictions on compatibility with previous Series 32000 microprocessors and the extensions to the architecture that are implemented by the NS32532.

B.1 RESTRICTIONS ON COMPATIBILITY

If the following restrictions are observed, then a program that executes on an earlier Series 32000 microprocessor will produce identical results when executed on the NS32532 in an appropriately configured system:

1. The program is not time-dependent. For example, the program should not use instruction loops to control real-time delays.
2. The program does not use any encodings of instructions, operands, addresses, or control fields identified to be reserved or undefined. For example, if the count operand's value for an LSHi instruction is not within the range specified by the *Series 32000 Instruction Set Reference Manual,* then the results produced by the NS32532 may differ from those of the NS32032.
3. Either the program does not depend on the use of a Memory Management Unit (MMU), or it is written for operation with the NS32382 MMU and does not use the bus-error or debugging features of the NS32382.
4. The program does not depend on the detection of bus errors according to the implementation of the NS32332. For example, the NS32532 distinguishes between restartable and nonrestartable bus errors by transferring control to the appropriate bus-error exception service procedure through one of two distinct entries in the Interrupt Dispatch Table. In contrast, the NS32332 uses a single entry in the Interrupt Dispatch Table for all bus errors.
5. The program does not modify itself. Refer to Section B.4 for more information.
6. The program does not depend on the execution of certain complex instructions to be non-interruptible. Refer to Section B.5 on. "Memory-Mapped I/O" for more information.
7. The program does not use the custom slave instructions CATSTO and CATST1, as they are not supported by the NS32532 and will result in a Trap (UND) when their execution is attempted.

B.2 ARCHITECTURE EXTENSIONS

The NS32532 implements the following extensions of the Series 32000 architecture using previously reserved control bits, instruction encodings, and memory locations. Extensions implemented earlier in the NS32332, such as 32-bit addressing, are not listed.

1. The DC, LDC, IC, and LIC bits in the CFG register have been defined to control the on-chip Instruction and Data Caches. The DE-bit in the CFG register has been defined to enable Direct-Exception Mode.
2. The V-flag in the PSR register has been defined to enable the Integer-Overflow Trap.
3. The DCR, BPC, DSR, and CAR registers have been defined to control debugging features. Access to these registers has been added to the definition of the LPR and SPR instructions.
4. Access to the CFG and SP1 registers has been added to the definition of the LPR and SPR instructions.
5. The CINV instruction has been defined to invalidate control of the on-chip Instruction and Data Caches.
6. Direct-Exception Mode has been added to support faster interrupt service time and systems without module tables.
7. A new entry has been added to the Interrupt Dispatch Table for supporting vectors to distinguish between restartable and nonrestartable bus errors. Two additional entries support Trap (OVF) and Trap (DBG).
8. Restrictions have been eliminated for recovery from Trap (ABT) for operands with access class of write that cross page boundaries. Restrictions still exist however, for the operands of the MOVMi instruction.

B.3 INTEGER OVERFLOW TRAP

A new trap condition is recognized for integer arithmetic overflow. Trap (OVF) is enabled by the V-flag in the PSR. This new trap is important because detection of integer overflow conditions is required for certain programming languages, such as ADA, and the PSR flags do not indicate the occurrence of overflow for ASHi, DIVi and MULi instructions.

Appendix B. Compatibility Issues (Continued)

More details on integer overflow are given in Section 3.2.5, where a description of all the cases in which an overflow condition is detected is also provided.

INTEGER ARITHMETIC

The V-flag in the PSR enables Trap (OVF) to occur following execution of an integer arithmetic instruction whose result cannot be represented exactly in the destination operand's location.

If the number of bits required to represent the resulting quotient of a DEI instruction exceeds half the number of bits of the destination, then the contents of both the quotient and remainder stored in the destination are undefined.

The ADDR instruction can be used in place of integer arithmetic instructions to perform certain calculations. In this case however, integer overflow is not detected by the CPU.

LOGICAL INSTRUCTIONS

The V-flag in the PSR enables Trap (OVF) to occur following execution of an ASHi instruction whose result cannot be represented exactly in the destination operand's location.

ARRAY INSTRUCTIONS

The V-flag in the PSR enables Trap (OVF) to occur following execution of a CHECKi instruction whose source operand is out of bounds.

PROCESSOR CONTROL INSTRUCTIONS

The V-flag in the PSR enables Trap (OVF) to occur following execution of an ACBi instruction if the sum of the "inc" value and the "index" operand cannot be represented exactly in the "index" operand's location.

B.4 SELF-MODIFYING CODE

The Series 32000 architecture does not have special provisions to optimally support self-modifying programs. Nevertheless, on the NS32332 and previous Series 32000 microprocessors it is possible to execute self-modifying code according to the following sequence:

1. Modify the appropriate instruction.
2. Execute a JUMP instruction or other instruction that causes the microprocessor's instruction queue to be flushed.
3. Execute the modified instruction.

For example, an interactive debugger may follow the sequence above after reaching a breakpoint in a program being monitored.

The same program may not produce identical results when executed on the NS32532 due to effects of the Instruction Cache and branch prediction. In order to execute self-modifying code on the NS32532 it is necessary to do the following:

1. Modify the appropriate instruction.
2. If the modified instruction is on a cacheable page, execute CINV to invalidate the contents of the Instruction Cache.
3. Execute an instruction that causes a serializing operation. See Section 3.1.3.3.
4. Execute the modified instruction.

B.5 MEMORY-MAPPED I/O

As was mentioned in Section 3.1.3.2, certain peripheral devices exhibit characteristics identified as "destructive-reading" and "side-effects of writing" that impose requirements for special handling of memory-mapped I/O references. The NS32532 supports two methods to use on references to memory-mapped peripheral devices that exhibit either or both of these characteristics.

For peripheral devices that exhibit only side-effects of writing, correct operation can be ensured either by locating the device between addresses FF000000 (hex) and FF7FFFFF (hex) in the virtual address space or by observing the first 2 restrictions listed below. For peripheral devices that exhibit destructive-reading, all the following restrictions must be observed to ensure correct operation:

1. References to the device must be inhibited while the CPU asserts the output signal $\overline{\text{IOINH}}$.
2. The input signal $\overline{\text{IODEC}}$ must be asserted by the system on references to the device.
3. The device cannot be used for instruction fetches, reads of effective addresses, or Page Table Entries.
4. If an instruction that reads a source operand from the device crosses a page boundary, then no Trap (ABT) or restartable bus error can occur during fetches from the page with higher addresses.
5. No Trap (ABT) for a data reference or other exception can occur during execution of an instruction that reads a source operand from the device. (Exceptions that are recognized after completion of an instruction, like Trap (OVF) and Trap (DBG), cause no problem.)
6. The device can be used as a source operand only for instructions in the list below.

ABSi	CBITi	MOVMi	SBITIi
ADDi	CBITIi	MOVXi	SUBi
ADDCi	CMPi	MOVZi	SUBCi
ADDPi	CMPQi	NEGi	SUBPi
ADDQi	COMi	NOTi	TBITi
ANDi	IBITi	ORi	XORi
ASHi	LSHi	ROTi	
BICi	MOVi	SBITi	

This restriction arises because the CPU can respond to interrupt requests during the execution of complex instruction in order to reduce interrupt latency. Thus, the CPU may read the source operands for a DEID instruction (extended-precision divide), begin calculating the instruction's results, and then respond to an interrupt request before completing the instruction. In such an event, the instruction can be executed again and completed correctly after the interrupt service procedure returns unless one of the source operands was altered by destructive-reading.

Appendix C. Instruction Set Extensions

The following sections describe the differences and extensions to the Series 32000 instruction set (as presented in the "Series 32000 Instruction Set Reference Manual") implemented by the NS32352.

No changes or additions have been made to the user-mode instruction set, and only a few privileged instructions have been added.

Appendix C. Instruction Set Extensions (Continued)

C.1 PROCESSOR SERVICE INSTRUCTIONS

The CFG register, User Stack Pointer (SP1), and Debug Registers can be loaded and stored using privileged forms of the LPRi and SPRi instructions.

When the SETCFG instruction is executed, the CFG register bits 0 through 3 are loaded from the instruction's short field, bits 4 through 7 are forced to 1, and bits 8 through 12 are forced to 0.

The contents of the on-chip Instruction Cache and Data Cache can be invalidated by executing the privileged instruction CINV. While executing the CINV instruction, the CPU generates 2 slave bus cycles on the system interface to display the first 3 bytes of the instruction and the source operand. External circuitry can thereby detect the execution of the CINV instruction for use in monitoring the contents of the on-chip caches.

C.2 MEMORY MANAGEMENT INSTRUCTIONS

The NS32532 on-chip MMU does not implement the BAR, BDR, BEAR, and BMR registers of the NS32382. These registers are used in the NS32382 to support bus error and debugging features. When an attempt is made to access one of these 4 registers by executing an LMR or SMR instruction, a Trap (UND) occurs. More generally, a Trap (UND) occurs whenever an attempt is made to execute an LMR or SMR instruction and the most-significant bit of the short-field is 0.

While executing an LMR instruction, the CPU generates 2 slave bus cycles on the system interface to display the first 3 bytes of the instruction and the source operand. External circuitry can thereby detect the execution of an LMR instruction for use in monitoring the contents of the on-chip Translation Lookaside Buffer.

Like the NS32382 MMU, the F-flag in the PSR is set and no Trap (ABT) occurs when a RDVAL or WRVAL instruction is executed and the Protection Level in the Level-1 Page Table Entry indicates that the access is not allowed. In the NS32082 MMU, an abort occurs when the Level-1 PTE is invalid, regardless of the Protection Level.

C.3 INSTRUCTION DEFINITIONS

This section provides a description of the operations and encodings of the new NS32532 privileged instructions.

Load and Store Processor Registers

Syntax:	**LPRi**	**procreg,**	**src**
		short	gen
			read.i
	SPRi	**procreg**	**dest**
		short	gen
			write.i

The LPRi and SPRi instructions can be used to load and store the User Stack Pointer (USP or SP1), the Configuration Register (CFG) and the Debug Registers in addition to the Processor Registers supported by the previous Series 32000 CPUs. Access to these registers is privileged.

Figure C-1 and Table C-1 show the instruction formats and the new 'short' field encodings for LPRi and SPRi.

Flags Affected: No flags affected by loading or storing the USP, CFG, or Debug Registers.

Traps: Illegal Instruction Trap (ILL) occurs if an attempt is made to load or store the USP, CFG or Debug Registers while the U-flag is 1.

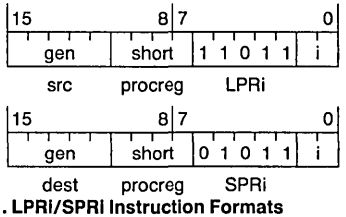


FIGURE C-1. LPRi/SPRi Instruction Formats

TABLE C-1. LPRi/SPRi New 'Short' Field Encodings

Register	procreg	short field
Debug Condition Register	DCR	0001
Breakpoint Program Counter	BPC	0010
Debug Status Register	DSR	0011
Compare Address Register	CAR	0100
User Stack Pointer	USP	1011
Configuration Register	CFG	1100

Cache Invalidate

Syntax: **CINV** **options, src**

gen

read. D

The CINV instruction invalidates the contents of locations in the on-chip Instruction Cache and Data Cache. The instruction can be used to invalidate either the entire contents of the on-chip caches or only a 16-byte block. In the latter case, the 28 most-significant bits of the source operand specify the physical address of the aligned 16-byte block; the 4 least-significant bits of the source operand are ignored. If the specified block is not located in the on-chip caches, then the instruction has no effect. If the entire cache contents is to be invalidated, then the source operand is read, but its value is ignored.

Options are specified by listing the letters A (invalidate All), I (Instruction Cache), and D (Data Cache). If neither the I nor D option is specified, the instruction has no effect.

In the instruction encoding, the options are represented in the A, I, and D fields as follows:

A: 0—invalidate only a 16-byte block
1—invalidate the entire cache

I: 0—do not affect the Instruction Cache
1—invalidate the Instruction Cache

D: 0—do not affect the Data Cache
1—invalidate the Data Cache

Flags Affected: None

Traps: Illegal Operation Trap (ILL) occurs if an attempt is made to execute this instruction while the U-flag is 1.

Examples:

1. CINV A, D, I, R3 1E A7 1B
2. CINV I, R3 1E 27 19

Example 1 invalidates the entire Instruction Cache and Data Cache.

Example 2 invalidates the 16-byte block whose physical address in the Instruction Cache is contained in R3.

Appendix C. Instruction Set Extensions (Continued)

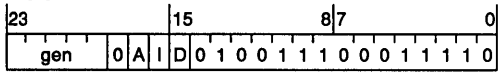


FIGURE C-2. CINV Instruction Format

Load and Store Memory Management Register

Syntax: **LMR** **mmreg,** **src**
short gen
read.D

SMR **mmureg,** **dest**
short gen
write.D

The LMR and SMR instruction load and store the on-chip MMU registers as 32-bit quantities to and from any general operand. For reasons of system security, these instructions are privileged. In order to be executable, they must also be enabled by setting the M bit in the CFG register.

The instruction formats as well as the 'short' field encodings are shown in *Figure C-3* and Table C-2 respectively.

It is to be noted that the IVAR0 and IVAR1 registers are write-only, and as such, they can only be loaded by the LMR instruction.

Flags Affected: none

Traps: Undefined Instruction Trap (UND) occurs if an attempt is made to execute this instruction while either of the following conditions is true:

1. The M-bit in the CFG register is 0.
2. The U-Flag in the PSR is 0 and the most-significant bit of the short field is 0.

Illegal Instruction Trap (ILL) occurs if an attempt is made to execute this instruction while the M-bit in the CFG register and the U-flag in the PSR are both 1.

TABLE C-2. LMR/SMR 'Short' Field Encodings

Register	mmureg	short field
Memory Management Control Reg	MCR	1001
Memory Management Status Reg	MSR	1010
Translation Exception Address Reg	TEAR	1011
Page Table Base Register 0	PTB0	1100
Page Table Base Register 1	PTB1	1101
Invalidate Virtual Address 0	IVAR0	1110
Invalidate Virtual Address 1	IVAR1	1111

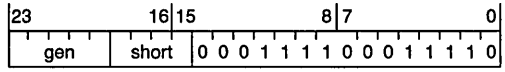
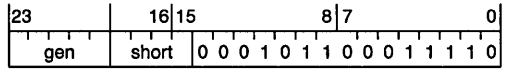


FIGURE C-3. LMR/SMR Instruction Formats



Appendix D. Instruction Execution Times

The NS32532 achieves its performance by using an advanced implementation incorporating a 4-stage Instruction Pipeline, a Memory Management Unit, an Instruction Cache and a Data Cache into a single integrated circuit.

As a consequence of this advanced implementation, the performance evaluation for the NS32532 is more complex than for the previous microprocessors in the Series 32000 family. In fact, it is no longer possible to determine the execution time for an instruction using only a set of tables for operations and addressing modes. Rather, it is necessary to consider dependencies between the various instructions executing in the pipeline, as well as the occurrence of misses for the on-chip caches.

The following sections explain the method to evaluate the performance of the NS32532 by calculating various timing parameters for an instruction sequence. Due to the high degree of parallelism in the NS32532, the evaluation techniques presented here include some simplifications and approximations.

D.1 INTERNAL ORGANIZATION AND INSTRUCTION EXECUTION

The NS32532 is organized internally as 8 functional units as shown in *Figure 1*. The functional units operate in parallel to execute instructions in the 4-stage pipeline. The structure of this pipeline is shown in *Figure 3-2*. The Instruction Fetch and Instruction Decode pipeline stages are implemented in the loader along with the 8-byte instruction queue and the buffer for a decoded instruction. The Address Calculation pipeline stage is implemented in the address unit. The Execute pipeline stage is implemented in the Execution Unit along with the write data buffer that holds up to two results directed to memory.

The Address Unit and Execution Unit can process instructions at a peak rate of 2 clock cycles per instruction, enabling a sustained pipeline throughput at 30 MHz of 15 MIPS (million instructions per second) for sequences of register-to-register, immediate-to-register, register-to-memory and memory-to-register instructions. Nevertheless, the execution of instructions in the pipeline is reduced from the peak throughput of 2 cycles by the following causes of delay:

1. Complex operations, like division, require more than 2 cycles in the Execution Unit, and complex addressing modes, like memory relative, require more than 2 cycles in the Address Unit.

Appendix D. Instruction Execution Times (Continued)

2. Dependencies between instructions can limit the flow through the pipeline. A data dependency can arise when the result of one instruction is the source of a following instruction. Control dependencies arise when branching instructions are executed. Section D.3 describes the types of instruction dependencies that impact performance and explains how to calculate the pipeline delays.
3. Cache and TLB misses can cause the flow of instructions through the pipeline to be delayed, as can non-aligned references. Section D.4 explains the performance impact for these forms of storage delays.

The effective time T_{eff} needed to execute an instruction is given by the following formula:

$$T_{eff} = T_e + T_d + T_s$$

T_e is the execution time in the pipeline in the absence of data dependencies between instructions and storage delays, T_d is the delay due to data dependencies, and T_s is the effect of storage delays.

D.2 BASIC EXECUTION TIMES

Instruction flow in sequence through the pipeline stages implemented by the Loader, Address Unit, and Execution Unit. In almost all cases, the Loader is at least as fast at decoding an instruction as the Address Unit is at processing the instruction. Consequently, the effects of the Loader can be ignored when analyzing the smooth flow of instructions in the pipeline, and it is only necessary to consider the times for the Address Unit and Execution Unit. The time required by the Loader to fetch and decode instructions is significant only when there are control dependencies between instructions or Instruction Cache misses, both of which are explained later.

The time for the pipeline to advance from one instruction to the next is typically determined by the maximum time of the Address Unit and Execution Unit to complete processing of the instruction on which they are operating. For example, if the Execution Unit is completing instruction *n* in 2 cycles and the Address Unit is completing instruction *n+1* in 4 cycles, then the pipeline will advance in 4 cycles. For certain instructions, such as RESTORE, the Address Unit waits until the Execution Unit has completed the instruction before proceeding to the next instruction. When such an instruction is in the Execution Unit, the time for the pipeline to advance is equal to the sum of the time for the Execution Unit to complete instruction *n* and the time for the Address Unit to complete instruction *n+1*. The processing times for the Loader, Address Unit, and Execution Unit are explained below.

D.2.1 Loader Timing

The Loader can process an instruction field on each clock cycle, where a *field* is one of the following:

- An opcode of 1 to 3 bytes including addressing mode specifiers.
- Up to 2 index bytes, if scaled index addressing mode is used.
- A displacement.
- An immediate value of 8, 16 or 32 bits.

The Loader requires additional time in the following cases:

- 1 additional cycle when 2 consecutive double-word fields begin at an odd address.
- 2 cycles in total to process a double-precision floating-point immediate value.

D.2.2 Address Unit Timing

The processing time of the Address Unit depends on the instruction's operation and the number and type of its general addressing modes. The basic time for most instructions is 2 cycles. A relatively small number of instructions require an additional address unit time, as shown in the timing tables in Section D.5.5. Non-pipelined floating-point instructions as well as Custom-Slave instructions require an additional 3 cycles plus 2 cycles for each quad-word operand in memory.

For instructions with 2 general addressing modes, 2 additional cycles are required when both addressing modes refer to memory. Certain general addressing modes require an additional processing time, as shown in Table D-1. For example, the instruction **MOVD 4(8(FP)), TOS** requires 7 cycles in the Address Unit; 2 cycles for the basic time, an additional 2 cycles because both modes refer to memory, and an additional 3 cycles for Memory Relative addressing mode.

TABLE D-1. Additional Address Unit Processing Time for Complex Addressing Modes

Mode	Additional Cycles
Memory Relative	3
External	8
Scaled Indexing	2

D.2.3 Execution Unit Timing

The Execution Unit processing times for the various NS32532 instructions are provided in Section D.5.5. Certain operations cause a break in the instruction flow through the pipeline.

Some of these operation simply stop the Address Unit, while others flush the instruction queue as well. The information on how to evaluate the penalty resulting from instruction flow breaks is provided in the following sections.

D.3 INSTRUCTION DEPENDENCIES

Interactions between instructions in the pipeline can cause delays. Two types of interactions can arise, as described below.

D.3.1 Data Dependencies

In certain circumstances the flow of instructions in the pipeline will be delayed when the result of an instruction is used as the source of a succeeding instruction. Such interlocks are automatically detected by the microprocessor and handled with complete transparency to software.

D.3.1.1 Register Interlocks

When an instruction uses a base register that is the destination of either of the previous 2 instructions, a delay occurs. The delay is 3 cycles when, as in the following example, the base register is modified by the immediately preceding instruction. Modifications of the Stack Pointer resulting from the use of TOS addressing mode do not cause any delay. Also, there is no delay for a data dependency when the instruction that modifies the register is one for which the Address Unit stops.

Appendix D. Instruction Execution Times (Continued)

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD 4(R0),R2  ; R0 is base register,
                      delay 3 cycles
```

The delay is 1 cycle when the register is modified 2 instructions before its use as a base register, as shown in this example.

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD 4(SP),R3  ; R0 not used
n+2: MOVD 4(R0),R2  ; R0 is base register,
                      delay 1 cycle
```

When an instruction uses an index register that is the destination of the previous instruction, a delay of 1 cycle occurs, as shown in the example below. If the register is modified 2 or more instructions prior to its use as an index register, then no delay occurs.

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD 4(SP)[R0:B],R2
                    ; R0 is index register,
                      delay 1 cycle
```

Bypass circuitry in the Execution Unit generally avoids delay when a register modified by one instruction is used as the source operand of the following instruction, as in the following example.

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD R0,R2     ; R0 is source register,
                      no delay
```

For the uncommon case where the operand in the source register is larger than the destination of the previous instruction, a delay of 2 cycles occurs. Here is an example.

```
  n: ADDB R1,R0     ; modify byte in R0
n+1: MOVD R0,R2     ; R0 dw source operand,
                      2 cycle delay
```

Note: The Address Unit does not make any differentiation between CPU and FPU registers. Therefore, register interlocks can occur between integer and floating-point instructions.

D.3.1.2 Memory Interlocks

When an instruction reads a source operand (or address for effective address calculation) from memory that depends on the destination of either of the previous 2 instructions, a delay occurs. The CPU detects a dependency between a read and a write reference in the following cases, which include some false dependencies in addition to all actual dependencies:

- Either reference crosses a double-word boundary
- Address bits 0 through 11 are equal
- Address bits 2 through 11 are equal and either reference is for a word
- Address bits 2 through 11 are equal and either reference is for a double-word

The delay for a memeory interlock is 4 cycles when, as in the following example, the memory location is modified by the immediately preceding instruction.

```
  n: ADDQD 1,4(SP)  ; modify 4(SP)
n+1: CMPD 10,4(SP)  ; read, 4(SP),
                      4 cycle delay
```

The delay is 2 cycles when the memory location is modified 2 instructions before its use as a source operand or effective address, as shown in this example.

```
  n: ADDQD 1,4(SP)  ; modify 4(SP)
n+1: MOVD R0,R1     ; no reference to 4(SP)
n+2: CMPD 10, 4(SP) ; read 4(SP),
                      2 cycles delay
```

Certain sequences of read and write references can cause a delay of 1 cycle although there is no data dependency between the references. This arises because the Data Cache is occupied for 2 cycles on write references. In the absence of data dependencies, read references are given priority over write references. Therefore, this delay only occurs when an instruction with destination in memory is followed 2 instructions later by an instruction that refers to memory (read or write) and 3 instructions later by an instruction that reads from memory. Here is an example:

```
  n: MOVD R0,4(SP)  ; memory write
n+1: MOVD R6,R7     ; any instruction
n+2: MOVD 8(SP),R0  ; memory read or write
n+3: MOVD 12(SP),R1 ; memory read
                      delayed 1 cycle
```

D.3.2 Control Dependencies

The flow of instructions through the pipeline is delayed when the address from which to fetch an instruction depends on a previous instruction, such as when a conditional branch is excuted. The Loader includes special circuitry to handle branch instructions (ACB, BR, Bcond, and BSR) that serves to reduce such delays. When a branch instruction is decoded, the Loader calculates the destination address and selects between the sequential and non-sequential instruction streams. The non-sequential stream is selected for unconditional branches. For conditional branches the selection is based on the branch's direction (forward or backward) as well as the tested condition. The branch is predicted taken in any of the following cases.

- The branch is backward.
- The tested condition is either NE or LE.

Measurements have shown that the correct stream is selected for 64% of conditional branches and 71% of total branches.

If the Loader selects the non-sequential stream, then the destination address is transferred to the Instruction Cache. For conditional branches, the Loader saves the address of the alternate stream (the one not selected). When a conditional branch instruction reaches the Execution Unit, the condition is resolved, and the Execution Unit signals the Loader whether or not the branch was taken. If the branch had been incorrectly predicted, the Instruction Cache begins fetching instructions from the correct stream.

The delay for handling a branch instruction depends on whether the branch is taken and whether it is predicted correctly. Unconditional branches have the same delay as correctly predicted, taken conditional branches.

Another form of delay occurs when 2 consecutive conditional branch instructions are executed. This delay of 2 cycles arises from contention for the register that holds the alternate stream address in the Loader.

Control dependencies also arise when JUMP, RET, and other non-branch instructions alter the sequential execution of instructions.

Appendix D. Instruction Execution Times (Continued)

D.4 STORAGE DELAYS

The flow of instructions in the pipeline can be delayed by off-chip memory references that result from misses in the on-chip storage buffers and by misalignment of instructions and operands. These considerations are explained in the following sections. The delays reported assume no wait states on the external bus and no interference between instruction and data references.

D.4.1 Instruction Cache Misses

An Instruction Cache miss causes a 5 cycle gap in the fetching of instructions. When the miss occurs for a non-sequential instruction fetch, the pipeline is idle for the entire gap, so the delay is 5 cycles. When the miss occurs for a sequential fetch, the pipeline is not idle for the entire gap because instructions that have been prefetched ahead and buffered can be executed. The delay for misses on non-sequential instruction fetches can be estimated to be approximately half the gap, or 2.5 cycles.

D.4.2 Data Cache Misses

A Data Cache miss causes a delay of 2 cycles. When a burst read cycle is used to fill the cache block, then 3 additional cycles are required to update the Data Cache. In case a burst cycle is used and either of the 2 instructions following the instruction that caused the miss also reads from memory, then an additional delay occurs: 3 cycle delay when the instruction that reads from memory immediately follows the miss, and 2 cycle delay when the memory read occurs 2 instructions after the miss.

D.4.3 TLB Misses

There is a delay for the MMU to translate a virtual address whenever there is a TLB miss for an instruction fetch, data read or data write and whenever the M-bit in the Page Table Entry (PTE) must be set for a data write that hits in the TLB. The delay for the MMU to handle a TLB miss is 15 cycles when no update to the PTEs is necessary. When only the Level-1 PTE must be updated, the delay is 17 cycles; when only the Level-2 PTE must be updated, the delay is 22 cycles. When both PTEs must be updated, the delay is 24 cycles.

D.4.4 Instruction and Operand Alignment

When a data reference (either read or write) crosses a double-word boundary, there is a delay of 2 cycles.

When the opcode for a non-sequential instruction crosses a double-word boundary, there is a delay of 1 cycle. No delay occurs in the same situation for a sequential instruction. There is also a delay of 2 cycles when an instruction fetch is located on a different page from the previous fetch and there is a hit in the Instruction Cache. This delay, which is due to the time required to translate the new page's address, also occurs following any serializing operation.

D.5 EXECUTION TIME CALCULATIONS

This section provides the necessary information to calculate the T_e portion of the effective time required by the CPU to execute an instruction.

The effects of data dependencies and storage delays are not taken into account in the evaluation of T_e, rather, they should be separately evaluated through a careful examination of the instruction sequence.

The following assumptions are made:

— The entire instruction, with displacements and immediate operands, is present in the instruction queue when needed.

— All memory operands are available to the Execution Unit and Address Unit when needed.

— Memory writes are performed at full speed through the write buffer.

— Where possible, the values of operands are taken into consideration when they affect instruction timing, and a range of times is given. When this is not done, the worst case is assumed.

D.5.1 Definitions

T_{eu} Time required by the Execution Unit to execute an instruction.

T_{au} Total processing time in the Address Unit.

T_{ad} Extra time needed by the Address Unit, in addition to the basic time, to process more complex cases. T_{ad} can be evaluated as follows:

$T_{ad} = T_x + T_{y1} + T_{y2}$

$T_x = 2$ if the instruction has two general operands and both of them are in memory.

0 otherwise.

T_{y1} and T_{y2} are related to operands 1 and 2 respectively. Their values are given below.

$T_{y(1,2)} = 3$ if Memory Relative

8 if External

2 if Scaled Indexing

0 if any other addressing mode

The following parameters are only used for floating-point execution time calculations.

T_{anp} Additional Address Unit time needed to process floating-point instructions in non-pipelined mode. (Section D.2.2).

T_{anp} may be totally hidden for pipelined instructions. For non-pipelined instructions it can be calculated as follows:

$T_{anp} = 3 + 2 *$ (Number of 64-bit operands in memory)

T_{tcs} Time required to transfer ID and Opcode, if no operand needs to be transferred to the slave. Otherwise, it is the time needed to transfer the last 32 bits of operand data to the slave. In the latter case the transfer of ID and Opcode as well as any operand data except the last 32 bits is included in the Execution Unit timing.

T_{tsc} Time required by the CPU to complete the floating-point instruction upon receiving the DONE signal from the slave. This includes the time to process the DONE signal itself in addition to the time needed to read the result (if any) from the slave.

Appendix D. Instruction Execution Times (Continued)

I This parameter is related to the floating-point operand size as follows:

Standard floating (32 bits): I = 0

Long floating (64 bits): I = 1

D.5.2 Notes on Table Use

1. In the T_{eu} column the notation n1 → n2 means n1 minimum, n2 maximum.
2. In the notes column, notations held within angle brackets <> indicate alternatives in the operand addressing modes which affect the execution time. A table entry which is affected by the operand addressing may have multiple values, corresponding to the alternatives. This addressing notations are:
   - <I> Immediate
   - <R> CPU register
   - <M> Memory
   - <F> FPU register, either 32 or 64 bits
   - <m> Memory, except Top of Stack
   - <T> Top of Stack
   - <x> Any addressing mode
   - <ab> a and b represent the addressing modes of operands 1 and 2 respectively. Both of them can be any addressing mode. (e.g., <MR> means memory to CPU register).
3. The notation 'Break K' provides pipeline status information after executing the instruction to which 'Break K' applies. The value of K is interpreted as follows:
   - K = 0 The Address Unit was stopped by the instruction but the pipeline was not flushed. The Address Unit can start processing the next instruction immediately.
   - K > 0 The pipeline was flushed by the instruction. The Address Unit must wait for K cycles before it can start processing the next instruction.
   - K < 0 The Address Unit was stopped at the beginning of the instruction but it was restarted |K| cycles before the end of it. The Address Unit can start processing the next instruction |K| cycles before the end of the instruction to which 'Break K' applies.
4. Some instructions must wait for pending writes to complete before being able to execute. The number of cycles that these instructions must wait for, is between 6 and 7 for the first operand in the write buffer and 2 for the second operand, if any.
5. The CBITIi and SBITIi instructions will execute a RMW access after waiting for pending writes. The extra time required for the RMW access is only 3 cycles since the read portion is overlapped with the time in the Execution Unit.
6. The keyword defined for the Bcond instruction have the following meaning:
   - BTPC Branch Taken, Predicted Correctly
   - BTPI Branch Taken, Predicted Incorrectly
   - BNTPC Branch Not Taken, Predicted Correctly
   - BNTPI Branch Not Taken, Predicted Incorrectly

D.5.3 T_{eff} Evaluation

The T_e portion of the effective execution time for a certain instruction in an instruction sequence is obtained by performing the following steps:

1. Label the current and previous instruction in the sequence with n and n−1 respectively.
2. Obtain from the tables the values of T_{eu} and T_{au} for instruction n and T_{eu} for instruction n−1.
3. For floating-point instructions, obtain the values of T_{tcs} and T_{tsc}.
4. Use the following formula to determine the execution time T_e.

   $T_e = T_{dpf}(n) + \text{func}\ (T_{au}(n), T_{eu}(n-1), T_{flt}(n-1), \text{Break}\ (n-1)) + T_{eu}(n) + T_{flt}(n)$

   T_{dpf} is the delay incurred before an instruction can begin execution. It must be considered only when the floating-point pipelined mode is enabled.

   For a non-floating-point instruction, it represents the time needed to complete all the instructions in the FIFO. For a floating-point instruction, it is only relevant if the FIFO is full, and represents the time to complete the first instruction in the FIFO.

   func provides the amount of processing time in the Address Unit that cannot be hidden. Its definition is given below.

func = 0	if $T_{au}(n) \le (T_{eu}(n-1) + T_{flt}(n-1))$ AND NOT Break (n−1)
$T_{au}(n) - T_{eu}(n-1)$	if $T_{au}(n) > (T_{eu}(n-1) + T_{flt}(n-1))$ AND NOT Break (n−1)
$T_{au}(n) + K$	if $(T_{au}(n) + K) > 0$ AND Break (n−1)
0	if $(T_{au}(n) + K) \le 0$ AND Break (n−1)

   K is the value associated with Break (n−1).

Appendix D. Instruction Execution Times (Continued)

T_{flt} only applies to floating-point instructions and is always 0 for other instructions. It is evaluated as follows:

if pipelined mode is disabled, then

$T_{flt} = t_{tcs} + T_{tsc} + T_{fpu}$

else

Tflt = 0 if group A instruction.

$\max(T_{prv}, T_{tcs}) + T_{tsc}$ if group B instruction.

T_{fpu} is the execution time in the Floating-Point Unit. T_{prv} is the time needed by the CPU and FPU to complete all the floating-point instructions in the FIFO.

5. Calculate the total execution time T_{eff} by using the following formula:

$$T_{eff} = T_e + T_d + T_s$$

Where T_d and T_s are dependent on the instruction sequence, and can be obtained using the information provided in Section D.4.

D.5.4 Instruction Timing Example

This section presents a simple instruction timing example for a procedure that recursively evaluates the Fibonacci function. In this example there are no data dependencies or storage buffer misses; only the basic instruction execution times in the pipeline, control dependencies, and instruction alignment are considered.

The following is the source of the procedure in C.

```
unsigned fib(x)
int   x;
{
      if (x > 2)
            return (fib(x-1) + fib(x-2));
      else
            return(1);
}
```

The assembly code for the procedure with comments indicating the execution time is shown below. The procedure requires 26 cycles to execute when the actual parameter is less than or equal to 2 (branch taken) and 99 cycles when the actual parameter is equal to 3 (recursive calls).

```
_fib:   movd    r3,tos     ; 2 cycles
        movd    r4,tos     ; 2 cycles
        movd    r1,r3      ; 2 cycles
        cmpqd   $(2),r3    ; 2 cycles
        bge     .L1        ; 2 cycles,  Break 4 If Branch Taken
        movd    r3,r1      ; 2 cycles
        addqd   $(-2),r1   ; 2 cycles
        bsr     _fib       ; 3 cycles
        movd    r0,r4      ; 4 cycles + 4 Cycles due to RET
        movd    r3,r1      ; 2 cycles
        addqd   $(-1),r1   ; 2 cycles
        bsr     _fib       ; 3 cycles
        addd    r4,r0      ; 4 cycles + 1 cycle alignment + 4 cycles due to RET
        movd    tos,r4     ; 2 cycles
        movd    tos,r3     ; 2 cycles
        ret     $(0)       ; 4 cycles, break 4
        .align 4
_L1:    movqd   $(1),r0    ; 4 cycles + 4 cycles due to BGE
        movd    tos,r4     ; 2 cycles
        movd    tos,r3     ; 2 cycles
        ret     $(0)       ; 4 cycles, Break 4
```

Appendix D. Instruction Execution Times (Continued)

D.5.5 Execution Timing Tables

The following tables provide the execution timing information for all the NS32532 instructions. The table for the floating-point instructions provides only the CPU portion of the total execution time. The FPU execution times can be found in the NS32381 and NS32580 datasheets.

D.5.5.1 Basic and Memory Management Instructions

Mnemonic	T_{eu}	T_{au}	Notes
ABSi	5	2 + T_{ad}	
ACBi	5	2 + T_{ad}	If incorrect prediction then Break 1
ADDi	2	2 + T_{ad}	
ADDCi	2	2 + T_{ad}	
ADDPi	9	2 + T_{ad}	
ADDQi	2	2 + T_{ad}	
ADDR	2	4 + T_{ad}	
ADJSPi	5 3	2 + T_{ad} 2 + T_{ad}	i = B, W Break 0 i = D Break 0
ANDi	2	2 + T_{ad}	
ASHi	9	2 + T_{ad}	
B_{COND}	2 → 3 2 2 2	2 2 2 2	BTPC BTPI Break 2 BNTPC BNTPI Break 2 (see Note 5 in Section D.5.2)
BICi	2	2 + T_{ad}	
BICPSRi	6	2 + T_{ad}	Wait for pending writes. Break 5
BISPSRi	6	2 + T_{ad}	Wait for pending writes. Break 5
BPT	30 21	2 2	Modular Direct Break 5
BR	2 → 3	2	
BSR	2 → 3	3 + T_{ad}	
CASEi	7	2 + T_{ad}	Break 5
CBITi	10 14	2 2 + T_{ad}	<R> <M> Break 0
CBITIi	18	2 + T_{ad}	<M> Wait for pending writes. Execute interlocked RMW access. Break 5
CHECKi	10	2 + T_{ad}	Break −3. If SRC is out of bounds and the V bit in the PSR is set, then add trap time.
CINV	10	2 + T_{ad}	Wait for pending writes. Break 5
CMPi	2	2 + T_{ad}	
CMPMi	6 + 8 * n		n = number of elements. Break 0
CMPQi	2	2 + T_{ad}	
CMPSi	7 + 13 * n	2 + T_{ad}	n = number of elements. Break 0
CMPST	6 + 20 * n	2 + T_{ad}	n = number of elements. Break 0
COMi	2	2 + T_{ad}	
CVTP	5	4 + T_{ad}	
CXP	17	13	Break 5
CXPD	21	11 + T_{ad}	Break 5
DEIi	28 + 4 * i	5 + T_{ad}	i = 0/4/12 for B/W/D. Break 0
DIA	3	2	Break 5
DIVi	(30 → 40) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
ENTER	15 + 2 * n	3	n = number of registers saved. Break 0
EXIT	8 + 2 * n	2	n = number of registers restored
EXTi	12 13	8 8 + T_{ad}	<R> <M> Break −3
EXSi	11 14	6 6 + T_{ad}	<R> <M> Break −3

Appendix D. Instruction Execution Times (Continued)

D.5.5.1 Basic and Memory Management Instructions (Continued)

Mnemonic	T_{eu}	T_{au}	Notes
FFSi	11 + 3 * i	2 + T_{ad}	i = number of bytes
FLAG	4 32 21	2 2 2	No trap Trap, Modular Trap, Direct If trap then: {wait for pending writes; Break 5}
IBITi	10 14	2 2 + T_{ad}	<R> <M> Break 0
INDEXi	43	5 + T_{ad}	
INSi	15 18	8 8 + T_{ad}	<R> <M>
INSSi	14 19	6 6 + T_{ad}	<R> <M> Break 0
JSR	3	9 + T_{ad}	Break 5
JUMP	3	4 + T_{ad}	Break 5
LMR	11	2 + T_{ad}	Wait for pending writes. Break 5
LPRi	6 5 7	2 + T_{ad} 2 + T_{ad} 2 + T_{ad}	CPU Reg = FP, SP, USP, SP, MOD. Break 0 CPU Reg = CFG, INTBASE, DSR, BPC, UPSR. Wait for pending writes. Break 5 CPU Reg = DCR, PSR CAR. Wait for pending writes. Break 5
LSHi	3	2 + T_{ad}	
MEIi	13 + 2 * i	5 + T_{ad}	i = 0/4/12 for B/W/D. Break 0
MODi	(34 → 49) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
MOVi	2	2 + T_{ad}	
MOVMi	5 + 4 * n	2 + T_{ad}	n = number of elements. Break 0
MOVQi	2	2 + T_{ad}	
MOVSi	 12 + 4 * n 14 + 8 * n	 2 + T_{ad} 2 + T_{ad}	n = number of elements. No options. B, W and/or U Options in effect. Break 0
MOVST	16 + 9 * n	2 + T_{ad}	n = number of elements. Break 0
MOVSVi	9	2 + T_{ad}	Wait for pending writes. Break 5
MOVUSi	11	2 + T_{ad}	Wait for pending writes. Break 5
MOVXii	2	2 + T_{ad}	
MOVZii	2	2 + T_{ad}	
MULi	13 + 2 * i 24	2 + T_{ad} 2 + T_{ad}	i = 0/4/12 for B/W/D. General case. If MULD and 0 ≤ SRC ≤ 255
NEGi	2	2 + T_{ad}	
NOP	2	2	
NOTi	3	2 + T_{ad}	
ORi	2	2 + T_{ad}	
QUOi	(30 → 40) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
RDVAL	10	2 + T_{ad}	Wait for pending writes. Break 5
REMi	(32 → 42) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
RESTORE	7 + 2 * n	2	n = number of registers restored. Break 0
RET	4	3	Break 4
RETI	19 13 29 22	5 5 5 5	Noncascaded, Modular Noncascaded, Direct Cascaded, Modular Cascaded, Direct Wait for pending writes. Break 5
RETT	14 8	5 5	Modular Direct Wait for pending writes. Break 5
ROTi	7	2 + T_{ad}	
RXP	8	5	Break 5
SCONDi	3	2 + T_{ad}	
SAVE	8 + 2 * n	2	n = number of registers. Break 0
SBITi	10 14	2 2 + T_{ad}	<R> <M> Break 0

Appendix D. Instruction Execution Times (Continued)

D.5.5.1 Basic and Memory Management Instructions (Continued)

Mnemonic	T_{eu}	T_{au}	Notes
SBITIi	10 18	2 2 + T_{ad}	<R> <M> Wait for pending writes. Execute interlocked RMW access. Break 5
SETCFG	6	2	Break 5
SKPSi	8 + 6 * n	2 + T_{ad}	n = number of elements. Break 0
SKPST	6 + 20 * n	2 + T_{ad}	n = number of elements. Break 0
SMR	7	2 + T_{ad}	Wait for pending writes. Break 5
SPRi	5 3	2 + T_{ad} 2 + T_{ad}	CPU Reg = PSR, CAR CPU Reg = all others
SUBi	2	2 + T_{ad}	
SUBCi	2	2 + T_{ad}	
SUBPi	6	2 + T_{ad}	
SVC	32 21	2 2	Modular Direct Wait for pending writes. Break 5
TBITi	8 11	2 2 + T_{ad}	<R> <M> Break 0
WAIT	3	2	Wait for pending writes. Wait for interrupt
WRVAL	10	2 + T_{ad}	Wait for pending writes. Break 5
XORi	2	2 + T_{ad}	

Appendix D. Instruction Execution Times (Continued)

D.5.5.2 Floating-Point Instructions, CPU Portion

Mnemonic	T_{eu}	T_{au}	T_{tcs}	T_{tsc}	Group	Notes
MOVf, NEGf, ABSf, SQRTf, LOGBf	2 4 + 3 * I 6 + 3 * I 6 + 3 * I 11 + 4 * I 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} 2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2 2 2	1 1 1 1 3 + 2 * I 3 + 2 * I	A A B B B B	<FF> <MF> <IF> <TF> <FM> Break − (1 + I) <MM>, <IM> Break − (1 + I)
ADDf, SUBf, MULf, DIVf, SCALBf	2 4 + 3 * I 6 + 3 * I 6 + 3 * I 17 + 7 * I 19 + 10 * I	2 + T_{anp} 2 + T_{anp} 2 + T_{anp} 2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2 2 2	1 1 1 1 3 + 2 * I 3 + 2 * I	A A B B B B	<FF> <MF> <IF> <TF> <FM> Break − (1 + I) <MM>, <IM> Break − (1 + I)
ROUNDfi, TRUNCfi, FLOORfi	11 11 + 4 * I 13 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	3 + 2 * I 3 + 2 * I 3 + 2 * I 3 + 2 * I	B B B B	<FR> Break − 1 <FM> Break − (1 + I) <MR>, <IR> Break − 1 <MM>, <IM> Break − (1 + I)
CMPf	18 20 + 3 * I 23 + 3 * I 25 + 6 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2		B B B B	<FF> <MF> <FM> <MM>, <IM>, <MI>, <II> Break 3
POLYf, DOTf, MACf	2 4 + 3 * I 6 + 3 * I 11 + 4 * I 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2 2	1 1 1 1 1	A A B A B	<FF> <MF> <IF>, <TF> <FM> Break − (1 + I) <MM>, <MI>, <IM>, <II> Break − (1 + I)
MOVif	6 13 6 + 3 * I 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	1 1 	B B B B	<RF> <RM> Break − 1 <MF>, <IF>, <TF> <MM>, <IM> Break − (1 + I)
LFSR	6 6 + 3 * I 6 + 3 * I 6 + 3 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} 2 + T_{anp}	2 2 2 2	1 1 1 1	B B B B	<R> <M> <I> <T>
SFSR	11	2 + T_{anp} + T_{ad}	2	3	B	Break − 1
MOVFL	4 6 15 17	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	1 1 	B B B B	<FF> <MF>, <IF>, <TF> <FM> Break 0 <MM>, <IM> Break 0
MOVLF	4 9 15 20	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	1 1 	B B B B	<FF> <MF>, <IF>, <TF> <FM> Break 0 <MM>, <IM> Break 0

National Semiconductor

PRELIMINARY

NS32332-10/NS32332-15
32-Bit Advanced Microprocessors

General Description

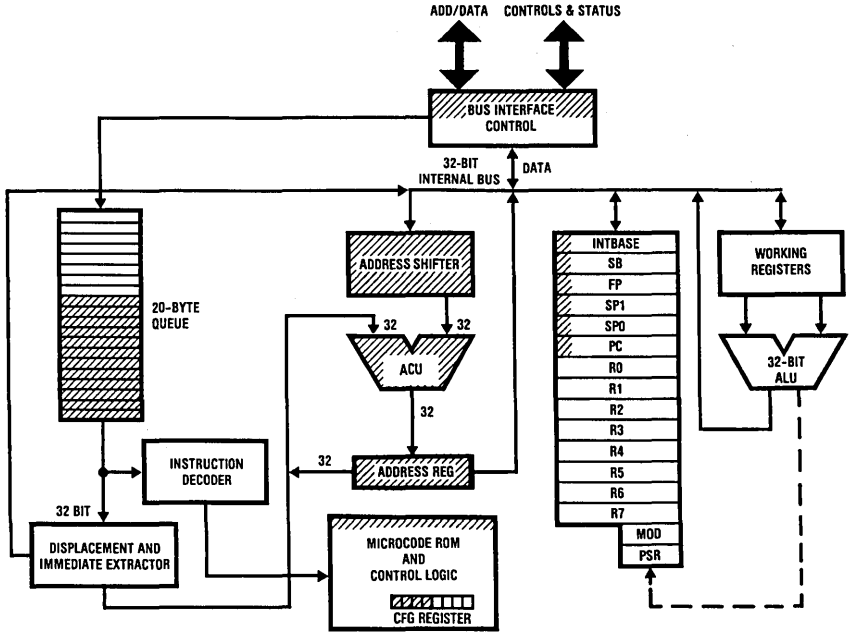
The NS32332 is a 32-bit, virtual memory microprocessor with 4 GByte addressing and an enhanced internal implementation. It is fully object code compatible with other Series 32000® microprocessors, and it has the added features of 32-bit addressing, higher instruction execution throughput, cache support, and expanded bus handling capabilities. The new bus features include bus error and retry support, dynamic bus sizing, burst mode memory accessing, and enhanced slave processor communication protocol. The higher clock frequency and added features of the NS32332 enable it to deliver 2 to 3 times the performance of the NS32032.

The NS32332 microprocessor is designed to work with both the 16- and 32-bit slave processors of the Series 32000 family.

Features

- 32-bit architecture and implementation
- 4 Gbyte uniform addressing space
- Software compatible with the Series 32000 Family
- Powerful instruction set
  - — General 2-address capability
  - — Very high degree of symmetry
  - — Address modes optimized for high level languages
- Supports both 16- and 32-bit Slave Processor Protocol
  - — Memory management support via NS32082 or NS32382
  - — Floating point support via NS32081 or NS32381
- Extensive bus feature
  - — Burst mode memory accessing
  - — Cache memory support
  - — Dynamic bus configuration (8-, 16-, 32-bits)
  - — Fast bus protocol
- High speed XMOS™ technology
- 84 Pin grid array package

Block Diagram



TL/EE/8673-1

FIGURE 1

*Shaded areas indicate enhancements from the NS32032.

Table of Contents

List of Illustrations

List of Illustrations (Continued)

List of Tables

1.0 Product Introduction

The Series 32000 Microprocessor family is a new generation of devices using National's XMOS and CMOS technologies. By combining state-of-the-art MOS technology with a very advanced architectural design philosophy, this family brings mainframe computer processing power to VLSI processors.

The Series 32000 family supports a variety of system configurations, extending from a minimum low-cost system to a powerful 4 gigabyte system. The architecture provides complete upward compatibility from one family member to another. The family consists of a selection of CPUs supported by a set of peripherals and slave processors that provide sophisticated interrupt and memory management facilities as well as high-speed floating-point operations. The architectural features of the Series 32000 family are described briefly below:

Powerful Addressing Modes. Nine addressing modes available to all instructions are included to access data structures efficiently.

Data Types. The architecture provides for numerous data types, such as byte, word, doubleword, and BCD, which may be arranged into a wide variety of data structures.

Symmetric Instruction Set. While avoiding special case instructions that compilers can't use, the Series 32000 family incorporates powerful instructions for control operations, such as array indexing and external procedure calls, which save considerable space and time for compiled code.

Memory-to-Memory Operations. The Series 32000 CPUs represent two-address machines. This means that each operand can be referenced by any one of the addressing modes provided. This powerful memory-to-memory architecture permits memory locations to be treated as registers for all useful operations. This is important for temporary operands as well as for context switching.

Memory Management. Either the NS32382 or the NS32082 Memory Management Unit may be added to the system to provide advanced operating system support functions, including dynamic address translation, virtual memory management, and memory protection.

Large, Uniform Addressing. The NS32332 has 32-bit address pointers that can address up to 4 gigabytes without requiring any segmentation; this addressing scheme provides flexible memory management without added-on expense.

Modular Software Support. Any software package for the Series 32000 family can be developed independent of all other packages, without regard to individual addressing. In addition, ROM code is totally relocatable and easy to access, which allows a significant reduction in hardware and software cost.

Software Processor Concept. The Series 32000 architecture allows future expansions of the instruction set that can be executed by special slave processors, acting as extensions to the CPU. This concept of slave processors is unique to the Series 32000 family. It allows software compatibility even for future components because the slave hardware is transparent to the software. With future advances in semiconductor technology, the slaves can be physically integrated on the CPU chip itself.

To summarize, the architectural features cited above provide three primary performance advantages and characteristics:

- High-Level Language Support
- Easy Future Growth Path
- Application Flexibility

1.1 NS32332 KEY FEATURES

The NS32332 is a 32-bit CPU in the Series 32000 family. It is totally software compatible with the NS32032, NS32016, and NS32008 CPUs but with an enhanced internal implementation.

The NS32332 design goals were to achieve two to three times the throughput of the NS32032 and to provide the full 32-bit addressing inherent in the architecture.

The basic approaches to higher throughput were: fewer clock cycles per instruction, better bus use, and higher clock frequency.

An examination of the block diagram of the NS32332 shows it to be identical to that of the NS32032, except for enhanced bus interface control, a 20-byte (rather than 8-byte) instruction prefetch queue, and special hardware in the address unit. The new addressing hardware consists of a high-speed ALU, a barrel shifter on one of its inputs, and an address register. Of the throughput improvement not due to increased clock frequency, about 15% is derived from the new address unit hardware, 15% from the bus enhancements, 10% from the larger prefetch queue, and 60% from microcode improvements.

Other important aspects of the enhanced bus interface circuitry of the NS32332 are a burst access mode, designed to work with nibble and static column RAMs, read and write timing designed to support caches, and support for bus error processing.

An enhanced slave processor communication protocol is designed to achieve improved performance with the NS32382 MMU and NS32381 FPU, while still working directly with the previous NS32082 MMU and NS32081 FPU.

2.0 Architectural Description

2.1 PROGRAMMING MODEL

The Series 32000 architecture has 8 general purpose and 8 dedicated registers. All registers are 32 bits wide except the STATUS and MODULE register. These two registers are each 16 bits wide.

2.1.1 General Purpose Registers

There are eight registers for meeting high speed general storage requirements, such as holding temporary variables and addresses. The general purpose registers are free for any use by the programmer. They are thirty-two bits in length. If a general register is specified for an operand that is eight or sixteen bits long, only the low part of the register is used; the high part is not referenced or modified.

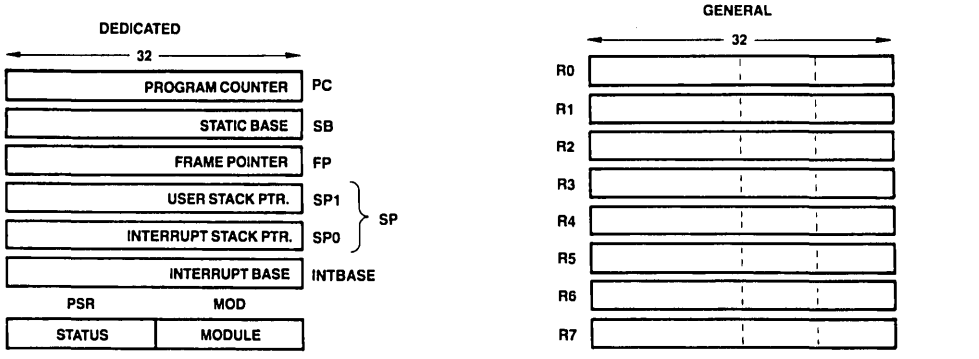
2.1.2 Dedicated Registers

The eight dedicated registers of the processor are assigned specific functions.

PC: The PROGRAM COUNTER register is a pointer to the first byte of the instruction currently being executed. The PC is used to reference memory in the program section.

SP0, SP1: The SP0 register points to the lowest address of the last item stored on the INTERRUPT STACK. This stack is normally used only by the operating system. It is used

2.0 Architectural Description (Continued)



TL/EE/8673-2

FIGURE 2-1. The General and Dedicated Registers

primarily for storing temporary data, and holding return information for operating system subroutines and interrupt and trap service routines. The SP1 register points to the lowest address of the last item stored on the USER STACK. This stack is used by normal user programs to hold temporary data and subroutine return information.

In this document, reference is made to the SP register. The terms "SP register" or "SP" refer to either SP0 or SP1, depending on the setting of the S bit in the PSR register. If the S bit in the PSR is 0 the SP refers to SP0. If the S bit in the PSR is 1 then SP refers to SP1.

Stacks in the Series 32000 family grow downward in memory. A Push operation pre-decrements the Stack Pointer by the operand length. A Pop operation post-increments the Stack Pointer by the operand length.

FP: The FRAME POINTER register is used by a procedure to access parameters and local variables on the stack. The FP register is set up on procedure entry with the ENTER instruction and restored on procedure termination with the EXIT instruction.

The frame pointer holds the address in memory occupied by the old contents of the frame pointer.

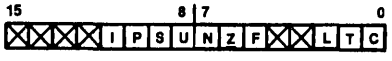
SB: The STATIC BASE register points to the global variables of a software module. This register is used to support relocatable global variables for software modules. The SB register holds the lowest address in memory occupied by the global variables of a module.

INTBASE: The INTERRUPT BASE register holds the address of the dispatch table for interrupts and traps (Sec. 3.8). The INTBASE register holds the lowest address in memory occupied by the dispatch table.

MOD: The MODULE register holds the address of the module descriptor of the currently executing software module. The MOD register is sixteen bits long, therefore the module table must be contained within the first 64K bytes of memory.

PSR: The PROCESSOR STATUS REGISTER (PSR) holds the status codes for the microprocessor.

The PSR is sixteen bits long, divided into two eight-bit halves. The low order eight bits are accessible to all programs, but the high order eight bits are accessible only to programs executing in Supervisor Mode.



TL/EE/8673-3

FIGURE 2-2. Processor Status Register

C: The C bit indicates that a carry or borrow occurred after an addition or subtraction instruction. It can be used with the ADDC and SUBC instructions to perform multiple-precision integer arithmetic calculations. It may have a setting of 0 (no carry or borrow) or 1 (carry or borrow).

T: The T bit causes program tracing. If this bit is a 1, a TRC trap is executed after every instruction (Sec. 3.8.5).

L: The L bit is altered by comparison instructions. In a comparison instruction the L bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as unsigned integers. Otherwise, it is set to "0". In Floating Point comparisons, this bit is always cleared.

F: The F bit is a general condition flag, which is altered by many instructions (e.g., integer arithmetic instructions use it to indicate overflow).

Z: The Z bit is altered by comparison instructions. In a comparison instruction the Z bit is set to "1" if the second operand is equal to the first operand; otherwise it is set to "0".

N: The N bit is altered by comparison instructions. In a comparison instruction the N bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as signed integers. Otherwise, it is set to "0".

U: If the U bit is "1" no privileged instructions may be executed. If the U bit is "0" then all instructions may be executed. When U = 0 the processor is said to be in Supervisor Mode; when U = 1 the processor is said to be in User Mode. A User Mode program is restricted from executing certain instructions and accessing certain registers which could interfere with the operating system. For example, a User Mode program is prevented from changing the setting of the flag used to indicate its own privilege mode. A Supervisor Mode program is assumed to be a trusted part of the operating system, hence it has no such restrictions.

S: The S bit specifies whether the SP0 register or SP1 register is used as the stack pointer. The bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use the SP0 register) or 1 (use the SP1 register).

2.0 Architectural Description (Continued)

P: The P bit prevents a TRC trap from occurring more than once for an instruction (Sec. 3.8.5.). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I: If I = 1, then all interrupts will be accepted (Sec. 3.8.). If I = 0, only the NMI interrupt is accepted. Trap enables are not affected by this bit.

2.1.3 The Configuration Register (CFG)*

Within the Control section of the CPU is the CFG Register, which declares the presence and type of external devices. It is referenced by only one instruction, SETCFG, which is intended to be executed only as part of system initialization after reset. The format of the CFG Register is shown in *Figure 2-3.*

*The NS32332 CPU has four new bits in the CFG Register, namely P, FC, FM and FF.

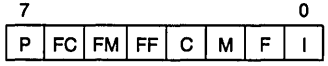


FIGURE 2-3. CFG Register

The CFG I bit declares the presence of external interrupt vectoring circuitry (specifically, the Interrupt Control Unit). If the CFG I bit is set, interrupts requested through the $\overline{INT}$ pin are "Vectored." If it is clear, these interrupts are "Non-Vectored." See Sec. 3.8.

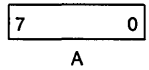
The F, M and C bits declare the presence of the FPU, MMU and Custom Slave Processors. If these bits are not set, the corresponding instructions are trapped as being undefined.

The FF, FM, FC bits define the Slave Communication Protocol to be used in FPU, MMU and Custom Slave instructions (Sec. 3.4.9). If these bits are not set, the corresponding instructions will use the 16-bit protocol (32032 compatible). If these bits are set, the corresponding instructions will use the new (fast) 32-bit protocol.

The P bit improves the efficiency of the Write Validation Buffer in the CPU. It is set if the Virtual Memory has page size(s) larger than or equal to 4 Kbytes. It is reset otherwise. In Systems where the MMU is not present, the P bit is not used.

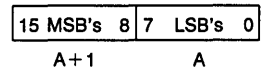
2.1.4 Memory Organization

The main memory is a uniform linear address space. Memory locations are numbered sequentially starting at zero and ending at $2^{32} - 1$. The number specifying a memory location is called an address. The contents of each memory location is a byte consisting of eight bits. Unless otherwise noted, diagrams in this document show data stored in memory with the lowest address on the right and the highest address on the left. Also, when data is shown vertically, the lowest address is at the top of a diagram and the highest address at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number zero, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.



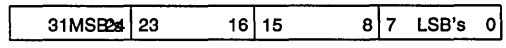
Byte at Address A

Two contiguous bytes are called a word. Except where noted (Sec. 2.2.1), the least significant byte of a word is stored at the lower address, and the most significant byte of the word is stored at the next higher address. In memory, the address of a word is the address of its least significant byte, and a word may start at any address.



Word at Address A

Two contiguous words are called a double word. Except where noted (Sec. 2.2.1), the least significant word of a double word is stored at the lowest address and the most significant word of the double word is stored at the address two greater. In memory, the address of a double word is the address of its least significant byte, and a double word may start at any address.



Double Word at Address A

Although memory is addressed as bytes, it is actually organized as double-words. Note that access time to a word or a double-word depends upon its address, e.g. double-words that are aligned to start at addresses that are multiples of four will be accessed more quickly than those not so aligned. This also applies to words that cross a double-word boundary.

2.1.5 Dedicated Tables

Two of the dedicated registers (MOD and INTBASE) serve as pointers to dedicated tables in memory.

The INTBASE register points to the Interrupt Dispatch and Cascade tables.

The MOD register contains a pointer into the Module Table, whose entries are called Module Descriptors. A Module Descriptor contains four pointers. The MOD register contains the address of the Module Descriptor for the currently running module. It is automatically up-dated by the Call External Procedure instructions (CXP and CXPD).

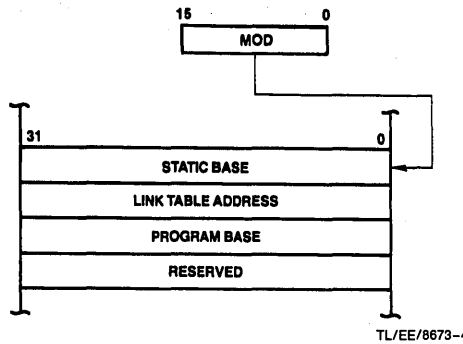


FIGURE 2-4. Module Descriptor Format

The format of a Module Descriptor is shown in *Figure 2-4.* The Static Base entry contains the address of static data assigned to the running module. It is loaded into the CPU Static Base register by the CXP and CXPD instructions. The Program Base entry contains the address of the first byte of instruction code in the module. Since a module may have multiple entry points, the Program Base pointer serves only as a reference to find them.

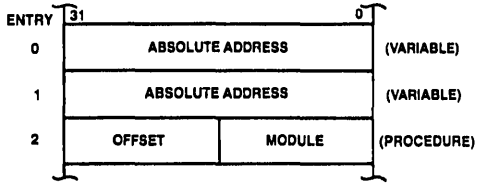
2.0 Architectural Description (Continued)

The Link Table Address points to the Link Table for the currently running module. The Link Table provides the information needed for:

1) Sharing variables between modules. Such variables are accessed through the Link Table via the External addressing mode.
2) Transferring control from one module to another. This is done via the Call External Procedure (CXP) instruction.

The format of a Link Table is given in *Figure 2-5*. A Link Table Entry for an external variable contains the 32-bit address of that variable. An entry for an external procedure contains two 16-bit fields: Module and Offset. The Module field contains the new MOD register contents for the module being entered. The Offset field is an unsigned number giving the position of the entry point relative to the new module's Program Base pointer.

For further details of the functions of these tables, see the Series 32000 Instruction Set Reference Manual.



TL/EE/8673-5

FIGURE 2-5. A Sample Link Table

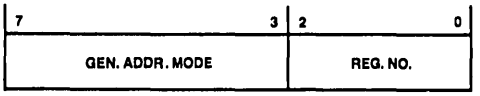
2.2 INSTRUCTION SET

2.2.1 General Instruction Format

Figure 2-6 shows the general format of a Series 32000 instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing Mode ("Gen") fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

Index Bytes appear when either or both Gen fields specify Scaled Index. In this case, the Gen field specifies only the Scale Factor (1, 2, 4 or 8), and the Index Byte specifies which General Purpose Register to use as the index, and which addressing mode calculation to perform before indexing. See *Figure 2-7*.

Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected address modes. Each Disp/Imm field may contain one or two displacements, or one immediate value. The size of a Displacement field is encoded with the top bits of that field, as shown in *Figure 2-8*, with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the Opcode field. Both Displacement and Immediate fields are stored most significant byte first. Note that this is different from the memory representation of data (Sec. 2.1.4).



TL/EE/8673-7

FIGURE 2-7. Index Byte Format

Some instructions require additional, "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition (Sec. 2.2.3).

2.2.2 Addressing Modes

The CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."

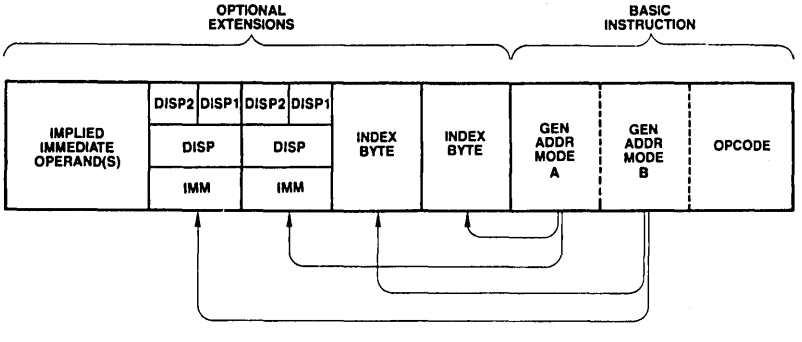
Addressing modes are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode, within the instruction that acts upon that variable. Extraneous data movement is therefore minimized.

Addressing Modes fall into nine basic types:

Register: The operand is available in one of the eight General Purpose Registers. In certain Slave Processor instructions, an auxiliary set of eight registers may be referenced instead.

Register Relative: A General Purpose Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.

Memory Space. Identical to Register Relative above, except that the register used is one of the dedicated registers PC, SP, SB or FP. These registers point to data areas generally needed by high-level languages.



TL/EE/8673-6

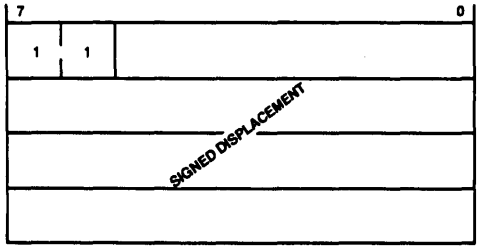
FIGURE 2-6. General Instruction Format

2.0 Architectural Description (Continued)

BYTE DISPLACEMENT: RANGE −64 TO +63

WORD DISPLACEMENT: RANGE −8192 TO +8191

DOUBLE WORD DISPLACEMENT: RANGE $-(2^{29}-2^{24})$ to $+(2^{29}-1)$*



TL/EE/8673-8

FIGURE 2-8. Displacement Encodings

***Note: The pattern "11100000" for the most significant byte of the displacement is reserved by National for future enhancements. Therefore, it should never be used by the user program. This causes the lower limit of the displacement range to be $-(2^{29}-2^{24})$ instead of -2^{29}.**

Memory Relative: A pointer variable is found within the memory space pointed to by the SP, SB or FP register. A displacement is added to that pointer to generate the Effective Address of the operand.

Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.

Absolute: The address of the operand is specified by a displacement field in the instruction.

External: A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.

Top of Stack: The currently-selected Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.

Scaled Index: Although encoded as an addressing mode. Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any General Purpose Register by 1, 2, 4 or 8 and adding it into the total, yielding the final Effective Address of the operand.

Table 2-1 is a brief summary of the addressing modes. For a complete description of their actions, see the Instruction Set Reference Manual.

2.2.3 Instruction Set Summary

Table 2-2 presents a brief description of the Series 32000 instruction set. The Format column refers to the Instruction Format tables (Appendix A). The Instruction column gives the instruction as coded in assembly language, and the Description column provides a short description of the function provided by that instruction. Further details of the exact operations performed by each instruction may be found in the Instruction Set Reference Manual.

Notations:

i = Integer length suffix: B = Byte
W = Word
D = Double Word

f = Floating Point length suffix: F = Standard Floating
L = Long Floating

gen = General operand. Any addressing mode can be specified.

short = A 4-bit value encoded within the Basic Instruction (see Appendix A for encodings).

imm = Implied immediate operand. An 8-bit value appended after any addressing extensions.

disp = Displacement (addressing constant): 8, 16 or 32 bits. All three lengths legal.

reg = Any General Purpose Register: R0–R7.

areg = Any Dedicated/Address Register: SP, SB, FP, MOD, INTBASE, PSR, US (bottom 8 PSR bits).

mreg = Any Memory Management Status/Control Register.

creg = A Custom Slave Processor Register (Implementation Dependent).

cond = Any condition code, encoded as a 4-bit field within the Basic Instruction (see Appendix A for encodings).

2.0 Architectural Description (Continued)

TABLE 2-1
NS32332 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0 or F0	None: Operand is in the specified register
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R7 or F7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Relative			
10000	Frame memory relative	disp2(disp1(FP))	Disp2 + Pointer; Pointer found at address Disp1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1(SP))	
10010	Static memory relative	disp2(disp1(SB))	
Reserved			
10011	(Reserved for Future Use)		
Immediate			
10100	Immediate	value	None: Operand is input from instruction queue.
Absolute			
10101	Absolute	@disp	Disp.
External			
10110	External	EXT (disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top of Stack			
10111	Top of stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register; "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	*+disp	
Scaled Index			
11100	Index, bytes	mode[Rn:B]	EA (mode) + Rn.
11101	Index, words	mode[Rn:W]	EA (mode) + 2× Rn.
11110	Index, double words	mode[Rn:D]	EA (mode) + 4× Rn.
11111	Index, quad words	mode[Rn:Q]	EA (mode) + 8 × Rn. 'Mode' and 'n' are contained within the Index Byte. EA (mode) denotes the effective address generated using mode.

2.0 Architectural Description (Continued)

TABLE 2-2

Series 32000 Instruction Set Summary

MOVES

Format	Operation	Operands	Description
4	MOVi	gen,gen	Move a value.
2	MOVQi	short,gen	Extend and move a signed 4-bit constant.
7	MOVMi	gen,gen,disp	Move Multiple: disp bytes (1 to 16).
7	MOVZBW	gen,gen	Move with zero extension.
7	MOVZiD	gen,gen	Move with zero extension.
7	MOVXBW	gen,gen	Move with sign extension.
7	MOVXiD	gen,gen	Move with sign extension.
4	ADDR	gen,gen	Move Effective Address.

INTEGER ARITHMETIC

Format	Operation	Operands	Description
4	ADDI	gen,gen	Add.
2	ADDQi	short,gen	Add signed 4-bit constant.
4	ADDCi	gen,gen	Add with carry.
4	SUBi	gen,gen	Subtract.
4	SUBCi	gen,gen	Subtract with carry (borrow).
6	NEGi	gen,gen	Negate (2's complement).
6	ABSi	gen,gen	Take absolute value.
7	MULi	gen,gen	Multiply
7	QUOi	gen,gen	Divide, rounding toward zero.
7	REMi	gen,gen	Remainder from QUO.
7	DIVi	gen,gen	Divide, rounding down.
7	MODi	gen,gen	Remainder from DIV (Modulus).
7	MEIi	gen,gen	Multiply to Extended Integer.
7	DEIi	gen,gen	Divide Extended Integer.

PACKED DECIMAL (BCD) ARITHMETIC

Format	Operation	Operands	Description
6	ADDPi	gen,gen	Add Packed.
6	SUBPi	gen,gen	Subtract Packed.

INTEGER COMPARISON

Format	Operation	Operands	Description
4	CMPi	gen,gen	Compare.
2	CMPQi	short,gen	Compare to signed 4-bit constant.
7	CMPMi	gen,gen,disp	Compare Multiple: disp bytes (1 to 16).

LOGICAL AND BOOLEAN

Format	Operation	Operands	Description
4	ANDi	gen,gen	Logical AND.
4	ORi	gen,gen	Logical OR.
4	BICi	gen,gen	Clear selected bits.
4	XORi	gen,gen	Logical Exclusive OR.
6	COMi	gen,gen	Complement all bits.
6	NOTi	gen,gen	Boolean complement: LSB only.
2	Scondi	gen	Save condition code (cond) as a Boolean variable of size i.

2.0 Architectural Description (Continued)

SHIFTS

Format	Operation	Operands	Description
6	LSHi	gen,gen	Logical Shift, left or right.
6	ASHi	gen,gen	Arithmetic Shift, left or right.
6	ROTi	gen,gen	Rotate, left or right.

BITS

Format	Operation	Operands	Description
4	TBITi	gen,gen	Test bit.
6	SBITi	gen,gen	Test and set bit.
6	SBITIi	gen,gen	Test and set bit, interlocked
6	CBITi	gen,gen	Test and clear bit.
6	CBITIi	gen,gen	Test and clear bit, interlocked.
6	IBITi	gen,gen	Test and invert bit.
8	FFSi	gen,gen	Find first set bit

BIT FIELDS

Bit fields are values in memory that are not aligned to byte boundaries. Examples are PACKED arrays and records used in Pascal. "Extract" instructions read and align a bit field. "Insert" instructions write a bit field from an aligned source.

Format	Operation	Operands	Description
8	EXTi	reg,gen,gen,disp	Extract bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented).
7	EXTSi	gen,gen,imm,imm	Extract bit field (short form).
7	INSSi	gen,gen,imm,imm	Insert bit field (short form).
8	CVTP	reg,gen,gen	Convert to Bit Field Pointer.

ARRAYS

Format	Operation	Operands	Description
8	CHECKi	reg,gen,gen	Index bounds check.
8	INDEXi	reg,gen,gen	Recursive indexing step for multiple-dimensional arrays.

STRINGS

String instructions assign specific functions to the General Purpose Registers:

R4 - Comparison Value

R3 - Translation Table Pointer

R2 - String 2 Pointer

R1 - String 1 Pointer

R0 - Limit Count

Options on all string instructions are:

B (Backward): Decrement string pointers after each step rather than incrementing.

U (Until match): End instruction if String 1 entry matches R4.

W (While match): End instruction if String 1 entry does not match R4.

All string instructions end when R0 decrements to zero.

Format	Operation	Operands	Descriptions
5	MOVSi	options	Move String 1 to String 2.
	MOVST	options	Move string, translating bytes.
5	CMPSi	options	Compare String 1 to String 2.
	CMPST	options	Compare translating, String 1 bytes.
5	SKPSi	options	Skip over String 1 entries
	SKPST	options	Skip, translating bytes for Until/While.

2.0 Architectural Description (Continued)

JUMPS AND LINKAGE

Format	Operation	Operands	Description
3	JUMP	gen	Jump.
0	BR	disp	Branch (PC Relative).
0	Bcond	disp	Conditional branch.
3	CASEi	gen	Multiway branch.
2	ACBi	short,gen,disp	Add 4-bit constant and branch if non-zero.
3	JSR	gen	Jump to subroutine.
1	BSR	disp	Branch to subroutine.
1	CXP	disp	Call external procedure.
3	CXPD	gen	Call external procedure using descriptor.
1	SVC		Supervisor Call.
1	FLAG		Flag Trap.
1	BPT		Breakpoint Trap.
1	ENTER	[reg list],disp	Save registers and allocate stack frame (Enter Procedure).
1	EXIT	[reg list]	Restore registers and reclaim stack frame (Exit Procedure).
1	RET	disp	Return from subroutine.
1	RXP	disp	Return from external procedure call.
1	RETT	disp	Return from trap. (Privileged)
1	RETI		Return from interrupt. (Privileged)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save General Purpose Registers.
1	RESTORE	[reg list]	Restore General Purpose Registers.
2	LPRi	areg,gen	Load Dedicated Register. (Privileged if PSR or INTBASE)
2	SPRi	areg,gen	Store Dedicated Register. (Privileged if PSR or INTBASE)
3	ADJSPi	gen	Adjust Stack Pointer.
3	BISPSRi	gen	Set selected bits in PSR. (Privileged if not Byte length)
3	BICPSRi	gen	Clear selected bits in PSR. (Privileged if not Byte length)
5	SETCFG	[option list]	Set Configuration Register. (Privileged)

FLOATING POINT

Format	Operation	Operands	Description
11	MOVf	gen,gen	Move a Floating Point value.
9	MOVLF	gen,gen	Move and shorten a Long value to Standard.
9	MOVFL	gen,gen	Move and lengthen a Standard value to Long.
9	MOVif	gen,gen	Convert any integer to Standard or Long Floating.
9	ROUNDfi	gen,gen	Convert to integer by rounding.
9	TRUNCfi	gen,gen	Convert to integer by truncating, toward zero.
9	FLOORfi	gen,gen	Convert to largest integer less than or equal to value.
11	ADDf	gen,gen	Add.
11	SUBf	gen,gen	Subtract.
11	MULf	gen,gen	Multiply.
11	DIVf	gen,gen	Divide.
11	CMPf	gen,gen	Compare.
11	NEGf	gen,gen	Negate.
11	ABSf	gen,gen	Take absolute value.
12	POLYf	gen,gen	Polynomial Step.
12	DOTf	gen,gen	Dot Product.
12	SCALBf	gen,gen	Binary Scale.
12	LOGBf	gen,gen	Binary Log.
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

2.0 Architectural Description (Continued)

MEMORY MANAGEMENT

Format	Operation	Operands	Description
14	LMR	mreg,gen	Load Memory Management Register. (Privileged)
14	SMR	mreg,gen	Store Memory Management Register. (Privileged)
14	RDVAL	gen	Validate address for reading. (Privileged)
14	WRVAL	gen	Validate address for writing. (Privileged)
8	MOVSUi	gen,gen	Move a value from Supervisor Space to User Space. (Privileged)
8	MOVUSi	gen,gen	Move a value from User Space to Supervisor Space. (Privileged)

MISCELLANEOUS

Format	Operation	Operands	Description
1	NOP		No Operation.
1	WAIT		Wait for interrupt.
1	DIA		Diagnose. Single-byte "Branch to Self" for hardware breakpointing. Not for use in programming.

CUSTOM SLAVE

Format	Operation	Operands	Description
15.5	CCAL0c	gen,gen	Custom Calculate.
15.5	CCAL1c	gen,gen	
15.5	CCAL2c	gen,gen	
15.5	CCAL3c	gen,gen	
15.5	CMOV0c	gen,gen	Custom Move.
15.5	CMOV1c	gen,gen	
15.5	CMOV2c	gen,gen	
15.5	CMOV3c	gen,gen	
15.5	CCMP0c	gen,gen	Custom Compare.
15.5	CCMP1c	gen,gen	
15.1	CCV0ci	gen,gen	Custom Convert.
15.1	CCV1ci	gen,gen	
15.1	CCV2ci	gen,gen	
15.1	CCV3ic	gen,gen	
15.1	CCV4DQ	gen,gen	
15.1	CCV5QD	gen,gen	
15.1	LCSR	gen	Load Custom Status Register.
15.1	SCSR	gen	Store Custom Status Register.
15.0	CATST0	gen	Custom Address/Test. (Privileged)
15.0	CATST1	gen	(Privileged)
15.0	LCR	creg,gen	Load Custom Register. (Privileged)
15.0	SCR	creg,gen	Store Custom Register. (Privileged)

3.0 Functional Description

The following is a functional description of the NS32332 CPU.

3.1 POWER AND GROUNDING

The NS32332 requires a single 5-volt power supply, applied on 7 pins. The Logic Voltage pins ($V_{CC}L1$ and $V_{CC}L2$) supply the power to the on-chip logic. The Buffer Voltage pins (V_{CCB1} to V_{CCB5}) supply the power to the output drivers of the chip. The Logic Voltage pins and the Buffer Voltage pins should be connected together by a power (V_{CC}) plane on the printed circuit board.

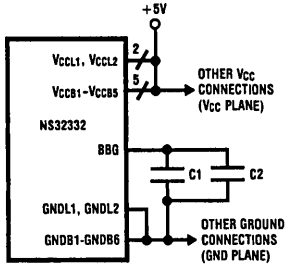
The NS32332 grounding connections are made on 8 pins. The Logic Ground pins (GNDL1 and GNDL2) are the ground pins for the on-chip logic. The Buffer Ground pins (GNDB1 to GNDB6) are the ground pins for the output drivers of the chip. The Logic Ground pins and the Buffer Ground pins should be connected together by a ground plane on the printed circuit board.

In addition to V_{CC} and Ground, the NS32332 CPU uses an internally-generated negative voltage. It is necessary to filter this voltage externally by attaching a pair of capacitors (*Figure 3.1*) from the BBG pin to Ground.

Recommended values for these are:

C1: 1 μF, Tantalum

C2: 1000 pF, Low inductance. This should be either a disc or monolithic capacitor.



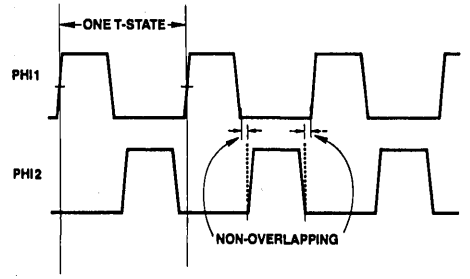
TL/EE/8673-11

FIGURE 3-1. Recommended Supply Connections

3.2 CLOCKING

The NS32332 inputs clocking signals from the Timing Control Unit (TCU), which presents two non-overlapping phases of a single clock frequency. These phases are called PHI1 (pin A7) and PHI2 (pin B8). Their relationship to each other is shown in *Figure 3-2*.

Each rising edge of PHI1 defines a transition in the timing state ("T-State") of the CPU. One T-State represents the execution of one microinstruction within the CPU, and/or one step of an external bus transfer. See Sec. 4 for complete specifications of PHI1 and PHI2.



TL/EE/8673-9

FIGURE 3-2. Clock Timing Relationships

As the TCU presents signals with very fast transitions, it is recommended that the conductors carrying PHI1 and PHI2 be kept as short as possible, and that they not be connected anywhere except from the TCU to the CPU and, if present, the MMU. A TTL Clock signal (CTTL) is provided by the TCU for all other clocking.

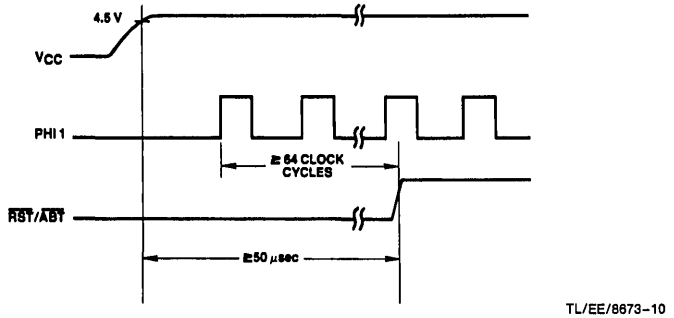
3.3 RESETTING

The $\overline{RST}/\overline{ABT}$ pin serves both as a Reset for on-chip logic and as the Abort input for Memory-Managed systems. For its use as the Abort Command, see Sec. 3.5.2.

The $\overline{DT}/\overline{SDONE}$ pin is sampled on the rising edge of PHI1, one cycle before the reset signal is deasserted to select the data timing during write cycles. If $\overline{DT}/\overline{SDONE}$ is sampled high, AD0–AD31 are floated during state T2 and the data is output during state T3. This mode must be selected if an MMU is used (Section 3.5). If $\overline{DT}/\overline{SDONE}$ is sampled low, the data is output during state T2. See *Figure 3-7*.

The CPU may be reset at any time by pulling the $\overline{RST}/\overline{ABT}$ pin low for at least 64 clock cycles. Upon detecting a reset, the CPU terminates instruction processing, resets its internal logic, and clears the Program Counter (PC) and Processor Status Register (PSR) to all zeroes.

On application of power, $\overline{RST}/\overline{ABT}$ must be held low for at least 50 μsec after V_{CC} is stable. This is to ensure that all



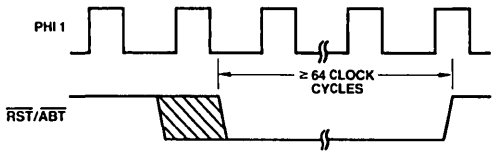
TL/EE/8673-10

FIGURE 3-3. Power-on Reset Requirements

3.0 Functional Description (Continued)

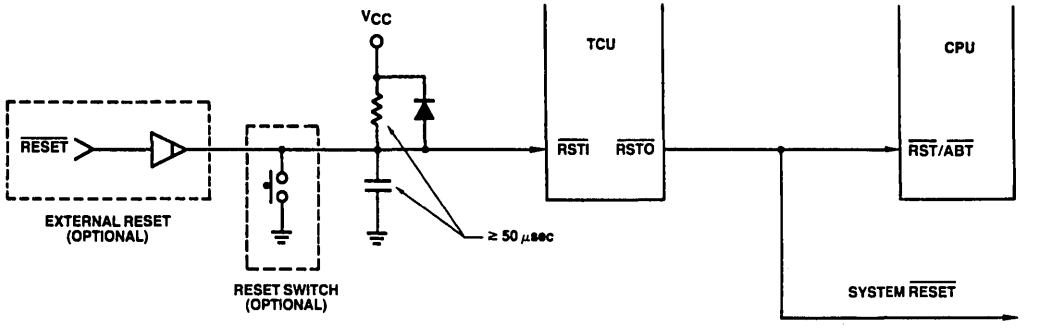
on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must also remain active for not less than 64 clock cycles. See *Figures 3-3* and *3-4*.

The Timing Control Unit (TCU) provides circuitry to meet the Reset requirements of the NS32332 CPU. *Figure 3-5a* shows the recommended connections for a non-Memory-Managed system. *Figure 3-5b* shows the connections for a Memory-Managed system.



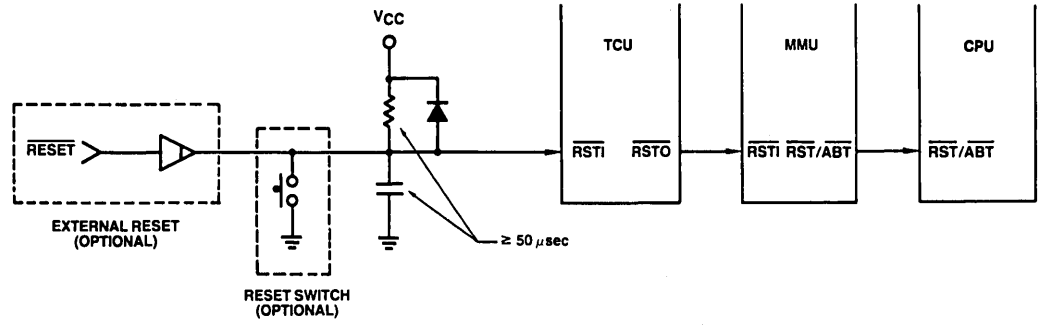
TL/EE/8673-12

FIGURE 3-4. General Reset Timing



TL/EE/8673-13

FIGURE 3-5a. Recommended Reset Connections, Non-Memory-Managed System



TL/EE/8673-14

FIGURE 3-5b. Recommended Reset Connections, Memory-Managed System

3.4 BUS CYCLES

The NS32332 CPU will perform Bus cycles for one of the following reasons:

1) To write or read data to or from memory or peripheral interface device. Peripheral input and output are memory mapped in the Series 32000 family.
2) To fetch instructions into the 20-byte instruction queue. This happens whenever the bus would otherwise be idle and the queue is not already full.
3) To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
4) To transfer information to or from a Slave Processor.

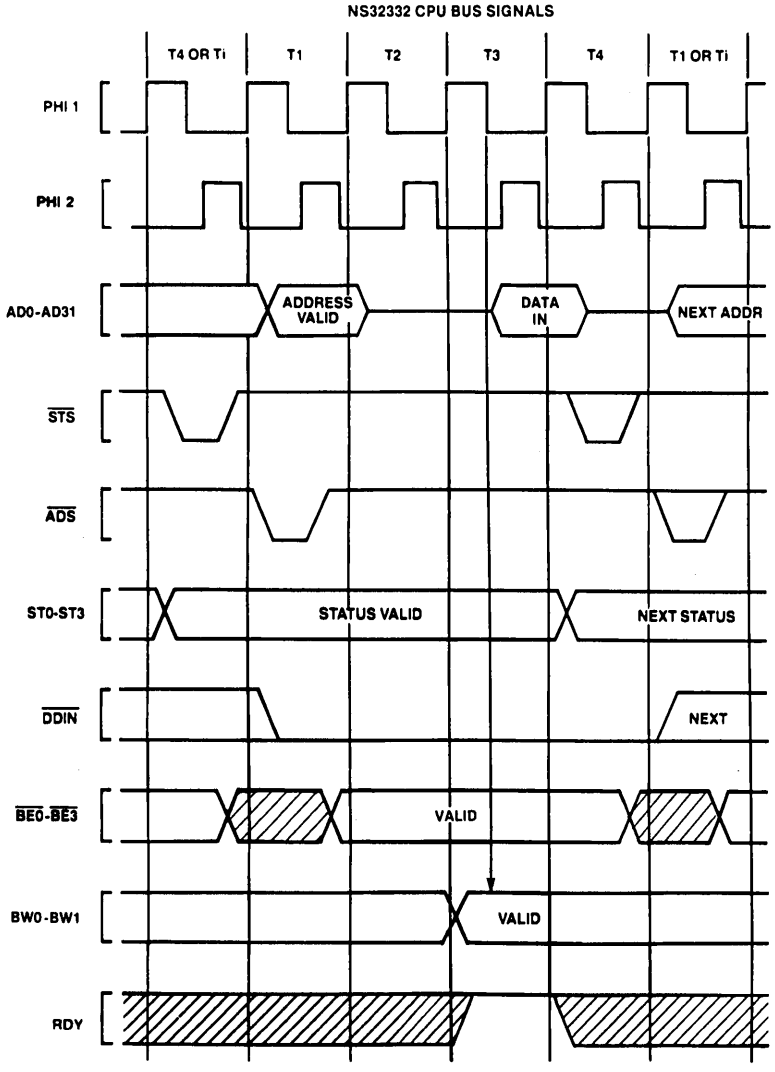
In terms of bus timing, cases 1 through 3 above are identical. For timing specifications, see Sec. 4. The only external difference between them is the 4-bit code placed on the Bus Status pins (ST0–ST3). Slave Processor cycles differ in that separate control signals are applied (Sec. 3.4.6).

For case 1 (only Read) and case 2, the NS32332 supports Burst cycles which are suitable for memories that can handle "nibble mode" accesses. (Sec. 3.4.2).

The sequence of events in a non-Slave, non-Burst Bus cycle is shown in *Figure 3-6* for a Read cycle, and *Figure 3-7* for a Write cycle. The cases shown assume that the selected memory or interface device is capable of communicating with the CPU at full speed. If it is not, then cycle extension may be requested through the RDY line (Sec. 3.4.1).

A full speed Bus cycle is performed in four cycles of the PHI1 clock, labeled T1 through T4. Clock cycles not associated with a Bus cycle are designated Ti (for idle).

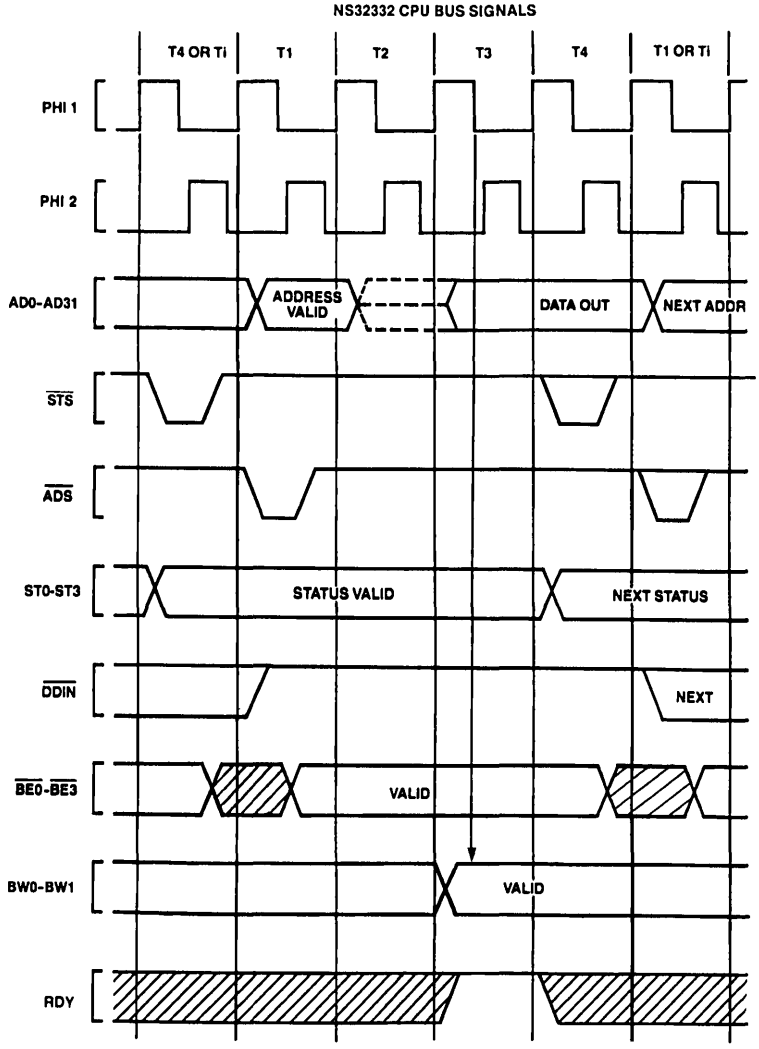
3.0 Functional Description (Continued)



TL/EE/8673–15

FIGURE 3-6. Read Cycle Timing

3.0 Functional Description (Continued)



TL/EE/8673–16

FIGURE 3-7. Write Cycle Timing

3.0 Functional Description (Continued)

During T4 or Ti which preceed T1 of the current Bus cycle, the CPU applies a Status Code on pins ST0-ST3. It also provides a low-going pulse on the $\overline{STS}$ pin to indicate that the status code is valid.

The $\overline{ADS}$ signal has the dual purpose of informing the external circuitry that a Bus cycle is starting and of providing control to an external latch for demultiplexing address bits 0–31 from AD0–AD31 pins. (See *Figure 3-8.*)

During this time, the control signal $\overline{DDIN}$, which indicates the direction of the transfer, and $\overline{BE0}$–$\overline{BE3}$ which indicate which of the four bus bytes to be referenced, become valid. Note that during Instruction Fetch cycles $\overline{BE0}$–$\overline{BE3}$ are all active, but in operand Read or Write cycles they indicate the byte(s) to be referenced.

Note: If a burst cycle occurs during an operand read, all the memory banks should be enabled, during the burst cycle, regardless of $\overline{BE}$n. The CPU $\overline{BE}$n lines, in this case, are valid in the middle of T3 of the burst cycle—thus, there may not be enough time to selectively enable the different memory banks, unless a WAIT state is added. See *Figure 4-6.*

During T2 the CPU floats AD0–AD31 lines unless $\overline{DT/SDONE}$ is sampled low on the rising edge of reset and the bus cycle is a write cycle. T2 is a time window to be used for virtual to physical address translation by the Memory Management Unit, if virtual memory is used in the system.

The T3 state provides for access time requirements and it occurs at least once in a bus cycle. In the middle of T3 on the falling edge of PHI1, the RDY line is sampled to determine whether the bus cycle will be extended (Sec. 3.4.1).

If the CPU is performing a Read cycle, the Data Bus (AD0–AD31) is sampled on the falling edge of PHI2 of the last T3 state. See Sec. 4. Data must, however, be held at least until the beginning of T4. The T4 state finishes the Bus cycle. Data from the CPU during Write cycles remains valid throughout T4. Note that the Bus Status lines (ST0–ST3) change at the beginning of T4, anticipating the following bus cycle (if any).

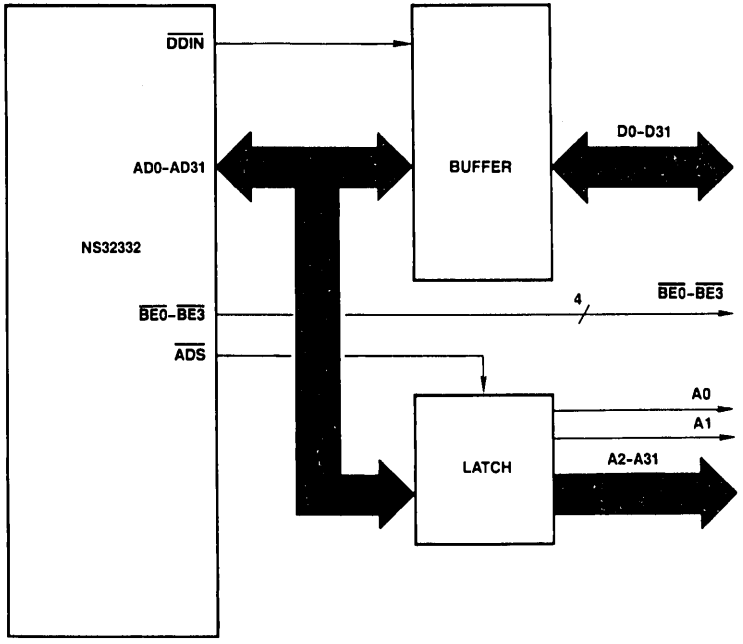
3.4.1 Cycle Extension

To allow sufficient strobe widths and access times for any speed of memory or peripheral device, the NS32332 provides for extension of a bus cycle. Any type of bus cycle except a Slave Processor cycle can be extended.

In *Figures 3-7* and *3-8*, note that during T3 all bus control signals from the CPU and TCU are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the RDY (Ready) pin.

In the middle of T3 on the falling edge of PHI1, the RDY line is sampled by the CPU. If RDY is high, the next T-state will be T4, ending the bus cycle. If RDY is low, then another T3 state will be inserted and the RDY line will again be sampled on the falling edge of PHI1. Each additional T3 state after the first is referred to as a "WAIT STATE". See *Figure 3-9.*

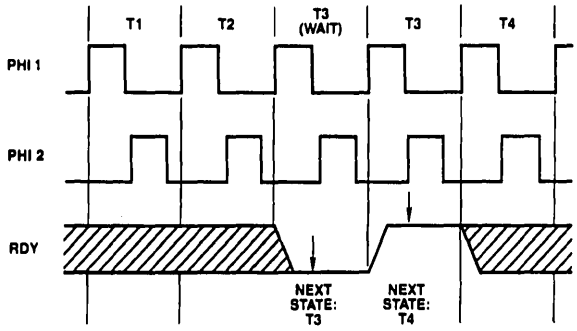
Figure 3-10 illustrates a typical Read cycle, with two WAIT states requested through the RDY pin.



TL/EE/8673-17

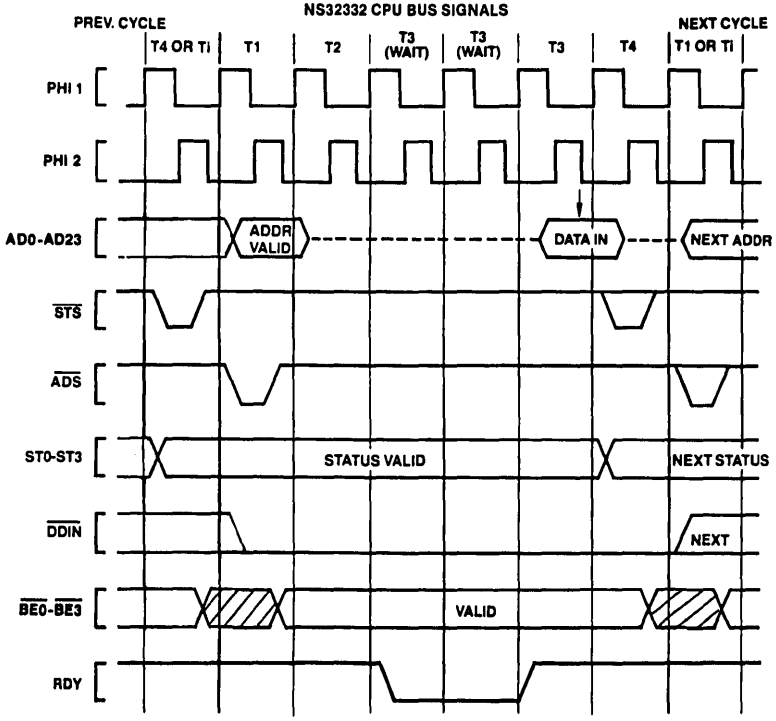
FIGURE 3-8. Bus Connections

3.0 Functional Description (Continued)



TL/EE/8673-18

FIGURE 3-9. RDY Pin Timing



TL/EE/8673-19

FIGURE 3-10. Extended Cycle Example

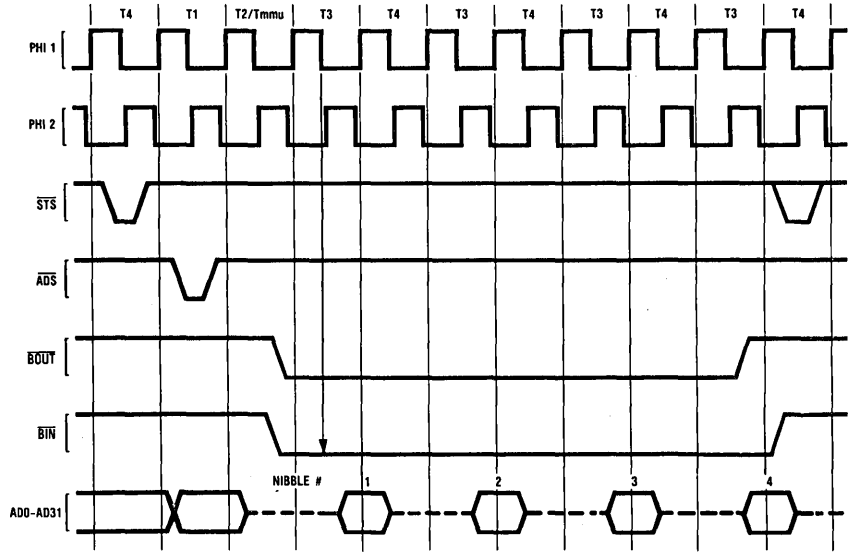
3.0 Functional Description (Continued)

3.4.2 Burst Cycles

The NS32332 is capable of performing Burst cycles in order to increase the bus throughput. Burst is available in instruction Fetch cycles and operand Read cycles only. Burst is not supported in operand Write cycles or Slave cycles.

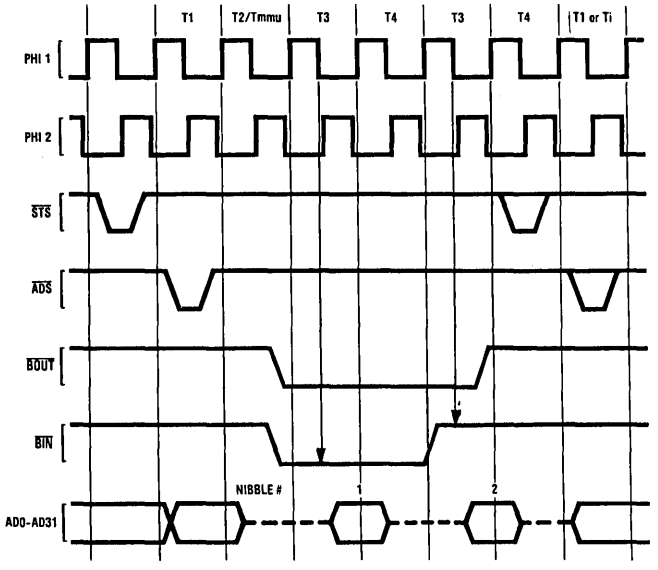
The sequence of events for Burst cycles is shown in *Figure 3-11*. The cases shown assume that the selected memory is capable of communicating with the CPU at full speed. If it is not, then cycle extension may be requested through the RDY line (Sec. 3.4.1).

A Burst cycle is composed of two parts. The first part is a regular cycle (i.e. T1 through T4), in which the CPU outputs the new status and asserts all the other relevant control signals discussed in Sec. 3.4. In addition, the Burst Out Signal ($\overline{BOUT}$) is activated by the CPU indicating that the CPU can perform Burst cycles. If the selected memory allows



TL/EE/8673-20

(a) Normal Termination of Burst



TL/EE/8673-21

(b) External Termination of Burst

FIGURE 3-11. Burst Cycles (For Read Only)

3.0 Functional Description (Continued)

Burst cycles, it will notify the CPU by activating the burst in signal ($\overline{BIN}$). $\overline{BIN}$ is sampled by the CPU in the middle of T3 on the falling edge of PHI1. If the memory does not allow burst ($\overline{BIN}$ high), the cycle will terminate through T4 and $\overline{BOUT}$ will go inactive immediately. If the memory allows burst ($\overline{BIN}$ low), and the CPU has not deasserted $\overline{BOUT}$, the second part of the Burst cycle will be performed (see *Figure 3-11*) and $\overline{BOUT}$ will remain active until termination of the Burst.

The second part consists of up to 3 nibbles. In each nibble, a data item is read by the CPU. The duration of each nibble is 2 clock cycles labeled T3 and T4.

The Burst chain will be terminated in the following cases:

1. The CPU has reached a 16 byte boundary i.e. the byte address of the current nibble is x...x1111 (binary).
2. The CPU detects that the instructions being prefetched (in Burst Mode) are no longer needed due to an alteration of the flow of control. This happens, for example, when a branch instruction is executed or an exception occurs.

Note: In 16-bit bus systems (see Sec. 3.4.7) the Burst chain will be terminated by the CPU on an 8-byte boundary i.e. address x..x111 (binary) and in 8-bit bus system on a 4-byte boundary i.e. address x...x11 (binary).

3. The data operand has been completely read. This applies to burst read cycles for non-aligned operands or when the bus width is either 8 or 16 bits.
4. $\overline{BIN}$, sampled in the current nibble's last T3, is not active any more. (See *Figure 3.11b*).
5. Bus Error or Bus Retry occurs (see Sec. 3.4.8).
6. A $\overline{HOLD}$ Request occurs.

Any nibble's T3 may be extended with WAIT states using the RDY line as described in Sec. 3.4.2.

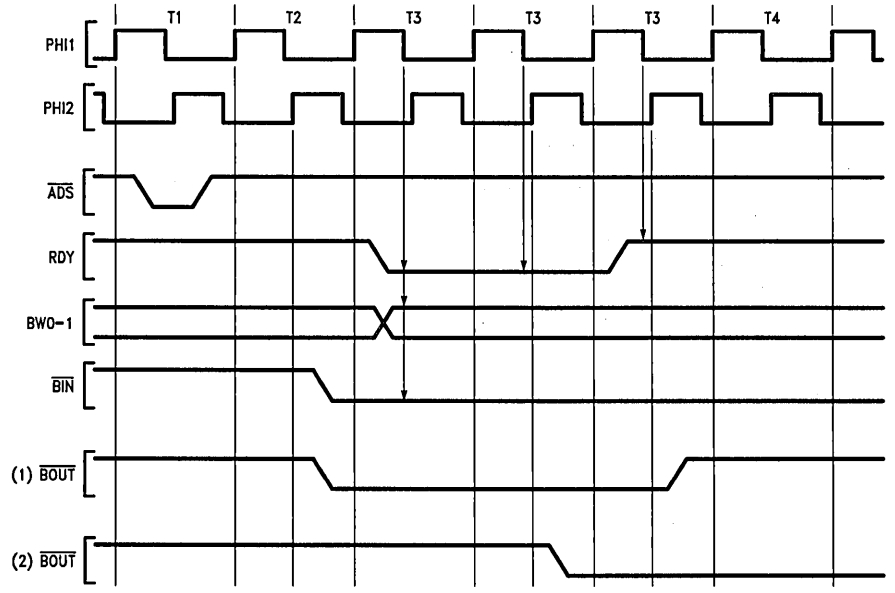
The control signals $\overline{BOUT}$, ST0–ST3, and $\overline{DDIN}$ remain stable during the Burst chain.

$\overline{BE0}$–$\overline{BE3}$ are adjusted for every nibble in operand cycles.

$\overline{BOUT}$ is initially set by the CPU according to the known bus width. Its state may change in a subsequent T3 as a result of a change in the bus width. *Figure 3-12* shows the resulting $\overline{BOUT}$ timing.

Note: If the selected memory is capable of handling burst transfers, it should activate $\overline{BIN}$ regardless of the state of $\overline{BOUT}$.

The reason is that $\overline{BOUT}$ may be activated by the CPU after the $\overline{BIN}$ sampling time. The $\overline{BOUT}$ signal indicates when the CPU is going to burst, and should not be interpreted as a 'Burst Request' signal.



TL/EE/8673–88

Note 1: CPU deasserts $\overline{BOUT}$.

Note 2: CPU asserts $\overline{BOUT}$.

FIGURE 3-12. $\overline{BOUT}$ Timing Resulting from a Bus Width Change

3.0 Functional Description (Continued)

3.4.3 Bus Status

The NS32332 CPU presents four bits of Bus Status information on pins ST0–ST3. The various combinations on these pins indicate why the CPU is performing a bus cycle, or, if it is idle on the bus, then why is it idle.

Referring to *Figures 3-6* and *3-7*, note that Bus Status leads the corresponding Bus Cycle, going valid one clock cycle before T1, and changing to the next state at T4. This allows the system designer to fully decode the Bus Status and, if desired, latch the decoded signals before $\overline{ADS}$ initiates the Bus Cycle.

The Bus Status pins are interpreted as a four-bit value, with ST0 the least significant bit. Their values decode as follows:

0000 – The bus is idle because the CPU does not yet need to perform a bus access.

0001 – The bus is idle because the CPU is executing the WAIT instruction.

0010 – (Reserved for future use.)

0011 – The bus is idle because the CPU is waiting for a Slave Processor to complete an instruction.

0100 – Interrupt Acknowledge, Master.

The CPU is performing a Read cycle. To acknowledge receipt of a Non-Maskable Interrupt (on $\overline{NMI}$) it will read from address $FFFFFF00_{16}$, but will ignore any data provided.

To acknowledge receipt of a Maskable Interrupt (on $\overline{INT}$) it will read from address $FFFFFE00_{16}$, expecting a vector number to be provided from the Master Interrupt Control Unit. If the vectoring mode selected by the last SETCFG instruction was Non-Vectored, then the CPU will ignore the value it has read and will use a default vector instead. See Sec. 3.4.5.

0101 – Interrupt Acknowledge, Cascaded.

The CPU is reading a vector number from a Cascaded Interrupt Control Unit. The address provided is the address of ICU's Hardware Vector register. See Sec. 3.4.6.

0110 – End of Interrupt, Master.

The CPU is performing a Read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction. See Sec. 3.4.6.

0111 – End of Interrupt, Cascaded.

The CPU is reading from a Cascaded Interrupt Control Unit to indicate that it is returning (through RETI) from an interrupt service routine requested by that unit. See Sec. 3.4.6.

1000 – Sequential Instruction Fetch.

The CPU is reading the next sequential word from the instruction stream into the Instruction Queue. It will do so whenever the bus would otherwise be idle and the queue is not already full.

1001 – Non-Sequential Instruction Fetch.

The CPU is performing the first fetch of instruction code after the Instruction Queue is purged. This will occur as a result of any jump or branch, or any interrupt or trap, or execution of certain instructions.

1010 – Data Transfer.

The CPU is reading or writing an operand of an instruction.

1011 – Read RMW Operand.

The CPU is reading an operand which will subsequently be modified and rewritten. If memory protection circuitry would not allow the following Write cycle, it must abort this cycle.

1100 – Read for Effective Address Calculation.

The CPU is reading information from memory in order to determine the Effective Address of an operand. This will occur whenever an instruction uses the Memory Relative or External addressing mode.

1101 – Transfer Slave Processor Operand.

The CPU is either transferring an instruction operand to or from a Slave Processor, or it is issuing the Operation Word of a Slave Processor instruction. See Sec. 3.9.1.

1110 – Read Slave Processor Status.

The CPU is reading a Status Word from a Slave Processor. This occurs after the Slave Processor has signalled completion of an instruction. The transferred word tells the CPU whether a trap should be taken, and in some instructions it presents new values for the CPU Processor Status Register bits N, Z, L or F. See Sec. 3.9.1.

1111 – Broadcast Slave ID.

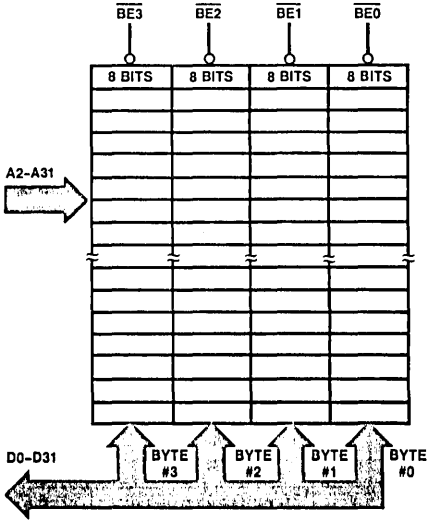
The CPU is initiating the execution of a Slave Processor instruction. The ID Byte (first byte of the instruction) is sent to all Slave Processors, one of which will recognize it. From this point the CPU is communicating with only one Slave Processor. See Sec. 3.9.1.

3.0 Functional Description (Continued)

3.4.4 Data Access Sequences

The 32-bit address provided by the NS32332 is a byte address; that is, it uniquely identifies one of up to 4 billion eight-bit memory locations. An important feature of the NS32332 is that the presence of a 32-bit data bus imposes no restrictions on data alignment; any data item, regardless of size, may be placed starting at any memory address. The NS32332 provides special control signals. Byte Enable ($\overline{BE0}$–$\overline{BE3}$) which facilitate individual byte accessing on a 32-bit bus.

Memory is organized as four eight-bit banks, each bank receiving the double-word address (A2–A31) in parallel. One bank, connected to Data Bus pins AD0–AD7 is enabled when $\overline{BE0}$ is low. The second bank, connected to data bus pins AD8–AD15 is enabled when $\overline{BE1}$ is low. The third and fourth banks are enabled by $\overline{BE2}$ and $\overline{BE3}$, respectively. See *Figure 3-13*.



TL/EE/8673–22

FIGURE 3-13. Memory Interface

Since operands do not need to be aligned with respect to the double-word bus access performed by the CPU, a given double-word access can contain one, two, three, or four bytes of the operand being addressed, and these bytes can begin at various positions, as determined by A1, A0. Table 3-1 lists the 10 resulting access types.

TABLE 3-1
Bus Access Types

Type	Bytes Accessed	A1,A0	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
1	1	00	1	1	1	0
2	1	01	1	1	0	1
3	1	10	1	0	1	1
4	1	11	0	1	1	1
5	2	00	1	1	0	0
6	2	01	1	0	0	1
7	2	10	0	0	1	1
8	3	00	1	0	0	0
9	3	01	0	0	0	1
10	4	00	0	0	0	0

Accesses of operands requiring more than one bus cycle are performed sequentially, with no idle T-States separating them. The number of bus cycles required to transfer an operand depends on its size and its alignment. Table 3-2 lists the bus cycles performed for each situation.

3.4.4.1 Bit Accesses

The Bit Instructions perform byte accesses to the byte containing the designated bit. The Test and Set Bit instruction (SBIT), for example, reads a byte, alters it, and rewrites it, having changed the contents of one bit.

3.4.4.2 Bit Field Accesses

An access to a Bit Field in memory always generates a Double-Word transfer at the address containing the least significant bit of the field. The Double Word is read by an Extract instruction; an Insert instruction reads a Double Word, modifies it, and rewrites it.

3.4.4.3 Extending Multiple Accesses

The Extending Multiply Instruction (MEI) will return a result which is twice the size in bytes of the operand it reads. If the multiplicand is in memory, the most-significant half of the result is written first (at the higher address), then the least-significant half. This is done in order to support retry if this instruction is aborted.

3.4.5 Instruction Fetches

Instructions for the NS32332 CPU are "prefetched"; that is, they are input before being needed into the next available entry of the twenty-byte Instruction Queue. The CPU performs two types of Instruction Fetch cycles: Sequential and Non-Sequential. These can be distinguished from each other by their differing status combinations on pins ST0–ST3 (Sec. 3.4.3).

A Sequential Fetch will be performed by the CPU whenever the Data Bus would otherwise be idle and the Instruction Queue is not currently full. Sequential Fetches are always type 10 Read cycles (Table 3-1).

A Non-Sequential Fetch occurs as a result of any break in the normally sequential flow of a program. Any jump or branch instruction, a trap or an interrupt will cause the next Instruction Fetch cycle to be Non-Sequential. In addition, certain instructions flush the instruction queue, causing the next instruction fetch to display Non-Sequential status. Only the first bus cycle after a break displays Non-Sequential status, and that cycle depends on the destination address.

If a non-sequential fetch is followed by additional sequential fetches which are burst continuation of the non-sequential fetch, then the Status Bus (ST0–ST3) remains the same.

Note 1: During instruction fetch cycles, $\overline{BE0}$–$\overline{BE3}$ are all active regardless of the alignment.

Note 2: During Operand Access cycles $\overline{BE0}$–$\overline{BE3}$ are activated as if the bus is 32 bits wide, regardless of the real width.

3.4.6 Interrupt Control Cycles

Activating the $\overline{INT}$ or $\overline{NMI}$ pin on the CPU will initiate one or more bus cycles whose purpose is interrupt control rather than the transfer of instructions or data. Execution of the Return from Interrupt instruction (RETI) will also cause Interrupt Control bus cycles. These differ from instruction or data transfers only in the status presented on pins ST0–ST3. All Interrupt Control cycles are single-byte Read cycles.

This section describes only the Interrupt Control sequences associated with each interrupt and with the return from its service routine.

3.0 Functional Description (Continued)

TABLE 3-2
Access Sequences

Cycle	Type	Address	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Data Bus: Byte 3	Byte 2	Byte 1	Byte 0
A. ***Word at address ending with 11***									BYTE 1 | BYTE 0	← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	1	A + 1	1	1	1	0	X	X	X	Byte 1
B. ***Double word at address ending with 01***							BYTE 3	BYTE 2	BYTE 1	BYTE 0 ← A
1.	9	A	0	0	0	1	Byte 2	Byte 1	Byte 0	X
2.	1	A + 3	1	1	1	0	X	X	X	Byte 3
C. ***Double word at address ending with 10***							BYTE 3	BYTE 2	BYTE 1	BYTE 0 ← A
1.	7	A	0	0	1	1	Byte 1	Byte 0	X	X
2.	5	A + 2	1	1	0	0	X	X	Byte 3	Byte 2
D. ***Double word at address ending with 11***							BYTE 3	BYTE 2	BYTE 1	BYTE 0 ← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	8	A + 1	1	0	0	0	X	Byte 3	Byte 2	Byte 1
E. ***Quad word at address ending with 00***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	10	A	0	0	0	0	Byte 3	Byte 2	Byte 1	Byte 0
Other bus cycles (instruction prefetch or slave) can occur here.										
2.	10	A + 4	0	0	0	0	Byte 7	Byte 6	Byte 5	Byte 4
F. ***Quad word at address ending with 01***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	9	A	0	0	0	1	Byte 2	Byte 1	Byte 0	X
2.	1	A + 3	1	1	1	0	X	X	X	Byte 3
Other bus cycles (instruction prefetch or slave) can occur here.										
3.	9	A + 4	0	0	0	1	Byte 6	Byte 5	Byte 4	X
4.	1	A + 7	1	1	1	0	X	X	X	Byte 7
G. ***Quad word at address ending with 10***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	7	A	0	0	1	1	Byte 1	Byte 0	X	X
2.	5	A + 2	1	1	0	0	X	X	Byte 3	Byte 2
Other bus cycles (instruction prefetch or slave) can occur here.										
3.	7	A + 4	0	0	1	1	Byte 5	Byte 4	X	X
4.	5	A + 6	1	1	0	0	X	X	Byte 7	Byte 6
H. ***Quad word at address ending with 11***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	8	A + 1	1	0	0	0	X	Byte 3	Byte 2	Byte 1
Other bus cycles (instruction prefetch or slave) can occur here.										
1.	4	A + 4	0	1	1	1	Byte 4	X	X	X
2.	8	A + 5	1	0	0	0	X	Byte 7	Byte 6	Byte 5

X = Don't Care

3.0 Functional Description (Continued)

TABLE 3-3
Interrupt Sequences

Cycle	Status	Address	$\overline{DDIN}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Data Bus Byte 3	Byte 2	Byte 1	Byte 0
					A. Non-Maskable Interrupt Control Sequences						
Interrupt Acknowledge											
1	0100	$FFFFFF00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
None: Performed through Return from Trap (RETT) instruction.											
					B. Non-Vectored Interrupt Control Sequences						
Interrupt Acknowledge											
1	0100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
1	0110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	X
					C. Vectored Interrupt Sequences: Non-Cascaded.						
Interrupt Acknowledge											
1	0100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Range: 0-127
Interrupt Return											
1	0110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Same as in Previous Int. Ack. Cycle
					D. Vectored Interrupt Sequences: Cascaded						
Interrupt Acknowledge											
1	0100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: range −16 to −1
(The CPU here uses the Cascade Index to find the Cascade Address.)											
2	0101	Cascade Address	0			See Note		Vector, range 9–255; on appropriate byte of data bus.			
Interrupt Return											
1	0110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: Same as in previous Int. Ack. Cycle
(The CPU here uses the Cascade Index to find the Cascade Address)											
2	0111	Cascade Address	0			See Note		X	X	X	X

X = Don't Care

Note: $\overline{BE0}$-$\overline{BE3}$ signals will be activated according to the cascaded ICU address. The cycle type can be 1, 2, 3 or 4, when reading the interrupt vector. The vector value can be in the range 0–255.

3.0 Functional Description (Continued)

3.4.7 Dynamic Bus Configuration

The NS32332 interfaces to external data buses with 3 different widths: 8-bit, 16-bit and 32-bit. The NS32332 can switch from one bus width to another dynamically i.e. on a cycle by cycle basis.

This feature allows the user to include in his system different bus sizes for different purposes, like 8-bit bus for bootstrap ROM and 32-bit bus for cache memory, etc.

In each memory cycle, the bus width is determined by the inputs BW0 and BW1.

Four combinations exist:

BW1	BW0	
0	0	reserved
0	1	8-bit bus
1	0	16-bit bus
1	1	32-bit bus

The dynamic bus configuration is not applicable for slave cycles (see Sec. 3.4.1).

The BW0–BW1 lines are sampled by the CPU in T3 with the falling edge of PHI1 (see *Figure 3-14*).

If the bus width didn't change from the previous memory cycle, the CPU terminates the cycle normally.

If the bus width of the current cycle is different from the bus width of the previous cycle, then two WAIT states (see Sec. 3.4.1) must be inserted in order to let the CPU switch to the new width.

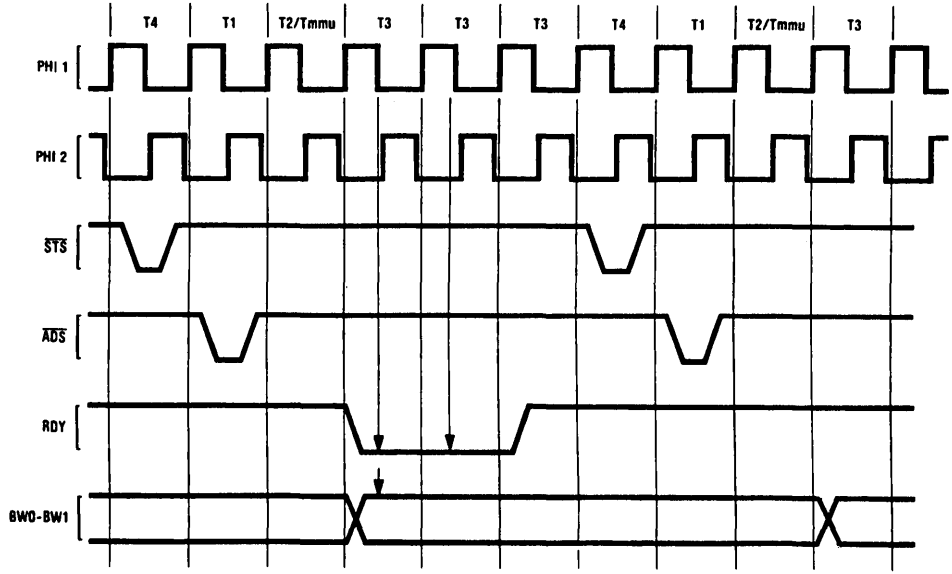
The additional 2 WAIT states count from the moment BW0 BW1 change. This can be overlapped with the wait states due to slow memories.

Note: BW0–BW1 can only be changed during the first T3 state of a memory access cycle. They should be externally latched and should not be changed at any other time.

In write cycles, the appropriate data will be present on the appropriate data lines. The CPU presents the data during T3 in a way that would fit any bus width.

If the operand being written is a byte, it will be duplicated on the 4 bytes AD0–AD31 depending on the operand address:

Address A0–1 =	00	XX	XX	XX	OP
	01	XX	XX	OP	OP
	10	XX	OP	XX	OP
	11	OP	XX	OP	OP



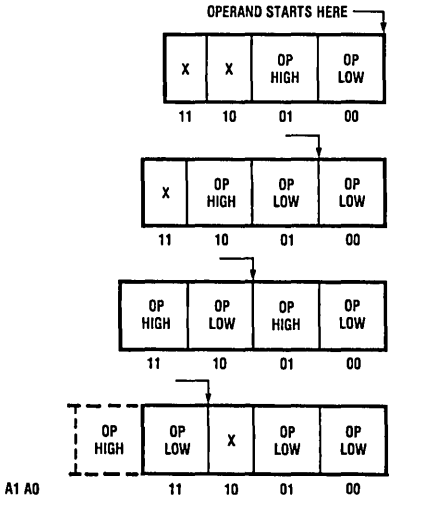
TL/EE/8673-23

FIGURE 3-14. Bus width changes. Two wait states are required after the signals BW0–BW1 change.

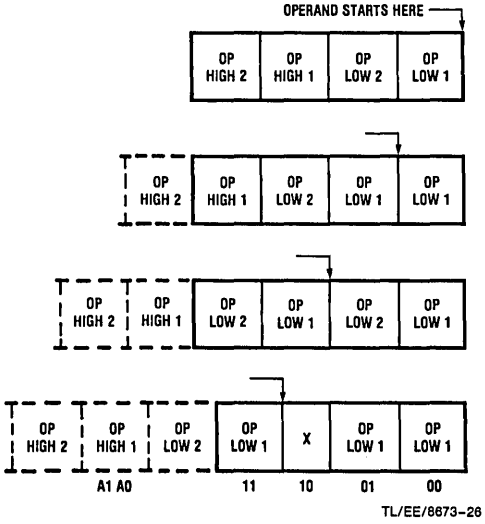
3.0 Functional Description (Continued)

If the operand being written is a word, 4 cases exist. The operand address can be x...x00 (binary) or x...x01 (binary) or x...x10 or x...x11 (binary).

See the duplications for each case:



TL/EE/8673-25



TL/EE/8673-26

If the operand being written is a double word 4 cases exist: The operand address can be x...x00 (binary) or x...x01 (binary) or x...x10 (binary) or x...x11 (binary).

See the duplications for each case:

Note that the organization of the operand described applies to the initial part of the operand cycle. For instance, if the CPU writes a double word operand to a 16-bit bus and the operand address is x...x11 (binary) it needs three memory cycles.

The description above applies to the first cycle. In the other 2 memory cycles belonging to the same operand, the data will be presented on the data bus lines to fit 16-bit bus width and take into account the operand length.

Example:

The CPU has to write a double word DDCCBBAA to address HEX 987653 which is in a 16-bit bus area. In the first cycle, the CPU does not know the width until T3 so it generates a cycle to address 987653 which activates the $\overline{BE3}$ line and puts on the data bus AA XX AA AA (X = don't care). After this cycle, the CPU knows it has a 16-bit bus and it generates a cycle to address 987654 which activates the BE0, BE1 and BE2 lines and puts on the data bus XX XX CC BB. The last cycle will address 987656, activate BE2, and put on the data bus XX XX XX DD. The BE0–BE3 lines are always activated as if the bus is 32-bit wide, regardless of BW0–BW1 state.

The CPU does not support a change of the bus width during a sequence of several memory references belonging to the same operand e.g. nonaligned double word. In other words, any operand should not be split between two memory spaces having different bus widths.

Instruction Fetches do not fall in this category and an Instruction Fetch can have its own bus width regardless of the bus width in the previous cycle.

3.4.8 Bus Exceptions

Any bus cycle may have a bus error during its execution. The error may be corrected during the current cycle or may be incorrectable. The NS32332 can handle both types of errors by means of BUS RETRY and BUS ERROR.

3.4.8.1 Bus Retry

If a bus error can be corrected, the CPU may be requested to repeat the erroneous bus cycle. The request is done by asserting the $\overline{BRT}$ (Bus Retry) signal.

The CPU response to Bus Retry depends on the cycle type:

Instruction Fetch Cycle—If the RETRY occurs during an instruction fetch, the fetch cycle will be retried as soon as possible. If the RETRY is requested during a burst chain, the burst is stopped and the fetch is retried. The only delay in retrying the instruction fetch may result from pending operand requests (and, of course, from hold or wait requests).

The fetch cycle will be retried only if there are no more than four bytes in the queue.

Operand Read Cycle—If the RETRY occurs on an operand read, the bus cycle is immediately repeated. If the data read is "multiple" e.g. non-aligned, only the problematic part will be repeated. For instance, if the cycle is a non-aligned double word and the second half failed, only the second part will be repeated. The same applies for a RETRY occurring during a burst chain. The repeated cycle will begin where the read operand failed (rather than the first address of the burst) and will finish the original burst.

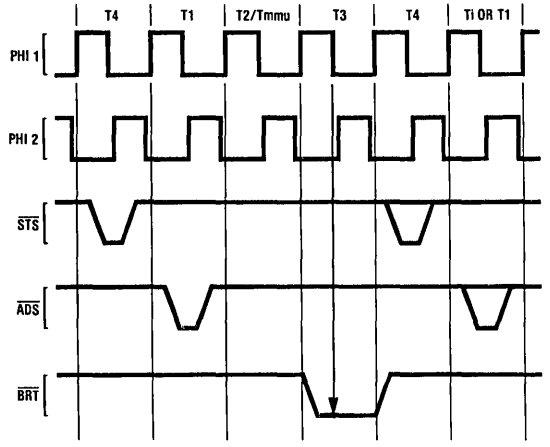
3.0 Functional Description (Continued)

Operand Write Cycle—If the RETRY occurs on a write, the bus cycle is immediately repeated. If the operand write is "multiple" e.g. non-aligned, only the problematic part will be repeated. For instance, if the cycle is a non-aligned double word and the second half failed, only the second part will be repeated.

A Bus Retry is requested by activating the $\overline{BRT}$ line (see *Figure 3-15*). $\overline{BRT}$ is sampled by the CPU during T3 on the falling edge of PHI1. If $\overline{BRT}$ is inactive, the cycle will be terminated in a regular way. In this case $\overline{BRT}$ must also be kept inactive during T4. If $\overline{BRT}$ is active, $\overline{BRT}$ will be sampled again during T4 on the falling edge of PHI1. If $\overline{BRT}$ is inactive, the cycle will be terminated in a regular way. If $\overline{BRT}$ is active, T4 will be followed by an idle state and the cycle will be repeated, i.e. a new T4 for setting the Status Bus and issuing $\overline{STS}$ and then T1 through T4 will be performed.

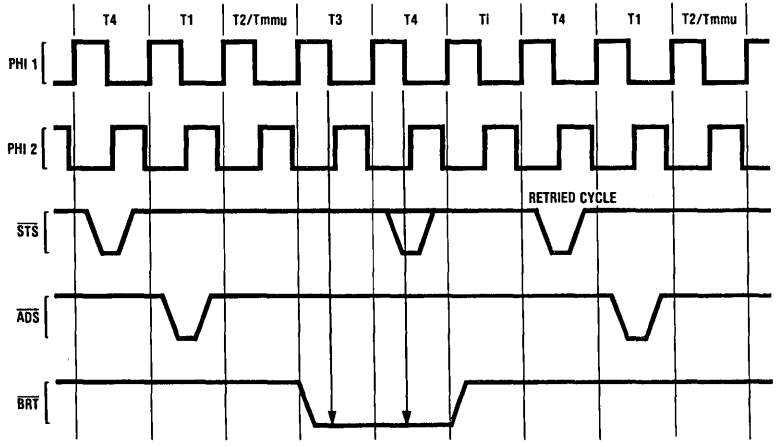
Although the decision about Retry is taken by the CPU on T4, $\overline{BRT}$ must have an early activation in T3 as described above in order to prevent the internal pipeline to advance. Holding the pipeline allows the repeated cycle to override the original one. If $\overline{BRT}$ is activated only in T3 and not in T4, there might be one cycle penalty in the performance of the execution unit in operand read cycles.

Retry is applicable for regular memory cycles and burst cycles, but not for Slave cycles.



TL/EE/8673-27

(a) Bus Cycle Not Retried



TL/EE/8673-28

(b) Bus Cycle Retried

FIGURE 3-15. Bus Cycle Retry

3.0 Functional Description (Continued)

3.4.8.2 Bus Error

If a Bus Error is incorrectable the CPU may be requested to abort the current process and branch to an appropriate routine to handle the error. The request is performed by activating the $\overline{BER}$ signal.

$\overline{BER}$ is sampled by the CPU during T4 on the falling edge of PHI1. If $\overline{BER}$ is active the bus will go to Tidle after T4 and the CPU will jump to the Bus Error handler (see Sec. 3.8).

The CPU response to Bus Error depends on the cycle type:

Instruction Fetch Cycles—If the bus error occurs on an instruction fetch, additional fetches are inhibited including the one which failed. If, after inhibiting instruction fetches, some operand cycles are still pending within the CPU, they are executed normally, delaying the access to the bus error exception. If and when the internal instruction queue becomes empty, the CPU will enter the BUS ERROR exception. This arrangement enables the CPU to ignore bus errors which belong to fetch ahead cycles if these fetches are not to be used as a result of a jump.

Operand Read Cycles—If the bus error occurs on an operand read, the bus error is immediately accepted, and the CPU enters the BUS ERROR exception.

Operand Write Cycles—If the bus error occurs on an operand write, the exception is immediately accepted.

Note 1: When a bus error occurs, the instruction that caused the error is generally not re-executable.

The process that was being executed should either be aborted or should be restarted from the last checkpoint.

Note 2: Bus error has top priority and is accepted even during the acknowledge sequence of another CPU exception (i.e. Abort, Interrupt, etc.).

It is the responsibility of the user software to detect such an occurence and to take the appropriate corrective actions.

3.4.8.3 Fatal Bus Error

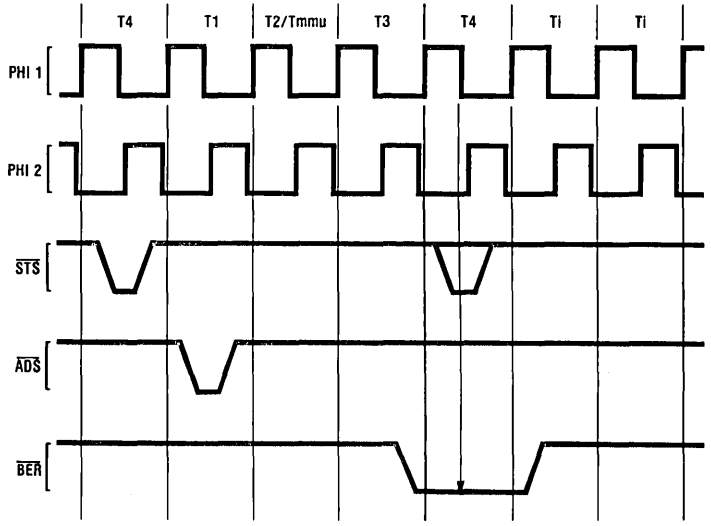
As previously mentioned, the CPU response to a bus error is to interrupt the current activity and enter the error routine.

An exception to this rule occurs when a bus error is signalled to the CPU during the acknowledge of a previous bus error. In this case the second error is interpreted by the CPU as a fatal bus error.

The CPU will respond to this event by halting execution and floating $\overline{ADS}$, $\overline{BE0}$–$\overline{BE3}$, $\overline{DDIN}$, $\overline{STS}$ and AD0–AD31.

The Halt condition is indicated by the setting of ST0–ST3 to zero and by the assertion of $\overline{MC}/\overline{EXS}$ for more than one clock cycle (see Sec. 4.1.3).

The CPU can exit this condition only through a hardware reset.



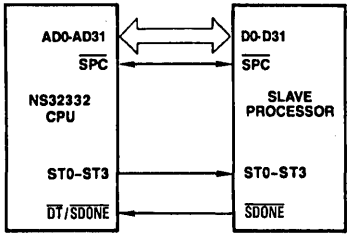
TL/EE/8673–30

FIGURE 3-16. Bus Error During Read or Write Cycle

3.0 Functional Description (Continued)

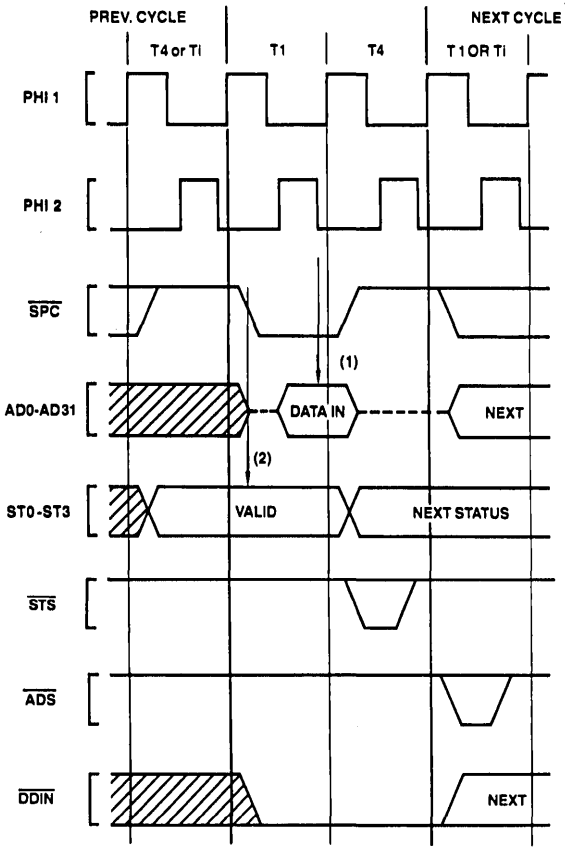
3.4.9 Slave Processor Communication

The $\overline{SPC}$ pin is used as the data strobe for Slave Processor transfers. In this role, it is referred to as Slave Processor Control ($\overline{SPC}$). In a Slave Processor bus cycle, data is transferred on the Data Bus and the status lines (ST0–ST3) are monitored by each Slave Processor in order to determine the type of transfer being performed. $\overline{SPC}$ is bidirectional, but is driven by the CPU during all Slave Processor bus cycles. See Sec. 3.9 for full protocol sequences.



TL/EE/8673-31

FIGURE 3-17. Slave Processor Connections



TL/EE/8673-32

Notes:

(1) CPU samples Data Bus here.

(2) Slave Processor samples CPU Status here.

FIGURE 3-18. CPU Read from Slave Processor

3.0 Functional Description (Continued)

3.4.9.1 Slave Processor Bus Cycles

A Slave Processor bus cycle always takes exactly two clock cycles, labeled T1 and T4 (see *Figures 3-18* and *3-19*). During a Read cycle, $\overline{SPC}$ is active from the beginning of T1 to the beginning of T4, and the data is sampled at the end of T1. The Cycle Status pins lead the cycle by one clock period, and are sampled at the leading edge of $\overline{SPC}$. During a Write cycle, the CPU applies data and activates $\overline{SPC}$ at T1, removing $\overline{SPC}$ at T4. The Slave Processor latches status on the leading edge of $\overline{SPC}$ and latches data on the trailing edge.

The CPU does not pulse the address ($\overline{ADS}$) and status ($\overline{STS}$) strobes during a slave protocol. The direction of a transfer is determined by the sequence ("protocol") established by the instruction under execution; but the CPU indicates the direction on the $\overline{DDIN}$ pin for hardware debugging purposes.

3.4.9.2 Slave Operand Transfer Sequences

A Slave Processor operand is transferred in one or more slave operand cycles. The NS32332 supports two slave protocols which can be selected by the configuration register (CFG).

1. The regular Slave protocol is fully compatible with NS32032, NS32016 and NS32008 slave protocols.

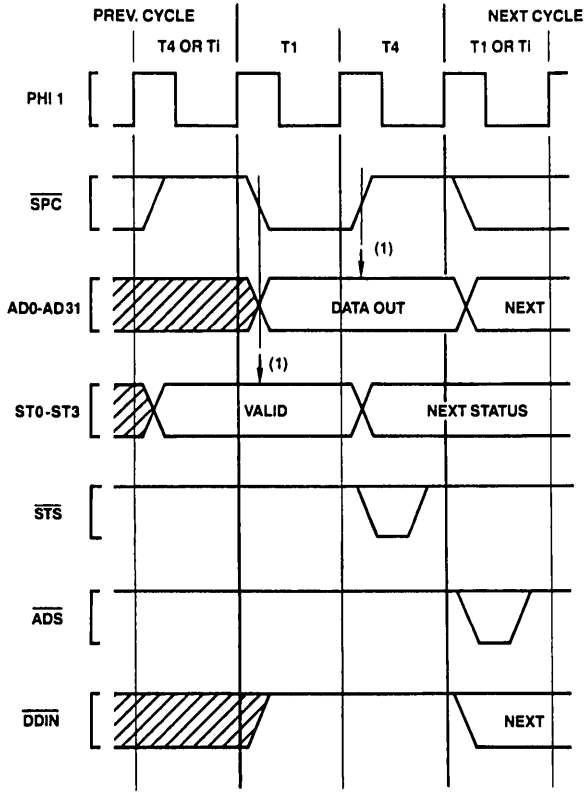
   In this protocol the NS32332 uses only the two least significant bytes of the data bus for slave cycles. This allows the NS32332 CPU to work with the current slaves (like NS32082, NS32081 etc.)

   A byte operand is transferred on the least significant byte of the data bus (AD0–AD15).

   A double word is transferred in a consecutive pair of bus cycles least significant word first. A quadword is transferred in two pairs of slave cycles.

2. The fast slave protocol is unique to the NS32332 CPU. In this protocol the NS32332 uses the full width of the data bus (AD0–AD31) for slave cycles.

   A byte operand is transferred on the least significant byte of the data bus (AD0–AD7), a word operand is transferred on bits AD0–AD15 and a double word operand is transferred on bits AD0–AD31. A quad word is transferred in two pairs of slave cycles with other bus cycles possibly occurring between them.



TL/EE/8673–33

Note:

(1) Arrows indicate points at which the Slave Processor samples.

FIGURE 3-19. CPU Write to Slave Processor

3.0 Functional Description (Continued)

3.5 MEMORY MANAGEMENT OPTION

The NS32332 CPU, in conjunction with the Memory Management Unit (MMU), provides full support for address translation, memory protection, and memory allocation techniques up to and including Virtual Memory.

When an MMU is used, the states T2 and TMMU are overlapped. During this time the CPU places AD0–AD31 into the TRI-STATE mode, allowing the MMU to assert the translated address and issue the physical address strobe $\overline{PAV}$. *Figure 3-20* shows the Bus Cycle timing with address translation.

Note 1: If an NS32382 MMU is used, the CPU can be selected to output data during write cycles in state T2, by forcing $\overline{DT}/\overline{SDONE}$ low during reset. This can be done because the NS32382 uses a separate physical address bus.

However, if a write cycle causes an MMU page table lookup, the CPU data will be valid in state T3. After $\overline{FLT}$ is deasserted, regardless of the data timing selected.

DT/$\overline{SDONE}$ must always be forced high during reset if an NS32082 MMU is used since, in this case, no separate physical address bus is provided.

Note 2: If an NS32082 MMU is used, in order for it to operate properly, it must be set to the 32-Bit mode by forcing a A24/$\overline{HBF}$ low during reset. In this mode the bus lines AD16–AD24 are floated after the MMU address has been latched, since they are used by the CPU to transfer data.

3.5.1 The $\overline{FLT}$ (Float) Pin

The $\overline{FLT}$ signal is used by the CPU for address translation support. Activating $\overline{FLT}$ during Tmmu causes the CPU to wait longer than Tmmu for address translation and validation. This feature is used occasionally by the MMU in order to update its Translation Lookaside Buffer (TLB) from page tables in memory, or to update certain status bits within them.

Figure 3-21 shows the effect of $\overline{FLT}$. Upon sampling $\overline{FLT}$ low, late in Tmmu, the CPU enters idle T-States (Tf) during which it:

1) Sets AD0–AD31, and $\overline{DDIN}$ to the TRI-STATE condition ("floating").
2) Suspends further internal processing of the current instruction. This ensures that the current instruction remains abortable with retry. (See $\overline{RST}/\overline{ABT}$ description.)

The above conditions remain in effect until $\overline{FLT}$ again goes high. See Sec. 4.

3.5.2 Aborting Bus Cycles

The $\overline{RST}/\overline{ABT}$ pin, apart from its Reset function (Sec. 3.3), also serves as the means to "abort", or cancel, a bus cycle and the instruction, if any, which initiated it. An Abort request is distinguished from a Reset in that the $\overline{RST}/\overline{ABT}$ pin is held active for only one clock cycle.

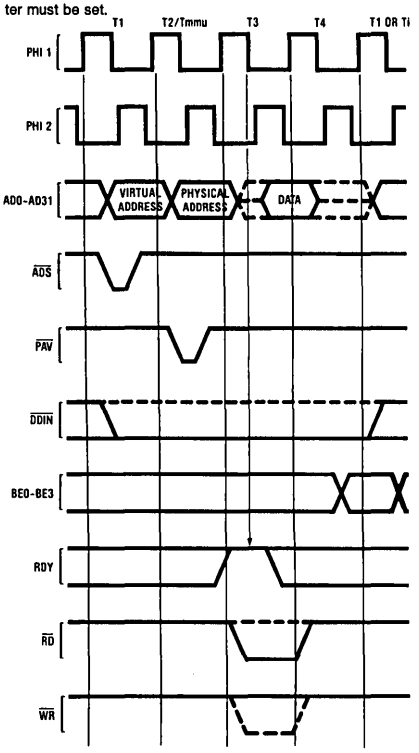
If $\overline{RST}/\overline{ABT}$ is pulled low during Tmmu or Tf, this signals that the cycle must be aborted. Since it is the MMU $\overline{PAV}$ signal which triggers a physical cycle, the rest of the system remains unaware that a cycle was started.

The MMU will abort a bus cycle for either of two reasons:

1) The CPU is attempting to access a virtual address which is not currently resident in physical memory. The referenced page must be brought into physical memory from mass storage to make it accessible to the CPU.
2) The CPU is attempting to perform an access which is not allowed by the protection level assigned to that page.

When a bus cycle is aborted by the MMU, the instruction that caused it to occur is also aborted in such a manner that it is guaranteed re-executable later.

Note: To guarantee correct instruction reexecution, Bit M in the CFG Register must be set.



TL/EE/8673–87

FIGURE 3-20. Read (Write) Cycle with Address Translation

3.5.2.1 Instruction Abort

Upon aborting an instruction, the CPU immediately interrupts the instruction and performs an abort acknowledge using the ABT vector in the Interrupt Table (see Sec. 3.8). The Return Address pushed on the Interrupt Stack is the address of the aborted instruction, so that a Return from Trap (RETT) instruction will automatically retry it.

The one exception to this sequence occurs if the aborted bus cycle was an instruction prefetch. If so, it is not yet certain that the aborted prefetched code is to be executed. Instead of causing an interrupt, the CPU only aborts the bus cycle, and stops prefetching. If the information in the Instruction Queue runs out, meaning that the instruction will actually be executed, the Abort will occur, in effect aborting the instruction that was being fetched.

3.5.2.2 Hardware Considerations

In order to guarantee instruction retry, certain rules must be followed in applying an Abort to the CPU. These rules are followed by the Memory Management Unit.

1) If $\overline{FLT}$ has not been applied to the CPU, the Abort pulse must occur during Tmmu.

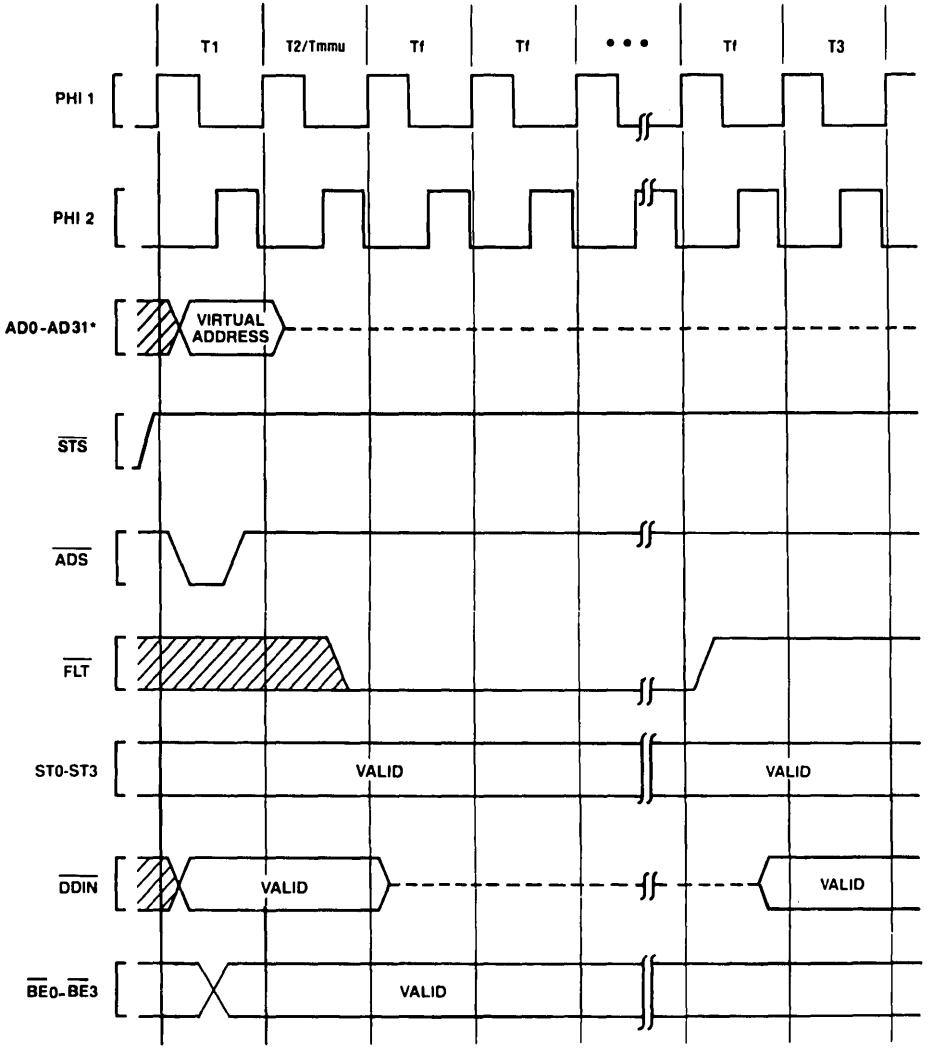
3.0 Functional Description (Continued)

2) If $\overline{FLT}$ has been applied to the CPU, the Abort pulse must be applied before the T-State in which $\overline{FLT}$ goes inactive. The CPU will not actually respond to the Abort command until $\overline{FLT}$ is removed.
3) The Write half of a Read-Modify-Write operand access may not be aborted. The CPU guarantees that this will never be necessary for Memory Management functions by applying a special RMW status (Status Code 1011) during the Read half of the access. When the CPU presents RMW status, that cycle must be aborted if it would be illegal to write to any of the accessed addresses.

If $\overline{RST}/\overline{ABT}$ is pulsed at any time other than as indicated above, it will abort either the instruction currently under execution or the next instruction and will act as a very high-priority interrupt. However, the program that was running at the time is not guaranteed recoverable.

3.6 BUS ACCESS CONTROL

The NS32332 CPU has the capability of relinquishing its access to the bus upon request from a DMA device or another CPU. This capability is implemented on the $\overline{HOLD}$ (Hold Request) and $\overline{HLDA}$ (Hold Acknowledge) pins. By asserting HOLD low, an external device requests access to the bus. On receipt of $\overline{HLDA}$ from the CPU, the device may perform bus cycles, as the CPU at this point has set the



*See MMU data sheet for details on physical address timing and MMU initiated Bus cycles.

TL/EE/8673-34

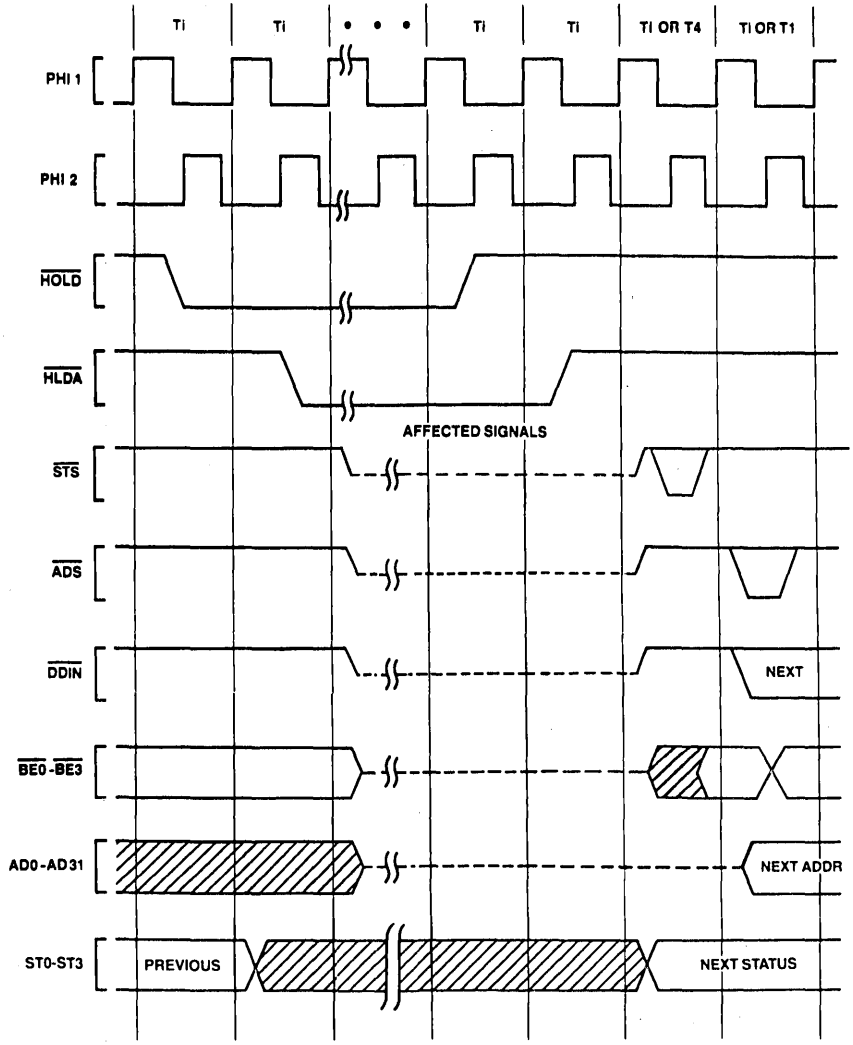
FIGURE 3-21. $\overline{FLT}$ Timing

3.0 Functional Description (Continued)

AD0–AD31, $\overline{ADS}$, $\overline{STS}$, $\overline{DDIN}$ and $\overline{BE0}$–$\overline{BE3}$ pins to the TRI-STATE condition. To return control of the bus to the CPU, the device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

How quickly the CPU releases the bus depends on whether it is idle on the bus at the time the $\overline{HOLD}$ request is made, as the CPU must always complete the current bus cycle. *Figure 3-22* shows the timing sequence when the CPU is idle. In this case, the CPU grants the bus during the immediately following clock cycle. *Figure 3-23* shows the sequence if the CPU is using the bus at the time that the $\overline{HOLD}$ request is made. If the request is made during or before the clock cycle shown (two clock cycles before T4), the CPU will release the bus during the clock cycle following T4. If the request occurs closer to T4, the CPU may already have decided to initiate another bus cycle. In that case it will not grant the bus until after the next T4 state. Note that this situation will also occur if the CPU is idle on the bus but has initiated a bus cycle internally.

In a Memory-Managed system, the $\overline{HLDA}$ signal is connected in a daisy-chain through the MMU, so that the MMU can release the bus if it is using it.



TL/EE/8673–35

FIGURE 3-22. $\overline{HOLD}$ Timing, Bus Initially Idle

3.0 Functional Description (Continued)

3.7 INSTRUCTION STATUS

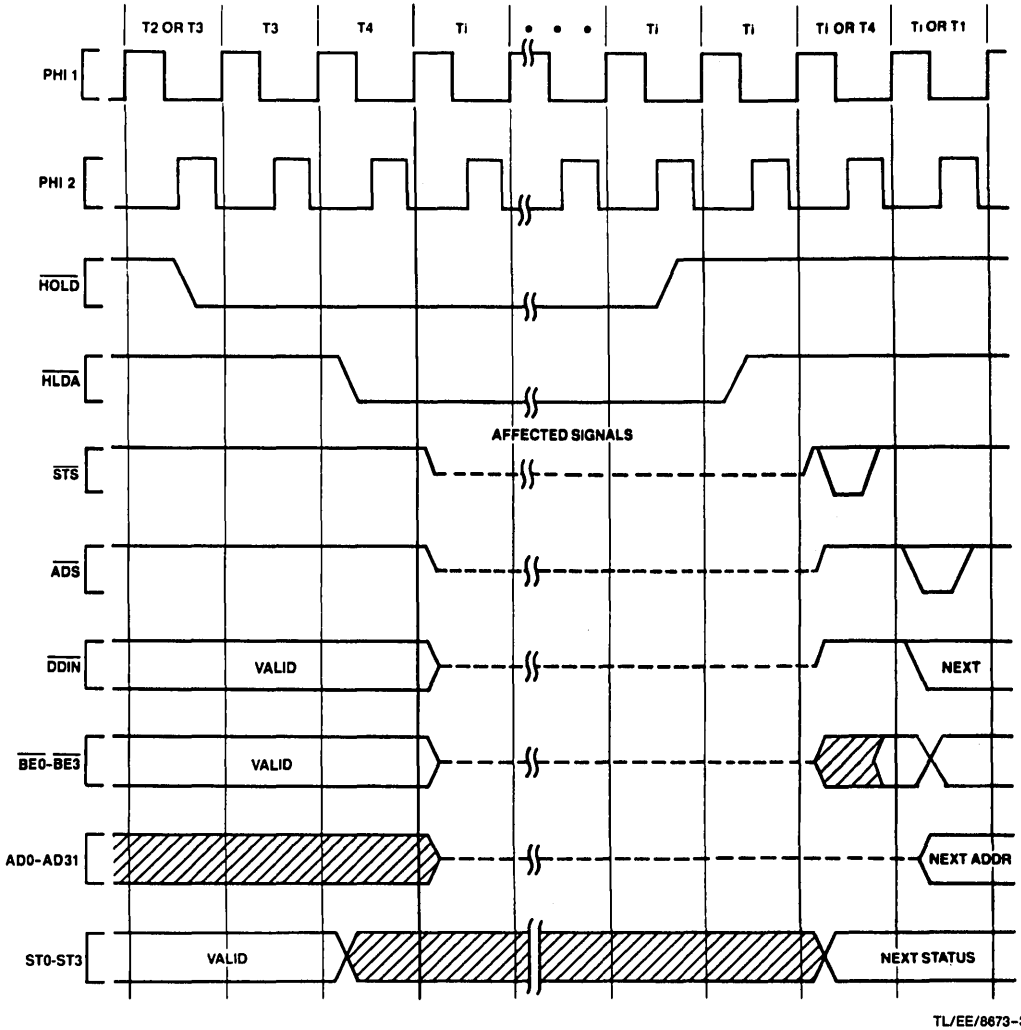
In addition to the four bits of Bus Cycle status (ST0–ST3), the NS32332 CPU also presents Instruction Status information on four separate pins. These pins differ from ST0–ST3 in that they are synchronous to the CPU's internal instruction execution section rather than to its bus interface section.

$\overline{PFS}$ (Program Flow Status) is pulsed low as each instruction begins execution. It is intended for debugging purposes.

U/$\overline{S}$ originates from the U bit of the Processor Status Register, and indicates whether the CPU is currently running in User or Supervisor mode. It is sampled by the MMU for mapping, protection, and debugging purposes. U/$\overline{S}$ line is updated every T4.

$\overline{ILO}$ (Interlocked Operation) is activated during an SBITI (Set Bit, Interlocked) or CBITI (Clear Bit, Interlocked) instruction. It is made available to external bus arbitration circuitry in order to allow these instructions to implement the semaphore primitive operations for multi-processor communication and resource sharing.

While $\overline{ILO}$ is active, the CPU inhibits instruction fetches. In order to prevent MMU cycles during $\overline{ILO}$, the CPU executes a dummy Read cycle with status code 1011 (RMW) prior to activating $\overline{ILO}$. Thereafter, $\overline{ILO}$ is activated and the Read is performed again but with status code 1010 (operand transfer). Refer to *Figure 3-24*.



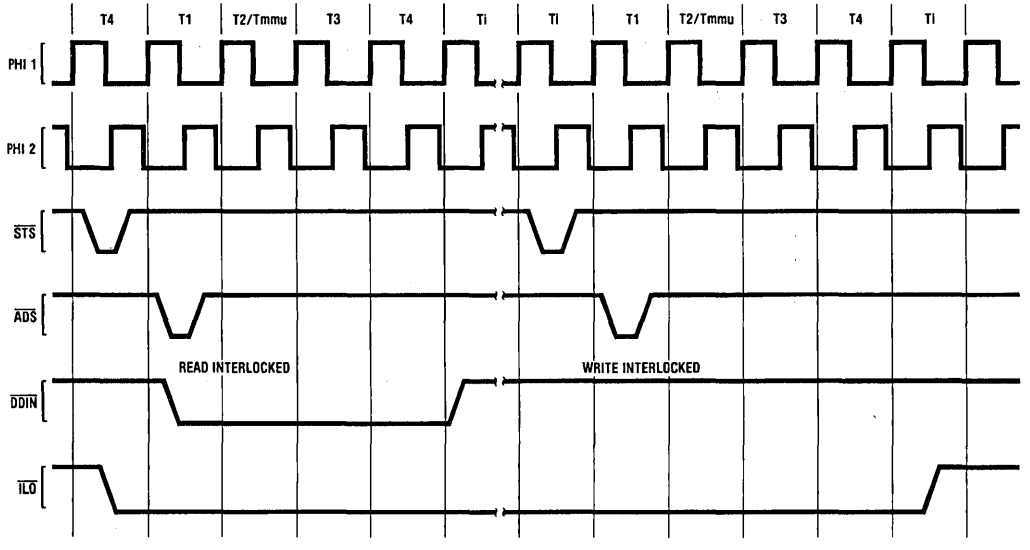
TL/EE/8673-36

FIGURE 3-23. $\overline{HOLD}$ Timing, Bus Initially Not Idle

3.0 Functional Description (Continued)

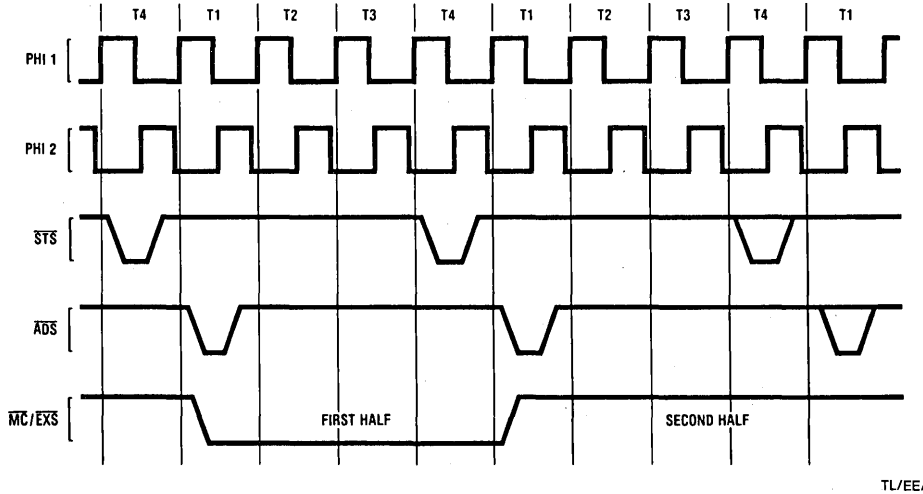
$\overline{MC}/\overline{EXS}$ (Multiple Cycle/Exception Status) is activated during the access of the first part of an operand that crosses a double-word address boundary. The activation of this signal is independent of the selected bus width. Its timing is shown in *Figure 3-25*. The MMU or other external circuitry can use it as an early indication of a CPU access to an operand that crosses a page boundary.

$\overline{MC}/\overline{EXS}$ is also activated during the first non-sequential instruction fetch (status code 1001) following an abort, and when the CPU enters the idle state (Status Code 0000) following a fatal bus error.



TL/EE/8673-37

FIGURE 3-24. $\overline{ILO}$ Timing



TL/EE/8673-38

FIGURE 3-25. Non-aligned Write Cycle—$\overline{MC}/\overline{EXS}$ Timing

3.0 Functional Description (Continued)

3.8 NS32332 INTERRUPT STRUCTURE

$\overline{INT}$, on which maskable interrupts may be requested,

$\overline{NMI}$, on which non-maskable interrupts may be requested, and

$\overline{RST}/\overline{ABT}$, which may be used to abort a bus cycle and any associated instruction. See Sec. 3.5.2.

In addition there is a set of internally-generated "traps" which cause interrupt service to be performed as a result either of exceptional conditions (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., the Supervisor Call instruction).

3.8.1 General Interrupt/Trap Sequence

Upon receipt of an interrupt or trap request, the CPU goes through three major steps:

1) Adjustment of Registers.

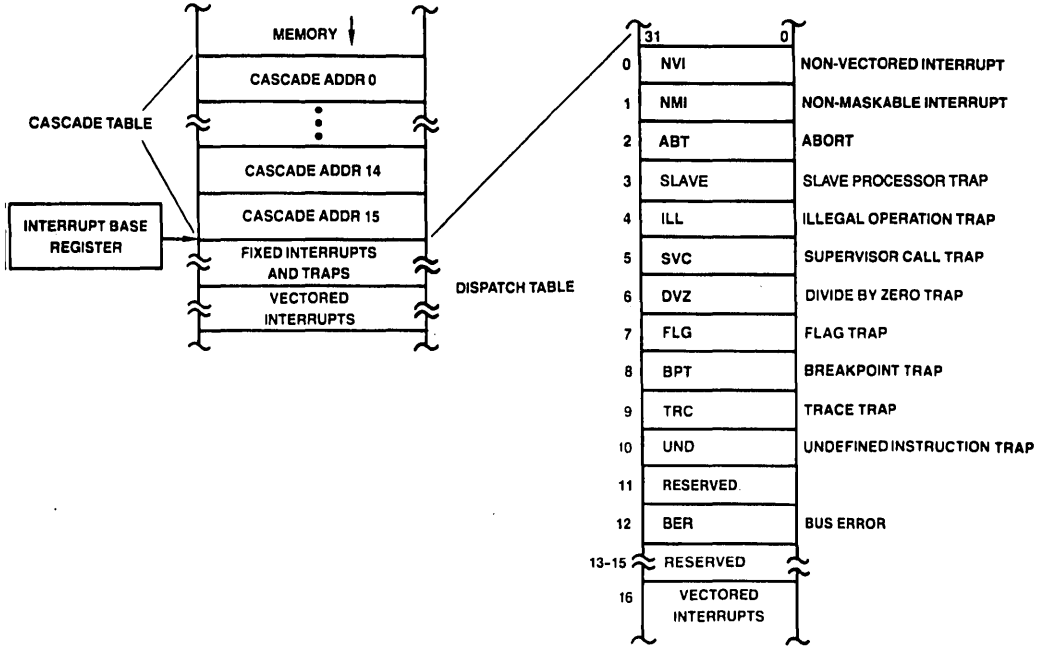
Depending on the source of the interrupt or trap, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR) and the currently-selected Stack Pointer (SP). A copy of the PSR is made, and the PSR is then set to reflect Supervisor Mode and selection of the Interrupt Stack.

2) Vector Acquisition.

A Vector is either obtained from the Data Bus or is supplied by default.

3) Service Call.

The Vector is used as an index into the Interrupt Dispatch Table, whose base address is taken from the CPU Interrupt Base (INTBASE) Register. See *Figure 3-26*. A 32-bit External Procedure Descriptor is read from the table entry, and an External Procedure Call is performed using it. The MOD Register (16 bits) and Program Counter (32 bits) are pushed on the Interrupt Stack.

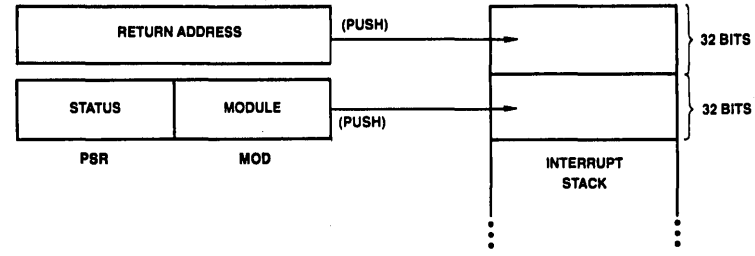


TL/EE/8673-39

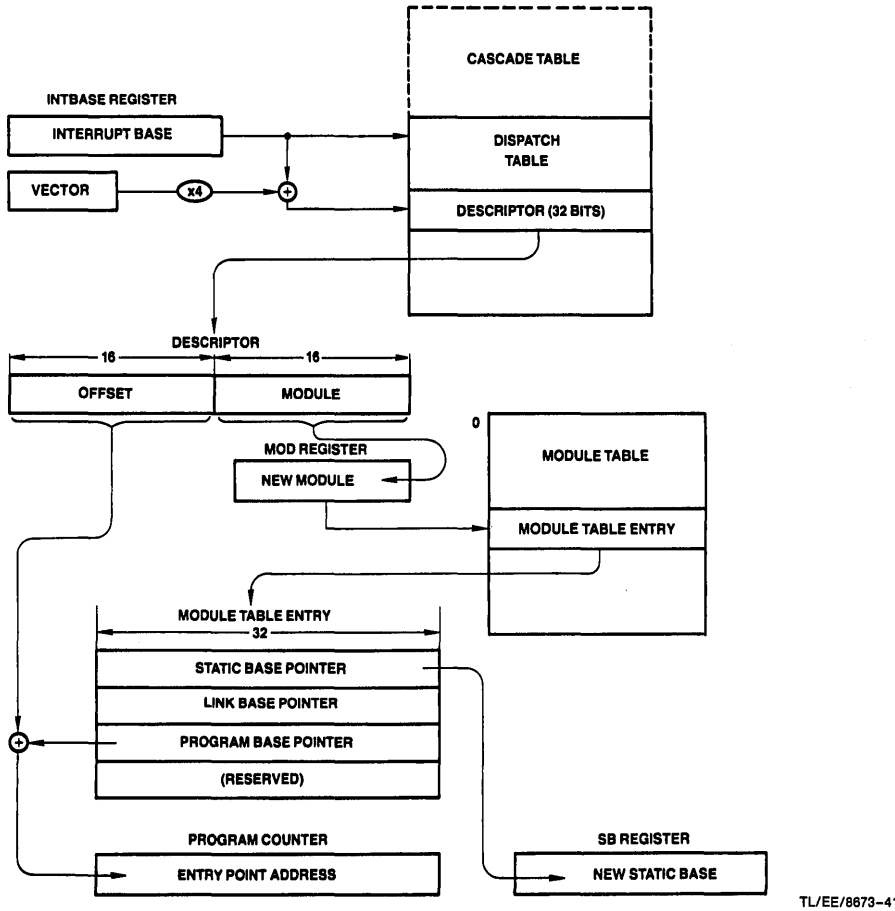
FIGURE 3-26. Interrupt Dispatch Table

3.0 Functional Description (Continued)

This process is illustrated in *Figure 3-27*, from the viewpoint of the programmer.



TL/EE/8673-40



TL/EE/8673-41

FIGURE 3-27. Interrupt/Trap Service Routine Calling Sequence

3.0 Functional Description (Continued)

3.8.2 Interrupt/Trap Return

To return control to an interrupted program, one of two instructions is used. The RETT (Return from Trap) instruction *(Figure 3-28)* restores the PSR, MOD, PC and SB registers to their previous contents and, since traps are often used deliberately as a call mechanism for Supervisor Mode procedures, it also discards a specified number of bytes from the original stack as surplus parameter space. RETT is used to return from any trap or interrupt except the Maskable Interrupt. For this, the RETI (Return from Interrupt) instruction is used, which also informs any external Interrupt Control Units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not pop parameters. See *Figure 3-29*.

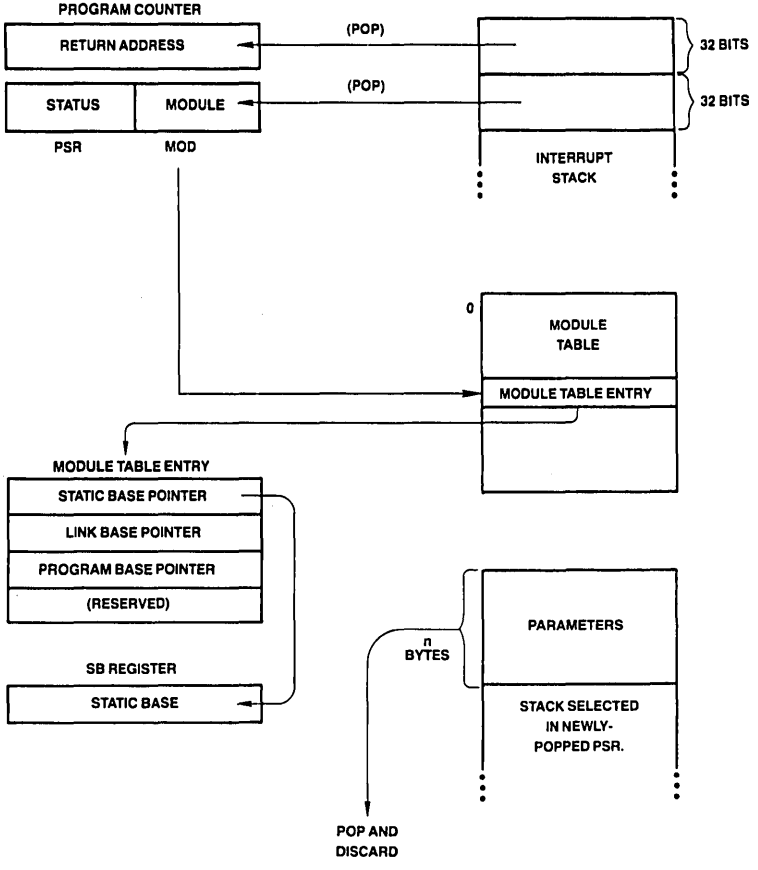
3.8.3 Maskable Interrupts (The $\overline{\text{INT}}$ Pin)

The $\overline{\text{INT}}$ pin is a level-sensitive input. A continuous low level is allowed for generating multiple interrupt requests. The input is maskable, and is therefore enabled to generate interrupt requests only while the Processor Status Register I bit is set. The I bit is automatically cleared during service of an $\overline{\text{INT}}$, $\overline{\text{NMI}}$ or Abort request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

The $\overline{\text{INT}}$ pin may be configured via the SETCFG instruction as either Non-Vectored (CFG Register bit I = 0) or Vectored (bit I = 1).

3.8.3.1 Non-Vectored Mode

In the Non-Vectored mode, an interrupt request on the $\overline{\text{INT}}$ pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.



TL/EE/8673–42

FIGURE 3-28. Return from Trap (RETT n) Instruction Flow

3.0 Functional Description (Continued)

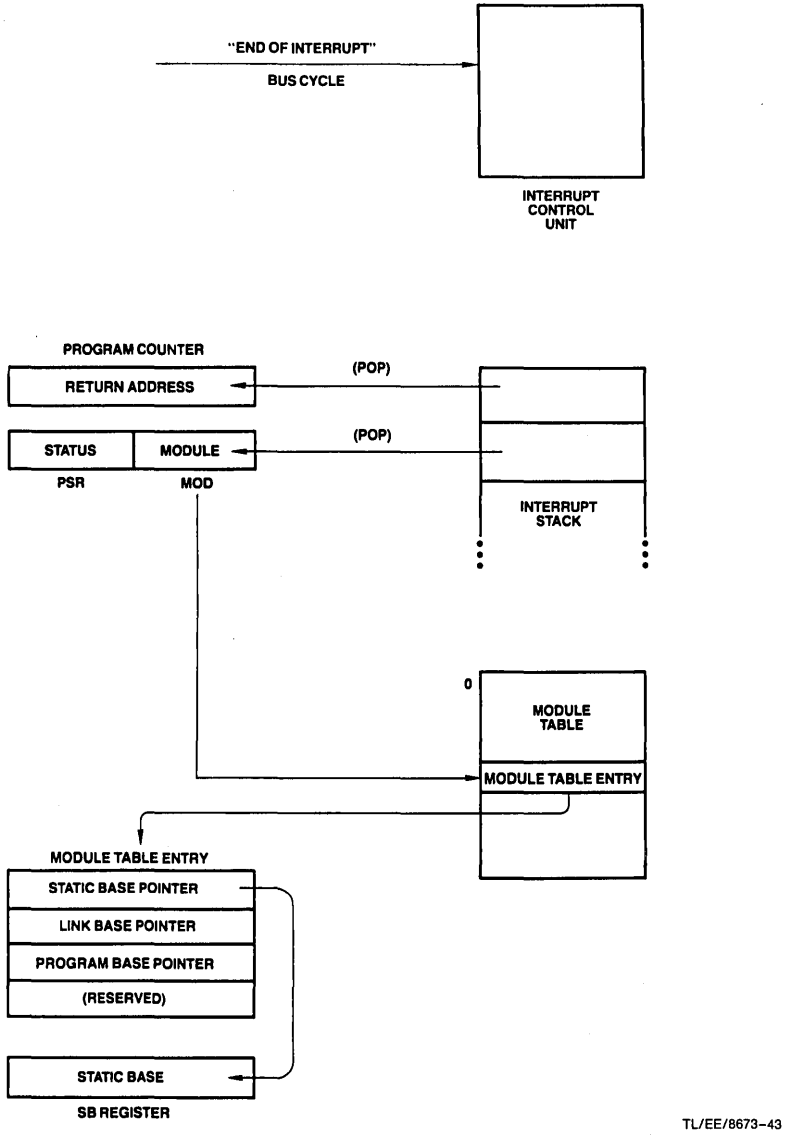


FIGURE 3-29. Return from Interrupt (RETI) Instruction Flow

3.0 Functional Description (Continued)

3.8.3.2 Vectored Mode: Non-Cascaded Case

In the Vectored mode, the CPU uses an Interrupt Control Unit (ICU) to prioritize many interrupt requests. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an "Interrupt Acknowledge, Master" bus cycle (Sec. 3.4.3) reading a vector value from the low-order byte of the Data Bus. This vector is then used as an index into the Dispatch Table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) instruction, which performs an End of Interrupt bus cycle, informing the ICU that it may re-prioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it needs also to inform a Cascaded ICU (see below).

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range of 0 through 127; that is, they must be positive numbers in eight bits. By providing a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU (see below).

3.8.3.3 Vectored Mode: Cascaded Case

In order to allow more levels of interrupt, provision is made in the CPU to transparently support cascading. Note that the Interrupt output from a Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin. Refer to the ICU data sheet for details.

In a system which uses cascading, two tasks must be performed upon initialization:

1) For each Cascaded ICU in the system, the Master ICU must be informed of the line number on which it receives the cascaded requests.
2) A Cascade Table must be established in memory. The Cascade Table is located in a NEGATIVE direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

Figure 3-26 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0 to 15) and subtract 16 from it, giving an index in the range −16 to −1. Multiply this value by 4, and add the resulting negative number to the contents of the INTBASE Register. The 32-bit entry at this address must be set to the address of the Hardware Vector Register of the Cascaded ICU. This is referred to as the "Cascade Address."

Upon receipt of an interrupt request from a Cascaded ICU, the Master ICU interrupts the CPU and provides the negative Cascade Table index instead of a (positive) vector number. The CPU, seeing the negative value, uses it as an index into the Cascade Table and reads the Cascade Address from the referenced entry. Applying this address, the CPU performs an "Interrupt Acknowledge, Cascaded" bus cycle (Sec. 3.4.3), reading the final vector value. This vector is interpreted by the CPU as an unsigned byte, and can therefore be in the range of 0 through 255.

In returning from a Cascaded interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any Maskable Interrupt. The CPU performs an "End of Interrupt, Master" bus cycle (Sec. 3.4.3), whereupon the Master ICU again provides the negative Cascade Table index. The CPU, seeing a negative value, uses it to find the corresponding Cascade Address from the Cascade Table. Applying this address, it performs an "End of Interrupt, Cascaded" bus cycle (Sec. 3.4.3), informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the interrupt mask register of the interrupt controller.

However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the $\overline{INT}$ line before the ICU deasserted it. This could cause the ICU to provide an invalid vector. To avoid this problem the above operation should be performed with the CPU interrupt disabled.

3.8.4 Non-Maskable Interrupt (The $\overline{NMI}$ Pin)

The Non-Maskable Interrupt is triggered whenever a falling edge is detected on the $\overline{NMI}$ pin. The CPU performs an "Interrupt Acknowledge, Master" bus cycle (Sec. 3.4.3) when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is $FFFFFF00_{16}$. The vector value used for the Non-Maskable Interrupt is taken as 1, regardless of the value read from the bus.

The service procedure returns from the Non-Maskable Interrupt using the Return from Trap (RETT) instruction. No special bus cycles occur on return.

For the full sequence of events in processing the Non-Maskable Interrupt, see Sec. 3.8.7.1.

3.8.5 Traps

A trap is an internally-generated interrupt request caused as a direct and immediate result of the execution of an instruction. The Return Address pushed by any trap except Trap (TRC) is the address of the first byte of the instruction during which the trap occurred. Traps do not disable interrupts, as they are not associated with external events. Traps recognized by the NS32332 CPU are:

Trap (SLAVE): An exceptional condition was detected by the Floating Point Unit or another Slave Processor during the execution of a Slave Instruction. This trap is requested via the Status Word returned as part of the Slave Processor Protocol (Sec. 3.9.1).

Trap (ILL): Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR bit U = 1).

Trap (SVC): The Supervisor Call (SVC) instruction was executed.

Trap (DVZ): An attempt was made to divide an integer by zero. (The Slave trap is used for Floating Point division by zero.)

Trap (FLG): The FLAG instruction detected a "1" in the CPU PSR F bit.

Trap (BPT): The Breakpoint (BPT) instruction was executed.

Trap (TRC): The instruction just completed is being traced. See below.

Trap (UND): An undefined opcode was encountered by the CPU.

3.0 Functional Description (Continued)

A special case is the Trace Trap (TRC), which is enabled by setting the T bit in the Processor Status Register (PSR). At the beginning of each instruction, the T bit is copied into the PSR P (Trace "Pending") bit. If the P bit is set at the end of an instruction, then the Trace Trap is activated. If any other trap or interrupt request is made during a traced instruction, its entire service procedure is allowed to complete before the Trace Trap occurs. Each interrupt and trap sequence handles the P bit for proper tracing, guaranteeing one and only one Trace Trap per instruction, and guaranteeing that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

Note: A slight difference exists between the NS32332 and previous Series 32000 CPUs when tracing is enabled.

The NS32332 always clears the P bit in the PSR before pushing the PSR on the stack. Previous CPUs do not clear it when a trap (ILL) occurs.

The result is that an instruction that causes a trap (ILL) exception is traced by previous Series 32000 CPUs, but is never traced by the NS32332.

3.8.6 Prioritization

The NS32332 CPU internally prioritizes simultaneous interrupt and trap requests as follows:

1) Traps other than Trace (Highest priority)
2) Abort
3) Bus Error
4) Non-Maskable Interrupt
5) Maskable Interrupts
6) Trace Trap (Lowest priority)

3.8.7 Interrupt/Trap Sequences: Detailed Flow

For purposes of the following detailed discussion of interrupt and trap service sequences, a single sequence called "Service" is defined in *Figure 3-30*. Upon detecting any interrupt request or trap condition, the CPU first performs a sequence dependent upon the type of interrupt or trap. This sequence will include pushing the Processor Status Register and establishing a Vector and a Return Address. The CPU then performs the Service sequence.

For the sequence followed in processing either Maskable or Non-Maskable interrupts (on the $\overline{INT}$ or $\overline{NMI}$ pins, respectively), see Sec. 3.8.7.1 For Abort Interrupts, see Sec. 3.8.7.4. For the Trace Trap, see Sec. 3.8.7.3, and for all other traps see Sec. 3.8.7.2.

3.8.7.1 Maskable/Non-Maskable Interrupt Sequence

This sequence is performed by the CPU when the $\overline{NMI}$ pin receives a falling edge, or the $\overline{INT}$ pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or, in the case of the String instructions, at the next interruptible point during its execution.

1. If a String instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P bit.
   b. Set "Return Address" to the address of the first byte of the interrupted instruction.

   Otherwise, set "Return Address" to the address of the next instruction.
2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, T, P and I.
3. If the interrupt is Non-Maskable:
   a. Read a byte from address $FFFFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master, Sec. 3.4.3). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.
4. If the interrupt is Non-Vectored:
   a. Read a byte from address $FFFFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Sec. 3.4.3). Discard the byte read.
   b. Set "Vector" to 0.
   c. Go to Step 8.
5. Here the interrupt is Vectored. Read "Byte" from address $FFFFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Sec. 3.4.3).
6. If "Byte" ≥ 0, then set "Vector" to "Byte" and go to Step 8.
7. If "Byte" is in the range −16 through −1, then the interrupt source is Cascaded. (More negative values are reserved for future use.) Perform the following:
   a. Read the 32-bit Cascade Address from memory. The address is calculated as INTBASE +4* Byte.
   b. Read "Vector," applying the Cascade Address just read and Status Code 0101 (Interrupt Acknowledge, Cascaded: Sec. 3.4.3).
8. Perform Service (Vector, Return Address), *Figure 3-30*.

Service (Vector, Return Address):

1) Read the 32-bit External Procedure Descriptor from the Interrupt Dispatch Table: address is Vector* 4 + INTBASE Register contents.
2) Move the Module field of the Descriptor into the MOD Register.
3) Read the Program Base pointer from memory address MOD + 8, and add to it the Offset field from the Descriptor, placing the result in the Program Counter.
4) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.
5) Flush queue: Non-sequentially fetch first instruction of Interrupt routine.
6) Push the PSR copy onto the Interrupt Stack as a 16-bit value.
7) Push MOD Register into the Interrupt Stack as a 16-bit value.
8) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.

FIGURE 3-30. Service Sequence

Invoked during all interrupt/trap sequences.

3.8.7.2 Trap Sequence: Traps Other Than Trace

1) Restore the currently selected Stack Pointer and the Processor Status Register to their original values at the start of the trapped instruction.
2) Set "Vector" to the value corresponding to the trap type.

SLAVE:	Vector = 3.
ILL:	Vector = 4.
SVC:	Vector = 5.
DVZ:	Vector = 6.
FLG:	Vector = 7.
BPT:	Vector = 8.
UND:	Vector = 10.

3) Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, P and T.

3.0 Functional Description (Continued)

4) Set "Return Address" to the address of the first byte of the trapped instruction.

5) Perform Service (Vector, Return Address), *Figure 3-30.*

3.8.7.3 Trace Trap Sequence

1) In the Processor Status Register (PSR), clear the P bit.

2) Copy the PSR into a temporary register, then clear PSR bits S, U and T.

3) Set "Vector" to 9.

4) Set "Return Address" to the address of the next instruction.

5) Perform Service (Vector, Return Address), *Figure 3-30.*

3.8.7.4 Abort Sequence

1) Restore the currently selected Stack Pointer to its original contents at the beginning of the aborted instruction.

2) Clear the PSR P bit.

3) Copy the PSR into a temporary register, then clear PSR bits S, U, T and I.

4) Set "Vector" to 2.

5) Set "Return Address" to the address of the first byte of the aborted instruction.

6) Perform Service (Vector, Return Address), *Figure 3-30.*

3.8.7.5 Bus Error Sequence

1) The same as Abort sequence above, but set vector to 12.

3.9 SLAVE PROCESSOR INSTRUCTIONS

The NS32332 CPU recognizes three groups of instructions being executable by external Slave Processor:

Floating Point Instruction Set

Memory Management Instruction Set

Custom Instruction Set

Each Slave Instruction Set is validated by a bit in the Configuration Register (Sec. 2.1.3). Any Slave Instruction which does not have its corresponding Configuration Register bit set will trap as undefined, without any Slave Processor communication attempted by the CPU. This allows software simulation of a non-existent Slave Processor.

In addition, each slave instruction will be performed either through the regular (32032 compatible) slave protocol or through a fast slave protocol according to the relevent bit in the configuration register (Sec. 2.1.3).

A combination of one slave communicating with an old protocol and another with a new protocol is allowed, e.g. 16-bit FPU (32081) and 32-bit MMU (32382) or vice versa.

3.9.1 16-Bit Slave Processor Protocol (32032 Compatible)

Slave Processor instructions have a three-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word. The ID Byte has three functions:

1) It identifies the instruction as being a Slave Processor instruction.

2) It specifies which Slave Processor will execute it.

3) It determines the format of the following Operation Word of the instruction.

Upon receiving a Slave Processor instruction, the CPU initiates the sequence outlined in *Figure 3-31.* While applying Status Code 1111 (Broadcast ID, Sec. 3.4.3), the CPU transfers the ID Byte on bits AD0–AD7 and a non-used byte xxxxxxx1 (x = don't care) on bits AD24–AD31. All Slave Processors input this byte and decode it. The Slave Processor selected by the ID Byte is activated, and from this point the CPU is communicating only with it. If any other slave protocol was in progress (e.g., an aborted Slave instruction), this transfer cancels it.

The CPU next sends the Operation Word while applying Status Code 1101 (Transfer Slave Operand, Sec. 3.4.3). Upon receiving it, the Slave Processor decodes it, and at this point both the CPU and the Slave Processor are aware of the number of operands to be transferred and their sizes. The operation Word is swapped on the Data Bus, that is, bits 0–7 appear on pins AD8–AD15 and bits 8–15 appear on pins AD0–AD7.

Using the Address Mode fields within the Operation Word, the CPU starts fetching operand and issuing them to the Slave Processor. To do so, it references any Addressing Mode extensions which may be appended to the Slave Processor instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the Slave processor. The Status Code applied is 1101 (Transfer Slave Processor Operand, Sec. 3.4.3).

After the CPU has issued the last operand, the Slave Processor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low. To allow for this $\overline{SPC}$ is normally held high only by an internal pull-up device of approximately 5 kΩ.

While the Slave Processor is executing the instruction, the CPU is free to prefetch instructions into its queue. If it fills the queue before the Slave Processor finishes, the CPU will wait, applying Status Code 0011 (Waiting for Slave, Sec. 3.4.3).

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a Status Word from the Slave Processor, applying Status Code 1110 (Read Slave Status, Sec. 3.4.3). This word has the format shown in *Figure 3-34.* If the Q bit ("Quit", Bit 0) is set, this indicates that an error was detected by the Slave Processor. The CPU will not continue the protocol, but will immediately trap through the SLAVE vector in the Interrupt Table. Certain Slave Processor instructions cause CPU PSR bits to be loaded from the Status Word.

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The Read cycles from the Slave Processor are performed by the CPU while applying Status Code 1101 (Transfer Slave Operand, Sec. 3.4.3).

An exception to the protocol above is the LMR (Load Memory Management Register) instruction, and a corresponding Custom Slave instruction (LCR: Load Custom Register). In executing these instructions, the protocol ends after the CPU has issued the last operand. The CPU does not wait for an acknowledgement from the Slave Processor, and it does not read status.

3.0 Functional Description (Continued)

Status Combinations:

Send ID (ID): Code 1111
Xfer Operand (OP): Code 1101
Read Status (ST): Code 1110

Step	Status	Action
1	ID	CPU Send ID Byte.
2	OP	CPU Sends Operaton Word.
3	OP	CPU Sends Required Operands
4	—	Slave Starts Execution. CPU Pre-fetches.
5	—	Slave Pulses $\overline{SPC}$ Low.
6	ST	CPU Reads Status Word. (Trap? Alter Flags?)
7	OP	CPU Reads Results (If Any).

FIGURE 3-31. 16-Bit Slave Processor Protocol

3.9.2 32-Bit Fast Slave Protocol

Upon receiving a Slave Processor instruction, the CPU initiates the sequence outlined in *Figure 3-32*. While applying Status code 1111 (Broadcast ID Sec. 3.4.2), the CPU transfers the ID Byte on bits AD24–AD31, the operation word on bits AD8–AD23 in a swapped order of bytes and a non-used byte XXXXXXX1 (X = don't care) on bits AD0–AD7 (*Figure 3-33*).

Using the addressing mode fields within the Operation word, the CPU fetches operands and sends them to the Slave Processor. Since the CPU is solely responsible for memory accesses, addressing mode extensions are not sent to the Slave Processor. The Status Code applied is 1101 (Transfer Slave Processor Operand Sec. 3.4.2). After the CPU has issued the last operand, the Slave Processor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SDONE}$ or $\overline{SPC}$ low for one clock cycle.

Unlike the old protocol, the SLAVE may request the CPU to read the status by activating the $\overline{SDONE}$ or $\overline{SPC}$ line for two clock cycles instead of one. The CPU will then read the slave status word and update the PSR Register, unless a trap is signalled. If this happens, the CPU will immediately abort the protocol and start a trap sequence using either the SLAVE or the UND vector in the interrupt table as specified in the Status Word.

Note: The PSR update is presently restricted to three instructions: CMPf, RDVAL, WRVAL and their custom slave equivalents.

While the Slave Processor is executing the instruction, the CPU is free to prefetch instructions into its queue. If it fills its queue before the Slave Processor finishes, the CPU will wait applying status code 0011 (waiting for Slave, Sec. 3.4.2).

Upon receiving the pulse on either $\overline{SDONE}$ or $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read the result from the Slave Processor and transfer it to the destination. The Read cycles from the Slave Processor are performed by the CPU while applying Status Code 1101 (Transfer Slave Operand, Sec. 3.4.2).

Status Combinations:

Send ID (ID): Code 1111
Xfer Operand (OP): Code 1101
Read Status (ST): Code 1110

Step	Status	Action
1	ID	CPU sends ID and Operation Word.
2	OP	CPU sends required operands (if any).
3	—	Slave starts execution (CPU prefetches).*
4	—	Slave pulses $\overline{SDONE}$ or $\overline{SPC}$ low.
5	ST	CPU Reads Status word (only if $\overline{SDONE}$ or $\overline{SPC}$ pulse is two clock cycles wide).
6	OP	CPU Reads Results (if any).

FIGURE 3-32. 32-Bit Fast Slave Protocol

Certain Slave Processor instructions affect CPU PSR. For these instructions only the CPU will perform a Read Slave status cycle as described in 3.9.1.1 before reading the result. The relevent PSR bits will be loaded from the status word.

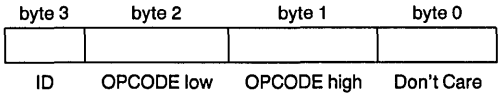


FIGURE 3-33. ID and Opcode Format for Fast Slave Protocol

3.9.3 Floating Point Instructions

Table 3-4 gives the protocols followed for each Floating Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Appendix A.

The Operand class columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see Instruction Set Reference Manual).

The Operand Issued columns show the sizes of the operands issued to the Floating Point Unit by the CPU. "D" indicates a 32-bit Double Word. "i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). "f" indicates that the instruction specifies a Floating Point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the Slave Processor Status Word (*Figure 3-34*).

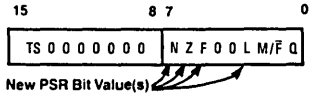
3.0 Functional Description (Continued)

TABLE 3-4
Floating Point Instruction Protocols.

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
POLYf	read.f	read.f	f	f	f to F0	none
DOTf	read.f	read.f	f	f	f to F0	none
SCALBf	read.f	rmw.f	f	f	f to Op.2	none
LOGBf	read.f	write.f	f	N/A	f to Op.2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none

Note 1:
D = Double Word
i = Integer size (B,W,D) specified in mnemonic.
f = Floating Point type (F,L) specified in mnemonic.
N/A = Not Applicable to this instruction.

Any operand indicated as being of type "f" will not cause a transfer if the Register addressing mode is specified. This is because the Floating Point Registers are physically on the Floating Point Unit and are therefore available without CPU assistance.



TL/EE/8673–44

FIGURE 3-34. Slave Processor Status Word Format

Note 1: Q is the Trap Bit. It is set to 1 by the Slave whenever a trap is requested.

Note 2: TS is the Trap Select Bit. When a trap is requested (Q = 1), TS tells the CPU whether a SLAVE or an UND trap is to be generated. TS is 0 for a slave trap and 1 for an UND trap.

Note 3: M/$\overline{F}$ should be set for a RDVAL, WRVAL, or Custom Slave Equivalent instruction. It should be cleared for CMPf and CCMP0c and CCMPc. When M/$\overline{F}$ is cleared, the F bit should also be cleared.

3.9.4 Memory Management Instructions

Table 3-5 gives the protocols for Memory Management instructions. Encodings for these instructions may be found in Appendix A.

In executing the RDVAL and WRVAL instructions, the CPU calculates and issues the 32-bit Effective Address of the single operand. The CPU then performs a single-byte Read cycle from that address, allowing the MMU to safely abort the instruction if the necessary information is not currently in physical memory. Upon seeing the memory cycle complete, the MMU continues the protocol, and returns the validation result in the F bit of the Slave Status Word.

The size of a Memory Management operand is always a 32-bit Double Word. For further details of the Memory Management Instruction set, see the Instruction Set Reference Manual and the MMU Data Sheet.

3.0 Functional Description (Continued)

TABLE 3-5
Memory Management Instruction Protocols.

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
RDVAL*	addr	N/A	D	N/A	N/A	F
WRVAL*	addr	N/A	D	N/A	N/A	F
LMR*	read.D	N/A	D	N/A	N/A	none
SMR*	write.D	N/A	N/A	N/A	D to Op. 1	none

Note:

In the RDVAL and WRVAL instructions, the CPU issues the address as a Double Word, and performs a single-byte Read cycle from that memory address. For details, see the Instruction Set Reference Manual and the Memory Management Unit Data Sheet.

D = Double Word

* = Privileged Instruction: will trap if CPU is in User Mode.

N/A = Not Applicable to this Instruction.

3.9.5 Custom Slave Instructions

Provided in the NS32332 is the capability of communicating with a user-defined, "Custom" Slave Processor. The instruction set provided for a Custom Slave Processor defines the instruction formats, the operand classes and the communication protocol. Left to the user are the interpretations of the Op Code fields, the programming model of the Custom Slave and the actual types of data transferred. The protocol specifies only the size of an operand, not its data type.

Table 3-6 lists the relevant information for the Custom Slave instruction set. The designation "c" is used to represent an operand which can be a 32-bit ("D") or 64-bit ("Q") quantity in any format; the size is determined by the suffix on the mnemonic. Similarly, an "i" indicates an integer size (Byte, Word, Double Word) selected by the corresponding mnemonic suffix.

Any operand indicated as being of type "c" will not cause a transfer if the register addressing mode is specified. It is assumed in this case that the slave processor is already holding the operand internally.

For the instruction encodings, see Appendix A.

3.0 Functional Description (Continued)

TABLE 3-6
Custom Slave Instruction Protocols.

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
CCAL0c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL1c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL2c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL3c	read.c	rmw.c	c	c	c to Op. 2	none
CMOV0c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV1c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV2c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV3c	read.c	write.c	c	N/A	c to Op.2	none
CCMP0c	read.c	read.c	c	c	N/A	N,Z,L
CCMP1c	read.c	read.c	c	c	N/A	N,Z,L
CCV0ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV1ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV2ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV3ic	read.i	write.c	i	N/A	c to Op. 2	none
CCV4DQ	read.D	write.Q	D	N/A	Q to Op. 2	none
CCV5QD	read.Q	write.D	Q	N/A	D to Op. 2	none
LCSR	read.D	N/A	D	N/A	N/A	none
SCSR	N/A	write.D	N/A	N/A	D to OP. 2	none
CATST0*	addr	N/A	D	N/A	N/A	F
CATST1*	addr	N/A	D	N/A	N/A	F
LCR*	read.D	N/A	D	N/A	N/A	none
SCR*	write.D	N/A	N/A	N/A	D to Op.1	none

Note:
D = Double Word
i = Integer size (B,W,D) specified in mnemonic.
c = Custom size (D:32 bits or Q:64 bits) specified in mnemonic.
* = Privileged instruction: will trap if CPU is in User Mode.
N/A = Not Applicable to this instruction.

4.0 Device Specifications

4.1 NS32332 PIN DESCRIPTIONS

The following is a brief description of all NS32332 pins. The descriptions reference portions of the Functional Description, Section 3.

Unless otherwise indicated, reserved pins should be left open.

4.1.1 Supplies

Logic Power ($V_{CCL1, 2}$): +5V positive supply.

Buffers Power ($V_{CCB1, 2, 3, 4, 5}$): +5V positive supply.

Logic Ground (GNDL1, GNDL2): Ground reference for on-chip logic.

Buffer Grounds (GNDB1, GNDB2, GNDB3, GNDB4, GNDB5, GNDB6): Ground references for on-chip drivers.

Back Bias Generator (BBG): Output of on-chip substrate voltage generator.

4.1.2 Input Signals

Clocks (PHI1, PHI2): Two-phase clocking signals.

Ready (RDY): Active high. While RDY is not active, the CPU adds wait cycles to the current bus cycle. Not applicable for slave cycles.

Hold Request ($\overline{HOLD}$): Active low. Causes the CPU to release the bus for DMA or multiprocessing purposes.

Note: If the $\overline{HOLD}$ signal is generated asynchronously, it's set up and hold times may be violated. In this case it is recommended to synchronize it with CTTL to minimize the possibility of metastable states.

The CPU provides only one synchronization stage to minimize the $\overline{HLDA}$ latency. This is to avoid speed degradations in cases of heavy $\overline{HOLD}$ activity (i.e. DMA controller cycles interleaved with CPU cycles.)

Interrupt ($\overline{INT}$): Active low. Maskable Interrupt request.

Non-Maskable Interrupt ($\overline{NMI}$): Active low. Non-Maskable Interrupt request.

Reset/Abort ($\overline{RST}/\overline{ABT}$): Active low. If held active for one clock cycle and released, this pin causes an ABORT. If held longer, it is interpreted as RESET.

Bus Error ($\overline{BER}$): Active low. When active, indicates that an error occurred during a bus cycle. It is treated by the CPU as the highest priority exception after RESET. Not applicable for slave cycles.

Bus Retry ($\overline{BRT}$): Active low. When active, the CPU will re-execute the last bus cycle. Not applicable for slave cycles.

Bus Width (BW1, BW0): Define the bus width (8, 16, 32) in every bus cycle. 01–8 bits, 10–16 bits, 11–32 bits. 00 is a reserved combination. Not applicable for slave cycles.

Burst In ($\overline{BIN}$): Active low. When active, the CPU may perform burst cycles.

Float ($\overline{FLT}$): Active low. Float command input. In non-memory managed systems, this pin should be tied to V_{CC} through a 10 kΩ resistor.

Data Timing/Slave Done ($\overline{DT}/\overline{SDONE}$): Active low. Used by a 32-bit slave processor to acknowledge the completion of an instruction and/or indicate that the slave status should be read (Section 3.9.2). Sampled during reset to select the data timing during write cycles (Section 3.3).

4.1.3 Output Signals

Address Strobe ($\overline{ADS}$): Active low. Controls address latches, indicates the start of a bus cycle.

Data Direction In ($\overline{DDIN}$): Active low. Indicates the directions of data transfers.

Byte Enables ($\overline{BE0}$–$\overline{BE3}$): Active low. Enable the access of bytes 0–3 in a 32 bit system.

Status (ST0–ST3): Bus cycle status code, ST0 least significant. Encodings are:

0000 — Idle: CPU Inactive on Bus.
0001 — Idle: WAIT Instruction.
0010 — (Reserved).
0011 — Idle: Waiting for Slave.
0100 — Interrupt Acknowledge, Master.
0101 — Interrupt Acknowledge, Cascaded.
0110 — End of Interrupt, Master.
0111 — End of Interrupt, Cascaded.
1000 — Sequential Instruction Fetch.
1001 — Non-Sequential Instruction Fetch.
1010 — Data Transfer.
1011 — Read Read-Modify-Write Operand.
1100 — Read for Effective Address.
1101 — Transfer Slave Operand.
1110 — Read Slave Status Word.
1111 — Broadcast Slave ID.

Status Strobe ($\overline{STS}$): Active low. Indicates that a new status (ST0–ST3) is valid. Not applicable for slave cycles.

Multiple Cycle/Exception Status ($\overline{MC}/\overline{EXS}$): Active low. This signal is activated during the access of the first part of an operand that crosses a double word address boundary.

It is also activated in conjunction with status codes 1001 and 0000 during Abort Acknowledge and when a fatal bus error occurs.

Note: $\overline{MC}/\overline{EXS}$ indicates a fatal bus error only when it has been active for more than one clock cycle.

Hold Acknowledge ($\overline{HLDA}$): Active low. Activated by the CPU in response to HOLD input. Indicates that the CPU has released the bus.

User/Supervisor (U/$\overline{S}$): User or Supervisor Mode status.

Interlocked Operation ($\overline{ILO}$): Active low. Indicates that an interlocked cycle is being performed.

Program Flow Status ($\overline{PFS}$): Active low. A pulse that indicates the beginning of an instruction execution.

Burst Out ($\overline{BOUT}$): Active low. When active, indicates that the CPU will perform burst cycles.

4.1.4 Input/Output Signals

Address/Data 0–31 (AD0–AD31): Multiplexed address and data lines.

Slave Processor Control ($\overline{SPC}$): Active low. Used by the CPU as a data strobe output for slave processor transfers. Used by a 16-bit slave processor to acknowledge the completion of an instruction.

4.0 Device Specifications (Continued)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

4.2 ABSOLUTE MAXIMUM RATINGS

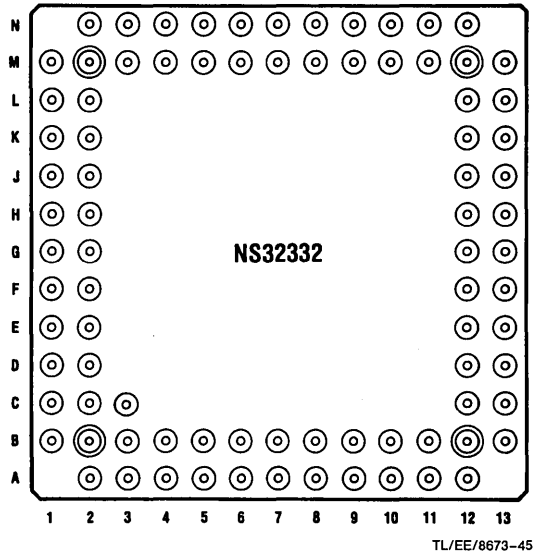
Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	3 Watt

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS $T_A = 0°$ to $+70°C$, $V_{CC} = 5V \pm 5\%$, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		$V_{CC}+0.5$	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{CH}	High Level Clock Voltage	PHI1, PHI2 pins only	$V_{CC}-0.5$		$V_{CC}+0.5$	V
V_{CL}	Low Level Clock Voltage	PHI1, PHI2 pins only	−0.5		0.3	V
V_{CRT}	Clock Input Ringing Tolerance	PHI1, PHI2 pins only	−0.5		0.5	V
V_{OH}	High Level Output Voltage	$I_{OH} = -400\ \mu A$	2.4			V
V_{OL}	Low Level Output Voltage	$I_{OL} = 2$ mA			0.45	V
I_{ILS}	$\overline{SPC}$ and $\overline{DT}/SDONE$ Input Current (low)	$V_{IN} = 0.4V$, $\overline{SPC}$ in input mode	0.05		1.0	mA
I_I	Input Load Current	$0 \le V_{IN} \le V_{CC}$, Input Pins except PHI1, PHI2, $\overline{DT}/SDONE$	−20		20	μA
I_L	Leakage Current (Output and I/O pins in TRI-STATE/Input Mode)	$0.4 \le V_{IN} \le V_{CC}$	−80		80	μA
I_{CC}	Active Supply Current	$I_{OUT} = 0$, $T_A = 25°C$		450	600	mA

Connection Diagram*



TL/EE/8673-45

Bottom View

NS32332 Pinout Descriptions
84 Pin Grid Array

Desc	Pin	Desc	Pin	Desc	Pin
GNDB1	B1	AD29	N6	$\overline{BOUT}$	E12
AD6	B2	AD30	M6	$\overline{SPC}$	D13
AD7	C1	AD31	N7	$\overline{MC}/EXS$	D12
AD8	C2	VCCL1	M7	VCCB5	C13
AD9	D1	VCCL2	N8	$\overline{ADS}$	C12
AD10	D2	$\overline{INT}$	M8	GNDB6	B13
AD11	E1	$\overline{NMI}$	N9	$\overline{DDIN}$	A12
GNDB2	E2	RESERVED	M9	$\overline{BE0}$	B12
AD12	F1	RESERVED	N10	$\overline{BE1}$	A11
AD13	F2	RESERVED	M10	$\overline{BE2}$	B11
AD14	G1	RESERVED	N11	$\overline{BE3}$	A10
AD15	G2	$\overline{ILO}$	M11	$\overline{HLDA}$	B10
VCCB2	H1	VCCB4	N12	$\overline{HOLD}$	A9
AD16	H2	ST3	M13	RDY	B9
AD17	J1	ST2	M12	$\overline{DT}/SDONE$	A8
AD18	J2	ST1	L13	PHI 2	B8
AD19	K1	ST0	L12	PHI 1	A7
GNDB3	K2	$\overline{STS}$	K13	BBG	B7
AD20	L1	GNDB5	K12	GNDL2	A6
AD21	L2	$\overline{PFS}$	J13	GNDL1	B6
AD22	M1	U/$\overline{S}$	J12	VCCB1	A5
AD23	N2	BW1	H13	AD0	B5
VCCB3	M2	BW0	H12	AD1	A4
AD24	N3	$\overline{BIN}$	G13	AD2	B4
AD25	M3	$\overline{FLT}$	G12	AD3	A3
AD26	N4	$\overline{RST}/\overline{ABT}$	F13	AD4	B3
AD27	M4	$\overline{BRT}$	F12	AD5	A2
GNDB4	N5	$\overline{BER}$	E13	POSITION PIN	C3
AD28	M5				

Order Number NS32332U-10 or NS32332U-15
See NS Package Number U84C

FIGURE 4-1. Pin Grid Array Package

*AMP sockets are recommended for use with NS32332 CPU. AMP sockets are manufactured by AMP INCORPORATED, Harrisburg PA.

4.0 Device Specifications (Continued)

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 2.0V on the rising or falling edges of the clock phases PHI1 and PHI2 and 0.8V or 2.0V on all other signals as illustrated below, unless specifically stated otherwise.

ABBREVIATIONS:

L.E. — leading edge
T.E. — trailing edge
R.E. — rising edge
F.E. — falling edge

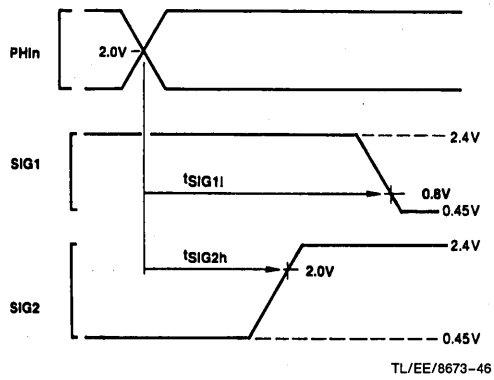


FIGURE 4-2. Timing Specification Standard (Signal Valid After Clock Edge)

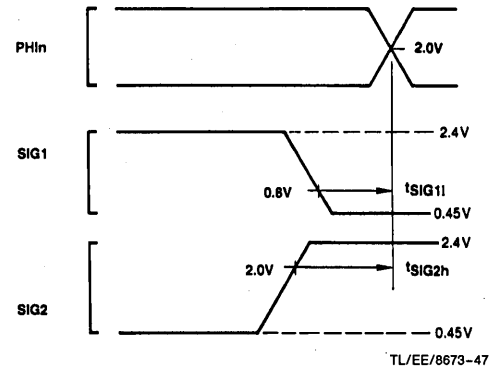


FIGURE 4-3. Timing Specification Standard (Signal Valid Before Clock Edge)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32332-10, NS32332-15

Maximum times assume capacitive loading of 100 pF.

AD0–31, $\overline{ADS}$ and $\overline{BOUT}$ timings are defined with a capacitive loading of 75 pF.

Symbol	Figure	Description	Reference/ Conditions	NS32332-10		NS32332-15		Units
				Min	Max	Min	Max	
t_{ALv}	4-5	Address bits 0–31 valid	after R.E., PHI1 T1		30		20	ns
t_{ALh}	4-5	Address bits 0–31 hold	after R.E., PHI1 T2/Tmmu	10		6		ns
t_{Dv}	4-5	Data valid (write cycle)	after R.E., PHI1 T3 or T2		50		38	ns
t_{Dh}	4-5	Data hold (write cycle)	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{ALADSs}	4-4	Address bits 0–31 setup	before $\overline{ADS}$ T.E.	25		20		ns
t_{ALADSh}	4-18	Address bits 0–31 hold	after $\overline{ADS}$ T.E.	10		10		ns
t_{ALf}	4-4	Address bits 0–31 floating (no MMU)	after R.E., PHI1 T2/Tmmu		25		24	ns
t_{ALMf}	4-18	Address bits 0–31 floating (by $\overline{FLT}$ line)	after R.E., PHI1 Tf		40		40	ns
t_{STSa}	4-3,4-5	$\overline{STS}$ signal active (low)	after R.E., PHI1 T4 of previous bus cycle or Ti		35		25	ns
t_{STSia}	4-3,4-5	$\overline{STS}$ signal inactive	after R.E., PHI2 T4 of previous bus cycle or Ti		45		30	ns
t_{STSw}	4-3	$\overline{STS}$ pulse width	at 0.8V (both edges)	35		24		ns
t_{BErv}	4-4,4-6	$\overline{BEn}$ signals valid (Operand Read Cycles Only)	after R.E., PHI2, T4 or Ti		140		95	ns
t_{BEv}	4-5,4-6	$\overline{BEn}$ signals valid	after R.E., PHI2, T4 or Ti		85		58	ns
t_{BEh}	4-4	$\overline{BEn}$ signals hold	after R.E., PHI2, T4	0		0		ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32332-10, NS32332-15 (Continued)

Symbol	Figure	Description	Reference/ Conditions	NS32332-10		NS32332-15		Units
				Min	Max	Min	Max	
t_{STv}	4-5	Status (ST0–ST3) valid	after R.E., PHI1 T4 (before T1, see note)		50		35	ns
t_{STSTSs}	4-5	Status Signals Setup	Before $\overline{STS}$ T.E.	10		6		ns
t_{STh}	4-5	Status (ST0–ST3) hold	after R.E., PHI1 T4 (after T1)	0		0		ns
t_{DDINv}	4-4	$\overline{DDIN}$ signal valid	after R.E., PHI1 T1		35		25	ns
t_{DDINh}	4-4	$\overline{DDIN}$ signal hold	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{ADSa}	4-5	$\overline{ADS}$ signal active (low)	after R.E., PHI1 T1		25		17	ns
t_{ADSia}	4-5	$\overline{ADS}$ signal inactive	after R.E., PHI2 T1		45		29	ns
t_{ADSw}	4-5	$\overline{ADS}$ pulse width	at 0.8V (both edges)	35		24		ns
t_{MCa}	4-4,4-5	$\overline{MC}$ signal active (low)	after R.E., PHI1 T1		70		50	ns
t_{MCia}	4-4,4-5	$\overline{MC}$ signal inactive	after R.E., PHI1 T1 or T3 (burst)		70		50	ns
t_{ALf}	4-15	AD0–AD31 floating (caused by $\overline{HOLD}$)	after R.E., PHI1 T1		25		24	ns
t_{ADSf}	4-15, 4-17	$\overline{ADS}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{BEf}	4-15, 4-17	$\overline{BEn}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{DDINf}	4-15, 4-17	$\overline{DDIN}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		45	ns
t_{HLDAa}	4-15, 4-16	$\overline{HLDA}$ signal active (low)	after R.E., PHI1 T4		60		45	ns
t_{HLDAia}	4-18	$\overline{HLDA}$ signal inactive	after R.E., PHI1 Ti		60		45	ns
t_{ADSr}	4-18	$\overline{ADS}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{BEr}	4-18	$\overline{BEn}$ signals return from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{DDINr}	4-18	$\overline{DDIN}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{DDINf}	4-19	$\overline{DDIN}$ signal floating (caused by $\overline{FLT}$)	after $\overline{FLT}$ F.E.		50		45	ns
t_{DDINr}	4-20	$\overline{DDIN}$ signal returns from floating (caused by $\overline{FLT}$)	after $\overline{FLT}$ R.E.		40		28	ns
t_{SPCa}	4-21	$\overline{SPC}$ output active (low)	after R.E., PHI1 T1		30		21	ns
t_{SPCia}	4-21	$\overline{SPC}$ output inactive	after R.E., PHI1 T4	2	35	2	26	ns
t_{SPCnf}	4-24	$\overline{SPC}$ output nonforcing	after R.E., PHI2 T4		10		8	ns
t_{Dv}	4-21	Data valid (slave processor write)	after R.E., PHI1 T1		50		38	ns
t_{Dh}	4-21	Data hold (slave processor write)	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{PFSw}	4-26	$\overline{PFS}$ pulse width	at 0.8V (both edges)	70		45		ns
t_{PFSa}	4-26	$\overline{PFS}$ pulse active (low)	after R.E., PHI2		50		38	ns
t_{PFSia}	4-26	$\overline{PFS}$ pulse inactive	after R.E., PHI2		50		38	ns
t_{USv}	4-33	U/$\overline{S}$ signal valid	after R.E., PHI1 T4		48		35	ns
t_{USh}	4-33	U/$\overline{S}$ signal hold	after R.E., PHI1 T4	10		6		ns
t_{NSPF}	4-28	Nonsequential fetch to next $\overline{PFS}$ clock cycle	after R.E., PHI1 T1	4		4		t_{Cp}

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32332-10, NS32332-15 (Continued)

Symbol	Figure	Description	Reference/ Conditions	NS32332-10		NS32332-15		Units
				Min	Max	Min	Max	
t_{PFNS}	4-27	$\overline{PFS}$ clock cycle to next non-sequential fetch	before R.E., PHI1 T1	4		4		t_{Cp}
t_{STSf}	4-15, 4-16	$\overline{STS}$ floating (HOLD)	after R.E., PHI1 Ti		55		44	ns
t_{STSr}	4-18	$\overline{STS}$ not floating (HOLD)	after R.E., PHI1 Ti, T4		55		40	ns
t_{BOUTa}	4-6, 4-10	$\overline{BOUT}$ output active	after R.E., PHI2 Tmmu		100		66	ns
t_{BOUTia}	4-6, 4-10	$\overline{BOUT}$ output inactive	after R.E., PHI2 T3 or T4		75		40	ns
t_{ILOa}	4-14	$\overline{ILO}$ signal active	after R.E., PHI1 T4		50		38	ns
t_{ILOia}	4-14	$\overline{ILO}$ signal inactive	after R.E., PHI1 Ti		50		38	ns

Note: Every memory cycle starts with T4, during which Cycle Status is applied. If the CPU was idling, the sequence will be: ". . . Ti, T4, T1 . . .". If the CPU was not idling, the sequence will be: ". . . T4, T1 . . .".

4.4.2.2 Input Signal Requirements: NS32332-10, NS32332-15

Symbol	Figure	Description	Reference/ Conditions	NS32332-10		NS32332-15		Units
				Min	Max	Min	Max	
t_{PWR}	4-31	Power stable to $\overline{RST}$ R.E.	after V_{CC} reaches 4.5V	50		33		μs
t_{DIs}	4-4	Data in setup (read cycle)	before F.E., PHI2 T3	12		10		ns
t_{DIh}	4-4	Data in hold (read cycle)	after R.E., PHI1 T4	3		3		ns
t_{HLDa}	4-15 4-16,	$\overline{HOLD}$ active setup time	before F.E., PHI2 T2/Tmmu or T3 or Ti	25		17		ns
t_{HLDia}	4-18	$\overline{HOLD}$ inactive setup time	before F.E., PHI2 Ti	25		17		ns
t_{HLDh}	4-15, 4-17, 4-18	$\overline{HOLD}$ hold time	after R.E., PHI1 Ti or T3	0		0		ns
t_{FLTa}	4-19	$\overline{FLT}$ active (low) setup time	before F.E., PHI2 Tmmu	25		17		ns
t_{FLTia}	4-20	$\overline{FLT}$ inactive setup time	before F.E., PHI2 T3	25		17		ns
t_{RDYs}	4-4, 4-5, 4-6	RDY setup time	before F.E., PHI1 T3	20		12		ns
t_{RDYh}	4-4, 4-5, 4-6	RDY hold time	after R.E., PHI2 T3	4		3		ns
t_{ABTs}	4-29	$\overline{ABT}$ setup time ($\overline{FLT}$ inactive)	before F.E., PHI2 T2/Tmmu	20		13		ns
t_{ABTs}	4-30	$\overline{ABT}$ setup time ($\overline{FLT}$ active)	before F.E., PHI2 Tf	20		13		ns
t_{ABTh}	4-29, 4-30	$\overline{ABT}$ hold time	after R.E., PHI1 T3	0		0		ns
t_{RSTs}	4-31, 4-32	$\overline{RST}$ setup time	before F.E., PHI1	20		13		ns
t_{RSTw}	4-31, 4-32	$\overline{RST}$ pulse width	at 0.8V (both edges)	64		64		t_{Cp}
t_{INTs}	4-34	$\overline{INT}$ setup time	before F.E., PHI2	20		13		ns
t_{NMIw}	4-35	$\overline{NMI}$ pulse width	at 0.8V (both edges)	40		27		ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32332-10, NS32332-15 (Continued)

Symbol	Figure	Description	Reference/ Conditions	NS32332-10		NS32332-15		Units
				Min	Max	Min	Max	
t_{DIs}	4-24	Data setup (slave read cycle)	before F.E., PHI2 T1	12		10		ns
t_{DIh}	4-24	Data hold (slave read cycle)	after R.E., PHI1 T4	3		3		ns
t_{DTs}	4-31	$\overline{DT}$ setup time	before F.E., PHI1	0		0		ns
t_{DTh}	4-31	$\overline{DT}$ hold time	after R.E., PHI1	0		0		ns
t_{SPCd}	4-24	$\overline{SPC}$ pulse delay from slave	after R.E., PHI2 T4	10		8		ns
t_{SPCs}	4-24	$\overline{SPC}$ setup time	before F.E., PHI1	25		15		ns
t_{SPCw}	4-24	$\overline{SPC}$ pulse width	at 0.8V (both edges)	20	100	13	66	ns
t_{SDNd}	4-23	$\overline{SDONE}$ pulse delay from slave	after R.E., PHI2 T4	10		8		ns
t_{SDNs}	4-23	$\overline{SDONE}$ setup time	before F.E., PHI1	25		15		ns
t_{SDNw}	4-23	$\overline{SDONE}$ pulse width	at 0.8V (both edges)	20	100	13	66	ns
t_{SDNSTw}	4-23	$\overline{SDONE}$ pulse width (to force CPU to read slave status)	at 0.8V (both edges)	175	275	115	200	ns
t_{BWs}	4-4, 4-5 4-6	$\overline{BW}$ 0–1 setup time	before F.E., PHI1 T3	25		13		ns
t_{BWh}	4-6	$\overline{BW}$0-1 hold time	after R.E., PHI1 T3 of Next Memory Access Cycle	0		0		ns
t_{BINs}	4-6, 4-7	$\overline{BIN}$ setup time (for each cycle of the burst)	before F.E., PHI1 T3	25		12		ns
t_{BINh}	4-6, 4-7	$\overline{BIN}$ hold time	after R.E., PHI1 T4	0		0		ns
t_{BERs}	4-12, 4-13	$\overline{BER}$ setup time	before F.E., PHI1 T4	25		14		ns
t_{BERh}	4-12, 4-13	$\overline{BER}$ hold time (see note)	after R.E., PHI1 Ti	0		0		ns
t_{BRTs}	4-8, 4-9, 4-10, 4-11	$\overline{BRT}$ setup time	before F.E., PHI1 T3 and T4	25		14		ns
t_{BRTh}	4-8, 4-9, 4-10	$\overline{BRT}$ hold time	after R.E., PHI1 T4 or Ti	0		0		ns

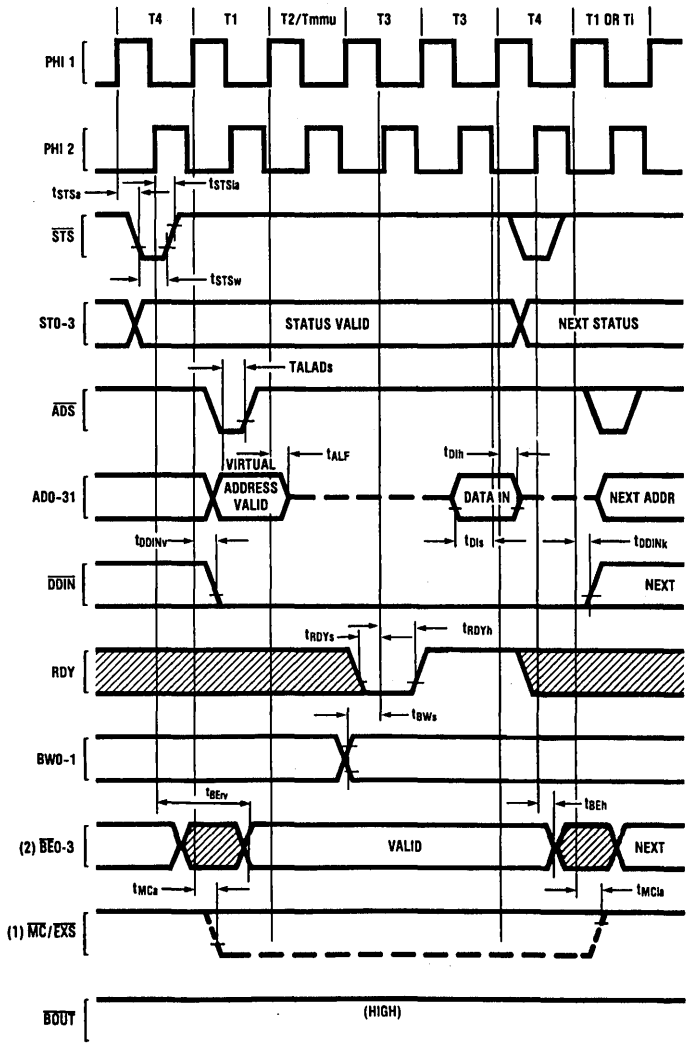
Note: A Ti state follows T4 when $\overline{BER}$ is asserted. $\overline{BER}$ should be deasserted at the latest in the beginning of the cycle following this Ti state.

4.4.2.3 Clocking Requirements: NS32332-10, NS32332-15

Symbol	Figure	Description	Reference/ Conditions	NS32332-10		NS32332-15		Units
				Min	Max	Min	Max	
t_{Cp}	4-25	Clock period	R.E., PHI1, PHI2 to next R.E., PHI1, PHI2	100	250	66	250	ns
$t_{CLw(1,2)}$	4-25	PHI1, PHI2 Pulse Width	At 2.0V on PHI1, PHI2 (Both Edges)	0.5 t_{cp} − 10 ns		0.5 t_{cp} − 6 ns		
$t_{CLh(1,2)}$	4-25	PHI1, PHI2 high time	At V_{CC}−0.9V on PHI1, PHI2 (Both Edges)	0.5 t_{cp} − 15 ns		0.5 t_{cp} − 10 ns		
t_{CLl}	4-25	PHI1, PHI2 low time	At 0.8V on PHI1, PHI2 (Both Edges)	0.5 t_{cp} −5 ns		0.5 t_{cp} −5 ns		
$t_{nOVL(1,2)}$	4-25	Non-overlap time	0.8V on F.E., PHI1, PHI2 to 0.8V on R.E., PHI2, PHI1	−2	2	−2	2	ns
t_{nOVLas}		Non-overlap asymmetry ($t_{nOVL(1)} - t_{nOVL(2)}$)	At 0.8V on PHI1, PHI2	−3	3	−3	3	ns
t_{CLhas}		PHI1, PHI2 asymmetry ($t_{CLh(1)} - t_{CLh(2)}$)	At V_{CC}−0.9V on PHI1, PHI2	−5	5	−3	3	ns

4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams

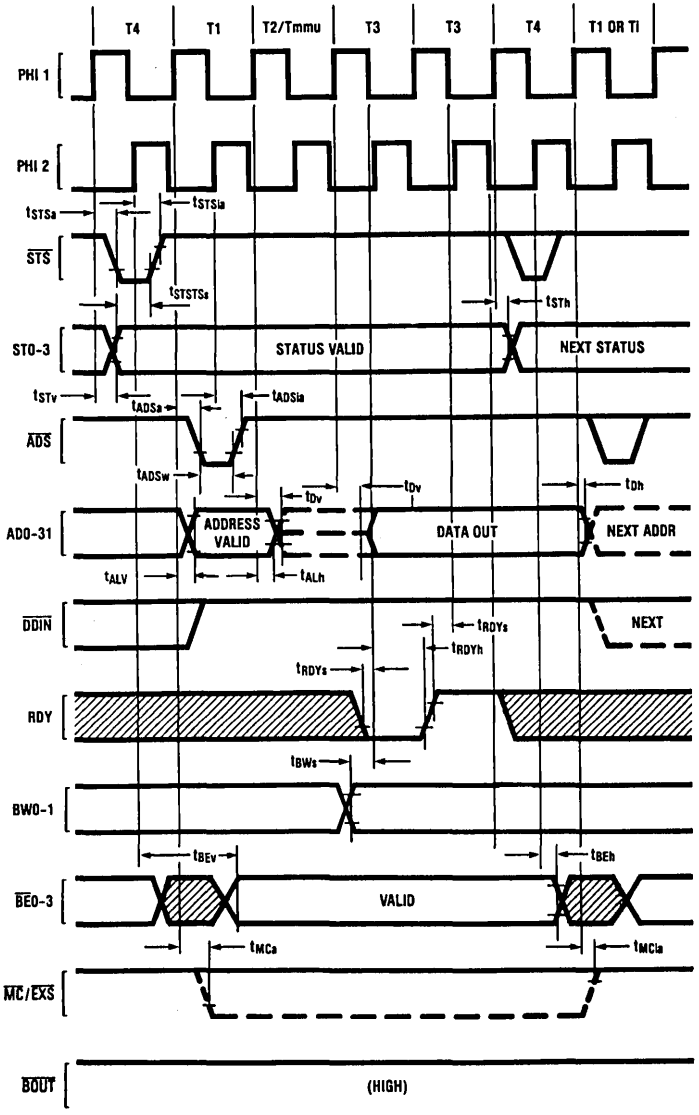


TL/EE/8673-48

Note 1: Asserted (low) when the bus transaction crosses a double-word boundary (address bits A0–1 wrap around during the transaction).
Note 2: $\overline{BE0}$–$\overline{BE3}$ are all active during instruction fetch cycles.

FIGURE 4-4. NS32332 Read Cycle Timing

4.0 Device Specifications (Continued)

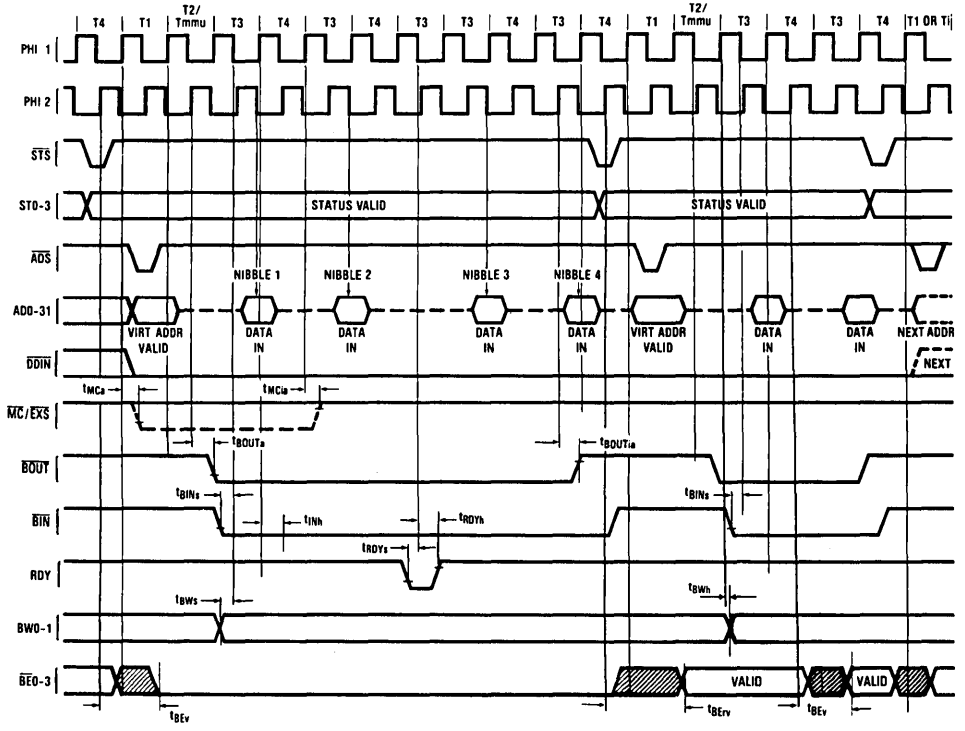


TL/EE/8673–49

Note: If $\overline{DT}/\overline{SDONE}$ is sampled low during reset, the CPU outputs the data during T2/TMMU (see Section 3.3).

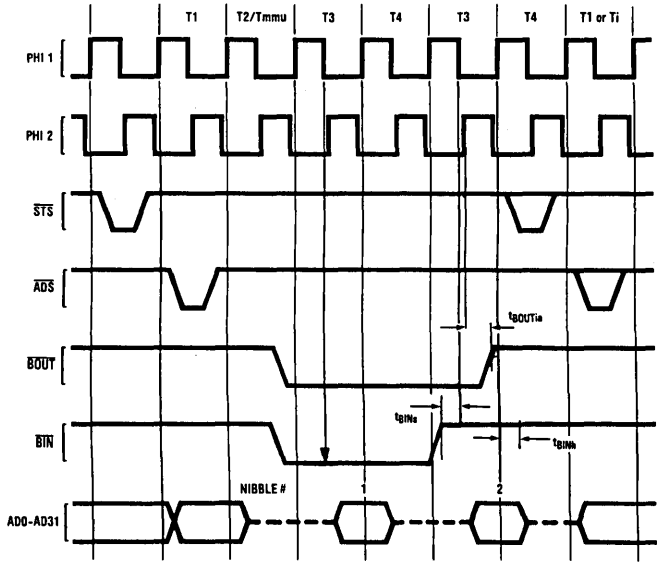
FIGURE 4-5. NS32332 Write Cycle Timing

4.0 Device Specifications (Continued)



TL/EE/8673-50

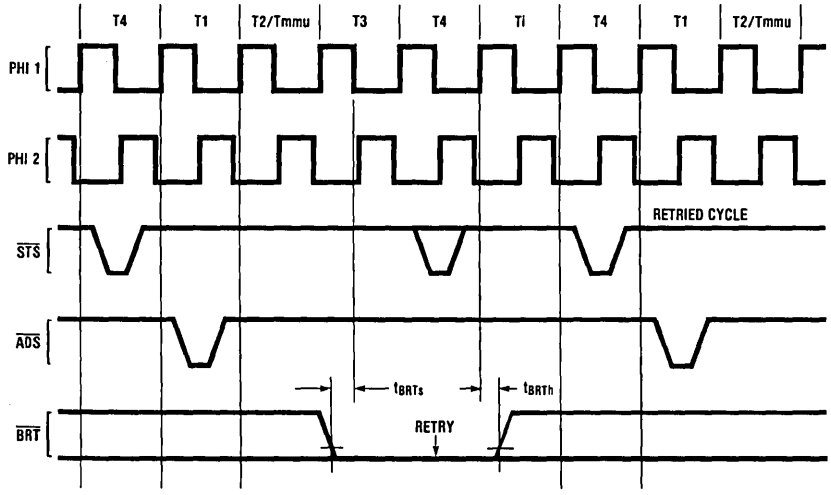
FIGURE 4-6. NS32332 Burst Cycle Timing (Instruction fetches followed by Operand Reads)



TL/EE/8673-94

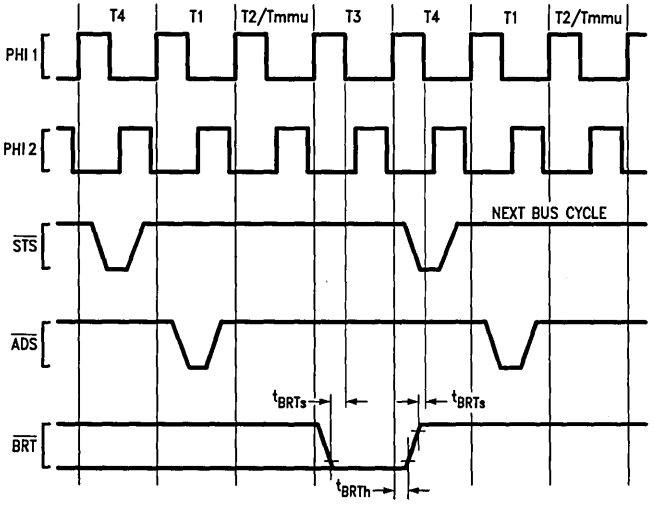
FIGURE 4-7. External Termination of Burst Cycle

4.0 Device Specifications (Continued)



TL/EE/8673-51

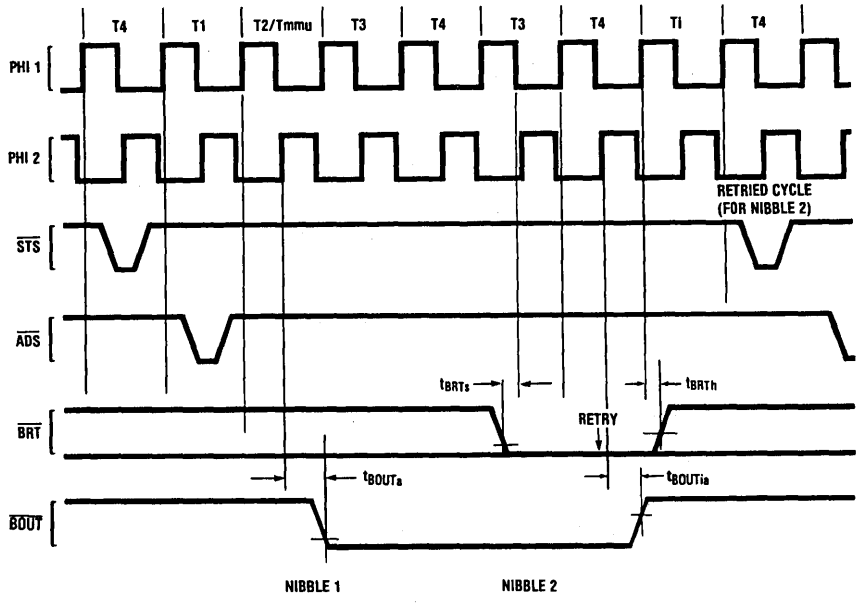
FIGURE 4-8. Bus Retry During Normal Bus Cycle



TL/EE/8673-52

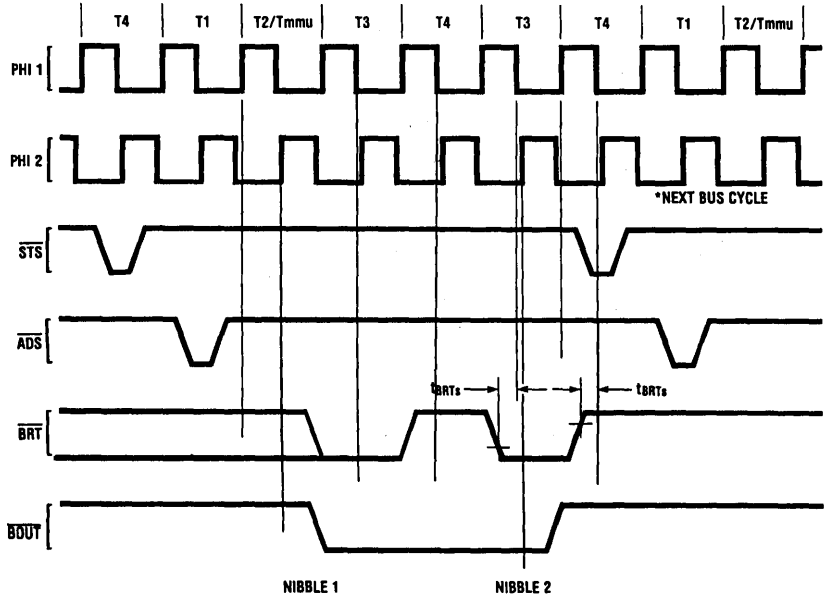
FIGURE 4-9. $\overline{BRT}$ Activated, but no Bus Retry

4.0 Device Specifications (Continued)



TL/EE/8673-53

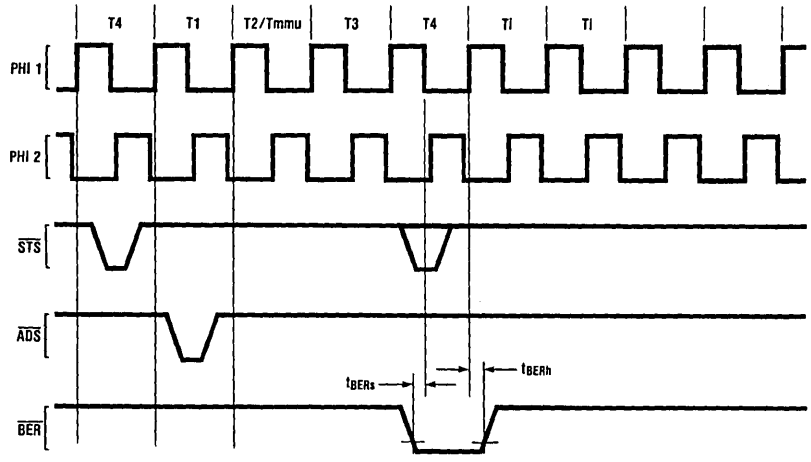
FIGURE 4-10. Bus Retry During Burst Bus Cycle



TL/EE/8673-54

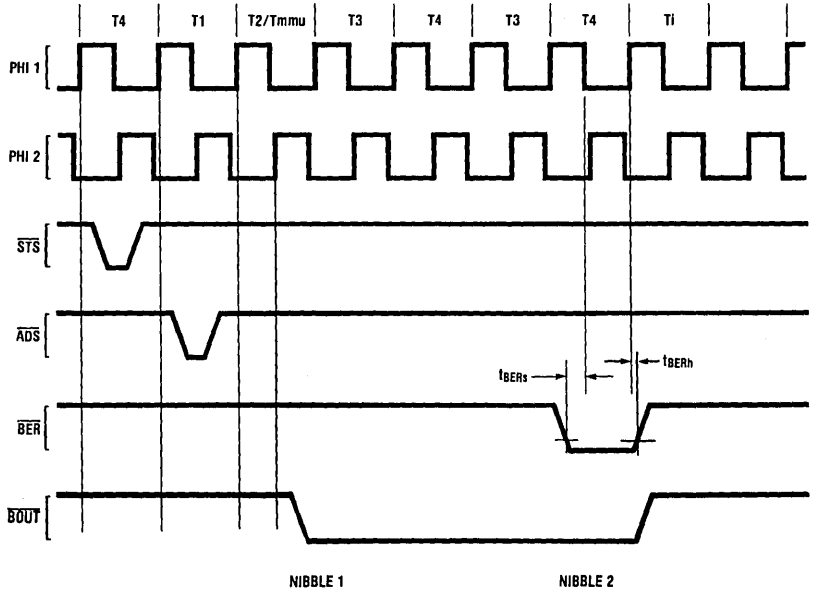
FIGURE 4-11. $\overline{BRT}$ Activated During Burst Bus Cycle, but no Bus Retry

4.0 Device Specifications (Continued)



TL/EE/8673-55

FIGURE 4-12. Bus Error During Normal Bus Cycle

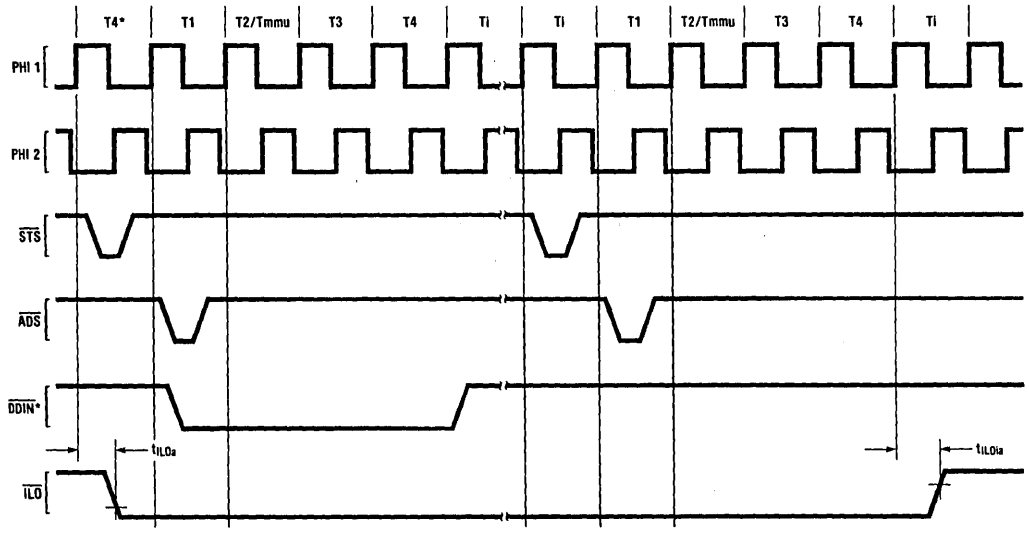


TL/EE/8673-56

FIGURE 4-13. Bus Error During Burst Bus Cycle

2

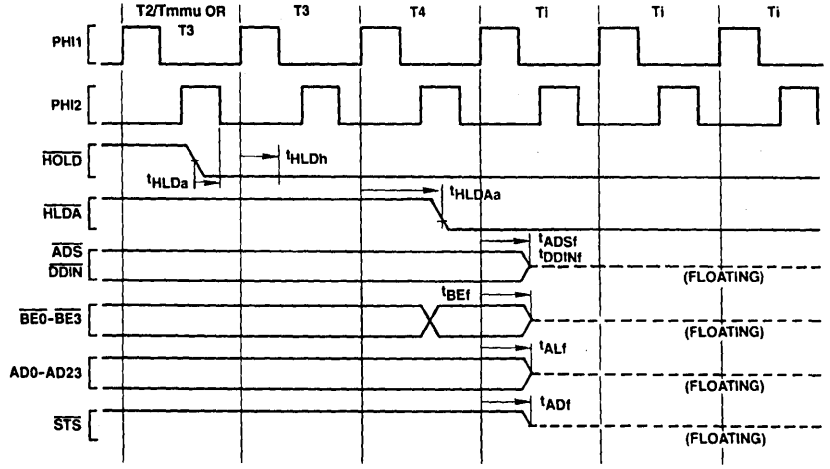
4.0 Device Specifications (Continued)



*End of Dummy Read cycle with the address of the interlocked operand.

TL/EE/8673-57

FIGURE 4-14. Timing of Interlocked Bus Transactions

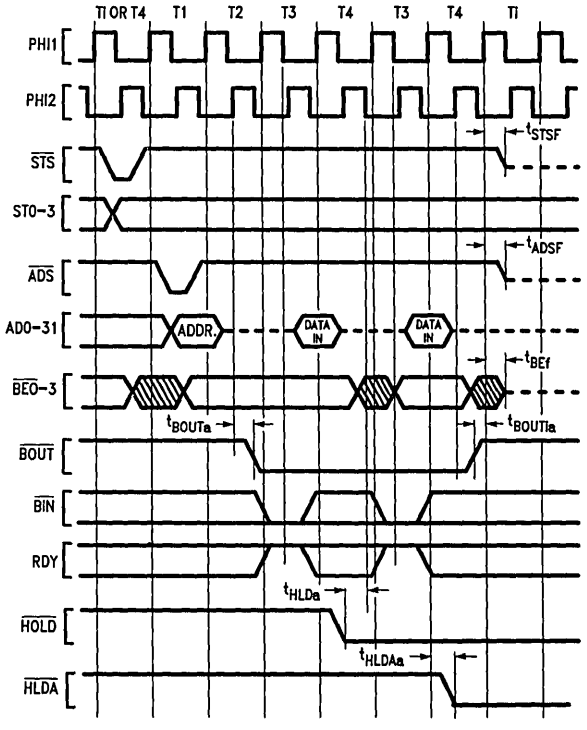


TL/EE/8673-58

FIGURE 4-15. Floating by HOLD Timing (CPU Not Idle Initially)

Note: Whenever the CPU is not idling (not in Ti), the HOLD signal must be active before the falling edge of PHI2 of the clock cycle that appears two clock cycles before T4 (TX1) and stay low until after the rising edge of PHI1 of the clock cycle that precedes T4 (TX2) for the request to be acknowledged.

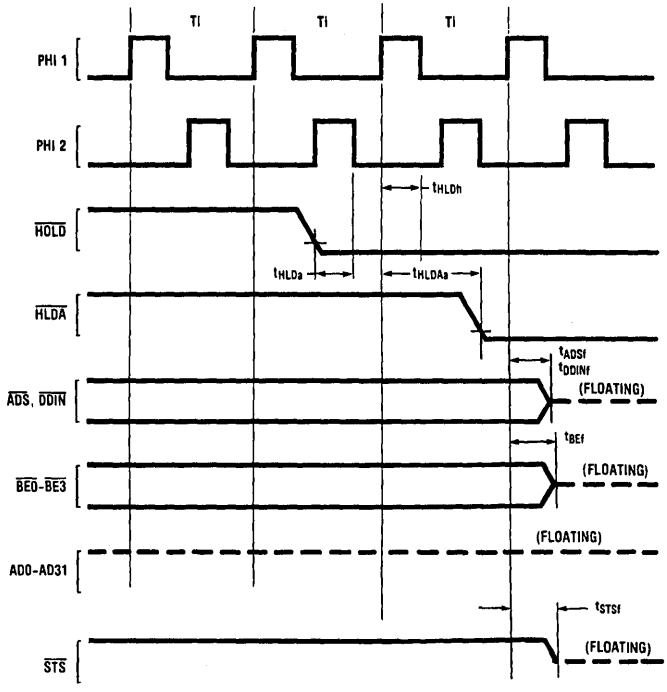
4.0 Device Specifications (Continued)



TL/EE/8673-90

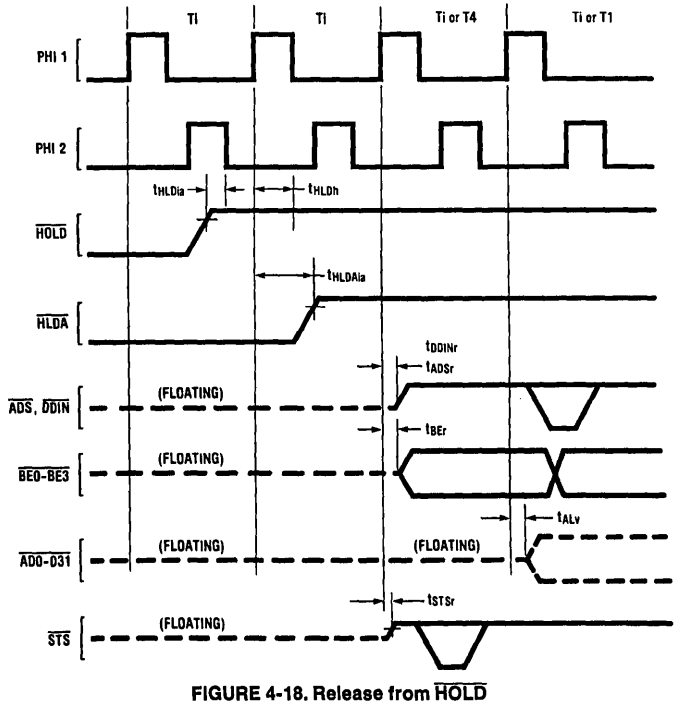
FIGURE 4-16. Floating by $\overline{HOLD}$ Timing (Burst Cycle Ended by $\overline{HOLD}$ Assertion)

4.0 Device Specifications (Continued)



TL/EE/8673-59

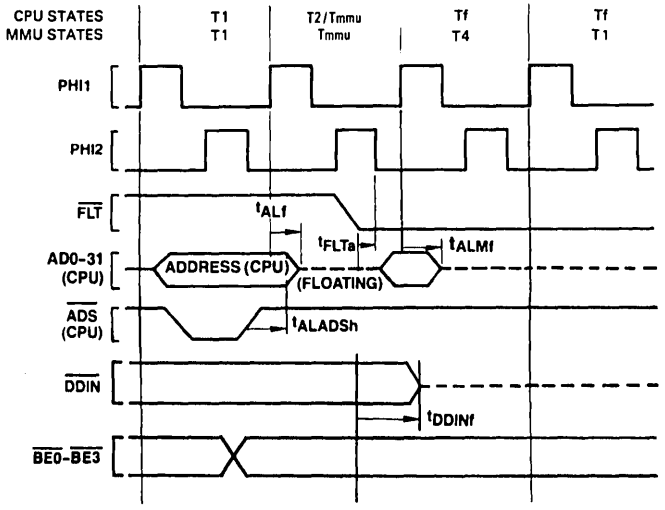
FIGURE 4-17. Floating by $\overline{\text{HOLD}}$ Timing (CPU Initially Idle)



TL/EE/8673-60

FIGURE 4-18. Release from $\overline{\text{HOLD}}$

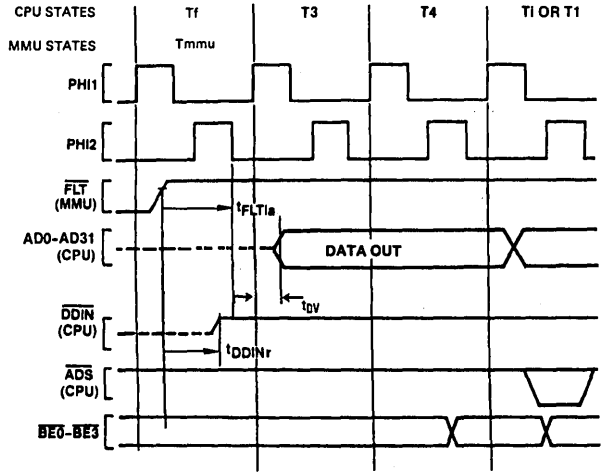
4.0 Device Specifications (Continued)



TL/EE/8673-61

Note: The bus lines AD0–31 are temporarily driven in T2/TMMU and T_f when $\overline{FLT}$ is asserted only if $\overline{DT}/\overline{SDONE}$ is sampled low during reset (see Section 3.3).

FIGURE 4-19. $\overline{FLT}$ Initiated Cycle Timing

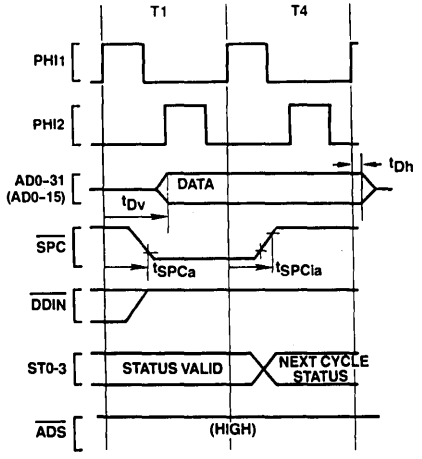


TL/EE/8673-62

FIGURE 4-20. Release from $\overline{FLT}$ Timing (CPU Write Cycle)

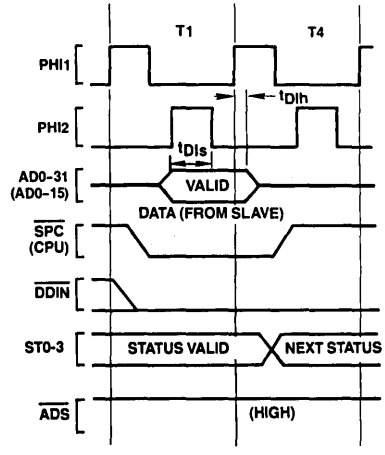
Note: When $\overline{FLT}$ is deasserted the CPU restarts driving $\overline{DDIN}$ before the MMU releases it. This, however, does not cause any conflict, since both CPU and MMU force $\overline{DDIN}$ to the same logic level.

4.0 Device Specifications (Continued)



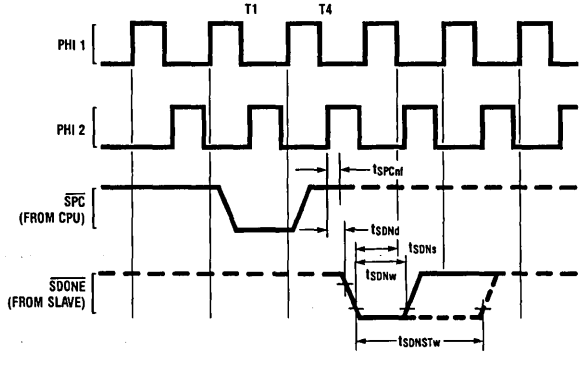
TL/EE/8673-64

FIGURE 4-21. Slave Processor Write Timing



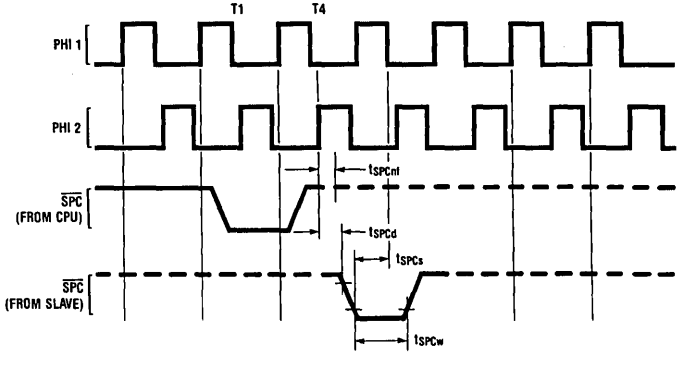
TL/EE/8673-65

FIGURE 4-22. Slave Processor Read Timing



TL/EE/8673-63

FIGURE 4-23. $\overline{DT}/\overline{SDONE}$ Timing (32-Bit Slave Protocol)

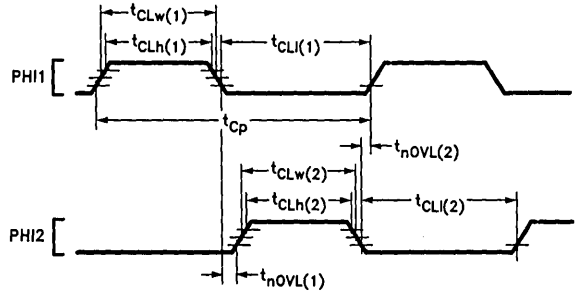


TL/EE/8673-66

FIGURE 4-24. $\overline{SPC}$ Timing (16-Bit Slave Protocol)

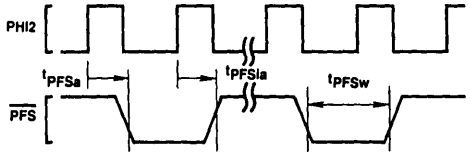
Note: After transferring last operand to a Slave Processor, CPU turns OFF driver and holds $\overline{SPC}$ high with internal 5 kΩ pullup.

4.0 Device Specifications (Continued)



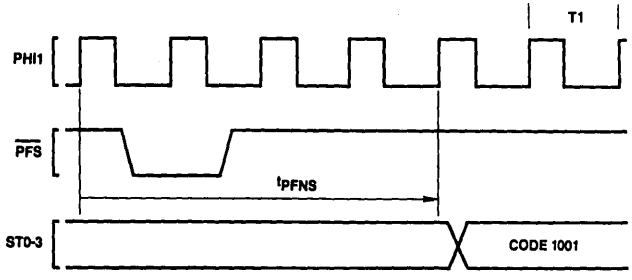
TL/EE/8673-91

FIGURE 4-25. Clock Waveforms



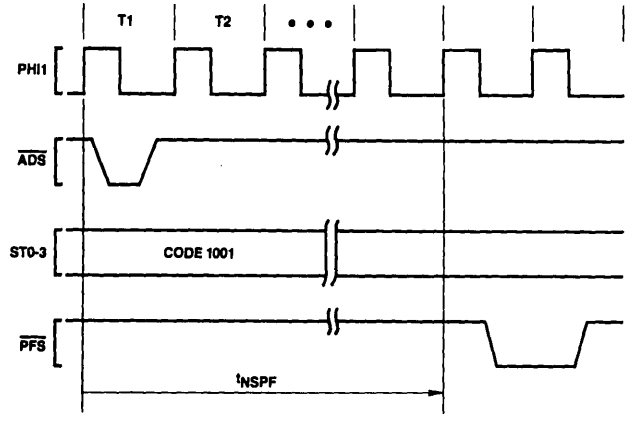
TL/EE/8673-68

FIGURE 4-26. Relationship of $\overline{PFS}$ to Clock Cycles



TL/EE/8673-69

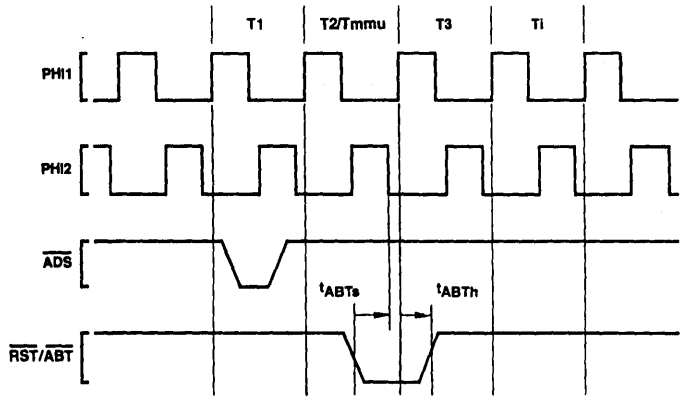
FIGURE 4-27. Guaranteed Delay, $\overline{PFS}$ to Non-Sequential Fetch



TL/EE/8673-70

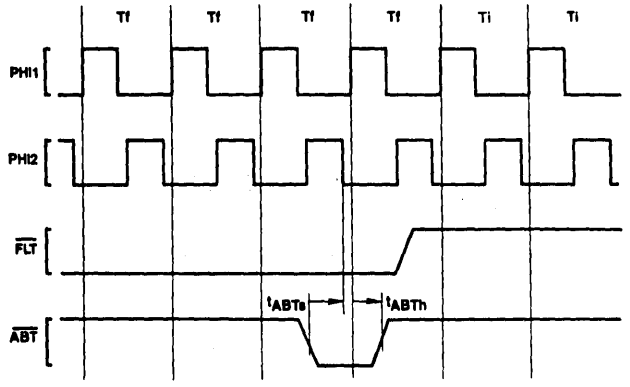
FIGURE 4-28. Guaranteed Delay, Non-Sequential Fetch to $\overline{PFS}$

4.0 Device Specifications (Continued)



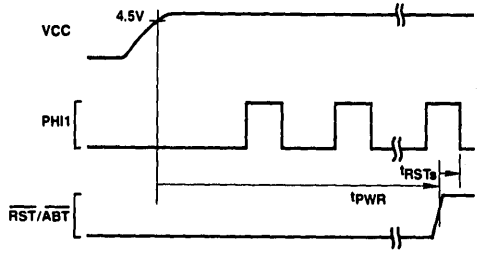
TL/EE/8673-71

FIGURE 4-29. Abort Timing, $\overline{FLT}$ Not Applied



TL/EE/8673-72

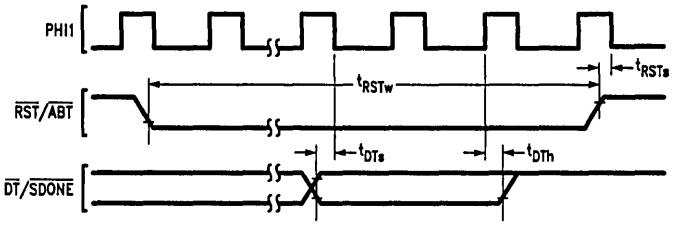
FIGURE 4-30. Abort Timing, $\overline{FLT}$ Applied



TL/EE/8673-73

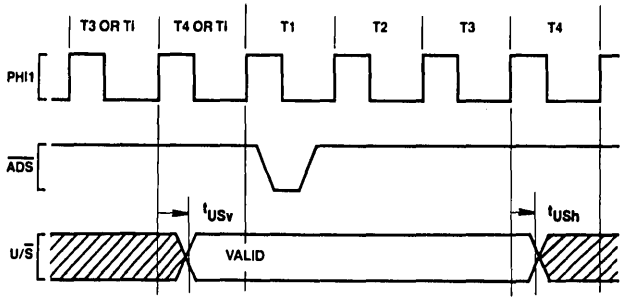
FIGURE 4-31. Power-On Reset

4.0 Device Specifications (Continued)



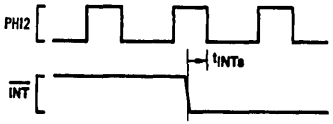
TL/EE/8673-92

FIGURE 4-32. Non-Power-On Reset



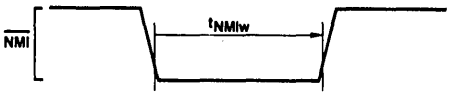
TL/EE/8673-75

FIGURE 4-33. U/S Relationship to Any Bus Cycle — Guaranteed Valid Interval



TL/EE/8673-76

FIGURE 4-34. INT Interrupt Signal Detection



TL/EE/8673-77

FIGURE 4-35. NMI Interrupt Signal Timing

Appendix A: Instruction Formats

NOTATIONS

i= Integer Type Field
- B = 00 (Byte)
- W = 01 (Word)
- D = 11 (Double Word)

f= Floating Point Type Field
- F = 1 (Std. Floating: 32 bits)
- L = 0 (Long Floating: 64 bits)

c= Custom Type Field
- D = 1 (Double Word)
- Q = 0 (Quad Word)

op= Operation Code
Valid encodings shown with each format.

gen, gen 1, gen 2= General Addressing Mode Field
See Sec. 2.2 for encodings.

reg= General Purpose Register Number

cond= Condition Code Field
- 0000 = EQual: Z = 1
- 0001 = Not Equal: Z = 0
- 0010 = Carry Set: C = 1
- 0011 = Carry Clear: C = 0
- 0100 = HIgher: L = 1
- 0101 = Lower or Same: L = 0
- 0110 = Greater Than: N = 1
- 0111 = Less or Equal: N = 0
- 1000 = Flag Set: F = 1
- 1001 = Flag Clear: F = 0
- 1010 = LOwer: L = 0 and Z = 0
- 1011 = Higher or Same: L = 1 or Z = 1
- 1100 = Less Than: N = 0 and Z = 0
- 1101 = Greater or Equal: N = 1 or Z = 1
- 1110 = (Unconditionally True)
- 1111 = (Unconditionally False)

short= Short Immediate value. May contain
- quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
- cond: Condition Code (above), in Scond.
- areg: CPU Dedicated Register, in LPR, SPR.
  - 0000 = US
  - 0001 – 0111 = (Reserved)
  - 1000 = FP
  - 1001 = SP
  - 1010 = SB
  - 1011 = (Reserved)
  - 1100 = (Reserved)
  - 1101 = PSR
  - 1110 = INTBASE
  - 1111 = MOD
- Options: in String Instructions

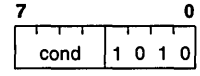
U/W	B	T

  - T = Translated
  - B = Backward
  - U/W = 00: None
    - 01: While Match
    - 11: Until Match

Configuration bits in SETCFG Instruction:

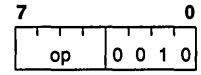
P	FC	FM	FF	C	M	F	I

mreg: NS32382 Register number, in LMR, SMR.
- 0000 = BAR
- 0001 = (Reserved)
- 0010 = BMR
- 0011 = BDR
- 0100 = (Reserved)
- 0101 = (Reserved)
- 0110 = BEAR
- 0111 = (Reserved)
- 1000 = (Reserved)
- 1001 = MCR
- 1010 = MSR
- 1011 = TEAR
- 1100 = PTB0
- 1101 = PTB1
- 1110 = IVAR0
- 1111 = IVAR1



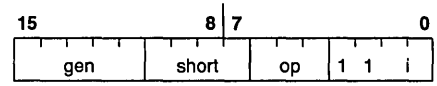
Format 0

Bcond (BR)



Format 1

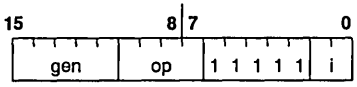
BSR	-0000	ENTER	-1000
RET	-0001	EXIT	-1001
CXP	-0010	NOP	-1010
RXP	-0011	WAIT	-1011
RETT	-0100	DIA	-1100
RETI	-0101	FLAG	-1101
SAVE	-0110	SVC	-1110
RESTORE	-0111	BPT	-1111



Format 2

ADDQ	-000	ACB	-100
CMPQ	-001	MOVQ	-101
SPR	-010	LPR	-110
Scond	-011		

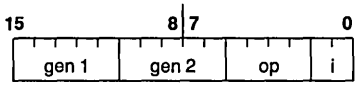
Appendix A: Instruction Formats (Continued)



Format 3

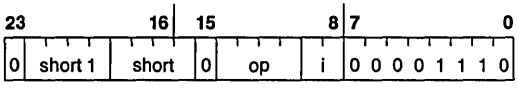
CXPD	-0000	ADJSP	-1010
BICPSR	-0010	JSR	-1100
JUMP	-0100	CASE	-1110
BISPSR	-0110		

Trap (UND) on XXX1, 1000



Format 4

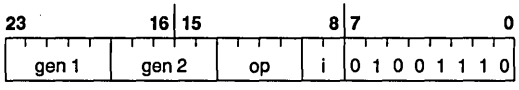
ADD	-0000	SUB	-1000
CMP	-0001	ADDR	-1001
BIC	-0010	AND	-1010
ADDC	-0100	SUBC	-1100
MOV	-0101	TBIT	-1101
OR	-0110	XOR	-1110



Format 5

MOVS	-0000	SETCFG*	-0010
CMPS	-0001	SKPS	-0011

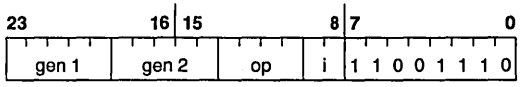
Trap (UND) on 1XXX, 01XX



Format 6

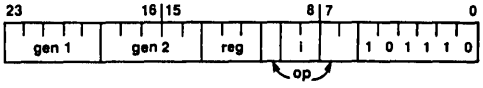
ROT	-0000	NEG	-1000
ASH	-0001	NOT	-1001
CBIT	-0010	Trap (UND)	-1010
CBITI	-0011	SUBP	-1011
Trap (UND)	-0100	ABS	-1100
LSH	-0101	COM	-1101
SBIT	-0110	IBIT	-1110
SBITI	-0111	ADDP	-1111

*Short 1 in format 5 applies only for SETCFG instruction. In other instructions this field is 0.



Format 7

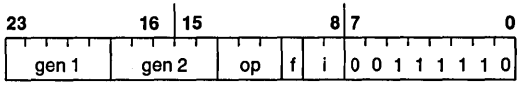
MOVM	-0000	MUL	-1000
CMPM	-0001	MEI	-1001
INSS	-0010	Trap (UND)	-1010
EXTS	-0011	DEI	-1011
MOVXBW	-0100	QUO	-1100
MOVZBW	-0101	REM	-1101
MOVZiD	-0110	MOD	-1110
MOVXiD	-0111	DIV	-1111



TL/EE/8673-78

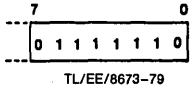
Format 8

EXT	-0 00	INDEX	-1 00
CVTP	-0 01	FFS	-1 01
INS	-0 10		
CHECK	-0 11		
MOVSU	-110, reg = 001		
MOVUS	-110, reg = 011		



Format 9

MOVif	-000	ROUND	-100
LFSR	-001	TRUNC	-101
MOVLF	-010	SFSR	-110
MOVFL	-011	FLOOR	-111

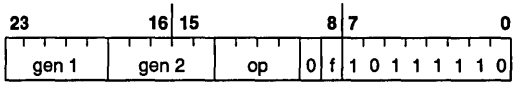


TL/EE/8673-79

Format 10

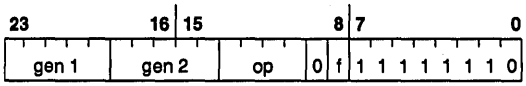
Trap (UND) Always

Appendix A: Instruction Formats (Continued)



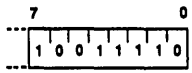
Format 11

ADDf	-0000	DIVf	-1000
MOVf	-0001	Note 1	-1001
CMPf	-0010	Trap (UND)	-1010
Note 3	-0011	Trap (UND)	-1011
SUBf	-0100	MULf	-1100
NEGf	-0101	ABSf	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111



Format 12

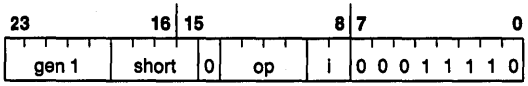
Note 2	-0000	Note 2	-1000
Note 1	-0001	Note 1	-1001
POLYf	-0010	Trap (UND)	-1010
DOTf	-0011	Trap (UND)	-1011
SCALBf	-0100	Note 2	-1100
LOGBf	-0101	Note 1	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111



TL/EE/6673–81

Format 13

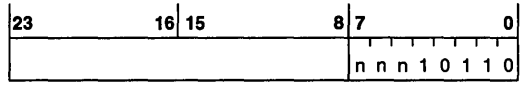
Trap (UND) Always



Format 14

RDVAL	-0000	LMR	-0010
WRVAL	-0001	SMR	-0011

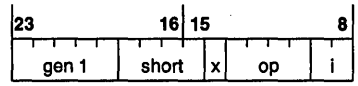
Trap (UND) on 01XX, 1XXX



Format 15

(Custom Slave)

nnn **Operation Word Format**

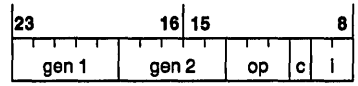


000

Format 15.0

CATST0	-0000	LCR	-0010
CATST1	-0001	SCR	-0011

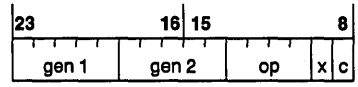
Trap (UND) on all others



001

Format 15.1

CCV3	-000	CCV2	-100
LCSR	-001	CCV1	-101
CCV5	-010	SCSR	-110
CCV4	-011	CCV0	-111



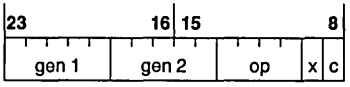
101

Format 15.5

CCAL0	-0000	CCAL3	-1000
CMOV0	-0001	CMOV3	-1001
CCMP0	-0010	Trap (UND)	-1010
CCMP1	-0011	Trap (UND)	-1011
CCAL1	-0100	CCAL2	-1100
CMOV2	-0101	CMOV1	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111

Appendix A: Instruction Formats (Continued)

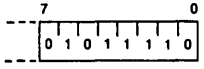
111



Format 15.7

Note 2	-0000	Note 2	-1000
Note 1	-0001	Note 1	-1001
Note 3	-0010	Trap (UND)	-1010
Note 3	-0011	Trap (UND)	-1011
Note 2	-0100	Note 2	-1100
Note 1	-0101	Note 1	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111

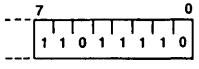
If nnn = 010, 011, 100, 110 then Trap (UND) Always.



TL/EE/8673-82

Format 16

Trap (UND) Always



TL/EE/8673-83

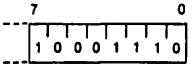
Note 1: Opcode not defined; CPU treats like MOV_f or $CMOV_c$. First operand has access class of read; second operand has access class of write; f or c field selects 32- or 64-bit data.

Note 2: Opcode not defined; CPU treats like ADD_f or $CCAL_c$. First operand has access class of read; second operand has access class of read-modify-write; f or c field selects 32- or 64-bit data.

Note 3: Opcode not defined; CPU treats like CMP_f or $CCMP_c$. First operand has access class of read; second operand has access class of read; f or c field selects 32- or 64-bit data.

Format 17

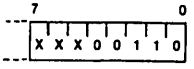
Trap (UND) Always



TL/EE/8673-84

Format 18

Trap (UND) Always

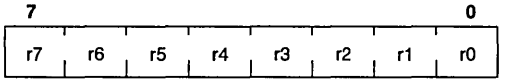


TL/EE/8673-85

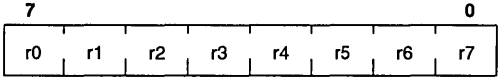
Format 19

Trap (UND) Always

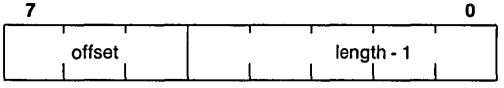
Implied Immediate Encodings:



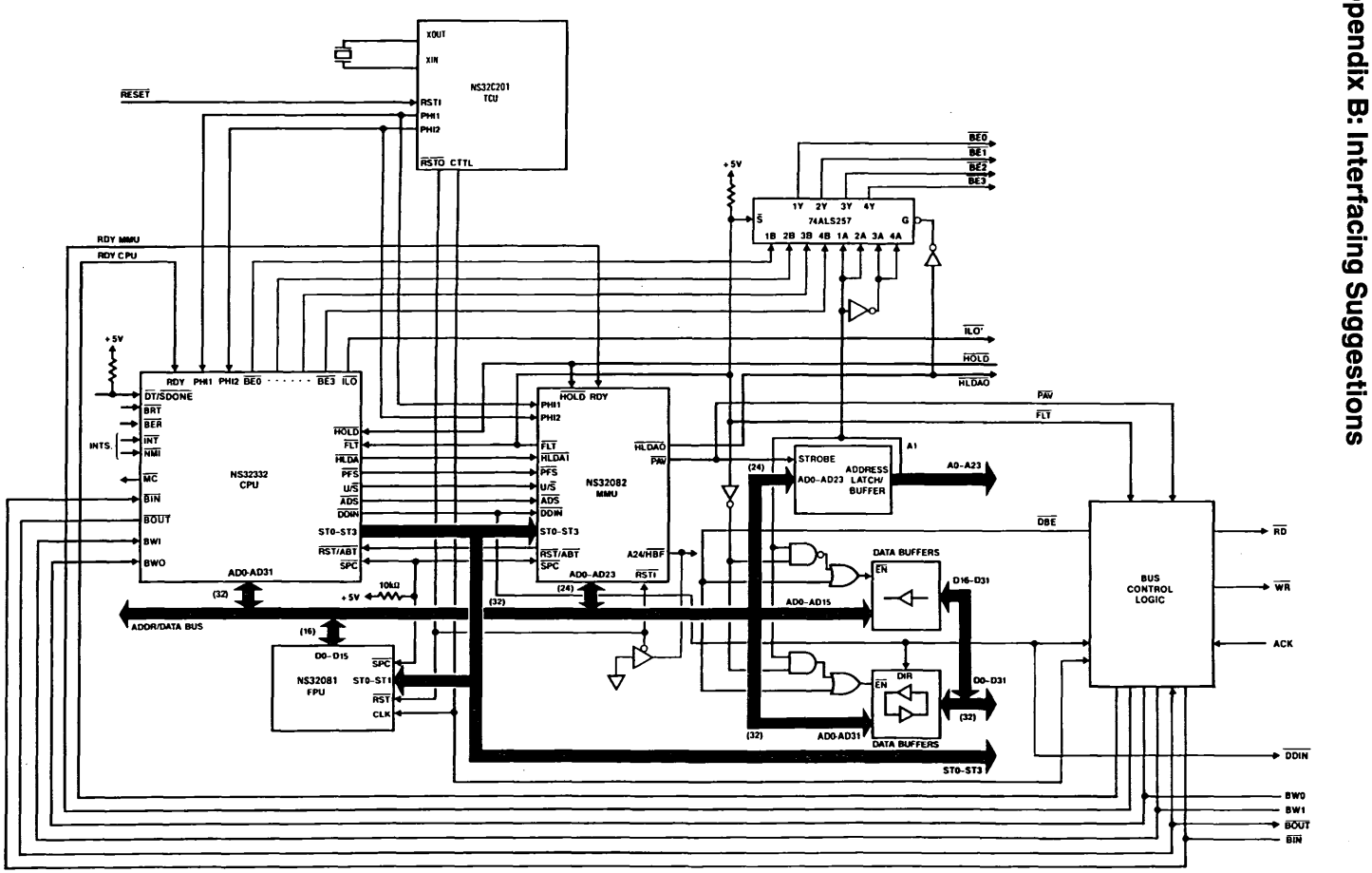
Register Mark, appended to SAVE, ENTER



Register Mark, appended to RESTORE, EXIT



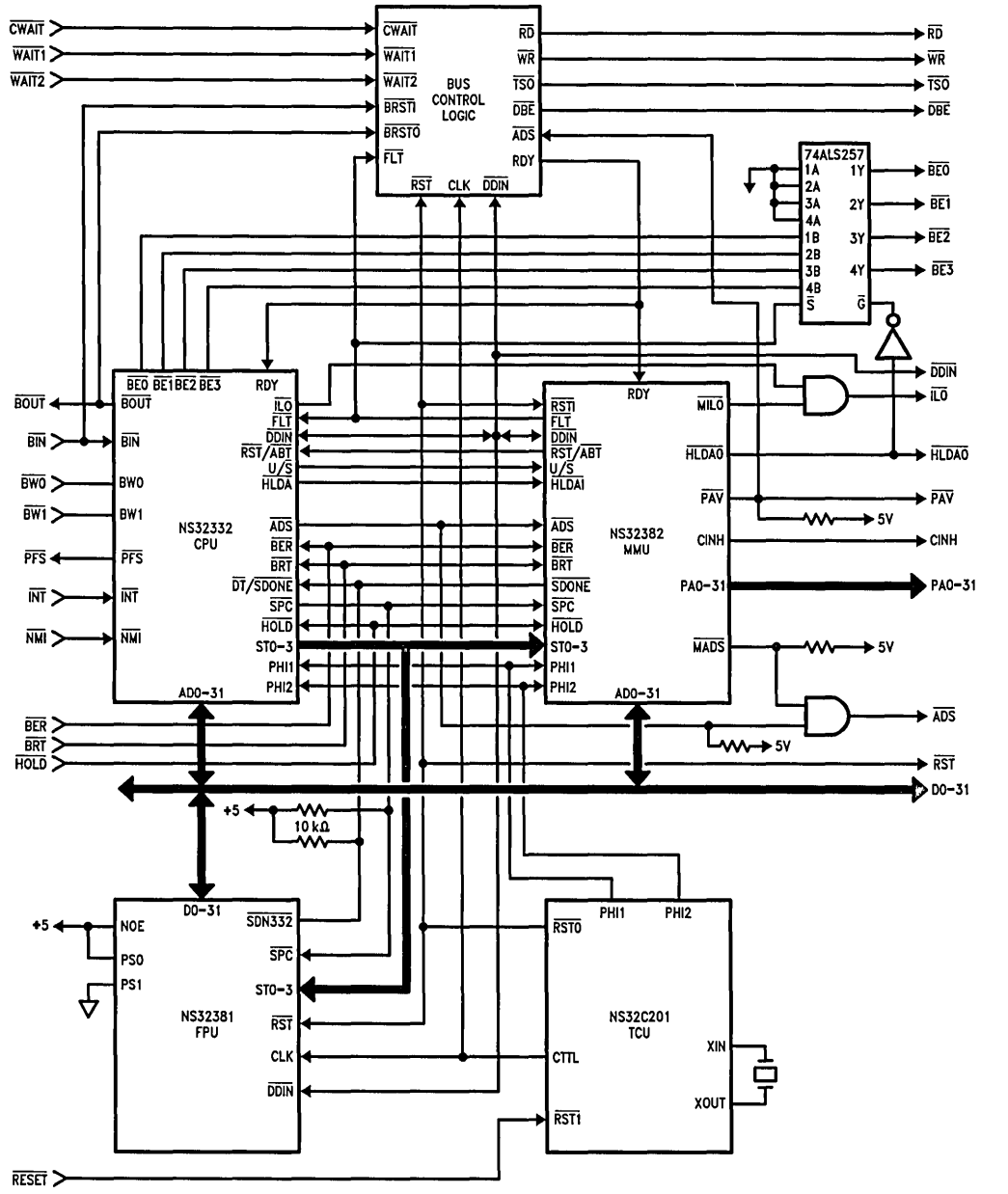
Offset/Length Modifier appended to INSS, EXTS



TL/EE/8673-86

FIGURE B-1. System Connection Diagram (32332, 32081 & 32082)

Appendix B: Interfacing Suggestions (Continued)



TL/EE/8673-93

FIGURE B-2. System Connection Diagram (32332, 32381 & 32382)

National Semiconductor

NS32C032-10/NS32C032-15 High-Performance Microprocessors

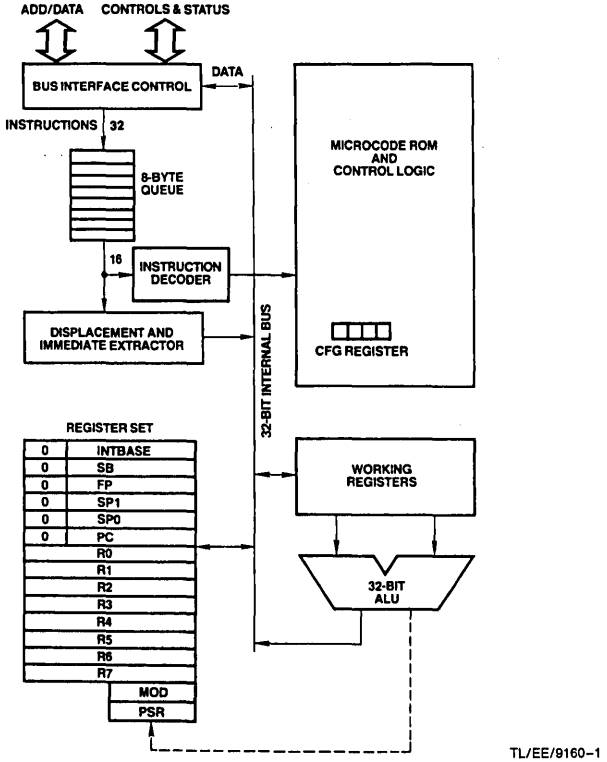
General Description

The NS32C032 is a 32-bit, virtual memory microprocessor with a 16-MByte linear address space and a 32-bit external data bus. It has a 32-bit ALU, eight 32-bit general purpose registers, an eight-byte prefetch queue, and a slave processor interface. The NS32C032 is fabricated with National Semiconductor's advanced CMOS process, and is fully object code compatible with other Series 32000® processors. The Series 32000 instruction set is optimized for modular, high-level languages (HLL). The set is very symmetric, it has a two address format, and it incorporates HLL oriented addressing modes. The capabilities of the NS32C032 can be expanded with the use of the NS32081 floating point unit (FPU), and the NS32082 demand-paged virtual memory management unit (MMU). Both devices interface to the NS32C032 as slave processors. The NS32C032 is a general purpose microprocessor that is ideal for a wide range of computational intensive applications.

Features

- 32-bit architecture and implementation
- Virtual memory support
- 16-MByte linear address space
- 32-bit data bus
- Powerful instruction set
  - — General 2-address capability
  - — Very high degree of symmetry
  - — Addressing modes optimized for high-level languages
- Series 32000 slave processor support
- High-speed CMOS technology
- 68-pin leadless chip carrier

Block Diagram



TL/EE/9160–1

FIGURE 1

Table of Contents

List of Illustrations

List of Illustrations (Continued)

List of Tables

1.0 Product Introduction

The Series 32000 microprocessor family is a new generation of devices using National's XMOS and CMOS technologies. By combining state-of-the-art MOS technology with a very advanced architectural design philosophy, this family brings mainframe computer processing power to VLSI processors.

The Series 32000 family supports a variety of system configurations, extending from a minimum low-cost system to a powerful 4 gigabyte system. The architecture provides complete upward compatibility from one family member to another. The family consists of a selection of CPUs supported by a set of peripherals and slave processors that provide sophisticated interrupt and memory management facilities as well as high-speed floating-point operations. The architectural features of the Series 32000 family are described briefly below:

Powerful Addressing Modes. Nine addressing modes available to all instructions are included to access data structures efficiently.

Data Types. The architecture provides for numerous data types, such as byte, word, doubleword, and BCD, which may be arranged into a wide variety of data structures.

Symmetric Instruction Set. While avoiding special case instructions that compilers can't use, the Series 32000 family incorporates powerful instructions for control operations, such as array indexing and external procedure calls, which save considerable space and time for compiled code.

Memory-to-Memory Operations. The Series 32000 CPUs represent two-address machines. This means that each operand can be referenced by any one of the addressing modes provided. This powerful memory-to-memory architecture permits memory locations to be treated as registers for all useful operations. This is important for temporary operands as well as for context switching.

Memory Management. Either the NS32382 or the NS32082 Memory Management Unit may be added to the system to provide advanced operating system support functions, including dynamic address translation, virtual memory management, and memory protection.

Large, Uniform Addressing. The NS32C032 has 24-bit address pointers that can address up to 16 megabytes without requiring any segmentation; this addressing scheme provides flexible memory management without added-on expense.

Modular Software Support. Any software package for the Series 32000 family can be developed independent of all other packages, without regard to individual addressing. In addition, ROM code is totally relocatable and easy to access, which allows a significant reduction in hardware and software cost.

Software Processor Concept. The Series 32000 architecture allows future expansions of the instruction set that can be executed by special slave processors, acting as extensions to the CPU. This concept of slave processors is unique to the Series 32000 family. It allows software compatibility even for future components because the slave hardware is transparent to the software. With future advances in semiconductor technology, the slaves can be physically integrated on the CPU chip itself.

To summarize, the architectural features cited above provide three primary performance advantages and characteristics:

- High-Level Language Support
- Easy Future Growth Path
- Application Flexibility

2.0 Architectural Description

2.1 PROGRAMMING MODEL

The Series 32000 architecture includes 16 registers on the NS32C032 CPU.

2.1.1 General Purpose Registers

There are eight registers for meeting high speed general storage requirements, such as holding temporary variables and addresses. The general purpose registers are free for any use by the programmer. They are thirty-two bits in length. If a general register is specified for an operand that is eight or sixteen bits long, only the low part of the register is used; the high part is not referenced or modified.

2.1.2 Dedicated Registers

The eight dedicated registers of the NS32C032 are assigned specific functions.

PC: The PROGRAM COUNTER register is a pointer to the first byte of the instruction currently being executed. The PC is used to reference memory in the program section. (In the NS32C032 the upper eight bits of this register are always zero.)

SP0, SP1: The SP0 register points to the lowest address of the last item stored on the INTERRUPT STACK. This stack is normally used only by the operating system. It is used primarily for storing temporary data, and holding return information for operating system subroutines and interrupt and trap service routines. The SP1 register points to the lowest address of the last item stored on the USER STACK. This stack is used by normal user programs to hold temporary data and subroutine return information.

In this document, reference is made to the SP register. The terms "SP register" or "SP" refer to either SP0 or SP1, depending on the setting of the S bit in the PSR register. If the S bit in the PSR is 0 the SP refers to SP0. If the S bit in the PSR is 1 then SP refers to SP1. (In the NS32C032 the upper eight bits of these registers are always zero).

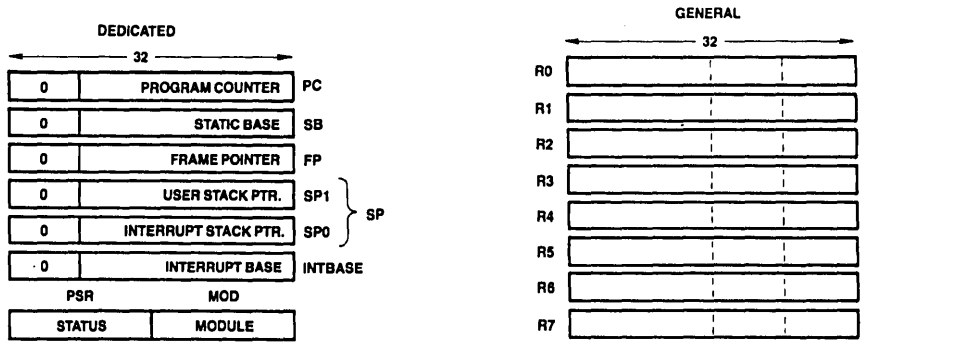
Stacks in the Series 32000 family grow downward in memory. A Push operation pre-decrements the Stack Pointer by the operand length. A Pop operation post-increments the Stack Pointer by the operand length.

FP: The FRAME POINTER register is used by a procedure to access parameters and local variables on the stack. The FP register is set up on procedure entry with the ENTER instruction and restored on procedure termination with the EXIT instruction.

The frame pointer holds the address in memory occupied by the old contents of the frame pointer. (In the NS32C032 the upper eight bits of this register are always zero.)

SB: The STATIC BASE register points to the global variables of a software module. This register is used to support relocatable global variables for software modules.

2.0 Architectural Description (Continued)



TL/EE/9160-3

FIGURE 2-1. The General and Dedicated Registers

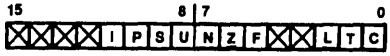
The SB register holds the lowest address in memory occupied by the global variables of a module. (In the NS32C032 the upper eight bits of this register are always zero.)

INTBASE: The INTERRUPT BASE register holds the address of the dispatch table for interrupts and traps (Sec. 3.8). The INTBASE register holds the lowest address in memory occupied by the dispatch table. (In the NS32C032 the upper eight bits of this register are always zero.)

MOD: The MODULE register holds the address of the module descriptor of the currently executing software module. The MOD register is sixteen bits long, therefore the module table must be contained within the first 64K bytes of memory.

PSR: The PROCESSOR STATUS REGISTER (PSR) holds the status codes for the NS32C032 microprocessor.

The PSR is sixteen bits long, divided into two eight-bit halves. The low order eight bits are accessible to all programs, but the high order eight bits are accessible only to programs executing in Supervisor Mode.



TL/EE/9160-4

FIGURE 2-2. Processor Status Register

C: The C bit indicates that a carry or borrow occurred after an addition or subtraction instruction. It can be used with the ADDC and SUBC instructions to perform multiple-precision integer arithmetic calculations. It may have a setting of 0 (no carry or borrow) or 1 (carry or borrow).

T: The T bit causes program tracing. If this bit is a 1, a TRC trap is executed after every instruction (Sec. 3.8.5).

L: The L bit is altered by comparison instructions. In a comparison instruction the L bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as unsigned integers. Otherwise, it is set to "0". In Floating Point comparisons, this bit is always cleared.

F: The F bit is a general condition flag, which is altered by many instructions (e.g., integer arithmetic instructions use it to indicate overflow).

Z: The Z bit is altered by comparison instructions. In a comparison instruction the Z bit is set to "1" if the second operand is equal to the first operand; otherwise it is set to "0".

N: The N bit is altered by comparison instructions. In a comparison instruction the N bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as signed integers. Otherwise, it is set to "0".

U: If the U bit is "1" no privileged instructions may be executed. If the U bit is "0" then all instructions may be executed. When U = 0 the NS32C032 is said to be in Supervisor Mode; when U = 1 the NS32C032 is said to be in User Mode. A User Mode program is restricted from executing certain instructions and accessing certain registers which could interfere with the operating system. For example, a User Mode program is prevented from changing the setting of the flag used to indicate its own privilege mode. A Supervisor Mode program is assumed to be a trusted part of the operating system, hence it has no such restrictions.

S: The S bit specifies whether the SP0 register or SP1 register is used as the stack pointer. The bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use the SP0 register) or 1 (use the SP1 register).

P: The P bit prevents a TRC trap from occurring more than once for an instruction (Sec. 3.8.5.). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I: If I = 1, then all interrupts will be accepted (Sec. 3.8.). If I = 0, only the NMI interrupt is accepted. Trap enables are not affected by this bit.

2.1.3 The Configuration Register (CFG)

Within the Control section of the NS32C032 CPU is the four-bit CFG Register, which declares the presence of certain external devices. It is referenced by only one instruction, SETCFG, which is intended to be executed only as part of system initialization after reset. The format of the CFG Register is shown in *Figure 2-3*.

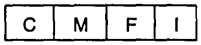


FIGURE 2-3. CFG Register

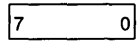
2.0 Architectural Description (Continued)

The CFG I bit declares the presence of external interrupt vectoring circuitry (specifically, the NS32202 Interrupt Control Unit). If the CFG I bit is set, interrupts requested through the $\overline{INT}$ pin are "Vectored." If it is clear, these interrupts are "Non-Vectored." See Sec. 3.8.

The F, M and C bits declare the presence of the FPU, MMU and Custom Slave Processors. If these bits are not set, the corresponding instructions are trapped as being undefined.

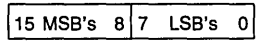
2.1.4 Memory Organization

The main memory of the NS32C032 is a uniform linear address space. Memory locations are numbered sequentially starting at zero and ending at $2^{24} - 1$. The number specifying a memory location is called an address. The contents of each memory location is a byte consisting of eight bits. Unless otherwise noted, diagrams in this document show data stored in memory with the lowest address on the right and the highest address on the left. Also, when data is shown vertically, the lowest address is at the top of a diagram and the highest address at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number zero, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.



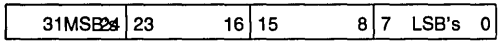
Byte at Address A

Two contiguous bytes are called a word. Except where noted (Sec. 2.2.1), the least significant byte of a word is stored at the lower address, and the most significant byte of the word is stored at the next higher address. In memory, the address of a word is the address of its least significant byte, and a word may start at any address.



Word at Address A

Two contiguous words are called a double word. Except where noted (Sec. 2.2.1), the least significant word of a double word is stored at the lowest address and the most significant word of the double word is stored at the address two greater. In memory, the address of a double word is the address of its least significant byte, and a double word may start at any address.



Double Word at Address A

Although memory is addressed as bytes, it is actually organized as double-words. Note that access time to a word or a double-word depends upon its address, e.g. double-words that are aligned to start at addresses that are multiples of four will be accessed more quickly than those not so aligned. This also applies to words that cross a double-word boundary.

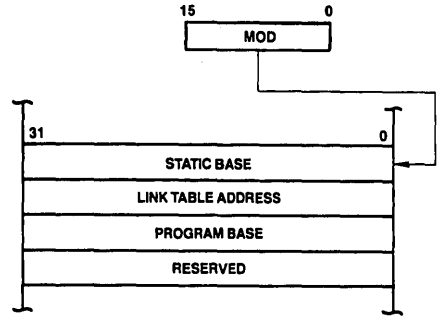
2.1.5 Dedicated Tables

Two of the NS32C032 dedicated registers (MOD and INTBASE) serve as pointers to dedicated tables in memory.

The INTBASE register points to the Interrupt Dispatch and Cascade tables. These are described in Sec. 3.8.

The MOD register contains a pointer into the Module Table, whose entries are called Module Descriptors. A Module Descriptor contains four pointers, three of which are used by NS32C032. The MOD register contains the address of the Module Descriptor for the currently running module. It is automatically up-dated by the Call External Procedure instructions (CXP and CXPD).

The format of a Module Descriptor is shown in *Figure 2-4*. The Static Base entry contains the address of static data assigned to the running module. It is loaded into the CPU Static Base register by the CXP and CXPD instructions. The Program Base entry contains the address of the first byte of instruction code in the module. Since a module may have multiple entry points, the Program Base pointer serves only as a reference to find them.



TL/EE/9160-5

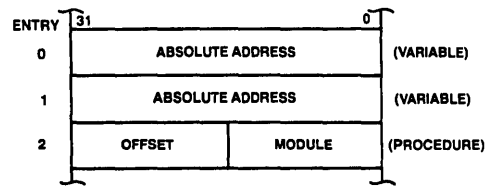
FIGURE 2-4. Module Descriptor Format

The Link Table Address points to the Link Table for the currently running module. The Link Table provides the information needed for:

1) Sharing variables between modules. Such variables are accessed through the Link Table via the External addressing mode.
2) Transferring control from one module to another. This is done via the Call External Procedure (CXP) instruction.

The format of a Link Table is given in *Figure 2-5*. A Link Table Entry for an external variable contains the 32-bit address of that variable. An entry for an external procedure contains two 16-bit fields: Module and Offset. The Module field contains the new MOD register contents for the module being entered. The Offset field is an unsigned number giving the position of the entry point relative to the new module's Program Base pointer.

For further details of the functions of these tables, see the Series 32000 Instruction Set Reference Manual.



TL/EE/9160-6

FIGURE 2-5. A Sample Link Table

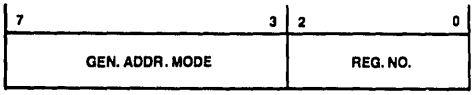
2.0 Architectural Description (Continued)

2.2 INSTRUCTION SET

2.2.1 General Instruction Format

Figure 2-6 shows the general format of a Series 32000 instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing Mode ("Gen") fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

Index Bytes appear when either or both Gen fields specify Scaled Index. In this case, the Gen field specifies only the Scale Factor (1, 2, 4 or 8), and the Index Byte specifies which General Purpose Register to use as the index, and which addressing mode calculation to perform before indexing. See *Figure 2-7*.



TL/EE/9160-8

FIGURE 2-7. Index Byte Format

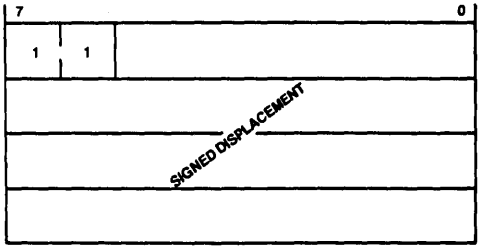
Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected address modes. Each Disp/Imm field may contain one or two displacements, or one immediate value. The size of a Displacement field is encoded with the top bits of that field, as shown in *Figure 2-8*, with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the Opcode field. Both Displacement and Immediate fields are stored most significant byte first. Note that this is different from the memory representation of data (Sec. 2.1.4).

Some instructions require additional, "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition (Sec. 2.2.3).

Byte Displacement: Range −64 to +63

Word Displacement: Range −8192 to +8191

Double Word Displacement: Range (Entire Addressing Space)

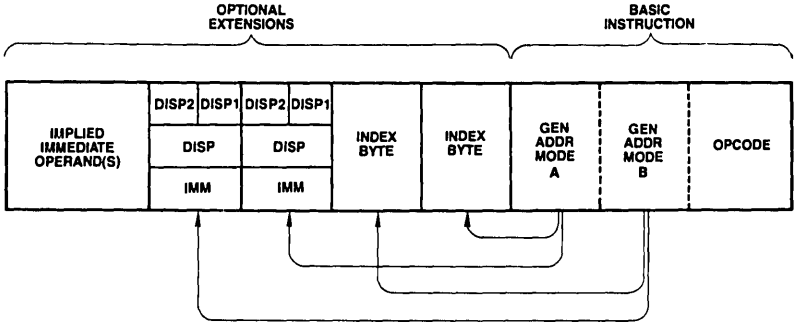


TL/EE/9160-11

FIGURE 2-8. Displacement Encodings

2.2.2 Addressing Modes

The NS32C032 CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."



TL/EE/9160-7

FIGURE 2-6. General Instruction Format

2.0 Architectural Description (Continued)

Addressing modes in the NS32C032 are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode, within the instruction that acts upon that variable. Extraneous data movement is therefore minimized.

NS32C032 Addressing Modes fall into nine basic types:

Register: The operand is available in one of the eight General Purpose Registers. In certain Slave Processor instructions, an auxiliary set of eight registers may be referenced instead.

Register Relative: A General Purpose Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.

Memory Space. Identical to Register Relative above, except that the register used is one of the dedicated registers PC, SP, SB or FP. These registers point to data areas generally needed by high-level languages.

Memory Relative: A pointer variable is found within the memory space pointed to by the SP, SB or FP register. A displacement is added to that pointer to generate the Effective Address of the operand.

Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.

Absolute: The address of the operand is specified by a displacement field in the instruction.

External: A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.

Top of Stack: The currently-selected Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.

Scaled Index: Although encoded as an addressing mode. Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any General Purpose Register by 1, 2, 4 or 8 and adding it into the total, yielding the final Effective Address of the operand.

Table 2-1 is a brief summary of the addressing modes. For a complete description of their actions, see the Instruction Set Reference Manual.

2.2.3 Instruction Set Summary

Table 2-2 presents a brief description of the NS32C032 instruction set. The Format column refers to the Instruction Format tables (Appendix A). The Instruction column gives the instruction as coded in assembly language, and the Description column provides a short description of the function provided by that instruction. Further details of the exact operations performed by each instruction may be found in the Instruction Set Reference Manual.

Notations:

i = Integer length suffix: B = Byte
W = Word
D = Double Word

f = Floating Point length suffix: F = Standard Floating
L = Long Floating

gen = General operand. Any addressing mode can be specified.

short = A 4-bit value encoded within the Basic Instruction (see Appendix A for encodings).

imm = Implied immediate operand. An 8-bit value appended after any addressing extensions.

disp = Displacement (addressing constant): 8, 16 or 32 bits. All three lengths legal.

reg = Any General Purpose Register: R0–R7.

areg = Any Dedicated/Address Register: SP, SB, FP, MOD, INTBASE, PSR, US (bottom 8 PSR bits).

mreg = Any Memory Management Status/Control Register.

creg = A Custom Slave Processor Register (Implementation Dependent).

cond = Any condition code, encoded as a 4-bit field within the Basic Instruction (see Appendix A for encodings).

2.0 Architectural Description (Continued)

TABLE 2-1
NS32C032 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0 or F0	None: Operand is in the specified register
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R7 or F7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Relative			
10000	Frame memory relative	disp2(disp1(FP))	Disp2 + Pointer; Pointer found at address Disp1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1(SP))	
10010	Static memory relative	disp2(disp1(SB))	
Reserved			
10011	(Reserved for Future Use)		
Immediate			
10100	Immediate	value	None: Operand is input from instruction queue.
Absolute			
10101	Absolute	@disp	Disp.
External			
10110	External	EXT (disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top of Stack			
10111	Top of stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register; "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	* + disp	
Scaled Index			
11100	Index, bytes	mode[Rn:B]	EA (mode) + Rn.
11101	Index, words	mode[Rn:W]	EA (mode) + 2× Rn.
11110	Index, double words	mode[Rn:D]	EA (mode) + 4× Rn.
11111	Index, quad words	mode[Rn:Q]	EA (mode) + 8 × Rn. 'Mode' and 'n' are contained within the Index Byte. EA (mode) denotes the effective address generated using mode.

2.0 Architectural Description (Continued)

TABLE 2-2
NS32C032 Instruction Set Summary

MOVES

Format	Operation	Operands	Description
4	MOVi	gen,gen	Move a value.
2	MOVQi	short,gen	Extend and move a signed 4-bit constant.
7	MOVMi	gen,gen,disp	Move Multiple: disp bytes (1 to 16).
7	MOVZBW	gen,gen	Move with zero extension.
7	MOVZiD	gen,gen	Move with zero extension.
7	MOVXBW	gen,gen	Move with sign extension.
7	MOVXiD	gen,gen	Move with sign extension.
4	ADDR	gen,gen	Move Effective Address.

INTEGER ARITHMETIC

Format	Operation	Operands	Description
4	ADDI	gen,gen	Add.
2	ADDQi	short,gen	Add signed 4-bit constant.
4	ADDCi	gen,gen	Add with carry.
4	SUBi	gen,gen	Subtract.
4	SUBCi	gen,gen	Subtract with carry (borrow).
6	NEGi	gen,gen	Negate (2's complement).
6	ABSi	gen,gen	Take absolute value.
7	MULi	gen,gen	Multiply
7	QUOi	gen,gen	Divide, rounding toward zero.
7	REMi	gen,gen	Remainder from QUO.
7	DIVi	gen,gen	Divide, rounding down.
7	MODi	gen,gen	Remainder from DIV (Modulus).
7	MEIi	gen,gen	Multiply to Extended Integer.
7	DEIi	gen,gen	Divide Extended Integer.

PACKED DECIMAL (BCD) ARITHMETIC

Format	Operation	Operands	Description
6	ADDPi	gen,gen	Add Packed.
6	SUBPi	gen,gen	Subtract Packed.

INTEGER COMPARISON

Format	Operation	Operands	Description
4	CMPi	gen,gen	Compare.
2	CMPQi	short,gen	Compare to signed 4-bit constant.
7	CMPMi	gen,gen,disp	Compare Multiple: disp bytes (1 to 16).

LOGICAL AND BOOLEAN

Format	Operation	Operands	Description
4	ANDi	gen,gen	Logical AND.
4	ORi	gen,gen	Logical OR.
4	BICi	gen,gen	Clear selected bits.
4	XORi	gen,gen	Logical Exclusive OR.
6	COMi	gen,gen	Complement all bits.
6	NOTi	gen,gen	Boolean complement: LSB only.
2	Scondi	gen	Save condition code (cond) as a Boolean variable of size i.

2.0 Architectural Description (Continued)

TABLE 2-2 (Continued)
NS32C032 Instruction Set Summary (Continued)

SHIFTS

Format	Operation	Operands	Description
6	LSHi	gen,gen	Logical Shift, left or right.
6	ASHi	gen,gen	Arithmetic Shift, left or right.
6	ROTi	gen,gen	Rotate, left or right.

BITS

Format	Operation	Operands	Description
4	TBITi	gen,gen	Test bit.
6	SBITi	gen,gen	Test and set bit.
6	SBITIi	gen,gen	Test and set bit, interlocked
6	CBITi	gen,gen	Test and clear bit.
6	CBITIi	gen,gen	Test and clear bit, interlocked.
6	IBITi	gen,gen	Test and invert bit.
8	FFSi	gen,gen	Find first set bit

BIT FIELDS

Bit fields are values in memory that are not aligned to byte boundaries. Examples are PACKED arrays and records used in Pascal. "Extract" instructions read and align a bit field. "Insert" instructions write a bit field from an aligned source.

Format	Operation	Operands	Description
8	EXTi	reg,gen,gen,disp	Extract bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented).
7	EXTSi	gen,gen,imm,imm	Extract bit field (short form).
7	INSSi	gen,gen,imm,imm	Insert bit field (short form).
8	CVTP	reg,gen,gen	Convert to Bit Field Pointer.

ARRAYS

Format	Operation	Operands	Description
8	CHECKi	reg,gen,gen	Index bounds check.
8	INDEXi	reg,gen,gen	Recursive indexing step for multiple-dimensional arrays.

STRINGS

String instructions assign specific functions to the General Purpose Registers:

R4 - Comparison Value

R3 - Translation Table Pointer

R2 - String 2 Pointer

R1 - String 1 Pointer

R0 - Limit Count

Options on all string instructions are:

B (Backward): Decrement string pointers after each step rather than incrementing.

U (Until match): End instruction if String 1 entry matches R4.

W (While match): End instruction if String 1 entry does not match R4.

All string instructions end when R0 decrements to zero.

Format	Operation	Operands	Descriptions
5	MOVSi	options	Move String 1 to String 2.
	MOVST	options	Move string, translating bytes.
5	CMPSi	options	Compare String 1 to String 2.
	CMPST	options	Compare translating, String 1 bytes.
5	SKPSi	options	Skip over String 1 entries
	SKPST	options	Skip, translating bytes for Until/While.

2.0 Architectural Description (Continued)

TABLE 2-2 (Continued)
NS32C032 Instruction Set Summary (Continued)

JUMPS AND LINKAGE

Format	Operation	Operands	Description
3	JUMP	gen	Jump.
0	BR	disp	Branch (PC Relative).
0	Bcond	disp	Conditional branch.
3	CASEi	gen	Multiway branch.
2	ACBi	short,gen,disp	Add 4-bit constant and branch if non-zero.
3	JSR	gen	Jump to subroutine.
1	BSR	disp	Branch to subroutine.
1	CXP	disp	Call external procedure.
3	CXPD	gen	Call external procedure using descriptor.
1	SVC		Supervisor Call.
1	FLAG		Flag Trap.
1	BPT		Breakpoint Trap.
1	ENTER	[reg list],disp	Save registers and allocate stack frame (Enter Procedure).
1	EXIT	[reg list]	Restore registers and reclaim stack frame (Exit Procedure).
1	RET	disp	Return from subroutine.
1	RXP	disp	Return from external procedure call.
1	RETT	disp	Return from trap. (Privileged)
1	RETI		Return from interrupt. (Privileged)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save General Purpose Registers.
1	RESTORE	[reg list]	Restore General Purpose Registers.
2	LPRi	areg,gen	Load Dedicated Register. (Privileged if PSR or INTBASE)
2	SPRi	areg,gen	Store Dedicated Register. (Privileged if PSR or INTBASE)
3	ADJSPi	gen	Adjust Stack Pointer.
3	BISPSRi	gen	Set selected bits in PSR. (Privileged if not Byte length)
3	BICPSRi	gen	Clear selected bits in PSR. (Privileged if not Byte length)
5	SETCFG	[option list]	Set Configuration Register. (Privileged)

FLOATING POINT

Format	Operation	Operands	Description
11	MOVf	gen,gen	Move a Floating Point value.
9	MOVLF	gen,gen	Move and shorten a Long value to Standard.
9	MOVFL	gen,gen	Move and lengthen a Standard value to Long.
9	MOVif	gen,gen	Convert any integer to Standard or Long Floating.
9	ROUNDfi	gen,gen	Convert to integer by rounding.
9	TRUNCfi	gen,gen	Convert to integer by truncating, toward zero.
9	FLOORfi	gen,gen	Convert to largest integer less than or equal to value.
11	ADDf	gen,gen	Add.
11	SUBf	gen,gen	Subtract.
11	MULf	gen,gen	Multiply.
11	DIVf	gen,gen	Divide.
11	CMPf	gen,gen	Compare.
11	NEGf	gen,gen	Negate.
11	ABSf	gen,gen	Take absolute value.
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

MEMORY MANAGEMENT

Format	Operation	Operands	Description
14	LMR	mreg,gen	Load Memory Management Register. (Privileged)
14	SMR	mreg,gen	Store Memory Management Register. (Privileged)
14	RDVAL	gen	Validate address for reading. (Privileged)
14	WRVAL	gen	Validate address for writing. (Privileged)
8	MOVSUi	gen,gen	Move a value from Supervisor Space to User Space. (Privileged)
8	MOVUSi	gen,gen	Move a value from User Space to Supervisor Space. (Privileged)

2.0 Architectural Description (Continued)

TABLE 2-2 (Continued)
NS32C032 Instruction Set Summary (Continued)

MISCELLANEOUS

Format	Operation	Operands	Description
1	NOP		No Operation.
1	WAIT		Wait for interrupt.
1	DIA		Diagnose. Single-byte "Branch to Self" for hardware breakpointing. Not for use in programming.

CUSTOM SLAVE

Format	Operation	Operands	Description
15.5	CCAL0c	gen,gen	Custom Calculate.
15.5	CCAL1c	gen,gen	
15.5	CCAL2c	gen,gen	
15.5	CCAL3c	gen,gen	
15.5	CMOV0c	gen,gen	Custom Move.
15.5	CMOV1c	gen,gen	
15.5	CMOV2c	gen,gen	
15.5	CMOV3c	gen,gen	
15.5	CCMP0c	gen,gen	Custom Compare.
15.5	CCMP1c	gen,gen	
15.1	CCV0ci	gen,gen	Custom Convert.
15.1	CCV1ci	gen,gen	
15.1	CCV2ci	gen,gen	
15.1	CCV3ic	gen,gen	
15.1	CCV4DQ	gen,gen	
15.1	CCV5QD	gen,gen	
15.1	LCSR	gen	Load Custom Status Register.
15.1	SCSR	gen	Store Custom Status Register.
15.0	CATST0	gen	Custom Address/Test. (Privileged)
15.0	CATST1	gen	(Privileged)
15.0	LCR	creg,gen	Load Custom Register. (Privileged)
15.0	SCR	creg,gen	Store Custom Register. (Privileged)

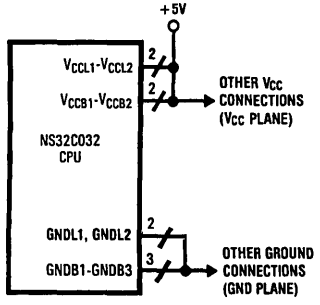
3.0 Functional Description

3.1 POWER AND GROUNDING

The NS32C032 requires a single 5-volt power supply, applied on 4 pins. The Logic Voltage pins ($V_{CC}L1$ and $V_{CC}L2$) supply the power to the on-chip logic. The Buffer Voltage pins (V_{CCB1} and V_{CCB2}) supply the power to the output drivers of the chip. The Logic Voltage pins and the Buffer Voltage pins should be connected together by a power (V_{CC}) plane on the printed circuit board.

The NS32C032 grounding connections are made on 5 pins. The Logic Ground pins (GNDL1 and GNDL2) are the ground pins for the on-chip logic. The Buffer Ground pins (GNDB1 to GNDB3) are the ground pins for the output drivers of the chip. The Logic Ground pins and the Buffer Ground pins should be connected together by a ground plane on the printed circuit board.

Both power and ground connections are shown below (Figure 3-1).



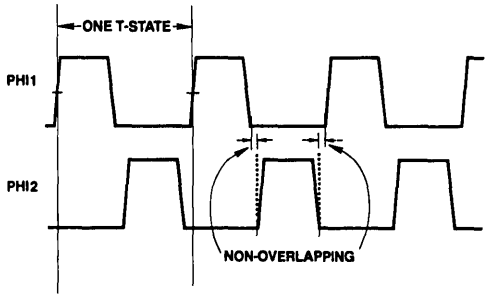
TL/EE/9160-12

FIGURE 3-1. Recommended Supply Connections

3.2 CLOCKING

The NS32C032 inputs clocking signals from the Timing Control Unit (TCU), which presents two non-overlapping phases of a single clock frequency. These phases are called PHI1 (pin 26) and PHI2 (pin 27). Their relationship to each other is shown in *Figure 3-2*.

Each rising edge of PHI1 defines a transition in the timing state ("T-State") of the CPU. One T-State represents the execution of one microinstruction within the CPU, and/or one step of an external bus transfer. See Section 4 for complete specifications of PHI1 and PHI2.



TL/EE/9160-13

FIGURE 3-2. Clock Timing Relationships

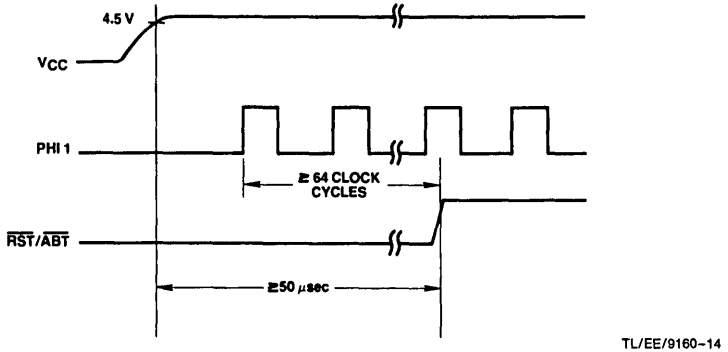
As the TCU presents signals with very fast transitions, it is recommended that the conductors carrying PHI1 and PHI2 be kept as short as possible, and that they not be connected anywhere except from the TCU to the CPU and, if present, the MMU. A TTL Clock signal (CTTL) is provided by the TCU for all other clocking.

3.3 RESETTING

The $\overline{RST}/\overline{ABT}$ pin serves both as a Reset for on-chip logic and as the Abort input for Memory-Managed systems. For its use as the Abort Command, see Sec. 3.5.4.

The CPU may be reset at any time by pulling the $\overline{RST}/\overline{ABT}$ pin low for at least 64 clock cycles. Upon detecting a reset, the CPU terminates instruction processing, resets its internal logic, and clears the Program Counter (PC) and Processor Status Register (PSR) to all zeroes.

On application of power, $\overline{RST}/\overline{ABT}$ must be held low for at least 50 μsec after V_{CC} is stable. This is to ensure that all on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must also remain



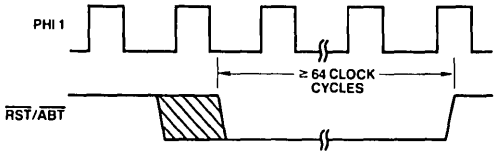
TL/EE/9160-14

FIGURE 3-3. Power-on Reset Requirements

3.0 Functional Description (Continued)

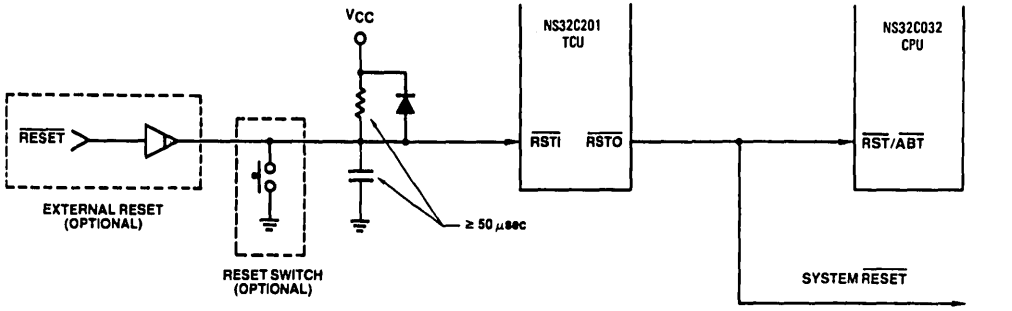
active for not less than 64 clock cycles. The rising edge must occur while PHI1 is high. See *Figures 3-3* and *3-4*.

The NS32C201 Timing Control Unit (TCU) provides circuitry to meet the Reset requirements of the NS32C032 CPU. *Figure 3-5a* shows the recommended connections for a non-Memory-Managed system. *Figure 3-5b* shows the connections for a Memory-Managed system.



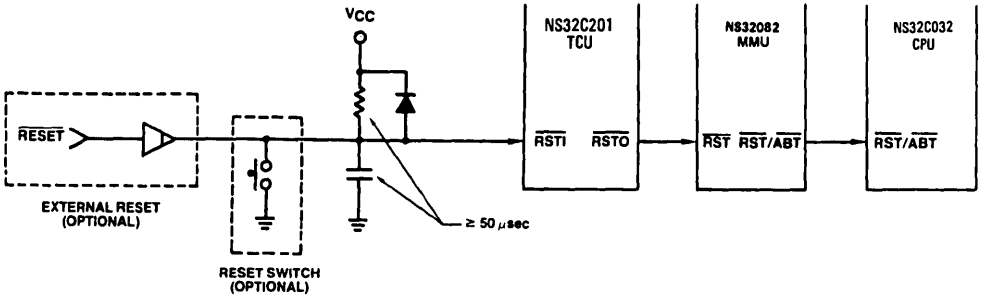
TL/EE/9160-15

FIGURE 3-4. General Reset Timing



TL/EE/9160-16

FIGURE 3-5a. Recommended Reset Connections, Non-Memory-Managed System



TL/EE/9160-17

FIGURE 3-5b. Recommended Reset Connections, Memory-Managed System

3.4 BUS CYCLES

The NS32C032 CPU has a strap option which defines the Bus Timing Mode as either With or Without Address Translation. This section describes only bus cycles under the No Address Translation option. For details of the use of the strap and of bus cycles with address translation, see Sec. 3.5.

The CPU will perform a bus cycle for one of the following reasons:

1) To write or read data, to or from memory or a peripheral interface device. Peripheral input and output are memory-mapped in the Series 32000 family.
2) To fetch instructions into the eight-byte instruction queue. This happens whenever the bus would otherwise be idle and the queue is not already full.
3) To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
4) To transfer information to or from a Slave Processor.

In terms of bus timing, cases 1 through 3 above are identical. For timing specifications, see Sec. 4. The only external difference between them is the four-bit code placed on the Bus Status pins (ST0–ST3). Slave Processor cycles differ in that separate control signals are applied (Sec. 3.4.6).

The sequence of events in a non-Slave bus cycle is shown below in *Figure 3-7* for a Read cycle and *Figure 3-8* for a Write cycle. The cases shown assume that the selected memory or interface device is capable of communicating with the CPU at full speed. If it is not, then cycle extension may be requested through the RDY line (Sec. 3.4.1).

3.0 Functional Description (Continued)

A full-speed bus cycle is performed in four cycles of the PHI1 clock signal, labeled T1 through T4. Clock cycles not associated with a bus cycle are designated Ti (for "Idle").

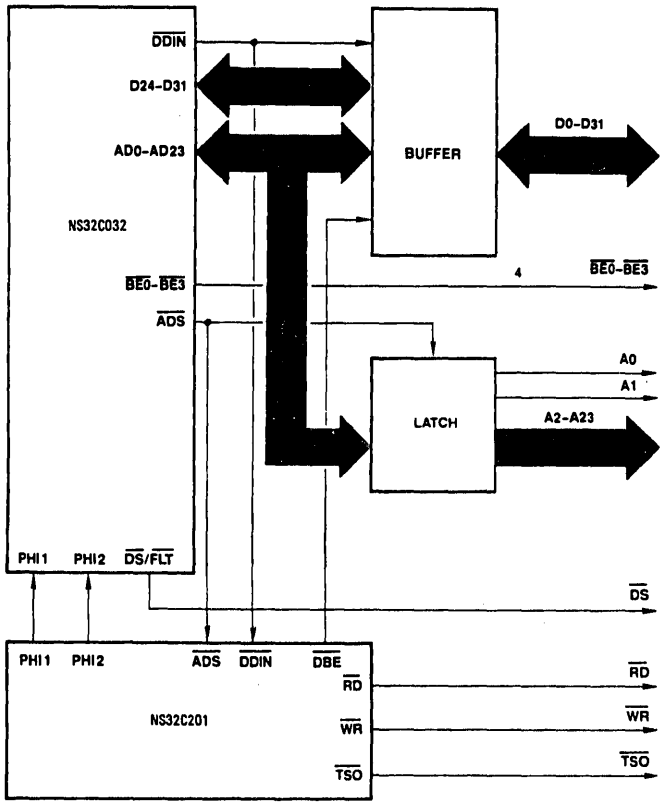
During T1, the CPU applies an address on pins AD0–AD23. It also provides a low-going pulse on the ADS pin, which serves the dual purpose of informing external circuitry that a bus cycle is starting and of providing control to an external latch for demultiplexing Address bits 0–23 from the AD0–AD23 pins. See *Figure 3-6*. During this time also the status signals DDIN, indicating the direction of the transfer, and $\overline{BE0}$–$\overline{BE3}$, indicating which of the four bus bytes are to be referenced, become valid.

During T2 the CPU switches the Data Bus, AD0–AD31 to either accept or present data. It also starts the data strobe ($\overline{DS}$), signalling the beginning of the data transfer. Associated signals from the NS32C201 Timing Control Unit are also activated at this time: $\overline{RD}$ (Read Strobe) or $\overline{WR}$ (Write Strobe), $\overline{TSO}$ (Timing State Output, indicating that T2 has been reached) and DBE (Data Buffer Enable).

The T3 state provides for access time requirements, and it occurs at least once in a bus cycle. At the end of T2 or T3, on the falling edge of the PHI2 clock, the RDY (Ready) line is sampled to determine whether the bus cycle will be extended (Sec. 3.4.1).

If the CPU is performing a Read cycle, the Data Bus (AD0–AD31) is sampled at the falling edge of PHI2 of the last T3 state. See Section 4. Data must, however, be held at least until the beginning of T4. $\overline{DS}$ and $\overline{RD}$ are guaranteed not to go inactive before this point, so the rising edge of either of them may safely be used to disable the device providing the input data.

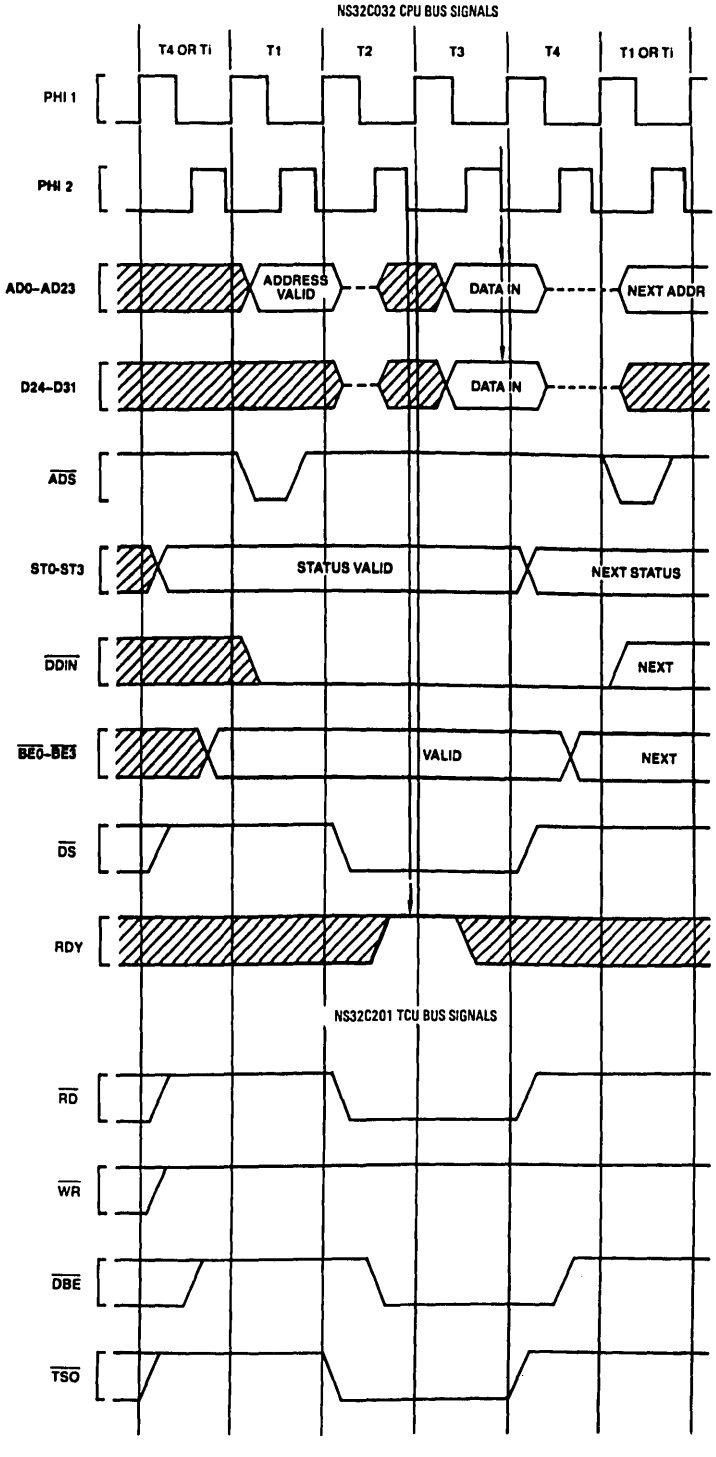
The T4 state finishes the bus cycle. At the beginning of T4, the $\overline{DS}$, $\overline{RD}$ or $\overline{WR}$, and $\overline{TSO}$ signals go inactive, and at the rising edge of PHI2, $\overline{DBE}$ goes inactive, having provided for necessary data hold times. Data during Write cycles remains valid from the CPU throughout T4. Note that the Bus Status lines (ST0–ST3) change at the beginning of T4, anticipating the following bus cycle (if any).



TL/EE/9160–18

FIGURE 3-6. Bus Connections

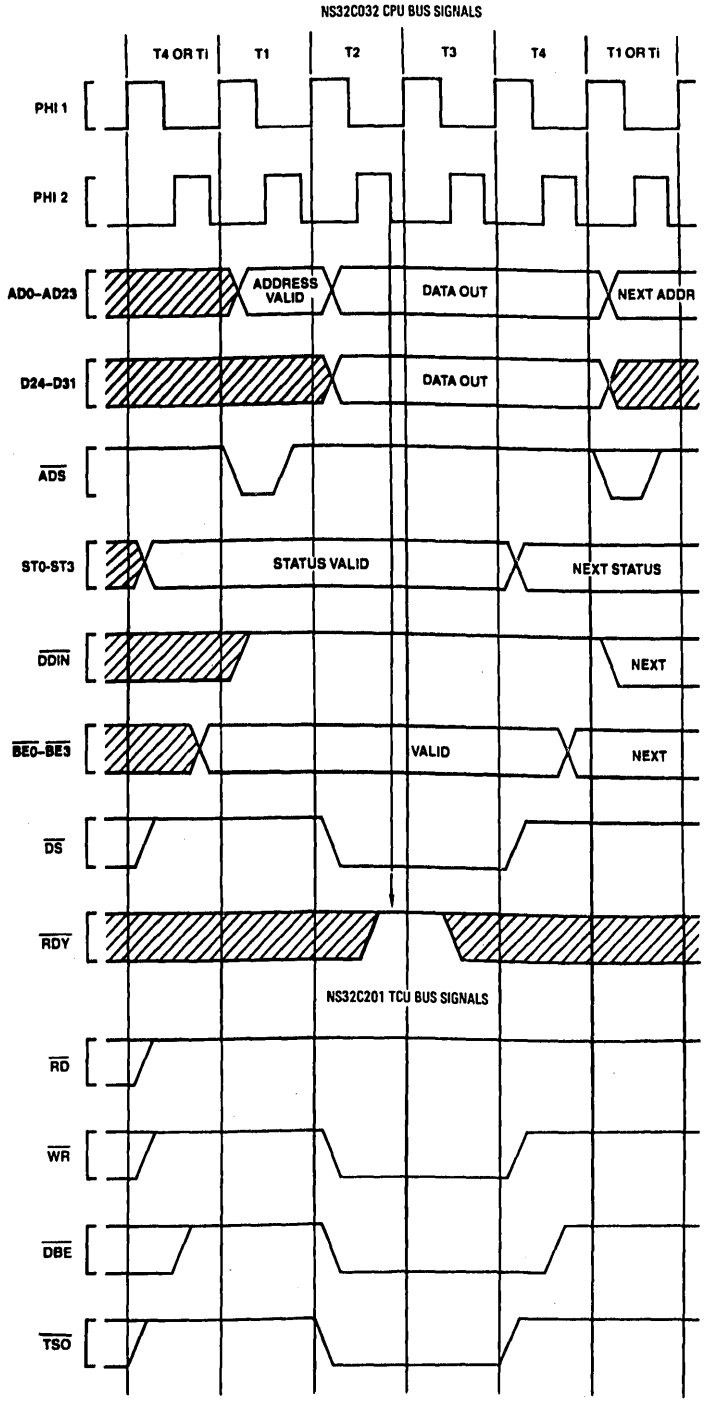
3.0 Functional Description (Continued)



TL/EE/9160–20

FIGURE 3-7. Read Cycle Timing

3.0 Functional Description (Continued)



TL/EE/9160–19

FIGURE 3-8. Write Cycle Timing

3.0 Functional Description (Continued)

3.4.1 Cycle Extension

To allow sufficient strobe widths and access times for any speed of memory or peripheral device, the NS32C032 provides for extension of a bus cycle. Any type of bus cycle except a Slave Processor cycle can be extended.

In *Figures 3-7* and *3-8*, note that during T3 all bus control signals from the CPU and TCU are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the RDY (Ready) pin.

At the end of T2 on the falling edge of PHI2, the RDY line is sampled by the CPU. If RDY is high, the next T-states will be T3 and then T4, ending the bus cycle. If RDY is low, then another T3 state will be inserted after the next T-state and the RDY line will again be sampled on the falling edge of PHI2. Each additional T3 state after the first is referred to as a "WAIT STATE". See *Figure 3-9*.

The RDY pin is driven by the NS32C201 Timing Control Unit, which applies WAIT States to the CPU as requested on three sets of pin:

1) $\overline{\text{CWAIT}}$ (Continuous WAIT), which holds the CPU in WAIT states until removed.
2) $\overline{\text{WAIT1}}$, $\overline{\text{WAIT2}}$, $\overline{\text{WAIT4}}$, $\overline{\text{WAIT8}}$ (Collectively $\overline{\text{WAITn}}$), which may be given a four-bit binary value requesting a specific number of WAIT States from 0 to 15.
3) $\overline{\text{PER}}$ (Peripheral), which inserts five additional WAIT states and causes the TCU to reshape the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ strobes. This provides the setup and hold times required by most MOS peripheral interface devices.

Combinations of these various WAIT requests are both legal and useful. For details of their use, see the NS32C201 Data Sheet.

Figure 3-10 illustrates a typical Read cycle, with two WAIT states requested through the TCU $\overline{\text{WAITn}}$ pins.

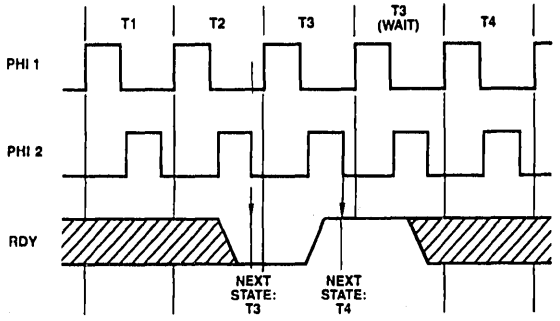


FIGURE 3-9. RDY Pin Timing

3.4.2 Bus Status

The NS32C032 CPU presents four bits of Bus Status information on pins ST0–ST3. The various combinations on these pins indicate why the CPU is performing a bus cycle, or, if it is idle on the bus, then why is it idle.

Referring to *Figures 3-7* and *3-8*, note that Bus Status leads the corresponding Bus Cycle, going valid one clock cycle before T1, and changing to the next state at T4. This allows the system designer to fully decode the Bus Status and, if desired, latch the decoded signals before $\overline{\text{ADS}}$ initiates the Bus Cycle.

The Bus Status pins are interpreted as a four-bit value, with ST0 the least significant bit. Their values decode as follows:

0000 – The bus is idle because the CPU does not need to perform a bus access.

0001 – The bus is idle because the CPU is executing the WAIT instruction.

0010 – (Reserved for future use.)

0011 – The bus is idle because the CPU is waiting for a Slave Processor to complete an instruction.

0100 – Interrupt Acknowledge, Master.

The CPU is performing a Read cycle. To acknowledge receipt of a Non-Maskable Interrupt (on $\overline{\text{NMI}}$) it will read from address FFFF00_{16}, but will ignore any data provided.

To acknowledge receipt of a Maskable Interrupt (on $\overline{\text{INT}}$) it will read from address FFFE00_{16}, expecting a vector number to be provided from the Master NS32202 Interrupt Control Unit. If the vectoring mode selected by the last SETCFG instruction was Non-Vectored, then the CPU will ignore the value it has read and will use a default vector instead, having assumed that no NS32202 is present. See Sec. 3.4.5.

0101 – Interrupt Acknowledge, Cascaded.

The CPU is reading a vector number from a Cascaded NS32202 Interrupt Control Unit. The address provided is the address of the NS32202 Hardware Vector register. See Sec. 3.4.5.

0110 – End of Interrupt, Master.

The CPU is performing a Read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction. See Sec. 3.4.5.

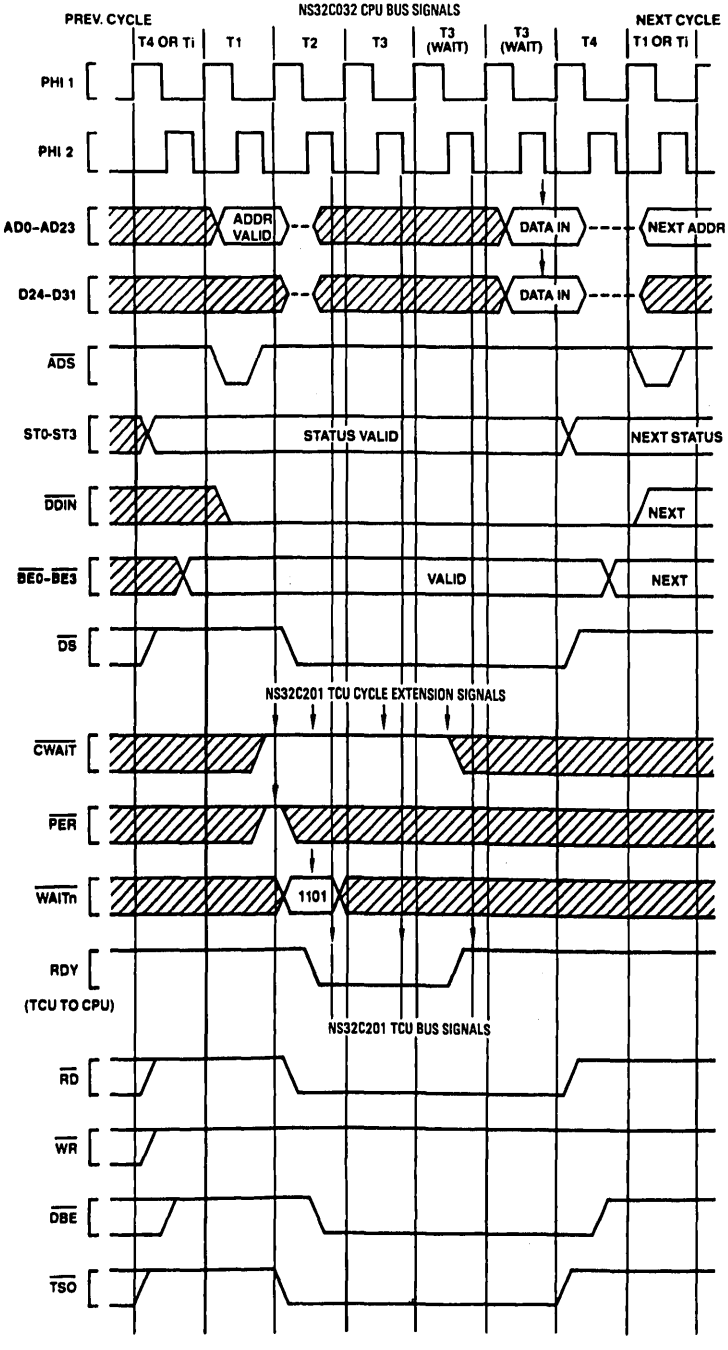
0111 – End of Interrupt, Cascaded.

The CPU is reading from a Cascaded Interrupt Control Unit to indicate that it is returning (through RETI) from an interrupt service routine requested by that unit. See Sec. 3.4.5.

1000 – Sequential Instruction Fetch.

The CPU is reading the next sequential word from the instruction stream into the Instruction

3.0 Functional Description (Continued)



TL/EE/9160-22

FIGURE 3-10. Extended Cycle Example

Note: Arrows on $\overline{\text{CWAIT}}$, $\overline{\text{PER}}$, $\overline{\text{WAIT}}$n indicate points at which the TCU samples. Arrows on AD0–AD15 and RDY indicate points at which the CPU samples.

3.0 Functional Description (Continued)

Queue. It will do so whenever the bus would otherwise be idle and the queue is not already full.

1001 – Non-Sequential Instruction Fetch.

The CPU is performing the first fetch of instruction code after the Instruction Queue is purged. This will occur as a result of any jump or branch, or any interrupt or trap, or execution of certain instructions.

1010 – Data Transfer.

The CPU is reading or writing an operand of an instruction.

1011 – Read RMW Operand.

The CPU is reading an operand which will subsequently be modified and rewritten. If memory protection circuitry would not allow the following Write cycle, it must abort this cycle.

1100 – Read for Effective Address Calculation.

The CPU is reading information from memory in order to determine the Effective Address of an operand. This will occur whenever an instruction uses the Memory Relative or External addressing mode.

1101 – Transfer Slave Processor Operand.

The CPU is either transferring an instruction operand to or from a Slave Processor, or it is issuing the Operation Word of a Slave Processor instruction. See Sec. 3.9.1.

1110 – Read Slave Processor Status.

The CPU is reading a Status Word from a Slave Processor. This occurs after the Slave Processor has signalled completion of an instruction. The transferred word tells the CPU whether a trap should be taken, and in some instructions it presents new values for the CPU Processor Status Register bits N, Z, L or F. See Sec. 3.9.1.

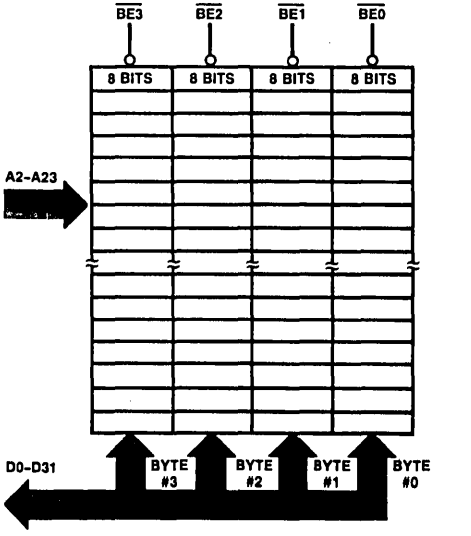
1111 – Broadcast Slave ID.

The CPU is initiating the execution of a Slave Processor instruction. The ID Byte (first byte of the instruction) is sent to all Slave Processors, one of which will recognize it. From this point the CPU is communicating with only one Slave Processor. See Sec. 3.9.1.

3.4.3 Data Access Sequences

The 24-bit address provided by the NS32C032 is a byte address; that is, it uniquely identifies one of up to 16,777,216 eight-bit memory locations. An important feature of the NS32C032 is that the presence of a 32-bit data bus imposes no restrictions on data alignment; any data item, regardless of size, may be placed starting at any memory address. The NS32C032 provides special control signals. Byte Enable ($\overline{BE0}$–$\overline{BE3}$) which facilitate individual byte accessing on a 32-bit bus.

Memory is organized as four eight-bit banks, each bank receiving the double-word address (A2–A23) in parallel. One bank, connected to Data Bus pins AD0–AD7 is enabled when $\overline{BE0}$ is low. The second bank, connected to data bus pins AD8–AD15 is enabled when $\overline{BE1}$ is low. The third and fourth banks are enabled by $\overline{BE2}$ and $\overline{BE3}$, respectively. See *Figure 3-11.*



TL/EE/9160–23

FIGURE 3-11. Memory Interface

Since operands do not need to be aligned with respect to the double-word bus access performed by the CPU, a given double-word access can contain one, two, three, or four bytes of the operand being addressed, and these bytes can begin at various positions, as determined by A1, A0. Table 3-1 lists the 10 resulting access types.

TABLE 3-1
Bus Access Types

Type	Bytes Accessed	A1,A0	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
1	1	00	1	1	1	0
2	1	01	1	1	0	1
3	1	10	1	0	1	1
4	1	11	0	1	1	1
5	2	00	1	1	0	0
6	2	01	1	0	0	1
7	2	10	0	0	1	1
8	3	00	1	0	0	0
9	3	01	0	0	0	1
10	4	00	0	0	0	0

Accesses of operands requiring more than one bus cycle are performed sequentially, with no idle T-States separating them. The number of bus cycles required to transfer an operand depends on its size and its alignment. Table 3-2 lists the bus cycles performed for each situation.

3.0 Functional Description (Continued)

TABLE 3-2
Access Sequences

Cycle	Type	Address	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Data Bus Byte 3	Byte 2	Byte 1	Byte 0
A. ***Word at address ending with 11***									BYTE 1 | BYTE 0	← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	1	A + 1	1	1	1	0	X	X	X	Byte 1
B. ***Double word at address ending with 01***							BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	9	A	0	0	0	1	Byte 2	Byte 1	Byte 0	X
2.	1	A + 3	1	1	1	0	X	X	X	Byte 3
C. ***Double word at address ending with 10***							BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	7	A	0	0	1	1	Byte 1	Byte 0	X	X
2.	5	A + 2	1	1	0	0	X	X	Byte 3	Byte 2
D. ***Double word at address ending with 11***							BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	8	A + 1	1	0	0	0	X	Byte 3	Byte 2	Byte 1
E. ***Quad word at address ending with 00***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	10	A	0	0	0	0	Byte 3	Byte 2	Byte 1	Byte 0
Other bus cycles (instruction prefetch or slave) can occur here.										
2.	10	A + 4	0	0	0	0	Byte 7	Byte 6	Byte 5	Byte 4
F. ***Quad word at address ending with 01***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	9	A	0	0	0	1	Byte 2	Byte 1	Byte 0	X
2.	1	A + 3	1	1	1	0	X	X	X	Byte 3
Other bus cycles (instruction prefetch or slave) can occur here.										
3.	9	A + 4	0	0	0	1	Byte 6	Byte 5	Byte 4	X
4.	1	A + 7	1	1	1	0	X	X	X	Byte 7
G. ***Quad word at address ending with 10***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	7	A	0	0	1	1	Byte 1	Byte 0	X	X
2.	5	A + 2	1	1	0	0	X	X	Byte 3	Byte 2
Other bus cycles (instruction prefetch or slave) can occur here.										
3.	7	A + 4	0	0	1	1	Byte 5	Byte 4	X	X
4.	5	A + 6	1	1	0	0	X	X	Byte 7	Byte 6
H. ***Quad word at address ending with 11***							BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	8	A + 1	1	0	0	0	X	Byte 3	Byte 2	Byte 1
Other bus cycles (instruction prefetch or slave) can occur here.										
1.	4	A + 4	0	1	1	1	Byte 4	X	X	X
2.	8	A + 5	1	0	0	0	X	Byte 7	Byte 6	Byte 5

X = Don't Care

3.0 Functional Description (Continued)

3.4.3.1 Bit Accesses

The Bit Instructions perform byte accesses to the byte containing the designated bit. The Test and Set Bit instruction (SBIT), for example, reads a byte, alters it, and rewrites it, having changed the contents of one bit.

3.4.3.2 Bit Field Accesses

An access to a Bit Field in memory always generates a Double-Word transfer at the address containing the least significant bit of the field. The Double Word is read by an Extract instruction; an Insert instruction reads a Double Word, modifies it, and rewrites it.

3.4.3.3 Extending Multiply Accesses

The Extending Multiply Instruction (MEI) will return a result which is twice the size in bytes of the operand it reads. If the multiplicand is in memory, the most-significant half of the result is written first (at the higher address), then the least-significant half. This is done in order to support retry if this instruction is aborted.

3.4.4 Instruction Fetches

Instructions for the NS32C032 CPU are "prefetched"; that is, they are input before being needed into the next available entry of the eight-byte Instruction Queue. The CPU performs two types of Instruction Fetch cycles: Sequential and Non-Sequential. These can be distinguished from each other by their differing status combinations on pins ST0–ST3 (Sec. 3.4.2).

A Sequential Fetch will be performed by the CPU whenever the Data Bus would otherwise be idle and the Instruction Queue is not currently full. Sequential Fetches are always type 10 Read cycles (Table 3-1).

A Non-Sequential Fetch occurs as a result of any break in the normally sequential flow of a program. Any jump or branch instruction, a trap or an interrupt will cause the next Instruction Fetch cycle to be Non-Sequential. In addition, certain instructions flush the instruction queue, causing the next instruction fetch to display Non-Sequential status. Only the first bus cycle after a break displays Non-Sequential status, and that cycle depends on the destination address.

Note: During non-sequential fetches, $\overline{BE0}$–$\overline{BE3}$ are all active regardless of the alignment.

3.4.5 Interrupt Control Cycles

Activating the $\overline{INT}$ or $\overline{NMI}$ pin on the CPU will initiate one or more bus cycles whose purpose is interrupt control rather than the transfer of instructions or data. Execution of the Return from Interrupt instruction (RETI) will also cause Interrupt Control bus cycles. These differ from instruction or data transfers only in the status pesented on pins ST0–ST3. All Interrupt Control cycles are single-byte Read cycles.

This section describes only the Interrupt Control sequences associated with each interrupt and with the return from its service routine. For full details of the NS32C032 interrupt structure, see Sec. 3.8.

3.0 Functional Description (Continued)

TABLE 3-3
Interrupt Sequences

Cycle	Status	Address	$\overline{DDIN}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Data Bus Byte 3	Byte 2	Byte 1	Byte 0
A. Non-Maskable Interrupt Control Sequences											
Interrupt Acknowledge											
1	0100	$FFFF00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
None: Performed through Return from Trap (RETT) instruction.											
B. Non-Vectored Interrupt Control Sequences											
Interrupt Acknowledge											
1	0100	$FFFE00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
1	0110	$FFFE00_{16}$	0	1	1	1	0	X	X	X	X
C. Vectored Interrupt Sequences: Non-Cascaded.											
Interrupt Acknowledge											
1	0100	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Range: 0-127
Interrupt Return											
1	0110	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Same as in Previous Int. Ack. Cycle
D. Vectored Interrupt Sequences: Cascaded											
Interrupt Acknowledge											
1	0100	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: range −16 to −1
(The CPU here uses the Cascade Index to find the Cascade Address.)											
2	0101	Cascade Address	0	See Note				Vector, range 9–255; on appropriate byte of data bus.			
Interrupt Return											
1	0110	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: Same as in previous Int. Ack. Cycle
(The CPU here uses the Cascade Index to find the Cascade Address)											
2	0111	Cascade Address	0	See Note				X	X	X	X

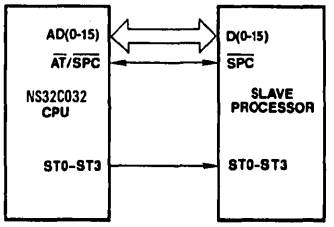
X = Don't Care

Note: $\overline{BE0}$-$\overline{BE3}$ signals will be activated according to the cascaded ICU address. The cycle type can be 1, 2, 3 or 4, when reading the interrupt vector. The vector value can be in the range 0–255.

3.0 Functional Description (Continued)

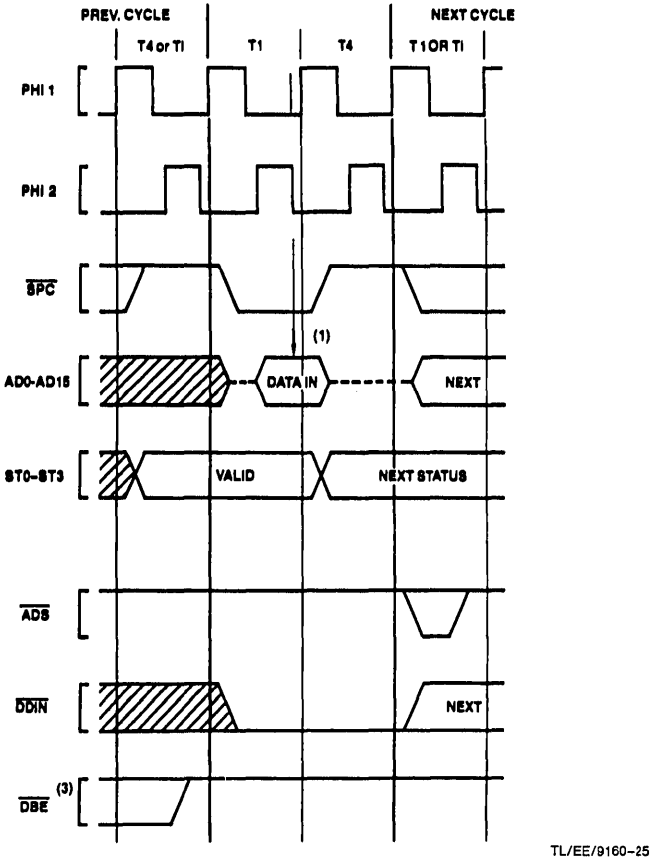
3.4.6 Slave Processor Communication

In addition to its use as the Address Translation strap (Sec. 3.5.1), the $\overline{AT}/\overline{SPC}$ pin is used as the data strobe for Slave Processor transfers. In this role, it is referred to as Slave Processor Control ($\overline{SPC}$). In a Slave Processor bus cycle, data is transferred on the Data Bus (AD0–AD15), and the status lines (ST0–ST3) are monitored by each Slave Processor in order to determine the type of transfer being performed. $\overline{SPC}$ is bidirectional, but is driven by the CPU during all Slave Processor bus cycles. See Sec. 3.9 for full protocol sequences.



TL/EE/9160-24

FIGURE 3-12. Slave Processor Connections



TL/EE/9160-25

Note:

(1) CPU samples Data Bus here.

(2) $\overline{DBE}$ and all other NS32C201 TCU bus signals remain inactive because no $\overline{ADS}$ pulse is received from the CPU.

FIGURE 3-13. CPU Read from Slave Processor

3.0 Functional Description (Continued)

3.4.6.1 Slave Processor Bus Cycles

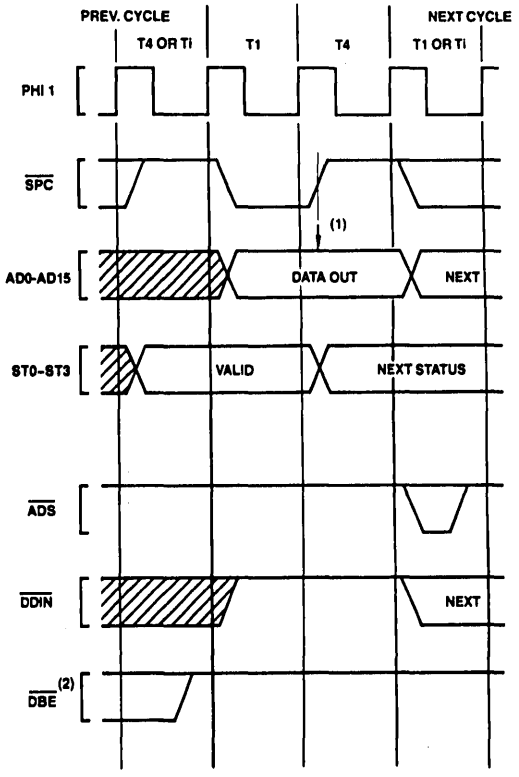
A Slave Processor bus cycle always takes exactly two clock cycles, labeled T1 and T4 (see *Figures 3-13* and *3-14*). During a Read cycle $\overline{SPC}$ is active from the beginning of T1 to the beginning of T4, and the data is sampled at the end of T1. The Cycle Status pins lead the cycle by one clock period, and are sampled at the leading edge of $\overline{SPC}$. During a Write cycle, the CPU applies data and activates $\overline{SPC}$ at T1, removing $\overline{SPC}$ at T4. The Slave Processor latches status on the leading edge of $\overline{SPC}$ and latches data on the trailing edge.

Since the CPU does not pulse the Address Strobe ($\overline{ADS}$), no bus signals are generated by the NS32C201 Timing Control Unit. The direction of a transfer is determined by the sequence ("protocol") established by the instruction under execution; but the CPU indicates the direction on the $\overline{DDIN}$ pin for hardware debugging purposes.

3.4.6.2 Slave Operand Transfer Sequences

A Slave Processor operand is transferred in one or more Slave bus cycles. A Byte operand is transferred on the least-significant byte of the Data Bus (AD0–AD7), and a Word operand is transferred on bits AD0–AD15. A Double Word is transferred in a consecutive pair of bus cycles, least-significant word first. A Quad Word is transferred in two pairs of Slave cycles, with other bus cycles possibly occurring between them. The word order is from least-significant word to most-significant.

Note that the NS32C032 uses only the two least significant bytes of the data bus for slave cycles. This is to maintain compatibility with existing slave processors.



TL/EE/9160–26

Note:

(1) Slave Processor samples Data Bus here.

(2) $\overline{DBE}$, being provided by the NS32C201 TCU, remains inactive due to the fact that no pulse is presented on $\overline{ADS}$. TCU signals $\overline{RD}$, $\overline{WR}$ and $\overline{TSO}$ also remain inactive.

FIGURE 3-14. CPU Write to Slave Processor

3.0 Functional Description (Continued)

3.5 MEMORY MANAGEMENT OPTION

The NS32C032 CPU, in conjunction with the NS32082 Memory Management Unit (MMU), provides full support for address translation, memory protection, and memory allocation techniques up to and including Virtual Memory.

3.5.1 Address Translation Strap

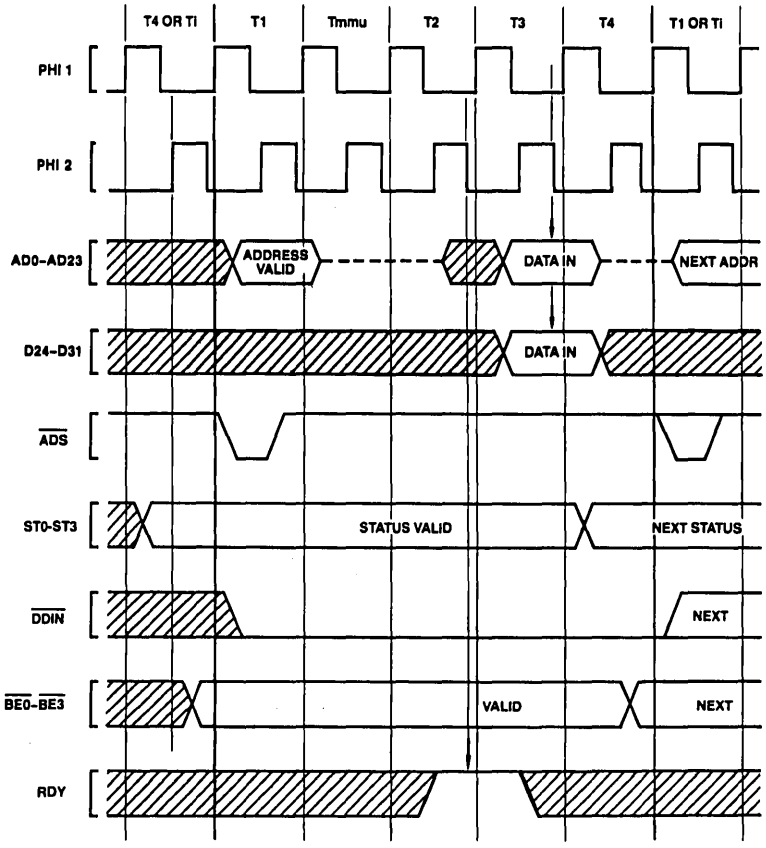
The Bus Interface Control section of the NS32C032 CPU has two bus timing modes: With or Without Address Translation. The mode of operation is selected by the CPU by sampling the $\overline{AT}/\overline{SPC}$ (Address Translation/Slave Processor Control) pin on the rising edge of the $\overline{RST}$ (Reset) pulse.

If $\overline{AT}/\overline{SPC}$ is sampled as high, the bus timing is as previously described in Sec. 3.4. If it is sampled as low, two changes occur:

1) An extra clock cycle, Tmmu, is inserted into all bus cycles except Slave Processor transfers.
2) The $\overline{DS}/\overline{FLT}$ pin changes in function from a Data Strobe output ($\overline{DS}$) to a Float Command input ($\overline{FLT}$).

The NS32082 MMU will itself pull the CPU $\overline{AT}/\overline{SPC}$ pin low when it is reset. In non-Memory-Managed systems this pin should be pulled up to V_{CC} through a 10 kΩ resistor.

Note that the Address Translation strap does not specifical-



TL/EE/9160-27

FIGURE 3-15. Read Cycle with Address Translation (CPU Action)

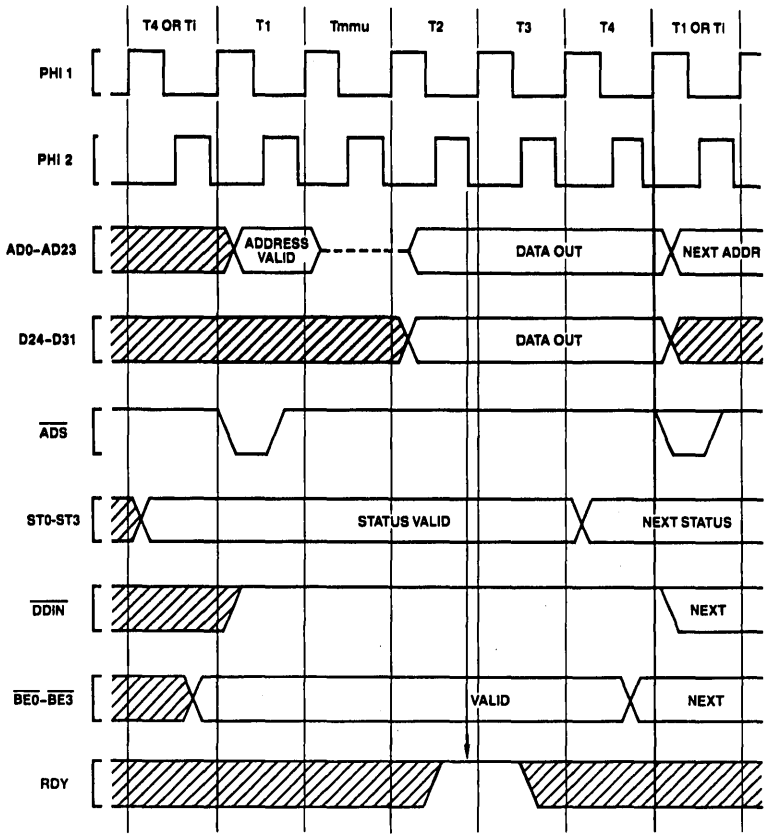
3.0 Functional Description (Continued)

ly declare the presence of an NS32082 MMU, but only the presence of external address translation circuitry. MMU instructions will still trap as being undefined unless the SETCFG (Set Configuration) instruction is executed to declare the MMU instruction set valid. See Sec. 2.1.3.

3.5.2 Translated Bus Timing

Figures 3-15 and *3-16* illustrate the CPU activity during a Read cycle and a Write cycle in Address Translation mode. The additional T-State, Tmmu, is inserted between T1 and T2. During this time the CPU places AD0–AD23 into the TRI-STATE® mode, allowing the MMU to assert the translated address and issue the physical address strobe $\overline{PAV}$. T2 through T4 of the cycle are identical to their counterparts without Address Translation. Note that in order for the NS32082 MMU to operate correctly it must be set to the 32032 mode by forcing A24/$\overline{HBF}$ low during reset. In this mode the bus lines AD16–AD23 are floated after the MMU address has been latched, since they are used by the CPU to transfer data.

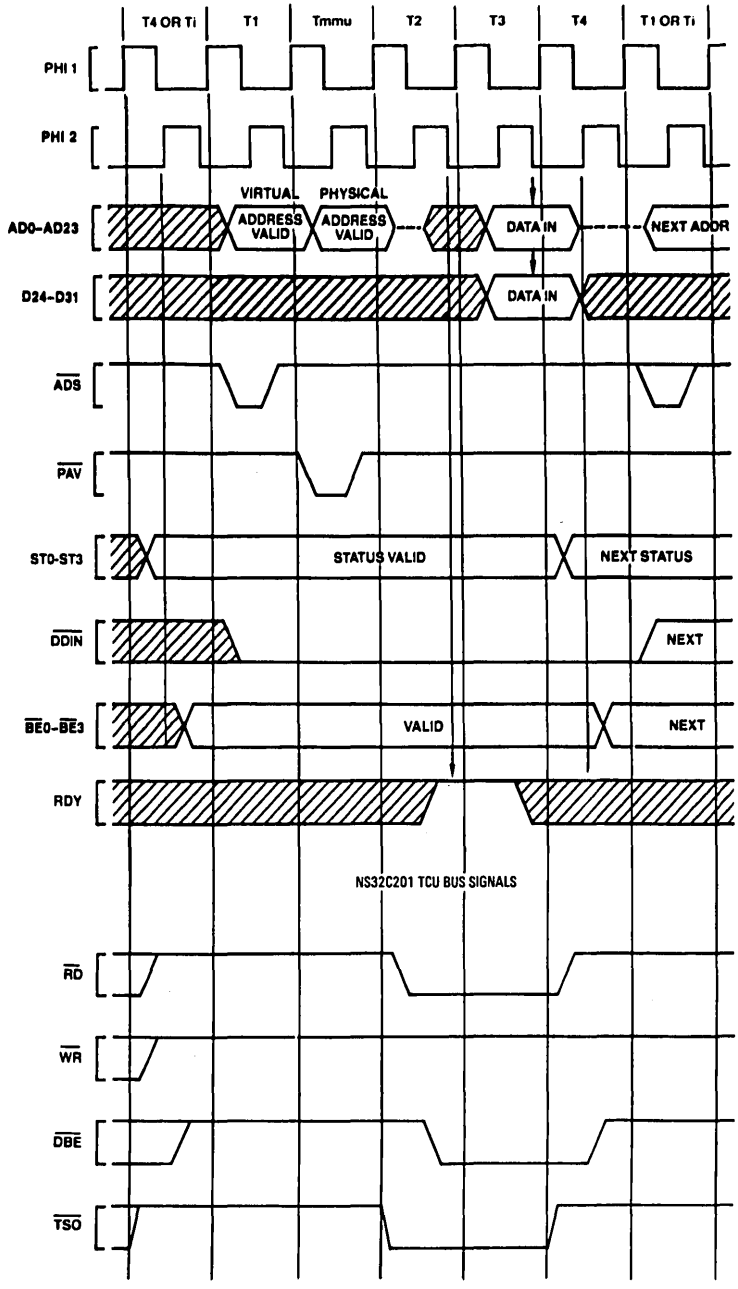
Figures 3-17 and *3-18* show a Read cycle and a Write cycle as generated by the 32C032/32082/32C201 group. Note that with the CPU $\overline{ADS}$ signal going only to the MMU, and with the MMU $\overline{PAV}$ signal substituting for $\overline{ADS}$ everywhere else, Tmmu through T4 look exactly like T1 through T4 in a non-Memory-Managed system. For the connection diagram, see Appendix B.



TL/EE/9160–28

FIGURE 3-16. Write Cycle with Address Translation (CPU Action)

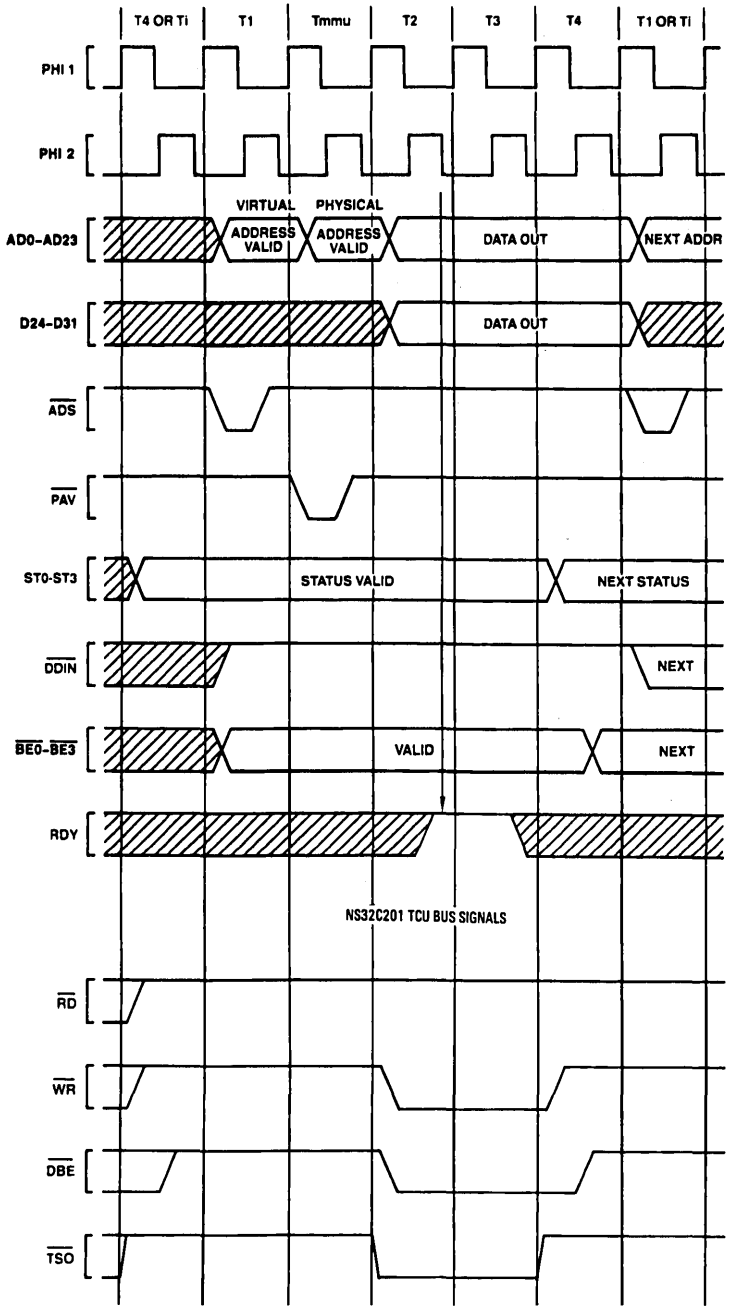
3.0 Functional Description (Continued)



TL/EE/9160-29

FIGURE 3-17. Memory-Managed Read Cycle

3.0 Functional Description (Continued)



TL/EE/9160-30

FIGURE 3-18. Memory-Managed Write Cycle

3.0 Functional Description (Continued)

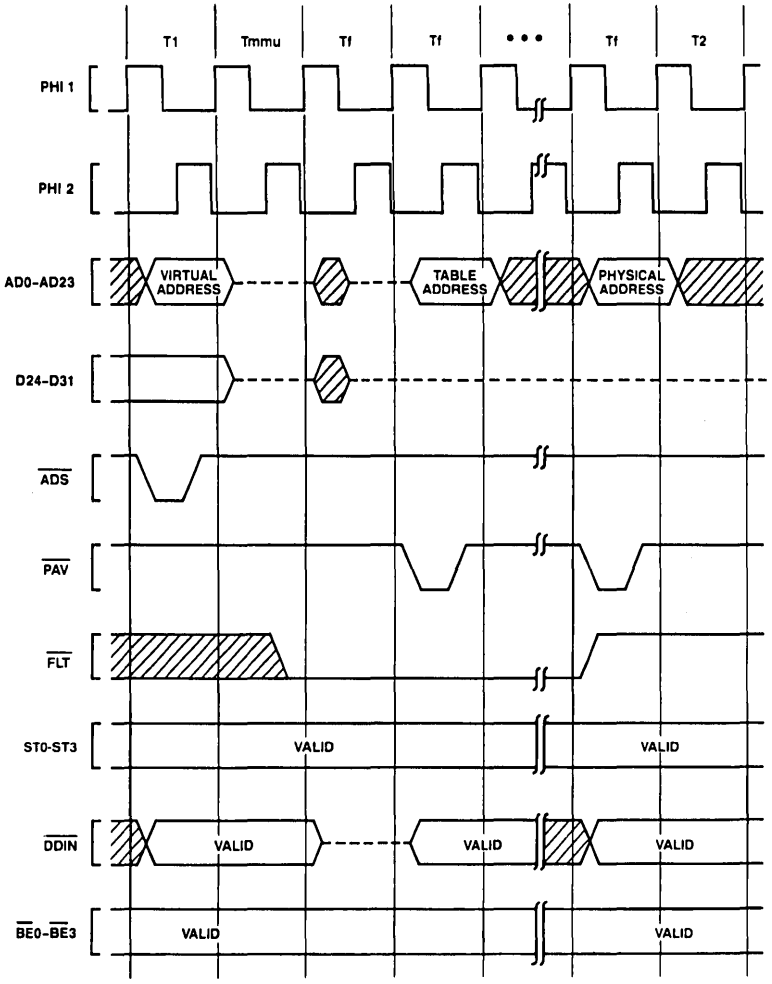
3.5.3 The $\overline{FLT}$ (Float) Pin

The $\overline{FLT}$ pin is used by the CPU for address translation support. Activating $\overline{FLT}$ during Tmmu causes the CPU to wait longer than Tmmu for address translation and validation. This feature is used occasionally by the NS32082 MMU in order to update its translation look-aside buffer (TLB) from page tables in memory, or to update certain status bits within them.

Figure 3-19 shows the effect of $\overline{FLT}$. Upon sampling $\overline{FLT}$ low, late in Tmmu, the CPU enters idle T-States (Tf) during which it:

1) Sets AD0–AD23, D24–D31 and $\overline{DDIN}$ to the TRI-STATE condition ("floating").
2) Suspends further internal processing of the current instruction. This ensures that the current instruction remains abortable with retry. (See $\overline{RST}/\overline{ABT}$ description, Sec. 3.5.4.)

Note that the AD0–AD23 pins may be briefly asserted during the first idle T-State. The above conditions remain in effect until $\overline{FLT}$ again goes high. See the Timing Specifications, Sec. 4.



TL/EE/9160-31

FIGURE 3-19. $\overline{FLT}$ Timing

3.0 Functional Description (Continued)

3.5.4 Aborting Bus Cycles

The $\overline{RST}/\overline{ABT}$ pin, apart from its Reset function (Sec. 3.3), also serves as the means to "abort", or cancel, a bus cycle and the instruction, if any, which initiated it. An Abort request is distinguished from a Reset in that the $\overline{RST}/\overline{ABT}$ pin is held active for only one clock cycle.

If $\overline{RST}/\overline{ABT}$ is pulled low during Tmmu or Tf, this signals that the cycle must be aborted. The CPU itself will enter T2 and then Ti, thereby terminating the cycle. Since it is the MMU $\overline{PAV}$ signal which triggers a physical cycle, the rest of the system remains unaware that a cycle was started.

The NS32082 MMU will abort a bus cycle for either of two reasons:

1) The CPU is attempting to access a virtual address which is not currently resident in physical memory. The referenced page must be brought into physical memory from mass storage to make it accessible to the CPU.
2) The CPU is attempting to perform an access which is not allowed by the protection level assigned to that page.

When a bus cycle is aborted by the MMU, the instruction that caused it to occur is also aborted in such a manner that it is guaranteed re-executable later. The information that is changed irrecoverably by such a partly-executed instruction does not affect its re-execution.

3.5.4.1 The Abort Interrupt

Upon aborting an instruction, the CPU immediately performs an interrupt through the ABT vector in the Interrupt Table (see Sec. 3.8). The Return Address pushed on the Interrupt Stack is the address of the aborted instruction, so that a Return from Trap (RETT) instruction will automatically retry it.

The one exception to this sequence occurs if the aborted bus cycle was an instruction prefetch. If so, it is not yet certain that the aborted prefetched code is to be executed. Instead of causing an interrupt, the CPU only aborts the bus cycle, and stops prefetching. If the information in the Instruction Queue runs out, meaning that the instruction will actually be executed, the ABT interrupt will occur, in effect aborting the instruction that was being fetched.

3.5.4.2 Hardware Considerations

In order to guarantee instruction retry, certain rules must be followed in applying an Abort to the CPU. These rules are followed by the NS32082 Memory Management Unit.

1) If $\overline{FLT}$ has not been applied to the CPU, the Abort pulse must occur during or before Tmmu. See the Timing Specifications, *Figure 4-22.*
2) If $\overline{FLT}$ has been applied to the CPU, the Abort pulse must be applied before the T-State in which $\overline{FLT}$ goes inactive. The CPU will not actually respond to the Abort command until $\overline{FLT}$ is removed. See *Figure 4-23.*
3) The Write half of a Read-Modify-Write operand access may not be aborted. The CPU guarantees that this will never be necessary for Memory Management functions by applying a special RMW status (Status Code 1011) during the Read half of the access. When the CPU presents RMW status, that cycle must be aborted if it would be illegal to write to any of the accessed addresses.

If $\overline{RST}/\overline{ABT}$ is pulsed at any time other than as indicated above, it will abort either the instruction currently under execution or the next instruction and will act as a very high-priority interrupt. However, the program that was running at the time is not guaranteed recoverable.

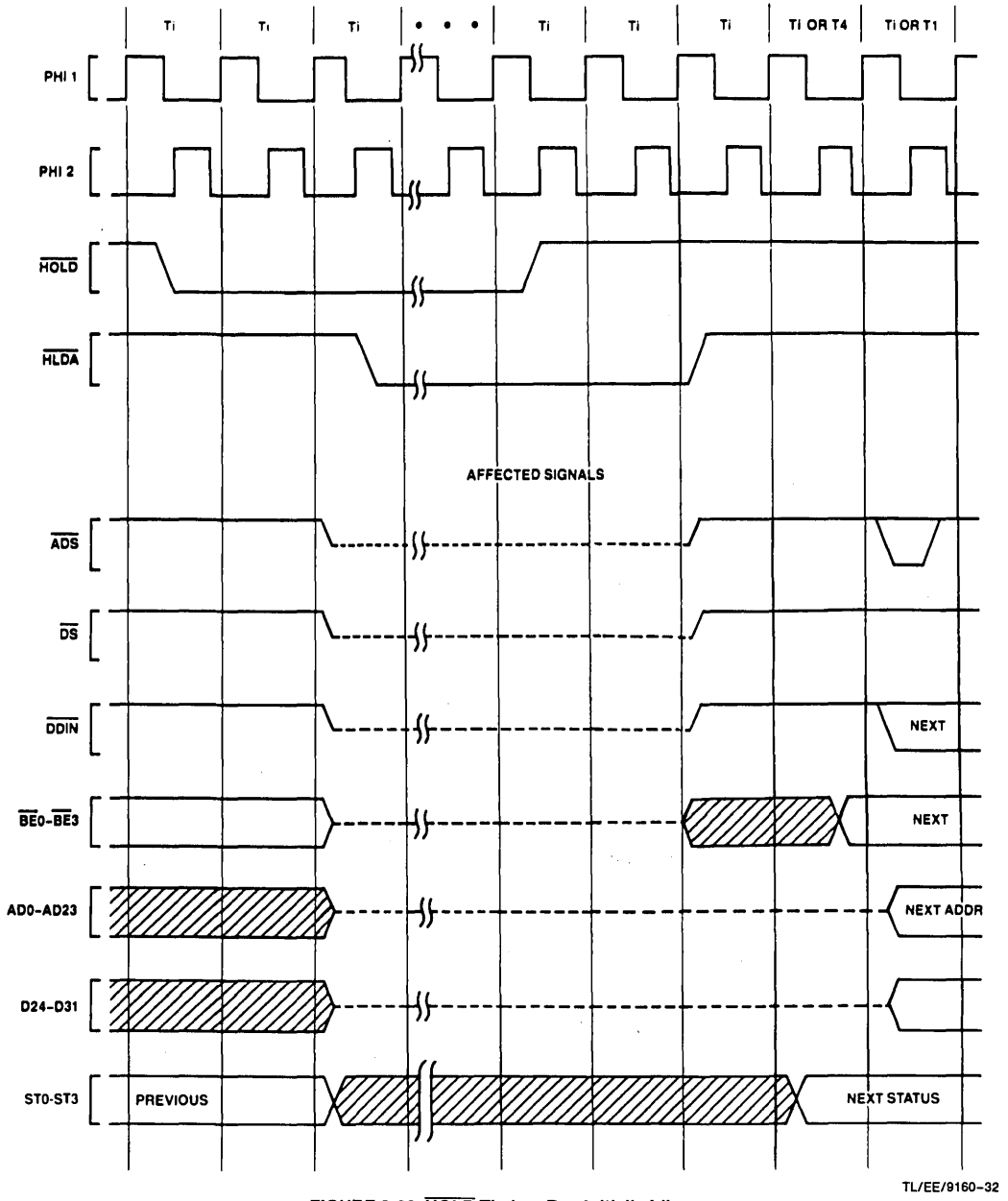
3.6 BUS ACCESS CONTROL

The NS32C032 CPU has the capability of relinquishing its access to the bus upon request from a DMA device or another CPU. This capability is implemented on the $\overline{HOLD}$ (Hold Request) and $\overline{HLDA}$ (Hold Acknowledge) pins. By asserting HOLD low, an external device requests access to the bus. On receipt of $\overline{HLDA}$ from the CPU, the device may perform bus cycles, as the CPU at this point has set the AD0–AD23, D24–D31, $\overline{ADS}$, $\overline{DDIN}$ and $\overline{BE0}$–$\overline{BE3}$ pins to the TRI-STATE condition. To return control of the bus to the CPU, the device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

How quickly the CPU releases the bus depends on whether it is idle on the bus at the time the $\overline{HOLD}$ request is made, as the CPU must always complete the current bus cycle. *Figure 3-20* shows the timing sequence when the CPU is idle. In this case, the CPU grants the bus during the immediately following clock cycle. *Figure 3-21* shows the sequence if the CPU is using the bus at the time that the $\overline{HOLD}$ request is made. If the request is made during or before the clock cycle shown (two clock cycles before T4), the CPU will release the bus during the clock cycle following T4. If the request occurs closer to T4, the CPU may already have decided to initiate another bus cycle. In that case it will not grant the bus until after the next T4 state. Note that this situation will also occur if the CPU is idle on the bus but has initiated a bus cycle internally.

In a Memory-Managed system, the $\overline{HLDA}$ signal is connected in a daisy-chain through the NS32082, so that the MMU can release the bus if it is using it.

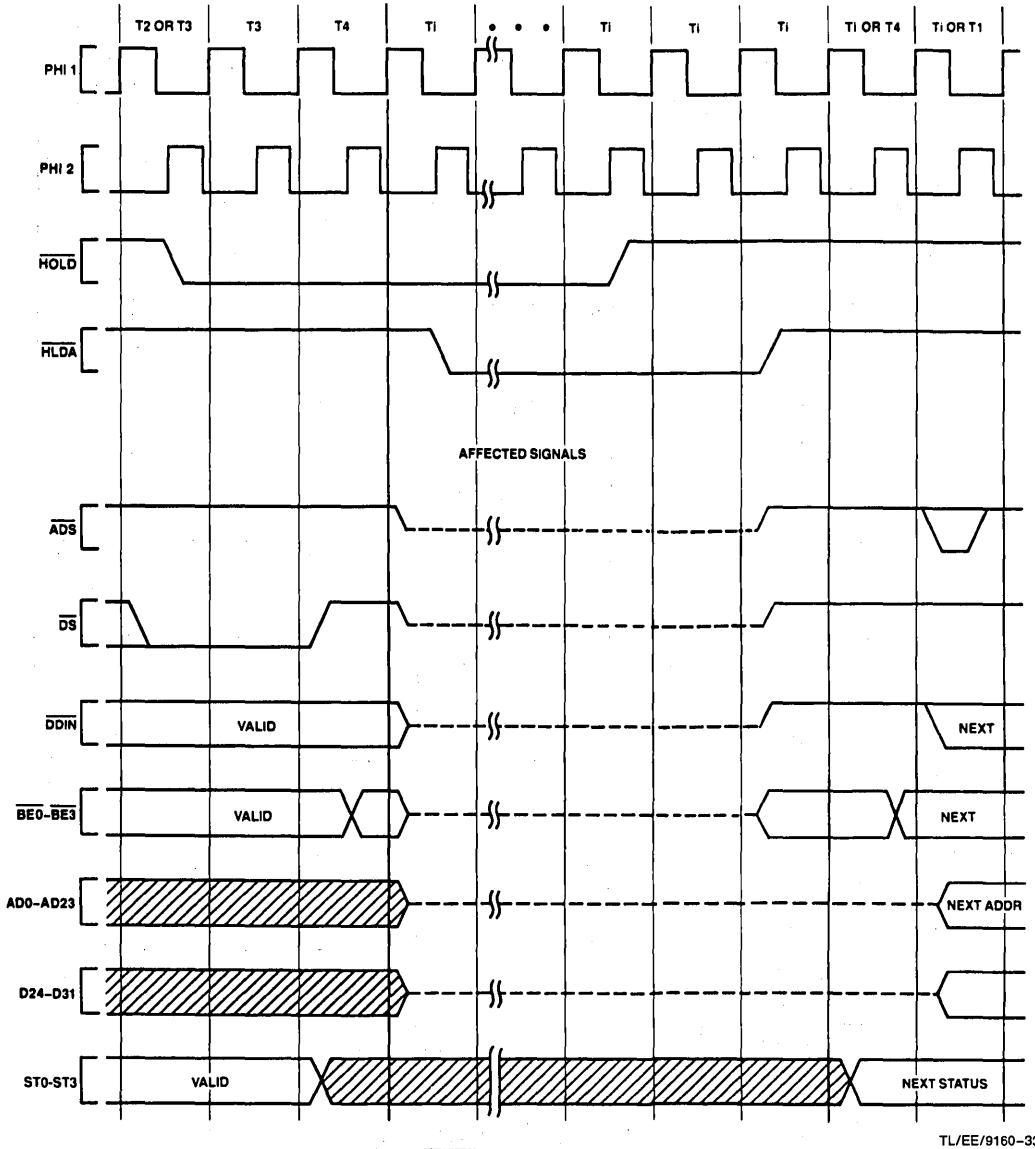
3.0 Functional Description (Continued)



TL/EE/9160–32

FIGURE 3-20. HOLD Timing, Bus Initially Idle

3.0 Functional Description (Continued)



TL/EE/9160–33

FIGURE 3-21. HOLD Timing, Bus Initially Not Idle

3.0 Functional Description (Continued)

3.7 INSTRUCTION STATUS

In addition to the four bits of Bus Cycle status (ST0–ST3), the NS32C032 CPU also presents Instruction Status information on three separate pins. These pins differ from ST0–ST3 in that they are synchronous to the CPU's internal instruction execution section rather than to its bus interface section.

$\overline{PFS}$ (Program Flow Status) is pulsed low as each instruction begins execution. It is intended for debugging purposes, and is used that way by the NS32082 Memory Management Unit.

U/$\overline{S}$ originates from the U bit of the Processor Status Register, and indicates whether the CPU is currently running in User or Supervisor mode. It is sampled by the MMU for mapping, protection, and debugging purposes. Although it is not synchronous to bus cycles, there are guarantees on its validity during any given bus cycle. See the Timing Specifications, *Figure 4-21.*

$\overline{ILO}$ (Interlocked Operation) is activated during an SBITI (Set Bit, Interlocked) or CBITI (Clear Bit, Interlocked) instruction. It is made available to external bus arbitration circuitry in order to allow these instructions to implement the semaphore primitive operations for multi-processor communication and resource sharing. As with the U/$\overline{S}$ pin, there are guarantees on its validity during the operand accesses performed by the instructions. See the Timing Specification Section, *Figure 4-19.*

3.8 NS32C032 INTERRUPT STRUCTURE

$\overline{INT}$, on which maskable interrupts may be requested,

$\overline{NMI}$, on which non-maskable interrupts may be requested, and

$\overline{RST}$/$\overline{ABT}$, which may be used to abort a bus cycle and any associated instruction. See Sec. 3.5.4.

In addition there is a set of internally-generated "traps" which cause interrupt service to be performed as a result either of exceptional conditions (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., the Supervisor Call instruction).

3.8.1 General Interrupt/Trap Sequence

Upon receipt of an interrupt or trap request, the CPU goes through three major steps:

1) Adjustment of Registers.

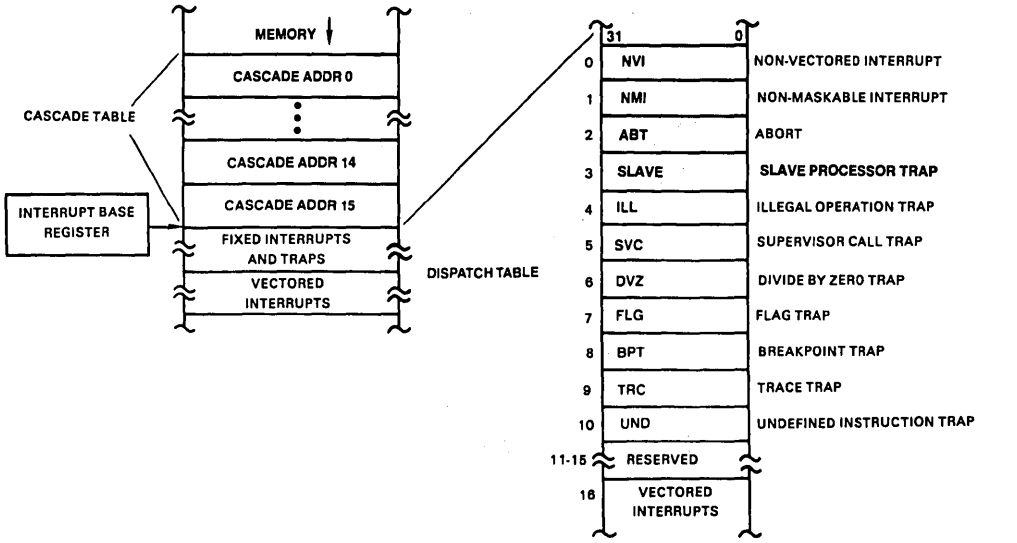
   Depending on the source of the interrupt or trap, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR) and the currently-selected Stack Pointer (SP). A copy of the PSR is made, and the PSR is then set to reflect Supervisor Mode and selection of the Interrupt Stack.

2) Vector Acquisition.

   A Vector is either obtained from the Data Bus or is supplied by default.

3) Service Call.

   The Vector is used as an index into the Interrupt Dispatch Table, whose base address is taken from the CPU Interrupt Base (INTBASE) Register. See *Figure 3-22.* A 32-bit External Procedure Descriptor is read from the table entry, and an External Procedure Call is performed using it. The MOD Register (16 bits) and Program Counter (32 bits) are pushed on the Interrupt Stack.



TL/EE/9160–34

FIGURE 3-22. Interrupt Dispatch and Cascade Tables

3.0 Functional Description (Continued)

This process is illustrated in *Figure 3-23*, from the viewpoint of the programmer.

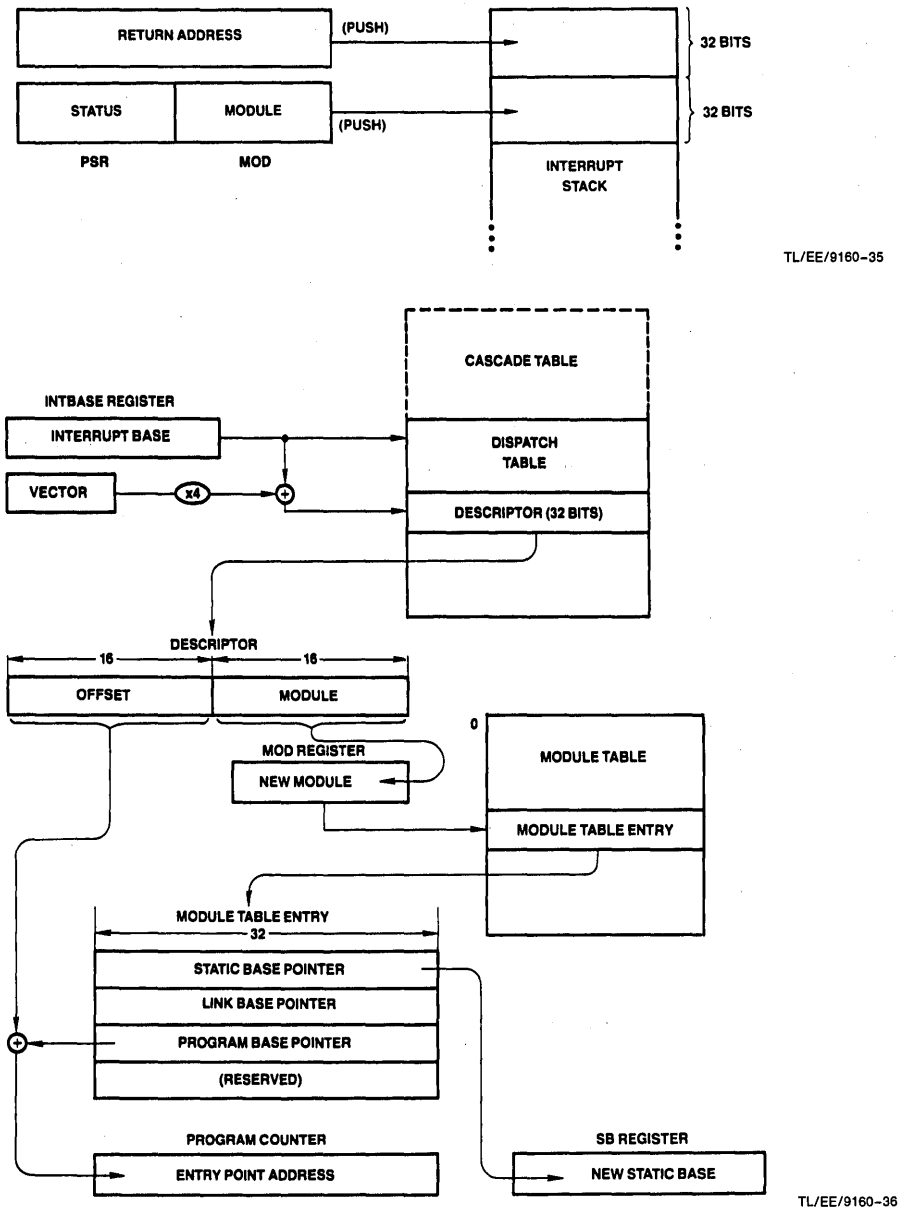


FIGURE 3-23. Interrupt/Trap Service Routine Calling Sequence

3.0 Functional Description (Continued)

3.8.2 Interrupt/Trap Return

To return control to an interrupted program, one of two instructions is used. The RETT (Return from Trap) instruction (*Figure 3-24*) restores the PSR, MOD, PC and SB registers to their previous contents and, since traps are often used deliberately as a call mechanism for Supervisor Mode procedures, it also discards a specified number of bytes from the original stack as surplus parameter space. RETT is used to return from any trap or interrupt except the Maskable Interrupt. For this, the RETI (Return from Interrupt) instruction is used, which also informs any external Interrupt Control Units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not pop parameters. See *Figure 3-25*.

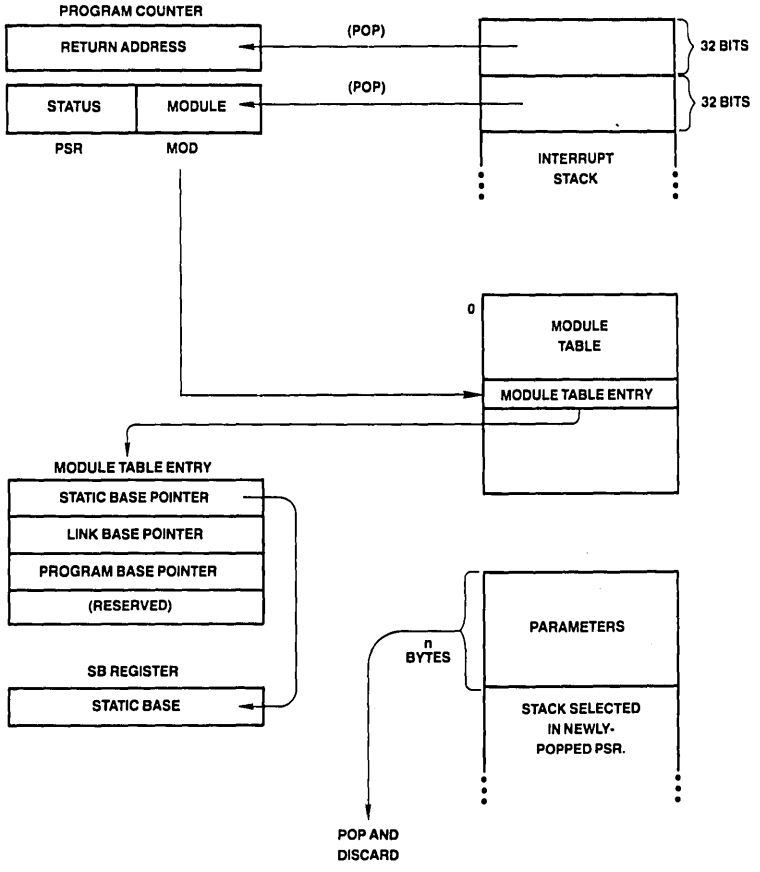
3.8.3 Maskable Interrupts (The $\overline{\text{INT}}$ Pin)

The $\overline{\text{INT}}$ pin is a level-sensitive input. A continuous low level is allowed for generating multiple interrupt requests. The input is maskable, and is therefore enabled to generate interrupt requests only while the Processor Status Register I bit is set. The I bit is automatically cleared during service of an $\overline{\text{INT}}$, $\overline{\text{NMI}}$ or Abort request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

The $\overline{\text{INT}}$ pin may be configured via the SETCFG instruction as either Non-Vectored (CFG Register bit I = C) or Vectored (bit I = 1).

3.8.3.1 Non-Vectored Mode

In the Non-Vectored mode, an interrupt request on the $\overline{\text{INT}}$ pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.



TL/EE/9160-37

FIGURE 3-24. Return from Trap (RETT n) Instruction Flow

3.0 Functional Description (Continued)

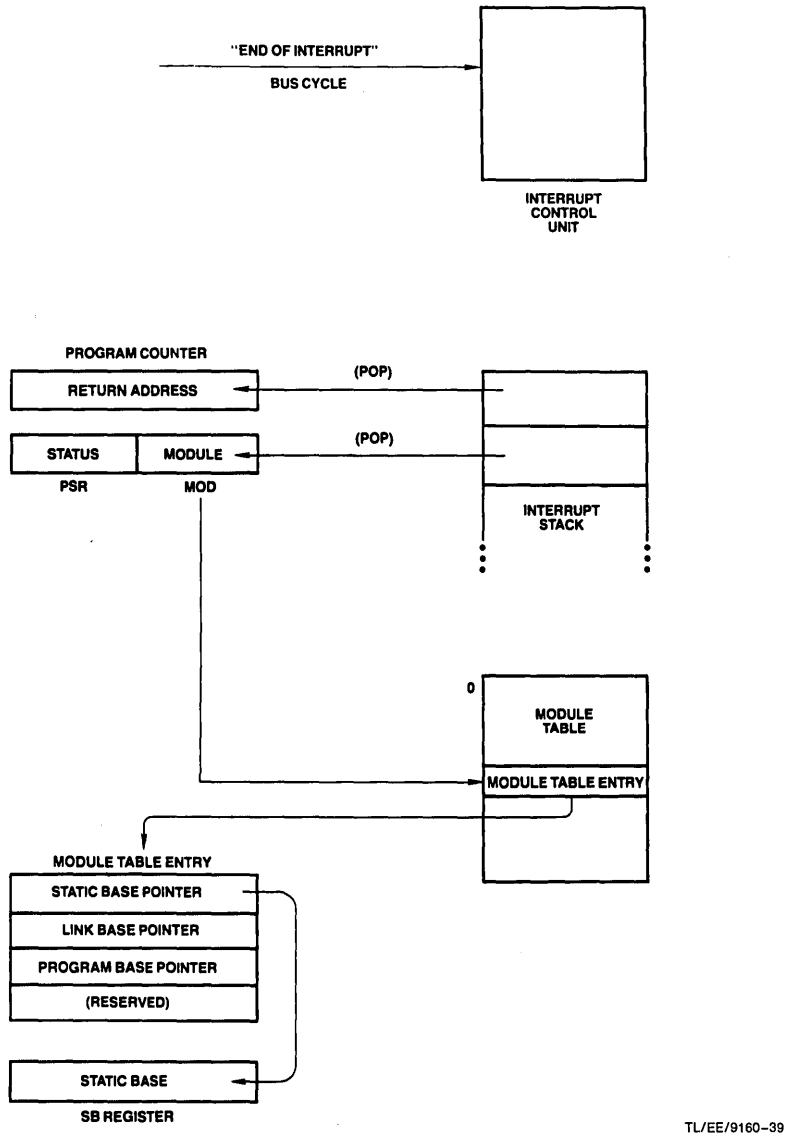


FIGURE 3-25. Return from Interrupt (RETI) Instruction Flow

3.0 Functional Description (Continued)

3.8.3.2 Vectored Mode: Non-Cascaded Case

In the Vectored mode, the CPU uses an Interrupt Control Unit (ICU) to prioritize up to 16 interrupt requests. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an "Interrupt Acknowledge, Master" bus cycle (Sec. 3.4.2) reading a vector value from the low-order byte of the Data Bus. This vector is then used as an index into the Dispatch Table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) instruction, which performs an End of Interrupt bus cycle, informing the ICU that it may re-prioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it needs also to inform a Cascaded ICU (see below).

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range of 0 through 127; that is, they must be positive numbers in eight bits. By providing a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU (see below).

3.8.3.3 Vectored Mode: Cascaded Case

In order to allow up to 256 levels of interrupt, provision is made both in the CPU and in the NS32202 Interrupt Control Unit (ICU) to transparently support cascading. *Figure 3-27*, shows a typical cascaded configuration. Note that the Interrupt output from a Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin.

In a system which uses cascading, two tasks must be performed upon initialization:

1) For each Cascaded ICU in the system, the Master ICU must be informed of the line number (0 to 15) on which it receives the cascaded requests.

2) A Cascade Table must be established in memory. The Cascade Table is located in a NEGATIVE direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

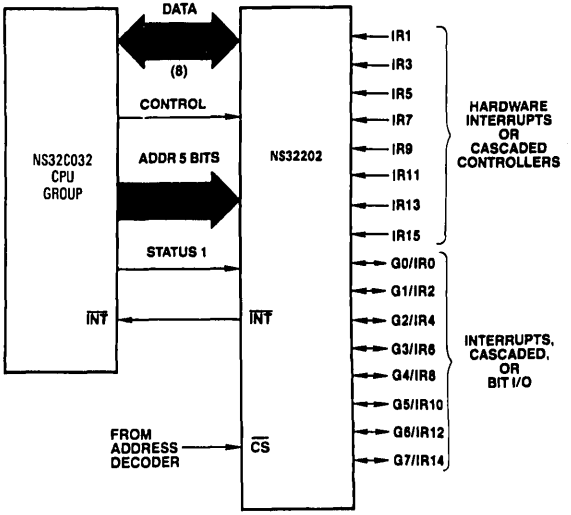
Figure 3-22 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0 to 15) and subtract 16 from it, giving an index in the range −16 to −1. Multiply this value by 4, and add the resulting negative number to the contents of the INTBASE Register. The 32-bit entry at this address must be set to the address of the Hardware Vector Register of the Cascaded ICU. This is referred to as the "Cascade Address."

Upon receipt of an interrupt request from a Cascaded ICU, the Master ICU interrupts the CPU and provides the negative Cascade Table index instead of a (positive) vector number. The CPU, seeing the negative value, uses it as an index into the Cascade Table and reads the Cascade Address from the referenced entry. Applying this address, the CPU performs an "Interrupt Acknowledge, Cascaded" bus cycle (Sec. 3.4.2), reading the final vector value. This vector is interpreted by the CPU as an unsigned byte, and can therefore be in the range of 0 through 255.

In returning from a Cascaded interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any Maskable Interrupt. The CPU performs an "End of Interrupt, Master" bus cycle (Sec. 3.4.2), whereupon the Master ICU again provides the negative Cascade Table index. The CPU, seeing a negative value, uses it to find the corresponding Cascade Address from the Cascade Table. Applying this address, it performs an "End of Interrupt, Cascaded" bus cycle (Sec. 3.4.2), informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the Interrupt Mask Register of the Interrupt Controller.

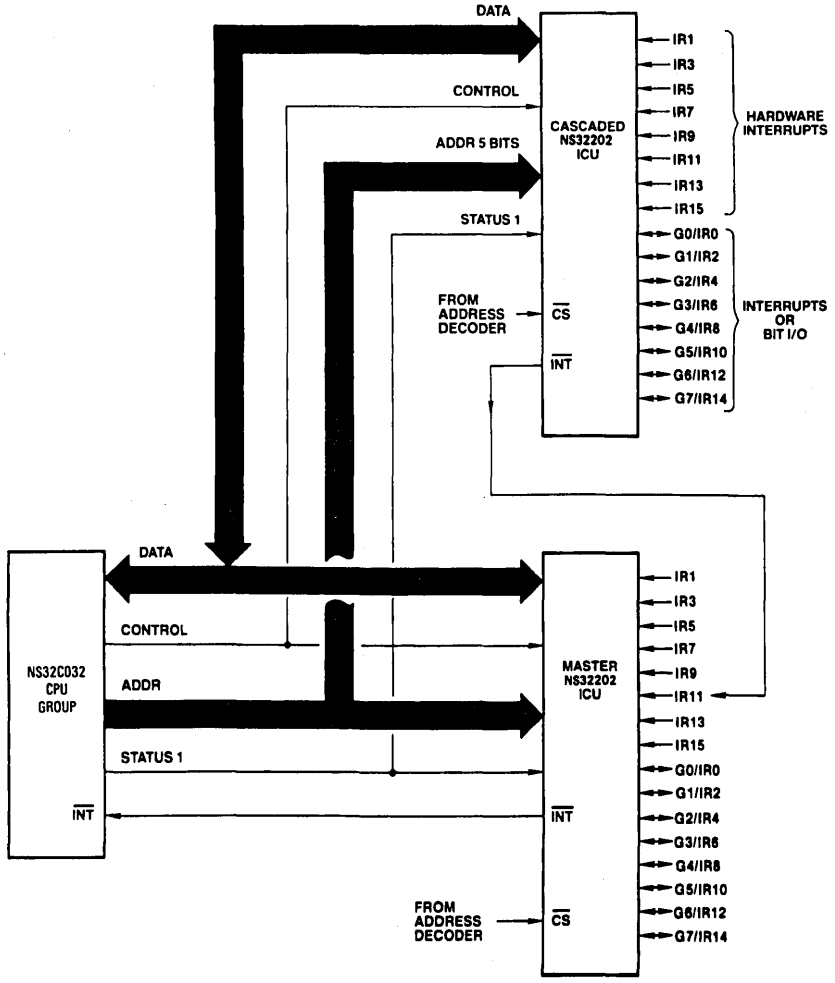
However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the $\overline{INT}$ line before the ICU deasserted it. This could cause the ICU to provide an invalid vector. To avoid this problem the above operation should be performed with the CPU interrupt disabled.



TL/EE/9160–40

FIGURE 3-26. Interrupt Control Unit Connections (16 Levels)

3.0 Functional Description (Continued)



TL/EE/9160-41

FIGURE 3-27. Cascaded Interrupt Control Unit Connections

3.8.4 Non-Maskable Interrupt (The $\overline{\text{NMI}}$ Pin)

The Non-Maskable Interrupt is triggered whenever a falling edge is detected on the $\overline{\text{NMI}}$ pin. The CPU performs an "Interrupt Acknowledge, Master" bus cycle (Sec. 3.4.2) when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is FFFF00_{16}. The vector value used for the Non-Maskable Interrupt is taken as 1, regardless of the value read from the bus.

The service procedure returns from the Non-Maskable Interrupt using the Return from Trap (RETT) instruction. No special bus cycles occur on return.

For the full sequence of events in processing the Non-Maskable Interrupt, see Sec. 3.8.7.1.

3.8.5 Traps

A trap is an internally-generated interrupt request caused as a direct and immediate result of the execution of an instruction. The Return Address pushed by any trap except Trap (TRC) is the address of the first byte of the instruction during which the trap occurred. Traps do not disable interrupts, as they are not associated with external events. Traps recognized by the NS32C032 CPU are:

Trap (SLAVE): An exceptional condition was detected by the Floating Point Unit or another Slave Processor during the execution of a Slave Instruction. This trap is requested via the Status Word returned as part of the Slave Processor Protocol (Sec. 3.9.1).

3.0 Functional Description (Continued)

Trap (ILL): Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR bit U = 1).

Trap (SVC): The Supervisor Call (SVC) instruction was executed.

Trap (DVZ): An attempt was made to divide an integer by zero. (The FPU trap is used for Floating Point division by zero.)

Trap (FLG): The FLAG instruction detected a "1" in the CPU PSR F bit.

Trap (BPT): The Breakpoint (BPT) instruction was executed.

Trap (TRC): The instruction just completed is being traced. See below.

Trap (UND): An undefined opcode was encountered by the CPU.

A special case is the Trace Trap (TRC), which is enabled by setting the T bit in the Processor Status Register (PSR). At the beginning of each instruction, the T bit is copied into the PSR P (Trace "Pending") bit. If the P bit is set at the end of an instruction, then the Trace Trap is activated. If any other trap or interrupt request is made during a traced instruction, its entire service procedure is allowed to complete before the Trace Trap occurs. Each interrupt and trap sequence handles the P bit for proper tracing, guaranteeing one and only one Trace Trap per instruction, and guaranteeing that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

3.8.6 Prioritization

The NS32016 CPU internally prioritizes simultaneous interrupt and trap requests as follows:

1) Traps other than Trace (Highest priority)
2) Abort
3) Non-Maskable Interrupt
4) Maskable Interrupts
5) Trace Trap (Lowest priority)

3.8.7 Interrupt/Trap Sequences: Detailed Flow

For purposes of the following detailed discussion of interrupt and trap service sequences, a single sequence called "Service" is defined in *Figure 3-28*. Upon detecting any interrupt request or trap condition, the CPU first performs a sequence dependent upon the type of interrupt or trap. This sequence will include pushing the Processor Status Register and establishing a Vector and a Return Address. The CPU then performs the Service sequence.

For the sequence followed in processing either Maskable or Non-Maskable interrupts (on the $\overline{INT}$ or $\overline{NMI}$ pins, respectively), see Sec. 3.8.7.1 For Abort Interrupts, see Sec. 3.8.7.4. For the Trace Trap, see Sec. 3.8.7.3, and for all other traps see Sec. 3.8.7.2.

3.8.7.1 Maskable/Non-Maskable Interrupt Sequence

This sequence is performed by the CPU when the $\overline{NMI}$ pin receives a falling edge, or the $\overline{INT}$ pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or, in the case of the String instructions, at the next interruptible point during its execution.

1. If a String instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P bit.
   b. Set "Return Address" to the address of the first byte of the interrupted instruction.
   Otherwise, set "Return Address" to the address of the next instruction.
2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, T, P and I.
3. If the interrupt is Non-Maskable:
   a. Read a byte from address $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master, Sec. 3.4.2). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.
4. If the interrupt is Non-Vectored:
   a. Read a byte from address $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Sec. 3.4.2). Discard the byte read.
   b. Set "Vector" to 0.
   c. Go to Step 8.
5. Here the interrupt is Vectored. Read "Byte" from address $FFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Sec. 3.4.2).
6. If "Byte" ≥ 0, then set "Vector" to "Byte" and go to Step 8.
7. If "Byte" is in the range −16 through −1, then the interrupt source is Cascaded. (More negative values are reserved for future use.) Perform the following:
   a. Read the 32-bit Cascade Address from memory. The address is calculated as INTBASE + 4* Byte.
   b. Read "Vector," applying the Cascade Address just read and Status Code 0101 (Interrupt Acknowledge, Cascaded: Sec. 3.4.2).
8. Push the PSR copy (from Step 2) onto the Interrupt Stack as a 16-bit value.
9. Perform Service (Vector, Return Address), *Figure 3-28*.

Service (Vector, Return Address):

1) Read the 32-bit External Procedure Descriptor from the Interrupt Dispatch Table: address is Vector* 4 + INTBASE Register contents.
2) Move the Module field of the Descriptor into the MOD Register.
3) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.
4) Read the Program Base pointer from memory address MOD + 8, and add to it the Offset field from the Descriptor, placing the result in the Program Counter.
5) Flush queue: Non-sequentially fetch first instruction of Interrupt routine.
6) Push MOD Register into the Interrupt Stack as a 16-bit value. (The PSR has already been pushed as a 16-bit value.)
7) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.

FIGURE 3-28. Service Sequence

Invoked during all interrupt/trap sequences.

3.0 Functional Description (Continued)

3.8.7.2 Trap Sequence: Traps Other Than Trace

1) Restore the currently selected Stack Pointer and the Processor Status Register to their original values at the start of the trapped instruction.
2) Set "Vector" to the value corresponding to the trap type.
   - SLAVE: Vector = 3.
   - ILL: Vector = 4.
   - SVC: Vector = 5.
   - DVZ: Vector = 6.
   - FLG: Vector = 7.
   - BPT: Vector = 8.
   - UND: Vector = 10.
3) Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, P and T.
4) Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5) Set "Return Address" to the address of the first byte of the trapped instruction.
6) Perform Service (Vector, Return Address), *Figure 3-28.*

3.8.7.3 Trace Trap Sequence

1) In the Processor Status Register (PSR), clear the P bit.
2) Copy the PSR into a temporary register, then clear PSR bits S, U and T.
3) Push the PSR copy onto the Interrupt Stack as a 16-bit value.
4) Set "Vector" to 9.
5) Set "Return Address" to the address of the next instruction.
6) Perform Service (Vector, Return Address), *Figure 3-28.*

3.8.7.4 Abort Sequence

1) Restore the currently selected Stack Pointer to its original contents at the beginning of the aborted instruction.
2) Clear the PSR P bit.
3) Copy the PSR into a temporary register, then clear PSR bits S, U, T and I.
4) Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5) Set "Vector" to 2.
6) Set "Return Address" to the address of the first byte of the aborted instruction.
7) Perform Service (Vector, Return Address), *Figure 3-28.*

3.9 SLAVE PROCESSOR INSTRUCTIONS

The NS32C032 CPU recognizes three groups of instructions being executable by external Slave Processor:

Floating Point Instruction Set
Memory Management Instruction Set
Custom Instruction Set

Each Slave Instruction Set is validated by a bit in the Configuration Register (Sec. 2.1.3). Any Slave Instruction which does not have its corresponding Configuration Register bit set will trap as undefined, without any Slave Processor communication attempted by the CPU. This allows software simulation of a non-existent Slave Processor.

3.9.1 Slave Processor Protocol

Slave Processor instructions have a three-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word. The ID Byte has three functions:

1) It identifies the instruction as being a Slave Processor instruction.
2) It specifies which Slave Processor will execute it.
3) It determines the format of the following Operation Word of the instruction.

Upon receiving a Slave Processor instruction, the CPU initiates the sequence outlined in *Figure 3-29.* While applying Status Code 1111 (Broadcast ID, Sec. 3.4.2), the CPU transfers the ID Byte on the least-significant byte of the Data Bus (AD0–AD7). All Slave Processors input this byte and decode it. The Slave Processor selected by the ID Byte is activated, and from this point the CPU is communicating only with it. If any other slave protocol was in progress (e.g., an aborted Slave instruction), this transfer cancels it.

The CPU next sends the Operation Word while applying Status Code 1101 (Transfer Slave Operand, Sec. 3.4.2). Upon receiving it, the Slave Processor decodes it, and at this point both the CPU and the Slave Processor are aware of the number of operands to be transferred and their sizes. The operation Word is swapped on the Data Bus, that is, bits 0–7 appear on pins AD8–AD15 and bits 8–15 appear on pins AD0–AD7.

Using the Address Mode fields within the Operation Word, the CPU starts fetching operand and issuing them to the Slave Processor. To do so, it references any Addressing Mode extensions which may be appended to the Slave Processor instruction. Since the CPU is solely responsible

Status Combinations:

Send ID (ID): Code 1111
Xfer Operand (OP): Code 1101
Read Status (ST): Code 1110

Step	Status	Action
1	ID	CPU Send ID Byte.
2	OP	CPU Sends Operaton Word.
3	OP	CPY Sends Required Operands
4	—	Slave Starts Execution. CPU Pre-fetches.
5	—	Slave Pulses $\overline{SPC}$ Low.
6	ST	CPU Reads Status Word. (Trap? Alter Flags?)
7	OP	CPU Reads Results (If Any).

FIGURE 3-29. Slave Processor Protocol

3.0 Functional Description (Continued)

for memory accesses, these extensions are not sent to the Slave processor. The Status Code applied is 1101 (Transfer Slave Processor Operand, Sec. 3.4.2).

After the CPU has issued the last operand, the Slave Processor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low. To allow for this, and for the Address Translation strap function, $\overline{AT}/\overline{SPC}$ is normally held high only by an internal pull-up device of approximately 5 kΩ.

While the Slave Processor is executing the instruction, the CPU is free to prefetch instructions into its queue. If it fills the queue before the Slave Processor finishes, the CPU will wait, applying Status Code 0011 (Waiting for Slave, Sec. 3.4.2).

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a Status Word from the Slave Processor, applying Status Code 1110 (Read Slave Status, Sec. 3.4.2). This word has the format shown in *Figure 3-30*. If the Q bit ("Quit", Bit 0) is set, this indicates that an error was detected by the Slave Processor. The CPU will not continue the protocol, but will immediately trap through the Slave vector in the Interrupt Table. Certain Slave Processor instructions cause CPU PSR bits to be loaded from the Status Word.

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The Read cycles from the Slave Processor are performed by the CPU while applying Status Code 1101 (Transfer Slave Operand, Sec. 3.4.2).

An exception to the protocol above is the LMR (Load Memory Management Register) instruction, and a corresponding Custom Slave instruction (LCR: Load Custom Register). In executing these instructions, the protocol ends after the CPU has issued the last operand. The CPU does not wait for an acknowledgement from the Slave Processor, and it does not read status.

3.9.2 Floating Point Instructions

Table 3-4 gives the protocols followed for each Floating Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Appendix A.

The Operand class columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see Instruction Set Reference Manual).

The Operand Issued columns show the sizes of the operands issued to the Floating Point Unit by the CPU. "D" indicates a 32-bit Double Word. "i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). "f" indicates that the instruction specifies a Floating Point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

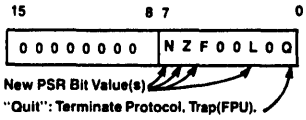
The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the Slave Processor Status Word (*Figure 3-30*).

TABLE 3-4
Floating Point Instruction Protocols.

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none

Note:
D = Double Word
i = Integer size (B,W,D) specified in mnemonic.
f = Floating Point type (F,L) specified in mnemonic.
N/A = Not Applicable to this instruction.

3.0 Functional Description (Continued)



TL/EE/9160–42

FIGURE 3-30. Slave Processor Status Word Format

Any operand indicated as being of type "f" will not cause a transfer if the Register addressing mode is specified. This is because the Floating Point Registers are physically on the Floating Point Unit and are therefore available without CPU assistance.

3.9.3 Memory Management Instructions

Table 3-5 gives the protocols for Memory Management instructions. Encodings for these instructions may be found in Appendix A.

In executing the RDVAL and WRVAL instructions, the CPU calculates and issues the 32-bit Effective Address of the single operand. The CPU then performs a single-byte Read cycle from that address, allowing the MMU to safely abort the instruction if the necessary information is not currently in physical memory. Upon seeing the memory cycle complete, the MMU continues the protocol, and returns the validation result in the F bit of the Slave Status Word.

The size of a Memory Management operand is always a 32-bit Double Word. For further details of the Memory Management Instruction set, see the Instruction Set Reference Manual and the NS32082 MMU Data Sheet.

TABLE 3-5

Memory Management Instruction Protocols.

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
RDVAL*	addr	N/A	D	N/A	N/A	F
WRVAL*	addr	N/A	D	N/A	N/A	F
LMR*	read.D	N/A	D	N/A	N/A	none
SMR*	write.D	N/A	N/A	N/A	D to Op. 1	none

Note:

In the RDVAL and WRVAL instructions, the CPU issues the address as a Double Word, and performs a single-byte Read cycle from that memory address. For details, see the Instruction Set Reference Manual and the NS32082 Memory Management Unit Data Sheet.

D = Double Word

* = Privileged Instruction: will trap if CPU is in User Mode.

N/A = Not Applicable to this instruction.

3.0 Functional Description (Continued)

3.9.4 Custom Slave Instructions

Provided in the NS32C032 is the capability of communicating with a user-defined, "Custom" Slave Processor. The instruction set provided for a Custom Slave Processor defines the instruction formats, the operand classes and the communication protocol. Left to the user are the interpretations of the Op Code fields, the programming model of the Custom Slave and the actual types of data transferred. The protocol specifies only the size of an operand, not its data type.

Table 3-6 lists the relevant information for the Custom Slave instruction set. The designation "c" is used to represent an operand which can be a 32-bit ("D") or 64-bit ("Q") quantity in any format; the size is determined by the suffix on the mnemonic. Similarly, an "i" indicates an integer size (Byte, Word, Double Word) selected by the corresponding mnemonic suffix.

Any operand indicated as being of type "c" will not cause a transfer if the register addressing mode is specified. It is assumed in this case that the slave processor is already holding the operand internally.

For the instruction encodings, see Appendix A.

TABLE 3-6
Custom Slave Instruction Protocols.

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
CCAL0c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL1c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL2c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL3c	read.c	rmw.c	c	c	c to Op. 2	none
CMOV0c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV1c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV2c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV3c	read.c	write.c	c	N/A	c to Op.2	none
CCMP0c	read.c	read.c	c	c	N/A	N,Z,L
CCMP1c	read.c	read.c	c	c	N/A	N,Z,L
CCV0ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV1ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV2ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV3ic	read.i	write.c	i	N/A	c to Op. 2	none
CCV4DQ	read.D	write.Q	D	N/A	Q to Op. 2	none
CCV5QD	read.Q	write.D	Q	N/A	D to Op. 2	none
LCSR	read.D	N/A	D	N/A	N/A	none
SCSR	N/A	write.D	N/A	N/A	D to OP. 2	none
CATST0*	addr	N/A	D	N/A	N/A	F
CATST1*	addr	N/A	D	N/A	N/A	F
LCR*	read.D	N/A	D	N/A	N/A	none
SCR*	write.D	N/A	N/A	N/A	D to Op.1	none

Note:
D = Double Word
i = Integer size (B,W,D) specified in mnemonic.
c = Custom size (D:32 bits or Q:64 bits) specified in mnemonic.
* = Privileged instruction: will trap if CPU is in User Mode.
N/A = Not Applicable to this instruction.

4.0 Device Specifications

4.1 NS32C032 PIN DESCRIPTIONS

The following is a brief description of all NS32C032 pins. The descriptions reference portions of the Functional Description. Sec. 3.

Unless otherwise indicated reserved pins should be left open.

4.1.1 Supplies

Logic Power ($V_{CCL1, 2}$): +5V positive supply.

Buffers Power ($V_{CCB1, 2}$): +5V positive supply.

Logic Ground (GNDL1, GNDL2): Ground reference for on-chip logic.

Buffer Grounds (GNDB1, GNDB2, GNDB3): Ground references for on-chip drivers.

4.1.2 Input Signals

Clocks (PHI1, PHI2): Two-phase clocking signals. Sec. 3.2.

Ready (RDY): Active high. While RDY is inactive, the CPU extends the current bus cycle to provide for a slower memory or peripheral reference. Upon detecting RDY active, the CPU terminates the bus cycle. Sec. 3.4.1.

Hold Request ($\overline{HOLD}$): Active low. Causes the CPU to release the bus for DMA or multiprocessing purposes. Sec. 3.6.

Note 1: $\overline{HOLD}$ must not be asserted until $\overline{HLDA}$ from a previous $\overline{HOLD}/\overline{HLDA}$ sequence is deasserted.

Note 2: If the $\overline{HOLD}$ signal is generated asynchronously, it's set up and hold times may be violated.

In this case it is recommended to synchronize it with CTTL to minimize the possibility of metastable states.

The CPU provides only one synchronization stage to minimize the $\overline{HLDA}$ latency. This is to avoid speed degradations in cases of heavy $\overline{HOLD}$ activity (i.e., DMA controller cycles interleaved with CPU cycles.)

Interrupt ($\overline{INT}$): Active low. Maskable Interrupt request. Sec. 3.8.

Non-Maskable Interrupt ($\overline{NMI}$): Active low. Non-Maskable Interrupt request. Sec. 3.8.

Reset/Abort ($\overline{RST}/\overline{ABT}$): Active low. If held active for one clock cycle and released, this pin causes an Abort Command, Sec. 3.5.4. If held longer, it initiates a Reset. Sec. 3.3.

4.1.3 Output Signals

Address Strobe ($\overline{ADS}$): Active low. Controls address latches: indicates start of a bus cycle. Sec. 3.4.

Data Direction In ($\overline{DDIN}$): Active low. Status signal indicating direction of data transfer during a bus cycle. Sec. 3.4.

Byte Enable ($\overline{BE0}$–$\overline{BE3}$): Active low. Four control signals enabling data transfers on individual bus bytes. Sec. 3.4.3.

Status (ST0–ST3): Bus cycle status code, ST0 least significant. Sec. 3.4.2. Encodings are:

0000 — Idle: CPU Inactive on Bus.
0001 — Idle: WAIT Instruction.
0010 — (Reserved).
0011 — Idle: Waiting for Slave.
0100 — Interrupt Acknowledge, Master.
0101 — Interrupt Acknowledge, Cascaded.
0110 — End of Interrupt, Master.
0111 — End of Interrupt, Cascaded.
1000 — Sequential Instruction Fetch.
1001 — Non-Sequential Instruction Fetch.
1010 — Data Transfer.
1011 — Read Read-Modify-Write Operand.
1100 — Read for Effective Address.
1101 — Transfer Slave Operand.
1110 — Read Slave Status Word.
1111 — Broadcast Slave ID.

Hold Acknowledge ($\overline{HLDA}$): Active low. Applied by the CPU in response to $\overline{HOLD}$ input, indicating that the bus has been released for DMA or multiprocessing purposes. Sec. 3.6.

User/Supervisor (U/$\overline{S}$): User or Supervisor Mode status. Sec. 3.7. High state indicates User Mode, low indicates Supervisor Mode. Sec. 3.7.

Interlocked Operation ($\overline{ILO}$): Active low. Indicates that an interlocked instruction is being executed. Sec. 3.7.

Program Flow Status ($\overline{PFS}$): Active low. Pulse indicates beginning of an instruction execution. Sec. 3.7.

4.1.4 Input-Output Signals

Address/Data 0–23 (AD0–AD23): Multiplexed Address/Data information. Bit 0 is the least significant bit of each. Sec. 3.4.

Data Bits 24–31 (D24–D31): The high order 8 bits of the data bus.

Address Translation/Slave Processor Control ($\overline{AT}$/$\overline{SPC}$): Active low. Used by the CPU as the data strobe output for Slave Processor transfers; used by Slave Processors to acknowledge completion of a slave instruction. Sec. 3.4.6; Sec. 3.9. Sampled on the rising edge of Reset pulse as Address Translation Strap. Sec. 3.5.1.

In non-memory-managed systems, this pin should be pulled-up to V_{CC} through a 10 kΩ resistor.

Data Strobe/Float ($\overline{DS}/\overline{FLT}$): Active low. Data Strobe output, Sec. 3.4, or Float Command input, Sec. 3.5.3. Pin function is selected on $\overline{AT}/\overline{SPC}$ pin, Sec. 3.5.1.

4.0 Device Specifications (Continued)

4.2 ABSOLUTE MAXIMUM RATINGS

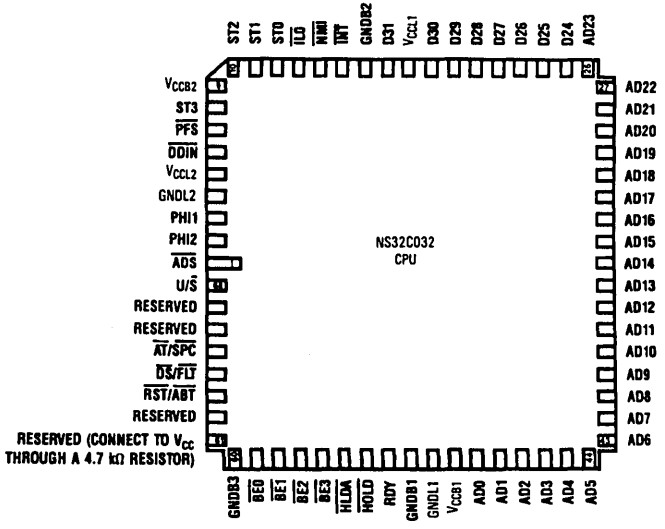
If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	1.5 Watt

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_A = 0° to +70°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		V_{CC} +0.5	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{CH}	High Level Clock Voltage	PHI1, PHI2 pins only	0.85 V_{CC}		V_{CC} +0.5	V
V_{CL}	Low Level Clock Voltage	PHI1, PHI2 pins only	−0.5		0.10 V_{CC}	V
V_{CRT}	Clock Input Ringing Tolerance	PHI1, PHI2 pins only	−0.5		0.6	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	0.85 V_{CC}			V
V_{OL}	Low Level Output Voltage	I_{OL} = 2 mA			0.10 V_{CC}	V
I_{ILS}	$\overline{AT}/\overline{SPC}$ Input Current (low)	V_{IN} = 0.4V, $\overline{AT}/\overline{SPC}$ in input mode	0.05		1.0	mA
I_I	Input Load Current	0 ≤ V_{IN} ≤ V_{CC}, All inputs except PHI1, PHI2, $\overline{AT}/\overline{SPC}$	−20		20	μA
I_L	Leakage Current Output and I/O Pins in TRI-STATE/Input Mode	0.4 ≤ V_{OUT} ≤ V_{CC}	−20		20	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C		70	100	mA



TL/EE/9160-2

Bottom View

FIGURE 4-1. NS32C032 Connection Diagram

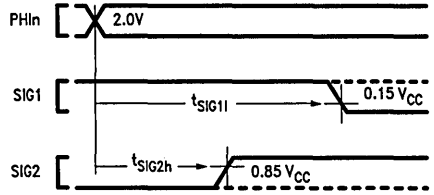
Order Number NS32C032-10E, NS32C032-15E, NS32C032-10V or NS32C032-15V
See NS Package Number E68B or V68A

4.0 Device Specifications (Continued)

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 2.0V on the rising or falling edges of the clock phases PHI1 and PHI2; to 15% or 85% of V_{CC} on all the CMOS output signals, and to 0.8V or 2.0V on all the TTL input signals as illustrated in *Figures 4-2* and *4-3* unless specifically stated otherwise.



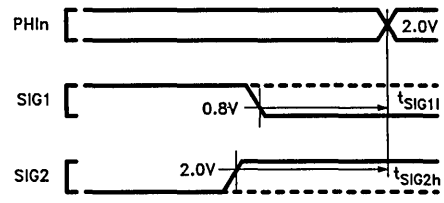
TL/EE/9160–43

FIGURE 4-2. Timing Specification Standard (CMOS Output Signals)

ABBREVIATIONS:

L.E. — leading edge R.E. — rising edge

T.E. — trailing edge F.E. — falling edge



TL/EE/9160–44

FIGURE 4-3. Timing Specification Standard (TTL Input Signals)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32C032-10, NS32C032-15

Maximum times assume capacitive loading of 100 pF.

Name	Figure	Description	Reference/Conditions	NS32C032-10		NS32C032-15		Units
				Min	Max	Min	Max	
t_{ALv}	4-4	Address bits 0–23 valid	after R.E., PHI1 T1		40		35	ns
t_{ALh}	4-4	Address bits 0–23 hold	after R.E., PHI1 Tmmu or T2	5		5		ns
t_{Dv}	4-4	Data valid (write cycle)	after R.E., PHI1 T2		50		35	ns
t_{Dh}	4-4	Data hold (write cycle)	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{ALADSs}	4-5	Address bits 0–23 setup	before $\overline{ADS}$ T.E.	25		20		ns
t_{ALADSh}	4-10	Address bits 0–23 hold	after $\overline{ADS}$ T.E.	15		10		ns
t_{ALf}	4-5	Address bits 0–23 floating (no MMU)	after R.E., PHI1 T2		25		20	ns
t_{ADf}	4-5	Data bits D24–D31 floating (no MMU)	after R.E., PHI1 T2		25		20	ns
t_{ALMf}	4-9	Address bits 0–23 floating (with MMU)	after R.E., PHI1 Tmmu		25		20	ns
t_{ADMf}	4-9	Data bits 21–31 floating (with MMU)	after R.E., PHI1 Tmmu		25		20	ns
t_{BEv}	4-4	$\overline{BEn}$ signals valid	after R.E., PHI2 T4		60		45	ns
t_{BEh}	4-4	$\overline{BEn}$ signals hold	after R.E., PHI2 T4 or Ti	0		0		ns
t_{STv}	4-4	Status (ST0–ST3) valid	after R.E., PHI1 T4 (before T1, see note)		45		35	ns
t_{STh}	4-4	Status (ST0–ST3) hold	after R.E., PHI1 T4 (after T1)	0		0		ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32C032-8, NS32C032-10 (Continued)

Name	Figure	Description	Reference/ Conditions	NS32C032-10		NS32C032-15		Units
				Min	Max	Min	Max	
t_{DDINv}	4-5	$\overline{DDIN}$ signal valid	after R.E., PHI1 T1		50		35	ns
t_{DDINh}	4-5	$\overline{DDIN}$ signal hold	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{ADSa}	4-4	$\overline{ADS}$ signal active (low)	after R.E., PHI1 T1		35		26	ns
t_{ADSia}	4-4	$\overline{ADS}$ signal inactive	after R.E., PHI2 T1		40		30	ns
t_{ADSw}	4-4	$\overline{ADS}$ pulse width	at 15% V_{CC} (both edges)	30		25		ns
t_{DSa}	4-4	$\overline{DS}$ signal active (low)	after R.E., PHI1 T2		40		30	ns
t_{DSia}	4-4	$\overline{DS}$ signal inactive	after R.E., PHI1 T4		40		30	ns
t_{ALf}	4-6	AD0–AD23 floating (caused by $\overline{HOLD}$)	after R.E., PHI1 T1		25		20	ns
t_{ADf}	4-6	D24–D31 floating (caused by $\overline{HOLD}$)	after R.E., PHI1 T1		25		20	ns
t_{DSf}	4-6	$\overline{DS}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{ADSf}	4-6	$\overline{ADS}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{BEf}	4-6	$\overline{BEn}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{DDINf}	4-6	$\overline{DDIN}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{HLDAa}	4-6	$\overline{HLDA}$ signal active (low)	after R.E., PHI1 Ti		30		25	ns
t_{HLDAia}	4-8	$\overline{HLDA}$ signal inactive	after R.E., PHI1 Ti		40		30	ns
t_{DSr}	4-8	$\overline{DS}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{ADSr}	4-8	$\overline{ADS}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{BEr}	4-8	$\overline{BEn}$ signals return from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{DDINr}	4-8	$\overline{DDIN}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{DDINf}	4-9	$\overline{DDIN}$ signal floating (caused by $\overline{FLT}$)	after $\overline{FLT}$ F.E.		55		50	ns
t_{DDINr}	4-10	$\overline{DDIN}$ signal returns from floating (caused by $\overline{FLT}$)	after $\overline{FLT}$ R.E.		40		30	ns
t_{SPCa}	4-13	$\overline{SPC}$ output active (low)	after R.E., PHI1 T1		35		26	ns
t_{SPCia}	4-13	$\overline{SPC}$ output inactive	after R.E., PHI1 T4		35		26	ns
t_{SPCnf}	4-15	$\overline{SPC}$ output nonforcing	after R.E., PHI2 T4		30		25	ns
t_{Dv}	4-13	Data valid (slave processor write)	after R.E., PHI1 T1		50		35	ns
t_{Dh}	4-13	Data hold (slave processor write)	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{PFSw}	4-18	$\overline{PFS}$ pulse width	at 15% V_{CC} (both edges)	50		40		ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32C032-10, NS32C032-15 (Continued)

Name	Figure	Description	Reference/ Conditions	NS32C032-10		NS32C032-15		Units
				Min	Max	Min	Max	
t_{PFSa}	4-18	$\overline{PFS}$ pulse active (low)	after R.E., PHI2		40		35	ns
t_{PFSia}	4-18	$\overline{PFS}$ pulse inactive	after R.E., PHI2		40		35	ns
t_{ILOs}	4-20a	$\overline{ILO}$ signal setup	before R.E., PHI1 T1 of first interlocked read cycle	50		35		ns
t_{ILOh}	4-20b	$\overline{ILO}$ signal hold	after R.E., PHI1 T3 of last interlocked write cycle	10		7		ns
t_{ILOa}	4-21	$\overline{ILO}$ signal active (low)	after R.E., PHI1		35		30	ns
t_{ILOia}	4-21	$\overline{ILO}$ signal inactive	after R.E., PHI1		35		30	ns
t_{USv}	4-22	U/$\overline{S}$ signal valid	after R.E., PHI1 T4		35		30	ns
t_{USh}	4-22	U/$\overline{S}$ signal hold	after R.E., PHI1 T4	8		6		ns
t_{NSPF}	4-19b	Nonsequential fetch to next $\overline{PFS}$ clock cycle	after R.E., PHI1 T1	4		4		t_{Cp}
t_{PFNS}	4-19a	$\overline{PFS}$ clock cycle to next non-sequential fetch	before R.E., PHI1 T1	4		4		t_{Cp}
t_{LXPF}	4-29	Last operand transfer of an instruction to next $\overline{PFS}$ clock cycle	before R.E., PHI1 T1 of first of first bus cycle of transfer	0		0		t_{Cp}

Note: Every memory cycle starts with T4, during which Cycle Status is applied. If the CPU was idling, the sequence will be: " . . . Ti, T4, T1 . . . ". If the CPU was not idling, the sequence will be: " . . . T4, T1 . . . ".

4.4.2.2 Input Signal Requirements: NS32C032-10, NS32C032-15

Name	Figure	Description	Reference/Conditions	NS32C032-10		NS32C032-15		Units
				Min	Max	Min	Max	
t_{PWR}	4-25	Power stable to $\overline{RST}$ R.E.	after V_{CC} reaches 4.5V	50		50		μs
t_{DIs}	4-5	Data in setup (read cycle)	before F.E., PHI2 T3	15		10		ns
t_{DIh}	4-5	Data in hold (read cycle)	after R.E., PHI1 T4	3		3		ns
t_{HLDa}	4-6	$\overline{HOLD}$ active (low) setup time (see note)	before F.E., PHI2 TX1	25		17		ns
t_{HLDia}	4-8	$\overline{HOLD}$ inactive setup time	before F.E., PHI2 Ti	25		17		ns
t_{HLDh}	4-6	$\overline{HOLD}$ hold time	after R.E., PHI1 TX2	0		0		ns
t_{FLTa}	4-9	$\overline{FLT}$ active (low) setup time	before F.E., PHI2 Tmmu	25		17		ns
t_{FLTia}	4-10	$\overline{FLT}$ inactive setup time	before F.E., PHI2 T2	25		17		ns
t_{RDYs}	4-11, 4-12	RDY setup time	before F.E., PHI2 T2 or T3	15		10		ns
t_{RDYh}	4-11, 4-12	RDY hold time	after F.E., PHI1 T3	5		5		ns
t_{ABTs}	4-23	$\overline{ABT}$ setup time ($\overline{FLT}$ inactive)	before F.E., PHI2 Tmmu	20		13		ns
t_{ABTs}	4-24	$\overline{ABT}$ setup time ($\overline{FLT}$ active)	before F.E., PHI2 T_f	20		13		ns
t_{ABTh}	4-23	$\overline{ABT}$ hold time	after R.E., PHI1	0		0		ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements NS32C032-10, NS32C032-15 (Continued)

Name	Figure	Description	Reference/ Conditions	NS32C032-10		NS32C032-15		Units
				Min	Max	Min	Max	
t_{RSTs}	4-25, 4-26	$\overline{RST}$ setup time	before F.E., PHI1	10		8		ns
t_{RSTw}	4-26	$\overline{RST}$ pulse width	at 0.8V (both edges)	64		64		t_{Cp}
t_{INTs}	4-27	$\overline{INT}$ setup time	before F.E., PHI1	20		15		ns
t_{NMIw}	4-28	$\overline{NMI}$ pulse width	at 0.8V (both edges)	70		70		ns
t_{DIs}	4-14	Data setup (slave read cycle)	before F.E., PHI2 T1	15		10		ns
t_{DIh}	4-14	Data hold (slave read cycle)	after R.E., PHI1 T4	3		3		ns
t_{SPCd}	4-15	$\overline{SPC}$ pulse delay from slave	after R.E., PHI2 T4	30		25		ns
t_{SPCs}	4-15	$\overline{SPC}$ setup time	before F.E., PHI1	30		25		ns
t_{SPCw}	4-15	$\overline{SPC}$ pulse width from slave processor (async input)	at 0.8V (both edges)	25		20		ns
t_{ATs}	4-16	$\overline{AT}/\overline{SPC}$ setup for address translation strap	before R.E., PHI1 of cycle during which $\overline{RST}$ pulse is removed	1		1		t_{Cp}
t_{ATh}	4-16	$\overline{AT}/\overline{SPC}$ hold for address translation strap	after F.E., PHI1 of cycle during which $\overline{RST}$ pulse is removed	2		2		t_{Cp}

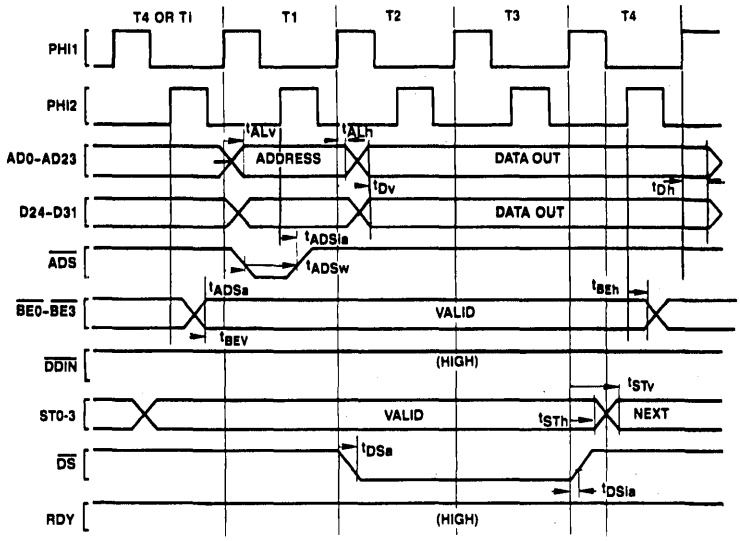
Note: This setup time is necessary to ensure prompt acknowledgement via $\overline{HLDA}$ and the ensuing floating of CPU off the buses. Note that the time from the receipt of the $\overline{HOLD}$ signal until the CPU floats is a function of the time $\overline{HOLD}$ signal goes low, the state of the RDY input (in MMU systems), and the length of the current MMU cycle.

4.4.2.3 Clocking Requirements: NS32C032-10, and NS32C032-15

Name	Figure	Description	Reference/ Conditions	NS32C032-10		NS32C032-15		Units
				Min	Max	Min	Max	
t_{Cp}	4-17	Clock Period	R.E., PHI1, PHI2 to next R.E., PHI1, PHI2	100	250	66	250	ns
$t_{CLw(1,2)}$	4-17	PHI1, PHI2 Pulse Width	At 2.0V on PHI1, PHI2 (Both Edges)	0.5 t_{Cp} −10 ns		0.5 t_{Cp} −6 ns		
$t_{CLh(1,2)}$	4-17	PHI1, PHI2 High Time	At 90% V_{CC} on PHI1, PHI2	0.5 t_{Cp} −15 ns		0.5 t_{Cp} −10 ns		
$t_{CLl(1,2)}$	4−17	PHI1, PHI2 Low Time	At 15% V_{CC} on PHI1, PHI2	0.5 t_{CP} −6 ns		0.5 t_{CP} −5 ns		ns
$t_{nOVL(1,2)}$	4-17	Non-Overlap Time	At 15% V_{CC} on PHI1, PHI2	−2	2	−2	2	ns
t_{nOVLas}		Non-Overlap Asymmetry ($t_{nOVL(1)} - t_{nOVL(2)}$)	At 15% V_{CC} on PHI1, PHI2	−3	3	−3	3	ns
t_{CLwas}		PHI1, PHI2 Asymmetry ($t_{CLw(1)} - t_{CLw(2)}$)	At 2.0V on PHI1, PHI2	−5	5	−3	3	ns

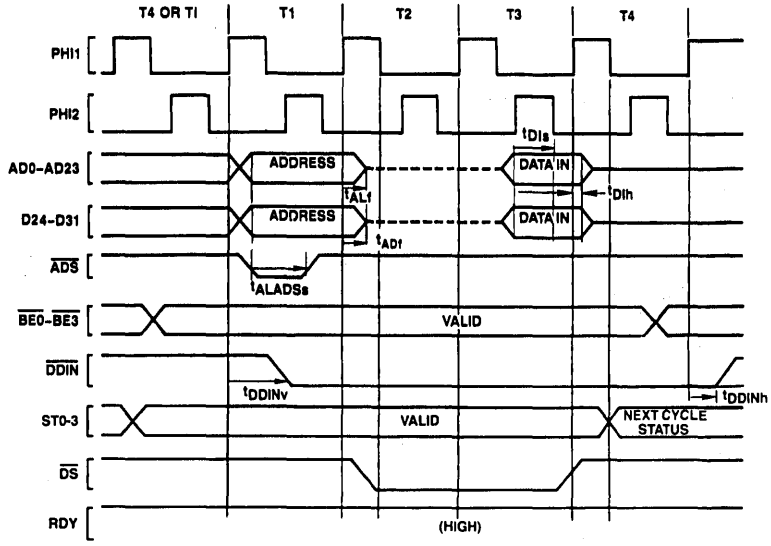
4.0 Device Specifications

4.4.3 Timing Diagrams



TL/EE/9160-45

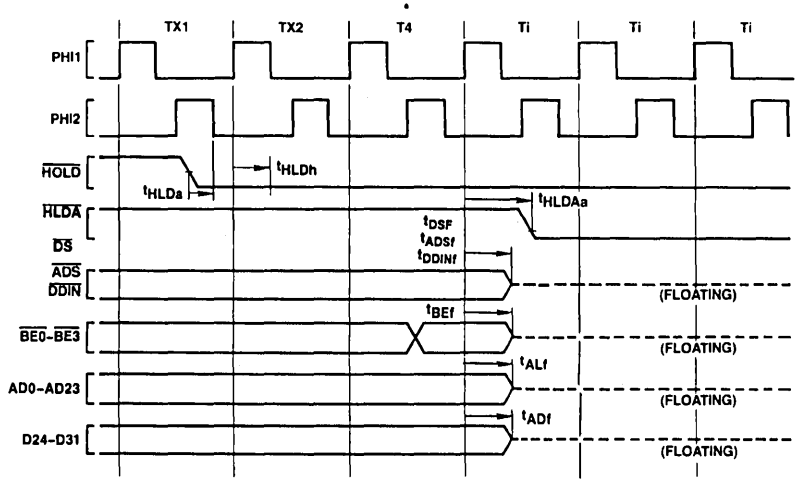
FIGURE 4-4. Write Cycle



TL/EE/9160-46

FIGURE 4-5. Read Cycle

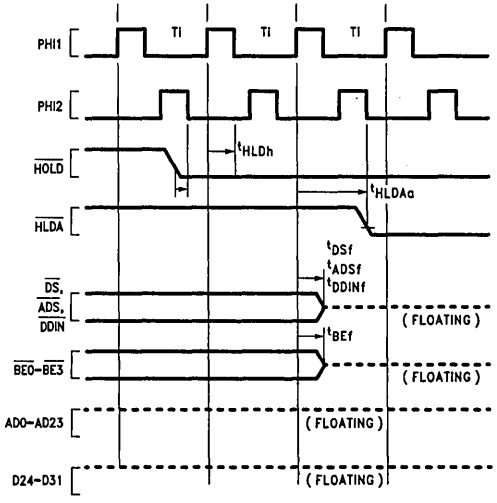
4.0 Device Specifications (Continued)



TL/EE/9160-47

FIGURE 4-6. Floating by $\overline{\text{HOLD}}$ Timing (CPU Not Idle Initially).

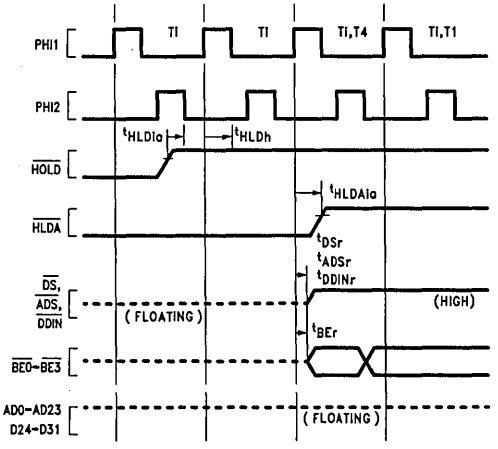
Note that whenever the CPU is not idling (not in Ti), the $\overline{\text{HOLD}}$ request ($\overline{\text{HOLD}}$ low) must be active tHLDa before the falling edge of PHI2 of the clock cycle that appears two clock cycles before T4 (TX1) and stay low until tHLDh after the rising edge of PHI1 of the clock cycle that precedes T4 (TX2) for the request to be acknowledged.



TL/EE/9160-48

FIGURE 4-7. Floating by $\overline{\text{HOLD}}$ Timing (CPU Initially Idle)

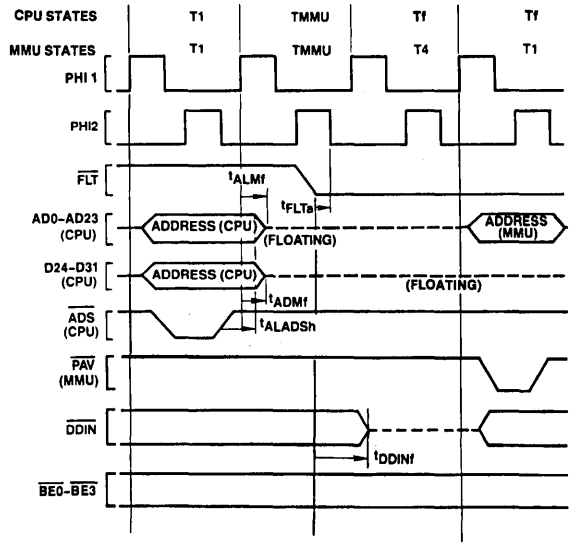
Note that during Ti1 the CPU is already idling.



TL/EE/9160-49

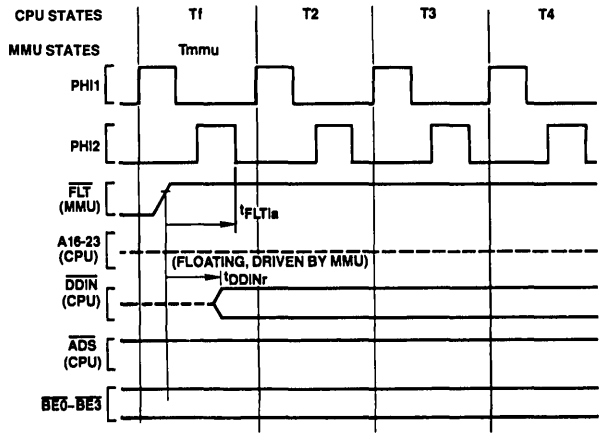
FIGURE 4-8. Release from $\overline{\text{HOLD}}$

4.0 Device Specifications (Continued)



TL/EE/9160-50

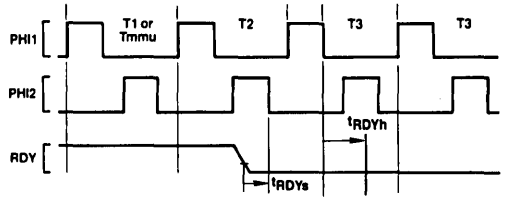
FIGURE 4-9. FLT Initiated Float Cycle Timing



TL/EE/9160-51

FIGURE 4-10. Release from FLT Timing

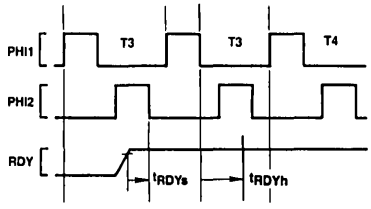
Note that when FLT is deasserted the CPU restarts driving DDIN before the MMU releases it. This, however, does not cause any conflict, since both CPU and MMU force DDIN to the same logic level.



TL/EE/9160-52

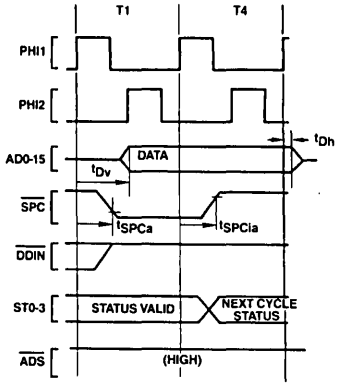
FIGURE 4-11. Ready Sampling (CPU Initially READY)

4.0 Device Specifications (Continued)



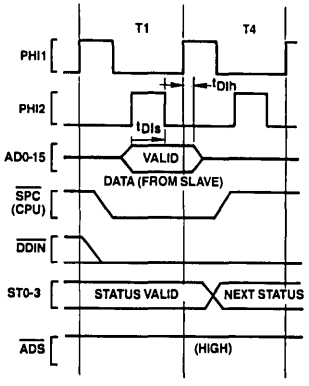
TL/EE/9160-53

FIGURE 4-12. Ready Sampling (CPU Initially NOT READY)



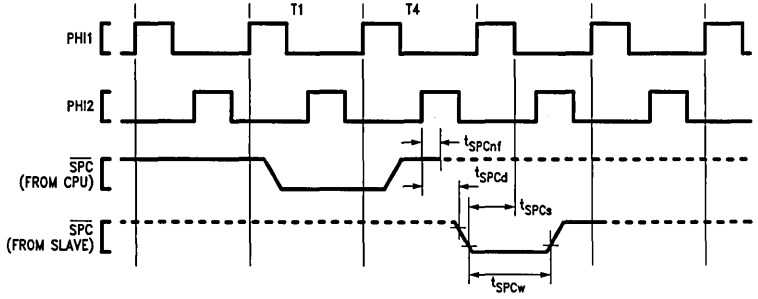
TL/EE/9160-54

FIGURE 4-13. Slave Processor Write Timing



TL/EE/9160-55

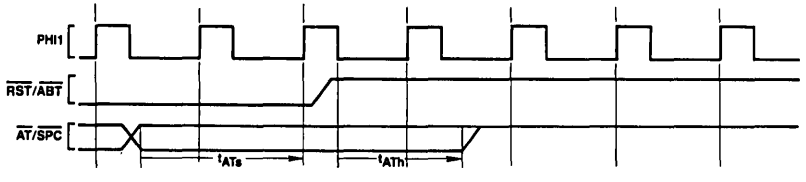
FIGURE 4-14. Slave Processor Read Timing



TL/EE/9160-56

FIGURE 4-15. $\overline{SPC}$ Timing

After transferring last operand to a Slave Processor, CPU turns OFF driver and holds $\overline{SPC}$ high with internal 5 kΩ pullup.

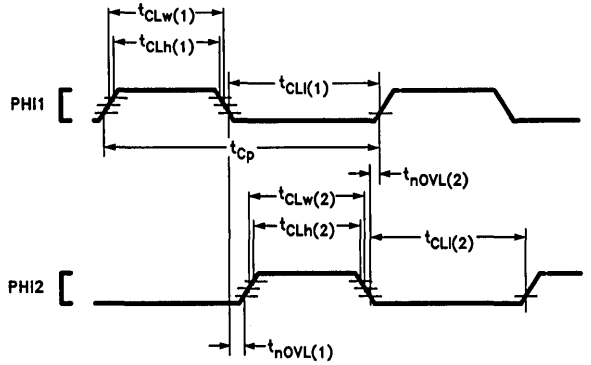


TL/EE/9160-57

FIGURE 4-16. Reset Configuration Timing

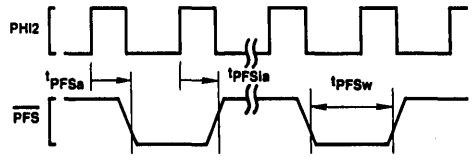
2

4.0 Device Specifications (Continued)



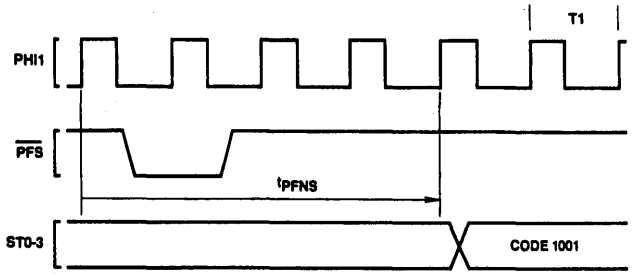
TL/EE/9160-58

FIGURE 4-17. Clock Waveforms



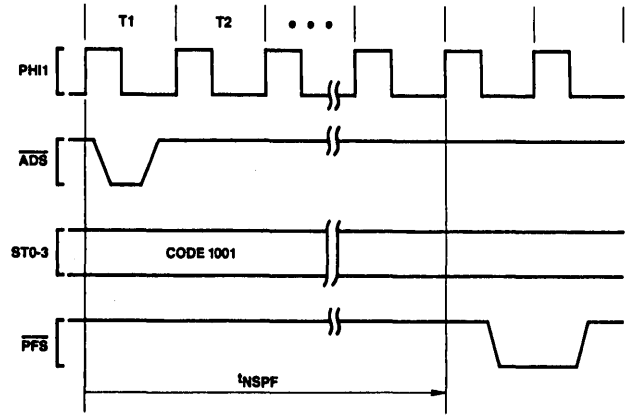
TL/EE/9160-59

FIGURE 4-18. Relationship of $\overline{PFS}$ to Clock Cycles



TL/EE/9160-60

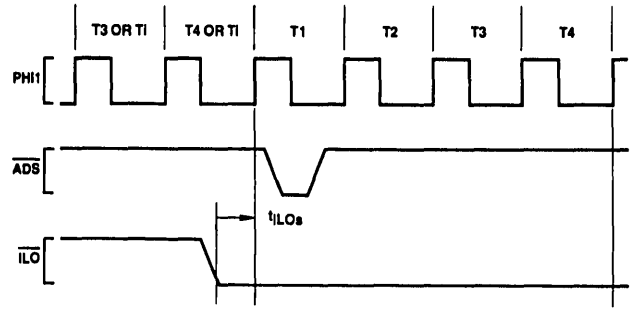
FIGURE 4-19a. Guaranteed Delay, $\overline{PFS}$ to Non-Sequential Fetch



TL/EE/9160-61

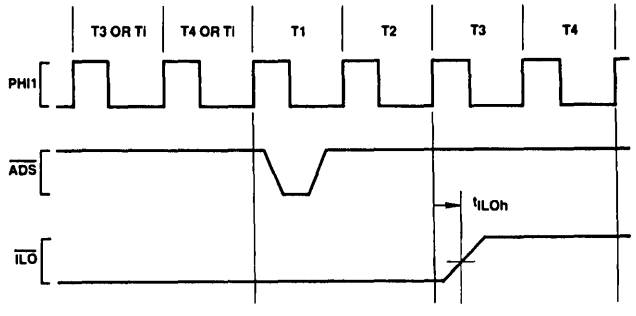
FIGURE 4-19b. Guaranteed Delay, Non-Sequential Fetch to $\overline{PFS}$

4.0 Device Specifications (Continued)



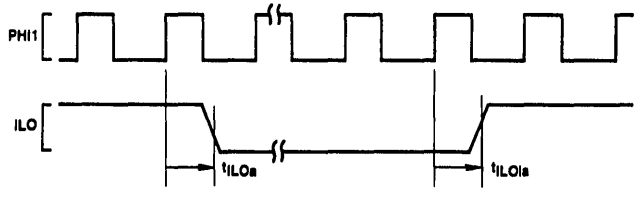
TL/EE/9160-62

FIGURE 4-20a. Relationship of $\overline{ILO}$ to First Operand Cycle of an Interlocked Instruction



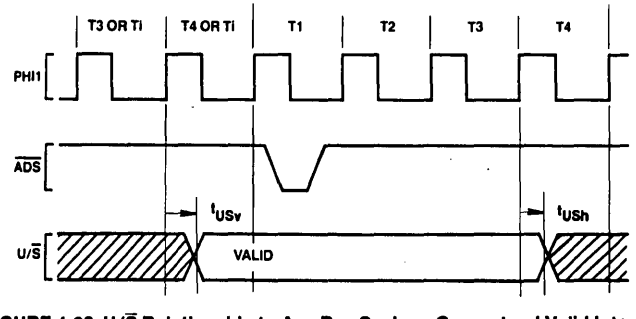
TL/EE/9160-63

FIGURE 4-20b. Relationship of $\overline{ILO}$ to Last Operand Cycle of an Interlocked Instruction



TL/EE/9160-64

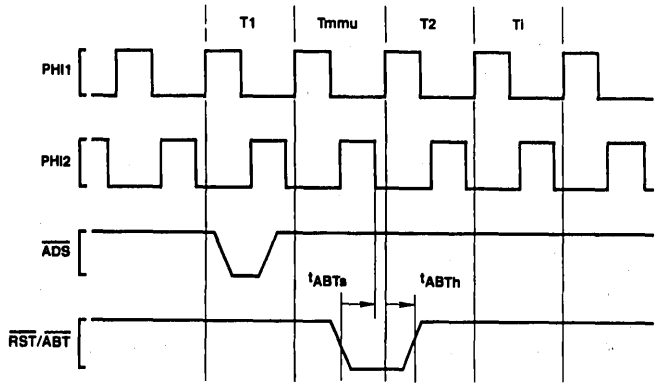
FIGURE 4-21. Relationship of $\overline{ILO}$ to Any Clock Cycle



TL/EE/9160-65

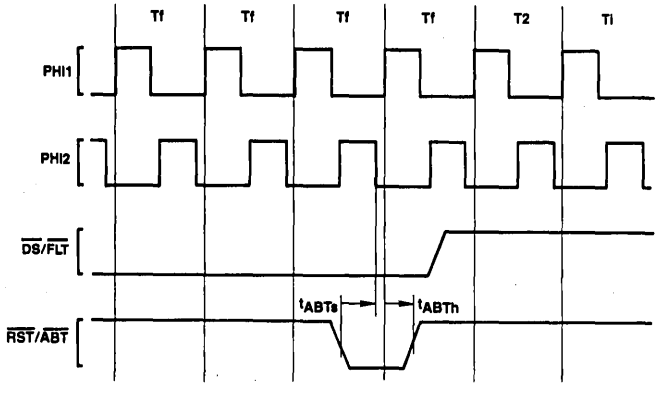
FIGURE 4-22. U/$\overline{S}$ Relationship to Any Bus Cycle — Guaranteed Valid Interval

4.0 Device Specifications (Continued)



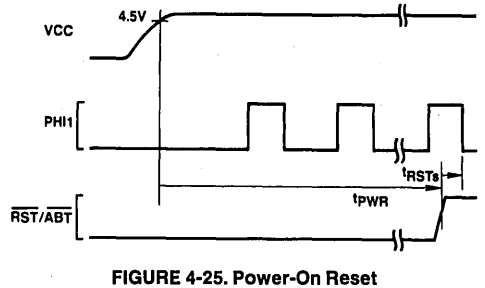
TL/EE/9160-66

FIGURE 4-23. Abort Timing, $\overline{FLT}$ Not Applied



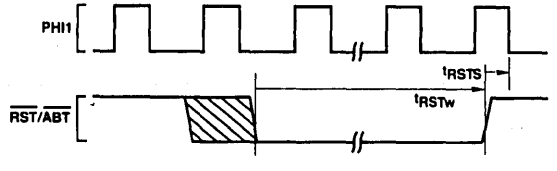
TL/EE/9160-67

FIGURE 4-24. Abort Timing, $\overline{FLT}$ Applied



TL/EE/9160-68

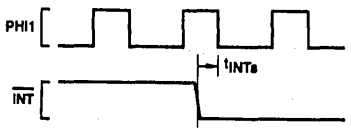
FIGURE 4-25. Power-On Reset



TL/EE/9160-69

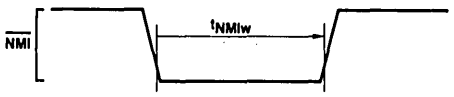
FIGURE 4-26. Non-Power-On Reset

4.0 Device Specifications (Continued)



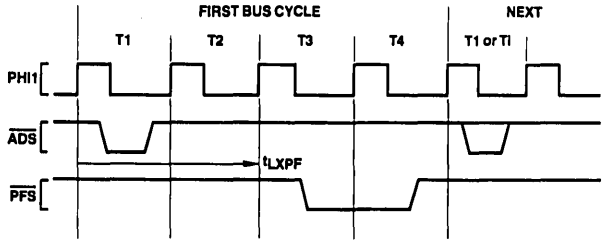
TL/EE/9160-70

FIGURE 4-27. $\overline{INT}$ Interrupt Signal Detection



TL/EE/9160-71

FIGURE 4-28. $\overline{NMI}$ Interrupt Signal Timing



TL/EE/9160-72

FIGURE 4-29. Relationship Between Last Data Transfer of an Instruction and $\overline{PFS}$ Pulse of Next Instruction

Note: In a transfer of a Read-Modify-Write type operand, this is the Read transfer, displaying RMW Status (Code 1011).

Appendix A: Instruction Formats

NOTATIONS

i= Integer Type Field
B = 00 (Byte)
W = 01 (Word)
D = 11 (Double Word)

f= Floating Point Type Field
F = 1 (Std. Floating: 32 bits)
L = 0 (Long Floating: 64 bits)

c= Custom Type Field
D = 1 (Double Word)
Q = 0 (Quad Word)

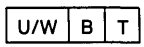
op= Operation Code
Valid encodings shown with each format.

gen, gen 1, gen 2= General Addressing Mode Field
See Sec. 2.2 for encodings.

reg= General Purpose Register Number

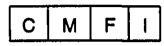
cond= Condition Code Field
0000 = EQual: Z = 1
0001 = Not Equal: Z = 0
0010 = Carry Set: C = 1
0011 = Carry Clear: C = 0
0100 = HIgher: L = 1
0101 = Lower or Same: L = 0
0110 = Greater Than: N = 1
0111 = Less or Equal: N = 0
1000 = Flag Set: F = 1
1001 = Flag Clear: F = 0
1010 = LOwer: L = 0 and Z = 0
1011 = Higher or Same: L = 1 or Z = 1
1100 = Less Than: N = 0 and Z = 0
1101 = Greater or Equal: N = 1 or Z = 1
1110 = (Unconditionally True)
1111 = (Unconditionally False)

short= Short Immediate value. May contain
quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
cond: Condition Code (above), in Scond.
areg: CPU Dedicated Register, in LPR, SPR.
0000 = US
0001 – 0111 = (Reserved)
1000 = FP
1001 = SP
1010 = SB
1011 = (Reserved)
1100 = (Reserved)
1101 = PSR
1110 = INTBASE
1111 = MOD
Options: in String Instructions

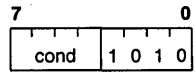


T = Translated
B = Backward
U/W = 00: None
01: While Match
11: Until Match

Configuration bits, in SETCFG:

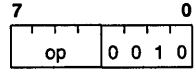


mreg: NS32082 Register number, in LMR, SMR.
0000 = BPR0
0001 = BPR1
0010 = (Reserved)
0011 = (Reserved)
0100 = (Reserved)
0101 = (Reserved)
0110 = (Reserved)
0111 = (Reserved)
1000 = (Reserved)
1001 = (Reserved)
1010 = MSR
1011 = BCNT
1100 = PTB0
1101 = PTB1
1110 = (Reserved)
1111 = EIA



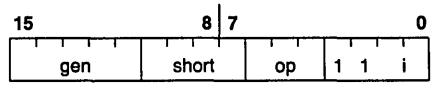
Format 0

Bcond (BR)



Format 1

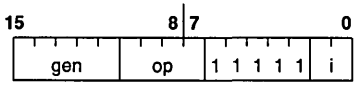
BSR	-0000	ENTER	-1000
RET	-0001	EXIT	-1001
CXP	-0010	NOP	-1010
RXP	-0011	WAIT	-1011
RETT	-0100	DIA	-1100
RETI	-0101	FLAG	-1101
SAVE	-0110	SVC	-1110
RESTORE	-0111	BPT	-1111



Format 2

ADDQ	-000	ACB	-100
CMPQ	-001	MOVQ	-101
SPR	-010	LPR	-110
Scond	-011		

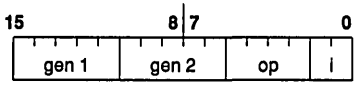
Appendix A: Instruction Formats (Continued)



Format 3

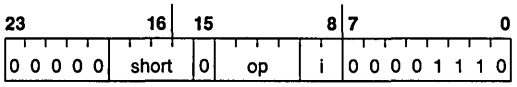
CXPD	-0000	ADJSP	-1010
BICPSR	-0010	JSR	-1100
JUMP	-0100	CASE	-1110
BISPSR	-0110		

Trap (UND) on XXX1, 1000



Format 4

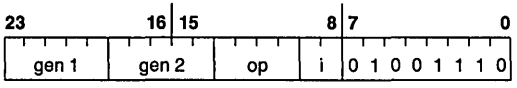
ADD	-0000	SUB	-1000
CMP	-0001	ADDR	-1001
BIC	-0010	AND	-1010
ADDC	-0100	SUBC	-1100
MOV	-0101	TBIT	-1101
OR	-0110	XOR	-1110



Format 5

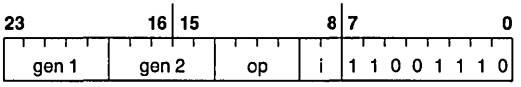
MOVS	-0000	SETCFG	-0010
CMPS	-0001	SKPS	-0011

Trap (UND) on 1XXX, 01XX



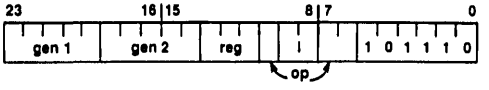
Format 6

ROT	-0000	NEG	-1000
ASH	-0001	NOT	-1001
CBIT	-0010	Trap (UND)	-1010
CBITI	-0011	SUBP	-1011
Trap (UND)	-0100	ABS	-1100
LSH	-0101	COM	-1101
SBIT	-0110	IBIT	-1110
SBITI	-0111	ADDP	-1111



Format 7

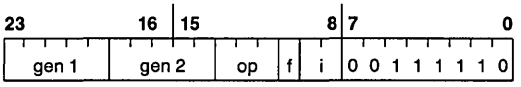
MOVM	-0000	MUL	-1000
CMPM	-0001	MEI	-1001
INSS	-0010	Trap (UND)	-1010
EXTS	-0011	DEI	-1011
MOVXBW	-0100	QUO	-1100
MOVZBW	-0101	REM	-1101
MOVZiD	-0110	MOD	-1110
MOVXiD	-0111	DIV	-1111



TL/EE/9160-73

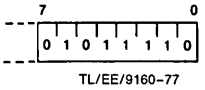
Format 8

EXT	-0 00	INDEX	-1 00
CVTP	-0 01	FFS	-1 01
INS	-0 10		
CHECK	-0 11		
MOVSU	-110, reg = 001		
MOVUS	-110, reg = 011		



Format 9

MOVif	-000	ROUND	-100
LFSR	-001	TRUNC	-101
MOVLF	-010	SFSR	-110
MOVFL	-011	FLOOR	-111

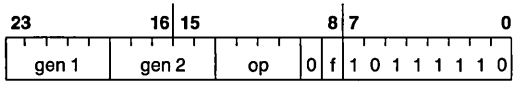


TL/EE/9160-77

Format 10

Trap (UND) Always

Appendix A: Instruction Formats (Continued)



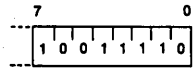
Format 11

ADDf	-0000	DIVf	-1000
MOVf	-0001	Trap (SLAVE)	-1001
CMPf	-0010	Trap (UND)	-1010
Trap (SLAVE)	-0011	Trap (UND)	-1011
SUBf	-0100	MULf	-1100
NEGf	-0101	ABSf	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111

TL/EE/9160–75

Format 12

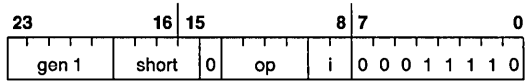
Trap (UND) Always



TL/EE/9160–76

Format 13

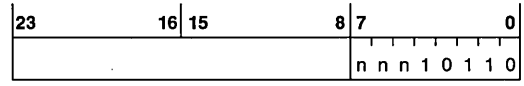
Trap (UND) Always



Format 14

RDVAL	-0000	LMR	-0010
WRVAL	-0001	SMR	-0011

Trap (UND) on 01XX, 1XXX



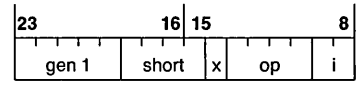
Operation Word ID Byte

Format 15

(Custom Slave)

nnn **Operation Word Format**

000

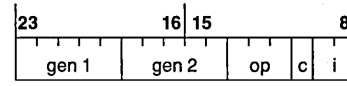


Format 15.0

CATST0	-0000	LCR	-0010
CATST1	-0001	SCR	-0011

Trap (UND) on all others

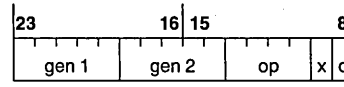
001



Format 15.1

CCV3	-000	CCV2	-100
LCSR	-001	CCV1	-101
CCV5	-010	SCSR	-110
CCV4	-011	CCV0	-111

101

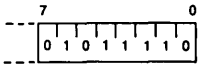


Format 15.5

CCAL0	-0000	CCAL3	-1000
CMOV0	-0001	CMOV3	-1001
CCMP0	-0010	Trap (UND)	-1010
CCMP1	-0011	Trap (UND)	-1011
CCAL1	-0100	CCAL2	-1100
CMOV2	-0101	CMOV1	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111

If nnn = 010, 011, 100, 110, 111
then Trap (UND) Always

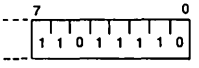
Appendix A: Instruction Formats (Continued)



TL/EE/9160-77

Format 16

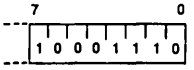
Trap (UND) Always



TL/EE/9160-78

Format 17

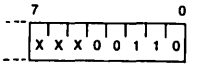
Trap (UND) Always



TL/EE/9160-79

Format 18

Trap (UND) Always

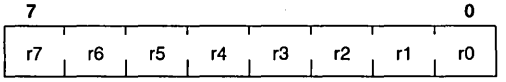


TL/EE/9160-80

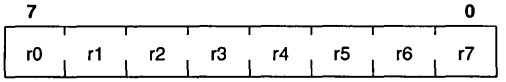
Format 19

Trap (UND) Always

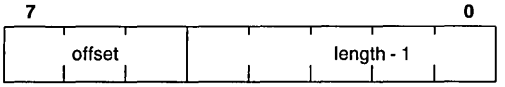
Implied Immediate Encodings:



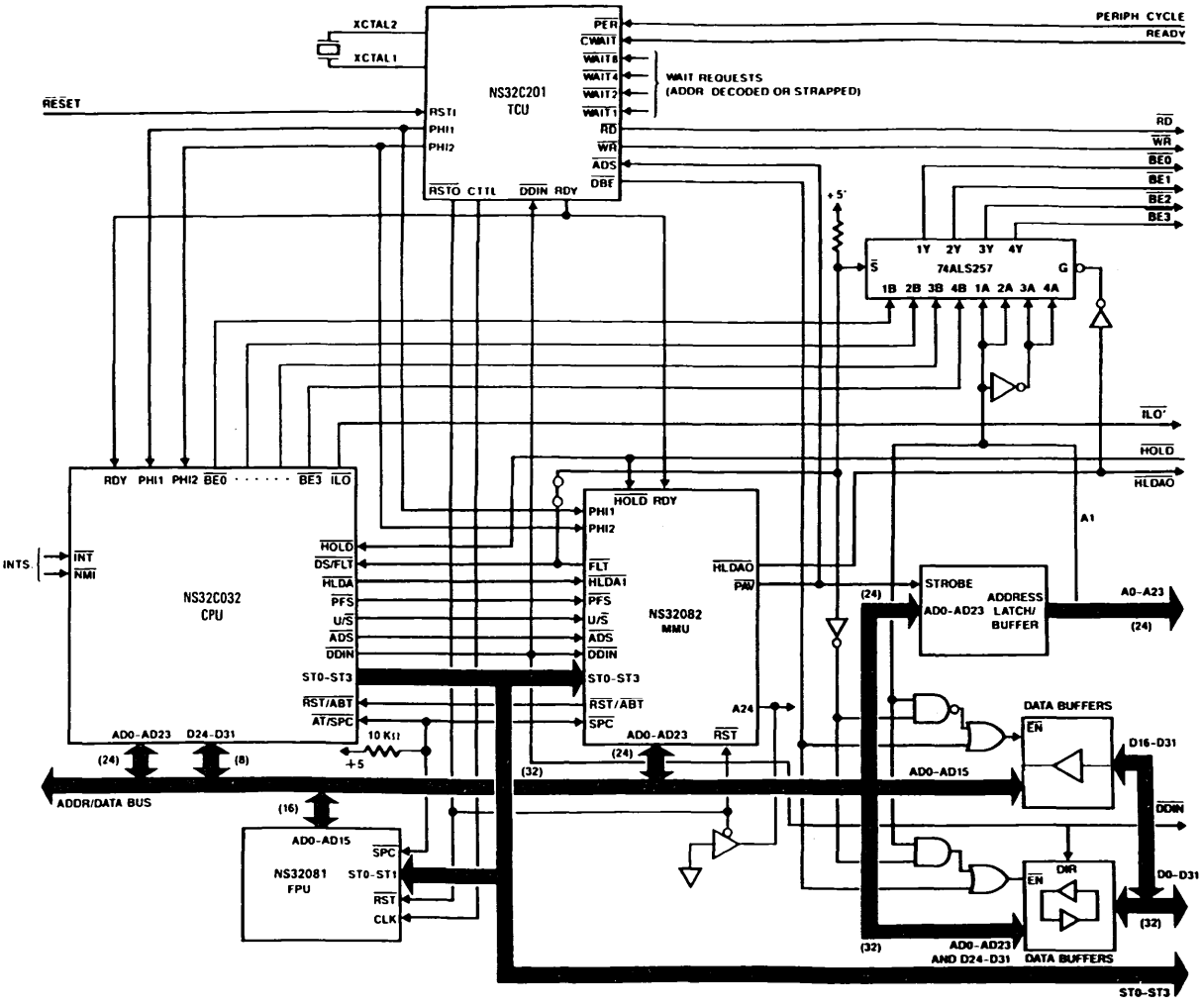
Register Mark, appended to SAVE, ENTER



Register Mark, appended to RESTORE, EXIT



Offset/Length Modifier appended to INSS, EXTS



TL/EE/9160–81

FIGURE B-1. System Connection Diagram

National Semiconductor

PRELIMINARY

NS32C016-10/NS32C016-15
High-Performance Microprocessors

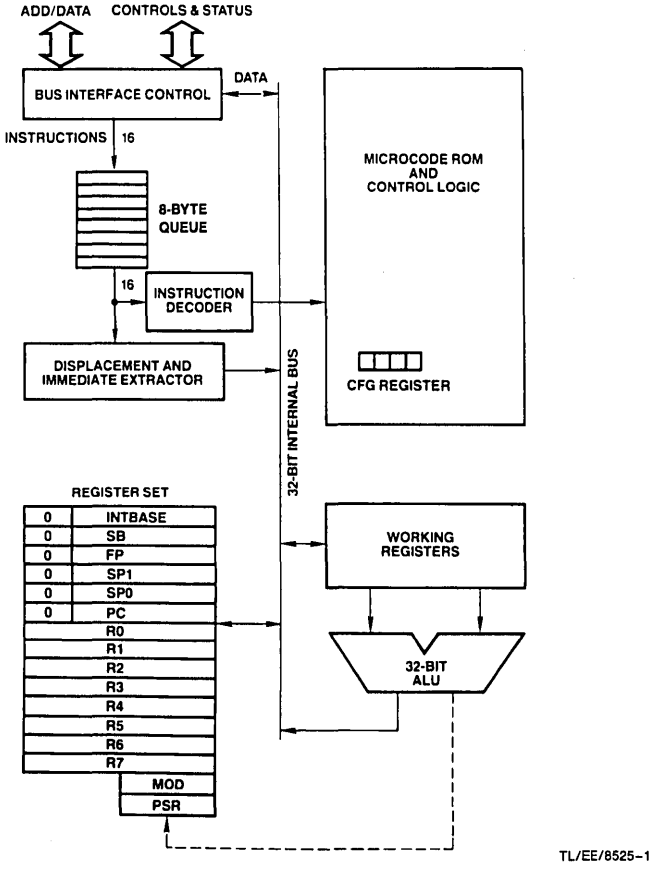
General Description

The NS32C016 is a 32-bit, CMOS microprocessor with TTL compatible inputs. The NS32C016 has a 16M byte linear address space and a 16-bit external data bus. It is fabricated with National Semiconductor's advanced CMOS process and is fully object code compatible with other Series 32000® CPU's. The NS32C016 has a 32-bit ALU, eight 32-bit general purpose registers, an eight-byte prefetch queue and a highly symmetric architecture. It also incorporates a slave processor interface and provides for full virtual memory capability in conjunction with the NS32082 memory management unit (MMU). High performance floating-point instructions are provided with the NS32081 floating-point unit (FPU). The NS32C016 is intended for a wide range of high performance computer applications.

Features

- 32-bit architecture and implementation
- 16M byte uniform addressing space
- Powerful instruction set
  - — General 2-address capability
  - — Very high degree of symmetry
  - — Addressing modes optimized for high-level Language references
- High-speed CMOS technology
- TTL compatible inputs
- Single 5V supply
- 48-pin dual-in-line package

Block Diagram



TL/EE/8525–1

2

Table of Contents

List of Illustrations

List of Illustrations (Continued)

List of Illustrations (Continued)

List of Tables

1.0 Product Introduction

The Series 32000 Microprocessor family is a new generation of devices using National's XMOS and CMOS technologies. By combining state-of-the-art MOS technology with a very advanced architectural design philosophy, this family brings mainframe computer processing power to VLSI processors.

The Series 32000 family supports a variety of system configurations, extending from a minimum low-cost system to a powerful 4 gigabyte system. The architecture provides complete upward compatibility from one family member to another. The family consists of a selection of CPUs supported by a set of peripherals and slave processors that provide sophisticated interrupt and memory management facilities as well as high-speed floating-point operations. The architectural features of the Series 32000 family are described briefly below:

Powerful Addressing Modes. Nine addressing modes available to all instructions are included to access data structures efficiently.

Data Types. The architecture provides for numerous data types, such as byte, word, doubleword, and BCD, which may be arranged into a wide variety of data structures.

Symmetric Instruction Set. While avoiding special case instructions that compilers can't use, the Series 32000 family incorporates powerful instructions for control operations, such as array indexing and external procedure calls, which save considerable space and time for compiled code.

Memory-to-Memory Operations. The Series 32000 CPUs represent two-address machines. This means that each operand can be referenced by any one of the addressing modes provided. This powerful memory-to-memory architecture permits memory locations to be treated as registers for all useful operations. This is important for temporary operands as well as for context switching.

Memory Management. Either the NS32382 or the NS32082 Memory Management Unit may be added to the system to provide advanced operating system support functions, including dynamic address translation, virtual memory management, and memory protection.

Large, Uniform Addressing. The NS32C016 has 24-bit address pointers that can address up to 16 megabytes without any segmentation; this addressing scheme provides flexible memory management without added-on expense.

Modular Software Support. Any software package for the Series 32000 family can be developed independent of all other packages, without regard to individual addressing. In addition, ROM code is totally relocatable and easy to access, which allows a significant reduction in hardware and software cost.

Software Processor Concept. The Series 32000 architecture allows future expansions of the instruction set that can be executed by special slave processors, acting as extensions to the CPU. This concept of slave processors is unique to the Series 32000 family. It allows software compatibility even for future components because the slave hardware is transparent to the software. With future advances in semiconductor technology, the slaves can be physically integrated on the CPU chip itself.

To summarize, the architectural features cited above provide three primary performance advantages and characteristics:

- High-Level Language Support
- Easy Future Growth Path
- Application Flexibility

2.0 Architectural Description

2.1 PROGRAMMING MODEL

The Series 32000 architecture includes 16 registers on the NS32C016 CPU.

2.1.1 General Purpose Registers

There are eight registers for meeting high speed general storage requirements, such as holding temporary variables and addresses. The general purpose registers are free for any use by the programmer. They are thirty-two bits in length. If a general register is specified for an operand that is eight or sixteen bits long, only the low part of the register is used; the high part is not referenced or modified.

2.1.2 Dedicated Registers

The eight dedicated registers of the NS32C016 are assigned specific functions.

PC: The PROGRAM COUNTER register is a pointer to the first byte of the instruction currently being executed. The PC is used to reference memory in the program section. (In the NS32C016 the upper eight bits of this register are always zero.)

SP0, SP1: The SP0 register points to the lowest address of the last item stored on the INTERRUPT STACK. This stack is normally used only by the operating system. It is used primarily for storing temporary data, and holding return information for operating system subroutines and interrupt and

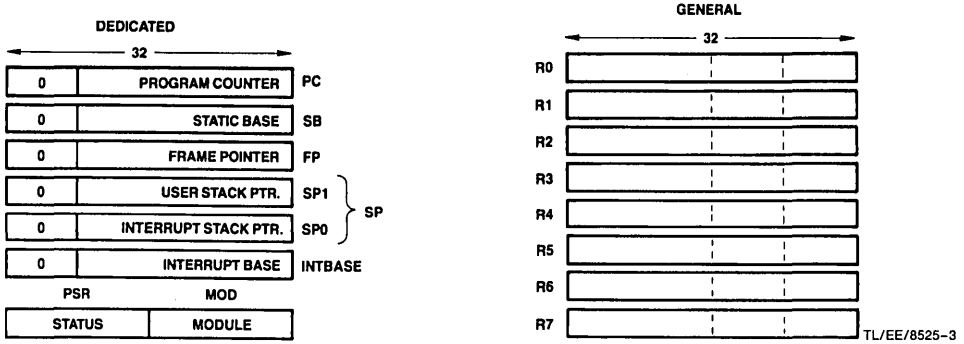


FIGURE 2-1. The General and Dedicated Registers

2.0 Architectural Description (Continued)

trap service routines. The SP1 register points to the lowest address of the last item stored on the USER STACK. This stack is used by normal user programs to hold temporary data and subroutine return information.

In this document, reference is made to the SP register. The terms "SP register" or "SP" refer to either SP0 or SP1, depending on the setting of the S bit in the PSR register. If the S bit in the PSR is 0 then SP refers to SP0. If the S bit in the PSR is 1 then SP refers to SP1. (In the NS32C016 the upper eight bits of these registers are always zero.)

Stacks in the Series 32000 family grow downward in memory. A Push operation pre-decrements the Stack Pointer by the operand length. A Pop operation post-increments the Stack Pointer by the operand length.

FP: The FRAME POINTER register is used by a procedure to access parameters and local variables on the stack. The FP register is set up on procedure entry with the ENTER instruction and restored on procedure termination with the EXIT instruction.

The frame pointer holds the address in memory occupied by the old contents of the frame pointer. (In the NS32C016 the upper eight bits of this register are always zero.)

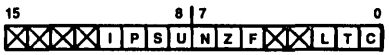
SB: The STATIC BASE register points to the global variables of a software module. This register is used to support relocatable global variables for software modules. The SB register holds the lowest address in memory occupied by the global variables of a module. (In the NS32C016 the upper eight bits of this register are always zero.)

INTBASE: The INTERRUPT BASE register holds the address of the dispatch table for interrupts and traps (Section 3.8). The INTBASE register holds the lowest address in memory occupied by the dispatch table. (In the NS32C016 the upper eight bits of this register are always zero.)

MOD: The MODULE register holds the address of the module descriptor of the currently executing software module. The MOD register is sixteen bits long, therefore the module table must be contained within the first 64k bytes of memory.

PSR: The PROCESSOR STATUS REGISTER (PSR) holds the status codes for the NS32C016 microprocessor.

The PSR is sixteen bits long, divided into two eight-bit halves. The low order eight bits are accessible to all programs, but the high order eight bits are accessible only to programs executing in Supervisor Mode.



TL/EE/8525–78

FIGURE 2-2. Processor Status Register

C: The C bit indicates that a carry or borrow occurred after an addition or subtraction instruction. It can be used with the ADDC and SUBC instructions to perform multiple-precision integer arithmetic calculations. It may have a setting of 0 (no carry or borrow) or 1 (carry or borrow).

T: The T bit causes program tracing. If this bit is a 1, a TRC trap is executed after every instruction (Section 3.8.5).

L: The L bit is altered by comparison instructions. In a comparison instruction the L bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as unsigned integers. Otherwise, it is set to "0". In Floating Point comparisons, this bit is always cleared.

F: The F bit is a general condition flag, which is altered by many instructions (e.g., integer arithmetic instructions use it to indicate overflow).

Z: The Z bit is altered by comparison instructions. In a comparison instruction the Z bit is set to "1" if the second operand is equal to the first operand; otherwise it is set to "0".

N: The N bit is altered by comparison instructions. In a comparison instruction the N bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as signed integers. Otherwise, it is set to "0".

U: If the U bit is "1" no privileged instructions may be executed. If the U bit is "0" then all instructions may be executed. When U=0 the NS32C016 is said to be in Supervisor Mode; when U=1 the NS32C016 is said to be in User Mode. A User Mode program is restricted from executing certain instructions and accessing certain registers which could interfere with the operating system. For example, a User Mode program is prevented from changing the setting of the flag used to indicate its own privilege mode. A Supervisor Mode program is assumed to be a trusted part of the operating system, hence it has no such restrictions.

S: The S bit specifies whether the SP0 register or SP1 register is used as the stack pointer. The bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use the SP0 register) or 1 (use the SP1 register).

P: The P bit prevents a TRC trap from occurring more than once for an instruction (Section 3.8.5). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I: If I=1, then all interrupts will be accepted (Section 3.8). If I=0, only the NMI interrupt is accepted. Trap enables are not affected by this bit.

2.1.3 The Configuration Register (CFG)

Within the Control section of the NS32C016 CPU is the four-bit CFG Register, which declares the presence of certain external devices. It is referenced by only one instruction, SETCFG, which is intended to be executed only as part of system initialization after reset. The format of the CFG Register is shown in *Figure 2-3.*

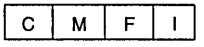


FIGURE 2-3. CFG Register

The CFG I bit declares the presence of external interrupt vectoring circuitry (specifically, the NS32202 Interrupt Control Unit). If the CFG I bit is set, interrupts requested through the $\overline{INT}$ pin are "Vectored." If it is clear, these interrupts are "Non-Vectored." See Section 3.8.

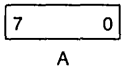
The F, M and C bits declare the presence of the FPU, MMU and Custom Slave Processors. If these bits are not set, the corresponding instructions are trapped as being undefined.

2.1.4 Memory Organization

The main memory of the NS32C016 is a uniform linear address space. Memory locations are numbered sequentially starting at zero and ending at $2^{24} - 1$. The number specifying a memory location is called an address. The contents of each memory location is a byte consisting of eight bits. Unless otherwise noted, diagrams in this document show data stored in memory with the lowest address on the right and the highest address on the left. Also, when data is shown vertically, the lowest address is at the top of a diagram and

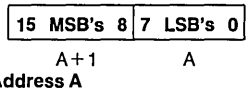
2.0 Architectural Description (Continued)

the highest address at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number zero, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.



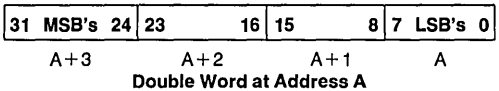
Byte at Address A

Two contiguous bytes are called a word. Except where noted (Section 2.2.1), the least significant byte of a word is stored at the lower address, and the most significant byte of the word is stored at the next higher address. In memory, the address of a word is the address of its least significant byte, and a word may start at any address.



Word at Address A

Two contiguous words are called a double word. Except where noted (Section 2.2.1), the least significant word of a double word is stored at the lowest address and the most significant word of the double word is stored at the address two greater. In memory, the address of a double word is the address of its least significant byte, and a double word may start at any address.



Double Word at Address A

Although memory is addressed as bytes, it is actually organized as words. Therefore, words and double words that are aligned to start at even addresses (multiples of two) are accessed more quickly than words and double words that are not so aligned.

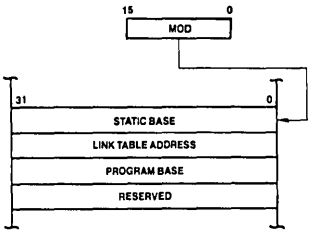
2.1.5 Dedicated Tables

Two of the NS32C016 dedicated registers (MOD and INTBASE) serve as pointers to dedicated tables in memory.

The INTBASE register points to the Interrupt Dispatch and Cascade tables. These are described in Section 3.8.

The MOD register contains a pointer into the Module Table, whose entries are called Module Descriptors. A Module Descriptor contains four pointers, three of which are used by the NS32C016. The MOD register contains the address of the Module Descriptor for the currently running module. It is automatically updated by the Call External Procedure instructions (CXP and CXPD).

The format of a Module Descriptor is shown in *Figure 2-4*. The Static Base entry contains the address of static data assigned to the running module. It is loaded into the CPU Static Base register by the CXP and CXPD instructions. The Program Base entry contains the address of the first byte of instruction code in the module. Since a module may have multiple entry points, the Program Base pointer serves only as a reference to find them.



TL/EE/8525–4

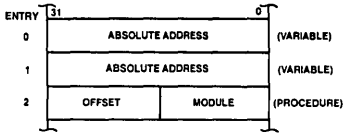
FIGURE 2-4. Module Descriptor Format

The Link Table Address points to the Link Table for the currently running module. The Link Table provides the information needed for:

1) Sharing variables between modules. Such variables are accessed through the Link Table via the External addressing mode.
2) Transferring control from one module to another. This is done via the Call External Procedure (CXP) instruction.

The format of a Link Table is given in *Figure 2-5*. A Link Table Entry for an external variable contains the 32-bit address of that variable. An entry for an external procedure contains two 16-bit fields: Module and Offset. The Module field contains the new MOD register contents for the module being entered. The Offset field is an unsigned number giving the position of the entry point relative to the new module's Program Base pointer.

For further details of the functions of these tables, see the Series 32000 Instruction Set Reference Manual.



TL/EE/8525–5

FIGURE 2-5. A Sample Link Table

2.2 INSTRUCTION SET

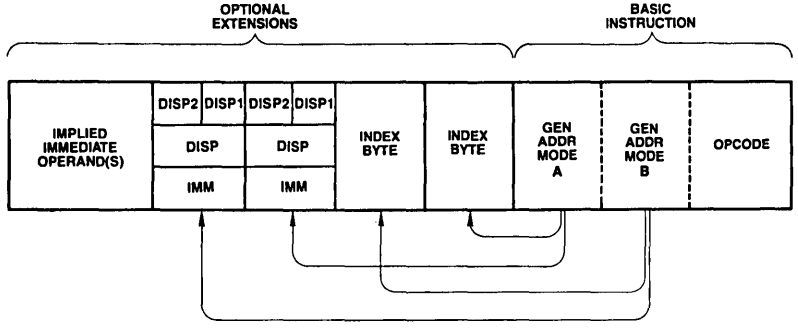
2.2.1 General Instruction Format

Figure 2-6 shows the general format of a Series 32000 instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to 5-bit General Addressing Mode ("Gen") fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

Index Bytes appear when either or both Gen fields specify Scaled Index. In this case, the Gen field specifies only the Scale Factor (1, 2, 4 or 8), and the Index Byte specifies which General Purpose Register to use as the index, and which addressing mode calculation to perform before indexing. See *Figure 2-7*.

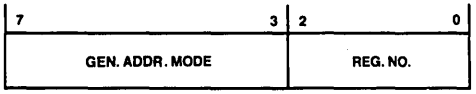
Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Disp/Imm field may contain

2.0 Architectural Description (Continued)



TL/EE/8525-6

FIGURE 2-6. General Instruction Format



TL/EE/8525-7

FIGURE 2-7. Index Byte Format

one of two displacements, or one immediate value. The size of a Displacement field is encoded within the top bits of that field, as shown in *Figure 2-8,* with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the Opcode field. Both Displacement and Immediate fields are stored most-significant byte first. Note that this is different from the memory representation of data (Section 2.1.4).

Some instructions require additional "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition (Section 2.2.3).

2.2.2 Addressing Modes

The NS32C016 CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."

Addressing modes in the NS32C016 are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode, within the instruction that acts upon that variable. Extraneous data movement is therefore minimized.

NS32C016 Addressing Modes fall into nine basic types:

Register: The operand is available in one of the eight General Purpose Registers. In certain Slave Processor instructions, an auxiliary set of eight registers may be referenced instead.

Register Relative: A General Purpose Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.

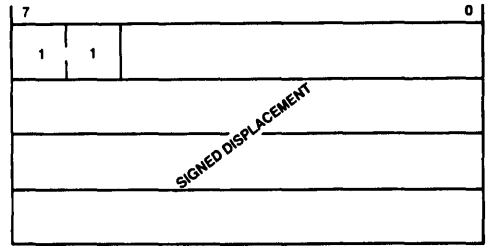
Memory Space: Identical to Register Relative above, except that the register used is one of the dedicated registers PC, SP, SB or FP. These registers point to data areas generally needed by high-level languages.

Memory Relative: A pointer variable is found within the memory space pointed to by the SP, SB or FP register. A

Byte Displacement: Range −64 to +63

Word Displacement: Range −8192 to +8191

Double Word Displacement: Range (Entire Addressing Space)



TL/EE/8525-8

FIGURE 2-8. Displacement Encodings

displacement is added to that pointer to generate the Effective Address of the operand.

Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.

Absolute: The address of the operand is specified by a displacement field in the instruction.

External: A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.

Top of Stack: The currently-selected Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.

2.0 Architectural Description (Continued)

Scaled Index: Although encoded as an addressing mode, Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any General Purpose Register by 1, 2, 4 or 8 and adding into the total, yielding the final Effective Address of the operand.

Table 2-1 is a brief summary of the addressing modes. For a complete description of their actions, see the Series 32000 Instruction Set Reference Manual.

TABLE 2-1. NS32C016 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0 or F0	None: Operand is in the specified register.
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R6 or F7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Relative			
10000	Frame memory relative	disp2(disp1 (FP))	Disp2 + Pointer; Pointer found at address Disp 1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1 (SP))	
10010	Static memory relative	disp2(disp1 (SB))	
Reserved			
10011	(Reserved for Future Use)		
Immediate			
10100	Immediate	value	None: Operand is input from instruction queue.
Absolute			
10101	Absolute	@disp	Disp.
External			
10110	External	EXT (disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top Of Stack			
10111	Top of stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register; "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	*+ disp	
Scaled Index			
11100	Index, bytes	mode[Rn:B]	EA (mode) + Rn.
11101	Index, words	mode[Rn:W]	EA (mode) + 2×Rn.
11110	Index, double words	mode[Rn:D]	EA (mode) + 4×Rn.
11111	Index, quad words	mode[Rn:Q]	EA (mode) + 8×Rn. "Mode" and "n" are contained within the Index Byte. EA (mode) denotes the effective address generated using mode.

2.0 Architectural Description (Continued)

2.2.3 Instruction Set Summary

Table 2-2 presents a brief description of the NS32C016 instruction set. The Format column refers to the Instruction Format tables (Appendix A). The Instruction column gives the instruction as coded in assembly language, and the Description column provides a short description of the function provided by that instruction. Further details of the exact operations performed by each instruction may be found in the Series 32000 Instruction Set Reference Manual.

Notations:

i = Integer length suffix: B = Byte
W = Word
D = Double Word

f = Floating Point length suffix: F = Standard Floating
L = Long Floating

gen = General operand. Any addressing mode can be specified.

short = A 4-bit value encoded within the Basic Instruction (see Appendix A for encodings).

imm = Implied immediate operand. An 8-bit value appended after any addressing extensions.

disp = Displacement (addressing constant): 8, 16 or 32 bits. All three lengths legal.

reg = Any General Purpose Register: R0–R7.

areg = Any Dedicated/Address Register: SP, SB, FP, MOD, INTBASE, PSR, US (bottom 8 PSR bits).

mreg = Any Memory Management Status/Control Register.

creg = A Custom Slave Processor Register (Implementation Dependent).

cond = Any condition code, encoded as a 4-bit field within the Basic Instruction (see Appendix A for encodings).

TABLE 2-2. NS32C016 Instruction Set Summary

MOVES

Format	Operation	Operands	Description
4	MOVi	gen,gen	Move a value.
2	MOVQi	short,gen	Extend and move a signed 4-bit constant.
7	MOVMi	gen,gen,disp	Move multiple: disp bytes (1 to 16).
7	MOVZBW	gen,gen	Move with zero extension.
7	MOVZiD	gen,gen	Move with zero extension.
7	MOVXBW	gen,gen	Move with sign extension.
7	MOVXiD	gen,gen	Move with sign extension.
4	ADDR	gen,gen	Move effective address.

INTEGER ARITHMETIC

Format	Operation	Operands	Description
4	ADDi	gen,gen	Add.
2	ADDQi	short,gen	Add signed 4-bit constant.
4	ADDCi	gen,gen	Add with carry.
4	SUBi	gen,gen	Subtract.
4	SUBCi	gen,gen	Subtract with carry (borrow).
6	NEGi	gen,gen	Negate (2's complement).
6	ABSi	gen,gen	Take absolute value.
7	MULi	gen,gen	Multiply.
7	QUOi	gen,gen	Divide, rounding toward zero.
7	REMi	gen,gen	Remainder from QUO.
7	DIVi	gen,gen	Divide, rounding down.
7	MODi	gen,gen	Remainder from DIV (Modulus).
7	MEIi	gen,gen	Multiply to extended integer.
7	DEIi	gen,gen	Divide extended integer.

PACKED DECIMAL (BCD) ARITHMETIC

Format	Operation	Operands	Description
6	ADDPi	gen,gen	Add packed.
6	SUBPi	gen,gen	Subtract packed.

2.0 Architectural Description (Continued)

TABLE 2-2. NS32C016 Instruction Set Summary (Continued)

INTEGER COMPARISON

Format	Operation	Operands	Description
4	CMPi	gen,gen	Compare.
2	CMPQi	short,gen	Compare to signed 4-bit constant.
7	CMPMi	gen,gen,disp	Compare multiple: disp bytes (1 to 16).

LOGICAL AND BOOLEAN

Format	Operation	Operands	Description
4	ANDi	gen,gen	Logical AND.
4	ORi	gen,gen	Logical OR.
4	BICi	gen,gen	Clear selected bits.
4	XORi	gen,gen	Logical exclusive OR.
6	COMi	gen,gen	Complement all bits.
6	NOTi	gen,gen	Boolean complement: LSB only.
2	Scondi	gen	Save condition code (cond) as a Boolean variable of size i.

SHIFTS

Format	Operation	Operands	Description
6	LSHi	gen,gen	Logical shift, left or right.
6	ASHi	gen,gen	Arithmetic shift, left or right.
6	ROTi	gen,gen	Rotate, left or right.

BITS

Format	Operation	Operands	Description
4	TBITi	gen,gen	Test bit.
6	SBITi	gen,gen	Test and set bit.
6	SBITIi	gen,gen	Test and set bit, interlocked.
6	CBITi	gen,gen	Test and clear bit.
6	CBITIi	gen,gen	Test and clear bit, interlocked.
6	IBITi	gen,gen	Test and invert bit.
8	FFSi	gen,gen	Find first set bit.

BIT FIELDS

Bit fields are values in memory that are not aligned to byte boundaries. Examples are PACKED arrays and records used in Pascal. "Extract" instructions read and align a bit field. "Insert" instructions write a bit field from an aligned source.

Format	Operation	Operands	Description
8	EXTi	reg,gen,gen,disp	Extract bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented).
7	EXTSi	gen,gen,imm,imm	Extract bit field (short form).
7	INSSi	gen,gen,imm,imm	Insert bit field (short form).
8	CVTP	reg,gen,gen	Convert to bit field pointer.

ARRAYS

Format	Operation	Operands	Description
8	CHECKi	reg,gen,gen	Index bounds check.
8	INDEXi	reg,gen,gen	Recursive indexing step for multiple-dimensional arrays.

2.0 Architectural Description (Continued)

TABLE 2-2. NS32C016 Instruction Set Summary (Continued)

STRINGS

String instructions assign specific functions to the General Purpose Registers:

R4 — Comparison Value
R3 — Translation Table Pointer
R2 — String 2 Pointer
R1 — String 1 Pointer
R0 — Limit Count

Options on all string instructions are:

B (Backward): Decrement strong pointers after each step rather than incrementing.

U (Until match): End instruction if String 1 entry matches R4.

W (While match): End instruction if String 1 entry does not match R4.

All string instructions end when R0 decrements to zero.

Format	Operation	Operands	Description
5	MOVSi	options	Move string 1 to string 2.
	MOVST	options	Move string, translating bytes.
5	CMPSi	options	Compare string 1 to string 2.
	CMPST	options	Compare, translating string 1 bytes.
5	SKPSi	options	Skip over string 1 entries.
	SKPST	options	Skip, translating bytes for until/while.

JUMPS AND LINKAGE

Format	Operation	Operands	Description
3	JUMP	gen	Jump.
0	BR	disp	Branch (PC Relative).
0	Bcond	disp	Conditional branch.
3	CASEi	gen	Multiway branch.
2	ACBi	short,gen,disp	Add 4-bit constant and branch if non-zero.
3	JSR	gen	Jump to subroutine.
1	BSR	disp	Branch to subroutine.
1	CXP	disp	Call external procedure
3	CXPD	gen	Call external procedure using descriptor.
1	SVC		Supervisor call.
1	FLAG		Flag trap.
1	BPT		Breakpoint trap.
1	ENTER	[reg list], disp	Save registers and allocate stack frame (Enter Procedure).
1	EXIT	[reg list]	Restore registers and reclaim stack frame (Exit Procedure).
1	RET	disp	Return from subroutine.
1	RXP	disp	Return from external procedure call.
1	RETT	disp	Return from trap. (Privileged)
1	RETI		Return from interrupt. (Privileged)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save general purpose registers.
1	RESTORE	[reg list]	Restore general purpose registers.
2	LPRi	areg,gen	Load dedicated register. (Privileged if PSR or INTBASE)
2	SPRi	areg,gen	Store dedicated register. (Privileged if PSR or INTBASE)
3	ADJSPi	gen	Adjust stack pointer.
3	BISPSRi	gen	Set selected bits in PSR. (Privileged if not Byte length)
3	BICPSRi	gen	Clear selected bits in PSR. (Privileged if not Byte length)
5	SETCFG	[option list]	Set configuration register. (Privileged)

2.0 Architectural Description (Continued)

TABLE 2-2. NS32C016 Instruction Set Summary (Continued)

FLOATING POINT

Format	Operation	Operands	Description
11	MOVf	gen,gen	Move a floating point value.
9	MOVLF	gen,gen	Move and shorten a long value to standard.
9	MOVFL	gen,gen	Move and lengthen a standard value to long.
9	MOVif	gen,gen	Convert any integer to standard or long floating.
9	ROUNDfi	gen,gen	Convert to integer by rounding.
9	TRUNCfi	gen,gen	Convert to integer by truncating, toward zero.
9	FLOORfi	gen,gen	Convert to largest integer less than or equal to value.
11	ADDf	gen,gen	Add.
11	SUBf	gen,gen	Subtract.
11	MULf	gen,gen	Multiply.
11	DIVf	gen,gen	Divide.
11	CMPf	gen,gen	Compare.
11	NEGf	gen,gen	Negate.
11	ABSf	gen,gen	Take absolute value.
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

MEMORY MANAGEMENT

Format	Operation	Operands	Description
14	LMR	mreg,gen	Load memory management register. (Privileged)
14	SMR	mreg,gen	Store memory management register. (Privileged)
14	RDVAL	gen	Validate address for reading. (Privileged)
14	WRVAL	gen	Validate address for writing. (Privileged)
8	MOVSUi	gen,gen	Move a value from supervisor space to user space. (Privileged)
8	MOVUSi	gen,gen	Move a value from user space to supervisor space. (Privileged)

MISCELLANEOUS

Format	Operation	Operands	Description
1	NOP		No operation.
1	WAIT		Wait for interrupt.
1	DIA		Diagnose. Single-byte "Branch to Self" for hardware breakpointing. Not for use in programming.

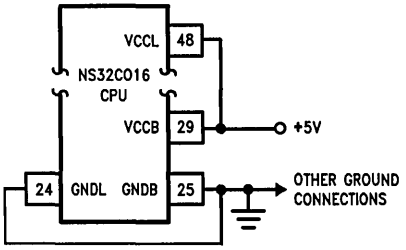
CUSTOM SLAVE

Format	Operation	Operands	Description
15.5	CCAL0c	gen,gen	Custom calculate.
15.5	CCAL1c	gen,gen	
15.5	CCAL2c	gen,gen	
15.5	CCAL3c	gen,gen	
15.5	CMOV0c	gen,gen	Custom move.
15.5	CMOV1c	gen,gen	
15.5	CMOV2c	gen,gen	
15.5	CMOV3c	gen,gen	
15.5	CCMP0c	gen,gen	Custom compare.
15.5	CCMP1c	gen,gen	
15.1	CCV0ci	gen,gen	Custom convert.
15.1	CCV1ci	gen,gen	
15.1	CCV2ci	gen,gen	
15.1	CCV3ic	gen,gen	
15.1	CCV4DQ	gen,gen	
15.1	CCV5QD	gen,gen	
15.1	LCSR	gen	Load custom status register.
15.1	SCSR	gen	Store custom status register.
15.0	CATST0	gen	Custom address/test. (Privileged)
15.0	CATST1	gen	(Privileged)
15.0	LCR	creg,gen	Load custom register. (Privileged)
15.0	SCR	creg,gen	Store custom register. (Privileged)

3.0 Functional Description

3.1 POWER AND GROUNDING

Power and ground connections for the NS32C016 are made on four pins. On-chip logic is connected to power through the logic power pin (VCCL, pin 48) and to ground through the logic ground pin (GNDL, pin 24). On-chip output drivers are connected to power through the buffer power pin (VCCP, pin 29) and to ground through the buffer ground pin (GNDB, pin 25). For optimal noise immunity, it is recommended that single conductors be connected directly from VCCL to VCCB and from GNDL to GNDB, as shown below (*Figure 3-1*).



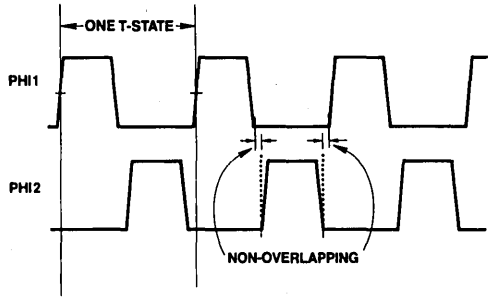
TL/EE/8525-9

FIGURE 3-1. Recommended Supply Connections

3.2 CLOCKING

The NS32C016 inputs clocking signals from the NS32C201 Timing Control Unit (TCU), which presents two non-overlapping phases of a single clock frequency. These phases are called PHI1 (pin 26) and PHI2 (pin 27). Their relationship to each other is shown in *Figure 3-2*.

Each rising edge of PHI1 defines a transition in the timing state ("T-State") of the CPU. One T-State represents the execution of one microinstruction within the CPU, and/or one step of an external bus transfer. See Section 4 for complete specifications of PHI1 and PHI2.



TL/EE/8525-10

FIGURE 3-2. Clock Timing Relationships

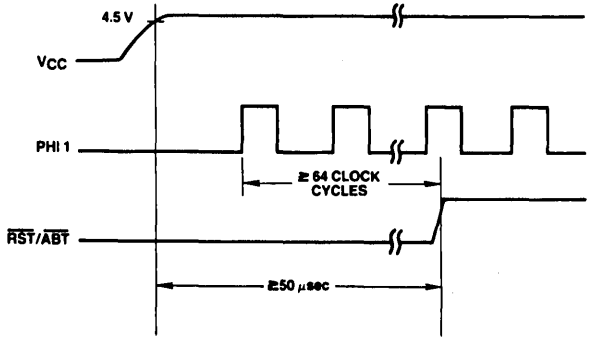
As the TCU presents signals with very fast transitions, it is recommended that the conductors carrying PHI1 and PHI2 be kept as short as possible, and that they not be connected anywhere except from the TCU to the CPU and, if present, the MMU. A TTL Clock signal (CTTL) is provided by the TCU for all other clocking.

3.3 RESETTING

The $\overline{RST}/\overline{ABT}$ pin serves both as a Reset for on-chip logic and as the Abort input for Memory-Managed systems. For its use as the Abort Command, see Section 3.5.4.

The CPU may be reset at any time by pulling the $\overline{RST}/\overline{ABT}$ pin low for at least 64 clock cycles. Upon detecting a reset, the CPU terminates instruction processing, resets its internal logic, and clears the Program Counter (PC) and Processor Status Register (PSR) to all zeroes.

On application of power, $\overline{RST}/\overline{ABT}$ must be held low for at least 50 μs after V_{CC} is stable. This is to ensure that all on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must also remain active



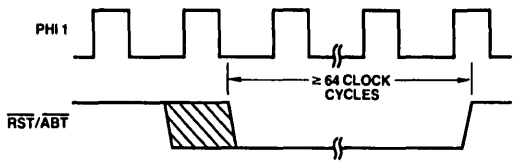
TL/EE/8525-11

FIGURE 3-3. Power-On Reset Requirements

3.0 Functional Description (Continued)

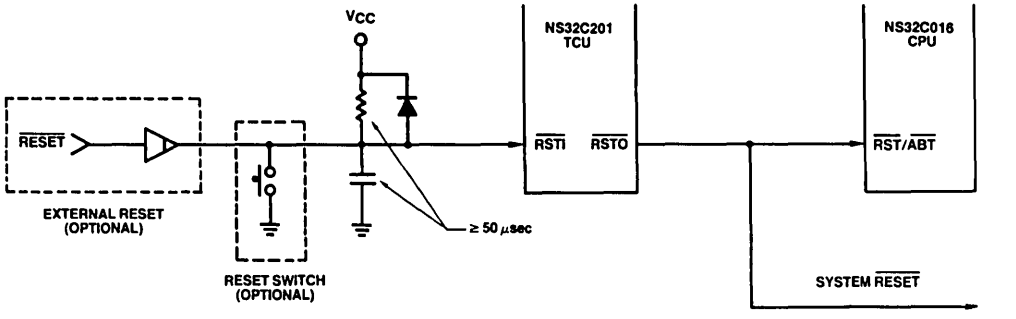
for not less than 64 clock cycles. The rising edge must occur while PHI1 is high. See *Figures 3-3* and *3-4.*

The NS32C201 Timing Control Unit (TCU) provides circuitry to meet the Reset requirements of the NS32C016 CPU. *Figure 3-5a* shows the recommended connections for a non-Memory-Managed system. *Figure 3-5b* shows the connections for a Memory-Managed system.



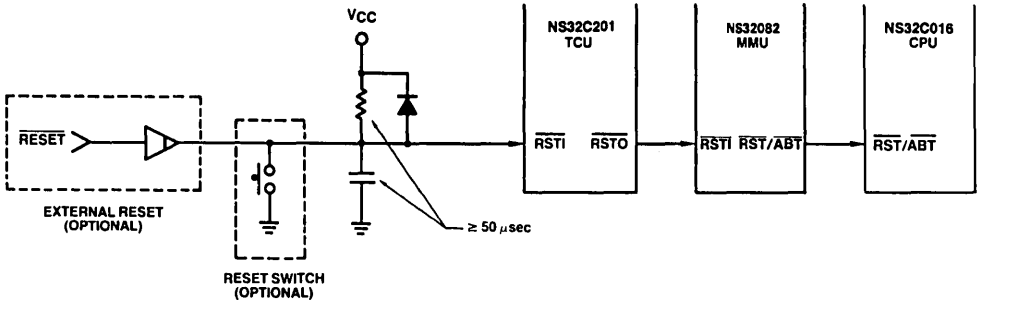
TL/EE/8525–12

FIGURE 3-4. General Reset Timing



TL/EE/8525–13

FIGURE 3-5a. Recommended Reset Connections, Non-Memory-Managed System



TL/EE/8525–14

FIGURE 3-5b. Recommended Reset Connections, Memory-Managed System

3.4 BUS CYCLES

The NS32C016 CPU has a strap option which defines the Bus Timing Mode as either With or Without Address Translation. This section describes only bus cycles under the No Address Translation option. For details of the use of the strap and of bus cycles with address translation, see Section 3.5.

The CPU will perform a bus cycle for one of the following reasons:

1) To write or read data, to or from memory or a peripheral interface device. Peripheral input and output are memory-mapped in the Series 32000 family.
2) To fetch instructions into the eight-byte instruction queue. This happens whenever the bus would otherwise be idle and the queue is not already full.
3) To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
4) To transfer information to or from a Slave Processor.

In terms of bus timing, cases 1 through 3 above are identical. For timing specifications, see Section 4. The only external difference between them is the four-bit code placed on the Bus Status pins (ST0–ST3). Slave Processor cycles differ in that separate control signals are applied (Section 3.4.6).

The sequence of events in a non-Slave bus cycle is shown in *Figure 3-7* for a Read cycle and *Figure 3-8* for a Write cycle. The cases shown assume that the selected memory or interface device is capable of communicating with the CPU at full speed. If it is not, then cycle extension may be requested through the RDY line (Section 3.4.1).

3.0 Functional Description (Continued)

A full-speed bus cycle is performed in four cycles of the PHI1 clock signal, labeled T1 through T4. Clock cycles not associated with a bus cycle are designated Ti (for "Idle").

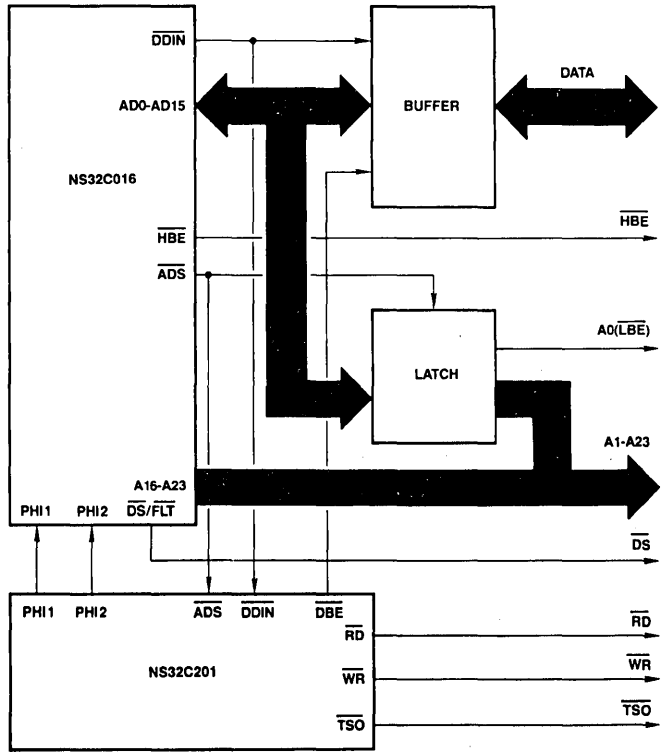
During T1, the CPU applies an address on pins AD0–AD15 and A16–A23. It also provides a low-going pulse on the $\overline{ADS}$ pin, which serves the dual purpose of informing external circuitry that a bus cycle is starting and of providing control to an external latch for demultiplexing Address bits 0–15 from the AD0–AD15 pins. See *Figure 3-6*. During this time also the status signals $\overline{DDIN}$, indicating the direction of the transfer, and $\overline{HBE}$, indicating whether the high byte (AD8–AD15) is to be referenced, become valid.

During T2 the CPU switches the Data Bus, AD0–AD15, to either accept or present data. Note that the signals A16–A23 remain valid, and need not be latched. It also starts the data strobe ($\overline{DS}$), signaling the beginning of the data transfer. Associated signals from the NS32C201 Timing Control Unit are also activated at this time: $\overline{RD}$ (Read Strobe) or $\overline{WR}$ (Write Strobe), $\overline{TSO}$ (Timing State Output, indicating that T2 has been reached) and $\overline{DBE}$ (Data Buffer Enable).

The T3 state provides for access time requirements, and it occurs at least once in a bus cycle. At the end of T2, on the falling edge of the PHI2 clock, the RDY (Ready) line is sampled to determine whether the bus cycle will be extended (Section 3.4.1).

If the CPU is performing a Read cycle, the Data Bus (AD0–AD15) is sampled at the falling edge of PHI2 of the last T3 state, see Section 4. Data must, however, be held at least until the beginning of T4. $\overline{DS}$ and $\overline{RD}$ are guaranteed not to go inactive before this point, so the rising edge of either of them may safely be used to disable the device providing the input data.

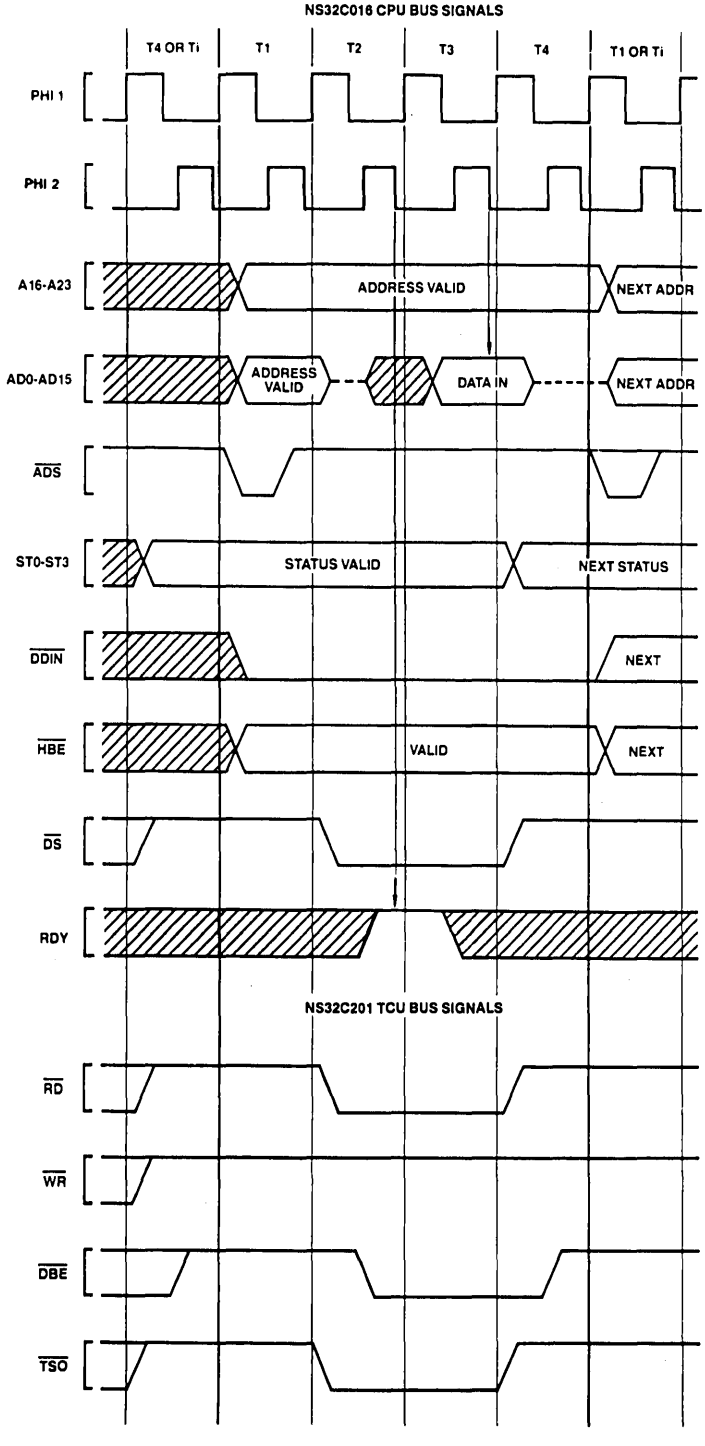
The T4 state finishes the bus cycle. At the beginning of T4, the $\overline{DS}$, $\overline{RD}$, or $\overline{WR}$, and $\overline{TSO}$ signals go inactive, and at the rising edge of PHI2, $\overline{DBE}$ goes inactive, having provided for necessary data hold times. Data during Write cycles remains valid from the CPU throughout T4. Note that the Bus Status lines (ST0–ST3) change at the beginning of T4, anticipating the following bus cycle (if any).



TL/EE/8525-15

FIGURE 3-6. Bus Connections

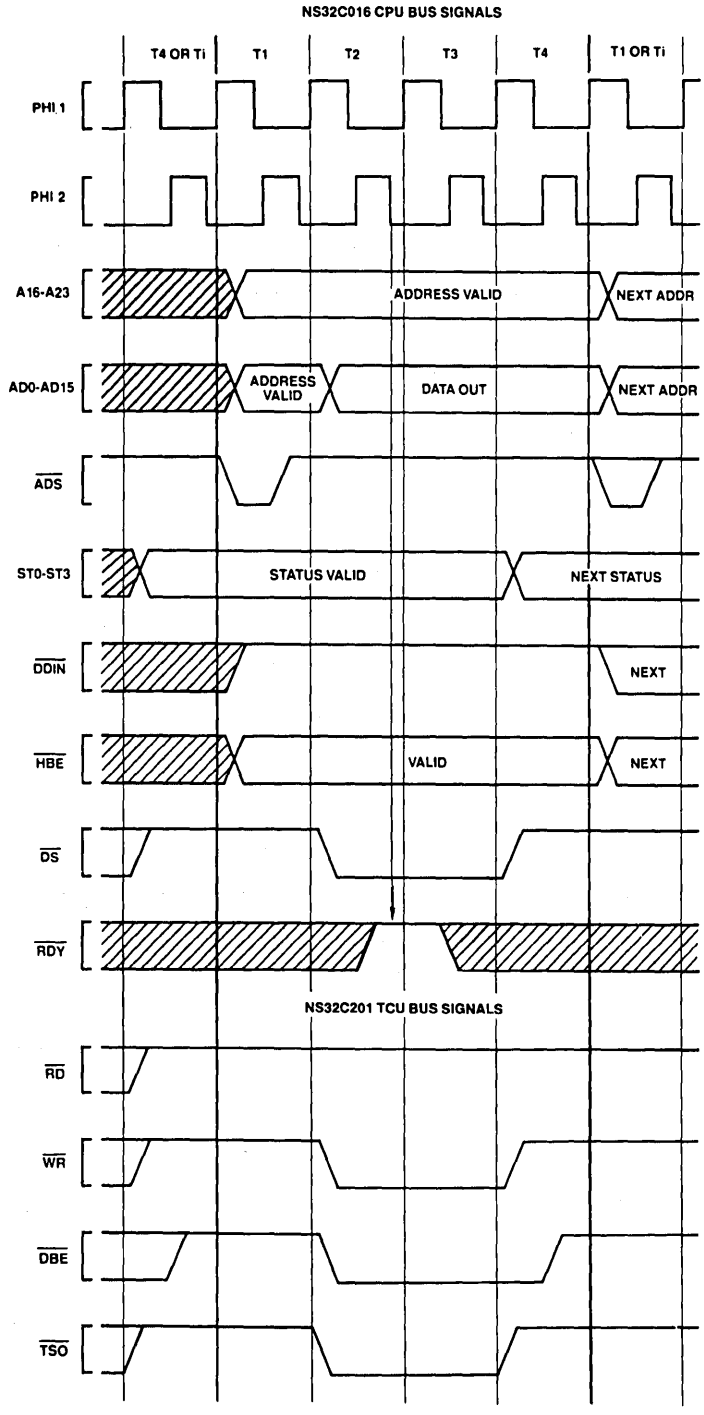
3.0 Functional Description (Continued)



TL/EE/8525-16

FIGURE 3-7. Read Cycle Timing

3.0 Functional Description (Continued)



TL/EE/8525–17

FIGURE 3-8. Write Cycle Timing

3.0 Functional Description (Continued)

3.4.1 Cycle Extension

To allow sufficient strobe widths and access times for any speed of memory or peripheral device, the NS32C016 provides for extension of a bus cycle. Any type of bus cycle except a Slave Processor cycle can be extended.

In *Figures 3-7* and *3-8*, note that during T3 all bus control signals from the CPU and TCU are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the RDY (Ready) pin.

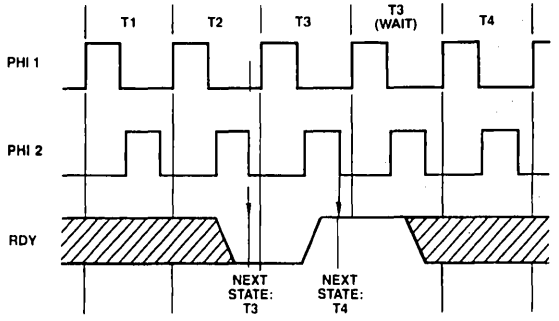
At the end of T2 on the falling edge of PHI2, the RDY line is sampled by the CPU. If RDY is high, the next T-states will be T3 and then T4, ending the bus cycle. If it is sampled low, then another T3 state will be inserted after the next T-state and the RDY line will again be sampled on the falling edge of PHI2. Each additional T3 state after the first is referred to as a "wait state." See *Figure 3-9*.

The RDY pin is driven by the NS32C201 Timing Control Unit, which applies WAIT States to the CPU as requested on three sets of pins:

1) $\overline{\text{CWAIT}}$ (Continues WAIT), which holds the CPU in WAIT states until removed.
2) $\overline{\text{WAIT1}}$, $\overline{\text{WAIT2}}$, $\overline{\text{WAIT4}}$, $\overline{\text{WAIT8}}$ (Collectively $\overline{\text{WAITn}}$), which may be given a four-bit binary value requesting a specific number of WAIT States from 0 to 15.
3) $\overline{\text{PER}}$ (Peripheral), which inserts five additional WAIT states and causes the TCU to reshape the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ strobes. This provides the setup and hold times required by most MOS peripheral interface devices.

Combinations of these various WAIT requests are both legal and useful. For details of their use, see the NS32C201 TCU Data Sheet.

Figure 3-10 illustrates a typical Read cycle, with two WAIT states requested through the TCU $\overline{\text{WAITn}}$ pins.



TL/EE/8525–18

FIGURE 3-9. RDY Pin Timing

3.4.2 Bus Status

The NS32C016 CPU presents four bits of Bus Status information on pins ST0–ST3. The various combinations on these pins indicate why the CPU is performing a bus cycle, or, if it is idle on the bus, then why it is idle.

Referring to *Figures 3-7* and *3-8*, note that Bus Status leads the corresponding Bus Cycle, going valid one clock cycle before T1, and changing to the next state at T4. This allows the system designer to fully decode the Bus Status and, if desired, latch the decoded signals before $\overline{\text{ADS}}$ initiates the Bus Cycle.

The Bus Status pins are interpreted as a four-bit value, with ST0 the least significant bit. Their values decode as follows:

0000 — The bus is idle because the CPU does not need to perform a bus access.

0001 — The bus is idle because the CPU is executing the WAIT instruction.

0010 — (Reserved for future use.)

0011 — The bus is idle because the CPU is waiting for a Slave Processor to complete an instruction.

0100 — Interrupt Acknowledge, Master.

The CPU is performing a Read cycle. To acknowledge receipt of a Non-Maskable Interrupt (on $\overline{\text{NMI}}$) it will read from address FFFF00_{16}, but will ignore any data provided.

To acknowledge receipt of a Maskable Interrupt (on $\overline{\text{INT}}$) it will read from address FFFE00_{16}, expecting a vector number to be provided from the Master NS32202 Interrupt Control Unit. If the vectoring mode selected by the last SETCFG instruction was Non-Vectored, then the CPU will ignore the value it has read and will use a default vector instead, having assumed that no NS32202 is present. See Section 3.4.5.

0101 — Interrupt Acknowledge, Cascaded.

The CPU is reading a vector number from a Cascaded NS32202 Interrupt Control Unit. The address provided is the address of the NS32202 Hardware Vector register. See Section 3.4.5.

0110 — End of Interrupt, Master.

The CPU is performing a Read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction. See Section 3.4.5.

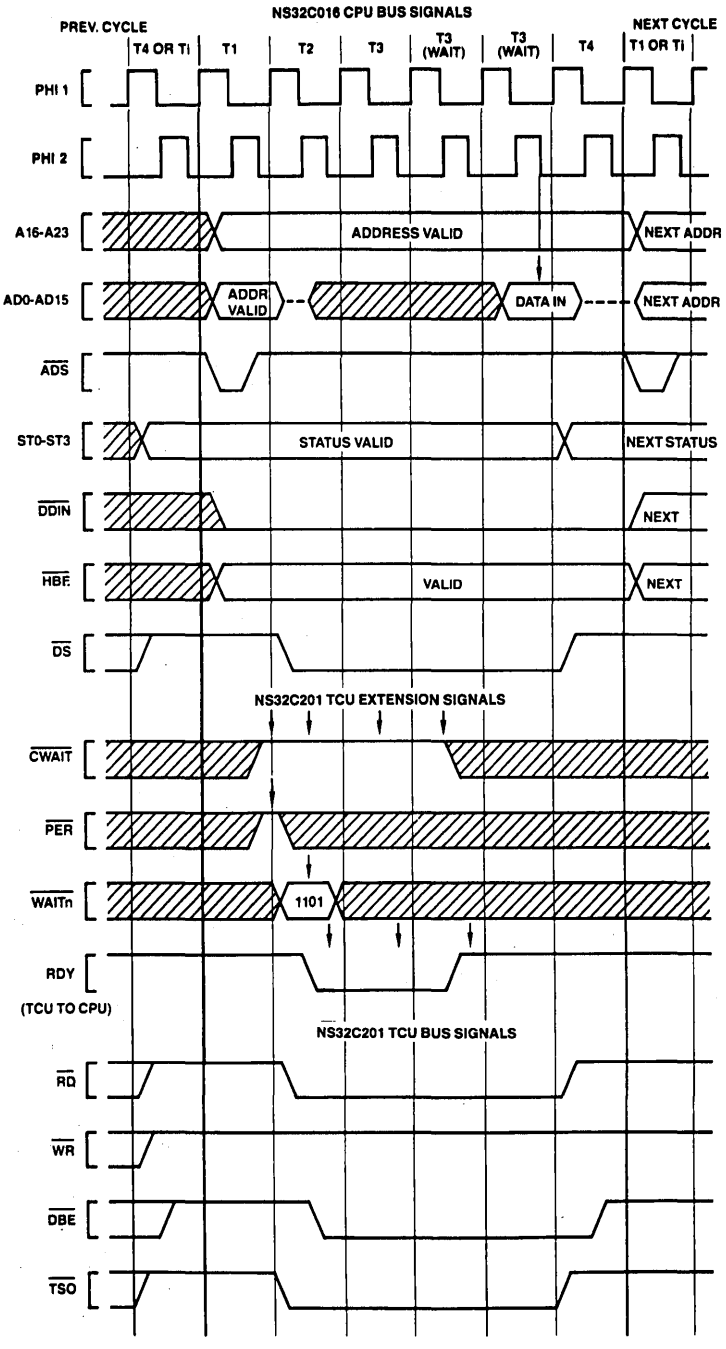
0111 — End of Interrupt, Cascaded.

The CPU is reading from a Cascaded Interrupt Control Unit to indicate that it is returning (through RETI) from an interrupt service routine requested by that unit. See Section 3.4.5.

1000 — Sequential Instruction Fetch.

The CPU is reading the next sequential word from the instruction stream into the Instruction Queue. It will do so whenever the bus would otherwise be idle and the queue is not already full.

3.0 Functional Description (Continued)



TL/EE/8525-19

FIGURE 3-10. Extended Cycle Example

Note: Arrows on $\overline{\text{CWAIT}}$, $\overline{\text{PER}}$, $\overline{\text{WAITn}}$ indicate points at which the TCU samples. Arrows on AD0–AD15 and RDY indicate points at which the CPU samples.

3.0 Functional Description (Continued)

1001 — Non-Sequential Instruction Fetch.
The CPU is performing the first fetch of instruction code after the Instruction Queue is purged. This will occur as a result of any jump or branch, or any interrupt or trap, or execution of certain instructions.

1010 — Data Transfer.
The CPU is reading or writing an operand of an instruction.

1011 — Read RMW Operand.
The CPU is reading an operand which will subsequently be modified and rewritten. If memory protection circuitry would not allow the following Write cycle, it must abort this cycle.

1100 — Read for Effective Address Calculation.
The CPU is reading information from memory in order to determine the Effective Address of an operand. This will occur whenever an instruction uses the Memory Relative or External addressing mode.

1101 — Transfer Slave Processor Operand.
The CPU is either transferring an instruction operand to or from a Slave Processor, or it is issuing the Operation Word of a Slave Processor instruction. See Section 3.9.1.

1110 — Read Slave Processor Status.
The CPU is reading a Status Word from a Slave Processor. This occurs after the Slave Processor has signalled completion of an instruction. The transferred word tells the CPU whether a trap should be taken, and in some instructions it presents new values for the CPU Processor Status Register bits N, Z, L or F. See Section 3.9.1.

1111 — Broadcast Slave ID.
The CPU is initiating the execution of a Slave Processor instruction. The ID Byte (first byte of the instruction) is sent to all Slave Processors, one of which will recognize it. From this point the CPU is communicating with only one Slave Processor. See Section 3.9.1.

3.4.3 Data Access Sequences

The 24-bit address provided by the NS32C016 is a byte address; that is, it uniquely identifies one of up to 16,777,216 eight-bit memory locations. An important feature of the NS32C016 is that the presence of a 16-bit data bus imposes no restrictions on data alignment; any data item, regardless of size, may be placed starting at any memory address. The NS32C016 provides a special control signal, High Byte Enable ($\overline{HBE}$), which facilitates individual byte addressing on a 16-bit bus.

Memory is organized as two eight-bit banks, each bank receiving the word address (A1–A23) in parallel. One bank, connected to Data Bus pins AD0–AD7, is enabled to respond to even byte addresses; i.e., when the least significant address bit (A0) is low. The other bank, connected to Data Bus pins AD8–AD15, is enabled when $\overline{HBE}$ is low. See *Figure 3-11.*

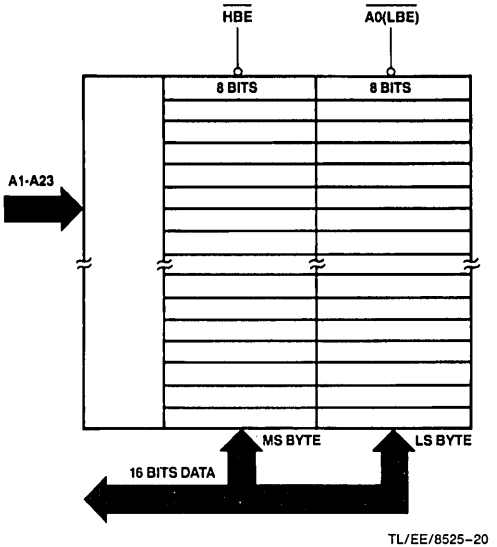


FIGURE 3-11. Memory Interface

Any bus cycle falls into one of three categories: Even Byte Access, Odd Byte Access, and Even Word Access. All accesses to any data type are made up of sequences of these cycles. Table 3-1 gives the state of A0 and $\overline{HBE}$ for each category.

TABLE 3-1. Bus Cycle Categories

Category	$\overline{HBE}$	A0
Even Byte	1	0
Odd Byte	0	1
Even Word	0	0

Accesses of operands requiring more than one bus cycle are performed sequentially, with no idle T-States separating them. The number of bus cycles required to transfer an operand depends on its size and its alignment (i.e., whether it starts on an even byte address or an odd byte address). Table 3-2 lists the bus cycle performed for each situation. For the timing of A0 and $\overline{HBE}$, see Section 3.4.

3.0 Functional Description (Continued)

TABLE 3-2. Access Sequences

A. Odd Word Access Sequence

Cycle	Type	Address	$\overline{HBE}$	A0	High Bus (BYTE 1)	Low Bus (BYTE 0)	
							← A
1	Odd Byte	A	0	1	Byte 0	Don't Care	
2	Even Byte	A+1	1	0	Don't Care	Byte 1	

B. Even Double-Word Access Sequence

Cycle	Type	Address	$\overline{HBE}$	A0	BYTE 3	BYTE 2	BYTE 1	BYTE 0	
									← A
1	Even Word	A	0	0			Byte 1	Byte 0	
2	Even Word	A+2	0	0			Byte 3	Byte 2	

C. Odd Double-Word Access Sequence

Cycle	Type	Address	$\overline{HBE}$	A0	BYTE 3	BYTE 2	BYTE 1	BYTE 0	
									← A
1	Odd Byte	A	0	1			Byte 0	Don't Care	
2	Even Word	A+1	0	0			Byte 2	Byte 1	
3	Even Byte	A+3	1	0			Don't Care	Byte 3	

D. Even Quad-Word Access Sequence

Cycle	Type	Address	$\overline{HBE}$	A0	BYTE 7	BYTE 6	BYTE 5	BYTE 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0	
													← A
1	Even Word	A	0	0							Byte 1	Byte 0	
2	Even Word	A+2	0	0							Byte 3	Byte 2	
Other bus cycles (instruction prefetch or slave) can occur here.													
3	Even Word	A+4	0	0							Byte 5	Byte 4	
4	Even Word	A+6	0	0							Byte 7	Byte 6	

E. Odd Quad-Word Access Sequence

Cycle	Type	Address	$\overline{HBE}$	A0	BYTE 7	BYTE 6	BYTE 5	BYTE 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0	
													← A
1	Odd Byte	A	0	1							Byte 0	Don't Care	
2	Even Word	A+1	0	0							Byte 2	Byte 1	
3	Even Byte	A+3	1	0							Don't Care	Byte 3	
Other bus cycles (instruction prefetch or slave) can occur here.													
4	Odd Byte	A+4	0	1							Byte 4	Don't Care	
5	Even Word	A+5	0	0							Byte 6	Byte 5	
6	Even Byte	A+7	1	0							Don't Care	Byte 7	

3.0 Functional Description (Continued)

3.4.3.1 Bit Accesses

The Bit Instructions perform byte accesses to the byte containing the designated bit. The Test and Set Bit instruction (SBIT), for example, reads a byte, alters it, and rewrites it, having changed the contents of one bit.

3.4.3.2 Bit Field Accesses

An access to a Bit Field in memory always generates a Double-Word transfer at the address containing the least significant bit of the field. The Double Word is read by an Extract instruction; an Insert instruction reads a Double Word, modifies it, and rewrites it.

3.4.3.3 Extending Multiply Accesses

The Extending Multiply Instruction (MEI) will return a result which is twice the size in bytes of the operand it reads. If the multiplicand is in memory, the most-significant half of the result is written first (at the higher address), then the least-significant half. This is done in order to support retry if this instruction is aborted.

3.4.4 Instruction Fetches

Instructions for the NS32C016 CPU are "prefetched"; that is, they are input before being needed into the next available entry of the eight-byte Instruction Queue. The CPU performs two types of Instruction Fetch cycles: Sequential and Non-Sequential. These can be distinguished from each other by their differing status combinations on pins ST0–ST3 (Section 3.4.2).

A Sequential Fetch will be performed by the CPU whenever the Data Bus would otherwise be idle and the Instruction Queue is not currently full. Sequential Fetches are always Even Word Read cycles (Table 3-1).

A Non-Sequential Fetch occurs as a result of any break in the normally sequential flow of a program. Any jump or branch instruction, a trap or an interrupt will cause the next Instruction Fetch cycle to be Non-Sequential. In addition, certain instructions flush the instruction queue, causing the next instruction fetch to display Non-Sequential status. Only the first bus cycle after a break displays Non-Sequential status, and that cycle is either an Even Word Read or an Odd Byte Read, depending on whether the destination address is even or odd.

3.4.5 Interrupt Control Cycles

Activating the $\overline{\text{INT}}$ or $\overline{\text{NMI}}$ pin on the CPU will initiate one or more bus cycles whose purpose is interrupt control rather than the transfer of instructions or data. Execution of the Return from Interrupt instruction (RETI) will also cause Interrupt Control bus cycles. These differ from instruction or data transfers only in the status presented on pins ST0–ST3. All Interrupt Control cycles are single-byte Read cycles.

This section describes only the Interrupt Control sequences associated with each interrupt and with the return from its service routine. For full details of the NS32C016 interrupt structure, see Section 3.8.

3.0 Functional Description (Continued)

TABLE 3-3. Interrupt Sequences

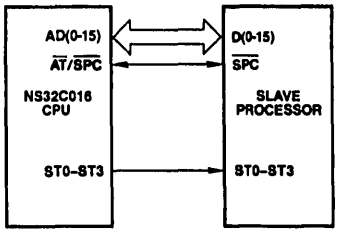
Cycle	Status	Address	$\overline{DDIN}$	$\overline{HBE}$	A0	High Bus	Low Bus
			A. Non-Maskable Interrupt Control Sequences.				
Interrupt Acknowledge							
1	0100	$FFFF00_{16}$	0	1	0	Don't Care	Don't Care
Interrupt Return							
None: Performed through Return from Trap (RETT) instruction.							
			B. Non-Vectored Interrupt Control Sequences.				
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Don't Care
Interrupt Return							
None: Performed through Return from Trap (RETT) instruction.							
			C. Vectored Interrupt Sequences: Non-Cascaded.				
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Vector: Range: 0–127
Interrupt Return							
1	0110	$FFFE00_{16}$	0	1	0	Don't Care	Vector: Same as in Previous Int. Ack. Cycle
			D. Vectored Interrupt Sequences: Cascaded.				
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Cascade Index: range −16 to −1
(The CPU here uses the Cascade Index to find the Cascade Address.)							
2	0101	Cascade Address	0	1 or 0*	0 or 1*	Vector, range 0–255; on appropriate half of Data Bus for even/odd address	
Interrupt Return							
1	0110	$FFFE00_{16}$	0	1	0	Don't Care	Cascade Index: same as in previous Int. Ack. Cycle
(The CPU here uses the Cascade Index to find the Cascade Address.)							
2	0111	Cascade Address	0	1 or 0*	0 or 1*	Don't Care	Don't Care

* If the Cascaded ICU Address is Even (A0 is low), then the CPU applies $\overline{HBE}$ high and reads the vector number from bits 0–7 of the Data Bus.
If the address is Odd (A0 is high), then the CPU applies $\overline{HBE}$ low and reads the vector number from bits 8–15 of the Data Bus. The vector number may be in the range 0–255.

3.0 Functional Description (Continued)

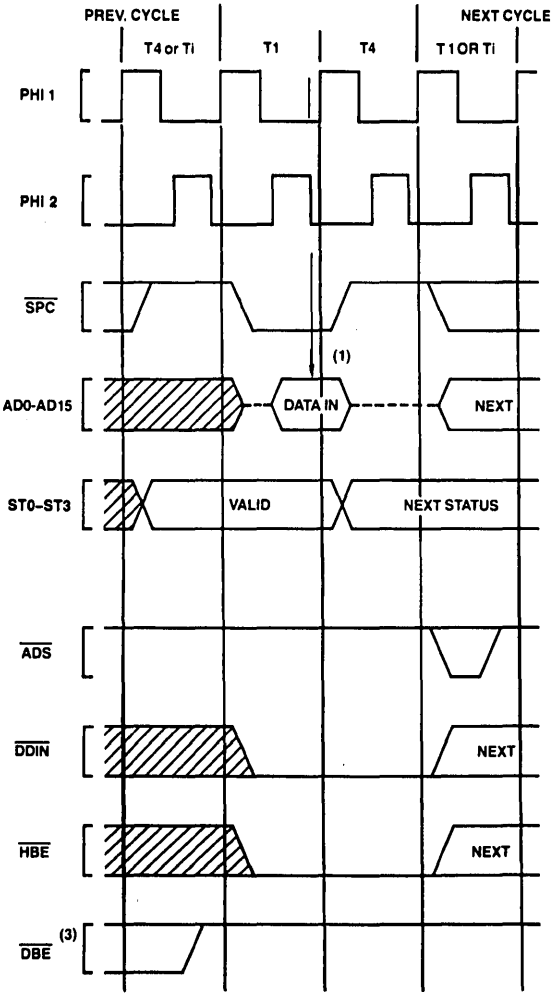
3.4.6 Slave Processor Communication

In addition to its use as the Address Translation strap (Section 3.5.1), the $\overline{AT}/\overline{SPC}$ pin is used as the data strobe for Slave Processor transfers. In this role, it is referred to as Slave Processor Control ($\overline{SPC}$). In a Slave Processor bus cycle, data is transferred on the Data Bus (AD0–AD15), and the status lines ST0–ST3 are monitored by each Slave Processor in order to determine the type of transfer being performed. $\overline{SPC}$ is bidirectional, but is driven by the CPU during all Slave Processor bus cycles. See Section 3.9 for full protocol sequences.



TL/EE/8525-21

FIGURE 3-12. Slave Processor Connections



TL/EE/8525-22

Notes:

(1) CPU samples Data Bus here.

(2) $\overline{DBE}$ and all other NS32C201 TCU bus signals remain inactive because no $\overline{ADS}$ pulse is received from the CPU.

FIGURE 3-13. CPU Read from Slave Processor

3.0 Functional Description (Continued)

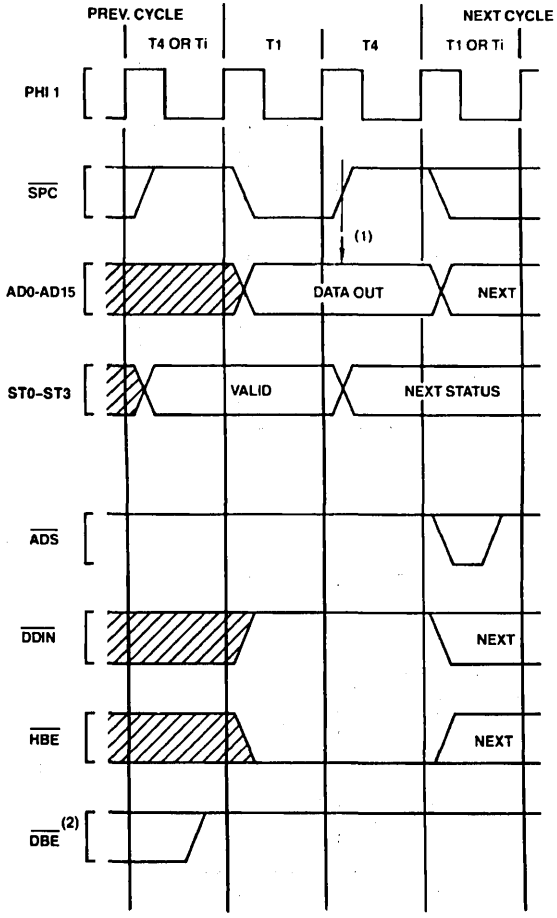
3.4.6.1 Slave Processor Bus Cycles

A Slave Processor bus cycle always takes exactly two clock cycles, labeled T1 and T4 (see *Figures 3-13* and *3-14*). During a Read cycle $\overline{SPC}$ is active from the beginning of T1 to the beginning of T4, and the data is sampled at the end of T1. The Cycle Status pins lead the cycle by one clock period, and are sampled at the leading edge of $\overline{SPC}$. During a Write cycle, the CPU applies data and activates $\overline{SPC}$ at T1, removing $\overline{SPC}$ at T4. The Slave Processor latches status on the leading edge of $\overline{SPC}$ and latches data on the trailing edge.

Since the CPU does not pulse the Address Strobe ($\overline{ADS}$), no bus signals are generated by the NS32C201 Timing Control Unit. The direction of a transfer is determined by the sequence ("protocol") established by the instruction under execution; but the CPU indicates the direction on the $\overline{DDIN}$ pin for hardware debugging purposes.

3.4.6.2 Slave Operand Transfer Sequences

A Slave Processor operand is transferred in one or more Slave bus cycles. A Byte operand is transferred on the least-significant byte of the Data Bus (AD0–AD7), and a Word operand is transferred on the entire bus. A Double Word is transferred in a consecutive pair of bus cycles, least-significant word first. A Quad Word is transferred in two pairs of Slave cycles, with other bus cycles possibly occurring between them. The word order is from least-significant word to most-significant.



TL/EE/8525–23

Notes:

(1) Slave Processor samples Data Bus here.

(2) $\overline{DBE}$, being provided by the NS32C201 TCU, remains inactive due to the fact that no pulse is presented on $\overline{ADS}$. $\overline{TCU}$ signals $\overline{RD}$, $\overline{WR}$ and $\overline{TSO}$ also remain inactive.

FIGURE 3-14. CPU Write to Slave Processor

3.0 Functional Description (Continued)

3.5 MEMORY MANAGEMENT OPTION

The NS32C016 CPU, in conjunction with the NS32082 Memory Management Unit (MMU), provides full support for address translation, memory protection, and memory allocation techniques up to and including Virtual Memory.

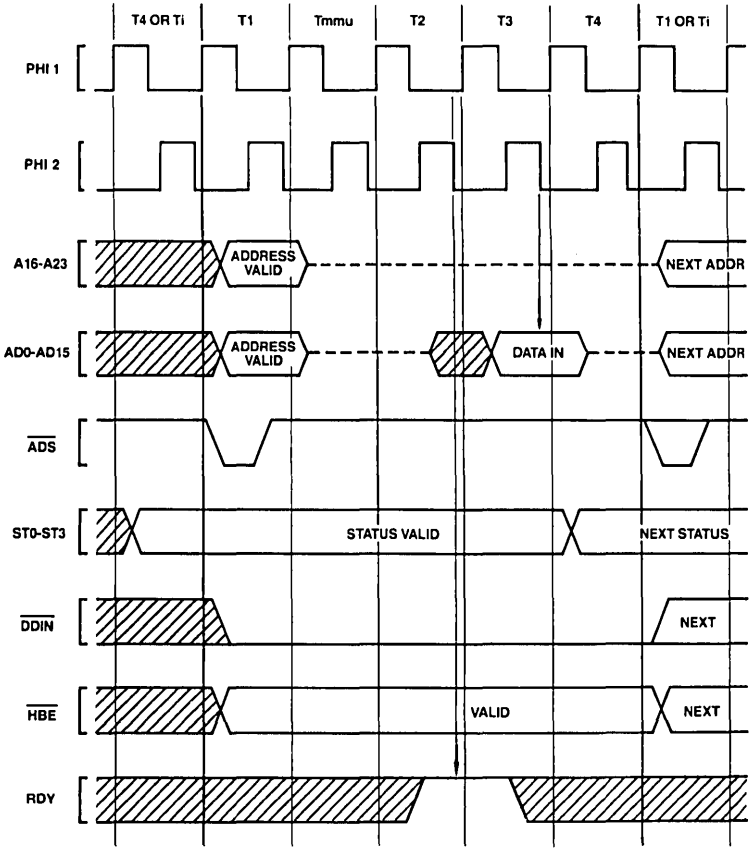
3.5.1 Address Translation Strap

The Bus Interface Control section of the NS32C016 CPU has two bus timing modes: With or Without Address Translation. The mode of operation is selected by the CPU by sampling the $\overline{AT}/\overline{SPC}$ (Address Translation/Slave Processor Control) pin on the rising edge of the $\overline{RST}$ (Reset) pulse. If $\overline{AT}/\overline{SPC}$ is sampled as high, the bus timing is as previously described in Section 3.4. If it is sampled as low, two changes occur:

1) An extra clock cycle, Tmmu, is inserted into all bus cycles except Slave Processor transfers.
2) The $\overline{DS}/\overline{FLT}$ pin changes in function from a Data Strobe output ($\overline{DS}$) to a Float Command input ($\overline{FLT}$).

The NS32082 MMU will itself pull the CPU $\overline{AT}/\overline{SPC}$ pin low when it is reset. In non-Memory-Managed systems this pin should be pulled up to V_{CC} through a 10 kΩ resistor.

Note that the Address Translation strap does not specifically declare the presence of an NS32082 MMU, but only the



TL/EE/8525-24

FIGURE 3-15. Read Cycle with Address Translation (CPU Action)

3.0 Functional Description (Continued)

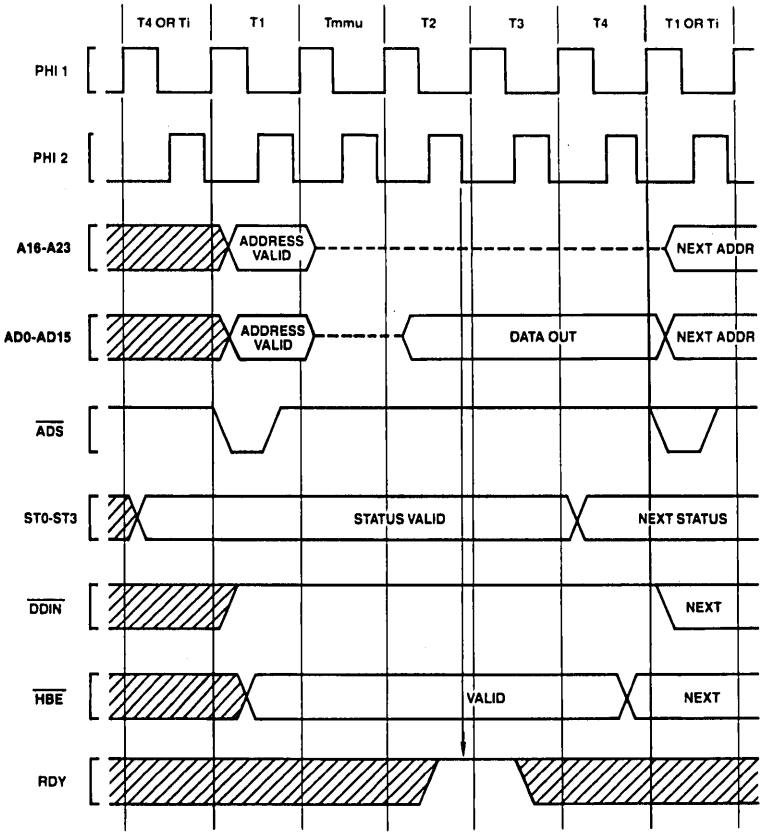
presence of external address translation circuitry. MMU instructions will still trap as being undefined unless the SETCFG (Set Configuration) instruction is executed to declare the MMU instruction set valid. See Section 2.1.3.

3.5.2 Translated Bus Timing

Figures 3-15 and *3-16* illustrate the CPU activity during a Read cycle and a Write cycle in Address Translation mode. The additional T-State, Tmmu, is inserted between T1 and T2. During this time the CPU places AD0–AD15 and A16–A23 into the TRI-STATE® mode, allowing the MMU to assert the translated address and issue the physical address strobe $\overline{PAV}$. T2 through T4 of the cycle are identical to their counter-parts without Address Translation, with the exception that the CPU Address lines A16–A23 remain in the TRI-STATE condition. This allows the MMU to continue asserting the translated address on those pins.

Note that in order for the NS32082 MMU to operate correctly, it must be set to the 32C016 mode by forcing A24 high during reset.

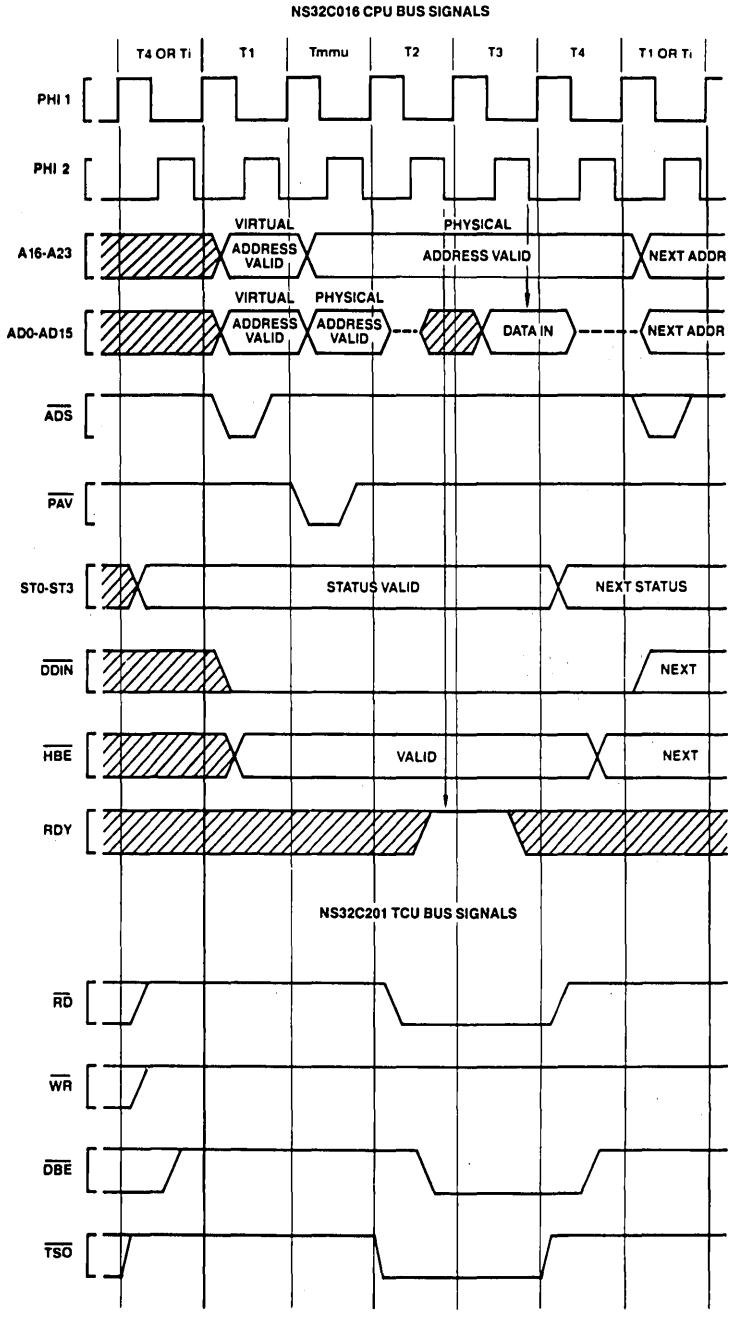
Figures 3-17 and *3-18* show a Read cycle and a Write cycle as generated by the 32C016/32082/32C201 group. Note that with the CPU $\overline{ADS}$ signal going only to the MMU, and with the MMU $\overline{PAV}$ signal substituting for $\overline{ADS}$ everywhere else, Tmmu through T4 look exactly like T1 through T4 in a non-Memory-Managed system. For the connection diagram, see Appendix B.



TL/EE/8525–25

FIGURE 3-16. Write Cycle with Address Translation (CPU Action)

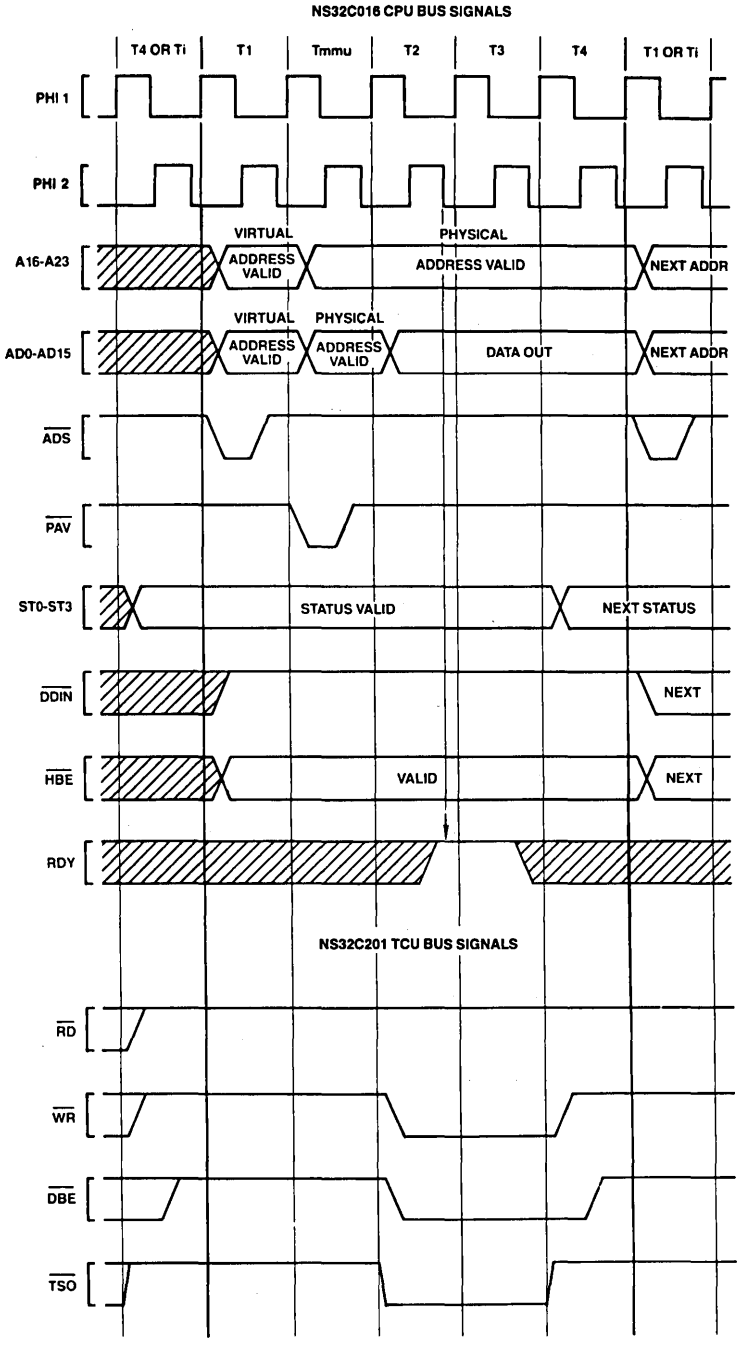
3.0 Functional Description (Continued)



TL/EE/8525–26

FIGURE 3-17. Memory-Managed Read Cycle

3.0 Functional Description (Continued)



TL/EE/8525-27

FIGURE 3-18. Memory-Managed Write Cycle

3.0 Functional Description (Continued)

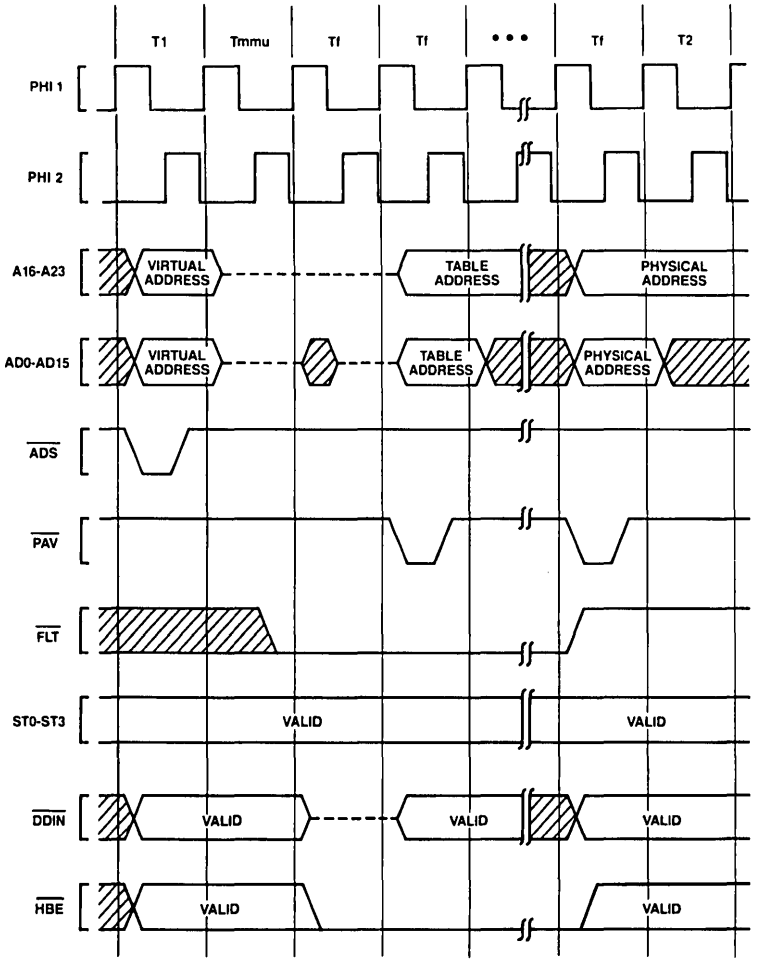
3.5.3 The $\overline{FLT}$ (Float) Pin

The $\overline{FLT}$ pin is used by the CPU for address translation support. Activating $\overline{FLT}$ during Tmmu causes the CPU to wait longer than Tmmu for address translation and validation. This feature is used occasionally by the NS32082 MMU in order to update its internal translation Look-Aside Buffer (TLB) from page tables in memory, or to update certain status bits within them.

Figure 3-19 shows the effects of $\overline{FLT}$. Upon sampling $\overline{FLT}$ low, late in Tmmu, the CPU enters idle T-States (Tf) during which it:

1) Sets AD0–AD15, A16–A23 and $\overline{DDIN}$ to the TRI-STATE condition ("floating").
2) Sets $\overline{HBE}$ low.
3) Suspends further internal processing of the current instruction. This ensures that the current instruction remains abortable with retry. (See $\overline{RST}/\overline{ABT}$ description, Section 3.5.4.)

Note that the AD0–AD15 pins may be briefly asserted during the first idle T-State. The above conditions remain in effect until $\overline{FLT}$ again goes high. See the Timing Specifications, Section 4.



TL/EE/8525-28

FIGURE 3-19. $\overline{FLT}$ Timing

3.0 Functional Description (Continued)

3.5.4 Aborting Bus Cycles

The $\overline{RST}/\overline{ABT}$ pin, apart from its Reset function (Section 3.3), also serves as the means to "abort," or cancel, a bus cycle and the instruction, if any, which initiated it. An Abort request is distinguished from a Reset in that the $\overline{RST}/\overline{ABT}$ pin is held active for only one clock cycle.

If $\overline{RST}/\overline{ABT}$ is pulled low during Tmmu or Tf, this signals that the cycle must be aborted. The CPU itself will enter T2 and then Ti, thereby terminating the cycle. Since it is the MMU $\overline{PAV}$ signal which triggers a physical cycle, the rest of the system remains unaware that a cycle was started.

The NS32082 MMU will abort a bus cycle for either of two reasons:

1) The CPU is attempting to access a virtual address which is not currently resident in physical memory. The reference page must be brought into physical memory from mass storage to make it accessible to the CPU.
2) The CPU is attempting to perform an access which is not allowed by the protection level assigned to that page.

When a bus cycle is aborted by the MMU, the instruction that caused it to occur is also aborted in such a manner that it is guaranteed re-executable later. The information that is changed irrecoverably by such a partly-executed instruction does not affect its re-execution.

3.5.4.1 The Abort Interrupt

Upon aborting an instruction, the CPU immediately performs an interrupt through the ABT vector in the Interrupt Table (see Section 3.8). The Return Address pushed on the Interrupt Stack is the address of the aborted instruction, so that a Return from Trap (RETT) instruction will automatically retry it.

The one exception to this sequence occurs if the aborted bus cycle was an instruction prefetch. If so, it is not yet certain that the aborted prefetched code is to be executed. Instead of causing an interrupt, the CPU only aborts the bus cycle, and stops prefetching. If the information in the Instruction Queue runs out, meaning that the instruction will actually be executed, the ABT interrupt will occur, in effect aborting the instruction that was being fetched.

3.5.4.2 Hardware Considerations

In order to guarantee instruction retry, certain rules must be followed in applying an Abort to the CPU. These rules are followed by the NS32082 Memory Management Unit.

1) If $\overline{FLT}$ has not been applied to the CPU, the Abort pulse must occur during or before Tmmu. See the Timing Specifications, *Figure 4-23.*
2) If $\overline{FLT}$ has been applied to the CPU, the Abort pulse must be applied before the T-State in which $\overline{FLT}$ goes inactive. The CPU will not actually respond to the Abort command until $\overline{FLT}$ is removed. See *Figure 4-24.*
3) The Write half of a Read-Modify-Write operand access may not be aborted. The CPU guarantees that this will never be necessary for Memory Management funtions by applying a special RMW status (Status Code 1011) during the Read half of the access. When the CPU presents RMW status, that cycle must be aborted if it would be illegal to write to any of the accessed addresses.

If $\overline{RST}/\overline{ABT}$ is pulsed at any time other than as indicated above, it will abort either the instruction currently under execution or the next instruction and will act as a very high-priority interrupt. However, the program that was running at the time is not guaranteed recoverable.

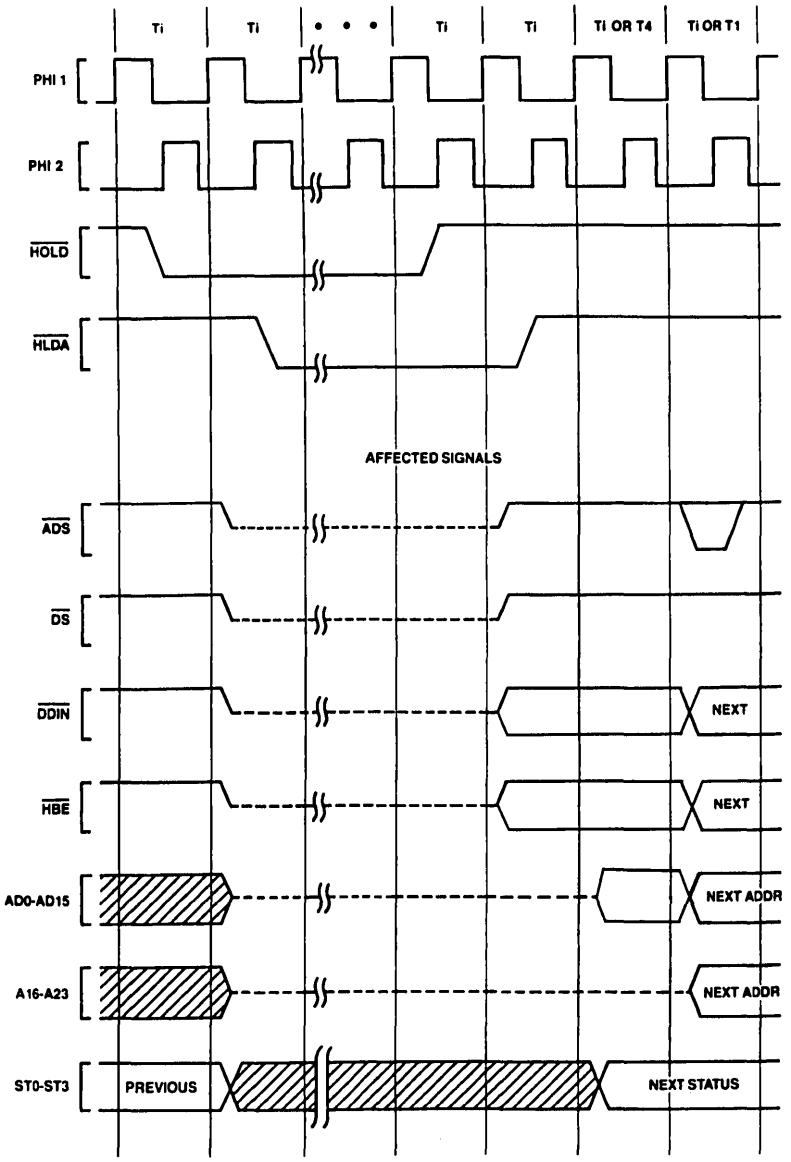
3.6 BUS ACCESS CONTROL

The NS32C016 CPU has the capability of relinquishing its access to the bus upon request from a DMA device or another CPU. This capability is implemented on the $\overline{HOLD}$ (Hold Request) and $\overline{HLDA}$ (Hold Acknowledge) pins. By asserting $\overline{HOLD}$ low, an external device requests access to the bus. On receipt of $\overline{HLDA}$ from the CPU, the device may perform bus cycles, as the CPU at this point has set the AD0–AD15, A16–A23, $\overline{ADS}$, $\overline{DDIN}$ and $\overline{HBE}$ pins to the TRI-STATE condition. To return control of the bus to the CPU, the device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

How quickly the CPU releases the bus depends on whether it is idle on the bus at the time the $\overline{HOLD}$ request is made, as the CPU must always complete the current bus cycle. *Figure 3-20* shows the timing sequence when the CPU is idle. In this case, the CPU grants the bus during the immediately following clock cycle. *Figure 3-21* shows the sequence if the CPU is using the bus at the time that the $\overline{HOLD}$ request is made. If the request is made during or before the clock cycle shown (two clock cycles before T4), the CPU will release the bus during the clock cycle following T4. If the request occurs closer to T4, the CPU may already have decided to initiate another bus cycle. In that case it will not grant the bus until after the next T4 state. Note that this situation will also occur if the CPU is idle on the bus but has initiated a bus cycle internally.

In a Memory-Managed system, the $\overline{HLDA}$ signal is connected in a daisy-chain through the NS32082, so that the MMU can release the bus if it is using it.

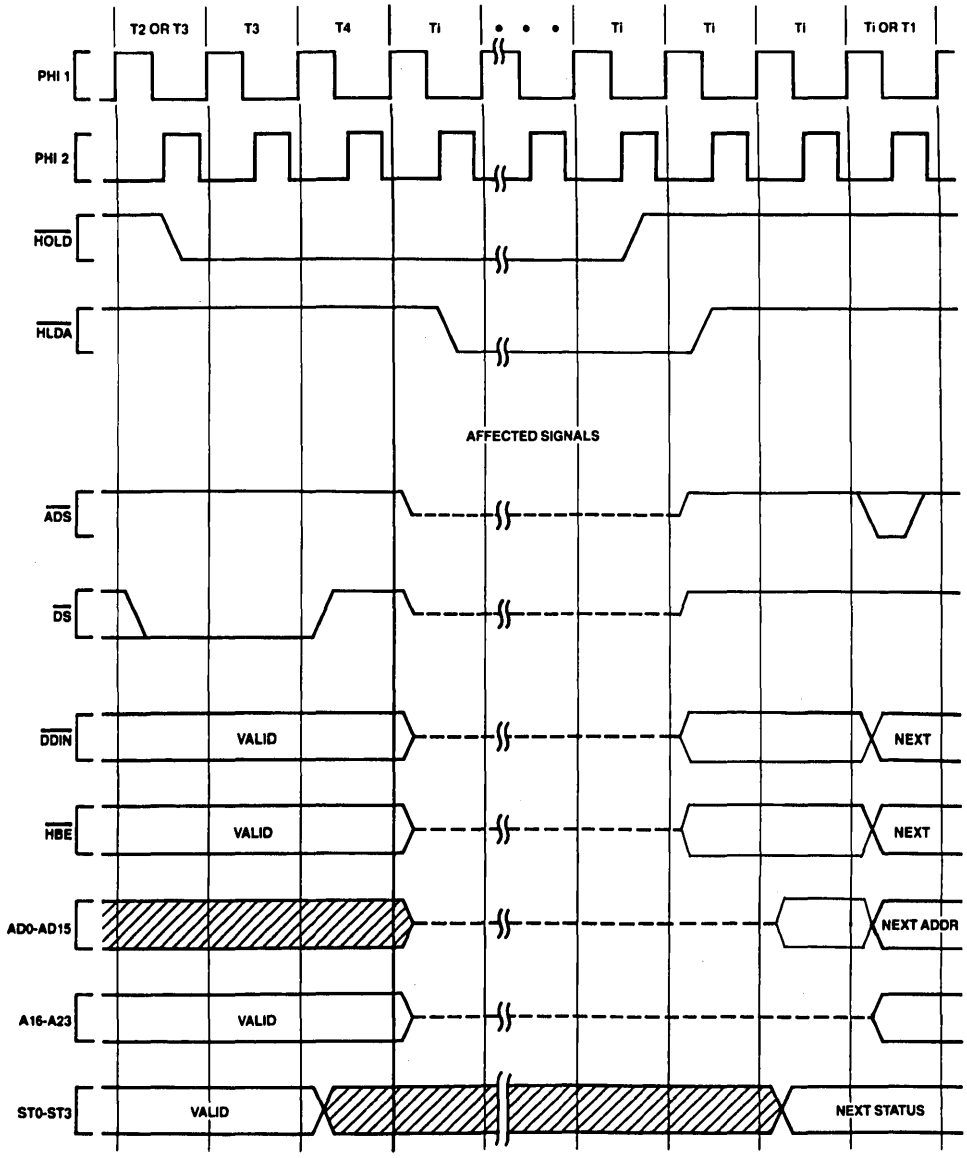
3.0 Functional Description (Continued)



TL/EE/8525-29

FIGURE 3-20. HOLD Timing, Bus Initially Idle

3.0 Functional Description (Continued)



TL/EE/8525–30

FIGURE 3-21. HOLD Timing, Bus Initially Not Idle

3.0 Functional Description (Continued)

3.7 INSTRUCTION STATUS

In addition to the four bits of Bus Cycle status (ST0–ST3), the NS32C016 CPU also presents Instruction Status information on three separate pins. These pins differ from ST0–ST3 in that they are synchronous to the CPU's internal instruction execution section rather than to its bus interface section.

$\overline{PFS}$ (Program Flow Status) is pulsed low as each instruction begins execution. It is intended for debugging purposes, and is used that way by the NS32082 Memory Management Unit.

U/$\overline{S}$ originates from the U bit of the Processor Status Register, and indicates whether the CPU is currently running in User or Supervisor mode. It is sampled by the MMU for mapping, protection and debugging purposes. Although it is not synchronous to bus cycles, there are guarantees on its validity during any given bus cycle. See the Timing Specifications, *Figure 4-22.*

$\overline{ILO}$ (Interlocked Operation) is activated during an SBITI (Set Bit, Interlocked) or CBITI (Clear Bit, Interlocked) instruction. It is made available to external bus arbitration circuitry in order to allow these instructions to implement the semaphore primitive operations for multi-processor communication and resource sharing. As with the U/$\overline{S}$ pin, there are guarantees on its validity during the operand accesses performed by the instructions. See the Timing Specification Section, *Figure 4-20.*

3.8 NS32C016 INTERRUPT STRUCTURE

$\overline{INT}$, on which maskable interrupts may be requested,

$\overline{NMI}$, on which non-maskable interrupts may be requested, and

$\overline{RST}$/$\overline{ABT}$, which may be used to abort a bus cycle and any associated instruction. See Section 3.5.4.

In addition, there is a set of internally-generated "traps" which cause interrupt service to be performed as a result either of exceptional conditions (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., the Supervisor Call instruction).

3.8.1 General Interrupt/Trap Sequence

Upon receipt of an interrupt or trap request, the CPU goes through three major steps:

1) Adjustment of Registers.

   Depending on the source of the interrupt or trap, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR) and the currently-selected Stack Pointer (SP). A copy of the PSR is made, and the PSR is then set to reflect Supervisor Mode and selection of the Interrupt Stack.

2) Vector Acquisition.

   A Vector is either obtained from the Data Bus or is supplied by default.

3) Service Call.

   The Vector is used as an index into the Interrupt Dispatch Table, whose base address is taken from the CPU Interrupt Base (INTBASE) Register. See *Figure 3-22.* A 32-bit External Procedure Descriptor is read from the table entry, and an External Procedure Call is performed using it. The MOD Register (16 bits) and Program Counter (32 bits) are pushed on the Interrupt Stack.

This process is illustrated in *Figure 3-23,* from the viewpoint of the programmer.

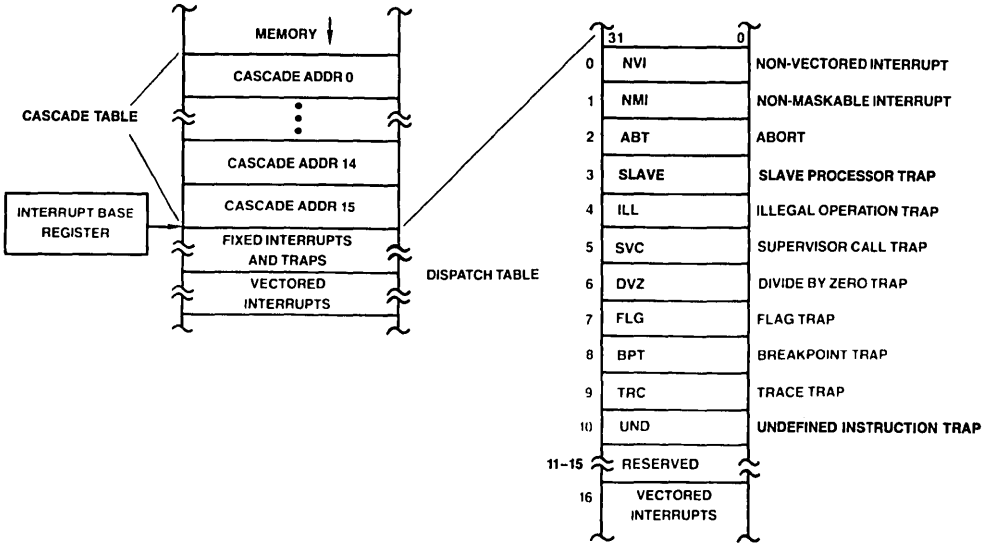
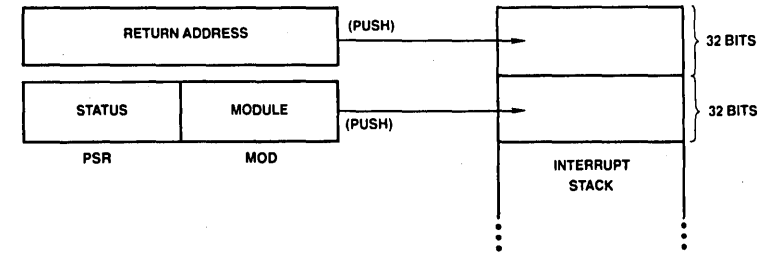
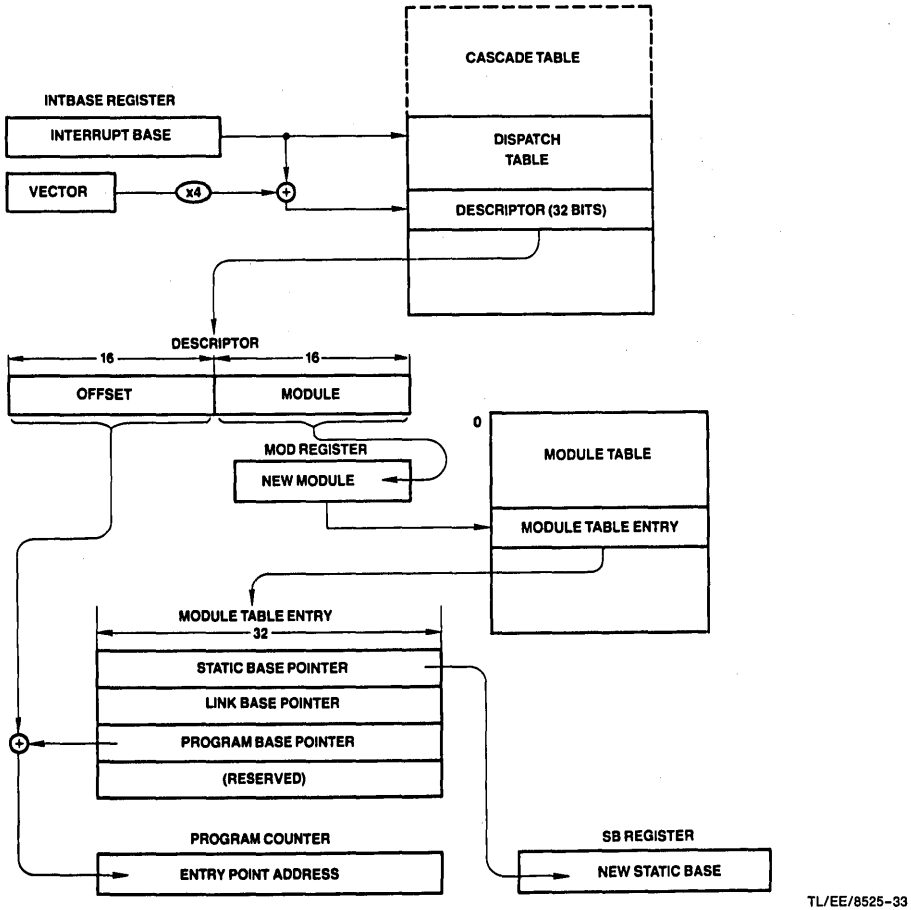


FIGURE 3-22. Interrupt Dispatch and Cascade Tables

3.0 Functional Description (Continued)



TL/EE/8525-32



TL/EE/8525-33

FIGURE 3-23. Interrupt/Trap Service Routine Calling Sequence

3.0 Functional Description (Continued)

3.8.2 Interrupt/Trap Return

To return control to an interrupted program, one of two instructions is used. The RETT (Return from Trap) instruction (*Figure 3-24*) restores the PSR, MOD, PC and SB registers to their previous contents and, since traps are often used deliberately as a call mechanism for Supervisor Mode procedures, it also discards a specified number of bytes from the original stack as surplus parameter space. RETT is used to return from any trap or interrupt except the Maskable Interrupt. For this, the RETI (Return from Interrupt) instruction is used, which also informs any external Interrupt Control Units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not pop parameters. See *Figure 3-25*.

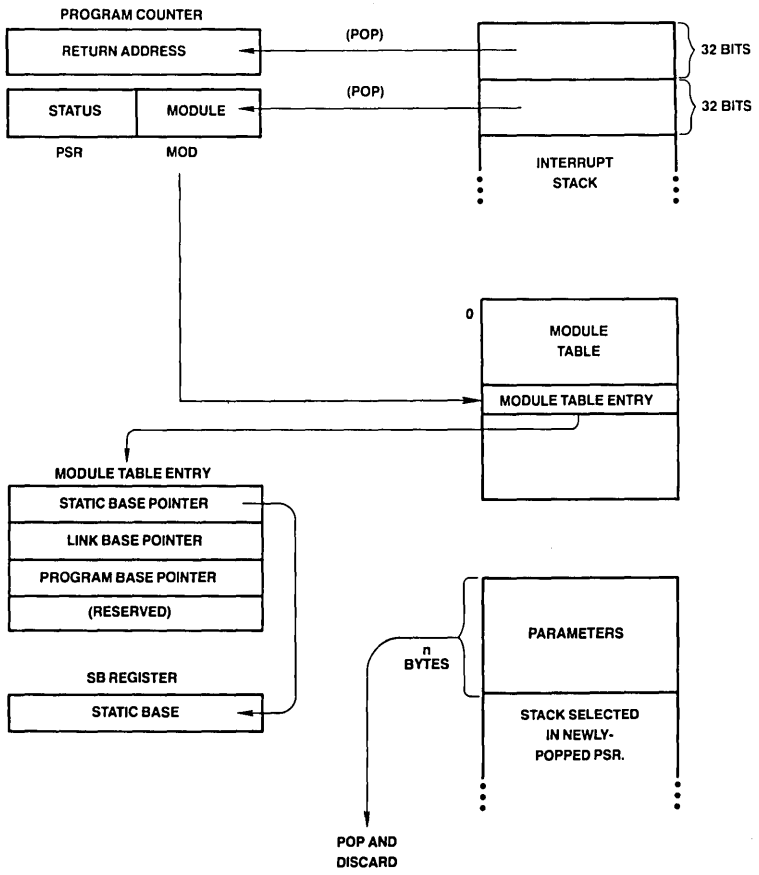
3.8.3 Maskable Interrupts (The $\overline{\text{INT}}$ Pin)

The $\overline{\text{INT}}$ pin is a level-sensitive input. A continuous low level is allowed for generating multiple interrupt requests. The input is maskable, and is therefore enabled to generate interrupt requests only while the Processor Status Register I bit is set. The I bit is automatically cleared during service of an $\overline{\text{INT}}$, $\overline{\text{NMI}}$ or Abort request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

The $\overline{\text{INT}}$ pin may be configured via the SETCFG instruction as either Non-Vectored (CFG Register bit I=0) or Vectored (bit I=1).

3.8.3.1 Non-Vectored Mode

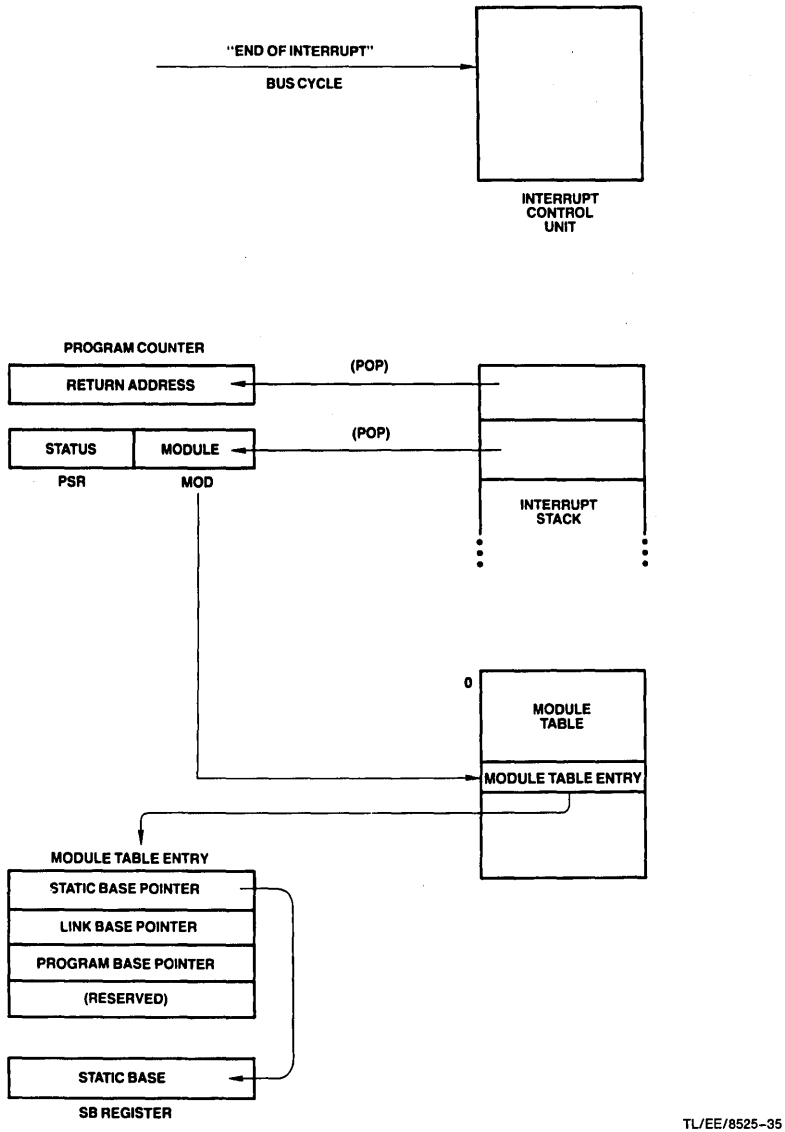
In the Non-Vectored mode, an interrupt request on the $\overline{\text{INT}}$ pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.



TL/EE/8525-34

FIGURE 3-24. Return from Trap (RETT n) Instruction Flow

3.0 Functional Description (Continued)



TL/EE/8525–35

FIGURE 3-25. Return from Interrupt (RET I) Instruction Flow

3.0 Functional Description (Continued)

3.8.3.2 Vectored Mode: Non-Cascaded Case

In the Vectored mode, the CPU uses an Interrupt Control Unit (ICU) to prioritize up to 16 interrupt requests. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an "Interrupt Acknowledge, Master" bus cycle (Section 3.4.2) reading a vector value from the low-order byte of the Data Bus. This vector is then used as an index into the Dispatch Table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) instruction, which performs an End of Interrupt bus cycle, informing the ICU that it may re-prioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it needs also to inform a Cascaded ICU (see below).

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range of 0 through 127; that is, they must be positive numbers in eight bits. By providing a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU (see below).

3.8.3.3 Vectored Mode: Cascaded Case

In order to allow up to 256 levels of interrupt, provision is made both in the CPU and in the NS32202 Interrupt Control Unit (ICU) to transparently support cascading. *Figure 3-27* shows a typical cascaded configuration. Note that the Interrupt output from a Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin.

In a system which uses cascading, two tasks must be performed upon initialization:

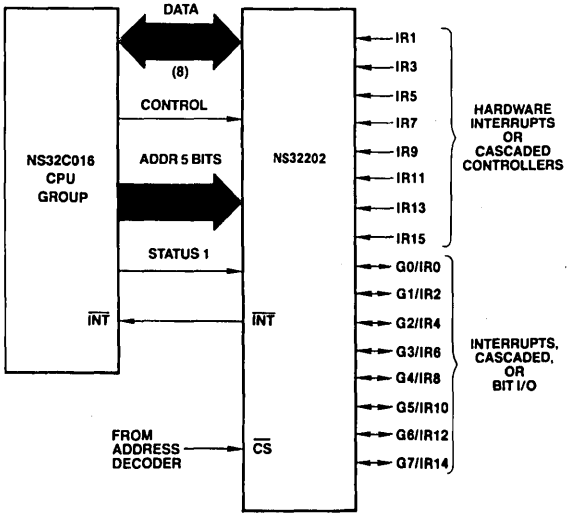
1) For each Cascaded ICU in the system, the Master ICU must be informed of the line number (0 to 15) on which it receives the cascaded requests.
2) A Cascade Table must be established in memory. The Cascade Table is located in a NEGATIVE direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

Figure 3-22 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0 to 15) and subtract 16 from it, giving an index in the range −16 to −1. Multiply this value by 4, and add the resulting negative number to the contents of the INTBASE Register. The 32-bit entry at this address must be set to the address of the Hardware Vector Register of the Cascaded ICU. This is referred to as the "Cascade Address."

Upon receipt of an interrupt request from a Cascaded ICU, the Master ICU interrupts the CPU and provides the negative Cascade Table index instead of a (positive) vector number. The CPU, seeing the negative value, uses it as an index into the Cascade Table and reads the Cascade Address from the referenced entry. Applying this address, the CPU performs an "Interrupt Acknowledge, Cascaded" bus cycle (Section 3.4.2), reading the final vector value. This vector is interpreted by the CPU as an unsigned byte, and can therefore be in the range of 0 through 255.

In returning from a Cascaded interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any Maskable Interrupt. The CPU performs an "End of Interrupt, Master" bus cycle (Section 3.4.2), whereupon the Master ICU again provides the negative Cascaded Table index. The CPU, seeing a negative value, uses it to find the corresponding Cascade Address from the Cascade Table. Applying this address, it performs an "End of Interrupt, Cascaded" bus cycle (Section 3.4.2), informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

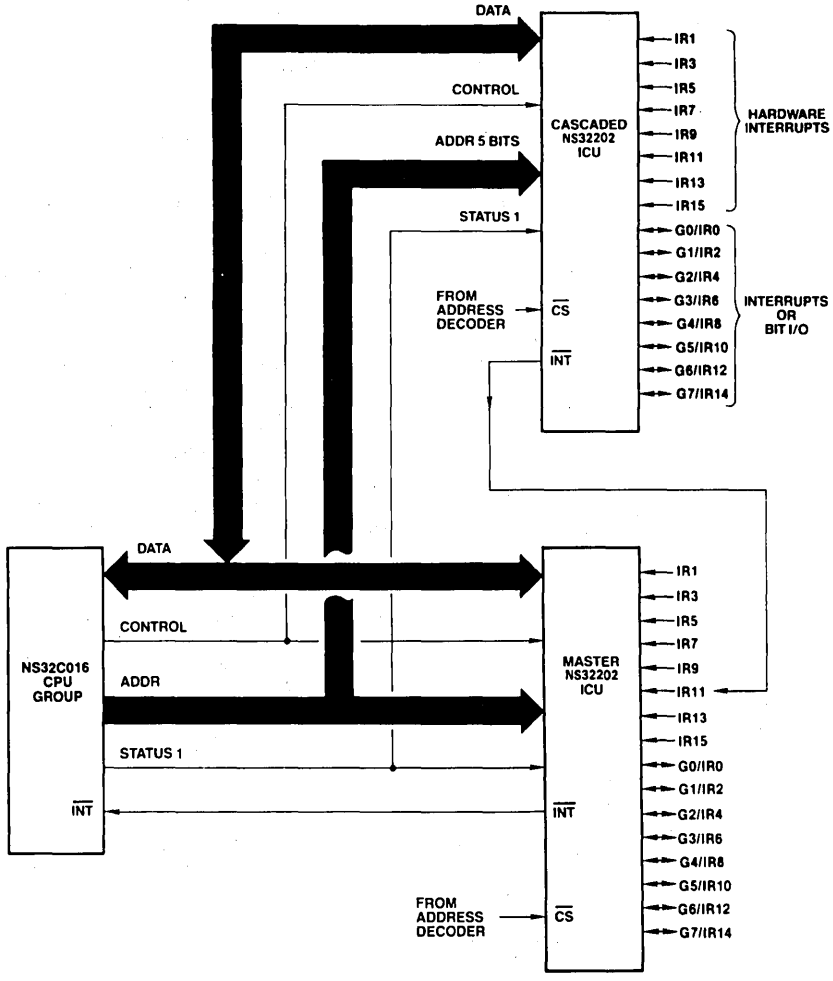
Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the Interrupt Mask Register of the Interrupt Controller. However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the $\overline{INT}$ line before the ICU deasserted it. This could cause the ICU to provide an invalid vector. To avoid this problem the above operation should be performed with the CPU interrupt disabled.



TL/EE/8525-36

FIGURE 3-26. Interrupt Control Unit Connections (16 Levels)

3.0 Functional Description (Continued)



TL/EE/8525-37

FIGURE 3-27. Cascaded Interrupt Control Unit Connections

3.8.4 Non-Maskable Interrupt (The $\overline{\text{NMI}}$ Pin)

The Non-Maskable Interrupt is triggered whenever a falling edge is detected on the $\overline{\text{NMI}}$ pin. The CPU performs an "Interrupt Acknowledge, Master" bus cycle (Section 3.4.2) when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is $FFFF00_{16}$. The vector value used for the Non-Maskable Interrupt is taken as 1, regardless of the value read from the bus.

The service procedure returns from the Non-Maskable Interrupt using the Return from Trap (RETT) instruction. No special bus cycles occur on return.

For the full sequence of events in processing the Non-Maskable Interrupt, see Section 3.8.7.1.

3.8.5 Traps

A trap is an internally-generated interrupt request caused as a direct and immediate result of the execution of an instruction. The Return Address pushed by any trap except Trap (TRC) below is the address of the first byte of the instruction during which the trap occurred. Traps do not disable interrupts, as they are not associated with external events. Traps recognized by NS32C016 CPU are:

Trap (SLAVE): An exceptional condition was detected by the Floating Point Unit or another Slave Processor during the execution of a Slave Instruction. This trap is requested via the Status Word returned as part of the Slave Processor Protocol (Section 3.9.1).

3.0 Functional Description (Continued)

Trap (ILL): Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR bit U=1).

Trap (SVC): The Supervisor Call (SVC) instruction was executed.

Trap (DVZ): An attempt was made to divide an integer by zero. (The Slave trap is used for Floating Point division by zero.)

Trap (FLG): The FLAG instruction detected a "1" in the CPU PSR F bit.

Trap (BPT): The Breakpoint (BPT) instruction was executed.

Trap (TRC): The instruction just completed is being traced. See below.

Trap (UND): An undefined opcode was encountered by the CPU.

A special case is the Trace Trap (TRC), which is enabled by setting the T bit in the Processor Status Register (PSR). At the beginning of each instruction, the T bit is copied into the PSR P (Trace "Pending") bit. If the P bit is set at the end of an instruction, then the Trace Trap is activated. If any other trap or interrupt request is made during a traced instruction, its entire service procedure is allowed to complete before the Trace Trap occurs. Each interrupt and trap sequence handles the P bit for proper tracing, guaranteeing one and only one Trace Trap per instruction, and guaranteeing that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

3.8.6 Prioritization

The NS32C016 CPU internally prioritizes simultaneous interrupt and trap requests as follows:

1) Traps other than Trace (Highest priority)
2) Abort
3) Non-Maskable Interrupt
4) Maskable Interrupts
5) Trace Trap (Lowest priority)

3.8.7 Interrupt/Trap Sequences: Detail Flow

For purposes of the following detailed discussion of interrupt and trap service sequences, a single sequence called "Service" is defined in *Figure 3-28.* Upon detecting any interrupt request or trap condition, the CPU first performs a sequence dependent upon the type of interrupt or trap. This sequence will include pushing the Processor Status Register and establishing a Vector and a Return Address. The CPU then performs the Service sequence.

For the sequenced followed in processing either Maskable or Non-Maskable Interrupts (on the $\overline{INT}$ or $\overline{NMI}$ pins, respectively), see Section 3.8.7.1. For Abort interrupts, see Section 3.8.7.4. For the Trace Trap, see Section 3.8.7.3, and for all other traps see Section 3.8.7.2.

3.8.7.1 Maskable/Non-Maskable Interrupt Sequence

This sequence is performed by the CPU when the $\overline{NMI}$ pin receives a falling edge, or the $\overline{INT}$ pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or, in the case of the String instructions, at the next interruptible point during its execution.

1. If a String instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P bit.
   b. Set "Return Address" to the address of the first byte of the interrupted instruction.

   Otherwise, set "Return Address" to the address of the next instruction.
2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, T, P and I.
3. If the interrupt is Non-Maskable:
   a. Read a byte from address $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Section 3.4.2). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.
4. If the interrupt is Non-Vectored:
   a. Read a byte from address $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Section 3.4.2). Discard the byte read.
   b. Set "Vector" to 0.
   c. Go to Step 8.
5. Here the interrupt is Vectored. Read "Byte" from address $FFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Section 3.4.2).
6. If "Byte" ≥ 0, then set "Vector" to "Byte" and go to Step 8.
7. If "Byte" is in the range -16 through -1, then the interrupt source is Cascaded. (More negative values are reserved for future use.) Perform the following:
   a. Read the 32-bit Cascade Address from memory. The address is calculated as INTBASE + 4* Byte.
   b. Read "Vector," applying the Cascade Address just read and Status Code 0101 (Interrupt Acknowledge, Cascaded: Section 3.4.2).
8. Push the PSR copy (from Step 2) onto the Interrupt Stack as a 16-bit value.
9. Perform Service (Vector, Return Address), *Figure 3-28.*

Service (Vector, Return Address):

1) Read the 32-bit External Procedure Descriptor from the Interrupt Dispatch Table: address is Vector*4+INTBASE Register contents.
2) Move the Module field of the Descriptor into the MOD Register.
3) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.
4) Read the Program Base pointer from memory address MOD+8, and add to it the Offset field from the Descriptor, placing the result in the Program Counter.
5) Flush Queue: Non-sequentially fetch first instruction of Interrupt Routine.
6) Push MOD Register onto the Interrupt Stack as a 16-bit value. (The PSR has already been pushed as a 16-bit value.)
7) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.

FIGURE 3-28. Service Sequence
Invoked during all interrupt/trap sequences

3.0 Functional Description (Continued)

3.8.7.2 Trap Sequence: Traps Other Than Trace

1) Restore the currently selected Stack Pointer and the Processor Status Register to their original values at the start of the trapped instruction.
2) Set "Vector" to the value corresponding to the trap type.

   SLAVE: Vector=3.
   ILL: Vector=4.
   SVC: Vector=5.
   DVZ: Vector=6.
   FLG: Vector=7.
   BPT: Vector=8.
   UND: Vector=10.
3) Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, P and T.
4) Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5) Set "Return Address" to the address of the first byte of the trapped instruction.
6) Perform Service (Vector, Return Address), *Figure 3-28.*

3.8.7.3 Trace Trap Sequence

1) In the Processor Status Register (PSR), clear the P bit.
2) Copy the PSR into a temporary register, then clear PSR bits S, U and T.
3) Push the PSR copy onto the Interrupt Stack as a 16-bit value.
4) Set "Vector" to 9.
5) Set "Return Address" to the address of the next instruction.
6) Perform Service (Vector, Return Address), *Figure 3-28.*

3.8.7.4 Abort Sequence

1) Restore the currently selected Stack Pointer to its original contents at the beginning of the aborted instruction.
2) Clear the PSR P bit.
3) Copy the PSR into a temporary register, then clear PSR bits S, U, T and I.
4) Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5) Set "Vector" to 2.
6) Set "Return Address" to the address of the first byte of the aborted instruction.
7) Perform Service (Vector, Return Address), *Figure 3-28.*

3.9 SLAVE PROCESSOR INSTRUCTIONS

The NS32C016 CPU recognizes three groups of instructions as being executable by external Slave Processors:

Floating Point Instruction Set
Memory Management Instruction Set
Custom Instruction Set

Each Slave Instruction Set is validated by a bit in the Configuration Register (Section 2.1.3). Any Slave Instruction which does not have its corresponding Configuration Register bit set will trap as undefined, without any Slave Processor communication attempted by the CPU. This allows software simulation of a non-existent Slave Processor.

3.9.1 Slave Processor Protocol

Slave Processor instructions have a three-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word. The ID Byte has three functions:

1) It identifies the instruction as being a Slave Processor instruction.
2) It specifies which Slave Processor will execute it.
3) It determines the format of the following Operation Word of the instruction.

Upon receiving a Slave Processor instruction, the CPU initiates the sequence outlined in *Figure 3-29.* While applying Status Code 1111 (Broadcast ID, Section 3.4.2), the CPU transfers the ID Byte on the least-significant half of the Data Bus (AD0–AD7). All Slave Processors input this byte and decode it. The Slave Processor selected by the ID Byte is activated, and from this point the CPU is communicating only with it. If any other slave protocol was in progress (e.g., an aborted Slave instruction), this transfer cancels it.

The CPU next sends the Operation Word while applying Status Code 1101 (Transfer Slave Operand, Section 3.4.2). Upon receiving it, the Slave Processor decodes it, and at this point both the CPU and the Slave Processor are aware of the number of operands to be transferred and their sizes. The Operation Word is swapped on the Data Bus; that is, bits 0–7 appear on pins AD8–AD15 and bits 8–15 appear on pins AD0–AD7.

Using the Addressing Mode fields within the Operation Word, the CPU starts fetching operands and issuing them to the Slave Processor. To do so, it references any Addressing Mode extensions which may be appended to the Slave Processor instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the Slave Processor. The Status Code applied is 1101 (Transfer Slave Processor Operand, Section 3.4.2).

Status Combinations:
Send ID (ID): Code 1111
Xfer Operand (OP): Code 1101
Read Status (ST): Code 1110

Step	Status	Action
1	ID	CPU Send ID Byte.
2	OP	CPU Sends Operation Word.
3	OP	CPU Sends Required Operands.
4	—	Slave Starts Execution. CPU Pre-Fetches.
5	—	Slave Pulses $\overline{SPC}$ Low.
6	ST	CPU Reads Status Word. (Trap? Alter Flags?)
7	OP	CPU Reads Results (If Any).

FIGURE 3-29. Slave Processor Protocol

3.0 Functional Description (Continued)

After the CPU has issued the last operand, the Slave Processor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low. To allow for this, and for the Address Translation strap function, $\overline{AT}/\overline{SPC}$ is normally held high only by an internal pull-up device of approximately 5 kΩ.

While the Slave Processor is executing the instruction, the CPU is free to prefetch instructions into its queue. If it fills the queue before the Slave Processor finishes, the CPU will wait, applying Status Code 0011 (Waiting for Slave, Section 3.4.2).

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a Status Word from the Slave Processor, applying Status Code 1110 (Read Slave Status, Section 3.4.2). This word has the format shown in *Figure 3-30*. If the Q bit ("Quit", Bit 0) is set, this indicates that an error was detected by the Slave Processor. The CPU will not continue the protocol, but will immediately trap through the Slave vector in the Interrupt Table. Certain Slave Processor instructions cause CPU PSR bits to be loaded from the Status Word.

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The Read cycles from the Slave Processor are performed by the CPU while applying Status Code 1101 (Transfer Slave Operand, Section 3.4.2).

An exception to the protocol above is the LMR (Load Memory Management Register) instruction, and a corresponding Custom Slave instruction (LCR: Load Custom Register). In executing these instructions, the protocol ends after the CPU has issued the last operand. The CPU does not wait for an acknowledgement from the Slave Processor, and it does not read status.

3.9.2 Floating Point Instructions

Table 3-4 gives the protocols followed for each Floating Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Appendix A.

The Operand class columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see Series 32000 Instruction Set Reference Manual).

The Operand Issued columns show the sizes of the operands issued to the Floating Point Unit by the CPU. "D" indicates a 32-bit Double Word. "i" indicates that the instruction specifies an integer size for the operand (B=Byte, W=Word, D=Double Word). "f" indicates that the instruction specifies a Floating Point size for the operand (F=32-bit Standard Floating, L=64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the Slave Processor Status Word (*Figure 3-30*).

TABLE 3-4. Floating Point Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none

Notes:

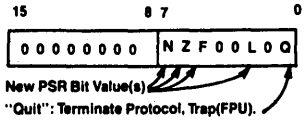
D = Double Word

i = integer size (B,W,D) specified in mnemonic.

f = Floating Point type (F,L) specified in mnemonic.

N/A = Not Applicable to this instruction.

3.0 Functional Description (Continued)



TL/EE/8525-38

FIGURE 3-30. Slave Processor Status Word Format

Any operand indicated as being of type "f" will not cause a transfer if the Register addressing mode is specified. This is because the Floating Point Registers are physically on the Floating Point Unit and are therefore available without CPU assistance.

3.9.3 Memory Management Instructions

Table 3-5 gives the protocols for Memory Management instructions. Encodings for these instructions may be found in Appendix A.

In executing the RDVAL and WRVAL instructions, the CPU calculates and issues the 32-bit Effective Address of the single operand. The CPU then performs a single-byte Read cycle from that address, allowing the MMU to safely abort the instruction if the necessary information is not currently in physical memory. Upon seeing the memory cycle complete, the MMU continues the protocol, and returns the validation result in the F bit of the Slave Status Word.

The size of a Memory Management operand is always a 32-bit Double Word. For further details of the Memory Management Instruction set, see the Series 32000 Instruction Set Reference Manual and the NS32082 MMU Data Sheet.

TABLE 3-5. Memory Management Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
RDVAL*	addr	N/A	D	N/A	N/A	F
WRVAL*	addr	N/A	D	N/A	N/A	F
LMR*	read.D	N/A	D	N/A	N/A	none
SMR*	write.D	N/A	N/A	N/A	D to Op. 1	none

Note:

In the RDVAL and WRVAL instructions, the CPU issues the address as a Double Word, and performs a single-byte Read cycle from that memory address. For details, see the Series 32000 Instruction Set Reference Manual and the NS32082 Memory Management Unit Data Sheet.

D = Double Word

* = Privileged Instruction: will trap if CPU is in User Mode.

N/A = Not Applicable to this Instruction.

3.0 Functional Description (Continued)

3.9.4 Custom Slave Instructions

Provided in the NS32C016 is the capability of communicating with a user-defined, "Custom" Slave Processor. The instruction set provided for a Custom Slave Processor defines the instruction formats, the operand classes and the communication protocol. Left to the user are the interpretations of the Op Code fields, the programming model of the Custom Slave and the actual types of data transferred. The protocol specifies only the size of an operand, not its data type.

Table 3-6 lists the relevant information for the Custom Slave instruction set. The designation "c" is used to represent an operand which can be a 32-bit ("D") or 64-bit ("Q") quantity in any format; the size is determined by the suffix on the mnemonic. Similarly, an "i" indicates an integer size (Byte, Word, Double Word) selected by the corresponding mnemonic suffix.

Any operand indicated as being of type 'c' will not cause a transfer if the register addressing mode is specified. It is assumed in this case that the slave processor is already holding the operand internally.

For the instruction encodings, see Appendix A.

TABLE 3-6. Custom Slave Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
CCAL0c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL1c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL2c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL3c	read.c	rmw.c	c	c	c to Op. 2	none
CMOV0c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV1c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV2c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV3c	read.c	write.c	c	N/A	c to Op. 2	none
CCMP0c	read.c	read.c	c	c	N/A	N,Z,L
CCMP1c	read.c	read.c	c	c	N/A	N,Z,L
CCV0ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV1ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV2ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV3ic	readi	write.c	i	N/A	c to Op. 2	none
CCV4DQ	read.D	write.Q	D	N/A	Q to Op. 2	none
CCV5QD	read.Q	write.D	Q	N/A	D to Op. 2	none
LCSR	read.D	N/A	D	N/A	N/A	none
SCSR	N/A	write.D	N/A	N/A	D to Op. 2	none
CATST0*	addr	N/A	D	N/A	N/A	F
CATST1*	addr	N/A	D	N/A	N/A	F
LCR*	read.D	N/A	D	N/A	N/A	none
SCR*	write.D	N/A	N/A	N/A	D to Op.1	none

Notes:

D = Double Word

i = integer size (B,W,D) specified in mnemonic.

c = Custom size (D:32 bits or Q:64 bits) specified in mnemonic.

* = Privileged instruction: will trap if CPU is in User Mode.

N/A = Not Applicable to this instruction.

4.0 Device Specifications

4.1 NS32C016 PIN DESCRIPTIONS

The following is a brief description of all NS32C016 pins. The descriptions reference portions of the Functional Description, Section 3.

4.1.1 Supplies

Logic Power (V_{CCL}): +5V positive supply for on-chip logic. Section 3.1.

Buffer Power (V_{CCB}): +5V positive supply for on-chip output buffers. Section 3.1.

Logic Ground (GNDL): Ground reference for on-chip logic. Section 3.1.

Buffer Ground (GNDB): Ground reference for on-chip drivers connected to output pins. Section 3.1.

4.1.2 Input Signals

Clocks (PHI1, PHI2): Two-phase clocking signals. Section 3.2.

Ready (RDY): Active high. While RDY is inactive, the CPU extends the current bus cycle to provide for a slower memory or peripheral reference. Upon detecting RDY active, the CPU terminates the bus cycle. Section 3.4.1.

Hold Request ($\overline{HOLD}$): Active low. Causes the CPU to release the bus for DMA or multiprocessing purposes. Section 3.6.

Note: If the $\overline{HOLD}$ signal is generated asynchronously, its set up and hold times may be violated. In this case it is recommended to synchronize it with CTTL to minimize the possibility of metastable states.

The CPU provides only one synchronization stage to minimize the $\overline{HLDA}$ latency. This is to avoid speed degradations in cases of heavy $\overline{HOLD}$ activity (i.e. DMA controller cycles interleaved with CPU cycles).

Interrupt ($\overline{INT}$): Active low. Maskable Interrupt request. Section 3.8.

Non-Maskable Interrupt ($\overline{NMI}$): Active low. Non-Maskable Interrupt request. Section 3.8.

Reset/Abort ($\overline{RST}/\overline{ABT}$): Active low. If held active for one clock cycle and released, this pin causes an Abort Command, Section 3.5.4. If held longer, it initiates a Reset, Section 3.3.

4.1.3 Output Signals

Address Bits 16–23 (A16–A23): These are the most significant 8 bits of the memory address bus. Section 3.4.

Address Strobe ($\overline{ADS}$): Active low. Controls address latches; indicates start of a bus cycle. Section 3.4.

Data Direction In ($\overline{DDIN}$): Active low. Status signal indicating direction of data transfer during a bus cycle. Section 3.4.

High Byte Enable ($\overline{HBE}$): Active low. Status signal enabling transfer on the most significant byte of the Data Bus. Section 3.4; Section 3.4.3.

Note: In the current NS32C016, the $\overline{HBE}$ signal is forced low by the CPU when $\overline{FLT}$ is asserted by the MMU. However, in future revisions of the CPU, $\overline{HBE}$ will no longer be affected by $\overline{FLT}$. Therefore, in a memory managed system, an external 'AND' gate is required. This is shown in *Figure B-1* in Appendix B.

Status (ST0–ST3): Active high. Bus cycle status code, ST0 least significant. Section 3.4.2. Encodings are:

0000—Idle: CPU Inactive on Bus.
0001—Idle: WAIT Instruction.
0010—(Reserved)
0011—Idle: Waiting for Slave.
0100—Interrupt Acknowledge, Master.
0101—Interrupt Acknowledge, Cascaded.
0110—End of Interrupt, Master.
0111—End of Interrupt, Cascaded.
1000—Sequential Instruction Fetch.
1001—Non-Sequential Instruction Fetch.
1010—Data Transfer.
1011—Read Read-Modify-Write Operand.
1100—Read for Effective Address.
1101—Transfer Slave Operand.
1110—Read Slave Status Word.
1111—Broadcast Slave ID.

Hold Acknowledge ($\overline{HLDA}$): Active low. Applied by the CPU in response to $\overline{HOLD}$ input, indicating that the bus has been released for DMA or multiprocessing purposes. Section 3.6.

User/Supervisor (U/$\overline{S}$): User or Supervisor Mode status. Section 3.7. High state indicates User Mode, low indicates Supervisor Mode. Section 3.7.

Interlocked Operation ($\overline{ILO}$): Active low. Indicates that an interlocked instruction is being executed. Section 3.7.

Program Flow Status ($\overline{PFS}$): Active Low. Pulse indicates beginning of an instruction execution. Section 3.7.

4.1.4 Input-Output Signals

Address/Data 0–15 (AD0–AD15): Multiplexed Address/Data information. Bit 0 is the least significant bit of each. Section 3.4.

Address Translation/Slave Processor Control ($\overline{AT}/\overline{SPC}$): Active low. Used by the CPU as the data strobe output for Slave Processor transfers; used by Slave Processors to acknowledge completion of a slave instruction. Section 3.4.6; Section 3.9. Sampled on the rising edge of Reset as Address Translation Strap. Section 3.5.1.

In non-memory-managed systems this pin should be pulled up to V_{CC} through a 10 kΩ resistor.

Data Strobe/Float ($\overline{DS}/\overline{FLT}$): Active low. Data Strobe output, Section 3.4, or Float Command input, Section 3.5.3. Pin function is selected on $\overline{AT}/\overline{SPC}$ pin, Section 3.5.1.

4.0 Device Specifications (Continued)

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	1.5 Watt

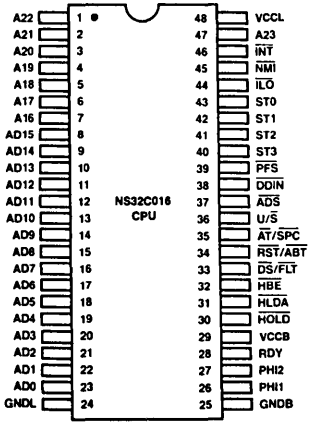
Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS: $T_A = -40°C$ to $+85°C$, $V_{CC} = 5V \pm 10\%$, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		$V_{CC} + 0.5$	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{CH}	High Level Clock Voltage	PHI1, PHI2 pins only	0.90 V_{CC}		$V_{CC} + 0.5$	V
V_{CL}	Low Level Clock Voltage	PHI1, PHI2 pins only	−0.5		0.10 V_{CC}	V
V_{CRT}	Clock Input Ringing Tolerance	PHI1, PHI2 pins only	−0.5		0.6	V
V_{OH}	High Level Output Voltage	$I_{OH} = -400\ \mu A$	0.90 V_{CC}			V
V_{OL}	Low Level Output Voltage	$I_{OL} = 2$ mA			0.10 V_{CC}	V
I_{ILS}	$\overline{AT}/\overline{SPC}$ Input Current (low)	$V_{IN} = 0.4V$, $\overline{AT}/\overline{SPC}$ in input mode	0.05		1.0	mA
I_I	Input Load Current	$0 \le V_{IN} \le V_{CC}$, All inputs except PHI1, PHI2, $\overline{AT}/\overline{SPC}$	−20		20	μA
I_L	Leakage Current Output and IO Pins in TRI-STATE/ Input Mode	$0.4 \le V_{IN} \le V_{CC}$	−20		20	μA
I_{CC}	Active Supply Current	$I_{OUT} = 0$, $T_A = 25°C$		70	100	mA

Connection Diagram

Dual-In-Line Package



TL/EE/8525-2

Top View

FIGURE 4-1

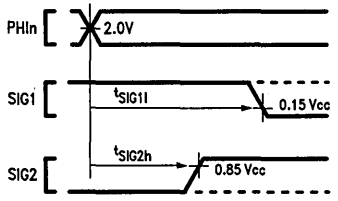
Order Number NS32C016D-10, NS32C016D-15, NS32C016N-10 or NS32C016N-15
See NS Package Number D48A or N48A

4.0 Device Specifications (Continued)

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 2.0V on the rising or falling edges of the clock phases PHI1 and PHI2; to 15% or 85% of V_{CC} on all the CMOS output signals, and to 0.8V or 2.0V on all the TTL input signals as illustrated in *Figures 4-2* and *4-3* unless specifically stated otherwise.



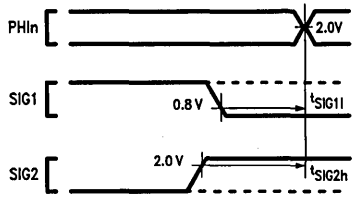
TL/EE/8525-39

FIGURE 4-2. Timing Specification Standard (CMOS Output Signals)

ABBREVIATIONS:

L.E. — leading edge R.E. — rising edge

T.E. — trailing edge F.E. — falling edge



TL/EE/8525-40

FIGURE 4-3. Timing Specification Standard (TTL Input Signals)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32C016-10 and NS32C016-15

Maximum times assume capacitive loading of 75 pF, on the address/data bus signals and 50 pF on all other signals.

Name	Figure	Description	Reference/Conditions	NS32C016-10		NS32C016-15		Units
				Min	Max	Min	Max	
t_{ALv}	4-4	Address bits 0–15 valid	after R.E., PHI1 T1		40		35	ns
t_{ALh}	4-4	Address bits 0–15 hold	after R.E., PHI1 Tmmu or T2	5		5		ns
t_{Dv}	4-4	Data valid (write cycle)	after R.E., PHI1 T2		50		35	ns
t_{Dh}	4-4	Data hold (write cycle)	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{AHv}	4-4	Address bits 16–23 valid	after R.E., PHI1 T1		40		35	ns
t_{AHh}	4-4	Address bits 16–23 hold	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{ALADSs}	4-5	Address bits 0–15 set up	before $\overline{ADS}$ T.E.	25		20		ns
t_{AHADSs}	4-5	Address bits 16–23 set up	before $\overline{ADS}$ T.E.	25		20		ns
t_{ALADSh}	4-9	Address bits 0–15 hold	after $\overline{ADS}$ T.E.	15		10		ns
t_{AHADSh}	4-9	Address bits 16–23 hold	after $\overline{ADS}$ T.E.	15		10		ns
t_{ALf}	4-5	Address bits 0–15 floating (no MMU)	after R.E., PHI1 T2		25		20	ns
t_{ALMf}	4-9	Address bits 0–15 floating (with MMU)	after R.E., PHI1 TMMU		25		20	ns
t_{AHMf}	4-9	Address bits 16–23 floating (with MMU)	after R.E., PHI1 TMMU		25		20	ns
t_{HBEv}	4-4	$\overline{HBE}$ signal valid	after R.E., PHI1 T1		45		35	ns
t_{HBEh}	4-4	$\overline{HBE}$ signal hold	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{STv}	4-4	Status (ST0–ST3) valid	after R.E., PHI1 T4 (before T1, see note)		45		35	ns
t_{STh}	4-4	Status (ST0–ST3) hold	after R.E., PHI1 T4 (after T1)	0		0		ns
t_{DDINv}	4-5	$\overline{DDIN}$ signal valid	after R.E., PHI1 T1		50		35	ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32C016-10 and NS32C016-15 (Continued)

Name	Figure	Description	Reference/Conditions	NS32C016-10		NS32C016-15		Units
				Min	Max	Min	Max	
t_{DDINh}	4-5	$\overline{DDIN}$ signal hold	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{ADSa}	4-4	$\overline{ADS}$ signal active (low)	after R.E., PHI1 T1		35		26	ns
t_{ADSia}	4-4	$\overline{ADS}$ signal inactive	after R.E., PHI2 T1		40		30	ns
t_{ADSw}	4-4	$\overline{ADS}$ pulse width	at 15% V_{CC} (both edges)	30		25		ns
t_{DSa}	4-4	$\overline{DS}$ signal active (low)	after R.E., PHI1 T2		40		30	ns
t_{DSia}	4-4	$\overline{DS}$ signal inactive	after R.E., PHI1 T4		40		30	ns
t_{ALf}	4-6	AD0–AD15 floating	after R.E., PHI1 T1 (caused by $\overline{HOLD}$)		25		20	ns
t_{AHf}	4-6	A16–A23 floating	after R.E., PHI1 T1 (caused by $\overline{HOLD}$)		25		20	ns
t_{DSf}	4-6	$\overline{DS}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{ADSf}	4-6	$\overline{ADS}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{HBEf}	4-6	$\overline{HBE}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{DDINf}	4-6	$\overline{DDIN}$ floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		50		40	ns
t_{HLDAa}	4-6	$\overline{HLDA}$ signal active (low)	after R.E., PHI1 Ti		30		25	ns
t_{HLDAia}	4-8	$\overline{HLDA}$ signal inactive	after R.E., PHI1 Ti		40		30	ns
t_{DSr}	4-8	$\overline{DS}$ signal returns from floating (caused by HOLD)	after R.E., PHI1 Ti		55		40	ns
t_{ADSr}	4-8	$\overline{ADS}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{HBEr}	4-8	$\overline{HBE}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{DDINr}	4-8	$\overline{DDIN}$ signal returns from floating (caused by $\overline{HOLD}$)	after R.E., PHI1 Ti		55		40	ns
t_{DDINf}	4-9	$\overline{DDIN}$ signal floating (caused by $\overline{FLT}$)	after $\overline{FLT}$ F.E.		55		50	ns
t_{HBEl}	4-9	$\overline{HBE}$ signal low (caused by $\overline{FLT}$)	after $\overline{FLT}$ F.E.		40		30	ns
t_{DDINr}	4-10	$\overline{DDIN}$ signal returns from floating (caused by $\overline{FLT}$)	after $\overline{FLT}$ R.E.		40		30	ns
t_{HBEr}	4-10	$\overline{HBE}$ signal returns from low (caused by $\overline{FLT}$)	after $\overline{FLT}$ R.E.		35		25	ns
t_{SPCa}	4-13	$\overline{SPC}$ output active (low)	after R.E., PHI1 T1		35		26	ns
t_{SPCia}	4-13	$\overline{SPC}$ output inactive	after R.E., PHI1 T4		35		26	ns
t_{SPCnf}	4-15	$\overline{SPC}$ output nonforcing	after R.E., PHI2 T4		30		25	ns
t_{Dv}	4-13	Data valid (slave processor write)	after R.E., PHI1 T1		50		35	ns
t_{Dh}	4-13	Data hold (slave processor write)	after R.E., PHI1 next T1 or Ti	0		0		ns
t_{PFSw}	4-18	$\overline{PFS}$ pulse width	at 15% V_{CC} (both edges)	50		40		ns
t_{PFSa}	4-18	$\overline{PFS}$ pulse active (low)	after R.E., PHI2		40		35	ns
t_{PFSia}	4-18	$\overline{PFS}$ pulse inactive	after R.E., PHI2		40		35	ns
t_{ILOs}	4-20a	$\overline{ILO}$ signal setup	before R.E., PHI1 T1 of first interlocked write cycle	50		35		ns
t_{ILOh}	4-20b	$\overline{ILO}$ signal hold	after R.E., PHI1 T3 of last interlocked read cycle	10		7		ns
t_{ILOa}	4-21	$\overline{ILO}$ signal active (low)	after R.E., PHI1		35		30	ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32C016-10 and NS32C016-15 (Continued)

Name	Figure	Description	Reference/Conditions	NS32C016-10		NS32C016-15		Units
				Min	Max	Min	Max	
t_{ILOia}	4-21	$\overline{ILO}$ signal inactive	after R.E., PHI1		35		30	ns
t_{USV}	4-22	U/$\overline{S}$ signal valid	after R.E., PHI1 T4		35		30	ns
t_{USh}	4-22	U/$\overline{S}$ signal hold	after R.E., PHI1 T4	8		6		ns
t_{NSPF}	4-19b	Nonsequential fetch to next $\overline{PFS}$ clock cycle	after R.E., PHI1 T1	4		4		t_{Cp}
t_{PFNS}	4-19a	$\overline{PFS}$ clock cycle to next nonsequential fetch	before R.E., PHI1 T1	4		4		t_{Cp}
t_{LXPF}	4-29	Last operand transfer of an instruction to next $\overline{PFS}$ clock cycle	before R.E., PHI1 T1 of first bus cycle of transfer	0		0		t_{Cp}

Note: Every memory cycle starts with T4, during which Cycle Status is applied. If the CPU was idling, the sequence will be: ". . . Ti, T4, T1 . . .". If the CPU was not idling, the sequence will be: ". . . T4, T1 . . .".

4.4.2.2 Input Signal Requirements: NS32C016-10 and NS32C016-15

Name	Figure	Description	Reference/Conditions	NS32C016-10		NS32C016-15		Units
				Min	Max	Min	Max	
t_{PWR}	4-25	Power stable to $\overline{RST}$ R.E.	after V_{CC} reaches 4.5V	50		33		μs
t_{DIs}	4-5	Data in setup (read cycle)	before F.E., PHI2 T3	15		10		ns
t_{DIh}	4-5	Data in hold (read cycle)	after F.E., PHI1 T4	3		3		ns
t_{HLDa}	4-6	$\overline{HOLD}$ active (low) setup time (see note)	before F.E., PHI2 TX1	25		17		ns
t_{HLDia}	4-8	$\overline{HOLD}$ inactive setup time	before F.E., PHI2 Ti	25		17		ns
t_{HLDh}	4-6	$\overline{HOLD}$ hold time	after R.E., PHI1 TX2	0		0		ns
t_{FLTa}	4-9	$\overline{FLT}$ active (low) setup time	before F.E., PHI2 Tmmu	25		17		ns
t_{FLTia}	4-10	$\overline{FLT}$ inactive setup time	before F.E., PHI2 T2	25		17		ns
t_{RDYs}	4-11, 4-12	RDY setup time	before F.E., PHI2 T2 or T3	15		10		ns
t_{RDYh}	4-11, 4-12	RDY hold time	after F.E., PHI1 T3	5		5		ns
t_{ABTs}	4-23	$\overline{ABT}$ setup time ($\overline{FLT}$ inactive)	before F.E., PHI2 Tmmu	20		13		ns
t_{ABTs}	4-24	$\overline{ABT}$ setup time ($\overline{FLT}$ active)	before F.E., PHI2 Tf	20		13		ns
t_{ABTh}	4-23	$\overline{ABT}$ hold time	after R.E., PHI1	0		0		ns
t_{RSTs}	4-25, 4-26	$\overline{RST}$ setup time	before F.E., PHI1	10		8		ns
t_{RSTw}	4-26	$\overline{RST}$ pulse width	at 0.8V (both edges)	64		64		t_{Cp}
t_{INTs}	4-27	$\overline{INT}$ setup time	before R.E., PHI1	20		15		ns
t_{NMIw}	4-28	$\overline{NMI}$ pulse width	at 0.8V (both edges)	70		70		ns
t_{DIs}	4-14	Data setup (slave read cycle)	before F.E., PHI2 T1	15		10		ns
t_{DIh}	4-14	Data hold (slave read cycle)	after R.E., PHI1 T4	3		3		ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32C016-10 and NS32C016-15 (Continued)

Name	Figure	Description	Reference/Conditions	NS32C016-10		NS32C016-15		Units
				Min	Max	Min	Max	
t_{SPCd}	4-15	$\overline{SPC}$ pulse delay from slave	after R.E., PHI2 T4	30		25		ns
t_{SPCs}	4-15	$\overline{SPC}$ setup time	before F.E., PHI1	30		25		ns
t_{SPCw}	4-15	$\overline{SPC}$ pulse width from slave processor (async. input)	at 0.8V (both edges)	20		20		ns
t_{ATs}	4-16	$\overline{AT}/\overline{SPC}$ setup for address translation strap	before R.E., PHI1 of cycle during which $\overline{RST}$ pulse is removed	1		1		t_{Cp}
t_{ATh}	4-16	$\overline{AT}/\overline{SPC}$ hold for address translation strap	after F.E., PHI1 of cycle during which $\overline{RST}$ pulse is removed	2		2		t_{Cp}

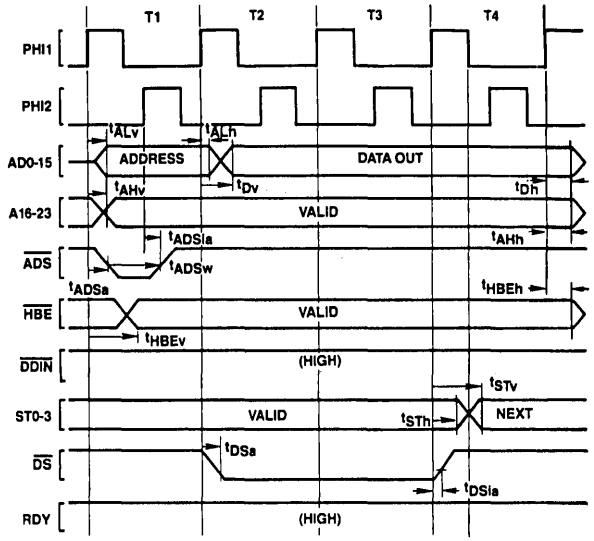
Note: This setup time is necessary to ensure prompt acknowledgement via $\overline{HLDA}$ and the ensuing floating of CPU off the buses. Note that the time from the receipt of the $\overline{HOLD}$ signal until the CPU floats is a function of the time $\overline{HOLD}$ signal goes low, the state of the RDY input (in MMU systems), and the length of the current MMU cycle.

4.4.2.3 Clocking Requirements: NS32C016-10 and NS32C016-15

Name	Figure	Description	Reference/Conditions	NS32C016-10		NS32C016-15		Units
				Min	Max	Min	Max	
t_{Cp}	4-17	Clock period	R.E., PHI1, PHI2 to next R.E., PHI1, PHI2	100	250	66	250	ns
t_{CLw}	4-17	PHI1, PHI2 pulse width	At 2.0V on PHI1, PHI2 (both edges)	$0.5t_{Cp}$ −10 ns		$0.5t_{Cp}$ −6 ns		
t_{CLh}	4-17	PHI1, PHI2 High Time	At 90% V_{CC} on PHI1, PHI2	$0.5t_{Cp}$ −15 ns		$0.5t_{Cp}$ −10 ns		
t_{CLl}	4-17	PHI1, PHI2 Low Time	At 15% V_{CC} on PHI1, PHI2	$0.5t_{Cp}$ −5 ns		$0.5t_{Cp}$ −5 ns		
$t_{nOVL(1,2)}$	4-17	Non-overlap time	At 15% V_{CC} on PHI1, PHI2	−2	2	−2	2	ns
t_{nOVLas}		Non-overlap asymmetry ($t_{nOVL(1)} - t_{nOVL(2)}$)	At 15% V_{CC} on PHI1, PHI2	−3	3	−3	3	ns
t_{CLwas}		PHI1, PHI2 asymmetry ($t_{CLw(1)} - t_{CLw(2)}$)	At 2.0V on PHI1, PHI2	−5	5	−3	3	ns

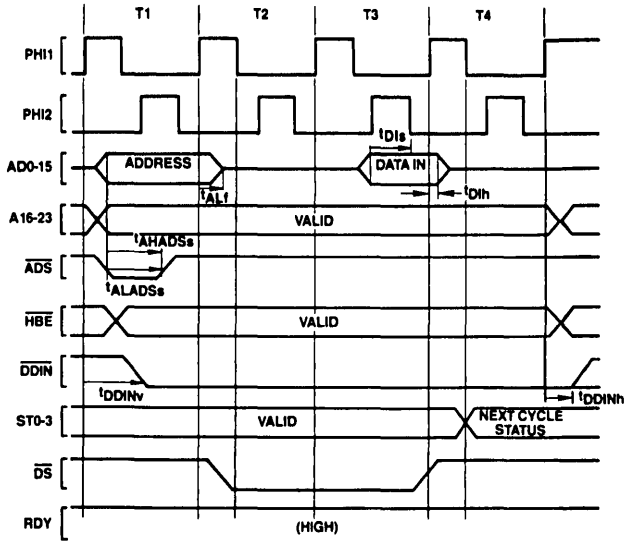
4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams



TL/EE/8525-41

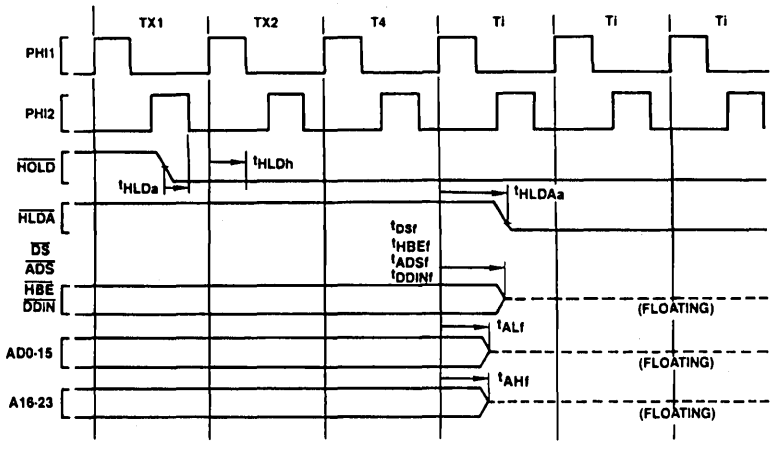
FIGURE 4-4. Write Cycle



TL/EE/8525-42

FIGURE 4-5. Read Cycle

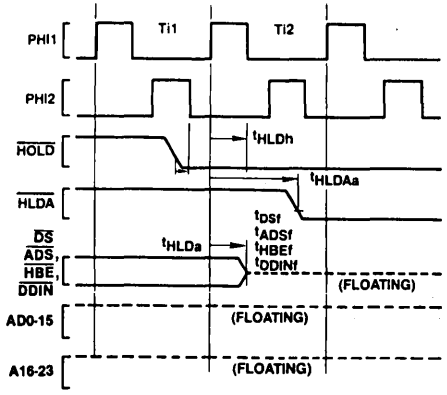
4.0 Device Specifications (Continued)



TL/EE/8525-43

FIGURE 4-6. Floating by $\overline{\text{HOLD}}$ Timing (CPU Not Idle Initially)

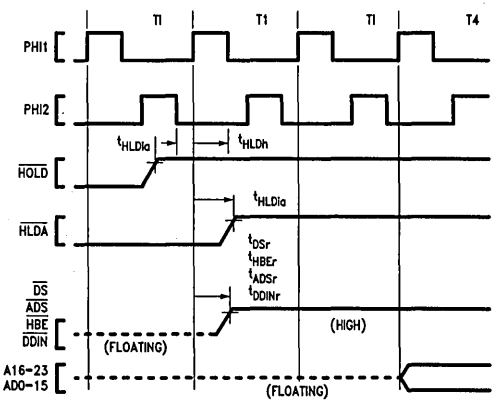
Note that whenever the CPU is not idling (not in TI), the $\overline{\text{HOLD}}$ request ($\overline{\text{HOLD}}$ low) must be active t_{HLDa} before the falling edge of PHI2 of the clock cycle that appears two clock cycles before T4 (TX1) and stay low until t_{HLDh} after the rising edge of PHI1 of the clock cycle that precedes T4 (TX2) for the request to be acknowledged.



TL/EE/8525-44

FIGURE 4-7. Floating by $\overline{\text{HOLD}}$ Timing (CPU Initially Idle)

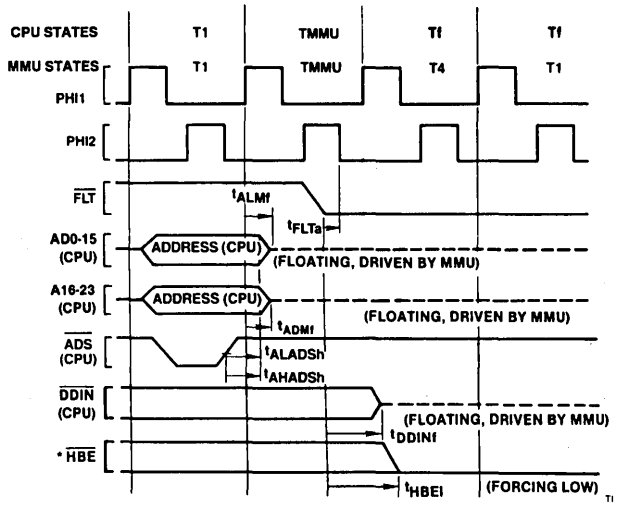
Note that during TI1 the CPU is already idling.



TL/EE/8525-80

FIGURE 4-8. Release from $\overline{\text{HOLD}}$

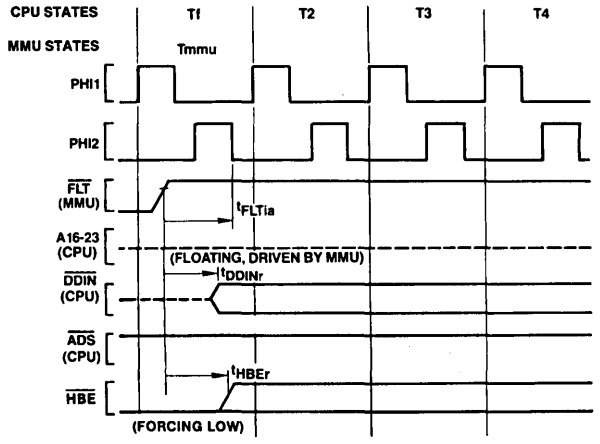
4.0 Device Specifications (Continued)



TL/EE/8525–46

*Note: In future higher speed versions of the NS32C016, $\overline{HBE}$ will no longer be affected by $\overline{FLT}$. See *Figure B-1* in Appendix B for the required modification to the interface logic.

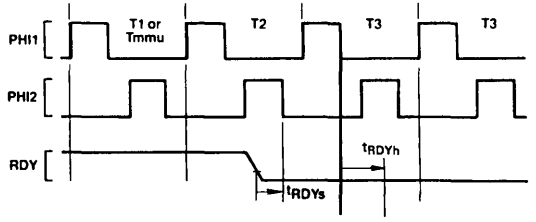
FIGURE 4-9. $\overline{FLT}$ Initiated Cycle Timing



TL/EE/8525–47

Note that when $\overline{FLT}$ is deasserted the CPU restarts driving $\overline{DDIN}$ before the MMU releases it. This, however, does not cause any conflict, since both CPU and MMU force $\overline{DDIN}$ to the same logic level.

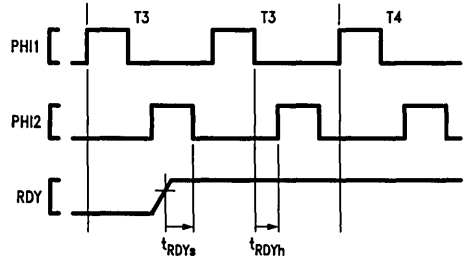
FIGURE 4-10. Release from $\overline{FLT}$ Timing



TL/EE/8525–48

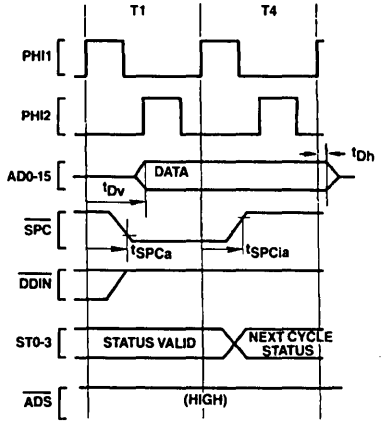
FIGURE 4-11. Ready Sampling (CPU Initially READY)

4.0 Device Specifications (Continued)



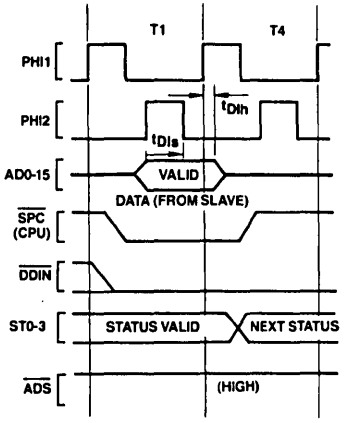
TL/EE/8525-49

FIGURE 4-12. Ready Sampling (CPU Initially NOT READY)



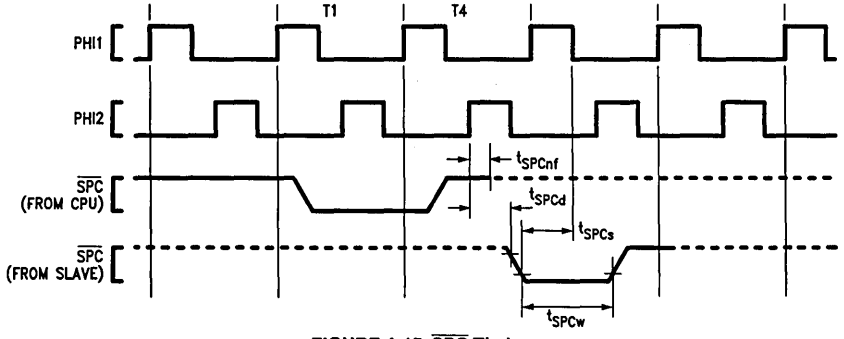
TL/EE/8525-50

FIGURE 4-13. Slave Processor Write Timing



TL/EE/8525-51

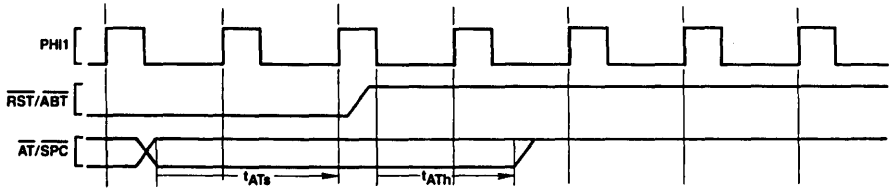
FIGURE 4-14. Slave Processor Read Timing



TL/EE/8525-81

FIGURE 4-15. $\overline{SPC}$ Timing

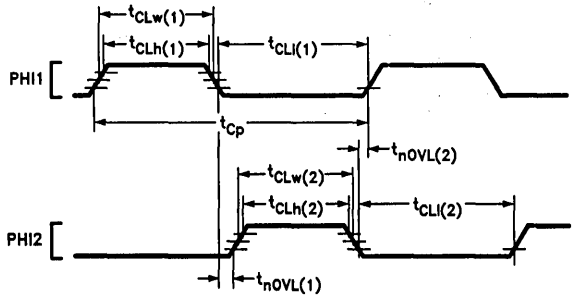
After transferring last operand to a Slave Processor, CPU turns OFF driver and holds $\overline{SPC}$ high with internal 5 kΩ pullup.



TL/EE/8525-53

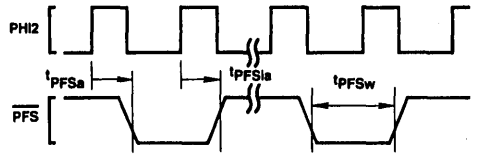
FIGURE 4-16. Reset Configuration Timing

4.0 Device Specifications (Continued)



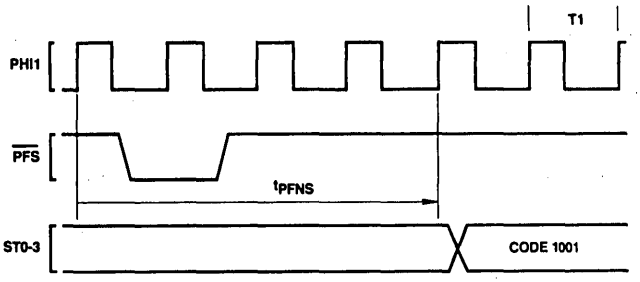
TL/EE/8525-54

FIGURE 4-17. Clock Waveforms



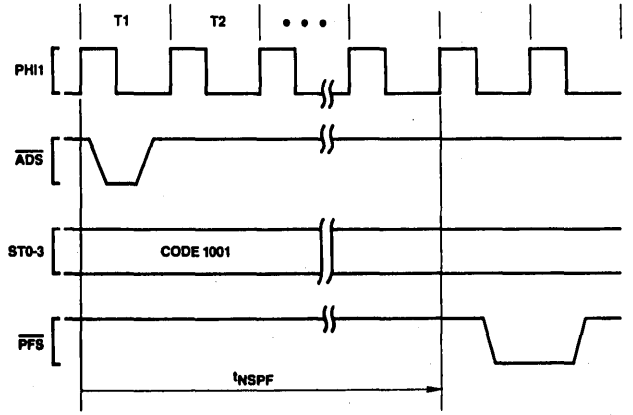
TL/EE/8525-55

FIGURE 4-18. Relationship of $\overline{PFS}$ to Clock Cycles



TL/EE/8525-56

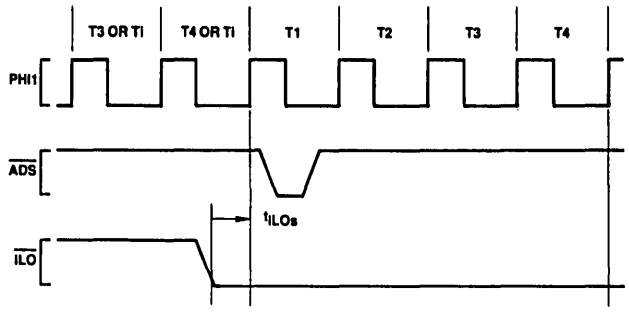
FIGURE 4-19a. Guaranteed Delay, $\overline{PFS}$ to Non-Sequential Fetch



TL/EE/8525-57

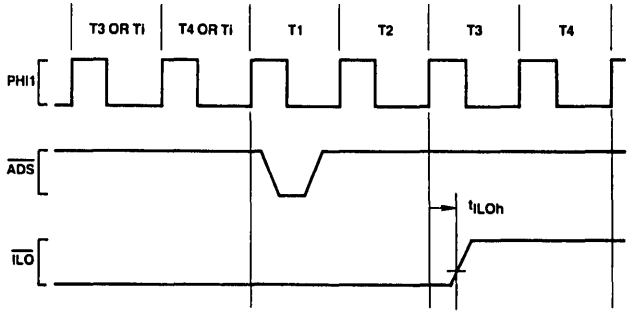
FIGURE 4-19b. Guaranteed Delay, Non-Sequential Fetch to $\overline{PFS}$

4.0 Device Specifications (Continued)



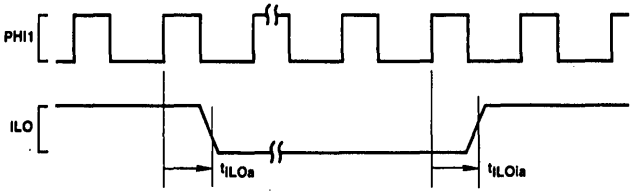
TL/EE/8525-58

FIGURE 4-20a. Relationship of $\overline{ILO}$ to First Operand Cycle of an Interlocked Instruction



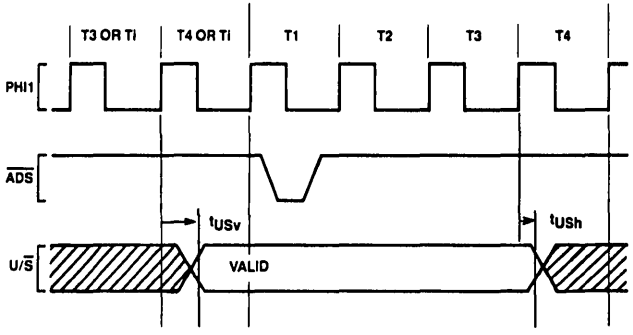
TL/EE/8525-59

FIGURE 4-20b. Relationship of $\overline{ILO}$ to Last Operand Cycle of an Interlocked Instruction



TL/EE/8525-60

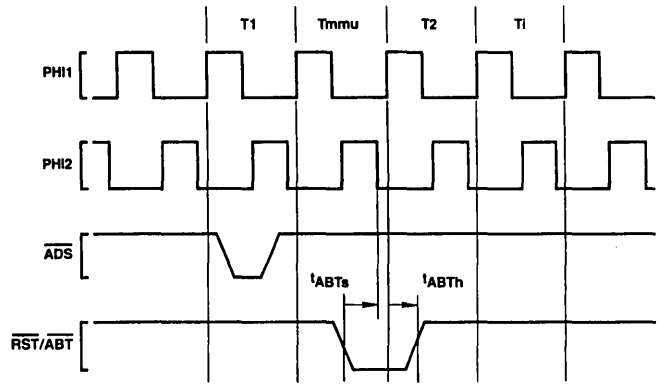
FIGURE 4-21. Relationship of $\overline{ILO}$ to Any Clock Cycle



TL/EE/8525-61

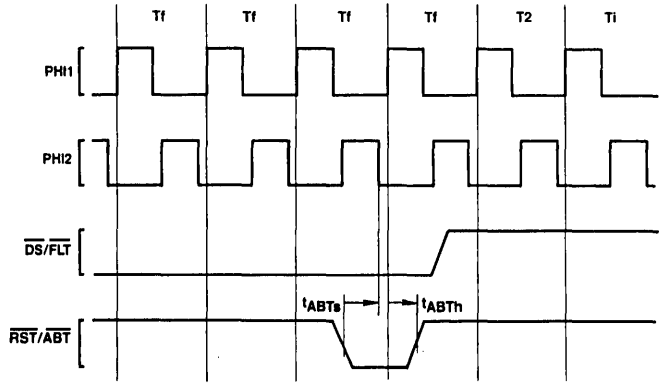
FIGURE 4-22. U/$\overline{S}$ Relationship to Any Bus Cycle—Guaranteed Valid Interval

4.0 Device Specifications (Continued)



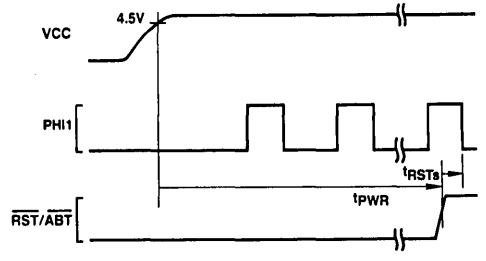
TL/EE/8525-62

FIGURE 4-23. Abort Timing, $\overline{FLT}$ Not Applied



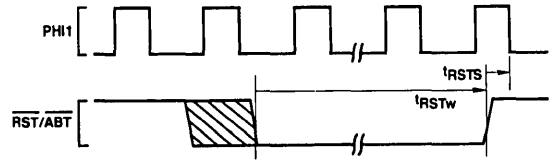
TL/EE/8525-63

FIGURE 4-24. Abort Timing, $\overline{FLT}$ Applied



TL/EE/8525-64

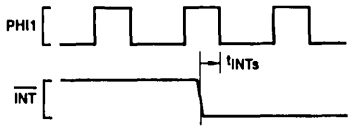
FIGURE 4-25. Power-On Reset



TL/EE/8525-65

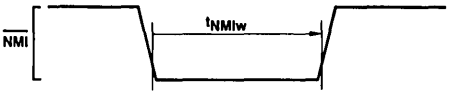
FIGURE 4-26. Non-Power-On Reset

4.0 Device Specifications (Continued)



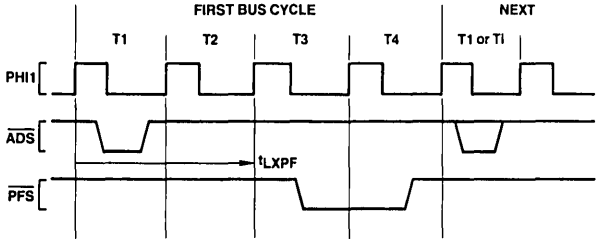
TL/EE/8525-66

FIGURE 4-27. $\overline{INT}$ Interrupt Signal Detection



TL/EE/8525-67

FIGURE 4-28. $\overline{NMI}$ Interrupt Signal Timing



TL/EE/8525-68

FIGURE 4-29. Relationship Between Last Data Transfer of an Instruction and $\overline{PFS}$ Pulse of Next Instruction

NOTE:

In a transfer of a Read-Modify-Write type operand, this is the Read transfer, displaying RMW Status (Code 1011).

Appendix A: Instruction Formats

NOTATIONS:

i = Integer Type Field
- B = 00 (Byte)
- W = 01 (Word)
- D = 11 (Double Word)

f = Floating Point Type Field
- F = 1 (Std. Floating: 32 bits)
- L = 0 (Long Floating: 64 bits)

c = Custom Type Field
- D = 1 (Double Word)
- Q = 0 (Quad Word)

op = Operation Code
- Valid encodings shown with each format.

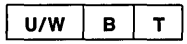
gen, gen 1, gen 2 = General Addressing Mode Field
- See Sec. 2.2 for encodings.

reg = General Purpose Register Number

cond = Condition Code Field
- 0000 = EQual: Z = 1
- 0001 = Not Equal: Z = 0
- 0010 = Carry Set: C = 1
- 0011 = Carry Clear: C = 0
- 0100 = Higher: L = 1
- 0101 = Lower or Same: L = 0
- 0110 = Greater Than: N = 1
- 0111 = Less or Equal: N = 0
- 1000 = Flag Set: F = 1
- 1001 = Flag Clear: F = 0
- 1010 = LOwer: L = 0 and Z = 0
- 1011 = Higher or Same: L = 1 or Z = 1
- 1100 = Less Than: N = 0 and Z = 0
- 1101 = Greater or Equal: N = 1 or Z = 1
- 1110 = (Unconditionally True)
- 1111 = (Unconditionally False)

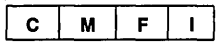
short = Short Immediate Value. May contain:
- quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
- cond: Condition Code (above), in Scond.
- areg: CPU Dedicated Register, in LPR, SPR.
  - 0000 = US
  - 0001 – 0111 = (Reserved)
  - 1000 = FP
  - 1001 = SP
  - 1010 = SB
  - 1011 = (Reserved)
  - 1100 = (Reserved)
  - 1101 = PSR
  - 1110 = INTBASE
  - 1111 = MOD

Options: in String Instructions

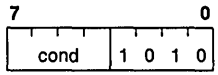


T = Translated
B = Backward
U/W = 00: None
- 01: While Match
- 11: Until Match

Configuration bits, in SETCFG:

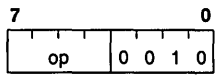


mreg: NS32082 Register number, in LMR, SMR.
- 0000 = BPR0
- 0001 = BPR1
- 0010 = (Reserved)
- 0011 = (Reserved)
- 0100 = (Reserved)
- 0101 = (Reserved)
- 0110 = (Reserved)
- 0111 = (Reserved)
- 1000 = (Reserved)
- 1001 = (Reserved)
- 1010 = MSR
- 1011 = BCNT
- 1100 = PTB0
- 1101 = PTB1
- 1110 = (Reserved)
- 1111 = EIA



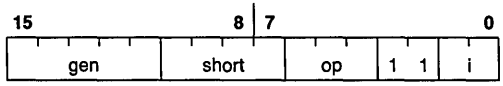
Format 0

Bcond (BR)



Format 1

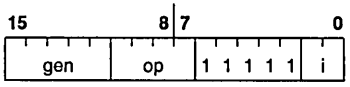
BSR	–0000	ENTER	–1000
RET	–0001	EXIT	–1001
CXP	–0010	NOP	–1010
RXP	–0011	WAIT	–1011
RETT	–0100	DIA	–1100
RETI	–0101	FLAG	–1101
SAVE	–0110	SVC	–1110
RESTORE	–0111	BPT	–1111



Format 2

ADDQ	–000	ACB	–100
CMPQ	–001	MOVQ	–101
SPR	–010	LPR	–110
Scond	–011		

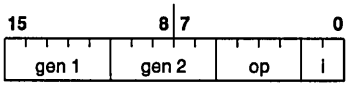
Appendix A: Instruction Formats (Continued)



Format 3

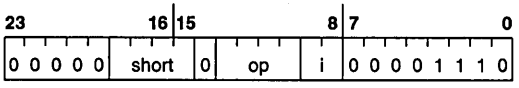
CXPD	−0000	ADJSP	−1010
BICPSR	−0010	JSR	−1100
JUMP	−0100	CASE	−1110
BISPSR	−0110		

Trap (UND) on XXX1, 1000



Format 4

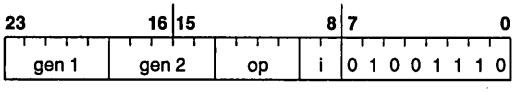
ADD	−0000	SUB	−1000
CMP	−0001	ADDR	−1001
BIC	−0010	AND	−1010
ADDC	−0100	SUBC	−1100
MOV	−0101	TBIT	−1101
OR	−0110	XOR	−1110



Format 5

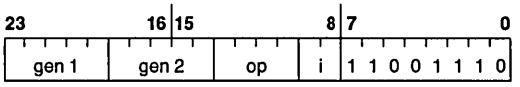
MOVS	−0000	SETCFG	−0010
CMPS	−0001	SKPS	−0011

Trap (UND) on 1XXX, 01XX



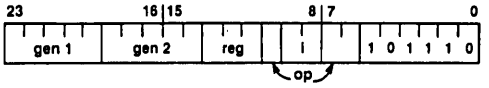
Format 6

ROT	−0000	NEG	−1000
ASH	−0001	NOT	−1001
CBIT	−0010	Trap (UND)	−1010
CBITI	−0011	SUBP	−1011
Trap (UND)	−0100	ABS	−1100
LSH	−0101	COM	−1101
SBIT	−0110	IBIT	−1110
SBITI	−0111	ADDP	−1111



Format 7

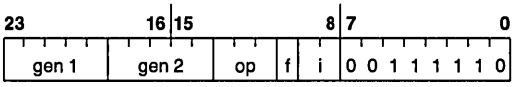
MOVM	−0000	MUL	−1000
CMPM	−0001	MEI	−1001
INSS	−0010	Trap (UND)	−1010
EXTS	−0011	DEI	−1011
MOVXBW	−0100	QUO	−1100
MOVZBW	−0101	REM	−1101
MOVZiD	−0110	MOD	−1110
MOVXiD	−0111	DIV	−1111



TL/EE/8525-69

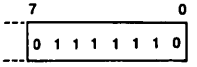
Format 8

EXT	−000	INDEX	−100
CVTP	−001	FFS	−101
INS	−010		
CHECK	−011		
MOVSU	−110, reg=001		
MOVUS	−110, reg=011		



Format 9

MOVif	−000	ROUND	−100
LFSR	−001	TRUNC	−101
MOVLF	−010	SFSR	−110
MOVFL	−011	FLOOR	−111

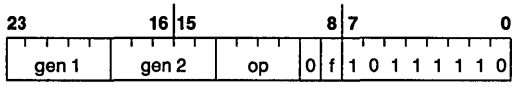


TL/EE/8525-70

Format 10

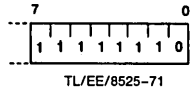
Trap (UND) Always

Appendix A: Instruction Formats (Continued)



Format 11

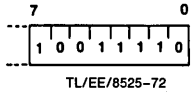
ADDf	−0000	DIVf	−1000
MOVf	−0001	Trap (SLAVE)	−1001
CMPf	−0010	Trap (UND)	−1010
Trap (SLAVE)	−0011	Trap (UND)	−1011
SUBf	−0100	MULf	−1100
NEGf	−0101	ABSf	−1101
Trap (UND)	−0110	Trap (UND)	−1110
Trap (UND)	−0111	Trap (UND)	−1111



TL/EE/8525-71

Format 12

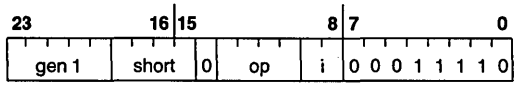
Trap (UND) Always



TL/EE/8525-72

Format 13

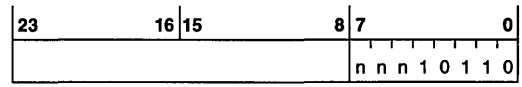
Trap (UND) Always



Format 14

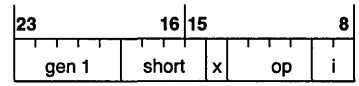
RDVAL	−0000	LMR	−0010
WRVAL	−0001	SMR	−0011

Trap (UND) on 01XX, 1XXX



Format 15
(Custom Slave)

nnn **Operation Word Format**

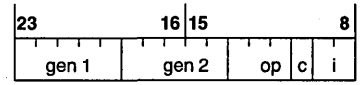


000

Format 15.0

CATST0	−0000	LCR	−0010
CATST1	−0001	SCR	−0011

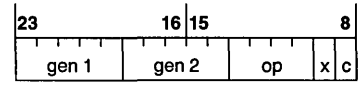
Trap (UND) on all others



001

Format 15.1

CCV3	−000	CCV2	−100
LCSR	−001	CCV1	−101
CCV5	−010	SCSR	−110
CCV4	−011	CCV0	−111



101

Format 15.5

CCAL0	−0000	CCAL3	−1000
CMOV0	−0001	CMOV3	−1001
CCMP0	−0010	Trap (UND)	−1010
CCMP1	−0011	Trap (UND)	−1011
CCAL1	−0100	CCAL2	−1100
CMOV2	−0101	CMOV1	−1101
Trap (UND)	−0110	Trap (UND)	−1110
Trap (UND)	−0111	Trap (UND)	−1111

If nnn = 010, 011, 100, 110, 111
then Trap (UND) Always

Appendix A: Instruction Formats (Continued)

TL/EE/8525-73

Format 16

Trap (UND) Always

TL/EE/8525-75

Format 17

Trap (UND) Always

TL/EE/8525-76

Format 18

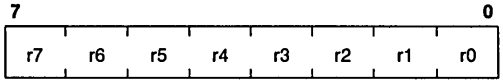
Trap (UND) Always

TL/EE/8525-74

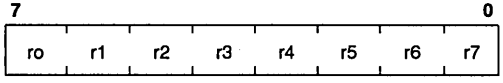
Format 19

Trap (UND) Always

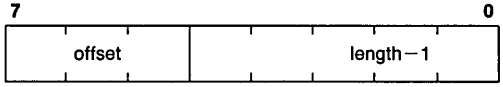
Implied Immediate Encodings:



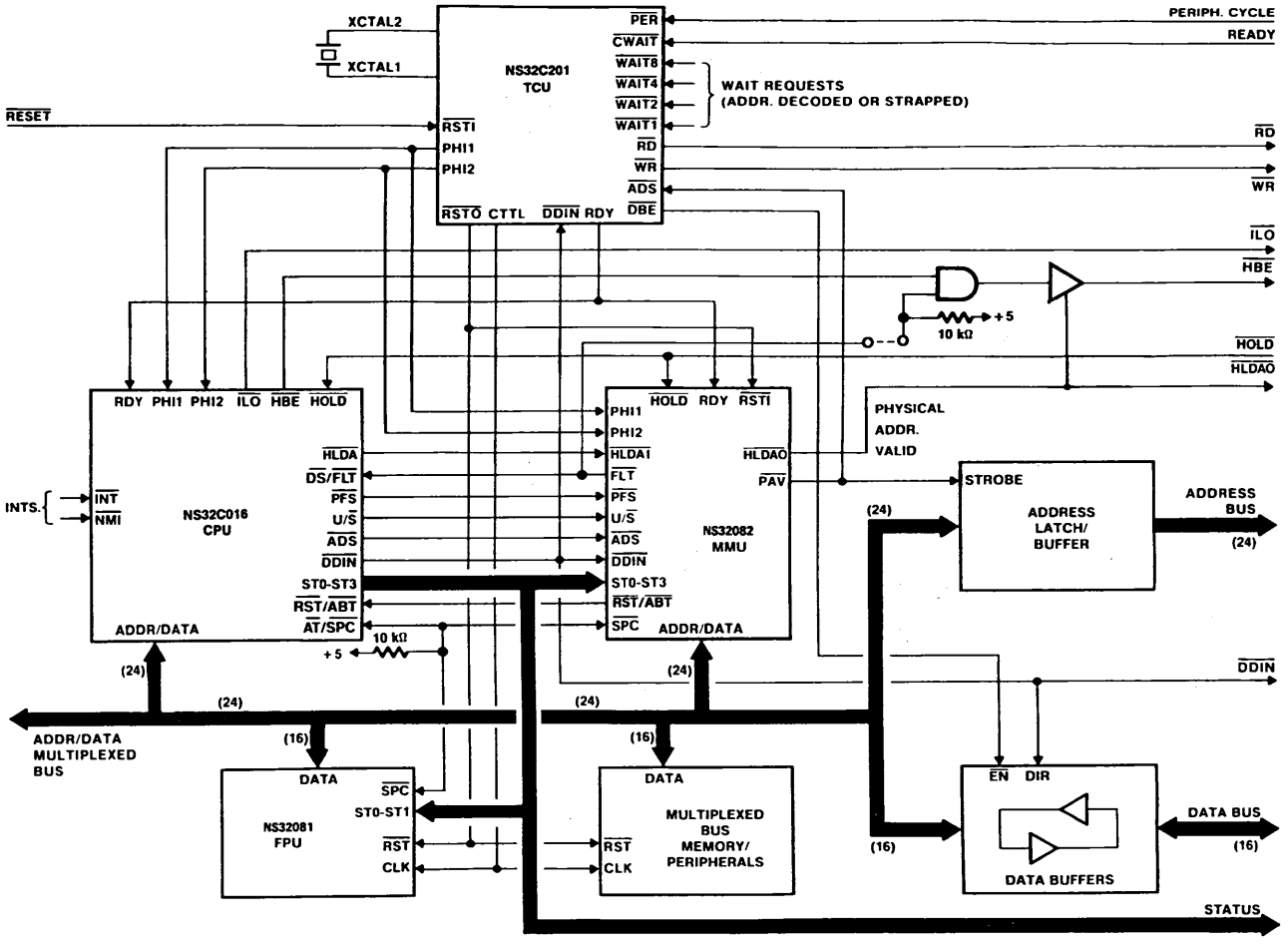
Register Mask, appended to SAVE, ENTER



Register Mask, appended to RESTORE, EXIT



Offset/Length Modifier appended to INSS, EXTS



TL/EE/8525-77

FIGURE B-1. System Connection Diagram

Section 3

Slave Processors

Section 3 Contents

National Semiconductor

PRELIMINARY

NS32382-10/NS32382-15 Memory Management Units

General Description

The NS32382 Memory Management Unit (MMU) provides hardware support for demand-paged virtual memory implementations. The NS32382 functions as a slave processor in Series 32000 microprocessor-based systems. Its specific capabilities include fast dynamic translation, protection, and detailed status to assist an operating system in efficiently managing up to 4 Gbytes of physical memory. Support for multiple address spaces, virtual machines, and program debugging is provided.

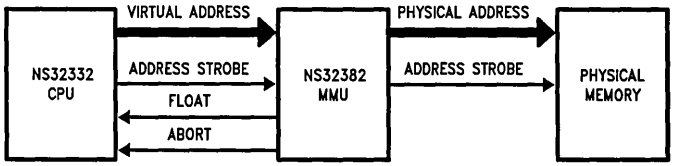
High-speed address translation is performed on-chip through a 32-entry fully associative translation look-aside buffer (TLB), which maintains itself from tables in memory with no software intervention. Protection violations and page faults (references to non-resident pages) are automatically detected by the MMU, which invokes the instruction abort feature of the CPU.

Additional features for program debugging include three breakpoint registers which provide the programmer with powerful stand-alone debugging capability.

Features

- Compatible with the NS32332 CPU
- Totally automatic mapping of 4 Gbyte virtual address space using memory based tables
- On-chip translation look-aside buffer allows 97% of translations to occur in one clock for most applications
- Full hardware support for virtual memory and virtual machines
- Implements "referenced" bits for simple, efficient working set management
- Protection mechanisms implemented via access level checking and dual space mapping
- Program debugging support
- Dedicated 32-bit physical address bus
- Non-cacheable page support
- 125-pin PGA (Pin grid array) package

Conceptual Address Translation Model



TL/EE/9142-1

3

Table Of Contents

List of Illustrations

List of Illustrations (Continued)

Tables

1.0 Product Introduction

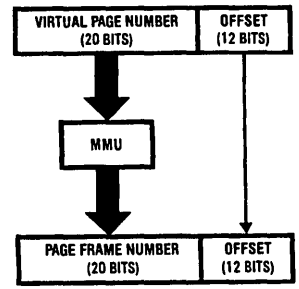
The NS32382 MMU provides hardware support for three basic features of the Series 32000; dynamic address translation, access level checking and software debugging. Dynamic Address Translation is required to implement demand-paged virtual memory. Access level checking is performed during address translation, ensuring that unauthorized accesses do not occur. Because the MMU resides on the local bus and is in an ideal location to monitor CPU activity, debugging functions are also included.

The MMU is intended for use in implementing demand-paged virtual memory. The concept of demand-paged virtual memory is illustrated in *Figure 1-1.* At any point in time, a program sees a uniform addressing space of up to 4 gigabytes (the "virtual" space), regardless of the actual size of the memory physically present in the system (the "physical" space). The full virtual space is recorded as an image on a mass storage device. Portions of the virtual space needed by a running program are copied into physical memory when needed.

To make the virtual information directly available to a running program, a mapping must be established between the virtual addresses asserted by the CPU and the physical addresses of the data being referenced.

To perform this mapping, the MMU divides the virtual memory space into 4 Kbyte blocks called "pages". It interprets the 32-bit address from the CPU as a 20-bit "page number" followed by a 12-bit offset, which indicates the position of a byte within the selected page. Similarly, the MMU divides the physical memory into 4 Kbyte frames, each of which can hold a virtual page.

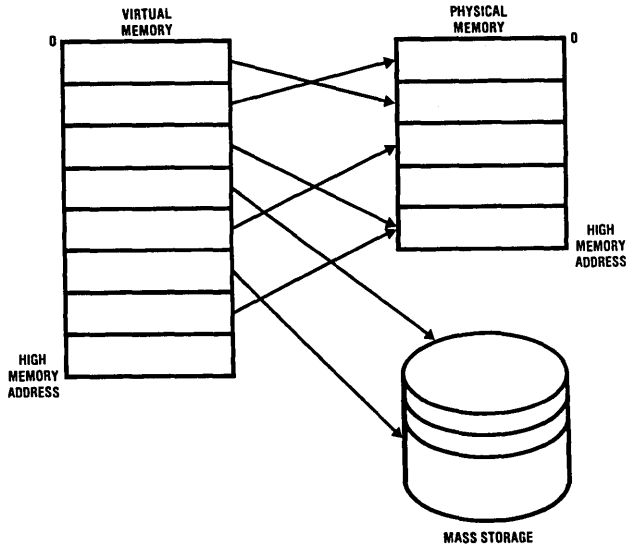
The translation process is therefore modeled as accepting a virtual page number from the CPU and substituting the corresponding physical page frame number for it, as shown in *Figure 1-2.* The offset is not changed. The translated page frame number is 20 bits long. Physical addresses issued by the MMU are 32 bits wide.



TL/EE/9142-3

FIGURE 1-2. NS32382 Address Translation Model

Generally, in virtual memory systems the available physical memory space is smaller than the maximum virtual memory space. Therefore, not all virtual pages are simultaneously resident. Nonresident pages are not directly addressable by the CPU. Whenever the CPU issues a virtual address for a nonresident or nonexistent page, a "page fault" will result. The MMU signals this condition by invoking the Abort feature of the CPU. The CPU then halts the memory cycle, restores its internal state to the point prior to the instruction being executed, and enters the operating system through the abort trap vector.



TL/EE/9142-2

FIGURE 1-1. The Virtual Memory Model

1.0 Product Introduction (Continued)

The operating system reads from the MMU the virtual address which caused the abort. It selects a page frame which is either vacant or not recently used and, if necessary, writes this frame back to mass storage. The required virtual page is then copied into the selected page frame.

The MMU is informed of this change by updating the page tables (Section 3.2), and the operating system returns control to the aborted program using the RETT instruction. Since the return address supplied by the abort trap is the address of the aborted instruction, execution resumes by retrying the instruction.

This sequence is called paging. Since a page fault encountered in normal execution serves as a demand for a given page, the whole scheme is called demand-paged virtual memory.

The MMU also provides debugging support. It may be programmed to monitor the CPU bus for a single or a range of virtual addresses in real time.

1.1 PROGRAMMING CONSIDERATIONS

When a CPU instruction is aborted as a result of a page fault, some memory resident data might have been already modified by the instruction before the occurrence of the abort.

This could compromise the restartability of the instruction when the CPU returns from the abort routine.

To guarantee correct results following the re-execution of the aborted instruction, the following actions should not be attempted:

a) No instruction should try to overlay part of a source operand with part of the result. It is, however, permissible to rewrite the result into the source operand exactly, if page faults are being generated only by invalid pages and not by write protection violations (for example, the instruction "ABSW X, X", which replaces X with its absolute value). Also, never write to any memory location which is necessary for calculating the effective address of either operand (i.e. the pointer in "Memory Relative" addressing mode; the Link Table pointer or Link Table Entry in "External" addressing mode).

b) No instruction should perform a conversion in place from one data type to another larger data type (Example: MOVWF X, X which replaces the 16-bit integer value in memory location X with its 32-bit floating-point value). The addressing mode combination "TOS, TOS" is an exception, and is allowed. This is because the least-significant part of the result is written to the possibly invalid page before the source operand is affected. Also, integer conversions to larger integers always work correctly in place, because the low-order portion of the result always matches the source value.

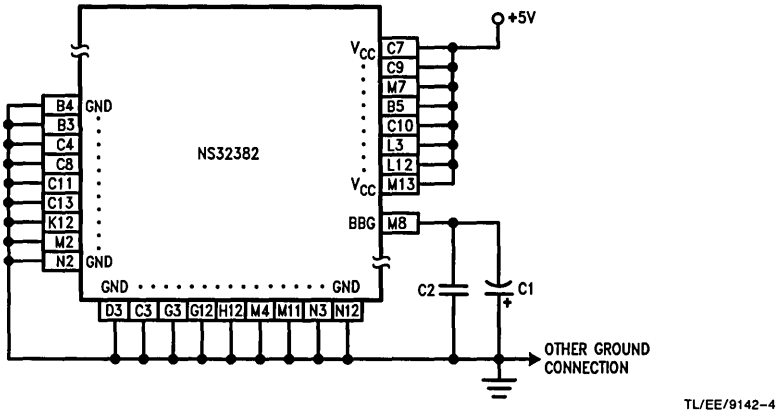
c) When performing the MOVM instruction, the entire source and destination blocks must be considered "operands" as above, and they must not overlap.

2.0 Functional Description

2.1 POWER AND GROUNDING

The NS32382 requires a single 5V power supply, applied on eight (V_{CC}) pins. These pins should be connected together by a power (V_{CC}) plane on the printed circuit board. See *Figure 2-1*.

The grounding connections are made on eighteen (GND) pins.



TL/EE/9142–4

C1 = 1 μF, Tantalum.

C2 = 1000 pF, low inductance. This should be either a disc or monolithic ceramic capacitor.

FIGURE 2-1. Recommended Supply Connections

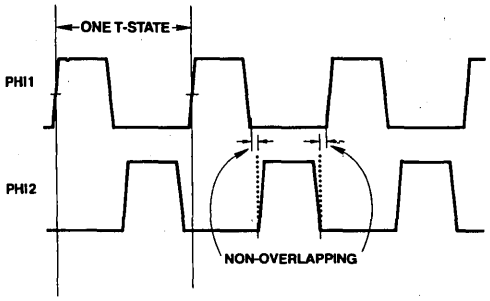
2.0 Functional Description (Continued)

These pins should be connected together by a ground (GND) plane on the printed circuit board.

In addition to V_{CC} and Ground, the NS32382 MMU uses an internally-generated negative voltage (BBG), output of the on-chip substrate voltage generator. It is necessary to filter this voltage externally by attaching a pair of capacitors *(Figure 2-1)* from the BBG pin to ground.

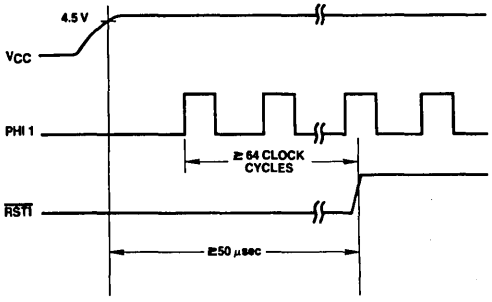
2.2 CLOCKING

The NS32382 inputs clocking signals from the NS32301 Timing Control Unit (TCU), which presents two non-overlapping phases of a single clock frequency. These phases are called PHI1 (pin B8) and PHI2 (pin B9). Their relationship to each other is shown in *Figure 2-2*.



TL/EE/9142–5

FIGURE 2-2. Clock Timing Relationships



TL/EE/9142–6

FIGURE 2-3. Power-On Reset Requirements

Each rising edge of PHI1 defines a transition in the timing state ("T-State") of the MMU. One T-State represents one hardware cycle within the MMU, and/or one step of an external bus transfer. See Section 4 for complete specifications of PHI1 and PHI2.

As the TCU presents signals with very fast transitions, it is recommended that the conductors carrying PHI1 and PHI2 be kept as short as possible, and that they not be connected to any devices other than the CPU and MMU. A TTL Clock signal (CTTL) is provided by the TCU for all other clocking.

2.3 RESETTING

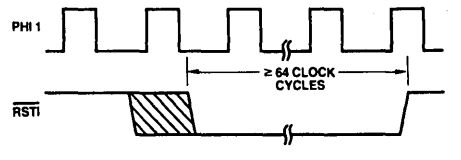
The $\overline{RSTI}$ input pin is used to reset the NS32382. The MMU responds to $\overline{RSTI}$ by terminating processing, resetting its internal logic and clearing the MCR and MSR registers.

Only the MCR and MSR registers are changed on reset. No other program accessible registers are affected.

The $\overline{RST}/\overline{ABT}$ signal is activated by the MMU on reset. This signal should be used to reset the CPU.

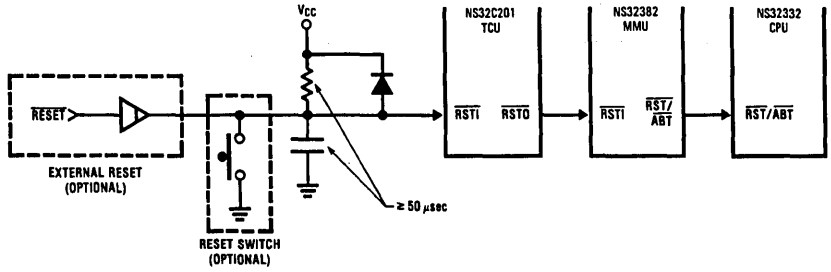
On application of power, $\overline{RSTI}$ must be held low for at least 50 μs after V_{CC} is stable. This is to ensure that all on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must also remain active for not less than 64 clock cycles. See *Figures 2-3* and *2-4*.

The NS32C201 Timing Control Unit (TCU) provides circuitry to meet the Reset requirements of the NS32382 MMU. *Figure 2-5* shows the recommended connections.



TL/EE/9142–7

FIGURE 2-4. General Reset Timing



TL/EE/9142–8

FIGURE 2-5. Recommended Reset Connections, Memory-Managed System

2.0 Functional Description (Continued)

2.4 BUS OPERATION

2.4.1 Interconnections

The MMU runs synchronously with the CPU, sharing with it a single multiplexed address/data bus. The interconnections used by the MMU for bus control, when used in conjunction with the NS32332, are shown in *Figure A-1* (Appendix A).

The CPU issues 32-bit virtual addresses on the bus, and status information on other pins, pulsing the signal $\overline{ADS}$ low. These are monitored by the MMU. The MMU issues 32-bit physical addresses on the Physical Address bus, pulsing the $\overline{PAV}$ line low. The $\overline{PAV}$ pulse triggers the address latches and signals the NS32C201 TCU to begin a bus cycle. The TCU in turn generates the necessary bus control signals and synchronizes the insertion of WAIT states, by providing the signal RDY to the MMU and CPU. Note that it is the MMU rather than the CPU that actually triggers bus activity in the system.

The functions of other interface signals used by the MMU to control bus activity are described below.

The ST0–ST3 pins indicate the type of cycle being initiated by the CPU. ST0 is the least-significant bit of the code. Table 2-1 shows the interpretations of the status codes presented on these lines.

Status codes that are relevant to the MMU's function during a memory reference are:

1000, 1001 Instruction Fetch status, used by the debugging features to distinguish between data and instruction references.

1010 Data Transfer. A data value is to be transferred.

1011 Read RMW Operand. Although this is always a Read cycle, the MMU treats it as a Write cycle for purposes of protection and breakpointing.

1100 Read for Effective Address. Data used for address calculation is being transferred.

The MMU ignores all other status codes. The status codes 1101, 1110 and 1111 are also recognized by the MMU in conjunction with pulses on the $\overline{SPC}$ line while it is executing Slave Processor instructions, but these do not occur in a context relevant to address translation.

TABLE 2-1. ST0–ST3 Encodings (ST0 is the Least Significant)

0000 — Idle: CPU Inactive on Bus
0001 — Idle: WAIT Instruction
0010 — (Reserved)
0011 — Idle: Waiting for Slave
0100 — Interrupt Acknowledge, Master
0101 — Interrupt Acknowledge, Cascaded
0110 — End of Interrupt, Master
0111 — End of Interrupt, Cascaded
1000 — Sequential Instruction Fetch
1001 — Non-Sequential Instruction Fetch
1010 — Data Transfer
1011 — Read Read-Modify-Write Operand
1100 — Read for Effective Address
1101 — Transfer Slave Operand
1110 — Read Slave Status Word
1111 — Broadcast Slave ID and Operation Word

The $\overline{DDIN}$ line indicates the direction of the transfer: 0 = Read, 1 = Write.

$\overline{DDIN}$ is monitored by the MMU during CPU cycles to detect write operations, and is driven by the MMU during MMU-initiated bus cycles.

The U/$\overline{S}$ pin indicates the privilege level at which the CPU is making the access: 0 = Supervisor Mode, 1 = User Mode. It is used by the MMU to select the address space for translation and to perform protection level checking. Normally, the U/$\overline{S}$ pin is a direct reflection of the U bit in the CPU's Processor Status Register (PSR). The MOVUS and MOVSU CPU instructions, however, toggle this pin on successive operand accesses in order to move data between virtual spaces.

The MMU uses the $\overline{FLT}$ line to take control of the bus from the CPU. It does so as necessary for updating its internal TLB from the Page Tables in memory, and for maintaining the contents of the status bits (R and M) in the Page Table Entries.

The MMU also aborts invalid accesses attempted by the CPU. This is done by pulsing the $\overline{RST}/\overline{ABT}$ pin low for one clock period. (A pulse longer than one clock period is interpreted by the CPU as a Reset command.)

2.4.2 CPU-Initiated Bus Cycles

A CPU-initiated bus cycle is performed in a minimum of four clock cycles: T1, T2, T3 and T4, as shown in *Figure 2-6*.

During period T1, the CPU places the virtual address to be translated on the bus, and the MMU latches it internally and begins translation. The MMU also samples the $\overline{DDIN}$ pin, the status lines ST0–ST3, and the U/$\overline{S}$ pin in the previous T4 cycle to determine how the CPU intends to use the bus.

During period T2 the CPU removes the virtual address from the bus and the MMU takes one of three actions:

1) If the translation for the virtual address is resident in the MMU's TLB, and the access being attempted by the CPU does not violate the protection level of the page being referenced, the MMU presents the translated address on PA0–PA31 and generates a $\overline{PAV}$ pulse to trigger a bus cycle in the rest of the system. See *Figure 2-6*.

2) If the translation for the virtual address is resident in the MMU's TLB, but the access being attempted by the CPU is not allowed due to the protection level of the page being referenced, the MMU generates a pulse on the $\overline{RST}/\overline{ABT}$ pin to abort the CPU's access. No $\overline{PAV}$ pulse is generated. See *Figure 2-7*.

3) If the translation for the virtual address is not resident in the TLB, or if the CPU is writing to a page whose M bit is not yet set, the MMU takes control of the bus asserting the $\overline{FLT}$ signal as shown in *Figure 2-8*. This causes the CPU to float its bus and wait. The MMU then initiates a sequence of bus cycles as described in Section 2.4.3.

From state T2 through T4 data is transferred on the bus between the CPU and memory, and the TCU provides the strobes for the transfer.

Whenever the MMU generates an Abort pulse on the $\overline{RST}/\overline{ABT}$ pin, the CPU enters state T3 and then Ti (idle), ending the bus cycle. Since no $\overline{PAV}$ pulse is issued by the MMU, the rest of the system remains unaware that an access has been attempted.

2.0 Functional Description (Continued)

2.4.3 MMU-Initiated Cycles

Bus cycles initiated by the MMU are always nested within CPU-initiated bus cycles; that is, they appear after the MMU has accepted a virtual address from the CPU and has set the $\overline{FLT}$ line active. The MMU will initiate memory cycles in the following cases:

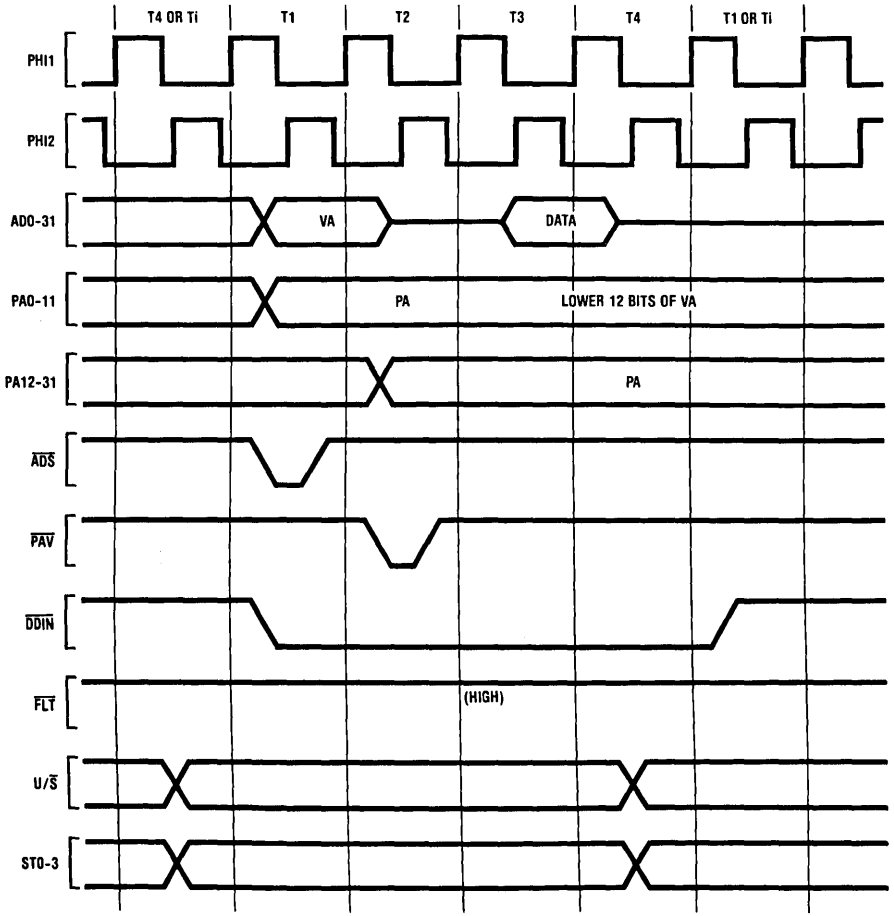
1) There is no translation in the MMU's TLB for the virtual address issued by the CPU, meaning that the MMU must reference the Page Tables in memory to obtain the translation.
2) There is a translation for that virtual address in the TLB, but the page is being written for the first time (the M bit in its Level-2 Page Table Entry is 0). The MMU treats this case as if there were no translation in the TLB, and performs a Page Table lookup in order to set the M bit in the Level-2 Page Table Entry as well as in the TLB.

Having made the necessary memory references, the MMU either aborts the CPU access or it provides the translated address and allows the CPU's access to continue to T3.

Figure 2-8 shows the sequence of events in a Page Table lookup. After asserting $\overline{FLT}$, the MMU waits for one additional clock cycle, then reads the Level-1 Page Table Entry and the Level-2 Page Table Entry in two consecutive memory Read cycles. There are no idle clock cycles between MMU-initiated bus cycles unless a bus request is made on the $\overline{HOLD}$ line (Section 2.6).

During the Page Table lookup the MMU drives the $\overline{DDIN}$ signal. The status lines ST0–ST3 and the U/$\overline{S}$ pin are not released by the CPU, and retain their original settings while the MMU uses the bus. The Byte Enable signals from the CPU, $\overline{BE0}$–$\overline{BE3}$, should be handled externally for correct memory referencing.

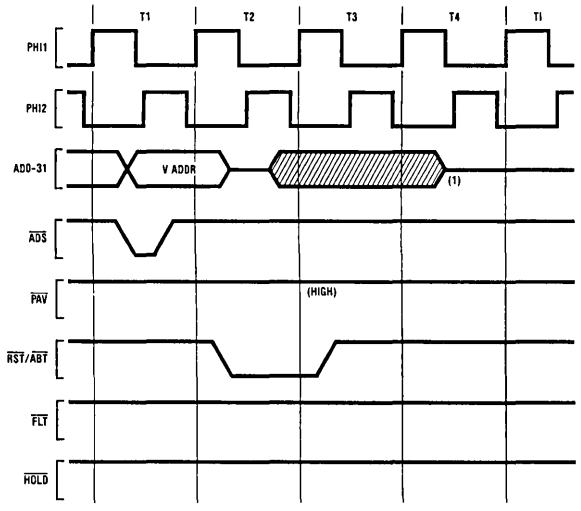
In the clock cycle immediately after T4 of the last lookup cycle, the MMU issues the translated address and pulses $\overline{MADS}$. In the subsequent cycle it removes $\overline{FLT}$ and pulses $\overline{PAV}$ to continue the CPU's access.



TL/EE/9142-9

FIGURE 2-6. CPU Read Cycle; Translation in TLB (TLB Hit)

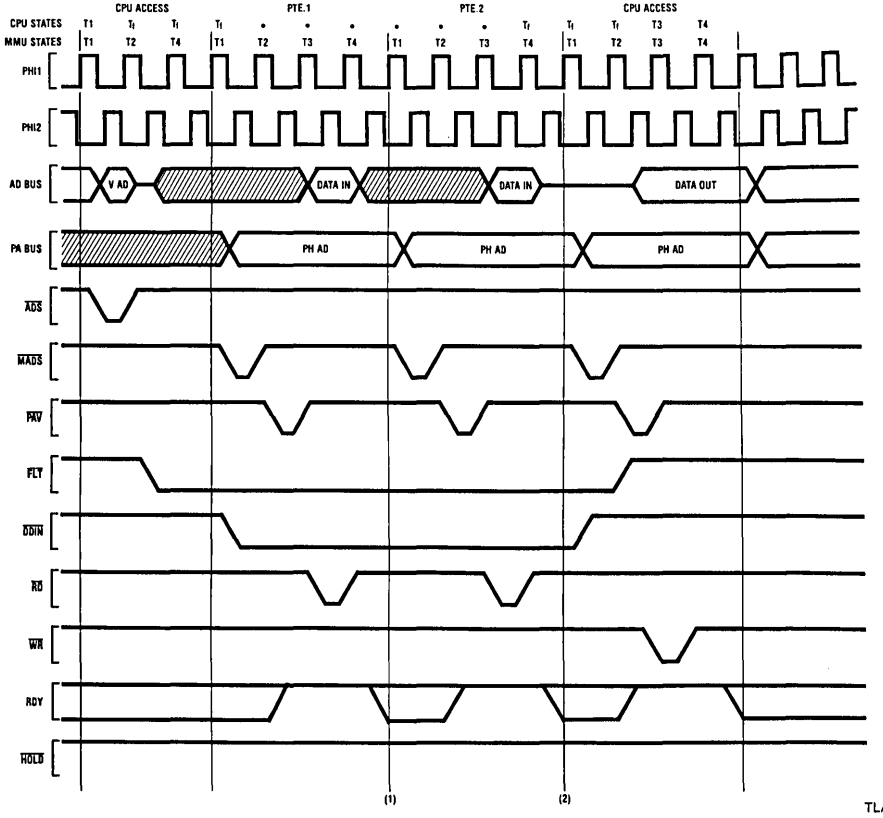
2.0 Functional Description (Continued)



TL/EE/9142-10

Note 1: The CPU drives the bus if a write cycle is aborted.

FIGURE 2-7. Abort Resulting from Protection Violation or a Breakpoint; Translation in TLB



TL/EE/9142-11

Note 1: If the R bit on the Level-1 PTE must be set, a write cycle is inserted here.

Note 2: If either the R or the M bit on the Level-2 PTE must be set, a write cycle is inserted here.

FIGURE 2-8. Page Table Lookup

2.0 Functional Description (Continued)

If the V bit (Bit 0) in any of the Page Table Entries is zero, or the protection level field PL (bits 1 and 2) indicates that the CPU's attempted access is illegal, the MMU does not generate any further memory cycles, but instead issues an Abort pulse during the clock cycle after T4 and removes the $\overline{FLT}$ signal.

If the R and/or M bit (bit 7 or 8) must be updated, the MMU does this immediately in a single Write cycle. All bits except those updated are rewritten with their original values.

At most, the MMU writes two double words to memory during a translation: the first to the Level-1 table to update the R bit, and the second to the Level-2 table to update the R and/or M bits.

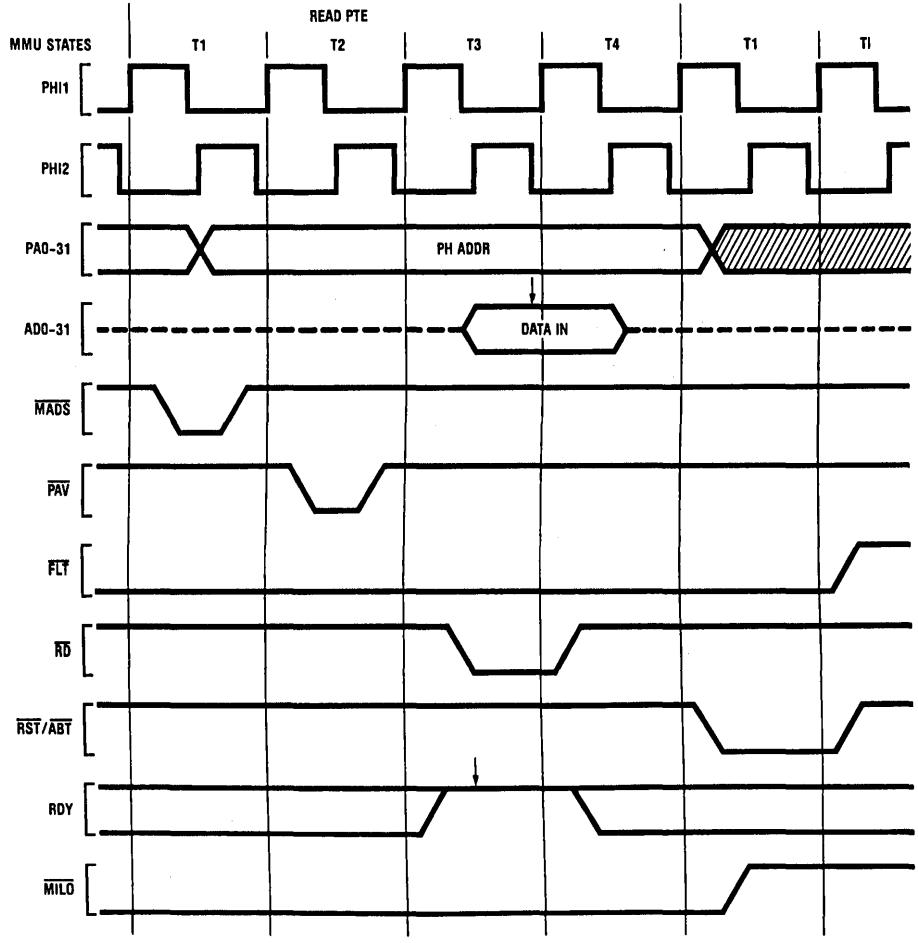
2.4.4 Cycle Extension

To allow sufficient strobe widths and access time requirements for any speed of memory or peripheral device, the NS32382 provides for extension of a bus cycle. Any type of bus cycle, CPU-initiated or MMU-initiated, can be extended, except Slave Processor cycles, which are not memory or peripheral references.

In *Figures 2-6* and *2-8*, note that during T3 all bus control signals are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the RDY (Ready) pin.

In the middle of T3, on the falling edge of clock phase PHI1, the RDY line is sampled by the CPU and/or the MMU. If RDY is high, the next state after T3 will be T4, ending the bus cycle. If it is low, the next state after T3 will be another T3 and the RDY line will be sampled again. RDY is sampled in each following clock period, with insertion of additional T3 states, until it is sampled high. Each additional T3 state inserted is called a "WAIT state".

The RDY pin is driven by the NS32C201 Timing Control Unit, which applies WAIT states to the CPU and MMU as requested on its own WAIT request input pins.



TL/EE/9142-12

FIGURE 2-9. Abort Resulting after a Page Table Lookup

2.0 Functional Description (Continued)

2.4.5 Bus Retry

The Bus Retry input signal ($\overline{BRT}$) provides a system with the capability of repeating a bus cycle upon the occurrence of a "soft" or correctable error. The system first determines that a correctable error has occurred and then activates the $\overline{BRT}$ input. The MMU then samples this input on the falling edge of PHI1 in both T3 and T4 of a bus cycle. A valid bus retry will be issued as a result of a low being sampled in both T3 and T4.

If the MMU gets a Bus Retry when it is controlling the bus, it will re-run the bus cycle until $\overline{BRT}$ is deactivated.

Any Pending Hold request will not be acknowledged by the MMU if a bus retry is detected and during Hold Acknowledge, the MMU will not recognize the Bus Retry signal.

2.4.6 Bus Error

The Bus Error input signal $\overline{BER}$ will be activated (low) when a "hard" or uncorrectable error occurs within the system (e.g. bus timeout, double ECC error). $\overline{BER}$ will be sampled on the falling edge of PHI1 in T4. If the MMU detects Bus Error while it is controlling the bus, it will store the virtual address which caused the error in the BEAR (Bus Error Address Register), and set the ME bit in the MSR to indicate MMU ERROR. An abort signal $\overline{ABT}$ will be generated and further memory accesses by the MMU will be inhibited. The 32382 then returns bus control to the CPU by releasing the $\overline{FLT}$ signal, ($\overline{FLT}$ = 1). Any pending Hold request will not be acknowledged by the MMU if a bus error is detected.

If the Bus Error signal is received when the CPU is controlling the bus, the MMU will store the virtual address in BEAR, and set the CE bit in the MSR to indicate CPU ERROR.

During the Hold Acknowledge, the MMU will ignore the $\overline{BER}$ signal.

2.4.7 Interlocked Bus Transfers

Both the 32332 CPU and the 32382 MMU are capable of executing interlocked cycles to access a stream of data from memory without intervention from other devices.

Before executing an interlocked access, the 32332 CPU performs a dummy read with Read-Modify-Write status (1011). The MMU handles the dummy read as if it were a real RMW access. The TLB entries will be searched and page table look-up will be performed if a miss occurs. The access level is checked and the CPU will be aborted if write privilege is not currently assigned. The Reference (R) and the Modify (M) bits in the first and second level PTEs, as well as those in the Translation look-aside Buffer, will be updated. By executing the dummy read, the CPU is assured of no MMU intervention when the actual interlocked access is performed.

The 32382 MMU executes interlocked Read-Modify-Write memory cycles to access Page Table Entries (PTEs) and update the Reference (R) and Modify (M) bit in the PTEs when necessary. If the R and/or M bit(s) do not require updating, the write portion of the RMW cycle will not be executed. The memory cycles to access PTEs during execution of RDVAL and WRVAL instructions are not interlocked since R and M bits are not updated.

During interlocked access cycles, the $\overline{MILO}$ signal from the MMU will be asserted. $\overline{MILO}$ has the same timing as $\overline{ILO}$ from the CPU. $\overline{MILO}$ is asserted in the clock cycle immediately before the Read-Modify-Write access and de-activated in the clock cycle following T4 of the write cycle.

The write portion of the Read-Modify-Write access will not be executed if any one of the following conditions occurs:

(1) A bus error has occurred in the read portion of the interlocked access.

(2) The R and/or M bit(s) in the PTE(s) do not require updating.

(3) A protection violation has occurred.

(4) An invalid PTE is detected.

If a bus retry is encountered in an interlocked access, $\overline{MILO}$ will continue to be asserted, and the access will be retried.

2.5 SLAVE PROCESSOR INTERFACE

The CPU and MMU execute four instructions cooperatively. These are LMR, SMR, RDVAL and WRVAL, as described in Section 2.5.2. The MMU takes the role of a Slave Processor in executing these instructions, accepting them as they are issued to it by the CPU. The CPU calculates all effective addresses and performs all operand transfers to and from memory and the MMU. The MMU does not take control of the bus except as necessary in normal operation; i.e., to translate and validate memory addresses as they are presented by the CPU.

The sequence of transfers ("protocol") followed by the CPU and MMU involves a special type of bus cycle performed by the CPU. This "Slave Processor" bus cycle does not involve the issuing of an address, but rather performs a fast data transfer whose purpose is pre-determined by the form of the instruction under execution and by status codes asserted by the CPU.

2.5.1 Slave Processor Bus Cycles

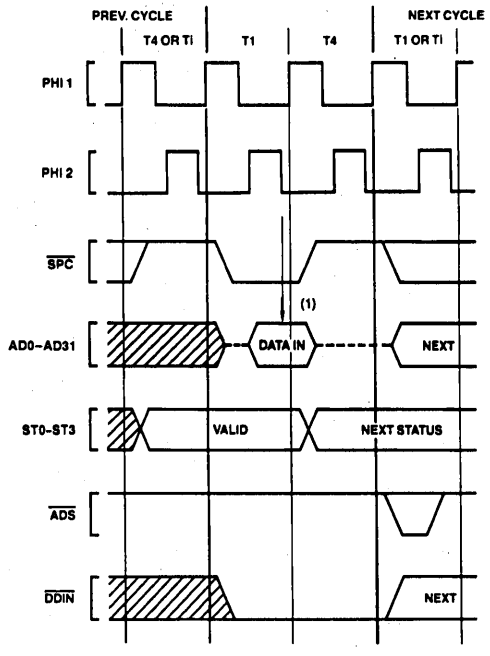
The interconnections between the CPU and MMU for Slave Processor communication are shown in *Figure A-1* (Appendix A). The $\overline{SPC}$ signal is pulsed by the CPU as a low-active data strobe for Slave Processor transfers. Since $\overline{SPC}$ is normally in a high-impedance state, it must be pulled high with a 10 kΩ resistor, as shown. The MMU also monitors the status lines ST0–ST3 to follow the protocol for the instruction being executed.

Data is transferred between the CPU and the MMU with Slave Processor bus cycles, illustrated in *Figures 2-10* and *2-11*. Each bus cycle transfers one double-word (32 bits) to or from the MMU.

Slave Processor bus cycles are performed by the CPU in two clock periods, which are labeled T1 and T4. During T1, the CPU activates $\overline{SPC}$ and, if it is writing to the MMU, it presents data on the bus. During T4, the CPU deactivates $\overline{SPC}$ and, if it is reading from the MMU, it latches data from the bus. The CPU guarantees that data written to the MMU is held through T4 to provide for the MMU's hold time requirements. The CPU also guarantees that the status code on ST0–ST3 becomes valid, at the latest, during the clock period preceding T1. The status code changes during T4 to anticipate the next bus cycle, if any.

Note that Slave Processor bus cycles are never extended with WAIT states. The RDY line is not sampled.

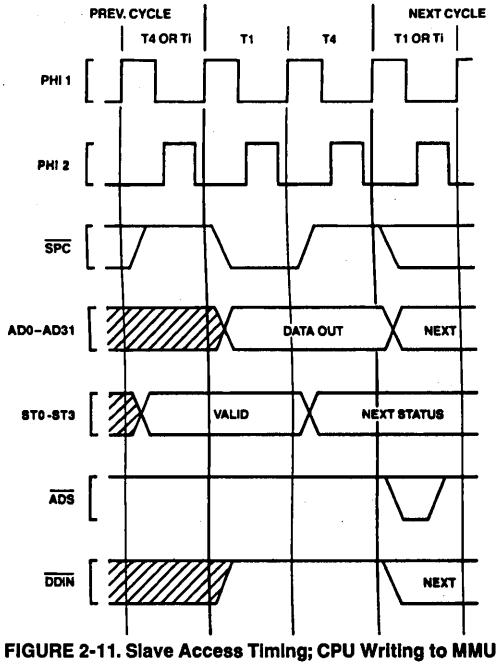
2.0 Functional Description (Continued)



TL/EE/9142–13

Note 1: CPU samples Data Bus here.

FIGURE 2-10. Slave Access Timing; CPU Reading from MMU



TL/EE/9142–14

FIGURE 2-11. Slave Access Timing; CPU Writing to MMU

2.0 Functional Description (Continued)

2.5.2 Instruction Protocols

MMU instructions have a three-byte Basic Instruction field consisting of an ID byte followed by an Operation Word. See *Figure 3-10* for the MMU instruction encodings. The ID Byte has three functions:

1) It identifies the instruction as being a Slave Processor instruction.
2) It specifies that the MMU will execute it.
3) It determines the format of the following Operation Word of the instruction.

The CPU initiates an MMU instruction by issuing the ID Byte and the Operation Word, using Slave Processor bus cycles. While applying status code IIII, the CPU transfers the ID byte on bits AD24–AD31, the operation word on bits AD8–AD23 in a swapped order of bytes and a non-used byte XXXXXXX1 (X = Don't Care) on bits AD0–AD7.

Other actions are taken by the CPU and the MMU according to the instruction under execution, as shown in Tables 2-2, 2-3 and 2-4.

In executing the LMR instruction (Load MMU Register, Table 2-2), the CPU issues the ID Byte, the Operation Word, and then the operand value to be loaded by the MMU. The register to be loaded is specified in a field within the Operation Word of the instruction.

The CPU then waits for the MMU to signal the completion of the instruction by pulsing $\overline{\text{SDONE}}$ low.

In executing the SMR instruction (Store MMU Register, Table 2-3), the CPU also issues the ID Byte and the Operation Word of the instruction to the MMU. It then waits for the MMU to signal (by pulsing $\overline{\text{SDONE}}$ low) that it is ready to present the specified register's contents to the CPU. Upon receiving this "Done" pulse, the CPU reads the contents of the selected register in one Slave Processor bus cycle, and places this result value into the instruction's destination (a CPU general-purpose register or a memory location).

In executing the RDVAL (Read-Validate) or WRVAL (Write-Validate) instruction, the CPU first performs the effective address calculation and obtains the address to be validated. It then issues the ID Byte and the Operation Word to the MMU. It initiates a one-byte Read cycle from the memory address whose protection level is being tested. It does so while presenting status code 1010; this being the only place that this status code appears during a RDVAL or WRVAL instruction. This memory access triggers a special address translation from the MMU. The translation is performed by the MMU using User-Mode mapping, and any protection violation occurring during this memory cycle does not cause an Abort. The MMU will, however, abort the CPU if the Level-1 Page Table Entry is invalid.

Upon completion of the address translation, the MMU pulses $\overline{\text{SDONE}}$ for two clock cycles to acknowledge that the instruction may continue execution and an MMU status read is required.

TABLE 2-2. LMR Instruction Protocol

CPU Action	Status	MMU Action
Issues ID Byte and Operation Word, pulsing $\overline{\text{SPC}}$.	1111	Accepts and decodes instruction.
Accesses memory for effective address calculation and operand fetching or instruction prefetching.	XXXX	Translates CPU addresses.
Issues operand value to MMU, pulsing $\overline{\text{SPC}}$.	1101	Accepts operand value from bus; places it into referenced MMU register.
Waits for $\overline{\text{SDONE}}$ pulse from MMU.	0011	Sends completion signal by pulsing $\overline{\text{SDONE}}$ low.

TABLE 2-3. SMR Instruction Protocol

CPU Action	Status	MMU Action
Issues ID Byte and Operation Word, pulsing $\overline{\text{SPC}}$.	1111	Accepts and decodes instruction.
Accesses memory for effective address calculation or instruction prefetching.	XXXX	Translates CPU addresses.
Waits for $\overline{\text{SDONE}}$ pulse from MMU.	0011	Sends completion signal by pulsing $\overline{\text{SDONE}}$ low.
Reads results from MMU, pulsing $\overline{\text{SPC}}$.	1101	Presents data value from referenced MMU register on bus.

TABLE 2-4. RDVAL/WRVAL Instruction Protocol

CPU Action	Status	MMU Action
Performs effective address calculation and obtains address to be validated.	XXXX	Translates CPU addresses.
Issues ID Byte and operation word, pulsing $\overline{\text{SPC}}$.	1111	Accepts and decodes instruction.
CPU may prefetch instructions.	XXXX	
Performs dummy one-byte memory read from operand's location.	1010	Translates CPU address, using User-Mode mapping, and performs requested test on the address presented by the CPU. Aborts the CPU if there is no protection violation and the level-1 page table entry is invalid. Aborts on protection violations are temporarily suppressed.
Waits for $\overline{\text{SDONE}}$ pulse from MMU	XXXX	Pulses $\overline{\text{SDONE}}$ low for two clock cycles.
Sends $\overline{\text{SPC}}$ pulse and reads Status Word from MMU; places bit 5 of this word into the F bit of the PSR register.	1110	Presents Status Word on bus, indicating in bit 5 the result of the test.

2.0 Functional Description (Continued)

The CPU then reads a status word from the MMU. Bit 5 of this Status Word indicates the result of the instruction:

0 if the CPU in User Mode could have made the corresponding access to the operand at the specified address (Read in RDVAL, Write in WRVAL),

1 if the CPU would have been aborted for a protection violation.

Bit 5 of the Status Word is placed by the CPU into the F bit of the PSR register, where it can be tested by subsequent instructions as a condition code.

2.6 BUS ACCESS CONTROL

The NS32382 MMU has the capability of relinquishing its access to the bus upon request from a DMA device. It does this by using $\overline{\text{HOLD}}$, $\overline{\text{HLDAI}}$ and $\overline{\text{HLDAO}}$.

Details on the interconnections of these pins are provided in *Figure A-1* (Appendix A).

Requests for DMA are presented in parallel to both the CPU and MMU on the $\overline{\text{HOLD}}$ pin of each. The component that currently controls the bus then activates its Hold Acknowledge output to grant bus access to the requesting device. When the CPU grants the bus, the MMU passes the CPU's $\overline{\text{HLDA}}$ signal to its own $\overline{\text{HLDAO}}$ pin. When the MMU grants the bus, it does so by activating its $\overline{\text{HLDAO}}$ pin directly, and the CPU is not involved. $\overline{\text{HLDAI}}$ in this case is ignored.

Refer to *Figures 4-15* and *4-16* for details on bus granting sequences.

2.7 BREAKPOINTING

The MMU provides the ability to monitor references to memory locations in real time, generating a Breakpoint trap on occurrence of any type of reference made by a program to a specified virtual address or range of addresses.

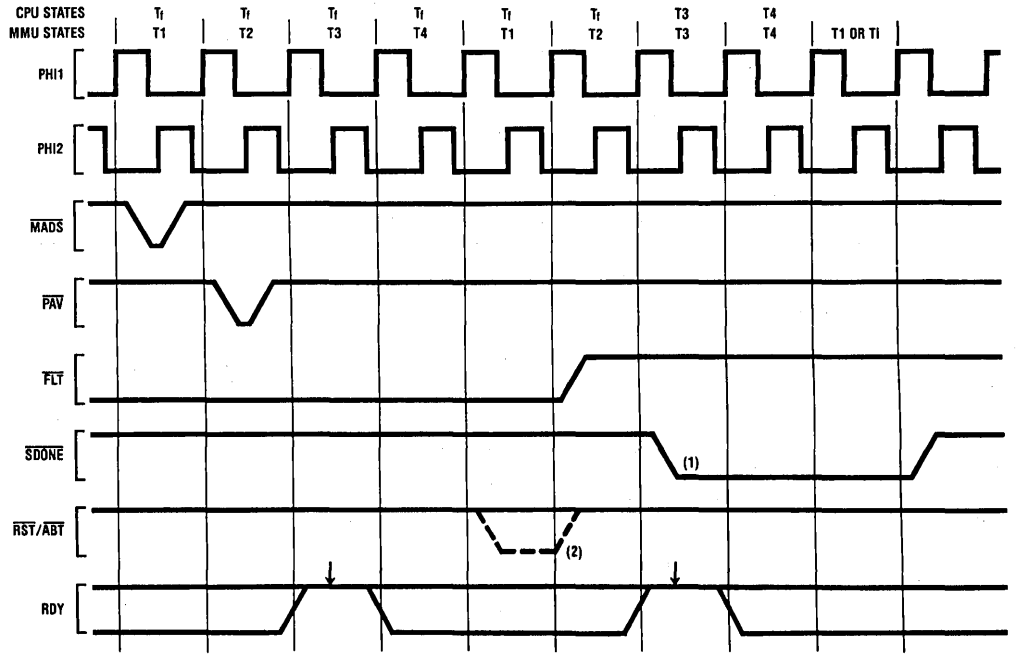
Breakpoint monitoring is enabled and regulated by the setting of appropriate bits in the BAR, BMR, BDR, MCR and MSR registers. See Sections 3.7 through 3.11.

The MMU compares the 32-bit address stored in the BAR register with the virtual address from the CPU. Selected bits can be masked off by the data pattern stored in the BMR register. Only those bit positions which are set in the BMR register will be used in the comparison process, bit positions which are cleared become "Don't Cares".

If a Breakpoint condition is detected, an abort will be issued to the CPU and the BP bit in the MSR register will be set. The virtual address that triggered the Breakpoint is then stored in the BDR register.

The dummy read addresses generated by the CPU during RDVAL/WRVAL operations, are not subject to Breakpoint address comparison. See Section 2.5.2.

When a Breakpoint is enabled, the NS32332 burst cycles should be inhibited by keeping the $\overline{\text{BIN}}$ signal high. The reason being that the CPU addresses are not incremented during burst. It is therefore possible for the CPU to skip over the address specified in the BAR register during burst cycle.



TL/EE/9142-15

Note 1: If there is a protection violation or an invalid Level-2 PTE then $\overline{\text{SDONE}}$ is issued two clock cycles earlier in T1.

Note 2: If there is no protection violation and the Level-1 PTE is not valid, an abort is generated and $\overline{\text{SDONE}}$ is not pulsed.

FIGURE 2-12. $\overline{\text{FLT}}$ Deassertion During RDVAL/WRVAL Execution

3.0 Architectural Description

3.1 PROGRAMMING MODEL

The MMU contains a set of registers through which the CPU controls and monitors management and debugging functions. These registers are not memory-mapped. They are examined and modified by executing the Slave Processor instructions LMR (Load Memory Management Register) and SMR (Store Memory Management Register). These instructions are explained in Section 3.14, along with the other Slave Processor instructions executed by the MMU.

A brief description of the MMU registers is provided below. Details on their formats and functions are provided in the following sections.

PTB0, PTB1—Page Table Base Registers. They hold the physical memory addresses of the LEVEL-1 Page Tables referenced by the MMU for address translation. See Section 3.3.

IVAR0, IVAR1—Invalidate Virtual Address Registers. These WRITE-ONLY registers are used to remove invalid Page Table Entries from the Translation Buffer.

TEAR—Translation Exception Address Registers. This register contains the virtual address which caused the translation exception.

BEAR—Bus Error Address Register. This register contains the virtual address which triggered the bus error.

BAR—Breakpoint Address Register. Used to hold a virtual address for breakpoint address comparison.

BMR—Breakpoint Mask Register. The contents of this register indicate which bit positions of the virtual address are to be compared.

BDR—Breakpoint Data Register. This register contains the virtual address that triggered a breakpoint.

MCR—Memory Management Control Register. Contains the control field for selecting the various features provided by the MMU.

MSR—Memory Management Status Register. Contains basic status fields for all MMU functions. See Section 3.11.

3.2 MEMORY MANAGEMENT FUNCTIONS

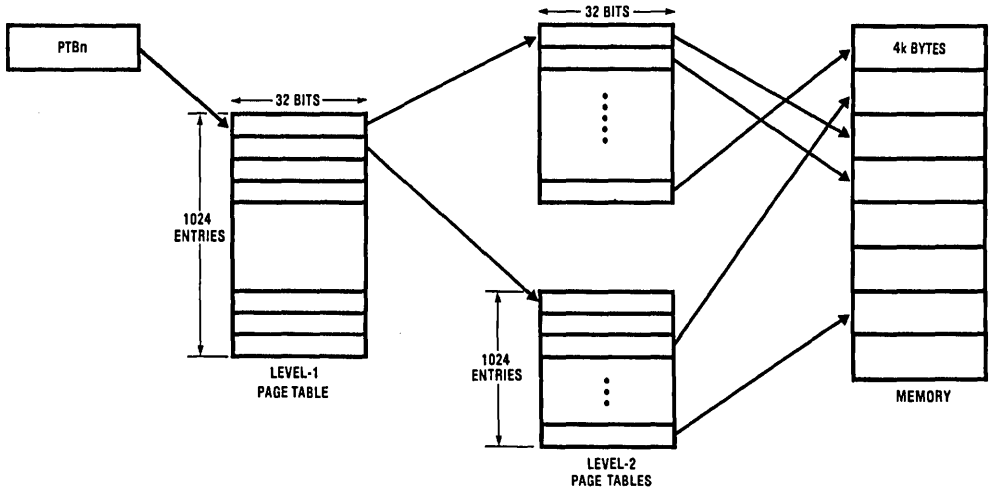
The NS32382 uses sets of tables in physical memory (the "Page Tables") to define the mapping from virtual to physical addresses. These tables are found by the MMU using one of its two Page Table Base registers: PTB0 or PTB1. Which register is used depends on the currently selected address space. See Section 3.2.2.

3.2.1. Page Tables Structure

The page tables are arranged in a two-level structure, as shown in *Figure 3-1*. Each of the MMU's PTBn registers may point to a Level-1 page table. Each entry of the Level-1 page table may in turn point to a Level-2 page table. Each Level-2 page table entry contains translation information for one page of the virtual space.

The Level-1 page table must remain in physical memory while the PTBn register contains its address and translation is enabled. Level-2 Page Tables need not reside in physical memory permanently, but may be swapped into physical memory on demand as is done with the pages of the virtual space.

The Level-1 Page Table contains 1024 32-bit Page Table Entries (PTE's) and therefore occupies 4 Kbyte. Each entry of the Level-1 Page Table contains fields used to construct the physical base address of a Level-2 Page Table. These fields are a 20-bit PFN field, providing bits 12-31 of the physical address. The remaining bits (0-11) are assumed zero, placing a Level-2 Page Table always on a 4 Kbyte (page) boundary.



TL/EE/9142-18

FIGURE 3-1. Two-Level Page Tables

3.0 Architectural Description (Continued)

Level-2 Page Tables contain 1024 32-bit Page Table entries, and so occupy 4 Kbytes (1 page). Each Level-2 Page Table Entry points to a final 4 Kbyte physical page frame. In other words, its PFN provides the Page Frame Number portion (bits 12-31) of the translated address *(Figure 3-3)*. The OFFSET field of the translated address is taken directly from the corresponding field of the virtual address.

3.2.2 Virtual Address Spaces

When the Dual Space option is selected for address translation in the MCR (Sec. 3.10) the MMU uses two maps: one for translating addresses presented to it in Supervisor Mode and another for User Mode addresses. Each map is referenced by the MMU using one of the two Page Table Base registers: PTB0 or PTB1. The MMU determines the CPU's current mode by monitoring the state of the U/$\overline{S}$ pin and applying the following rules.

1) While the CPU is in Supervisor Mode (U/$\overline{S}$ pin = 0), the CPU is said to be presenting addresses belonging to Address Space 0, and the MMU uses the PTB0 register as its reference for looking up translations from memory.
2) While the CPU is in User Mode (U/$\overline{S}$ pin = 1), and the MCR DS bit is set to enable Dual Space translation, the CPU is said to be presenting addresses belonging to Address Space 1, and the MMU uses the PTB1 register to look up translations.
3) If Dual Space translation is not selected in the MCR, there is no Address Space 1, and all addresses presented in both Supervisor and User modes are considered by the MMU to be in Address Space 0. The privilege level of the CPU is used then only for access level checking.

Note: When the CPU executes a Dual-Space Move instruction (MOVUSi or MOVSUi), it temporarily enters User Mode by switching the state of the U/$\overline{S}$ pin. Accesses made by the CPU during this time are treated by the MMU as User-Mode accesses for both mapping and access level checking. It is possible, however, to force the MMU to assume Supervisor-Mode privilege on such accesses by setting the Access Override (AO) bit in the MCR (Sec. 3.10).

3.2.3 Page Table Entry Formats

Figure 3-2 shows the formats of Level-1 and Level-2 Page Table Entries (PTE's).

The bits are defined as follows:

V Valid. The V bit is set and cleared only by software.
- V=1 => The PTE is valid and may be used for translation by the MMU.
- V=0 => The PTE does not represent a valid translation. Any attempt to use this PTE will cause the MMU to generate an Abort trap.

PL Protection Level. This two-bit field establishes the types of accesses permitted for the page in both User Mode and Supervisor Mode, as shown in Table 3-1.

The PL field is modified only by software. In a Level-1 PTE, it limits the maximum access level allowed for all pages mapped through that PTE.

TABLE 3-1. Access Protection Levels

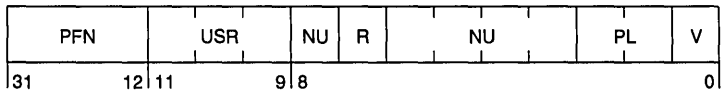
Mode	U/$\overline{S}$	Protection Level Bits (PL)			
		00	01	10	11
User	1	no access	no access	read only	full access
Supervisor	0	read only	full access	full access	full access

NU Not Used. These bits are reserved by National for future enhancements. Their values should be set to zero.

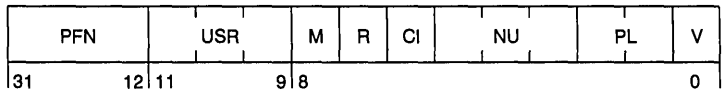
CI Cache Inhibit. This bit appears only in Level-2 PTE's. It is used to specify non-cacheable pages.

R Referenced. This is a status bit, set by the MMU and cleared by the operating system, that indicates whether the page mapped by this PTE has been referenced within a period of time determined by the operating system. It is intended to assist in implementing memory allocation strategies. In a Level-1 PTE, the R bit indicates only that the Level-2 Page Table has been referenced for a translation, without necessarily implying that the translation was successful. In a Level-2 PTE, it indicates that the page mapped by the PTE has been successfully referenced.
- R=1 => The page has been referenced since the R bit was last cleared.
- R=0 => The page has not been referenced since the R bit was last cleared.

M Modified. This is a status bit, set by the MMU whenever a write cycle is successfully performed to the page mapped by this PTE. It is initialized to zero by the operating system when the page is brought into physical memory.



First Level PTE



Second Level PTE

FIGURE 3-2. Page Table Entries (PTE's)

3.0 Architectural Description (Continued)

M=1=> The page has been modified since it was last brought into physical memory.

M=0=> The page has not been modified since it was last brought into physical memory.

In Level-1 Page Table Entries, this bit position is undefined, and is unaltered.

USR User bits. These bits are ignored by the MMU and their values are not changed.

They can be used by the user software.

PFN Page Frame Number. This 20-bit field provides bits 12-31 of the physical address. See *Figure 3-3.*

3.2.4 Physical Address Generation

When a virtual address is presented to the MMU by the CPU and the translation information is not in the TLB, the MMU performs a page table lookup in order to generate the physical address.

The Page Table structure is traversed by the MMU using fields taken from the virtual address. This sequence is diagrammed in *Figure 3-3.*

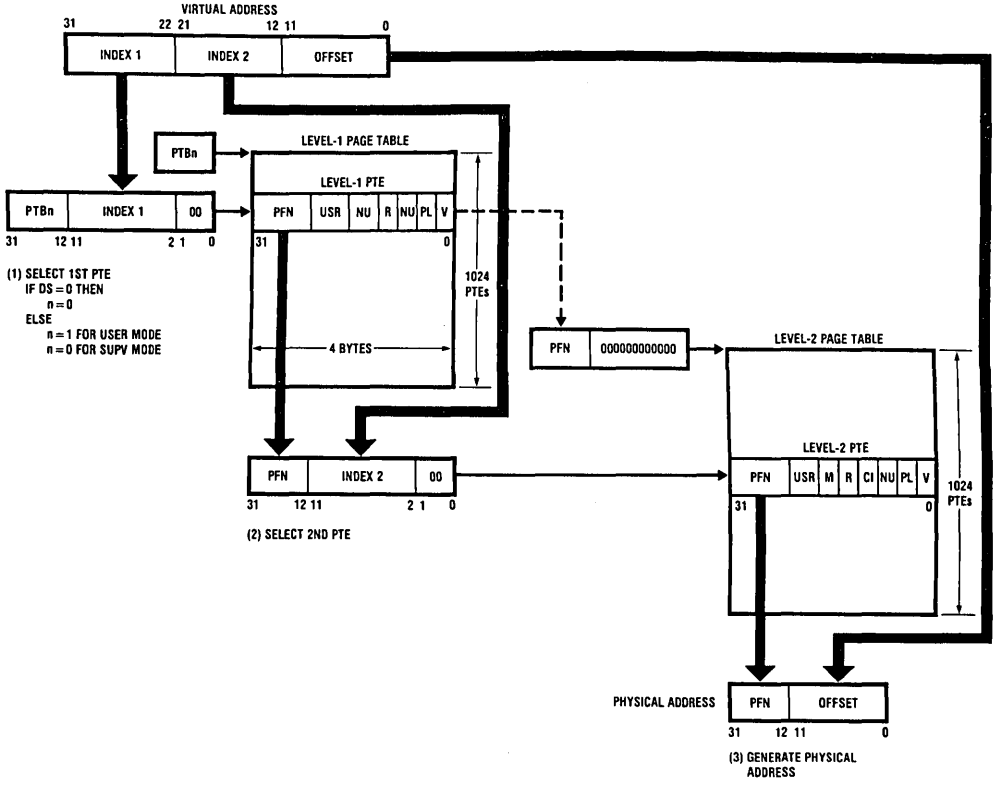
Bits 12–31 of the virtual address hold the 20-bit Page Number, which in the course of the translation is replaced with the 20-bit Page Frame Number of the physical address. The virtual Page Number field is further divided into two fields, INDEX 1 and INDEX 2.

Bits 0–11 constitute the OFFSET field, which identifies a byte's position within the accessed page. Since the byte position within a page does not change with translation, this value is not used, and is simply echoed by the MMU as bits 0–11 of the final physical address.

The 10-bit INDEX 1 field of the virtual address is used as an index into the Level-1 Page Table, selecting one of its 1024 entries. The address of the entry is computed by adding INDEX 1 (scaled by 4) to the contents of the current Page Table Base register. The PFN field of that entry gives the base address of the selected Level-2 Page Table.

The INDEX 2 field of the virtual address (10 bits) is used as the index into the Level-2 Page Table, by adding it (scaled by 4) to the base address taken from the Level-1 Page Table Entry. The PFN field of the selected entry provides the entire Page Frame Number of the translated address.

The offset field of the virtual address is then appended to this frame number to generate the final physical address.



TL/EE/9142–20

FIGURE 3-3. Virtual to Physical Address Translation

3.0 Architectural Description (Continued)

3.3 PAGE TABLE BASE REGISTERS (PTB0, PTB1)

The PTBn registers hold the physical addresses of the Level-1 Page Tables.

The format of these registers is shown in *Figure 3-4.* The least-significant 12 bits are permanently zero, so that each register always points to a 4 Kbyte boundary in memory.

The PTBn registers may be loaded or stored using the MMU Slave Processor instructions LMR and SMR (Section 3.14).

3.4 INVALIDATE VIRTUAL ADDRESS REGISTERS (IVAR0, IVAR1)

The Invalidate Virtual Address registers are write-only registers. When a virtual address is written to IVAR0 or IVAR1 using the LMR instruction, the translation for that virtual address is purged, if present, from the TLB. This must be done whenever a Page Table Entry has been changed in memory, since the TLB might otherwise contain an incorrect translation value.

Another technique for purging TLB entries is to load a PTBn register. This automatically purges all entries associated with the addressing space mapped by that register. Turning off translation (clearing the MCR TU and/or TS bits) does not purge any entries from the TLB.

The format of the $IVAR_n$ registers is shown in *Figure 3-5.*

3.5 TRANSLATION EXCEPTION ADDRESS REGISTER (TEAR)

The TEAR Register is loaded when a translation exception occurs. It contains the 32-bit virtual address which caused the translation exception and is a read-only register. TEAR has the same format as the IVARn registers of *Figure 3-5.*

For more details on the updating of TEAR, refer to the note at the end of Section 3.11.

3.6 BUS ERROR ADDRESS REGISTER (BEAR)

The BEAR Register is loaded when a CPU or MMU bus error occurs. It contains the 32-bit virtual address which triggered the bus error and is a read-only register. BEAR has the same format as the IVARn registers of *Figure 3-5.*

3.7 BREAKPOINT ADDRESS REGISTER (BAR)

The Breakpoint Address Register is used to hold a virtual address for breakpoint address comparison during instruction and operand accesses. It is 32 bits in length and its format is shown in *Figure 3-6.*

3.8 BREAKPOINT MASK REGISTER (BMR)

The Breakpoint Mask Register provides corresponding bit positions for each of the virtual address bits that are to be compared when the Breakpoint Address Compare Function is enabled. Bits which are set in this register are used for matching virtual address bits while bits which are cleared are treated as "don't cares". This allows a breakpoint to be generated upon an access to any location within a block of addresses. The BMR Register format is shown in *Figure 3-6.*

3.9 BREAKPOINT DATA REGISTER (BDR)

The Breakpoint Data Register holds the virtual address that triggered the breakpoint.

It is a read-only register and its format is shown in *Figure 3-6.*

3.10 MEMORY MANAGEMENT CONTROL REGISTER (MCR)

The MCR Register controls the various features provided by the MMU. It is 32 bits in length and has the format shown in *Figure 3-7.* All bits will be cleared on reset. The bits 8 to 31 are RESERVED for future use and must be loaded with zeros.

When MCR is read as a 32-bit word, bits 8 to 31 will be returned as zeros. Details on the MCR bits are given below.

TU Translate User-Mode Addresses. While this bit is "1", the MMU translates all addresses presented while the CPU is in User Mode. While it is "0", the MMU echoes all User-Mode virtual addresses without performing translation or access level checking.

Note: Altering the TU bit has no effect on the contents of the TLB.

TS Translate Supervisor-Mode Addresses. While this bit is "1", the MMU translates all addresses presented while the CPU is in Supervisor Mode. While it is "0", the MMU echoes all Supervisor-Mode virtual addresses without translation or access level checking.

Note: Altering the TS bit has no effect on the contents of the TLB.

DS Dual-Space Translation. While this bit is "1", Supervisor Mode addresses and User Mode addresses are translated independently of each other, using separate mappings. While it is "0", both Supervisor Mode addresses and User Mode addresses are translated using the same mapping. See Section 3.2.2.

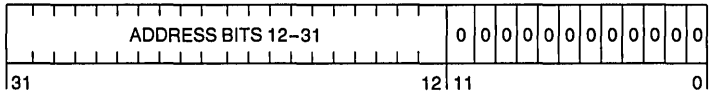


FIGURE 3-4. Page Table Base Registers (PTB0, PTB1)

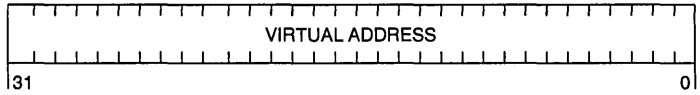


FIGURE 3-5. Address Registers (IVAR0, IVAR1, TEAR, BEAR)

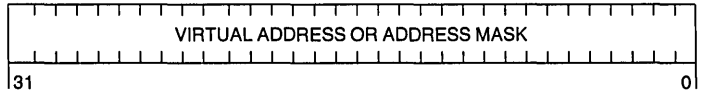


FIGURE 3-6. Breakpoint Registers (BAR, BMR, BDR)

3.0 Architectural Description (Continued)

AO Access Level Override. This bit may be set to temporarily cause User Mode accesses to be given Supervisor Mode privilege. See Section 3.13.

BR Break on Read. If BR is 1, a break is generated when data is read from the breakpoint address. Instruction fetches do not trigger a Read breakpoint. If BR is 0, this condition is disabled.

BW Break on Write. If BW is 1, a break is generated when data is written to the breakpoint address or when data is read from the breakpoint address as the first part of a read-modify-write access. If BW is 0, this condition is disabled.

BE Break on Execution. If BE is 1, a break is generated when the instruction at the breakpoint address is fetched. If BE is 0, this condition is disabled.

BAS Breakpoint Address Space. This bit selects the address space for breakpointing.

BAS = 0 Selects Address Space 0 (PTB0).

BAS = 1 Selects Address Space 1 (PTB1).

3.11 MEMORY MANAGEMENT STATUS REGISTER (MSR)

The Memory Management Status Register provides status information for translation exceptions as well as bus errors.

When either a translation exception or a bus error occurs, the corresponding bits in the MSR are updated.

The MSR register can be loaded with an LMR instruction. Its format is shown in *Figure 3-8*. Bits 19 through 31 are reserved for future use and are returned as zeros when read. Bits 8 and 18 are also reserved.

Upon reset, all MSR bits are cleared to zero. Details on the function of each bit are given below.

TEX Translation Exception. This 2-bit field specifies the cause of the current address translation exception. Combinations appearing in this field are summarized below.

00 No Translation Exception

01 First Level PTE Invalid

10 Second Level PTE Invalid

11 Protection Violation

Note: During address translation, if a protection violation and an invalid PTE are detected at the same time, the TEX field is set to indicate a protection violation.

DDT Data Direction. This bit indicates the direction of the transfer that the CPU was attempting when the translation exception occurred.

DDT = 0 = > Read Cycle

DDT = 1 = > Write Cycle

UST User/Supervisor. This is the state of the U/$\overline{S}$ pin from the CPU during the access cycle that triggered the translation exception.

STT CPU Status. This 4-bit field is set on an address translation exception to the value of the CPU Status Bus (ST0–ST3).

BP Break. This bit is set to indicate that a breakpoint condition has been detected by the MMU.

CE CPU Error. This bit is set when a bus error occurs while the CPU is in control of the bus.

ME MMU Error. This bit is set when a bus error occurs while the MMU is in control of the bus.

DDE Data Direction. This bit indicates the direction of the transfer that the CPU was attempting when the bus error occurred.

DDE = 0 = > Read Cycle

DDE = 1 = > Write Cycle

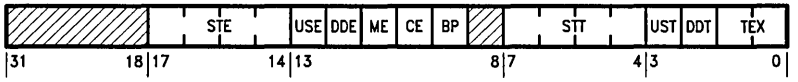
USE User/Supervisor. This is the state of the U/$\overline{S}$ pin from the CPU during the access cycle that triggered the bus error.

STE CPU Status. This 4-bit field is set to the value of the CPU status bus (ST0–ST3) when a bus error is detected.

Note: The MSR and TEAR registers are updated whenever a translation exception occurs, regardless of whether a CPU abort will result. As a consequence, after an abort is recognized, MSR and TEAR may be overwritten with new data and thus the original contents may be lost. This happens if the CPU, while executing the abort routine, performs instruction prefetch cycles from an invalid page. To ensure correct operation the reading of MSR and TEAR should be performed before any instruction prefetch crosses a page boundary, unless the next page is valid. This may place some restrictions in the relocation of the abort routine.

TL/EE/9142–24

FIGURE 3-7. Memory Management Control Register (MCR)



TL/EE/9142–25

FIGURE 3-8. Memory Management Status Register (MSR)

3.0 Architectural Description (Continued)

3.12 TRANSLATION LOOKASIDE BUFFER (TLB)

The Translation Lookaside Buffer is an on-chip fully associative memory. It provides direct virtual to physical mapping for the 32 most recently used pages, requiring only one clock period to perform the address translation.

The efficiency of the MMU is greatly increased by the TLB, which bypasses the much longer Page Table lookup in over 97% of the accesses made by the CPU.

Entries in the TLB are allocated and replaced by the MMU itself; the operating system is not involved. The TLB entries cannot be read or written by software; however, they can be purged from it under program control.

Figure 3-9 models the TLB. Information is placed into the TLB whenever the MMU performs a lookup from the Page Tables in memory. If the retrieved mapping is valid (V=1 in both levels of the Page Tables), and the access attempted is permitted by the protection level, an entry of the TLB is loaded from the information retrieved from memory. The recipient entry is selected by an on-chip circuit that implements a Least-Recently-Used (LRU) algorithm. The MMU places the virtual page number (20 bits) and the Address Space qualifier bit into the Tag field of the TLB entry.

The Value portion of the entry is loaded from the Page Tables as follows:

The Translation field (20 bits) is loaded from the PFN field of the Level-2 Page Table Entry.

The CI and M bits are loaded from the Level-2 Page Table Entry.

The PL field (2 bits) is loaded to reflect the net protection level imposed by the PL fields of the Level-1 and Level-2 Page Table Entries.

(Not shown in the figure are additional bits associated with each TLB entry which flag it as full or empty, and which select it as the recipient when a Page Table lookup is performed.)

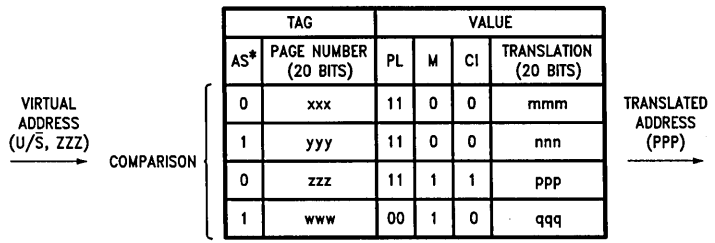
When a virtual address is presented to the MMU for translation, the high-order 20 bits (page number) and the Address Space qualifier are compared associatively to the corresponding fields in all entries of the TLB. When the Tag portion of a TLB entry completely matches the input values, the Value portion is produced as output. If the protection level is not violated, and the M bit does not need to be changed, then the physical address Page Frame number is output in the next clock cycle. If the protection level is violated, the MMU instead activates the Abort output. If no TLB entry matches, or if the matching entry's M bit needs to be changed, the MMU performs a page-table lookup from memory.

Note that for a translation to be loaded into the TLB it is necessary that the Level-1 and Level-2 Page Table Entries be valid (V bit = 1). Also, it is guaranteed that in the process of loading a TLB entry (during a Page Table lookup) the Level-1 and Level-2 R bits will be set in memory if they were not already set. For these reasons, there is no need to replicate either the V bit or the R bit in the TLB entries.

Whenever a Page Table Entry in memory is altered by software, it is necessary to purge any matching entry from the TLB, otherwise the MMU would be translating the corresponding addresses according to obsolete information. TLB entries may be selectively purged by writing a virtual address to one of the IVARn registers using the LMR instruction. The TLB entry (if any) that matches that virtual address is then purged, and its space is made available for another translation. Purging is also performed by the MMU whenever an address space is remapped by altering the contents of the PTB0 or PTB1 register. When this is done, the MMU purges all the TLB entries corresponding to the address space mapped by that register. Turning translation on or off (via the MCR TU and TS bits) does not affect the contents of the TLB.

3.13 ADDRESS TRANSLATION ALGORITHM

The MMU either translates the 32-bit virtual address to a 32-bit physical address or reports a translation error. This process is described algorithmically in the following pages. See also *Figure 3-3*.



TL/EE/9142-26

FIGURE 3-9. TLB Model

*AS represents the virtual address space qualifier.

MMU Page Table Lookup and Access Validation Algorithm

```
Legend:
x = y           x is assigned the value y
x == y          Comparison expression, true if x is equal to y
x AND y         Boolean AND expression, true only if assertions x and y are both true
x OR y          Boolean inclusive OR expression, true if either of assertions x and y is true
;               Delimiter marking end of statement
{ . . . }       Delimiters enclosing a statement block
item(i)         Bit number i of structure "item"
item(i:j)       The field from bit number i through bit number j of structure "item"
item.x          The bit or field named "x" in structure "item"
DONE            Successful end of translation; MMU provides translated address
ABORT           Unsuccessful end of translation; MMU aborts CPU access
This algorithm represents for all cases a valid definition of address translation.
Bus activity implied here occurs only if the TLB does not contain the mapping,
or if the reference requires that the MMU alter the M bit of the Page Table Entry.
Otherwise, the MMU provides the translated address in one clock period.
Input (from CPU):
     U (1 if U/S is high)
     W (1 if DDIN input is high)
     VA  Virtual address consisting of:
        INDEX_1 (from pins A31-A22)
        INDEX_2 (from pins A21-A12)
        OFFSET  (from pins A11-A0)
     ACCESS_LEVEL       The access level of a reference is a 2-bit value synthesized by the MMU from CPU status:
        bit 1 = U AND NOT MCR.A0  (U from U/S input pin)
        bit 0 = 1 for Write cycle, or Read cycle of an "rmw" class operand access
                0 otherwise.

Output:
     PA Physical Address on pins PA0-PA31;
     CI Cache Inhibit Signal
     Abort pulse on RST/ABT pin.
Uses:
     MCR        Control Register:
                        fields TU, TS and DS
```

MMU Page Table Lookup and Access Validation Algorithm (Continued)

```
     PTB0       Page Table Base Register 0
     PTB1       Page Table Base Register 1
     PTE_1      Level-1 Page Table Entry:
                       fields PFN, PL, V and R
     PTEP_1     Pointer, holding address of PTE_1
     PTE_2      Level-2 Page Table Entry:
                       fields PFN, PL, V, M, R and CI
     PTEP_2     Pointer, holding address of PTE_2
IF ( (MCR.TU == 0) AND (U == 1) )  OR  ( (MCR.TS == 0) AND (U ==0) )  If translation not enabled then echo
     THEN  {  PA(0:31) = VA(0:31) ; CINH PIN = 0 ; DONE  }  ;              virtual address as physical address.
     ,

IF (MCR.DS == 1) AND (U == 1)                                        If Dual Space mode and CPU in User Mode
   THEN {  PTEP_1(31:12) = PTB1(31:12) ;                                  then form Level-1 PTE address
           PTEP_1(11:2) = VA.INDEX_1 ; PTEP_1(1:0)=0  }                         from PTB1 register,
   ELSE {  PTEP_1(31:12) = PTB0(31:12) ;                                  else form Level-1 PTE address
           PTEP_1(11:2) = VA.INDEX_1; PTEP_1(1:0) = 0                           from PTB0 register.
        } ;

                                              - - - LEVEL 1 PAGE TABLE LOOKUP - - -

IF ( ACCESS_LEVEL > PTE_1.PL ) OR (PTE_1.V == 0)                     If protection violation or invalid Level-2 page
   THEN ABORT ;                                                           table then abort the access.

IF PTE_1.R == 0 THEN PTE_1.R = 1 ;                                   Otherwise, set Reference bit if not already set,

PTEP_2(31:11) = PTE_1.PFN ;                                                    and form Level-2 PTE address.
PTEP_2(11:2) = VA.INDEX_2 ; PTEP_2(1:0) = 0 ;

                                              - - - LEVEL 2 PAGE TABLE LOOKUP - - -

IF ( ACCESS_LEVEL > PTE_2.PL ) OR ( PTE_2.V ==0 )                        If protection violation or invalid page
   THEN ABORT ;                                                          then abort the access.

IF PTE_2.R == 0 THEN PTE_2.R == 1 ;                                  Otherwise, set Referenced bit if not already set,
IF ( W == 1) AND ( PTE_2.M == 0 ) THEN PTE_2.M = 1 ;                      if Write cycle set Modified bit if not
                                                                               already set,
PA(31:11) = PTE_2.PFN ; PA(11:0) = VA.OFFSET ; CINH = PTE_2.CI ;                  and generate physical address.
DONE ;
```

3.0 Architectural Description (Continued)

3.14 INSTRUCTION SET

Four instructions of the Series 32000 instruction set are executed cooperatively by the CPU and MMU. These are:

LMR	Load Memory Management Register
SMR	Store Memory Management Register
RDVAL	Validate Address for Reading
WRVAL	Validate Address for Writing

The format of the MMU slave instructions is shown in *Figure 3-10*. Table 3-2 shows the encodings of the "short" field for selecting the various MMU internal registers.

TABLE 3-2. "Short" Field Encodings

"Short" Field	Register
0000	BAR
0001	RESERVED
0010	BMR
0011	BDR
0110	BEAR
1001	MCR
1010	MSR
1011	TEAR
1100	PTB0
1101	PTB1
1110	IVAR0
1111	IVAR1

Note: All other codes are illegal. They will cause unpredictable registers to be selected if used in an instruction.

For reasons of system security, all MMU instructions are privileged, and the CPU does not issue them to the MMU in User Mode. Any such attempt made by a User-Mode program generates the Illegal Operation trap, Trap (ILL). In addition, the CPU will not issue MMU instructions unless its CFG register's M bit has been set to validate the MMU instruction set. If this has not been done, MMU instructions are not recognized by the CPU, and an Undefined Instruction trap, Trap (UND), results.

The LMR and SMR instructions load and store MMU registers as 32-bit quantities to and from any general operand (including CPU General-Purpose Registers).

The RDVAL and WRVAL instructions probe a memory address and determine whether its current protection level would allow reading or writing, respectively, if the CPU were in User Mode. Instead of triggering an Abort trap, these instructions have the effect of setting the CPU PSR F bit if the type of access being tested for would be illegal. The PSR F bit can then be tested as a condition code.

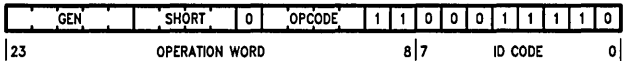
Note: The Series 32000 Dual-Space Move instructions (MOVSUi and MOVUSi), although they involve memory management action, are not Slave Processor instructions. The CPU implements them by switching the state of its U/$\overline{S}$ pin at appropriate times to select the desired mapping and protection from the MMU.

For full architectural details of these instructions, see the Series 32000 Instruction Set Reference Manual.

4.0 Device Specifications

4.1 NS32382 PIN DESCRIPTIONS

The following is a brief description of all NS32382 pins. The descriptions reference portions of the Functional Description, Section 2.0.



TL/EE/9142–27

FIGURE 3-10. MMU Slave Instruction Format

4.0 Device Specifications (Continued)

4.1.1 Supplies

Power (V_{CC}): Eight pins, connected to the +5V supply.

Back Bias Generator (BBG): Output of on-chip substrate voltage generator.

Ground (GND): Eighteen pins, connected to ground.

4.1.2 Input Signals

Clocks (PHI1, PHI2): Two-phase clocking signals. Section 2.2.

Ready (RDY): Active high. Used by slow memories to extend MMU originated memory cycles. Section 2.4.4.

Hold Request ($\overline{HOLD}$): Active low. Causes a release of the bus for DMA or multiprocessing purposes. Section 2.6.

Hold Acknowledge In ($\overline{HLDAI}$): Active low. Applied by the CPU in response to $\overline{HOLD}$ input, indicating that the CPU has released the bus for DMA or multiprocessing purposes. Section 2.6.

Reset Input ($\overline{RSTI}$): Active low. System reset. Section 2.3.

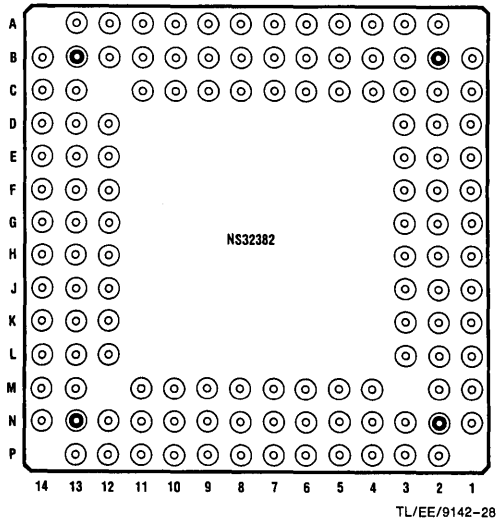
Status Lines (ST0–ST3): Status code input from the CPU. Active from T4 of previous bus cycle through T3 of current bus cycle. Section 2.4.

User/Supervisor Mode (U/$\overline{S}$): This signal is provided by the CPU. It is used by the MMU for protection and for selecting the address space (in dual address space mode only). Section 2.4.

Address Strobe Input ($\overline{ADS}$): Active low. Pulse indicating that a virtual address is present on the bus.

Bus Error ($\overline{BER}$): Active low. When active, indicates that an error occurred during a bus cycle. Not applicable for slave cycles.

Connection Diagram



Bottom View

FIGURE 4-1. Pin Grid Array Package

Order Number NS32382U-10 or NS32382U-15
See NS Package Number U125A

NS32382 Pinout Descriptions
125 Pin Grid Array

Desc	Pin	Desc	Pin	Desc	Pin	Desc	Pin
NC	A2	V_{CC}	C7	AD22	H1	PA4	M9
$\overline{SPC}$	A3	GND	C8	AD21	H2	PA7	M10
NC	A4	V_{CC}	C9	AD20	H3	GND	M11
$\overline{SDONE}$	A5	V_{CC}	C10	GND	H12	V_{CC}	M13
$\overline{MILO}$	A6	GND	C11	PA22	H13	PA13	M14
$\overline{HLDAI}$	A7	GND	C13	PA21	H14	NC	N1
$\overline{RSTI}$	A8	CINH	C14	AD19	J1	GND	N2
$\overline{BER}$	A9	AD29	D1	AD18	J2	GND	N3
$\overline{BRT}$	A10	AD31	D2	AD17	J3	AD9	N4
$\overline{RST}/\overline{ABT}$	A11	GND	D3	PA20	J12	AD5	N5
ST0	A12	$\overline{ADS}$	D12	PA19	J13	AD2	N6
ST1	A13	RESERVED	D13	PA18	J14	AD0	N7
NC	B1	PA31	D14	AD14	K1	PA0	N8
NC	B2	AD27	E1	AD15	K2	PA3	N9
GND	B3	AD30	E2	AD16	K3	PA6	N10
GND	B4	U/$\overline{S}$	E3	GND	K12	PA9	N11
V_{CC}	B5	PA30	E12	PA17	K13	GND	N12
$\overline{HOLD}$	B6	PA29	E13	PA16	K14	NC	N13
RDY	B7	PA28	E14	AD13	L1	PA12	N14
PHI2	B8	AD25	F1	AD12	L2	AD11	P2
PHI1	B9	AD26	F2	V_{CC}	L3	AD10	P3
$\overline{PAV}$	B10	AD28	F3	V_{CC}	L12	AD8	P4
$\overline{FLT}$	B11	PA27	F12	PA14	L13	AD6	P5
ST2	B12	PA26	F13	PA15	L14	AD4	P6
ST3	B13	PA25	F14	NC	MI	AD1	P7
RESERVED	B14	AD23	G1	GND	M2	PA1	P8
NC	C1	AD24	G2	GND	M4	PA2	P9
$\overline{MADS}$	C2	GND	G3	AD7	M5	PA5	P10
GND	C3	GND	G12	AD3	M6	PA8	P11
GND	C4	PA24	G13	V_{CC}	M7	PA10	P12
$\overline{DDIN}$	C5	PA23	G14	BBG	M8	PA11	P13
$\overline{HLDAO}$	C6						

4.0 Device Specifications (Continued)

Bus Retry ($\overline{BRT}$): Active low. When active, the MMU will re-execute the last bus cycle. Not applicable for slave cycles.

Slave Processor Control ($\overline{SPC}$): Active low. Used as a data strobe for slave processor transfers.

4.1.3 Output Signals

Reset Output/Abort ($\overline{RST}/\overline{ABT}$): Active Low. Held active longer than one clock cycle to reset the CPU. Pulsed low during T2 to abort the current CPU instruction.

Float Output ($\overline{FLT}$): Active low. Floats the CPU from the bus when the MMU accesses page table entries. Section 2.4.3.

Physical Address Valid ($\overline{PAV}$): Active low. Pulse generated during T2 indicating that a physical address is present on the bus.

Hold Acknowledge Output ($\overline{HLDAO}$): Active low. When active, indicates that the bus has been released.

Cache Inhibit (CINH): This output signal reflects the state of the CI bit in the second level Page Table Entry (PTE). It is used to specify non-cacheable pages. During MMU generated bus cycles and when the MMU is in No-Translation mode, CINH will be held low.

Slave Done ($\overline{SDONE}$): Active low. Used by the MMU to inform the CPU of the completion of a slave instruction. It floats when it is not active.

MMU Address Strobe ($\overline{MADS}$): Active low. This signal is asserted in T1 of an MMU initiated cycle. It indicates that the physical address is available on the physical address bus. $\overline{MADS}$ is floated during hold acknowledge.

MMU Interlock ($\overline{MILO}$): Active low. This signal is asserted by the MMU when it performs a read-modify-write operation to up-date the R and/or the M bit in the Page Table Entry (PTE). It is inactive during Hold Acknowledge.

Physical Address Bus (PA0–PA31): These 32 signal lines carry the physical address. They float during Hold Acknowledge.

4.1.4 Input-Output Signals

Data Direction In ($\overline{DDIN}$): Active low. Status signal indicating direction of data transfer during a bus cycle. Driven by the MMU during a page-table lookup.

Address/Data 0–31 (AD0–AD31): Multiplexed Address/Data Information. Bit 0 is the least significant bit.

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	2.5W

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_A = 0 to +70°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		V_{CC} + 0.5	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{CH}	High Level Clock Voltage	PHI1, PHI2 Pins Only	V_{CC} − 0.5		V_{CC} + 0.5	V
V_{CL}	Low Level Clock Voltage	PHI1, PHI2 Pins Only	−0.5		0.3	V
V_{CRT}	Clock Input Ringing Tolerance	PHI1, PHI2 Pins Only	−0.5		0.5	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low Level Output Voltage	I_{OL} = 2 mA			0.45	V
I_{ILS}	$\overline{SPC}$ Input Current (Low)	V_{IN} = 0.4V, $\overline{SPC}$ in Input Mode	0.05		1.0	mA
I_I	Input Load Current	0 ≤ V_{IN} ≤ V_{CC}, All Inputs Except PHI1, PHI2, $\overline{AT}/\overline{SPC}$	−20		20	μA
I_L	Leakage Current (Output and I/O Pins in TRI-STATE/Input Mode)	0.4 ≤ V_{OUT} ≤ V_{CC}	−20		20	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C		350	500	mA

4.0 Device Specifications (Continued)

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 2.0V on the rising or falling edges of the clock phases PHI1 and PHI2, and 0.8V or 2.0V on all other signals as illustrated in *Figures 4-2* and *4-3*, unless specifically stated otherwise.

ABBREVIATIONS:

L.E. — leading edge R.E. — rising edge
T.E. — trailing edge F.E. — falling edge

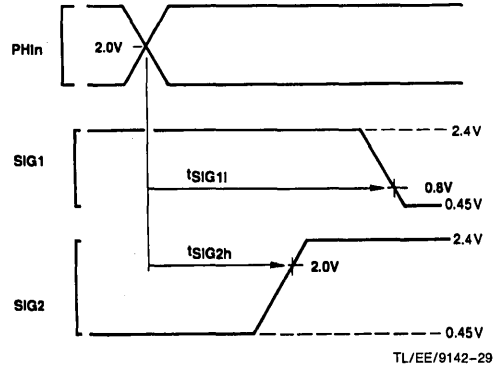


FIGURE 4-2. Timing Specification Standard (Signal Valid after Clock Edge)

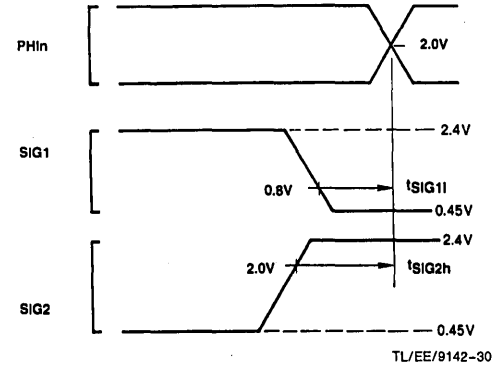


FIGURE 4-3. Timing Specification Standard (Signal Valid before Clock Edge)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32382-10, NS32382-15.
Maximum times assume capacitive loading of 50 pF.

Name	Figure	Description	Reference/Conditions	NS32382-10		NS32382-15		Units
				Min	Max	Min	Max	
t_{PALv}	4-4	PA0-11 Valid ($\overline{FLT}$ = 1)	After R.E., PHI1 T1		75		50	ns
t_{PAHv}	4-4	PA12-31 Valid ($\overline{FLT}$ = 1)	After R.E., PHI1 T2		30		20	ns
t_{PAVa}	4-4	$\overline{PAV}$ Signal Active	After R.E., PHI1 T2		25		17	ns
t_{PAVia}	4-4	$\overline{PAV}$ Signal Inactive	After R.E., PHI2 T2		40		27	ns
t_{PAVw}	4-4	$\overline{PAV}$ Pulse Width	At 0.8V (Both Edges)	35		22		ns
t_{PALh}	4-4	PA0-11 Hold ($\overline{FLT}$ = 1)	After R.E., PHI1 (Next) T1	0		0		ns
t_{PAHh}	4-4	PA12-31 Hold ($\overline{FLT}$ = 1)	After R.E., PHI1 (Next) T2	0		0		ns
t_{CIv}	4-4, 4-15,	CINH Signal Valid ($\overline{FLT}$ = 1) ($\overline{FLT}$ = 0)	After R.E., PHI1 T2 After R.E., PHI1 T1		40		27	ns
t_{CIh}	4-4	CINH Signal Hold ($\overline{FLT}$ = 1)	After R.E., PHI1 (Next) T2	0		0		
t_{DDINv}	4-5, 4-7, 4-15	$\overline{DDIN}$ Signal Valid ($\overline{FLT}$ = 0)	After R.E., PHI1 T1		35		25	ns
t_{DDINh}	4-5	$\overline{DDIN}$ Signal Hold ($\overline{FLT}$ = 0)	After R.E., PHI1 (Next) T1	0		0		ns
t_{Dv}	4-6	AD0-AD31 Valid (Memory Write)	After R.E., PHI1 T2		50		38	ns
t_{Dh}	4-6	AD0-AD31 Hold (Memory Write)	After R.E., PHI1 (Next) T1	0		0		ns
t_{MAv}	4-6	PA0-31 Valid ($\overline{FLT}$ = 0)	After R.E., PHI1 T1		30		20	ns
t_{MAh}	4-6	PA0-31 Hold ($\overline{FLT}$ = 0)	After R.E., PHI1 (Next) T1	0		0		ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32382-10, NS32382-15.
Maximum times assume capacitive loading of 50 pF. (Continued)

Name	Figure	Description	Reference/Conditions	NS32382-10		NS32382-15		Units
				Min	Max	Min	Max	
t_{MADSa}	4-6, 15	$\overline{MADS}$ Signal Active ($\overline{FLT}$ = 0)	After R.E., PHI1 T1		25		17	ns
t_{MADSia}	4-6	$\overline{MADS}$ Signal Inactive	After R.E., PHI2 T1	5	35	5	25	ns
t_{MADSw}	4-6	$\overline{MADS}$ Pulse Width	At 0.8V (Both Edges)	35		22		ns
t_{DDINf}	4-7, 4-9, 11	$\overline{DDIN}$ Floating	After R.E., PHI1 T3 After R.E., PHI1 T1		25		25	ns
t_{MILOa}	4-5, 4-15	$\overline{MILO}$ Signal Active	After R.E., PHI1 T4		50		38	
t_{MILOia}	4-7, 4-15	$\overline{MILO}$ Signal Inactive	After R.E., PHI1 T1 or Ti		50		38	ns
t_{ABTa}	4-8	$\overline{RST}/\overline{ABT}$ Signal Active (Abort)	After R.E., PHI1 T1 or T2		50		40	ns
t_{ABTia}	4-8	$\overline{RST}/\overline{ABT}$ Signal Inactive (Abort)	After R.E., PHI1 T2 or T3	2	50	2	40	ns
t_{ABTw}	4-8	$\overline{RST}/\overline{ABT}$ Pulse Width (Abort)	At 0.8V (Both Edges)	60		40		ns
t_{FLTa}	4-5	$\overline{FLT}$ Signal Active	After R.E., PHI1 T2		50		40	ns
t_{FLTia}	4-7, 4-9	$\overline{FLT}$ Signal Inactive	After R.E., PHI1 T2		40		30	ns
t_{Df}	4-12	Data Bits Floating (Slave Processor Read)	After R.E., PHI1 T4		25		18	
t_{Dv}	4-12	AD0-AD31 Valid (CPU Slave Read)	After R.E., PHI1 T1		50		38	ns
t_{Dh}	4-12	AD0-AD31 Hold (CPU Slave Read)	After R.E., PHI1 T4	4		3		ns
t_{SDNa}	4-14	$\overline{SDONE}$ Signal Active	After R.E., PHI2		50		35	ns
t_{SDNia}	4-14	$\overline{SDONE}$ Signal Inactive	Ater R.E., PHI1		50		35	ns
t_{SDNw}	4-14	$\overline{SDONE}$ Pulse Width	At 0.8V (Both Edges)	25	90	17	60	ns
t_{SDNdw}	4-14	$\overline{SDONE}$ Double Pulse Width	At 0.8V (Both Edges)	225	275	140	180	ns
t_{SDNf}	4-14	$\overline{SDONE}$ Signal Floating	After R.E., PHI2		40		25	ns
t_{HLDAOa}	4-15	$\overline{HLDAO}$ Signal Active ($\overline{FLT}$ = 0)	After R.E., PHI1 Ti		60		40	ns
$t_{HLDAOia}$	4-15	$\overline{HLDAO}$ Signal Inactive ($\overline{FLT}$ = 0)	After R.E., PHI1 T4		60		40	ns
t_{MADSz}	4-15	$\overline{MADS}$ Signal Floated by $\overline{HOLD}$	After R.E., PHI1 Ti		40		25	ns
t_{PAVz}	4-15	$\overline{PAV}$ Signal Floated by $\overline{HOLD}$	After R.E., PHI1 Ti		40		25	ns
t_{PAVr}	4-15	$\overline{PAV}$ Return from Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 T1		40		25	ns
t_{Dz}	4-15	AD0-AD31 Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 Ti		25		18	ns
t_{MAz}	4-15	PA0-31 Floated by $\overline{HOLD}$	After R.E., PHI1 Ti		25		18	ns
t_{DDINz}	4-15	$\overline{DDIN}$ Signal Floated by $\overline{HOLD}$	After R.E., PHI1 Ti		40		25	ns
t_{CIz}	4-15	CINH Signal Floated by $\overline{HOLD}$	After R.E., PHI1 Ti		25		18	ns
t_{MILOia}	4-15	$\overline{MILO}$ Signal Inactive by $\overline{HOLD}$ ($\overline{FLT}$ = 0)	After R.E., PHI1 Ti		50		38	ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32382-10, NS32382-15.
Maximum times assume capacitive loading of 50 pF. (Continued)

Name	Figure	Description	Reference/Conditions	NS32382-10		NS32382-15		Units
				Min	Max	Min	Max	
t_{MILOa}	4–15	$\overline{MILO}$ Signal Active ($\overline{FLT}$ = 0)	After R.E., PHI1 T4		50		38	ns
t_{HLDAOa}	4–16	$\overline{HLDAO}$ Signal Active ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti		45		30	ns
$t_{HLDAOia}$	4–16	$\overline{HLDAO}$ Signal Inactive ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti or T4		45		30	ns
t_{MADSz}	4–16	$\overline{MADS}$ Signal Floated by $\overline{HLDAI}$ ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti		25		18	ns
t_{MADSr}	4–16	$\overline{MADS}$ Return from Floating ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti or T4		30		20	ns
t_{PAVz}	4–16	$\overline{PAV}$ Signal Floated $\overline{HLDAI}$ ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti		25		18	ns
t_{PAVr}	4–16	$\overline{PAV}$ Return from Floating ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti or T4		30		20	ns
t_{Dz}	4–16	AD0–AD31 Signals Floating ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti		25		18	ns
t_{Dr}	4–16	AD0–AD31 Return from Floating ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti or T4		30		20	ns
t_{MAz}	4–16	PA0–31 Signals Floated by $\overline{HLDAI}$ ($\overline{FLT}$ = 1)	After R.E., PHI1 T1		25		18	ns
t_{MAr}	4–16	PA0–31 Return from Floating ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti or T4		30		20	ns
t_{Clz}	4–16	CINH Signal Floated by $\overline{HLDAI}$ ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti		25		18	ns
t_{Clr}	4–16	CINH Return from Floating ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti or T4		30		20	ns
t_{RSTOa}	4–18	$\overline{RST}/\overline{ABT}$ Signal Active (Reset)	After R.E., PHI2 Ti		50		40	ns
t_{RSTOia}	4–18	$\overline{RST}/\overline{ABT}$ Signal Inactive (Reset)	After R.E. PHI2 Ti		50		40	ns
t_{RSTOw}	4–18	$\overline{RST}/\overline{ABT}$ Pulse Width (Reset)	At 0.8V (Both Edges)	64		64		t_{cp}

4.4.2.2 Input Signal Requirements: NS32382-10, NS32382-15

Name	Figure	Description	Reference/Conditions	NS32382-10		NS32382-15		Units
				Min	Max	Min	Max	
t_{DIs}	4–5	Input Data Setup ($\overline{FLT}$ = 0)	Before F.E., PHI2 T3	12		10		ns
t_{DIh}	4–5	Input Data Hold ($\overline{FLT}$ = 0)	After R.E., PHI1 T4	3		3		ns
t_{RDYs}	4–5	RDY Setup	Before F.E., PHI1 T3	20		12		ns
t_{RDYh}	4–5	RDY Hold	After R.E., PHI2 T3	4		3		ns
t_{SPCs}	4-12	$\overline{SPC}$ Input Setup	Before F.E., PHI2 T1	45		35		ns
t_{SPCh}	4-12	$\overline{SPC}$ Input Hold	After R.E., PHI1 T4	0		0		ns
t_{USs}	4–4, 4-12	U/$\overline{S}$ Setup	Before F.E., PHI2 T4	25		20		ns
t_{USh}	4–4, 4–12	U/$\overline{S}$ Hold	After R.E., PHI1 (Next) T4	0		0		ns
t_{STs}	4–4, 4–12	ST0–3 Setup	Before F.E., PHI2 T4	40		25		ns
t_{STh}	4–4, 4–12	ST0–3 Hold	After R.E., PHI1 (Next) T4	0		0		ns
t_{DIs}	4–13	Data In Setup (Slave Processor Write)	Before F.E., PHI2 T1	40		22		ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32382-10, NS32382-15 (Continued)

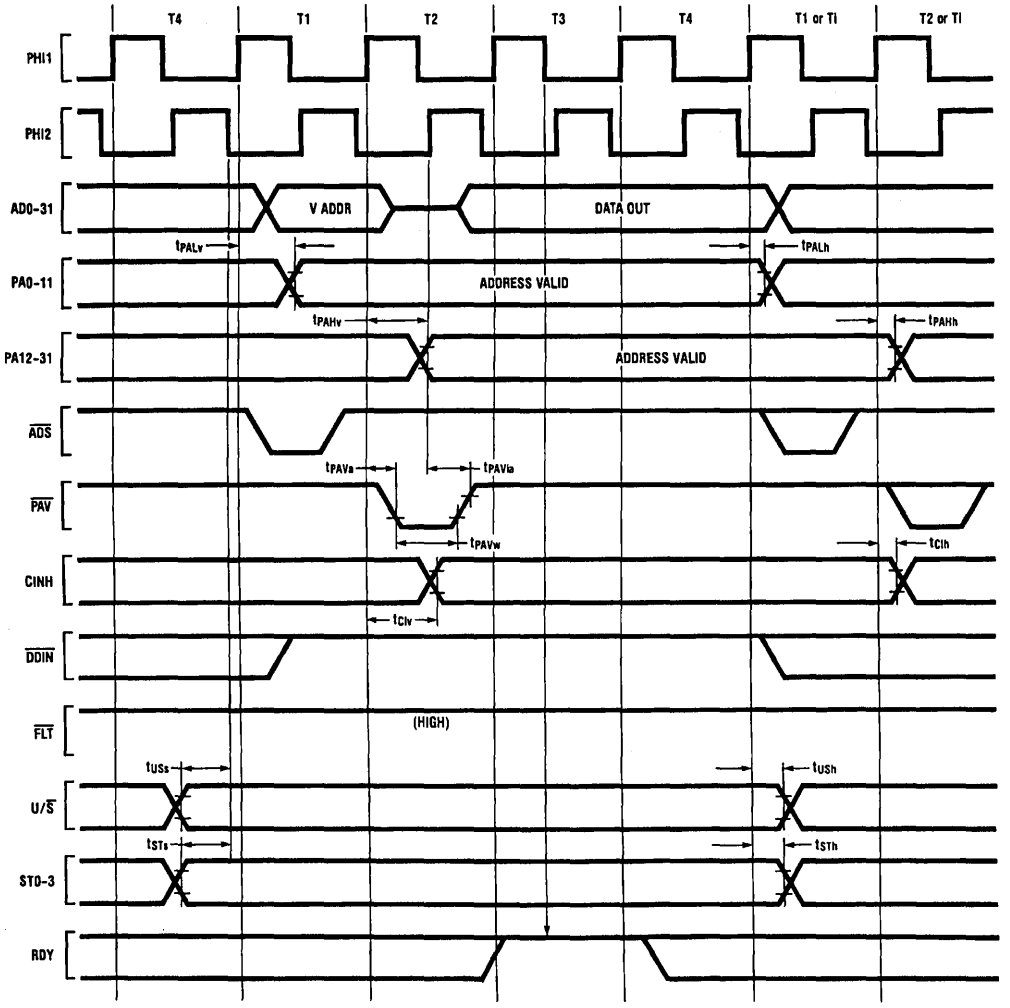
Name	Figure	Description	Reference/Conditions	NS32382-10		NS32382-15		Units
				Min	Max	Min	Max	
t_{DIh}	4–13	Data In Hold (Slave Processor Write)	After R.E., PHI1 (Next) Ti	3		3		ns
t_{HOLDs}	4-15	$\overline{HOLD}$ Setup ($\overline{FLT}$ = 0)	Before F.E., PHI2 T3	15		15		ns
t_{HOLDh}	4-15	$\overline{HOLD}$ Hold ($\overline{FLT}$ = 0)	After R.E., PHI1 T4	0		0		ns
t_{HLDAis}	4-16	$\overline{HLDAI}$ Signal Setup ($\overline{FLT}$ = 1)	Before F.E., PHI2 Ti	25		15		ns
t_{HLDAih}	4-16	$\overline{HLDAI}$ Signal Hold ($\overline{FLT}$ = 1)	After R.E., PHI1 Ti or T4	0		0		ns
t_{BRTs}	4-10	$\overline{BRT}$ Signal Setup ($\overline{FLT}$ = 0)	Before F.E., PHI1 T3 or T4	25		14		ns
t_{BRTh}	4-10	$\overline{BRT}$ Signal Hold ($\overline{FLT}$ = 0)	After R.E., PHI2 T3 or T4	0		0		ns
t_{BERs}	4-11	$\overline{BER}$ Signal Setup ($\overline{FLT}$ = 0)	Before F.E., PHI1 T4	25		14		ns
t_{BERh}	4-11	$\overline{BER}$ Signal Hold ($\overline{FLT}$ = 0)	After R.E., PHI2 T4	0		0		ns
t_{RSTIs}	4-18	Reset Input Setup	Before F.E., PHI1 Ti	20		10		ns
t_{RSTIw}	4–18	Reset Input Width	At 0.8V (Both Edges)	64		64		t_{cp}

4.4.2.3 Clocking Requirements: NS32382-10, NS32382-15

Name	Figure	Description	Reference/ Conditions	NS32382-10		NS32382-15		Units
				Min	Max	Min	Max	
t_{Cp}	4–17	Clock Period	R.E., PHI1, PHI2 to Next R.E., PHI1, PHI2	100	250	66	250	ns
$t_{CLw(1,2)}$	4–17	PHI1, PHI2 Pulse Width	At 2.0V on PHI1, PHI2 (Both Edges)	0.5 t_{cp} −10 ns		0.5 t_{cp} −6 ns		
$t_{CLh(1,2)}$	4–17	PHI1, PHI2 High Time	At V_{CC} − 0.9V on PHI1, PHI2 (Both Edges)	0.5 t_{cp} −15 ns		0.5 t_{cp} −10 ns		
t_{CLl}	4–17	PHI1, PHI2 Low Time	At 0.8V on PHI1, PHI2 (Both Edges)	0.5 t_{Cp} −5 ns		0.5 t_{Cp} −5 ns		
$t_{nOVL(1,2)}$	4–17	Non-Overlap Time	0.8V on F.E., PHI1, PHI2 to 0.8V on R.E., PHI2, PHI1	−2	5	−2	5	ns
t_{nOVLas}		Non-Overlap Asymmetry ($t_{nOVL(1)} - t_{nOVL(2)}$)	At 0.8V on PHI1, PHI2	−4	4	−3	3	ns
t_{CLhas}		PHI1, PHI2 Asymmetry $t_{CLh(1)} - t_{CLh(2)}$)	At V_{CC} − 0.9V on PHI1, PHI2	−5	5	−3	3	ns

4.0 Device Specifications (Continued)

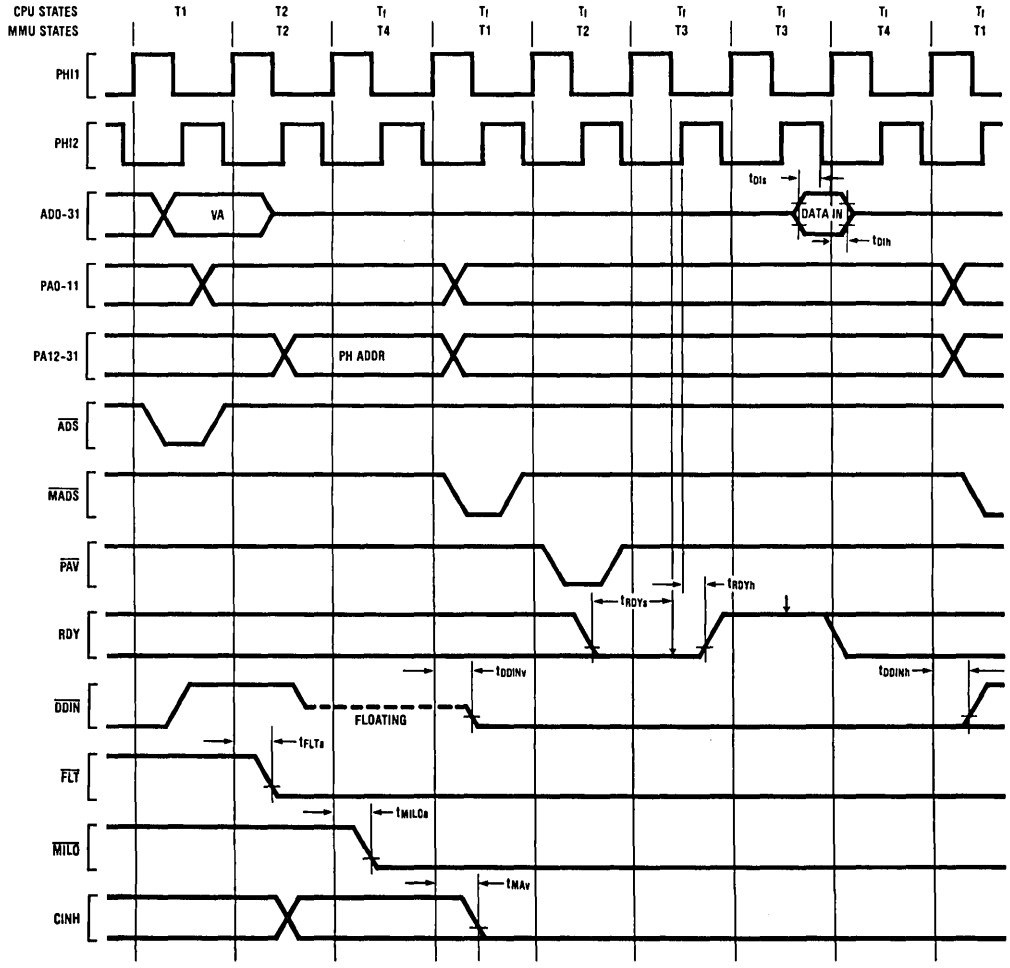
4.4.3 Timing Diagrams



TL/EE/9142-31

FIGURE 4-4. CPU Write Cycle Timing; Translation in TLB

4.0 Device Specifications (Continued)



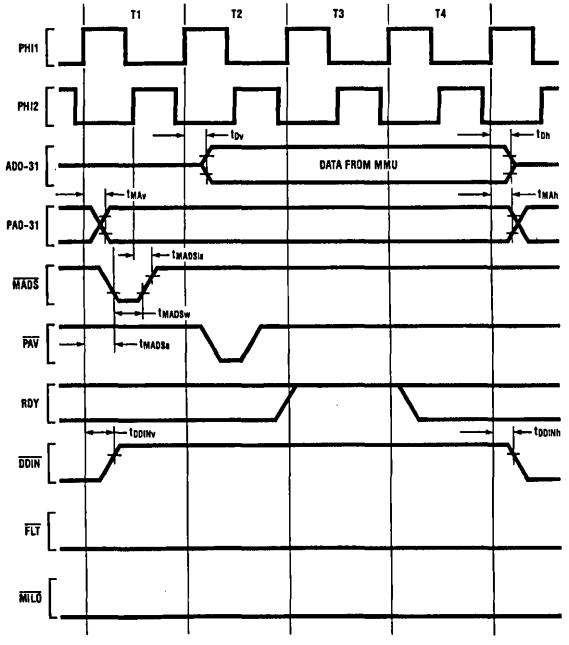
TL/EE/9142-32

FIGURE 4-5. MMU Read Cycle Timing (1-Wait State); After a TLB Miss

Note: After $\overline{FLT}$ is deasserted, $\overline{DDIN}$ may be driven temporarily by both CPU and MMU. This, however, does not cause any conflict. Since CPU and MMU force $\overline{DDIN}$ to the same logic level.

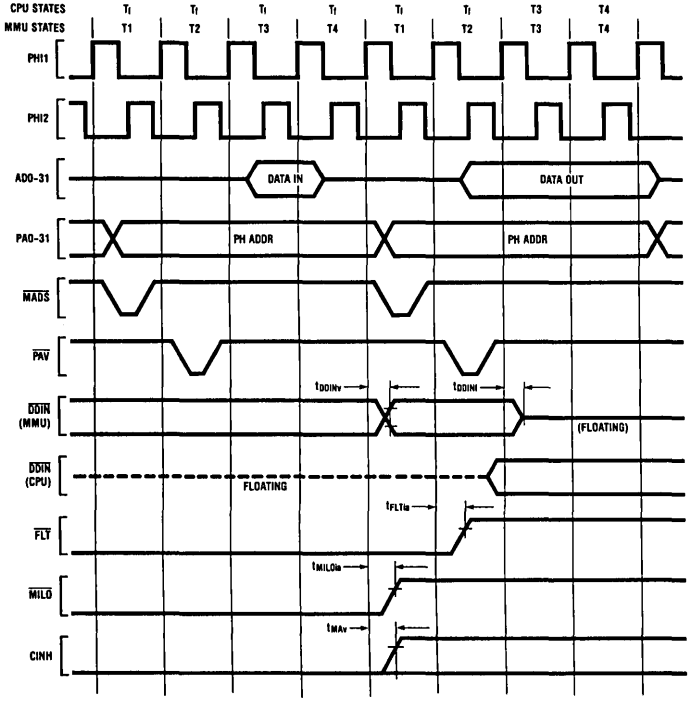
3

4.0 Device Specifications (Continued)



TL/EE/9142-33

FIGURE 4-6. MMU Write Cycle Timing; after a TLB Miss

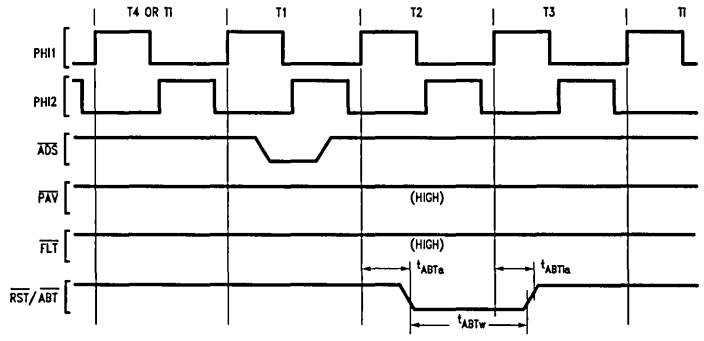


TL/EE/9142-34

FIGURE 4-7. $\overline{\text{FLT}}$ Deassertion Timing

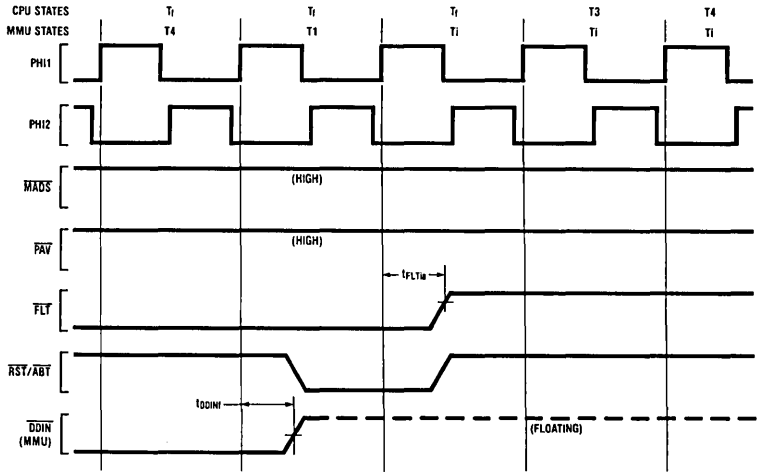
Note: After $\overline{\text{FLT}}$ is deasserted, $\overline{\text{DDIN}}$ may be driven temporarily by both CPU and MMU. This, however, does not cause any conflict. Since CPU and MMU force $\overline{\text{DDIN}}$ to the same logic level.

4.0 Device Specifications (Continued)



TL/EE/9142-35

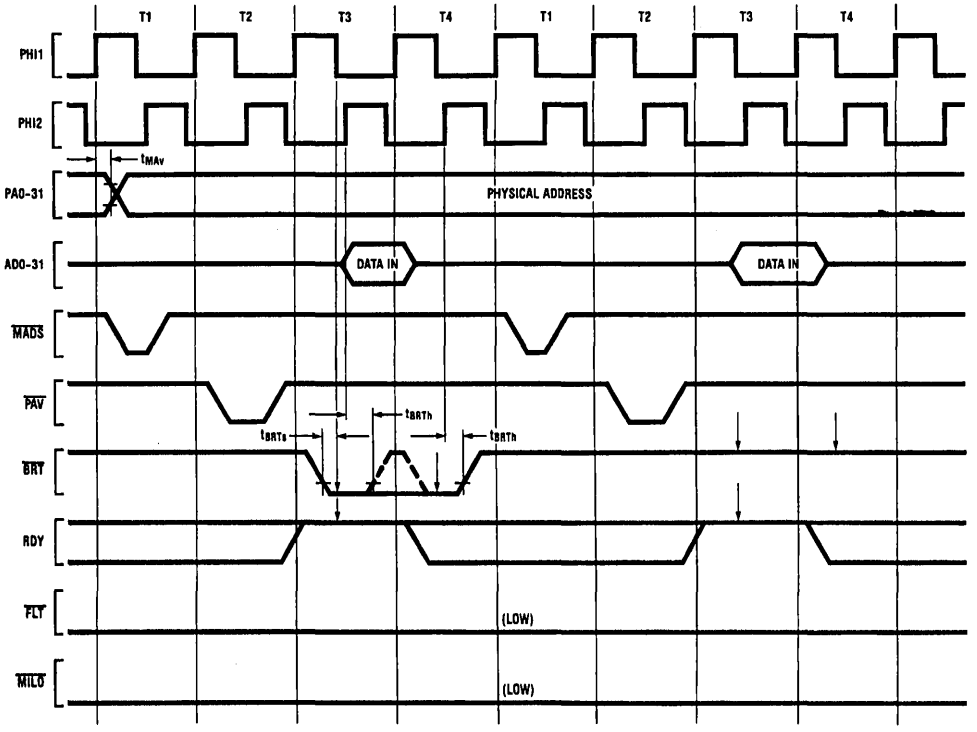
FIGURE 4-8. Abort Timing ($\overline{FLT}$ = 1)



TL/EE/9142-36

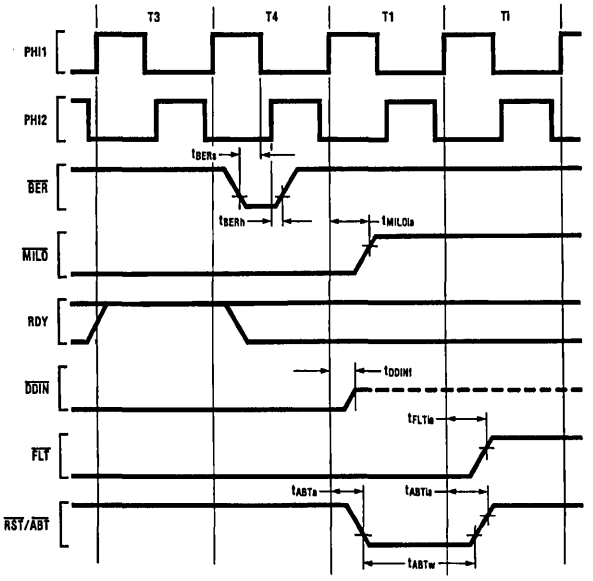
FIGURE 4-9. Abort Timing ($\overline{FLT}$ = 0)

4.0 Device Specifications (Continued)



TL/EE/9142-37

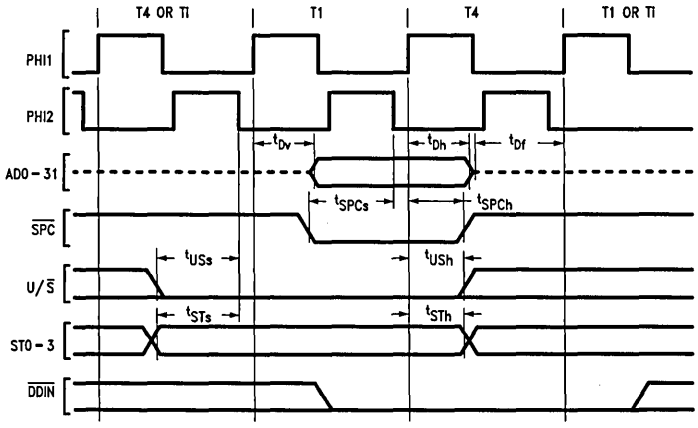
FIGURE 4-10. MMU Bus Retry Timing



TL/EE/9142-53

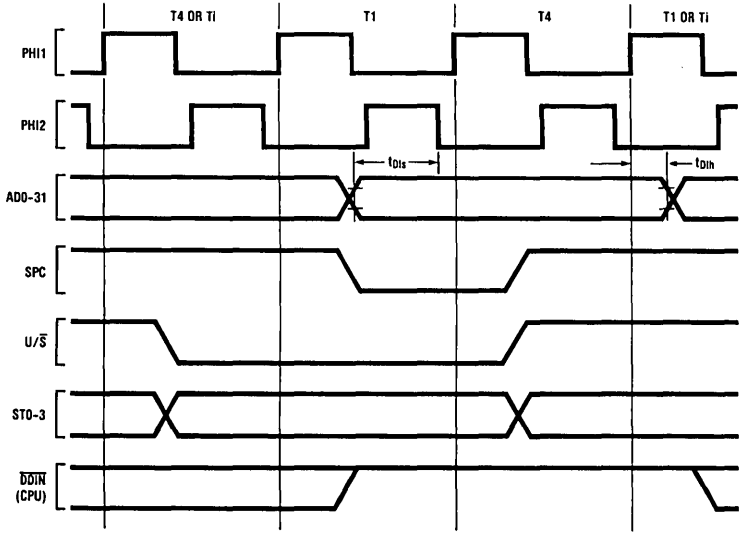
FIGURE 4-11. Bus Error Timing

4.0 Device Specifications (Continued)



TL/EE/9142–38

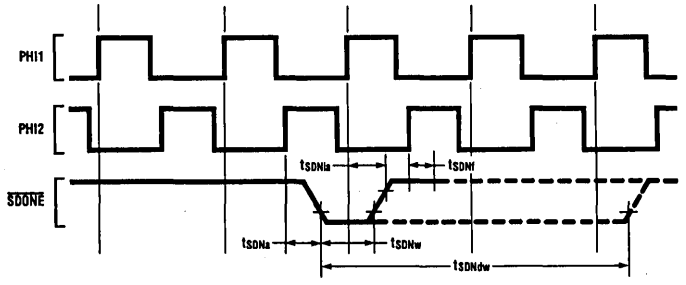
FIGURE 4-12. Slave Access Timing; CPU Reading from MMU



TL/EE/9142–39

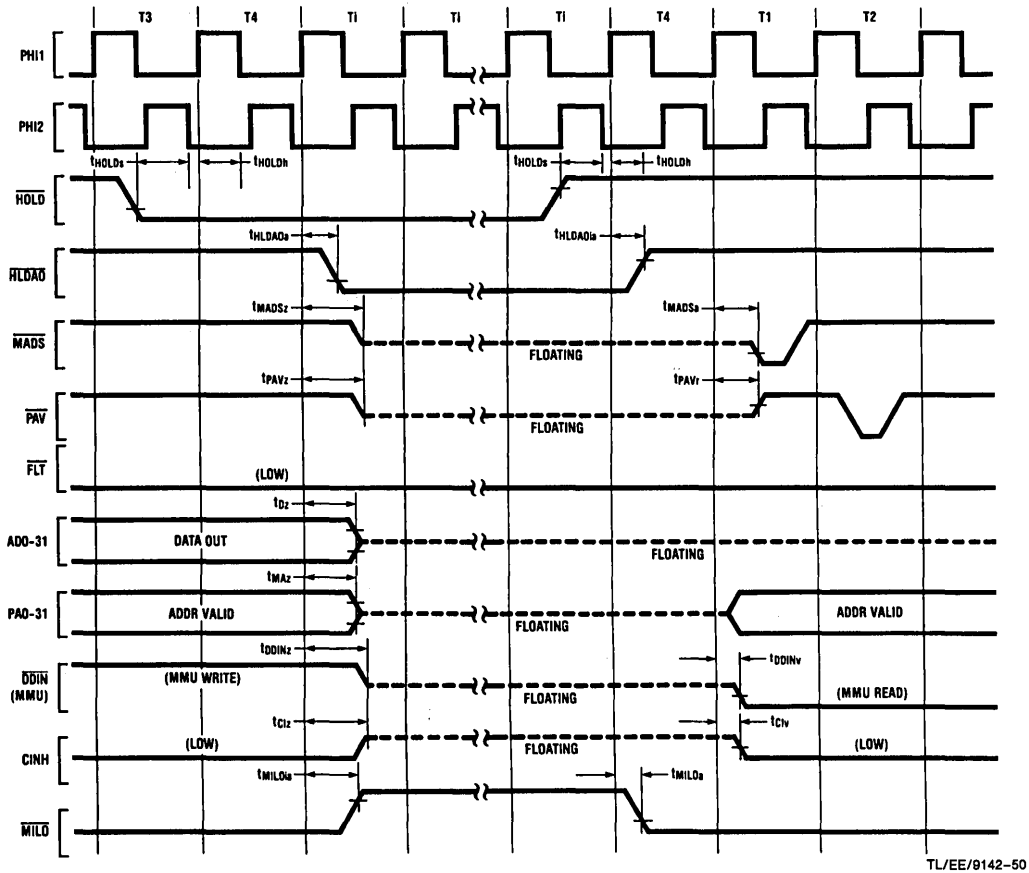
FIGURE 4-13. Slave Access Timing; CPU Writing to MMU

4.0 Device Specifications (Continued)



TL/EE/9142-40

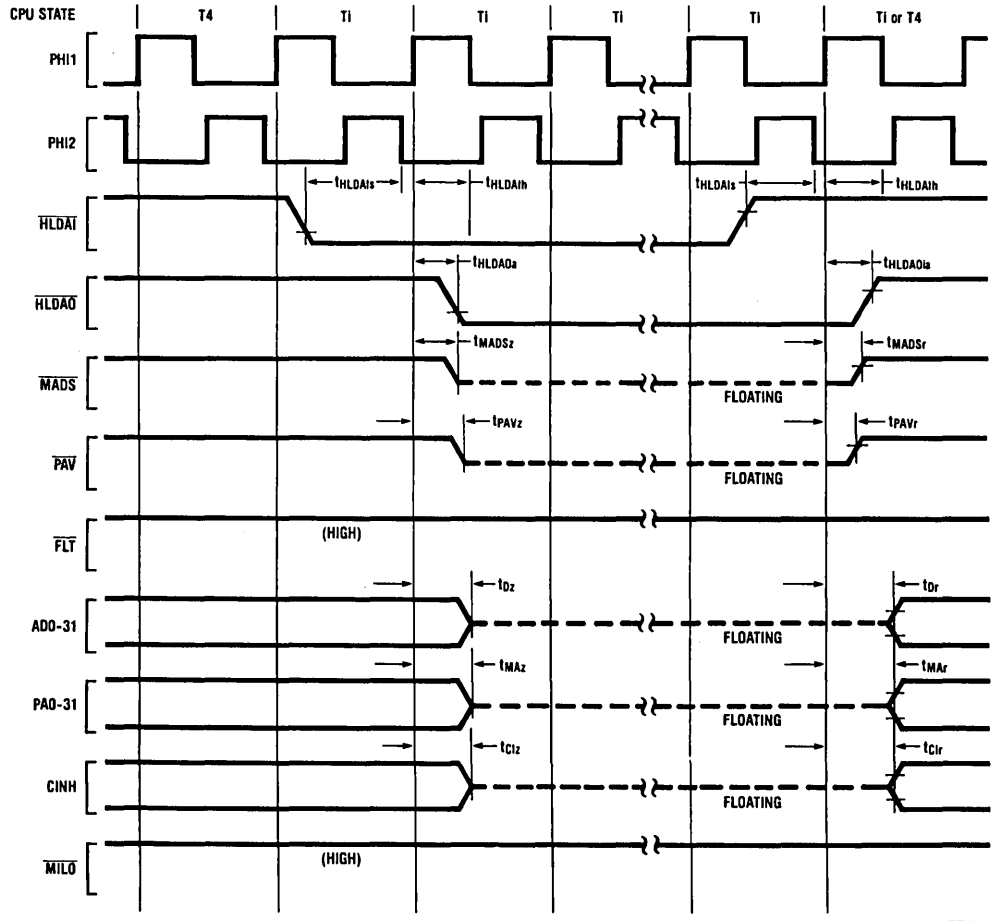
FIGURE 4-14. $\overline{\text{SDONE}}$ Timing



TL/EE/9142-50

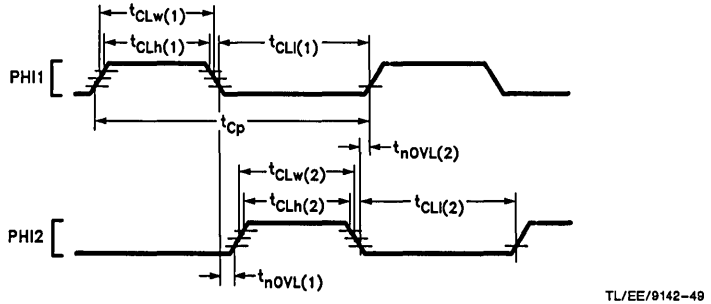
FIGURE 4-15. Hold Timing ($\overline{\text{FLT}}$ = 0)

4.0 Device Specifications (Continued)



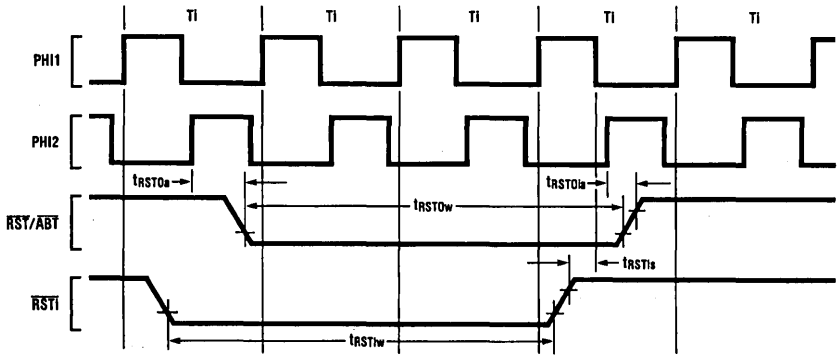
TL/EE/9142-51

FIGURE 4-16. Hold Timing ($\overline{FLT}$ = 1)



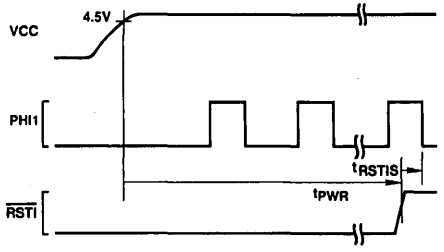
TL/EE/9142-49

FIGURE 4-17. Clock Waveforms



TL/EE/9142–45

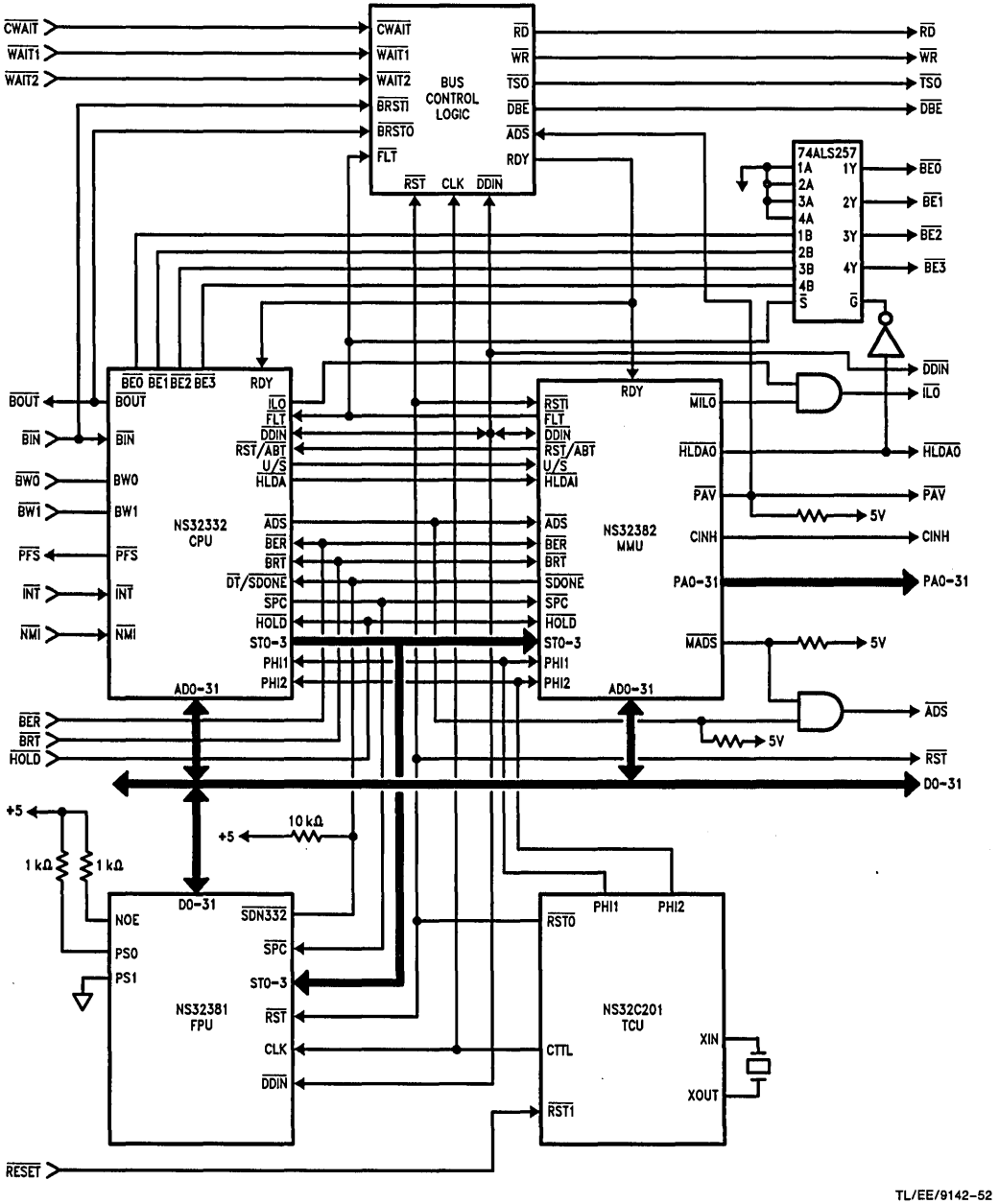
FIGURE 4-18. Non Power-On Reset Timing



TL/EE/9142–46

FIGURE 4-19. Power-On Reset

Appendix A: Interfacing Suggestions



TL/EE/9142-52

FIGURE A-1. System Connection Diagram

National Semiconductor

NS32082-10 Memory Management Unit

General Description

The NS32082 Memory Management Unit (MMU) provides hardware support for demand-paged virtual memory implementations. The NS32082 functions as a slave processor in Series 32000 microprocessor-based systems. Its specific capabilities include fast dynamic translation, protection, and detailed status to assist an operating system in efficiently managing up to 32 Mbytes of physical memory. Support for multiple address spaces, virtual machines, and program debugging is provided.

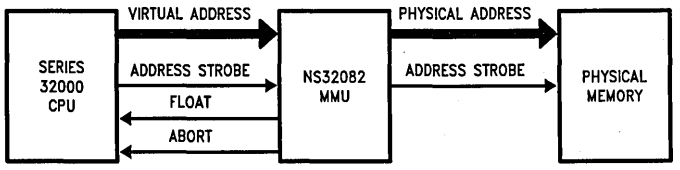
High-speed address translation is performed on-chip through a 32-entry fully associative translation look-aside buffer (TLB), which maintains itself from tables in memory with no software intervention. Protection violations and page faults (references to non-resident pages) are automatically detected by the MMU, which invokes the instruction abort feature of the CPU.

Additional features for program debugging include two breakpoint registers and a breakpoint counter, which provide the programmer with powerful stand-alone debugging capability.

Features

- Totally automatic mapping of 16 Mbyte virtual address space using memory based tables
- On-chip translation look-aside buffer allows 97% of translations to occur in one clock for most applications
- Full hardware support for virtual memory and virtual machines
- Implements "referenced" bits for simple, efficient working set management
- Protection mechanisms implemented via access level checking and dual space mapping
- Program debugging support
- Compatible with NS32016, NS32032 and NS32332 CPUs
- 48-pin dual-in-line package

Conceptual Address Translation Model



TL/EE/8692-1

Table Of Contents

List of Illustrations

List of Illustrations (Continued)

Tables

1.0 Product Introduction

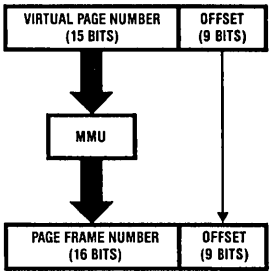
The NS32082 MMU provides hardware support for three basic features of the Series 32000; dynamic address translation, access level checking and software debugging. Dynamic Address Translation is required to implement demand-paged virtual memory. Access level checking is performed during address translation, ensuring that unauthorized accesses do not occur. Because the MMU resides on the local bus and is in an ideal location to monitor CPU activity, debugging functions are also included.

The MMU is intended for use in implementing demand-paged virtual memory. The concept of demand-paged virtual memory is illustrated in *Figure 1-1*. At any point in time, a program sees a uniform addressing space of up to 16 megabytes (the "virtual" space), regardless of the actual size of the memory physically present in the system (the "physical" space). The full virtual space is recorded as an image on a mass storage device. Portions of the virtual space needed by a running program are copied into physical memory when needed.

To make the virtual information directly available to a running program, a mapping must be established between the virtual addresses asserted by the CPU and the physical addresses of the data being referenced.

To perform this mapping, the MMU divides the virtual memory space into 512-byte blocks called "pages." It interprets the 24-bit address from the CPU as a 15-bit "page number" followed by a 9-bit offset, which indicates the position of a byte within the selected page. Similarly, the MMU divides the physical memory into 512-byte frames, each of which can hold a virtual page.

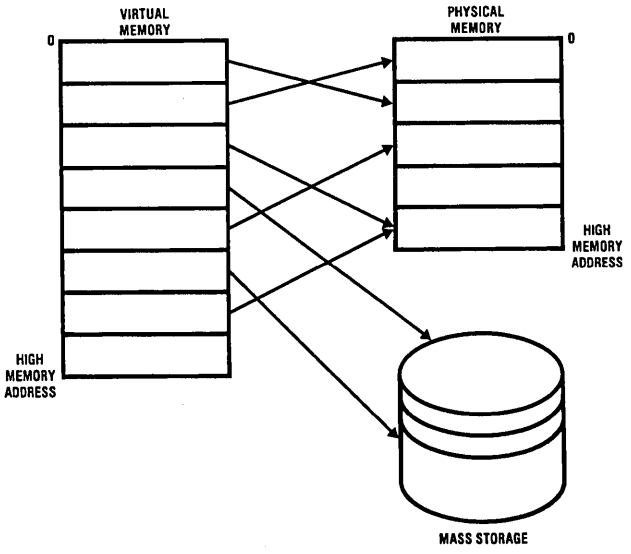
The translation process is therefore modeled as accepting a virtual page number from the CPU and substituting the corresponding physical page frame number for it, as shown in *Figure 1-2*. The offset is not changed. The translated page frame number is 16 bits long, including an additional address bit (A24) intended for physical bank selection. Physical addresses issued by the MMU are 25 bits wide.



TL/EE/8692-3

FIGURE 1-2. NS32082 Address Translation Model

Generally, in virtual memory systems the available physical memory space is smaller than the maximum virtual memory space. Therefore, not all virtual pages are simultaneously resident. Nonresident pages are not directly addressable by the CPU. Whenever the CPU issues a virtual address for a nonresident or nonexistent page, a "page fault" will result. The MMU signals this condition by invoking the Abort feature of the CPU. The CPU then halts the memory cycle,



TL/EE/8692-2

FIGURE 1-1. The Virtual Memory Model

1.0 Product Introduction (Continued)

restores its internal state to the point prior to the instruction being executed, and enters the operating system through the abort trap vector.

The operating system reads from the MMU the virtual address which caused the abort. It selects a page frame which is either vacant or not recently used and, if necessary, writes this frame back to mass storage. The required virtual page is then copied into the selected page frame.

The MMU is informed of this change by updating the page tables (Section 3.2), and the operating system returns control to the aborted program using the RETT instruction. Since the return address supplied by the abort trap is the address of the aborted instruction, execution resumes by retrying the instruction.

This sequence is called paging. Since a page fault encountered in normal execution serves as a demand for a given page, the whole scheme is called demand-paged virtual memory.

The MMU also provides debugging support. It may be programmed to monitor the bus for two virtual or physical addresses in real time. A counter register is associated with one of these, providing a "break-on-N-occurrences" capability.

1.1 PROGRAMMING CONSIDERATIONS

When a CPU instruction is aborted as a result of a page fault, some memory resident data might have been already modified by the instruction before the occurrence of the abort.

This could compromise the restartability of the instruction when the CPU returns from the abort routine.

To guarantee correct results following the re-execution of the aborted instruction, the following actions should not be attempted:

a) No instruction should try to overlay part of a source operand with part of the result. It is, however, permissible to rewrite the result into the source operand exactly if page faults are being generated only by invalid pages and not by write protection violations (for example, the instruction "ABSW X, X", which replaces X with its absolute value). Also, never write to any memory location which is necessary for calculating the effective address of either operand (i.e. the pointer in "Memory Relative" addressing mode; the Link Table pointer or Link Table Entry in "External" addressing mode).

b) No instruction should perform a conversion in place from one data type to another larger data type (Example: MOVWF X, X which replaces the 16-bit integer value in memory location X with its 32-bit floating-point value). The addressing mode combination "TOS, TOS" is an exception, and is allowed. This is because the least-significant part of the result is written to the possibly invalid page before the source operand is affected. Also, integer conversions to larger integers always work correctly in place, because the low-order portion of the result always matches the source value.

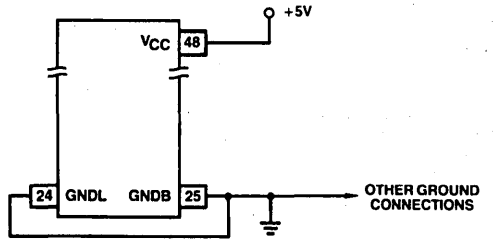
c) When performing the MOVM instruction, the entire source and destination blocks must be considered "operands" as above, and they must not overlap.

2.0 Functional Description

2.1 POWER AND GROUNDING

The NS32082 requires a single 5V power supply, applied on pin 48 (V_{CC}).

Grounding connections are made on two pins. Logic Ground (GNDL, pin 24) is the common pin for on-chip logic, and Buffer Ground (GNDB, pin 25) is the common pin for the output drivers. For optimal noise immunity, it is recommended that GNDL be attached through a single conductor directly to GNDB, and that all other grounding connections be made only to GNDB, as shown below (*Figure 2-1*).



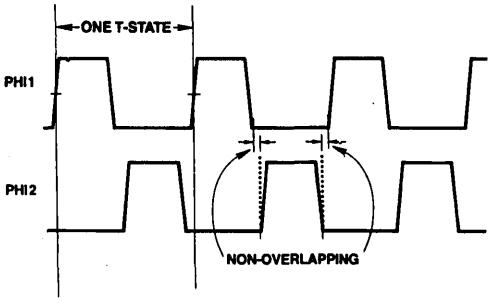
TL/EE/8692–4

FIGURE 2-1. Recommended Supply Connections

2.2 CLOCKING

The NS32082 inputs clocking signals from the NS32201 Timing Control Unit (TCU), which presents two non-overlapping phases of a single clock frequency. These phases are called PHI1 (pin 26) and PHI2 (pin 27). Their relationship to each other is shown in *Figure 2-2*.

Each rising edge of PHI1 defines a transition in the timing state ("T-State") of the MMU. One T-State represents one hardware cycle within the MMU, and/or one step of an external bus transfer. See Section 4 for complete specifications of PHI1 and PHI2.



TL/EE/8692–5

FIGURE 2-2. Clock Timing Relationships

As the TCU presents signals with very fast transitions, it is recommended that the conductors carrying PHI1 and PHI2 be kept as short as possible, and that they not be connected to any devices other than the CPU and MMU. A TTL Clock signal (CTTL) is provided by the TCU for all other clocking.

2.0 Functional Description (Continued)

2.3 RESETTING

The $\overline{RSTI}$ input pin is used to reset the NS32082. The MMU responds to $\overline{RSTI}$ by terminating processing, resetting its internal logic and clearing the appropriate bits in the MSR register.

Only the MSR register is changed on reset. No other program accessible registers, including the TLB are affected.

The $\overline{RST}/\overline{ABT}$ signal is activated by the MMU on reset. This signal should be used to reset the CPU. $\overline{AT}/\overline{SPC}$ is held low for five clock cycles after the rising edge of $\overline{RSTI}$ to indicate to the CPU that the address translation mode must be selected.

The A24/$\overline{HBF}$ signal is sampled by the MMU on the rising edge of $\overline{RSTI}$. It indicates the bus size of the attached CPU. A24/$\overline{HBF}$ must be sampled high for a 16-bit bus and low for a 32-bit bus.

On application of power, $\overline{RSTI}$ must be held low for at least 50 μs after V_{CC} is stable. This is to ensure that all on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must also remain active for not less than 64 clock cycles. The rising edge must occur while PHI1 is high. See *Figures 2-3* and *2-4*.

The NS32201 Timing Control Unit (TCU) provides circuitry to meet the Reset requirements of the NS32082 MMU. *Figure 2-5* shows the recommended connections.

2.4 BUS OPERATION

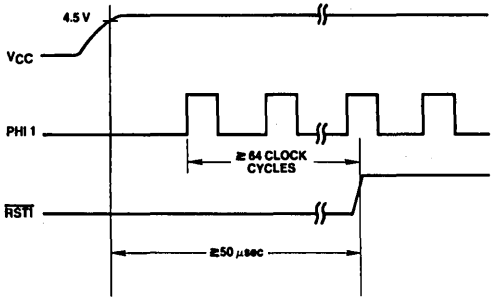
2.4.1 Interconnections

The MMU runs synchronously with the CPU, sharing with it a single multiplexed address/data bus. The interconnections used by the MMU for bus control, when used in conjunction with the NS32016, are shown in *Figure A-1* (Appendix A).

The CPU issues 24-bit virtual addresses on the bus, and status information on other pins, pulsing the signal $\overline{ADS}$ low. These are monitored by the MMU. The MMU issues 25-bit physical addresses on the bus, pulsing the $\overline{PAV}$ line low. The $\overline{PAV}$ pulse triggers the address latches and signals the NS32201 TCU to begin a bus cycle. The TCU in turn generates the necessary bus control signals and synchronizes the insertion of WAIT states, by providing the signal RDY to the MMU and CPU. Note that it is the MMU rather than the CPU that actually triggers bus activity in the system.

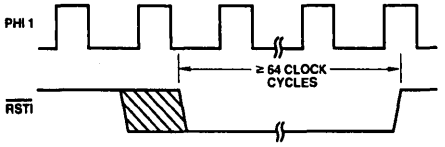
The functions of other interface signals used by the MMU to control bus activity are described below.

The ST0–ST3 pins indicate the type of cycle being initiated by the CPU. ST0 is the least-significant bit of the code. Table 2-1 shows the interpretations of the status codes presented on these lines.



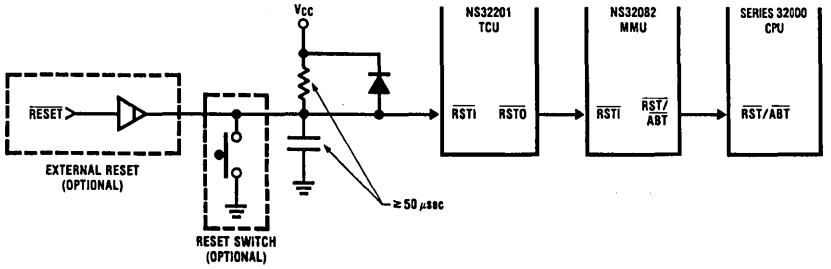
TL/EE/8692-6

FIGURE 2-3. Power-On Reset Requirements



TL/EE/8692-7

FIGURE 2-4. General Reset Timing



TL/EE/8692-8

FIGURE 2-5. Recommended Reset Connections, Memory-Managed System

2.0 Functional Description (Continued)

Status codes that are relevant to the MMU's function during a memory reference are:

1000, 1001 Instruction Fetch status, used by the debugging features to distinguish between data and instruction references.

1010 Data Transfer. A data value is to be transferred.

1011 Read RMW Operand. Although this is always a Read cycle, the MMU treats it as a Write cycle for purposes of protection and breakpointing.

1100 Read for effective address. Data used for address calculation is being transferred.

All other status codes are treated as data accesses if they occur in conjunction with a pulse on the $\overline{ADS}$ pin. Note that these include Interrupt Acknowledge and End of Interrupt cycles performed by the CPU. The status codes 1101, 1110 and 1111 are also recognized by the MMU in conjunction with pulses on the $\overline{SPC}$ line while it is executing Slave Processor instructions, but these do not occur in a context relevant to address translation.

TABLE 2-1. ST0–ST3 Encodings (ST0 is the Least Significant)

0000 — Idle: CPU Inactive on Bus
0001 — Idle: WAIT Instruction
0010 — (Reserved)
0011 — Idle: Waiting for Slave
0100 — Interrupt Acknowledge, Master
0101 — Interrupt Acknowledge, Cascaded
0110 — End of Interrupt, Master
0111 — End of Interrupt, Cascaded
1000 — Sequential Instruction Fetch
1001 — Non-Sequential Instruction Fetch
1010 — Data Transfer
1011 — Read Read-Modify-Write Operand
1100 — Read for Effective Address
1101 — Transfer Slave Operand
1110 — Read Slave Status Word
1111 — Broadcast Slave ID

The $\overline{DDIN}$ line indicates the direction of the transfer: 0 = Read, 1 = Write.

$\overline{DDIN}$ is monitored by the MMU during CPU cycles to detect write operations, and is driven by the MMU during MMU-initiated bus cycles.

The U/$\overline{S}$ pin indicates the privilege level at which the CPU is making the access: 0 = Supervisor Mode, 1 = User Mode. It is used by the MMU to select the address space for translation and to perform protection level checking. Normally, the U/$\overline{S}$ pin is a direct reflection of the U bit in the CPU's Processor Status Register (PSR). The MOVUS and MOVSU CPU instructions, however, toggle this pin on successive operand accesses in order to move data between virtual spaces.

The MMU uses the $\overline{FLT}$ line to take control of the bus from the CPU. It does so as necessary for updating its internal TLB from the Page Tables in memory, for maintaining the contents of the status bits (R and M) in the Page Table Entries, and for implementing bus timing adjustments needed by the debugging features.

The MMU also aborts invalid accesses attempted by the CPU. This is done by pulsing the $\overline{RST}/\overline{ABT}$ pin low for one clock period. (A pulse longer than one clock period is interpreted by the CPU as a Reset command).

Because the MMU performs only 16-bit transfers, some additional circuitry is needed to interface it to the 32-bit data bus of an NS32032-based system. However, since the MMU never writes to the most-significant word of a Page Table Entry, the only special requirement is that it must be able to read from the top half of the bus. This can be accomplished as shown in *Figure A-2* (Appendix A) by using a 16-bit unidirectional buffer and some gating circuitry that enables it whenever an MMU-initiated bus cycle accesses an address ending in binary "10".

The bus connections required in conjunction with the NS32332 CPU are somewhat more complex (see the NS32332 data sheet), but the sequences of events documented here still hold.

2.4.2 CPU-Initiated Bus Cycles

A CPU-initiated bus cycle is performed in a minimum of five clock cycles (four in the case of the NS32332): T1, TMMU, T2, T3 and T4, as shown in *Figure 2-6*.

During period T1, the CPU places the virtual address to be translated on the bus, and the MMU latches it internally and begins translation. The MMU also samples the $\overline{DDIN}$ pin, the status lines ST0–ST3, and the U/$\overline{S}$ pin to determine how the CPU intends to use the bus.

During period TMMU the CPU floats its bus drivers and the MMU takes one of three actions:

1) If the translation for the virtual address is resident in the MMU's TLB, and the access being attempted by the CPU does not violate the protection level of the page being referenced, the MMU presents the translated address and generates a $\overline{PAV}$ pulse to trigger a bus cycle in the rest of the system. See *Figure 2-6*.
2) If the translation for the virtual address is resident in the MMU's TLB, but the access being attempted by the CPU is not allowed due to the protection level of the page being referenced, the MMU generates a pulse on the $\overline{RST}/\overline{ABT}$ pin to abort the CPU's access. No $\overline{PAV}$ pulse is generated. See *Figure 2-7*.
3) If the translation for the virtual address is not resident in the TLB, or if the CPU is writing to a page whose M bit is not yet set, the MMU takes control of the bus asserting the $\overline{FLT}$ signal as shown in *Figure 2-8*. This causes the CPU to float its bus and wait. The MMU then initiates a sequence of bus cycles as described in Section 2.4.3.

From state T2 through T4 data is transferred on the bus between the CPU and memory, and the TCU provides the strobes for the transfer. During this time the MMU floats

2.0 Functional Description (Continued)

pins AD0–AD15, and handles pins A16–A24 according to the mode of operation (16-bit or 32-bit) selected during reset (Section 2.3).

In 16-bit bus mode, the MMU drives address lines A16–A24 from TMMU through T4 and they need not be latched externally. This is appropriate for the NS32016 CPU, which uses only AD0–AD15 for data transfers. In 32-bit bus mode, the MMU asserts the physical address on pins A16–A24 only during TMMU, and floats them from T2 through T4 because the CPU uses them for data transfer. In this case the physical address presented on these lines must be latched externally using $\overline{PAV}$.

Whenever the MMU generates an Abort pulse on the $\overline{RST}/\overline{ABT}$ pin, the CPU enters state T2 and then Ti (idle), ending the bus cycle. Since no $\overline{PAV}$ pulse is issued by the MMU, the rest of the system remains unaware that an access has been attempted. The MMU requires that no further memory references be attempted by the CPU for at least two clock cycles after the T2 state, as shown in *Figure 2-7*. This requirement is met by all Series 32000 CPU's. During this time, the RDY line must remain high. This requirement is met by the NS32201 TCU.

2.4.3 MMU-Initiated Cycles

Bus cycles initiated by the MMU are always nested within CPU-initiated bus cycles; that is, they appear after the MMU has accepted a virtual address from the CPU and has set the $\overline{FLT}$ line active. The MMU will initiate memory cycles in the following cases:

1) There is no translation in the MMU's TLB for the virtual address issued by the CPU, meaning that the MMU must reference the Page Tables in memory to obtain the translation.
2) There is a translation for that virtual address in the TLB, but the page is being written for the first time (the M bit in its Level-2 Page Table Entry is 0). The MMU treats this case as if there were no translation in the TLB, and performs a Page Table lookup in order to set the M bit in the Level-2 Page Table Entry as well as in the TLB.

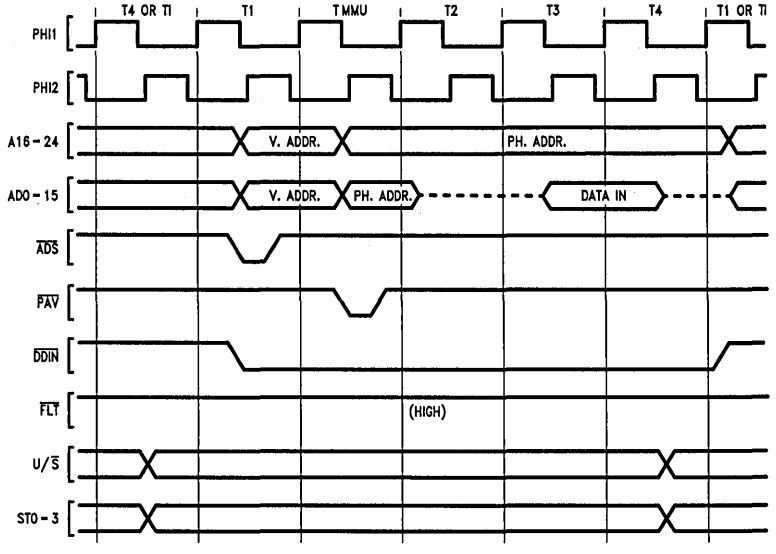
Having made the necessary memory references, the MMU either aborts the CPU access or it provides the translated address and allows the CPU's access to continue to T2.

Figure 2-8 shows the sequence of events in a Page Table lookup. After asserting $\overline{FLT}$, the MMU waits for one additional clock cycle, then reads the Level-1 Page Table Entry and the Level-2 Page Table Entry in four consecutive memory Read cycles. Note that the MMU performs two 16-bit transfers to read each Page Table Entry, regardless of the width of the CPU's data bus. There are no idle clock cycles between MMU-initiated bus cycles unless a bus request is made on the $\overline{HOLD}$ line (Section 2.6).

During the Page Table lookup the MMU drives the $\overline{DDIN}$ signal. The status lines ST0–ST3 and the U/$\overline{S}$ pin are not released by the CPU, and retain their original settings while the MMU uses the bus. The Byte Enable signals from the CPU ($\overline{HBE}$ in 16-bit systems, $\overline{BE0}$–$\overline{BE3}$ in 32-bit systems) should in general be handled externally for correct memory referencing. (The current NS32016 CPU does, however, handle $\overline{HBE}$ in a manner that is acceptable in many systems at clock rates of 12.5 MHz or less.)

In the clock cycle immediately after T4 of the last lookup cycle, the MMU removes the $\overline{FLT}$ signal, issues the translated address, and pulses $\overline{PAV}$ to continue the CPU's access.

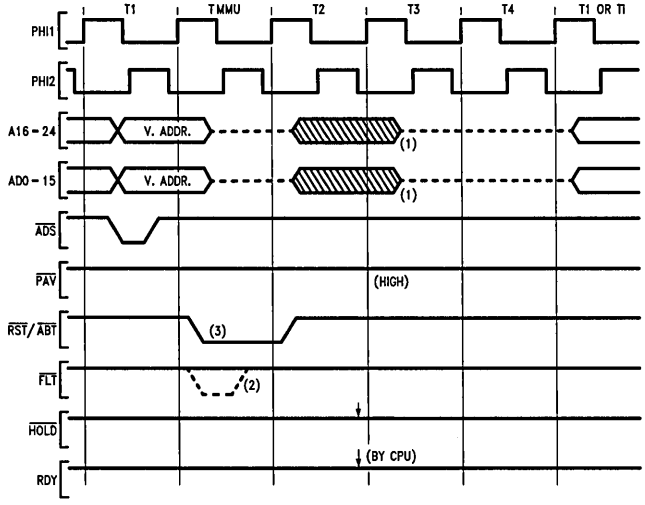
Note that when the MMU sets $\overline{FLT}$ active, the clock cycle originally called TMMU is redesignated Tf. Clock cycles in which the $\overline{PAV}$ pulse occurs are designated TMMU.



TL/EE/8692-9

FIGURE 2-6. CPU Read Cycle; Translation in TLB (TLB Hit)

2.0 Functional Description (Continued)



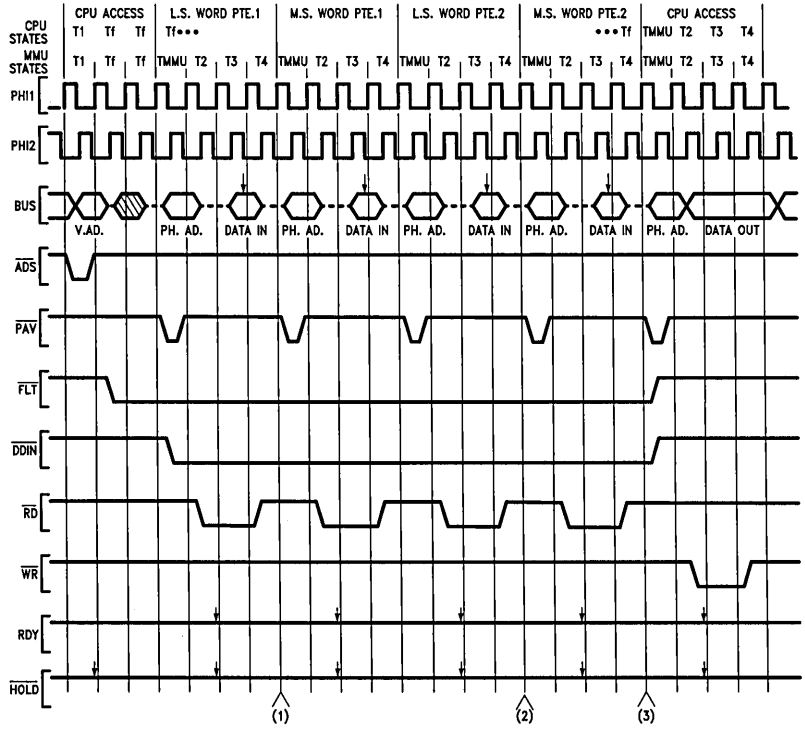
TL/EE/8692-10

Note 1: The CPU drives the bus if a write cycle is aborted.

Note 2: FLT may be pulsed if a breakpoint on physical address is enabled or an execution breakpoint is triggered.

Note 3: If this bus cycle is a write cycle to a write-protected page, FLT is asserted for two clock cycles and the abort pulse is delayed by one clock cycle.

FIGURE 2-7. Abort Resulting from Protection Violation; Translation in TLB



TL/EE/8692-11

Note 1: If the R bit on the Level-1 PTE must be set, a write cycle is inserted here.

Note 2: If either the R or the M bit on the Level-2 PTE must be set, a write cycle is inserted here.

Note 3: If a breakpoint on physical address is enabled, an extra clock cycle is inserted here.

FIGURE 2-8. Page Table Lookup

2.0 Functional Description (Continued)

The Page Table Entries are read starting with the low-order word. If the V bit (bit 0) of the low-order word is zero, or the protection level field PL (bits 1 and 2) indicates that the CPU's attempted access is illegal, the MMU does not generate any further memory cycles, but instead issues an Abort pulse during the clock cycle after T4 and removes the $\overline{FLT}$ signal. The CPU continues to T2 and then becomes idle on the bus, as shown in *Figure 2-9*.

If the R and/or M bit (bit 3 or 4) of the low-order word must be updated, the MMU does this immediately in a single Write cycle, before reading the high-order word of the Page Table Entry. All bits except those updated are rewritten with their original values.

At most, the MMU writes two 16-bit words to memory during a translation: the first to the Level-1 table to update the R bit, and the second to the Level-2 table to update the R and/or M bits.

2.4.4 Cycle Extension

To allow sufficient strobe widths and access time requirements for any speed of memory or peripheral device, the NS32082 provides for extension of a bus cycle. Any type of bus cycle, CPU-initiated or MMU-initiated, can be extended, except Slave Processor cycles, which are not memory or peripheral references.

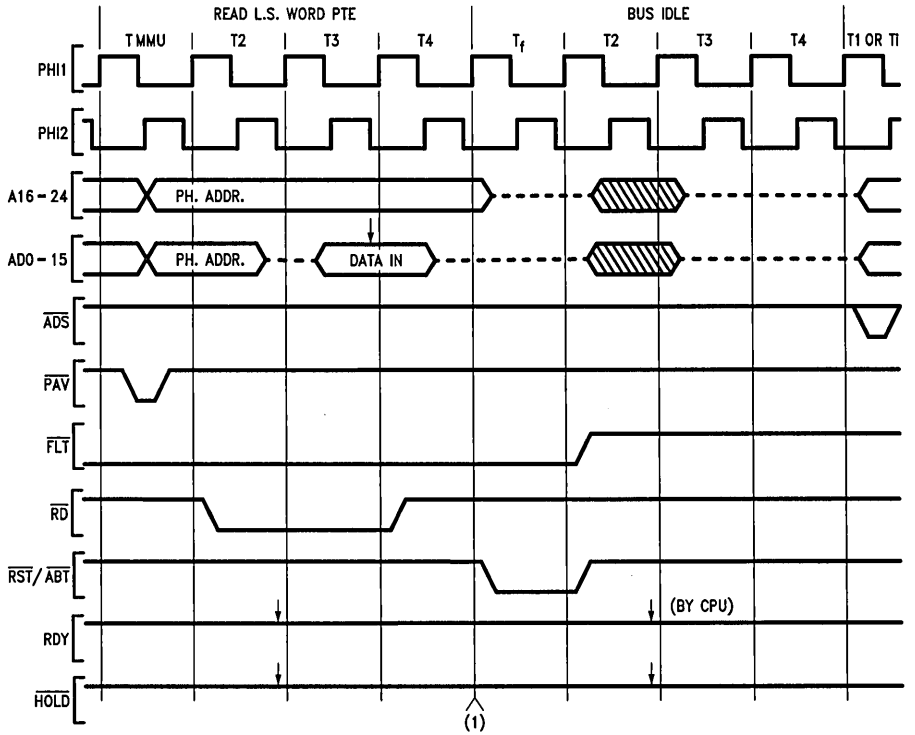
In *Figures 2-6* and *2-8*, note that during T3 all bus control signals are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the RDY (Ready) pin.

Immediately before T3 begins, on the falling edge of clock phase PHI2, the RDY line is sampled by the CPU and/or the MMU. If RDY is high, the next state after T3 will be T4, ending the bus cycle. If it is low, the next state after T3 will be another T3 and the RDY line will be sampled again. RDY is sampled in each following clock period, with insertion of additional T3 states, until it is sampled high. Each additional T3 state inserted is called a "WAIT state."

During CPU bus cycles, the MMU monitors the RDY pin only if the 16-bit mode is selected. This is necessary since the MMU drives the address lines A16–A24, and needs to detect the end of the bus cycle in order to float them.

If the 32-bit mode is selected, the above address lines are floated following the TMMU state. The MMU will be ready to perform another translation after three clock cycles, and the RDY line is ignored.

The RDY pin is driven by the NS32201 Timing Control Unit, which applies WAIT states to the CPU and MMU as requested on its own WAIT request input pins.



TL/EE/8692–12

Note 1: If a breakpoint on physical address is enabled, an extra clock cycle is inserted here.

FIGURE 2-9. Abort Resulting after a Page Table Lookup

2.0 Functional Description (Continued)

2.5 SLAVE PROCESSOR INTERFACE

The CPU and MMU execute four instructions cooperatively. These are LMR, SMR, RDVAL and WRVAL, as described in Section 2.5.2. The MMU takes the role of a Slave Processor in executing these instructions, accepting them as they are issued to it by the CPU. The CPU calculates all effective addresses and performs all operand transfers to and from memory and the MMU. The MMU does not take control of the bus except as necessary in normal operation; i.e., to translate and validate memory addresses as they are presented by the CPU.

The sequence of transfers ("protocol") followed by the CPU and MMU involves a special type of bus cycle performed by the CPU. This "Slave Processor" bus cycle does not involve the issuing of an address, but rather performs a fast data transfer whose purpose is pre-determined by the form of the instruction under execution and by status codes asserted by the CPU.

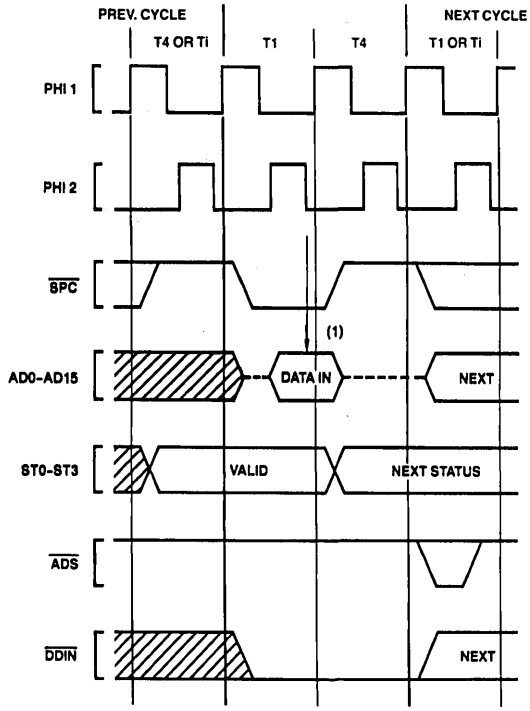
2.5.1 Slave Processor Bus Cycles

The interconnections between the CPU and MMU for Slave Processor communication are shown in *Figures A-1* and *A-2* (Appendix A). The low-order 16 bits of the bus are used for data transfers. The $\overline{SPC}$ signal is bidirectional. It is pulsed by the CPU as a low-active data strobe for Slave Processor transfers, and is also pulsed low by the MMU to acknowledge, when necessary, that it is ready to continue execution of an MMU instruction. Since $\overline{SPC}$ is normally in a high-impedance state, it must be pulled high with a 10 kΩ resistor, as shown. The MMU also monitors the status lines ST0–ST3 to follow the protocol for the instruction being executed.

Data is transferred between the CPU and the MMU with Slave Processor bus cycles, illustrated in *Figures 2-10* and *2-11*. Each bus cycle transfers one byte or one word (16 bits) to or from the MMU.

Slave Processor bus cycles are performed by the CPU in two clock periods, which are labeled T1 and T4. During T1, the CPU activates $\overline{SPC}$ and, if it is writing to the MMU, it presents data on the bus. During T4, the CPU deactivates $\overline{SPC}$ and, if it is reading from the MMU, it latches data from the bus. The CPU guarantees that data written to the MMU is held through T4 to provide for the MMU's hold time requirements. The CPU also guarantees that the status code on ST0–ST3 becomes valid, at the latest, during the clock period preceding T1. The status code changes during T4 to anticipate the next bus cycle, if any.

Note that Slave Processor bus cycles are never extended with WAIT states. The RDY line is not sampled.

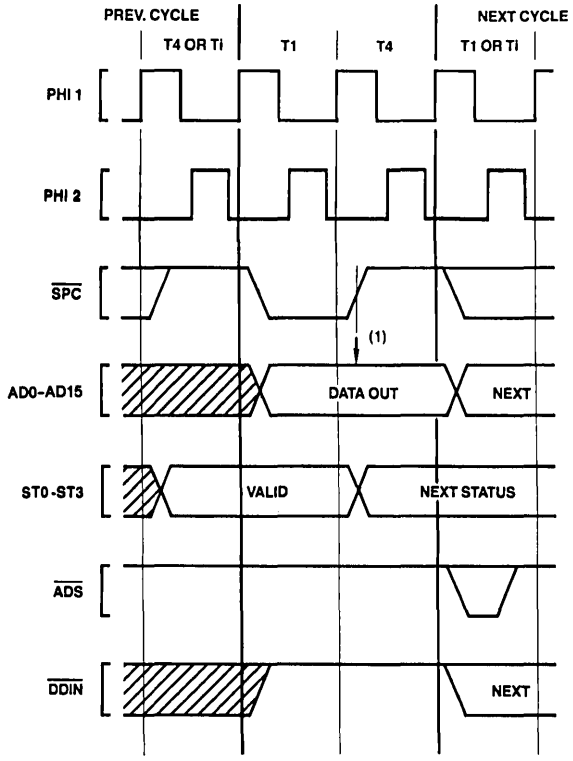


TL/EE/8692–13

Note 1: CPU samples Data Bus here.

FIGURE 2-10. Slave Access Timing; CPU Reading from MMU

2.0 Functional Description (Continued)



Note 1: MMU samples Data Bus here.

FIGURE 2-11. Slave Access Timing; CPU Writing to MMU

2.5.2 Instruction Protocols

MMU instructions have a three-byte Basic Instruction field consisting of an ID byte followed by an Operation Word. See *Figure 3-10* for the MMU instruction encodings. The ID Byte has three functions:

1) It identifies the instruction as being a Slave Processor instruction.
2) It specifies that the MMU will execute it.
3) It determines the format of the following Operation Word of the instruction.

The CPU initiates an MMU instruction by issuing first the ID Byte and then the Operation Word, using Slave Processor bus cycles. The ID Byte is sent on the least-significant byte of the bus, in conjunction with status code 1111 (Broadcast ID Byte). The Operation Word is sent on the entire 16-bit data bus, with status code 1101 (Transfer Operation Word / Operand). The Operation Word is sent with its bytes swapped; i.e., its least-significant byte is presented to the MMU on the most-significant half of the 16-bit bus.

Other actions are taken by the CPU and the MMU according to the instruction under execution, as shown in Tables 2-2, 2-3 and 2-4.

In executing the LMR instruction (Load MMU Register, Table 2-2), the CPU issues the ID Byte, the Operation Word, and then the operand value to be loaded by the MMU. The register to be loaded is specified in a field within the Operation Word of the instruction.

In executing the SMR instruction (Store MMU Register, Table 2-3), the CPU also issues the ID Byte and the Operation Word of the instruction to the MMU. It then waits for the MMU to signal (by pulsing $\overline{SPC}$ low) that it is ready to present the specified register's contents to the CPU. Upon receiving this "Done" pulse, the CPU reads first a "Status Word" (dictated by the protocol for Slave Processor instructions) which the MMU provides as a word of all zeroes. The CPU then reads the contents of the selected register in two successive Slave Processor bus cycles, and places this result value into the instruction's destination (a CPU general-purpose register or a memory location).

In executing the RDVAL (Read-Validate) or WRVAL (Write-Validate) instruction, the CPU again issues the ID Byte and the Operation Word to the MMU. However, its next action is to initiate a one-byte Read cycle from the memory address whose protection level is being tested. It does so while presenting status code 1010; this being the only place that this status code appears during a RDVAL or WRVAL instruction. This memory access triggers a special address translation from the MMU. The translation is performed by the MMU using User-Mode mapping, and any protection violation occurring during this memory cycle does not cause an Abort. The MMU will, however, abort the CPU if the Level-1 Page Table Entry is invalid.

Upon completion of the address translation, the MMU pulses $\overline{SPC}$ to acknowledge that the instruction may continue execution.

2.0 Functional Description (Continued)

TABLE 2-2. LMR Instruction Protocol

CPU Action	Status	MMU Action
Issues ID Byte of instruction, pulsing $\overline{SPC}$.	1111	Accepts ID Byte.
Sends Operation Word of Instruction, pulsing $\overline{SPC}$.	1101	Decodes instruction.
Issues low-order word of new register value to MMU, pulsing $\overline{SPC}$.	1101	Accepts word from bus; places it into low-order half of referenced MMU register.
Issues high-order word of new register value to MMU, pulsing $\overline{SPC}$.	1101	Accepts word from bus; places it into high-order half of referenced MMU register.

TABLE 2-3. SMR Instruction Protocol

CPU Action	Status	MMU Action
Issues ID Byte of Instruction, pulsing $\overline{SPC}$.	1111	Accepts ID Byte.
Sends Operation Word of instruction, pulsing $\overline{SPC}$.	1101	Decodes instruction.
Waits for Done pulse from MMU.	xxxx	Sends Done pulse on $\overline{SPC}$.
Pulses $\overline{SPC}$ and reads Status Word from MMU.	1110	Presents Status Word (all zeroes) on bus.
Pulses $\overline{SPC}$, reading low-order word of result from MMU.	1101	Presents low-order word of referenced MMU register on bus.
Pulses $\overline{SPC}$, reading high-order word of result from MMU.	1101	Presents high-order word of referenced MMU register on bus.

TABLE 2-4. RDVAL/WRVAL Instruction Protocol

CPU Action	Status	MMU Action
Issues ID Byte of instruction, pulsing $\overline{SPC}$.	1111	Accepts ID Byte.
Sends Operation Word of instruction, pulsing $\overline{SPC}$.	1101	Decodes instruction.
Performs dummy one-byte memory read from operand's location.	1010	Translates CPU's address, using User-Mode mapping, and performs requested test on the address presented by the CPU. Aborts the CPU if the level-1 page table entry is invalid. Starts a Memory Cycle from the Translated Address if the translation is successful. Aborts on protection violations are temporarily suppressed.
Waits for Done pulse from MMU	xxxx	Sends Done pulse on $\overline{SPC}$.
Sends $\overline{SPC}$ pulse and reads Status Word from MMU; places bit 5 of this word into the F bit of the PSR register.	1110	Presents Status Word on bus, indicating in bit 5 the result of the test.

If the translation is successful the MMU will also start a dummy memory cycle from the translated address. See *Figure 2-12*. Note that, during this time the CPU will monitor the RDY line. Therefore, for proper operation, the RDY line must be kept high if the memory cycle is not performed.

The CPU then reads from the MMU a Status Word. Bit 5 of this Status Word indicates the result of the instruction:

0 if the CPU in User Mode could have made the corresponding access to the operand at the specified address (Read in RDVAL, Write in WRVAL),

1 if the CPU would have been aborted for a protection violation.

Bit 5 of the Status Word is placed by the CPU into the F bit of the PSR register, where it can be tested by subsequent instructions as a condition code.

Note: The MMU sets the R bit on RDVAL; R and M bits on WRVAL.

2.6 BUS ACCESS CONTROL

The NS32082 MMU has the capability of relinquishing its access to the bus upon rquest from a DMA device. It does this by using $\overline{HOLD}$, $\overline{HLDAI}$ and $\overline{HLDAO}$.

Details on the interconnections of these pins are provided in *Figures A-1* and *A-2* (Appendix A).

Requests for DMA are presented in parallel to both the CPU and MMU on the $\overline{HOLD}$ pin of each. The component that currently controls the bus then activates its Hold Acknowledge output to grant bus access to the requesting device. When the CPU grants the bus, the MMU passes the CPU's $\overline{HLDA}$ signal to its own $\overline{HLDAO}$ pin. When the MMU grants the bus, it does so by activating its $\overline{HLDAO}$ pin directly, and the CPU is not involved. $\overline{HLDAI}$ in this case is ignored.

Refer to *Figures 4-14, 4-15* and *4-16* for details on bus granting sequences.

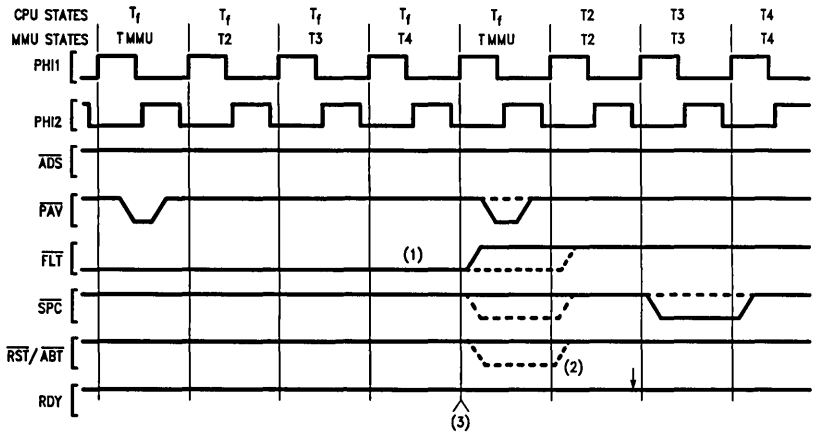
2.7 BREAKPOINTING

The MMU provides the ability to monitor references to two memory locations in real time, generating a Breakpoint trap on occurrence of any specified type of reference to either location made by a program. In addition, a Breakpoint trap may be inhibited until a specified number of such references have been performed.

Breakpoint monitoring is enabled and regulated by the setting of appropriate bits in the MSR and BPR0-1 registers. See Sections 3.5 and 3.7.

A Breakpoint trap is signalled to the CPU as either a Non-Maskable Interrupt or an Abort trap, depending on the setting of the AI bit in the MSR register.

2.0 Functional Description (Continued)



TL/EE/8692-15

Note 1: $\overline{FLT}$ is asserted if the translation is not in the TLB or a WRVAL instruction is executed and the M Bit is not set.

Note 2: If the Level-1 PTE is not valid, an abort is generated, $\overline{SPC}$ is issued in TMMU and $\overline{FLT}$ is deasserted in T_2.

Note 3: If a protection violation occurs or the Level-2 PTE is invalid, an Idle State is inserted here, $\overline{PAV}$ is not pulsed and $\overline{SPC}$ is pulsed during this Idle State.

FIGURE 2-12. $\overline{FLT}$ Deassertion During RDVAL/WRVAL Execution

The MSR register also indicates which breakpoint register triggered the break, and the direction (read or write) and type of memory cycle that was detected. The breakpoint address is not placed into the EIA register, as this register holds the addresses of address translation errors only. The breakpoint address is, however, available in the indicated Breakpoint register.

On occurrence of any trap generated by the MMU, including the Breakpoint trap, the BEN bit in the MSR register is immediately cleared, disabling any further Breakpoint traps.

Enabling breakpoints may cause variations in the bus timing given in the previous sections. Specifically:

1) While either breakpoint is enabled to monitor physical addresses, the MMU inserts an additional clock period into all bus cycles by asserting the $\overline{FLT}$ line for one clock. See *Figure 2-13.*

2) If the CPU initiates an instruction prefetch from a location at which a breakpoint is enabled on Execution, the MMU asserts the $\overline{FLT}$ line to the CPU, performs the memory cycle itself, and issues an edited instruction word to the CPU. See *Figure 2-14* and Section 2.7.1.

Note: Instructions which use two operands, a read-type and a write-type (e.g., MOVD 0(r1).0(r2), with the first operand valid and protected to allow user reads, and the second operand either invalid (page fault) or write protected, cause a read-type break event to occur for the first operand regardless of the outcome of the instruction. Each time the instruction is retried, the read-event is recorded. Hence, the breakpoint count register may reflect a different count than a casual assumption would lead one to. The same effect can occur on a RMW type operand with read only protection.

2.7.1 Breakpoints on Execution

The Series 32000 CPUs have an instruction prefetch which requires synchronization with execution breakpoints. In consideration of this, the MMU only issues an execution breakpoint when an instruction is prefetched with a nonsequential status code and the conditions specified in a breakpoint register are met. This guarantees that the instruction prefetch queue is empty and there are not pending instructions in the pipeline. There are three cases to consider:

Case 1: A nonsequential instruction prefetch is made to a breakpointed address.

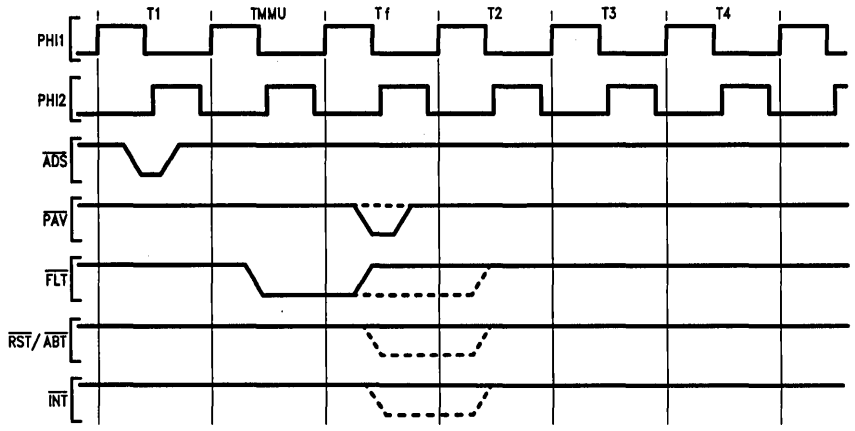
Response: The queue is necessarily empty. The breakpoint is issued.

Case 2, 3: A sequential prefetch is made to a breakpointed address OR a prefetch is made to an even address and the breakpoint is on the next odd address.

Response: In these cases, there may be instructions pending in the queue which must finish before the breakpoint is fired. Instead of putting the opcode byte (the one specified by the breakpointed address) in the queue, a DIA instruction is substituted for it. DIA is a single byte instruction which branches to itself, causing a queue flush. When the DIA executes, the breakpoint address is again issued, this time with nonsequential fetch status and the problem is reduced to case 1.

Note: Execution breakpoints cannot be used when the MMU is connected to either an NS32032 or an NS32332 CPU.

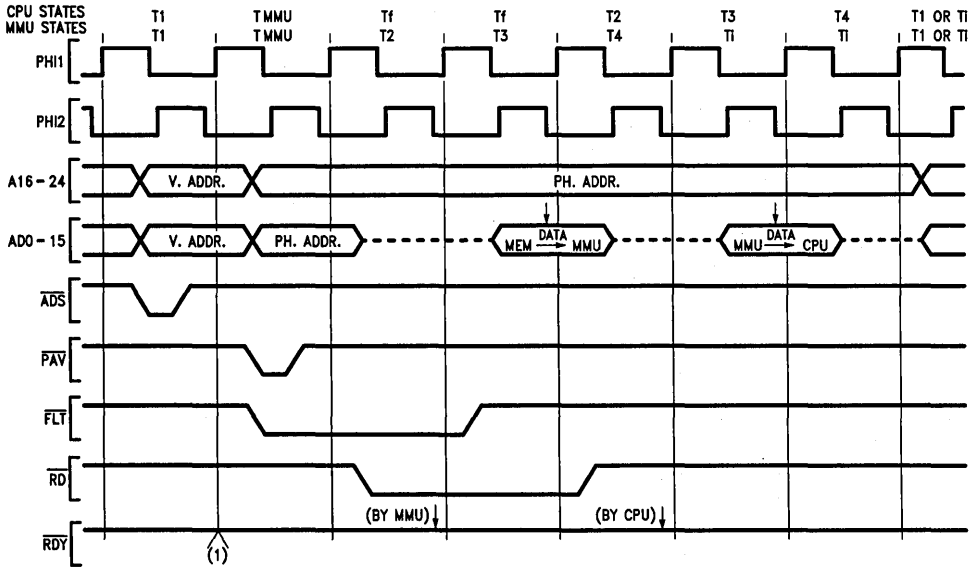
2.0 Functional Description (Continued)



TL/EE/8692-16

Note: If a breakpoint condition is met and abort on breakpoint is enabled, the bus cycle is aborted. In this case FLT is stretched by one clock cycle.

FIGURE 2-13. Bus Timing with Breakpoint on Physical Address Enabled



TL/EE/8692-17

Note 1: If a breakpoint on physical address is enabled, an extra clock cycle is inserted here.

FIGURE 2-14. Execution Breakpoint Timing; Insertion of DIA Instruction

3.0 Architectural Description

3.1 PROGRAMMING MODEL

The MMU contains a set of registers through which the CPU controls and monitors management and debugging functions. These registers are not memory-mapped. They are examined and modified by executing the Slave Processor instructions LMR (Load Memory Management Register) and SMR (Store Memory Management Register). These instructions are explained in Section 3.11, along with the other Slave Processor instructions executed by the MMU.

A brief description of the MMU registers is provided below. Details on their formats and functions are provided in the following sections.

PTB0, PTB1—Page Table Base Registers. They hold the physical memory addresses of the Page Tables referenced by the MMU for address translation. See Section 3.3.

EIA—Error/Invalidate Register. Dual-function register, used to display error addresses and also to purge cached translation information from the TLB. See Section 3.4.

BPR0, BPR1—Breakpoint Registers. Specify the conditions under which a breakpoint trap is generated. See Section 3.5.

BCNT—Breakpoint Counter Register. 24-bit counter used to count BPR0 events. Allows the breakpoint trap from the BPR0 register to be inhibited until a specified number of events have occurred. See Section 3.6.

MSR—Memory Management Status Register. Contains basic control and status fields for all MMU functions. See Section 3.7.

3.2 MEMORY MANAGEMENT FUNCTIONS

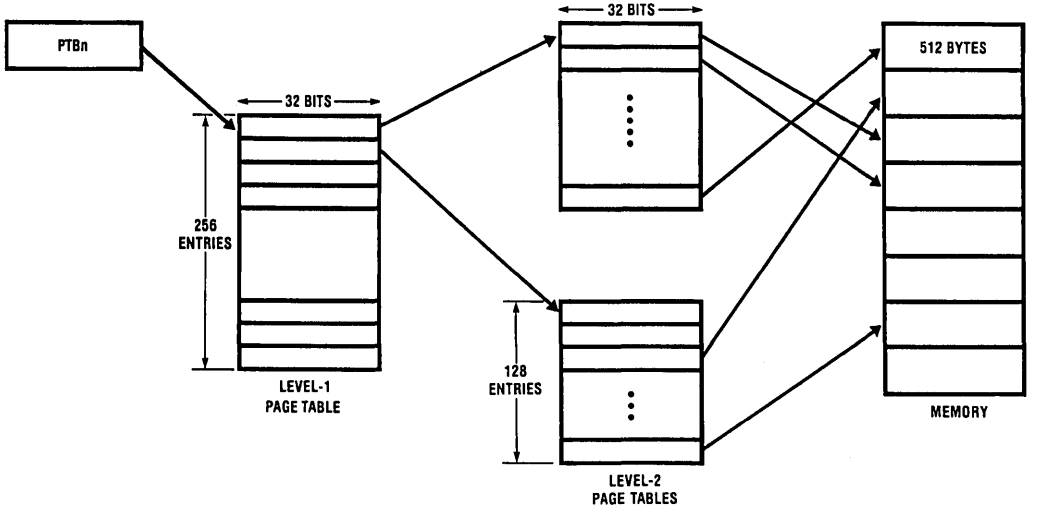
The NS32082 uses sets of tables in physical memory (the "Page Tables") to define the mapping from virtual to physical addresses. These tables are found by the MMU using one of its two Page Table Base registers: PTB0 or PTB1. Which register is used depends on the currently selected address space. See Section 3.2.2.

3.2.1. Page Table Structure

The page tables are arranged in a two-level structure, as shown in *Figure 3-1*. Each of the MMU's PTBn registers may point to a Level-1 page table. Each entry of the Level-1 page table may in turn point to a Level-2 page table. Each Level-2 page table entry contains translation information for one page of the virtual space.

The Level-1 page table must remain in physical memory while the PTBn register contains its address and translation is enabled. Level-2 Page Tables need not reside in physical memory permanently, but may be swapped into physical memory on demand as is done with the pages of the virtual space.

The Level-1 Page Table contains 256 32-bit Page Table Entries (PTE'S) and therefore occupies 1 Kbyte. Each entry of the Level-1 Page Table contains fields used to construct the physical base address of a Level-2 Page Table. These fields are a 15-bit PFN field, providing bits 9-23 of the physical address, and an MS bit providing bit 24. The remaining bits (0–8) are assumed zero, placing a Level-2 Page Table always on a 512-byte (page) boundary.



TL/EE/8692–18

FIGURE 3-1. Two-Level Page Tables

3.0 Architectural Description (Continued)

Level-2 Page Tables contain 128 32-bit Page Table entries, and so occupy 512 bytes (1 page). Each Level-2 Page Table Entry points to a final 512-byte physical page frame. In other words, its PFN and MS fields provide the Page Frame Number portion (bits 9-24) of the translated address (*Figure 3-3*). The OFFSET field of the translated address is taken directly from the corresponding field of the virtual address.

3.2.2 Virtual Address Spaces

When the Dual Space option is selected for address translation in the MSR (Sec. 3.7) the MMU uses two maps: one for translating addresses presented to it in Supervisor Mode and another for User Mode addresses. Each map is referenced by the MMU using one of the two Page Table Base registers: PTB0 or PTB1. The MMU determines the CPU's current mode by monitoring the state of the U/$\overline{S}$ pin and applying the following rules.

1) While the CPU is in Supervisor Mode (U/$\overline{S}$ pin = 0), the CPU is said to be presenting addresses belonging to Address Space 0, and the MMU uses the PTB0 register as its reference for looking up translations from memory.
2) While the CPU is in User Mode (U/$\overline{S}$ pin = 1), and the MSR DS bit is set to enable Dual Space translation, the CPU is said to be presenting addresses belonging to Address Space 1, and the MMU uses the PTB1 register to look up translations.
3) If Dual Space translation is not selected in the MSR, there is no Address Space 1, and all addresses presented in both Supervisor and User modes are considered by the MMU to be in Address Space 0. The privilege level of the CPU is used then only for access level checking.

Note: When the CPU executes a Dual-Space Move instruction (MOVUSi or MOVSUi), it temporarily enters User Mode by switching the state of the U/$\overline{S}$ pin. Accesses made by the CPU during this time are treated by the MMU as User-Mode accesses for both mapping and access level checking. It is possible, however, to force the MMU to assume Supervisor-Mode privilege on such accesses by setting the Access Override (AO) bit in the MSR (Sec. 3.7).

3.2.3 Page Table Entry Formats

Figure 3-2 shows the formats of Level-1 and Level-2 Page Table Entries (PTE's). Their formats are identical except for the "M" bit, which appears only in a Level-2 PTE.

The bits are defined as follows:

V Valid. The V bit is set and cleared only by software.

V=1=> The PTE is valid and may be used for translation by the MMU.

V=0=> The PTE does not represent a valid translation. Any attempt to use this PTE will cause the MMU to generate an Abort trap. While V=0, the operating system may use all other bits except the PL field for any desired function.

PL Protection Level. This two-bit field establishes the types of accesses permitted for the page in both User Mode and Supervisor Mode, as shown in Table 3-1.

The PL field is modified only by software. In a Level-1 PTE, it limits the maximum access level allowed for all pages mapped through that PTE.

TABLE 3-1. Access Protection Levels

Mode	U/$\overline{S}$	Protection Level Bits (PL)			
		00	01	10	11
User	1	no access	no access	read only	full access
Supervisor	0	read only	full access	full access	full access

R Referenced. This is a status bit, set by the MMU and cleared by the operating system, that indicates whether the page mapped by this PTE has been referenced within a period of time determined by the operating system. It is intended to assist in implementing memory allocation strategies. In a Level-1 PTE, the R bit indicates only that the Level-2 Page Table has been referenced for a translation, without necessarily implying that the translation was successful. In a Level-2 PTE, it indicates that the page mapped by the PTE has been successfully referenced.

R=1=> The page has been referenced since the R bit was last cleared.

R=0=> The page has not been referenced since the R bit was last cleared.

Note: The RDVAL and WRVAL instructions set the Level-1 and Level-2 bits for the page whose protection level is tested. See Sections 2.5.2 and 3.11.

M Modified. This is a status bit, set by the MMU whenever a write cycle is successfully performed to the page mapped by this PTE. It is initialized to zero by the operating system when the page is brought into physical memory.

M=1=> The page has been modified since it was last brought into physical memory.

M=0=> The page has not been modified since it was last brought into physical memory.

In Level-1 Page Table Entries, this bit position is undefined, and is altered in an undefined manner by the MMU while the V bit is 1.

Note: The WRVAL instruction sets the M bit for the page whose protection level is tested. See Sections 2.5.2 and 3.11.

NSC Reserved. These bits are ignored by the MMU and their values are not changed.

They are reserved by National, and therefore should not be used by the user software.

USR User bits. These bits are ignored by the MMU and their values are not changed.

They can be used by the user software.

TL/EE/8692-19

FIGURE 3-2. A Page Table Entry

3.0 Architectural Description (Continued)

PFN Page Frame Number. This 15-bit field provides bits 9-23 of the Page Frame Number of the physical address. See *Figure 3-3*.

MS Memory System. This bit represents the most significant bit of the physical address, and is presented by the MMU on pin A24. This bit is treated by the MMU no differently than any other physical address bit, and can be used to implement a 32-Mbyte physical addressing space if desired.

3.2.4 Physical Address Generation

When a virtual address is presented to the MMU by the CPU and the translation information is not in the TLB, the MMU performs a page table lookup in order to generate the physical address.

The Page Table structure is traversed by the MMU using fields taken from the virtual address. This sequence is diagrammed in *Figure 3-3*.

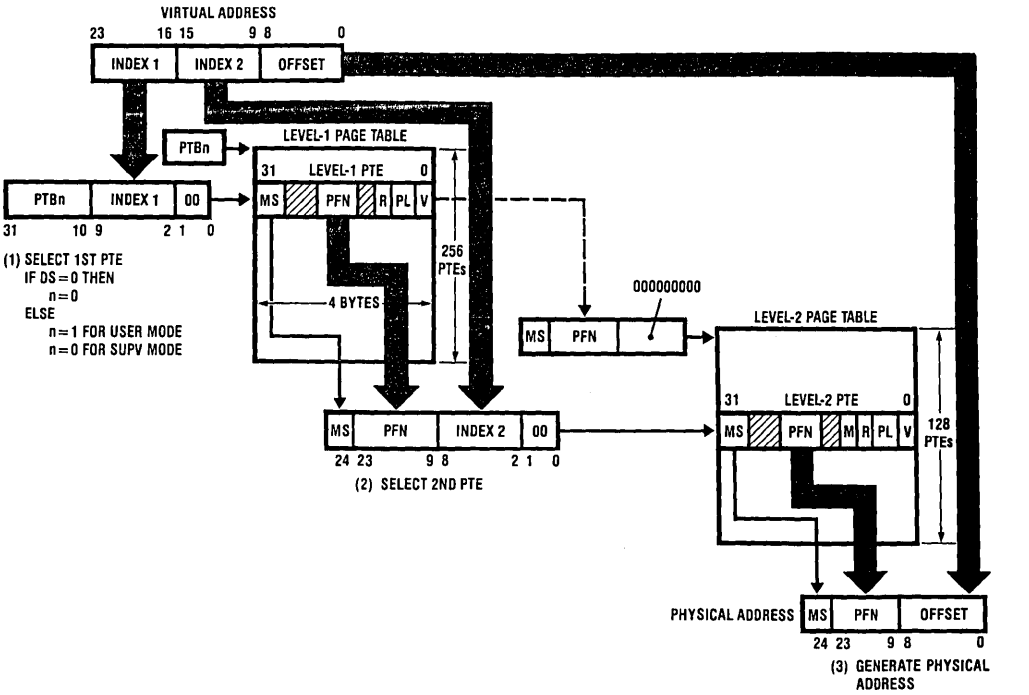
Bits 9–23 of the virtual address hold the 15-bit Page Number, which in the course of the translation is replaced with the 16-bit Page Frame Number of the physical address. The virtual Page Number field is further divided into two fields, INDEX 1 and INDEX 2.

Bits 0–8 constitute the OFFSET field, which identifies a byte's position within the accessed page. Since the byte position within a page does not change with translation, this value is not used, and is simply echoed by the MMU as bits 0–8 of the final physical address.

The 8-bit INDEX 1 field of the virtual address is used as an index into the Level-1 Page Table, selecting one of its 256 entries. The address of the entry is computed by adding INDEX 1 (scaled by 4) to the contents of the current Page Table Base register. The PFN and MS fields of that entry give the base address of the selected Level-2 Page Table.

The INDEX 2 field of the virtual address (7 bits) is used as the index into the Level-2 Page Table, by adding it (scaled by 4) to the base address taken from the Level-1 Page Table Entry. The PFN and MS fields of the selected entry provide the entire Page Frame Number of the translated address.

The offset field of the virtual address is then appended to this frame number to generate the final physical address.



TL/EE/8692-20

FIGURE 3-3. Virtual to Physical Address Translation

3.0 Architectural Description (Continued)

3.3 PAGE TABLE BASE REGISTERS (PTB0, PTB1)

The PTBn registers hold the physical addresses of the Level-1 Page Tables.

The format of these registers is shown in *Figure 3-4.* The least-significant 10 bits are permanently zero, so that each register always points to a 1 Kbyte boundary in memory.

The PTBn registers may be loaded or stored using the MMU Slave Processor instructions LMR and SMR (Section 3.11).

3.4 ERROR/INVALIDATE ADDRESS REGISTER (EIA)

The Error/Invalidate Address register is a dual-purpose register.

1) When it is read using the SMR instruction, it presents the virtual address which last generated an address translation error.
2) When a virtual address is written into it using the LMR instruction, the translation for that virtual address is purged, if present, from the TLB. This must be done whenever a Page Table Entry has been changed in memory, since the TLB might otherwise contain an incorrect translation value.

The format of the EIA register is shown in *Figure 3-5.* When a translation error occurs, the cause of the error is reported by the MMU in the appropriate fields of the MSR register (Section 3.7). The ADDRESS field of the EIA register holds the virtual address at which the error occurred, and the AS bit indicates the address space that was in use.

In writing a virtual address to the EIA register, the virtual address is specified in the low-order 24 bits, and the AS bit specifies the address space. A TLB entry is purged only if it matches both the ADDRESS and AS fields.

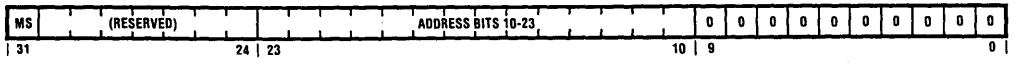
Another technique for purging TLB entries is to load a PTBn register. This automatically purges all entries associated with the addressing space mapped by that register. Turning off translation (clearing the MSR TU and/or TS bits) does not purge any entries from the TLB.

3.5 BREAKPOINT REGISTERS (BPR0, BPR1)

The Breakpoint registers BPR0 and BPR1 specify the addresses and conditions on which a Breakpoint trap will be generated. They are each 32 bits in length and have the format shown in *Figure 3-6.* All implemented bits of BPR0 and BPR1 are readable and writable.

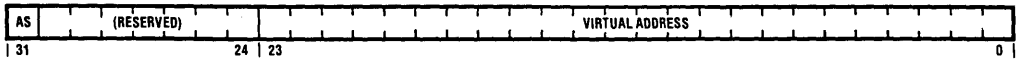
Bits 0 through 23 and bit 31 (AS) specify the breakpoint address. This address may be either virtual or physical, as specified in the VP bit.

Bits 24 and 25 are not implemented. Bit 26 (CE) is not implemented in register BPR1.



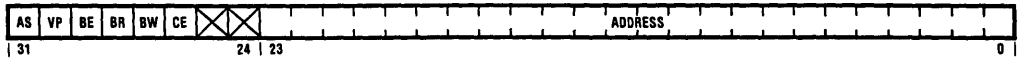
TL/EE/8692-21

FIGURE 3-4. Page Table Base Registers (PTB0, PTB1)



TL/EE/8692-22

FIGURE 3-5. EIA Register



TL/EE/8692-23

FIGURE 3-6. Breakpoint Registers (BPR0, BPR1)

3.0 Architectural Description (Continued)

Bits 26 through 30 specify the breakpoint conditions. Breakpoint conditions define how the breakpoint address is compared and which conditions permit a break to be generated. A Breakpoint register can be selectively disabled by setting all of these bits to zero.

AS Address Space. This bit depends on the setting of the VP bit. For virtual addresses, this bit contains the AS (Address Space) qualifier of the virtual address (Section 3.2.2). For physical addresses, this bit contains the MS (Memory System) bit of the physical address.

VP Virtual/Physical. If VP is 0, the breakpoint address is compared against each referenced virtual address. If VP is 1, the breakpoint address is compared against each physical address that is referenced by the CPU (i.e. after translation).

BE Break on Execution. If BE is 1, a break is generated immediately before the instruction at the breakpoint address is executed. While this option is enabled, the breakpoint address must be the address of the first byte of an instruction. If BE is 0, this condition is disabled.

Note: This option cannot be used in systems based on any CPU with a 32-bit wide bus.

The BE bit should only be set when the CPU has a 16-bit bus (i.e. NS32016, NS32C016). In other systems, use instead the BPT instruction placed in memory, to signal a break.

BR Break on Read. If BR is 1, a break is generated when data is read from the breakpoint address. Instruction fetches do not trigger a Read breakpoint. If BR is 0, this condition is disabled.

BW Break on Write. If BW is 1, a break is generated when data is written to the breakpoint address or when data is read from the breakpoint address as the first part of a read-modify-write access. If BW is 0, this condition is disabled.

CE Counter Enable. This bit is implemented only in the BPR0 register. If CE is 1, no break is generated unless the Breakpoint Count register (BCNT, see below) is zero. The BCNT register decrements when the condition for the breakpoint in register BPR0 is met and the BCNT register is not already zero. If CE is 0, the BCNT register is disabled, and breaks from BPR0 occur immediately.

Note 1: The bits BR, BW and CE should not all be set. The counting performed by the MMU becomes inaccurate, and in Abort Mode (MSR AI bit set), it can trap a program in such a way as to make it impossible to retry the breakpointed instruction correctly.

Note 2: An execution breakpoint should not be counted (BE and CE bits both set) if it is placed at an address that is the destination of a branch, or if it follows a queue-flushing instruction. See Table 3-2. The counting performed by the MMU will be inaccurate if interrupts occur during the fetch of that address.

TABLE 3-2. Instructions Causing Non-Sequential Fetches

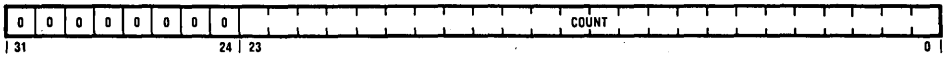
Branch	
ACBi	Add, Compare and Branch: unless result is zero
BR	Branch (Unconditional)
BSR	Branch to Subroutine
Bcond	Branch (Conditional): only if condition is met
CASEi	Case Branch
CXP	Call External Procedure
CXPD	Call External Procedure with Descriptor
DIA	Diagnose
JSR	Jump to Subroutine
JUMP	Jump
RET	Return from Subroutine
RXP	Return from External Procedure
BPT	Breakpoint Trap
FLAG	Trap on Flag
RETI	Return from Interrupt: if MSR loaded properly by supervisor
RETT	Return from Trap: if MSR loaded properly by supervisor
SVC	Supervisor Call

Also all traps or interrupts not generated by the MMU.

Branch to Following Instruction	
BICPSRi	Bit Clear in PSR
BISPSRi	Bit Set in PSR
LMR	Load Memory Management Register
LPRi	Load Processor Register: unless UPSR is the register specified
MOVSUi	Move Value from Supervisor to User Space
MOVUSi	Move Value from User to Supervisor Space
WAIT	Wait: fetches next instruction before waiting

3.6 BREAKPOINT COUNT REGISTER (BCNT)

The Breakpoint Count register (BCNT) permits the user to specify the number of breakpoint conditions given by register BPR0 that should be ignored before generating a Breakpoint trap. The BCNT register is 32 bits in length, containing a counter in its low-order 24 bits, as shown in *Figure 3-7.* The high-order eight bits are not used.



TL/EE/8692-24

FIGURE 3-7. Breakpoint Count Register (BCNT)

3.0 Architectural Description (Continued)

The BCNT register affects the generation of Breakpoint traps only when it is enabled by the CE bit in the BPR0 register. When the BPR0 breakpoint condition is encountered, and the BPR0 CE bit is 1, the contents of the BCNT register are checked against zero. If the BCNT contents are zero, a breakpoint trap is generated. If the contents are not equal to zero, no breakpoint trap is generated and the BCNT register is decremented by 1.

If the CE bit in the BPR0 register is 0, the BCNT register is ignored and the BPR0 condition breaks the program execution regardless of the BCNT register's contents. The BCNT register contents are unaffected.

3.7 MEMORY MANAGEMENT STATUS REGISTER (MSR)

The Memory Management Status Register (MSR) provides overall control and status fields for both address translation and debugging functions. The format of the MSR register is shown in *Figure 3-8*.

The MSR fields relevant to either of the above functions are described in the following sub-sections.

3.7.1 MSR Fields for Address Translation.

Control Functions

The address translation control bits in the MSR, ad exception of the R bit, are both readable (using the SMR instruction) and writable (using LMR).

R Reset. When read, this bit's contents are undefined. Whenever a "1" is written into it, MSR status fields TE, B, TET, ED, BD, EST and BST are cleared to all zeroes. (The BN bit is not affected.)

TU Translate User-Mode Addresses. While this bit is "1", the MMU translates all addresses presented while the CPU is in User Mode. While it is "0", the MMU echoes all User-Mode virtual addresses without performing translation or access level checking. This bit is cleared by a hardware Reset.

Note: Altering the TU bit has no effect on the contents of the TLB.

TS Translate Supervisor-Mode Addresses. While this bit is "1", the MMU translates all addresses presented while the CPU is in Supervisor Mode. While it is "0", the MMU echoes all Supervisor-Mode virtual addresses without translation or access level checking. This bit is cleared by a hardware Reset.

Note: Altering the TS bit has no effect on the contents of the TLB.

DS Dual-Space Translation. While this bit is "1", Supervisor Mode addresses and User Mode addresses are translated independently of each other, using separate mappings. While it is "0", both Supervisor Mode addresses and User Mode addresses are translated using the same mapping. See Section 3.2.2.

AO Access Level Override. This bit may be set to temporarily cause User Mode accesses to be given Supervisor Mode privilege. See Section 3.10.

Status Fields

The MSR status fields may be read using the MSR instruction, but are not writable. Instead, all status fields (except the BN bit) may be cleared by loading a "1" into the R bit using the LMR instruction.

TE Translation Error. This bit is set by the MMU to indicate that an address translation error has occurred. This bit is cleared by a hardware reset.

TET Translation Error Type. This three-bit field shows the reason(s) for the last address translation error reported by the MMU. The format of the TET field is shown below.

IL2	IL1	PL

PL Protection Level error. The access attempted by the CPU was not allowed by the protection level assigned to the page it attempted to access (forbidden by either of the Page Table Entry PL fields).

IL1 Invalid Level 1. The Level-1 Page Table Entry was invalid (V bit = 0).

IL2 Invalid Level 2. The Level-2 Page Table Entry was invalid (V bit = 0).

These error indications are not mutually exclusive. A protection level error and an invalid translation error can be reported simultaneously by the MMU.

ED Error Direction. This bit indicates the direction of the transfer that the CPU was attempting on the most recent address translation error.

ED=0=>Write cycle.

ED=1=>Read cycle.

EST Error Status. This 3-bit field is set on an address translation error to the low-order three bits of the CPU status bus. Combinations appearing in this field are summarized below.

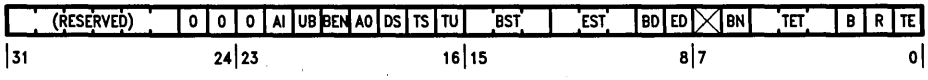
000 Sequential instruction fetch

001 Non-sequential instruction fetch

010 Operand transfer (read or write)

011 The Read action of a read-modify-write transfer (operands of access class "rmw" only: See the Series 32000 Instruction Set Reference Manual for further details).

100 A read transfer which is part of an effective address calculation (Memory Relative or External mode)



TL/EE/8692-25

Note: In some Series 32000 documentation, the bits TE, R and B are jointly referenced with the keyword "ERC".

FIGURE 3-8. Memory Management Status Register (MSR)

3.0 Architectural Description (Continued)

3.7.2 MSR Fields for Debugging

Control Functions

Breakpoint control bits in the MSR are both readable (using the SMR instruction) and writable (using LMR).

BEN Breakpoint Enable. Setting this bit enables both Breakpoint Registers (BPR0, BPR1) to monitor CPU activity. This bit is cleared by a hardware reset or whenever a Breakpoint trap or an address translation error occurs. If only one breakpoint register must be enabled, the other register should be disabled by clearing all of its control bits (bits 26–31) to zeroes.

Note: When the BEN bit is set (using the LMR instruction), the MMU enables breakpoints only after two non-sequential instruction fetch cycles have been completed by the CPU. See Section 3.9.

UB User-Only Breakpointing. When this bit is set in conjunction with the BEN bit, it limits the Breakpoint Registers to monitor addresses only while the CPU is in User Mode.

AI Abort/Interrupt. This bit selects the action taken by the MMU on a breakpoint. While AI is "0" the MMU generates a pulse on the $\overline{INT}$ pin (this can be used to generate a non-maskable interrupt). While AI is "1" the MMU generates an Abort pulse instead.

Status Fields

The MSR status fields may be read using the SMR instruction, but are not writable. Instead, all status fields (except the BN bit) may be cleared by loading a "1" into the R bit using the LMR instruction. See Section 3.7.1.

B Break. This bit is set to indicate that a breakpoint trap has been generated by the MMU.

BN Breakpoint Number. The BN bit contains the register number for the most recent breakpoint trap generated by the MMU. If BN is 1, the breakpoint was triggered by the BPR1 register. If BN is 0, the breakpoint was triggered by the BPR0 register. If both registers trigger a breakpoint simultaneously, the BN bit is set to 1.

BD Break Direction. This bit indicates the direction of the transfer that the CPU was attempting on the access that triggered the most recent breakpoint trap. It is loaded from the complement of the $\overline{DDIN}$ pin.

BD = 0 => Write cycle.

BD = 1 => Read cycle.

BST Breakpoint Status. This 3-bit field is loaded on a Breakpoint trap from the low-order three bits of the CPU status bus. Combinations appearing in this field are summarized below.

000 No break has occurred since the field was last reset.

001 Instruction fetch

010 Operand transfer (read or write)

011 The Read action of a read-modify-write transfer (operands of access class "rmw" only: See the Series 32000 Instruction Set Reference Manual for further details).

100 A read transfer which is part of an effective address calculation (Memory Relative or External mode)

Note: The BST field encodings 000 and 001 differ from those of the EST field (Section 3.7.1) because the MMU inserts a DIA instruction into the instruction stream in implementing Execution breakpoints (Section 2.7.1). One side effect of this is that a breakpoint trap is never triggered directly by a sequential instruction fetch cycle.

3.8 TRANSLATION LOOKASIDE BUFFER (TLB)

The Translation Lookaside Buffer is an on-chip fully associative memory. It provides direct virtual to physical mapping for the 32 most recently used pages, requiring only one clock period to perform the address translation.

The efficiency of the MMU is greatly increased by the TLB, which bypasses the much longer Page Table lookup in over 97% of the accesses made by the CPU.

Entries in the TLB are allocated and replaced by the MMU itself; the operating system is not involved. The TLB entries cannot be read or written by software; however, they can be purged from it under program control.

Figure 3-9 models the TLB. Information is placed into the TLB whenever the MMU performs a lookup from the Page Tables in memory. If the retrieved mapping is valid (V = 1 in both levels of the Page Tables), and the access attempted is permitted by the protection level, an entry of the TLB is loaded from the information retrieved from memory. The recipient entry is selected by an on-chip circuit that implements a Least-Recently-Used (LRU) algorithm. The MMU places the virtual page number (15 bits) and the Address Space qualifier bit into the Tag field of the TLB entry.

The Value portion of the entry is loaded from the Page Tables as follows:

The Translation field (16 bits) is loaded from the MS bit and PFN field of the Level-2 Page Table Entry.

The M bit is loaded from the Level-2 Page Table Entry.

The PL field (2 bits) is loaded to reflect the net protection level imposed by the PL fields of the Level-1 and Level-2 Page Table Entries.

(Not shown in the figure are additional bits associated with each TLB entry which flag it as full or empty, and which select it as the recipient when a Page Table lookup is performed.)

When a virtual address is presented to the MMU for translation, the high-order 15 bits (page number) and the Address Space qualifier are compared associatively to the corresponding fields in all entries of the TLB. When the Tag portion of a TLB entry completely matches the input values, the Value portion is produced as output. If the protection level is not violated, and the M bit does not need to be changed, then the physical address Page Frame number is output in the next clock cycle. If the protection level is violated, the MMU instead activates the Abort output. If no TLB entry matches, or if the matching entry's M bit needs to be changed, the MMU performs a page-table lookup from memory.

Note that for a translation to be loaded into the TLB it is necessary that the Level-1 and Level-2 Page Table Entries be valid (V bit = 1). Also, it is guaranteed that in

3.0 Architectural Description (Continued)

the process of loading a TLB entry (during a Page Table lookup) the Level-1 and Level-2 R bits will be set in memory if they were not already set. For these reasons, there is no need to replicate either the V bit or the R bit in the TLB entries.

Whenever a Page Table Entry in memory is altered by software, it is necessary to purge any matching entry from the TLB, otherwise the MMU would be translating the corresponding addresses according to obsolete information. TLB entries may be selectively purged by writing a virtual address to the EIA register using the LMR instruction. The TLB entry (if any) that matches that virtual address is then purged, and its space is made available for another translation. Purging is also performed by the MMU whenever an address space is remapped by altering the contents of the PTB0 or PTB1 register. When this is done, the MMU purges all the TLB entries corresponding to the address space mapped by that register. Turning translation on or off (via the MSR TU and TS bits) does not affect the contents of the TLB.

Note: If the value in the PTB0 register must be changed, it is strongly recommended that the translation be disabled before loading the new value, otherwise the purge performed may be incomplete. This is due to instruction prefetches and/or memory read cycles occurring during the LMR instruction which may restore TLB entries from the old map.

3.9 ENTRY/RE-ENTRY INTO PROGRAMS UNDER DEBUGGING

Whenever the MSR is written, breakpoints are disabled. After two non-sequential instruction fetch cycles have completed, they are again enabled if the new BEN bit value is '1'. The recommended sequence for entering a program under test is:

```
LMR    MSR, New_Value
RETT   n        ; or RETI
```

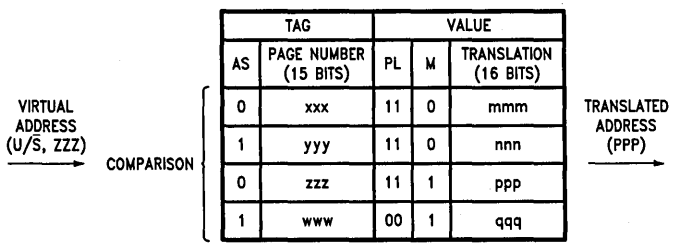
executed with interrupts disabled (CPU PSR I bit off).

This feature allows a debugger or monitor program to return control to a program being debugged without the risk of a false breakpoint trap being triggered during the return.

The LMR instruction performs the first non-sequential fetch cycle, in effect branching to the next sequential instruction. The RETT (or RETI) instruction performs the second non-sequential fetch as its last memory reference, branching to the first (next) instruction of the program under debug. The non-sequential fetch caused by the RETT instruction, which might not have occurred otherwise, is not monitored.

3.10 ADDRESS TRANSLATION ALGORITHM

The MMU either translates the 24-bit virtual address to a 25-bit physical address or reports a translation error. This process is described algorithmically in the following pages. See also *Figure 3-3*.



TAG		VALUE		
AS	PAGE NUMBER (15 BITS)	PL	M	TRANSLATION (16 BITS)
0	xxx	11	0	mmm
1	yyy	11	0	nnn
0	zzz	11	1	ppp
1	www	00	1	qqq

TL/EE/8692-26

FIGURE 3-9. TLB Model

MMU Page Table Lookup and Access Validation Algorithm

```
Legend:
x = y           x is assigned the value y
x == y          Comparison expression, true if x is equal to y
x AND y         Boolean AND expression, true only if assertions x and y are both true
x OR y          Boolean inclusive OR expression, true if either of assertions x and y is true
;               Delimiter marking end of statement
{ . . . }       Delimiters enclosing a statement block
item(i)         Bit number i of structure "item"
item(i:j)       The field from bit number i through bit number j of structure "item"
item.x          The bit or field named "x" in structure "item"
DONE            Successful end of translation; MMU provides translated address
ABORT           Unsuccessful end of translation; MMU aborts CPU access
This algorithm represents for all cases a valid definition of address translation.
Bus activity implied here occurs only if the TLB does not contain the mapping,
or if the reference requires that the MMU alter the M bit of the Page Table Entry.
Otherwise, the MMU provides the translated address in one clock period.
Input (from CPU):
     U (1 if U/S is high)
     W (1 if DDIN input is high)
     VA  Virtual address consisting of:
        INDEX_1 (from pins A23-A16)
        INDEX_2 (from pins AD15-AD9)
        OFFSET  (from pins AD8-AD0)
     ACCESS_LEVEL       The access level of a reference is a 2-bit value synthesized by the MMU from CPU status:
        bit 1 = U AND NOT MSR.A0  (U from U/S input pin)
        bit 0 = 1 for Write cycle, or Read cycle of an "rmw" class operand access
                0 otherwise.

Output:
     PA Physical Address on pins A24-A16, AD15-AD0;
        or
     Abort pulse on RST/ABT pin.
Uses:
     MSR        Status Register:
                        fields TU, TS and DS
```

MMU Page Table Lookup and Access Validation Algorithm (Continued)

```
     PTB0       Page Table Base Register 0
     PTB1       Page Table Base Register 1
     PTE_1      Level-1 Page Table Entry:
                        fields PFN, PL, V, R and MS
     PTEP_1     Pointer, holding address of PTE_1
     PTE_2      Level-2 Page Table Entry:
                        fields PFN, PL, V, M, R and MS
     PTEP_2     Pointer, holding address of PTE_2
IF ( (MSR.TU == 0) AND (U == 1) )  OR  ( (MSR.TS == 0) AND (U ==0) )  If translation not enabled then echo
     THEN  {  PA(0:23) = VA(0:23) ; PA(24) = 0 ; DONE  }  ;                  virtual address as physical address.

IF (MSR.DS == 1) AND (U == 1)                                          If Dual Space mode and CPU in User Mode
   THEN {  PTEP_1(24) = PTB1.MS ; PTEP_1(23:10) = PTB1(23:10) ;             then form Level-1 PTE address
           PTEP_1(9:2) = VA.INDEX_1 ; PTEP_1(1:0)=0  }                             from PTB1 register,
   ELSE {  PTEP_1(24) = PTB0.MS ; PTEP_1(23:10) = PTB0(23:10) ;             else form Level-1 PTE address
           PTEP_1(9:2) = VA.INDEX_1; PTEP_1(1:0) = 0                               from PTB0 register.
        } ;

                                              - - - LEVEL 1 PAGE TABLE LOOKUP - - -

IF ( ACCESS_LEVEL > PTE_1.PL ) OR (PTE_1.V == 0)                       If protection violation or invalid Level-2 page
   THEN ABORT ;                                                            table then abort the access.

IF PTE_1.R == 0 THEN PTE_1.R = 1 ;                                     Otherwise, set Reference bit if not already set,
PTE_1(4) = (undefined value);                                                  (the M bit position may be garbaged)

PTEP_2(24) = PTE_1.MS ; PTEP_2(23:9) = PTE_1.PFN ;                             and form Level-2 PTE address.
PTEP_2(8:2) = VA.INDEX_2 ; PTEP_2(1:0) = 0 ;

                                              - - - LEVEL 2 PAGE TABLE LOOKUP - - -

IF ( ACCESS_LEVEL > PTE_2.PL ) OR ( PTE_2.V ==0 )                            If protection violation or invalid page
   THEN ABORT ;                                                              then abort the access.

IF PTE_2.R == 0 THEN PTE_2.R == 1 ;                                    Otherwise, set Referenced bit if not already set,
IF ( W == 1) AND ( PTE_2.M == 0 ) THEN PTE_2.M = 1 ;                         if Write cycle set Modified bit if not
                                                                                already set,
PA(24) = PTE_2.MS ; PA(23:9) = PTE_2.PFN ; PA(8:0) = VA.OFFSET ;                 and generate physical address.
DONE ;
```

3.0 Architectural Description (Continued)

3.11 INSTRUCTION SET

Four instructions of the Series 32000 instruction set are executed cooperatively by the CPU and MMU. These are:

LMR Load Memory Management Register
SMR Store Memory Management Register

RDVAL Validate Address for Reading
WRVAL Validate Address for Writing

The format of the MMU slave instructions is shown in *Figure 3-10*. Table 3-3 shows the encodings of the "short" field for selecting the various MMU internal registers.

TABLE 3-3. "Short" Field Encodings

"Short" Field	Register
0000	BPR0
0001	BPR1
1010	MSR
1011	BCNT
1100	PTB0
1101	PTB1
1111	EIA

Note: All other codes are illegal. They will cause unpredictable registers to be selected if used in an instruction.

For reasons of system security, all MMU instructions are privileged, and the CPU does not issue them to the MMU in User Mode. Any such attempt made by a User-Mode program generates the Illegal Operation trap, Trap (ILL). In addition, the CPU will not issue MMU instructions unless its CFG register's M bit has been set to validate the MMU instruction set. If this has not been done, MMU instructions are not recognized by the CPU, and an Undefined Instruction trap, Trap (UND), results.

The LMR and SMR instructions load and store MMU registers as 32-bit quantities to and from any general operand (including CPU General-Purpose Registers).

The RDVAL and WRVAL instructions probe a memory address and determine whether its current protection level would allow reading or writing, respectively, if the CPU were in User Mode. Instead of triggering an Abort trap, these instructions have the effect of setting the CPU PSR F bit if the type of access being tested for would be illegal. The PSR F bit can then be tested as a condition code.

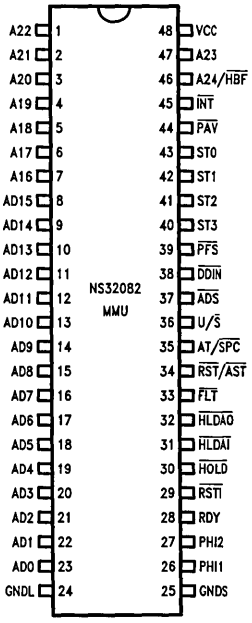
Note: The Series 32000 Dual-Space Move instructions (MOVSUi and MOVUSi), although they involve memory management action, are not Slave Processor instructions. The CPU implements them by switching the state of its U/$\overline{S}$ pin at appropriate times to select the desired mapping and protection from the MMU.

For full architectural details of these instructions, see the Series 32000 Instruction Set Reference Manual.

4.0 Device Specifications

4.1 NS32082 PIN DESCRIPTIONS

The following is a brief description of all NS32082 pins. The descriptions reference portions of the Functional Description, Section 2.0.



TL/EE/8692-28

Top View

Order Number NS16082D
See NS Package Number D48A

FIGURE 4-1. Dual-In-Line Package Connection Diagram

4.1.1 Supplies

Power (V_{CC}): +5V positive supply. Section 2.1.

Logic Ground (GNDL): Ground reference for on-chip logic. Section 2.1.

Buffer Ground (GNDB): Ground reference for on-chip drivers connected to output pins. Section 2.1.

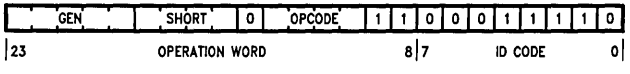
4.1.2 Input Signals

Clocks (PHI1, PHI2): Two-phase clocking signals. Section 2.2.

Ready (RDY): Active high. Used by slow memories to extend MMU originated memory cycles. Section 2.4.4.

Hold Request ($\overline{HOLD}$): Active low. Causes a release of the bus for DMA or multiprocessing purposes. Section 2.6.

Hold Acknowledge In ($\overline{HLDAI}$): Active low. Applied by the CPU in response to $\overline{HOLD}$ input, indicating that the CPU has released the bus for DMA or multiprocessing purposes. Section 2.6.



TL/EE/8692-27

FIGURE 3-10. MMU Slave Instruction Format

4.0 Device Specifications (Continued)

Reset Input ($\overline{RSTI}$): Active low. System reset. Section 2.3.

Status Lines (ST0–ST3): Status code input from the CPU. Active from T4 of previous bus cycle through T3 of current bus cycle. Section 2.4.

Program Flow Status ($\overline{PFS}$): Active low. Pulse issued by the CPU at the beginning of each instruction.

User/Supervisor Mode (U/$\overline{S}$): This signal is provided by the CPU. It is used by the MMU for protection and for selecting the address space (in dual address space mode only). Section 2.4.

Address Strobe Input ($\overline{ADS}$): Active low. Pulse indicating that a virtual address is present on the bus.

4.1.3 Output Signals

Reset Output/Abort ($\overline{RST}$/$\overline{ABT}$): Active Low. Held active longer than one clock cycle to reset the CPU. Pulsed low during T2 or TMMU to abort the current CPU instruction.

Interrupt Output ($\overline{INT}$): Active low. Pulse used by the debug functions to inform the CPU that a break condition has occurred.

Float Output ($\overline{FLT}$): Active low. Floats the CPU from the bus when the MMU accesses page table entries or performs a physical breakpoint check. Section 2.4.3.

Physical Address Valid ($\overline{PAV}$): Active low. Pulse generated during TMMU indicating that a physical address is present on the bus.

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	1.5W

Hold Acknowledge Output ($\overline{HLDAO}$): Active low. When active, indicates that the bus has been released.

4.1.4 Input-Output Signals

Data Direction In ($\overline{DDIN}$): Active low. Status signal indicating direction of data transfer during a bus cycle. Driven by the MMU during a page-table lookup.

Address Translation/Slave Processor Control ($\overline{AT}$/$\overline{SPC}$): Active low. Used by the CPU as the data strobe output for Slave Processor transfers; used by the MMU to acknowledge completion of an MMU instruction. Section 2.3 and 2.5. Held low during reset to select the address translation mode on the CPU.

M.S. Bit of Physical Address/High Byte Float (A24/$\overline{HBF}$): Most significant bit of physical address. Sampled on the rising edge of the reset input to select 16 or 32-bit bus mode. This pin outputs a low level if address translation is not enabled. It is floated during T2–T4 if 32-bit bus mode is selected.

Address Bits 16–23 (A16–A23): High order bits of the address bus. These signals are floated by the MMU during T2–T4 if 32-bit bus mode is selected.

Address/Data 0–15 (AD0–AD15): Multiplexed Address/Data Information. Bit 0 is the least significant bit.

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_A = 0 to +70°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		V_{CC} + 0.5	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{CH}	High Level Clock Voltage	PHI1, PHI2 pins only	V_{CC} − 0.35		V_{CC} + 0.5	V
V_{CL}	Low Level Clock Voltage	PHI1, PHI2 pins only	−0.5		0.3	V
V_{CLT}	Low Level Clock Voltage, Transient (ringing tolerance)	PHI1, PHI2 pins only	−0.5		0.6	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low Level Output Voltage	I_{OL} = 2 mA			0.45	V
I_{ILS}	$\overline{AT}$/$\overline{SPC}$ Input Current (low)	V_{IN} = 0.4V, $\overline{AT}$/$\overline{SPC}$ in input mode	0.05		1.0	mA
I_I	Input Load Current	0 ≤ V_{IN} ≤ V_{CC}, All inputs except PHI1, PHI2, $\overline{AT}$/$\overline{SPC}$	−20		20	μA
I_L	Leakage Current (Output and I/O Pins in TRI-STATE/Input Mode)	0.4 ≤ V_{IN} ≤ V_C	−20		30	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C		200	300	mA

4.0 Device Specifications (Continued)

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 2.0V on the rising or falling edges of the clock phases PHI1 and PHI2, and 0.8V or 2.0V on all other signals as illustrated in *Figures 4-2* and *4-3*, unless specifically stated otherwise.

ABBREVIATIONS:

L.E. — leading edge R.E. — rising edge
T.E. — trailing edge F.E. — falling edge

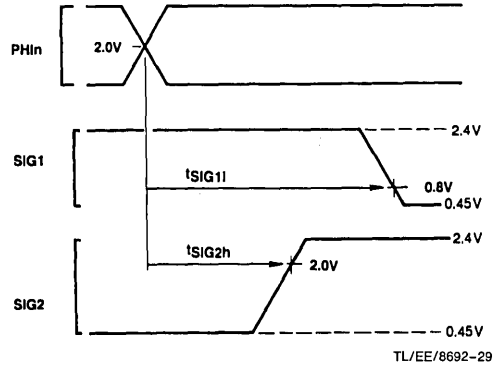


FIGURE 4-2. Timing Specification Standard (Signal Valid after Clock Edge)

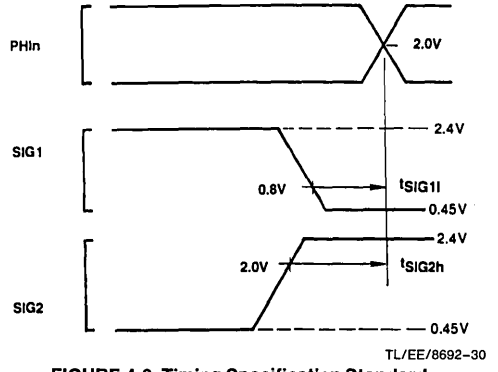


FIGURE 4-3. Timing Specification Standard (Signal Valid before Clock Edge)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32082-10.
Maximum times assume capacitive loading of 100 pF.

Name	Figure	Description	Reference/Conditions	NS32082-10		Units
				Min	Max	
t_{ALv}	4–4	Address Bits 0–15 Valid	After R.E., PHI1 TMMU or T1		40	ns
t_{ALh}	4–4	Address Bits 0–15 Hold	After R.E., PHI1 T2	5		ns
t_{AHv}	4–4, 4–6	Address Bits 16–24 Valid	After R.E., PHI1 TMMU or T1		40	ns
t_{AHh}	4–4	Address Bits 16–24 Hold	After R.E., PHI1 T2	5		ns
t_{ALPAVs}	4–5	Address Bits 0–15 Set Up	Before $\overline{PAV}$ T.E.	25		ns
t_{AHPAVs}	4–5	Address Bits 16–24 Set Up	Before $\overline{PAV}$ T.E.	25		ns
t_{ALPAVh}	4–5	Address Bits 0–15 Hold	After $\overline{PAV}$ T.E.	15		ns
t_{AHPAVh}	4–5	Address Bits 16–24 Hold	After $\overline{PAV}$ T.E.	15		ns
t_{ALf}	4–10	AD0–AD15 Floating	After R.E., PHI1 T2		25	ns
t_{AHf}	4–7, 4–10	A16–A24 Floating	After R.E., PHI1 T2 or T1		25	ns
t_{ALz}	4–15, 4–16	AD0–AD15 Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 Ti		25	ns
t_{AHZ}	4–15, 4–16	A16–A24 Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 Ti		25	ns
t_{ALr}	4–15, 4–16	AD0–AD15 Return from Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 T1		50	ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32082-10. (Continued)

Name	Figure	Description	Reference/Conditions	NS32082-10		Units
				Min	Max	
t_{AHr}	4–15, 4–16	A16–A24 Return from Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 T1		50	ns
t_{Dv}	4–6	Data Valid (Memory Write)	After R.E., PHI1 T2		50	ns
t_{Dh}	4–6	Data Hold (Memory Write)	After R.E., PHI1 next T1 or Ti	0		ns
t_{Df}	4–11	Data Bits Floating (Slave Processor Read)	After R.E., PHI1 T1 or Ti		10	ns
t_{Dv}	4–11	Data Valid (Slave Processor Read)	After R.E., PHI1 T1		50	ns
t_{Dh}	4–11	Data Hold (Slave Processor Read)	After R.E., PHI1 next T1 or Ti	0		ns
t_{DDINv}	4–5, 4–7	$\overline{DDIN}$ Signal Valid	After R.E., PHI1 T1 or T_{MMU}		50	ns
t_{DDINh}	4–5	$\overline{DDIN}$ Signal Hold	After R.E., PHI1 T1 or Ti	0		ns
t_{DDINf}	4–7	$\overline{DDIN}$ Signal Floating	After R.E., PHI1 T2		25	ns
t_{DDINz}	4–16	$\overline{DDIN}$ Signal Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 Ti		50	ns
t_{DDINr}	4–16	$\overline{DDIN}$ Return from Floating (Caused by $\overline{HOLD}$)	After R.E., PHI1 T1 or Ti		50	ns
t_{DDINAf}	4–9	$\overline{DDIN}$ Floating after Abort ($\overline{FLT}$ = 0)	After R.E., PHI2 T2		25	ns
t_{PAVa}	4–4	$\overline{PAV}$ Signal Active	After R.E., PHI1 T_{MMU} or T1		35	ns
t_{PAVia}	4–4	$\overline{PAV}$ Signal Inactive	After R.E., PHI2 T_{MMU} or T1		40	ns
t_{PAVw}	4–4	$\overline{PAV}$ Pulse Width	At 0.8V (Both Edges)	30		ns
t_{PAVdz}	4–14, 4–15	$\overline{PAV}$ Floating Delay	After $\overline{HLDAI}$ F.E.		25	ns
t_{PAVdr}	4–14, 4–15	$\overline{PAV}$ Return from Floating	After $\overline{HLDAI}$ R.E.		25	ns
t_{PAVz}	4–16	$\overline{PAV}$ Floating (Caused by $\overline{HOLD}$)	After R.E., PHI2 T4		30	ns
t_{PAVr}	4–16	$\overline{PAV}$ Return from Floating (Caused by $\overline{HOLD}$)	After R.E., PHI2 Ti		30	ns
t_{FLTa}	4–5, 4–10	$\overline{FLT}$ Signal Active	After R.E., PHI1 T_{MMU}		55	ns
t_{FLTia}	4–7, 4–10	$\overline{FLT}$ Signal Inactive	After R.E., PHI1 T_{MMU}, T_f or T2		35	ns
t_{ABTa}	4–8, 4–10	Abort Signal Active	After R.E., PHI1 T_{MMU} or T1		55	ns
t_{ABTia}	4–8, 4–10	Abort Signal Inactive	After R.E., PHI1 T2		55	ns
t_{ABTw}	4–8, 4–10	Abort Pulse Width	At 0.8V (Both Edges)	70		ns
t_{INTa}	4–4, 4–10	$\overline{INT}$ Signal Active	After R.E., PHI1 T_{MMU} or T_f		55	ns
t_{INTia}	4–4, 4–10	$\overline{INT}$ Signal Inactive	After R.E., PHI1 T2		55	ns
t_{INTw}	4–10	$\overline{INT}$ Pulse Width	At 0.8V (Both Edges)	70		ns
t_{SPCa}	4–13	$\overline{SPC}$ Signal Active	After R.E., PHI1 T1		40	ns
t_{SPCia}	4–13	$\overline{SPC}$ Signal Inactive	After R.E., PHI1 T4		40	ns

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32082-10. (Continued)

Name	Figure	Description	Reference/Conditions	NS32082-10		Units
				Min	Max	
t_{SPCf}	4–13	$\overline{SPC}$ Signal Floating	After F.E., PHI1 T4		25	ns
t_{SPCw}	4–13	$\overline{SPC}$ Pulse Width	At 0.8V (Both Edges)	70		ns
t_{HLD0da}	4–14	$\overline{HLDA0}$ Assertion Delay	After $\overline{HLDAI}$ F.E.		50	ns
$t_{HLD0dia}$	4–14, 4–15	$\overline{HLDA0}$ Deassertion Delay	After $\overline{HLDAI}$ R.E.		50	ns
t_{HLD0a}	4–15, 4–16	$\overline{HLDA0}$ Signal Active	After R.E., PHI1 Ti		30	ns
t_{HLD0ia}	4–16	$\overline{HLDA0}$ Signal Inactive	After R.E, PHI1 Ti		30	ns
t_{ATa}	4–18	$\overline{AT}/\overline{SPC}$ Signal Active	After R.E., PHI1		35	ns
t_{ATia}	4–18	$\overline{AT}/\overline{SPC}$ Signal Inactive	After R.E., PHI1		35	ns
t_{ATf}	4–18	$\overline{AT}/\overline{SPC}$ Signal Floating	After F.E., PHI1		25	ns
t_{RST0a}	4–18	$\overline{RST}/\overline{ABT}$ Asserted (Low)	After R.E. PHI1		30	ns
t_{RST0ia}	4–18	$\overline{RST}/\overline{ABT}$ Deasserted (High)	After R.E. PHI1 Ti		30	ns

4.4.2.2 Input Signal Requirements: NS32082-10

Name	Figure	Description	Reference/Conditions	NS32082-10		Units
				Min	Max	
t_{DIs}	4–5	Data In Set Up (Memory Read)	Before F.E., PHI2 T3	15		ns
t_{DIh}	4–5	Data In Hold (Memory Read)	After R.E., PHI1 T4	3		ns
t_{DIs}	4–12	Data In Set Up (Slave Processor Write)	Before F.E., PHI2 T1	20		ns
t_{DIh}	4–12	Data In Hold (Slave Processor Write)	After R.E., PHI1 T4	3		ns
t_{RDYs}	4–5	RDY Signal Set Up	Before F.E., PHI2 T2 or T3	15		ns
t_{RDYh}	4–5	RDY Signal Hold	After F.E., PHI1 T3	5		ns
t_{USs}	4–4, 4-11	U/$\overline{S}$ Signal Set Up	Before F.E., PHI2 T4 or Ti	35		ns
t_{USh}	4–4, 4–11	U/$\overline{S}$ Signal Hold	After R.E., PHI1 Next T4	0		ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32082-10 (Continued)

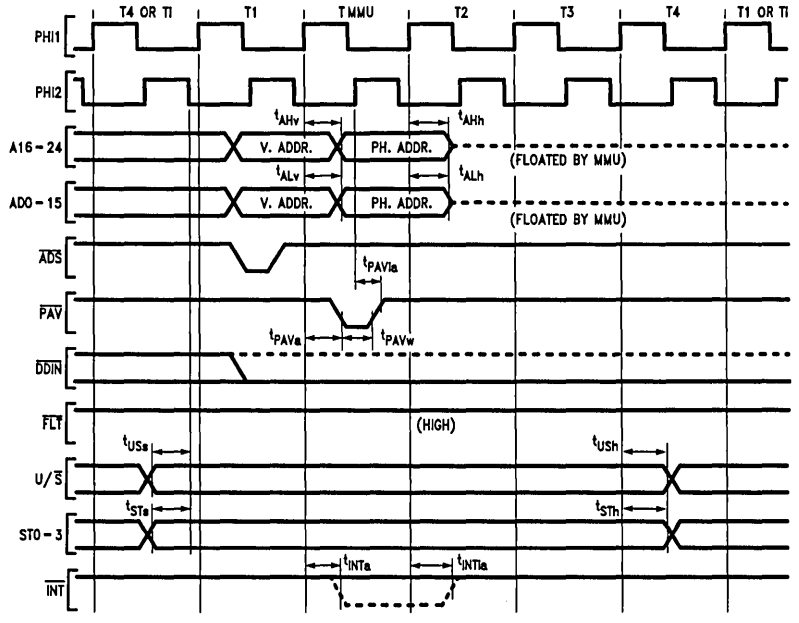
Name	Figure	Description	Reference/Conditions	NS32082-10		Units
				Min	Max	
t_{STs}	4–4, 4–11	Status Signals Set Up	Before F.E., PHI2 T4 or Ti	35		ns
t_{STh}	4–4, 4–11	Status Signals Hold	After R.E., PHI1 Next T4	0		ns
t_{SPCs}	4-11	$\overline{SPC}$ Input Set Up	Before F.E., PHI2 T1	45		ns
t_{SPCh}	4-11	$\overline{SPC}$ Input Hold	After R.E., PHI1 T4	0		ns
t_{HLDs}	4-16	$\overline{HOLD}$ Signal Set Up	Before F.E., PHI2 T4 or Ti	25		ns
t_{HLDh}	4-16	$\overline{HOLD}$ Signal Hold	After F.E., PHI2 T4 or Ti	0		ns
t_{HLDIs}	4-15	$\overline{HLDAI}$ Signal Set Up	Before F.E., PHI2 Ti	20		ns
t_{HLDIh}	4-15	$\overline{HLDAI}$ Signal Hold	After F.E., PHI2 Ti	0		ns
t_{HBFs}	4-18	A24/$\overline{HBF}$ Signal Set Up	Before F.E., PHI2	10		ns
t_{HBFh}	4-18	A24/$\overline{HBF}$ Signal Hold	After F.E., PHI2	0		ns
t_{RSTIs}	4-18	Reset Input Set Up	Before F.E., PHI1	20		ns
t_{PWR}	4-19	Power Stable to $\overline{RSTI}$ R.E.	After V_{CC} Reaches 4.5V	50		μs
t_{RSTIw}		$\overline{RSTI}$ Pulse Width	At 0.8V (Both Edges)	64		t_{cp}

4.4.2.3 Clocking Requirements: NS32082-10

Name	Figure	Description	Reference/ Conditions	NS32082-10		Units
				Min	Max	
t_{Cp}	4–17	Clock Period	R.E., PHI1, PHI2 to Next R.E., PHI1, PHI2	100	250	ns
t_{CLw}	4–17	PHI1, PHI2 Pulse Width	At 2.0V on PHI1, PHI2 (Both Edges)	$0.5t_{Cp}$ − 10 ns		
t_{CLh}	4–17	PHI1, PHI2 High Time	At V_{CC} − 0.9V on PHI1, PHI2 (Both Edges)	$0.5t_{Cp}$ − 15 ns		
t_{CLl}	4–17	PHI1, PHI2 Low Time	At 0.8V on PHI1, PHI2	$0.5t_{Cp}$ − 5 ns		
$t_{nOVL\,(1,2)}$	4–17	Non-overlap Time	0.8V on F.E. PHI1, PHI2 to 0.8V on R.E., PHI2, PHI1	−2	5	ns
t_{nOVLas}		Non-overlap Asymmetry ($t_{nOVL(1)} - t_{nOVL(2)}$)	At 0.8V on PHI1, PHI2	−4	4	ns
t_{CLwas}		PHI1, PHI2 Asymmetry $t_{CLw(1)} - t_{CLw(2)}$	At 2.0V on PHI1, PHI2	−5	5	ns

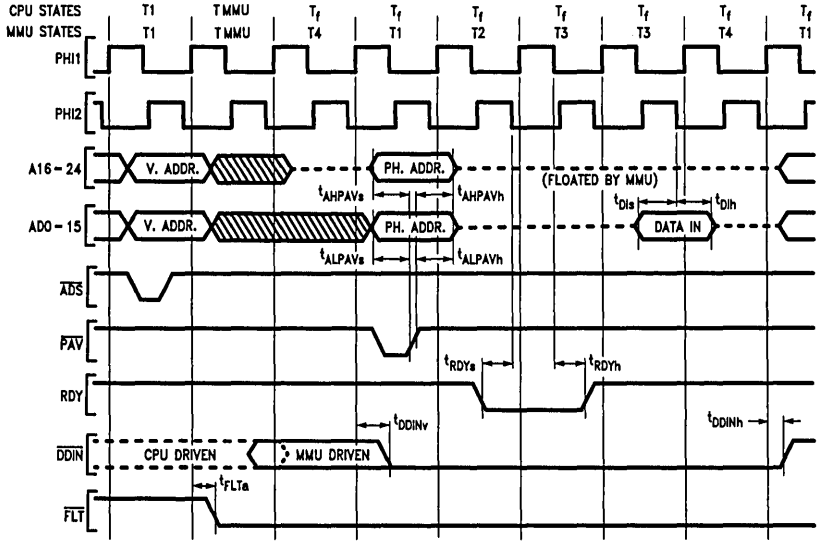
4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams



TL/EE/8692-31

FIGURE 4-4. CPU Read (Write) Cycle Timing (32-Bit Mode); Translation in TLB

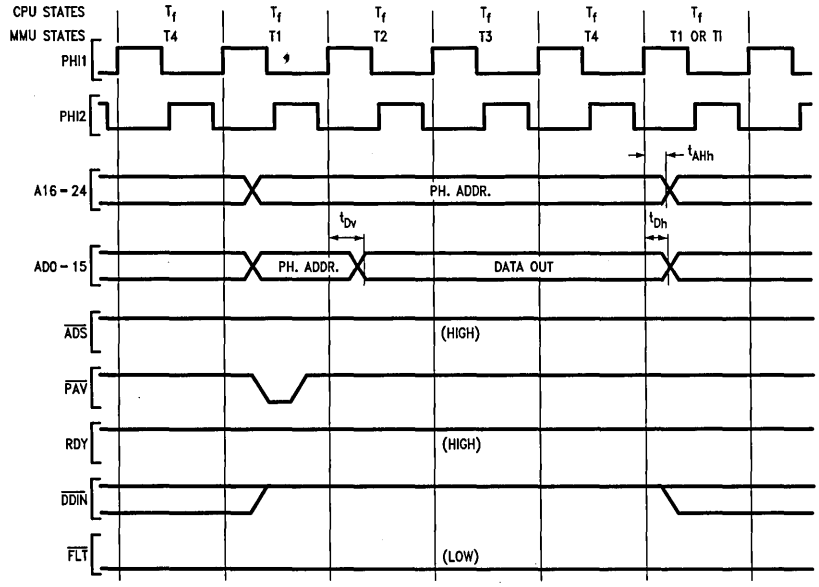


TL/EE/8692-32

FIGURE 4-5. MMU Read Cycle Timing (32-Bit Mode); After a TLB Miss

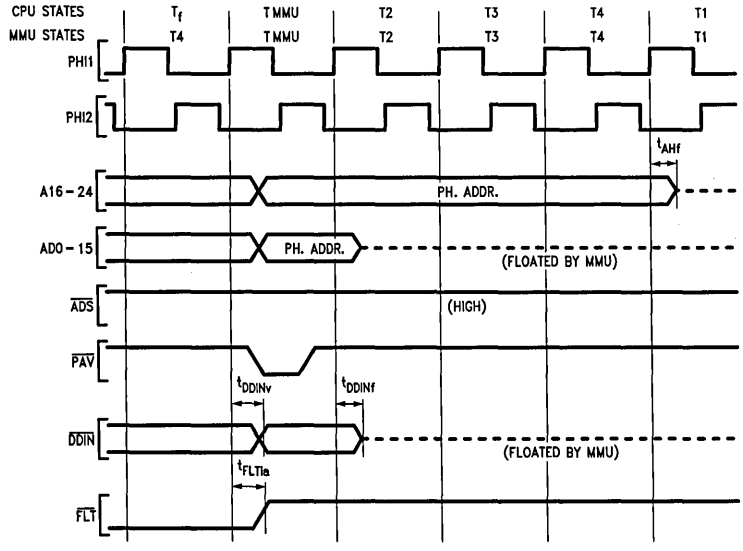
Note: After $\overline{FLT}$ is asserted, $\overline{DDIN}$ may be driven temporarily by both CPU and MMU. This, however, does not cause any conflict, since both CPU and MMU force $\overline{DDIN}$ to the same logic level.

4.0 Device Specifications (Continued)



TL/EE/8692–33

FIGURE 4-6. MMU Write Cycle Timing; after a TLB Miss

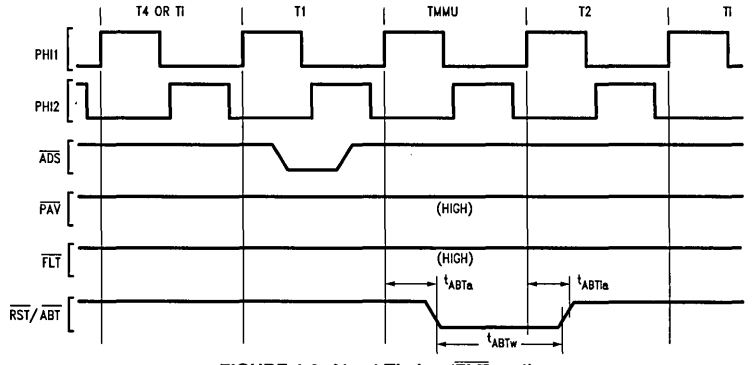


TL/EE/8692–34

FIGURE 4-7. $\overline{\text{FLT}}$ Deassertion Timing

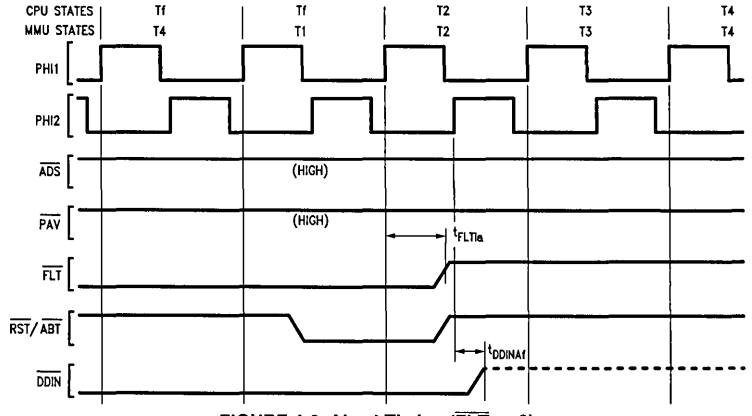
Note: After $\overline{\text{FLT}}$ is deasserted, $\overline{\text{DDIN}}$ may be driven temporarily by both CPU and MMU. This, however, does not cause any conflict. Since CPU and MMU force $\overline{\text{DDIN}}$ to the same logic level.

4.0 Device Specifications (Continued)



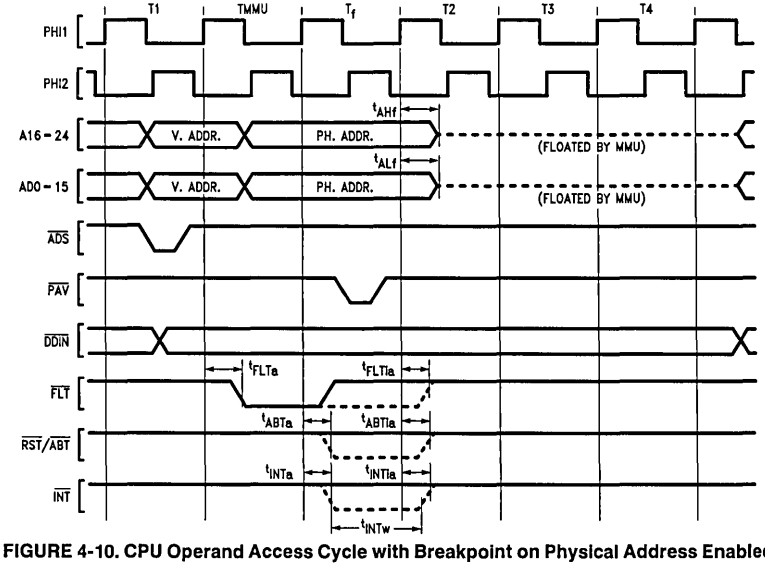
TL/EE/8692-35

FIGURE 4-8. Abort Timing ($\overline{FLT}$ = 1)



TL/EE/8692-36

FIGURE 4-9. Abort Timing ($\overline{FLT}$ = 0)

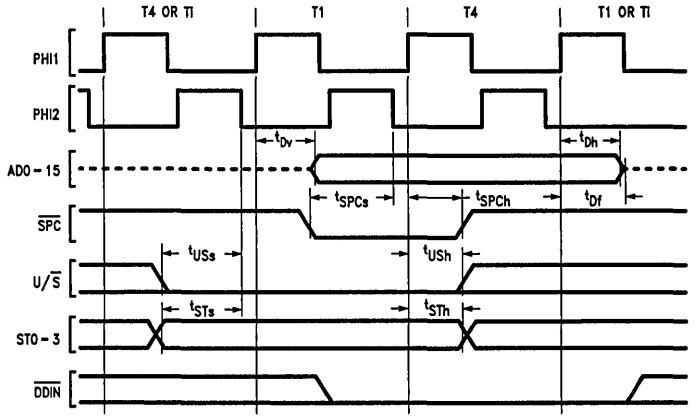


TL/EE/8692-37

FIGURE 4-10. CPU Operand Access Cycle with Breakpoint on Physical Address Enabled

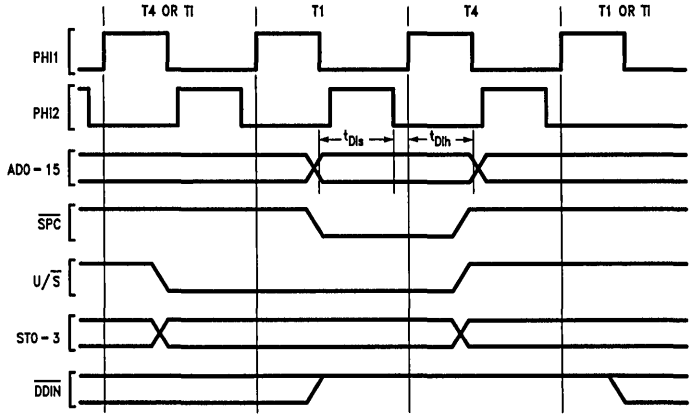
Note: If a breakpoint condition is met and abort on breakpoint is enabled, the bus cycle is aborted. In this case $\overline{FLT}$ is stretched by one clock cycle.

4.0 Device Specifications (Continued)



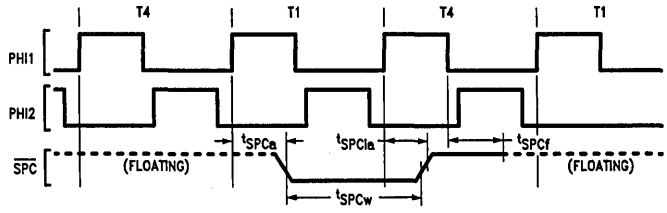
TL/EE/8692-38

FIGURE 4-11. Slave Access Timing; CPU Reading from MMU



TL/EE/8692-39

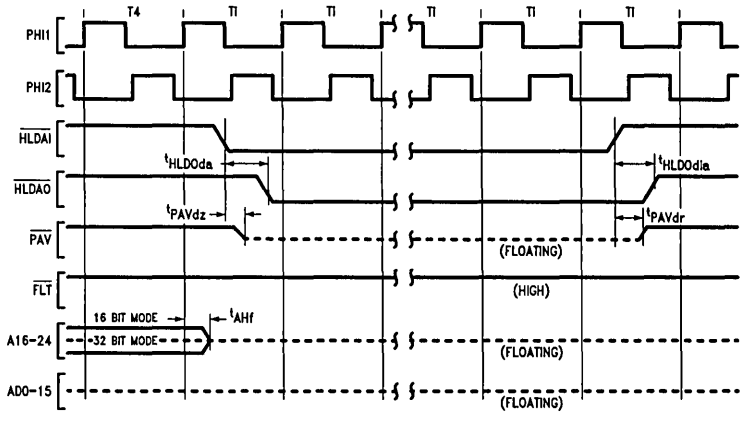
FIGURE 4-12. Slave Access Timing; CPU Writing to MMU



TL/EE/8692-40

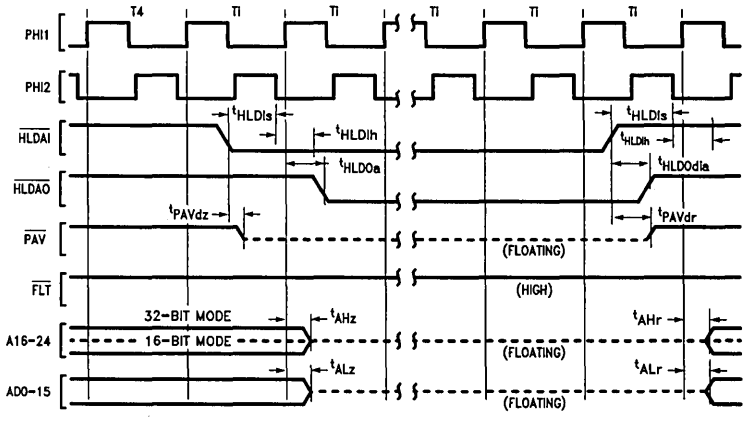
FIGURE 4-13. $\overline{SPC}$ Pulse from the MMU

4.0 Device Specifications (Continued)



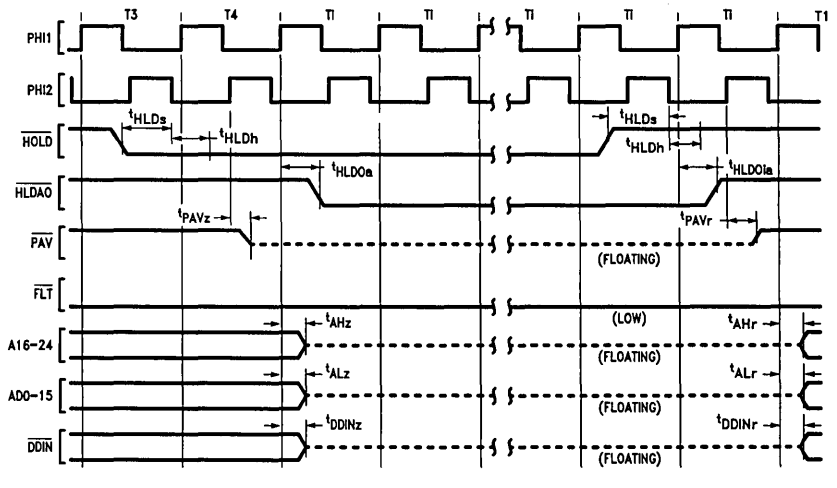
TL/EE/8692-41

FIGURE 4-14. Hold Timing ($\overline{FLT}$ = 1); SMR Instruction Not Being Executed



TL/EE/8692-42

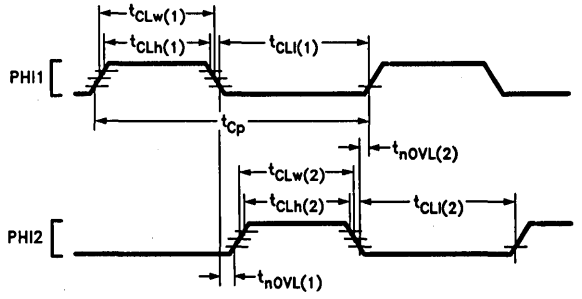
FIGURE 4-15. Hold Timing ($\overline{FLT}$ = 1); SMR Instruction Being Executed



TL/EE/8692-43

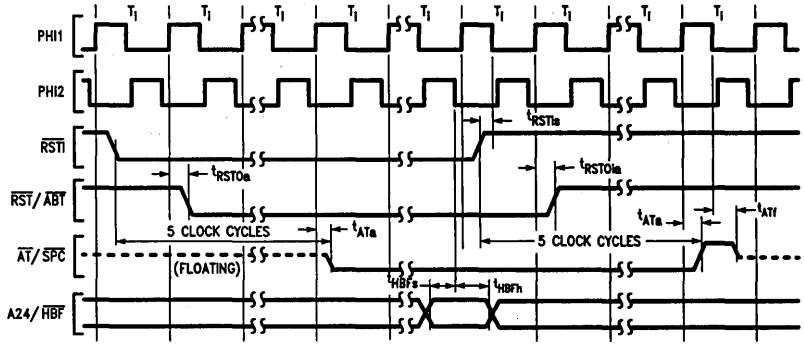
FIGURE 4-16. Hold Timing ($\overline{FLT}$ = 0)

4.0 Device Specifications (Continued)



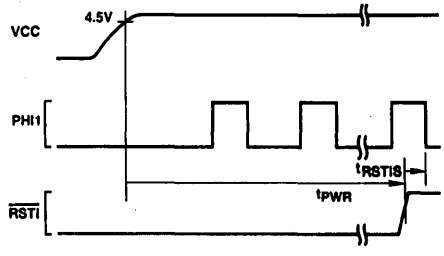
TL/EE/8692-49

FIGURE 4-17. Clock Waveforms



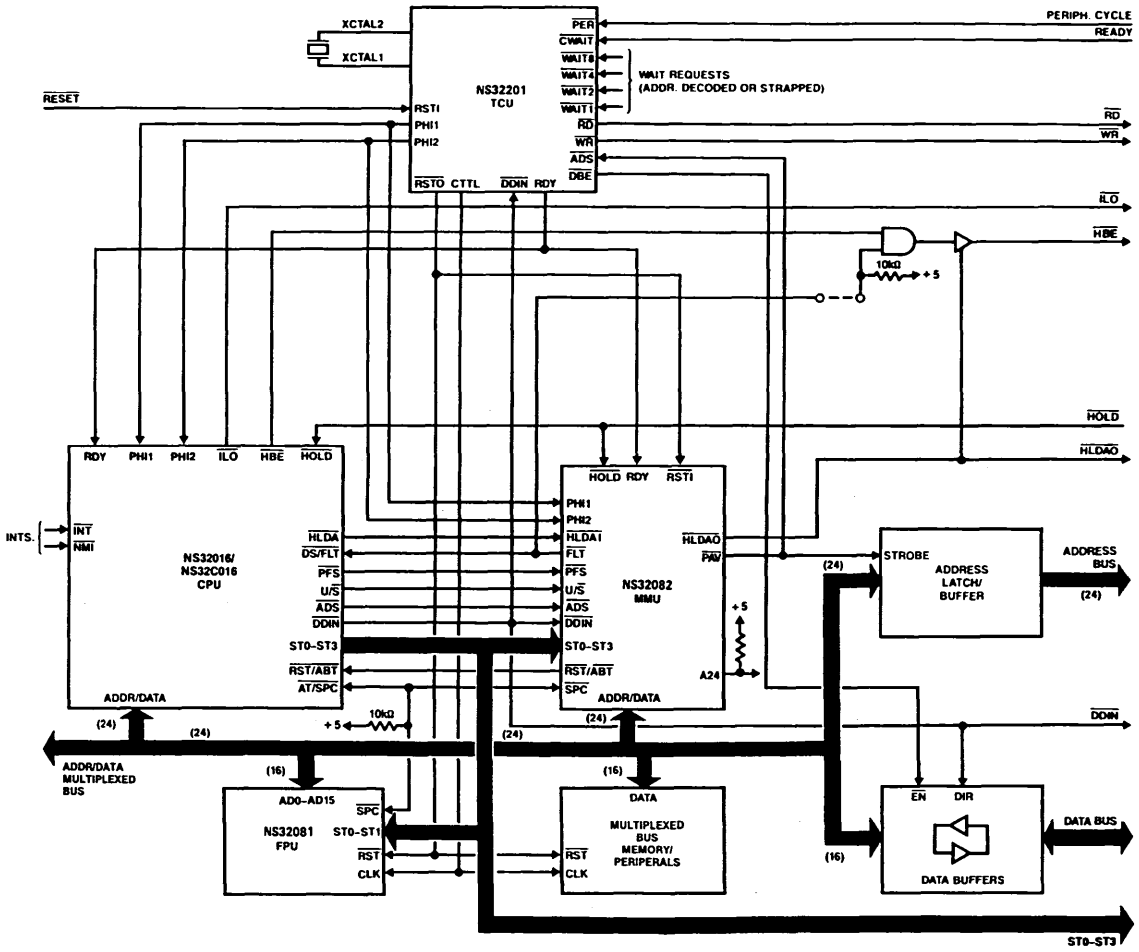
TL/EE/8692-45

FIGURE 4-18. Reset Timing



TL/EE/8692-46

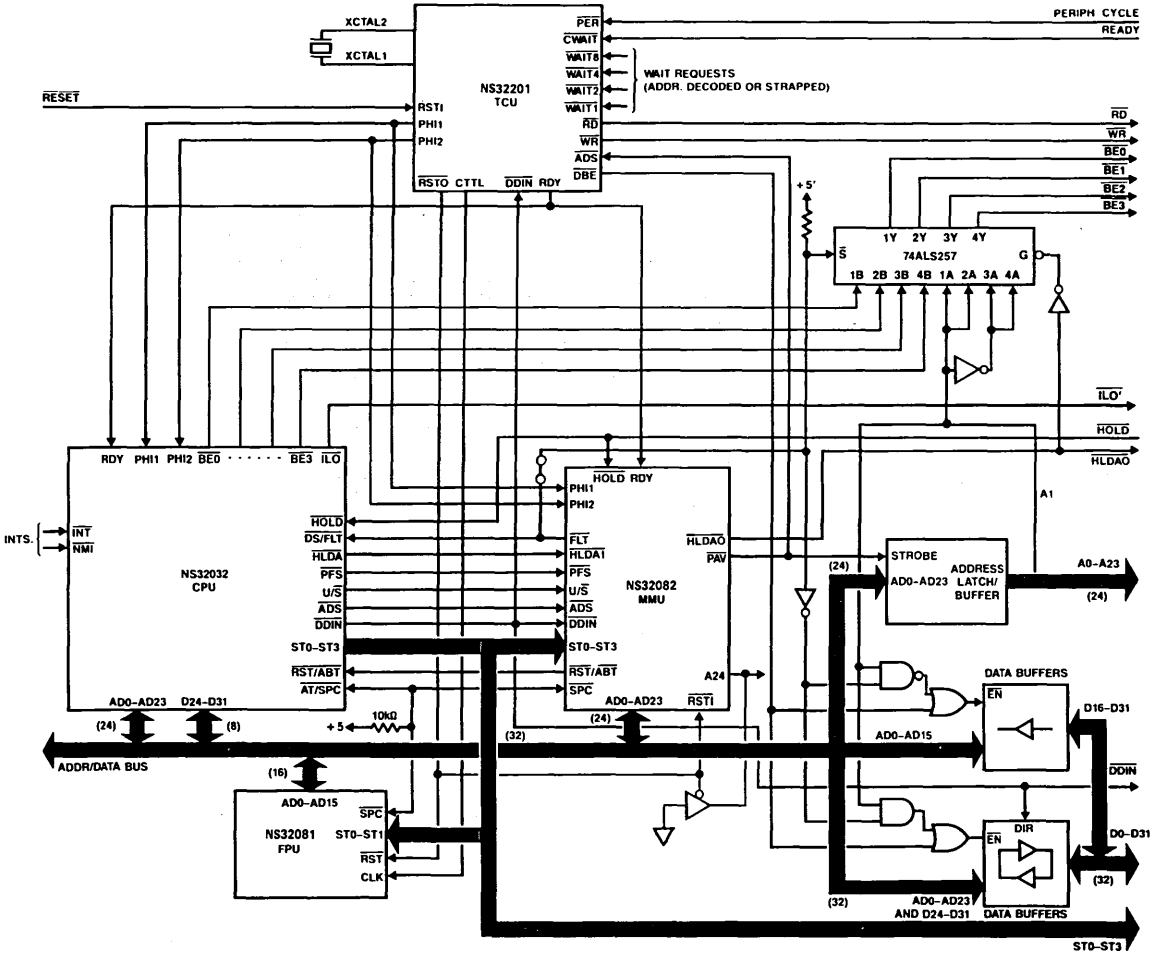
FIGURE 4-19. Power-On Reset



TL/EE/8692-47

Note: The "AND" gate on the $\overline{HBE}$ line is not needed when an NS32016 is used.

FIGURE A-1. System Connection Diagram



TL/EE/8692–48

FIGURE A-2. System Connection Diagram

National Semiconductor

PRELIMINARY

NS32381-15/NS32381-20/NS32381-25/NS32381-30 Floating-Point Unit

General Description

The NS32381 is a second generation, CMOS, floating-point slave processor that is fully software compatible with its forerunner, the NS32081 FPU. The NS32381 FPU functions with National's Embedded System Processors™, the NS32GX32 and the NS32CG16, and with any Series 32000 CPU, from the NS32008 to the NS32532, in a tightly coupled slave configuration. The performance of the NS32381 has been increased over the NS32081 by architecture improvements, hardware enhancements, and higher clock frequencies. Key improvements include the addition of a 32-bit slave protocol, an early done algorithm to increase CPU/FPU parallelism, an expanded register set, an automatic power down feature, expanded math hardware, and additional instructions.

The NS32381 FPU contains eight 64-bit data registers and a Floating-Point Status Register (FSR). The FPU executes 20 instructions, and operates on both single and double-precision operands. Three separate processors in the NS32381 manipulate the mantissa, sign, and exponent.

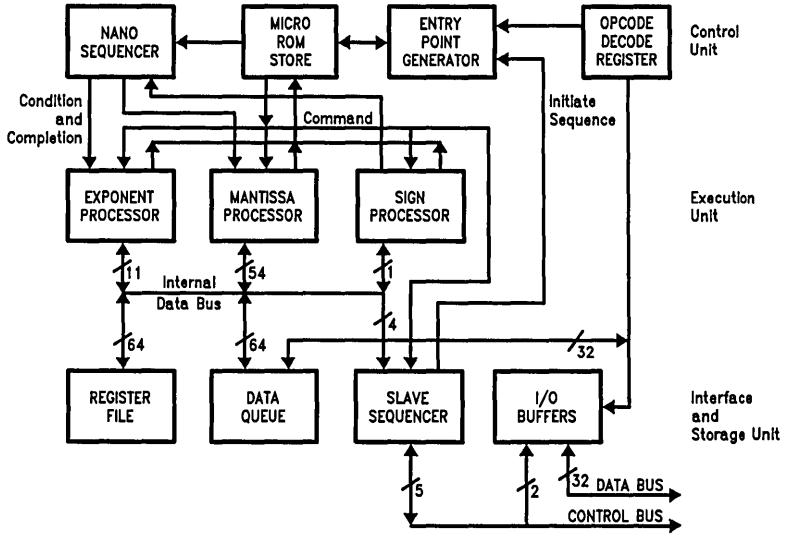
The CPU and NS32381 FPU form a tightly coupled computer cluster, which appears to the user as a single processing unit. The CPU and FPU communication is handled automatically, and is user transparent.

The FPU is fabricated with National's advanced double-metal CMOS process. It is available in a 68-pin Pin Grid Array (PGA) package or 68-pin Plastic package.

Features

- Compatible with NS32008, NS32016, NS32C016, NS32032, NS32C032, NS32332, NS32532, NS32CG16 and NS32GX32 microprocessors
- Selectable 16-bit or 32-bit Slave Protocol
- Format compatible with IEEE Standard 754-1985 for binary floating point arithmetic
- Early done algorithm
- Single (32-bit) and double (64-bit) precision operations
- Eight on-chip (64-bit) data registers
- Automatic power down mode
- Full upward compatibility with existing 32000 software
- High speed double-metal CMOS design
- 68-pin PGA package
- 68-pin plastic package

FPU Block Diagram



TL/EE/9157-1

FIGURE 1-1

Table of Contents

List of Illustrations

List of Tables

1.0 Product Introduction

The NS32381 Floating-Point Unit (FPU) provides high speed floating-point operations for the Series 32000 family, and is fabricated using National high-speed CMOS technology. It operates as a slave processor for transparent expansion of the Series 32000 CPU's basic instruction set. The FPU can also be used with other microprocessors as a peripheral device by using additional TTL and CMOS interface logic. The NS32381 is compatible with the IEEE Floating-Point Formats.

1.1 IEEE FEATURES SUPPORTED-STANDARD 754-1985

a) Basic floating-point number formats

b) Add, subtract, multiply, divide and compare operations

c) Conversions between different floating-point formats

d) Conversions between floating-point and integer formats

e) Round floating-point number to integer (round to nearest, round toward negative infinity and round toward zero, in double or single-precision)

f) Exception signaling and handling (invalid operation, divide by zero, overflow, underflow and inexact)

1.2 OPERAND FORMATS

The N32381 FPU operates on two floating-point data types—single precision (32 bits) and double precision (64 bits). Floating-point instruction mnemonics use the suffix F (Floating) to select the single precision data type, and the suffix L (Long Floating) to select the double precision data type.

A floating-point number is divided into three fields, as shown in *Figure 1-2*.

The F field is the fractional portion of the represented number. In Normalized numbers (Section 1.2.1), the binary point is assumed to be immediately to the left of the most significant bit of the F field, with an implied 1 bit to the left of the binary point. Thus, the F field represents values in the range $1.0 \leq x < 2.0$.

TABLE 1-1. Sample F Fields

F Field	Binary Value	Decimal Value
000...0	1.000...0	1.000...0
010...0	1.010...0	1.250...0
100...0	1.100...0	1.500...0
110...0	1.110...0	1.750...0
	↑	
	Implied Bit	

The E field contains an unsigned number that gives the binary exponent of the represented number. The value in the E field is biased; that is, a constant bias value must be subtracted from the E field value in order to obtain the true exponent. The bias value is $011\ldots11_2$, which is either 127 (single precision) or 1023 (double precision). Thus, the true exponent can be either positive or negative, as shown in Table 1-2.

TABLE 1-2. Sample E Fields

E Field	F Field	Represented Value
011...110	100...0	$1.5\times2^{-1} = 0.75$
011...111	100...0	$1.5\times2^{0} = 1.50$
100...000	100...0	$1.5\times2^{1} = 3.00$

Two values of the E field are not exponents. 11...11 signals a reserved operand (Section 1.2.3). 00...00 represents the number zero if the F field is also all zeroes, otherwise it signals a reserved operand.

The S bit indicates the sign of the operand. It is 0 for positive and 1 for negative. Floating-point numbers are in sign-magnitude form, that is, only the S bit is complemented in order to change the sign of the represented number.

1.2.1 Normalized Numbers

Normalized numbers are numbers which can be expressed as floating-point operands, as described above, where the E field is neither all zeroes nor all ones.

The value of a Normalized number can be derived by the formula:

$$(-1)^{S} \times 2^{(E\text{-}Bias)} \times (1 + F)$$

The range of Normalized numbers is given in Table 1-3.

1.2.2 Zero

There are two representations for zero—positive and negative. Positive zero has all-zero F and E fields, and the S bit is zero. Negative zero also has all-zero F and E fields, but its S bit is one.

1.2.3 Reserved Operands

The IEEE Standard for Binary Floating-Point Arithmetic provides for certain exceptional forms of floating-point operands. The NS32381 FPU treats these forms as reserved operands. The reserved operands are:

- Positive and negative infinity
- Not-a-Number (NaN) values
- Denormalized numbers

Both Infinity and NaN values have all ones in their E fields. Denormalized numbers have all zeroes in their E fields and non-zero values in their F fields.

The NS32381 FPU causes an Invalid Operation trap (Section 2.1.2.2) if it receives a reserved operand, unless the operation is simply a move (without conversion). The FPU does not generate reserved operands as results.

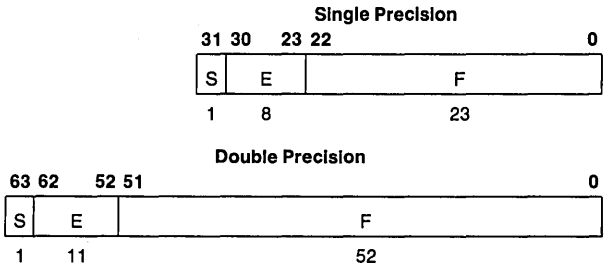


FIGURE 1-2. Floating-Point Operand Formats

1.0 Product Introduction (Continued)

TABLE 1-3. Normalized Number Ranges

	Single Precision	Double Precision
Most Positive	$2^{127} \times (2 - 2^{-23})$ $= 3.40282346 \times 10^{38}$	$2^{1023} \times (2 - 2^{-52})$ $= 1.7976931348623157 \times 10^{308}$
Least Positive	2^{-126} $= 1.17549436 \times 10^{-38}$	2^{-1022} $= 2.2250738585072014 \times 10^{-308}$
Least Negative	$-(2^{-126})$ $= -1.17549436 \times 10^{-38}$	$-(2^{-1022})$ $= -2.2250738585072014 \times 10^{-308}$
Most Negative	$-2^{127} \times (2 - 2^{-23})$ $= -3.40282346 \times 10^{38}$	$-2^{1023} \times (2 - 2^{-52})$ $= -1.7976931348623157 \times 10^{308}$

Note: The values given are extended one full digit beyond their represented accuracy to help in generating rounding and conversion algorithms.

1.2.4 Integers

In addition to performing floating-point arithmetic, the NS32381 FPU performs conversions between integer and floating-point data types. Integers are accepted or generated by the FPU as two's complement values of byte (8 bits), word (16 bits) or double word (32 bits) length.

See *Figure 1-3* for the Integer Format and Table 1-4 for the Integer Fields.

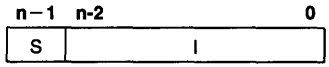


FIGURE 1-3. Integer Format

TABLE 1-4. Integer Fields

S	Value	Name
0	I	Positive Integer
1	$I - 2^n$	Negative Integer

Note: n represents the number of bits in the word, 8 for byte, 16 for word and 32 for double-word.

1.2.5 Memory Representations

The NS32381 FPU does not directly access memory. However, it is cooperatively involved in the execution of a set of two-address instructions with its Series 32000 Family CPU. The CPU determines the representation of operands in memory.

In the Series 32000 family of CPUs, operands are stored in memory with the least significant byte at the lowest byte address. The only exception to this rule is the Immediate addressing mode, where the operand is held (within the instruction format) with the most significant byte at the lowest address.

2.0 Architectural Description

2.1 PROGRAMMING MODEL

The Series 32000 architecture includes nine registers that are implemented on the NS32381 Floating-Point Unit (FPU).

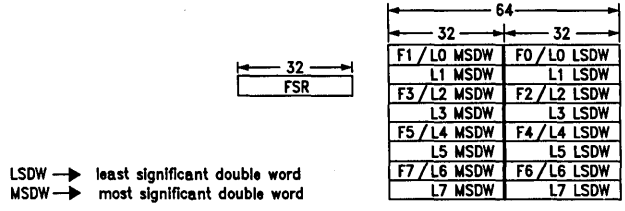
2.1.1 Floating-Point Registers

There are eight registers (L0–L7) on the NS32381 FPU for providing high-speed access to floating-point operands. Each is 64 bits long. A floating-point register is referenced whenever a floating-point instruction uses the Register addressing mode (Section 2.2.2) for a floating-point operand. All other Register mode usages (i.e., integer operands) refer to the General Purpose Registers (R0–R7) of the CPU, and the FPU transfers the operand as if it were in memory.

Note: These registers are all upward compatible with the 32-bit NS32081 registers, (F0–F7), such that when the Register addressing mode is specified for a double precision (64-bit) operand, a pair of 32-bit registers holds the operand. The programmer specifies the even register of the pair which contains the least significant half of the operand and the next consecutive register contains the most significant half.

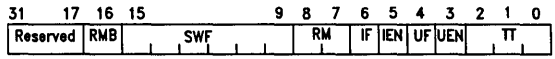
2.1.2 Floating-Point Status Register (FSR)

The Floating-Point Status Register (FSR) selects operating modes and records any exceptional conditions encountered during execution of a floating-point operation. *Figure 2-2* shows the format of the FSR.



TL/EE/9157-36

FIGURE 2-1. Register Set



TL/EE/9157-37

FIGURE 2-2. The Floating-Point Status Register

2.0 Architectural Description (Continued)

2.1.2.1 FSR Mode Control Fields

The FSR mode control fields select FPU operation modes. The meanings of the FSR mode control bits are given below.

Rounding Mode (RM): Bits 7 and 8. This field selects the rounding method. Floating-point results are rounded whenever they cannot be exactly represented. The rounding modes are:

00 Round to nearest value. The value which is nearest to the exact result is returned. If the result is exactly halfway between the two nearest values the even value (LSB = 0) is returned.

01 Round toward zero. The nearest value which is closer to zero or equal to the exact result is returned.

10 Round toward positive infinity. The nearest value which is greater than or equal to the exact result is returned.

11 Round toward negative infinity. The nearest value which is less than or equal to the exact result is returned.

Underflow Trap Enable (UEN): Bit 3. If this bit is set, the FPU requests a trap whenever a result is too small in absolute value to be represented as a normalized number. If it is not set, any underflow condition returns a result of exactly zero.

Inexact Result Trap Enable (IEN): Bit 5. If this bit is set, the FPU requests a trap whenever the result of an operation cannot be represented exactly in the operand format of the destination. If it is not set, the result is rounded according to the selected rounding mode.

2.1.2.2 FSR Status Fields

The FSR Status Fields record exceptional conditions encountered during floating-point data processing. The meanings of the FSR status bits are given below:

Trap Type (TT): bits 0-2. This 3-bit field records any exceptional condition detected by a floating-point instruction. The TT field is loaded with zero whenever any floating-point instruction except LFSR or SFSR completes without encountering an exceptional condition. It is also set to zero by a hardware reset or by writing zero into it with the Load FSR (LFSR) instruction. Underflow and Inexact Result are always reported in the TT field, regardless of the settings of the UEN and IEN bits.

000 No exceptional condition occurred.

001 Underflow. A non-zero floating-point result is too small in magnitude to be represented as a normalized floating-point number in the format of the destination operand. This condition is always reported in the TT field and UF bit, but causes a trap only if the UEN bit is set. If the UEN bit is not set, a result of Positive Zero is produced, and no trap occurs.

010 Overflow. A result (either floating-point or integer) of a floating-point instruction is too great in magnitude to be held in the format of the destination operand. Note that rounding, as well as calculations, can cause this condition.

011 Divide by zero. An attempt has been made to divide a non-zero floating-point number by zero. Dividing zero by zero is considered an Invalid Operation instead (below).

100 Illegal Instruction. Any instruction forms not included in the NS32381 Instruction Set are detected by the FPU as being illegal.

101 Invalid Operation. One of the floating-point operands of a floating-point instruction is a Reserved operand, or an attempt has been made to divide zero by zero using the DIVf instruction.

110 Inexact Result. The result (either floating-point or integer) of a floating-point instruction cannot be represented exactly in the format of the destination operand, and a rounding step must alter it to fit. This condition is always reported in the TT field and IF bit unless any other exceptional condition has occurred in the same instruction. In this case, the TT field always contains the code for the other exception and the IF bit is not altered. A trap is caused by this condition only if the IEN bit is set; otherwise the result is rounded and delivered, and no trap occurs.

111 (Reserved for future use.)

Underflow Flag (UF): Bit 4. This bit is set by the FPU whenever a result is too small in absolute value to be represented as a normalized number. Its function is not affected by the state of the UEN bit. The UF bit is cleared only by writing a zero into it with the Load FSR instruction or by a hardware reset.

Inexact Result Flag (IF): Bit 6. This bit is set by the FPU whenever the result of an operation must be rounded to fit within the destination format. The IF bit is set only if no other error has occurred. It is cleared only by writing a zero into it with the Load FSR instruction or by a hardware reset.

Register Modify Bit (RMB): Bit 16. This bit is set by the FPU whenever writing to a floating point data register. The RMB bit is cleared only by writing a zero with the LFSR instruction or by a hardware reset. This bit can be used in context switching to determine whether the FPU registers should be saved.

2.1.2.3 FSR Software Field (SWF)

Bits 9-15 of the FSR hold and display any information written to them (using the LFSR and SFSR instructions), but are not otherwise used by FPU hardware. They are reserved for use with NSC floating-point extension software.

2.0 Architectural Description (Continued)

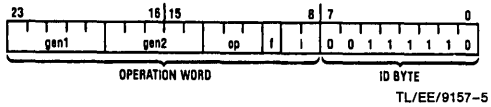
2.2 INSTRUCTION SET

2.2.1 Floating-Point Instruction Set

This section describes the floating-point instructions executed by the FPU in conjunction with the CPU. These instructions form a subset of the Series 32000® instruction set and take 9, 11, and 12 encoding formats. A list of all the Series 32000 instructions as well as details on their formats and addressing modes can be found in the appropriate CPU data sheets.

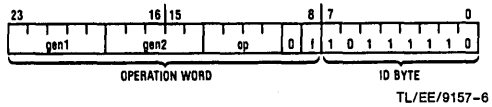
Certain notations in the following instruction description tables serve to relate the assembly language form of each instruction to its binary format in *Figure 2-3*.

Format 9



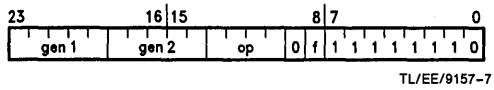
TL/EE/9157–5

Format 11



TL/EE/9157–6

Format 12



TL/EE/9157–7

FIGURE 2-3. Floating-Point Instruction Formats

The Format column indicates which of the three formats in *Figure 2-3* represents each instruction.

The Op column indicates the binary pattern for the field called "op" in the applicable format.

The Instruction column gives the form of each instruction as it appears in assembly language. The form consists of an instruction mnemonic in upper case, with one or more suffixes (i or f) indicating data types, followed by a list of operands (gen1, gen2).

An i suffix on an instruction mnemonic indicates a choice of integer data types. This choice affects the binary pattern in the i field of the corresponding instruction format as follows:

Suffix i	Data Type	i Field
B	Byte	00
W	Word	01
D	Double Word	11

An f suffix on an instruction mnemonic indicates a choice of floating-point data types. This choice affects the setting of the f bit of the corresponding instruction format as follows:

Suffix f	Data Type	f Bit
F	Single Precision	1
L	Double Precision (Long)	0

An operand designation (gen1, gen2) indicates a choice of addressing mode expressions. This choice affects the binary pattern in the corresponding gen1 or gen2 field of the instruction format. Refer to Table 2-1 for the options available and their patterns.

Further details of the exact operations performed by each instruction are found in the Series 32000 Instruction Set Reference Manual.

Movement and Conversion

The following instructions move the gen1 operand to the gen2 operand, leaving the gen1 operand intact.

Format	Op	Instruction		Description
11	0001	MOVf	gen1, gen2	Move without conversion
9	010	MOVLF	gen1, gen2	Move, converting from double precision to single precision.
9	011	MOVFL	gen1, gen2	Move, converting from single precision to double precision.
9	000	MOVif	gen1, gen2	Move, converting from any integer type to any floating-point type.
9	100	ROUNDfi	gen1, gen2	Move, converting from floating-point to the nearest integer.
9	101	TRUNCfi	gen1, gen2	Move, converting from floating-point to the nearest integer closer to zero.
9	111	FLOORfi	gen1, gen2	Move, converting from floating-point to the largest integer less than or equal to its value.

Note: The MOVLF instruction f bit must be 1 and the i field must be 10.
The MOVFL instruction f bit must be 0 and the i field must be 11.

Arithmetic Operations

The following instructions perform floating-point arithmetic operations on the gen1 and gen2 operands, leaving the result in the gen2 operand.

Note: POLY and DOT use the additional third implied operand.
POLY and DOT put their result to LO/FO register and not to GEN2.

Format	Op	Instruction		Description
11	0000	ADDf	gen1, gen2	Add gen1 to gen2.
11	0100	SUBf	gen1, gen2	Subtract gen1 from gen2.
11	1100	MULf	gen1, gen2	Multiply gen2 by gen1.

2.0 Architectural Description (Continued)

	Format	Op	Instruction		Description
	11	1000	DIVf	gen1, gen2	Divide gen2 by gen1.
	11	0101	NEGf	gen1, gen2	Move negative of gen1 to gen2.
	11	1101	ABSf	gen1, gen2	Move absolute value of gen1 to gen2.
(N)	12	0100	SCALBf	gen1, gen2	Move gen2*2^{gen1} to gen2, for integral values of gen1 without computing 2^{gen1}.
(N)	12	0101	LOGBf	gen1, gen2	Move the unbiased exponent of gen1 to gen2.
(N)	12	0011	DOTf	gen1, gen2	Move (gen1*gen2) + L0 to L0.(*)
(N)	12	0010	POLYf	gen1, gen2	Move (L0*gen1) + gen2 to L0.(*)

Notes:

(N): Indicates NEW instruction.

(*)The third impled operand used by these instructions can be either F0 or L0 depending on whether 'floating' or 'long' data type is specified in the opcode.

Comparison

The Compare instruction compares two floating-point values, sending the result to the CPU PSR Z and N bits for use as condition codes. See *Figure 3-11*. The Z bit is set if the gen1 and gen2 operands are equal; it is cleared otherwise. The N bit is set if the gen1 operand is greater than the gen2 operand; it is cleared otherwise. The CPU PSR L bit is unconditionally cleared. Positive and negative zero are considered equal.

Format	Op	Instruction		Description
11	0010	CMPf	gen1, gen2	Compare gen1 to gen2.

Floating-Point Status Register Access

The following instructions load and store the FSR as a 32-bit integer.

Format	Op	Instruction		Description
9	001	LFSR	gen1	Load FSR
9	110	SFSR	gen2	Store FSR

Note: All instructions support all of the NS32000 family data formats (for external operands) and all addressing modes are supported.

Rounding

The FPU supports all IEEE rounding options: Round toward nearest value or even significant if a tie. Round toward zero, Round toward positive infinity and Round toward negative infinity.

2.3 EXCEPTIONS

The FPU supports five types of exceptions: Invalid operation, Division by zero, Overflow, Underflow and Inexact Result. When an exception occurs, the FPU may or may not generate a trap depending upon the bit setting in the FSR Register. The user can disable the Inexact Result and the Underflow traps. If an undefined Floating-Point instruction is passed to the FPU an Illegal Instruction trap will occur. The user can't disable trap on Illegal Instruction.

Upon detecting an exceptional condition in executing a floating-point instruction, the FPU requests a TRAP by pulsing the $\overline{SPC}$ line for one clock cycle, pulsing the $\overline{SDN332}$ line for two and a half clock cycles and pulsing the $\overline{FSSR}$ line for one clock cycle. (The user will connect the correct lines according to the CPU being used).

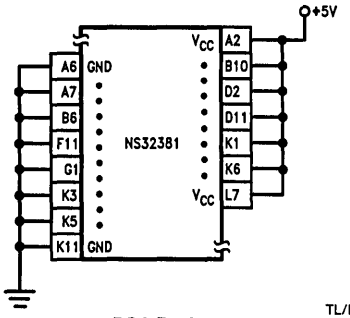
In addition, the FPU sets the Q bit in the status word register. The CPU responds by reading the status word register (refer to Section 3.6.1 for its format) while applying status h'E (transferring status word) on the status lines. A trapped instruction returns no result (even if the destination is FPU register) and does not affect the CPU PSR. The FPU records exceptional cause in the trap type (TT) field of the FSR. If an illegal opcode is detected, the FPU sets the TS bit in the slave processor status word register, indicating a trap (UND).

3.0 Functional Description

3.1 POWER AND GROUNDING

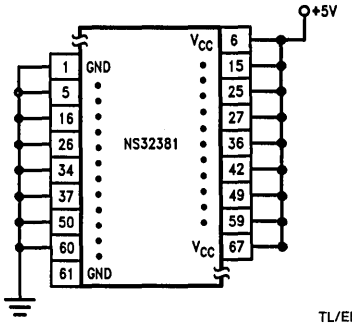
The NS32381 requires a single 5V power supply, applied on the V_{CC} pins. These pins should be connected together by a power (V_{CC}) plane on the printed circuit board. See *Figure 3-1*.

The grounding connections are made on the GND pins. These pins should be connected together by a ground (GND) plane on the printed circuit board. See *Figure 3-1*.



TL/EE/9157-8

PGA Package

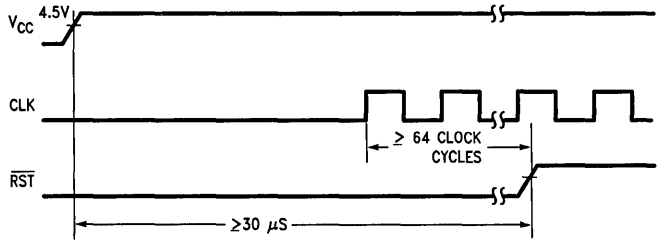


TL/EE/9157-43

PLCC Package

FIGURE 3-1. Recommended Supply Connections

3.0 Functional Description (Continued)



TL/EE/9157-9

FIGURE 3-2. Power-On Reset Requirements

3.2 AUTOMATIC POWER DOWN MODE

The NS32381 supports a power down mode in which the device consumes only 10% of its original power at 30 MHz. The NS32381 enters the power down mode (internal clocks are stopped with phase two high) if it does not receive an $\overline{SPC}$ pulse from the CPU within 256 clocks.

The FPU exits the power down mode and returns to normal operation after it receives an $\overline{SPC}$ from the CPU. There is no extra delay caused by the FPU being in the power down mode.

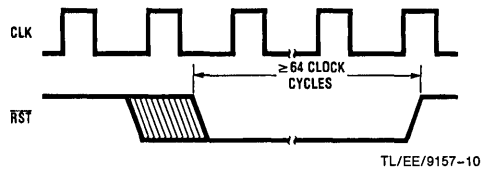
3.3 CLOCKING

The NS32381 FPU requires a single-phase TTL clock input on its CLK pin (pin A8). Different Clock sources can be used to provide the CLK signal depending on the application. For example, it can come from the BCLK of the NS32532 CPU. It can also come from the CTTL pin of the NS32C201 Timing Control Unit, if it is required.

3.4 RESETTING

The $\overline{RST}$ pin serves as a reset for on-chip logic. The FPU may be reset at any time by pulling the $\overline{RST}$ pin low for at least 64 clock cycles. Upon detecting a reset, the FPU terminates instruction processing, resets its internal logic, and clears the FSR to all zeroes.

On application of power, $\overline{RST}$ must be held low for at least 30 μs after V_{CC} is stable. This ensures that all on-chip voltages are completely stable before operation. See *Figures 3-2* and *3-3*.



TL/EE/9157-10

FIGURE 3-3. General Reset Timing

3.5 BUS OPERATION

Instructions and operands are passed to the NS32381 FPU with slave processor bus cycles. Each bus cycle transfers either one byte (8 bits), one word (16 bits) or one double word (32 bits) to or from the FPU. During all bus cycles, the $\overline{SPC}$ line is driven by the CPU as an active low data strobe, and the FPU monitors pins ST0–ST3 to keep track of the sequence (protocol) established for the instruction being executed. This is necessary in a virtual memory environment, allowing the FPU to retry an aborted instruction.

3.5.1 Bus Cycles

A bus cycle is initiated by the CPU, which asserts the proper status on (ST0–ST3) and pulses $\overline{SPC}$ low. The status lines are sampled by the FPU on the leading (falling) edge of the $\overline{SPC}$ pulse except for the 32532 CPU. When used with the 32532 CPU, the status lines are sampled on the rising edge of CLK in the T2 state. If the transfer is from the FPU (a slave processor read cycle), the FPU asserts data on the data bus for the duration of the $\overline{SPC}$ pulse. If the transfer is to the FPU (a slave processor write cycle), the FPU latches data from the data bus on the trailing (rising) edge of the $\overline{SPC}$ pulse. *Figures 3-5, 3-6, 3-7* and *3-8* illustrate these sequences.

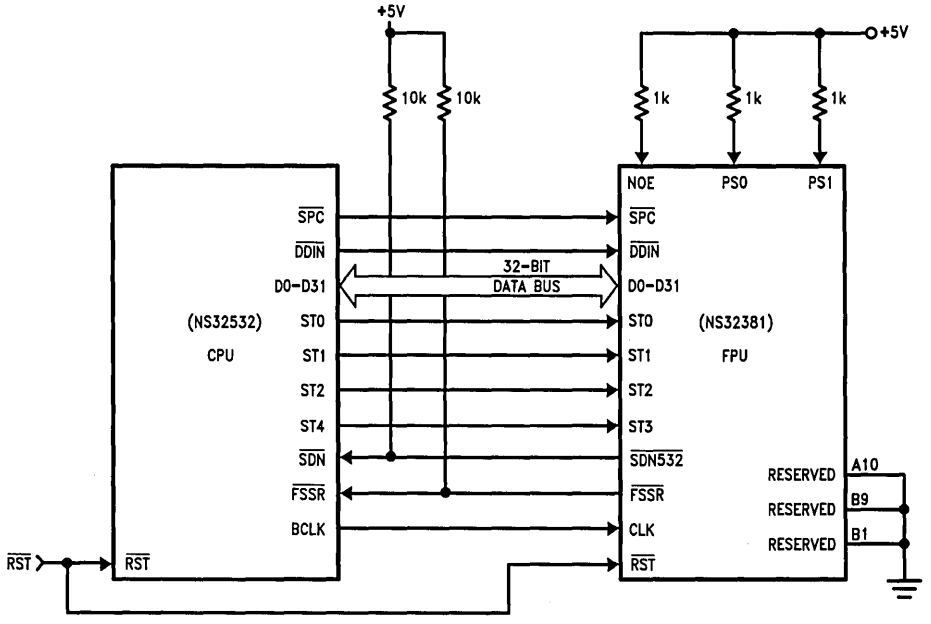
The direction of the transfer and the role of the bidirectional $\overline{SPC}$ line are determined by the instruction protocol being performed. $\overline{SPC}$ is always driven by the CPU during slave processor bus cycles. Protocol sequences for each instruction are given in Section 3.6.

3.5.2 Operand Transfer Sequences

An operand is transferred in one or more bus cycles. For the 16-Bit Slave Protocol a 1-byte operand is transferred on the least significant byte of the data bus (D0–D7). A 2-byte operand is transferred on the entire bus. A 4-byte or 8-byte operand is transferred in consecutive bus cycles, least significant word first.

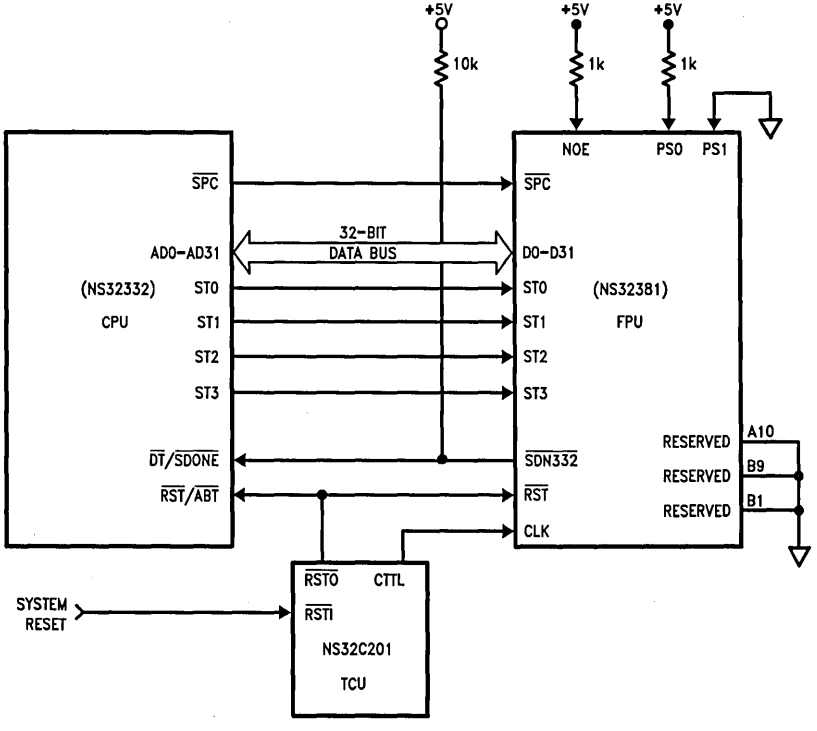
For the 32-Bit Slave Protocol a 4-byte operand is transferred on the entire data bus in a single bus cycle and an 8-byte operand is transferred in two consecutive bus cycles with the most significant byte transferred on data bits (D0–D7). The complete operand transfer of bytes B0–B7 where B0 is the least significant byte would appear on the data bus as B4, B5, B6, B7 followed by B0, B1, B2, B3 in the second bus cycle.

3.0 Functional Description (Continued)



TL/EE/9157-38

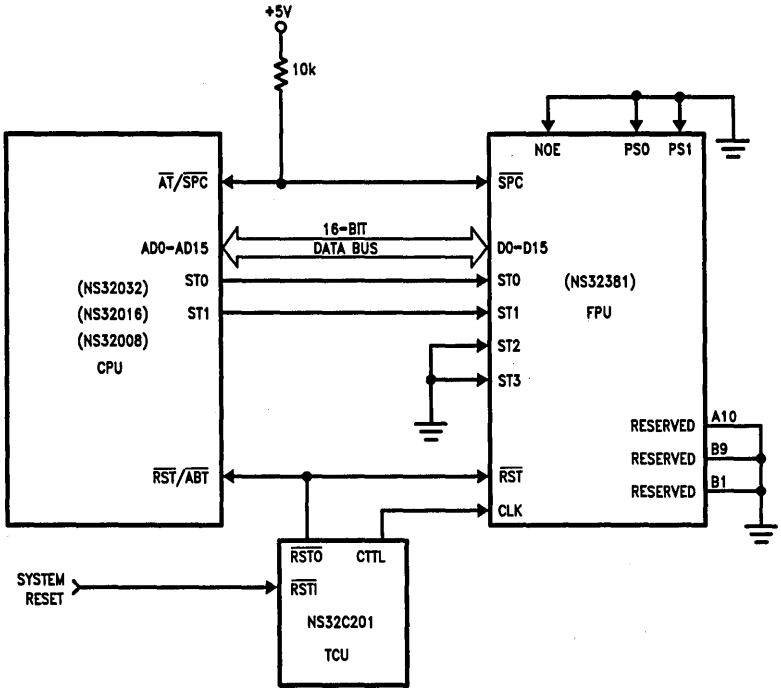
FIGURE 3-4a. System Connection Diagram with the NS32532 CPU



TL/EE/9157-39

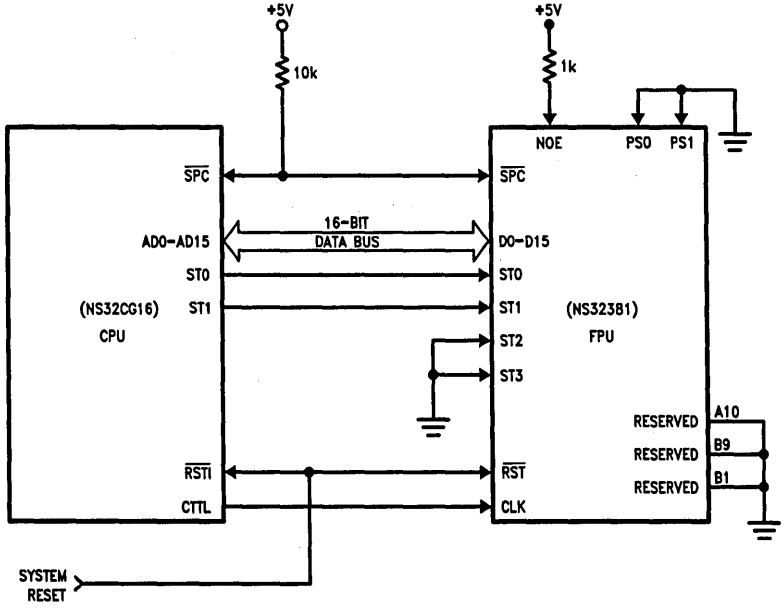
FIGURE 3-4b. System Connection Diagram with the NS32332 CPU

3.0 Functional Description (Continued)



TL/EE/9157-40

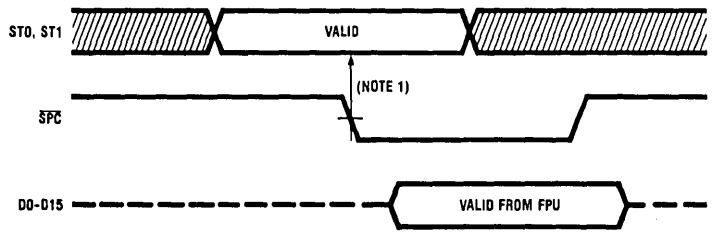
FIGURE 3-4c. System Connection Diagram with the NS32008, NS32016 or NS32032 CPU



TL/EE/9157-41

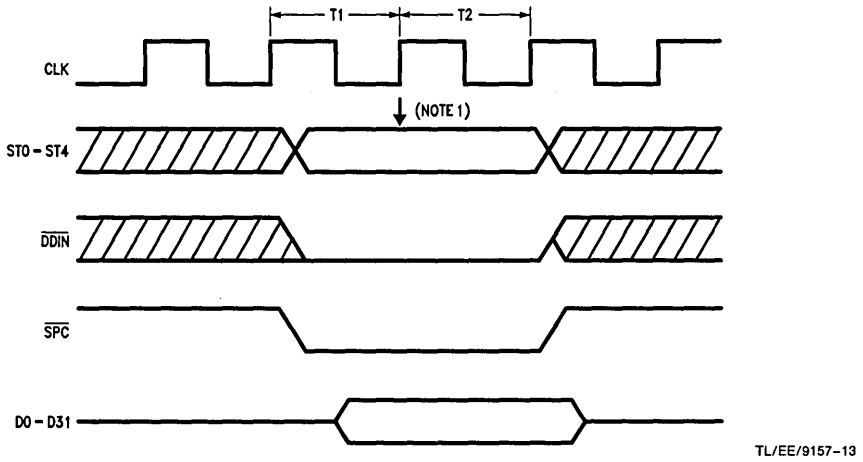
FIGURE 3-4d. System Connection Diagram with the NS32CG16 CPU

3.0 Functional Description (Continued)



Note 1: FPU samples CPU status here.

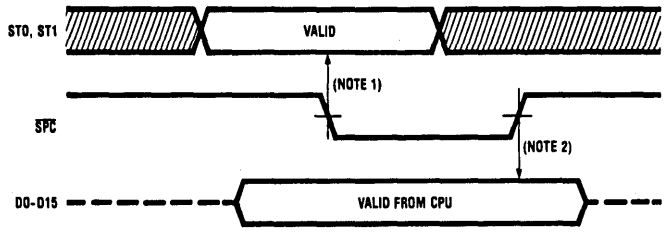
FIGURE 3-5. Slave Processor Read Cycle (NS32008, NS32016, NS32032 and NS32332 CPUs)



Note 1: FPU samples CPU status here.

FIGURE 3-6. Slave Processor Read Cycle (NS32532 CPU)

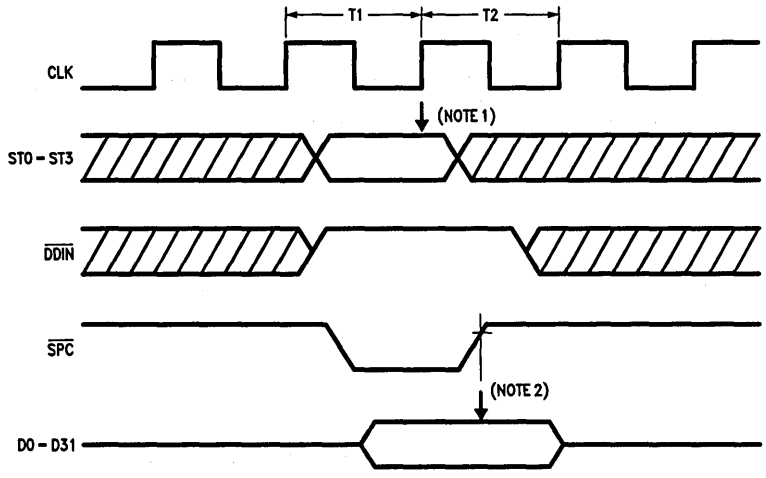
3.0 Functional Description (Continued)



TL/EE/9157–14

Note 1: FPU samples CPU status here.
Note 2: FPU samples data bus here.

FIGURE 3-7. Slave Processor Write Cycle (NS32008, NS32016, NS32032 and NS32332 CPU)



TL/EE/9157–15

Note 1: FPU samples CPU status here.
Note 2: FPU samples data bus here.

FIGURE 3-8. Slave Processor Write Cycle (NS32532 CPU)

3.0 Functional Description (Continued)

3.6 INSTRUCTION PROTOCOLS

3.6.1 General Protocol Sequences

The NS32381 supports both the 16-bit and 32-bit General Slave protocol sequences. See Tables 3-1, 3-2 and *Figures 3-12, 3-13* respectively.

Slave Processor instructions have a three-byte Basic Instruction field, consisting of an ID byte followed by an Operation Word. See *Figure 3-9* for the ID and Opcode format 16-bit Slave Protocol and *Figure 3-10* for the ID and Opcode Format 32-bit Slave Protocol. The ID Byte has three functions:

1) It identifies the instruction to the CPU as being a Slave Processor instruction.

2) It specifies which Slave Processor will execute it.

3) It determines the format of the following Operation Word of the instruction.

Upon receiving a slave processor instruction, the CPU initiates a sequence outlined in either Table 3-1 or 3-2, depending on the PS0 and PS1, to allow for the 16-bit or 32-bit slave protocol. The NS32008, NS32016, NS32C016, NS32032, NS32C032 and NS32CG16 all communicate with the NS32381 using the 16-bit Slave Protocol. The NS32332, NS32532 and NS32GX32 CPUs communicate with the NS32381 using a 32-bit Slave Protocol; a different version is provided for each CPU.

TABLE 3-1. 16-Bit General Slave Instruction Protocol

Step	Status	Action
1	ID (1111)	CPU sends ID Byte
2	OP (1101)	CPU sends Operation Word
3	OP (1101)	CPU sends required operands (if any)
4	—	Slaves starts execution (CPU prefetches)
5	—	Slave pulses $\overline{SPC}$ low
6	ST (1110)	CPU Reads Status Word
7	OP (1101)	CPU Reads Result (if destination is memory and if no TRAP occurred)

TABLE 3-2. 32-Bit General Slave Instruction Protocol

Step	Status	Action
1	ID (1111)	CPU sends ID and Operation Word
2	OP (1101)	CPU sends required operands (if any)
3	—	Slaves starts execution (CPU prefetches)
4	—	Slave signals DONE or TRAP or CMPf
5	ST (1110)	CPU Reads Status Word (If TRAP was signaled or a CMPf instruction was executed)
6	OP (1101)	CPU Reads Result (if destination is memory and if no TRAP occurred)

TABLE 3-3. Floating-Point Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Destination	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none
SCALBf	read.f	rmw.f	f	f	f to Op.2	none
LOGBf	read.f	write.f	f	N/A	f to Op.2	none
DOTf	read.f	read.f	f	f	*f to F0/L0	none
POLYf	read.f	read.f	f	f	*f to F0/L0	none

D = Double Word

i = Integer size (B, W, D) specified in mnemonic.

f = Floating-Point type (F, L) specified in mnemonic.

N/A = Not Applicable to this instruction.

*The "returned value" can go to either F0 or L0 depending on the "f" bit in the opcode, i.e., whether "floating" or "long" data type is used.

3.0 Functional Description (Continued)

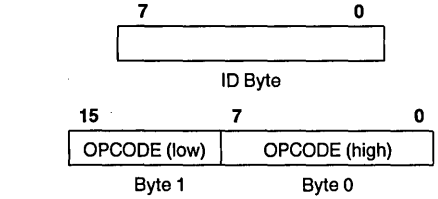


FIGURE 3-9. ID and OPCODE Format 16-Bit Slave Protocol

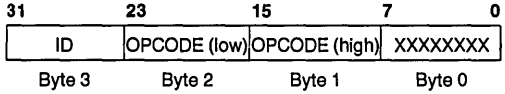


FIGURE 3-10. ID and OPCODE Format 32-Bit Slave Protocol

For the 16-bit Slave Protocol the CPU applies Status Code 1111 (Broadcast ID), and sends the ID Byte on the least significant half of the Data Bus (D0–D7). The CPU next sends the Operation Word while applying Status Code 1101 (Transfer Slave Operand). The Operation Word is swapped on the Data Bus; that is, bits 0–7 appear on pins D8–D15, and bits 8–15 appear on pins D0–D7.

For the 32-bit Slave Protocol the CPU applies Status Code 1111 and sends the ID Byte (different ID for each format) in byte 3 (D24–D31) and the Operation Word in bytes 1 and 2 in a single double word transfer. The Operation Word is swapped such that OPCODE low appears on byte 2 (D16–D23) and OPCODE high appears on byte 1 (D8–D15). Byte 0 (D0–D7) is not used.

All Slave Processors input and decode the data from these transfers. The Slave Processor selected by the ID Byte is activated and from this point on the CPU is communicating with it only. If any other slave protocol is in progress (e.g., an aborted Slave instruction), this transfer cancels it. Both the CPU and FPU are aware of the number and size of the operands at this point.

Using the Addressing Mode fields within the Operation Word, the CPU starts fetching operands and issuing them to the FPU. To do so, it references any Addressing Mode extensions appended to the FPU instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the Slave Processor. The Status Code applied is 1101 (Transfer Slave Processor Operand).

After the CPU has issued the last operand, the FPU starts the actual execution of the instruction. A one clock cycle $\overline{SPC}$ pulse is used to indicate the completion of the instruction and for the CPU to continue with the 16-Bit Slave Protocol by reading the FPU's Status Word Register.

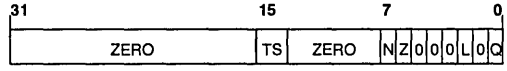
For the 32-bit Slave Protocol, upon completion of the instruction, the FPU will signal the CPU by pulsing either $\overline{SDNXXX}$ or $\overline{FSSR}$ (Force Slave Status Read).

A half clock cycle $\overline{SDN332}$ pulse with a NS32332 CPU, or a one clock cycle $\overline{SDN532}$ pulse with a NS32532 or NS32GX32 CPU, indicates a valid completion of the instruction and that there is no need for the CPU to read its Status Word Register.

But if there is a need for the CPU to read FPU's Status Word Register, a two and a half clock cycle $\overline{SDN332}$ (from NS32332) or a one clock cycle $\overline{FSSR}$ pulse (from NS32532 or NS32GX32) will be issued instead.

In all cases for both the 16-Bit and 32-Bit Slave Protocols the CPU will use $\overline{SPC}$ to read the Status Word from the FPU, while applying status code (1110). This word has the format shown in *Figure 3-11*. If the Q bit ("Quit", Bit 0) is set, this indicates that an error (TRAP) has been detected by the FPU. The CPU will not continue the protocol, but will immediately trap through the Slave vector in the Interrupt Table. If the instruction being performed is CMPf (Section 2.2.3) and the Q bit is not set, the CPU loads Processor Status Register (PSR) bits N, Z and L from the corresponding bits in the FPU Status Word. The FPU always sets the L bit to zero.

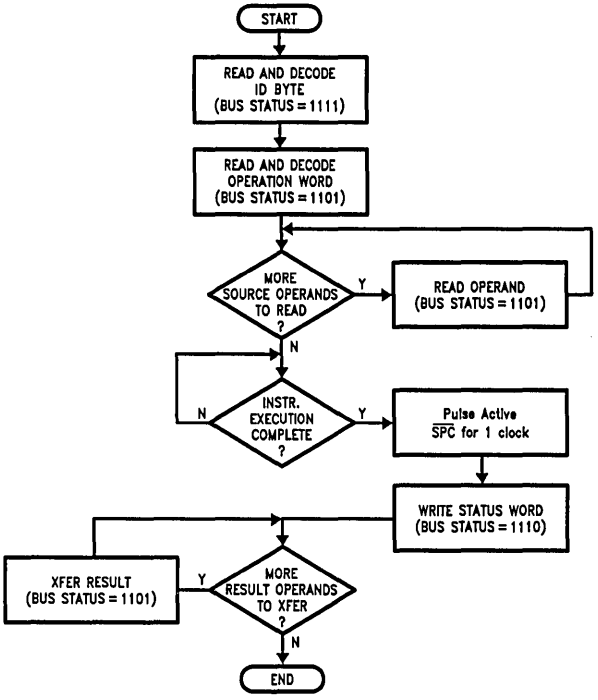
The last step will be for the CPU to read the result, provided there are no errors and the results destination is in memory. Here again the CPU uses $\overline{SPC}$ to read the result from the FPU and transfer it to its destination. These Read cycles from the FPU are performed by the CPU while applying Status Code 1101 (Transfer Slave Operand).



Bit		Description
(0)	Q:	Set to "1" if an FPU TRAP (error) occurred. Cleared to '0" by a valid CMPf.
(2)	L:	Cleared to "0" by the FPU.
(6)	Z:	Set to "1" if the second operand is equal to the first operand. Otherwise it is cleared to "0".
(7)	N:	Set to "1" if the second operand is less than the first operand. Otherwise it is cleared to "0".
(15)	TS:	Set to "1" if the TRAP is (UND) and cleared to "0" if the TRAP is (FPU).

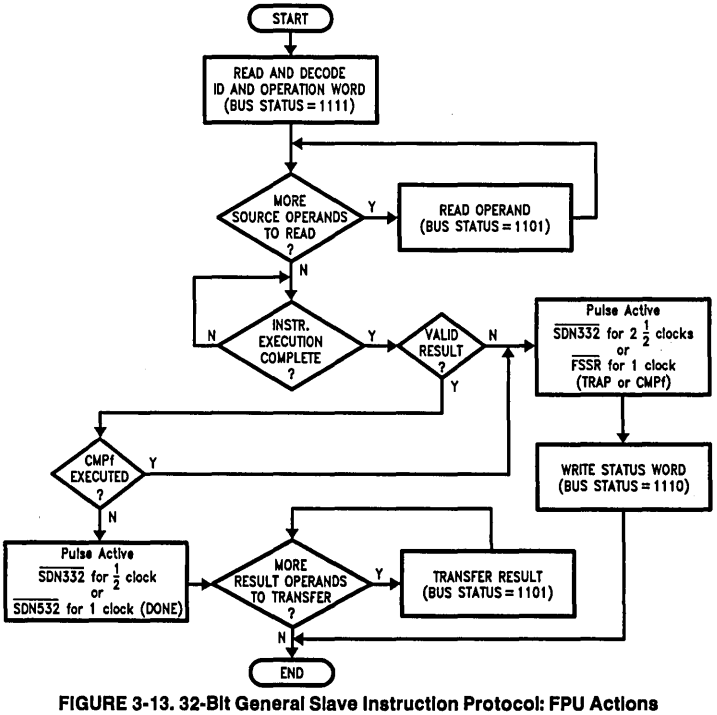
FIGURE 3-11. FPU Status Word Format

3.0 Functional Description (Continued)



TL/EE/9157–16

FIGURE 3-12. 16-Bit General Slave Instruction Protocol: FPU Actions



TL/EE/9157–17

FIGURE 3-13. 32-Bit General Slave Instruction Protocol: FPU Actions

3.0 Functional Description (Continued)

3.6.2 Early Done Algorithm

The NS32381 has the ability to modify the General Slave protocol sequences and to boost the performance of the FPU by 20% to 40%. This is called the Early Done Algorithm.

Early Done is defined by the fact that the destination of an instruction is an FPU register and that the instruction and range of operands cannot generate a TRAP (error). When these conditions are met the FPU will send a $\overline{\text{SDNXXX}}$ or $\overline{\text{SPC}}$ pulse after receiving all of the operands from the CPU and before executing the instruction. Hence this becomes an early done as compared to the General Slave Protocols.

In the case of the 16-bit Slave Protocol in which the CPU always reads the slave status word, the FPU will force all zeroes to be read. The CPU can then send the next instruction to the FPU and save the general protocol overhead. The FPU will start the new instruction immediately after finishing the previous instruction.

SFSR, CMPF and CMPL do not generate an Early Done.

3.6.3 Floating-Point Protocols

Table 3-3 gives the protocols followed for each floating-point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see section 2.2.3.

The Operand Class columns give the Access Classes for each general operand, defining how the addressing modes are interpreted by the CPU (see Series 32000 Instruction Set Reference Manual).

The Operand Issued columns show the sizes of the operands issued to the Floating-Point Unit by the CPU. "D" indicates a 32-bit Double Word. "i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). "f" indicates that the instruction specifies a floating-point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the FPU Status Word (*Figure 3-11*).

Any operand indicated as being of type "f" will not cause a transfer if the Register addressing mode is specified, because the Floating-Point Registers are physically on the Floating-Point Unit and are therefore available without CPU assistance.

4.0 Device Specifications

4.1 PIN DESCRIPTIONS

4.1.1 Supplies

The following is a brief description of all NS32381 pins.

V_{CC} — Power: +5V positive supply.

GND — Ground: Ground reference for both on-chip logic and drivers connected to output pins.

4.1.2 Input Signals

CLK — Clock: TTL-level clock signal.

*$\overline{\text{DDIN}}$ — Data Direction In: Active low. Status signal indicating the direction of data transfers during a bus cycle.

ST0–ST3 — Status: Bus cycle status code from CPU. ST0 is the least significant and rightmost bit.

1100— Reserved

1101— Transferring Operation Word or Operand

1110— Reading Status Word

1111— Broadcasting Slave ID

Note: The NS32332 generates four status lines and the NS32532 generates five. The user should connect the status lines as shown below:

NS32381	NS32332	NS32532
ST0	ST0	ST0
ST1	ST1	ST1
ST2	ST2	ST2
ST3	ST3	ST4

$\overline{\text{RST}}$ — Reset: Active low. Resets the last operation and clears the FSR register.

NOE — New Opcode Enable: Active high. This signal enables the new opcodes available in the NS32381.

PS0, PS1 — Protocol Select: Selects the slave protocol to be used. PS0 is the least significant and rightmost bit.

00—Selects 16-bit protocol.
01—Selects 32-bit protocol for NS32332.
10—Reserved.
11—Selects 32-bit protocol for NS32532.

4.1.3 Output Signals

$\overline{\text{SDN332}}$ — Slave Done 332: Active low. This signal is for use with the NS32332 CPU only. If held active for a half clock cycle and released this pin indicates the successful completion of a floating-point instruction by the FPU. Holding this pin active for two and a half clock cycles indicates TRAP or that the CMPf instruction has been executed.

$\overline{\text{SDN532}}$ — Slave Done 532: Active low. This signal is for use with the NS32532 CPU only. When active it indicates successful completion of a floating-point instruction by the FPU.

$\overline{\text{FSSR}}$ — Force Slave Status Read: Active low. This signal is for use with the NS32532 CPU only. When active it indicates TRAP or that the CMPf instruction has been executed.

4.1.4 Input/Output Signals

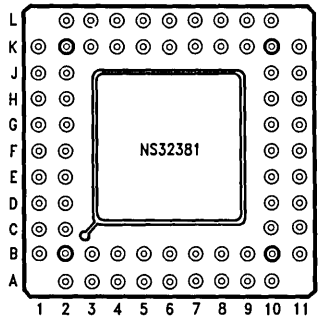
*D0–D31 — Data Bus: These are the 32 signal lines which carry data between the NS32381 and the CPU.

$\overline{\text{SPC}}$ — Slave Processor Control: Active low. This is the data strobe signal for slave transfers. For the 32-bit protocol, $\overline{\text{SPC}}$ is only an input signal.

*For the 16-bit Slave Protocol the upper sixteen data input signals (D16–D31) and $\overline{\text{DDIN}}$ should be left floating.

4.0 Device Specifications (Continued)

Connection Diagrams



TL/EE/9157–18

Bottom View

Order Number NS32381
See NS Package Number U68D

FIGURE 4-1. 68-Pin PGA Package
NS32381 Pinout Descriptions

Desc	Pin
V_{CC}	A2
D1	A3
D0	A4
PS1 (Note 1)	A5
GND	A6
GND	A7
CLK	A8
$\overline{RST}$	A9
Reserved (Note 2)	A10
Reserved (Note 2)	B1
D2	B2
D17	B3
D16	B4
PS0 (Note 1)	B5
GND	B6
NOE (Note 1)	B7
Reserved (Note 3)	B8
Reserved (Note 2)	B9
V_{CC}	B10
D15	B11
D18	C1
D3	C2
D31	C10
D14	C11
D19	D1
V_{CC}	D2
D30	D10
V_{CC}	D11
D4	E1
D20	E2
D13	E10
D29	E11
Reserved (Note 3)	F1
D5	F2

Desc	Pin
D28	F10
GND	F11
GND	G1
D21	G2
D12	G10
D27	G11
D6	H1
D22	H2
D11	H10
$\overline{SDN332}$	H11
D7	J1
D23	J2
$\overline{SPC}$	J10
$\overline{SDN532}$	J11
V_{CC}	K1
D8	K2
GND	K3
D26	K4
GND	K5
V_{CC}	K6
Reserved (Note 3)	K7
ST0	K8
ST1	K9
Reserved (Note 3)	K10
GND	K11
D24	L2
D25	L3
D9	L4
D10	L5
$\overline{DDIN}$	L6
V_{CC}	L7
ST2	L8
ST3	L9
$\overline{FSSR}$	L10

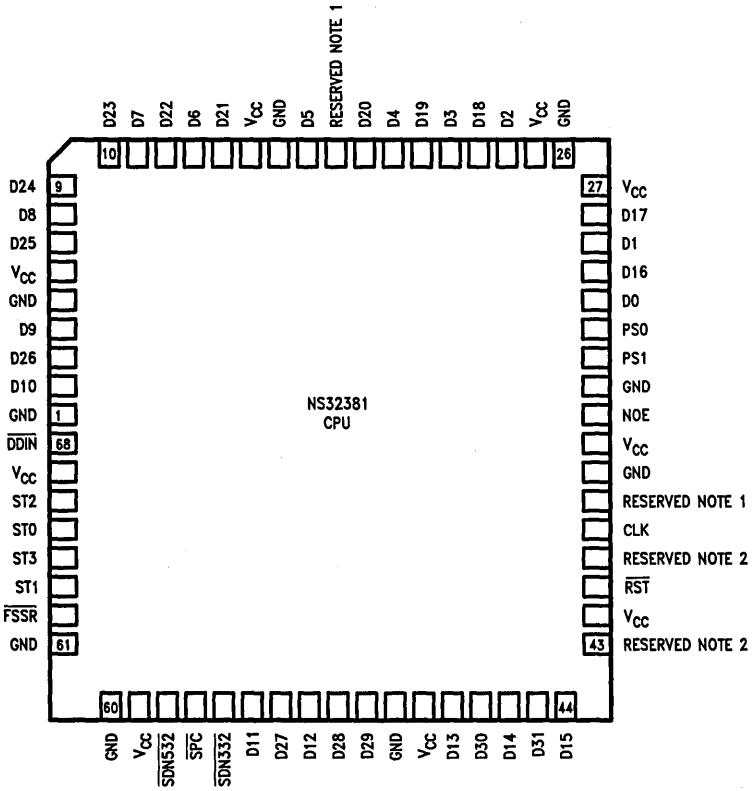
Note 1: CMOS input; never float.

Note 2: Pin should be grounded.

Note 3: Pin should be left floating.

4.0 Device Specifications (Continued)

Connection Diagrams (Continued)



TL/EE/9157–42

Bottom View

Order Number NS32381V-15, NS32381V-20, NS32381V-25 or NS32381V-30
See NS Package Number V68

FIGURE 4-2. 68-Pin Plastic Chip Carrier Package

Note 1: All these pins should be left open.

Note 2: All these pins should be grounded.

4.0 Device Specifications (Continued)

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Maximum Case Temperature	95°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7.0V
ESD Rating	2000V (in human body model)

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage*		2.0		V_{CC} +0.5	V
V_{IL}	Low Level Input Voltage*		−0.5		0.8	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low Level Output Voltage	I_{OL} = 2 mA			0.4	V
I_I	Input Load Current*	0 ≤ V_{IN} ≤ V_{CC}	−10.0		10.0	μA
V_{IH}	High Level Input Voltage for PS0, PS1, NOE		3.5		V_{CC} +0.5	V
V_{IL}	Low Level Input Voltage for PS0, PS1, NOE		−0.5		1.5	V
I_I	Input Load Current for PS0, PS1, NOE	0 ≤ V_{IN} ≤ V_{CC}	−100		100	μA
I_L	Leakage Current (Output and I/O Pins in TRI-STATE®/Input Mode)	0.4 ≤ V_{OUT} ≤ 2.4V	−20.0		20.0	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C, V_{CC} = 5V			300	mA
I_{CC}	Power Down Current	I_{OUT} = 0, T_A = 25°C, V_{CC} = 5V			60	mA

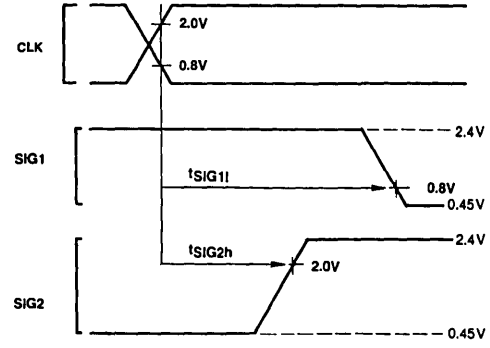
*Except PS0, PS1, NOE and Reserved pins.

Note: PS0, PS1 NOE pins have to be connected to either GND or V_{CC} (possible via resistor) as it is shown in *Figure 3-4a, 3-4b, 3-4c,* and *3-4d.*

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the Timing Specifications given in this section refer to 0.8V and 2.0V on all the input and output signals as illustrated in *Figures 4.3* and *4.4,* unless specifically stated otherwise.

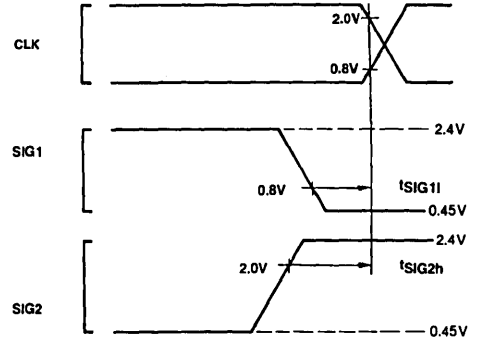


TL/EE/9157–19

FIGURE 4-3. Timing Specification Standard (Signal Valid after Clock Edge)

ABBREVIATIONS

L.E. — Leading Edge
R.E. — Rising Edge
T.E. — Trailing Edge
F.E. — Falling Edge



TL/EE/9157–20

FIGURE 4-4. Timing Specification Standard (Signal Valid before Clock Edge)

4.0 Device Specifications (Continued)

4.4.2 Timing Tables (Maximum times assume temperature range 0°C to 70°C)

4.4.2.1 Output Signal Propagation Delays for all CPUs (16-Bit Slave Protocol)

(Maximum times assume capacitive loading of 100 pF)

Symbol	Figure	Description	Reference/ Conditions	NS32381-15		NS32381-20		NS32381-25		Units
				Min	Max	Min	Max	Min	Max	
t_{SPCF_w}	4-18	$\overline{SPC}$ Pulse Width from FPU	At 0.8V (Both Edges)	$t_{CLK_p} - 10$	$t_{CLK_p} + 10$	$t_{CLK_p} - 10$	$t_{CLK_p} + 10$	$t_{CLK_p} - 10$	$t_{CLK_p} + 10$	ns
t_{SPCF_a}	4-18	$\overline{SPC}$ Output Active	After CLK R.E.		17		17		15	ns
$t_{SPCF_{ia}}$	4-18	$\overline{SPC}$ Output Inactive	After CLK R.E.		38		33		25	ns
t_{SPCF_f}(1)	4-18	$\overline{SPC}$ Output Floating	After CLK F.E.		35		30		25	ns

4.4.2.2 Output Signal Propagation Delays for the NS32008, NS32016 and NS32032 CPUs

Maximum times assumes capacitive loading of 100 pF

Symbol	Figure	Description	Reference/ Conditions	NS32381-15		NS32381-20		NS32381-25		Units
				Min	Max	Min	Max	Min	Max	
t_{D_v}	4-8	Data Valid (D0–D15)	After $\overline{SPC}$ L.E.		30				18	ns
t_{D_f}(1)	4-8	D0–D15 Floating	After $\overline{SPC}$ T.E.		30				30	ns

4.4.2.3 Output Signal Propagation Delays for the 32-Bit Slave Protocol NS32332 CPU

Maximum times assume capacitive loading of 100 pF unless otherwise specified

Symbol	Figure	Description	Reference/ Conditions	NS32381-15		Units
				Min	Max	
t_{D_v}	4-10	Data Valid	After $\overline{SPC}$ L.E.; 75 pF Cap. Loading		25	ns
t_{Dh}	4-10	Data Hold	After $\overline{SPC}$ T.E.	8		ns
t_{Df}(1)	4-10	Data Floating	After $\overline{SPC}$ T.E.		30	ns
t_{SDN_a}	4-12, 13	Slave Done Active	After CLK F.E.	3	28	ns
t_{SDN_h}	4-13	Slave Done Hold	After CLK R.E.		33	ns
t_{SDN_w}	4-12	Slave Done Pulse Width	At 0.8V (Both Edges)	$\frac{1}{2} t_{CLK_p} - 10$	$\frac{1}{2} t_{CLK_p} + 10$	ns
t_{SDN_f}(1)	4-12, 13	Slave Done Floating	After CLK R. E.		30	ns
t_{STRP_w}	4-13	Slave Done (TRAP) Pulse Width	At 0.8V (Both Edges)	$2\frac{1}{2} t_{CLK_p} - 10$	$2\frac{1}{2} t_{CLK_p} + 10$	ns

Note 1: Not 100% tested.

4.0 Device Specifications (Continued)

4.4.2.4 Output Signal Propagation Delays for the 32-Bit Slave Protocol NS32532 CPU
Maximum times assume capacitive loading of 50 pF

Symbol	Figure	Description	Reference/ Conditions	NS32381-						Units
				20		25		30		
				Min	Max	Min	Max	Min	Max	
t_{D_v}	4-14	Data Valid	After $\overline{SPC}$ L.E.		35		35		35	ns
t_{Dh}	4-14	Data Hold	After CLK R.E.	3		3		3		ns
t_{Df}(1)	4-14	Data Floating	After $\overline{SPC}$ T.E.		30		30		30	ns
t_{SD_a}	4-16	Slave Done Active	After CLK R.E.		35		25		20	ns
t_{SD_h}	4-16	Slave Done Hold	After CLK R.E.	2	33	2	25	2	20	ns
t_{SD_f}(1)	4-16	Slave Done Floating	After CLK R. E.		30		30		30	ns
t_{FSSR_a}	4-17	Forced Slave Status Read Active	After CLK R.E.		35		25		20	ns
t_{FSSR_h}	4-17	Forced Slave Status Read Hold	After CLK R.E.	2	33	2	25	2	20	ns
t_{FSSR_f}(1)	4-17	Forced Slave Status Read Floating	After CLK R.E.		30		30		30	ns

4.4.2.5 Input Signal Requirements with all CPUs

Symbol	Figure	Description	Reference/ Conditions	NS32381-								Units
				15		20		25		30		
				Min	Max	Min	Max	Min	Max	Min	Max	
t_{PWR}	4-5	Power-On Reset Duration	After CLK R.E.	30		30		30		30		μs
t_{RST_w}	4-6	Reset Pulse Width	At 0.8V (Both Edges)	64		64		64		64		t_{CLK_p}
t_{RST_s}	4-7	Reset Setup Time	Before CLK R.E.	10		14		12		11		ns
t_{RST_h}	4-7	Reset Hold	After CLK R.E.	0		0		0		0		ns

4.4.2.6 Input Signal Requirements with the NS32008, NS32016, NS32032 CPUs

Symbol	Figure	Description	Reference/ Conditions	NS32381-15		NS32381-20		NS32381-25		Units
				Min	Max	Min	Max	Min	Max	
t_{S_s}	4-8	Status (ST0–ST1) Setup	Before $\overline{SPC}$ L.E.	20		20		15		ns
t_{S_h}	4-8	Status (ST0–ST1) Hold	After $\overline{SPC}$ L.E.	20		20		17		ns
t_{D_s}	4-9	Data Setup (D0–D15)	Before $\overline{SPC}$ T.E.	25		20		15		ns
t_{D_h}	4-9	Data Hold (D0–D15)	After $\overline{SPC}$ T.E.	20		20		15		ns
t_{SPC_w}	4-8	$\overline{SPC}$ Pulse Width from CPU	At 0.8V (Both Edges)	35		35		28		ns

Note 1: Not 100% tested.

4.0 Device Specifications (Continued)

4.4.2.7 Input Signal Requirements with the 32-Bit Slave Protocol NS32332 CPU

Symbol	Figure	Description	Reference/ Conditions	NS32381-15		Units
				Min	Max	
t_{ST_s}	4-11	Status Setup	Before $\overline{SPC}$ L.E.	20		ns
t_{ST_h}	4-11	Status Hold	After $\overline{SPC}$ L.E.	20		ns
t_{D_s}	4-11	Data Setup	Before $\overline{SPC}$ T.E.	20		ns
t_{D_h}	4-11	Data Hold	After $\overline{SPC}$ T.E.	20		ns
t_{SPC_w}	4-11	$\overline{SPC}$ Pulse Width	At 0.8V (Both Edges)	35		ns

4.4.2.8 Input Signal Requirements with the 32-Bit Slave Protocol NS32532 CPU

Symbol	Figure	Description	Reference/ Conditions	NS32381						Units
				20		25		30		
				Min	Max	Min	Max	Min	Max	
t_{ST_s}	4-15	Status Setup	Before CLK (T2) R.E.	25		20		20		ns
t_{ST_h}	4-15	Status Hold	After CLK (T2) R.E.	20		10		10		ns
t_{DDIN_s}	4-15	Data Direction In Setup	Before $\overline{SPC}$ L.E.	0		0		0		ns
t_{DDIN_h}	4-15	Data Direction In Hold	After $\overline{SPC}$ T.E.	10		10		10		ns
t_{D_s}	4-15	Data Setup	Before $\overline{SPC}$ T.E.	6		6		4		ns
t_{D_h}	4-15	Data Hold	After $\overline{SPC}$ T.E.	20		10		10		ns
t_{SPC_s}	4-14, 15	$\overline{SPC}$ Setup	Before CLK R.E.	20		20		20		ns
t_{SPC_h}	4-14, 15	$\overline{SPC}$ Hold	After CLK R.E.	0		0		0		ns

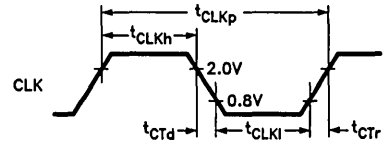
4.4.2.9 Clocking Requirements with all CPUs

Symbol	Figure	Description	Reference/ Conditions	NS32381								Units
				15		20		25		30		
				Min	Max	Min	Max	Min	Max	Min	Max	
t_{CLK_h}	4-4	Clock High Time	At 2.0 V (Both Edges)	25	1000	20	1000	16	1000	13	1000	ns
t_{CLK_l}	4-4	Clock Low Time	At 0.8V (Both Edges)	25	DC	20	DC	16	DC	13	DC	ns
t_{CT_r}(1)	4-4	Clock Rise Time	Between 0.8V and 2.0V		7		5		4		3	ns
t_{CT_d}(1)	4-4	Clock Fall Time	Between 2.0V and 0.8V		7		5		4		3	ns
t_{CLK_p}	4-4	Clock Period	CLK R.E. to Next CLK R.E.	66	DC	50	DC	40	DC	33.3	DC	ns

Note 1: Not 100% tested.

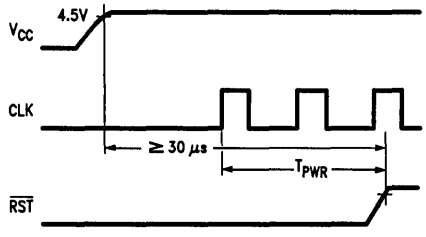
4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams



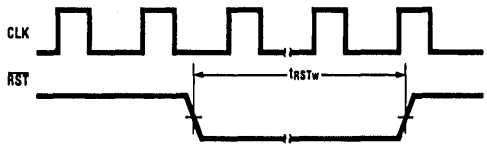
TL/EE/9157-21

FIGURE 4-5. Clock Timing



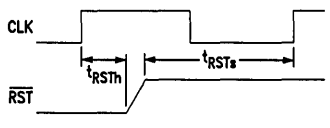
TL/EE/9157-22

FIGURE 4-6. Power-On Reset



TL/EE/9157-23

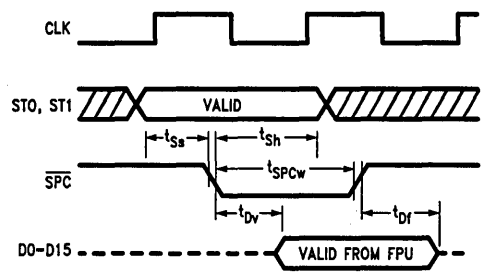
FIGURE 4-7. Non-Power-On Reset



TL/EE/9157-24

FIGURE 4-8. $\overline{RST}$ Release Timing

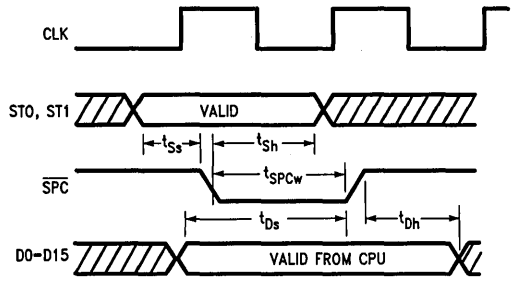
Note: The rising edge of $\overline{RST}$ must occur while CLK is high, as shown.



TL/EE/9157-25

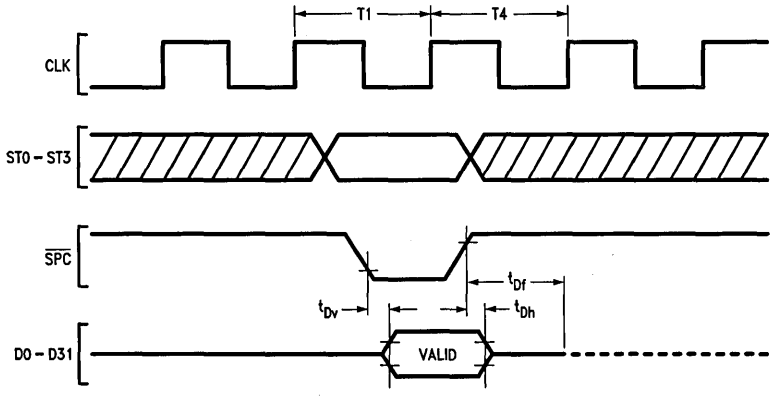
FIGURE 4-9. Read Cycle from FPU (NS32008, NS32016, NS32032 CPUs)

4.0 Device Specifications (Continued)



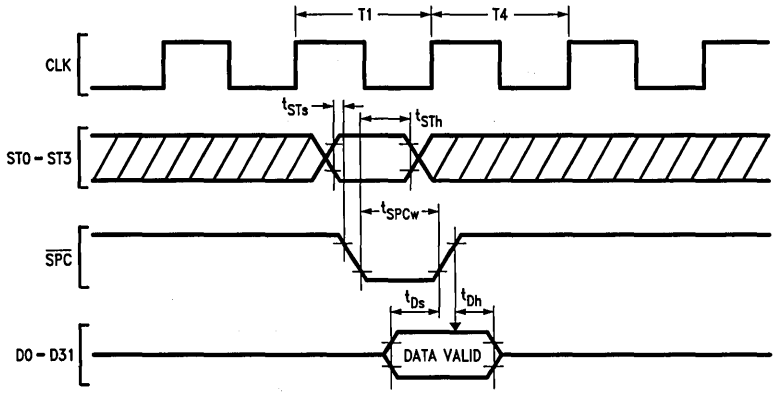
TL/EE/9157-26

FIGURE 4-10. Write Cycle to FPU (NS32008, NS32016, NS32032 CPUs)



TL/EE/9157-27

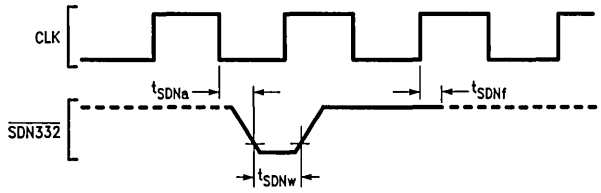
FIGURE 4-11. Read Cycle from FPU (NS32332 CPU)



TL/EE/9157-28

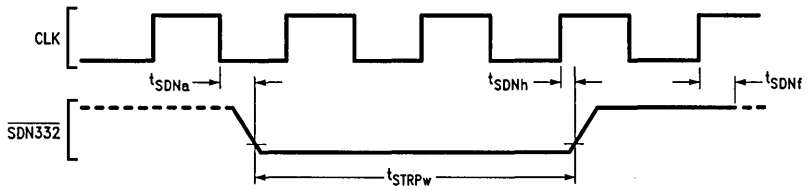
FIGURE 4-12. Write Cycle to FPU (NS32332 CPU)

4.0 Device Specifications (Continued)



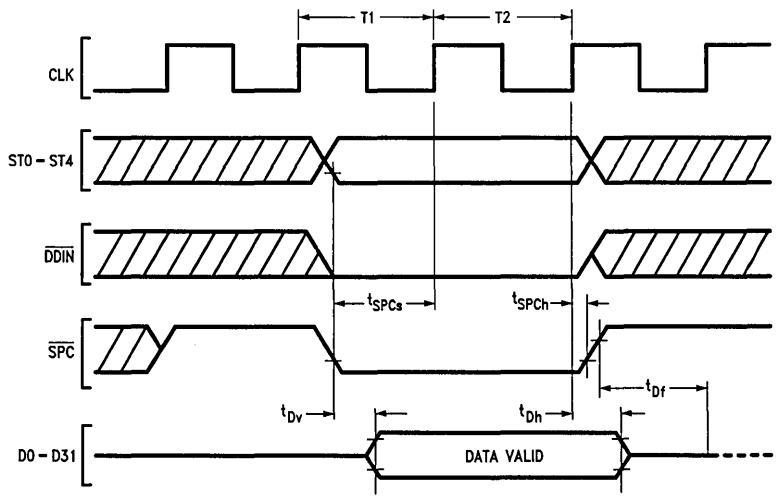
TL/EE/9157-29

FIGURE 4-13. $\overline{SDN332}$ Timing (NS32332 CPU)



TL/EE/9157-30

FIGURE 4-14. $\overline{SDN332}$ (TRAP) Timing (NS32332 CPU)

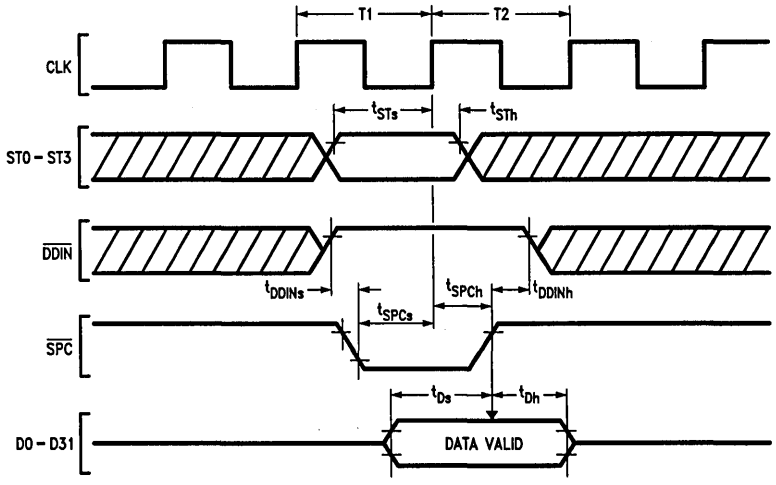


TL/EE/9157-31

FIGURE 4-15. Read Cycle from FPU (NS32532 CPU)

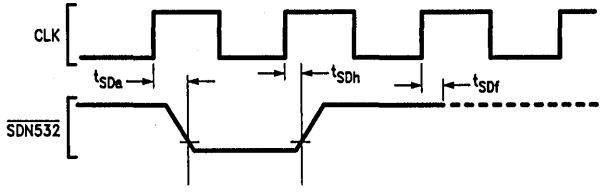
3

4.0 Device Specifications (Continued)



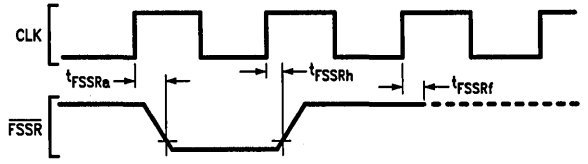
TL/EE/9157-32

FIGURE 4-16. Write Cycle to FPU (NS32532 CPU)



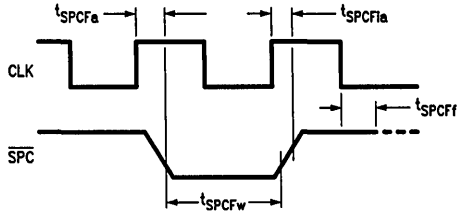
TL/EE/9157-33

FIGURE 4-17. SDN532 Timing (NS32532 CPU)



TL/EE/9157-34

FIGURE 4-18. FSSR Timing (NS32532 CPU)



TL/EE/9157-35

FIGURE 4-19. SPC Pulse from FPU

Appendix A

NS32381 PERFORMANCE ANALYSIS

The following performance numbers were taken from simulations using the 381 SIMPLE model. The timing terms have been designed to provide performance numbers which are CPU independent. Numbers were obtained from SIMPLE simulations, taking the average execution times using 'typical' operands.

Listed below are definitions of the timing terms:

EXT — (EXecution Time) This is the time from the last data sent to the FPU, until the early DONE is issued. (FPU Pipe is empty)

EDD — (Early Done Delta) This is the time from when the early DONE is issued until the execution of the next instruction may start.

Provided that the CPU can transfer the ID/OPCODE and any operands to the FPU during the EDD time, the average system execution time for an instruction (keeping the FPU pipe filled) is: EXT + EDD.

The system execution time for a single FPU instruction with FPU register destination and early done is: EXT plus the protocol time. (FPU pipe is initially empty)

Instruction	EXT*	EDD*	Total*
LFSR any, reg	5	8	13
MOVF any, reg	5	6	11
MOVL any, reg	5	8	13
MOVif any, reg	5	45	50
MOVFL any, reg	9	6	15
ADDF any, reg	11	31	42
ADDL any, reg	11	31	42
SUBF any, reg	11	31	42
SUBL any, reg	11	31	42
MULF any, reg	11	20	31
MULL any, reg	11	27	38
DIVF any, reg	11	45	56
DIVL any, reg	11	59	70
POLYF any, any	15	46	61
POLYL any, any	15	53	68
DOTF any, any	15	46	61
DOTL any, any	15	53	68

*Measured in the number of clock cycles.

NS32381 PERFORMANCE ANALYSIS

The following instructions do not generate an early done. In this case, EXT is the time from the last data sent to the FPU, until the normal DONE is issued. (FPU Pipe is empty)

Instruction	EXT
SFSR reg, mem	7
MOVLF any, any	18
ROUNDfi any, mem	46
FLOORfi any, mem	46
TRUNCfi any, mem	46
CMPF any, any	17
CMPL any, any	17
ABSf any, any	9
NEGf any, any	9
SCALBf any, any	49
LOGBf any, any	36

National Semiconductor

NS32081-10/NS32081-15 Floating-Point Units

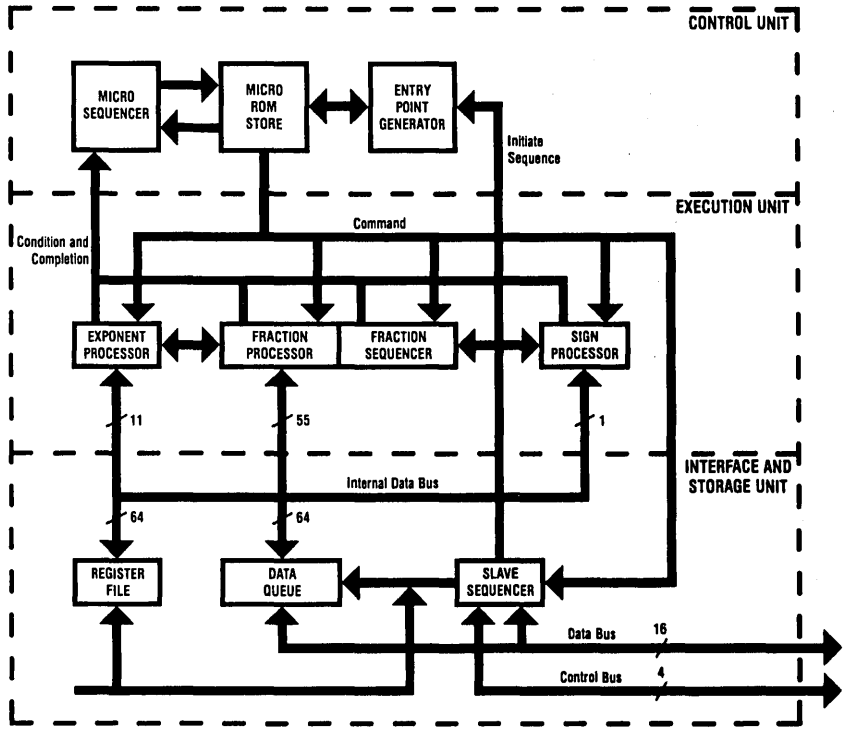
General Description

The NS32081 Floating-Point Unit functions as a slave processor in National Semiconductor's Series 32000® microprocessor family. It provides a high-speed floating-point instruction set for any Series 32000 family CPU, while remaining architecturally consistent with the full two-address architecture and powerful addressing modes of the Series 32000 micro-processor family.

Features

- Eight on-chip data registers
- 32-bit and 64-bit operations
- Supports proposed IEEE standard for binary floating-point arithmetic, Task P754
- Directly compatible with NS32016, NS32008 and NS32032 CPUs
- High-speed XMOS™ technology
- Single 5V supply
- 24-pin dual in-line package

Block Diagram



TL/EE/5234-1

Table of Contents

List of Illustrations

List of Tables

1.0 Product Introduction

The NS32081 Floating-Point Unit (FPU) provides high speed floating-point operations for the Series 32000 family, and is fabricated using National high-speed XMOS technology. It operates as a slave processor for transparent expansion of the Series 32000 CPU's basic instruction set. The FPU can also be used with other microprocessors as a peripheral device by using additional TTL interface logic. The NS32081 is compatible with the IEEE Floating-Point Formats by means of its hardware and software features.

1.1 OPERAND FORMATS

The NS32081 FPU operates on two floating-point data types—single precision (32 bits) and double precision (64 bits). Floating-point instruction mnemonics use the suffix F (Floating) to select the single precision data type, and the suffix L (Long Floating) to select the double precision data type.

A floating-point number is divided into three fields, as shown in *Figure 1-1*.

The F field is the fractional portion of the represented number. In Normalized numbers (Section 1.1.1), the binary point is assumed to be immediately to the left of the most significant bit of the F field, with an implied 1 bit to the left of the binary point. Thus, the F field represents values in the range $1.0 \leq x \leq 2.0$.

TABLE 1-1. Sample F Fields

F Field	Binary Value	Decimal Value
000...0	1.000...0	1.000...0
010...0	1.010...0	1.250...0
100...0	1.100...0	1.500...0
110...0	1.110...0	1.750...0
	↑	
	Implied Bit	

The E field contains an unsigned number that gives the binary exponent of the represented number. The value in the E field is biased; that is, a constant bias value must be subtracted from the E field value in order to obtain the true exponent. The bias value is $011\ldots11_2$, which is either 127 (single precision) or 1023 (double precision). Thus, the true exponent can be either positive or negative, as shown in Table 1-2.

TABLE 1-2. Sample E Fields

E Field	F Field	Represented Value
011...110	100...0	$1.5\times2^{-1} = 0.75$
011...111	100...0	$1.5\times2^{0} = 1.50$
100...000	100...0	$1.5\times2^{1} = 3.00$

Two values of the E field are not exponents. 11...11 signals a reserved operand (Section 2.1.3). 00...00 represents the number zero if the F field is also all zeroes, otherwise it signals a reserved operand.

The S bit indicates the sign of the operand. It is 0 for positive and 1 for negative. Floating-point numbers are in sign-magnitude form, that is, only the S bit is complemented in order to change the sign of the represented number.

1.1.1 Normalized Numbers

Normalized numbers are numbers which can be expressed as floating-point operands, as described above, where the E field is neither all zeroes nor all ones.

The value of a Normalized number can be derived by the formula:

$$(-1)^S \times 2^{(E\text{-}Bias)} \times (1 + F)$$

The range of Normalized numbers is given in Table 1-3.

1.1.2 Zero

There are two representations for zero—positive and negative. Positive zero has all-zero F and E fields, and the S bit is zero. Negative zero also has all-zero F and E fields, but its S bit is one.

1.1.3 Reserved Operands

The proposed IEEE Standard for Binary Floating-Point Arithmetic (Task P754) provides for certain exceptional forms of floating-point operands. The NS32081 FPU treats these forms as reserved operands. The reserved operands are:

- Positive and negative infinity
- Not-a-Number (NaN) values
- Denormalized numbers

Both Infinity and NaN values have all ones in their E fields. Denormalized numbers have all zeroes in their E fields and non-zero values in their F fields.

The NS32081 FPU causes an Invalid Operation trap (Section 2.1.2.2) if it receives a reserved operand, unless the operation is simply a move (without conversion). The FPU does not generate reserved operands as results.

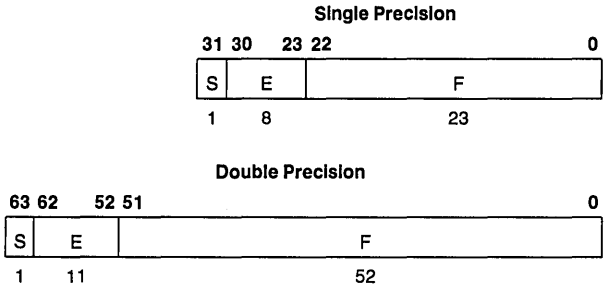


FIGURE 1-1. Floating-Point Operand Formats

1.0 Product Introduction (Continued)

TABLE 1-3. Normalized Number Ranges

	Single Precision	Double Precision
Most Positive	$2^{127}\times(2-2^{-23})$ $=3.40282346\times10^{38}$	$2^{1023}\times(2-2^{-52})$ $=1.7976931348623157\times10^{308}$
Least Positive	2^{-126} $=1.17549436\times10^{-38}$	2^{-1022} $=2.2250738585072014\times10^{-308}$
Least Negative	$-(2^{-126})$ $=-1.17549436\times10^{-38}$	$-(2^{-1022})$ $=-2.2250738585072014\times10^{-308}$
Most Negative	$-2^{127}\times(2-2^{-23})$ $=-3.40282346\times10^{38}$	$-2^{1023}\times(2-2^{-52})$ $=-1.7976931348623157\times10^{308}$

Note: The values given are extended one full digit beyond their represented accuracy to help in generating rounding and conversion algorithms.

1.1.4 Integers

In addition to performing floating-point arithmetic, the NS32081 FPU performs conversions between integer and floating-point data types. Integers are accepted or generated by the FPU as two's complement values of byte (8 bits), word (16 bits) or double word (32 bits) length.

1.1.5 Memory Representations

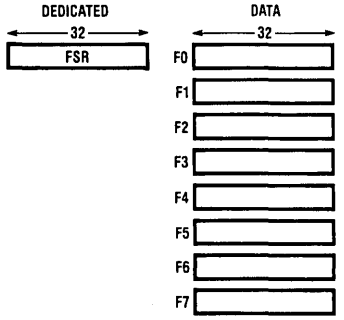
The NS32081 FPU does not directly access memory. However, it is cooperatively involved in the execution of a set of two-address instructions with its Series 32000 Family CPU. The CPU determines the representation of operands in memory.

In the Series 32000 family of CPUs, operands are stored in memory with the least significant byte at the lowest byte address. The only exception to this rule is the Immediate addressing mode, where the operand is held (within the instruction format) with the most significant byte at the lowest address.

2.0 Architectural Description

2.1 PROGRAMMING MODEL

The Series 32000 architecture includes nine registers that are implemented on the NS32081 Floating-Point Unit (FPU).



TL/EE/5234-4

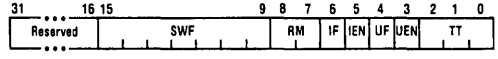
FIGURE 2-1. Register Set

2.1.1 Floating-Point Registers

There are eight registers (F0–F7) on the NS32081 FPU for providing high-speed access to floating-point operands. Each is 32 bits long. A floating-point register is referenced whenever a floating-point instruction uses the Register addressing mode (Section 2.2.2) for a floating-point operand. All other Register mode usages (i.e., integer operands) refer to the General Purpose Registers (R0–R7) of the CPU, and the FPU transfers the operand as if it were in memory. When the Register addressing mode is specified for a double precision (64-bit) operand, a pair of registers holds the operand. The programmer must specify the even register of the pair. The even register contains the least significant half of the operand and the next consecutive register contains the most significant half.

2.1.2 Floating-Point Status Register (FSR)

The Floating-Point Status Register (FSR) selects operating modes and records any exceptional conditions encountered during execution of a floating-point operation. *Figure 2-2* shows the format of the FSR.



TL/EE/5234-5

FIGURE 2-2. The Floating-Point Status Register

2.1.2.1 FSR Mode Control Fields

The FSR mode control fields select FPU operation modes. The meanings of the FSR mode control bits are given below.

Rounding Mode (RM): Bits 7 and 8. This field selects the rounding method. Floating-point results are rounded whenever they cannot be exactly represented. The rounding modes are:

00 Round to nearest value. The value which is nearest to the exact result is returned. If the result is exactly halfway between the two nearest values the even value (LSB=0) is returned.

01 Round toward zero. The nearest value which is closer to zero or equal to the exact result is returned.

2.0 Architectural Description (Continued)

10 Round toward positive infinity. The nearest value which is greater than or equal to the exact result is returned.

11 Round toward negative infinity. The nearest value which is less than or equal to the exact result is returned.

Underflow Trap Enable (UEN): Bit 3. If this bit is set, the FPU requests a trap whenever a result is too small in absolute value to be represented as a normalized number. If it is not set, any underflow condition returns a result of exactly zero.

Inexact Result Trap Enable (IEN): Bit 5. If this bit is set, the FPU requests a trap whenever the result of an operation cannot be represented exactly in the operand format of the destination. If it is not set, the result is rounded according to the selected rounding mode.

2.1.2.2 FSR Status Fields

The FSR Status Fields record exceptional conditions encountered during floating-point data processing. The meanings of the FSR status bits are given below:

Trap Type (TT): bits 0-2. This 3-bit field records any exceptional condition detected by a floating-point instruction. The TT field is loaded with zero whenever any floating-point instruction except LFSR or SFSR completes without encountering an exceptional condition. It is also set to zero by a hardware reset or by writing zero into it with the Load FSR (LFSR) instruction. Underflow and Inexact Result are always reported in the TT field, regardless of the settings of the UEN and IEN bits.

000 No exceptional condition occurred.

001 Underflow. A non-zero floating-point result is too small in magnitude to be represented as a normalized floating-point number in the format of the destination operand. This condition is always reported in the TT field and UF bit, but causes a trap only if the UEN bit is set. If the UEN bit is not set, a result of Positive Zero is produced, and no trap occurs.

010 Overflow. A result (either floating-point or integer) of a floating-point instruction is too great in magnitude to be held in the format of the destination operand. Note that rounding, as well as calculations, can cause this condition.

011 Divide by zero. An attempt has been made to divide a non-zero floating-point number by zero. Dividing zero by zero is considered an Invalid Operation instead (below).

100 Illegal Instruction. Two undefined floating-point instruction forms are detected by the FPU as being illegal. The binary formats causing this trap are:

xxxxxxxxxx0011xx10111110

xxxxxxxxxx1001xx10111110

101 Invalid Operation. One of the floating-point operands of a floating-point instruction is a Reserved operand, or an attempt has been made to divide zero by zero using the DIVf instruction.

110 Inexact Result. The result (either floating-point or integer) of a floating-point instruction cannot be represented exactly in the format of the destination operand, and a rounding step must alter it to fit. This condition is always reported in the TT field and IF bit unless any other exceptional condition has occurred in the same instruction. In this case, the TT field always contains the code for the other exception and the IF bit is not altered. A trap is caused by this condition only if the IEN bit is set; otherwise the result is rounded and delivered, and no trap occurs.

111 (Reserved for future use.)

Underflow Flag (UF): Bit 4. This bit is set by the FPU whenever a result is too small in absolute value to be represented as a normalized number. Its function is not affected by the state of the UEN bit. The UF bit is cleared only by writing a zero into it with the Load FSR instruction or by a hardware reset.

Inexact Result Flag (IF): Bit 6. This bit is set by the FPU whenever the result of an operation must be rounded to fit within the destination format. The IF bit is set only if no other error has occurred. It is cleared only by writing a zero into it with the Load FSR instruction or by a hardware reset.

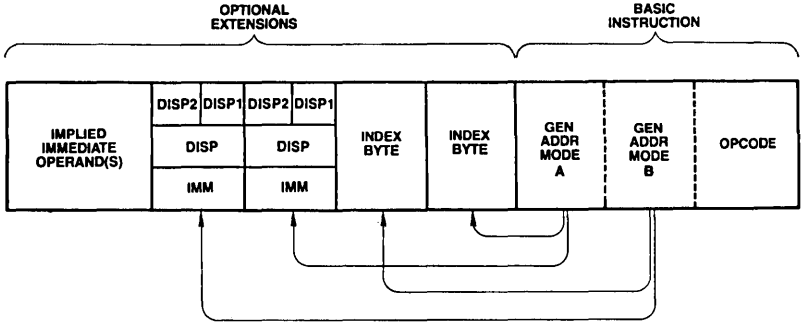
2.1.2.3 FSR Software Field (SWF)

Bits 9-15 of the FSR hold and display any information written to them (using the LFSR and SFSR instructions), but are not otherwise used by FPU hardware. They are reserved for use with NSC floating-point extension software.

2.2 INSTRUCTION SET

2.2.1 General Instruction Format

Figure 2-3 shows the general format of an Series 32000 instruction. The Basic Instruction is one to three bytes long



TL/EE/5234-6

FIGURE 2-3. General Instruction Format

2.0 Architectural Description (Continued)

and contains the opcode and up to two 5-bit General Addressing Mode (Gen) fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

The only form of extension issued to the NS32081 FPU is an Immediate operand. Other extensions are used only by the CPU to reference memory operands needed by the FPU.

Index Bytes appear when either or both Gen fields specify Scaled Index. In this case, the Gen field specifies only the Scale Factor (1, 2, 4 or 8), and the Index Byte specifies which General Purpose Register to use as the index, and which addressing mode calculation to perform before indexing. See *Figure 2-4*.

Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Disp/Imm field may contain one or two displacements, or one immediate value. The size of a Displacement field is encoded within the top bits of that field, as shown in *Figure 2-5*, with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the Opcode field. Both Displacement and Immediate fields are stored most significant byte first.

Some non-FPU instructions require additional, "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition.

2.2.2 Addressing Modes

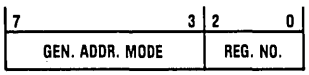
The Series 32000 Family CPUs generally access an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."

Addressing modes in the Series 32000 family are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode within the instruction which acts upon that variable. Extraneous data movement is therefore minimized.

Series 32000 Addressing Modes fall into nine basic types:

Register: In floating-point instructions, these addressing modes refer to a Floating-Point Register (F0–F7) if the operand is of a floating-point type. Otherwise, a CPU General Purpose Register (R0–R7) is referenced. See Section 2.1.1.

Register Relative: A CPU General Purpose Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.



TL/EE/5234–7

FIGURE 2-4. Index Byte Format

Memory Space: Identical to Register Relative above, except that the register used is one of the dedicated CPU registers PC, SP, SB or FP. These registers point to data areas generally needed by high-level languages.

Memory Relative: A pointer variable is found within the memory space pointed to by the CPU SP, SB or FP register. A displacement is added to that pointer to generate the Effective Address of the operand.

Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written. Floating-point operands as well as integer operands may be specified using Immediate mode.

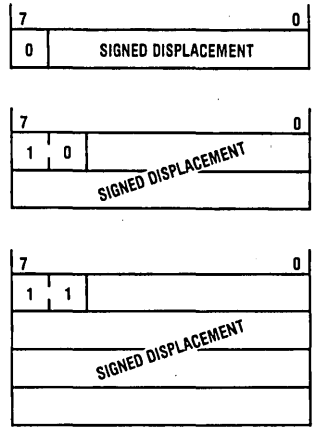
Absolute: The address of the operand is specified by a Displacement field in the instruction.

External: A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.

Top of Stack: The currently-selected CPU Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.

Scaled Index: Although encoded as an addressing mode, Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any General Purpose Register by 1, 2, 4 or 8 and adding it into the total, yielding the final Effective Address of the operand.

The following table, Table 2-1, is a brief summary of the addressing modes. For a complete description of their actions, see the Series 32000 Instruction Set Reference Manual.



TL/EE/5234–10

FIGURE 2-5. Displacement Encodings

2.0 Architectural Description (Continued)

TABLE 2-1. Series 32000 Family Addressing Modes

Encoding	Mode	Assembler Syntax	Effective Address
REGISTER			
00000	Register 0	R0 or F0	None: Operand is in the specified register.
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R7 or F7	
REGISTER RELATIVE			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
MEMORY SPACE			
11000	Frame memory	disp(FP)	Disp + Register; "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	*+disp	
MEMORY RELATIVE			
10000	Frame memory relative	disp2(disp1(FP))	Disp2+Pointer; Pointer found at address Disp1+Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1(SP))	
10010	Static memory relative	disp2(disp1(SB))	
IMMEDIATE			
10100	Immediate	value	None: Operand is issued from CPU instruction queue.
ABSOLUTE			
10101	Absolute	@disp	Disp.
EXTERNAL			
10110	External	EXT (disp1)+disp2	Disp2+Pointer; Pointer is found at Link Table Entry number Disp1.
TOP OF STACK			
10111	Top of Stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.
SCALED INDEX			
11100	Index, bytes	mode[Rn:B]	Mode + Rn.
11101	Index, words	mode[Rn:W]	Mode + 2 × Rn.
11110	Index, double words	mode[Rn:D]	Mode + 4 × Rn.
11111	Index, quad words	mode[Rn:Q]	Mode + 8 × Rn. "Mode" and "n" are contained within the Index Byte.
10011	(Reserved for Future Use)		

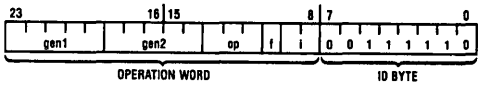
2.0 Architectural Description (Continued)

2.2.3 Floating-Point Instruction Set

The NS32081 FPU instructions occupy formats 9 and 11 of the Series 32000 Family instruction set (*Figure 2-6*). A list of all Series 32000 family instruction formats is found in the applicable CPU data sheet.

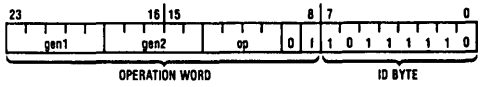
Certain notations in the following instruction description tables serve to relate the assembly language form of each instruction to its binary format in *Figure 2-6*.

Format 9



TL/EE/5234-11

Format 11



TL/EE/5234-12

FIGURE 2-6. Floating-Point Instruction Formats

The Format column indicates which of the two formats in *Figure 2-6* represents each instruction.

The Op column indicates the binary pattern for the field called "op" in the applicable format.

The Instruction column gives the form of each instruction as it appears in assembly language. The form consists of an instruction mnemonic in upper case, with one or more suffixes (i or f) indicating data types, followed by a list of operands (gen1, gen2).

An i suffix on an instruction mnemonic indicates a choice of integer data types. This choice affects the binary pattern in the i field of the corresponding instruction format (*Figure 2-6*) as follows:

Suffix i	Data Type	i Field
B	Byte	00
W	Word	01
D	Double Word	11

An f suffix on an instruction mnemonic indicates a choice of floating-point data types. This choice affects the setting of the f bit of the corresponding instruction format (*Figure 2-6*) as follows:

Suffix f	Data Type	f Bit
F	Single Precision	1
L	Double Precision (Long)	0

An operand designation (gen1, gen2) indicates a choice of addressing mode expressions. This choice affects the binary pattern in the corresponding gen1 or gen2 field of the instruction format (*Figure 2-6*). Refer to Table 2-1 for the options available and their patterns.

Further details of the exact operations performed by each instruction are found in the Series 32000 Instruction Set Reference Manual.

Movement and Conversion

The following instructions move the gen1 operand to the gen2 operand, leaving the gen1 operand intact.

Format	Op	Instruction		Description
11	0001	MOVf	gen1, gen2	Move without conversion
9	010	MOVLF	gen1, gen2	Move, converting from double precision to single precision.
9	011	MOVFL	gen1, gen2	Move, converting from single precision to double precision.
9	000	MOVif	gen1, gen2	Move, converting from any integer type to any floating-point type.
9	100	ROUNDfi	gen1, gen2	Move, converting from floating-point to the nearest integer.
9	101	TRUNCfi	gen1, gen2	Move, converting from floating-point to the nearest integer closer to zero.
9	111	FLOORfi	gen1, gen2	Move, converting from floating-point to the largest integer less than or equal to its value.

Note: The MOVLF instruction f bit must be 1 and the i field must be 10.
The MOVFL instruction f bit must be 0 and the i field must be 11.

Arithmetic Operations

The following instructions perform floating-point arithmetic operations on the gen1 and gen2 operands, leaving the result in the gen2 operand.

Format	Op	Instruction		Description
11	0000	ADDf	gen1, gen2	Add gen1 to gen2.
11	0100	SUBf	gen1, gen2	Subtract gen1 from gen2.
11	1100	MULf	gen1, gen2	Multiply gen2 by gen1.
11	1000	DIVf	gen1, gen2	Divide gen2 by gen1.
11	0101	NEGf	gen1, gen2	Move negative of gen1 to gen2.
11	1101	ABSf	gen1, gen2	Move absolute value of gen1 to gen2.

2.0 Architectural Description (Continued)

Comparison

The Compare instruction compares two floating-point values, sending the result to the CPU PSR Z and N bits for use as condition codes. See *Figure 3-7*. The Z bit is set if the gen1 and gen2 operands are equal; it is cleared otherwise. The N bit is set if the gen1 operand is greater than the gen2 operand; it is cleared otherwise. The CPU PSR L bit is unconditionally cleared. Positive and negative zero are considered equal.

Format	Op	Instruction		Description
11	0010	CMPf	gen1, gen2	Compare gen1 to gen2.

Floating-Point Status Register Access

The following instructions load and store the FSR as a 32-bit integer.

Format	Op	Instruction		Description
9	001	LFSR	gen1	Load FSR
9	110	SFSR	gen2	Store FSR

2.3 TRAPS

Upon detecting an exceptional condition in executing a floating-point instruction, the NS32081 FPU requests a trap by setting the Q bit of the status word transferred during the slave protocol (Section 3.5). The CPU responds by performing a trap using a default vector value of 3. See the Series 32000 Instruction Set Reference Manual and the applicable CPU data sheet for trap service details.

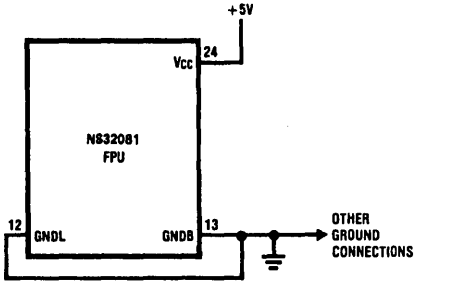
A trapped floating-point instruction returns no result, and does not affect the CPU Processor Status Register (PSR). The FPU displays the reason for the trap in the Trap Type (TT) field of the FSR (Section 2.1.2.2).

3.0 Functional Description

3.1 POWER AND GROUNDING

The NS32081 requires a single 5V power supply, applied on pin 24 (V_{CC}). See DC Electrical Characteristics table.

Grounding connections are made on two pins. Logic Ground (GNDL, pin 12) is the common pin for on-chip logic, and Buffer Ground (GNDB, pin 13) is the common pin for the output drivers. For optimal noise immunity, it is recommended that GNDL be attached through a single conductor directly to GNDB, and that all other grounding connections be made only to GNDB, as shown below (*Figure 3-1*).



TL/EE/5234-13

FIGURE 3-1. Recommended Supply Connections

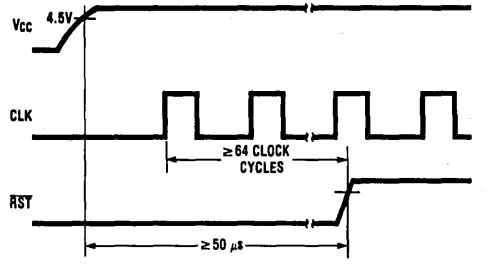
3.2 CLOCKING

The NS32081 FPU requires a single-phase TTL clock input on its CLK pin (pin 14). When the FPU is connected to a Series 32000 CPU, the CLK signal is provided from the CTTL pin of the NS32201 Timing Control Unit.

3.3 RESETTING

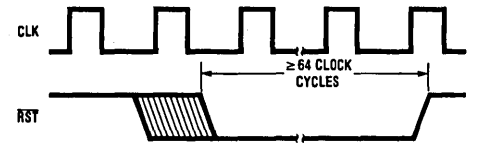
The $\overline{RST}$ pin serves as a reset for on-chip logic. The FPU may be reset at any time by pulling the $\overline{RST}$ pin low for at least 64 clock cycles. Upon detecting a reset, the FPU terminates instruction processing, resets its internal logic, and clears the FSR to all zeroes.

On application of power, $\overline{RST}$ must be held low for at least 50 μs after V_{CC} is stable. This ensures that all on-chip voltages are completely stable before operation. See *Figures 3-2* and *3-3*.



TL/EE/5234-14

FIGURE 3-2. Power-On Reset Requirements

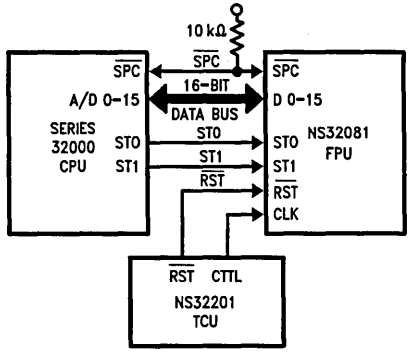


TL/EE/5234-15

FIGURE 3-3. General Reset Timing

3.4 BUS OPERATION

Instructions and operands are passed to the NS32081 FPU with slave processor bus cycles. Each bus cycle transfers either one byte (8 bits) or one word (16 bits) to or from the FPU. During all bus cycles, the $\overline{SPC}$ line is driven by the CPU as an active low data strobe, and the FPU monitors



TL/EE/5234-2

FIGURE 3-4. System Connection Diagram

3.0 Functional Description (Continued)

pins ST0 and ST1 to keep track of the sequence (protocol) established for the instruction being executed. This is necessary in a virtual memory environment, allowing the FPU to retry an aborted instruction.

3.4.1 Bus Cycles

A bus cycle is initiated by the CPU, which asserts the proper status on ST0 and ST1 and pulses $\overline{SPC}$ low. ST0 and ST1 are sampled by the FPU on the leading (falling) edge of the $\overline{SPC}$ pulse. If the transfer is from the FPU (a slave processor read cycle), the FPU asserts data on the data bus for the duration of the $\overline{SPC}$ pulse. If the transfer is to the FPU (a slave processor write cycle), the FPU latches data from the data bus on the trailing (rising) edge of the $\overline{SPC}$ pulse. *Figures 3-5* and *3-6* illustrate these sequences.

The direction of the transfer and the role of the bidirectional $\overline{SPC}$ line are determined by the instruction protocol being performed. $\overline{SPC}$ is always driven by the CPU during slave processor bus cycles. Protocol sequences for each instruction are given in Section 3.5.

3.4.2 Operand Transfer Sequences

An operand is transferred in one or more bus cycles. A 1-byte operand is transferred on the least significant byte of the data bus (D0–D7). A 2-byte operand is transferred on the entire bus. A 4-byte or 8-byte operand is transferred in consecutive bus cycles, least significant word first.

3.5 INSTRUCTION PROTOCOLS

3.5.1 General Protocol Sequence

Slave Processor instructions have a three-byte Basic Instruction field, consisting of an ID byte followed by an Operation Word. See Section 2.2.3 for FPU instruction encodings. The ID Byte has three functions:

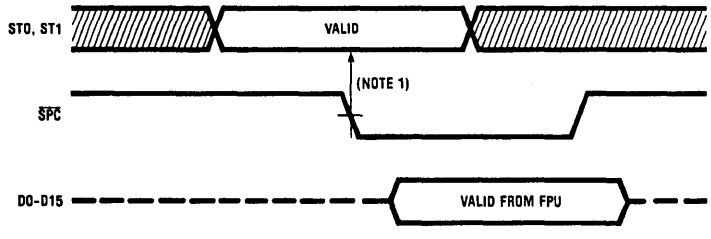
1) It identifies the instruction to the CPU as being a Slave Processor instruction.

2) It specifies which Slave Processor will execute it.

3) It determines the format of the following Operation Word of the instruction.

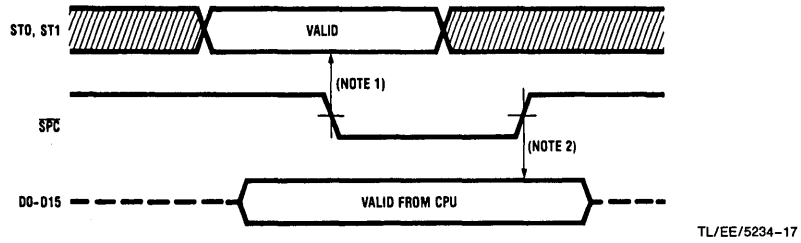
Upon receiving a Slave Processor instruction, the CPU initiates the sequence outlined in Table 3-2. While applying Status Code 11 (Broadcast ID. Table 3-1), the CPU transfers the ID Byte on the least significant half of the Data Bus (D0–D7). All Slave Processors input this byte and decode it. The Slave Processor selected by the ID Byte is activated, and from this point the CPU is communicating only with it. If any other slave protocol was in progress (e.g., an aborted Slave instruction), this transfer cancels it.

The CPU next sends the Operation Word while applying Status Code 01 (Transfer Slave Operand, Table 3-1). Upon receiving it, the FPU decodes it, and at this point both the CPU and the FPU are aware of the number of operands to be transferred and their sizes. The Operation Word is swapped on the Data Bus; that is, bits 0–7 appear on pins D8–D15, and bits 8–15 appear on pins D0–D7.



Note 1: FPU samples CPU status here.

FIGURE 3-5. Slave Processor Read Cycle



Note 1: FPU samples CPU status here.

Note 2: FPU samples data bus here.

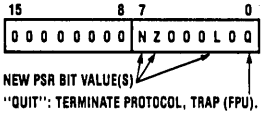
FIGURE 3-6. Slave Processor Write Cycle

3.0 Functional Description (Continued)

Using the Addressing Mode fields within the Operation Word, the CPU starts fetching operands and issuing them to the FPU. To do so, it references any Addressing Mode extensions appended to the FPU instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the Slave Processor. The Status Code applied is 01 (Transfer Slave Processor Operand, Table 3-1).

After the CPU has issued the last operand, the FPU starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low. To allow for this, the CPU releases the $\overline{SPC}$ signal, causing it to float. $\overline{SPC}$ must be held high by an external pull-up resistor.

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a Status Word from the FPU, applying Status Code 10. This word has the format shown in *Figure 3-7*. If the Q bit ("Quit", Bit 0) is set, this indicates that an error has been detected by the FPU. The CPU will not continue the protocol, but will immediately trap through the Slave vector in the Interrupt Table. If the instruction being performed is CMPf (Section 2.2.3) and the Q bit is not set, the CPU loads Processor Status Register (PSR) bits N, Z and L from the corresponding bits in the Status Word. The NS32081 FPU always sets the L bit to zero.



TL/EE/5234–18

FIGURE 3-7. FPU Protocol Status Word Format

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The Read cycles from the FPU are performed by the CPU while applying Status Code 01 (Section 4.1.2).

TABLE 3-1. General Instruction Protocol

Step	Status	Action
1	11	CPU sends ID Byte.
2	01	CPU sends Operation Word.
3	01	CPU sends required operands.
4	XX	FPU starts execution.
5	XX	FPU pulses $\overline{SPC}$ low.
6	10	CPU reads Status Word.
7	01	CPU reads result (if any).

3.5.2 Floating-Point Protocols

Table 3-2 gives the protocols followed for each floating-point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Section 2.2.3.

The Operand Class columns give the Access Classes for each general operand, defining how the addressing modes are interpreted by the CPU (see Series 32000 Instruction Set Reference Manual).

The Operand Issued columns show the sizes of the operands issued to the Floating-Point Unit by the CPU. "D" indicates a 32-bit Double Word. "i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). "f" indicates that the instruction specifies a floating-point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the Slave Processor Status Word (*Figure 3-7*).

Any operand indicated as being of type "f" will not cause a transfer if the Register addressing mode is specified, because the Floating-Point Registers are physically on the Floating-Point Unit and are therefore available without CPU assistance.

TABLE 3-2. Floating Point Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none

D = Double Word

i = Integer size (B, W, D) specified in mnemonic.

f = Floating-Point type (F, L) specified in mnemonic.

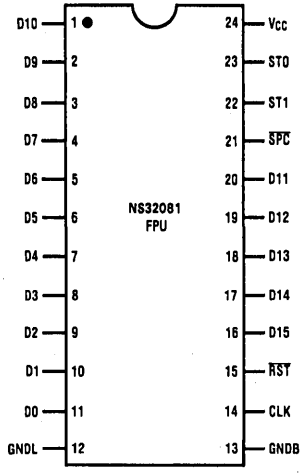
N/A = Not Applicable to this Instruction.

4.0 Device Specifications

4.1 PIN DESCRIPTIONS

The following are brief descriptions of all NS32081 FPU pins. The descriptions reference the relevant portions of the Functional Description, Section 3.

Dual-In-Line Package



TL/EE/5234–3

Top View

FIGURE 4-1. Connection Diagram

Order Number NS32081D-10 or NS32081D-15
See NS Package Number D24C

Order Number NS32081N-10 or NS32081N-15
See NS Package Number N24A

4.1.1 Supplies

Power (V_{CC}): +5V positive supply. Section 3.1.

Logic Ground (GNDL): Ground reference for on-chip logic. Section 3.1.

Buffer Ground (GNDB): Ground reference for on-chip drivers connected to output pins. Section 3.1.

4.1.2 Input Signals

Clock (CLK): TTL-level clock signal.

Reset ($\overline{RST}$): Active low. Initiates a Reset, Section 3.3.

Status (ST0, ST1): Input from CPU. ST0 is the least significant bit. Section 3.4 encodings are:

00—(Reserved)
01—Transferring Operation Word or Operand
10—Reading Status Word
11—Broadcasting Slave ID

4.1.3 Input/Output Signals

Slave Processor Control ($\overline{SPC}$): Active low. Driven by the CPU as the data strobe for bus transfers to and from the NS32081 FPU, Section 3.4. Driven by the FPU to signal completion of an operation, Section 3.5.1. Must be held high with an external pull-up resistor while floating.

Data Bus (D0–D15): 16-bit bus for data transfer. D0 is the least significant bit. Section 3.4.

4.2 ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7.0V
Power Dissipation	1.5W

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	HIGH Level Input Voltage		2.0		V_{CC} +0.5	V
V_{IL}	LOW Level Input Voltage		−0.5		0.8	V
V_{OH}	HIGH Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	LOW Level Output Voltage	I_{OL} = 4 mA			0.45	V
I_I	Input Load Current	$0 \le V_{IN} \le V_{CC}$	−10.0		10.0	μA
I_L	Leakage Current Output and I/O Pins in TRI-STATE/Input Mode	$0.45 \le V_{IN} \le 2.4V$	−20.0		20.0	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C		200	300	mA

4.0 Device Specifications (Continued)

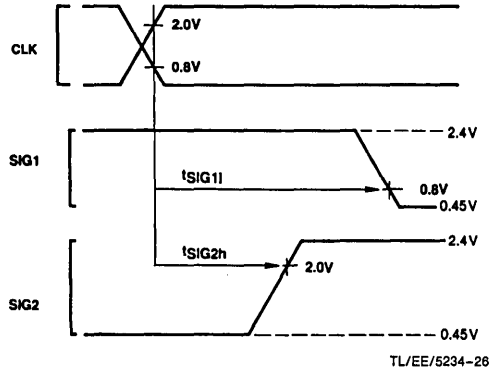
4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the Timing Specifications given in this section refer to 0.8V and 2.0V on all the input and output signals as illustrated in *Figures 4.2* and *4.3,* unless specifically stated otherwise.

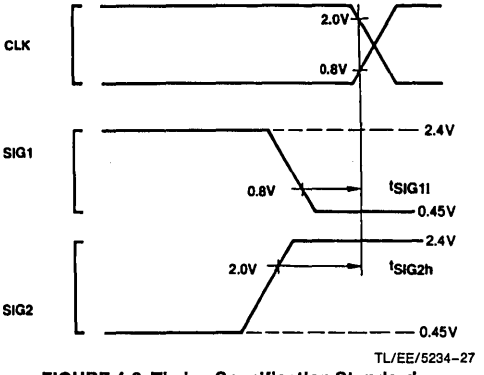
ABBREVIATIONS

L.E. — Leading Edge
T.E. — Trailing Edge
R.E. — Rising Edge
F.E. — Falling Edge



TL/EE/5234–26

FIGURE 4-2. Timing Specification Standard (Signal Valid After Clock Edge)



TL/EE/5234–27

FIGURE 4-3. Timing Specification Standard (Signal Valid Before Clock Edge)

4.0 Device Specifications (Continued)

4.4.2 Timing Tables

4.4.2.1 Output Signal Propagation Delays

Maximum times assume capacitive loading of 100 pF.

Name	Figure	Description	Reference/ Conditions	NS32081-10		NS32081-15		Units
				Min	Max	Min	Max	
t_{Dv}	4-7	Data Valid	After $\overline{SPC}$ L.E.		45		30	ns
t_{Df}	4-7	D_0–D_{15} Floating	After $\overline{SPC}$ T.E.		50	2	35	ns
t_{SPCFw}	4-9	$\overline{SPC}$ Pulse Width from FPU	At 0.8V (Both Edges)	$t_{CLKp} - 50$	$t_{CLKp} + 50$	$t_{CLKp} - 40$	$t_{CLKp} + 40$	ns
t_{SPCFl}	4-9	$\overline{SPC}$ Output Active	After CLK R.E.		55		38	ns
t_{SPCFh}	4-9	$\overline{SPC}$ Output Inactive	After CLK R.E.		55		38	ns
t_{SPCFnf}	4-9	$\overline{SPC}$ Output Nonforcing	After CLK F.E.		45		35	ns

4.4.2.2 Input Signal Requirements

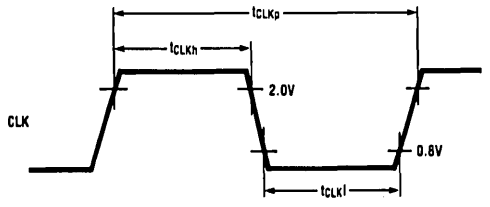
Name	Figure	Description	Reference/ Conditions	Min	Max	Min	Max	Units
t_{PWR}	4-5	Power Stable to $\overline{RST}$ R.E.	After V_{CC} Reaches 4.5V	50		50		μs
t_{RSTw}	4-6	$\overline{RST}$ Pulse Width	At 0.8V (Both Edges)	64		64		t_{CLKp}
t_{Ss}	4-7	Status (ST0–ST1) Setup	Before $\overline{SPC}$ L.E.	50		33		ns
t_{Sh}	4-7	Status (ST0–ST1) Hold	After $\overline{SPC}$ L.E.	40		35		ns
t_{Ds}	4-8	D0–D15 Setup Time	Before $\overline{SPC}$ T.E.	40		30		ns
t_{Dh}	4-8	D0–D15 Hold Time	After $\overline{SPC}$ T.E.	50		35		ns
t_{SPCw}	4-7	SPC Pulse Width from CPU	At 0.8V (Both Edges)	70		50		ns
t_{SPCs}	4-7	$\overline{SPC}$ Input Active	Before CLK R.E.	40		35		ns
t_{SPCh}	4-7	$\overline{SPC}$ Input Inactive	After CLK R.E.	0		0		ns
t_{RSTs}	4-10	$\overline{RST}$ Setup	Before CLK F.E.	10		10		ns
t_{RSTh}	4-10	$\overline{RST}$ R.E. Delay	After CLK R.E.	0		0		ns

4.4.2.3 Clocking Requirements

Name	Figure	Description	Reference/ Conditions	Min	Max	Min	Max	Units
t_{CLKh}	4-4	Clock High Time	At 2.0V (Both Edges)	42	1000	27	1000	ns
t_{CLKl}	4-4	Clock Low Time	At 0.8V (Both Edges)	42	1000	27	1000	ns
t_{CLKp}	4-4	Clock Period	CLK R.E. to Next CLK R.E.	100	2000	66		ns

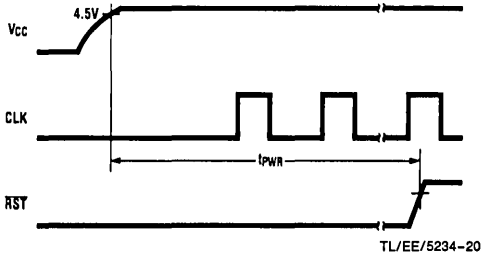
4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams



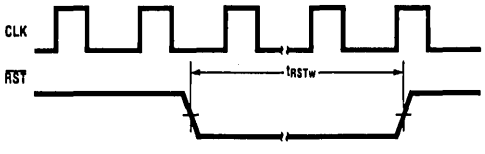
TL/EE/5234-19

FIGURE 4-4. Clock Timing



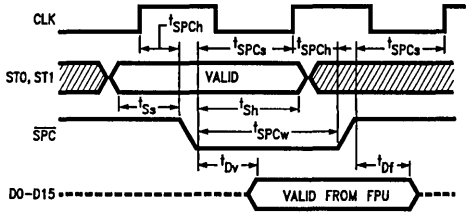
TL/EE/5234-20

FIGURE 4-5. Power-On Reset



TL/EE/5234-21

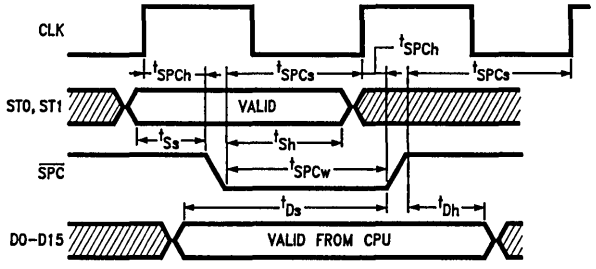
FIGURE 4-6. Non-Power-On Reset



TL/EE/5234-22

FIGURE 4-7. Read Cycle from FPU

Note: $\overline{SPC}$ pulse must be (nominally) 1 clock wide when writing into FPU.



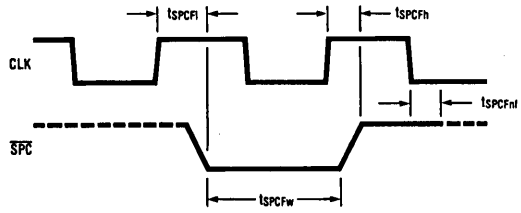
TL/EE/5234-23

FIGURE 4-8. Write Cycle to FPU

Note: $\overline{SPC}$ pulse may also be 2 clocks wide, but its edges must meet the t_{SPCs} and t_{SPCh} requirements with respect to CLK.

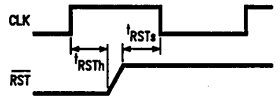
3

4.0 Device Specifications (Continued)



TL/EE/5234–24

FIGURE 4-9. $\overline{SPC}$ Pulse from FPU



TL/EE/5234–25

FIGURE 4-10. $\overline{RST}$ Release Timing

Note: The rising edge of $\overline{RST}$ must occur while CLK is high, as shown.

National Semiconductor

PRELIMINARY

NS32580-20/NS32580-25/NS32580-30 Floating Point Controller

General Description

The NS32580 Floating-Point Controller (FPC) is an interface device designed to couple the NS32532 Microprocessor with the Weitek WTL 3164 Floating-Point Data Path (FPDP). It is a new member of the Series 32000® family and it is fully upward compatible with the existing NS32081 floating-point software. Its performance reaches a peak of 10 Mflops when executing single and double precision ADD, SUB, MUL, and MAC instructions in a pipelined mode.

The FPC/FPDP supports the IEEE 754—1985 standard for Binary Floating-Point Arithmetic. An improved exception handling scheme allows enabling or disabling of each of the IEEE defined traps.

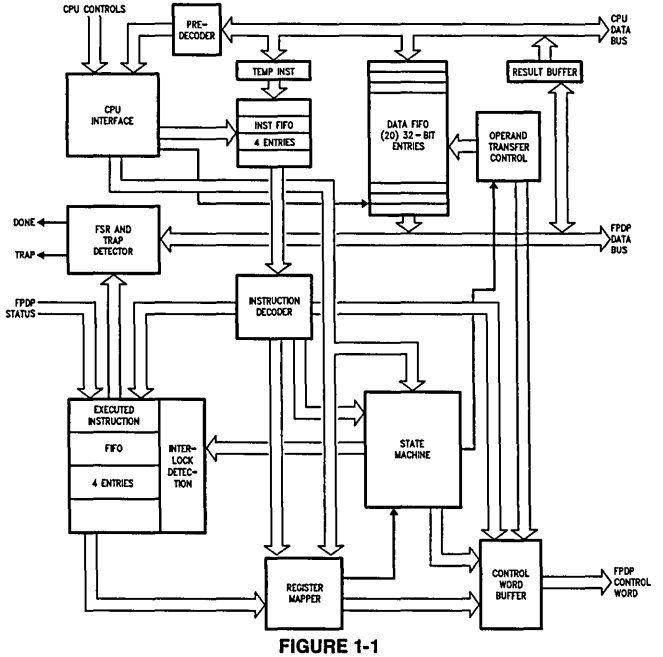
The NS32580 contains three FIFOs and a Floating-Point Status Register (FSR). It executes 18 instructions in conjunction with the WTL 3164 and with the NS32532 forms a tightly coupled computer cluster. The FPC/FPDP appears to the user as a single slave processing unit. The CPU and FPC/FPDP communication is handled automatically, and is user transparent.

The FPC is fabricated with National's advanced double-metal CMOS process and can operate at a frequency of 30 MHz.

Features

- Provides the NS32532 CPU with a complete interface controller for high-speed floating-point arithmetic
- 10 Mflops peak performance for single and double precision ADD, SUB, MUL and MAC instructions with the Weitek WTL 3164 FPDP
- Floating-point format compatible with IEEE 754—1985 standard
- Pipelined Slave Protocol with Data and Instruction FIFOs
- Improved exception handling including support of Infinities and Not a Number (NaN)
- Single (32-bit) and double (64-bit) precision operations
- Upward compatible with existing NS32081 software base
- 20 MHz, 25 MHz and 30 MHz operating frequencies
- 1 μm double-metal CMOS technology
- 172-pin PGA package

Block Diagram



TL/EE/9421-1

FIGURE 1-1

Table of Contents

List of Illustrations

List of Tables

1.0 Product Introduction

The NS32580 Floating-Point Controller (FPC) provides complete control for high speed floating-point operations between the NS32532 CPU and the Weitek WTL 3164 Floating-Point Data Path (FPDP). The FPC is fabricated using National high-speed CMOS technology and operates as a slave processor for transparent expansion of the Series 32000 CPU's basic instruction set. The NS32580 is compatible with the IEEE Floating-Point Formats by means of its hardware and software features.

1.1 IEEE FEATURES SUPPORTED

a. Basic floating-point number formats
b. Add, subtract, multiply, divide, sqrt, and compare operations
c. Conversions between different floating-point formats
d. Conversions between floating-point and integer formats
e. Round floating-point number to integer (round to nearest, round toward negative infinity and round toward zero, in double- or single-precision)
f. Exception signaling and handling (invalid operation, divide by zero, overflow, underflow and inexact)
g. Positive and negative infinity (Section 1.2.3)

Note: In addition to supporting the IEEE floating-point overflow, the NS32580 supports integer conversion overflow.

Also, the FPC-FPDP can accept Not-a-Number (NaN) as an operand and generate NaN as a result, but it does not conform to the IEEE 754-1985 Standard since it does not differentiate between signaling and quit Not-a-Number.

Note 1: ABSf NaN and NEGf NaN result in a signaling NaN but not a QUIT NaN.

Note 2: For NaN Op DNRM, where Op is ADDf, SUBf, MULf, DIVf and MACf, and with ROE = 1, the result is a QUIT NaN and not TRAP (INV).

Note 3: If ROE = 1, IVE = 0 and the operand is signalling NaN, the result is NaN with no TRAP (INV).

The remaining IEEE features can be supported in the software library. These items include:

a. Extended floating-point number formats
b. Mixed floating-point data formats
c. Conversions between basic formats, floating-point numbers and decimal strings
d. Remainder
e. Denormalized numbers

1.2 OPERAND FORMATS

The NS32580 FPC operates on two floating-point data types—single precision (32 bits) and double precision (64 bits). Floating-point instruction mnemonics use the suffix F (Floating) to select the single precision data type, and the suffix L (Long Floating) to select the double precision data type.

A floating-point number is divided into three fields, as shown in *Figure 1-2.*

The F field is the fractional portion of the represented number. In Normalized numbers (Section 1.2.1), the binary point is assumed to be immediately to the left of the most significant bit of the F field, with an implied 1 bit to the left of the binary point. Thus, the F field represents values in the range $1.0 \leq x < 2.0$, as shown in Table 1-1.

TABLE 1-1. Sample F Fields

F Field	Binary Value	Decimal Value
000 . . . 0	1.000 . . . 0	1.000 . . . 0
010 . . . 0	1.010 . . . 0	1.250 . . . 0
100 . . . 0	1.100 . . . 0	1.500 . . . 0
110 . . . 0	1.110 . . . 0	1.750 . . . 0
	↑	
	Implied Bit	

The E field contains an unsigned number that gives the binary exponent of the represented number. The value in the E field is biased; that is, a constant bias value must be subtracted from the E field value in order to obtain the true exponent. The bias value is $011 \ldots 11_2$, which is either 127 (single precision) or 1023 (double precision). Thus, the true exponent can be either positive or negative, as shown in Table 1-2.

TABLE 1-2. Sample E Fields

E Field	F Field	Represented Value
011 . . . 110	100 . . . 0	$1.5 \times 2^{-1} = 0.75$
011 . . . 111	100 . . . 0	$1.5 \times 2^{0} = 1.50$
100 . . . 000	100 . . . 0	$1.5 \times 2^{1} = 3.00$

Two values of the E field are not exponents. 11 . . . 11 signals Not-a-Number (NaN) or Infinity (Section 1.2.3). 00 . . . 00 represents the number zero (Section 1.2.2), if the F field is also all zeroes, otherwise it signals a reserved operand (Section 1.2.4).

The S bit indicates the sign of the operand. It is 0 for positive and 1 for negative. Floating-point numbers are in sign-magnitude form, that is, only the S bit is complemented in order to change the sign of the represented number.

1.2.1 Normalized Numbers

Normalized numbers are numbers which can be expressed as floating-point operands, as described above, where the E field is neither all zeroes nor all ones.

The value of a Normalized number can be derived by the formula:

$$(-1)^{S} \times 2^{(E\text{-}Bias)} \times (1 + F)$$

The range of Normalized numbers is given in Table 1-3.

1.2.2 Zero

There are two representatives for zero—positive and negative. Positive zero has all-zero F and E fields, and the S bit is zero. Negative zero also has all-zero F and E fields, but its S bit is one.

1.0 Product Introduction (Continued)

1.2.3 Reserved Operands

Infinity arithmetic is the limiting case of real arithmetic with operands of arbitrarily large magnitudes. The NS32580 does not treat infinity as a reserved operand and in ROUNDfi, TRUNCfi and FLOORfi instructions, when the operand is infinity, the FPC will return the TRAP "Integer overflow" instead of TRAP "Invalid Operation" with the Integer Conversion Overflow Flag, IOF, set to "1" and the Trap type to "2".

Another special case regarding infinity occurs when dividing infinity by zero. In this case NO TRAP "Divide by Zero" will be signaled and infinity will be returned as the result. See *Figures 1-3* and *1-4*.

The NS32580 FPC can treat NaN, not a number, either as a reserved operand (in NS32081 compatibility mode) or as not a reserved operand, depending upon the setting of the FSR ROE bit.

Denormalized numbers have all zeroes in their E fields and non-zero values in their F fields. They are treated as reserved operands except for those special cases listed in the compatibility table of Appendix A.

The NS32580 FPC causes an Invalid Operation Trap (Section 2.1.2.2) if it receives a reserved operand, unless the operation is simply a move (without conversion).

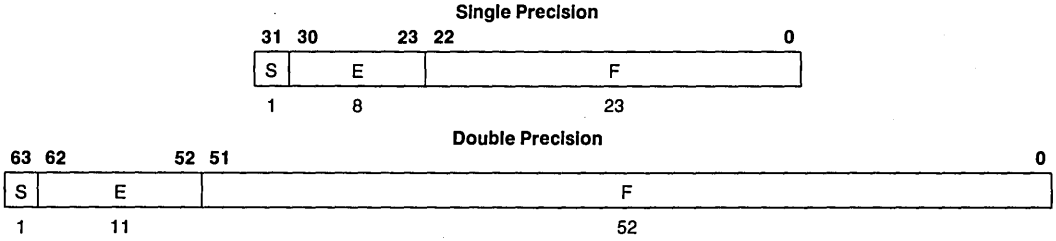


FIGURE 1-2. Floating-Point Operand Formats

TABLE 1-3. Normalized Number Ranges

	Single Precision	Double Precision
Most Positive	$2^{127} \times (2 - 2^{-23})$ $= 3.40282346 \times 10^{38}$	$2^{1023} \times (2 - 2^{-52})$ $= 1.7976931348623157 \times 10^{308}$
Least Positive	2^{-126} $= 1.17549436 \times 10^{-38}$	2^{-1022} $= 2.2250738585072014 \times 10^{-308}$
Least Negative	$-(2^{-126})$ $= -1.17549436 \times 10^{-38}$	$-(2^{-1022})$ $= -2.2250738585072014 \times 10^{-308}$
Most Negative	$-2^{127} \times (2 - 2^{-23})$ $= -3.40282346 \times 10^{38}$	$-2^{1023} \times (2 - 2^{-52})$ $= -1.7976931348623157 \times 10^{308}$

Note: The values given are extended one full digit beyond their represented accuracy to help in generating rounding and conversion algorithms.

E	F	Value	Name	Comments
255	Not 0	None	*NaN	ROE = 0 → Reserved Operand ROE = 1 → NaN Returned as Result
255	0	$(-1)^s$ * Infinity	*Infinity	Not a Reserved Operand
1–254	Any	$(-1)^s * 2^{e-127} * (1.f)$	Normalized Number	
0	Not 0	$(-1)^s * 2^{-126} * (0.f)$	*Denormalized Number	Reserved Operand
0	0	$(-1)^s * 0$	Zero	

FIGURE 1-3. Single-Precision Operand E and F Fields

E	F	Value	Name	Comments
2047	Not 0	None	*NaN	ROE = 0 → Reserved Operand ROE = 1 → NaN Returned as Result
2047	0	$(-1)^s$ * Infinity	*Infinity	Not a Reserved Operand
1–2046	Any	$(-1)^s * 2^{e-1023} * (1.f)$	Normalized Number	
0	Not 0	$(-1)^s * 2^{-1022} * (0.f)$	*Denormalized Number	Reserved Operand
0	0	$(-1)^s * 0$	Zero	

*Special cases listed in the compatibility table of Appendix A.

FIGURE 1-4. Double-Precision Operand E and F Fields

1.0 Product Introduction (Continued)

1.2.4 Integer Formats

The FPC-FPDP performs conversions between integer and floating point operands. Integers are accepted and generated by the FPC-FPDP as two's complement values of byte (8 bits), word (16 bits) or double-word (32 bits).

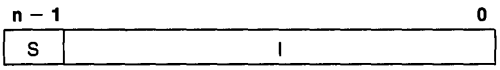


FIGURE 1-5. Integer Format

TABLE 1-4. Integer Fields

S	Value	Name
0	I	Positive Integer
1	$I - 2^n$	Negative Integer

n represents number of bits in the word, 8 for byte, 16 for word and 32 for double-word.

The FPDP supports only 32-bit integers, therefore, the FPC has to sign extend 8- and 16-bit integers prior to integer to floating-point number conversion.

In floating point to integer conversion, FPC has to check possible integer overflow, in case of 8- and 16-bit integer formats.

1.2.5 Memory Representations

The NS32580 FPC does not directly access memory. However, it is cooperatively involved in the execution of a set of two-address instructions with the NS32532 CPU. The CPU determines the representation of operands in memory.

In the Series 32000 family of CPUs, operands are stored in memory with the least significant byte at the lowest byte address. The only exception to this rule is the Immediate addressing mode, where the operand is held (within the instruction format) with the most significant byte at the lowest address.

2.0 Architectural Description

2.1 PROGRAMMING MODEL

The Series 32000 architecture implements nine floating point registers in the FPC; eight data registers and one floating-point status register.

2.1.1 Floating-Point Data Registers (L0–L7)

There are eight registers (L0–L7) in the FPC for providing high-speed access to floating-point operands. Each is 64 bits long. A floating-point register is referenced whenever a floating-point instruction uses the Register addressing mode (Section 2.2.2) for a floating-point operand. All other Register mode usages (i.e., integer operands) refer to the General Purpose Registers (R0–R7) of the CPU, and the FPC transfers the operand as if it were in memory.

Note: These registers are all upward compatible with the 32-bit NS32081 registers, (F0–F7), such that when the Register addressing mode is specified for a double precision (64-bit) operand, a pair of 32-bit registers holds the operand. The programmer specifies the even register of the pair which contains the least significant half of the operand and the next consecutive register contains the most significant half.

2.1.2 Floating-Point Status Register (FSR)

The Floating-Point Status Register selects operating modes and records any exceptional condition encountered during execution of a floating-point operation. The FPC FSR contains all the NS32081/NS32381 FSR bits and additional fields for better exception handling. The FSR is cleared to all zeros during reset.

DATA

← 32 Bits →	← 32 Bits →
F1 / L0 MSDW	F0 / L0 LSDW
L1 MSDW	L1 LSDW
F3 / L2 MSDW	F2 / L2 LSDW
L3 MSDW	L3 LSDW
F5 / L4 MSDW	F4 / L4 LSDW
L5 MSDW	L5 LSDW
F7 / L6 MSDW	F6 / L6 LSDW
L7 MSDW	L7 LSDW

LSDW → Least Significant Double Word

MSDW → Most Significant Double Word

FIGURE 2-1. Data Registers

2.1.2.1 FSR Mode Control Fields

The FSR mode control fields select FPC operation modes. The meanings of the FSR mode control bits are given below:

Rounding Mode (RM bit 8–7). This field selects the rounding method. Floating-point results are rounded whenever they cannot be represented exactly. The rounding modes are:

00 Round to nearest value. The value which is nearest to the exact result is returned. If the result is exactly halfway between the two nearest values the even value (lsb = 0) is returned.

01 Round toward zero. The nearest value which is closer to zero or equal to the exact result is returned.

10 Round toward positive infinity. The nearest value which is greater than or equal to the result is returned.

11 Round toward negative infinity. The nearest value which is less than or equal to the exact result is returned.

Underflow Trap Enable (UEN bit 3). If this bit is set, the FPC requests a trap whenever a result is too small in absolute value to be presented as a Normalized number. If it is not set, FPC returns a result of zero.

Inexact Result Trap Enable (IEN bit 5). If this bit is set, the FPC requests a trap whenever the result of an operation cannot be represented exactly in the operand format of the destination (and no other exception occurred in the same operation) or if the result of an operation overflows and the overflow trap is disabled. If IEN is not set, the result is rounded according to the selected rounding mode.

2.1.2.2 FSR Status Fields

The FSR Status Fields record exceptional conditions encountered during floating-point data processing. The meaning of the FSR status bits are given below:

Trap Type (TT bits 2–0). This 3-bit field indicates the reason for TRAP (FPU) requested by the FPC. The TT field is loaded with zero whenever any floating-point instruction except LFSR or SFSR completes without exception. It is also set to zero by a reset or by writing zero into it with the LFSR instruction. The TT field is updated regardless of the setting of the exception enable bits.

2.0 Architectural Description (Continued)

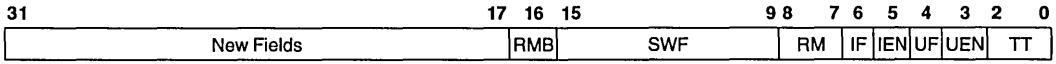


FIGURE 2-2. FSR (Compatible Fields)

000 No exceptional condition occurred.

001 Underflow. This condition occurs whenever a result is too close to zero to be represented as a Normalized number.

010 Overflow. This condition occurs whenever a result is too large in absolute value to be represented (float or integer).

011 Divide by Zero. This condition occurs whenever an attempt was made to divide a non-zero value by zero.

100 Illegal Instruction. An illegal or undefined Floating-Point instruction was passed to the FPC. If the T bit in the Status Word Register (SWR) is a "0", then it indicates that an illegal instruction was passed to the FPC. If the T bit in the SWR is a "1", then it indicates that an undefined instruction was passed to the FPC.

101 Invalid Operation. This condition occurs if:
1. NaN is used as a floating-point operand by any instruction except MOVf and the Reserved Operand Enable (ROE) bit in the FSR is disabled.
2. DNRM is used as a floating-point operand by any instruction except MOVf.
3. Both operands of the DIVf instruction are zero.
4. Sqrt when the floating-point number is negative.
5. Infinity plus negative infinity, infinity minus infinity.

110 Inexact Result. This condition occurs whenever the result of an operation cannot be exactly represented in the precision of the destination (and no other exception occurred in the same operation) or if the result of an operation overflows (floating-point or integer conversion overflow) and the overflow trap is disabled.

111 Reserved.

Underflow Flag (UF bit 4). This bit is set by the FPC whenever a result is too small in absolute value to be represented as a Normalized number. Its function is not affected by the state of the UEN bit. The UF bit is "sticky" therefore it can be cleared only by writing a zero into it with the Load FSR instruction or by a hardware reset.

Inexact Result Flag (IF bit 6). This bit is set by the FPC whenever the result of an operation must be rounded to fit within the destination format (and no other exception occurred in the same operation) or if the result of an operation overflows and the overflow trap is disabled. This situation applies both to floating-point and integer destinations. The IF bit is "sticky" therefore it is cleared only by writing a zero into it with the Load FSR instruction or by a hardware reset.

Register Modify Bit (RMB BIT 16). This bit is set by the FPC whenever writing to a floating-point data register. The RMB bit is cleared only by writing a zero with the LFSR instruction or by a hardware reset. This bit can be used in context switching to determine whether the FPC registers should be saved.

2.1.2.3 FSR Software Field (SWF)

Bits 15–9 of the FSR hold and display any information written to them using the LFSR and SFSR instructions, but are not otherwise used by FPC hardware. They are reserved for use with NSC floating-point extension software.

2.1.2.4 FSR New Fields

New fields were added to the FSR for better exception handling. In the FPC, the user can enable or disable each exception or combination of exceptions by using new "enable bits" implemented in the FSR. After reset the new fields are loaded to the default values (compatible with NS32081). Illegal Instruction always causes TRAP and can't be disabled. The bits are defined as follows:

CONTROL BITS

Reserved Operands Enable (ROE bit 17). If this bit is cleared, the FPC requests an Invalid Operation trap whenever a NaN has been detected by the FPC. When ROE is disabled, the FPC does not generate reserved operands as results. If the ROE is set then NaN will be returned as the result with no trap and the ROF bit is cleared. If Invalid Operation exception is disabled, the ROE bit is overwritten internally (the FPC does not change the ROE bit in the FSR) and the FPC can generate NaN as a result. ROE bit does not affect MOVf instruction.

Invalid Operation Enable (IVE bit 18). If this bit is cleared, the FPC requests a trap whenever the operation is invalid. If this bit is set to "1", the trap is disabled and if invalid operation occurred, NaN will be delivered as result.

Divide By Zero Enable (DZE bit 19). If this bit is cleared the FPC requests a trap whenever an attempt is made to divide by zero. If this bit is set the trap is disabled and if divide by zero occurred, infinity will be delivered as result.

Overflow Enable (OVE bit 20). If this bit is cleared, the FPC requests a trap whenever a floating-point result is too big in absolute value to be represented. If this bit is set, the overflow trap is disabled and if overflow occurred, Infinity or Maximum Number will be delivered as result.

Integer Conversion Overflow Enable (IOE bit 21). If this bit is cleared, the FPC requests a trap whenever an Integer result is too big to be represented. If this bit is set, the integer conversion overflow is disabled and if integer conversion overflow occurred, Max/Min integer will be delivered as result.

STATUS BITS

Reserved Operand Flag (ROF bit 22). This bit is set by the FPC whenever reserved operand DNRM or NaN (when ROE is cleared) is selected by the FPC. The ROF bit is "sticky" and can be cleared only by writing a zero with the Load FSR instruction or by a hardware reset.

Invalid Flag (IVF bit 23). This bit is set by the FPC whenever the operation is invalid. The IVF bit is "sticky" and can be cleared only by writing a zero with the Load FSR instruction or by a hardware reset.

Divide By Zero Flag (DZF bit 24). This bit is set by the FPC whenever an attempt is made to divide a non-zero value by zero. The DZF bit is "sticky" and can be cleared only by writing a zero with the Load FSR instruction or by a hardware reset.

2.0 Architectural Description (Continued)

31 ... 27	26	25	24	23	22	21	20	19	18	17	16
Reserved	IOF	OVF	DZF	IVF	ROF	IOE	OVE	DZE	IVE	ROE	RMB

FIGURE 2-3. New FSR Mode Control Fields

Overflow Flag (OVF bit 25). This bit is set by the FPC whenever a floating-point result is too large in absolute value to be represented. The OVF bit is "sticky" and can be cleared only by writing a zero with the Load FSR instruction or by a hardware reset.

Integer Conversion Overflow Flag (IOF bit 26). This bit is set by the FPC whenever an integer result is too large in absolute value to be represented. The IOF bit is "sticky" and can be cleared only by writing a zero with the Load FSR instruction or by a hardware reset.

Reserved Field

Bits 31–27 in the FSR are reserved by NSC for future use. User should not use this field.

2.1.2.5 FSR Default Values

During Reset the FSR is loaded to a default value (see Table 2-1). The default values for the FSR represent upward compatibility of the FPC-FPDP with the NS32081. The user can change the default values by loading the FSR register with new values.

TABLE 2-1. FSR Default State Summary

Bit Name	Default Value	Default State
TT (bits 2–0)	0	No exceptional condition occurred.
UEN (bit 3)	0	Underflow trap disabled.
UF (bit 4)	0	Underflow flag is cleared.
IEN (bit 5)	0	Inexact result trap disabled.
IF (bit 6)	0	Inexact flag is cleared.
RM (bits 8–7)	0	Round to nearest.
SWF (bits 15–9)	0	Undefined
RMB (bit 16)	0	RMB flag is cleared.
ROE(bit 17)	0	FPC requests a trap whenever an attempt is made to use reserved operand except for MOVf instruction.
IVE (bit 18)	0	FPC requests a trap whenever the operation is invalid.
DZE (bit 19)	0	FPC requests a trap whenever an attempt is made to divide by zero.
OVE (bit 20)	0	FPC requests a trap whenever a floating-point result is too big to be represented.

TABLE 2-1. FSR Default State Summary (Continued)

Bit Name	Default Value	Default State
IOE (bit 21)	0	FPC requests a trap whenever an integer conversion result is too big to be represented.
ROF (bit 22)	0	ROF flag is cleared.
IVF (bit 23)	0	IVF flag is cleared.
DZF (bit 24)	0	DZF flag is cleared.
OVF (bit 25)	0	OVF flag is cleared.
IOF (bit 26)	0	IOF flag is cleared.
RESERVED (bits 31–27)	0	Reserved field is cleared.

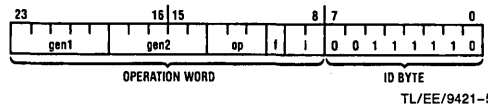
2.2 INSTRUCTION SET

2.2.1 Floating-Point Instruction Set

This section provides a description of the floating-point instructions executed by the FPC in conjunction with the CPU and the FPDP. These instructions form a small subset of the Series 32000 instruction set and their encodings use instruction formats 9, 11, and 12. A list of all the Series 32000 instructions as well as details on their formats and addressing modes can be found in the appropriate CPU data sheets.

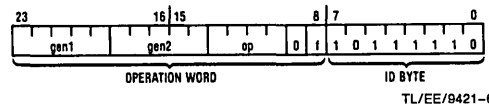
Certain notations in the following instruction description tables serve to relate the assembly language form of each instruction to its binary format in *Figure 2-4*.

Format 9



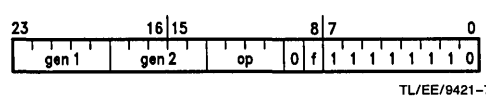
TL/EE/9421–5

Format 11



TL/EE/9421–6

Format 12



TL/EE/9421–7

FIGURE 2-4. Floating-Point Instruction Formats

2.0 Architectural Description (Continued)

The Format column indicates which of the three formats in *Figure 2-4* represents each instruction.

The Op column indicates the binary pattern for the field called "op" in the applicable format.

The Instruction column gives the form of each instruction as it appears in assembly language. The form consists of an instruction mnemonic in upper case, with one or more suffixes (i or f) indicating data types, followed by a list of operands (gen1, gen2).

An i suffix on an instruction mnemonic indicates a choice of integer data types. This choice affects the binary pattern in the i field of the corresponding instruction format as follows:

Suffix i	Data Type	i Field
B	Byte	00
W	Word	01
D	Double Word	11

An f suffix on an instruction mnemonic indicates a choice of floating-point data types. This choice affects the setting of the f bit of the corresponding instruction format as follows:

Suffix f	Data Type	f Bit
F	Single Precision	1
L	Double Precision (Long)	0

An operand designation (gen1, gen2) indicates a choice of addressing mode expressions. This choice affects the binary pattern in the corresponding gen1 or gen2 field of the instruction format.

Further details of the exact operations performed by each instruction are found in the Series 32000 Instruction Set Reference Manual.

Movement and Conversion

The following instructions move the gen1 operand to the gen2 operand, leaving the gen1 operand intact.

Format	Op	Instruction		Description
11	0001	MOVf	gen1, gen2	Move without conversion.
9	010	MOVLF	gen1, gen2	Move, converting from double precision to single precision.
9	011	MOVFL	gen1, gen2	Move, converting from single precision to double precision.
9	000	MOVif	gen1, gen2	Move, converting from any integer type to any floating-point type.
9	100	ROUNDfi	gen1, gen2	Move, converting from floating-point to the nearest integer.
9	101	TRUNCfi	gen1, gen2	Move, converting from floating-point to the nearest integer closer to zero.
9	111	FLOORfi	gen1, gen2	Move, converting from floating-point to the largest integer less than or equal to its value.

Note: The MOVLF instruction f bit must be 1 and the i field must be 10.
The MOVFL instruction f bit must be 0 and the i field must be 11.

Arithmetic Operations

The following instructions perform floating-point arithmetic operations on the gen1 and gen2 operands, leaving the result in the gen2 operand.

Format	Op	Instruction		Description
11	0000	ADDf	gen1, gen2	Add gen1 to gen2.
11	0100	SUBf	gen1, gen2	Subtract gen1 from gen2.
11	1100	MULf	gen1, gen2	Multiply gen2 by gen1.
11	1000	DIVf	gen1, gen2	Divide gen2 by gen1.
11	0101	NEGf	gen1, gen2	Move negative of gen1 to gen2.
11	1101	ABSf	gen1, gen2	Move absolute value of gen1 to gen2.
(N) 12	1010	MACf	gen1, gen2	Move (gen1*gen2) + L1 or F1 to L1 or F1 with two rounding errors.
(N) 12	0001	SQRTf	gen1, gen2	Move the square root of gen1 to gen2.

(N): Indicates NEW instruction.

Comparison

The compare instruction compares two floating-point operands, sending the result to the CPU PSR Z, N and L bits for use as condition codes.

Format	Opcode	Instruction		Description
11	0010	CMPf	gen1, gen2	Compare gen1 to gen2.

2.0 Architectural Description (Continued)

There are four possible results to the CMPf instruction (with normal operands):

Operands are equal	Z bit is set	N, L bits are cleared
Operand1 is less than Operand2		N, L, Z bits are cleared
Operand2 is less than Operand1	N bit is set	L, Z bits are cleared
Unordered (when at least one operand is NaN and ROE is set)	L bit is set	N, Z bits are cleared

Floating-Point Status Register Access

The following instructions load and store the FSR as a 32-bit integer. If the user specifies a register (gen1 in LFSR or gen2 in SFSR) it will be a general purpose register in the CPU.

Format	Opcode	Instruction		Description
9	001	LFSR	gen1	Load FSR with the content of gen1. (gen2 field = 0)
9	110	SFSR	gen2	Store FSR in gen2. (gen1 field = 0)

Note: All instructions support all of the NS32000 family data formats (for external operands) and all addressing modes are supported.

2.3 EXCEPTIONS

An exception for the FPC is a special floating-point condition with a default handling scheme. Seven types of exceptions are supported:

1) Underflows
2) Overflows
3) Divisions by zero
4) Illegal Instructions
5) Invalid Operations
6) Inexact results
7) Undefined Instructions

The FPC has improved exception handling. Except for Illegal and Undefined Instructions, the user can control all of the exception types. In addition, there are some specific exceptions that the user can control:

Overflows — Floating-Point overflow
— Integer conversion overflow

Invalid Operations — Reserved Operands

Most exceptions can be enabled to cause a CPU TRAP or to return a result without a TRAP on their occurrence. The TRAP is signaled by the FPC pulsing the $\overline{\text{FSSR}}$ line for one clock cycle. Illegal and Undefined instructions will always cause a TRAP if they are passed to the FPC.

When a TRAP occurs, the FPC sets the Q bit in the status word register. The CPU responds by reading the status word register while applying status (11110) on the status lines. If the TRAP is caused by an undefined opcode, the TS bit in the status word register will also be set by the FPC indicating a TRAP (UND). The TS bit is clear in all other cases.

When an exception occurs, the type field in the FSR register is also updated. A trapped instruction returns no result (even if the destination is an FPC register) and does not affect the CPU PSR. Instructions that end with a disabled exception will always return a result.

For each exception whose TRAP can be disabled, there is a flag bit to signal the occurrence of the exceptional condition whether or not the TRAP is enabled or disabled. These bits in the FSR can be used for polling the exception status while TRAPs are disabled.

2.0 Architectural Description (Continued)

TABLE 2-2. Exception Handling Summary

Exception Occurred	Enabled By	Q = 1; Trap Type	Disabled By	Q = 0; Default Result Returned	Flag Bits
Underflow	UEN = 1	001	UEN = 0	Zero	UF = 1
Floating-Point Overflow	OVE = 0	010	OVE = 1 IEN = 0	Infinity or Max NRM Number	OVF = 1
	OVE = 1 IEN = 1	110			OVF = 1 IF = 1
Integer Conversion Ov.	IOE = 0	010	IOE = 1 IEN = 0	Max or Min Integer	IOF = 1
	IOE = 1 IEN = 1	110			IOF = 1 IF = 1
Divide by Zero	DZE = 0	011	DZE = 1	Infinity	DZF = 1
Illegal Instruction	Always Enabled	T bit = 0 and 100	Cannot be Disabled	No Result	No Flags Affected
Invalid Operation	IVE = 0	101	IVE = 1	NaN	IVF = 1
Reserved Op. (NaN)	ROE = 0 IVE = 0	101	ROE = 0 IVE = 1	NaN	ROF = 1 IVF = 1
			ROE = 1 IVE = X	NaN	No Flags Affected
Reserved Op. (DNRM) (Note 1)	ROE = X IVE = 0	101	ROE = X IVE = 1	Undefined	ROF = 1 IVF = 1
Inexact Result	IEN = 1	110	IEN = 0	Correctly Rounded Result	IF = 1
Undefined Instruction	Always Enabled	T bit = 1 and 100	Cannot be Disabled	No Result	No Flags Affected

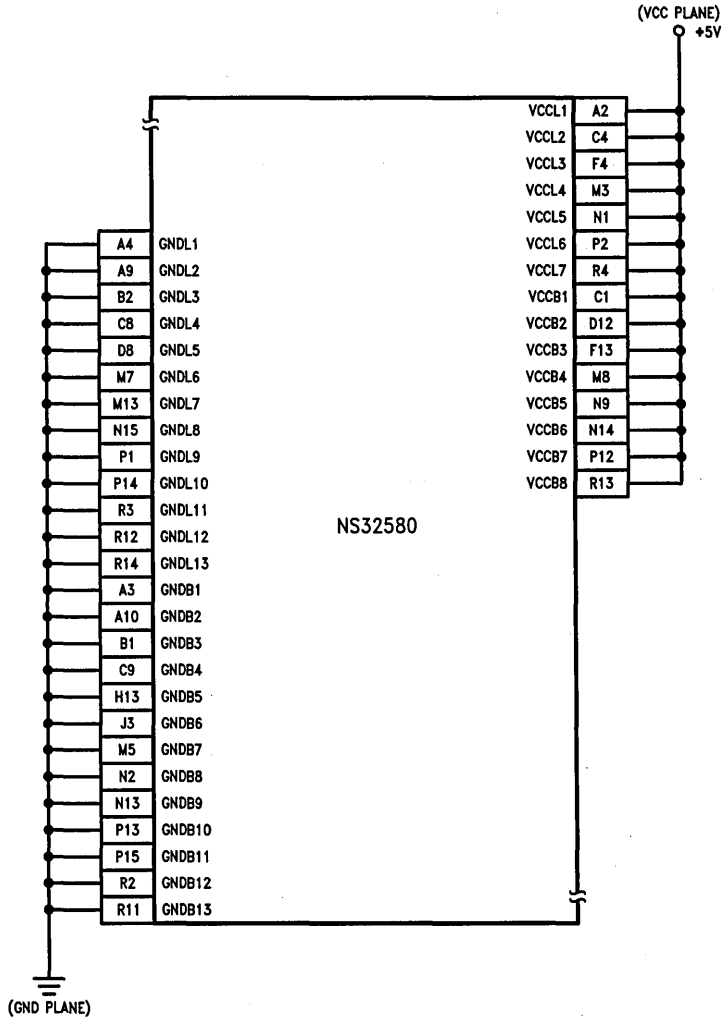
Exception Occurred	Enabled By	Q = 1; Trap Type	Disabled By	Status Word Register	Flag Bits
CMPf (NaN)	ROE = 0 IVE = 0	101	ROE = 0 IVE = 1	L = 1, N = Z = 0	ROF = 1 IVF = 1
			ROE = 1 IVE = X	L = 1, N = Z = 0	No Flags Affected
CMPf (DNRM)	ROE = X IVE = 0	101	ROE = X IVE = 1	N, L, Z Undefined	ROF = 1 IVF = 1

X = Don't Care

Note 1: For MULf 0 * DNRM
DIVf 0/DNRM
DIVf DNRM/Infinity
NS32580 returns a zero.
For DIVf Infinity/DNRM and MULf Infinity * DNRM, NS32580 returns an infinity.
For DIVf DNRM/0, TRAP (DVZ) will take place.

3

3.0 Functional Description



TL/EE/9421–8

FIGURE 3-1. Recommended Supply Connections

3.1 POWER AND GROUNDING

The NS32580 requires a single 5V power supply, applied on 15 pins. The logic voltage pins (VCCL1 to VCCL7) supply the power to the on-chip logic. The buffer voltage pins (VCCB1 to VCCB8) supply the power to the output drivers of the chip. All the voltage pins should be connected together by a power (V_{CC}) plane on the printed circuit board.

The NS32580 grounding connections are made on 26 pins. The logic ground pins (GNDL1 to GNDL13) are the ground pins for the on-chip logic. The buffer ground pins (GNDB1–GNDB13) are the ground pins for the output drivers of the chip. All the ground pins should be connected together by a ground plane on the printed circuit board.

Both power and ground connections are shown in *Figure 3-1.*

3.2 CLOCKING

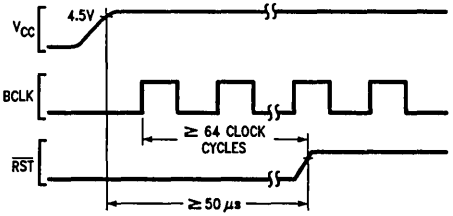
The NS32580 FPC requires a single-phase TTL clock input on its BCLK pin (pin C10) and an inverted TTL clock input on its $\overline{\text{BCLK}}$ pin (pin B8). When the FPC is connected to a NS32532 CPU these signals are provided directly from the CPU's BCLK and $\overline{\text{BCLK}}$ output signals.

3.3 RESETTING

The $\overline{\text{RST}}$ pin serves as a reset for on-chip logic. The FPC may be reset at any time by pulling the $\overline{\text{RST}}$ pin low for at least 64 clock cycles. Upon detecting a reset, the FPC terminates instruction processing, resets its internal logic, clears the FSR to all zeroes, and clears the FIFOs.

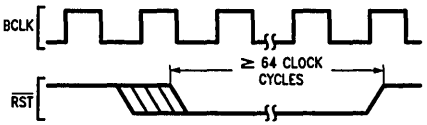
On application of power, $\overline{\text{RST}}$ must be held low for at least 50 μs after V_{CC} is stable. This ensures that all on-chip voltages are completely stable before operation. See *Figures 3-2* and *3-3.*

3.0 Functional Description (Continued)



TL/EE/9421-9

FIGURE 3-2. Power-On Reset Requirements



TL/EE/9421-10

FIGURE 3-3. General Reset Timing

3.4 BUS OPERATION

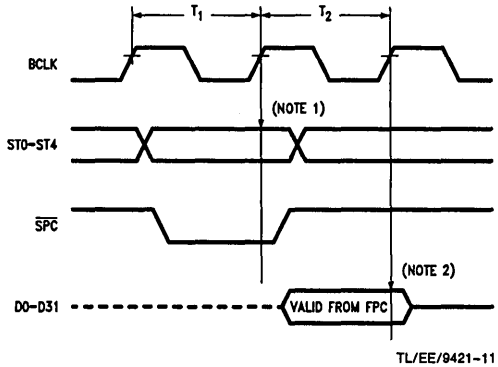
Instructions and operands are passed to the NS32580 FPC with slave processor bus cycles. Each bus cycle transfers one double-word (32 bits) to or from the FPC. During all bus cycles, the $\overline{SPC}$ line is driven by the CPU as an active low data strobe, and the FPC monitors pins ST0–ST4 to keep track of the sequence (protocol) established for the instruction being executed. This is necessary in a virtual memory environment, allowing the FPC to retry an aborted instruction.

A bus cycle is initiated by the CPU, which asserts the proper status on ST0–ST4 and pulses $\overline{SPC}$ low. The status lines are sampled by the FPC on the rising edge of BCLK in the T2 state. *Figures 3-4* and *3-5* illustrate these sequences.

3.4.1 Operand Transfers

The CPU fetches operands from memory, aligns them (if needed) and sends them to the slave (with status h'1D) as a 32-bit transfer. If the operand is double-precision the least significant half is transferred first (in 32000 mode). The FPC can not access the memory directly.

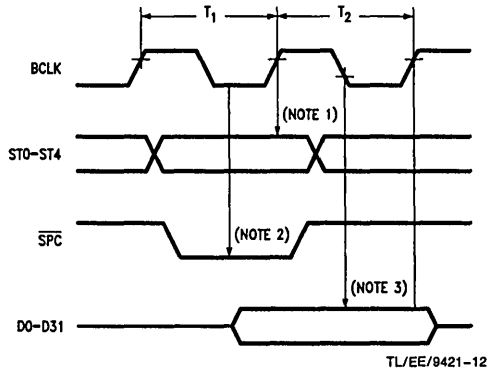
From the slave processor point of view there are four possible combinations of locations for operands: (For special cases see next paragraph.)



TL/EE/9421-11

Note 1: FPC samples CPU status here.
Note 2: CPU samples FPC data here.

FIGURE 3-4. Slave Processor Read Cycle from FPC



TL/EE/9421-12

Note 1: FPC samples CPU status here.
Note 2: FPC samples $\overline{SPC}$ here.
Note 3: FPC samples data here.

FIGURE 3-5. Slave Processor Write Cycle to FPC

Register to Register Instructions—Both operands reside in the register file inside the FPDP. No operand fetch or transfer from memory is needed.

Memory to Register—The source operand is in memory, therefore the CPU will transfer the operand (one 32-bit transfer for single-precision and two 32-bit transfers for double-precision). The result is going to the floating-point register in the register file located inside the FPDP.

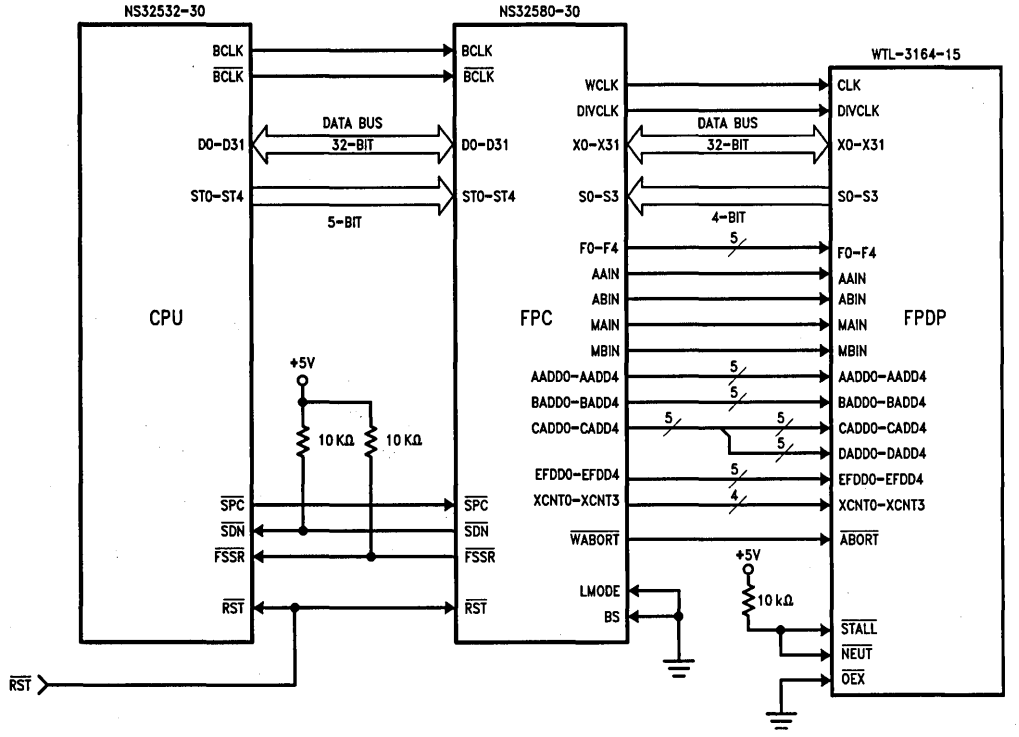
Register to Memory—The source operand resides inside the FPDP. If the instruction is monadic (one operand) the CPU will not transfer the operand to the FPC before the beginning of the instruction (all the information needed to start the operation resides inside the FPDP). For dyadic instructions, the CPU will fetch and transfer one operand from memory.

Memory to Memory—In monadic instructions the source operand is in memory and the CPU will transfer it to the FPC-FPDP. If the instruction is dyadic, two operands will be transferred from memory to the FPC-FPDP by the CPU (gen1 before gen2). The result in both cases is sent back to memory.

When the CPU transfers an operand from memory to the FPC-FPDP it is loaded into one of the registers that create the operand FIFO inside the FPDP. The FPC translates the incoming instruction (mem, reg or mem, mem) to a register-to-register instruction with the same register number. From the incoming instruction addressing mode it should know if the operands are coming from memory or already located in the register file.

The Data FIFO inside the FPC is 10 entries deep, single- or double-precision. If the destination of instruction is memory, the FPC will wait for completion of the instruction. Then, the result will be transferred to the FPC and $\overline{SDN}$ will be asserted. If the FPC receives a new ID and Opcode before the FPC receives all the operands for the last instruction or before the CPU reads the conplete result for the last instruction (can happen if page fault has been detected on a write and with only one instruction in the FPC's pipe), the FPC will abort the last instruction and will start the execution of the new instruction. The NS32532 CPU can "reset" the FPC at any time by asserting $\overline{SPC}$ with status 11110. In this case the FPC flushes the instructions currently being executed and the contents of the floating-point registers are undefined.

3.0 Functional Description (Continued)



TL/EE/9421-13

FIGURE 3-6. System Connection Diagram
***(For Two Cycle Latency in Little Endian Mode)**

*See Pin Description for other configurations.

3.5 INSTRUCTION PROTOCOLS

3.5.1 General Protocol Sequence

The FPC supports both the regular and the pipelined slave protocols provided by the NS32532. Detailed information on these protocols is provided in the NS32532 data sheet.

The basic operations performed by the CPU and the FPC are described below.

Floating-point instructions have a three-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word.

Upon receiving a floating-point instruction, the CPU initiates the sequence outlined in Table 3-1. While applying Status code 11111, the CPU transfers the ID Byte on bits D24–D31, the operation word on bits D8–D23 in a swapped order of bytes and a non-used byte XXXXXXXX (X = don't care) on bits D0–D7 *(Figure 3-7)*.

After transferring the instruction, the CPU sends to the FPC any source operands that are located in memory or the CPU General-Purpose registers.

The CPU action, at this point, depends on whether the regular or the pipelined slave protocol is selected. If the regular protocol is selected, the CPU waits for the FPC to complete the instruction. While the CPU is waiting, it can perform bus cycles to fetch instructions and read source operands for instructions that follow the floating-point instruction being executed. If there are no bus cycles to perform, the CPU is idle with a special Status indicating that it is waiting for a slave .

If the pipelined protocol is selected, the CPU may send a new floating-point instruction to the FPC before the previous instruction has been completed.

Although the CPU can advance as many as four floating-point instructions before receiving a completion pulse on $\overline{SDN}$ for the first instruction, full exception recovery is assured. This is accomplished through a FIFO mechanism which maintains the addresses of all the floating-point instructions sent to the FPC for execution.

Pipelined execution can occur only for instructions which do not require a result to be read from the FPC.

3.0 Functional Description (Continued)

In cases where a result is to be read back, the CPU will wait for instruction completion before issuing the next instruction. After the FPC asserts $\overline{SDN}$ or $\overline{FSSR}$, the CPU follows one of the two sequences described below.

If the FPC asserts $\overline{SDN}$, then the CPU checks whether the instruction stores any results to memory or the General-Purpose registers. The CPU reads any such results from the FPC by means of 1 or 2 bus cycles and updates the destination. If the instruction had been pipelined, the CPU simply updates the FIFO pointer to point to the next instruction in the FIFO.

If the FPC asserts $\overline{FSSR}$, then the NS32532 reads a 32-bit status word from the FPC. The CPU checks bit 0 in the FPC's status word to determine whether to update the PSR flags or to process an exception. *Figure 3-8* shows the format of the FPC's status word.

If the Q bit in the status word is 0, the CPU updates the N, Z and L flags in the PSR.

If the Q bit in the status word is set to 1, the CPU processes either a Trap (UND) if TS is 1 or a Trap (SLAVE) if TS is 0.

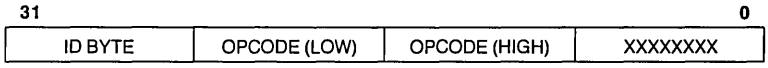


FIGURE 3-7. ID and Operation Word

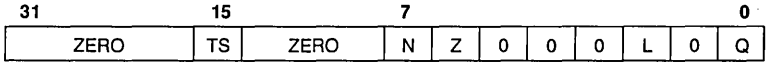


FIGURE 3-8. FPC Status Word

TABLE 3-1. Floating-Point Instruction Sequence

Step	Status	Action
1	ID (11111)	CPU sends ID and Operation Word
2	OP (11101)	CPU sends required operands (if any)
3	—	Slaves starts execution (CPU prefetches)
4	—	Slave signals completion by pulsing $\overline{SDN}$ or $\overline{FSSR}$.
5	ST (11110)	CPU Reads Status Word (If an exception occurred or if a CMPf instruction was executed)
6	OP (11101)	CPU Reads Result (if any)

3.0 Functional Description (Continued)

TABLE 3-2. Floating-Point Instruction Protocols

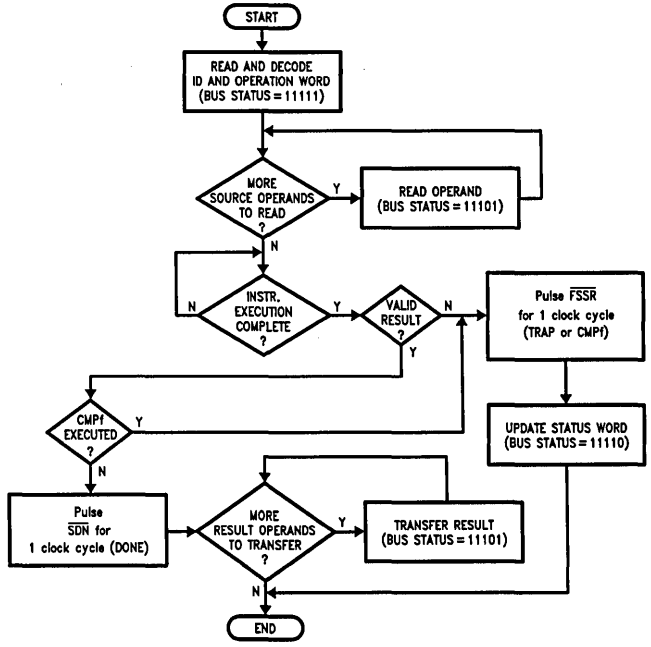
Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none
SQRTf	read.f	write.f	f	N/A	f to Op.2	none
MACf	read.f	read.f	f	f	f to L1/F1	none

D = Double Word

i = Integer size (B, W, D) specified in mnemonic.

f = Floating-Point type (F, L) specified in mnemonic.

N/A = Not Applicable to this instruction.



TL/EE/9421-14

FIGURE 3-9. Floating-Point Instruction Processing Flowchart

3.0 Functional Description (Continued)

With the pipelined protocol selected, the FPC can start execution of a new floating-point instruction every two clock cycles.

In the following example three floating-point instructions are executed in pipelined mode:

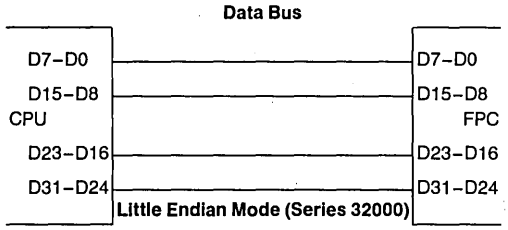
```
DIVF    O(RO), F1
ADDF    F2, F3
MULF    F4, F5
```

Step	Status	Action
1	ID (h'1F)	CPU sends ID and Opcode of DIVF instruction.
2	OP (h'1D)	CPU sends operand (R0).
3	—	Slave starts execution of DIVF instruction.
4	ID (h'1F)	CPU sends ID and Opcode of ADDF instruction.
5	—	Slave starts execution of ADDF instruction.
6	ID (h'1F)	CPU sends ID and Opcode of MULF instruction.
7	—	Slave starts execution of MULF instruction.
8	—	Slave pulses $\overline{SDN}$ or $\overline{FSSR}$ for the DIVF instruction. If an exception occurred, the rest of the instructions will be aborted.
9	ST (h'1E)	CPU Reads Status Word (if an exception occurred).
10	—	Slave pulses $\overline{SDN}$ or $\overline{FSSR}$ for the ADDF instruction. If an exception occurred, the rest of the instructions will be aborted.
11	ST (h'1E)	CPU Reads Status Word (if an exception occurred).
12	—	Slave pulses $\overline{SDN}$ or $\overline{FSSR}$ for the MULF instruction.
13	ST (h'1E)	CPU Reads Status Word (if an exception occurred).

Note: Instructions that can be pipelined include all instructions except CMPf in Format 11, as well as SQRTf and MAGf in Format 12. All other floating-point instructions will cause the pipe to break, that is, the instruction will be sent to the FPC but the pipe will stop until done or Trap.

3.5.2 Byte Sex

The FPC supports both little Endian and big Endian byte ordering, depending on the state of the BS pin. In little Endian mode (BS = "0"), the FPC receives the least significant half of a double-precision operand first and the most significant half afterward. In Big Endian mode (BS = "1"), the FPC receives the most significant half of a double-precision operand first and the least significant half afterward. The FPC will send the received operands to the correct destination registers inside the FPDP. In Big Endian mode, the user must swap the data bus between the CPU and FPC. See *Figure 3-10* for details. The BS pin is sampled by the FPC during Reset only.



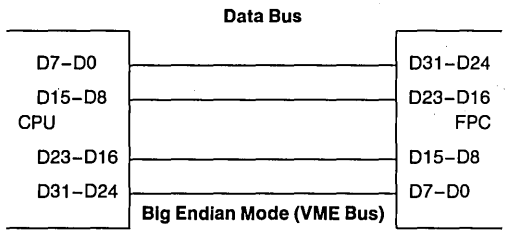


FIGURE 3-10. Byte Sex Connection Diagrams

3.5.3 Floating-Point Instruction Protocols

Table 3-2 gives the protocols followed for each floating-point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Section 2.2.3.

The Operand Class columns give the Access Classes for each general operand, defining how the addressing modes are interpreted by the CPU (see Series 32000 Instruction Set Reference Manual).

The Operand Issued columns show the sizes of the operands transferred to the Floating-Point Controller by the CPU. "D" indicates a 32-bit Double Word. "i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). "f" indicates that the instruction specifies a floating-point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the FPC Status Word *(Figure 3-9)*.

Any operand indicated as being of type "f" will not cause a transfer between CPU and FPC, if the Register addressing mode is specified, since the Floating-Point Registers are physically located in the FPC and are therefore available without CPU assistance.

3.6 FPDP INTERFACE

The FPC uses the Weitek WTL 3164 Floating-Point Data Path (FPDP) as the computational unit.

The FPDP is capable of supporting 32-bit and 64-bit IEEE floating-point operations. The FPDP consists of a Multiplier, ALU, Divide/Sqrt unit, 32 x 64-bit, Six-Port Register file, I/O port and control unit. There are six major internal 64-bit wide data buses used for data transfers between the different blocks inside the FPDP.

3.0 Functional Description (Continued)

Using six data buses allows the input of two double-precision operands to a selected unit and to output one double-precision result in one WCLK cycle, supporting pipelining of a new double-precision instruction every WCLK cycle. For a detailed description of the FPDP, refer to the appropriate data sheet from Weitek.

3.6.1 Controlling the FPDP

The FPC controls the FPDP on an instruction by instruction basis. The instruction control signals are delayed in the FPDP to match the FPDP pipeline stages.

This allows to specify all the controls for a Reg to Reg instruction in a single control word. There are two types of operations that can be executed concurrently on the FPDP. The first operation is a floating-point arithmetic operation done on operands from the register file. The second operation is a Load/Store operation using the X port of the FPDP.

3.6.2 Instruction Control

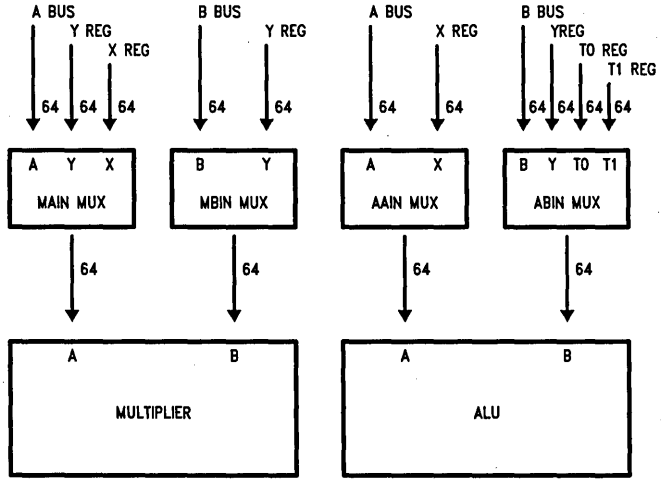
The FPC controls the FPDP using a 33-bit control word. The control word contains all the information needed for the execution of an instruction including the function to be executed, source operands and destination of the result. The controls are pipelined along with the instruction and affect the operation at the appropriate times. The control word is sampled with the rising edge of WCLK.

There are three functional fields in the control word:

1. The FUNC field defines the arithmetic operation to be executed.
2. The AAIN, ABIN, MAIN, MBIN, A ADD, B ADD, C ADD, D ADD bits specify the source and destination for arithmetic operations. Both C ADD and D ADD fields of the FPDP are connected to the D ADD field in the FPC control word.
3. The E/F ADD and XCNT fields control the Load and Store operations. E/F ADD selects the register to be loaded while XCNT selects the operation. XCNT encodings are provided in the following table.

5	1	1	1	1	5	5	5	5	5	4
FUNC	AAIN	ABIN	MAIN	MBIN	A ADD	B ADD	C ADD	D ADD	E/F ADD	XCNT
C41										C4

FIGURE 3-11. FPDP Control Word



TL/EE/9421–15

FIGURE 3-12. FPDP Multiplier and ALU Bus Control

3.0 Functional Description (Continued)

XCNT Field Encodings

The XCNT field specifies the I/O operation to be executed.

Code	Operation	Description
H'0	NOP	No Operation
H'1	EREG LS → XPAD	Transfer the Least Significant half of the register specified by EREG to the X-port (Store LS).
H'2	EREG MS → XPAD	Transfer the Most Significant half of the register specified by EREG to the X-port (Store MS).
H'3	EREG INT → XPAD	Transfer Integer operand in the register specified by EREG to the X-port (Store Int).
H'5	XPAD → XREG/FREG LS	Load the Least Significant half of the data in the X-port into the XREG LS and into the register specified by FREG.
H'6	XPAD → XREG/FREG MS	Load the Most Significant half of the data in the X-port into the XREG MS and into the register specified by FREG.
H'7	XPAD → XREG/FREG INT	Load the Integer operand in X-port into the XREG and into the register specified by FREG.

Data from the FPC is transferred to the FPDP through the XPAD Port. The data is loaded into the XREG and into a register in the register file specified by the E/F ADD.

Loading the data to both locations allows the immediate use of the data by the ALU and MULT, bypassing the register file. Loading the data to a register in the register file prevents data from being lost if the data from memory is needed a few cycles later.

The FPDP I/O Mode is determined by the control bits in the control register SR1 bits 4–0. The FPDP is used in Undelayed Single-Pump mode (code 00000).

3.6.3 "2 Cycle Mode" and "3 Cycle Mode"

The FPDP has two timing modes, "Two cycle latency" and "Three cycle latency". In "Two cycle latency" single- and double-precision operations have latency of two cycles. In "Three cycle latency", double-precision multiply has a three cycle latency, single-precision multiplies and single- or double-precision ALU operations have two cycle latency.

When using the "Three cycle latency" the Divide/Sqrt block uses the same clock as the other functional units in the FPDP. Although the "Three cycle latency" is not optimized for double-precision multiply it may be very useful if the WCLK frequency is higher than the FPDP speed rating.

The FPC has a pin to specify the desired mode. In "Three cycle latency" the LMODE pin should be connected to V_{CC} and in "Two cycle latency" it should be connected to GND. The LMODE line is sampled during reset. After reset, as part of the initialization cycle, the FPC updates the Multiply Latency bit in the FPDP control register SR0 bit-7 (0 = "Two cycle latency", 1 = "Three cycle latency").

In "Three cycle latency" the Divide/Sqrt block uses DCLK3 (same as WCLK), in "Two cycle latency" it uses DCLK2 (2 × WCLK). The FPC uses the latency pin to determine the length of some instructions (number of cycles before FPC can signal DONE or TRAP).

This feature allows the CPU to run at more than twice the maximum FPDP frequency.

The following table shows the system speed versus the FPDP speed and latency selection.

FPDP Speed Grade	WCLK "Two Cycle Latency"	WCLK "Three Cycle Latency"	Max System Speed
120 ns	120 ns	90 ns	45 ns
100 ns	100 ns	75 ns	38 ns
80 ns	80 ns	60 ns	30 ns
60 ns	60 ns	50 ns	25 ns

3.6.4 FPDP Mode Control Registers SR0, SR1

There are few options in the FPDP like Rounding, I/O, IEEE handling, Latency and others that can be controlled by writing into the control registers SR0 and SR1.

After reset and whenever the user changes the relevant fields in the FSR, the FPC updates the FPDP control registers.

Fast/IEEE Mode SR0 bit 0

"1" Set to Fast mode. An underflowed instruction with disabled underflow exception delivers zero to the destination register.

3.0 Functional Description (Continued)

Rounding

SR0 Bit-2	SR0 Bit-1	Rounding Mode
0	0	Round toward nearest value, if tie round toward even significant
0	1	Round toward zero
1	0	Round toward positive infinity
1	1	Round toward negative infinity

IntAbortOn SR0 Bit-3

"0" Internal abort off.

SR0 Bit-4

"0"

IlokOn SR0 Bit-5

"0" Disables Interlocks.

FpexSticky SR0 Bit-6

"0" FPEX is "Pulsed". In this mode, FPEX is asserted for one clock cycle.

Multiply Latency SR0 Bit-7

The FPDP has two multiply latency modes: Two cycle latency mode and Three cycle latency mode. See Section 3.6.3.

SR0 Bit-7	Latency Mode
0	Two Cycle Latency Mode
1	Three Cycle Latency Mode

I/O Mode SR1 Bits 4–0

0 0 0 0 0 Single-Pump Undelayed

The FPDP is used by the FPC in the undelayed single-pump mode for load and store operations.

FpexDelay SR1 Bit-5

"1" Delayed FPEX-Mode.

BypassOn SR1 Bit-6

"1" Enables bypassing of operands between instructions.

SR1 Bit-7

"0"

3.6.5 IEEE Enable Register SR2

The SR2 register has enable bits for each of the exception conditions. The FPC updates the enable bits after Reset and whenever the user changes the relevant bits in the FSR. (See LFSR Instruction.)

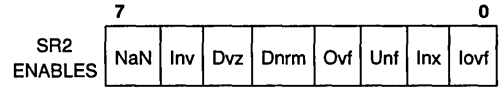


FIGURE 3-13. IEEE Enable Register (FPDP)

FPC updates the Inv, Dvz, Ovf and Iovf, Unf, Inx enable bits to reflect those enable bits in the FSR.

The NaN bit is affected by the ROE bit in the FSR. If the ROE is cleared then NaN should be enabled (signal exception upon detection of NaN). If ROE is set NaN will be disabled.

The Dnrm bit is always enabled and detection of Dnrm as operand for operation will cause a source exception.

Whenever the user changes the enable bit in the FSR, the same bit will be updated in the exception enable register in the FPDP.

Registers SR3–SR11 are not used by the FPC.

3.6.5.1 FPDP Status Lines (S0–3)

The status of operation in the FPDP can be obtained by using the FPDP status lines. The status is not "sticky", therefore, the FPC has to sample the status lines in the correct timing. If ALU and MULT instructions end in the same cycle, the ALU status is valid at the end of the cycle and the MULT status is valid at the beginning of the following cycle.

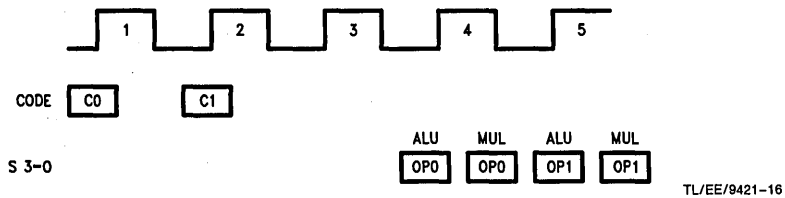


FIGURE 3-14. FPDP Status Timing

3.0 Functional Description (Continued)

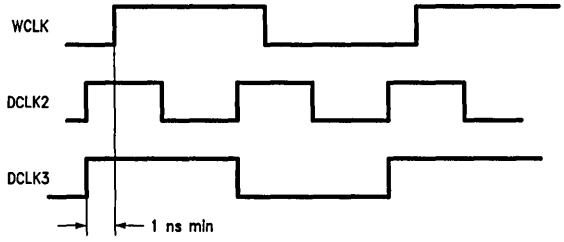
3.6.6 FPDP Clocking Requirements

The FPC uses BLCK and $\overline{\text{BLCK}}$ from the CPU to generate the clock signals required by the FPDP.

The FPDP requires two clock signals: DIVCLK and WCLK. DIVCLK is used by the DIVIDE/SQRT unit, while WCLK is used by all other functional units. The frequency of DIVCLK is dependent on the latency mode selected. It is either same or twice the frequency of WCLK for the "Three Cycle Latency" or "Two Cycle Latency" Modes respectively.

The WCLK frequency is always half the frequency of BCLK.

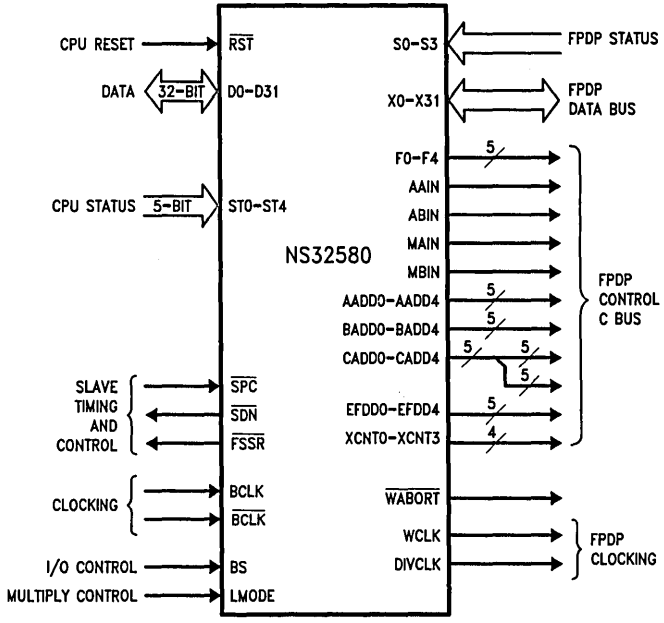
The FPC determines the DIVCLK frequency by using the LMODE pin.



TL/EE/9421-17

FIGURE 3-15. Divide/Sqrt Clock DCLK2/DCLK3

4.0 Device Specifications



TL/EE/9421-18

FIGURE 4-1. NS32580 Interface Signals

4.0 Device Specifications (Continued)

4.1 NS32580 PIN DESCRIPTIONS

Descriptions of the NS32580 pins are given in the following sections. *Figure 4-1* shows the NS32580 interface signals grouped according to related functions.

4.1.1 Supplies

VCCL1-7 **Logic Power**—+5V positive supplies for on-chip logic.

VCCB1-8 **Buffers Power**—+5V positive supplies for on-chip buffers.

GNDL1-13 **Logic Ground**—Ground references for on-chip logic.

GNDB1-13 **Buffers Ground**—Ground references for on-chip buffers.

4.1.2 Input Signals

BCLK **Bus Clock**—Input clock from NS32532.

$\overline{\text{BCLK}}$ **Bus Clock Inverse**—Inverted input clock from NS32532.

BS **Byte Sex**—Specifies the I/O byte ordering of the FPC. If connected to GND the FPC is in Little Endian mode. If connected to V_{CC} the FPC is in Big Endian mode. The BS line must be valid during and after Reset. See Section 3.5.2.

LMODE **Latency Mode**—Specifies the latency mode of the FPC-FPDP. If connected to GND the FPC-FPDP is in the "Two cycle latency", if connected to V_{CC} the FPC-FPDP is in the "Three cycle latency". LMODE line must be valid during and after Reset.

$\overline{\text{RST}}$ **Reset**—Active low. Resets the last operation, clears the FIFOs and the FSR register to its default state.

S0-3 **FPDP Status**—Indicates any exceptions or conditions that resulted from operations performed by the WTL 3164 floating-point data path.

$\overline{\text{SPC}}$ **Slave Processor Control**—Active low. Data strobe for slave transfers between the CPU and the FPC.

ST0-4 **CPU Status**—Bus cycle status code from CPU. ST0 is the least significant and rightmost bit.

1 1 1 0 0 —Reserved

1 1 1 0 1 —Transferring Operand

1 1 1 1 0 —Reading Status Word

1 1 1 1 1 —Broadcasting Slave ID

AADD0-4 **A Read Port Register Address**—Chooses the inputs to the A bus of the FPDP.

AAIN **ALU A Input Select**—Controls the A input multiplexers of the FPDP ALU.

ABIN **ALU B Input Select**—Controls the B input multiplexers of the FPDP ALU.

BADD0-4 **B Read Port Register Address**—Chooses the inputs to the B bus of the FPDP.

CADD0-4 **C Write Port Register Address**—C/D Bus Control. Chooses the destinations of C and D buses. These signals should be connected to both the (CADD0-4) and the (DADD0-4) lines of the FPDP.

4.1.3 Output Signals

DIVCLK **Divide/Square Root Clock**—Clock signal for the Divide/Sqrt unit in the FPDP.

EFDD0-4 **E and/or F Port Register Address**—Chooses the source and destination for the Load/Store operations of the FPDP.

F0-4 **Function Code**—Specifies the operation to be performed by the FPDP.

$\overline{\text{FSSR}}$ **Forced Slave Status Read**—Active low. When active, indicates that the FPC status word should be read by the CPU. It is floating before and after being active.

MAIN **Multiplier A Input Select**—Controls the A input multiplexers of the FPDP multiplier.

MBIN **Multiplier B Input Select**—Controls the B input multiplexers of the FPDP multiplier.

$\overline{\text{SDN}}$ **Slave Done**—Active low. When active, indicates successful completion by the FPC-FPDP of a floating-point instruction. It is floating before and after being active.

$\overline{\text{WABORT}}$ **FPDP Abort**—Aborts the current and previous instructions in the FPDP.

WCLK **FPDP Clock**—Clock signal for the FPDP. It is BCLK divided by two. i.e., if BCLK is 30 MHz, WCLK will be 15 MHz.

XCNT0-3 **X Port Control**—They are the Load/Store controls for the FPDP.

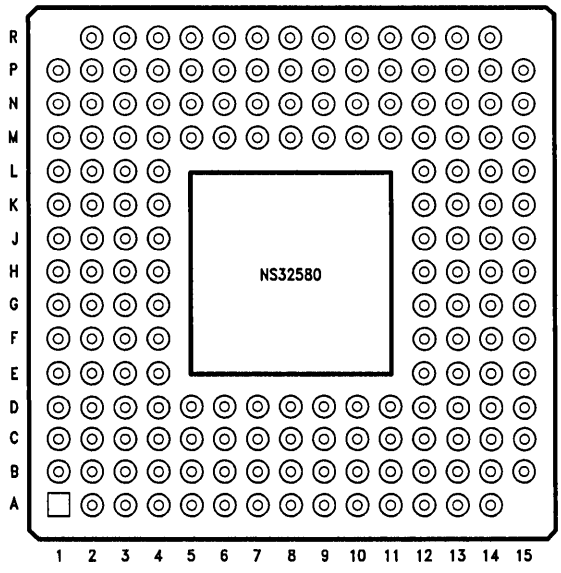
4.1.4 Input/Output Signals

D0-31 **CPU Data Bus**—Data bus between FPC and the CPU.

X0-31 **FPDP Data Bus**—Data bus between FPC and the FPDP X port.

4.0 Device Specifications (Continued)

Connection Diagram



TL/EE/9421–31

Bottom View

FIGURE 4-2. 172-Pin PGA Package

Order Number NS32580-20, NS32580-25 or NS32580-30
See NS Package Number U172B

4.0 Device Specifications (Continued)

NS32580 Pinout Descriptions

Desc	Pin	Desc	Pin	Desc	Pin	Desc	Pin
VCCL1	A2	X3	D1	D28	H12	VCCL5	N1
GNDB1	A3	X4	D2	GNDB5	H13	GNDB8	N2
GNDL1	A4	NC	D3	F0	H14	Reserved	N3
XCNT0	A5	D2	D4	F1	H15	D24	N4
XCNT3	A6	D17	D5	X13	J1	D25	N5
EFADD1	A7	D16	D6	X15	J2	D9	N6
EFADD2	A8	NC	D7	GNDB6	J3	D10	N7
GNDL2	A9	GNDL5	D8	D21	J4	NC	N8
GNDB2	A10	NC	D9	D12	J12	VCCB5	N9
CADD0	A11	NC	D10	D27	J13	ST2	N10
CADD2	A12	NC	D11	F2	J14	ST4	N11
CADD3	A13	VCCB2	D12	F3	J15	$\overline{\text{FSSR}}$	N12
BADD0	A14	D15	D13	X14	K1	GNDB9	N13
GNDB3	B1	AADD1	D14	X17	K2	VCCB6	N14
GNDL3	B2	AADD2	D15	D6	K3	GNDL8	N15
X0	B3	X5	E1	D22	K4	GNDL9	P1
XCNT1	B4	X7	E2	D11	K12	VCCL6	P2
XCNT2	B5	D18	E3	NC	K13	X21	P3
EFADD0	B6	D3	E4	SO	K14	X23	P4
EFADD3	B7	D31	E12	F4	K15	X25	P5
$\overline{\text{BCLK}}$	B8	D14	E13	X16	L1	X26	P6
WCLK	B9	AADD3	E14	X18	L2	X28	P7
DIVCLK	B10	AADD4	E15	D7	L3	X31	P8
EFADD4	B11	X6	F1	D23	L4	X30	P9
CADD1	B12	X9	F2	$\overline{\text{SPC}}$	L12	BS	P10
CADD4	B13	D19	F3	$\overline{\text{SDN}}$	L13	ST3	P11
BADD1	B14	VCCL3	F4	S2	L14	VCCB7	P12
BADD2	B15	D30	F12	S1	L15	GNDB10	P13
VCCB1	C1	VCCB3	F13	X19	M1	GNDL10	P14
X2	C2	MAIN	F14	Reserved	M2	GNDB11	P15
X1	C3	MBIN	F15	VCCL4	M3	GNDB12	R2
VCCL2	C4	X8	G1	D8	M4	GNDL11	R3
D1	C5	X10	G2	GNDB7	M5	VCCL7	R4
D0	C6	D4	G3	D26	M6	X20	R5
NC	C7	D20	G4	GNDL6	M7	X22	R6
GNDL4	C8	D13	G12	VCCB4	M8	X24	R7
GNDB4	C9	D29	G13	NC	M9	X27	R8
BCLK	C10	AAIN	G14	ST0	M10	X29	R9
$\overline{\text{RST}}$	C11	ABIN	G15	ST1	M11	LMODE	R10
NC	C12	X11	H1	NC	M12	GNDB13	R11
BADD3	C13	X12	H2	GNDL7	M13	GNDL12	R12
AADD0	C14	NC	H3	$\overline{\text{WABORT}}$	M14	VCCB8	R13
BADD4	C15	D5	H4	S3	M15	GNDL13	R14

Note: NC = No Connection

4.0 Device Specifications (Continued)

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	1.5W

ESD Rating is to be determined.

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±10%, GND = 0V

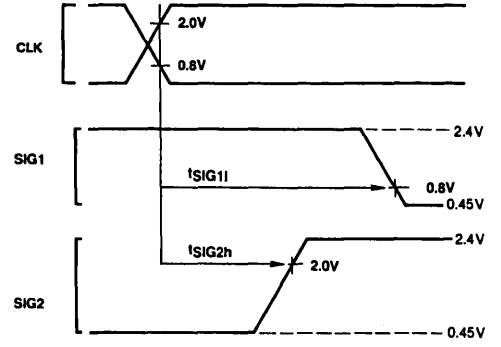
Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		V_{CC} + 0.5	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low Level Output Voltage	I_{OL} = 2 mA			0.4	V
I_I	Input Load Current	0 ≤ V_{IN} ≤ V_{CC}				
I_L	Leakage Current (Output and I/O Pins in TRI-STATE®/Input Mode)	0.4 ≤ V_{OUT} ≤ 2.4V				
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C			300	mA
C_{IN}	Input Capacitance					pF
C_{OUT}	Output Capacitance					pF

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the Timing Specifications given in this section refer to 0.8V and 2.0V on all the input and output signals as illustrated in *Figures 4.3* and *4.4*, unless specifically stated otherwise.

Note: These voltage levels shown are for 32532–32580 interface only. Levels for 32580–WTL3164 interface are specified in their appropriate timing diagrams.

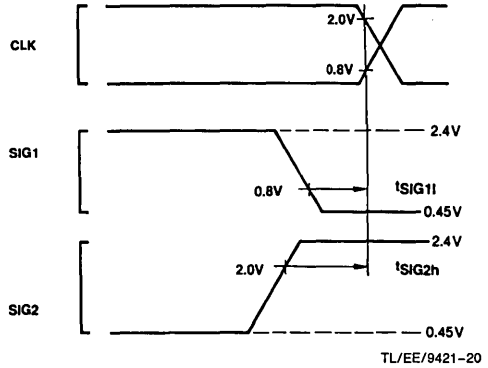


TL/EE/9421–19

FIGURE 4-3. Timing Specification Standard (Signal Valid after Clock Edge)

ABBREVIATIONS

L.E. — Leading Edge
T.E. — Trailing Edge
R.E. — Rising Edge
F.E. — Falling Edge



TL/EE/9421–20

FIGURE 4-4. Timing Specification Standard (Signal Valid before Clock Edge)

4.0 Device Specifications (Continued)

4.4.2 Timing Tables Maximum times assume temperature range 0°C to 70°C

4.4.2.1 Output Signal Propagation Delays Maximum times assume capacitive loading of 100 pF

Symbol	*Figure*	Description	Reference/ Conditions	NS32580-20		NS32580-25		NS32580-30		Units
				Min	Max	Min	Max	Min	Max	
t_{Dv}	*4-8*	CPU Data Valid	After R.E., BCLK T2		35		27		23	ns
t_{Doh}	*4-8*	CPU Data Hold	After R.E., BCLK Next T1/Ti	2		2		2		ns
t_{Dnf}	*4-8*	CPU Data Not Forcing	After R.E., BCLK Next T1/Ti		28		23		19	ns
t_{SDa}	*4-10*	$\overline{SDN}$ Signal Active	After R.E., BCLK		35		28		22	ns
t_{SDia}	*4-10*	$\overline{SDN}$ Signal Inactive	After R.E., Next BCLK	2		2		2		ns
t_{SDnf}	*4-10*	$\overline{SDN}$ Signal Not Forcing	After R.E., BCLK		25		20		17	ns
t_{FSSRa}	*4-11*	$\overline{FSSR}$ Signal Active	After R.E., BCLK		35		28		22	ns
t_{FSSRia}	*4-11*	$\overline{FSSR}$ Signal Inactive	After R.E., Next BCLK	2		2		2		ns
t_{FSSRnf}	*4-11*	$\overline{FSSR}$ Signal Not Forcing	After R.E., BCLK		25		20		17	ns
t_{Cv}	*4-14*	C Bus	After R.E., WCLK		83		63		50	ns
t_{ABRTv}	*4-14*	$\overline{WABORT}$ Valid	After R.E., WCLK		83		63		50	ns
t_{Ch}	*4-14*	C BUS	After R.E., WCLK	3		3		3		ns
t_{ABRTh}	*4-14*	$\overline{WABORT}$ Hold Time	After R.E., WCLK	5		5		5		ns
t_{XLv}	*4-14*	FPDP Data Valid	After R.E., WCLK		83		63		50	ns
t_{XLh}	*4-14*	FPDP Data Hold Time	After R.E., WCLK	3		3		3		ns
t_{D2p}	*4-13*	DCLK2 Period	From 2.0V R.E., to 2.0V R.E.	50		40		33.3		ns
t_{D2h}	*4-13*	DCLK2 High Time	From 2.0V R.E., to 0.8V F.E.	22		17		14.5		ns
t_{D2l}	*4-13*	DCLK2 Low Time	From 0.8V F.E. to 2.0V R.E.	22		17		14.5		ns
t_{D3p}	*4-13*	DCLK3 Period	From 2.0V R.E., to 2.0V R.E.	100		80		66.6		ns
t_{D3h}	*4-13*	DCLK3 High Time	From 2.0V R.E., to 0.8V F.E.	45		36		30		ns
t_{D3l}	*4-13*	DCLK3 Low Time	From 0.8V F.E., to 2.0V R.E.	45		36		30		ns
t_{WCLKp}	*4-13*	WCLK Period	From 2.0V R.E., to 2.0V R.E.	100		80		66.6		ns
t_{WCLKh}	*4-13*	WCLK High Time	From 2.0V R.E., to 0.8V F.E.	45		36		30		ns
t_{WCLKl}	*4-13*	WCLK Low Time	From 0.8V F.E. to 2.0V R.E.	45		36		30		ns
t_{DWd}	*4-13*	DCLK2/DCLK3 to WCLK Delay	From 2.0V R.E., to 2.0V R.E.	1	8	1	8	1	8	ns
t_{Wr}	*4-13*	FPDP Clock Rise Time	From 0.8V R.E., to 2.4V R.E.		4		4		4	ns
t_{Wf}	*4-13*	FPDP Clock Fall Time	From 2.4V F.E. to 0.8V F.E.		4		4		4	ns

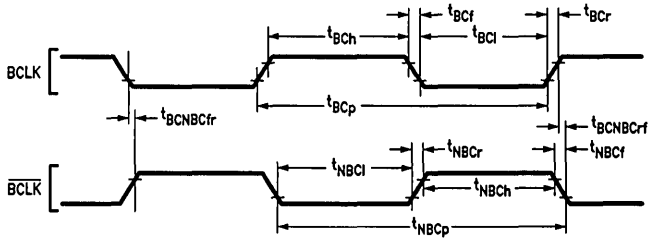
4.4.2.2 Input Signal Requirements NS32580-20, NS32580-25, NS32580-30

Symbol	*Figure*	Description	Reference/ Conditions	NS32580-20		NS32580-25		NS32580-30		Units
				Min	Max	Min	Max	Min	Max	
t_{BCp}	*4-5*	BCLK Period	R.E., BCLK to Next R.E., BLCK	50	100	40	100	33.3	100	ns
t_{BCh}	*4-5*	BCLK High Time	At 2.0V on BCLK (Both Edges)	0.5 t_{BCp} −5		0.5 t_{BCp} −4		0.5 t_{BCp} −3		
t_{BCl}	*4-5*	BCLK Low Time	At 0.8V on BCLK (Both Edges)	0.5 t_{BCp} −5		0.5 t_{BCp} −4		0.5 t_{BCp} −3		
t_{BCr}	*4-5*	BCLK Rise Time	0.8V to 2.0V on R.E., BCLK		5		4		3	ns
t_{BCf}	*4-5*	BCLK Fall Time	2.0V to 0.8V on F.E., BCLK		5		4		3	ns
t_{NBCp}	*4-5*	$\overline{BCLK}$ Period	R.E., $\overline{BCLK}$ to Next R.E., $\overline{BCLK}$	50	100	40	100	33.3	100	ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements NS32580-20, NS32580-25, NS32580-30 (Continued)

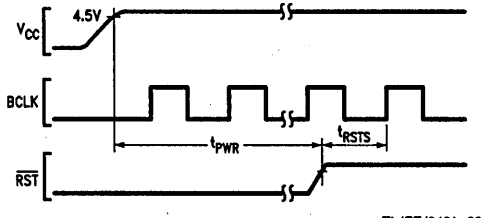
Symbol	*Figure*	Description	Reference/ Conditions	NS32580-20		NS32580-25		NS32580-30		Units
				Min	Max	Min	Max	Min	Max	
t_{NBCh}	*4-5*	$\overline{BCLK}$ High Time	At 2.0V on $\overline{BCLK}$ (Both Edges)	0.5 t_{NBCp} −5		0.5 t_{NBCp} −4		0.5 t_{NBCp} −3	120	ns
t_{NBCl}	*4-5*	$\overline{BCLK}$ Low Time	At 0.8V on $\overline{BCLK}$ (Both Edges)	0.5 t_{NBCp} −5		0.5 t_{NBCp} −4		0.5 t_{NBCp} −3	120	ns
t_{NBCr}	*4-5*	$\overline{BCLK}$ Rise Time	0.8V to 2.0V on R.E., $\overline{BCLK}$		5		4		3	ns
t_{NBCf}	*4-5*	$\overline{BCLK}$ Fall Time	2.0V to 0.8V on F.E., $\overline{BCLK}$		5		4		3	ns
$t_{BCNBCrf}$	*4-5*	Bus Clock Skew	2.0V on R.E., BCLK to 0.8V on F.E., $\overline{BCLK}$	−2	+2	−2	+2	−1	+1	ns
$t_{BCNBCfr}$	*4-5*	Bus Clock Skew	0.8V on F.E., BCLK to 2.0V on R.E., $\overline{BCLK}$	−2	+2	−2	+2	−1	+1	ns
t_{PWR}	*4-6*	Power Stable to R.E. of $\overline{RST}$	After V_{CC} Reaches 4.5V	50		40		30		μs
t_{RSTs}	*4-6, 4-7*	$\overline{RST}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{RSTw}	*4-7*	$\overline{RST}$ Pulse Width	At 0.8V (Both Edges)	64		64		64		t_{BCp}
t_{STs}	*4-8, 4-9*	CPU Status Setup Time	Before R.E., BCLK T2	36		30		24		ns
t_{STh}	*4-8, 4-9*	CPU Status Hold Time	After R.E., BCLK T2	15		12		10		ns
t_{SPCs}	*4-8, 4-9*	$\overline{SPC}$ Setup Time	Before R.E., BCLK T2	30		23		20		ns
t_{SPCh}	*4-8, 4-9*	$\overline{SPC}$ Hold Time	After R.E., BCLK T2	0	t_{BCp} +19	0	t_{BCp} +15	0	t_{BCp} +12	ns
t_{Ds}	*4-9*	Data Setup Time	Before R.E., BCLK T2	7		5		3		ns
t_{Dh}	*4-9*	Data Hold Time	After R.E., BCLK Next T1 or Ti	−4		−4		−4		ns
t_{SAs}	*4-12*	FPDP ALU Status Setup Time	Before R.E., WCLK	9		9		8		ns
t_{SAh}	*4-12*	FPDP ALU Status Hold Time	After R.E., WCLK	5		5		5		ns
t_{SMs}	*4-12*	FPDP Multiplier Status Setup Time	Before F.E., WCLK	9		9		8		ns
t_{SMh}	*4-12*	FPDP Multiplier Status Hold Time	After F.E., WCLK	5		5		5		ns
t_{XSs}	*4-14*	FPDP Data Setup Time	Before R.E., WCLK	9		9		9		ns
t_{XSh}	*4-14*	FPDP Data Hold Time	After R.E., WCLK	5		5		5		ns



TL/EE/9421-21

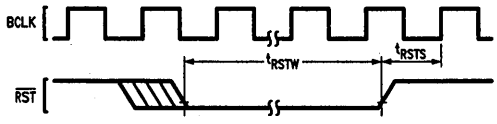
FIGURE 4-5. Clock Waveforms

4.0 Device Specifications (Continued)



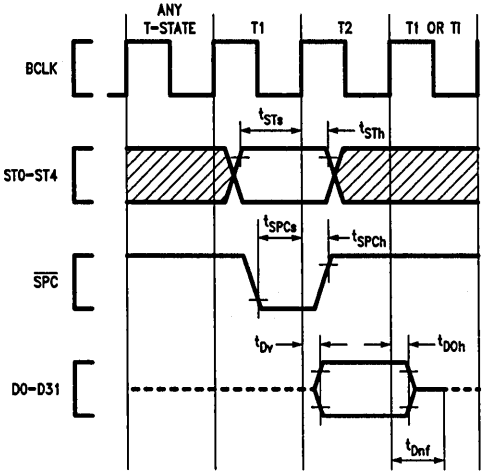
TL/EE/9421-22

FIGURE 4-6. Power-On Reset



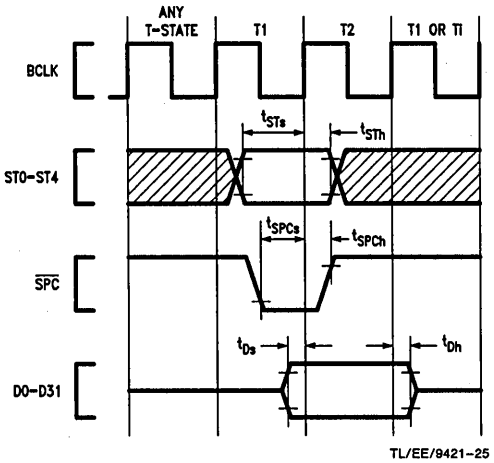
TL/EE/9421-23

FIGURE 4-7. Non-Power-On Reset



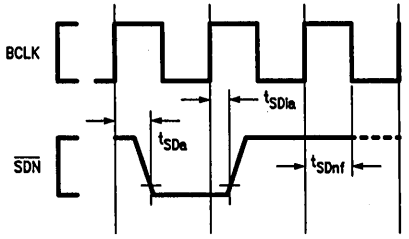
TL/EE/9421-24

FIGURE 4-8. Read Cycle from FPC



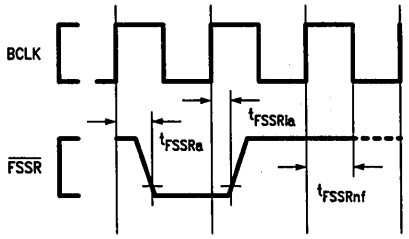
TL/EE/9421-25

FIGURE 4-9. Write Cycle to FPC



TL/EE/9421-26

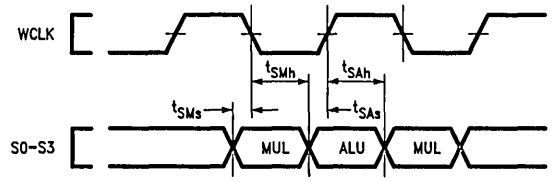
FIGURE 4-10. Slave Processor Done Timing



TL/EE/9421-27

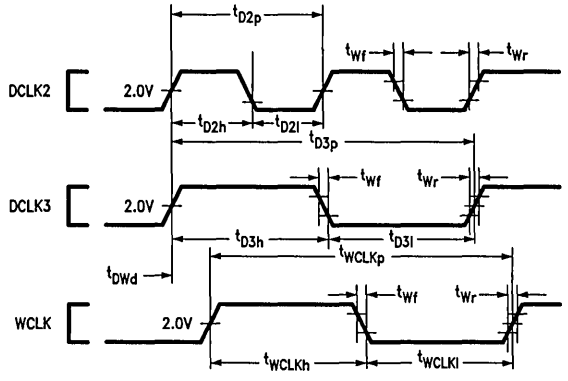
FIGURE 4-11. $\overline{FSSR}$ Signal Timing

4.0 Device Specifications (Continued)



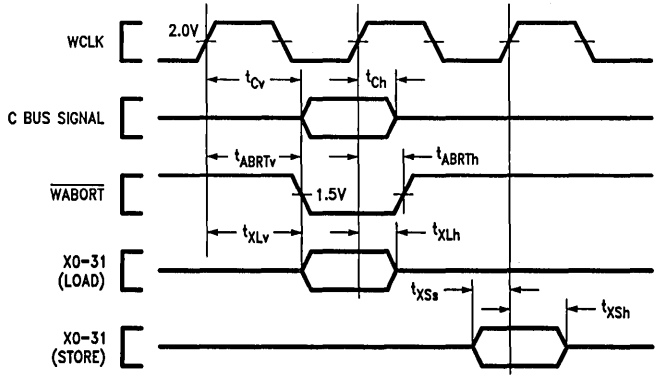
TL/EE/9421–28

FIGURE 4-12. FPDP Status Signal Timing



TL/EE/9421–29

FIGURE 4-13. FPDP Clock Signal Timing



TL/EE/9421–30

FIGURE 4-14. FPDP Output Signal Timing

3

Appendix A

COMPATIBILITY OF FPC-FPDP WITH NS32081/NS32381

NS32081	NS32381	NS32580
INSTRUCTIONS		
	NS32081 +	NS32081 +
	DOTf POLYf SCALBf LOGBf	MACf SQRTf
REGISTERS		
8 x 32 Bit	8 x 64 Bit	8 x 64 Bit
RESERVED OPERANDS		
DNRM	DNRM	DNRM*
NaN	NaN	NaN can be enabled or Disable.*
Infinity	Infinity	Infinity is NOT a reserved operand.*

NS32081	NS32381	NS32580
FSR		
	NS32081 FSR +	NS32081 FSR +
	RMB	RMB ROE IVE DZE OVE IOE ROF IVF DZF OVF IOF

*See compatibility table for special cases.

Compatibility Table

Special Case	NS32081/NS32381	NS32580
ROUNDfi (infinity)	TRAP (INV)	TRAP (OVF), IOF = 1
TRUNCfi (infinity)	TRAP (INV)	TRAP (OVF), IOF = 1
FLOORfi (infinity)	TRAP (INV)	TRAP (OVF), IOF = 1
DIVf 0, infinity	TRAP (DVZ)	Result = infinity
SQRTf (−DNRM)	TRAP (INV)	TRAP (INV), ROF = 0, IVF = 1
DIVf 0, DNRM	TRAP (INV)	TRAP (DVZ)
MULf (0, DNRM) or (DNRM, 0)	TRAP (INV)	Result = 0
DIVf DNRM, 0	TRAP (INV)	Result = 0
DIVf infinity, DNRM	TRAP (INV)	Result = 0
DIVf DNRM, infinity	TRAP (INV)	Result = infinity
MULf (infinity, DNRM) or (DNRM, infinity)	TRAP (INV)	Result = infinity
FSR.ROE = 1 and		
NEGf (NaN)	N/A	Result = −NaN
ABSf (NaN)	N/A	Result = |NaN|
ADDF	N/A	Result = NaN
SUBf (Nan, DNRM or DNRM, NaN)	N/A	Result = NaN
MULf (Nan, DNRM or DNRM, NaN)	N/A	Result = NaN
DIVf (Nan, DNRM or DNRM, NaN)	N/A	Result = NaN
MACf (Nan, DNRM or DNRM, NaN)	N/A	Result = NaN

Appendix B

PERFORMANCE ANALYSIS

The execution time is calculated from $\overline{SPC}$ (T1, T2 included) to $\overline{SDN}$ (including the $\overline{SDN}$ pulse)

Instruction	Latency reg, reg 2 cycles mode	Latency reg, reg 3 cycles mode	Throughput reg, reg 2 cycles mode	Throughput reg, reg 3 cycles mode	Pipe Break
ADDf/I	13	13	2	2	No
SUBf/I	13	13	2	2	No
MULf	13	13	2	2	No
MULI	13	15	2	4	No
DIVf	29	43	29	43	No
DIV1	43	71	43	71	No
MOVf/I	13	13	2	2	No
ABSf/I	13	13	2	2	No
NEGf/I	13	13	2	2	No
CMPf/I	13 + CPU	13 + CPU	—	—	Yes
FLOORfi	13 + CPU	13 + CPU	—	—	Yes
TRUNCfi	13 + CPU	13 + CPU	—	—	Yes
ROUNDfi	13 + CPU	13 + CPU	—	—	Yes
MOVFL	13 + CPU	13 + CPU	—	—	Yes
MOVLF	13 + CPU	13 + CPU	—	—	Yes
MOVif	17 + CPU	17 + CPU	—	—	Yes
MOVil	13 + CPU	13 + CPU	—	—	Yes
LFSR	13	13	—	—	Yes
SFSR	13 + CPU	13 + CPU	—	—	Yes
MACf	17	17	6	6	No
MACI	17	19	6	8	No
SQRTf	41	65	41	65	No
SQRTI	69	123	69	123	No

Appendix B (Continued)

Add the following CPU cycles to the base (reg, reg) number of cycles for the different cases:

Instruction	Latency 2 Cycles Mode	Latency 3 Cycles Mode	Throughput 2 Cycles Mode	Throughput 3 Cycles Mode	Pipe Break
MONADIC FLOAT (One Operand)					
mem, reg	0	0	2	2	see reg, reg
reg, mem	0 + CPU	0 + CPU	—	—	Yes
mem, mem	0 + CPU	0 + CPU	—	—	Yes
DYADIC FLOAT (Two Operands)					
mem, reg	0	0	2	2	see reg, reg
reg, mem	0 + CPU	0 + CPU	—	—	Yes
mem, mem	2 + CPU	2 + CPU	—	—	Yes
MONADIC LONG (One Operand)					
mem, reg	2	2	4	4	see reg, reg
reg, mem	2 + CPU	2 + CPU	—	—	Yes
mem, mem	2 + CPU	2 + CPU	—	—	Yes
DYADIC LONG (Two Operands)					
mem, reg	2	2	4	4	see reg, reg
reg, mem	6 + CPU	6 + CPU	—	—	Yes
mem, mem	6 + CPU	6 + CPU	—	—	Yes

Note: CPU stands for the time it takes the CPU to take the result from the FPC and resume operation.

Section 4
Peripherals

Section 4 Contents

National Semiconductor

PRELIMINARY

NS32C201-10/NS32C201-15 Timing Control Units

General Description

The NS32C201 Timing Control Unit (TCU) is a 24-pin device fabricated using National's microCMOS technology. It provides a two-phase clock, system control logic and cycle extension logic for the Series 32000® microprocessor family. The TCU input clock can be provided by either a crystal or an external clock signal whose frequency is twice the system clock frequency.

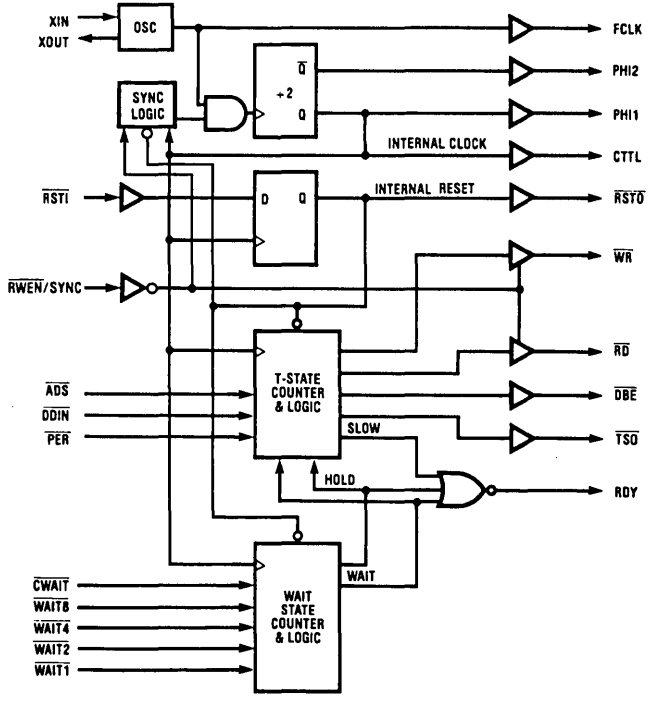
In addition to the two-phase clock for the CPU and MMU (PHI1 and PHI2), it also provides two system clocks for general use within the system (FCLK and CTTL). FCLK is a fast clock whose frequency is the same as the input clock, while CTTL is a replica of PHI1 clock.

The system control logic and cycle extension logic make the TCU very attractive by providing extremely accurate bus control signals, and allowing extensive control over the bus cycle timing.

Features

- Oscillator at twice the CPU clock frequency
- 2 phase full V_{CC} swing clock drivers (PHI1 and PHI2)
- 4-bit input ($\overline{WAITn}$) allowing precise specification of 0 to 15 wait states
- Cycle Hold for system arbitration and/or memory refresh
- System timing (FCLK, CTTL) and control ($\overline{RD}$, $\overline{WR}$, and $\overline{DBE}$) outputs
- General purpose Timing State Output ($\overline{TSO}$) that identifies internal states
- Peripheral cycle to accommodate slower MOS peripherals
- Provides "ready" (RDY) output for the Series 32000 CPUs
- Synchronous system reset generation from Schmitt trigger input
- Phase synchronization to a reference signal
- High-speed CMOS technology
- TTL compatible inputs
- Single 5V power supply
- 24-pin dual-in-line package

Block Diagram



TL/EE/8524-1

Table of Contents

List of Illustrations

1.0 Functional Description

1.1 POWER AND GROUNDING

The NS32C201 requires a single +5V power supply, applied to pin 24 (V_{CC}). See Electrical Characteristics. The Logic Ground on pin 12 (GND), is the common pin for the TCU.

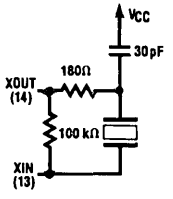
A 0.1 μF, ceramic decoupling capacitor must be connected across V_{CC} and GND, as close to the TCU as possible.

1.2 CRYSTAL OSCILLATOR CHARACTERISTICS

The NS32C201 has an internal oscillator that requires connections of the crystal and bias components to XIN and XOUT as shown in *Figure 1-1.* It is important that the crystal and the RC components be mounted in close proximity to the XIN, XOUT and V_{CC} pins to keep printed circuit trace lengths to an absolute minimum.

Typical Crystal Specifications:

Type ..... At-Cut
Tolerance ..... 0.005% at 25°C
Stability ..... 0.01% from 0° to 70°C
Resonance ..... Fundamental (parallel)
Capacitance ..... 20 pF
Maximum Series Resistance 50Ω



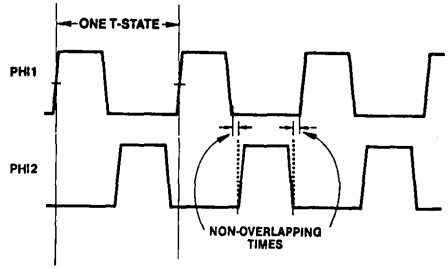
TL/EE/8524-3

CRYSTAL FREQUENCY (MHz)	R (OHM)
6-12	470
12-18	220
18-24	100
24-30	47

FIGURE 1-1. Crystal Connection Diagram

1.3 CLOCKS

The NS32C201 TCU has four clock output pins. The PHI1 and PHI2 clocks are required by the Series 32000 CPUs. These clocks are non-overlapping as shown in *Figure 1-2.*



TL/EE/8524-4

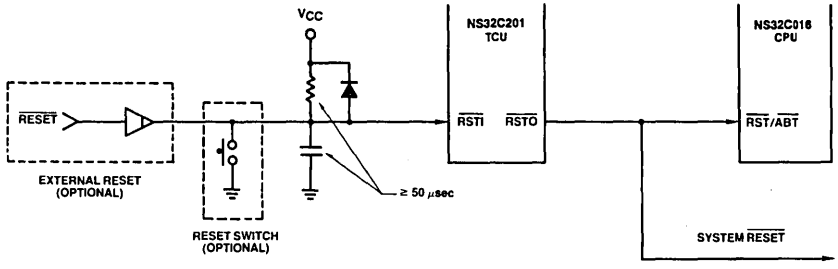
FIGURE 1.2. PHI1 and PHI2 Clock Signals

Each rising edge of PHI1 defines a transition in the timing state of the CPU.

As the TCU generates the various clock signals with very short transition timings, it is recommended that the conductors carrying PHI1 and PHI2 be kept as short as possible. It is also recommended that only the Series 32000 CPU and, if used, the MMU (Memory Management Unit) be connected to the PHI1 and PHI2 clocks.

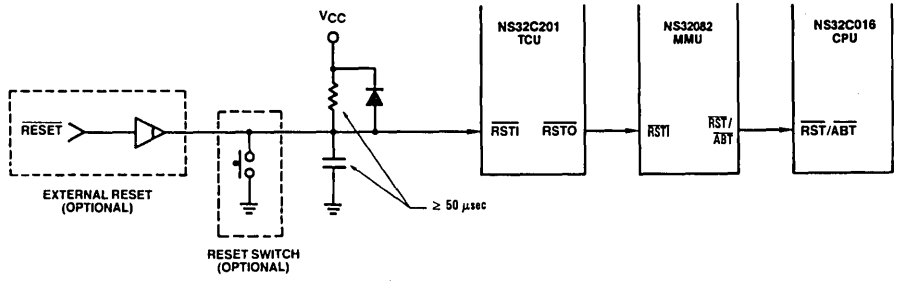
CTTL is a clock signal which runs at the same frequency as PHI1 and is closely balanced with it.

FCLK is a clock, running at the frequency of XIN input. This clock has a frequency that is twice the CTTL clock frequency. The exact phase relationship between PHI1, PHI2, CTTL and FLCK can be found in Section 2.



TL/EE/8524-5

FIGURE 1-3a. Recommended Reset Connections (Non Memory-Managed System)



TL/EE/8524-6

FIGURE 1-3b. Recommended Reset Connections (Memory-Managed System)

1.0 Functional Description (Continued)

1.4 RESETTING

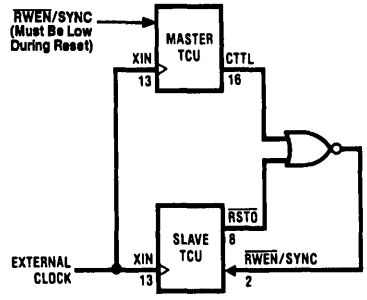
The NS32C201 TCU provides circuitry to meet the reset requirements of the Series 32000 CPUs. If the Reset Input line, $\overline{RSTI}$ is pulled low, the TCU asserts $\overline{RSTO}$ which resets the Series 32000 CPU. This Reset Output may also be used as a system reset signal. *Figure 1-3a* illustrates the reset connections for a non Memory-Managed system. *Figure 1-3b* illustrates the reset connections for a Memory-Managed system.

1.5 SYNCHRONIZING TWO OR MORE TCUs

During reset, (when $\overline{RSTO}$ is low), one or more TCUs can be synchronized with a reference (Master) TCU. The $\overline{RWEN}$/SYNC input to the slave TCU(s) is used for synchronization. The Slave TCU samples the $\overline{RWEN}$/SYNC input on the rising edge of XIN. When $\overline{RSTO}$ is low and CTTL is high (see *Figure 1-5*), if $\overline{RWEN}$/SYNC is sampled high, the phase of CTTL of the Slave TCU is shifted by one XIN clock cycle.

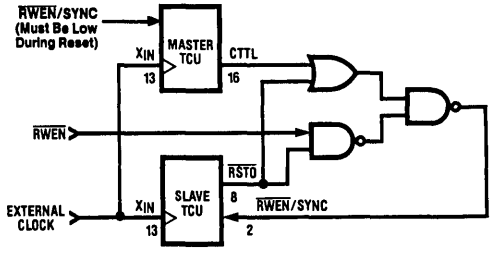
Two possible circuits for TCU synchronization are illustrated in *Figures 1-4a* and *1-4b*. It should be noted that when $\overline{RWEN}$/SYNC is high, the $\overline{RD}$ and $\overline{WR}$ signals will be TRI-STATE on the slave TCU.

Note: $\overline{RWEN}$/SYNC should not be kept constantly high during reset, otherwise the clock will be stopped and the device will not exit reset when $\overline{RSTI}$ is deasserted.



TL/EE/8524–7

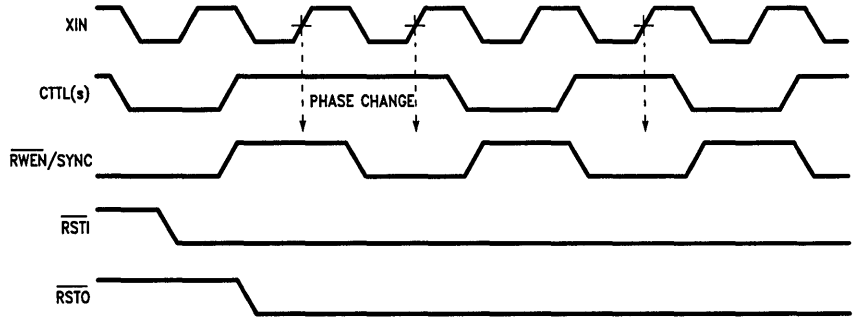
FIGURE 1-4a. Slave TCU Does Not Use $\overline{RWEN}$ During Normal Operation



TL/EE/8524–8

FIGURE 1-4b. Slave TCU Uses Both SYNC and $\overline{RWEN}$

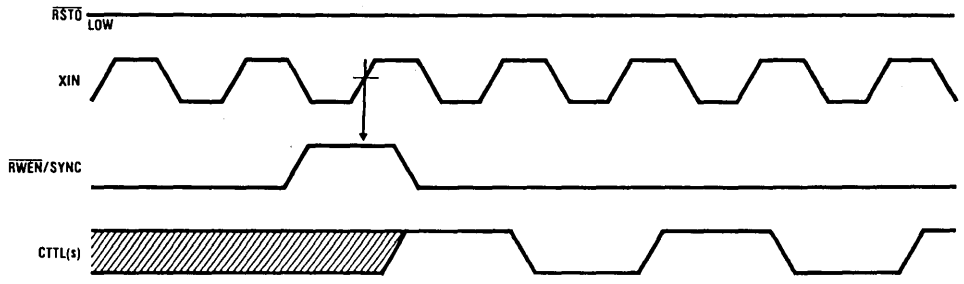
Note: When two or more TCUs are to be synchronized, the XIN of all the TCUs should be connected to an external clock source. For details on the external clock, see Switching Specifications in Section 2.



TL/EE/8524–9

FIGURE 1-5. Synchronizing Two TCUs

1.0 Functional Description (Continued)



TL/EE/8524–10

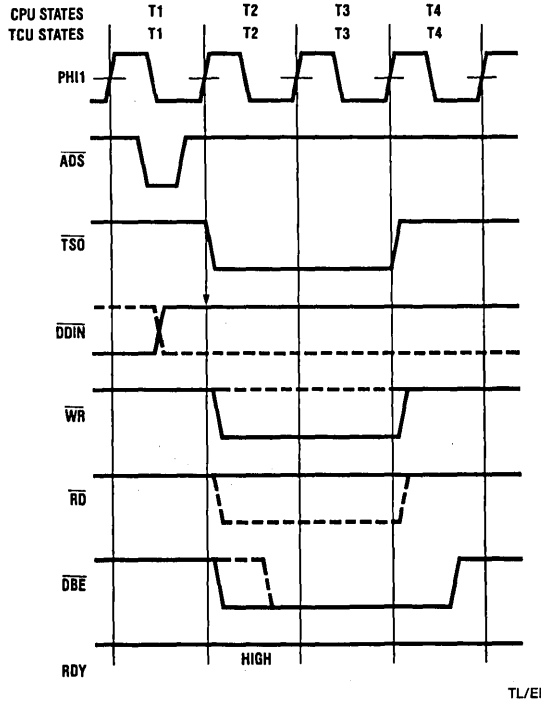
FIGURE 1-6. Synchronizing One TCU to An External Pulse

In addition to synchronizing two or more TCUs, the RWEN/SYNC input can be used to "fix" the phase of one TCU to an external pulse. The pulse to be used must be high for only one rising edge of XIN. Independent of CTTL's state at the XIN rising edge, the CTTL state following the XIN rising edge will be high. *Figure 1-6* shows the timing of this sequence.

1.6 BUS CYCLES

In addition to providing all the necessary clock signals, the NS32C201 TCU provides bus control signals to the system. The TCU senses the ADS signal from the CPU or MMU to start a bus cycle. The DDIN input signal is also sampled to determine whether a Read or Write cycle is to be generated. In addition to RD and WR, other signals are provided: DBE and TSO. DBE is used to enable data buffers. The leading edge of DBE is delayed a half clock period during Read cycles to avoid bus conflicts between data buffers and either the CPU or the MMU. This is shown in *Figure 1-7*.

The Timing State Output (TSO) is a general purpose signal that may be used by external logic for synchronizing to a System cycle. TSO is activated at the beginning of state T2 and returns to the high level at the beginning of state T4 of the CPU cycle. TSO can be used to gate the CWAIT signal when continuous waits are required. Another application of TSO is the control of interface circuitry for dynamic RAMs.



Notes:

1. The CPU and TCU view some timing states (T-states) differently. For clarity, references to T-states will sometimes be followed by (TCU) or (CPU). (CPU) also implies (MMU).
2. Arrows indicate when the TCU samples the input.
3. RWEN is assumed low (RD and WR enabled) unless specified differently.
4. For clarity, T-states for both the TCU and CPU are shown above the diagrams. (See Note 1.)

TL/EE/8524–11

FIGURE 1-7. Basic TCU Cycle (Fast Cycle)

1.0 Functional Description (Continued)

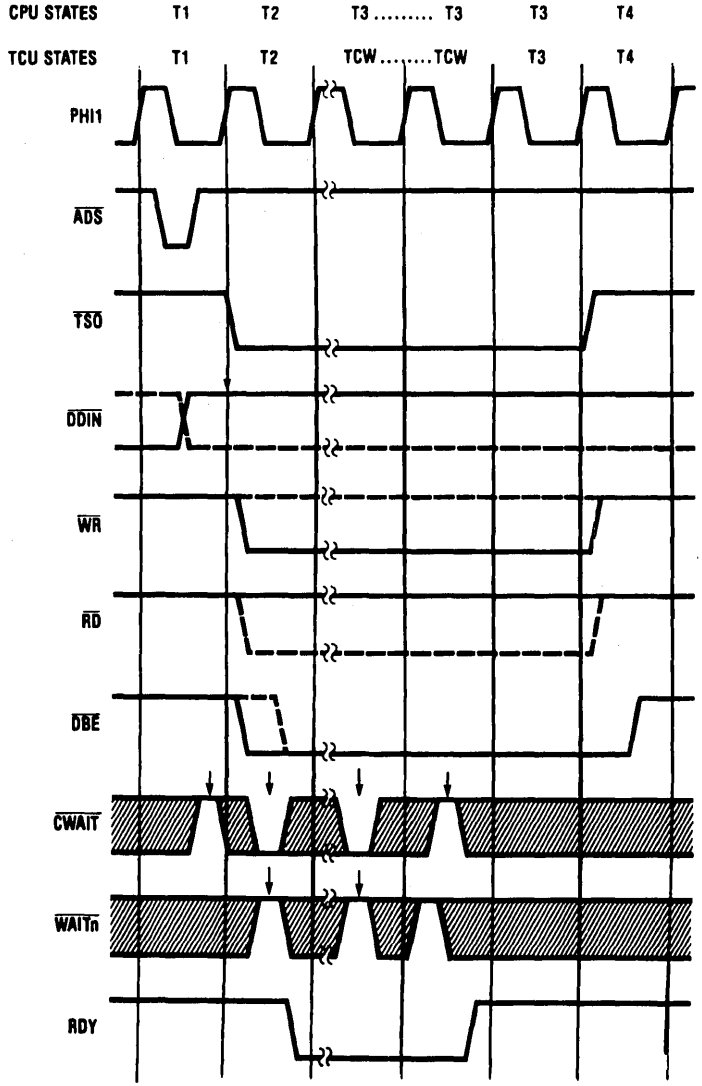
1.7 BUS CYCLE EXTENSION

The NS32C201 TCU uses the Wait input signals to extend normal bus cycles. A normal bus cycle consists of four PHI1 clock cycles. Whenever one or more Wait inputs to the TCU are activated, a bus cycle is extended by at least one PHI1 clock cycle. The purpose is to allow the CPU to access slow memories or peripherals. The TCU responds to the Wait signals by pulling the RDY signal low as long as Wait States are to be inserted in the Bus cycle.

There are three basic cycle extension modes provided by the TCU, as described below.

1.7.1 Normal Wait States

This is a normal Wait State insertion mode. It is initiated by pulling $\overline{\text{CWAIT}}$ or any of the $\overline{\text{WAITn}}$ lines low in the middle of T2. *Figure 1-8* shows the timing diagram of a bus cycle when $\overline{\text{CWAIT}}$ is sampled high at the end of T1 and low in the middle of T2.



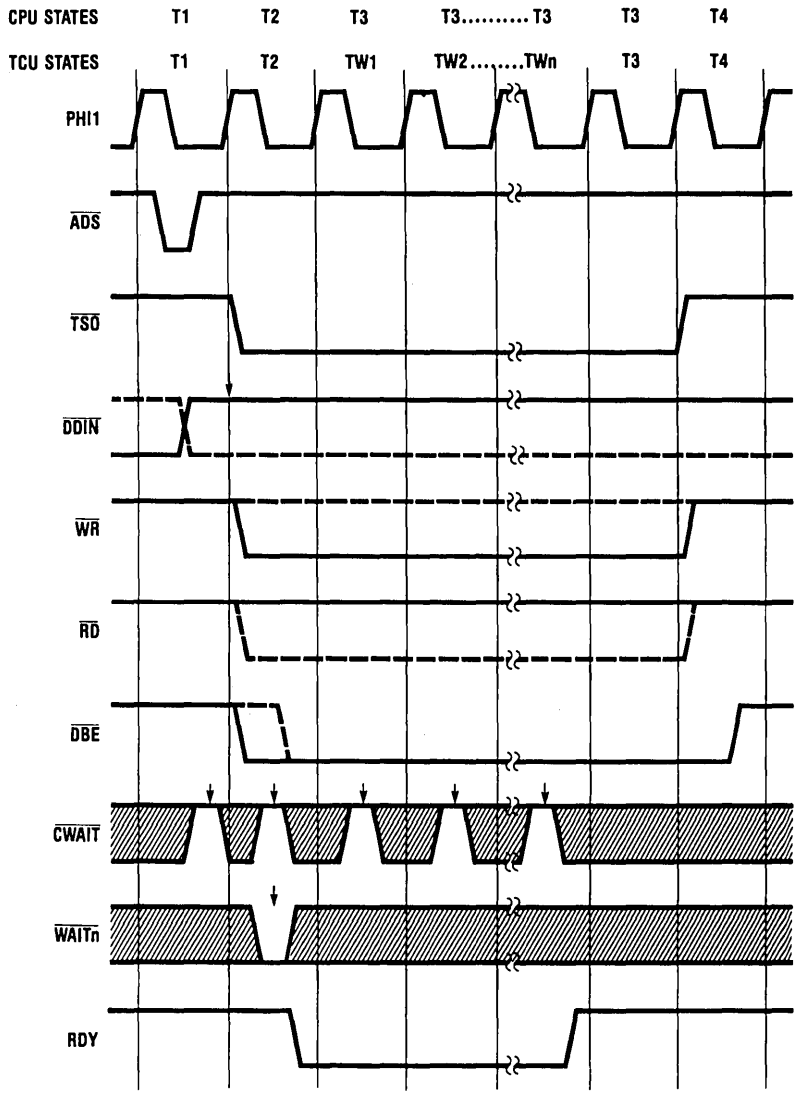
TL/EE/8524-12

FIGURE 1-8. Wait State Insertion Using $\overline{\text{CWAIT}}$ (Fast Cycle)

1.0 Functional Description (Continued)

The RDY signal goes low during T2 and remains low until $\overline{CWAIT}$ is sampled high by the TCU. RDY is pulled high by the TCU during the same PHI1 cycle in which the $\overline{CWAIT}$ line is sampled high.

If any of the $\overline{WAITn}$ signals are sampled low during T2 and $\overline{CWAIT}$ is high during the entire bus cycle, then the RDY line goes low for 1 to 15 clock cycles, depending on the binary weighted value of $\overline{WAITn}$. If, for example, $\overline{WAIT1}$ and $\overline{WAIT4}$ are sampled low, then five wait states will be inserted. This is shown in *Figure 1-9.*



TL/EE/8524–13

FIGURE 1-9. Wait State Insertion Using $\overline{WAITn}$ (Fast Cycle)

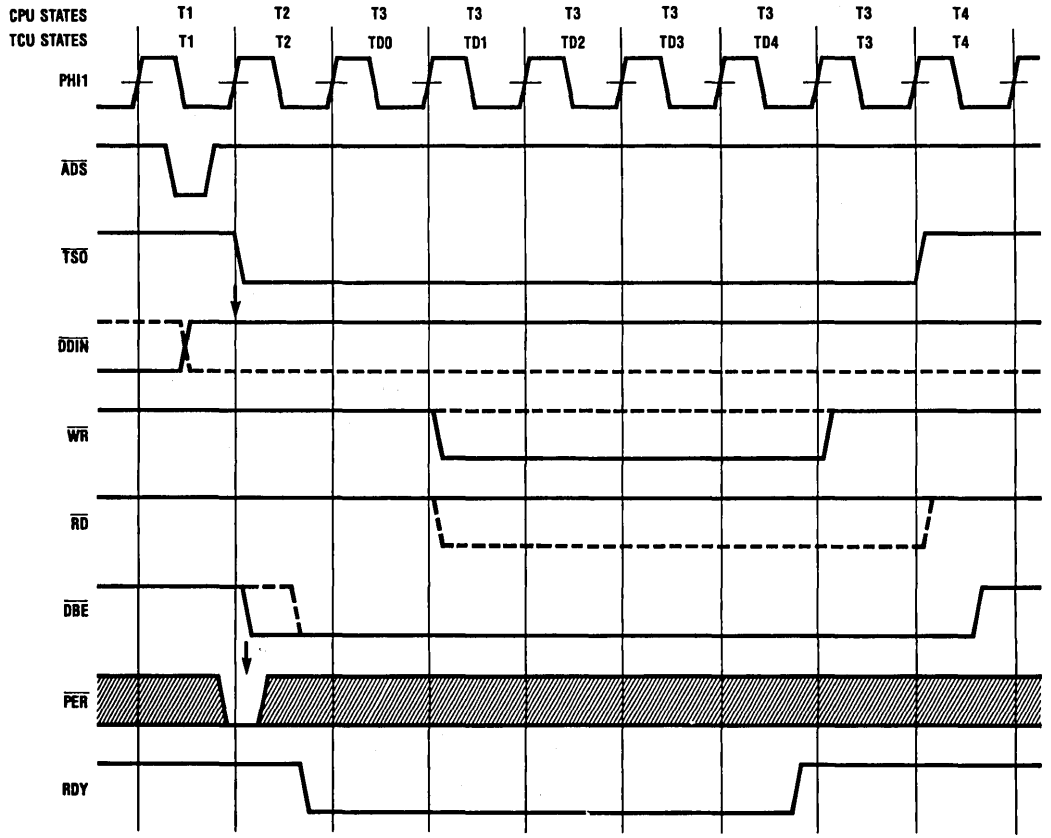
1.0 Functional Description (Continued)

1.7.2 Peripheral Cycle

This cycle is entered when the $\overline{PER}$ signal line is sampled low at the beginning of T2. The TCU adds five wait states identified as TD0–TD4 into a normal bus cycle. The $\overline{RD}$ and $\overline{WR}$ signals are also re-shaped so the setup and hold times for address and data will be increased.

This may be necessary when slower peripherals must be accessed.

Figure 1-10 shows the timing diagram of a peripheral cycle.



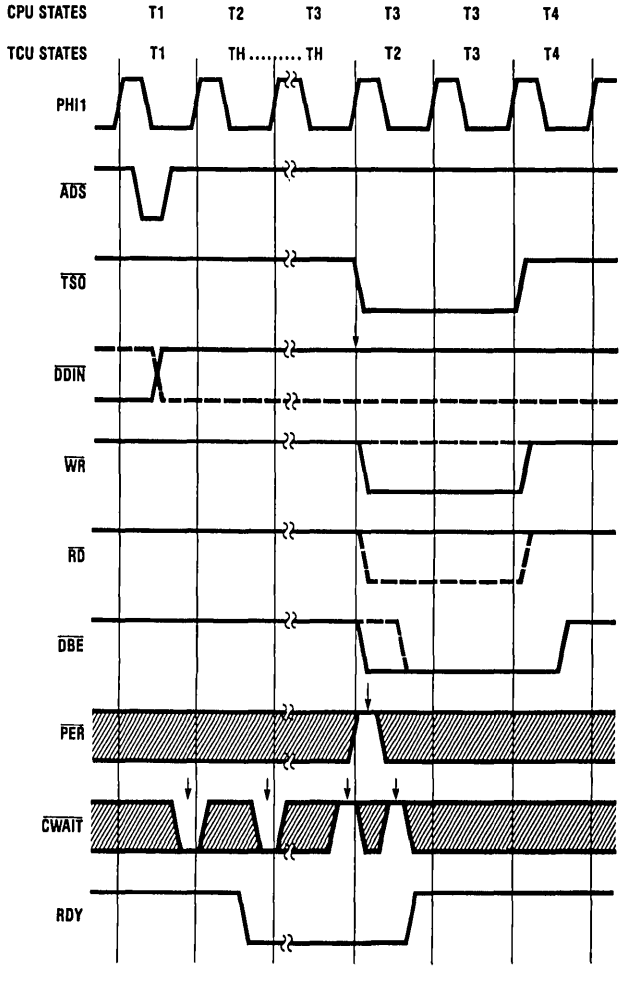
TL/EE/8524–14

FIGURE 1-10. Peripheral Cycle

1.0 Functional Description (Continued)

1.7.3 Cycle Hold

If the $\overline{CWAIT}$ input is sampled low at the end of state T1, the TCU will go into cycle hold mode and stay in this mode for as long as $\overline{CWAIT}$ is kept low. During this mode the control signals $\overline{RD}$, $\overline{WR}$, $\overline{TSO}$ and $\overline{DBE}$ are kept inactive; RDY is pulled low, thus causing wait states to be inserted into the bus cycle. The cycle hold feature can be used in applications involving dynamic RAMs. A timing diagram showing the cycle hold feature is shown in *Figure 1-11*.



TL/EE/8524–15

FIGURE 1-11. Cycle Hold Timing Diagram

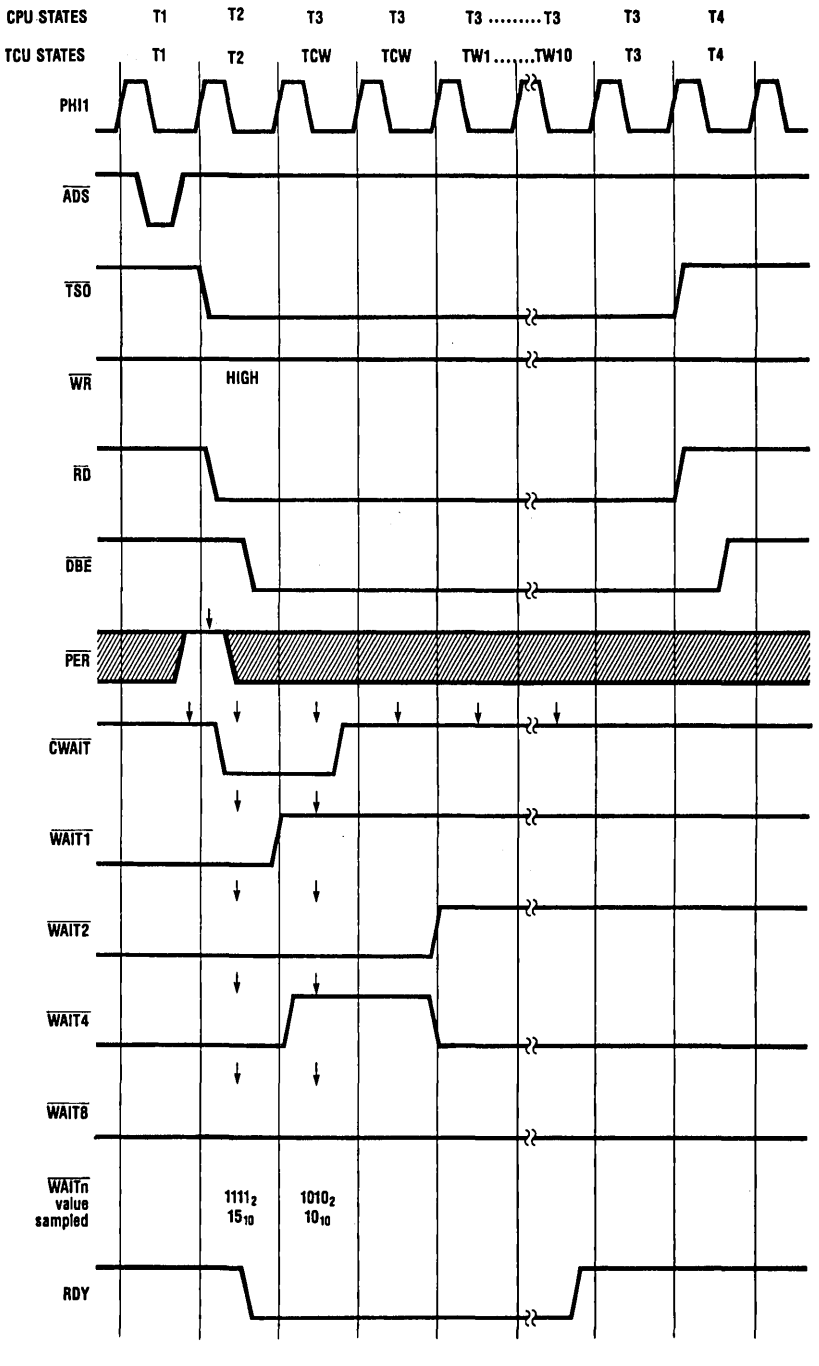
1.8 BUS CYCLE EXTENSION COMBINATIONS

Any combination of the TCU input signals used for extending a bus cycle can be activated at one time. The TCU will honor all of the requests according to a certain priority scheme. A cycle hold request is assigned top priority. It follows a peripheral cycle request, and then $\overline{CWAIT}$ and $\overline{WAITn}$ respectively.

If, for example, all the input signals $\overline{CWAIT}$, $\overline{PER}$ and $\overline{WAITn}$ are asserted at the beginning of the cycle, the TCU will enter the cycle hold mode. As soon as $\overline{CWAIT}$ goes high, the input signal $\overline{PER}$ is sampled to determine whether a peripheral cycle is requested.

Next, the TCU samples $\overline{CWAIT}$ again and $\overline{WAITn}$ to check whether additional wait states have to be inserted into the bus cycle. This sampling point depends on whether $\overline{PER}$ was sampled high or low. If $\overline{PER}$ was sampled high, then the sampling point will be in the middle of the TCU state T2, (*Figure 1-14*), otherwise it will occur three clock cycles later (*Figure 1-15*). *Figures 1-12* to *1-15* show the timing diagrams for different combinations of cycle extensions.

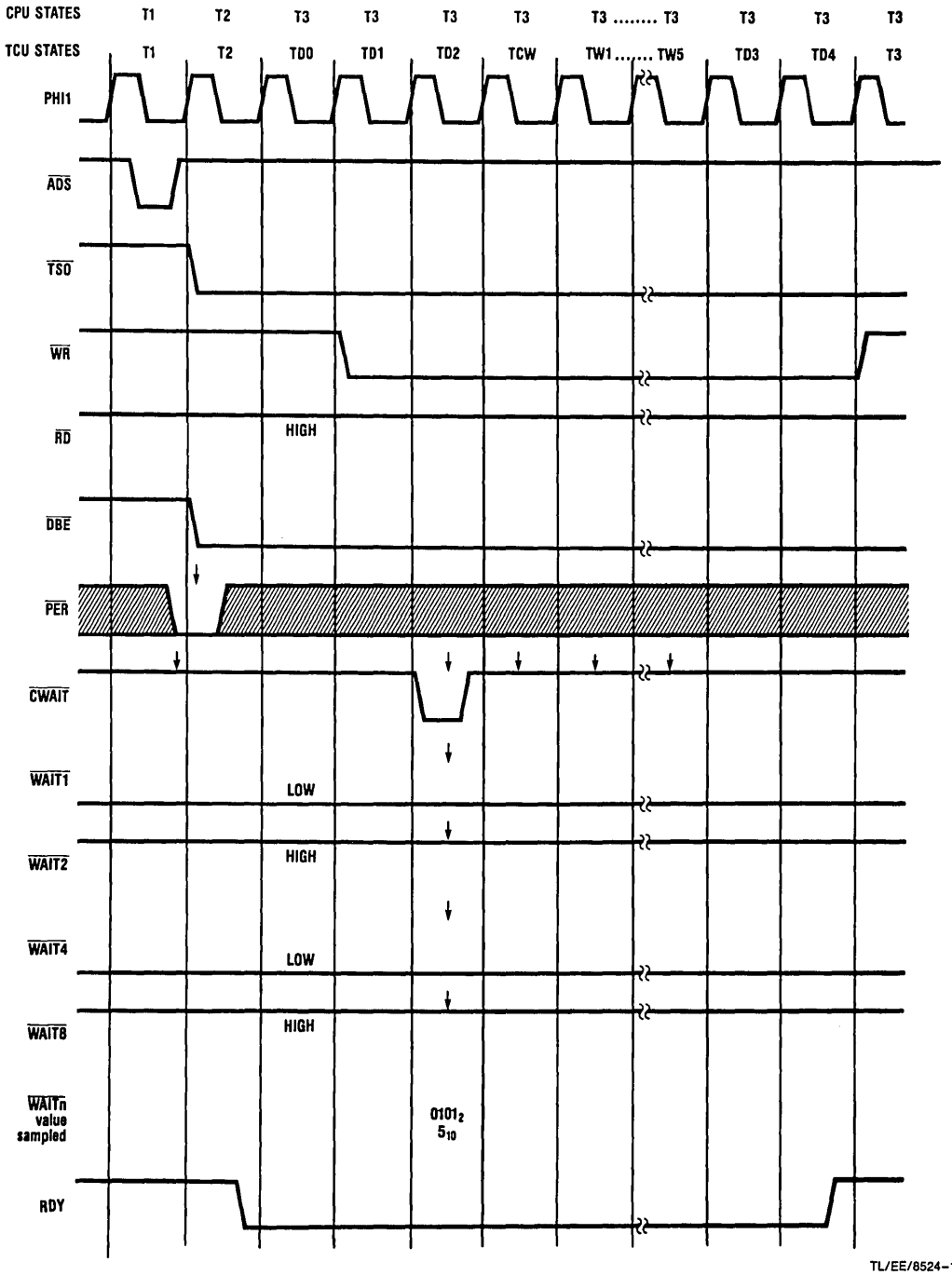
1.0 Functional Description (Continued)



TL/EE/8524-16

FIGURE 1-12. Fast Cycle With 12 Wait States (2 $\overline{\text{CWAIT}}$ and $\overline{\text{WAIT10}}$) (Read Cycle)

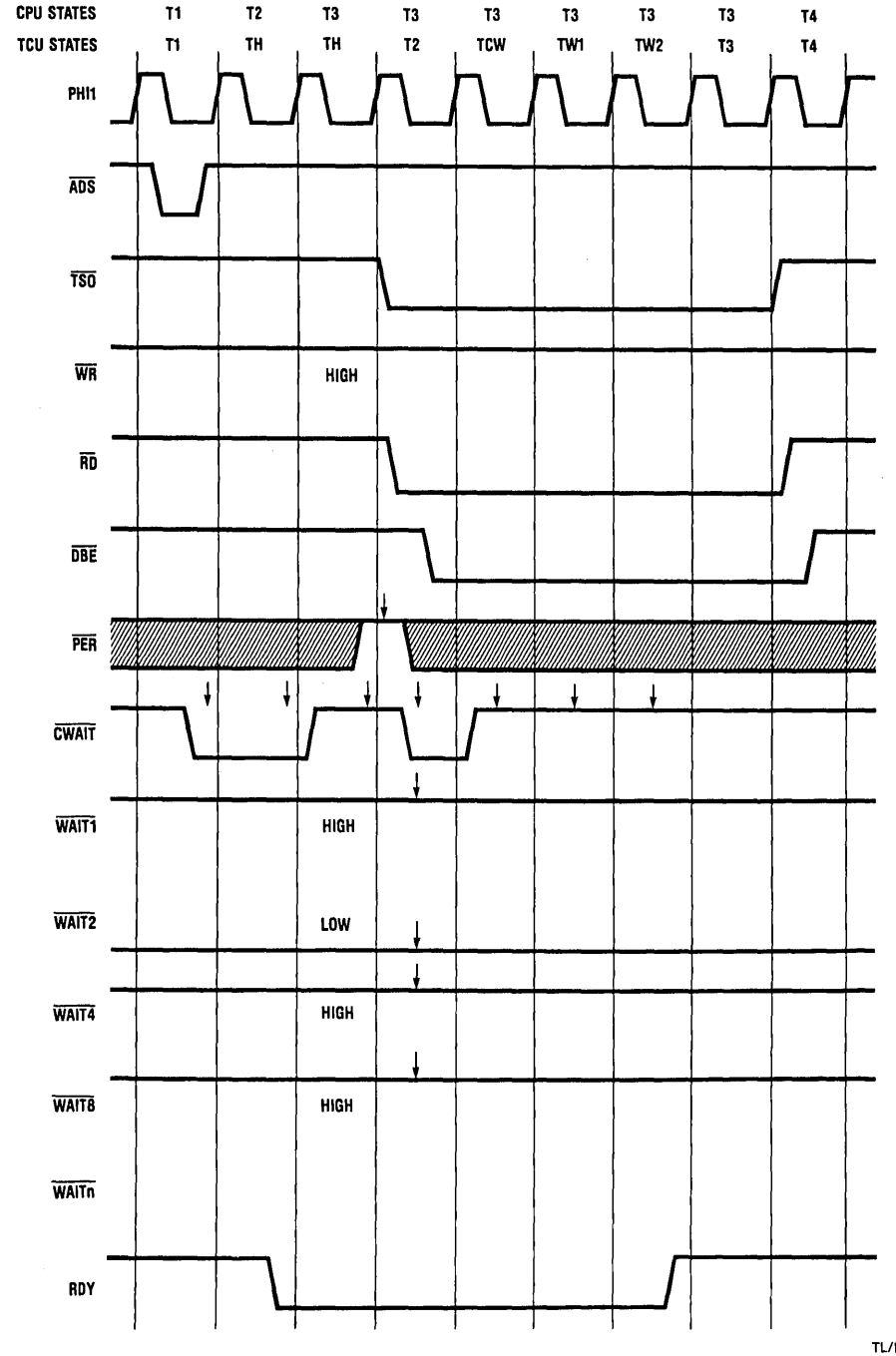
1.0 Functional Description (Continued)



TL/EE/8524–17

FIGURE 1-13. Peripheral Cycle with Six Wait States (1 $\overline{\text{CWAIT}}$ and $\overline{\text{WAIT5}}$) (Write Cycle)

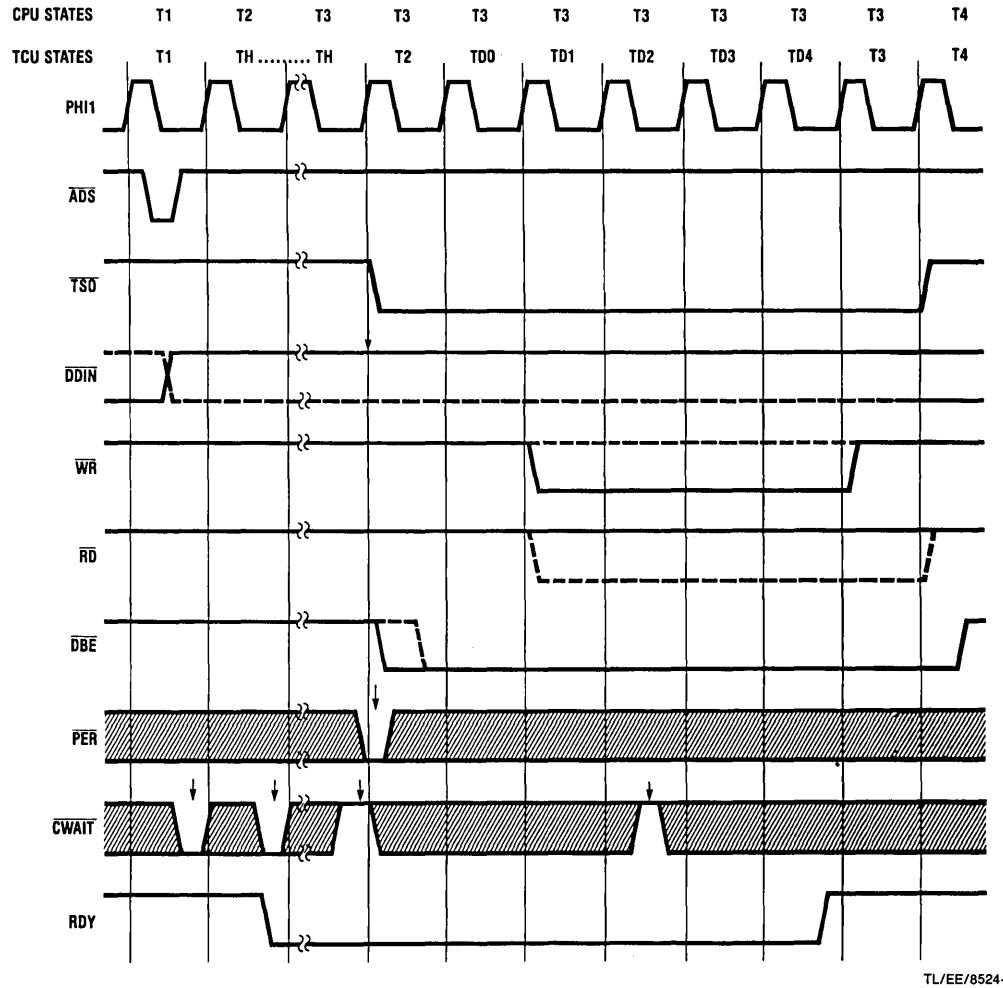
1.0 Functional Description (Continued)



TL/EE/8524-18

FIGURE 1-14. Cycle Hold with Three Wait States (1 CWAIT and WAIT2) (Read Cycle)

1.0 Functional Description (Continued)



TL/EE/8524–19

FIGURE 1-15. Cycle Hold of a Peripheral Cycle

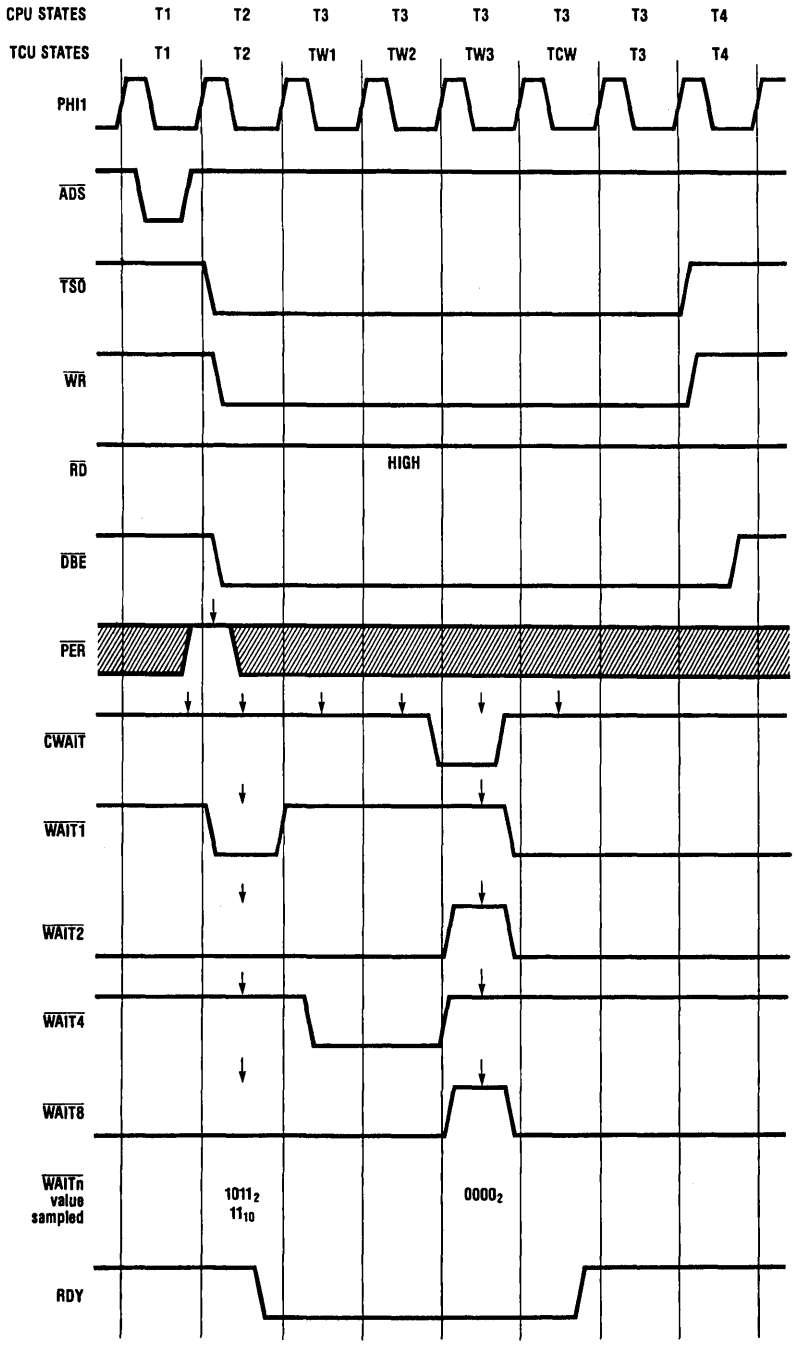
1.9 OVERRIDING WAITn WAIT STATES

The TCU handles the WAITn Wait States by means of an internal counter that is reloaded with the binary value corresponding to the state of the WAITn inputs each time CWAIT is sampled low, and is decremented when CWAIT is high.

This allows to either extend a bus cycle of a predefined number of clock cycles, or prematurely terminate it. To terminate a bus cycle, for example, CWAIT must be asserted for at least one clock cycle, and the WAITn inputs must be forced to their inactive state.

At least one wait state is always inserted when using this procedure as a result of CWAIT being sampled low. *Figure 1-16* shows the timing diagram of a prematurely terminated bus cycle where eleven wait states were being inserted.

1.0 Functional Description (Continued)



TL/EE/8524–20

FIGURE 1-16. Overriding $\overline{\text{WAITn}}$ Wait States (Write Cycle)

2.0 Device Specifications

2.1 PIN DESCRIPTIONS

The following is a description of all NS32C201 pins. The descriptions reference portions of the Functional Description, Section 1.

2.1.1 Supplies

Power (V_{CC}): +5V positive supply. Section 1.1.

Ground (GND): Power supply return. Section 1.1.

2.1.2 Input Signals

Reset Input ($\overline{RSTI}$): Active low. Schmitt triggered, asynchronous signal used to generate a system reset. Section 1.4.

Address Strobe ($\overline{ADS}$): Active low. Identifies the first timing state (T1) of a bus cycle.

Data Direction Input ($\overline{DDIN}$): Active low. Indicates the direction of the data transfer during a bus cycle. Implies a Read when low and a Write when high.

Note: In Rev. A of the NS32C201 this signal is CMOS compatible. In later revisions it is TTL compatible.

Read/Write Enable and Synchronization ($\overline{RWEN}$/SYNC): TRI-STATE® the $\overline{RD}$ and the $\overline{WR}$ outputs when high and enables them when low. Also used to synchronize the phase of the TCU clock signals, when two or more TCUs are used. Section 1.5.

Crystal or External Clock Source (XIN): Input from a crystal or an external clock source. Section 1.3.

Continuous Wait ($\overline{CWAIT}$): Active low. Initiates a continuous wait if sampled low in the middle of T2 during a Fast cycle, or in the middle of TD2, during a peripheral cycle. If $\overline{CWAIT}$ is low at the end of T1, it initiates a Cycle Hold. Section 1.7.1.

Four-Bit Wait State Inputs ($\overline{WAIT1}$, $\overline{WAIT2}$, $\overline{WAIT4}$ and $\overline{WAIT8}$): Active low. These inputs, (collectively called $\overline{WAITn}$), allow from zero to fifteen wait states to be specified. They are binary weighted. Section 1.7.1.

Peripheral Cycle ($\overline{PER}$): Active low. If active, causes the TCU to insert five wait states into a normal bus cycle. It also causes the Read and Write signals to be re-shaped to meet the setup and hold timing requirement of slower MOS peripherals. Section 1.7.2.

2.1.3 Output Signals

Reset Output ($\overline{RSTO}$): Active low. This signal becomes active when $\overline{RSTI}$ is low, initiating a system reset. $\overline{RSTO}$ goes high on the first rising edge of PHI1 after $\overline{RSTI}$ goes high. Section 1.4.

Read Strobe ($\overline{RD}$): (TRI-STATE) Active low. Identifies a Read cycle. It is decoded from $\overline{DDIN}$ and TRI-STATE by $\overline{RWEN}$/SYNC. Section 1.6.

Write Strobe ($\overline{WR}$): (TRI-STATE) Active low. Identifies a Write cycle. It is decoded from $\overline{DDIN}$ and TRI-STATE by $\overline{RWEN}$/SYNC. Section 1.6.

Note: $\overline{RD}$ and $\overline{WR}$ are mutually exclusive in any cycle. Hence they are never low at the same time.

Data Buffer Enable ($\overline{DBE}$): Active low. This signal is used to control the data bus buffers. It is low when the data buffers are to be enabled. Section 1.6.

Timing State Output ($\overline{TSO}$): Active low. The falling edge of $\overline{TSO}$ signals the beginning of state T2 of a bus cycle. The rising edge of $\overline{TSO}$ signals the beginning of state T4. Section 1.6.

Ready (RDY): Active high. This signal will go low and remain low as long as wait states are to be inserted in a bus cycle. It is normally connected to the RDY input of the CPU. Section 1.7.

Fast Clock (FCLK): This is a clock running at the same frequency as the crystal or the external source. Its frequency is twice that of the CPU clocks. Section 1.3.

CPU Clocks (PHI1 and PHI2): These outputs provide the Series 32000 CPU with two phase, non-overlapping clock signals. Their frequency is half that of the crystal or external source. Section 1.3.

System Clock (CTTL): This is a system version of the PHI1 clock. Hence, it operates at the CPU clock frequency. Section 1.3.

Crystal Output (XOUT): This line is used as the return path for the crystal (if used). It must be left open when an external clock source is used to drive XIN. Section 1.2.

2.0 Device Specifications (Continued)

2.2 ABSOLUTE MAXIMUM RATINGS (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage	7V
Input Voltages	$-0.5V$ to $V_{CC} + 0.5V$
Output Voltages	$-0.5V$ to $V_{CC} + 0.5V$
Storage Temperature	$-65°C$ to $+150°C$
Lead Temperature (Soldering, 10 sec.)	300°C
Continous Power Dissipation	1W

Note: *Absolute maximum ratings indicate limits beyond which permament damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

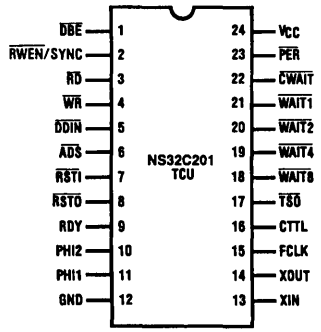
2.3 ELECTRICAL CHARACTERISTICS $T_A = -40°C$ to $+85°C$, $V_{CC} = 5V \pm 5\%$, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IL}	Input Low Voltage	All Inputs Except $\overline{RSTI}$ & XIN			0.8	V
V_{IH}	Input High Voltage	All Inputs Except $\overline{RSTI}$ & XIN	2.0			V
V_{T+}	$\overline{RSTI}$ Rising Threshold Voltage	$V_{CC} = 5.0V$	2.5		3.5	V
V_{HYS}	$\overline{RSTI}$ Hysteresis Voltage	$V_{CC} = 5.0V$	0.8		1.9	V
V_{XL}	XIN Input Low Voltage				0.20 V_{CC}	V
V_{XH}	XIN Input High Voltage		0.80 V_{CC}			V
I_{IL}	Input Low Current	$V_{IN} = 0V$			-10	μA
I_{IH}	Input High Current	$V_{IN} = V_{CC}$			10	μA
V_{OL}	Output Low Voltage	PHI1 & PHI2, I = 1 mA All Other Outputs Except XOUT, I = 2 mA			0.10 V_{CC}	V
V_{OH}	Output High Voltage	All Outputs Except XOUT, I = -1 mA	0.90 V_{CC}			V
I_L	Leakage Current on $\overline{RD}/\overline{WR}$	$0.4V \le V_{IN} \le V_{CC}$	-20		$+20$	μA
I_{CC}	Supply Current	$f_{xin} = 20$ MHz		100	120	mA

Note 1: All typical values are for $V_{CC} = 5V$ and $T_A = 25°C$.

Connection Diagram

Dual-In-Line Package



TL/EE/8524-2

Top View

Order Number NS32C201D or NS32C201N
See NS Package Number D24C or N24A

FIGURE 2.1

2.0 Device Specifications (Continued)

2.4 SWITCHING CHARACTERISTICS

2.4.1 Definitions

All the timing specifications given in this section refer to 2.0V on the rising or falling edges of the clock phases PHI1 and PHI2; to 15% or 85% of V_{CC} on all the CMOS output signals, and to 0.8V or 2.0V on all the TTL input signals, unless specifically stated otherwise.

2.4.2 Output Loading

Capacitive loading on output pins for the NS32C201.

RDY, $\overline{DBE}$, $\overline{TSO}$ ..... 50 pF
$\overline{RD}$, $\overline{WR}$ ..... 75 pF
CTTL ..... 50 ÷ 100 pF
FCLK ..... 100 pF
PHI1, PHI2 ..... 170 pF

ABBREVIATIONS

L.E.—Leading Edge
T.E.—Trailing Edge
R.E.—Rising Edge
F.E.—Falling Edge

2.4.3 Timing Tables

Symbol	Figure	Description	Reference/Conditions	NS32C201-10		NS32C201-15		Units
				Min	Max	Min	Max	
CLOCK-SIGNALS (XIN, FCLK, PHI1 & PHI2) TIMING								
t_{Cp}	2.2	Clock Period	PHI1 R.E. to Next PHI1 R.E.	100		66		ns
t_{CLh}	2.2	Clock High Time	At 90% V_{CC} on PHI1 (Both Edges)	0.5 t_{Cp} −15 ns	0.5 t_{Cp} −7 ns	0.5 t_{Cp} −10 ns	0.5 t_{Cp} −3 ns	
t_{CLl}	2.2	Clock Low Time	At 15% V_{CC} on PHI1	0.5 t_{Cp} −5 ns	0.5 t_{Cp} +10 ns	0.5 t_{Cp} −5 ns	0.5t_{CP} +6 ns	
$t_{CLw(1,2)}$	2.2	Clock Pulse Width	At 2.0V on PHI1, PHI2 (Both Edges)	0.5 t_{CP} −10 ns	0.5 t_{CP} −4 ns	0.5 t_{CP} −6 ns	0.5 t_{CP} −4 ns	
t_{CLwas}		PHI1, PHI2 Asymmetry (t_{CLw} (1) − t_{CLw} (2))	At 2.0V on PHI1, PHI2	−5	5	−3	3	ns
t_{CLR}	2.2	Clock Rise Time	15% to 90% V_{CC} on PHI1 R.E.		8		6	ns
t_{CLF}	2.2	Clock Fall Time	90% to 15% V_{CC} on PHI1 F.E.		8		6	ns
$t_{nOVL(1,2)}$	2.2	Clock Non-Overlap Time	At 15% V_{CC} on PHI1, PHI2	−2	+2	−2	+2	ns
t_{nOVLas}		Non-Overlap Asymmetry (t_{nOVL} (1) − t_{nOVL} (2))	At 15% V_{CC} on PHI1, PHI2	−4	4	−3	3	ns
t_{Xh}	2.2	XIN High Time (External Input)	At 80% V_{CC} on XIN (Both Edges)	16		10		ns
t_{Xl}	2.2	XIN Low Time (External Input)	At 15% V_{CC} on XIN (Both Edges)	16		10		ns
t_{XFr}	2.2	XIN to FCLK R.E. Delay	80% V_{CC} on XIN R.E. to FCLK R.E.	6	29	6	25	ns
t_{XFf}	2.2	XIN to FCLK F.E. Delay	15% V_{CC} on XIN F.E. to FCLK F.E.	6	29	6	25	ns
t_{XCr}	2.2	XIN to CTTL R.E. Delay	80% V_{CC} on XIN R.E. to CTTL R.E.	6	34	6	25	ns
t_{XPr}	2.2	XIN to PHI1 R.E. Delay	80% V_{CC} on XIN R.E. to PHI1 R.E.	6	32	6	25	ns
t_{FCr}	2.2	FCLK to CTTL R.E. Delay	FCLK R.E. to CTTL R.E.	0	6	0	6	ns
t_{FCf}	2.2	FCLK to CTTL F.E. Delay	FCLK R.E. to CTTL F.E.	−3	4	−3	4	ns
t_{FPr}	2.3	FCLK to PHI1 R.E. Delay	FCLK R.E. to PHI1 R.E.	−3	4	−3	4	ns
t_{FPf}	2.3	FCLK to PHI1 F.E. Delay	FCLK R.E. to PHI1 F.E.	−5	2	−5	2	ns
t_{Fw}	2.3	FCLK Pulse Width with Crystal	At 50% V_{CC} on FCLK (Both Edges)	0.25 t_{Cp} −5 ns	0.25 t_{Cp} +5 ns	0.25 t_{Cp} −5 ns	0.25 t_{Cp} +5 ns	
t_{PCf}	2.3	PHI2 R.E.to CTTL F.E. Delay	PHI2 R.E. to CTTL F.E.	−3	4	−3	3	ns
t_{CTw}	2.3	CTTL Pulse Width	At 50% V_{CC} on CTTL (Both Edges)	0.5 t_{Cp} −7 ns	0.5 t_{Cp} +1 ns	0.5 t_{Cp} −5 ns	0.5 t_{Cp} +1 ns	

Note 1: t_{XCr}, t_{FCr}, t_{FCf}, t_{PCf}, t_{CTh} are measured with 100 pF load on CTTL.

Note 2: PHI1 and PHI2 are interchangeable for the following parameters: t_{Cp}, t_{CLh}, t_{CLl}, t_{CLw}, t_{CLR}, t_{CLF}, t_{nOVL}, t_{XPr}, t_{FPr}, t_{FPf}.

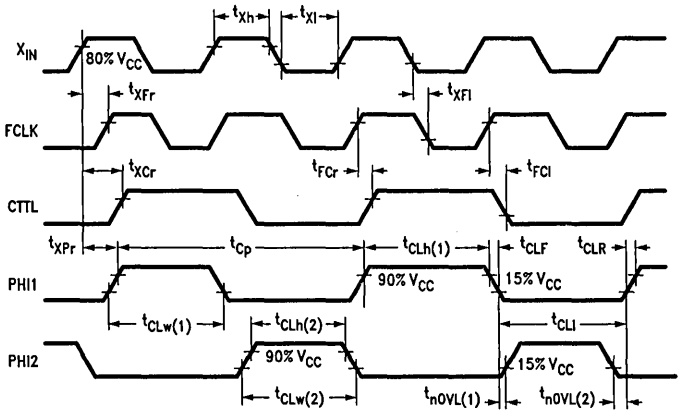
2.0 Device Specifications (Continued)

2.4.3 Timing Tables (Continued)

Symbol	Figure	Description	Reference/Conditions	NS32C201-10		NS32C201-15		Units
				Min	Max	Min	Max	
CTTL TIMING (CL = 50 pF)								
t_{PCr}	2.3	PHI1 to CTTL R.E. Delay	PHI1 R.E. to CTTL R.E.	−2	5	−2	3	ns
t_{CTR}	2.3	CTTL Rise Time	10% to 90% V_{CC} on CTTL R.E.		7		6	ns
t_{CTF}	2.3	CTTL Fall Time	90% to 10% V_{CC} on CTTL F.E.		7		6	ns
CTTL TIMING (CL = 100 pF)								
t_{PCr}	2.3	PHI1 to CTTL R.E. Delay	PHI1 R.E. to CTTL R.E.	−2	6	−2	4	ns
t_{CTR}	2.3	CTTL Rise Time	10% to 90% V_{CC} on CTTL R.E.		8		7	ns
t_{CTF}	2.3	CTTL Fall Time	90% to 10% V_{CC} on CTTL F.E.		8		7	ns
CONTROL INPUTS ($\overline{RST1}$, $\overline{ADS}$, $\overline{DDIN}$) TIMING								
t_{RSTs}	2.4	$\overline{RST1}$ Setup Time	Before PHI1 R.E.	20		15		
t_{ADs}	2.4	$\overline{ADS}$ Setup Time	Before PHI1 R.E.	25		20		ns
t_{ADw}	2.4	$\overline{ADS}$ Pulse Width	$\overline{ADS}$ L.E. to $\overline{ADS}$ T.E.	25		20		ns
t_{DDs}	2.4	$\overline{DDIN}$ Setup Time	Before PHI1 R.E.	15		13		ns
CONTROL OUTPUTS ($\overline{RSTO}$, $\overline{TSO}$, $\overline{RD}$, $\overline{WR}$, $\overline{DBE}$ & $\overline{RWEN}$/SYNC) TIMING								
t_{RSTr}	2.4	$\overline{RSTO}$ R.E. Delay	After PHI1 R.E.		21		10	ns
t_{Tf}	2.5	$\overline{TSO}$ L.E. Delay	After PHI1 R.E.		12		8	ns
t_{Tr}	2.5	$\overline{TSO}$ T.E. Delay	After PHI1 R.E.	3	18	3	10	ns
$t_{RWf(F)}$	2.5	$\overline{RD}$/$\overline{WR}$ L.E. Delay (Fast Cycle)	After PHI1 R.E.		30		21	ns
$t_{RWf(S)}$	2.6	$\overline{RD}$/$\overline{WR}$ L.E. Delay (Peripheral Cycle)	After PHI1 R.E.		25		15	ns
t_{RWr}	2.5/6	$\overline{RD}$/$\overline{WR}$ T.E. Delay	After PHI1 R.E.	3	20	3	15	ns
$t_{DBf(W)}$	2.5/6	$\overline{DBE}$ L.E. Delay (Write Cycle)	After PHI1 R.E.		25		15	ns
$t_{DBf(R)}$	2.5/6	$\overline{DBE}$ L.E. Delay (Read Cycle)	After PHI2 R.E.		20		11	ns
t_{DBr}	2.5/6	$\overline{DBE}$ T.E. Delay	After PHI2 R.E.		20		15	ns
t_{pLZ}	2.7	$\overline{RD}$,$\overline{WR}$ Low Level to TRI-STATE	After $\overline{RWEN}$/SYNC R.E.		25		20	ns
t_{pHZ}	2.7	$\overline{RD}$,$\overline{WR}$ High Level to TRI-STATE	After $\overline{RWEN}$/SYNC R.E.		20		15	ns
t_{pZL}	2.7	$\overline{RD}$,$\overline{WR}$ TRI-STATE to Low Level	After $\overline{RWEN}$/SYNC F.E.		25		18	ns
t_{pZH}	2.7	$\overline{RD}$,$\overline{WR}$ TRI-STATE to High Level	After $\overline{RWEN}$/SYNC F.E.		25		18	ns
WAIT STATES & CYCLE HOLD ($\overline{CWAIT}$, $\overline{WAITn}$, $\overline{PER}$ & $\overline{RDY}$) TIMING								
$t_{CWs(H)}$	2.8	$\overline{CWAIT}$ Setup Time (Cycle Hold)	Before PHI1 R.E.	30		20		ns
$t_{CWh(H)}$	2.8	$\overline{CWAIT}$ Hold Time (Cycle Hold)	After PHI1 R.E.	0		0		ns
$t_{CWs(W)}$	2.8/9	$\overline{CWAIT}$ Setup Time (Wait States)	Before PHI2 R.E.	10		6		ns
$t_{CWh(W)}$	2.9	$\overline{CWAIT}$ Hold Time (Wait States)	After PHI2 R.E.	20		10		ns
t_{Ws}	2.9	$\overline{WAITn}$ Setup Time	Before PHI2 R.E.	7		6		ns
t_{Wh}	2.9	$\overline{WAITn}$ Hold Time	After PHI2 R.E.	15		10		ns
t_{Ps}	2.10	$\overline{PER}$ Setup Time	Before PHI1 R.E.	7		5		ns
t_{Ph}	2.10	$\overline{PER}$ Hold Time	After PHI1 R.E.	30		20		ns
t_{Rd}	2.8/9/10	$\overline{RDY}$ Delay	After PHI2 R.E.		25		12	ns
SYNCHRONIZATION (SYNC) TIMING								
t_{Sys}	2.11	SYNC Setup Time	Before XIN R.E.	6		6		ns
t_{Syh}	2.11	SYNC Hold Time	After XIN R.E.	5		5		ns
t_{CS}	2.11	CTTL/SYNC Inversion Delay	CTTL (master) to $\overline{RWEN}$/SYNC (slave)		10		7	ns

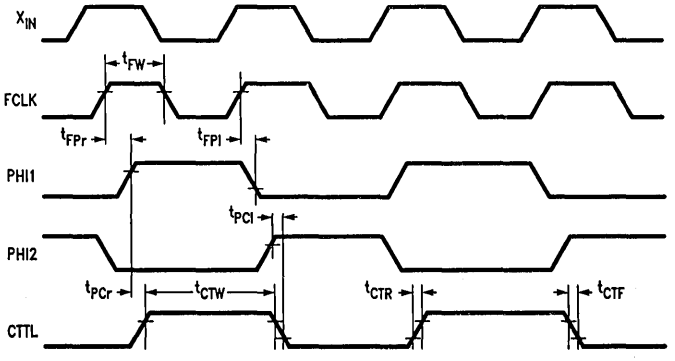
2.0 Device Specifications (Continued)

2.4.4 Timing Diagrams



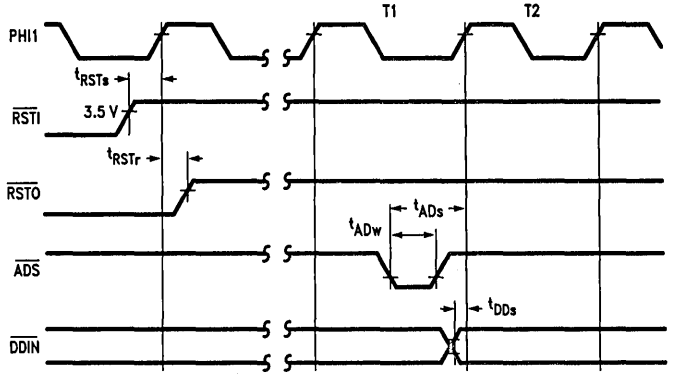
TL/EE/8524-21

FIGURE 2-2. Clock Signals (a)



TL/EE/8524-22

FIGURE 2-3. Clock Signals (b)

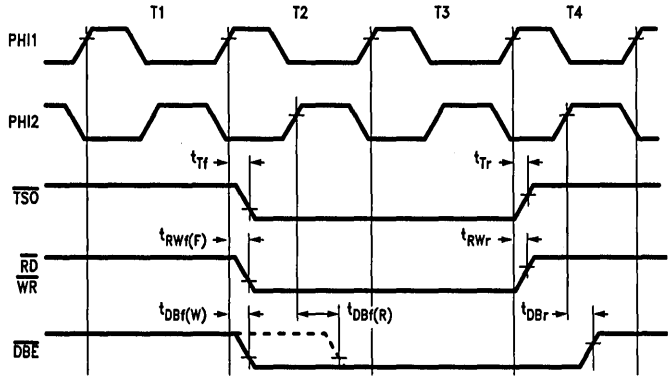


TL/EE/8524-23

FIGURE 2-4. Control Inputs

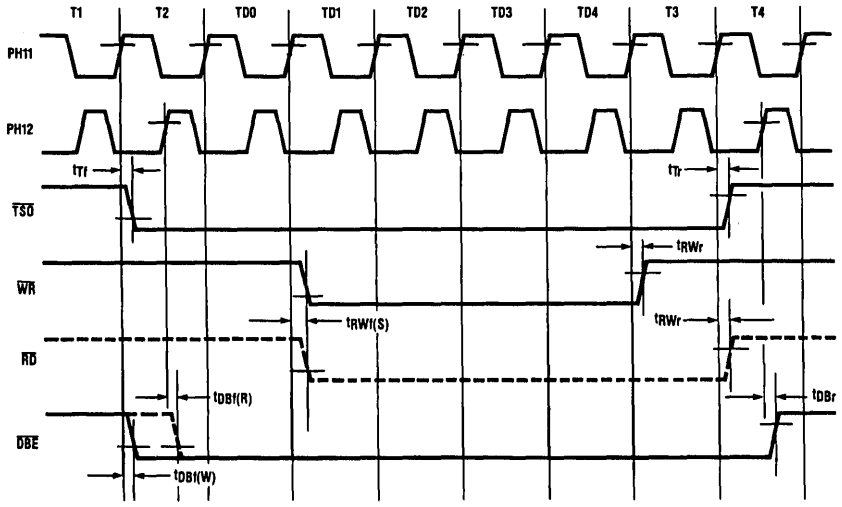
4

2.0 Device Specifications (Continued)



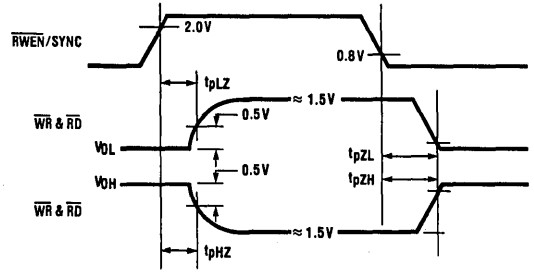
TL/EE/8524-24

FIGURE 2-5. Control Outputs (Fast Cycle)



TL/EE/8524-25

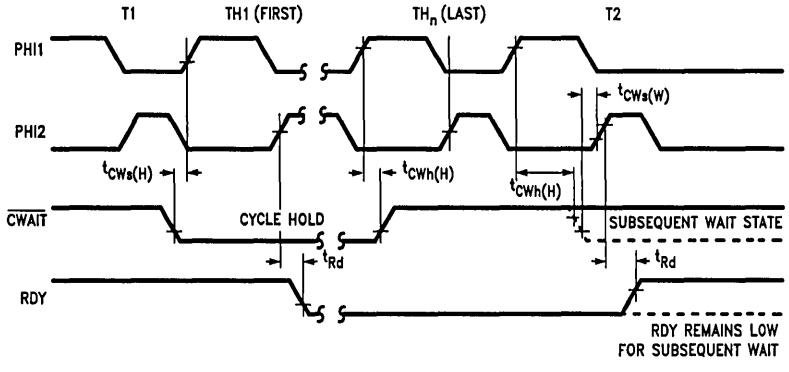
FIGURE 2-6. Control Outputs (Peripheral Cycle)



TL/EE/8524-26

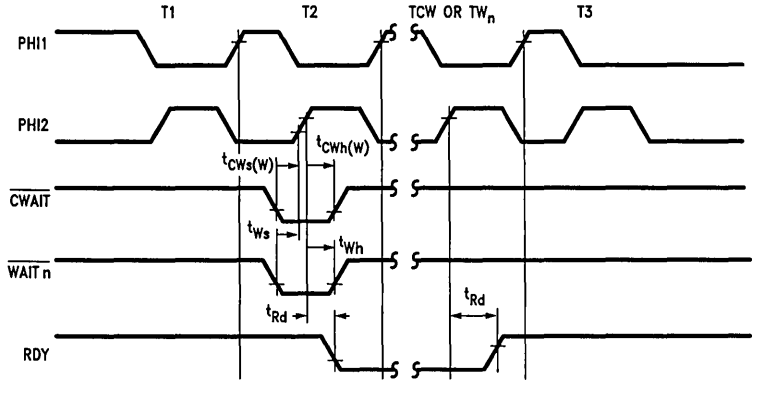
FIGURE 2-7. Control Outputs (TRI-STATE Timing)

2.0 Device Specifications (Continued)



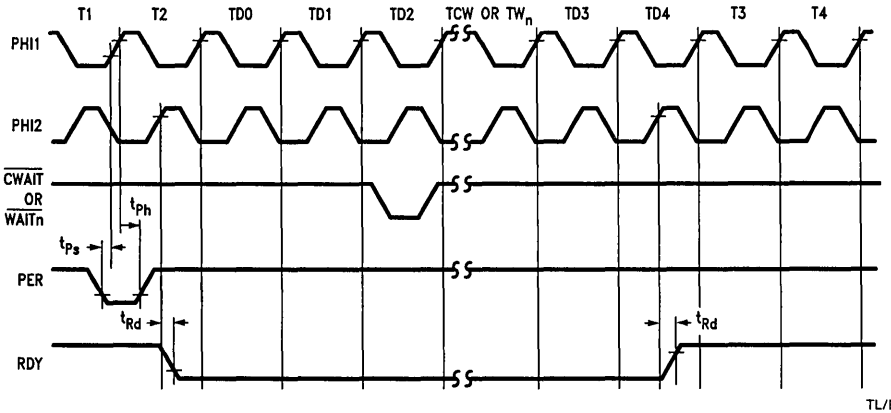
TL/EE/8524-27

FIGURE 2-8. Cycle Hold



TL/EE/8524-28

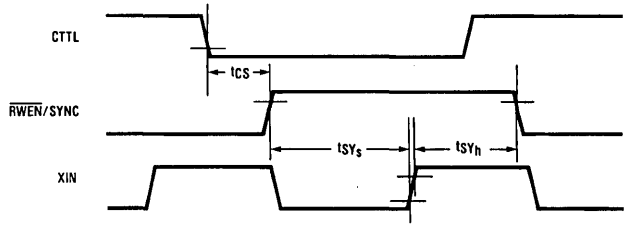
FIGURE 2-9. Wait State (Fast Cycle)



TL/EE/8524-29

FIGURE 2-10. Wait State (Peripheral Cycle)

2.0 Device Specifications (Continued)



TL/EE/8524–30

FIGURE 2-11. Synchronization Timing

National Semiconductor

NS32202-10 Interrupt Control Unit

General Description

The NS32202 Interrupt Control Unit (ICU) is the interrupt controller for the Series 32000® microprocessor family. It is a support circuit that minimizes the software and real-time overhead required to handle multi-level, prioritized interrupts. A single NS32202 manages up to 16 interrupt sources, resolves interrupt priorities, and supplies a single-byte interrupt vector to the CPU.

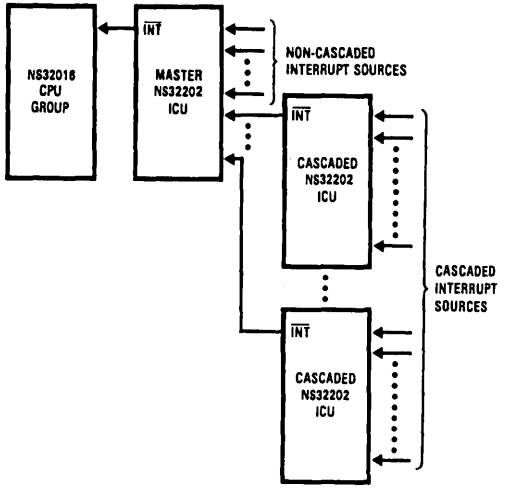
The NS32202 can operate in either of two data bus modes: 16-bit or 8-bit. In the 16-bit mode, eight hardware and eight software interrupt positions are available. In the 8-bit mode, 16 hardware interrupt positions are available, 8 of which can be used as software interrupts. In this mode, up to 16 additional ICUs may be cascaded to handle a maximum of 256 interrupts.

Two 16-bit counters, which may be concatenated under program control into a single 32-bit counter, are also available for real-time applications.

Features

- 16 maskable interrupt sources, cascadable to 256
- Programmable 8- or 16-bit data bus mode
- Edge or level triggering for each hardware interrupt with individually selectable polarities
- 8 software interrupts
- Fixed or rotating priority modes
- Two 16-bit, DC to 10 MHz counters, that may be concatenated into a single 32-bit counter
- Optional 8-bit I/O port available in 8-bit data bus mode
- High-speed XMOS™ technology
- Single, +5V supply
- 40-pin, dual in-line package

Basic System Configuration



TL/EE/5117–1

Table of Contents

List of Illustrations

1.0 Product Introduction

The NS32202 ICU functions as an overall manager in an interrupt-oriented system environment. Its many features and options permit the design of sophisticated interrupt systems.

Figure 1-1 shows the internal organization of the NS32202. As shown, the NS32202 is divided into five functional blocks. These are described in the following paragraphs:

1.1 I/O BUFFERS AND LATCHES

The I/O Buffers and Latches block is the interface with the system data bus. It contains bidirectional buffers for the data I/O pins. It also contains registers and logic circuits that control the operation of pins G0/IR0, . . . ,G7/IR14 when the ICU is in the 8-bit bus mode.

1.2 READ/WRITE LOGIC AND DECODERS

The Read/Write Logic and Decoders manage all internal and external data transfers for the ICU. These include Data, Control, and Status Transfers. This circuit accepts inputs from the CPU address and control buses. In turn, it issues commands to access the internal registers of the ICU.

1.3 TIMING AND CONTROL

The Timing and Control Block contains status elements that select the ICU operating mode. It also contains state machines that generate all the necessary sequencing and control signals.

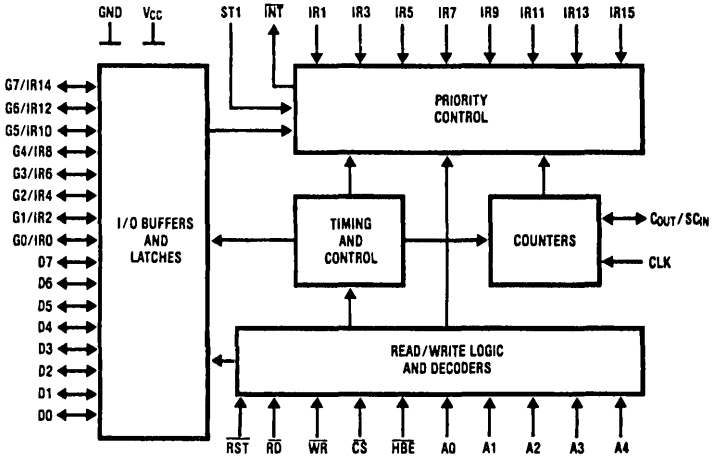
1.4 PRIORITY CONTROL

The Priority Control Block contains 16 units, one for each interrupt position. These units provide the following functions.

- Sensing the various forms of hardware interrupt signals e.g. level (high/low) or edge (rising/falling)
- Resolving priorities and generating an interrupt request to the CPU
- Handling cascaded arrangements
- Enabling software interrupts
- Providing for an automatic return from interrupt
- Enabling the assignment of any interrupt position to the internal counters
- Providing for rearrangement of priorities by assigning the first priority to any interrupt position
- Enabling automatic rotation of priorities

1.5 COUNTERS

This block contains two 16-bit counters, called the H-counter and the L-counter. These are down counters that count from an initial value to zero. Both counters have a 16-bit register (designated HCSV and LCSV) for loading their restarting values. They also have registers containing the current count values (HCCV and LCCV). Both sets of registers are fully described in Section 3.



TL/EE/5117-2

FIGURE 1-1. NS32202 ICU Block Diagram

1.0 Product Introduction (Continued)

The counters are under program control and can be used to generate interrupts. When the count reaches zero, either counter can generate an interrupt request to any of the 16 interrupt positions. The counter then reloads the start value from the appropriate registers and resumes counting. *Figure 1-2* shows typical counter output signals available from the NS32202.

The maximum input clock frequency is 2.5 MHz.

A divide-by-four prescaler is also provided. When the prescaler is used, the input clock frequency can be up to 10 MHz.

When intervals longer than provided by a 16-bit counter are needed, the L- and H-counters can be concatenated to form a 32-bit counter. In this case, both counters are controlled by the H-counter control bits. Refer to the discussion of the Counter Control Register in Section 3 for additional information. *Figure 1-3* summarizes counter read/write operations.

2.0 Functional Description

2.1 RESET

The ICU is reset when a logic low signal is present on the $\overline{RST}$ pin. At reset, most internal ICU registers are affected, and the ICU becomes inactive.

2.2 INITIALIZATION

After reset, the CPU must initialize the NS32202 to establish its configuration. Proper initialization requires knowledge of the ICU register's formats. Therefore, a flowchart of a recommended initialization sequence is shown in (*Figure 3-3*) after the discussion of the ICU registers.

The operation sequence shown in *Figure 3-3* ensures that all counter output pins remain inactive until the counters are completely initialized.

2.3 VECTORED INTERRUPT HANDLING

For details on the operation of the vectored interrupt mode for a particular Series 32000 CPU, refer to the data sheet for

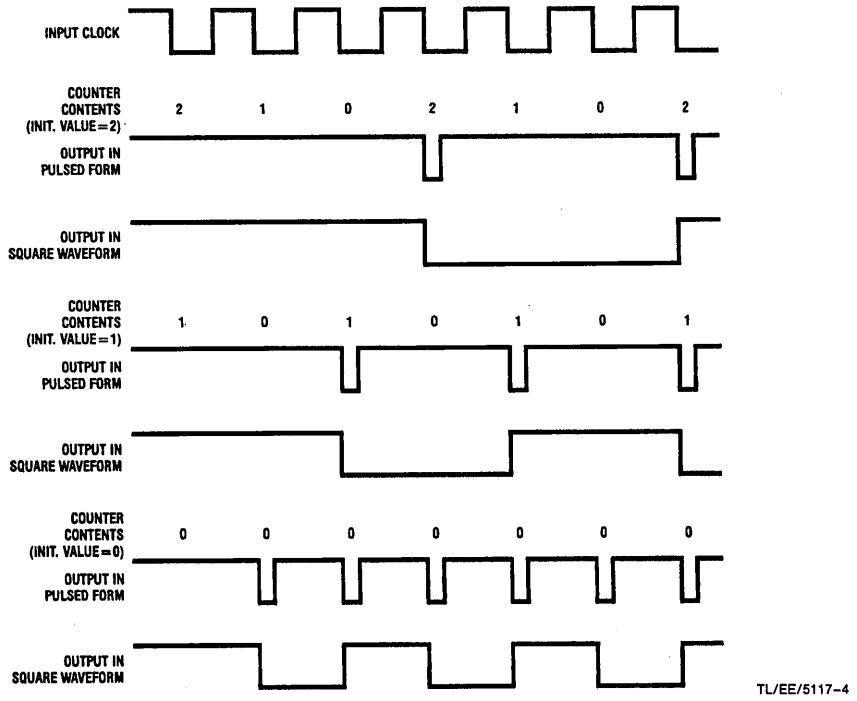
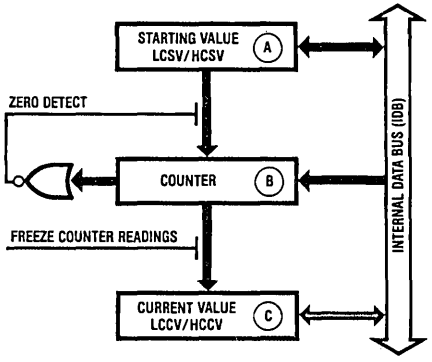


FIGURE 1-2. Counter Output Signals in Pulsed Form and Square Waveform for Three Different Initial Values

2.0 Functional Description (Continued)

that CPU. In this discussion, it is assumed that the NS32202 is working with a CPU in the vectored interrupt mode. Several ICU applications are discussed, including non-cascaded and cascaded operation. *Figures 2–1, 2–2,* and *2–3* show typical configurations of the ICU used with the NS32016 CPU.

A peripheral device issues an interrupt request by sending the proper signal to one of the NS32202 interrupt inputs. If the interrupt input is not masked, the ICU activates its Interrupt Output ($\overline{INT}$) pin and generates an interrupt vector byte. The interrupt vector byte identifies the interrupt source in its four least significant bits. When the CPU detects a low level on its Interrupt Input pin, it performs one or two interrupt acknowledge cycles depending on whether the interrupt request is from the master ICU or a cascaded ICU. *Figure 2–4* shows a flowchart of a typical CPU Interrupt Acknowledge sequence.



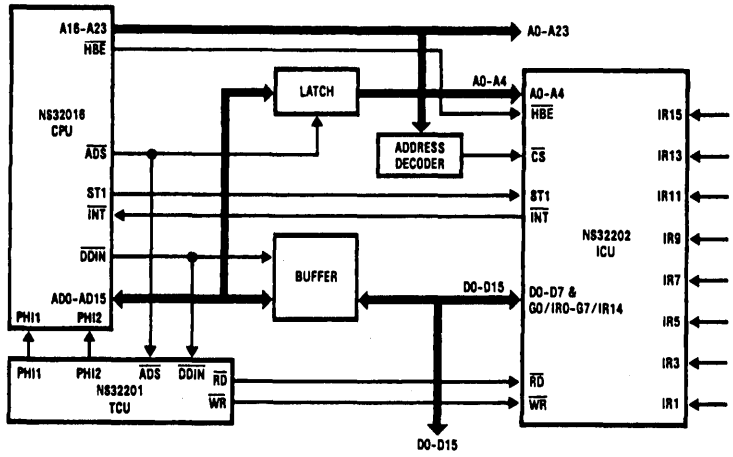
TL/EE/5117–5

BASIC OPERATIONS:

WRITING TO LCSV/HCSV	[A] ← (IDB)
READING LCSV/HCSV	[A] → (IDB)
WRITING TO LCCV/HCCV	[B] ← (IDB)
(only possible when counters are halted)	[C] ← (IDB)
READING LCCV/HCCV	[C] → (IDB)
(only possible when counter readings are frozen)	
COUNTER COUNTS AND READINGS ARE NOT FROZEN	[C] ← [B]
COUNTER RELOADS STARTING VALUE	[B] ← [A]
(occurs on the clock cycle following the one in which it reaches zero)	

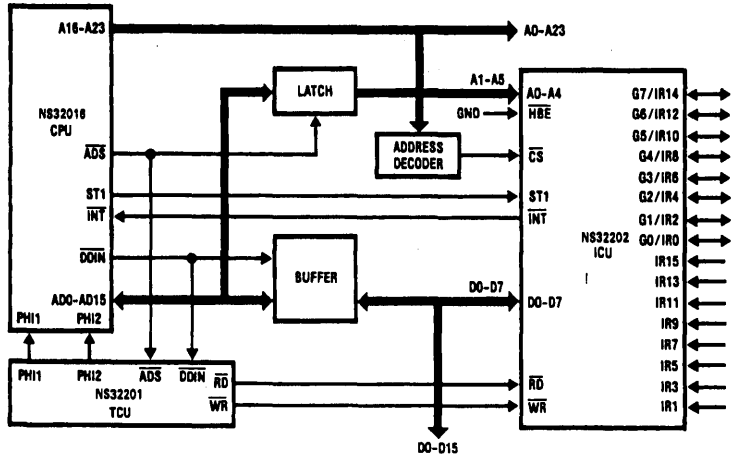
FIGURE 1–3. Counter Configuration and Basic Operations

2.0 Functional Description (Continued)



TL/EE/5117–6

FIGURE 2–1. Interrupt Control Unit Connections In 16-Bit Bus Mode

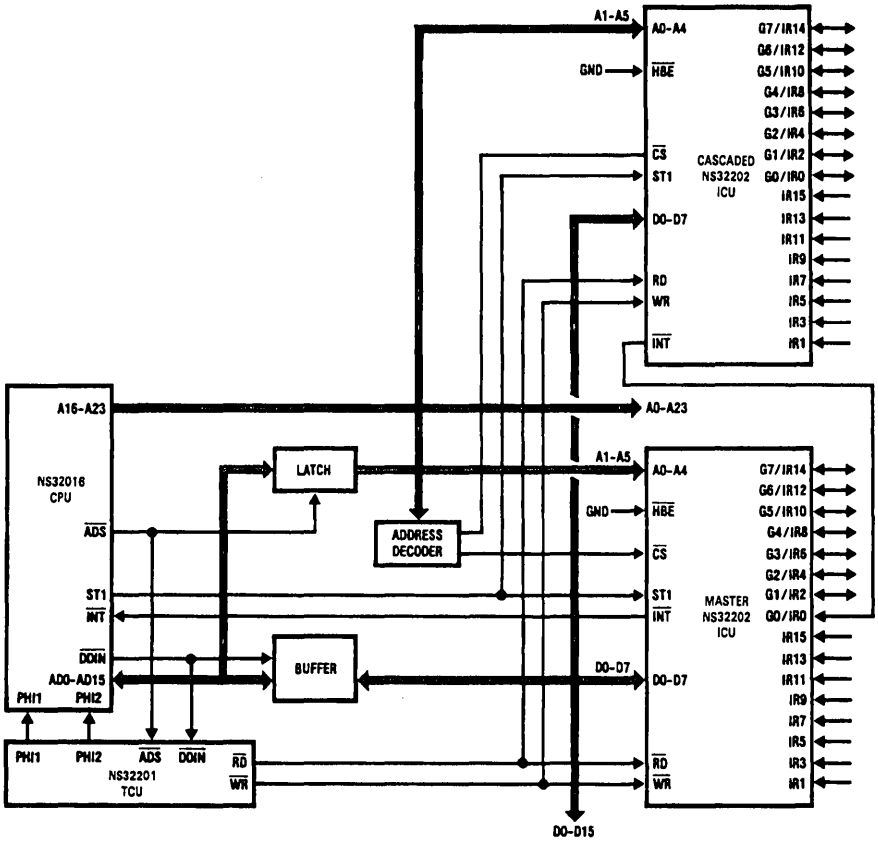


TL/EE/5117–7

NOTE: In the 8-Bit Bus Mode the Master ICU Registers appear at even addresses (A0 = 0) since the ICU communicates with the least significant byte of the CPU data bus.

FIGURE 2–2. Interrupt Control Unit Connections In 8-Bit Bus Mode

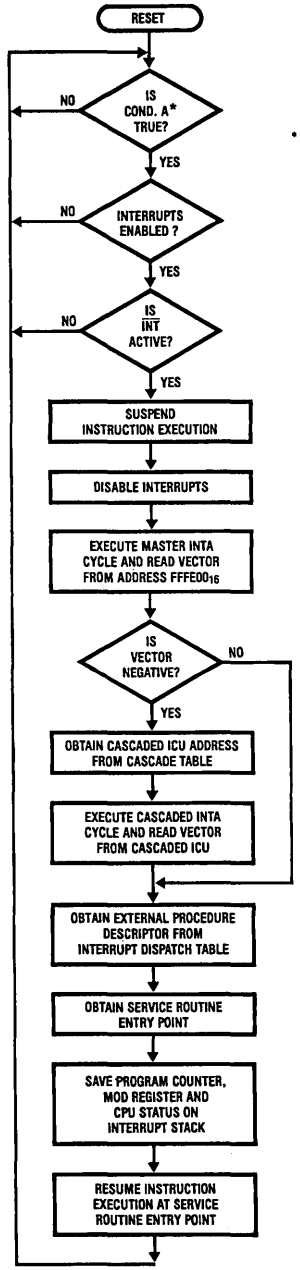
2.0 Functional Description (Continued)



TL/EE/5117-8

FIGURE 2-3. Cascaded Interrupt Control Unit Connections in 8-Bit Bus Mode

2.0 Functional Description (Continued)



* Cond. A is true if current instruction is terminated or an interruptible point in a string instruction is reached.

TL/EE/5117–9

FIGURE 2–4. CPU Interrupt Acknowledge Sequence

2.0 Functional Description (Continued)

In general, vectored interrupts are serviced by interrupt routines stored in system memory. The Dispatch Table stores up to 256 external procedure descriptors for the various service procedures. The CPU INTBASE register points to the top of the Dispatch Table. *Figure 2-5* shows the layout of the Dispatch Table. This figure also shows the layout of the Cascade Table, which is discussed with ICU cascaded operation.

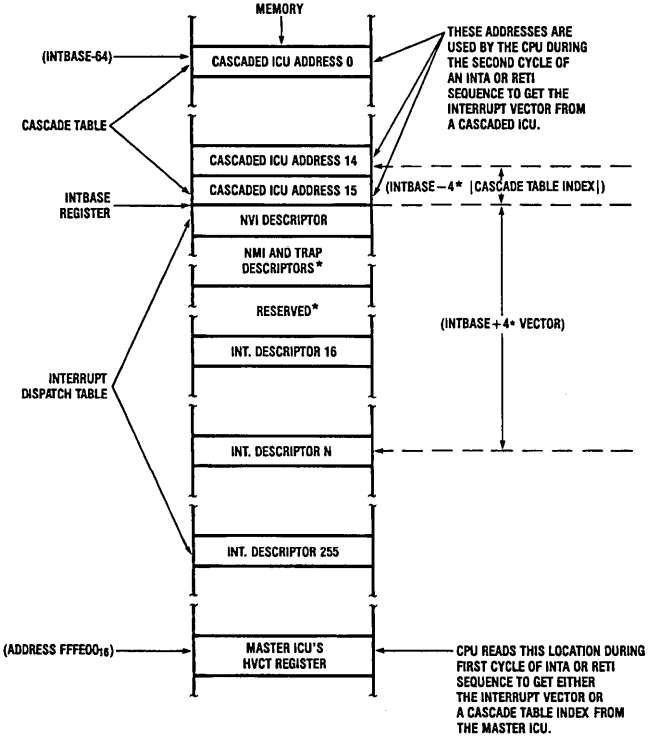
2.3.1 Non-Cascaded Operation. Whenever an interrupt request from a peripheral device is issued directly to the master ICU, a non-cascaded interrupt request to the CPU results. In a system using a single NS32202, up to 16 interrupt requests can be prioritized. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs a Master Interrupt-Acknowledge bus cycle, reading a vector byte from address $FFFE00_{16}$. This vector is then used as an index into the dispatch table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return-from-Interrupt (RET) instruction, which performs a Return-from-Interrupt bus cycle, informing the ICU that it may re-prioritize any interrupt requests still pending. *Figure 2-6* shows a typical CPU RETI sequence. In a system with only one ICU, the vectors provided must be in the range of 0 through 127; this can be ensured by writing 0XXXXXXX into the SVCT register. By providing a negative vector value, the master ICU flags the interrupt source as a cascaded ICU (see below).

2.3.2 Cascaded Operation. In cascaded operation, one or more of the interrupt inputs of the master ICU are connected to the Interrupt Output pin of one or more cascaded ICUs. Up to 16 cascaded ICUs may be used, giving a system total of 256 interrupts.

Note: The number of cascaded ICUs is practically limited to 15 because the Dispatch Table for the NS32016 CPU is constructed with entries 1 through 15 either used for NMI and Trap descriptors, or reserved for future use. Interrupt position 0 of the master ICU should not be cascaded, so it can be vectored through Dispatch Table entry 0, reserved for non-vectored interrupts. In this case, the non-vectored interrupt entry (entry 0) is also available for vectored interrupt operation, since the CPU is operating in the vectored interrupt mode.

The address of the master ICU should be $FFFE00_{16}$. (*) Cascaded ICUs can be located at any system address. A list of cascaded ICU addresses is maintained in the Cascade Table as a series of sixteen 32-bit entries.

(*)Note: The CPU status corresponding to both, master interrupt acknowledge and return from interrupt bus cycles, as well as address bit A8, could be used to generate the chip select ($\overline{CS}$) signal for accessing the master ICU during one of the above cycles. In this case the master ICU can reside at any system address. The only limitation is that the least significant 5 or 6 address bits (6 in the 8-bit bus mode) must be zero. The address bit A8 must be decoded to prevent an NMI bus cycle from reading the hardware vector register of the ICU. This could happen, since the NS32016 CPU performs a dummy read cycle from address $FFFF00_{16}$, with the same status as a master INTA cycle, when a non-maskable-interrupt is acknowledged.

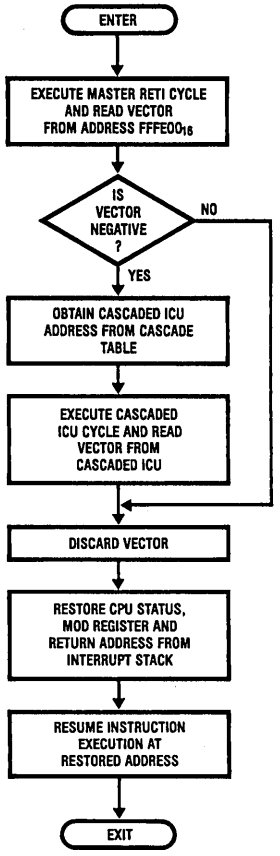


TL/EE/5117-10

* Table entries 1 to 15 should not be used by the ICU since they contain NMI and Trap Descriptors or are reserved for future use. (For more details refer to NS32016 data sheet.)

FIGURE 2-5. Interrupt Dispatch and Cascade Tables

2.0 Functional Description (Continued)



TL/EE/5117–11

FIGURE 2–6. CPU Return from Interrupt Sequence

The master ICU maintains a list (in the CSRC register pair) of its interrupt positions that are cascaded. It also provides a 4-bit (hidden) counter (in-service counter) for each interrupt position to keep track of the number of interrupts being serviced in the cascade ICUs. When a cascaded interrupt input is active, the master ICU activates its interrupt output and the CPU responds with a Master Interrupt Acknowledge Cycle. However, instead of generating a positive interrupt vector, the master ICU generates a negative Cascade Table index.

The CPU interprets the negative number returned from the master ICU as an index into the Cascade Table. The Cascade Table is located in a negative direction from the Dispatch Table, and it contains the virtual addresses of the hardware vector registers for any cascaded NS32202s in the system. Thus, the Cascade Table index supplied by the master ICU identifies the cascaded ICU that requested the interrupt.

Once the cascaded ICU is identified, the CPU performs a Cascaded Interrupt Acknowledge cycle. During this cycle, the CPU reads the final vector value directly from the cascaded ICU, and uses it to access the Dispatch Table. Each cascaded ICU, of course, has its own set of 16 unique interrupt vectors, one vector for each of its 16 interrupt positions.

The CPU interprets the vector value read during a Cascaded Interrupt Acknowledge cycle as an unsigned number. Thus, this vector can be in the range 0 through 255.

When a cascaded interrupt service routine completes its task, it must return control to the interrupted program with the same RETI instruction used in non-cascaded interrupt service routines. However, when the CPU performs a Master Return From Interrupt cycle, the CPU accesses the master ICU and reads the negative Cascade Table index identifying the cascaded ICU that originally received the interrupt request. Using the cascaded ICU address, the CPU now performs a Cascaded Return From Interrupt cycle, informing the cascaded ICU that the service routine is over. The byte provided by the cascaded ICU during this cycle is ignored.

2.4 INTERNAL ICU OPERATING SEQUENCE

The NS32202 ICU accepts two interrupt types, software and hardware.

Software interrupts are initiated when the CPU sets the proper bit in the Interrupt Pending (IPND) registers (R6, R7), located in the ICU. Bits are set and reset by writing the proper byte to either R6 or R7. Software interrupts can be masked, by setting the proper bit in the mask registers (R10, R11).

Hardware interrupts can be either internal or external to the ICU. Internal ICU hardware interrupts are initiated by the on-chip counter outputs. External hardware interrupts are initiated by devices external to the ICU, that are connected to any of the ICU interrupt input pins.

Hardware interrupts can be masked by setting the proper bit in the mask registers (R10, R11). If the Freeze bit (FRZ), located in the Mode Control Register (MCTL), is set, all incoming hardware interrupts are inhibited from setting their corresponding bits in the IPND registers. This prevents the ICU from recognizing any hardware interrupts.

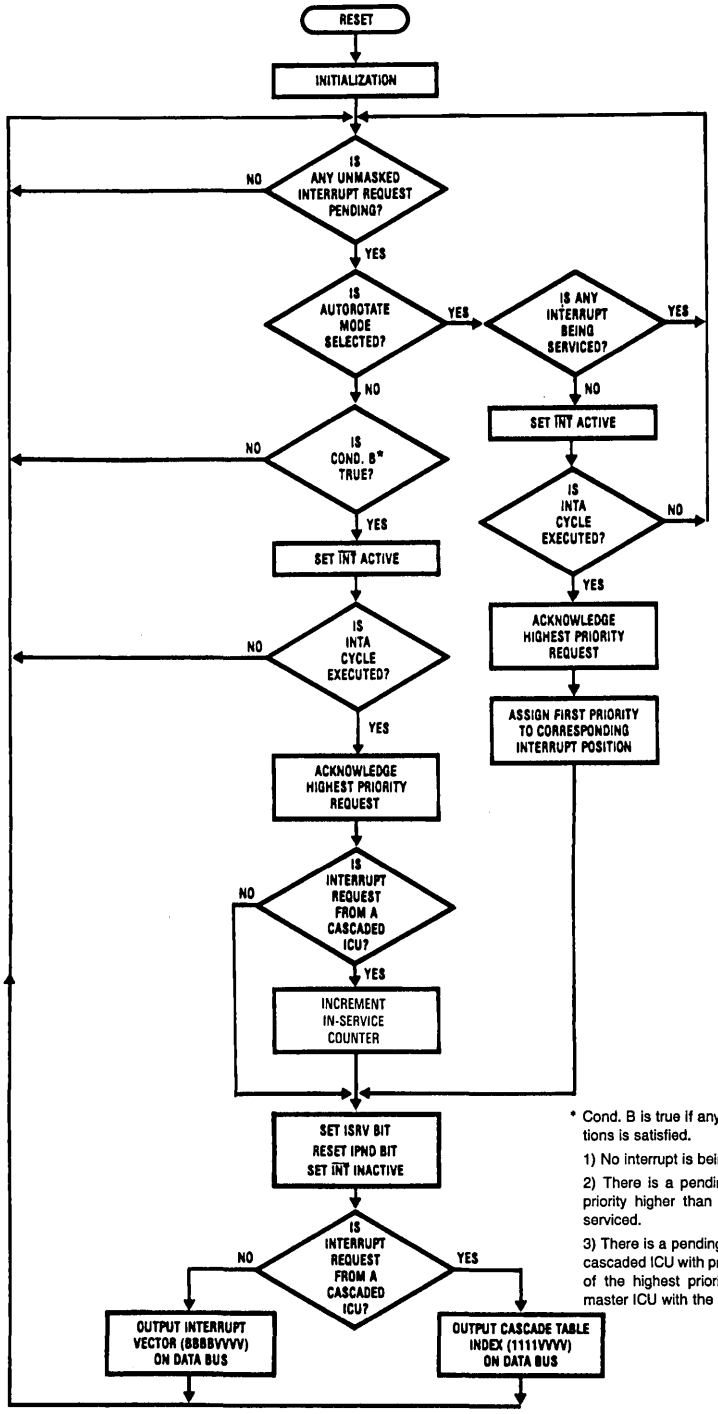
Once the ICU is initialized, it is enabled to accept interrupts. If an active interrupt is not masked, and has a higher priority than any interrupt currently being serviced, the ICU activates its Interrupt Output ($\overline{INT}$). *Figure 2–7* is a flowchart showing the ICU interrupt acknowledge sequence.

The CPU responds to the active $\overline{INT}$ line by performing an Interrupt Acknowledge bus cycle. During this cycle, the ICU clears the IPND bit corresponding to the active interrupt position and sets the corresponding bit in the Interrupt In-Service Registers (ISRV). The 4-bit in-service counter in the master ICU is also incremented by one if the fixed priority mode is selected and the interrupt is from a cascaded ICU. The ISRV bit remains set until the CPU performs a RETI bus cycle and the 4-bit in-service counter is decremented to zero. *Figure 2–8* is a flowchart showing ICU operation during a RETI bus cycle.

When the ISRV bit is set, the $\overline{INT}$ output is disabled. This output remains inactive until a higher priority interrupt position becomes active, or the ISRV bit is cleared.

An exception to the above occurs in the master ICU when the fixed priority mode is selected, and the interrupt input is connected to the $\overline{INT}$ output of a cascaded ICU. In this case the ISRV bit does not inhibit an interrupt of the same priority. This is to allow nesting of interrupts in a cascaded ICU.

2.0 Functional Description (Continued)



* Cond. B is true if any one of the following conditions is satisfied.

1) No interrupt is being serviced

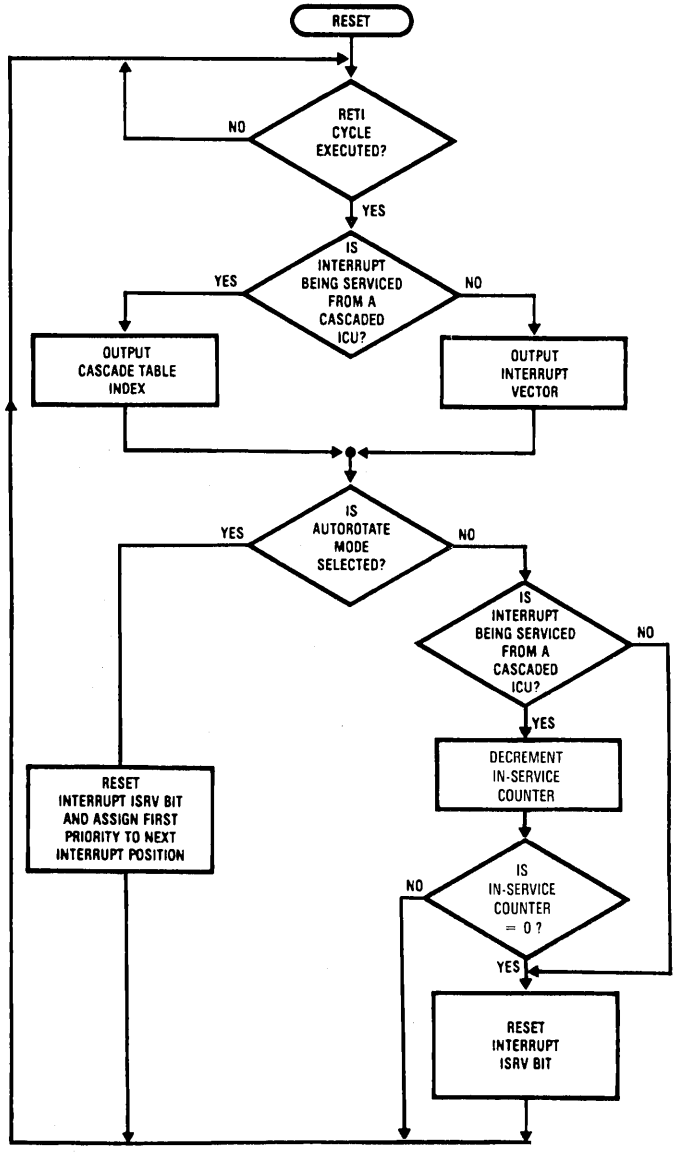
2) There is a pending unmasked interrupt with priority higher than that of the interrupt being serviced.

3) There is a pending unmasked interrupt from a cascaded ICU with priority higher or same as that of the highest priority interrupt position in the master ICU with the ISRV bit set.

TL/EE/5117–12

FIGURE 2–7. ICU Interrupt Acknowledge Sequence

2.0 Functional Description (Continued)



TL/EE/5117–13

FIGURE 2–8. ICU Return from Interrupt Sequence

2.0 Functional Description (Continued)

2.5 INTERRUPT PRIORITY MODES

The NS32202 ICU can operate in one of four interrupt priority modes: Fixed Priority; Auto-Rotate; Special Mask; and Polling. Each mode is described below.

2.5.1 Fixed Priority Mode

In the Fixed Priority Mode (also called Fully Nested Mode), each interrupt position is ranked in priority from 0 to 15, with 0 being the highest priority. In this mode, the processing of lower priority interrupts is nested with higher priority interrupts. Thus, while an interrupt is being serviced, any other interrupts of the same or lower priority are inhibited. The ICU does, however, recognize higher priority interrupt requests.

When the interrupt service routine executes its RETI instruction, the corresponding ISRV bit is cleared. This allows any lower priority interrupt request to be serviced by the CPU.

At reset, the default priority assignment gives interrupt IR0 priority 0 (highest priority), interrupt IR1 priority 1, and so forth. Interrupt IR15 is, of course, assigned priority 15, the lowest priority. The default priority assignment can be altered by writing an appropriate value into register FPRT (L) as explained in Section 3.9.

Note: When the ICU generates an interrupt request to the CPU for a higher priority interrupt while a lower priority interrupt is still being serviced by the CPU, the CPU responds to the interrupt request only if its internal interrupt enable flag is set. Normally, this flag is reset at the beginning of an interrupt acknowledge cycle and set during the RETI cycle. If the CPU is to respond to higher priority interrupts during any interrupt service routine, the service routine must set the internal CPU interrupt enable flag, as soon during the service routine as desired.

2.5.2 Auto-Rotate Mode

The Auto Rotate Mode is selected when the NTAR bit is set to 0, and is automatically entered after Reset. In this mode an interrupt source position is automatically assigned lowest priority after a request at that position has been serviced. Highest priority then passes to the next lower priority position. For example, when servicing of the interrupt request at position 3 is completed (ISRV bit 3 is cleared), interrupt position 3 is assigned lowest priority and position 4 assumes highest priority. The nesting of interrupts is inhibited, since the interrupt being serviced always has the highest priority.

This mode is used when the interrupting devices have to be assigned equal priority. A device requesting an interrupt, will have to wait, in the worst case, until each of the 15 other devices has been serviced at most once.

2.5.3 Special Mask Mode

The Special Mask Mode is used when it is necessary to dynamically alter the ICU priority structure while an interrupt is being serviced. For example, it may be desired in a particular interrupt service routine to enable lower priority interrupts during a part of the routine. To do so, the ICU must be programmed in fixed priority mode and the interrupt service routine must control its own in-service bit in the ISRV registers.

The bits of the ISRV registers are changed with either the Set Bit Interlocked or Clear Bit Interlocked instructions (SBITIW or CBITIW). The in-service bit is cleared to enable lower priority interrupts and set to disable them.

Note: For proper operation of the ICU, an interrupt service routine must set its ISRV bit before executing the RETI instruction. This prevents the RETI cycle from clearing the wrong ISRV bit.

2.5.4 Polling Mode

The Polling Mode gives complete control of interrupt priority to the system software. Either some or all of the interrupt positions can be assigned to the polling mode. To assign all interrupt positions to the polling mode, the CPU interrupt enable flag is reset. To assign only some of the interrupt positions to the polling mode, the desired interrupt positions are masked in the Interrupt Mask registers (IMSK). In either case, the polling operation consists of reading the Interrupt Pending (IPND) registers.

If necessary, the IPND read can be synchronized by setting the Freeze (FRZ) bit in the Mode Control register (MCTL). This prevents any change in the IPND registers during the read. The FRZ bit must be reset after the polling operation so the IPND contents can be updated. If an edge-triggered interrupt occurs while the IPND registers are frozen, the interrupt request is latched, and transferred to the IPND registers as soon as FRZ is reset.

The polling mode is useful when a single routine is used to service several interrupt levels.

3.0 Architectural Description

The NS32202 has thirty-two 8-bit registers that can be accessed either individually or in pairs. In 16-bit data bus mode, register pairs can be accessed with the CPU word or double-word reference instructions. *Figure 3-1* shows the ICU internal registers. This figure summarizes the name, function, and offset address for each register.

Because some registers hold similar data, they are grouped into functional pairs and assigned a single name. However, if a single register in a pair is referenced, either an L or an H is appended to the register name. The letters are placed in parentheses and stand for the low order 8 bits (L) and the high order 8 bits (H). For example, register R6, part of the Interrupt Pending (IPND) register pair, is referred to individually as IPND(L).

The following paragraphs give detailed descriptions of the registers shown in *Figure 3-1*.

3.1 HVCT — HARDWARE VECTOR REGISTER (R0)

The HVCT register is a single register that contains the interrupt vector byte supplied to the CPU during an Interrupt Acknowledge (INTA) or Return From Interrupt (RETI) cycle. The HVCT bit map is shown below:

7	6	5	4	3	2	1	0
B	B	B	B	V	V	V	V

3.0 Architectural Description (Continued)

REG. NUMBER AND ADDRESS IN HEX.		REG. NAME	REG. FUNCTION
	R0 (00_{16})	HVCT —	HARDWARE VECTOR
	R1 (01_{16})	SVCT —	SOFTWARE VECTOR
R3 (03_{16})	R2 (02_{16})	ELTG —	EDGE/LEVEL TRIGGERING
R5 (05_{16})	R4 (04_{16})	TPL —	TRIGGERING POLARITY
R7 (07_{16})	R6 (06_{16})	IPND —	INTERRUPTS PENDING
R9 (09_{16})	R8 (08_{16})	ISRV —	INTERRUPTS IN-SERVICE
R11 ($0B_{16}$)	R10 ($0A_{16}$)	IMSK —	INTERRUPT MASK
R13 ($0D_{16}$)	R12 ($0C_{16}$)	CSRC —	CASCADED SOURCE
R15 ($0F_{16}$)	R14 ($0E_{16}$)	FPRT —	FIRST PRIORITY
	R16 (10_{16})	MCTL —	MODE CONTROL
	R17 (11_{16})	OCASN —	OUTPUT CLOCK ASSIGNMENT
	R18 (12_{16})	CIPTR —	COUNTER INTERRUPT POINTER
	R19 (13_{16})	PDAT —	PORT DATA
	R20 (14_{16})	IPS —	INTERRUPT/PORT SELECT
	R21 (15_{16})	PDIR —	PORT DIRECTION
	R22 (16_{16})	CCTL —	COUNTER CONTROL
	R23 (17_{16})	CICTL —	COUNTER INTERRUPT CONTROL
R25 (19_{16})	R24 (18_{16})	LCSV —	L-COUNTER STARTING VALUE
R27 ($1B_{16}$)	R26 ($1A_{16}$)	HCSV —	H-COUNTER STARTING VALUE
R29 ($1D_{16}$)	R28 ($1C_{16}$)	LCCV —	L-COUNTER CURRENT VALUE
R31 ($1F_{16}$)	R30 ($1E_{16}$)	HCCV —	H-COUNTER CURRENT VALUE

FIGURE 3-1. ICU Internal Registers

3.0 Architectural Description (Continued)

The BBBB field is the bias which is programmed by writing $BBBB0000_2$ to the SVCT register (R1). The VVVV field identifies one of the 16 interrupt positions. The contents of the HVCT register provide various information to the CPU, as shown in *Figure 3-2*.

Note 1: The ICU always interprets a read of the HVCT register as either an INTA or RETI cycle. Since these cycles cause internal changes to the ICU, normal programs must never read the ICU HVCT register.

Note 2: If the HVCT register is read with ST1 = 0 (INTA cycle) and no unmasked interrupt is pending, the binary value BBBB1111 is returned and any pending edge-triggered interrupt in position 15 is cleared.

If the auto-rotate priority mode is selected, the FPRT register is also cleared, thus preventing any interrupt from being acknowledged. In this case a re-intialization of the FPRT register is required for the ICU to acknowledge interrupts again.

If a read of the HVCT register is performed with ST1 = 1 (RETI cycle), the binary value BBBB1111 is returned.

If the auto-rotate mode is selected, a priority rotation is also performed.

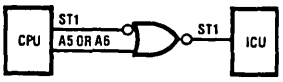
3.2 SVCT — SOFTWARE VECTOR REGISTER (R1)

The SVCT register is a copy of the HVCT register. It allows the programmer to read the contents of the HVCT register without initiating a INTA or RETI cycle in the ICU. It also allows a programmer to change the BBBB field of the HVCT register. The bit map of the SVCT register is the same as for the HVCT register.

During a write to SVCT, the four least significant bits are unaffected while the four most significant bits are written into both SVCT and HVCT (R1 and R0).

The SVCT register is updated dynamically by the ICU. The four least significant bits always contain the vector value that would be returned to the CPU if a INTA or RETI cycle were executed. Therefore, when reading the SVCT register, the state of the CPU ST1 pin is used to select either pending interrupt data or in-service interrupt data. For example, if the SVCT register is read with ST1 = 0 (as for an INTA cycle), the VVVV field contains the encoded value of the highest priority pending interrupt. On the other hand, if the SVCT register is read with ST1 = 1, the VVVV field contains the encoded value of the highest priority in-service interrupt.

Note: If the CPU ST1 output is connected directly to the ICU ST1 input, the vector read from SVCT is always the RETI vector. If both the INTA and RETI vectors are desired, additional logic must be added to drive the ICU ST1 input. A typical circuit is shown below. In this circuit, the state of the ICU ST1 input is controlled by both the CPU ST1 output and the selected address bit.



TL/EE/5117-14

3.3 ELTG — EDGE/LEVEL TRIGGERING REGISTERS (R2, R3)

The ELTG registers determine the input trigger mode for each of the 16 interrupt inputs. Each input is assigned a bit in this register pair. An interrupt input is level-triggered if its bit in ELTG is set to 1. The input is edge-triggered if its bit is cleared. At reset, all bits in ELTG are set to 1.

If odd-numbered interrupt positions must be used for software interrupts, the edge triggering mode must be selected and the corresponding interrupt inputs should be prevented from changing state.

3.4 TPL — TRIGGERING POLARITY REGISTERS (R4, R5)

The TPL registers determine the polarity of either the active level or the active edge for each of the 16 interrupt inputs. As with the ELTG registers, each input is assigned a bit. Possible triggering modes for the various combinations of ELTG and TPL bits are shown below.

ELTG BIT	TPL BIT	TRIGGERING MODE
0	0	Falling Edge
0	1	Rising Edge
1	0	Low Level
1	1	High Level

Software interrupt positions are not affected by their TPL bits. At reset, all TPL bits are set to 0.

Note 1: If edged-triggered interrupts are to be handled, the TPL register should be programmed before the ELTG register.

This prevents spurious interrupt requests from being generated during the ICU initialization from edge-triggered interrupt positions.

Note 2: Hardware interrupt inputs connected to cascaded ICUs must have their TPL bits set to 0.

3.5 IPND — INTERRUPT PENDING REGISTERS (R6, R7)

The IPND registers track interrupt requests that are pending but not yet serviced. Each interrupt position is assigned a bit in IPND. When an interrupt is pending, the corresponding bit in IPND is set. The IPND data are used by the ICU to generate interrupts to the CPU. These data are also used in polling operations.

	INTA CYCLE (ST1 = 0)		RETI CYCLE (ST1 = 1)	
	Highest priority pending interrupt is from:		Highest priority in-service interrupt was from:	
BBBB	cascaded ICU	any other source	cascaded ICU	any other source
	1111	programmed bias*	1111	programmed bias*
VVVV	encoded value of the highest priority pending interrupt		encoded value of the highest priority in-service interrupt	

*The Programmed bias for the master ICU must range from 0000 to 0111_2 because the CPU interprets a one in the most significant bit position as a Cascade Table Index indicator for a cascaded ICU.

FIGURE 3-2. HVCT Register Data Coding

3.0 Architectural Description (Continued)

The IPND registers are also used for requesting software interrupts. This is done by writing specially formatted data bytes to either IPND(L) or IPND(H). The formats differ for registers R6 and R7. These formats are shown below:

IPND(L) (R6) — S0000PPP

IPND(H) (R7) — S0001PPP

Where: S = Set (S = 1) or Clear (S = 0)

PPP = is a binary number identifying one of eight bits

Note: The data read from either R6 or R7 are different from that written to the register because the ICU returns the register contents, rather than the formatted byte used to set the register bits.

The ICU automatically clears a set IPND bit when the pending interrupt request is serviced. All pending interrupts in a register can be cleared by writing the pattern 'X1XXXXXX' to it (X = don't care). To avoid conflicts with asynchronous hardware interrupt requests, the IPND registers should be frozen before pending interrupts are cleared. Refer to the Mode Control Register description for details on freezing the IPND registers.

At reset, all IPND bits are set to 0.

Note: The edge sensing mechanism used for hardware interrupts in the NS32202 ICU is a latching device that can be cleared only by acknowledging the interrupt or by changing the trigger mode to level sensing. Therefore, before clearing pending interrupts in the IPND registers, any edge-triggered interrupt inputs must first be switched to the level-triggered mode. This clears the edge-triggered interrupts; the remaining interrupts can then be cleared in the manner described above. This applies to clearing the interrupts only. Edge-triggered interrupts can be set without changing the trigger mode.

3.6 ISRV — INTERRUPT IN-SERVICE REGISTERS (R8, R9)

The ISRV registers track interrupt requests that are currently being serviced. Each interrupt position is assigned a bit in ISRV. When an interrupt request is serviced by the ICU, its corresponding bit is set in the ISRV registers. Before generating an interrupt to the CPU, the ICU checks the ISRV registers to ensure that no higher priority interrupt is currently being serviced.

Each time the CPU executes an RETI instruction, the ICU clears the ISRV bit corresponding to the highest priority interrupt in service. The ISRV registers can also be written into by the CPU. This is done to implement the special mask priority mode.

At reset, the ISRV registers are set to 0.

Note: If the ICU initialization does not follow a hardware reset, the ISRV register should be cleared during initialization by writing zeroes into it.

3.7 IMSK — INTERRUPT MASK REGISTERS (R10, R11)

Each NS32202 interrupt position can be individually masked. A masked interrupt source is not acknowledged by the ICU. The IMSK registers store a mask bit for each of the ICU interrupt positions. If an interrupt position's IMSK bit is set to 1, the position is masked.

The IMSK registers are controlled by the system software. At reset, all IMSK bits are set to 1, disabling all interrupts.

Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the IMSK register. However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the $\overline{INT}$ line before the ICU deasserted it. This could cause the ICU to provide an invalid vector. To avoid this problem, the above operation should be performed with the CPU interrupt disabled.

3.8 CSRC — CASCADED SOURCE REGISTERS (R12, R13)

The CSRC registers track any cascaded interrupt positions. Each interrupt position is assigned a bit in the CSRC registers. If an interrupt position's CSRC bit is set, that position is connected to the $\overline{INT}$ output of another NS32202 ICU, i.e., it is a cascaded interrupt.

At reset, the CSRC registers are set to 0.

Note 1: If any cascaded ICU is used, the CSRC register should be cleared during initialization (if the initialization does not follow a hardware reset) by writing zeroes into it. This should be done before setting the bits corresponding to the cascaded interrupt positions. This operation ensures that the 4-bit in-service counters (associated with each interrupt position to keep track of cascaded interrupts) always get cleared when the ICU is re-initialized.

Note 2: Only the Master ICU should have any CSRC bits set. If CSRC bits are set in a cascaded ICU, incorrect operation results.

3.9 FPRT — FIRST PRIORITY REGISTERS (R14, R15)

The FPRT registers track the ICU interrupt position that currently holds first priority. Only one bit of the FPRT registers is set at one time. The set bit indicates the interrupt position with first (highest) priority.

The FPRT registers are automatically updated when the ICU is in the auto-rotate mode. The first priority interrupt can be determined by reading the FPRT registers. This operation returns a 16-bit word with only one bit set. An interrupt position can be assigned first priority by writing a formatted data byte to the FPRT(L) register. The format is shown below:

7	6	5	4	3	2	1	0
X	X	X	X	F	F	F	F

Where: XXXX = Don't Care

FFFF = A binary number from 0 to 15 indicating the interrupt position assigned first priority.

Note: The byte above is written only to the FPRT(L) register. Any data written to FPRT(H) is ignored.

At reset the FFFF field is set to 0, thus giving interrupt position 0 first priority.

3.10 MCTL — MODE CONTROL REGISTER (R16)

The contents of the MCTL set the operating mode of the NS32202 ICU. The MCTL bit map is shown below.

7	6	5	4	3	2	1	0
CFRZ	COUTD	COUTM	CLKM	FRZ	unused	NTAR	T16N8

3.0 Architectural Description (Continued)

CFRZ Determines whether or not the NS32202 counter readings are frozen. When frozen, the counters continue counting but the LCCV and HCCV registers are not updated. Reading of the true value of LCCV and HCCV is possible only while they are frozen.

CFRZ = 0 => LCCV and HCCV Not Frozen

CFRZ = 1 => LCCV and HCCV Frozen

COUTD Determines whether the COUT/SCIN pin is an input or an output. COUT/SCIN should be used as an input only for testing purposes. In this case an external sampling clock must be provided otherwise hardware interrupts will not be recognized.

COUTD = 0 => COUT/SCIN is Output

COUTD = 1 => COUT/SCIN is Input

COUTM When the COUT/SCIN pin is programmed as an output (COUTD=0), this bit determines whether the output signal is in pulsed form or in square wave form.

COUTM = 0 => Square Wave Form

COUTM = 1 => Pulsed Form

CLKM Used only in the 8-bit Bus Mode. This bit controls the clock wave form on any of the pins G0/IR0, . . . ,G3/IR6 programmed as counter output.

CLKM = 0 => Square Wave Form

CLKM = 1 => Pulsed Form

FRZ Freeze Bit. In order to allow a synchronous reading of the interrupt pending registers (IPND), their status may be frozen, causing the ICU to ignore incoming requests. This is of special importance if a polling method is used.

FRZ = 0 => IPND Not Frozen

FRZ = 1 => IPND Frozen

NTAR Determines whether the ICU is in the AUTO-ROTATE or FIXED Priority Mode. In AUTO-ROTATE mode, the interrupt source at the highest priority position, after being serviced, is assigned automatically lowest priority. In this mode, the interrupt in service always has highest priority and nesting of interrupts is therefore inhibited.

NTAR = 0 => Auto-Rotate Mode

NTAR = 1 => Fixed Mode

T16N8 Controls the data bus mode of operation.

T16N8 = 0 => 8-Bit Bus Mode

T16N8 = 1 => 16-Bit Bus Mode

At reset, all MCTL bits except COUTD, are reset to 0. COUTD is set to 1.

3.11 OCASN — OUTPUT CLOCK ASSIGNMENT REGISTER (R17)

Used only in the 8-bit Bus Mode. The four least significant bits of this register control the output clock assignments on pins G0/IR0, . . . ,G3/IR6. If any of these bits is set to 1, the clock generated by either the H-Counter or the H+L-Counter will be output to the corresponding pin. The four most significant bits of OCASN are not used. At Reset the four least significant bits are set to 0.

Note: The interrupt sensing mechanism on pins G0/IR0, . . . ,G3/IR6 is not disabled when any of these pins is programmed as clock output. Thus, to avoid spurious interrupts, the corresponding bits in register IPS should also be set to zero.

3.12 CIPTR — COUNTER INTERRUPT POINTER REGISTER (R18)

The CIPTR register tracks the assignment of counter outputs to interrupt positions. A bit map of this register is shown below.

7	6	5	4	3	2	1	0
H	H	H	H	L	L	L	L

Where: HHHH = A 4-bit binary number identifying the interrupt position assigned to the H-Counter (or the H+L-counter if the counters are concatenated).

LLLL = A 4-bit binary number identifying the interrupt position assigned to the L-counter.

Note: Assignment of a counter output to an interrupt position also requires control bits to be set in the CICTL register. If a counter output is assigned to an interrupt position, external hardware interrupts at that position are ignored.

At reset, all bits in the CIPTR are set to 1. (This means both counters are assigned to interrupt position 15.)

3.13 PDAT — PORT DATA REGISTER (R19)

Used only in the 8-bit Bus Mode. This register is used to input or output data through any of the pins G0/IR0, . . . ,G7/IR14 programmed as I/O ports by the IPS register. Any pin programmed as an output delivers the data written into PDAT. The input pins ignore it. Reading PDAT provides the logical value of all I/O pins, INPUT and OUTPUT.

3.14 IPS — INTERRUPT/PORT SELECT REGISTER (R20)

Used only in the 8-bit Bus Mode. This register controls the function of the pins G0/IR0, . . . ,G7/IR14. Each of these pins is individually programmed as an I/O port, if the corresponding bit of IPS is 0; as an interrupt source, if the corresponding bit is 1. The assignment of the H-Counter output to G0/IR0, . . . ,G3/IR6 by means of reg. OCASN overrides the assignment to these pins as I/O ports or interrupt inputs.

At Reset, all the IPS bits are set to 1.

Note: Whenever a bit in the IPS register is set to zero, to program the corresponding pin as an I/O port, any pending interrupt on the corresponding interrupt position will be cleared.

3.15 PDIR — PORT DIRECTION REGISTER (R21)

Used only in the 8-bit Bus Mode. This register determines the direction of any of the pins G0/IR0, . . . ,G7/IR14 programmed as I/O ports by the IPS register. A logic 1 indicates an input, while a logic 0 indicates an output.

At Reset, all the PDIR bits are set to 1.

3.16 CCTL — COUNTER CONTROL REGISTER (R22)

The CCTL register controls the operating modes of the counters. A bit map of CCTL is shown below.

7	6	5	4	3	2	1	0
CCON	CFNPS	COUT1	COUT0	CRUNH	CRUNL	CDCRH	CDCRL

CCON Determines whether the counters are independent or concatenated to form a single 32-bit counter (H+L-Counter). If a 32-bit counter is selected, the bits corresponding to the H-

3.0 Architectural Description (Continued)

Counter will control the H+L-Counter, while the bits corresponding to the L-Counter are not used.

CCON = 0 => Two 16-bit Counters

CCON = 1 => One 32-bit Counter

CFNPS — Determines whether the external clock is prescaled or not.

CFNPS = 0 => Clock Prescaled (divided by 4)

CFNPS = 1 => Clock Not Prescaled.

COUT1 & COUT0 — These bits are effective only when the COUT/SCIN pin is programmed as an OUTPUT (COUTD bit in reg. MCTL is 0). Their logic levels are decoded to provide different outputs for COUT/SCIN, as detailed in the table below:

COUT1	COUT0	COUT/SCIN Output Signal
0	0	Internal Sampling Oscillator
0	1	Zero Detect Of L-Counter
1	0	Zero Detect Of H-Counter
1	1	Zero Detect Of H+L-Counter*

*If the H- and L-Counters are not concatenated and COUT1/COUT0 are both 1, the COUT/SCIN pin is active when either counter reaches zero.

CRUNH — Determines the state of either the H-Counter or the H+L-Counter, depending upon the status of CCON.

CRUNH = 0 => H-Counter or H+L-Counter Halted

CRUNH = 1 => H-Counter or H+L-Counter Running

CRUNL — Effective only when CCON = 0. This bit determines whether the L-Counter is running or halted.

CRUNL = 0 => L-Counter Halted

CRUNL = 1 => L-counter Running

CDCRH — Effective only when CRUNH = 0 (Counter Halted). This bit is the single cycle decrement signal for either the H-Counter or the H+L-Counter.

CDCRH = 0 => No Effect

CDCRH = 1 => Decrement H-Counter or H+L-Counter

CDCRL — Effective only when CRUNL = 0 and CCON = 0. This bit is the single cycle decrement signal for the L-Counter.

CDCRL = 0 => No Effect

CDCRL = 1 => Decrement L-Counter

Note: The bits CDCRL and CDCRH are set when a logic 1 is written into them, but, they are automatically cleared after the end of the write operation. This is needed to accomplish the decrement operation. Therefore, these bits always contain 0 when read.

Reset does not affect the CCTL bits.

3.17 CICTL — COUNTER INTERRUPT CONTROL REGISTER (R23)

The CICTL register controls the counter interrupts and records counter interrupt status. Interrupts can be generated from either of the 16-bit counters. When the counters are concatenated, the interrupt control is through the H-Counter control bits. In this case the CIEL bit should be set to zero to avoid spurious interrupts from the L-Counter. A bit map of the CICTL register is shown following.

7	6	5	4	3	2	1	0
CERH	CIRH	CIEH	WENH	CERL	CIRL	CIEL	WENL

CERH — H-Counter Error Flag. This bit is set (1) when a second interrupt request from the H-Counter (or H+L-Counter) occurs before the first request is acknowledged.

CIRH — H-Counter Interrupt Request. It is set (1) when an interrupt is pending from the H-Counter (or H+L-Counter). It is automatically reset when the interrupt is acknowledged.

CIEH — H-Counter Interrupt Enable. When it is set, the H-Counter (or H+L-Counter) interrupt is enabled.

WENH — H-Counter Control Write Enable. When WEHN is set (1), bits CERH, CIRH, and CIEH can be written.

CERL — L-Counter Error Flag. This bit is set (1) when a second interrupt request from the L-Counter occurs before the first request is acknowledged.

CIRL — L-Counter Interrupt Request. It is set (1) when an interrupt is pending from the L-Counter. It is automatically reset when the interrupt is acknowledged.

CIEL — L-Counter Interrupt Enable. When it is set (1), the L-Counter interrupt is enabled.

WENL — L-Counter Control Write Enable. When WENL is set (1), bits CERL, CIRL, and CIEL can be written.

Note: Setting the write enable bits (WENH or WENL) and writing any of the other CICTL bits are concurrent operations. That is, the ICU will ignore any attempt to alter CICTL bits if the proper write enable bit is not set in the data byte.

At reset, all CICTL bits are set to 0. However, if the counters are running, the bits CIRL, CERL, CIRH and CERH may be set again after the reset signal is removed.

3.18 LCSV/HCSV — L-COUNTER STARTING VALUE/ H-COUNTER STARTING VALUE REGISTERS (R24, R25, R26, AND R27)

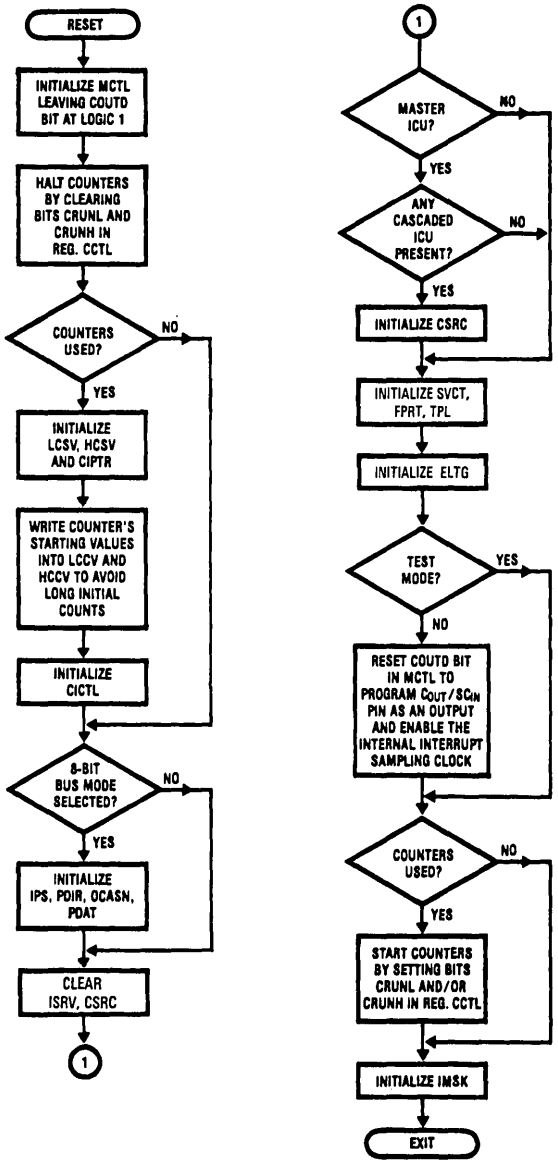
The LCSV and HCSV registers store the start values for the L-Counter and H-Counter, respectively. Each time a counter reaches zero, the start value is automatically reloaded from either LCSV or HCSV, one clock cycle after zero count is reached. Loading LCSV or HCSV from the CPU must be synchronized to avoid writing the registers while the reloading of the counters is occurring. One method is to halt the counters while the registers are loaded.

When the 16-bit counters are concatenated, the LCSV and HCSV registers hold the 32-bit start count, with the least significant byte in R24 and the most significant byte in R27.

3.19 LCCV/HCCV — L-COUNTER CURRENT VALUE/ H-COUNTER CURRENT VALUE REGISTERS (R28, R29, R30, AND R31)

The LCCV and HCCV registers hold the current value of the counters. If the CFRZ bit in the MCTL register is reset (0), these registers are updated on each clock cycle with the current value of the counters. LCCV and HCCV can be read only when the counter readings are frozen (CFRZ bit in the

3.0 Architectural Description (Continued)



TL/EE/5117–15

FIGURE 3–3. Recommended ICU's Initialization Sequence

4

3.0 Architectural Description (Continued)

MCTL register is 1). They can be written only when the counters are halted (CRUNL and/or CRUNH bits in the CCTL register are 0). This last feature allows new initial count values to be loaded immediately into the counters, and can be used during initialization to avoid long initial counts.

When the 16-bit counters are concatenated, the LCCV and HCCV registers hold the 32-bit current value, with the least significant byte in R28 and the most significant byte in R31.

3.20 REGISTER INITIALIZATION

Figure 3-3 shows a recommended initialization procedure for the ICU that sets up all the ICU registers for proper operation.

4.0 Device Specifications

4.1 NS32202 PIN DESCRIPTIONS

4.1.1 Power Supply

Power (V_{CC}): +5V DC Supply

Ground (GND): Power Supply Return

4.1.2 Input Signals

Reset ($\overline{RST}$): Active low. This signal initializes the ICU. (The ICU initializes to the 8-bit bus mode.)

Chip Select ($\overline{CS}$): Active low. This signal enables the ICU to respond to address, data, and control signals from the CPU.

Addresses (A0 through A4): Address lines used to select the ICU internal registers for read/write operations.

High Byte Enable ($\overline{HBE}$): Active low. Enables data transfers on the most-significant byte of the Data Bus. If the ICU is in the 8-bit Bus Mode, this signal is not used and should be connected to either GND or V_{CC}.

Read ($\overline{RD}$): Active low. Enables data to be read from the ICU's internal registers.

Write ($\overline{WR}$): Active low. Enables data to be written into the ICU's internal registers.

Status (ST1): Status signal from the CPU. When the Hardware Vector Register is read, this signal differentiates an INTA cycle from an RETI cycle. If ST1 = 0 the ICU initiates an INTA cycle. If ST1 = 1 an RETI cycle will result.

Interrupt Requests (IR1, IR3..., IR15): These eight inputs are used for hardware interrupts. Each may be individually triggered in one of four modes: Rising Edge, Falling Edge, Low Level, or High Level.

Counter Clock (CLK): External clock signal to drive the ICU internal counters.

4.1.3 Output Signals

Interrupt Output ($\overline{INT}$): Active low. This signal indicates that an interrupt is pending.

4.1.4 Input/Output Signals

Data Bus 0-7 (D0 through D7): Eight low-order data bus lines used in both 8-bit and 16-bit bus modes.

General Purpose I/O Lines (G0/IR0, G1/IR2,...,G7/IR14): These pins are the high-order data bits when the ICU is in the 16-bit bus mode. When the ICU is in the 8-bit bus mode, each of these can be individually assigned one of the following functions:

- Additional Hardware Interrupt Input (IR0 through IR14)
- General Purpose Data Input
- General Purpose Data Output
- Clock Output from H-Counter (Pins G0/IR0 through G3/IR6 only)

It should be noted that, for maximum flexibility in assigning interrupt priorities, the interrupt positions corresponding to pins G0/IR0,...,G7/IR14 and IR1,...,IR15 are interleaved.

Counter or Oscillator Output/Sampling Clock Input (COUT/SCIN): As an output, this pin provides either a clock signal generated by the ICU internal oscillator, or a zero detect signal from one or both of the ICU counters. As an input, it is used for an external clock, to override the internal oscillator used for interrupt sampling. This is done only for testing purposes.

4.0 Device Specifications (Continued)

4.2 ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7.0V
Power Dissipation	1.5 Watt

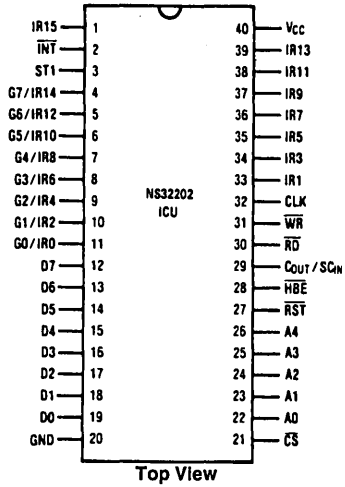
Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS

T_A = 0° to 70°C, V_{CC} = +5V ± 5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IL}	Input Low Voltage				0.8	V
V_{IH}	Input High Voltage		2.0			V
V_{OL}	Output Low Voltage	I_{OL} = 2 mA			0.45	V
V_{OH}	Output High Voltage	I_{OH} = −400 μA	2.4			V
I_L	Leakage Current (Output and I/O Pins in TRI-STATE/Input mode)	0.4 ≤ V_{IN} ≤ V_{CC}	−20		20	μA
I_I	Input Load Current	V_{in} = 0 to V_{CC}	−20		20	μA
I_{CC}	Power Supply Current	I_{out} = 0, T = 0°C			300	mA

Connection Diagram



TL/EE/5117-3

Top View

Order Number NS32202D-6, NS32202D-10
See NS Package Number D40C

FIGURE 4-1

4.0 Device Specifications (Continued)

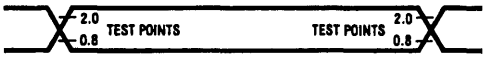
4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 0.8V or 2.0V on the input and output signals as illustrated in *Figure 1*, unless specifically stated otherwise.

Abbreviations:

L.E.—leading edge R.E.—rising edge
T.E.—trailing edge F.E.—falling edge



TL/EE/5117–16

FIGURE 4–2. Timing Specification Standard

4.4.1.1 Timing Tables

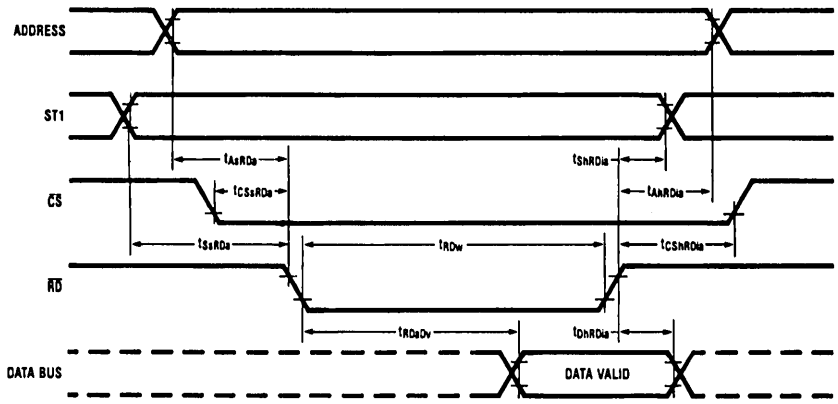
Symbol	Figure	Description	Reference/Conditions	NS32202-10		Units
				Min	Max	
READ CYCLE						
t_{AhRDia}	4-3	Address Hold Time	After $\overline{RD}$ T.E.	10		ns
t_{AsRDa}	4-3	Address Setup Time	Before $\overline{RD}$ L.E.	35		ns
$t_{CShRDia}$	4-3	$\overline{CS}$ Hold Time	After $\overline{RD}$ T.E.	15		ns
t_{CSsRDa}	4-3	$\overline{CS}$ Setup Time	Before $\overline{RD}$ L.E.	30		ns
t_{DhRDia}	4-3	Data Hold Time	After $\overline{RD}$ T.E.	5	50	ns
t_{RDaDv}	4-3	Data Valid	After $\overline{RD}$ L.E.		150	ns
t_{RDw}	4-3	$\overline{RD}$ Pulse Width	At 0.8V (Both Edges)	160		ns
t_{SsRDa}	4-3	ST1 Setup Time	Before $\overline{RD}$ L.E.	35		ns
t_{ShRDia}	4-3	ST1 Hold Time	After $\overline{RD}$ T.E.	−30		ns
WRITE CYCLE						
t_{AhWRia}	4-4	Address Hold Time	After $\overline{WR}$ T.E.	10		ns
t_{AsWRa}	4-4	Address Setup Time	Before $\overline{WR}$ L.E.	35		ns
$t_{CShWRia}$	4-4	$\overline{CS}$ Hold Time	After $\overline{WR}$ T.E.	15		ns
t_{CSsWRa}	4-4	$\overline{CS}$ Setup Time	Before $\overline{WR}$ L.E.	30		ns
t_{DhWRia}	4-4	Data Hold Time	After $\overline{WR}$ T.E.	10		ns
t_{DsWRia}	4-4	Data Setup Time	Before $\overline{WR}$ T.E.	70		ns
t_{WRiaPf}	4-4	Port Output Floating	After $\overline{WR}$ T.E. (To PDIR)		200	ns
t_{WRiaPv}	4-4	Port Output Valid	After $\overline{WR}$ T.E.		200	ns
t_{WRw}	4-4	$\overline{WR}$ Pulse Width	At 0.8V (Both Edges)	160		ns

4.0 Device Specifications (Continued)

4.4.1.1 Timing Tables (Continued)

Symbol	Figure	Description	Reference/Conditions	NS32202-10		Units
				Min	Max	
OTHER TIMINGS						
t_{COUTl}	4-8	Internal Sampling Clock Low Time	At 0.8V (Both Edges)	50		ns
t_{COUTp}	4-8	Internal Sampling Clock Period		400		ns
t_{SCINh}	4-7	External Sampling Clock High Time	At 2.0V (Both Edges)	100		ns
t_{SCINl}	4-7	External Sampling Clock Low Time	At 0.8V (Both Edges)	100		ns
t_{SCINp}	4-7	External Sampling Clock Period		800		ns
t_{Ch}	4-9	External Clock High Time (Without Prescaler)	At 2.0V (Both Edges)	100		ns
t_{Chp}	4-9	External Clock High Time (With Prescaler)	At 2.0V (Both Edges)	40		ns
t_{Cl}	4-9	External Clock Low Time (Without Prescaler)	At 0.8V (Both Edges)	100		ns
t_{Clp}	4-9	External Clock Low Time (With Prescaler)	At 0.8V (Both Edges)	40		ns
t_{Cy}	4-9	External Clock Period (Without Prescaler)		400		ns
t_{Cyp}	4-9	External Clock Period (With Prescaler)		100		ns
t_{GCOUTl}	4-9	Counter Output Transition Delay	After CLK F.E.		300	ns
t_{COUTw}	4-9	Counter Output Pulse Width in Pulsed Form	At 0.8V (Both Edges)	50		ns
t_{ACKIR}	4-5	Interrupt Request Delay	After Previous Interrupt Acknowledge	500		ns
t_{IRId}	4-5	$\overline{INT}$ Output Delay	After Interrupt Request Active		800	ns
t_{IRw}	4-5	Interrupt Request Pulse Width in Edge Trigger	At 0.8V (Both Edges)	50		ns
t_{RSTw}		$\overline{RST}$ Pulse Width	At 0.8V (Both Edges)	400		ns

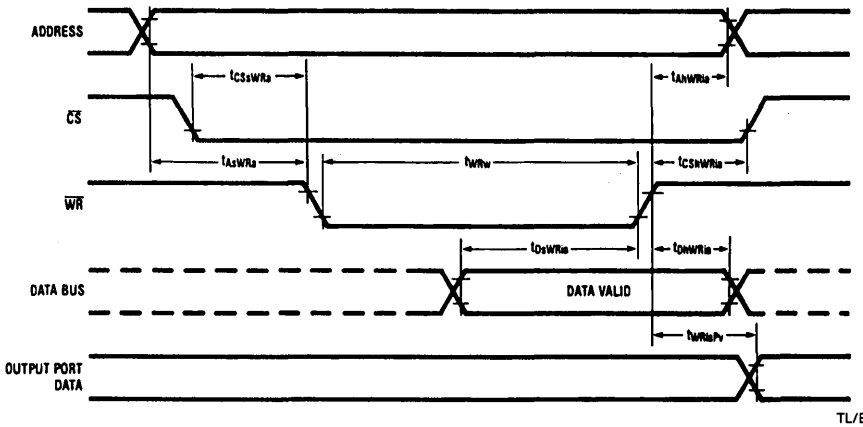
4.4.1.2 Timing Diagrams



TL/EE/5117-17

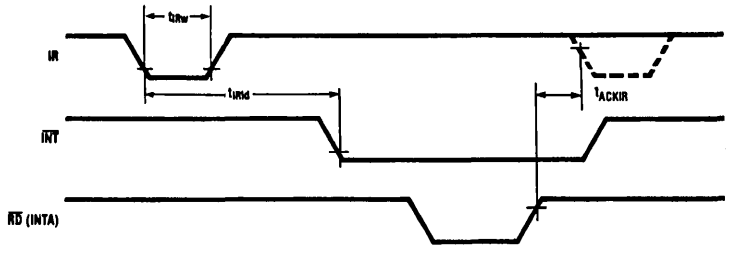
FIGURE 4-3. READ/INTA Cycle

4.0 Device Specifications (Continued)



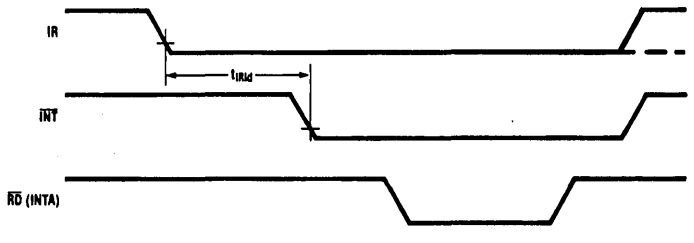
TL/EE/5117-18

FIGURE 4-4. Write Cycle



TL/EE/5117-19

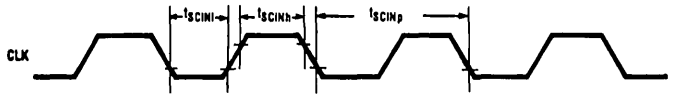
FIGURE 4-5. Interrupt Timing in Edge Triggering Mode



TL/EE/5117-20

FIGURE 4-6. Interrupt Timing in Level Triggering Mode

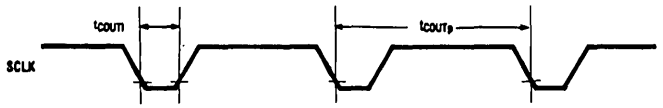
4.0 Device Specifications (Continued)



TL/EE/5117-21

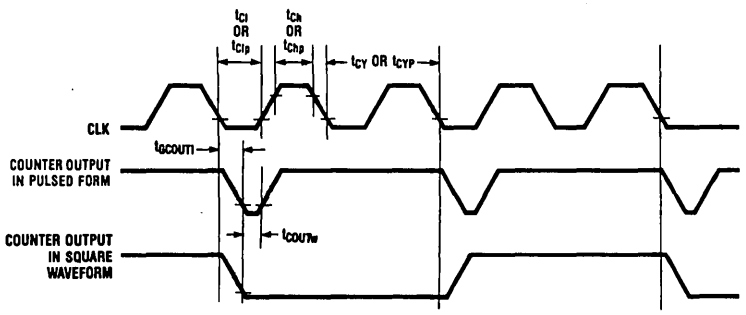
Note: Interrupts are sampled on the rising edge of CLK.

FIGURE 4-7. External Interrupt-Sampling-Clock to be Provided at Pin COUT/SCIN When in Test Mode



TL/EE/5117-22

FIGURE 4-8. Internal Interrupt-Sampling-Clock Provided at Pin COUT/SCIN



TL/EE/5117-23

FIGURE 4-9. Relationship Between Clock Input at Pin CLK and Counter Output Signals at Pins COUT/SCIN or G0/R0,...,G3/R6, in Both Pulsed Form and Square Waveform

National Semiconductor

PRELIMINARY

NS32203-10 Direct Memory Access Controller

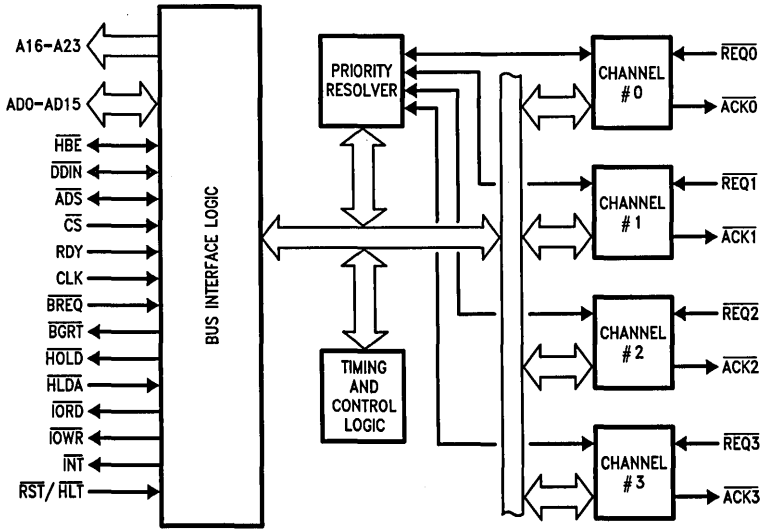
General Description

The NS32203 Direct Memory Access Controller (DMAC) is a support chip for the Series 32000® microprocessor family designed to relieve the CPU of data transfers between memory and I/O devices. The device is capable of packing data received from 8-bit peripherals into 16-bit words to reduce system bus loading. It can operate in local and remote configurations. In the local configuration it is connected to the multiplexed Series 32000 bus and shares with the CPU, the bus control signals from the NS32201 Timing Control Unit (TCU). In the remote configuration, the DMAC, in conjunction with its own TCU, communicates with I/O devices and/or memory through a dedicated bus, enabling rapid transfers between memory and I/O devices. The DMAC provides 4 16-bit I/O channels which may be configured as two complementary pairs to support chaining.

Features

- Direct or Indirect data transfers
- Memory to Memory, I/O to I/O or Memory to I/O transfers
- Remote or Local configurations
- 8-Bit or 16-Bit transfers
- Transfer rates up to 5 Megabytes per second
- Command Chaining on complementary channels
- Wide range of channel commands
- Search capability
- Interrupt Vector generation
- Simple interface with the Series 32000 Family of Microprocessors
- High Speed XMOS™ Technology
- Single +5V Supply
- 48-Pin Dual-In-Line Package

Block Diagram



TL/EE/8701–1

Table of Contents

List of Illustrations

1.0 Product Introduction

The NS32203 Direct Memory Access Controller (DMAC) is specifically designed to minimize the time required for high speed data transfers in a Series 32000-based computer system. It includes a wide variety of options and operating modes to enhance data throughput and system optimization, and to allow dynamic reconfiguration under program control.

The NS32203 can operate in two basic system configurations: local and remote. In the local configuration, the DMAC and the CPU share the same bus (address, data and control) and only one of them can perform data transfers on the bus at any one time. In this configuration, the DMAC and the CPU also share a Timing Control Unit (TCU) and a single set of address latches. Since this configuration yields a minimum part-count system, it offers a good cost/performance trade-off in many situations.

The remote configuration is intended to minimize the CPU bus use. In this configuration, the NS32203 I/O devices and optional buffer memory have their own dedicated bus (remote bus) so that an I/O transfer may be performed without loading the CPU bus (local bus).

Communication between the dedicated bus and the CPU bus may be initiated at any time by either the CPU or the NS32203. The DMAC accesses the CPU bus whenever a data transfer to/from memory or any I/O device residing on this bus is to be performed. The CPU, in turn, accesses the dedicated bus for reading status data or for programming either the DMAC or its I/O devices.

The NS32203 internal organization consists of seven functional blocks as illustrated in the block diagram. Descriptions of these blocks are given below.

DMA Channels. The NS32203 provides four channels. Each channel accepts a request from a peripheral I/O device and informs it when data transfer cycles are about to begin. A set of registers is provided for each channel to control the type of operation for that channel.

Bus Interface Unit. The bus interface unit controls all data transfers between peripheral I/O devices and memory whenever the DMAC is in control of the bus. This unit also controls the transfer of data between the CPU and the DMAC internal registers.

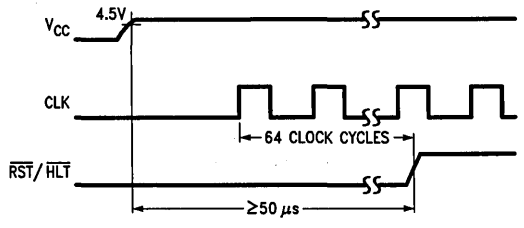
Timing and Control Logic. This block generates all the sequencing and control signals necessary for the operation of the DMAC.

Priority Resolver. This block resolves contentions among channels requesting service simultaneously.

2.0 Functional Description

2.1 RESETTING

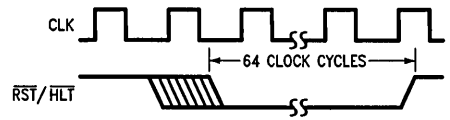
The $\overline{RST}/\overline{HLT}$ line serves both as a reset input for the on-chip logic and as a DMAC HALT input. Resetting is accomplished by pulling $\overline{RST}/\overline{HLT}$ low for at least 64 clock cycles. Upon detecting a Reset, the DMAC terminates any Data transfer in progress, resets its internal logic and enters an inactive state. On application of power, $\overline{RST}/\overline{HLT}$ must be held low for at least 50 μs after V_{CC} is stable. This is to ensure that all on-chip voltages are stable before operation. Whenever reset is applied, the rising edge must occur while the clock signal on the CLK pin is high (see *Figure 2-1* and *2-2*). The NS32201 TCU provides circuitry to meet the reset requirements. *Figure 2-3* shows the recommended connections. The HALT function is accomplished when $\overline{RST}/\overline{HLT}$ is activated for 1 or 2 clock cycles and then released. It can be used to stop any data transfer in progress in case of a bus error. As soon as HALT is acknowledged by the NS32203, the current transfer operation is terminated. See *Figure 4-18*.



TL/EE/8701–2

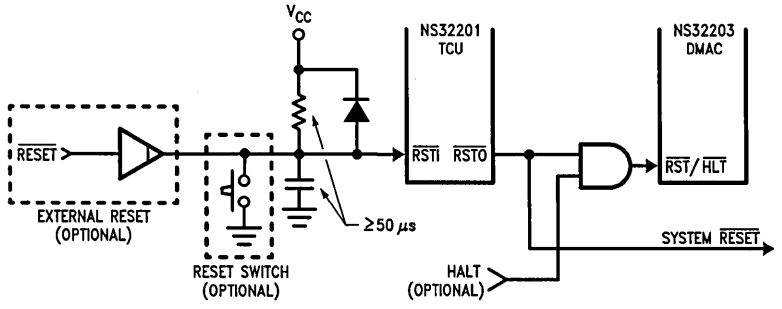
FIGURE 2-1. Power-On Reset Requirements

2.0 Functional Description (Continued)



TL/EE/8701-3

FIGURE 2-2. General Reset Timing



TL/EE/8701-4

FIGURE 2-3. Recommended Reset Connections

2.2 DATA TRANSFER OPERATIONS

After the NS32203 has been initialized by software, it is ready to transfer blocks of data, containing up to 64 kbytes, between memory and I/O devices, without further intervention required of the CPU. Upon receiving a transfer request from an I/O device, the DMAC performs the following operations:

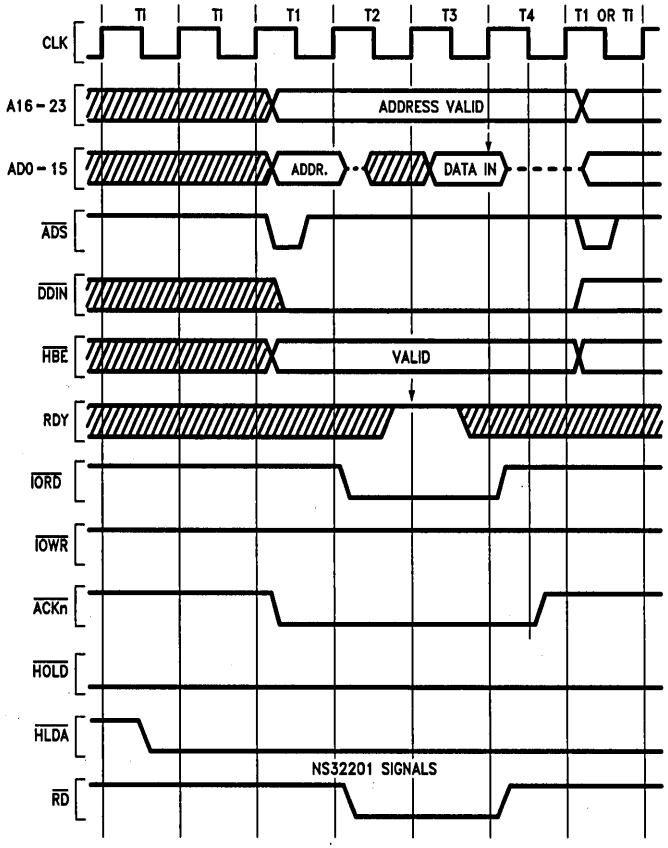
1) Acquires control of the bus
2) Acknowledge the requesting I/O device which is connected to the highest priority channel.
3) Starts executing data transfer cycles according to the values stored into the control registers of the channel being serviced.
4) Terminates data transfers and relinquishes control of the bus as soon as one of the programmed conditions is met.

Each channel can be programmed for indirect or direct data transfers. Detailed descriptions of these transfer types are provided in the following sub-sections.

2.2.1 Indirect Data Transfers

In this mode of operation, each byte or word transfer between source and destination requires at least two bus cycles. The data is first read into the DMAC and subsequently it is written into the destination. The bus cycles in this case are similar to the CPU bus cycles when the MMU is not used. This mode is slower than the direct mode, but is the only one that allows some data manipulation like Byte Search or Word Assembly/Disassembly. *Figure 2-4* and *2-5* show the read and write cycle timing diagrams related to indirect data transfers. If a search operation is specified, extra clock cycles may be added following each read cycle.

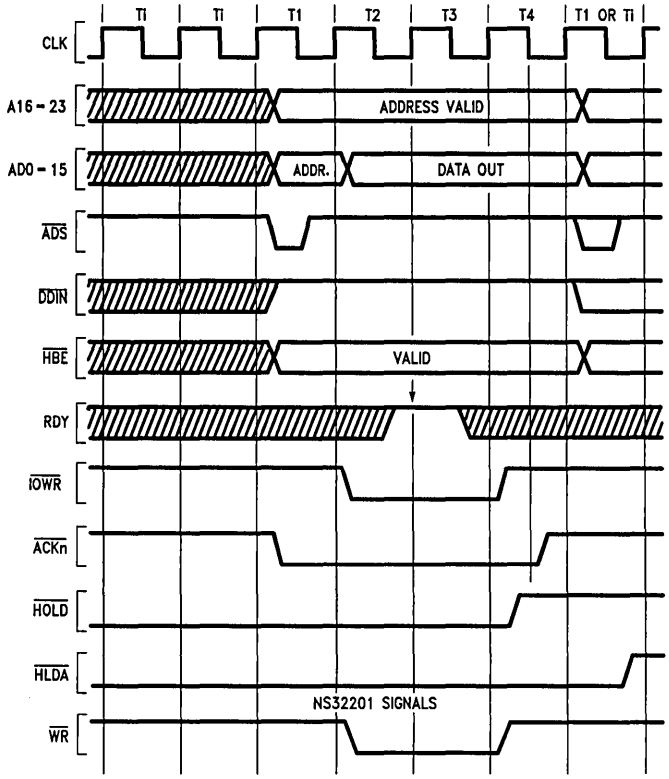
2.0 Functional Description (Continued)



TL/EE/8701–5

FIGURE 2-4. Indirect Read Cycle

2.0 Functional Description (Continued)



TL/EE/8701-6

FIGURE 2-5. Indirect Write Cycle (Single Transfer Mode)

Note: If burst mode is selected, $\overline{HOLD}$ is released at the end of the transfer operation.

2.0 Functional Description (Continued)

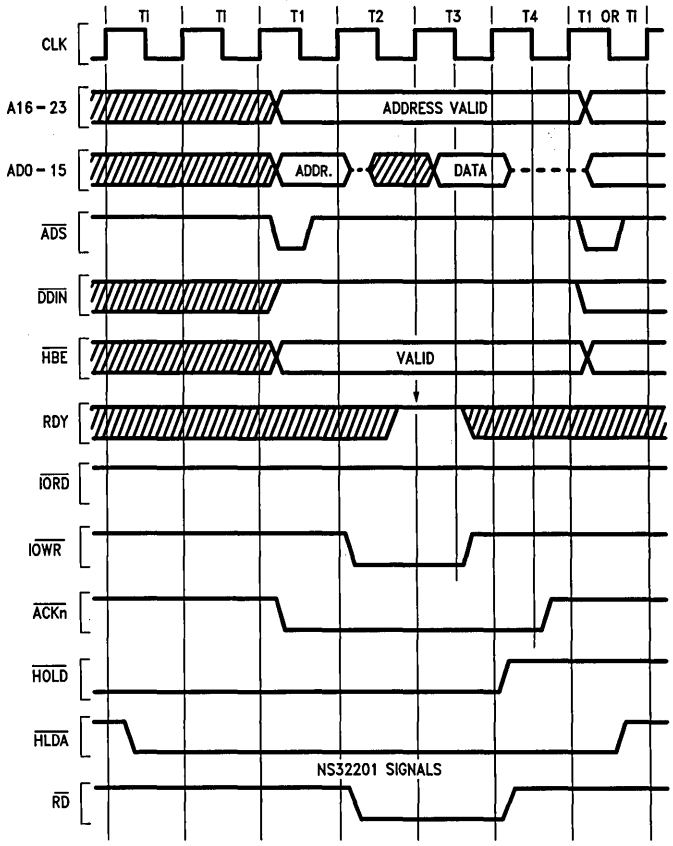
2.2.2 Direct (Flyby) Data Transfers

This mode of operation allows a very high data transfer rate between source and destination. Each data byte or word to be transferred requires only a single bus cycle instead of two separate read and write cycles, which are typical of the indirect mode. The DMAC accomplishes direct data transfers by activating $\overline{IORD}$, during memory write cycles, and $\overline{IOWR}$, during memory read cycles.

An I/O device, in the direct mode, is usually enabled by the proper acknowledge signal ($\overline{ACKn}$) from the DMAC. No search or word assembly/disassembly are possible during direct data transfers. *Figures 2-6* and *2-7* show the timing diagrams of direct memory-to-I/O and I/O-to-memory transfers respectively.

Note 1: In the direct mode each channel can control only one I/O device because the I/O device is hardwired to the $\overline{ACKn}$ output of the corresponding channel. in the indirect mode, a channel can control multiple devices as long as each device is selected through its own address rather than the $\overline{ACKn}$ output. However, the possiblity of selecting a single I/O device by the $\overline{ACKn}$ output is maintained in the indirect mode as well.

Note 2: Whenever the DMAC is either idle or is performing indirect transfers, it generates the $\overline{IORD}$ and $\overline{IOWR}$ signals as a replica of $\overline{RD}$ and $\overline{WR}$. This simplifies the logic required to access I/O devices wired for direct data transfers.



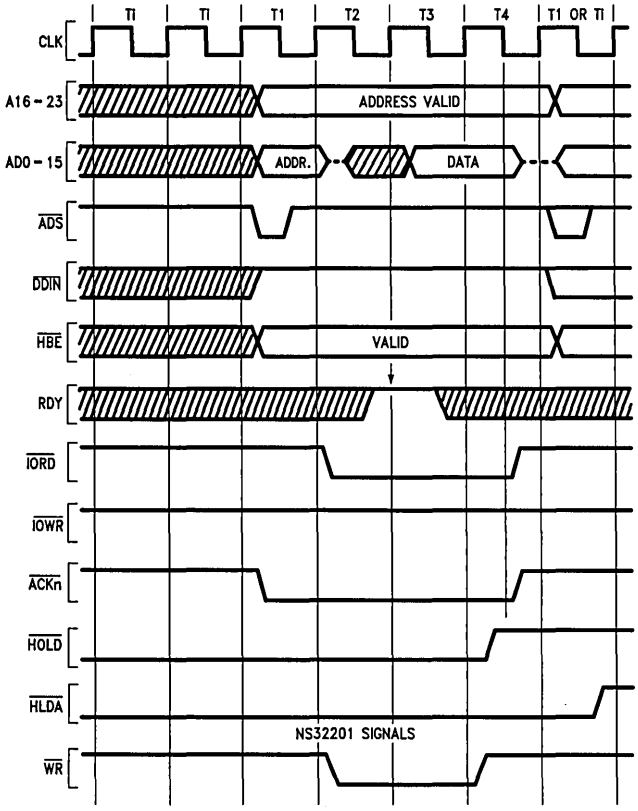
TL/EE/8701–7

FIGURE 2-6. Direct Memory-To-I/O Data Transfer (Single Transfer Mode)

2.0 Functional Description (Continued)

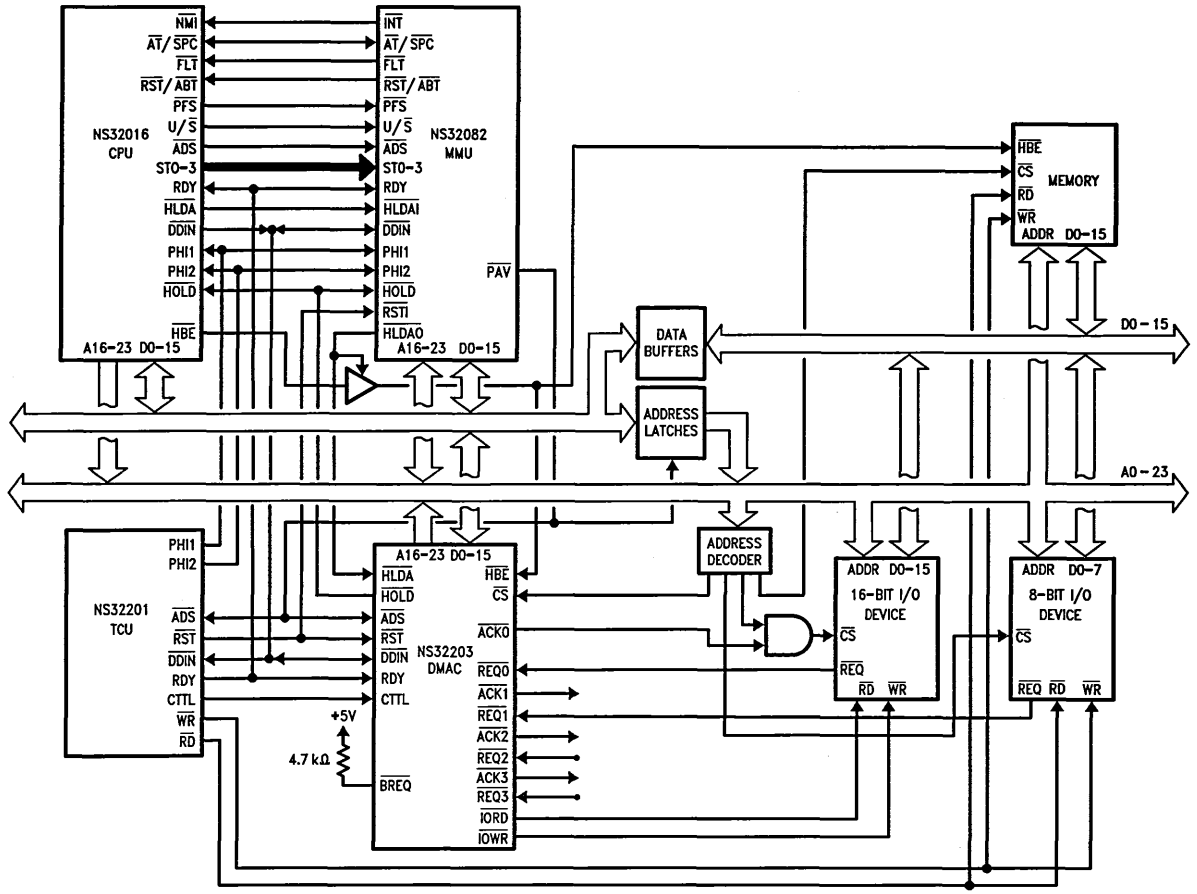
2.3 LOCAL CONFIGURATION

As previously mentioned, in the local configuration the DMAC shares with CPU and MMU the multiplexed address/data bus as well as the control signals from the NS32201 TCU. A typical local configuration is shown in *Figure 2-8*. The DMAC, in the local configuration, must gain control of the bus whenever a data transfer cycle is to be performed, even though it is directed to an I/O device and is related to an indirect data transfer. This causes the system to be quite sensitive to the volume of data handled by the DMAC. Thus, the overall system performance decreases as the volume of data increases. A possible solution to this problem is to use the remote configuration, described in the following section. A significant advantage of the local configuration is its simplicity.



TL/EE/8701-8

FIGURE 2-7. Direct I/O-To-Memory Data Transfer (Single Transfer Mode)



TL/EE/8701-9

FIGURE 2-8. NS32203 Interconnections In Local Configuration

Note 1: The 16 Bit I/O device is wired for direct transfers.
Note 2: The data buffers should not be enabled during direct data transfers or CPU accesses to the DMAC registers.

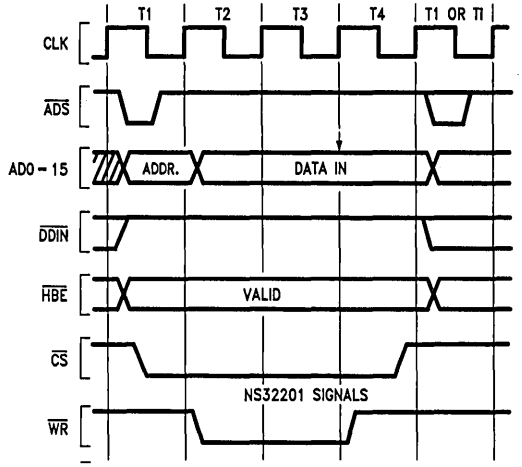
2.0 Functional Description (Continued)

2.4 REMOTE CONFIGURATION

The remote configuration is intended to minimize CPU Bus usage. In this configuration, the DMAC, buffer memory and I/O devices reside on a dedicated bus. Communication between the dedicated bus and the CPU bus is achieved by means of TRI-STATE buffers. Whenever the CPU needs to access the dedicated bus, it issues a bus request to the NS32203 by activating the $\overline{BREQ}$ signal. As the dedicated bus becomes idle, the DMAC pulls off the bus and acknowledges the CPU request by activating $\overline{BGRT}$. This output is also used as a control signal for the interconnection logic of the two buses.

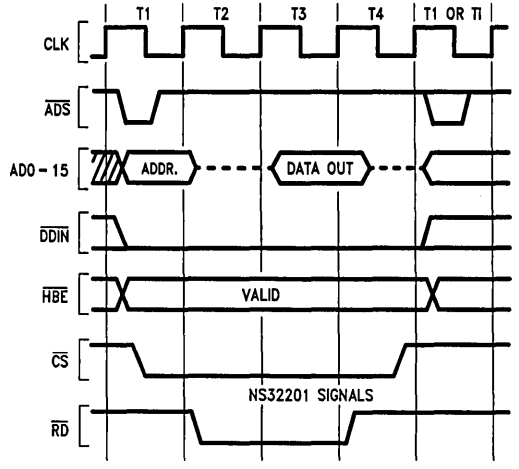
The CPU can either be interrupted by $\overline{BGRT}$ or it can poll $\overline{BGRT}$ to determine when the dedicated bus can be accessed. The DMAC, in turn, before accessing the CPU bus, has to gain control of it. This is accomplished through the usual request-acknowledge mechanism performed by means of the $\overline{HOLD}$ and $\overline{HLDA}$ signals.

Figure A-1 in Appendix A shows an interconnection diagram of a basic remote configuration. Both TCUs are clocked by the same clock signal. They are synchronized during reset by the $\overline{RWEN}$/SYNC signal so that their output clocks are in phase. *Figures 2-9* and *2-10* show the timing diagrams for read and write accesses to the NS32203 internal registers.



TL/EE/8701-10

FIGURE 2-9. Write to NS32203 Internal Registers



TL/EE/8701-11

FIGURE 2-10. Read from NS32203 Internal Registers

2.0 Functional Description (Continued)

2.5 DATA SOURCE (DESTINATION) ATTRIBUTES

Two types of data source (destination) are recognized: I/O device and memory. If the source (destination) is an I/O device, its address register is not changed after a data transfer; if it is memory, its address register is either incremented or decremented after any data transfer, according to the value of the corresponding direction bit. In the remote configuration, any data source (destination) may reside either on the CPU bus or on the dedicated bus. If it resides on the dedicated bus, the NS32203 does not activate the $\overline{HOLD}$ request line when an access to the source (destination) is performed, unless a direct transfer with a data destination (source) residing on the CPU bus is required.

Data can be transferred in either 8 bit or 16 bit units. The DMAC always considers the memory to be 16 bits wide. Thus, if an 8 bit transfer is specified, address bit A0 will determine the byte of the data-bus where the transfer takes place. If A0 = 0, the transfer occurs on the low order byte. If A0 = 1, it occurs on the high order byte. Different transfer widths can be specified for source and destination. However, some limitations exist in specifying these transfer widths when certain operations must be performed. These limitations are explained below.

1) If a transfer block has an odd number of bytes or is not word aligned, an 8 bit width for source and destination should be selected.
2) 16-bit I/O transfers can not be specified with 8 bit memory transfers.
3) Memory to memory transfers should have the same width.

Note 1: If source and destination are both memory, DMAC transfers can only be performed in indirect mode.

Note 2: If source and destination are both I/O devices and direct mode is being used, the source device is accessed by $\overline{IORD}$ and $\overline{ACK}$n; the destination device is accessed by $\overline{WR}$ (from the NS32201) and $\overline{CS}$ (from the address decoder). This allows a one direction data transfer only from one I/O device (source) to another. If data is to be transferred in both directions in direct mode between two I/O devices, two channels must be used (one for each direction of transfer), and extra hardware is required to control the read and write signals to the two I/O devices.

Note 3: When an 8-bit transfer is related to an I/O device, the other half of the 16-bit data bus is considered as DON'T CARE, and the $\overline{HBE}$/ signal may be activated.

2.6 WORD ASSEMBLY/DISASSEMBLY

This feature is automatically enabled when indirect transfers are selected, with data transferred between an 8-bit wide I/O device and a 16-bit I/O device or memory. For every 16-bit I/O device or memory access, the DMAC accesses the 8-bit I/O device twice, assembling two data bytes into a 16-bit word or breaking a 16-bit word into two data bytes, depending on the direction of transfer. The word assembly/disassembly feature allows a significant increase in the transfer speed and minimizes the CPU bus usage when the transfer occurs between an 8-bit I/O device residing on the dedicated bus, and a 16-bit I/O device or memory residing on the CPU bus. Word assembly/disassembly is not possible during direct data transfers.

Note: Requests from other channels are not acknowledged in the middle of a word assembly/disassembly. If this is unacceptable, 8 bit transfers should be specified for both source and destination.

2.7 AUTO TRANSFER

The NS32203 initiates a data transfer as a result of a request from an I/O device. In some cases a data transfer may be necessary without the corresponding request signal being asserted. This can happen, for example, when a block of data is to be moved from one memory region to another. In such cases, the auto transfer mode can be selected by setting an appropriate bit in the command register. The DMAC will initiate a data transfer regardless of the $\overline{REQ}$n signal for that channel.

Note: For proper operation, when auto transfer is required, the low order byte of the command register (containing the auto-transfer enable bit) should be written into after the other registers controlling the channel operation have been initialized.

2.8 SEARCH

The NS32203 provides a search capability that can be used to detect the occurrence of a certain data pattern. The search is performed by comparing each data byte with the search register, in conjunction with the mask register. An appropriate bit in the command register indicates whether the search continues 'UNTIL' a match occurs, or 'WHILE' a match exists. The search operation does not necessarily involve a data transfer. The DMAC allows a block of data to be searched without requiring any data transfer between source and destination. When performing a search, the user can specify whether or not the matched byte will be transferred. If 'INCLUSIVE SEARCH' is specified (INC = 1), the matched byte will be transferred, and the channel parameters will be updated accordingly. In this case, if a 16 bit word has been read from the data source and the search condition is satisfied by the low order byte, then the high order byte is transferred as well. If 'EXCLUSIVE SEARCH' is specified (INC = 0), the transfer will terminate with the last byte before the search condition was satisfied, and the parameters will point to the last transferred byte.

Search is not possible during direct transfers.

2.9 INTERRUPTS

The NS32203 provides interrupt circuitry that can be used to generate an interrupt whenever a data transfer is completed or a search condition is met. If an NS32202 ICU is used, the $\overline{INT}$ signal from the DMAC should be connected to an interrupt input of the ICU. When an interrupt occurs and the corresponding interrupt acknowledge (INTA) or return from interrupt (RETI) cycle is executed by the CPU, the NS32203 supplies its own vector as if it were a cascaded ICU. For such operation the virtual address of the interrupt vector register should be placed in the ICU cascade table, described in the NS32016 and NS32202 data sheets. See section 3.1.2.

2.10 TRANSFER MODES

When the NS32203 is in the inactive state and a channel requests service, the DMAC gains control of the bus and enters the active state. It is in this state that the data transfer takes place in one of the following modes:

SINGLE TRANSFER MODE

In single transfer mode, the NS32203 makes a single byte or word transfer for each $\overline{HOLD}$/$\overline{HLDA}$ handshake sequence.

In this case the request signal from the I/O device is edge sensitive, that is, a single transfer is performed each time a

2.0 Functional Description (Continued)

falling edge on $\overline{REQn}$ occurs. To perform multiple transfers, it is therefore necessary to temporarily deassert $\overline{REQn}$ after each transfer is initiated. If auto transfer mode is selected, the bus is released between two transfers for at least one clock cycle.

BURST (DEMAND) TRANSFER MODE

In burst transfer mode the DMAC will continue making data transfers until $\overline{REQn}$ goes inactive. Thus, the I/O device requesting service may suspend data transfer by bringing $\overline{REQn}$ inactive. Service may be resumed by asserting $\overline{REQn}$ again. If the auto transfer mode is selected, the DMAC will perform a single burst of data transfers until the end-transfer condition is reached.

Note 1: In either of the transfer modes described above, data transfers can only occur as long as the byte count is not zero or a search condition is not met. Whenever any of these conditions occur, the NS32203 terminates the current operation and releases the bus for at least one clock cycle.

Note 2: Whenever the DMAC releases $\overline{HOLD}$, it waits for $\overline{HLDA}$ to go inactive for at least one clock cycle before reasserting $\overline{HOLD}$ again to continue the transfer operation.

2.11 CHAINING

The NS32203 provides a chaining feature that allows the four DMAC channels to be regarded as two complementary pairs. Channels 0 and 1 form the first pair, while channels 2 and 3 form the second pair. Each pair is programmed independently by setting the corresponding bit in the configuration register. When two channels are complementary, only the even channel can perform transfer operations, while the odd one serves as temporary storage for the new control values and parameters loaded for the chaining operation. If an operation is being performed by the even channel of a pair and an end-condition is reached, the channel is not returned to the inactive state; rather, a new set of control values with or without parameters is loaded from the complementary channel and a new operation is started. During the reload operation the bus is released for at least two clock cycles. At the end of the second operation the channel returns to the inactive state, unless a new set of values has been loaded into the complementary channel by the CPU.

The chaining feature can be used to transfer blocks of data to/from non-contiguous memory segments. For example, the CPU can load channel 0 and 1 with control values and parameters for the first two blocks. After the operation for the first block is completed by channel 0, the control values and parameters stored in channel 1 are transferred to channel 0, during an update cycle, and a second operation is started. The CPU, being notified by an interrupt, can load channel 1 registers with control values and parameters for the third data block.

Note 1: Whenever a reload operation occurs, the register values of the complementary channel are affected. Thus, the CPU must always load a new set of values into the complementary channel if another chaining operation is required.

Note 2: When the chain option is selected, the CPU must be given the opportunity to acquire the bus for enough time between DMAC operations, in order for the complementary channel to be updated.

2.12 CHANNEL PRIORITIES

The NS32203 has four I/O channels, each of which can be connected to an I/O device. Since no dependency exists between the different I/O devices, a priority level is assigned to each I/O channel, and a priority resolver is provided to resolve multiple requests activated simultaneously.

The priority resolver checks the priorities on every cycle. If a channel is being serviced and a higher priority request is received, the channel operation is suspended and control passes to the higher priority channel, unless the lock bit for the lower priority channel is set. If the lock bit is set, that channel operation is continued until completion before control passes to the higher priority channel. The bus is always released for at least two clock cycles when control passes from one channel to another.

Two types of priority encodings are available as software selectable options.

The first is fixed priority which fixes the channels in priority order based on the decreasing values of their numbers. Channel 3 has the lowest priority, while channel 0 has the highest.

The second option is variable priority. The last channel that receives service becomes the lowest priority channel among all other channels with variable priority, while the channels which previously had lower priority will get their priorities increased. If variable priority is selected for all four channels, any I/O device requesting service is guaranteed to be acknowledged after no more than three higher priority services have occurred. This prevents any channel from monopolizing the system. Priority types can be intermixed for different channels.

As an example, let channels 0, 2 and 3 have variable priority and channel 1 fixed priority. Channel 2 receives service first, followed by channel 0. The priority levels among all channels will change as follows.

Priority	Initial Order	Next Order	Final Order
High 3	ch.0	ACK → ch.0	ch.3
2	ch.1	ch.1	ch.1 → fixed priority
1	ACK → ch.2	ch.3	ch.2
Low 0	ch.3	ch.2	ch.0

Whenever the PT bit (priority type) in the command register is changed, the priority levels of all the channels are reset to the initial order. If only one channel has variable priority, then no change in priority will occur from the initial order.

Note: If the lock bit is not set, three idle states are inserted between the write cycle of a previous burst indirect transfer and the next read cycle.

3.0 Architectural Description

The NS32203 has 128 8-bit registers that can be addressed either individually or in pairs, using the 7 least significant bits of the address bus and the high byte enable signal $\overline{HBE}$. Seventy-one of these registers are reserved, while the rest are accessible by the CPU for read/write operations. *Figure 3-1* shows the NS32203 internal registers together with their address offsets. Detailed descriptions of these registers are given in the following sections.

3.1 GLOBAL REGISTERS

The global registers consist of one configuration, one status and two interrupt vector registers. They are shared by all channels, and they control the overall operation of the NS32203.

3.1.1 CONF—Configuration Register

This register controls the hardware configuration of the NS32203 as well as the chaining feature.

3.0 Architectural Description (Continued)

The CONF register format is shown below:

7	6	5	4	3	2	1	0
XXXXX					C1	C0	CNF

CNF — Configuration Bit. Determines whether the NS32203 is in local or remote configuration.
CNF = 0 = > Local Configuration
CNF = 1 = > Remote Configuration

C0 — Chaining bit for channels 0 and 1. Determines whether or not channel 0 and 1 are complementary.
C0 = 0 = > Channels not complementary
C0 = 1 = > Channel 1 complementary to channel 0

C1 — Chaining bit for channels 2 and 3. Determines whether or not channels 2 and 3 are complementary.
C1 = 0 = > Channels not complementary
C1 = 1 = > Channel 3 complementary to channel 2

XXXXX — Reserved. These bits should be set to 0.

At reset, all CONF bits are reset to zero.

Note: The CNF bit should never be set by the software if the DMAC is wired for local configuration, otherwise bus conflicts will result.

	23 — 16	15 — 8	7 — 0	
Channel 0 Control Registers	COM(H) (02_{16})	COM(M) (01_{16})	COM(L) (00_{16})	Command
			SRCH (04_{16})	Search Pattern
			MSK (08_{16})	Search Mask
Channel 0 Parameter Registers	SRC(H) ($0E_{16}$)	SRC(M) ($0D_{16}$)	SRC(L) ($0C_{16}$)	Source Address
	DST(H) (12_{16})	DST(M) (11_{16})	DST(L) (10_{16})	Destination Address
		LNGT(H) (15_{16})	LNGT(L) (14_{16})	Block Length
Channel 1 Control Registers	COM(H) (22_{16})	COM(M) (21_{16})	COM(L) (20_{16})	Command
			SRCH (24_{16})	Search Pattern
			MSK (28_{16})	Search Mask
Channel 1 Parameter Registers	SRC(H) ($2E_{16}$)	SRC(M) ($2D_{16}$)	SRC(L) ($2C_{16}$)	Source Address
	DST(H) (32_{16})	DST(M) (31_{16})	DST(L) (30_{16})	Destination Address
		LNGT(H) (35_{16})	LNGT(L) (34_{16})	Block Length
Channel 2 Control Registers	COM(H) (42_{16})	COM(M) (41_{16})	COM(L) (40_{16})	Command
			SRCH (44_{16})	Search Pattern
			MSK (48_{16})	Search Mask
Channel 2 Parameters Registers	SRC(H) ($4E_{16}$)	SRC(M) ($4D_{16}$)	SRC(L) ($4C_{16}$)	Source Address
	DST(H) (52_{16})	DSC(M) 51_{16})	DST(L) (50_{16})	Destination Address
		LNGT(H) (55_{16})	LNGT(L) (54_{16})	Block Length
Channel 3 Control Registers	COM(H) (62_{16})	COM(M) (61_{16})	COM(L) (60_{16})	Command
			SRCH (64_{16})	Search Pattern
			MSK (68_{16})	Search Mask
Channel 3 Parameter Registers	SRC(H) ($6E_{16}$)	SRC(M) ($6D_{16}$)	SRC(L) ($6C_{16}$)	Source Address
	DST(H) (72_{16})	DST(M) (71_{16})	DST(L) (70_{16})	Destination Address
		LNGT(H) (75_{16})	LNGT(L) (74_{16})	Block Length
Global Registers			CONF (78_{16})	Configuration
			SVCT ($5C_{16}$)	Software Vector
			HVCT ($7C_{16}$)	Hardware Vector
		STAT(H) ($7F_{16}$)	STAT(L) ($7E_{16}$)	Status

FIGURE 3-1. NS32203 Internal Registers

3.0 Architectural Description (Continued)

3.1.2 HVCT — Hardware Vector Register

This register contains the interrupt vector byte that is supplied to the CPU during an interrupt acknowledge (INTA) or return from interrupt (RETI) cycle. The HVCT register format is shown below.

7	6	5	4	3	2	1	0
BIAS					E	CN	

CN — Channel number. Represents the number of the interrupting channel

E — Error code. Determines whether a normal operation completion or an error condition has occurred on the interrupting channel.

E = 0 => Normal Operation Completion

E = 1 => A second interrupt was generated by the same channel before the first interrupt was serviced.

BIAS — Programmable bias. This field is programmed by writing the pattern BBBBB000 into the HVCT register.

The NS32203 always interprets a read of the HVCT register as either an interrupt acknowledge (INTA) cycle or a return from interrupt (RETI) cycle. Since these cycles cause internal changes to the DMAC, normal programs should never read the HVCT register (see next section). The DMAC distinguishes an INTA cycle from a RETI cycle by the state of an internal flip-flop, called Interrupt Service Flip-Flop, that toggles every time the HVCT register is read. This flip-flop is cleared on reset or when the HVCT register is written into. When an interrupt is acknowledged by the CPU, the $\overline{INT}$ signal is deasserted unless another interrupt from a lower priority channel is pending. In this case the $\overline{INT}$ signal is deasserted when the acknowledge cycle for the second interrupt is performed.

For this reason, if the $\overline{INT}$ signal is connected to an interrupt input of the NS32202 ICU, the triggering mode of that interrupt position should be 'low level'.

Furthermore, if that ICU interrupt input is programmed for cascaded operation and nesting of interrupts from other devices connected to the ICU is to be allowed, then the ICU interrupt input connected to the DMAC should be masked off during the interrupt service routine, before the CPU interrupt is reenabled. This is because the DMAC does not provide interrupt nesting capability.

An interrupt from a certain channel can be acknowledged only after the return from interrupt from a previously acknowledged interrupt is performed.

3.1.3 SVCT — Software Vector Register

The SVCT register is an image of the HVCT register. It is a read-only register used for diagnostics. It allows the programmer to read the interrupt vector without affecting the interrupt logic of the NS32203. The format of the SVCT register is the same as that of the HVCT register.

3.1.4 STAT — Status Register

The status register contains status information of the NS32203, and can be used when the interrupts are not enabled. Each set bit is automatically cleared when a read operation is performed. The format of this register is shown in the following figure.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ME	CH	MN	TC	ME	CH	MN	TC	ME	CH	MN	TC	ME	CH	MN	TC
channel #3				channel #2				channel #1				channel #0			

The status of each channel is defined in a four-bit field as described below:

TC — Transfer Complete.

Indicates the completion of a channel operation, regardless of the state of the length register or whether a match/no match condition occurred.

MN — Match/No Match Bit.

This bit is set when a match/no match condition occurs.

CH — Channel Halted.

Set when a channel operation is halted by pulling the $\overline{RST}/\overline{HLT}$ pin.

ME — Multiple events. This bit is set when more than one of the above conditions have occurred.

Note: If an interrupt is enabled, the corresponding bit in the status register is not cleared upon read, unless the interrupt is acknowledged.

3.2 CONTROL REGISTERS

Each of the four channels has three control registers, consisting of a 24-bit command register, an 8-bit search register and an 8-bit mask register.

3.2.1 COM — Command Register

The command register controls the operation of the associated channel. It is divided into three separately addressable parts: COM(L), COM(M) and COM(H). The format of each part and bit functions are shown below.

COM(L) — Command Register (Low-Byte)

7	6	5	4	3	2	1	0
AT	LK	PT	UW	INC	DI	CC	

CC — Command Code

CC = 00 => Channel Disabled.

CC = 01 => Search

CC = 10 => Data Transfer

CC = 11 => Data Transfer and Search

DI — Direct/Indirect Transfers

DI = 0 => Indirect Transfers

DI = 1 => Direct Transfers

INC — Inclusive/Exclusive Search

INC = 0 => Exclusive Search

INC = 1 => Inclusive Search

UW — Search type

UW = 0 => Search UNTIL

UW = 1 => Search WHILE

PT — Priority type

PT = 0 => Fixed

PT = 1 => Variable

LK — Priority lock

LK = 0 => Priority Unlocked

LK = 1 => Priority Locked

4

3.0 Architectural Description (Continued)

AT — Auto transfer

AT = 0 => Auto Transfer Disabled

AT = 1 => Auto Transfer Enabled

At Reset, the CC bits in COM(L) are cleared, disabling the channel.

Note: The CC bits can be cleared by software during an indirect data transfer to stop the transfer. This, however, should not be done during direct data transfers. See section 3.3.3.

COM(M) - Command Register (Middle-Byte)

7	6	5	4	3	2	1	0
DD	DW	DL	DT	SD	SW	SL	ST

ST — Source Type

ST = 0 => I/O Device

ST = 1 => Memory

SL — Source Location

(Effective only in the remote configuration)

SL = 0 => Local

SL = 1 => Remote

SW — Source Width

SW = 0 => 8 Bits

SW = 1 => 16 Bits

SD — Source Direction

SD = 0 => Up

SD = 1 => Down

DT — Destination Type

DT = 0 => I/O Device

SD = 1 => Memory

DL — Destination Location

(Effective only in the remote configuration)

DL = 0 => Local

DL = 1 => Remote

DW — Destination Width

DW = 0 => 8 Bits

DW = 1 => 16 Bits

DD — Destination Direction.

DD = 0 => Up

DD = 1 => Down

COM(H) - Command Register (High-Byte)

7	6	5	4	3	2	1	0
HLI	MNI	TCI	AMN		ATC	DM	X

X — Reserved. (Should be set to 0)

TM — Transfer Mode

DM = 0 => Single Transfer

DM = 1 => Burst Transfer

ATC — Action after Transfer Complete

ATC = 0 => Disable Channel

ATC = 1 => Load Control Values and Parameters from Complementary Channel and Continue

AMN — Action after Match/No Match

AMN = 00 => Disable Channel

AMN = 01 => Continue

AMN = 10 => Load Control Values from Complementary Channel and Continue

AMN = 11 => Load Control Values and Parameters from Complementary Channel and Continue

TCI — Interrupt Mask on "Transfer Complete"

TCI = 0 => No Interrupt

TCI = 1 => Interrupt

MNI — Interrupt Mask on "Match/No Match"

MNI = 0 => No Interrupt

MNI = 1 => Interrupt

HLI — Interrupt Mask on "Channel Halted"

HLI = 0 => No Interrupt

HLI = 1 => Interrupt

3.2.2 SRCH — Search Register

This 8-bit register holds the value to be compared with the data transferred during the channel operation.

3.2.3 MSK — Mask Register

The 8-bit mask register determines which bits of the transferred data are compared with corresponding search register bits. If a mask register bit is set to 0, the corresponding search register bit is ignored in the compare operation. At reset, all the MSK bits are set to 0.

3.3 PARAMETER REGISTERS

Each channel has three parameter registers, consisting of a 24-bit source address register, a 24-bit destination address register and a 16-bit block length register.

3.3.1 SRC — Source Address Register

The source address register points to the physical address of the data source. When the data source is an I/O device, the register does not change during the transfer operation. When the data source is memory, the register is incremented or decremented by either one or two after each transfer.

3.3.2 DST — Destination Address Register

The destination address register points to the physical address of the data destination. When the data destination is an I/O device, the register does not change during the transfer operation. When the data destination is memory, the register is incremented or decremented by either one or two after each transfer.

3.3.3 LNGT — Block Length Register

The block length register holds the number of bytes in the block to be transferred. It is decremented by either one or two after each transfer.

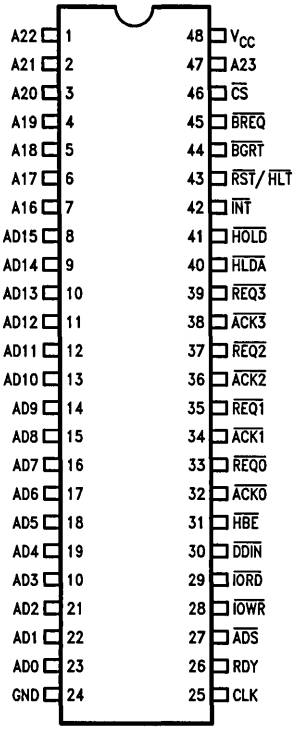
Note: A direct data transfer can be stopped by writing zeroes into the LNGT register. The number of bytes transferred can be determined in this case, from the value of either the SRC or the DST register.

4.0 Device Specifications

4.1. NS32203 PIN DESCRIPTIONS

The following is a brief description of all NS32203 pins. The descriptions reference portions of the Functional Description, Section 2.0.

Connection Diagram



TL/EE/8701-12

Top View

FIGURE 4-1. NS32203 Dual-In-Line Package

Order Number NS32203D or NS32203N
See NS Package Number D48A or N48A

4.1.1 SUPPLIES

Power (V_{CC}): +5V positive supply.

Ground (GND): Ground reference for on-chip logic.

4.1.2 INPUT SIGNALS

Reset/Halt ($\overline{RST}/\overline{HLT}$): Active low. If held active for 1 or 2 clock cycles and released, this signal halts the DMAC operation on the active channel. If held longer, it resets the DMAC. Section 2.1.

Chip Select ($\overline{CS}$): When low, the device is selected, enabling CPU access to the DMAC internal registers.

Ready (RDY): Active high. When inactive, the DMA Controller extends the current bus cycle for synchronization with slow memory or peripherals. Upon detecting RDY active, the DMAC terminates the bus cycle.

Channel Request 0–3 ($\overline{REQ0}$ - $\overline{REQ3}$): Active low. These lines are used by peripheral devices to request DMAC service.

Bus Request ($\overline{BREQ}$): Used only in the remote configuration. This signal, when asserted, forces the DMAC to stop transferring data and to release the bus. It must be activated by the CPU before any CPU access to the remote bus is performed. In the local configuration this signal should be connected to V_{CC} via a 4.7k resistor. Section 2.4.

Hold Acknowledge ($\overline{HLDA}$): Active low. When asserted, indicates that control of the system bus has been relinquished by the current bus master and the DMAC can take control of the bus.

Clock (CLK): Clock signal supplied by the CTTL output of the NS32201 TCU.

4.1.3 OUTPUT SIGNALS

Address Bits 16–23 (A16–A23): Most significant 8 bits of the address bus.

Hold Request ($\overline{HOLD}$): Active low. Used by the DMAC to request control of the system bus.

Channel Acknowledge 0–3 ($\overline{ACK0}$ - $\overline{ACK3}$): These lines indicate that a channel is active. When a channel's request is honored, the corresponding acknowledge line is activated to notify the peripheral device that it has been selected for a transfer cycle. Section 2.2.2.

Bus Grant ($\overline{BGRT}$): Used only in the remote configuration. This signal is used by the DMAC to inform the CPU that the remote bus has been relinquished by the DMAC and can be accessed by the CPU. Section 2.4.

I/O Read ($\overline{IORD}$): Active low. Enables data to be read from a peripheral device. Section 2.2.2.

I/O Write ($\overline{IOWR}$): Active low. Enables data to be written to a peripheral device. Section 2.2.2.

Interrupt ($\overline{INT}$): Active low. Used to generate an interrupt request when a programmed condition has occurred. Section 2.9.

4.1.4 INPUT/OUTPUT SIGNALS

Address/Data 0–15 (AD0–AD15): Multiplexed Address/Data bus lines. Also used by the CPU to access the DMAC internal registers.

High Byte Enable ($\overline{HBE}$): Active low. Enables data transfers on the most significant byte of the data bus.

Address Strobe ($\overline{ADS}$): Active low. Controls address latches and indicates the start of a bus cycle.

Data Direction In ($\overline{DDIN}$): Active low. Status signal indicating the direction of data flow in the current bus cycle.

4.0 Device Specifications (Continued)

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias	0°C to +70°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	1.1 Watt

Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_A = 0 to +70°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		V_{CC} + 0.5	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low Level Output Voltage	I_{OL} = 2 mA			0.45	V
I_I	Input Load Current	0 < V_{IN} ≤ V_{CC}	−20		20	μA
I_L	Leakage Current Output and I/O Pins in TRI-STATE/Input Mode	0.4 ≤ V_{IN} ≤ V_{CC}	−20		20	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C		180	300	mA

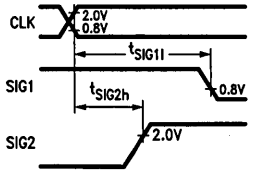
4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 0.8V and 2.0V on all the input and output signals as illustrated in *Figures 4-2* and *4-3*, unless specifically stated otherwise.

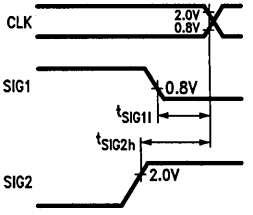
ABBREVIATIONS:

L.E. — leading edge R.E. — rising edge
T.E. — trailing edge F.E. — falling edge



TL/EE/8701–13

FIGURE 4-2. Timing Specification Standard (Signal Valid after Clock Edge)



TL/EE/8701–14

FIGURE 4-3. Timing Specification Standard (Signal Valid before Clock Edge)

4.0 Device Specifications (Continued)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32203-10

Maximum Times Assume Capacitive Loading of 100 pF.

Name	Figure	Description	Reference/ Conditions	NS32203-10		Units
				Min	Max	
t_{ALv}	4-7	Address Bits 0–15 Valid	After R.E., CLK T1		50	ns
t_{ALh}	4-9	Address Bits 0–15 Hold Time	After R.E., CLK T2	5		ns
t_{AHv}	4-7	Address Bits 16–23 Valid	After R.E., CLK T1		50	ns
t_{AHh}	4-7	Address Bits 16–23 Hold	After R.E., CLK T1 or Ti	5		ns
t_{ALADSs}	4-8	Address Bits 0–15 Set Up	Before $\overline{ADS}$ T.E.	25		ns
t_{AHADSs}	4-8	Address Bits 16–23 Set Up	Before $\overline{ADS}$ T.E.	25		ns
t_{ALADSh}	4-9	Address Bits 0–15 Hold Time	After $\overline{ADS}$ T.E.	15		μs
t_{ALf}	4-8	Address Bits 0–15 Floating	After R.E., CLK T2		25	ns
t_{Dv}	4-7	Data Valid (Write Cycle)	After R.E., CLK T2		50	ns
t_{Dh}	4-7	Data Hold (Write Cycle)	After R.E., CLK T1 or Ti	0		ns
t_{DOv}	4-5	Data Valid (Reading DMAC Registers)	After R.E., CLK T3		50	
t_{DOh}	4-5	Data Hold (Reading DMAC Registers)	After R.E., CLK T4	10		
t_{HBEv}	4-7	$\overline{HBE}$ Signal Valid	After R.E., CLK T1		50	ns
t_{HBEh}	4-7	$\overline{HBE}$ Signal Hold	After R.E., CLK T1 or Ti	0		ns
t_{DDINv}	4-8	$\overline{DDIN}$ Signal Valid	After R.E., CLK T1		65	ns
t_{DDINh}	4-8	$\overline{DDIN}$ Signal Hold	After R.E., CLK T1 or Ti	0		ns
t_{ADSa}	4-7	$\overline{ADS}$ Signal Active	After R.E., CLK T1		35	ns
t_{ADSia}	4-7	$\overline{ADS}$ Signal Inactive	After R.E., CLK T1		40	ns
t_{ADSw}	4-7	$\overline{ADS}$ Pulse Width	at 0.8V (Both Edges)	30		ns
t_{ALz}	4-12. 4-13	AD0–AD15 Floating	After R.E., CLK Ti		55	ns
t_{AHz}	4-12, 4-13	A16–A23 Floating	After R.E., CLK Ti		55	ns
t_{ADSz}	4-12, 4-13	$\overline{ADS}$ Floating	After R.E., CLK Ti		55	ns
t_{HBEz}	4-12, 4-13	$\overline{HBE}$ Floating	After R.E., CLK Ti		55	ns
t_{DDINz}	4-12, 4-13	$\overline{DDIN}$ Floating	After R.E., CLK Ti		55	ns
t_{HLDa}	4-11	$\overline{HOLD}$ Signal Active	After R.E., CLK Ti		50	ns
t_{HLDia}	4-12	$\overline{HOLD}$ Signal Inactive	After R.E., CLK Ti or T4		50	ns
t_{INTa}	4-19, 4-21	$\overline{INT}$ Signal Active	After R.E., CLK Ti		40	ns
t_{ACKa}	4-16, 4-17, 4-7	$\overline{ACKn}$ Signal Active	After R.E., CLK T1		50	ns
t_{ACKia}	4-16, 4-17, 4-7	$\overline{ACKn}$ Signal Inactive	After F.E., CLK T4		35	ns

4.0 Device Specifications (Continued)

Name	Figure	Description	Reference/ Conditions	NS32203-10		Units
				Min	Max	
t_{BGRTa}	4-13	$\overline{BGRT}$ Signal Active	After R.E., CLK		65	ns
t_{BGRTia}	4-14	$\overline{BGRT}$ Signal Inactive	After R.E., CLK		65	ns
t_{IORDa}	4-8, 4-9	$\overline{IORD}$ Active	After R.E., CLK T2		40	ns
t_{IORDia}	4-8	$\overline{IORD}$ Inactive (During Indirect Transfers)	After R.E., CLK T4		40	ns
t_{IORDia}	4-9	$\overline{IORD}$ Inactive (During Direct Transfers)	After F.E., CLK T4		40	ns
t_{IOWRa}	4-7, 4-10	$\overline{IOWR}$ Active	After R.E., CLK T2		40	ns
t_{IOWRia}	4-7	$\overline{IOWR}$ Inactive (During Indirect Transfers)	After R.E., CLK T4		40	ns
$t_{IOWRdia}$	4-10	$\overline{IOWR}$ Inactive (During Direct Transfers)	After F.E., CLK T3		40	ns
4.4.2.2 Input Signal Requirements: NS32203-10						
t_{PWR}	4-22	Power Stable to $\overline{RST}/\overline{HLT}$ R.E.	After V_{CC} Reaches 4.75V	50		µs
t_{RSTw}	4-23	$\overline{RST}/\overline{HLT}$ Pulse Width (Resetting the DMAC)	at 0.8V (Both Edges)	64		tCp
t_{RSTs}	4-24	$\overline{RST}/\overline{HLT}$ Set Up Time (Resetting the DMAC)	Before F.E., CLK	15		ns
t_{HLTs}	4-18	$\overline{RST}/\overline{HLT}$ Setup Time (Halting a DMAC Transfer)	Before R.E., CLK T3	25		ns
t_{HLTh}	4-19	$\overline{RST}/\overline{HLT}$ Hold Time (Halting a DMAC Transfer)	After R.E., CLK T4	10		ns
t_{DIs}	4-6	Data in Setup Time	Before R.E., CLK T3	15		ns
t_{DIh}	4-6	Data in Hold	After R.E., CLK T4	3		ns
t_{DIs}	4-6	Data in Setup Time (Writing to DMAC Registers)	After R.E., CLK T3	15		ns
t_{DIh}	4-6	Data in Hold (Writing to DMAC Registers)	After R.E., CLK T4	3		ns
t_{HLDAs}	4-11, 4-12	$\overline{HOLDA}$ Setup Time	Before R.E., CLK	25		ns
t_{HLDAh}	4-11	$\overline{HLDA}$ Hold Time	After R.E., CLK	10		ns
t_{RDYs}	4-15	RDY Setup Time	Before R.E., CLK T2 or T3	20		ns
t_{RDYh}	4-15	RDY Hold Time	After R.E., CLK T3	5		ns
t_{REQs}	4-16, 4-17	$\overline{REQn}$ Setup Time	Before R.E., CLK	50		ns
t_{REQh}	4-16, 4-17	$\overline{REQn}$ Hold Time	After R.E., CLK	10		
t_{BREQs}	4-13	$\overline{BREQ}$ Setup Time	Before R.E., CLK	25		ns

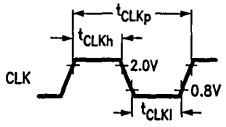
4.0 Device Specifications (Continued)

Name	Figure	Description	Reference/ Conditions	NS32203-10		Units
				Min	Max	
t_{BREQh}	4-13	$\overline{BREQ}$ Hold Time	After R.E., CLK	10		ns
$t_{ALADSis}$	4-6	Address Bits 0–5 Setup	Before $\overline{ADS}$ T.E.	20		ns
$t_{ALADSih}$	4-6	Address Bits 0–5 Hold	After $\overline{ADS}$ T.E.	20		ns
t_{HBEs}	4-6	$\overline{HBE}$ Setup Time	Before R.E., CLK T1	10		ns
t_{HBEih}	4-6	$\overline{HBE}$ Hold Time	After R.E., CLK T4	40		ns
t_{ADSs}	4-6	$\overline{ADS}$ L.E. Setup Time	Before R.E., CLK T1	40		ns
t_{ADSiw}	4-6	$\overline{ADS}$ Pulse Width	$\overline{ADS}$ L.E. to $\overline{ADS}$ T.E.	35		ns
t_{CSs}	4-6	$\overline{CS}$ Setup Time	Before R.E., CLK T1	15		ns
t_{CSh}	4-6	$\overline{CS}$ Hold Time	After R.E., CLK T4	40		ns
t_{DDINs}	4-6	$\overline{DDIN}$ Setup Time	Before R.E., CLK T2	30		ns
t_{DDINh}	4-6	$\overline{DDIN}$ Hold Time	After R.E., CLK T4	40		ns

4.4.2.3 Clocking Requirements: NS32203-10

Name	Figure	Description	Reference/ Conditions	NS32203-10		Units
				Min	Max	
t_{CLKh}	4-4	Clock High Time	At 2.0V (Both Edges)	42		ns
t_{CLK1}	4-4	Clock Low Time	At 0.8V (Both Edges)	42		ns
t_{CLKp}	4-4	Clock Period	R.E., CLK to Next R.E. CLK	100		ns

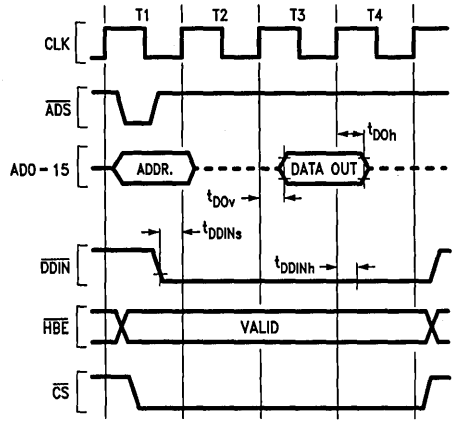
4.4.3 Timing Diagrams



TL/EE/8701–17

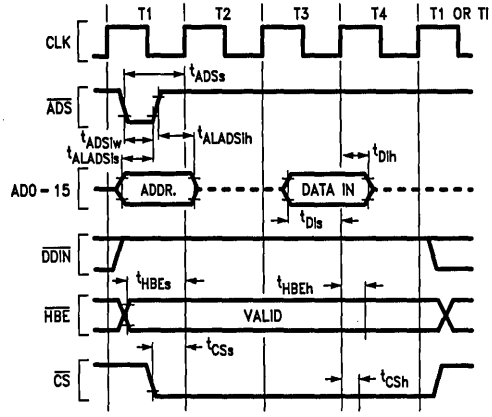
FIGURE 4-4. Clock Timing

4.0 Device Specifications (Continued)



TL/EE/8701–16

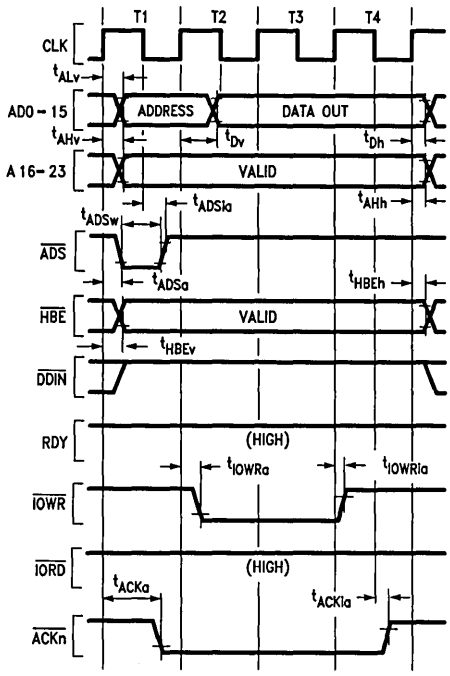
FIGURE 4-5. Read from DMAC Registers



TL/EE/8701–15

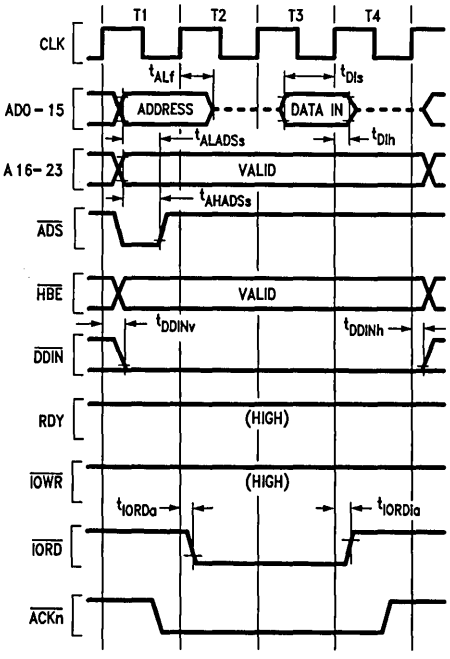
FIGURE 4-6. Write to DMAC Registers

4.0 Device Specifications (Continued)



TL/EE/8701-18

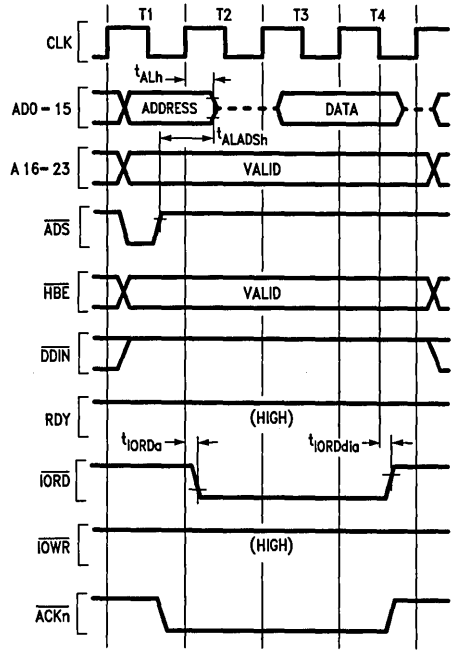
FIGURE 4-7. Indirect Write Cycle



TL/EE/8701-19

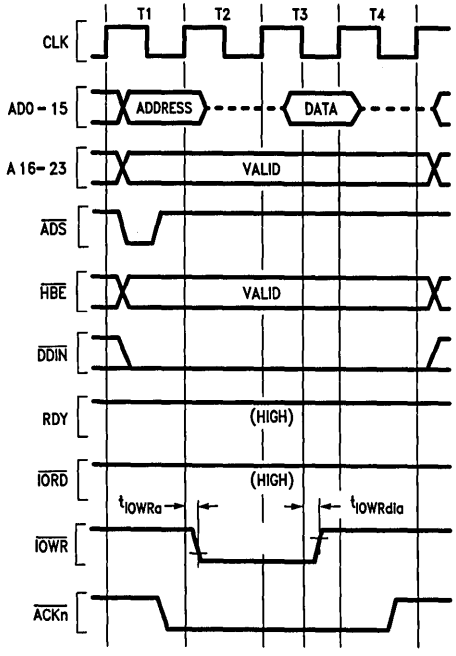
FIGURE 4-8. Indirect Read Cycle

4.0 Device Specifications (Continued)



TL/EE/8701-20

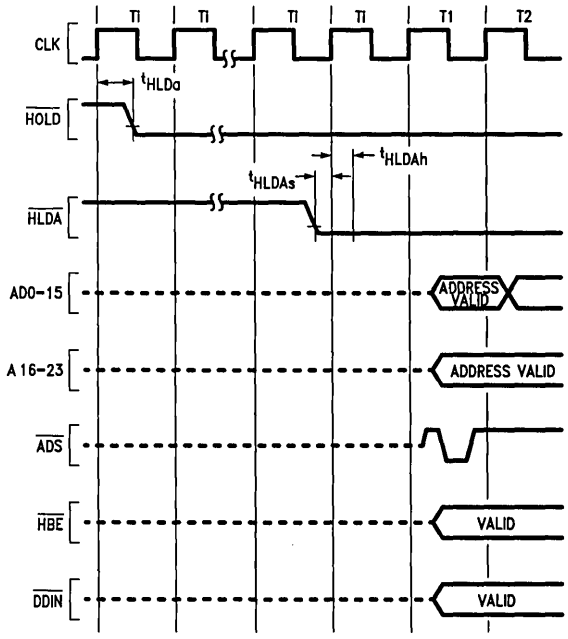
FIGURE 4-9. Direct I/O to Memory Transfer



TL/EE/8701-21

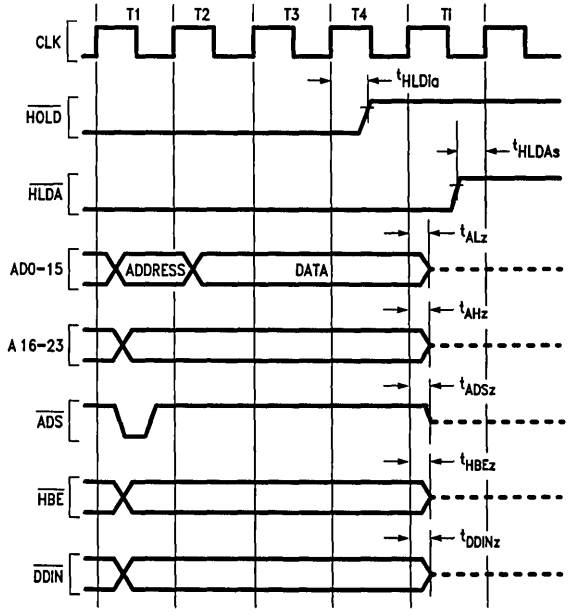
FIGURE 4-10. Direct Memory to I/O Transfer

4.0 Device Specifications (Continued)



TL/EE/8701-22

FIGURE 4-11. $\overline{\text{HOLD}}/\overline{\text{HOLDA}}$ Sequence Start



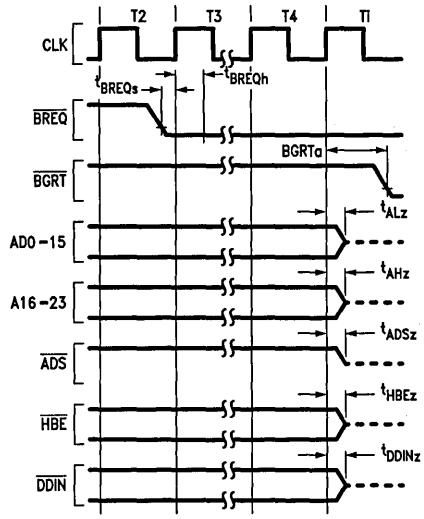
TL/EE/8701-23

FIGURE 4-12. $\overline{\text{HOLD}}/\overline{\text{HOLDA}}$ Sequence End

Note 1: DMAC in local configuration.

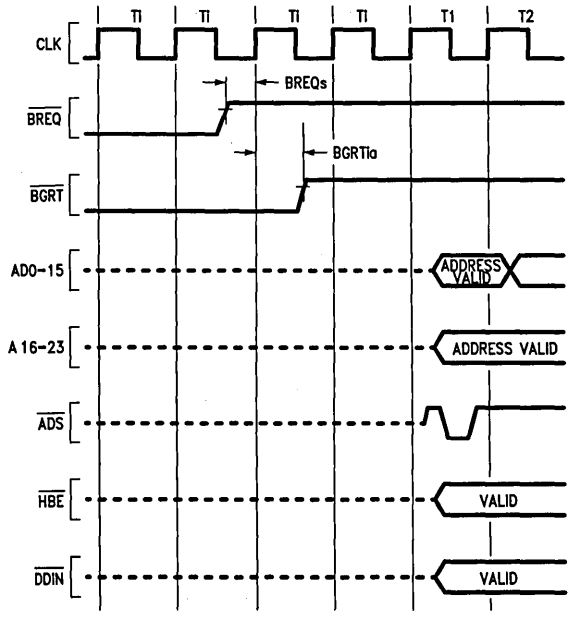
Note 2: The $\overline{\text{HOLD}}/\overline{\text{HOLDA}}$ sequence shown above is related to the single transfer mode. In burst transfer mode $\overline{\text{HOLD}}$ is deactivated two cycles later.

4.0 Device Specifications (Continued)



TL/EE/8701-24

FIGURE 4-13. Bus Request/Grant Sequence Start



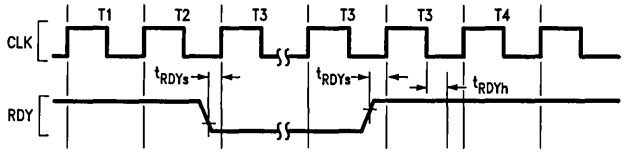
TL/EE/8701-25

FIGURE 4-14. Bus Request/Grant Sequence End

Note 1: DMAC in remote configuration.

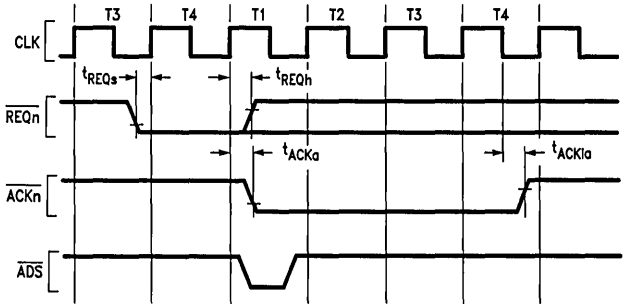
Note 2: If $\overline{\text{BREQ}}$ is asserted in the middle of a DMAC transfer, the transfer will always be completed.

4.0 Device Specifications (Continued)



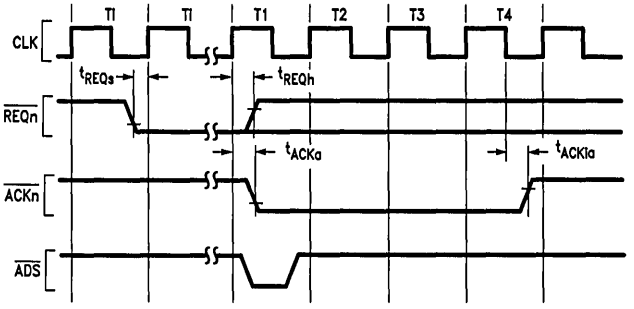
TL/EE/8701-26

FIGURE 4-15. Ready Sampling



TL/EE/8701-27

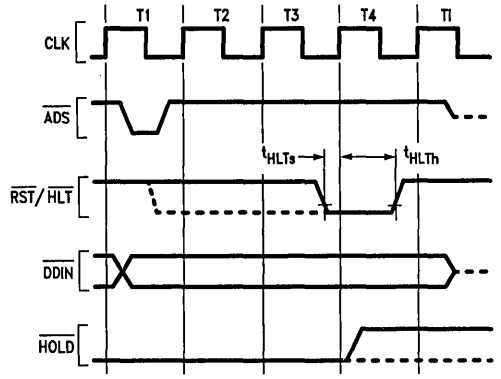
FIGURE 4-16. REQn/ACKn Sequence (DMAC Initially Not Idle)



TL/EE/8701-28

FIGURE 4-17. REQn/ACKn Sequence (DMAC Initially Idle)

4.0 Device Specifications (Continued)

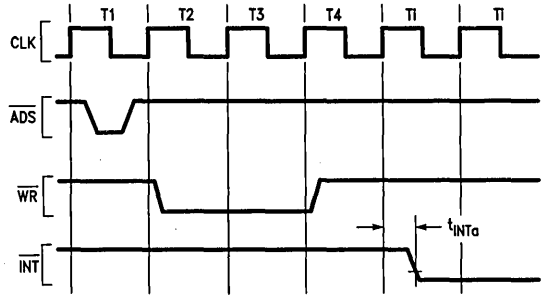


TL/EE/8701-29

FIGURE 4-18. Halted Cycle

Note 1: Halt may occur in previous T-States. It must be applied for 1 or 2 clock cycles.

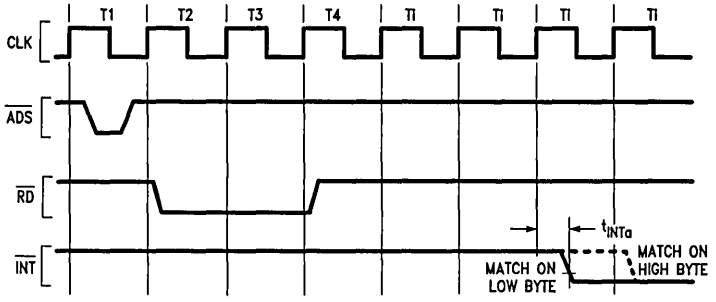
Note 2: If BREQ is asserted in the middle of a DMAC transfer, the transfer will always be completed.



TL/EE/8701-30

FIGURE 4-19. Interrupt on Transfer Complete

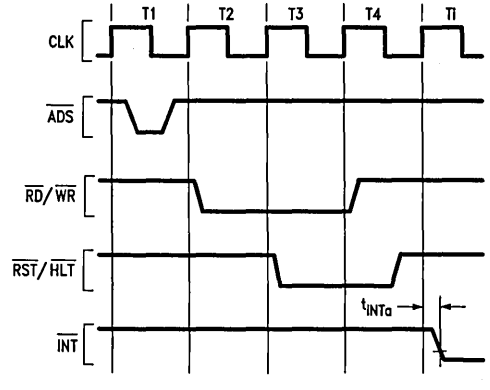
4.0 Device Specifications (Continued)



TL/EE/8701-31

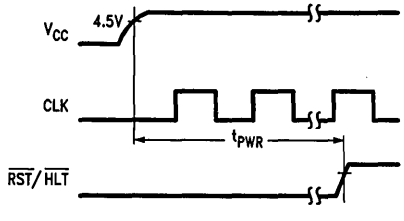
FIGURE 4-20. Interrupt on Match/No Match

Note: If inclusive search is specified a write cycle is performed before $\overline{INT}$ is activated.



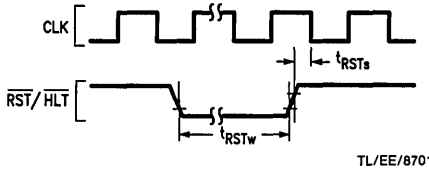
TL/EE/8701-32

FIGURE 4-21. Interrupt on Halt



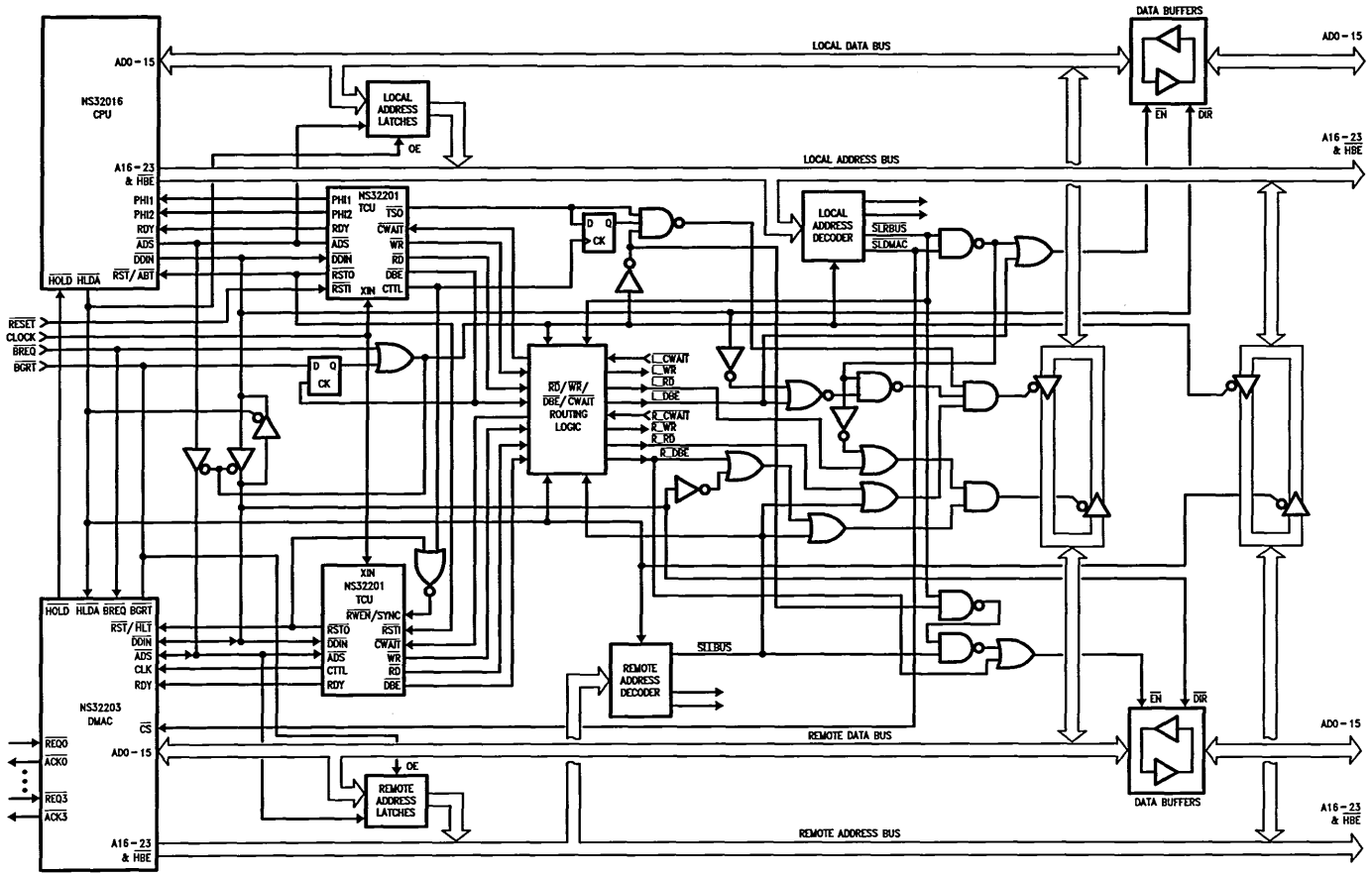
TL/EE/8701-33

FIGURE 4-22. Power on Reset



TL/EE/8701-34

FIGURE 4-23. Non Power on Reset



TL/EE/8701-35

FIGURE A-1. NS32203 Interconnections in Remote Configuration.

Note: This logic does not support direct (flyby) DMAC transfers.

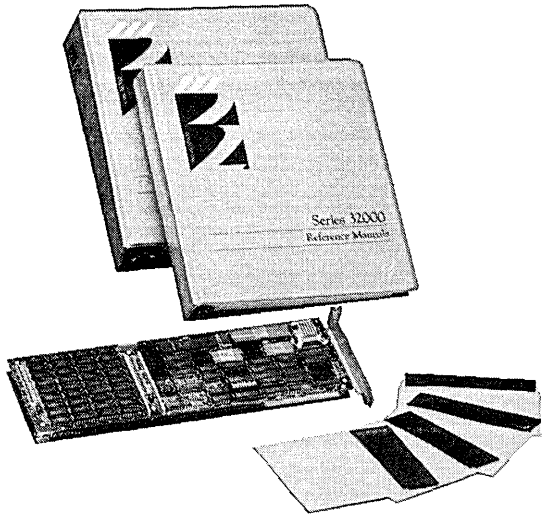
Section 5

Development Systems and Software Tools

Section 5 Contents

National Semiconductor

SYS32/30 PC Add-In Development Package



TL/EE/9420–1

- **15 MHz NS32332/NS32382 Add-In board for an IBM® PC/AT® or compatible system**
- **2–3 MIP system performance**
- **No wait-state, on-board memory in 4-, 8- or 16-Mbyte configurations**
- **Operating system derived from AT&T's UNIX® System V Release 3**
- **Multi-user support**
- **GENIX™ Native and Cross-Support (GNX™) language tools. Includes— assembler, linker, libraries, debuggers**
- **Support for other Series 32000® development products:**
  - **— SPLICE**
  - **— National's Series 32000 Development Board family**
  - **— Optimizing Compilers: C, FORTRAN 77, Pascal**
- **Easy-to-use DOS/UNIX interface**

Product Overview

The SYS32™/30 is a complete, high-performance development package that converts an IBM PC/AT or compatible computer into a powerful multi-user system for developing applications that use National Semiconductor Embedded System Processors™ or Series 32000 microprocessor family components. The SYS32/30 add-in processor board containing the Series 32000 device cluster with the NS32332 microprocessor allows programs to run on a personal computer at speeds greater than those of a VAX™ 11/780. The chip cluster on the processor board includes the NS32332 Central Processing Unit, NS32382 Memory Management Unit, NS32C201 Timing Control Unit and the NS32081 Floating-Point Unit.

Along with the processor board, the SYS32/30 package contains the Opus5™ operating system which is derived from GENIX V.3, National Semiconductor's

Product Overview (Continued)

port of AT&T's UNIX System V Release 3. Specially developed software is included to efficiently integrate the NS32332 processor board and the host PC/AT processor, allowing them to function as a complete UNIX computer system. National's Series 32000 GENIX Native and Cross-Support (GNX) language tools are included in the SYS32/30 package to provide stable and effective tools for software development. Optional compilers are available for FORTRAN 77, C, and Pascal.

Functional Description

15 MHz ADD-IN PROCESSOR BOARD FOR AN IBM PC/AT OR COMPATIBLE SYSTEM

The SYS32/30 development package contains a processor board designed around the Series 32000 chip set. This chip set includes the NS32332 Central Processing Unit, NS32382 Memory Management Unit, NS32C201 Timing Control Unit, and the NS32081 Floating-Point Unit.

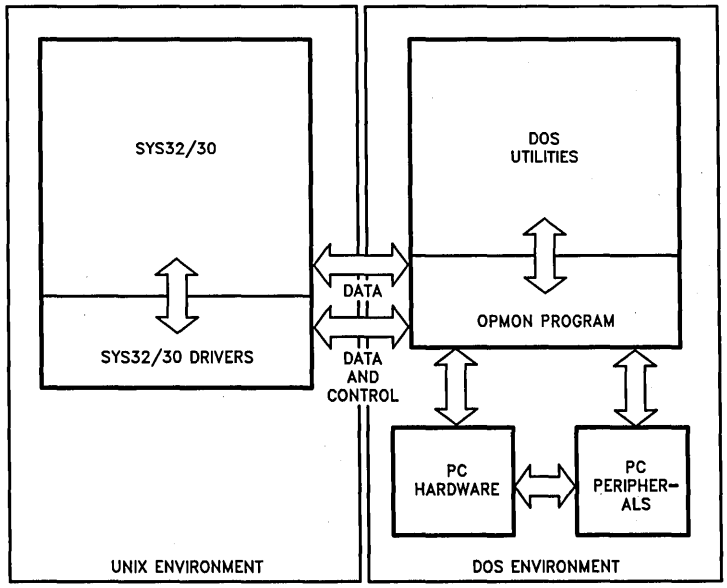
This processor board forms the high-performance center of the computer system with the host PC/AT processor. Peripherals are under the control of the PC/AT's microprocessor and are located either on the PC/AT motherboard or on other boards in the PC/AT chassis. The PC/AT handles all direct access to devices and serves as an integral dedicated I/O processor.

The SYS32/30 processor board plugs into the PC/AT bus, uses the standard control and data signals, and appears to the PC/AT as 16 bytes in the PC/AT Input/Output (I/O) space. Communication between the PC/AT and the board is accomplished via this address space. This architecture allows the board to interface to the PC/AT in the same manner as any other PC/AT peripheral. The PC/AT processes I/O commands while the SYS32/30 processor board continues with regular operation. I/O is requested via interrupt to the PC/AT, which then performs the data transfer using Direct Memory Access (DMA). (See *Figure 1*).

The processor board requires two slots in the PC/AT motherboard and plugs into a single long 16-bit bus slot. The space of the second slot is needed to accommodate the piggybacked memory board attached to the processor board. No additional connections are required.

2–3 MIPS SYSTEM PERFORMANCE

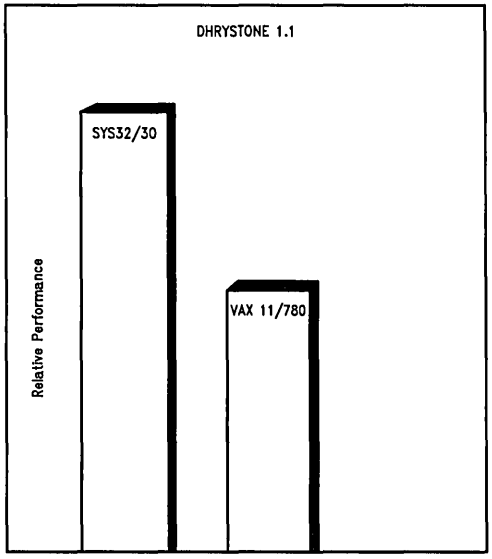
The NS32332 CPU and associated devices operating at 15 MHz provide computing power greater than that of a VAX 11/780. Sustained performance for the NS32332 device cluster is 2–3 VAX MIPS (Million Instructions Per Second). An example of relative performance using the widely recognized Dhrystone benchmark is shown in *Figure 2*.



TL/EE/9420–2

FIGURE 1

Functional Description (Continued)



TL/EE/9420–3

FIGURE 2. SYS32/30 Dhrystone Program Compiled with GNX Version 3 C Compiler VAX 11/780 Dhrystone Data Obtained from USENET

ON-BOARD MEMORY CONFIGURATIONS OF 4, 8 OR 16 MBYTES

The processor board is configured with either 4, 8, or 16 Mbytes of zero wait-state physical memory. It is possible to upgrade the 4- or 8-Mbyte configuration to 16 Mbytes through the purchase of an optional 16-Mbyte memory card.

OPERATING SYSTEM

The SYS32/30 operating system is derived from GENIX V.3, National Semiconductor's port of AT&T's UNIX System V Release 3.

The UNIX operating system is a powerful, multi-user, multitasking operating system that includes the following key features:

Demand-Paged Virtual Memory
Hierarchical file system
Source Code Control System (SCCS)
UNIX to UNIX copy (uucp)
"make" utility
Menu-driven system administration

The UNIX operating system has a proven reputation as an effective and productive environment for efficient software development. UNIX allows multiple users to work simultaneously on the same computer and project. The Source Code Control System (SCCS) automatically tracks program revisions as development work progresses. The "make" software saves valuable time in regenerating complex software systems after changes are made. The *uucp* software allows users on different UNIX systems to communicate using electronic mail and to transfer files over dial-up or serial communications links. Menu-driven system administration is available for system setup, adding users, controlling communication lines, installing software packages, changing passwords, and other administrative functions.

ADDITIONAL SUPPORT UTILITIES

Many of the popular utilities from the Berkeley 4.3 UNIX operating system, not contained in AT&T's UNIX System V Release 3, are supplied as part of the package. These utilities are listed in Table I.

TABLE I. Bsd 4.3 Utilities

C Shell	apply	banner
bsu	chsh	clear
ctags	expand	factor
from	head	last
leave	more	primes
scrpt	strings	test
unexpand	whereis	which

The Tools for Documenters package, derived from the AT&T Documenter's Workbench™ Utility, provides the Series 32000 programmer with the tools to prepare documentation. The major components of this package are shown in Table II.

TABLE II. Tools for Documenters Utilities

Name	Description
nroff	A text formatter for line printers
troff	A text formatter for typesetters
mm	A macro package
mmt	A macro package
eqn	A troff preprocessor for typesetting mathematics on a phototypesetter
neqn	A troff preprocessor for typesetting mathematics on a terminal
tbl	A preprocessor for formatting tables
pic	A preprocessor for graphic illustrations
col	A filter to nroff for processing multicolumn text output, as from tbl

NETWORKING CAPABILITY

The SYS32/30 based development system configured to support networking using the TCP/IP protocol allows project development using multiple systems, including SYS32/30 based systems, VAX/VMS™ (using TCP/IP), SUN-3/SunOS™ and VAX/ULTRIX. The

Functional Description (Continued)

compatibility design of the GNX language tools allows software modules developed on these networked systems to be linked together on a single system for execution as one program. Networking requires that additional hardware and software be installed in the system. Third party products that enable networking are listed in the SYS32/30 configuration guide.

MANUALS

A complete manual set for the operating system and related software is included in the SYS32/30 package. This includes:

Installation instructions for the PC Add-in board
Installation instructions for software
UNIX System V.3 reference manuals and user guides
GNX Language Tools Manuals
Tools for Documenters Reference Manual
Berkeley Utilities Manual

MULTI-USER SUPPORT

The SYS32/30 operating system is an interactive, multi-user, multitasking operating system. Many activities or jobs can be performed simultaneously when serial ports are added to the host system. These additional serial ports are used for terminals, printers, modems, I/O-to-development boards, I/O-to-target hardware, or for communication with National's SPLICE debugging tool. Information about third party products that provide additional serial ports is contained in the SYS32/30 configuration guide.

GNX LANGUAGE TOOLS

The GENIX Native and Cross-Support (GNX) language tools allow the user to compile, assemble, and link user programs to create executable files. These files can then be executed and debugged on a Series 32000 development board, target system application hardware, or a 32000/UNIX-based system such as the SYS32/30.

The GNX language tools include the assembler, linker, debuggers, libraries, and the monitor software for all Series 32000 development boards in both PROM and source code form.

The Series 32000 GNX language tools are based on AT&T's Common Object File Format (COFF). Under COFF, object modules created by any of the GNX compilers or the GNX assembler may be linked to object modules of any other translator in the GNX tools. Optimizing compilers are available for C, FORTRAN 77, and Pascal.

The COFF file format also allows object modules that have been created by the GNX tools on other development hosts (VAX/VMS or VAX/ULTRIX, for example) to be linked with modules created on the SYS32/30 system. This flexibility is most valuable where non-centralized software development is desired and the systems are able to transfer or share files via a common network. Information for configuring the SYS32/30 for integration into a network is contained in the configuration guide.

Compilers are available separately as optional software to allow individual selection of the application language. The C, FORTRAN 77 and Pascal compilers are the result of National's optimizing compiler project and reflect state-of-the-art compiler technology for optimizing execution speed. For additional details about the GNX tools consult the GNX tools data sheet.

SUPPORT FOR AN INTEGRATED DEVELOPMENT ENVIRONMENT

The SYS32/30 contains the functionality and compatibility needed to utilize other tools available from National Semiconductor for developing and debugging Series 32000-based applications. These tools include the SPLICE software debugger, NS32GG16-ISE, the Series 32000 Development Board set, and National's Embedded System Processor evaluation boards for the NS32CG16 and NS32GX32 processors.

The NS32CG16 ISE is a full featured emulator for development of NS32CG16 based systems. Software is developed on the SYS32/30, then transferred to the DOS partition of the development system for download by the ISE.

The SPLICE development tool provides a communication link between a Series 32000 target and a development system host. This connection allows users to download and map their software onto target memory and then debug this software using National Semiconductor's GNX debugger. Consult the SPLICE data sheet for more information.

The GNX debugger also directly supports the Hewlett-Packard HP64772 NS32532/NS32GX32 in-system emulator. This combination provides powerful integrated support for high-level source debugging and in-system emulation of the NS32532 or NS32GX32 processors.

The Series 32000 development boards and Embedded System Processor evaluation boards used with the SYS32/30 are specifically designed to assist the user in evaluating and developing hardware and software for embedded systems and the Series 32000 family of CPUs.

Functional Description (Continued)

DOS/UNIX INTEGRATION

The SYS32/30 PC add-in development package allows easy transfer of data between DOS and the UNIX operating system. A system console user can switch between either operating system using only a few keystrokes. A shell interface allows DOS commands to be executed from the UNIX shell, UNIX commands to be executed from DOS, and files to be transferred between the UNIX and DOS partitions on the system disk. In addition, the user can suspend the SYS32/30 operation, enter DOS, run an application, and then return to the SYS32/30 environment.

Series 32000 Application Development

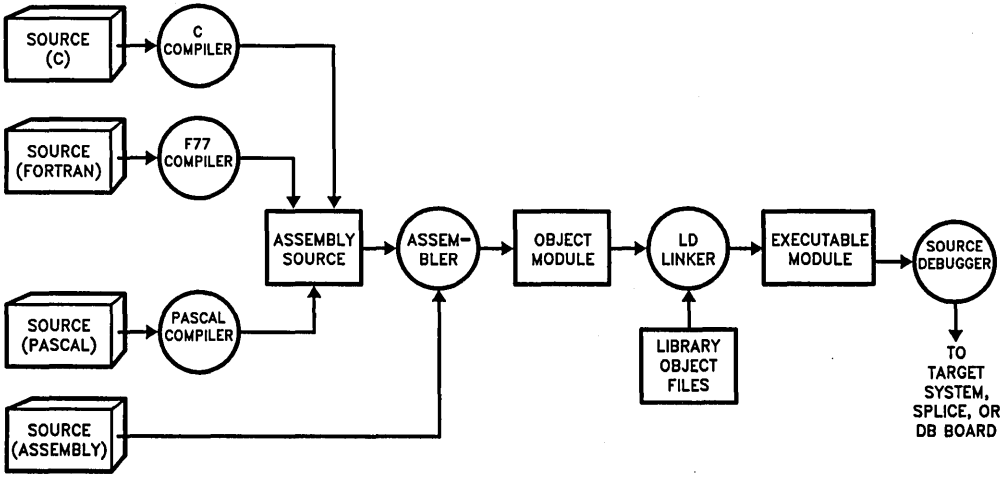
The SYS32/30 with the PC/AT operates as a local host computer system for integrating application software into target prototype boards containing Series 32000 components. Programs can be written in assembly language or in a higher level language. Optional compilers are available for C, FORTRAN 77, and Pascal.

During compilation, the compilers generate assembly code which is assembled by the GNX assembler. (See *Figure 3.*) The output of the assembler is an object file which can be linked to other object file and/or libraries, resulting in an executable file.

Since the SYS32/30 provides a Series 32000 native environment, the executable file may be run on the host SYS32/30 system or loaded into RAM on either a target system, an Embedded System Processor evaluation board or one of the Series 32000 development boards. The source-level software debuggers in the GNX tools provide powerful facilities for debugging software on the target system.

The GNX debugger is capable of downloading and controlling the execution of software on the target system. Executable monitor software is provided in PROMs in the SYS32/30 package for the Series 32000 development boards and the Embedded System Processor evaluation boards. Monitor software is also provided in source form in the GNX language tools so application designers can modify and port the monitor to suit the needs of their target system.

After debugging, the executable file created by linking can also be converted to PROM format using the GNX *nburn* utility.



TL/EE/9420–4

FIGURE 3

Configuring a System

The SYS32/30 PC Add-In package supports a variety of configurations. Based on developer needs, the final configuration may need extra serial I/O ports, and/or networking capability. A hard disk of sufficient size is also an important part of the configuration. A configuration guide that outlines available options and recommended products for configuring the SYS32/30 development system is available.

Host system elements required for SYS32/30 operation are:

— IBM PC/AT or compatible system
— Two full length slots in the motherboard
— 512 Kbytes of RAM
— PC-DOS 3.1 or later
— 1.2-Mbyte floppy disk drive
— Adequate hard disk storage (see the next section on disk size)

Note: The SYS32/30 processor board actually plugs into a single slot. The second slot is required to accommodate the space taken by the piggybacked memory board attached to the NS32332 processor board.

The SYS32/30 PC/AT Add-In Development Package runs on an IBM PC/AT or compatible computer. If an IBM PC/AT is not used for the host system, it is important to remember that compatibility can vary between IBM PC/AT compatible systems. The SYS32/30 processor board may not be adequately supported by systems that lack full IBM PC/AT compatibility. The configuration guide available contains a list of IBM PC/AT compatible systems that have the required compatibility.

HARD DISK CAPACITY

Several factors influence the size selected for a hard disk. Consideration should include the number of users for the system, space for user files, the size of the application to be developed, and extra software packages and compilers that must reside on the system.

For example, a 50-Mbyte hard disk is the minimum size recommended for a SYS32/30-based development environment. This provides sufficient space for a single-user account, the UNIX operating system and utilities, the GNX tools, compiler software, basic DOS software, and a moderate size application. Disk drives with even greater capacity than the minimum sizes indicated here should be considered for additional users or software and to provide for growth of the system.

When selecting hard disk drives or other peripheral devices, it is important that the device conform to the industry-standard for peripheral devices designed for use on the PC/AT bus.

Basic Kits

The SYS32/30 Add-In Development package is available in three basic kits:

NSS-SYS30-KIT1	For IBM-AT and compatible systems PC Add-In coprocessor board with 4 Mbytes on-board memory UNIX System V.3 based operating system GNX Language Tools Tools for Documenters Berkeley Utilities Installation instructions for the PC Add-In board Installation instructions for software UNIX System V.3 reference manuals and user guides GNX Language Tools Manuals Tools For Documenters Reference Manuals Berkeley Utilities Manual
NSS-SYS30-KIT2	Same as KIT1 except with 8 Mbytes of on-board memory
NSS-SYS30-KIT3	Same as KIT1 except with 16 Mbytes of on-board memory

MEMORY UPGRADE

To upgrade the memory size to 16 Mbytes after the purchase of KIT1 or KIT2, the following 16-Mbyte memory board must be purchased to replace the existing memory board:

NSS-SYS30-MEM16	16-Mbyte memory board.

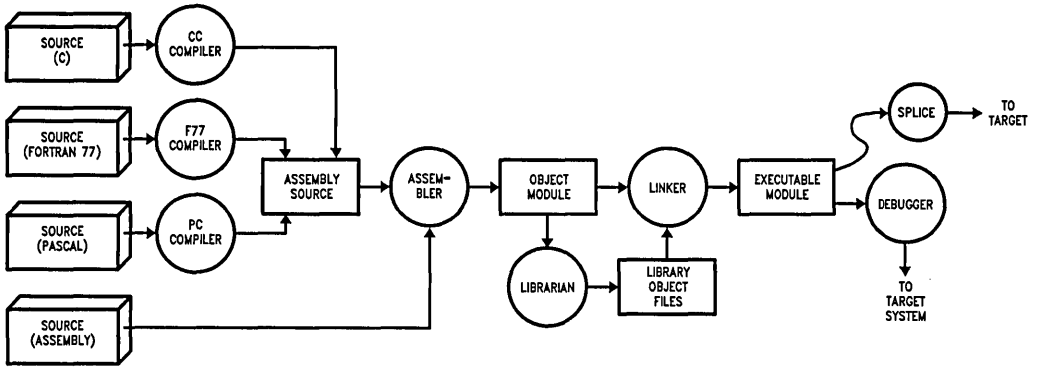
Optional Software Packages

(A prerequisite for use is the purchase of one of the above basic kits).

NSW-C-3-BHBF3	Optimizing C Compiler
NSW-F77-3-BHBF3	Optimizing FORTRAN 77 Compiler
NSW-PAS-3-BHBF3	Optimizing Pascal Compiler
NSW-NET-BHBF3	Networking software
NSP-SYS32/V3-MS	Additional operating system manual set

National Semiconductor

Series 32000® GENIX™ Native and Cross-Support (GNX) Development Tools (Version 3)



TL/EE/10418–1

- **Complete software development environment for Series 32000**
- **Supports software development on VAX™, Sun-3®, and SYS32™ development hosts**
- **Supports Common Object File Format (COFF)**
- **Includes versatile configuration definition utility**
- **Includes source code for board-level monitors**
- **Includes complete floating-point unit emulation software**
- **Supports optional C, FORTRAN 77, and Pascal optimizing compilers**
- **Supports SPLICE development tool**

Introduction

The Series 32000 GNX-Version 3 (GENIX Native and Cross-Support) development tools consist of assembler, linker, debuggers, monitors, basic I/O routines, libraries, optional high-level language compilers, and other tools to aid in the development of applications for the Series 32000 microprocessor family. The GNX tools allow users to compile, assemble, and link application programs to create executable files. These files can then be executed and debugged on Series 32000-based development hosts, such as the SYS32/20 and SYS32/30, or on a Series 32000-based target board. After debugging, the executable files can be converted to binary/hexedecimal files suitable as input to PROM programmers for burning PROMs.

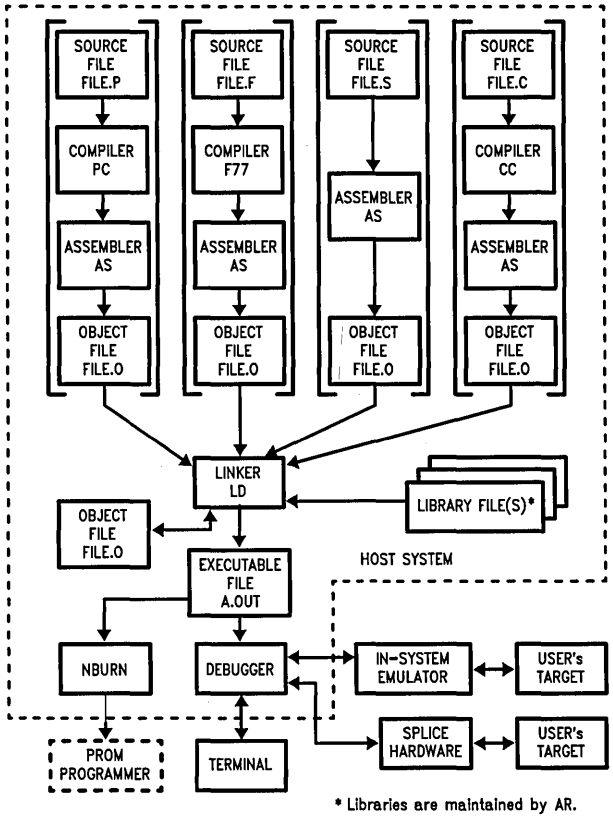
The Series 32000 GNX development tools are based on the Common Object File Format (COFF), as developed by AT&T and enhanced by National Semiconductor Corporation. This allows files developed on different hosts and in different high-level languages to be easily integrated.

Supported Development Hosts

The Series 32000 GNX development tools are available hosted for cross-development on the VAX se-

5

Supported Development Hosts (Continued)



TL/EE/10418-2

FIGURE 1. Sample Development Process

TABLE I. Commands for SYS32, VAX/UNIX, and VAX/VMS

SYS32	VAX/UNIX	VAX/VMS
ar	nar	nar
as	nasm	nasm
cc	nmcc	nmcc
	ncmp	ncmp
dbg32	dbg32	dbg32
f77	nf77	nf77
gts	gts	gts
ld	nmeld	nmeld
lorder	nlorder	
monfix	monfix	monfix
nburn	nburn	nburn
nm	nnm	nnm
pc	nmpc	nmpc
size	nsize	nsize
strip	nstrip	nstrip

ries of computers, running the VMS™, UNIX® (bsd), and ULTRIX operating systems and on a Sun-3 workstation running SunOS™. Also supported are National Semiconductor's SYS32/20 and SYS32/30 development environments. Table I summarizes the GNX commands for each environment.

The SYS32/20 and SYS32/30 PC-Add-In Development Packages are complete, high-performance packages that convert an IBM-PC/AT™ or compatible computer into a powerful multi-user system for developing applications that use the Series 32000 family. The SYS32 systems are based on the Series 32000 processor family; the SYS32/20 includes an NS32032 Central Processing Unit, and the SYS32/30 is based on the NS32332 CPU. Both the SYS32/20 and SYS32/30 run a derivative of the AT&T System V.3 UNIX operating system. Because these host systems are themselves based on the Series 32000 processor family, application code can be debugged on the host system without down-loading to target hardware.

Figure 1 illustrates a typical development process.

Tools Components

The GNX Development Tools comprise the following utilities and support libraries:

Ar

This utility maintains groups of files combined into a single archive file. **Ar** is used to create and update library files as used by the GNX linker **ld.**

As

The GNX assembler, **as**, assembles Series 32000 assembly language source programs and generates relocatable object modules. Relocatable object modules must be linked to create executable load modules.

DBG32

DBG32 is an interactive symbolic debugger. It can be used for remote debugging in conjunction with a host and any target hardware that includes a Series 32000 GNX monitor. DBG32 allows source-level debugging and includes an easy-to-use on-line help facility.

Floating-Point Enhancement and Emulation (FPEE) Library

When a floating-point unit (FPU) is not present, the floating-point enhancement and emulation (FPEE) library provides low-cost floating-point support by emulating the Series 32000 FPU instructions. When an FPU is present, FPEE enhances the FPU by providing additional functionality as recommended by Draft 10 of the ANSI/IEEE Task 754 Proposal for Binary Floating-Point Arithmetic (IEEE 754). FPEE meets the IEEE 754 standard for double-precision arithmetic.

The FPEE library is provided in source form and as a binary library suitable for its particular GNX tool-set environment. The source includes all support routines necessary to build the FPEE library. The FPEE library can be configured to enhance/emulate either the NS32081 FPU or the NS32381 FPU.

GNX Target Setup (GTS)

The GNX tools support the full line of Series 32000 central processing units and peripheral devices, based on user-defined parameters. The GNX Target Setup (GTS) utility allows users to easily define the characteristics of the target system at one time. This information is saved in a file on the host system, which is examined each time a GNX utility is invoked. These parameters are used to tailor the application code to characteristics of the particular hardware.

GTS operates both interactively and non-interactively and includes an easy-to-use interface and on-line help facility.

Ld

The GNX linker, **ld**, creates executable files by combining object files, providing relocation, and resolving external references. The linker also processes symbolic debugging information. The linker includes a powerful directives language, which allows the user to precisely control the linking process.

Lorder

Lorder finds ordering relations for object libraries. The input may be one or more object or library archive (see **ar**) files. The output of **lorder** can be processed to find an ordering of a library suitable for one-pass access by the linker.

Math Libraries

The math libraries (libm.a and lib381m.a) contain standard math functions that support both the NS32081 and NS32381 floating-point units. These functions are highly optimized for the Series 32000 architecture.

Table II contains a list of the available math functions.

TABLE II. Available Math Functions

acos	exp	fdrem	fmod	fpow	log1p
acosh	exp2	fexp	fneg	fpstrpvctr	log2
asin	expm1	fexp2	fp—gmathenv	frelation	neg
asinh	fabs	fexpm1	fp—getexptn	frem	nextdouble
atan	facos	ffabs	fp—getround	frint	nextfloat
atan2	facosh	ffinite	fp—gettrap	fsin	pi
atanh	fasin	ffloor	fp—procentry	fsinh	pow
bessel	fasinh	ffmod	fp—procexit	fsqrt	randomx
cabs	fatan	fhypot	fp—smathenv	ftan	relation
cbrt	fcabs	finf	fp—setexptn	ftan2	rem
ceil	fcbrt	finite	fp—setround	ftanh	rint
compound	fceil	flog	fp—settrap	gamma	sin
copysign	fcompound	flog10	fp—testrap	hypot	sinh
cos	fcopysign	flog1p	fp—tstexptn	inf	sqrt
cosh	fcos	flog2	fpgtrpvctrv	log	tan
drem	fcosh	floor	fpi	log10	tanh

Note: All math library functions are provided in single and double precision versions.

Tools Components (Continued)

Monitors

Mon16, mon32, mon332, mon332b, mon532 and mon32GX are PROM-based firmware monitors for use on a Series 32000-based development board. The monitors allow the user to load, execute, and debug development board programs with the **dbg32** debugger running on a host computer system. The monitors also provide run-time services, such as physical I/O, interrupt handling, and error handling in the form of supervisor calls.

Source to each monitor is provided so that it may be modified, assembled, linked, and installed on other Series-32000 based target boards.

Monfix

Monfix is a utility that creates a Series 32000 bootstrap program by modifying a Series 32000 GNX executable file.

Nburn

Nburn loads the specified bytes of a file to an EPROM burner in one of several user-specified formats, including ASCII-HEX and S-record.

Nm

The **nm** utility displays the symbol table of a Series 32000 GNX object file.

Size

The **size** utility displays size information for each section and optional header information of a Series 32000 GNX object file.

Strip

The **strip** utility strips symbol and line number information from a Series 32000 GNX object file.

Optional Compilers

A substantial amount of application code is developed in a high-level language; therefore, the speed and efficiency of the application are functions not only of processor speed, but also of quality of code generated by the high-level language compiler. An inefficient compiler can extract a significant performance penalty. Likewise, a significant performance improvement can be achieved for a much lower cost in software rather than hardware. For this reason. National Semiconductor has developed a line of optimizing compilers that generate extremely efficient code for the Series 32000 architecture.

Each of the optimizing compilers includes the state-of-the-art GNX optimizer, based on advanced optimization theory developed over the past 15 years. In addition, because all GNX-Version 3 optimizing compilers use a standard calling sequence, internal intermediate representation, and object file format, mixed-language programming is greatly simplified, aiding in the porting of existing applications to the Series 32000 architecture.

C Optimizing Compiler

The GNX-Version 3 C Optimizing Compiler fully implements the C programming language, as defined in *The C Programming Language* by B. Kernighan and D. Ritchie. The C Optimizing Compiler is also compatible with the UNIX System V C compiler, derived from the portable C compiler (pcc). Several features of the draft ANSI C standard (X3J11) are supported.

FORTRAN 77 Optimizing Compiler

The GNX-Version 3 FORTRAN 77 Optimizing Compiler fully implements the FORTRAN 77 programming language, as defined by the American Standard publication *Programming Language FORTRAN (ANSI X3.9-1978).* In addition, a command-line option is provided that forces the compiler to accept as input only programs that adhere to the FORTRAN 66 standard.

Pascal Optimizing Compiler

The GNX-Version 3 Pascal Optimizing Compiler fully implements the Pascal programming language, as defined by the International Standards Organization (ISO) standard **ISO dp7185 level 1**. Several useful extensions to the Pascal language are supported. A command-line option is provided that forces the compiler to accept as input only programs that adhere to the ISO standard.

SPLICE Support

The GNX development tools enable the use of the SPLICE development tool, which can be used to debug software/hardware on a Series 32000 target. SPLICE provides a communication link between a Series 32000 target and a development system host that allows users to down-load and map their software onto target memory and debug this software using the dbg32 debugger. The monitor resident on the SPLICE communicates with dbg32 on the development host.

Source Products

The GNX development tools, as well as the optional optimizing compilers, are available in source form for use in porting to other potential development environments. Source code is provided on a VAX/UNIX bsd tape. Contact Series 32000 Marketing for more information regarding GNX source availability.

Licensing

All binary versions of the Series 32000 GNX development tools require the execution of National Semiconductor's binary user agreement. Because the GNX development tools contain AT&T proprietary code, a System V source license is prerequisite for obtaining a source version of the GNX tools. Contact Series 32000 Marketing for more information regarding specific licensing issues.

Customer Support

National Semiconductor offers a full 90-day warranty period. Extended warranty provisions can be arranged by calling National Semiconductor's Technical Support Engineering Center at the toll-free number listed below.

National Semiconductor's Technical Support Engineering Center has highly trained technical specialists available to assist customers over the telephone with any product-related technical problems.

For more information, please call (800) 759-0105 (in the United States and Canada). Outside North America, please contact your local National Semiconductor office.

Ordering Information

Supported Host Environments and Order Codes:

SYS32/20:
NSW-ASM-3-BHAF3 (included with SYS32/20 kit)

SYS32/30:
NSW-ASM-3-BHBF3 (included with SYS32/30 kit)

VAX/VMS:
NSW-ASM-3-BRVM

VAX/ULTRIX (UNIX bsd):
NSW-ASM-3-BRVX

Micro VAX/VMS:
NSW-ASM-3-BCVM

Micro VAX/ULTRIX:
NSW-ASM-3-BCVX

Sun-3:
NSW-ASM-3-BCSX

Each software package is delivered with one copy of each appropriate manual. Additional manual sets may be ordered using the following order codes:

NSP-ASM-NX3-MS:
Manual set included with NSW-ASM-3-BHAF3 and NSW-ASM-3-BHBF3

NSP-ASM-X3-MS:
Manual set included with NSW-ASM-3-BRVX, NSW-ASM-3-BCVX, and NSW-ASM-3-BCSX

NSP-ASM-M3-MS:
Manual set included with NSW-ASM-3-BRVM and NSW-ASM-3-BCVM

NSP-C-V3-M:
Manual set delivered with Optimizing C compiler (all hosts)

NSP-F77-V3-M:
Manual set delivered with Optimizing FORTRAN 77 compiler (all hosts)

NSP-PAS-V3-M:
Manual set delivered with Optimizing Pascal compiler (all hosts)

For further information regarding National Semiconductor's software development tools and development hosts, please refer to the following datasheets:

GNX-Version 3 C Optimizing Compiler

GNX-Version 3 FORTRAN 77 Optimizing Compiler

GNX-Version 3 Pascal Optimizing Compiler

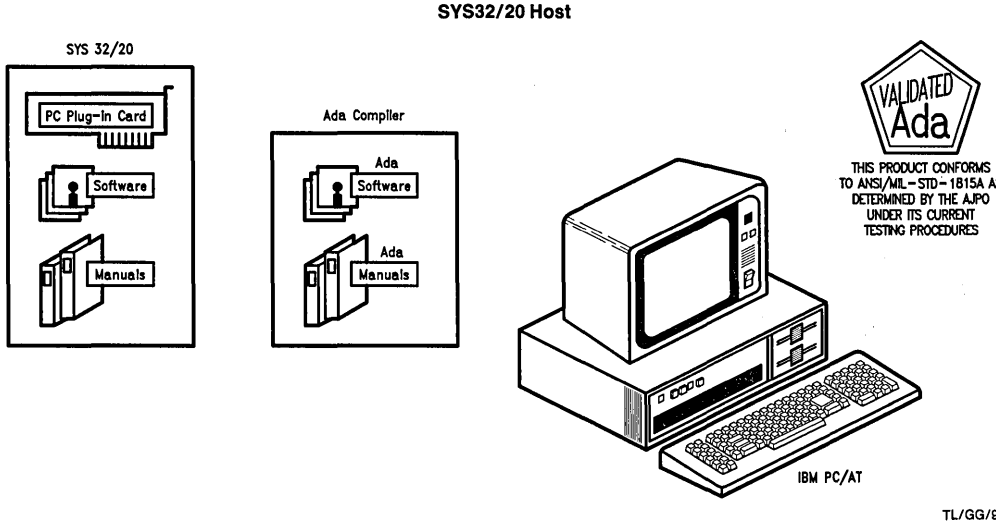
SYS32/20 PC-Add-In Development Package

SYS32/30 PC-Add-In Development Package

SPLICE Development Tool

National Semiconductor

Series 32000®
Ada Cross-Development System for SYS32™/20 Host



TL/GG/9307-2

- **Series 32000 cross-support development environment for SYS32/20**
- **Validated under 1.8 ACVC**
- **Derived from the VERDIX™ Ada Development System (VADS™)**
- **Compiler support for Ada Pragmas and Representation Attributes**
- **Comprehensive Support Services available from National**
- **Generates GNX™ Common Object File Format (COFF)**
- **Debugging Tools**
- **Program Generation Utilities**
- **SPLICE support**
- **Extensive Ada Library Management Utilities**
- **Run-time system to support bare-board environment**
- **Ada VRTX® Interface Package (Optional)**
- **Source to Ada Run-Time System (Optional)**

Product Overview

The Series 32000 Ada cross-compiler supports full Ada language program development on National's SYS32/20 host and is part of National's Validated Ada Development Environment (NVADE). NVADE provides a high performance Ada compiler that supports all required features of the Ada language and is fully compliant with ANSI/MIL-STD-1815A. NVADE also provides a comprehensive set of tools specifically tailored to provide the optimum Ada Programming Support Environment (APSE) for a host of application development.

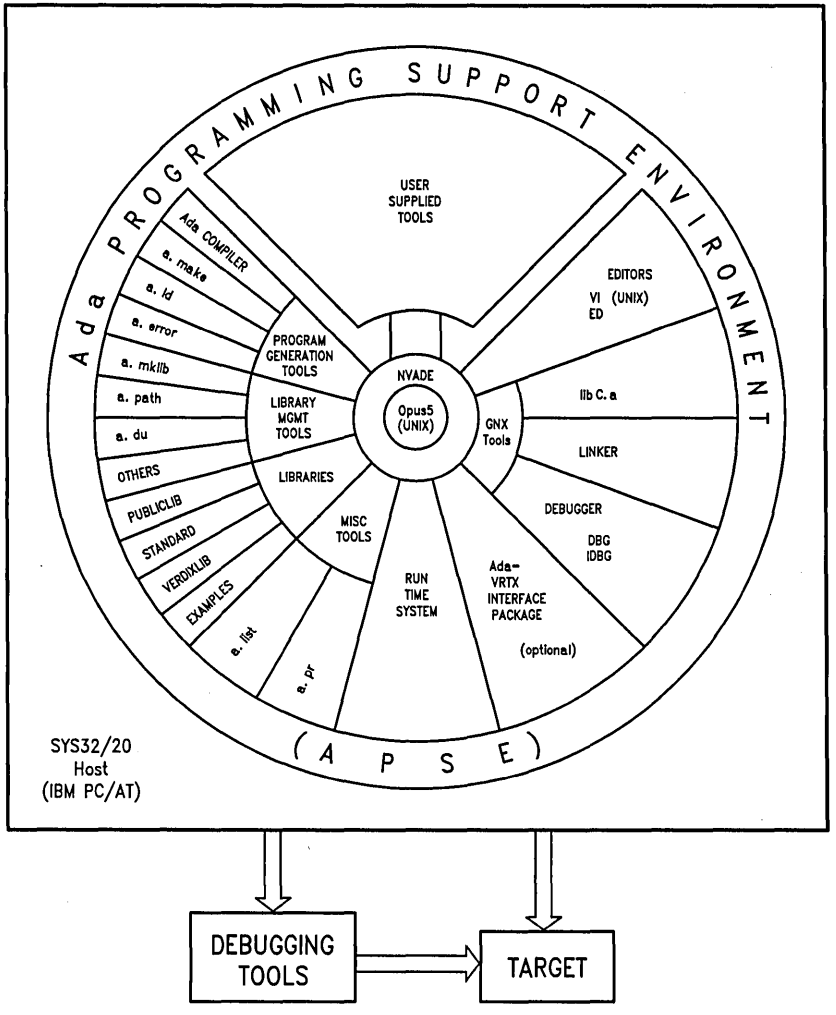
Product Overview (Continued)

The SYS32/20 Development system includes a high-performance add-in card that converts an IBM-PC/AT or compatible system into a Series 32000-based development environment.

Once compiled, the Ada program will execute on either a Series 32000 development board or a customer target board. This "production quality" Ada compiler focuses on high performance, and is intended for large-scale development of real-time, embedded control, or training simulator software applications. The Series 32000 Ada Cross-Development System includes the Ada compiler, program library utilities, program generation utilities, library management and a complete run-time system. This product directly interfaces with GNX language tools provided with the SYS32/20 system, including GNX linker, DBG and IDBG debuggers, library management tools and other utility programs.

The Series 32000 Ada Cross-Development System has been engineered and designed to run under OPUS5, the SYS32/20 Operating System derived from AT&T's UNIX™ System V. Therefore, rather than learning a new operating system, the programmer can immediately concentrate on Ada program development. To aid the user, complete on-line manual entries are provided. These can be configured to use either the UNIX man utility or a separate interactive help command, supplied with the product.

Series 32000 Ada Cross-Development System for SYS32/20 Host



TL/GG/9307-3

NVADE Components

Ada Compiler

The Ada Compiler accepts as input Ada source and generates Series 32000 code that can be downloaded to, and executed on, a Series 32000-based target development board.

The Series 32000 Ada Compiler supports the full Ada language. Features include shared or unshared generics, separates, in-lines, bit representation, machine-code insertion, monitor tasks and terminal I/O. The compiler generates GNX COFF (Common Object File Format) object files that can be linked with object files generated by other GNX compilers. The Ada compiler performs several optimizations, including value-tracking global register allocation, register assignment for commons and locals, common sub-expression removal, branch and dead code analysis, some constraint check removal, and local peephole optimizations. The Ada compiler operates as a re-entrant shareable process in the SYS32/20 host system, allowing the compiler to make full use of most operating system facilities.

In addition, the Ada compiler provides features to aid in the development of real-time, embedded control and training simulator software applications. Some of these include Ada Pragmas as specified in Chapter 13 of the Ada Language Reference Manual (LRM), such as: Inline, Interface, Interface_Object, Pack, Page, Priority, Share_Body, and Suppress. Also included is a Machine Code Package which provides an interface for handling machine code insertion and generics (Unchecked_Deallocation and Unchecked_Conversion) for controlling storage and type conversions.

Program Generation Utilities

An Ada make utility, similar in operation to that found in the UNIX operating system, is provided to simplify program compilation by maintaining program unit dependancy information. This utility determines which files must be recompiled to produce a current executable file. This utility can also be used to ensure that the named unit is up-to-date, recompiling dependencies as necessary. Also provided is a source code formatter, easily configurable for individual Ada coding standards.

Program Library Utilities

The Ada language imposes stringent requirements on an Ada Program Library. While the language provides for separate compilation of program units, each unit is compiled in the "context" of previously compiled units. The compiler must have access to this context, and the context must be carefully organized in the form of a Program Library. This library has been designed to enhance the compiler performance. A set of utilities is provided to manage, manipulate, and display Program Library information.

In addition, the Series 32000 Ada Cross-Development System permits Ada Program Libraries to be hierarchically organized, so that units not local to one library can be found in other libraries. Thus, programmers can work without interference on local versions of individual program units, while retrieving the remainder of the program from higher-level libraries.

NVADE also uses DIANA (Descriptive Intermediate Attributed Notation for Ada), which generates an intermediated representation for each unit. DIANA provides a tree-structured representation of an Ada program encoding the complete syntactic and semantic information of each individual Ada unit. The presence of DIANA as an integrated mechanism makes possible powerful editing, debugging and program query facilities, thus providing the means for simple and efficient incremental compilation.

Debuggers

The standard GNX debugger, DBG32, is used with the Series 32000 Ada Cross-Development System. DBG32 can be used to debug code on the SYS32/20 host and/or to download and remotely debug or execute code on a Series 32000 development board. DBG32 supports the use of National's SPLICE software debugging tool. Machine-level debug support is provided by the debugger.

Linker

Ada object files are linked by the standard GNX linker, which is called by the Ada compiler pre-linker. The GNX linker resolves references between object files and library routines and assigns relocated addresses to produce Series 32000 executable code.

Ada Run-Time System

The Series 32000 Ada Run-time System provides comprehensive support for tasking, debugging, exception handling and input/output.

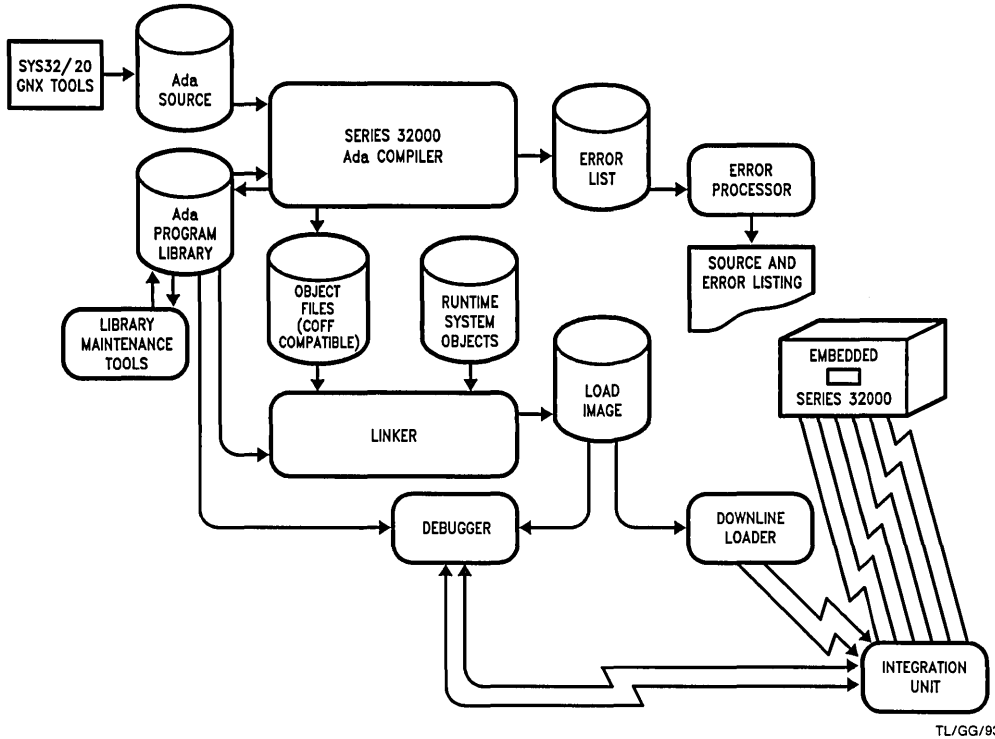
The Run-time System is linked with the user's generated Ada program. To facilitate resource utilization efficiency, major portions of the Run-time System have been optimized. Run-time source for customization is also available.

Ada-VRTX Interface Package (Optional)

The Ada Run-time System includes a large, rich, and elegant tasking system. VRTX (the Versatile Real-Time Executive) provides a small, simple, compact and fast tasking system and may be a preferred alternative to using the Ada Run-time System, particularly for embedded microprocessor applications where space and timing are critical. The Ada-VRTX interface package (AVIP) offers Ada language users a convenient means of interfacing with VRTX. AVIP allows Ada programmers the ability to call any VRTX service from their Ada program. (The exceptions are

Program Library Utilities (Continued)

Series 32000 Ada Cross-Development System for SYS32/20 Host
NVADE Modules and Run-Time Environment



TL/GG/9307–1

calls provided for the user-defined interrupt handlers and partition create and extend.) The actual operations performed by VRTX are identical in both assembly language and Ada. Thus, this package gives users both the elegant features of the Ada language and VRTX's unique tasking system.

Pre-Requisites

— SYS32/20 KIT (KIT 2 is recommended) installed on an IBM PC/AT
— DB32000, DB332-PLUS target development system board with power supply
— 60 mbyte hard disk capacity (minimum)
— IBM PC/AT with 1.2 mbyte floppy drive or IBM PC/AT with Tape Cartridge Unit
— A minimum of one available serial port

Supported Hardware/Software

— SYS32/20 HOST
— SYS32/20 Operating System (OPUS5)
— DB32000, DB332-PLUS target development system board with power supply
— In-System Emulator
— SPLICE II

Shipping Package

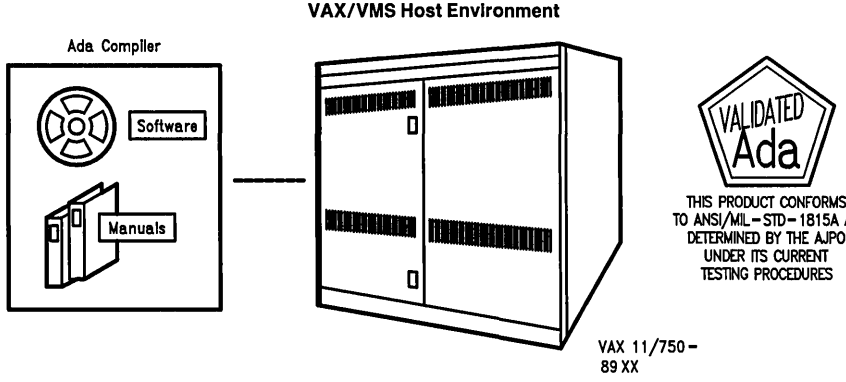
— Series 32000 Installation Instructions and Release Letter
— SYS32/20 Cartridge tape or high density floppy diskettes
— Ada Language Reference Manual (ANSI/MIL-STD 1815A)
— Ada Compiler and support tools documentation

Ordering Information

— NSW-Ada-BHAF Ada Cross-Development System, binary high density diskettes, SYS32/20
— NSW-Ada-BCAF Ada Cross-Development System, binary cartridge tape, SYS32/20
— NSW-ARTS-SHAF Ada Run-time System, source, high density diskettes, SYS32/20
— NSW-ARTS-SCAF Ada Run-time System, source, cartridge, SYS32/20
— NSW-AVIP-BHAF Ada-VRTX-Interface Package, Binary high density diskettes, SYS32/20
— NSW-AVIP-BCAF Ada-VRTX-Interface Package, Binary Cartridge tape, SYS32/20
— NSP-Ada-MS Manual set for the Ada Development System

National Semiconductor

Series 32000®
Ada Cross-Development System for VAX™/VMS™ Host



TL/GG/9364-1

- **Series 32000 cross-support development environment for VAX/VMS host**
- **Validated under 1.8 ACVC**
- **Runs under VAX/VMS 4.4 Operating Systems and future revisions of VMS**
- **Derived from the VERDIX™ Ada Development System (VADS™)**
- **Compiler Support for Ada Pragmas and Representation Attributes**
- **Comprehensive Support Services available from National**
- **Generates GNX™ Common Object File Format (COFF)**
- **Program Generation Utilities**
- **SPLICE support**
- **Extensive Ada Library Management Utilities**
- **Run-time system to support bare-board environment**
- **Debugging Tools**
- **Ada VRTX® Interface Package (Optional)**
- **Source to Ada Run-Time System (Optional)**

Product Overview

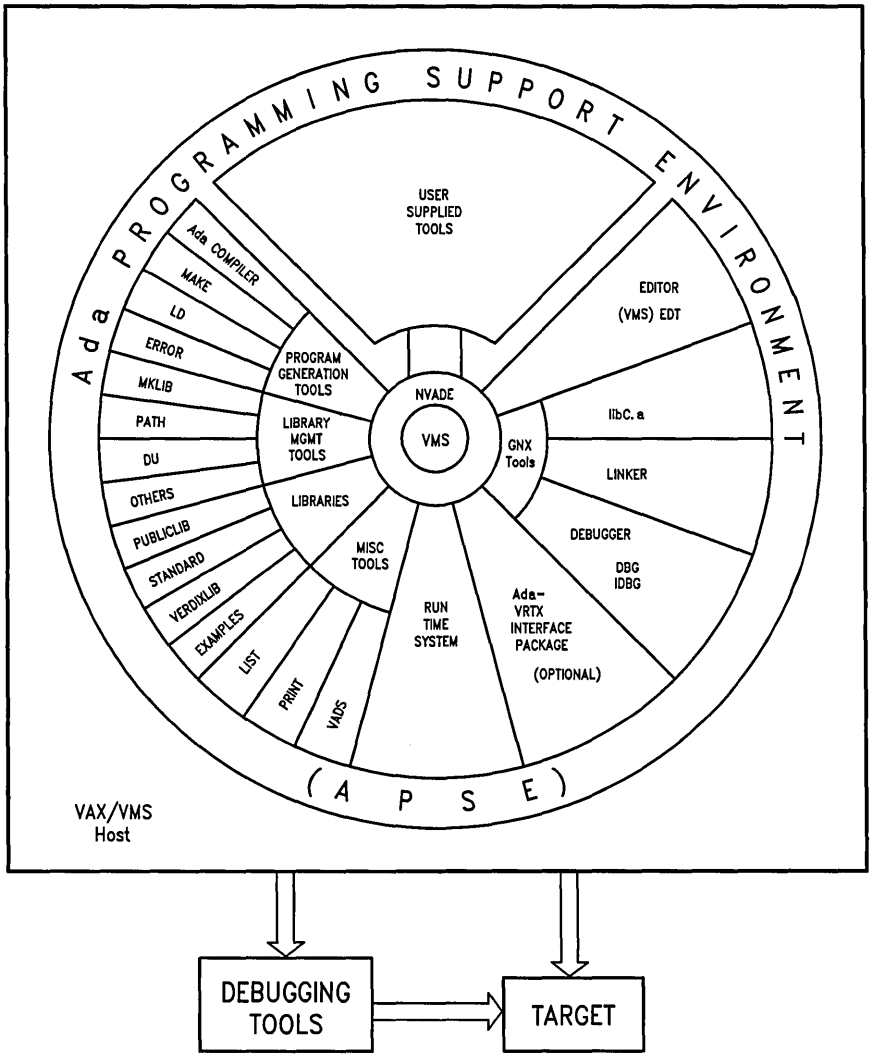
The Series 32000 Ada cross-compiler supports full Ada language program development on Digital Equipment Corporation's VAX/VMS hosts and is part of National's Validated Ada Development Environment (NVADE). NVADE provides a high performance Ada compiler that supports all required features of the Ada language and is fully compliant with ANSI/MIL-STD-1815A. NVADE also provides a comprehensive set of tools specifically tailored to provide the optimum Ada Programming Support Environment (APSE) for host application development.

Product Overview (Continued)

Once compiled, the Ada program will execute on either a Series 32000 development board or a customer target board. This "production quality" Ada compiler focuses on high performance, and is intended for large-scale development of Series 32000 real-time, embedded control, or training simulator software applications. The VAX/VMS Ada Cross-Development System includes the Ada compiler, program library utilities, program generation utilities, library management utilities and a complete run-time system. This product directly interfaces with VAX/VMS GNX language tools provided, including GNX linker, DBG and IDBG debuggers, library management tools and other utility programs.

The VAX/VMS Ada Cross-Development System has been engineered and designed to run under VAX/VMS 4.4 or later Operating Systems. Therefore, rather than learning a new operating system, the programmer can immediately concentrate on Ada program development.

Series 32000 Ada Cross-Development System for VAX/VMS Host



TL/GG/9364-2

NVADE Components

Ada Compiler

The Ada Compiler accepts as input Ada source and generates Series 32000 code that can be downloaded to, and executed on, a Series 32000-based target development board.

The Series 32000 Ada Compiler supports the full Ada language. Features include shared or unshared generics, separates, in-lines, bit representation, machine-code insertion, interrupt tasks, monitor tasks and terminal I/O. The compiler generates GNX COFF (Common Object File Format) object files that can be linked with object files generated by other GNX compilers. The Ada compiler performs several optimizations, including value-tracking global register allocation, register assignment for commons and locals, common sub-expression removal, branch and dead code analysis, some constraint check removal, and local peephole optimizations. The Ada compiler operates as a re-entrant shareable process in the VAX/VMS host system, allowing the compiler to make full use of most operating system facilities.

In addition, the Ada compiler provides features to aid in the development of real-time, embedded control, and training simulator software applications. Some of these include Ada Pragmas as specified in Chapter 13 of the Ada Language Reference Manual (LRM), such as: Inline, Interface, Interface_Object, Pack, Page, Priority, Share_Body and Suppress. Also included is a Machine Code Package which provides an interface for handling machine code insertion and generics (Unchecked_Deallocation and Unchecked_Conversion) for controlling storage and type conversions.

Program Generation Utilities

An Ada make utility, similar in operation to that found in the UNIX® operating system, is provided to simplify program compilation by maintaining program unit dependency information. This utility determines which files must be recompiled to produce a current executable file. This utility can also be used to ensure that the named unit is up-to-date, recompiling dependencies as necessary. Also provided is a source code formatter, easily configurable for individual Ada coding standards.

Program Library Utilities

The Ada Language imposes stringent requirements on an Ada Program Library. While the language provides for separate compilation of program units, each unit is compiled in the "context" of previously compiled units. The compiler must have access to this context, and the context must be carefully organized in the form of a Program Library. This library has been designed to enhance the compiler performance. A set of utilities is provided to manage, manipulate, and display Program Library information.

In addition, the Series 32000 Ada Cross-Development System permits Ada Program Libraries to be hierarchically organized, so that units not local to one library can be found in other libraries. Thus, programmers can work without interference on local versions of individual program units, while retrieving the remainder of the program from higher-level libraries.

NVADE also uses DIANA (Descriptive Intermediate Attributed Notation for Ada), which generates an intermediated representation for each unit. DIANA provides a tree-structured representation of an Ada program encoding the complete syntactic and semantic information of each individual Ada unit. The presence of DIANA as an integrated mechanism makes possible powerful editing, debugging and program query facilities, thus providing the means for simple and efficient incremental compilation.

Debuggers

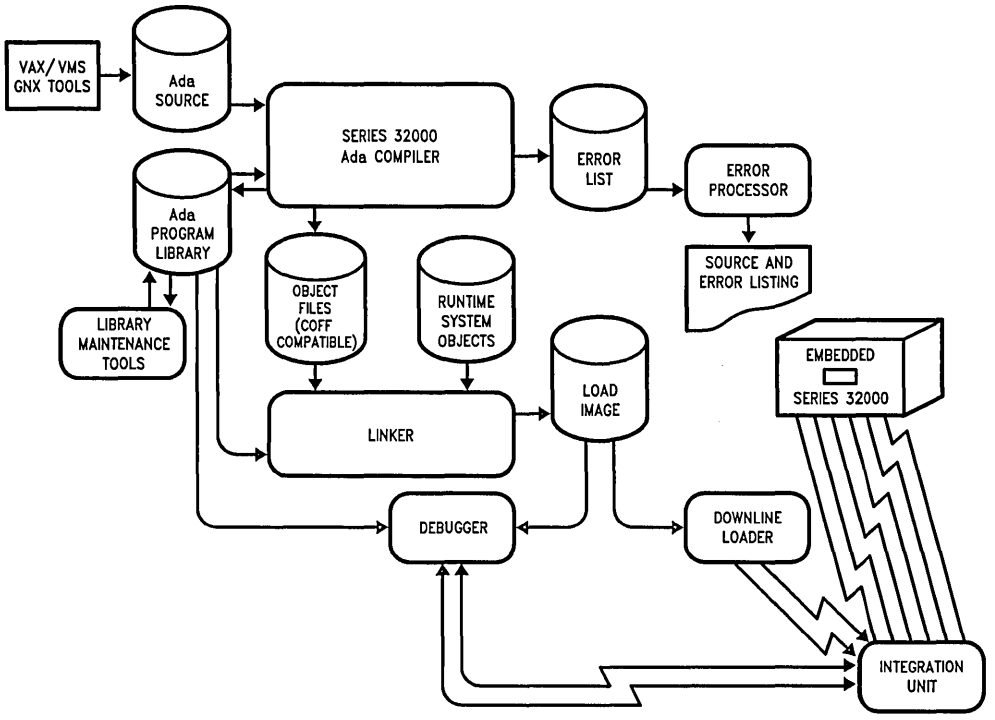
The standard GNX debugger, DBG32, is used with the Series 32000 Ada Cross-Development System. DBG32 can be used to debug code on the VAX host and/or to download and remotely debug or execute code on Series 32000 development board. DBG32 supports the use of National's SPLICE software debugging tool. Full machine-level debug support is provided by the debugger.

Linker

Ada object files are linked with the standard GNX linker, which is called by the Ada compiler pre-linker. The GNX linker resolves references between object files and library routines and assigns relocated addresses to produce Series 32000 executable code.

NVADE Components (Continued)

Series 32000 Ada Cross-Development System for VAX/VMS Host
NVADE Modules and Run-Time Environment



TL/GG/9364-3

Ada Run-Time System

The Series 32000 Ada Run-Time System provides comprehensive support for tasking, debugging, exception handling and input/output.

The Run-Time System is linked with the user's generated Ada program. To facilitate resource utilization efficiency, major portions of the Run-Time System have been optimized. Run-Time source code for customization is also available.

Ada-VRTX Interface Package (Optional)

The Ada Run-Time System consists of a large, rich and elegant tasking system. VRTX (the Versatile Real-Time Executive) provides a small, simple, compact and fast tasking system and may be a preferred alternative to using the Ada Run-Time System, particulary for embedded microprocessor applications where space and timing are critical. This Ada-VRTX interface package (AVIP) offers Ada language users a convenient means of interface with VRTX. AVIP allows Ada programmers the ability to call any VRTX service from their Ada program. (The only exceptions are the calls provided for user-defined interrupt handlers and for partition create and extend.) The actual operations performed by VRTX are identical in both assembly language and Ada. Thus, this package gives users both the elegant features of the Ada language and VRTX's unique tasking system.

PRE-REQUISITES

— VAX/VMS Host Computer 750-89XX
— VMS Operating System
— VAX/VMS GNX Assembler Package

Supported Hardware/Software

— All VAX/VMS computers
— DB32000, DB332-PLUS, VME532 target development system board with power supply

NVADE Components (Continued)

Shipping Package

— Series 32000 Installation Instructions and Applications Notes
— 1600 bpi magnetic tape (9-track VMS copy format)
— Ada Language Reference Manual (ANSI/MIL-STD 1815A)
— Ada Compiler and support tools documentation

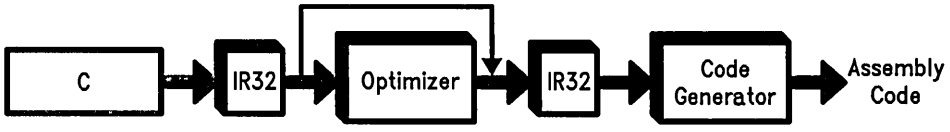
Ordering Information

Part Number	
NSW-Ada-BRVM-1	Binary Ada Cross Dev. System Tape, Vax-11/750, 11/780, 82XX
NSW-Ada-BRVM-2	Binary Ada Cross Dev. System Tape, Vax-11/785, 83XX
NSW-Ada-BRVM-3	Binary Ada Cross Dev. System Tape, Vax-8500, 8530, 8600
NSW-Ada-BRVM-4	Binary Ada Cross Dev. System Tape, Vax-8550, 8650, 8700
NSW-Ada-BRVM-5	Binary Ada Cross Dev. System Tape, Vax-88XX, 89XX
NSW-AVIP-BRVM-1	Binary Ada VRTX Int. Pckg. Tape, Vax-11/750, 11/780, 82XX
NSW-AVIP-BRVM-2	Binary Ada VRTX Int. Pckg. Tape, Vax-11/785, 83XX
NSW-AVIP-BRVM-3	Binary Ada VRTX Int. Pckg. Tape, Vax-8500, 8530, 8600
NSW-AVIP-BRVM-4	Binary Ada VRTX Int. Pckg. Tape, Vax-8550, 8650, 8700
NSW-AVIP-BRVM-5	Binary Ada VRTX Int. Pckg. Tape, Vax-88XX, 89XX
NSW-ARTS-SRVM-1	Source Ada RUNTIME SYSTEM Tape, Vax-11/750, 11/780, 82XX
NSW-ARTS-SRVM-2	Source Ada RUNTIME SYSTEM Tape, Vax-11/785, 83XX
NSW-ARTS-SRVM-3	Source Ada RUNTIME SYSTEM Tape, Vax-8500, 8530, 8600
NSW-ARTS-SRVM-4	Source Ada RUNTIME SYSTEM Tape, Vax-8550, 8650, 8700
NSW-ARTS-SRVM-5	Source Ada RUNTIME SYSTEM Tape, Vax-88XX, 89XX
NSP-Ada-VMS	Additional Manual Sets for VAX/VMS Ada Development System

National Semiconductor

PRELIMINARY

Series 32000® GNX-Version 3 C Optimizing Compiler



TL/EE/10363-1

- Generates high-quality code for the Series 32000 architecture
- Implements the C Language as defined by B. Kernighan and D. Ritchie in *The C Programming Language*
- Uses state-of-the-art optimization techniques
- Supports mixed-language programming
- Includes a complete run-time C library and highly optimized math library
- Incorporates many draft-proposed ANSI C standard (X3J11) features
- Compiles under UNIX®, ULTRIX™, and VMS™ operating systems

1.0 Introduction

A substantial amount of application code is developed in a high-level language. Therefore, the speed and efficiency of the application are functions not only of processor speed, but also of quality of code generated by the high-level language compiler. An inefficient compiler can extract a significant performance penalty. Likewise, a significant performance improvement can be achieved for much lower cost in software rather than hardware. For this reason, National Semiconductor has developed a line of optimizing compilers that generate extremely efficient code for the Series 32000 architecture.

1.1 Product Overview

The Series 32000 GNX-Version 3 C Optimizing Compiler is a member of National Semiconductor's optimizing compiler family, which also includes compilers that support the Pascal and FORTRAN 77 programming languages. Because all three optimizing compilers use a standard calling sequence, internal intermediate representation, and object file format, mixed-language programming is greatly simplified. The ability to use mixed-language programming simplifies the porting of pre-existing applications and code reuse. A detailed discussion of mixed-language programming is presented in the *GNX-Version 3 C Optimizing Compiler Reference Manual.*

The C Optimizing Compiler fully implements the C Language, as defined by B. Kernighan and D. Ritchie. The C Optimizing Compiler is also compatible with the UNIX Systtem V C compiler, derived from the fully portable C compiler (pcc). Several features of the draft ANSI C standard (X3J11) are supported.

The input to the C Optimizing Compiler is a C language source program. The output, controlled by command-line options, is either a Series 32000 executable module, a Series 32000 object module, or Series 32000 assembly code.

1.2 Native and Cross-Support

The GNX-Version 3 C Optimizing Compiler is available hosted as a cross-support compiler on the VAX™ series of computers, running the VMS, UNIX (bsd), and ULTRIX operating systems and on a Sun-3® workstation running SunOS™. Also supported are National Semiconductor's SYS32™/20 and SYS32/30 development environments.

1.3 GNX Development Tools

The GNX-Version 3 C Optimizing Compiler is an integral component of the GNX Cross-Development tool set. The GNX-Version 3 Assembler Package includes the Series 32000 assembler, the GNX linker, debuggers, libraries, and development board monitors. The GNX-Version 3 Assembler Package is a prerequisite for the GNX-Version 3 C Optimizing Compiler. See the *GNX-Version 3 Development Tools Datasheet* for more information on the GNX Tools.

1.0 Introduction (Continued)

The SYS32/20 and SYS32/30 PC-Add-In Development Packages are complete, high-performance packages that convert an IBM®-PC™/AT or compatible computer into a powerful multi-user system for developing applications that use the Series 32000 family. The SYS32 systems are based on the Series 32000 processor family; the SYS32/20 includes an NS32032 Central Processing Unit, and the SYS32/30 is based on the NS32332 CPU. Both the SYS32/20 and SYS32/30 run a derivative of the UNIX System V.3 operating system. Because these host systems are themselves based on the Series 32000 processor family, application code can be debugged on the host system without down-loading to target hardware.

2.0 Compiler Structure

The C Optimizing Compiler is a modular language processor consisting of five separate programs: the driver, the macro preprocessor (cpp), the parser (front end), the optimizer, and the code generator.

2.1 The Driver

The driver is a program that parses and interprets the command line and, in turn, sequentially calls each of the other programs, based on its input and the command-line options invoked. Under the UNIX operating system, the assembler and linker are also automatically invoked by the driver as required; under VMS, the assembler is invoked by the driver, and linking is done at the command line.

2.2 The Macro Preprocessor (cpp)

The macro preprocessor is the standard C preprocessor, known as cpp. The macro preprocessor's input is the C source program with preprocessor macros; its output is processed C code, with all preprocessor commands expanded and transformed as necessary. The macro preprocessor can be used to define constants, insert text from another file, or conditionally include or exclude source code from compilation based on a testable condition.

2.3 The C Language Parser (front end)

The front end of the C Optimizing Compiler is derived from the UNIX portable C compiler (pcc), with bug fixes and extensions included. The front end's input is C source code; its output is an intermediate representation that can be passed either to the optimizer or the code generator.

Among the extensions implemented in the front end are:

- Unsigned constants
- Enumerated types
- Improved structure manipulation; structures can be assigned, passed as parameters to functions, and returned by functions. Structure and union member names can be reused in other structures and unions in the same module. No limit is imposed on the size of structures.
- **Void** data type
- Signed and unsigned bitfields
- **Volatile** type; variables can be declared as type **volatile** to make them inaccessible to the optimizer. This is useful for mapping to external devices.
- **Const** keyword

The void, volatile, and const extensions conform to **ANSI C** standard (X3J11) features.

The output of the front end is a proprietary intermediate representation that can be either used as input to the optional optimizer phase or passed directly to the code generator. This intermediate language, known as **IR32**, is an attributed tree-structured representation. IR32 is completely high-level language independent; all of the GNX optimizing compilers produce the same internal representation. This allows a common back end to be shared by all GNX optimizing compilers.

2.4 The Optimizer

The state-of-the-art GNX optimizer is based on advanced optimization theory developed over the past 15 years. Depending on the compiler and application code characteristics, the GNX optimizer improves code performance from 15 to 200 percent beyond that of other compilers.

The GNX-Version 3 C optimizer is the most innovative component of the GNX Optimizing Compilers. The optimizer's input is an IR32 intermediate representation file; its output is an optimized IR32 file. The optimization pass is optional.

Unlike many other optimizers that are local in nature, optimizations are performed across the whole program by using sophisticated global-data-flow analysis.

The optimization process can be thought of as a five-step sequence. The sequence of optimizations has been carefully chosen to ensure that each optimization is performed to maximum effect and to provide more opportunities for later optimizations. These steps are as follows:

Step One—Local Optimizations

The source program is read-in one procedure at a time. A procedure is then partitioned into basic blocks: sequences of code that have branches only at entry or exit. Optimizations performed at this stage include:

- **Value Propagation**—replacing variables with their most recent values
- **Constant Folding**—evaluating expressions that consist solely of constants
- **Redundant Assignment Elimination**—eliminating assignments that are not used or that are reassigned prior to use

2.0 Compiler Structure (Continued)

The relationships between the various optimizations are illustrated as follows:

The program Sequence

```
a = 4;
if (a*8 < 0) b = 15;
else b = 20;
... code which uses b but
    not a ...
```

is translated by the compiler front end into the following intermediate code

```
a ← 4
if (a*8 >= 0) goto L1
b ← 15
goto L2
L1: b ← 20
L2: ...
```

which is transformed by "value propagation" into

```
a ← 4
if (4*8 >= 0) goto L1
b ← 15
goto L2
L1: b ← 20
L2: ...
```

which after "constant folding" becomes

```
a ← 4
if (true) goto L1
b ← 15
goto L2
L1: b ← 20
L2: ...
```

"dead code removal" results in

```
a ← 4
goto L1
L1: b ← 20
L2: ...
```

which is transformed by another "flow optimization" into

```
a ← 4
b ← 20
...
```

Since there is no further use of a, a ← 4 is a "redundant assignment:"

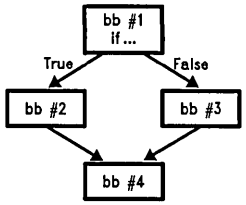
```
b ← 20
...
```

Step Two—Flow Optimizations

A flow graph is constructed. Each basic block is a node in the graph, with "arrows" drawn to represent program flow. Optimizations performed at this stage include:

- **Branch Elimination**—branches to branches are removed. Code may be reordered to eliminate branches.
- **Dead Code Removal**—code that will never be executed is removed.

The following diagram is an example of a flow graph:



TL/EE/10363-2

Step Three—Global-Data-Flow Analysis

Global-data-flow analysis is a process that identifies desirable global code transformations that can speed code execution. Since studies have shown that most programs spend 90 percent or more of their time in loops, particular attention is paid to transformations that allow loops to execute faster. This involves several techniques:

- **Fully Redundant Expression Elimination**—Expressions that are computed twice on the same path are instead computed only once, with the result saved, usually in a register.
- **Partially Redundant Expression Elimination**—If a path exists that contains a computation and a path exists that does not contain a computation, the computation is placed in each path. This makes the expression fully redundant, allowing it to be eliminated.
- **Loop Invariant Code Motion**—Values that are computed repeatedly inside of a loop are instead computed outside the loop and the result saved.
- **Strength Reduction**—Complex instructions are replaced by simpler substitutes (i.e., multiplications may be replaced with a sequence of additions).
- **Induction Variable Elimination**—Variables that maintain a fixed relation to other variables are replaced.

Step Four—Register Allocation

Register allocation is the process of placing variables in registers rather than main memory, allowing much faster access times. Proper allocation of registers can lead to significant improvement in execution speed. Most optimizing compilers attempt register allocation for local variables, to avoid problems caused by "aliasing," or referring to a variable in more than one way. By using a sophisticated algorithm, the GNX-Version 3 C Optimizing Compiler considers nearly all variables as candidates for register allocations.

2.0 Compiler Structure (Continued)

The algorithm used by the optimizer is called the **coloring algorithm**, derived from graph theory. The "live range" of each variable is constructed. The live range is the program path along which a variable has a value; assignment to a variable generally starts a new live range, which terminates with the last use of that value. Two variables that do not have intersecting live ranges can share a register. More frequently used variables are given priority for register allocation. In this way, maximum usage can be made of the registers. Other optimizations performed at this stage are:

- **Allocation Of Safe And Scratch Registers**—By convention, registers R0 through R2 and F0 through F3 are considered "scratch" registers; their values are not retained across procedure calls. Usage of these registers can reduce overhead of procedure calls.
- **Register Parameter Allocation**—For static routines, parameters are passed in registers whenever possible.

Step Five—Code Rewrite

Code is rewritten in IR32 to be passed to the code generator. Code is reorganized where necessary to increase performance.

2.5 The Code Generator

The code generator's input is an IR32 file; its output is assembly code that can be assembled by the GNX assembler into an object module.

The code generator matches expression trees with optimal code sequences. Several "peephole" optimizations are performed by the code generator: further reduction of arithmetic identities, stack and frame alignments, and strength reductions.

In addition, the target CPU and FPU are taken into consideration when code is produced. Sequences of code are chosen based on the characteristics of the target processor specified by the user. This further increases code efficiency.

3.0 Ordering Information

Supported Host Environments and Order Codes:

SYS32/20: NSW-C-3-BHAF3	**MicroVAX/VMS:** NSW-C-3-BCVM
SYS32/30: NSW-C-3-BHBF3	**MicroVAX/ULTRIX:** NSW-C-3-BCVX
VAX/VMS: NSW-C-3-BRVM	**Sun-3:** NSW-C-3-BCSX
VAX/ULTRIX (UNIX bsd): NSW-C-3-BRVX	

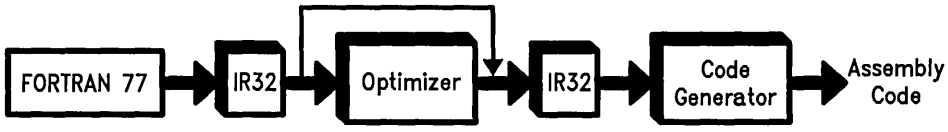
GNX-Version 3 Assembler and Cross-Development tools (required for use with the Optimizing C Compiler):

SYS32/30:	NSW-ASM-3-BHAF3 (provided with SYS32/20 system)
SYS32/30:	NSW-ASM-3-BHBF3 (provided with SYS32/30 system)
VAX/VMS:	NSW-ASM-3-BRVM
VAX/ULTRIX (UNIX bsd:)	NSW-ASM-3-BRVX
MicroVAX/VMS:	NSW-ASM-3-BCVM
MicroVAX/ULTRIX:	NSW-ASM-3-BCVX
Sun-3:	NSW-ASM-3-BCSX

For further information regarding National Semiconductor's software development tools and development hosts, please refer to the following datasheets:

GNX-Version 3 Development Tools
GNX-Version 3 FORTRAN 77 Compiler
GNX-Version 3 Pascal Compiler
SYS32/20 PC-Add-In-Development Package
SYS32/30 PC-Add-In-Development Package

Series 32000® GNX-Version 3 FORTRAN 77 Optimizing Compiler



TL/EE/10362–1

- **Generates high-quality code for the Series 32000 architecture**
- **Implements the FORTRAN 77 Language as described by the American Standard publication *Programming Language FORTRAN (ANSI X3.9-1978)***
- **Uses state-of-the-art optimization techniques**
- **Supports mixed-language programming**
- **Includes complete FORTRAN intrinsic function and I/O libraries**
- **Implements many extensions to standard FORTRAN 77**
- **Compiles under UNIX®, ULTRIX™, and VMS™ operating systems**

1.0 Introduction

A substantial amount of application code is developed in a high-level language. Therefore, the speed and efficiency of the application are functions not only of processor speed, but also of quality of code generated by the high-level language compiler. An inefficient compiler can extract a significant performance penalty. Likewise, a significant performance improvement can be achieved for much lower cost in software rather than hardware. For this reason, National Semiconductor has developed a line of optimizing compilers that generate extremely efficient code for the Series 32000 architecture.

1.1 Product Overview

The Series 32000 GNX-Version 3 FORTRAN 77 Optimizing Compiler is a member of National Semiconductor's optimizing compiler family, which also includes compilers that support the C and Pascal programming languages. Because all three optimizing compilers use a standard calling sequence, internal intermediate representation, and object file format, mixed-language programming is greatly simplified. The ability to use mixed-language programming simplifies the porting of pre-existing applications and code reuse. A detailed discussion of mixed-language programming is presented in the *GNX-Version 3 FORTRAN 77 Optimizing Compiler Reference Manual.*

The FORTRAN 77 Optimizing Compiler fully implements the FORTRAN 77 programming language, as defined by the American Standard publication *Programming Language FORTRAN (ANSI X3.9-1978).* In addition, a command-line option is provided that forces the compiler to accept as input only programs that adhere to the FORTRAN 66 standard.

The input to the FORTRAN 77 Optimizing Compiler is a FORTRAN 77 language source program. The output, controlled by command-line options, is either a Series 32000 executable module, a Series 32000 object module, or Series 32000 assembly code.

1.2 Native and Cross-support

The GNX-Version 3 FORTRAN 77 Optimizing Compiler is available hosted as a cross-support compiler on the VAX™ series of computers, running the VMS, UNIX (bsd), and ULTRIX operating systems. Also supported are National Semiconductor's SYS32™/20 and SYS32/30 development environments.

1.3 GNX Development Tools

The GNX-Version 3 FORTRAN 77 Optimizing Compiler is an integral component of the GNX Cross-development tool set. The GNX-Version 3 Assembler Package includes the Series 32000 assembler, the GNX linker, debuggers, libraries, and development board monitors. The GNX-Version 3 Assembler Package is a prerequisite for the GNX-Version 3 FORTRAN 77 Optimizing Compiler. See the *GNX-Version 3 Development Tools Datasheet* for more information on the GNX Tools.

1.0 Introduction (Continued)

The SYS32/20 and SYS32/30 PC-Add-In Development Packages are complete, high-performance packages that convert an IBM®-PC™/AT or compatible computer into a powerful multi-user system for developing applications that use the Series 32000 family. The SYS32 systems are based on the Series 32000 processor family; the SYS32/20 includes an NS32032 Central Processing Unit, and the SYS32/30 is based on the NS32332 CPU. Both the SYS32/20 and SYS32/30 run a derivative of the UNIX System V.3 operating system. Because these host systems are themselves based on the Series 32000 processor family, application code can be debugged on the host system without down-loading to target hardware.

2.0 Compiler Structure

The FORTRAN 77 Optimizing Compiler is a modular language processor consisting of five separate programs: the driver, the macro preprocessor (cpp), the parser (front end), the optimizer, and the code generator.

2.1 The Driver

The driver is a program that parses and interprets the command line and, in turn, sequentially calls each of the other programs, based on its input and the command-line options invoked. Under the UNIX operating system, the assembler and linker are also automatically invoked by the driver as required; under VMS, the assembler is invoked by the driver, and linking is done at the command line.

2.2 The Macro Preprocessor (cpp)

The macro preprocessor is the standard C-language preprocessor, known as **cpp**. Preprocessing is an optional step and is performed only if macros are defined in the FORTRAN 77 source code. The macro preprocessor's input is the FORTRAN 77 program with preprocessor macros; its output is processed FORTRAN 77 code, with all preprocessor commands expanded and transformed as necessary. The macro preprocessor can be used to define constants, insert text from another file, or conditionally include or exclude source code from compilation based on a testable condition.

2.3 FORTRAN 77 Language Parser (front end)

The FORTRAN 77 language parser, known as **f77_fe**, takes as input a FORTRAN 77 program. The output is an intermediate representation that can be passed either to the optimizer or the code generator. Several extensions to standard FORTRAN are implemented in the FORTRAN 77 language parser.

Among the extensions implemented in the front end are:

- Double Complex data type; each datum is represented by a pair of double-precision real variables.
- Short Integer data type; declarations of type **integer*2** are accepted
- Hollerith (nh) notation
- Variable-length program lines
- unlimited identifier length and underscores in identifier names
- non-integer constants (binary, octal, and hexadecimal)
- recursion; procedures may call themselves directly or through a chain of other procedures

Note: A command-line option is provided that will force the compiler to accept only code that conforms to the FORTRAN 77 (or FORTRAN 66) standard (ANSI X3.9-1978).

The output of the front end is a proprietary intermediate representation that can be either used as input to the optional optimizer phase or passed directly to the code generator. This intermediate language, known as **IR32**, is an attributed tree-structured representation. IR32 is completely high-level language independent; all of the GNX optimizing compilers produce the same internal representation. This allows a common back end to be shared by all GNX optimizing compilers.

2.4 The Optimizer

The state-of-the-art GNX optimizer is based on advanced optimization theory developed over the past 15 years. Depending on the compiler and application code characteristics, the GNX optimizer improves code performance from 15 to 200 percent beyond that of other compilers.

The GNX-Version 3 FORTRAN 77 optimizer is the most innovative component of the GNX Optimizing Compilers. The optimizer's input is an IR32 intermediate representation file; its output is an optimized IR32 file. The optimization pass is optional.

Unlike many other optimizers that are local in nature, optimizations are performed across the whole program by using sophisticated global-data-flow analysis.

The optimization process can be throught of as a five-step sequence. The sequence of optimizations has been carefully chosen to ensure that each optimization is performed to maximum effect and to provide more opportunities for later optimizations. These steps are as follows:

Step One—Local Optimizations

The source program is read-in one procedure at a time. A procedure is then partitioned into **basic blocks**: sequences of code that have branches only at entry or exit. Optimizations performed at this stage include:

- **Value Propagation**—replacing variables with their most recent values
- **Constant Folding**—evaluating expressions that consist solely of constants
- **Redundant Assignment Elimination**—eliminating assignments that are not used or that are reassigned prior to use

2.0 Compiler Structure (Continued)

The relationships between the various optimizations are illustrated as follows:

The program Sequence

```
        a = 4
        IF (a * 8 .LT. 0) THEN
          b = 15
        ELSE
          b = 20
        ENDIF
        ... code which uses b but not a ...
```

is translated by the Compiler front end into the following intermediate code

```
        a ← 4
        if (a * 8 >= 0) goto L1
        b ← 15
        goto L2
    L1: b ← 20
    L2: ...
```

which is transformed by "value propagation" into

```
        a ← 4
        if (4 * 8 >= 0) goto L1
        b ← 15
        goto L2
    L1: b ← 20
    L2: ...
```

which after "constant folding" becomes

```
        a ← 4
        if (true) goto L1
        b ← 15
        goto L2
    L1: b ← 20
    L2: ...
```

"dead code removal" results in

```
        a ← 4
        goto L1
    L1: b ← 20
    L2: ...
```

which is transformed by another "flow optimization" into

```
        a ← 4
        b ← 20
```

Since there is no further use of a, a ← 4 is a "redundant assignment:"

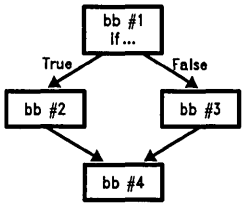
```
        b ← 20
        ...
```

Step Two—Flow Optimizations

A flow graph is constructed. Each basic block is a node in the graph, with "arrows" drawn to represent program flow. Optimizations performed at this stage include:

- **Branch elimination**—branches to branches are removed. Code may be reordered to eliminate branches.
- **Dead code removal**—code that will never be executed is removed.

The following diagram is an example of a flow graph:



TL/EE/10362-2

Step Three—Global-Data-Flow Analysis

Global-data-flow analysis is a process that identifies desirable global code transformations that can speed code execution. Since studies have shown that most programs spend 90 percent or more of their time in loops, particular attention is paid to transformations that allow loops to execute faster. This involves several techniques:

- **Fully redundant expression elimination**—Expressions that are computed twice on the same path are instead computed only once, with the result saved, usually in a register.
- **Partially redundant expression elimination**—If a path exists that contains a computation and a path exists that does not contain a computation, the computation is placed in each path. This makes the expression fully redundant, allowing it to be eliminated.
- **Loop invariant code motion**—Values that are computed repeatedly inside of a loop are instead computed outside the loop and the result saved.
- **Strength reduction**—Complex instructions are replaced by simpler substitutes (i.e., multiplications may be replaced with a sequence of additions).
- **Induction variable elimination**—Variables that maintain a fixed relation to other variables are replaced.

Step Four—Register Allocation

Register allocation is the process of placing variables in registers rather than main memory, allowing much faster access times. Proper allocation of registers can lead to significant improvement in execution speed. Most optimizing compilers attempt register allocation for local variables, to avoid problems caused by "aliasing," or referring to a variable in more than one way. By using a sophisticated algorithm, the GNX-Version 3 FORTRAN 77 Optimizing Compiler considers nearly all variables as candidates for register allocations.

2.0 Compiler Structure (Continued)

The algorithm used by the optimizer is called the **coloring algorithm,** derived from graph theory. The "live range" of each variable is constructed. The live range is the program path along which a variable has a value; assignment to a variable generally starts a new live range, which terminates with the last use of that value. Two variables that do not have intersecting live ranges can share a register. More frequently used variables are given priority for register allocation. In this way, maximum usage can be made of the registers. Other optimizations performed at this stage are:

- **Allocation of safe and scratch registers**—By convention, registers R0 through R2 and F0 through F3 are considered "scratch" registers; their values are not retained across procedure calls. Usage of these registers can reduce overhead of procedure calls.
- **Register Parameter Allocation**—for static routines, parameters are passed in registers whenever possible.

Step Five—Code Rewrite

Code is rewritten in IR32 to be passed to the code generator. Code is reorganized where necessary to increase performance.

2.5 The Code Generator

The code generator's input is an IR32 file; its output is assembly code that can be assembled by the GNX assembler into an object module.

The code generator matches expression trees with optimal code sequences. Several "peephole" optimizations are performed by the code generator: further reduction of arithmetic identities, stack and frame alignments, and strength reductions.

In addition, the target CPU and FPU are taken into consideration when code is produced. Sequences of code are chosen based on the characteristics of the target processor specified by the user. This further increases code efficiency.

3.0 Ordering Information

Supported Host Environments and Order Codes:

SYS32/20:	**VAX/ULTRIX (UNIX bsd):**
NSW-F77-3-BHAF3	NSW-F77-3-BRVX
SYS32/30:	**Micro VAX/VMS:**
NSW-F77-3-BHBF3	NSW-F77-3-BCVM
VAX/VMS:	**Micro VAX/ULTRIX:**
NSW-F77-3-BRVM	NSW-F77-3-BCVX

GNX-Version 3 Assembler and Cross-development tools (required for use with the Optimizing FORTRAN 77 Compiler):

SYS32/30:	NSW-ASM-3-BHAF3 (provided with SYS32/20 system)
SYS32/30:	NSW-ASM-3-BHBF3 (provided with SYS32/30 system)
VAX/VMS:	NSW-ASM-3-BRVM
VAX/ULTRIX (UNIX bsd):	NSW-ASM-3-BRVX
Micro VAX/VMS:	NSW-ASM-3-BCVM
Micro VAX/ULTRIX:	NSW-ASM-3-BCVX

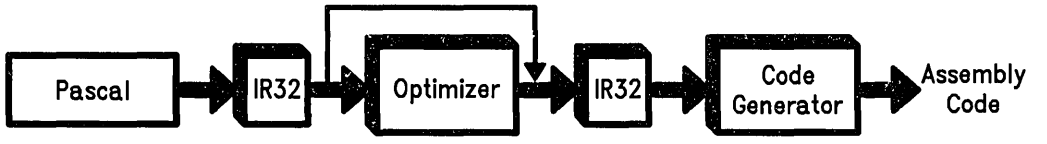
For further information regarding National Semiconductor's software development tools and development hosts, please refer to the following datasheets:

GNX-Version 3 Development Tools
GNX-Version 3 C Compiler
GNX-Version 3 Pascal Compiler
SYS32/20 PC-Add-In-Development Package
SYS32/30 PC-Add-In-Development Package

National Semiconductor

PRELIMINARY

Series 32000® GNX-Version 3 Pascal Optimizing Compiler



TL/EE/10365–1

- **Generates high-quality code for the Series 32000 architecture**
- **Implements the Pascal Language as described by the International Standards Organization (ISO) standard *ISO dp7185 level 1***
- **Uses state-of-the-art optimization techniques**
- **Supports mixed-language programming**
- **Includes a complete Pascal run-time library and highly optimized math library**
- **Implements many extensions to standard Pascal**
- **Compiles under UNIX®, ULTRIX™ and VMS™ operating systems**

1.0 Introduction

A substantial amount of application code is developed in a high-level language. Therefore, the speed and efficiency of the application are functions not only of processor speed, but also of quality of code generated by the high-level language compiler. An inefficient compiler can extract a significant performance penalty. Likewise, a significant performance improvement can be achieved for much lower cost in software rather than hardware. For this reason, National Semiconductor has developed a line of optimizing compilers that generate extremely efficient code for the Series 32000 architecture.

1.1 Product Overview

The Series 32000 GNX-Version 3 Pascal Optimizing Compiler is a member of National Semiconductor's optimizing compiler family, which also includes compilers that support the C and FORTRAN 77 programming languages. Because all three optimizing compilers use a standard calling sequence, internal intermediate representation, and object file format, mixed-language programming is greatly simplified. The ability to use mixed-language programming simplifies the porting of pre-existing applications and code reuse. A detailed discussion of mixed-language programming is presented in the *GNX-Version 3 Pascal Optimizing Compiler Reference Manual.*

The Pascal Optimizing Compiler fully implements the Pascal programming language, as defined by the International Standards Organization (ISO) standard **ISO dp7185 level 1**, with several useful extensions to the compiler extensions found in the University of California, Berkeley Pascal compiler (**pc**). In addition, a command-line option is provided that forces the compiler to accept as input only programs that adhere to the ISO standard.

The input to the Pascal Optimizing Compiler is a Pascal language source program. The output, controlled by command-line options, is either a Series 32000 executable module, a Series 32000 object module, or Series 32000 assembly code.

1.2 Native and Cross-Support

The GNX-Version 3 Pascal Optimizing Compiler is available hosted as a cross-support compiler on the VAX™ series of computers, running the VMS, UNIX (bsd), and ULTRIX operating systems. Also supported are National Semiconductor's SYS32™/20 and SYS32/30 development environments.

1.3 GNX Development Tools

The GNX-Version 3 Pascal Optimizing Compiler is an integral component of the GNX Cross-development tool set. The GNX-Version 3 Assembler Package includes the Series 32000 assembler, the GNX linker,

1.0 Introduction (Continued)

debuggers, libraries, and development board monitors. The GNX-Version 3 Assembler Package is a prerequisite for the GNX-Version 3 Pascal Optimizing Compiler. See the *GNX-Version 3 Development Tools Datasheet* for more information on the GNX Tools.

The SYS32/20 and SYS32/30 PC-Add-In Development Packages are complete, high-performance packages that convert an IBM-PC™/AT or compatible computer into a powerful multi-user system for developing applications that use the *Series 32000* family. The SYS32 systems are based on the *Series 32000* processor family; the SYS32/20 includes an NS32032 Central Processing Unit, and the SYS32/30 is based on the NS32332 CPU. Both the SYS32/20 and SYS32/30 run a derivative of the UNIX System V.3 operating system. Because these host systems are themselves based on the *Series 32000* processor family, application code can be debugged on the host system without down-loading to target hardware.

2.0 Compiler Structure

The Pascal Optimizing Compiler is a modular language processor consisting of five separate programs: the driver, the macro preprocessor (cpp), the parser (front end), the optimizer, and the code generator.

2.1 The Driver

The driver is a program that parses and interprets the command line and, in turn, sequentially calls each of the other programs, based on its input and the command-line options invoked. Under the UNIX operating system, the assembler and linker are also automatically invoked by the driver as required; under VMS, the assembler is invoked by the driver, and linking is done at the command line.

2.2 The Macro Preprocessor (cpp)

The macro preprocessor is the standard C-language preprocessor, known as **cpp**. Preprocessing is an optional step and is performed only if macros are defined in the Pascal source code. The macro preprocessor's input is the Pascal program with preprocessor macros; its output is processed Pascal code, with all preprocessor commands expanded and transformed as necessary. The macro preprocessor can be used to define constants, insert text from another file, or conditionally include or exclude source code from compilation based on a testable condition.

2.3 The Pascal Language Parser (front end)

The Pascal language parser, known as **pas_fe**, takes as input a Pascal program. The output is an intermediate representation that can be passed either to the optimizer or the code generator. Conformant array parameters, as defined in the ISO level 1 Standard, are fully supported. Several extensions to standard Pascal are implemented in the Pascal language parser.

Among the extensions implemented in the front end are:

- Separate compilation; programs can be divided into a number of files that can be compiled separately
- Longreal data type; double-precision (64-bit) floating point values
- String padding of constant strings with blanks
- Conversions of pointers to integers and vice versa
- Unlimited identifier length and underscores in identifier names
- Non-integer constants (binary, octal, and hexadecimal)
- Constant expressions; constants can be defined in terms of mathematical expressions
- predefined **argc** and **argv** functions; allows application programs to easily accept and process command-line arguments

Note: A command-line option is provided that will force the compiler to accept only code that conforms to the ISO Pascal standard *ISO dp7185 level 1.*

The output of the front end is a proprietary intermediate representation that can be either used as input to the optional optimizer phase or passed directly to the code generator. This intermediate language, known as **IR32**, is an attributed tree-structured representation. IR32 is completely high-level language independent; all of the GNX optimizing compilers produce the same internal representation. This allows a common back end to be shared by all GNX optimizing compilers.

2.4 The Optimizer

The state-of-the-art GNX optimizer is based on advanced optimization theory developed over the past 15 years. Depending on the compiler and application code characteristics, the GNX optimizer improves code performance from 15 to 200 percent beyond that of other compilers.

The GNX-Version 3 Pascal optimizer is the most innovative component of the GNX Optimizing Compilers. The optimizer's input is an IR32 intermediate representation file; its output is an optimized IR32 file. The optimization pass is optional.

Unlike many other optimizers that are local in nature, optimizations are performed across the whole program by using sophisticated global-data-flow analysis.

The optimization process can be thought of as a five-step sequence. The sequence of optimizations has been carefully chosen to ensure that each optimize is performed to maximum effect and to provide more opportunities for later optimizations. These steps are as follows:

Step One—Local Optimizations

The source program is read-in one procedure at a time. A procedure is then partitioned into **basic blocks**: sequences of code that have branches only

2.0 Compiler Structure (Continued)

at entry or exit. Optimizations performed at this stage include:

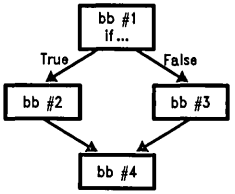
- **Value Propagation**—replacing variables with their most recent values
- **Constant Folding**—evaluating expressions that consist solely of constants
- **Redundant Assignment Elimination**—eliminating assignments that are not used or that are reassigned prior to use

Step Two—Flow Optimizations

A flow graph is constructed. Each basic block is a node in the graph, with "arrows" drawn to represent program flow. Optimizations performed at this stage include:

- **Branch elimination**—branches to branches are removed. Code may be reordered to eliminate branches.
- **Dead code removal**—code that will never be executed is removed.

The following diagram is an example of a flow graph:



TL/EE/10365-2

Step Three—Global-Data-Flow Analysis

Global-data-flow analysis is a process that identifies desirable global code transformations that can speed code execution. Since studies have shown that most programs spend 90 percent or more of their time in loops, particular attention is paid to transformations that allow loops to execute faster. This involves several techniques:

- **Fully redundant expression elimination**—Expressions that are computed twice on the same path are instead computed only once, with the result saved, usually in a register.
- **Partially redundant expression elimination**—If a path exists that contains a computation and a path exists that does not contain a computation, the computation is placed in each path. This makes the expression fully redundant, allowing it to be eliminated.
- **Loop invariant code motion**—Values that are computed repeatedly inside of a loop are instead computed outside the loop and the result saved.
- **Strength reduction**—Complex instructions are replaced by simpler substitutes (i.e., multiplications may be replaced with a sequence of additions).
- **Induction variable elimination**—Variables that maintain a fixed relation to other variables are replaced.

The relationship between the various optimizations are illustrated as follows:

The program sequence

```
a := 4;
if (a * 8 < 0) then b := 15;
b := 20;
... code which uses b but not a ...
```

is translated by the Compiler front end into the following intermediate code

```
a ← 4
if (a * 8 >= 0) goto L1
b ← 15
goto L2
L1: b ← 20
L2: ...
```

which is transformed by "value propagation" into

```
a ← 4
if (4 * 8 >= 0) goto L1
b ← 15
goto L2
L1: b ← 20
L2: ...
```

which after "constant folding" becomes

```
a ← 4
if (true) goto L1
b ← 15
goto L2
L1: b ← 20
L2: ...
```

"dead code removal" results in

```
a ← 4
goto L1
L1: b ← 20
L2: ...
```

which is transformed by another "flow optimization" into

```
a ← 4
b ← 20
...
```

Since there is no further use of a, a ← 4 is a "redundant assignment:"

```
b ← 20
...
```

Step Four—Register Allocation

Register allocation is the process of placing variables in registers rather than main memory, allowing much faster access times. Proper allocation of registers can lead to significant improvement in execution speed. Most optimizing compilers attempt register allocation for local variables, to avoid problems caused by "aliasing," or referring to a variable in more than one way. By using a sophisticated algorithm, the GNX-Version 3 Pascal Optimizing Compiler considers nearly all variables as candidates for register allocations.

2.0 Compiler Structure (Continued)

The algorithm used by the optimizer is called the **coloring algorithm**, derived from graph theory. The "live range" of each variable is constructed. The live range is the program path along which a variable has a value; assignment to a variable generally starts a new live range, which terminates with the last use of that value. Two variables that do not have intersecting live ranges can share a register. More frequently used variables are given priority for register allocation. In this way, maximum usage can be made of the registers. Other optimizations performed at this stage are:

- **Allocation of safe and scratch registers**—By convention, registers R0 through R2 and F0 through F3 are considered "scratch" registers; their values are not retained across procedure calls. Usage of these registers can reduce overhead of procedure calls.
- **Register Parameter Allocation**—For static routines, parameters are passed in registers whenever possible.

Step-Five—Code Rewrite

Code is rewritten in IR32 to be passed to the code generator. Code is reorganized where necessary to increase performance.

2.5 The Code Generator

The code generator's input is an IR32 file; its output is assembly code that can be assembled by the GNX assembler into an object module.

The code generator matches expression trees with optimal code sequences. Several "peephole" optimizations are performed by the code generator: further reduction of arithmetic identities, stack and frame alignments, and strength reductions.

In addition, the target CPU and FPU are taken into consideration when code is produced. Sequences of code are chosen based on the characteristics of the target processor specified by the user. This further increases code efficiency.

3.0 Ordering Information

Supported Host Environments and Order Codes:

SYS32/20:
NSW-PAS-3-BHAF3

SYS32/30:
NSW-PAS-3-BHBF3

VAX/VMS:
NSW-PAS-3-BRVM

VAX/ULTRIX (UNIX bsd):
NSW-PAS-3-BRVX

Micro VAX/VMS:
NSW-PAS-3-BCVM

Micro VAX/ULTRIX:
NSW-PAS-3-BCVX

GNX-Version 3 Assembler and Cross-development tools (required for use with the Optimizing Pascal Compiler):

SYS32/20:	NSW-ASM-3-BHAF3 (provided with SYS32/20 system)
SYS32/30:	NSW-ASM-3-BHBF3 (provided with SYS32/30 system)
VAX/VMS:	NSW-ASM-3-BRVM
VAX/ULTRIX (UNIX bsd):	NSW-ASM-3-BRVX
MicroVAX/VMS:	NSW-ASM-3-BCVM
MicroVAX/ULTRIX:	NSW-ASM-3-BCVX

For further information regarding National Semiconductor's software development tools and development hosts, please refer to the following datasheets:

GNX-Version 3 Development Tools

GNX-Version 3 C Compiler

GNX-Version 3 FORTRAN 77 Compiler

SYS32/20 PC-Add-In Development Package

SYS32/30 PC-Add-In Development Package

Section 6

Application Notes

Section 6 Contents

Instruction Execution Times of FPU NS32081 Considered for Stand-Alone Configurations

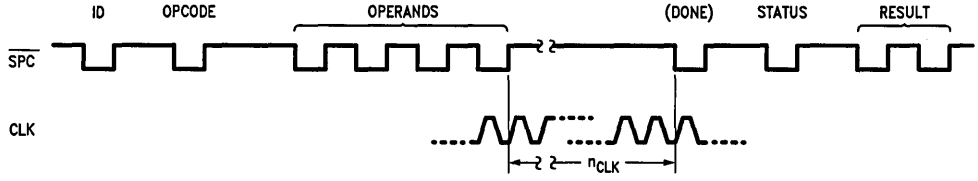
National Semiconductor
Application Brief 26
Systems & Applications Group

The table below gives execution timing information for the FPU NS32081.

The number of clock cycles nCLK is counted from the last SPC pulse, strobing the last operation word or operand into the FPU, and the Done-SPC pulse, which signals the CPU that the result is available (see *Figure 1*). The values are therefore independent of the operand's addressing modes and do not include the CPU/FPU protocol time. This makes it easy to determine the FPU execution times in stand-alone configurations.

The values are derived from measurements, the worst case is always assumed. The results are given in clock cycles (CLK).

Operation	Number of Clock-Cycles nCLK
Add, Subtract	63
Multiply Float	37
Multiply Long	51
Divide Float	78
Divide Long	108
Compare	38



TL/EE/8760-1

FIGURE 1

Use of the NS32332 with the NS32082 and the NS32201

National Semiconductor
Application Brief 27
Systems Applications Group

Care should be taken when the NS32332 is designed in a system with the NS32201 and the NS32082. Two configurations need to be considered, one with MMU and one without.

In a configuration without an MMU, TCU and CPU both run a four clock cycle bus (*Figure 1*). The RDY signal is the only incompatible signal between the CPU and TCU and therefore the RDY output of the TCU should not be directly connected to the RDY input of the NS32332. The NS32332 samples its RDY input in the middle of T3 while the NS32201 asserts its RDY output shortly after the middle of T2 and removes it shortly after the middle of T3, thus the NS32332 RDY input hold time (tRDYh) is not met. To meet tRDYh, the RDY output of the NS32201 should be clocked by the rising edge of the CTTL using a D-type flip-flop (74AS74) and then taken to the NS32332. It should be noted that the NS32332 outputs the data in a write cycle in T3 unless DT/SDONE pin is sampled low on the rising edge of the reset in which case the data is output during T2. The DT/SDONE pin is implemented as of revision B of the NS32332.

In a configuration with MMU the NS32332 runs a four clock cycle bus while the NS32082 runs a five cycle bus. Two options can be exercised.

The first option is extending the NS32332 bus cycle to five clocks by adding a blind wait state that bypasses the NS32201 (*Figure 2*). This configuration generally requires the minimum hardware modification for a 320xx based design to run the NS32332. Here the NS32201 output signals can be used to interface the NS32332 and the NS32082 to the memory or I/O. Additional wait states can be inserted by clocking the RDY output of the TCU.

The second option is to have the NS32332 run a four clock cycle bus (*Figure 3*). In this configuration the NS32201 output signals cannot be used to interface the NS32332 to memory or I/O; they can only be used to interface the NS32082 to the memory. In this configuration a revision N of the NS32082 should be used.

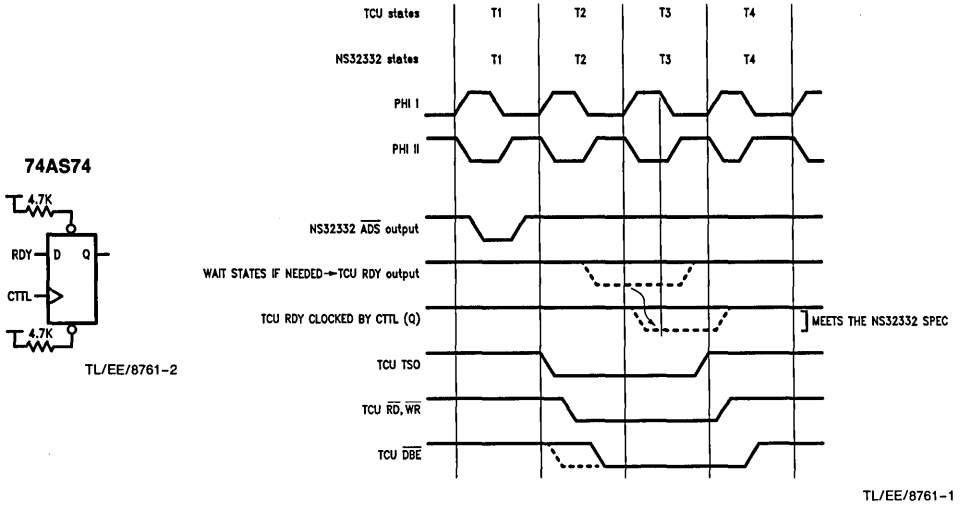
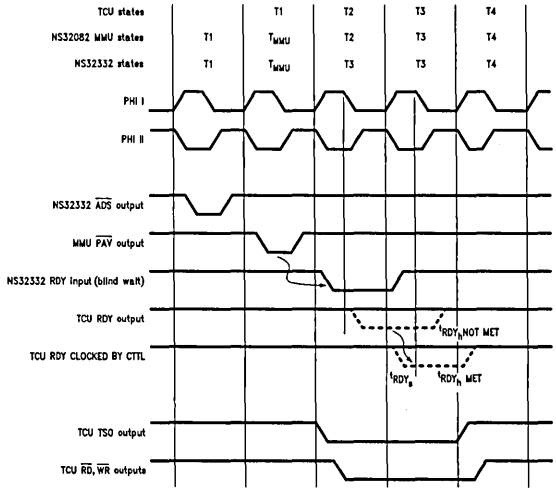
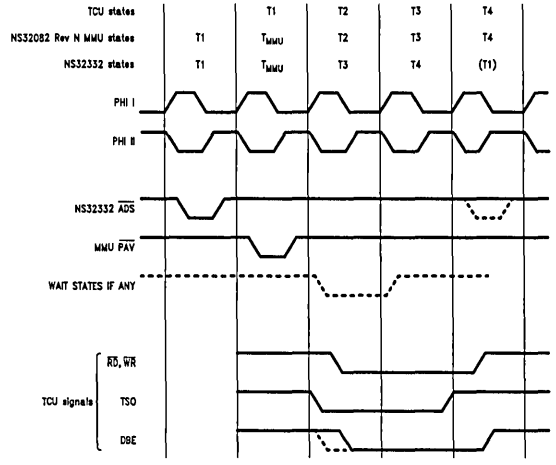


FIGURE 1. NS32332, TCU Timing Diagram, No Wait State, No MMU



TL/EE/8761–3

FIGURE 2. NS32332, MMU, TCU Timing Diagram when NS32332 is Run with 1 Wait State Similar to Timing Diagram of NS32332 Adapter to DB32000



TL/EE/8761–4

FIGURE 3. NS32332, MMU, TCU Timing Diagram with No Wait State

PC Board Layout for Floating-Point Units

National Semiconductor
Application Brief 40
Itzhak Nashelsky

For applications requiring floating-point capability, National Semiconductor offers two options:

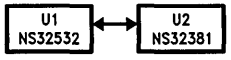
1. The NS32381: A low-cost floating-point unit (FPU) which interfaces directly to the NS32532 microprocessor.
2. The Weitek W3164: A high-performance floating-point solution which uses the NS32580 floating-point controller (FPC) to interface with the NS32532.

This application brief briefly explains how to lay-out a printed circuit (PC) board incorporating the NS32532 microprocessor and either FPU option. The board design provides maximum flexibility and can be used for either option.

Note: For detailed information regarding either the NS32381 FPU or the NS32580 FPC, refer to their data sheets.

The two FPU options are presented in *Figures 1* and *2*. To provide both floating-point options with minimal printed circuit board real estate, the NS32580's pin-out was designed to be fully compatible with that of the NS32381 FPU. *Figure 3* illustrates this pin compatibility and the location of the keying pins.

As a result, the layout of the PC board can be prepared using Option 2, leaving the decision for the final floating-point configuration to the user. Users who prefer Option 1, will therefore be able to insert the NS32381 into the NS32580's socket, leaving U3's socket unpopulated. This method was implemented in the VME532 board designed by National Semiconductor.

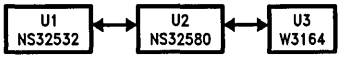


TL/EE/10351–1

FIGURE 1. Option 1, Using the NS32381 FPU

Note: Since the NS32381's package is smaller than that of the NS32580, special care should be taken while inserting the NS32381 into the NS32580's socket.

Also, to prevent damage cause by "shifted" insertion, it is recommended that four keying-pins be installed in the NS32580 socket in the center area (see *Figure 3*).



TL/EE/10351–2

FIGURE 2. Option 2, Using the W3164 and NS32580 FPC

```
o o o o o o o o o o o o o o
o o o o o o o o o o o o o o
o o x x x x x x x x x x o o
o o x x x x x x x x x x o o
o o x x k               k x x o o
o o x x                   x x o o
o o x x                   x x o o
o o x x                   x x o o
o o x x                   x x o o
o o x x                   x x o o
o o x x k               k x x o o
o o x x x x x x x x x x o o
o o x x x x x x x x x x o o
o o o o o o o o o o o o o o
o o o o o o o o o o o o o o
```

TL/EE/10351–3

x = Pins common to both NS32580 and NS32381.
o = Pins belonging to the NS32580 only.
k = Keying pins for the NS32381.

FIGURE 3. NS32580/NS32381 Pin-Out Compatibility

A Method for Efficient Task Switching Using the NS32381 FPU

National Semiconductor
Application Brief 44
Dan Biran

INTRODUCTION

Many microprocessor based embedded control systems are built as real-time multitasking systems where different functions of the system are controlled by different tasks. The multiple tasks in such a system have the appearance of all executing simultaneously, when in reality only one task is running on the processor at one time. (Readers not familiar with the concepts of *tasks* and *multitasking* can find explanations in most general textbooks about operating systems.)

A *task switch* is when one task stops executing and another begins executing. A task switch usually involves saving the values of the processor's registers onto the stack. In systems where both a Central Processor Unit (CPU) and a Floating Point Unit (FPU) are used, the registers of both processors must be saved. However, if the FPU has not been used during the execution of a task, saving its registers onto the stack is unnecessary and is an undesirable waste of time. This application brief is for the software designer of an embedded software system. It explains how to detect when the FPU has not been used in a task so the task switch time can be shortened by not saving the FPU registers.

METHOD

The Floating Status Register (FRS) *(Figure 1)* of the NS32381 has a Trap Type field (bits 0–2) that records any exceptional conditions detected by a floating point instruction. The Trap Type field is loaded with zero whenever any floating point instruction except LFSR (Load Floating Status Register) or SFSR (Store Floating Status Register) completes without encountering an exception condition.

Seven Trap Type codes are used to signal the different conditions (including the code "000" that is used to indicate "no exception"). One code "111" is not used.

Loading the FSR at the beginning of every task with a value that sets the Trap Type field to the unused code "111" lets the FSR be used later to determine whether the NS32381 has been used in the task. If the Trap Type code at the end of the task is still "111", it means that no floating point instruction has been executed since the FSR was loaded.

The execution figures below refer to a system that uses the NS32381 FPU with the National Semiconductor NS32GX32 CPU. This method also works with the NS32CG16 processor.

Saving the floating point registers onto the stack using routine 1 *(Figure 2)* described below takes 296 clock cycles. In cases where it is possible that the NS3281 has not been referenced in the current task, routine 2 *(Figure 3)* can be executed prior to saving the registers. This routine takes 43 cycles. In cases when the Floating Point Unit has not been referenced, 253 clock cycles (85.5%) are saved. If the FPU has been referenced, 43 cycles are added to the 296 cycles of the normal routine (extra 14.5%)

These numbers indicate that whenever the probability of not using the FPU is greater than 14.5% this method is efficient.

Routine #1

```
save_freg:  sfsr   tos
            movl   l0, tos
            movl   l1, tos
            movl   l2, tos
            movl   l3, tos
            movl   l4, tos
            movl   l5, tos
            movl   l6, tos
            movl   l7, tos
```

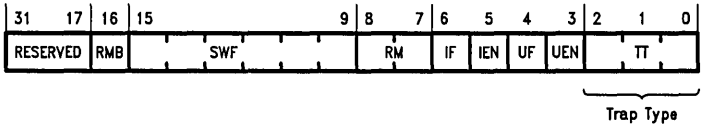
FIGURE 2

Routine #2

```
            sfsr   tos
            andb   h'7, r0
            cmpb   h'7, r0
            beq    end
save_freg:  sfsr   tos
            movl   l0, tos
            movl   l1, tos
            movl   l2, tos
            movl   l3, tos
            movl   l4, tos
            movl   l5, tos
            movl   l6, tos
            movl   l7, tos
            •
            •
            •
end:
```

FIGURE 3

NS32381 FPU Status Register (FSR)



TL/EE/10417–1

FIGURE 1

Interfacing the NS32081 as a Floating-Point Peripheral

National Semiconductor
Application Note 383
Microprocessor Applications
Engineering

This note is a guide for users who wish to interface the NS32081 Floating-Point Unit (FPU) as a peripheral unit to CPUs other than those of the Series 32000 family. This is not a particularly expensive procedure, but it requires some in-depth information not all of which is available in the NS32081 data sheet. Four basic topics will be covered here:

An overview of the architecture of the NS32081 as seen in a stand-alone environment.

The protocol used to sequence it through the execution of an instruction.

Special guidelines for connecting and programming the NS32081 as a peripheral component.

A sample application of these guidelines in the form of a circuit interfacing the NS32081 to the Motorola 68000 microprocessor.

References are made here to the NS32081 data sheet and the Series 32000 Instruction Set Reference Manual (Publication #420010099-001). The reader should have both these documents on hand.

1.0 Architecture Overview

1.1 REGISTER SET

The register set internal to the NS32081 FPU is shown in *Figure 1*. It consists of nine registers, each 32 bits in length:

FSR The Floating-Point Status Register. As given in the data sheet, this register holds status and mode information for the FPU. It is loaded by executing the LFSR instruction and examined using the SFSR instruction.

F0–F7 The Floating-Point Registers. Each can hold a single 32-bit single-precision floating-point value. To hold double-precision values, a register pair is referenced using the even-numbered register of the pair.

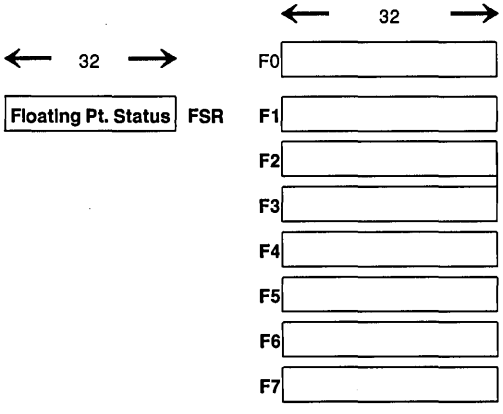


FIGURE 1. FPU Registers

Floating-point operands need not be held in registers; they may be supplied externally as part of the instruction sequence. Integer operands (appearing in conversion instructions) and values being transferred to or from the FSR must be supplied externally; they cannot be held in Floating-Point registers F0–F7.

1.2 INSTRUCTION SET AND ENCODING

The encodings used for NS32081 instructions are shown in *Figure 2*. They fall within two formats, labeled from Series 32000 tradition "Format 9" and "Format 11". These formats are distinguished by their least-significant byte (the "ID Byte"). Execution of an FPU instruction starts by passing first the ID Byte and then the rest of the instruction (the "Operation Word") to the FPU.

Fields within an instruction are interpreted by the FPU in the same manner as documented in Chapter 4 of the Series 32000 Instruction Set Reference Manual, with the exception of the 5-bit General Addressing Mode fields (*gen1, gen2*). Since the FPU does not itself perform memory accesses, it does not need to use these fields for addressing calculations. The only use it makes of these fields is to determine for each operand whether the value is to be found internal to the FPU (that is, within a register F0–F7, or whether it is to be transferred to and/or from the FPU. See *Figure 3*. A value of 0–7 in a *gen* field specifies one of the Floating-Point registers F0–F7, respectively, as the location of the corresponding operand. Any greater value specifies that the operand's location is external to the FPU and that its value will be transferred as part of the protocol. Any non-floating operand is always handled by the FPU as external, regardless of the addressing mode specified in its *gen* field. It is illegal to reference an odd-numbered register for a double-precision operand. If an odd register is referenced, the results are unpredictable.

1.3 PINOUT

The FPU is packaged in a 24-pin DIP (see *Figure 4*). The pin functions can be split into two groups: those that participate in the communication protocol between the FPU and the host system, and those that reflect the familiar requirements of LSI components.

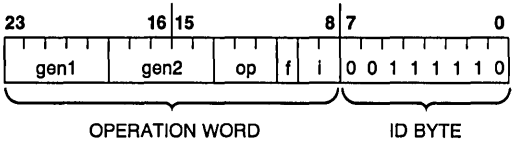
The protocol uses the following pins of the FPU:

D0–D15 The 16-bit data bus. The D0 pin holds the least-significant bit of data transferred on the bus.

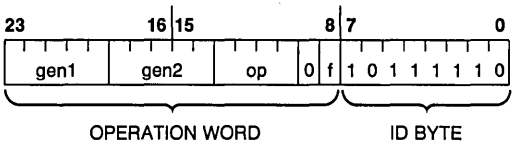
$\overline{SPC}$ A dual-purpose pin, low active. $\overline{SPC}$ is pulsed low from the host system as the data strobe for bus transfers. $\overline{SPC}$ is pulsed low by the FPU to signal that it has completed the internal execution phase of an instruction.

1.0 Architecture Overview (Continued)

ST0, ST1 The status code. This 2-bit value is sampled by the FPU on the falling edge of $\overline{SPC}$, and informs it of the current protocol phase. ST0 is the least-significant bit of the value. The need filled by the status code is most relevant to Series 32000-based systems, where it serves to allow retry of aborted instructions and to disambiguate the protocol when the $\overline{SPC}$ signal is bussed among multiple slave processors. In microprocessor-based peripheral applications, the status code can generally be provided from the CPU's address lines.



Format 9: LFSR/SFSR/Conversions



Format 11: Movement/Calculation

FIGURE 2. FPU Instruction Formats

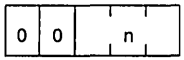
The pins providing for standard requirements are:

CLK The clock input. This is a TTL-level square wave which the FPU uses to sequence its internal calculations.

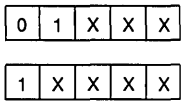
$\overline{RST}$ The reset input. This signal is used to reset the FPU's internal logic.

VCC The 5-volt positive supply.

GNDB, GNDL The grounding pins. GNDB serves as ground for the FPU's output buffers, and GNDL is used for the rest of the on-chip logic.



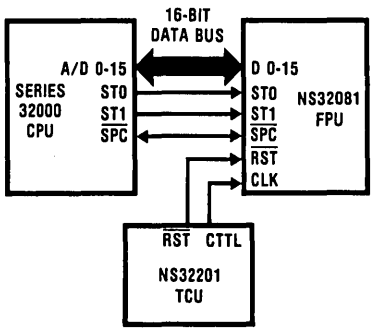
FPU Internal Register: Fn, n=0 . . . 7
Long Floating=Even Register Only



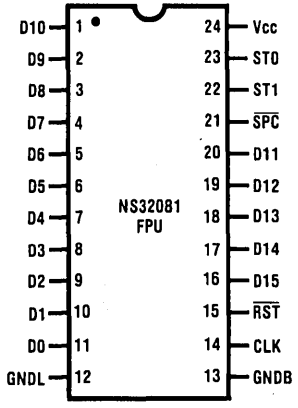
External to FPU

Note: All non-floating operands are always external.

FIGURE 3. FPU Addressing Modes



TL/EE/8388–1



Top View

TL/EE/8388–2

FIGURE 4. NS32081 FPU Connections

2.0 Protocol

The FPU requires a fixed sequence of transfers ("protocol") in its communication with the outside world. Each step of the protocol is identified by a status code (asserted to the FPU on pins ST0 and ST1) and by its position in the sequence, as shown in *Figure 5*.

Status Combinations:
- 11: Write ID Byte
- 01: Transfer Operation/Operand
- 10: Read Status Word

Step	Status	Action
1	11	CPU sends ID Byte on least-significant byte of bus.
2	01	CPU sends Operation Word, bytes swapped on bus.
3	01	CPU sends required operands, *gen1* first, least-significant word first.
4	xx	FPU starts internal execution.
5	xx	FPU pulses $\overline{SPC}$ low.
6	10	CPU reads Status Word (Error/Comparison Result).
7	01	CPU reads result (if any), least-significant word first.

FIGURE 5. FPU Instruction Protocol

Steps 1 and 2 transfer the instruction to the FPU. Step 1 transfers the first byte of the instruction (the ID Byte) and Step 2 transfers the rest of the instruction (the Operation Word). In Step 2, the two bytes of the Operation Word must be swapped on the bus; i.e. the most-significant byte of the Operation Word must be presented on the least-significant byte of the bus.

Step 3 is optional and repeatable depending on the instruction. It is used to transfer to the FPU any external operands that are required by the instruction. The operand specified by *gen1* is sent first, least-significant word first, followed by the operand specified by *gen2*. If an operand is only one byte in length, it is transferred on the least-significant half of the bus.

The FPU initiates Step 4 of the protocol, internal computation, upon receiving the last external operand word or, if there are no external operands, upon receiving the Operation Word of the instruction. During this time, the data bus may be used for any purpose by the rest of the system, as long as the $\overline{SPC}$ pin is kept pulled up by a resistor and is not actively driven.

Step 5 occurs when the FPU completes the instruction. The FPU pulses the $\overline{SPC}$ pin low to acknowledge that it is ready to continue the protocol. This pulse is called the "Done pulse". The bus is not used during this step, and remains floating.

In Step 6, the FPU is polled by reading a Status Word. This word indicates whether an exception has been detected by the FPU. In the Compare instruction (CMPf), it also displays the relationship between the operands and serves as the result. This transfer is mandatory, regardless of whether the information presented by the FPU is intended to be used. See *Figure 3-6* of the data sheet.

Step 7 is, like Step 3, optional and repeatable depending on the instruction. Any external result of an instruction is read from the FPU in this step, least-significant word first. If the result is a 1-byte value, it is presented by the FPU on the least-significant half of the bus (D0–D7).

Note: If in Step 6 the FPU indicates that an error has occurred, it is permissible, though not necessary, to continue the protocol through Step 7. No guarantee is made regarding the validity of the value read, but continuing through Step 7 will not cause any protocol problems.

If at any time within the protocol another ID byte is sent (ST = 11), the FPU will prepare itself internally to execute another instruction, throwing away the instruction that was in progress. This is done to support the Abort with Retry feature of the Series 32000 family.

Because of this feature, however, there is an important consideration when using the FPU in systems that support multitasking: the operating system must not allow a task using the FPU to be interrupted in the middle of an instruction protocol and then transfer control to another task that is also using the FPU. The partially-executed instruction would be thrown away, leaving the first task with a garbage result when it continues. This situation can be avoided easily in software but, depending on the system, some cooperation may be required from the user program. Other solutions involving some additional hardware are also possible.

3.0 Interfacing Guidelines

There are some special interfacing considerations that are required (see *Figure 6*):

1. The edges of the $\overline{SPC}$ pulse must have a fixed relationship to the clock signal (CLK) presented to the FPU. When writing information to the FPU, the pulse must start shortly after a rising edge of CLK and end shortly after the next rising edge of CLK. Failing to do so can cause the FPU to fail, often by causing it to freeze and not generate the Done pulse. This synchronous generation of $\overline{SPC}$ is also important when reading information from the FPU, but the $\overline{SPC}$ pulse is allowed to be two clocks in width. These requirements will be expressed in future NS32081 data sheets as a minimum setup time requirement between each edge of the $\overline{SPC}$ pulse and the next rising edge of CLK, currently set at 40 nanoseconds on the basis of preliminary characterization. The propagation delay in generating $\overline{SPC}$ through a Schottky flip-flop (e.g. 74S74) and a low-power Schottky buffer (e.g. 74LS125A) is therefore acceptable at 10 MHz. LS technology is recommended for the buffer to minimize undershoot when driving $\overline{SPC}$.
2. After the FPU generates the Done pulse, it is necessary to leave the $\overline{SPC}$ pin high for an additional two cycles of CLK before performing the Read Status Word transfer.
3. After performing the Read Status Word transfer, it is necessary to wait for an additional three cycles of CLK before reading a result from the FPU.

4.0 An Interface to the MC68000 Microprocessor

4.1 HARDWARE

A block diagram of the circuitry required to interface the MC68000 MPU to the NS32081 is shown in *Figure 7*.

First the easy part. Direct connections are possible on the data bus, which is numbered compatibly (D0–D15 on both parts), the status pins ST0–ST1 (connected to address lines A4–A5 from the 68000), and the clock (CLK on both). The system reset signal ($\overline{\text{RESET}}$ to and/or from the MC68000) should be synchronized with the clock before presenting it as $\overline{\text{RST}}$ to the FPU.

All that remains to be done is to generate $\overline{\text{SPC}}$ pulses that are within specifications whenever the 68000 accesses the FPU, and to detect the Done pulse from the FPU in a manner that will allow the 68000 to poll for it.

The approach selected for generating $\overline{\text{SPC}}$ pulses uses an address decoder that recognizes two separate address spaces; one to transfer information to or from the FPU ($\overline{\text{XFER}}$), and one to poll for the Done pulse ($\overline{\text{POLL}}$).

The 68000 signals $\overline{\text{AS}}$ (Address Strobe) and R/$\overline{\text{W}}$ (Read / not Write) are used to generate $\overline{\text{SPC}}$ timing.

Figure 8 shows the timing generated when the 68000 is writing to the FPU. The $\overline{\text{SPC}}$ pin is kept floating (held high by a pullup resistor) until bus state S4, at which point it is pulled low. On the next rising edge of CLK, $\overline{\text{SPC}}$ is actively pulled high, and is set floating afterward. It is not simply allowed to float high, as the resulting rise time can be unacceptable at speeds above about 4 MHz. A timing chain, required due to the 10-MHz 68000's treatment of its $\overline{\text{AS}}$ strobe, generates the signals TA, TB and TC, from which the $\overline{\text{SPC}}$ signal's state and enable are controlled.

Figure 9 shows the $\overline{\text{SPC}}$ timing for reading from the FPU. The basic difference is that $\overline{\text{SPC}}$ remains active for two clocks, so that the FPU holds data on the bus until it is sampled by the 68000. Again, $\overline{\text{SPC}}$ is actively driven high before being released.

Note: Although $\overline{\text{SPC}}$ must be driven high before being released, it must not be actively driven for more than two clocks after the trailing edge of $\overline{\text{SPC}}$. This is because the FPU can respond as quickly as three clocks after that edge with a Done pulse.

A simpler scheme in which the $\overline{\text{SPC}}$ pulse is identical for both reading and writing (1-clock wide always, but starting ½ clock later with CLK into the FPU inverted) was considered, but was rejected because the data hold time presented by the 68000 on a Write cycle would be inadequate at 10 MHz.

Any $\overline{\text{SPC}}$ pulse appearing while the $\overline{\text{XFER}}$ Select signal is inactive is interpreted as a Done pulse, which is latched in a flip-flop within the Done Detector block. When the 68000 performs a Read cycle from the address that generates the $\overline{\text{POLL}}$ select signal, the contents of the flip-flop are placed on data bus bit D15. Since this is the sign bit of a 16-bit value, the 68000 can perform a fast test of the bit using a MOVE.W instruction and a conditional branch (BPL) to wait for the FPU.

The schematic for the $\overline{\text{SPC}}$ generator and the Done pulse detector is given in *Figures 10a* and *10b*. The flip-flop labeled SPC generates the edges of the $\overline{\text{SPC}}$ pulse (on the signal $\overline{\text{SPCT}}$). The timing chain (TA, TB) provides the enable control to the buffer driving $\overline{\text{SPC}}$ to the FPU, as well as the signal to terminate the $\overline{\text{SPC}}$ pulse (either TB or TC, depending on the direction of the data transfer). Note that the timing chain assumes a full-speed memory cycle of four clocks in accessing the FPU, and will fail otherwise. The circuit generating the Data Acknowledge signal to the 68000 (DTACK, not shown) must guarantee this. In any system that must use a longer access, some modification to the timing chain will be necessary.

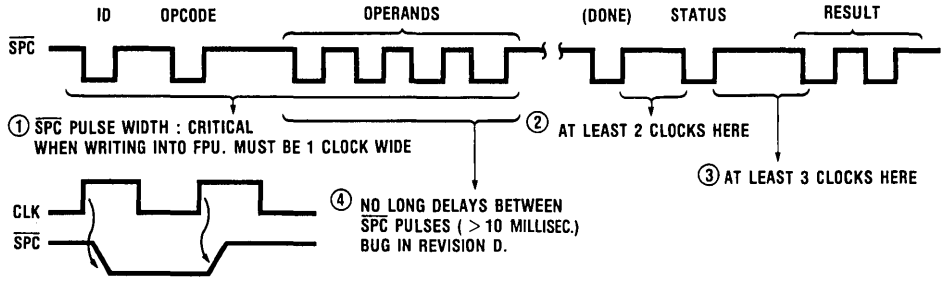
The flip-flop labeled DONE (*Figure 10b*) is the Done pulse detector. It is cleared by performing a data transfer into the FPU and is set by a Done pulse on $\overline{\text{SPC}}$. A buffer, enabled by the $\overline{\text{POLL}}$ select signal, connects its output to data bus bit 15.

4.2 SOFTWARE

Some notes on programming the FPU in a 68000 environment:

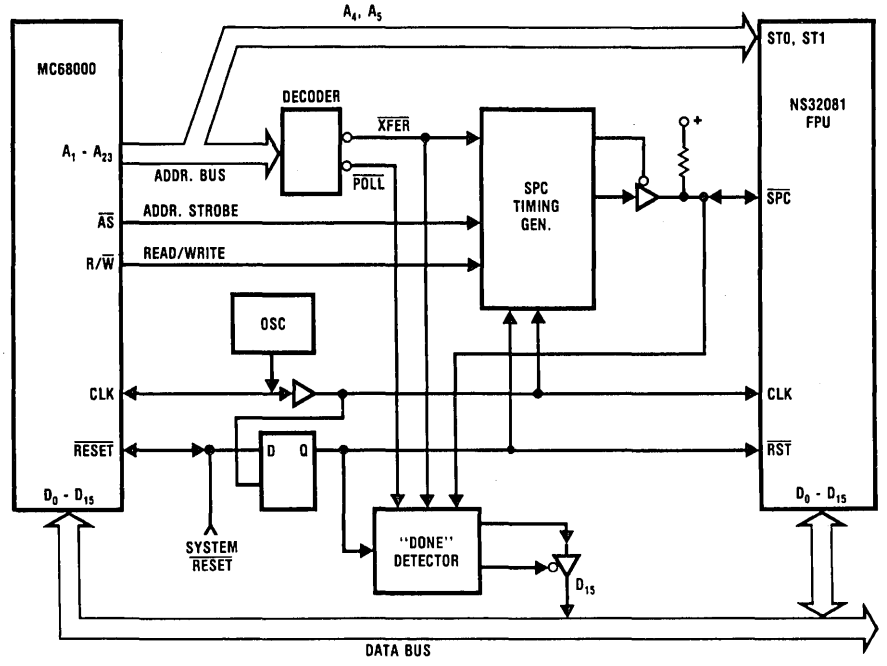
1. The byte addressing convention in the 68000 differs from that of the Series 32000 family. In particular, a byte with an even address is transferred on the most-significant half of the bus by the 68000, but the FPU expects to see it on the least-significant byte. When transferring a single byte to or from the FPU, either do so with an odd address specified, or transfer the byte as the least-significant half of a 16-bit value at an even address.
2. The 68000 transfers 32-bit operands by sending the most-significant 16 bits first. The FPU expects values to be transferred in the opposite order. Make certain that operands are transferred in the correct order (the 68000 SWAP instruction can be helpful for this).

A sample program that sequences the FPU through the execution of an ADDF instruction is listed in *Figure 11*. As this example is intended for clarity rather than efficiency, improvements are possible. The $\overline{\text{XFER}}$ select is assumed to be generated by addresses of the form 06xxxx (hex) and the $\overline{\text{POLL}}$ select is assumed to be generated by addresses of the form 07xxxx.



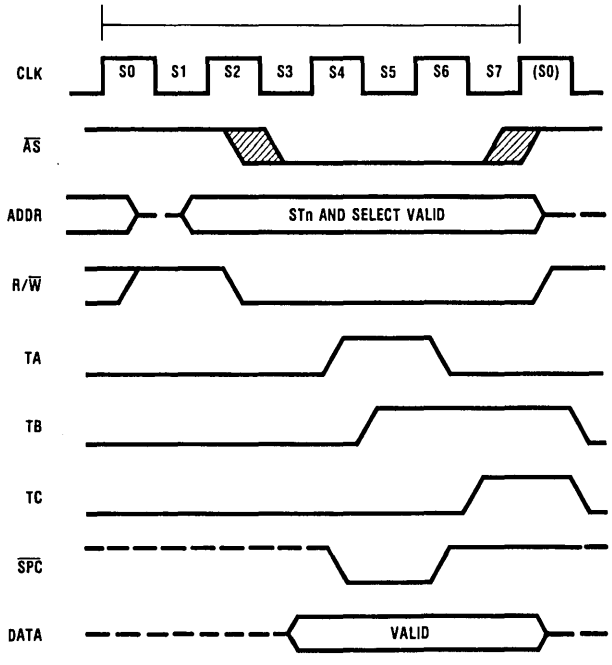
TL/EE/8388–3

FIGURE 6. Interfacing to FPU: Cautions



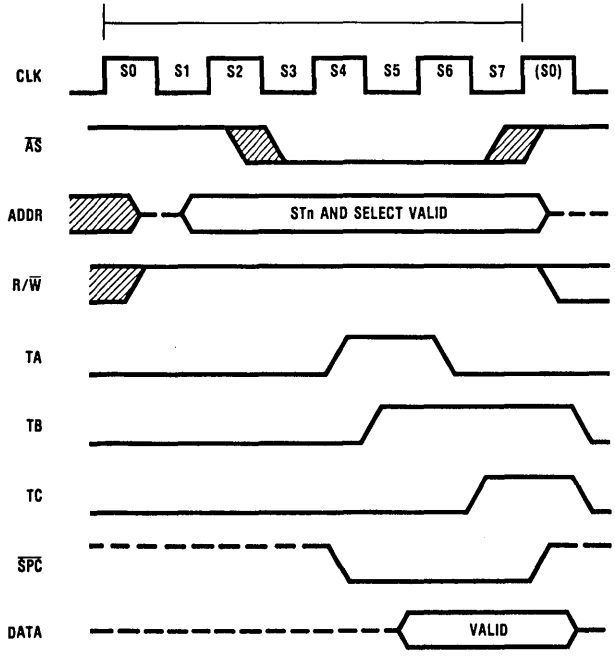
TL/EE/8388–4

FIGURE 7. 68000-32081 Interface Block Diagram



TL/EE/8388-5

FIGURE 8. 68000 Write to FPU



TL/EE/8388-6

FIGURE 9. 68000 Read from FPU

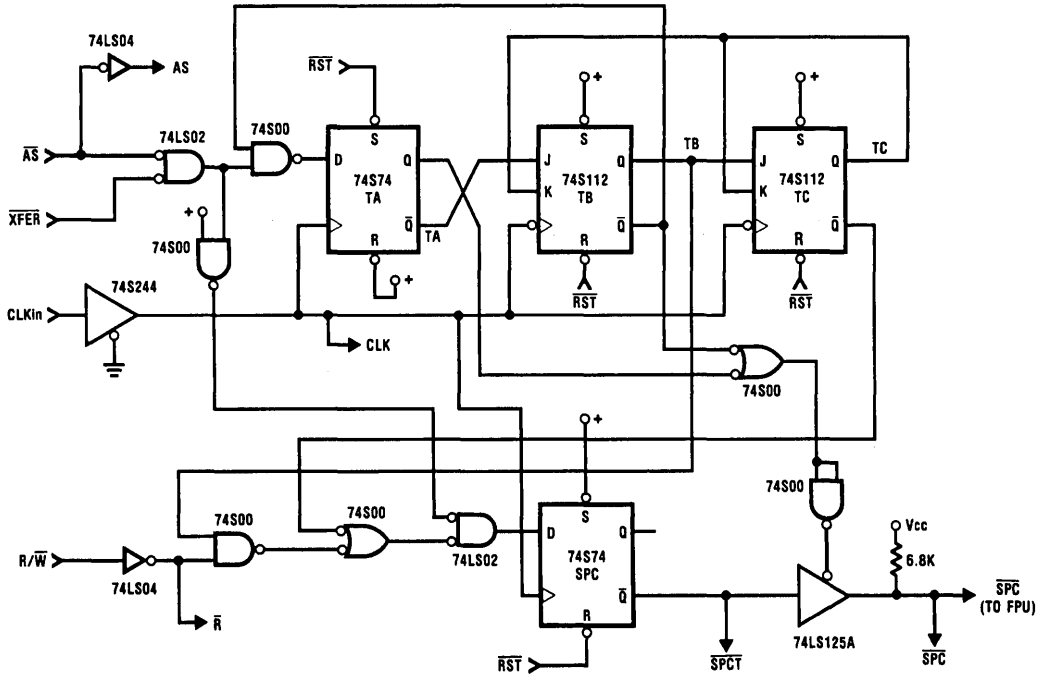


FIGURE 10a. Schematic: $\overline{SPC}$ Timing Generator

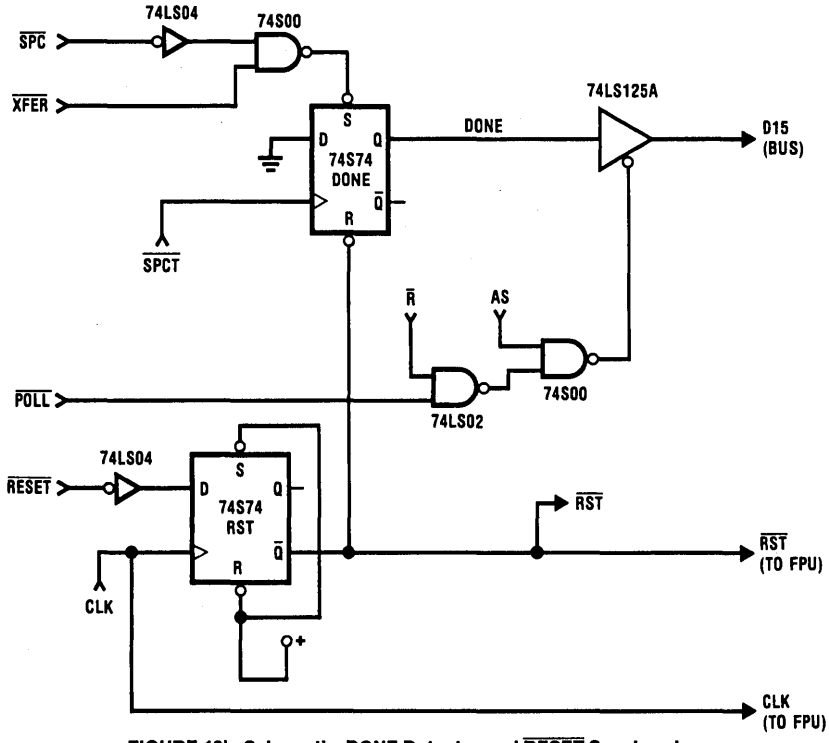


FIGURE 10b. Schematic: DONE Detector and $\overline{RESET}$ Synchronizer

```
*     Register Contents:
*
*     A0 = 00070000     Address of DONE flip-flop.
*     A1 = 00060010     Address for ST=1 transfer (Transfer Operand).
*     A2 = 00060020     Address for ST=2 transfer (Read Status Word).
*     A3 = 00060030     Address for ST=3 transfer (Broadcast ID).
*
*     D0 = 000000BE     ID byte for ADDF instruction.
*     D1 = 00000184     Operation Word for ADDF. (Note bytes swapped.)
*     D2 = 3F800000     First operand = 1.0.
*     D3 = 3F800000     Second operand = 1.0.
*     D4                Receives Status Word from FPU.
*     D5                Receives result from FPU.
*     D7                Scratch register (for DONE bit test).
*
START MOVE.W  D0,(A3)         Send ID byte.
      MOVE.W  D1,(A1)         Send Operation Word.
      SWAP    D2              Send operands. The swapping
      MOVE.L  D2,(A1)          is included because the
      SWAP    D2               FPU expects the least-
      SWAP    D3               significant word first.
      MOVE.L  D3,(A1)          (Can be avoided, with care.)
      SWAP    D3
*
POLL  MOVE.W  (A0),D7         Check the DONE flip-flop,
      BPL     POLL             loop until FPU is finished.
*                              (DONE bit is sign bit, tested
*                              by the MOVE instruction.)
*
      MOVE.W  (A2),D4         Read Status Word.
      MOVE.L  (A1),D5         Read result.
      SWAP    D5              Swap halves of result.
```

FIGURE 11. Single-Precision Addition (Demo Routine)

Using Dynamic RAM With Series 32000® CPUs

National Semiconductor
Application Note 405
Microprocessor Applications
Engineering

Recent advances in semiconductor technology have led to high-density, high-speed, low-cost dynamic random access memories (DRAMs), making large high-performance memory systems practical. DRAMs have complex timing and refresh requirements that can be met in different ways, depending on the size, speed, and processor interface requirements of the memory being designed. For low or intermediate performance, off-the-shelf components like the DP8419 can be used with a small amount of random logic. For higher performance, specialized high-speed circuitry must be designed

This application note presents the results of a timing analysis, and describes a DRAM interface for the NS32016 optimized for speed, simplicity and cost.

A future application note will discuss such features as error detection and correction, scrubbing, page mode and/or nibble mode support, in conjunction with future CPUs, such as the NS32332.

TIMING ANALYSIS RESULTS

Figures 1 and *2* show the number of CPU wait states required during a DRAM access cycle, for different CPU clock frequencies and DRAM access times.

Figure 1 is related to a DRAM interface using the DP8419 DRAM controller. Descriptions of the circuitry for use with the DP8419 and related timing diagrams are omitted. See the "DP8400 Memory Interface Family Applications" book for details.

Figure 2 shows the same data for a DRAM interface using standard TTL components, specially designed for the NS32016.

The special-purpose interface requires fewer wait states than the DP8419-based interface, especially at high frequencies.

These results assume a minimum amount of buffering between DRAM and CPU.

The results do not apply when CPU and DRAM reside on different circuit boards communicating through the system bus, since extra wait states may be required to provide for synchronization operations and extra levels of buffering.

INTERFACE DESCRIPTION

The DRAM interface presented here has been optimized for overall access time, while requiring moderate speed DRAMs, given the CPU clock frequency.

This may be significant when a relatively large DRAM array must be designed since a substantial saving can be achieved.

The result of these considerations has been the design of a high-speed DRAM interface capable of working with a CPU clock frequency of up to 15-MHz and 100-nsec DRAM chips, without wait states.

The only assumption has been that the DRAM array is directly accessible through the CPU local bus.

RAM Access Time In nsec	CPU Wait States Required							
250	0	1	1	1	1	2		
200	0	0	1	1	1	1	2	2
150	0	0	0	0	1	1	1	1
120	0	0	0	0	0	1	1	1
100	0	0	0	0	0	0	1	1
	6	7	8	9	10	11	12	13

CPU Clock Frequency in MHz

FIGURE 1. Memory Speed vs. CPU Wait States When Using the DP8419 DRAM Controller

RAM Access Time In nsec	CPU Wait States Required									
250	0	0	1	1	1	1				
200	0	0	0	0	1	1	1	1		
150	0	0	0	0	0	0	1	1	1	1
120	0	0	0	0	0	0	0	0	1	1
100	0	0	0	0	0	0	0	0	0	0
	6	7	8	9	10	11	12	13	14	15

CPU Clock Frequency in MHz

FIGURE 2. Memory Speed vs. CPU Wait States When Using Random Logic

This configuration presents some speed advantages; for example, the amount of buffering interposed between CPU and DRAM array is minimal. This translates into shorter propagation delays for address, data and other relevant signals.

Another advantage is that the interface can work in complete synchronization with the CPU. This significantly improves performance since no time is spent for synchronization. Reliability also improves since the possibility of metastable states in synchronizing flip-flops is eliminated.

A block diagram of the DRAM interface is shown in *Figure 3*. *Figures 4* through *7* show circuit diagrams and timing diagrams.

Interface operation details follow.

$\overline{\text{RAS}}$ AND $\overline{\text{CAS}}$ GENERATION

This is the most critical part of the entire interface circuit. To avoid wait states during a CPU read cycle, the DRAM must provide the data before the falling edge of clock phase PH12 during state T3. This requires that the $\overline{\text{RAS}}$ signal be generated early in the CPU bus cycle to meet the DRAM access time. On the other hand, the $\overline{\text{RAS}}$ signal can be asserted only after the row address is valid and the $\overline{\text{RAS}}$ precharge time from a previous CPU access or refresh cycle has elapsed.

The interface circuit shown in *Figures 4* and *5* relies on two advanced clock signals obtained from CTTL through a delay line and some standard TTL gates.

The advanced clock signals, CTTLA and $\overline{\text{CTTLB}}$, are used to clock the circuit that arbitrates between CPU access requests and refresh requests. The $\overline{\text{CTTLB}}$ signal is also used to enable an advanced $\overline{\text{RAS}}$ generation circuit, which causes the $\overline{\text{RAS}}$ signal to be asserted earlier than the CPU access-grant signal from the arbitration circuit. This speeds up the $\overline{\text{RAS}}$ signal by about 10 ns by avoiding the time required by the arbitration circuit to change state.

A different delay line is used to generate the $\overline{\text{CAS}}$ signal and to switch the multiplexers for the column addresses. Note that the $\overline{\text{CAS}}$ signal during write cycles is delayed until the beginning of CPU state T3, to guarantee that the data being written to the DRAM is valid at the time $\overline{\text{CAS}}$ is asserted. The $\overline{\text{CAS}}$ signal is deasserted after the trailing edge of $\overline{\text{RAS}}$ to guarantee the minimum pulse width requirement.

The timing diagrams in *Figures 6* and *7* show the signal sequences for both read and write cycles.

ADDRESS MULTIPLEXING

The multiplexing of the various addresses for the DRAM chips is accomplished via four 74AS153 multiplexer chips in addition to some standard TTL gates used to multiplex the top two address bits needed for 256k DRAMs. The resulting nine address lines are then buffered and sent to the DRAMs through series damping resistors. The function of these resistors is to minimize ringing.

REFRESH

The refresh circuitry includes an address counter, a timer and a number of flip-flops used to generate the refresh cycle and to latch the refresh request until the end of the refresh cycle.

The address counter is an 8-bit counter implemented by cascading the two 4-bit counters of a 74LS393 chip. This counter provides up to 256 refresh addresses and is incremented at the end of each refresh cycle.

The refresh timer is responsible for generating the refresh request signal whenever a refresh cycle is needed. This timer is implemented by cascading two 4-bit counters. Both counters are clocked by the $\overline{\text{CTTLB}}$ signal; the first is a presettable binary counter that divides the clock signal by a specified value; the second can be either a BCD or a binary counter depending on the CPU clock frequency.

With this arrangement, a refresh request is generated after a fixed time interval from the previous request, regardless of the CPU activity. A more sophisticated circuit that generates requests when the CPU is idle could also be implemented. However, such a circuit has not been considered here because the performance degradation due to the refresh is relatively small (less than 3.3 percent), and the improvement attainable by using a more sophisticated circuit would not justify the extra hardware required.

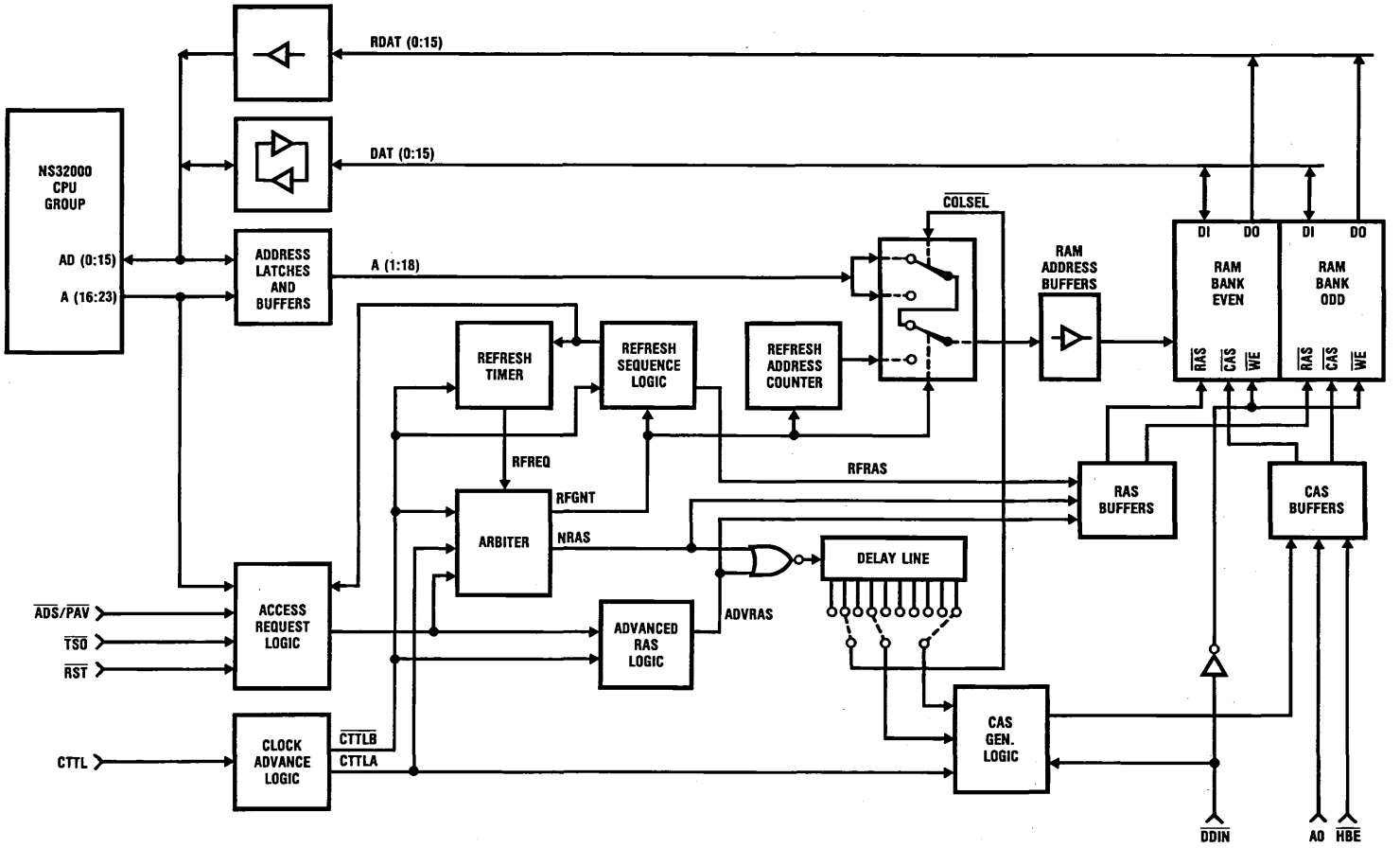
CONCLUSIONS

The DRAM interface described in this application uses two TTL-buffered delay lines to obtain speed advantages. One delay line is used to time the $\overline{\text{CAS}}$ signal and to enable the column address. The other is used to generate the advanced clock signals from CTTL.

Below 10 MHz, the advanced clocks might not be required, and the related delay line can be eliminated. When this is done, however, higher speed DRAMs must be used. If, on the other hand, advanced clocks must be used for frequencies lower than 10 MHz, a delay line with a larger delay (e.g. DDU-7J-100) might be needed.

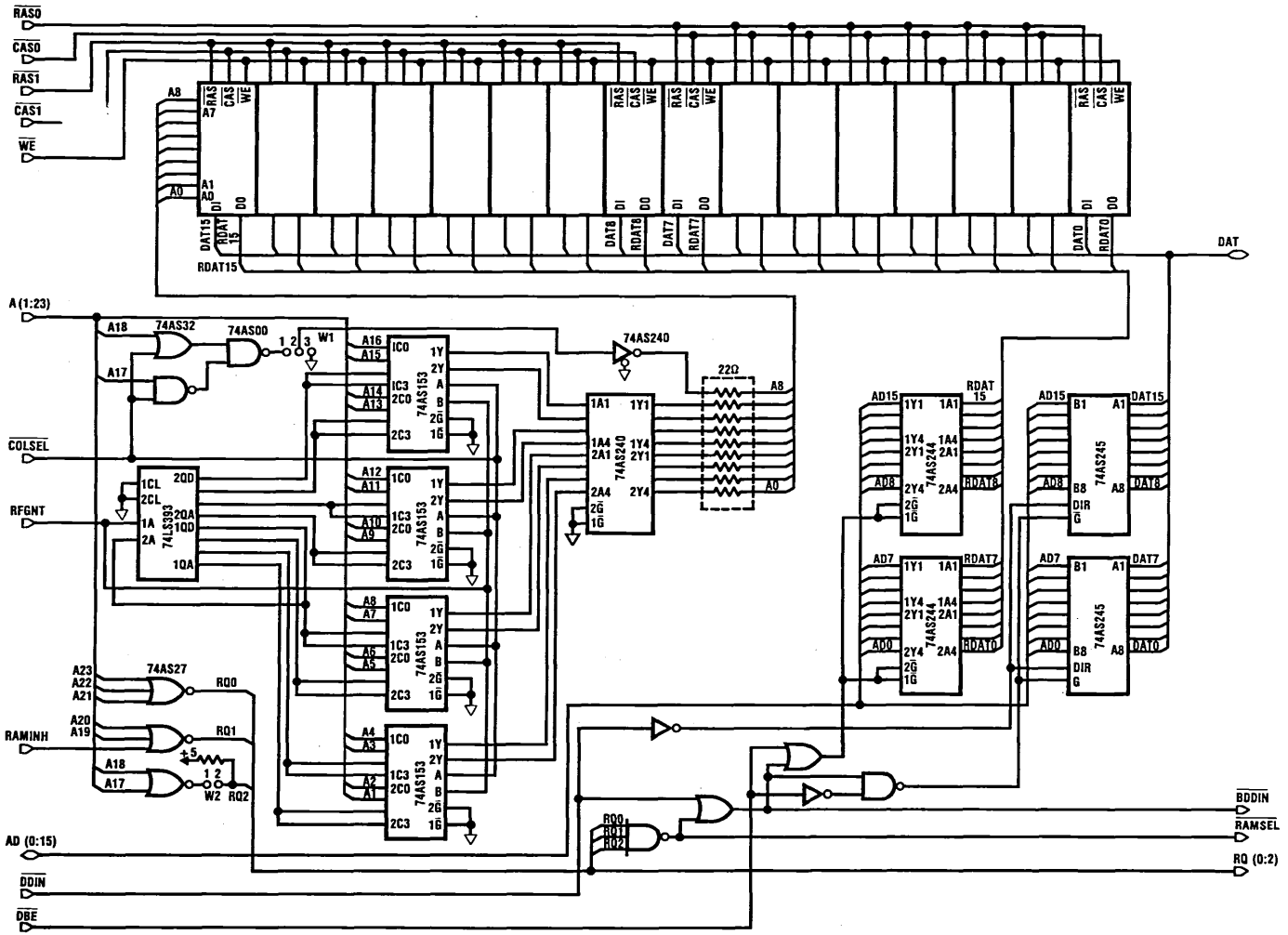
Delay lines are extremely versatile for this kind of application due to their accuracy and the fact that different delays are easily available to accommodate different DRAM types.

The savings attainable by using slower DRAM chips, in addition to the reliability improvement and cleaner design, make delay lines a valid alternative, even though their cost is relatively high in comparison to standard TTL gates.



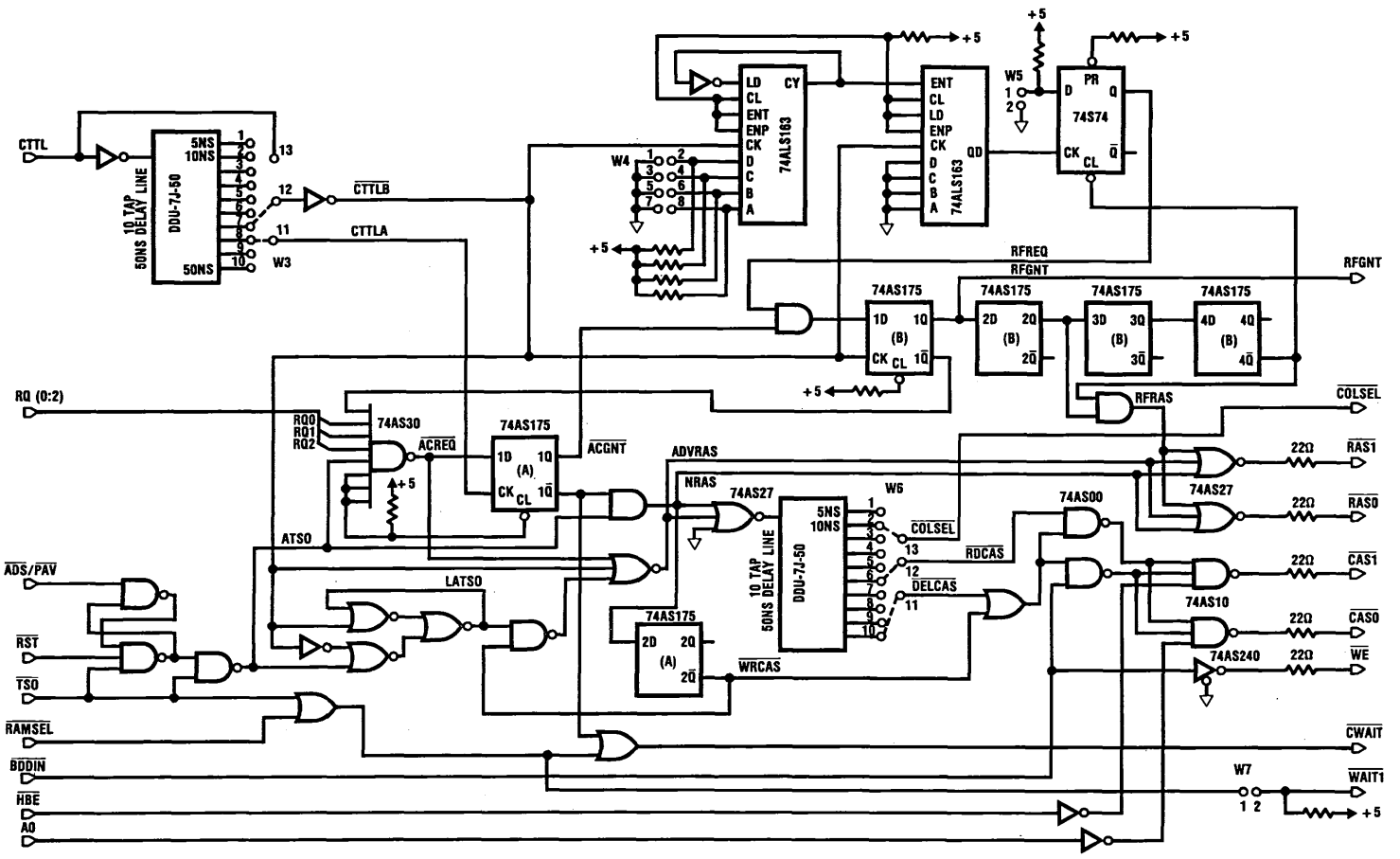
TL/EE/8517–1

FIGURE 3. DRAM Interface Block Diagram



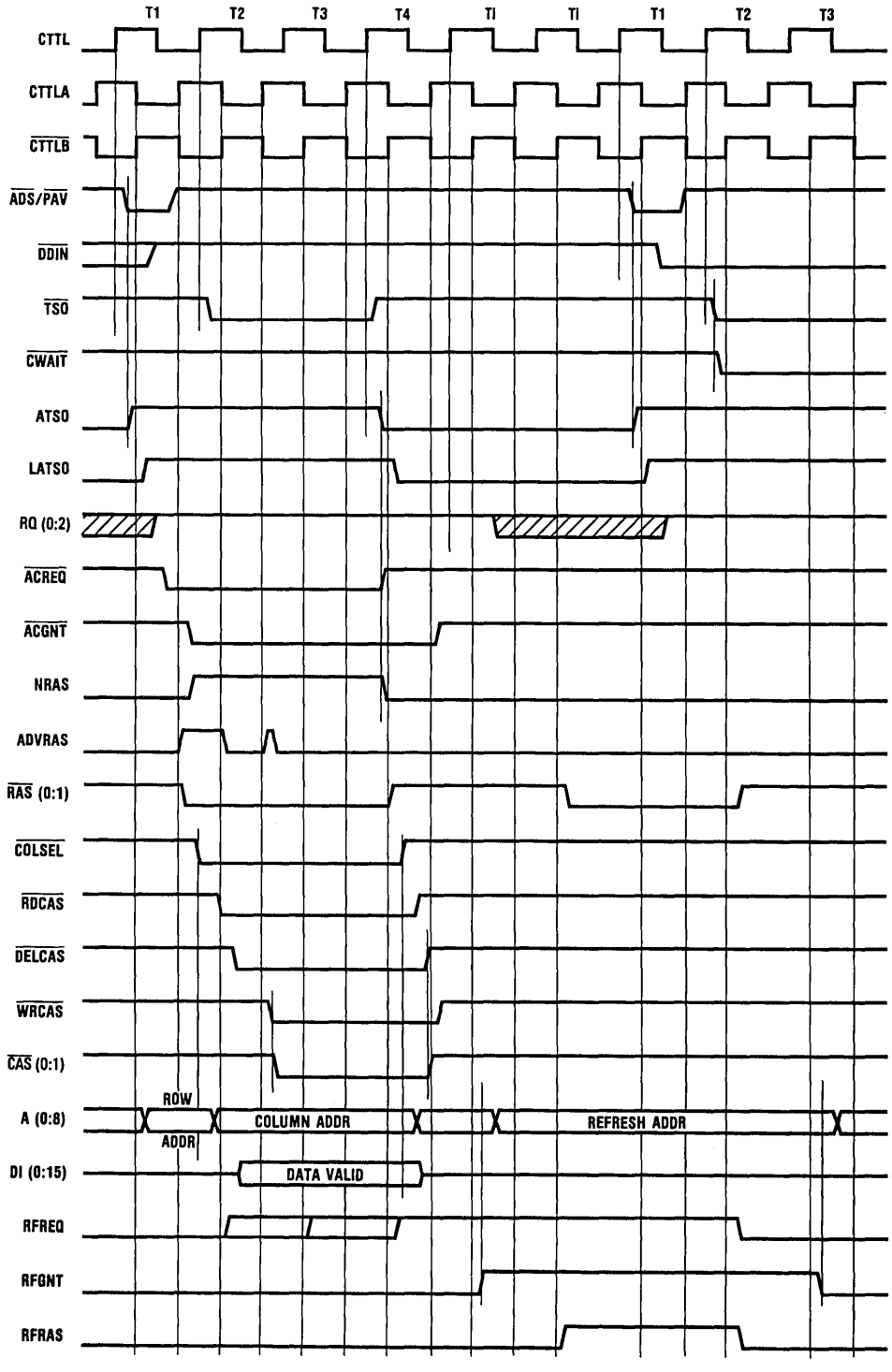
TL/EE/8517–2

FIGURE 4. DRAM Interface Circuit Diagram (a)



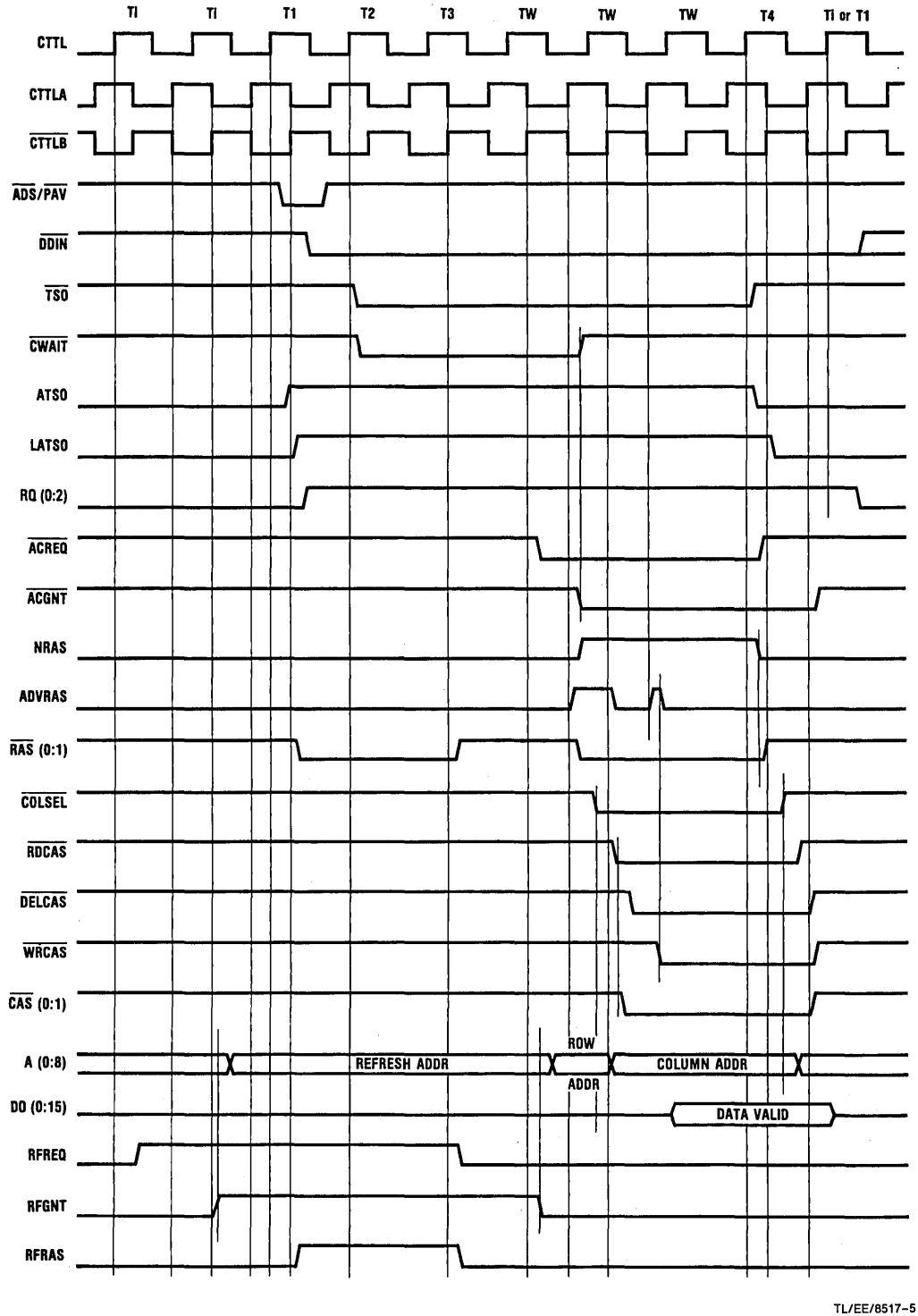
TL/EE/8517–3

FIGURE 5. DRAM Interface Circuit Diagram (b)



TL/EE/8517–4

FIGURE 6. Write Cycle Followed By a Refresh Cycle



TL/EE/8517–5

FIGURE 7. Refresh Cycle Followed By a Read Cycle

Effects of NS32082 Memory Management Unit on Processor Through Put

National Semiconductor
Application Note 464
Chris Siegl

INTRODUCTION

The purpose of this application note is to give a satisfactory answer to the question, "How great is the performance penalty for using the NS32082 memory management unit?" To arrive at a satisfactory answer a number of benchmarks have been run on the DB32000 board using the NS32032 with and without the NS32082 as well as the NS32016 with and without the NS32082. The benchmarks were compiled on two different compilers to show the differing effects of the MMU based on the degree of code optimization. The results are tabulated in a table along with the percent performance penalty.

The results show that the percentages vary over the wide range of 6% to 18.5% with generally a greater MMU impact with higher levels of code optimization in the compiler. The Whetstone benchmark has also been included to show the effects of the MMU on floating-point instructions. As can be seen in the tables the effects are much smaller with longer instructions such as the floating-point instructions. The last section of this ap-note rationalizes the differences in performance under varying conditions and gives some rules of thumb to use in applying this data to a specific case.

THE TEST SET-UP

To run this set of tests the DB32000 board was used. This board is a complete microprocessor system specifically designed to assist the user in evaluating and developing hardware and software for the NS32032 CPU, related slave processors (NS32081 FPU and NS32082 MMU) and support devices. Through the use of on board multiplexers the NS32016 and NS32008 CPU's can also be run on this board. The configuration of this board used for these tests consist of the NS32081 FPU (floating point unit), the NS32202 ICU (interrupt control unit), 256K of dynamic RAM, extensive ROM/EPROM capability, and two serial RS-232 ports as well as a parallel I/O port. See the DB32000 data sheet for more detailed information.

The TDS monitor (shipped installed on the DB32000 board) was then removed and replaced with MON32. This monitor is compatible with National's DBG16 debugger and allows downloading of code from a host computer through the debugger using an RS-232 link therefore allowing the host machine to be remote from the development environment. This can even be done over a modem line to the host.

A timing routine using the counters in the ICU was linked to the compiled benchmark programs before they were downloaded to the DB32000. A command to the debugger then started the timing program executing which in turn called the compiled benchmark after starting the ICU counters. After the benchmark completes, it returns to the timing routine where the counters are stopped and the execution time is read from the registers. This set-up and the timing program used are covered in detail in another application note titled "Using the DB32000 Evaluation Board for Benchmarking".

The SYS-32 Multi-User development system was used as the host. This system is based on the Series 32000 family, runs GENIX™ (National's version of Berkley 4.1 UNIX™) operating system in a demand paged virtual memory environment. The system supports up to eight simultaneous users, C and Pascal high level language compilers, a Series 32000 assembler, symbolic debugger and supports in-system emulation for the 32000 family. The minimum system configuration consists of 1.25 megabytes of RAM (expandable to 3.25 megabytes) 70 megabytes of hard disk (expandable to 490 megabytes) and a streamer tape drive for backup. For more detailed information on the SYS-32, please refer to the SYS-32 data sheet. The details of the DBG16 symbolic debugger's usage for down loading and execution of the benchmaks is covered in the ap-note "Using the DB32000 Evaluation Board for Benchmarks".

RESULTS

TABLES I, II and III show the results of running the benchmarks under the four different part combinations. As can be seen in tables the MMU penalty varies considerably from benchmark to benchmark and especially from one compiler to another. To set an understanding of why the variations are so big, we must look at how the 32000 family of CPU's operate in memory.

TABLE I
Benchmarks Executed on DB32000—All Processors Running at 10 MHz with no Wait States using Genix 4.1 C Compiler

Benchmark	NS32032 W MMU	NS32032 W/O MMU	MMU Penalty	NS32016 W MMU	NS32016 W/O MMU	MMU Penalty
Ackerman. c	4.72	4.32	9.3%	6.03	5.27	14.4%
BenchE. c	8.89	8.12	9.5%	11.97	10.50	14.0%
Puzzle. c	20.59	19.10	7.8%	26.96	23.65	14.0%
Sieve. c	19.42	18.09	7.4%	22.15	19.62	12.9%
Fibonacci. c	22.13	20.28	9.1%	26.31	23.61	11.4%
Longsearch. c	7.36	6.71	9.7%	10.31	8.70	18.5%

TABLE II
Benchmarks Executed on DB32000—All Processors Running at 10 MHz with no Wait States using Greenhill's C-32000 1.6.8 Compiler

Benchmark	NS32032 W MMU	NS32032 W/O MMU	MMU Penalty	NS32016 W MMU	NS32016 W/O MMU	MMU Penalty
Ackerman. c	3.75	3.30	13.6%	5.06	4.37	15.8%
BenchE. c	4.44	4.00	11.0%	4.76	4.48	6.3%
Puzzle. c	7.82	7.09	10.3%	9.61	8.57	12.1%
Sieve. c	17.71	16.41	7.9%	19.65	17.89	9.9%
Fibonacci. c	18.34	16.47	11.4%	24.87	21.17	17.5%
Longsearch. c	6.77	5.97	13.4%	8.75	7.48	17.0%

TABLE III
Benchmarks Executed on DB32000—All Processors Running at 10 MHz with no Wait States using Genix 4.1 Pascal Compiler

Benchmark	NS32032 W MMU	NS32032 W/O MMU	MMU Penalty	NS32016 W MMU	NS32016 W/O MMU	MMU Penalty
Whetstone. P	5.08	4.83	5.2%	6.17	5.63	9.6%

Both the NS32032 and the NS32016 have an eight byte queue for instruction prefetching. As a result of this queue having an MMU in the system has little effect on instruction fetching. An interesting test that helps in understanding this is to add wait states only to the code segment while using no waitstate RAM for the stacks and static data segments. These tests show a performance degradation of only 2 or 3% per waitstate. Another approach to demonstrating the same effect which is not dependent on a special hardware setup (controlling the number of wait states on different areas of memory space is done in hardware) is to generate a software loop which only uses the registers and immediate data for holding operands. A short example of such a program is shown in listing 1. Table IV shows the results obtained from timing this program both with and without the MMU. As can be seen from the times the penalty is very small, much less than 1%. This example clearly demonstrates that the queue is doing a good job of minimizing the effects of the MMU or waitstates on intruction fetching. This is why, even though the MMU lengthens each memory cycle by 25% (memory cycle goes from 4 t-states to 5) the net effect on performance is typically less than 10%. The penalty comes primarily from the lengthening of operand fetches. The NS32032 takes a much smaller penalty if the operands are primarily 32 bits or more in length. In that case the NS32032 is only doing half as many operand fetches as the NS32016, which has to do two accesses to get 32 bit operands. Another thing to note is that the performance times between NS32032 and the NS32016 is less than 1% in our software program loop test (see Table IV). This is because both processors are internally identical except in the queue and bus interface. If the queue keeps up and there are no stack or memory reference operations the execution time would be identical. The difference in time in this test is due to the queue not quite keeping up and the branch which purges the queue which the NS32032 reloads twice as fast.

TABLE IV
Benchmarks Executed on DB32000—All Processors Running at 10 MHz with No Wait States
(times are in microseconds)

Benchmark	NS32032 W MMU	NS32032 W/O MMU	MMU Penalty	NS32016 W MMU	NS32016 W/O MMU	MMU Penalty
Progloop.b.s	12622	12559	0.50%	12750	12668	0.65%
Progloop.w.s	13344	13291	0.40%	13432	13350	0.61%
Progloop.d.s	14988	14939	0.33%	15075	14992	0.55%

Tables I and II are the results of two different compilers using the same source files for input but generating code at different levels of optimization. The compiler in Table II optimizes to a much greater degree resulting in a much smaller ratio of instruction fetches to operand fetches while the table one compiler generates more code to do the same work. The number of operands does not decrease through optimization but extraneous code is eliminated, driving down the code to operand fetch ratio. As a result the penalty rises but is still in the neighborhood of 10%. The greater the complexity of the instruction the smaller the MMU penalty because the queue is more likely to keep up and a larger ratio of execution time to operands fetched especially with the NS32032. Table III gives the results of the Whetstone benchmark which illustrates this. The Whetstone benchmark is primarily floating point, the big NS32032 advantage comes from the operands being 32 or 64 bits in length. The NS32016 is making two times as many operand memory references as the NS32032 and therefore gets two times the MMU penalty.

CONCLUSIONS

After studying the above tests we can see the major factor effecting the performance penalty due to the MMU is the number of operand references and stack operations per unit of time. If operands are typically longer than 16 bits or the stack is heavily used, the NS32032 will show a much lower MMU penalty than the NS32016. However, even for the NS32016 the MMU penalty is seldom greater than 15% and typically half that for the NS32032. This penalty being so small makes a strong case for using the MMU even in systems not using a bulk memory device and benefiting from the page replacement aspects. The MMU can be useful in these non bulk memory applications for protection at the page level as well as for system debugging and program maintenance. If portions of the ROM based code require changes only the ROM holding the effected page table needs to be replaced with the new code being addable in any available ROM socket. The MMU with the on board breakpoint resistors and counter can often greatly simplify isolating bugs in the field where system disassembly on an ISE (In System Emulator) would be out of the question or inconvenient.

In bulk memory based systems there is no question that the performance improvements due to the MMU far outweigh the performance lost due to a longer memory cycle. For more details in this area see the technical note entitled "Series 32000 The Benefits of Demand Paged Virtual Memory".

LISTING 1

```
;#########################################################################
;       INLINE CODE LOOP
;       12-10-85 by Chris Siegl
;       all operands in registers
;#########################################################################
;       progloop.b.s = i's replaced by b at end of instructions - operands
;                      are bytes (8 bits)
;       progloop.w.s = i's replaced by w at end of instructions - operands
;                      are words (16 bits)
;       progloop.d.s = i's replaced by d at end of instructions - operands
;                      are double-words (32 bits)
        .program
-main::
        movi    0,r0      ;set loop counter to 0 for 256 loops
        movi    9,r3      ;put bcd values in r3 & r4
        movi    9,r4
        movi    r3,r1
        movi    r3,r2
        movi    r3,r5
        movi    r3,r6
```

```
loop:
        absi    r1,r2
        addi    r1,r2
        addci   r1,r2
        addpi   r3,r4
        subpi   r3,r4
        addqi   4,r1
        ashi    4,r1
        lshi    5,r1
        roti    6,r1
        andi    r2,r5
        comi    r2,r1
        ori     r2,r1
        xori    r2,r1
        nop
        muli    r5,r6
        absi    r1,r2
        addi    r1,r2
        addci   r1,r2
        addpi   r3,r4
        subpi   r3,r4
        addqi   4,r1
        ashi    4,r1
        lshi    5,r1
        roti    6,r1
        andi    r2,r5
        comi    r2,r1
        ori     r2,r1
        xori    r2,r1
        nop
        muli    r5,r6
        acbb    1,r0,loop
        rxp     0
        .endseg
```

Introduction to Bresenham's Line Algorithm Using the SBIT Instruction; Series 32000® Graphics Note 5

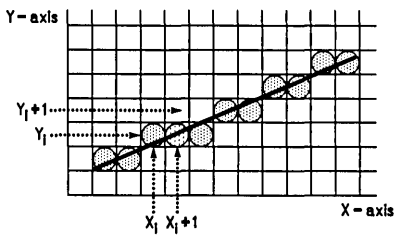
National Semiconductor
Application Note 524
Nancy Cossitt

1.0 INTRODUCTION

Even with today's achievements in graphics technology, the resolution of computer graphics systems will never reach that of the real world. A true real line can never be drawn on a laser printer or CRT screen. There is no method of accurately printing all of the points on the continuous line described by the equation $y = mx + b$. Similarly, circles, ellipses and other geometrical shapes cannot truly be implemented by their theoretical definitions because the graphics system itself is discrete, not real or continuous. For that reason, there has been a tremendous amount of research and development in the area of discrete or raster mathematics. Many algorithms have been developed which "map" real-world images into the discrete space of a raster device. Bresenham's line-drawing algorithm (and its derivatives) is one of the most commonly used algorithms today for describing a line on a raster device. The algorithm was first published in Bresenham's 1965 article entitled "Algorithm for Computer Control of a Digital Plotter". It is now widely used in graphics and electronic printing systems. This application note will describe the fundamental algorithm and show an implementation on National Semiconductor's Series 32000 microprocessor using the SBIT instruction, which is particularly well-suited for such applications. A timing diagram can be found in *Figure 8* at the end of the application note.

2.0 DESCRIPTION

Bresenham's line-drawing algorithm uses an iterative scheme. A pixel is plotted at the starting coordinate of the line, and each iteration of the algorithm increments the pixel one unit along the major, or x-axis. The pixel is incremented along the minor, or y-axis, only when a decision variable (based on the slope of the line) changes sign. A key feature of the algorithm is that it requires only integer data and simple arithmetic. This makes the algorithm very efficient and fast.



TL/EE/9665-1

FIGURE 1

The algorithm assumes the line has positive slope less than one, but a simple change of variables can modify the algorithm for any slope value. This will be detailed in section 2.2.

2.1 Bresenham's Algorithm for 0 < slope < 1

Figure 1 shows a line segment superimposed on a raster grid with horizontal axis X and vertical axis Y. Note that x_i and y_i are the integer abscissa and ordinate respectively of each pixel location on the grid.

Given (x_i, y_i) as the previously plotted pixel location for the line segment, the next pixel to be plotted is either $(x_i + 1, y_i)$ or $(x_i + 1, y_i + 1)$. Bresenham's algorithm determines which of these two pixel locations is nearer to the actual line by calculating the distance from each pixel to the line, and plotting that pixel with the smaller distance. Using the familiar equation of a straight line, $y = mx + b$, the y value corresponding to $x_i + 1$ is

$$y = m(x_i + 1) + b$$

The two distances are then calculated as:

$$d1 = y - y_i$$

$$d1 = m(x_i + 1) + b - y_i$$

$$d2 = (y_i + 1) - y$$

$$d2 = (y_i + 1) - m(x_i + 1) - b$$

and,

$$d1 - d2 = m(x_i + 1) + b - y_i - (y_i + 1) + m(x_i + 1) + b$$

$$d1 - d2 = 2m(x_i + 1) - 2y_i + 2b - 1$$

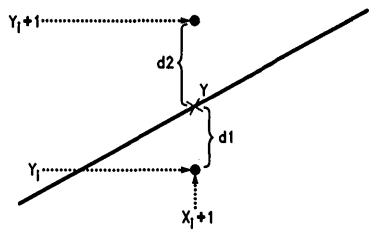
Multiplying this result by the constant dx, defined by the slope of the line $m = dy/dx$, the equation becomes:

$$dx(d1-d2) = 2dy(x_i) - 2dx(y_i) + c$$

where c is the constant $2dy + 2dxb - dx$. Of course, if $d2 > d1$, then $(d1-d2) < 0$, or conversely if $d1 > d2$, then $(d1-d2) > 0$. Therefore, a parameter p_i can be defined such that

$$p_i = dx(d1-d2)$$

$$p_i = 2dy(x_i) - 2dx(y_i) + c$$



TL/EE/9665-2

Distances d1 and d2 are compared.
The smaller distance marks next pixel to be plotted.

FIGURE 2

If $p_i > 0$, then d1 > d2 and y_{i+1} is chosen such that the next plotted pixel is $(x_i + 1, y_i)$. Otherwise, if $p_i < 0$, then d2 > d1 and $(x_i + 1, y_i + 1)$ is plotted. (See *Figure 2.*)

Similarly, for the next iteration, p_{i+1} can be calculated and compared with zero to determine the next pixel to plot. If $p_{i+1} < 0$, then the next plotted pixel is at $(x_{i+1} + 1, y_{i+1})$; if $p_{i+1} > 0$, then the next point is $(x_{i+1} + 1, y_{i+1} + 1)$. Note that in the equation for p_{i+1}, $x_{i+1} = x_i + 1$.

$$p_{i+1} = 2dy(x_i + 1) - 2dx(y_{i+1}) + c$$

Subtracting p_i from p_{i+1}, we get the recursive equation:

$$p_{i+1} = p_i + 2dy - 2dx(y_{i+1} - y_i)$$

Note that the constant c has conveniently dropped out of the formula. And, if $p_i < 0$ then $y_{i+1} = y_i$ in the above equation, so that:

$$p_{i+1} = p_i + 2dy$$

or, if $p_i > 0$ then $y_{i+1} = y_i + 1$, and

$$p_{i+1} = p_i + 2(dy-dx)$$

To further simplify the iterative algorithm, constants c1 and c2 can be initialized at the beginning of the program such that c1 = 2dy and c2 = 2(dy−dx). Thus, the actual meat of the algorithm is a loop of length dx, containing only a few integer additions and two compares *(Figure 3).*

2.2 For Slope < 0 and |Slope| > 1

The algorithm fails when the slope is negative or has absolute value greater than one (|dy| > |dx|). The reason for this is that the line will always be plotted with a positive slope if x_i and y_i are always incremented in the positive direction, and the line will always be "shorted" if |dx|<|dy| since the algorithm executes once for every x coordinate (i.e., dx times). However, a closer look at the algorithm must be taken to reveal that a few simple changes of variables will take care of these special cases.

For negative slopes, the change is simple. Instead of incrementing the pixel along the positive direction (+1) for each iteration, the pixel is incremented in the negative direction. The relationship between the starting point and the finishing point of the line determines which axis is followed in the negative direction, and which is in the positive. *Figure 4* shows all the possible combinations for slopes and starting points, and their respective incremental directions along the X and Y axis.

Another change of variables can be performed on the incremental values to accommodate those lines with slopes greater than 1 or less than −1. The coordinate system containing the line is rotated 90 degrees so that the X-axis now becomes the Y-axis and vice versa. The algorithm is then performed on the rotated line according to the sign of its slope, as explained above. Whenever the current position is incremented along the X-axis in the rotated space, it is actually incremented along the Y-axis in the original coordinate space. Similarly, an increment along the Y-axis in the rotated space translates to an increment along the X-axis in the original space. *Figure 4a., g.* and *h.* illustrates this translation process for both positive and negative lines with various starting points.

3.0 IMPLEMENTATION IN C

Bresenham's algorithm is easily implemented in most programming languages. However, C is commonly used for many application programs today, especially in the graphics area. The Appendix gives an implementation of Bresenham's algorithm in C. The C program was written and executed on a SYS32/20 system running UNIX on the NS32032 processor from National. A driver program, also written in C, passed to the function starting and ending points for each line to be drawn. *Figure 6* shows the output on an HP laser jet of 160 unique lines of various slopes on a bit map of 2,000 x 2,000 pixels. Each line starts and ends exactly 25 pixels from the previous line.

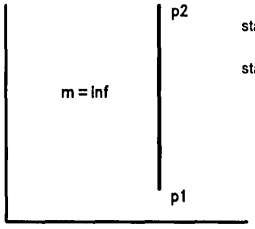
The program uses the variable *bit* to keep track of the current pixel position within the 2,000 x 2,000 bit map *(Figure 5).* When the Bresenham algorithm requires the current position to be incremented along the X-axis, the variable *bit* is incremented by either +1 or −1, depending on the sign of the slope. When the current position is incremented along the Y-axis (i.e., when p > 0) the variable *bit* is incremented by +warp or −warp, where *warp* is the vertical bit displacement of the bit map. The constant *last bit* is compared with *bit* during each iteration to determine if the line is complete. This ensures that the line starts and finishes according to the coordinates passed to the function by the driver program.

```
do while count < > dx
   if (p < 0) then p+ = c1
   else

                    p+ = c2
                    next_y = prev_y + y_inc
   next_x = prev_x + x_inc
   plot(next_x,next_y)
   count + = 1

/* PSEUDO CODE FOR BRESENHAM LOOP */
```

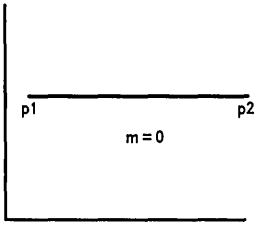
FIGURE 3



start p1: x_inc = y'_inc = 0
y_inc = x'_inc = +1
start p2: x_inc = y'_inc = 0
y_inc = x'_inc = −1

TL/EE/9665-3

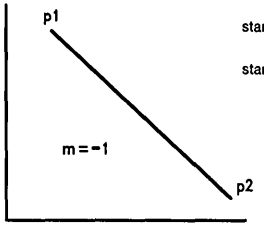
a.



start p1: x_inc = +1
y_inc = 0
start p2: x_inc = −1
y_inc = 0

TL/EE/9665-4

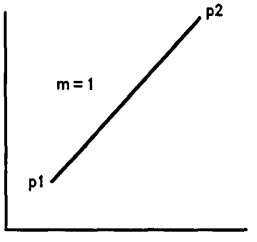
b.



start p1: x_inc = +1
y_inc = −1
start p2: x_inc = −1
y_inc = +1

TL/EE/9665-5

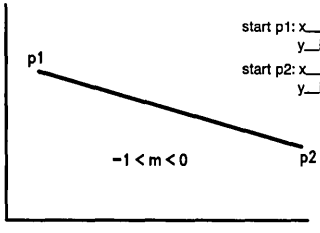
c.



start p1: x_inc = +1
y_inc = +1
start p2: x_inc = −1
y_inc = −1

TL/EE/9665-6

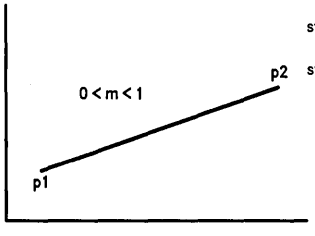
d.



start p1: x_inc = +1
y_inc = −1
start p2: x_inc = −1
y_inc = +1

TL/EE/9665-7

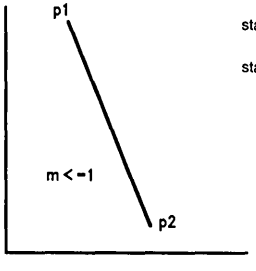
e.



start p1: x_inc = +1
y_inc = +1
start p2: x_inc = −1
y_inc = −1

TL/EE/9665-8

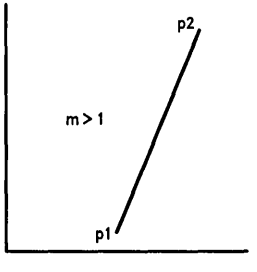
f.



start p1: x_inc = y'_inc = +1
y_inc = x'_inc = −1
start p2: x_inc = y'_inc = −1
y_inc = x'_inc = +1

TL/EE/9665-9

g.



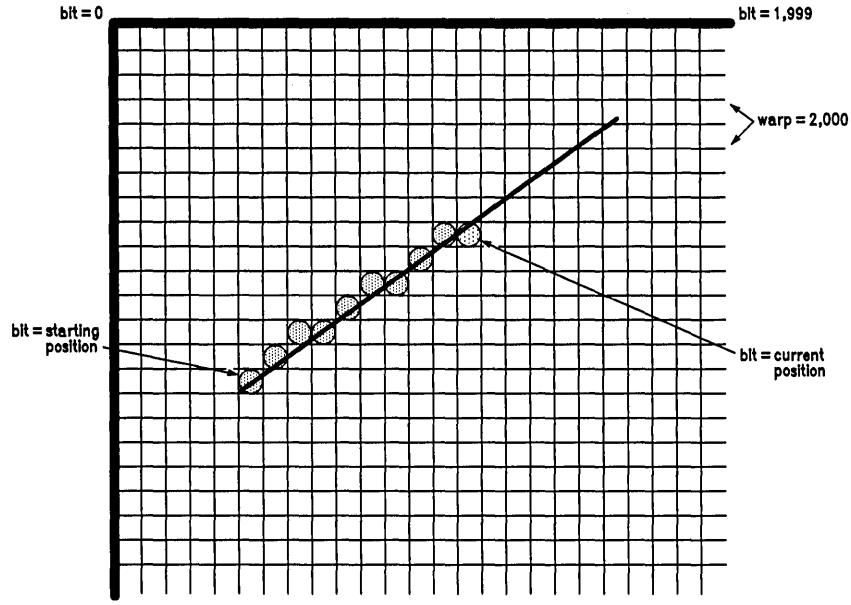
start p1: x_inc = y'_incl = −1
y_inc = x'_inc = +1
start p2: x_inc = y'_inc = +1
y_inc = x'_inc = −1

TL/EE/9665-10

h.

Note: a., g., and h. are rotated 90 degrees left and x', y' refer to the original axis.

FIGURE 4

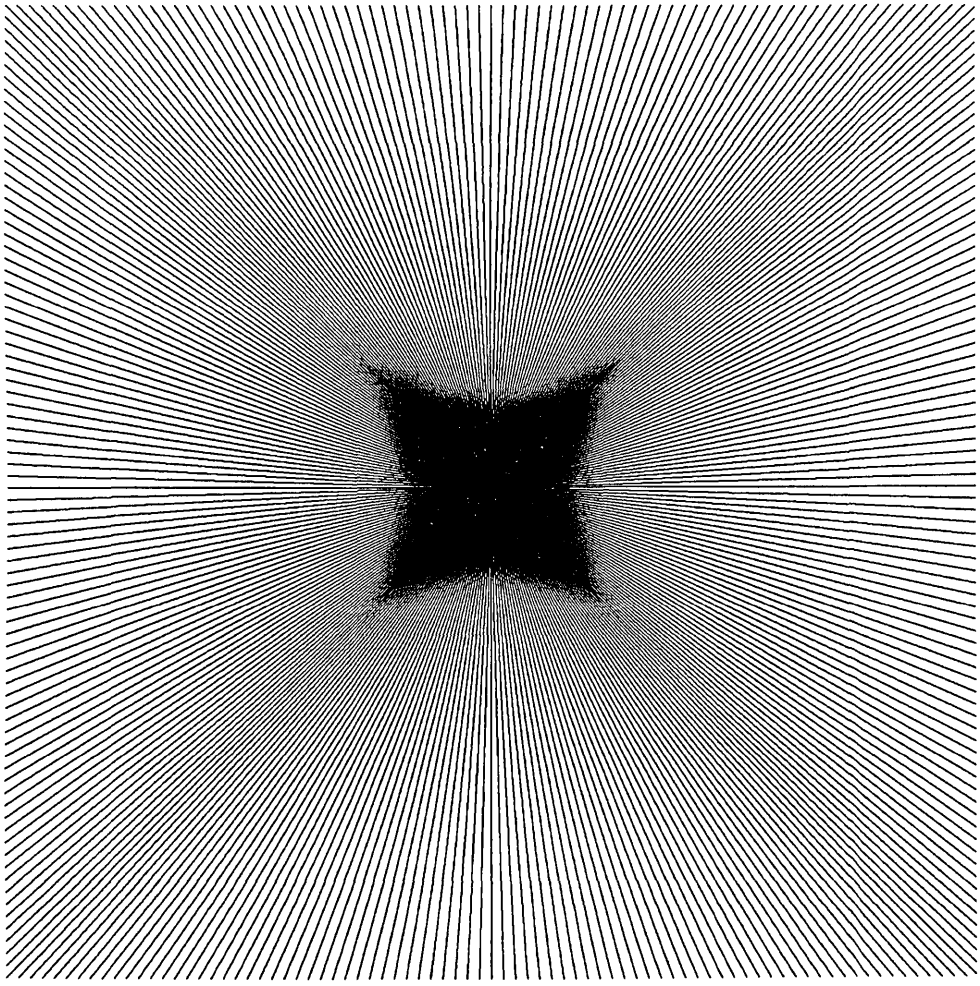


TL/EE/9665-11

Bit Map is 500 kbytes, 2k x 2k Bits
Base Address of Bit Map is 'Bit_Map'

FIGURE 5

Graphics Image (2000 x 2000 Pixels), 300 DPI



TL/EE/9665–12

FIGURE 6. Star-Burst Benchmark—This Star-Burst image was done on a 2k x 2k pixel bit map. Each line is 2k pixels in length and passes through the center of the image, bisecting the square. The lines are 25 pixel units apart, and are drawn using the LINE_DRAW.S routine. There are a total of 160 lines. The total time for drawing this Star-Burst is 2.9 sec on 10 MHz NS32C016.

4.0 IMPLEMENTATION IN SERIES 32000 ASSEMBLY: THE SBIT INSTRUCTION

National's Series 32000 family of processors is well-suited for the Bresenham's algorithm because of the SBIT instruction. *Figure 7* shows a portion of the assembly version of the Bresenham algorithm illustrating the use of the SBIT instruction. The first part of the loop, handles the algorithm for $p <$ 0 and .CASE2 handles the algorithm for $p > 0$. The main loop is unrolled in this manner to minimize unnecessary branches (compare loop structure of *Figure 7* to *Figure 3*). The SBIT instruction is used to plot the current pixel in the line.

The SBIT instruction uses *bit_map* as a base address from which it calculates the bit position to be set by adding the offset *bit* contained in register r1. For example, if *bit*, or R1, contains 2,000*, then the instruction:

sbitd r1,@ *bit_map*

will set the bit at position 2,000, given that *bit_map* is the memory location starting at bit 0 of this grid. In actuality, if *base* is a memory address, then the bit position set is:

offset MOD 8

within the memory byte whose address is:

base + (*offset* DIV 8)

So, for the above example,

2,000 MOD 8 = 0

bit_map + 2,000 DIV 8 = *bit_map* + 250

Thus, bit 0 of byte (*bit_map* + 250) is set. This bit corresponds to the first bit of the second row in *Figure 5*.

*All numbers are in decimal.

The SBIT instruction greatly increases the speed of the algorithm. Notice the method of setting the pixel in the C program given in the Appendix:

bit_map[*bit*/8] | = *bit_pos*[(bit & 7)]

This line of code contains a costly division and several other operations that are eliminated with the SBIT instruction. The SBIT instruction helps optimize the performance of the program. Notice also that the algorithm can be implemented using only 7 registers. This improves the speed performance by avoiding time-consuming memory accesses.

5.0 CONCLUSION

An optimized Bresenham line-drawing algorithm has been presented using the SYS32/20 system. Both Series 32000 assembly and C versions have been included. *Figure 8* presents the various timing results of the algorithm. Most of the optimization efforts have been concentrated in the main loop of the program, so the reader may spot other ways to optimize, especially in the set-up section of the algorithm.

Several variations of the Bresenham algorithm have been developed. One particular variation from Bresenham himself relies on "run-length" segments of the line for speed optimization. The algorithm is based on the original Bresenham algorithm, but uses the fact that typically the decision variable p has one sign for several iterations, changing only once in-between these "run-length" segments to make one vertical step. Thus, most lines are composed of a series of horizontal "run-lengths" separated by a single vertical jump. (Consider the special cases where the slope of the line is exactly 1, the slope is 0 or the slope is infinity.) This algorithm will be explored in the NS32CG16 Graphics Note 5, AN-522, "Line Drawing with the NS32CG16", where it will be optimized using special instructions of the NS32CG16.

```
# Main loop of Bresenham algorithm
.LOOP: #p < 0: move in x direction only
                cmpqd           $0,r4
                ble             .CASE2
                addd            r0,r4
                addd            r5,r1
                sbitd           r1,@_bit_map
                cmpd            r3,r1
                bne             .LOOP
                exit            [r3,r4,r5,r6,r7]
                ret             $0
                .align 4
.CASE2: #P > 0: move in x and y direction
                addd            r2,r4
                addd            r7,r1
                addd            r5,r1
                sbitd           r1,@_bit_map
                cmpd            r1,r3
                bne             .LOOP
                exit            [r3,r4,r5,r6,r7]
                ret             $0
```

FIGURE 7

Note: Instructions followed by the letter 'd' indicate "double word" operations.

```
Register and Memory
      Contents
r0 = c1 constant
r1 = bit current
     position
r2 = c2 constant
r3 = last_bit
r4 = p decision var
r5 = x_inc increment
r6 = unused register
r7 = y_inc increment
_bit_map = address of
first byte in bit map
```

Timing Performance
2k x 2k Bit Map
2k Pix/Vector 160 Lines per Star-Burst

Version	NS32000 Assembly with SBIT	
Parameter	**NS32C016-10**	**NS32C016-15**
Set-up Time Per Vector	45 μs	30 μs
Vectors/Sec	54	82
Pixels/Sec	109,776	164,771
Total Time Star-Burst Benchmark	2.9s	1.9s

FIGURE 8

Set-up time per line is measured from the start of LINE_DRAW.S only. The overhead of calling the LINE_DRAW routine, starting the timer and creating the endpoints of the vector are not included in this time. Set-up time does include all register set-up and branching for the Bresenham algorithm up to the entry point of the main loop.

Vectors/Second is determined by measuring the number of vectors per second the LINE_DRAW routine can draw, not including the overhead of the DRIVER.C and START.C routines, which start the timer and calculate the vector endpoints. All set-up of registers and branching for the Bresenham algorithm are included.

Pixels/Second is measured by dividing the Vectors/Second value by the number of pixels per line.

Total Time for the Star-Burst benchmark is measured from start of benchmark to end. It does include all overhead of START.C and DRIVER.C and all set-up for LINE_DRAW.S. This number can be used to approximate the number of pages per second for printing the whole Star-Burst image.

```
#       National Semiconductor Corporation.
#       CTP version 2.4     -- line_draw.s --

.file   "line_draw.s"
        .comm   _bit_map,49975ø
        .globl  _line_draw
        .set    WARP,2øøø
        .align 4
_line_draw:                             # initialize
        enter   [r3,r4,r5,r6,r7],12
        movd    12(fp),r5               # r5=ys
        movd    8(fp),r6                # r6=xs
        movd    r5,r1                   # initialize starting 'bit'
        muld    $(WARP),r1              # bit=warp*ys+xs
        addd    r6,r1                   # r1=bit
        movd    2ø(fp),r4               # r4=yf
        subd    r5,r4                   # r4=dy
        absd    r4,r3                   # r3=|dy|
        movd    16(fp),r2               # r2=xf
        subd    r6,r2                   # r2=dx
        absd    r2,r6                   # r6=|dx|
        cmpd    r3,r6                   # branch if slope<1
        ble     .LL1                    # must rotate axis for slope>1
        cmpqd   $(ø),r4                 # if dy<ø want x_inc<ø
        bge     .LL2                    # else x_inc is pos
        addr    WARP,r5                 # x_inc=+/-warp because of rotate
        br      .LL3
        .align 4
.LL2:
        addr    -WARP,r5
.LL3:
        cmpqd   $(ø),r2                 # if dx<ø want y_inc<ø
        bge     .LL4                    # else y_inc is pos
        movqd   $(1),r7                 # y_inc=+/-1 becaue of rotate
        br      .LL5
        .align 4
.LL4:
        movqd   $(-1),r7
.LL5:                                   # calculate c1,c2 and p
        movd    r6,rø
        addd    rø,rø                   # rø=c1=2*|dx| because of rotate
        subd    r3,r6                   # r6=|dx-dy|   r2=2*r6=c2
        addr    ø[r6:w],r2              # this muls r6 by 2 and puts in r2
        movd    rø,r4
        subd    r3,r4                   # r4=c2-|dy|=p in rotated space
        movd    2ø(fp),r3               # calculate last_bit
        muld    $(WARP),r3
        addd    16(fp),r3               # r3=last_bit
        br      .LL6
        .align 4
.LL1:                                   # slope<1 use original axis
        cmpqd   $(ø),r4                 # dy determines y_inc
        bge     .LL7
        addr    WARP,r7                 # dy>ø then y_inc=+warp
        br      .LL8
        .align 4
.LL7:
        addr    -WARP,r7                # dy<ø then y_inc=-warp
.LL8:
        cmpqd   $(ø),r2
        bge     .LL9
        movqd   $(1),r5                 # dx>ø then x_inc=+1
        br      .LL1ø
        .align 4
.LL9:
        movqd   $(-1),r5                # dx<ø then x_inc=-1
```

TL/EE/9665-13

```
.LL1ø:                                  # calculate c1,c2,p
        addr    ø[r3:w],rø              # rø=2*r3=c1
        movd    r3,r2
        subd    r6,r2
        addd    r2,r2                   # r2=2*|dy-dx|=c2
        movd    rø,r4
        subd    r6,r4                   # p=2*dy-dx=r4
        movd    2ø(fp),r3               # calculate last_bit=r3
        muld    $(WARP),r3
        addd    16(fp),r3
.LL6:                                   # main loop for algorithm
        cmpqd   $(ø),r4                 # check sign of p
        ble     .LL11                   # branch if pos
        addd    rø,r4                   # add c1 to p
        addd    r5,r1                   # inc bit by x_inc only
        sbitd   r1,@_bit_map            # plot bit
        cmpd    r3,r1                   # end only if bit=last_bit
        bne     .LL6
        exit    [r3,r4,r5,r6,r7]
        ret     $(ø)
        .align 4
.LL11:                                  # p>ø then inc in y dir
        addd    r2,r4                   # add c2 to p
        addd    r7,r1                   # add y_inc to bit
        addd    r5,r1                   # add x_inc to bit
        sbitd   r1,@_bit_map            # plot bit
        cmpd    r1,r3                   # end only when bit=last_bit
        bne     .LL6
        exit    [r3,r4,r5,r6,r7]
        ret     $(ø)
```

TL/EE/9665-14

```
/* This program calculates points on a line using Bresenham's iterative */
/* method.  */

#include<stdio.h>
#define xbytes  25Ø             /* number of bytes along x-axis*/
#define warp    xbytes * 8      /* number of bits along x_axis*/
#define maxy    1999            /* number of lines in y_axis*/
unsigned char   bit_map[xbytes*maxy];   /* array contains bit map*/
static unsigned char    bit_pos[]={1,2,4,8,16,32,64,128};
                                /* look-up table for setting bit */

line_draw(xs,ys,xf,yf)          /* starting (s) and finishing (f) points */

int     xs,ys,xf,yf;

{
        int     dx,dy,x_inc,y_inc,      /* deltas and increments */
                bit,last_bit,           /* current and last bit positions */
                p,c1,c2;                /* decision variable p and constants */

        dx=xf-xs;
        dy=yf-ys;
        bit=(ys*warp)+xs;               /* initialize bit to first bit pos */
        last_bit=(yf*warp)+xf;          /* calculate last bit on line */

        if (abs(dy) > abs(dx))
        {                               /* abs(slope)>1 must rotate space */
                                        /* see Figure 5 a.,g.,and h. */
                if (dy>Ø)
                        x_inc=warp;     /* x_axis is now original y_axis */
                else
                        x_inc= -warp;
                if (dx>Ø)
                        y_inc=1;        /* y_axis is now original x_axis */
                else
                        y_inc= -1;
                c1=2*abs(dx);           /* calculate Bresenham's constants */
                c2=2*(abs(dx)-abs(dy));
                p=2*abs(dx)-abs(dy);    /* p is decision variable now rotated */
        }
        else {                          /* abs(slope)<1 use original axis */
                if (dy>Ø)
                        y_inc=warp;     /* y_inc is +/-warp number of bits */
                else
                        y_inc= -warp;
                if (dx>Ø)
                        x_inc=1;        /* move forward one bit */
                else
                        x_inc= -1;      /* or backward one bit */
                c1=2*abs(dy);           /* calculate constants and p */
                c2=2*(abs(dy)-abs(dx));
                p=2*abs(dy)-abs(dx);
        }

/* Bresenham's Algorithm */
                do      /* do once for each x increment, i.e. dx times */
                {
                        if (p<Ø)                /* no y movement if p<Ø */
                                p+=c1;
                        else {                  /* move in y dir if p>Ø */
                                p+=c2;
                                bit+=y_inc;
                        }
                        bit+=x_inc;             /* always increment x */

                        /* bit is set by calculating bit MOD 8, which is */
```

TL/EE/9665-15

```
                        /* same as bit & 7, then looking up appropriate  */
                        /* bit in table bit_pos.  This bit pos is then set */
                        /* in byte bit/8 */

                        bit_map[bit/8] |= bit_pos[(bit&7)];
                } while (bit!=last_bit);
}
```

TL/EE/9665-16

```
 /* Program driver.c feeds line vectors to LINE_DRAW.S forming Star-Burst.   */

#include <stdio.h>
#define xbytes  25ø
#define maxx    1999
#define maxy    1999

unsigned char   bit_map[xbytes*maxy];

main()

{
    int i,count;

/* generate Star-Burst image */

        for (count=1;count<=1øøø;test++){

                for (i=ø;i<=maxy;i+=25)
                        line_draw(ø,i,maxx,maxy-i);
                for (i=ø;i<=maxx;i+=25)
                        line_draw(i,maxy,maxx-i,ø);
        }

}
```

TL/EE/9665-17

```
/* Start timer and call main procedure of DRIVER.C to draw lines */

start() {
    long *timer = (long *) øx6øø;
    *timer = ø;         /* write a zero to timer location */
            main(ø,ø);      /* Show argc as zero, argv ->ø */
    return(*timer);     /* return, in rø, the current time */
}
```

TL/EE/9665-18

Block Move Optimization Techniques Series 32000® Graphics Note 2

National Semiconductor
Application Note 526
Dave Rand

1.0 INTRODUCTION

This application note discusses fast methods of moving data in printer applications using the National Semiconductor Series 32000. Typically this data is moved to or from the band of RAM representing a small portion (or slice) of the total image. The length of data is fixed. The controller design may require moving data every few milliseconds to image the page, until a total of 1 page has been moved. This may be (at 300 DPI, for example) (8.5 × 300) × (11 × 300), or 1,051,875 bytes. In current controller designs the width is often rounded to a word boundary (usually 320 bytes at 300 DPI). This technique uses 1,056,000 bytes, or 528,000 words.

2.0 DESCRIPTION

The move string instructions (MOVSi) in the 32000 are very powerful, however, when all that is needed is a string copy, they may be overkill. The string instructions include string translation, conditionals and byte/word/double sizes. If the application needs only to move a block of data from one location to another, and that data is a known size (or at least a multiple of a known size), using unrolled MOVD instructions is a faster way of moving the data from A to B on the NS32032 and NS32332.

3.0 IMPLEMENTATION

A code sample follows which makes use of a block size of 128 bytes. To move 256 bytes, for example, R0 should contain 2 on entry.

```
; Version 1.0  Sun Mar 29 12:57:20 1987
;
;A subroutine to move blocks of memory. Uses a granularity of
;128 bytes.
;
;       Inputs:
;               r0 = number of 128 byte blocks to move
;               r1 = source block address
;               r2 = destination block address
;
;Listing continues on following page
;
```

TL/EE/9696-1

```
;       Outputs:
;               r0 = 0
;               r1 = source block address + (128 * blocks)
;               r2 = destination block address + (128 * blocks)
;
;Notes:
;       This algorithm corresponds closely to the MOVSD instruction,
;       except that r0 contains the number of 128 byte blocks, not
;       4 byte double words. The output values are the same as if a
;       MOVSD instruction were used.
;
movmem: cmpqd   0,r0            ;if no blocks to move
        beq     mvexit          ;exit now.
        .align  4
mvlp1:  movd    0(r1),0(r2)     ;move one block of data
        movd    4(r1),4(r2)
        movd    8(r1),8(r2)
        movd    12(r1),12(r2)
        movd    16(r1),16(r2)
        movd    20(r1),20(r2)
        movd    24(r1),24(r2)
        movd    28(r1),28(r2)
        movd    32(r1),32(r2)
        movd    36(r1),36(r2)
        movd    40(r1),40(r2)
        movd    44(r1),44(r2)
        movd    48(r1),48(r2)
        movd    52(r1),52(r2)
        movd    56(r1),56(r2)
        movd    60(r1),60(r2)
        movd    64(r1),64(r2)
        movd    68(r1),68(r2)
        movd    72(r1),72(r2)
        movd    76(r1),76(r2)
        movd    80(r1),80(r2)
        movd    84(r1),84(r2)
        movd    88(r1),88(r2)
        movd    92(r1),92(r2)
        movd    96(r1),96(r2)
        movd    100(r1),100(r2)
        movd    104(r1),104(r2)
        movd    108(r1),108(r2)
        movd    112(r1),112(r2)
        movd    116(r1),116(r2)
        movd    120(r1),120(r2)
        movd    124(r1),124(r2)
        addr    128(r1),r1      ;quick way of adding 128
        addr    128(r2),r2
        acbd    -1,r0,mvlp1     ;loop for rest of blocks
mvexit: ret     $0
```

TL/EE/9696-2

4.0 TIMING

All timing assumes word aligned data (double word aligned for 32-bit bus). Unaligned data is permitted, but will reduce the speed.

On the 32532 (no wait states, @ 30 MHz, 32-bit bus), this code executes in 204 clocks, assuming burst mode access is available. To move 256 bytes, this routine would take 13.6 μs. The MOVSD instruction takes about 156 clocks to move a 128-byte block. The MOVSD instruction is the best choice, therefore, on the 32532.

On the 32332 (no wait states, @ 15 MHz, 32-bit bus), this code executes in 458 clocks per 128-byte block. Thus, to move 256 bytes, this algorithm takes 61.1 μs. The loop overhead (the ADDR and ACBD instructions) is about 10%. Doubling the block size (to 256 bytes) would reduce the loop overhead to 5%, and reducing the block size (to 64 bytes) would increase the loop overhead to 20%. In comparison, the 32332 MOVSD instruction takes about 721 clocks to move a 128-byte block.

On the 32032 (no wait states. @ 10 MHz, 32-bit bus), this code executes in 634 clocks per 128-byte block. Thus, to move 256 bytes, this algorithm takes 126.8 μs. The loop overhead (the ADDR and ACBD instructions) is about 5%. Doubling the block size (to 256 bytes) would reduce the loop overhead to 2.5%, and reducing the block size (to 64 bytes) would increase the loop overhead to 10%. In comparison, the 32032 MOVSD instruction takes about 690 clocks to move a 128-byte block.

On the 32016 (1 wait state. @ 10 MHz, 16-bit bus), this code executes in 1150 clocks per 128-byte block. Thus, to move 256 bytes, this algorithm takes 230.0 μs. The loop overhead on the 32016 is about 2.5%. In comparison, the 32016 MOVSD instruction would take about 1,074 clocks. Thus, the MOVSD instruction is faster, and makes better use of the available bus bandwidth of the NS32016.

5.0 CONCLUSIONS

The MOVSi instructions on the NS32016 provide a very fast memory block move capability, with variable size. On the NS32332 and NS32032, however, unrolled MOVD instructions are faster due to the larger bus bandwidth of the NS32332 and NS32032.

Clearing Memory with the 32000; Series 32000® Graphics Note 3

National Semiconductor
Application Note 527
Dave Rand

1.0 INTRODUCTION

In printer applications, large amounts of RAM may need to be initialized to a zero value. This application note describes a fast method.

2.0 DESCRIPTION

While several different methods of initializing memory to all zeros are available, here is one that works very well on the Series 32000. While the current version clears memory only in blocks of 128 bytes, other block sizes are possible by extending the algorithm.

3.0 IMPLEMENTATION

This routine is written to clear blocks of 128 bytes. This provides an optimal tradeoff between loop size (granularity) and loop overhead. This can be modified to use a different size. For example, to use a block size of 64 bytes, simply delete 16 of the MOVQD 0,TOS instructions from the listing. As well, since the value of r1 is now the number of 64 byte groups, one of the ADDD R2,R2 instructions (prior to the loading of the stack pointer) must be removed. Since the 32000 has two stacks, interrupts will be handled properly using this code. If only a fixed buffer size needs to be cleared, the code can be further unrolled to clear that area (i.e., increase the number of MOVQD 0,TOS instructions.)

```
; Version 1.1  Sun Mar 29 10:22:19 1987
;
;Subroutine to clear a block of memory. The granularity of this
;algorithm is 128 bytes, to reduce the looping overhead.
;
;       Inputs:
;               r0 = start of block
;               r1 = number of 128-byte groups to clear
;
;       Outputs:
;               All registers preserved.
;
;
;Listing continues on following page
;
```

TL/EE/9697-1

```
clram:  cmpqd    0,r1              ;any blocks to clear?
        beq      clexit:w          ;no, exit now.
        save     [r0,r1,r2]        ;save our working registers
        movd     r1,r2             ;here we set r0 = r0 + (r1 * 128) + 4
        addd     r2,r2             ;length *= 2
        addd     r2,r2             ;*4
        addd     r2,r2             ;*8
        addd     r2,r2             ;*16
        addr     4(r0)[r2:q],r0    ;get starting point + 4
        sprd     sp,r2             ;save current stack
        lprd     sp,r0             ;move to last double
        .align   4
cl2:    movqd    0,tos             ;clear a double
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        movqd    0,tos
        acbd     -1,r1,cl2
        lprd     sp,r2           ;restore stack pointer
        restore  [r0,r1,r2]      ;restore our saved registers
clexit: ret      0
```

TL/EE/9697-2

FIGURE 1

```
clram:  cmpqd   0,r1            ;any blocks to clear?
        beq     clexit:w        ;no, exit now.
        .align  4
cl2:    movqd   0,00(r0)                ;clear a double
        movqd   0,04(r0)
        movqd   0,08(r0)
        movqd   0,12(r0)
        movqd   0,16(r0)
        movqd   0,20(r0)
        movqd   0,24(r0)
        movqd   0,28(r0)
        movqd   0,32(r0)
        movqd   0,36(r0)
        movqd   0,40(r0)
        movqd   0,44(r0)
        movqd   0,48(r0)
        movqd   0,52(r0)
        movqd   0,56(r0)
        movqd   0,60(r0)
        movqd   0,64(r0)
        movqd   0,68(r0)
        movqd   0,72(r0)
        movqd   0,76(r0)
        movqd   0,80(r0)
        movqd   0,84(r0)
        movqd   0,88(r0)
        movqd   0,92(r0)
        movqd   0,96(r0)
        movqd   0,100(r0)
        movqd   0,104(r0)
        movqd   0,108(r0)
        movqd   0,112(r0)
        movqd   0,116(r0)
        movqd   0,120(r0)
        movqd   0,124(r0)
        addd    $128,r0
        acbd    -1,r1,cl2
clexit: ret     0
```

TL/EE/9697-3

FIGURE 2

4.0 TIMING RESULTS

On the NS32016, NS32032 and NS32332, 4 clock cycles per write are required. To clear one page of 300 DPI 8½ x 11 (1,056,000 bytes), for example, requires 264,000 double words to be written. The optimal time for this, using 100% of the bus bandwidth on a 16 bit bus, would be 528,000 * 400 ns, or 211.2 ms, @ 10 MHz. All timing data assumes word aligned data (double word aligned for 32 bit bus). Unaligned data is permitted, but will reduce the speed somewhat.

On the NS32332 (no wait states. @15 MHz, 32 bit bus), this code clears the full page image in 178 ms.

On the NS32032 (no wait states. @10 MHz, 32 bit bus), this code clears the full page image in 324 ms.

On the NS32016 (1 wait state. @10 MHz, 16 bit bus), this code clears the full page image in 509 ms.

Doubling the block size (to 256 bytes) would increase the speed by 1%–2%, on the code sample.

On the NS32532, a better approach is to use the register indirect method of referencing memory, as is shown in *Figure 2*. With this approach, the page memory can be cleared in 19 ms, assuming a no wait state 30 MHz system, with a 32 bit bus. The optimal time, using 100% of the bus bandwidth of the NS32532 (2 clock bus cycle) would be 264,000 * 66.6 ns, or 17.6 ms.

Image Rotation Algorithm Series 32000® Graphics Note 4

National Semiconductor
Application Note 528
Dave Rand

1.0 INTRODUCTION

Fast image rotation of 90 and 270 degrees is important in printer applications, since both Portrait and Landscape orientation printing may be desired. With a fast image rotation algorithm, only the Portrait orientation fonts need to be stored. This minimizes ROM storage requirements.

This application note shows a fast image rotation algorithm that may be used to rotate an 8 pixel by 8 line image. Larger image sizes may be rotated by successive application of the rotation primitive.

2.0 DESCRIPTION

This Rotate Image algorithm (developed by the Electronic Imaging Group at National Semiconductor) does a very fast 8 by 8 (64 bit) rotation of font data. Note also that this algorithm does not exclusively deal with fonts, but any 64 bit image. Larger images can be rotated by breaking the image down into 8 x 8 segments, and using a 'source warp' constant to index into the source data.

The source data is pointed to by R0 on entry. A 'source warp' is contained in R1, and is added to R0 after each read of the source font. This allows the rotation of 16 by 16, 32 by 32 and larger fonts.

ROTIMG deals with the 8 by 8 destination character as 8 sequential bytes in two registers (R2 and R3), as follows:

Destination Font Matrix

Low Address

1
2
3
4
5
6
7
8

= R2	4	3	2	1
= R3	8	7	6	5

High Address

ROTIMG uses an external table (a pointer to the start of the table is located in register R4) to speed the rotation and to minimize the code. This table consists of 256 64 bit entries, or a total of 2,048 bytes. The table may be located code (PC) or data (SB) relative. The complete table is at the end of this document (see *Figure 1*). A few entries of the table are reproduced above.

Entry	Definition
0	0x00000000 00000000
1	0x00000000 00000001
2	0x00000000 00000100
3	0x00000000 00000101
...	
253	0x01010101 01010001
254	0x01010101 01010100
255	0x01010101 01010101

The bytes in the table are standard LSB to MSB format. Since there is no quad-byte assembler pseudo-op (other than LONG, which is floating point), we must reverse the 'double' declaration to get the correct byte ordering, as is shown below:

Entry	Definition
0	double 0,0
1	double 1,0
2	double 256,0
3	double 257,0
...	
253	double 16842753,16843009
254	double 0x01010100,0x01010101
255	double 0x01010101,0x01010101

Each byte within each eight byte table entry represents one bit of output data. By indexing into the table, and ORing the table's contents with R2 and R3, we set the destination byte if the corresponding source bit is set. In this manner, the character is rotated.

3.0 IMPLEMENTATION

What we are doing is setting the LS Bit of the destination byte if the source bit corresponding to that byte is set. We then shift the entire 64 bit destination left one bit, and repeat this process until we have set all eight bits, and processed all eight bytes of source information.

The source data for an 8 by 8 character ">" appears below:

Character Table for '>'

	Bit Number	Hex Value
	0 1 2 3 4 5 6 7	
Byte	0 0 1 0 0 0 0 0 0	02
	1 0 0 1 0 0 0 0 0	04
	2 0 0 0 1 0 0 0 0	08
	3 0 0 0 0 1 0 0 0	10
	4 0 0 0 0 1 0 0 0	10
	5 0 0 0 1 0 0 0 0	08
	6 0 0 1 0 0 0 0 0	04
	7 0 1 0 0 0 0 0 0	02

The ROTIMG algorithm, expressed in 32000 code, appears below:

```
#
#
#Rotate image emulation code
#
#    Inputs:
#         R0 = Source font address
#         R1 = Source font warp
#         R4 = Rotate table address
#
#    Outputs:
#         R2 = Destination font low 4 bytes (lsb->msb, 0 - 3)
#         R3 = Destination font high 4 bytes (lsb->msb, 4 - 7)
#
ROTIMG:  save [r0,r5,r6,r7]       #save registers we will use
         movqd    0,r2            #clear destination font
         movd     r2,r3           #clear high bits of dest.
         movd     r2,r5           #clear high bits of temp.
         addr     8,r6            #deal with 8 bytes of src.
rotlp:   movb     0(r0),r5        #get a byte of source
         addd     r1,r0           #add source warp
         addd     r2,r2           #shift destination left one bit
         addd     r3,r3           #top 32 bits too
         addrd    r4[r5:q],r7     #get pointer to table
         ord      0(r7),r2        #or in low bits
         ord      4(r7),r3        #or in high bits
         acbd     -1,r6,rotlp     #and back for more
         restore  [r0,r5,r6,r7]   #restore registers
         ret      $0              #and return
```

TL/EE/9698-1

Now, let's look at what happens to the data, given the example font of '>'.

Loop #	Source Font	R3	R2	
0	—	00000000	00000000	;0 destination
1	02 hex	00000000	00000100	;first bits in
2	04	00000000	00010200	;next bits in
3	08	00000000	01020400	;and so on
4	10	00000001	02040800	
5	10	00000003	04081000	
6	08	00000006	09102000	
7	04	0000000C	12214000	
8	02	00000018	24428100	;last iteration

Now, arranging this in the appropriate order gives us:

Destination Character Table for '>', 90 degree

	Bit Number	Hex Value
	01234567	
Byte	000000000	00
	110000001	81
	201000010	42
	300100100	24
	400011000	18
	500000000	00
	600000000	00
	700000000	00

Destination Character Table for '>', 270 degree

	Bit Number	Hex Value
	01234567	
Byte	000000000	00
	100000000	00
	200000000	00
	300011000	18
	400100100	24
	501000010	42
	610000001	81
	700000000	00

Note that by re-ordering the output data, we may rotate 90 or 270 degrees. This may also be accomplished by using a different table (see *Figure 2*).

4.0 TIMING

With unrolled 32000 code, the time for this algorithm is about 588 clocks on the 32016. Subtracting the font read time from this (about 113 clocks), the actual time for rotation is 475 clocks. On the 32332, the time is about 388 clocks. On the 32532, the unrolled loop time is 120–180 clocks, depending on burst mode availability. Repetition of the character data also affects the 32532, due to the data cache. See *Figure 3* for an unrolled code listing.

This table is used for the ROTIMG code. It is 256 entries of 64 bits each (8 bytes * 256 = 2048 bytes). There are two entries per line. This table is used for 90° rotation.

```
rottab1: .double  0x00000000,0x00000000,0x00000001,0x00000000 ;0,1
         .double  0x00000100,0x00000000,0x00000101,0x00000000 ;2,3
         .double  0x00010000,0x00000000,0x00010001,0x00000000 ;4,5
         .double  0x00010100,0x00000000,0x00010101,0x00000000 ;6,7
         .double  0x01000000,0x00000000,0x01000001,0x00000000 ;...
         .double  0x01000100,0x00000000,0x01000101,0x00000000
         .double  0x01010000,0x00000000,0x01010001,0x00000000
         .double  0x01010100,0x00000000,0x01010101,0x00000000
         .double  0x00000000,0x00000001,0x00000001,0x00000001
         .double  0x00000100,0x00000001,0x00000101,0x00000001
         .double  0x00010000,0x00000001,0x00010001,0x00000001
         .double  0x00010100,0x00000001,0x00010101,0x00000001
         .double  0x01000000,0x00000001,0x01000001,0x00000001
         .double  0x01000100,0x00000001,0x01000101,0x00000001
         .double  0x01010000,0x00000001,0x01010001,0x00000001
         .double  0x01010100,0x00000001,0x01010101,0x00000001
         .double  0x00000000,0x00000100,0x00000001,0x00000100
         .double  0x00000100,0x00000100,0x00000101,0x00000100
         .double  0x00010000,0x00000100,0x00010001,0x00000100
         .double  0x00010100,0x00000100,0x00010101,0x00000100
         .double  0x01000000,0x00000100,0x01000001,0x00000100
         .double  0x01000100,0x00000100,0x01000101,0x00000100
         .double  0x01010000,0x00000100,0x01010001,0x00000100
         .double  0x01010100,0x00000100,0x01010101,0x00000100
         .double  0x00000000,0x00000101,0x00000001,0x00000101
         .double  0x00000100,0x00000101,0x00000101,0x00000101
         .double  0x00010000,0x00000101,0x00010001,0x00000101
         .double  0x00010100,0x00000101,0x00010101,0x00000101
         .double  0x01000000,0x00000101,0x01000001,0x00000101
         .double  0x01000100,0x00000101,0x01000101,0x00000101
         .double  0x01010000,0x00000101,0x01010001,0x00000101
         .double  0x01010100,0x00000101,0x01010101,0x00000101
         .double  0x00000000,0x00010000,0x00000001,0x00010000
         .double  0x00000100,0x00010000,0x00000101,0x00010000
         .double  0x00010000,0x00010000,0x00010001,0x00010000
         .double  0x00010100,0x00010000,0x00010101,0x00010000
         .double  0x01000000,0x00010000,0x01000001,0x00010000
         .double  0x01000100,0x00010000,0x01000101,0x00010000
         .double  0x01010000,0x00010000,0x01010001,0x00010000
         .double  0x01010100,0x00010000,0x01010101,0x00010000
         .double  0x00000000,0x00010001,0x00000001,0x00010001
         .double  0x00000100,0x00010001,0x00000101,0x00010001
         .double  0x00010000,0x00010001,0x00010001,0x00010001
         .double  0x00010100,0x00010001,0x00010101,0x00010001
         .double  0x01000000,0x00010001,0x01000001,0x00010001
         .double  0x01000100,0x00010001,0x01000101,0x00010001
         .double  0x01010000,0x00010001,0x01010001,0x00010001
         .double  0x01010100,0x00010001,0x01010101,0x00010001
         .double  0x00000000,0x00010100,0x00000001,0x00010100
```

TL/EE/9698-2

FIGURE 1

```
.double  0x00000100,0x00010100,0x00000101,0x00010100
.double  0x00010000,0x00010100,0x00010001,0x00010100
.double  0x00010100,0x00010100,0x00010101,0x00010100
.double  0x01000000,0x00010100,0x01000001,0x00010100
.double  0x01000100,0x00010100,0x01000101,0x00010100
.double  0x01010000,0x00010100,0x01010001,0x00010100
.double  0x01010100,0x00010100,0x01010101,0x00010100
.double  0x00000000,0x00010101,0x00000001,0x00010101
.double  0x00000100,0x00010101,0x00000101,0x00010101
.double  0x00010000,0x00010101,0x00010001,0x00010101
.double  0x00010100,0x00010101,0x00010101,0x00010101
.double  0x01000000,0x00010101,0x01000001,0x00010101
.double  0x01000100,0x00010101,0x01000101,0x00010101
.double  0x01010000,0x00010101,0x01010001,0x00010101
.double  0x01010100,0x00010101,0x01010101,0x00010101
.double  0x00000000,0x01000000,0x00000001,0x01000000
.double  0x00000100,0x01000000,0x00000101,0x01000000
.double  0x00010000,0x01000000,0x00010001,0x01000000
.double  0x00010100,0x01000000,0x00010101,0x01000000
.double  0x01000000,0x01000000,0x01000001,0x01000000
.double  0x01000100,0x01000000,0x01000101,0x01000000
.double  0x01010000,0x01000000,0x01010001,0x01000000
.double  0x01010100,0x01000000,0x01010101,0x01000000
.double  0x00000000,0x01000001,0x00000001,0x01000001
.double  0x00000100,0x01000001,0x00000101,0x01000001
.double  0x00010000,0x01000001,0x00010001,0x01000001
.double  0x00010100,0x01000001,0x00010101,0x01000001
.double  0x01000000,0x01000001,0x01000001,0x01000001
.double  0x01000100,0x01000001,0x01000101,0x01000001
.double  0x01010000,0x01000001,0x01010001,0x01000001
.double  0x01010100,0x01000001,0x01010101,0x01000001
.double  0x00000000,0x01000100,0x00000001,0x01000100
.double  0x00000100,0x01000100,0x00000101,0x01000100
.double  0x00010000,0x01000100,0x00010001,0x01000100
.double  0x00010100,0x01000100,0x00010101,0x01000100
.double  0x01000000,0x01000100,0x01000001,0x01000100
.double  0x01000100,0x01000100,0x01000101,0x01000100
.double  0x01010000,0x01000100,0x01010001,0x01000100
.double  0x01010100,0x01000100,0x01010101,0x01000100
.double  0x00000000,0x01000101,0x00000001,0x01000101
.double  0x00000100,0x01000101,0x00000101,0x01000101
.double  0x00010000,0x01000101,0x00010001,0x01000101
.double  0x00010100,0x01000101,0x00010101,0x01000101
.double  0x01000000,0x01000101,0x01000001,0x01000101
.double  0x01000100,0x01000101,0x01000101,0x01000101
.double  0x01010000,0x01000101,0x01010001,0x01000101
.double  0x01010100,0x01000101,0x01010101,0x01000101
.double  0x00000000,0x01010000,0x00000001,0x01010000
.double  0x00000100,0x01010000,0x00000101,0x01010000
.double  0x00010000,0x01010000,0x00010001,0x01010000
.double  0x00010100,0x01010000,0x00010101,0x01010000
.double  0x01000000,0x01010000,0x01000001,0x01010000
.double  0x01000100,0x01010000,0x01000101,0x01010000
.double  0x01010000,0x01010000,0x01010001,0x01010000
.double  0x01010100,0x01010000,0x01010101,0x01010000
```

TL/EE/9698-3

FIGURE 1 (Continued)

```
.double  0x00000000,0x01010001,0x00000001,0x01010001
.double  0x00000100,0x01010001,0x00000101,0x01010001
.double  0x00010000,0x01010001,0x00010001,0x01010001
.double  0x00010100,0x01010001,0x00010101,0x01010001
.double  0x01000000,0x01010001,0x01000001,0x01010001
.double  0x01000100,0x01010001,0x01000101,0x01010001
.double  0x01010000,0x01010001,0x01010001,0x01010001
.double  0x01010100,0x01010001,0x01010101,0x01010001
.double  0x00000000,0x01010100,0x00000001,0x01010100
.double  0x00000100,0x01010100,0x00000101,0x01010100
.double  0x00010000,0x01010100,0x00010001,0x01010100
.double  0x00010100,0x01010100,0x00010101,0x01010100
.double  0x01000000,0x01010100,0x01000001,0x01010100
.double  0x01000100,0x01010100,0x01000101,0x01010100
.double  0x01010000,0x01010100,0x01010001,0x01010100
.double  0x01010100,0x01010100,0x01010101,0x01010100
.double  0x00000000,0x01010101,0x00000001,0x01010101
.double  0x00000100,0x01010101,0x00000101,0x01010101
.double  0x00010000,0x01010101,0x00010001,0x01010101
.double  0x00010100,0x01010101,0x00010101,0x01010101
.double  0x01000000,0x01010101,0x01000001,0x01010101
.double  0x01000100,0x01010101,0x01000101,0x01010101 ;250,251
.double  0x01010000,0x01010101,0x01010001,0x01010101 ;252,253
.double  0x01010100,0x01010101,0x01010101,0x01010101 ;254,255
```

TL/EE/9698–4

FIGURE 1 (Continued)

This table is used for the ROTIMG code. It is 256 entries of 64 bits each (8 bytes * 256 = 2048 bytes). There are two entries per line. This gives a 270° rotation.

```
rottab2:  .double  0x00000000,0x00000000,0x00000000,0x01000000
          .double  0x00000000,0x00010000,0x00000000,0x01010000
          .double  0x00000000,0x00000100,0x00000000,0x01000100
          .double  0x00000000,0x00010100,0x00000000,0x01010100
          .double  0x00000000,0x00000001,0x00000000,0x01000001
          .double  0x00000000,0x00010001,0x00000000,0x01010001
          .double  0x00000000,0x00000101,0x00000000,0x01000101
          .double  0x00000000,0x00010101,0x00000000,0x01010101
          .double  0x01000000,0x00000000,0x01000000,0x01000000
          .double  0x01000000,0x00010000,0x01000000,0x01010000
          .double  0x01000000,0x00000100,0x01000000,0x01000100
          .double  0x01000000,0x00010100,0x01000000,0x01010100
          .double  0x01000000,0x00000001,0x01000000,0x01000001
          .double  0x01000000,0x00010001,0x01000000,0x01010001
          .double  0x01000000,0x00000101,0x01000000,0x01000101
          .double  0x01000000,0x00010101,0x01000000,0x01010101
          .double  0x00010000,0x00000000,0x00010000,0x01000000
          .double  0x00010000,0x00010000,0x00010000,0x01010000
          .double  0x00010000,0x00000100,0x00010000,0x01000100
          .double  0x00010000,0x00010100,0x00010000,0x01010100
          .double  0x00010000,0x00000001,0x00010000,0x01000001
          .double  0x00010000,0x00010001,0x00010000,0x01010001
          .double  0x00010000,0x00000101,0x00010000,0x01000101
          .double  0x00010000,0x00010101,0x00010000,0x01010101
```

TL/EE/9698–5

FIGURE 2

```
.double  0x01010000,0x00000000,0x01010000,0x01000000
.double  0x01010000,0x00010000,0x01010000,0x01010000
.double  0x01010000,0x00000100,0x01010000,0x01000100
.double  0x01010000,0x00010100,0x01010000,0x01010100
.double  0x01010000,0x00000001,0x01010000,0x01000001
.double  0x01010000,0x00010001,0x01010000,0x01010001
.double  0x01010000,0x00000101,0x01010000,0x01000101
.double  0x01010000,0x00010101,0x01010000,0x01010101
.double  0x00000100,0x00000000,0x00000100,0x01000000
.double  0x00000100,0x00010000,0x00000100,0x01010000
.double  0x00000100,0x00000100,0x00000100,0x01000100
.double  0x00000100,0x00010100,0x00000100,0x01010100
.double  0x00000100,0x00000001,0x00000100,0x01000001
.double  0x00000100,0x00010001,0x00000100,0x01010001
.double  0x00000100,0x00000101,0x00000100,0x01000101
.double  0x00000100,0x00010101,0x00000100,0x01010101
.double  0x01000100,0x00000000,0x01000100,0x01000000
.double  0x01000100,0x00010000,0x01000100,0x01010000
.double  0x01000100,0x00000100,0x01000100,0x01000100
.double  0x01000100,0x00010100,0x01000100,0x01010100
.double  0x01000100,0x00000001,0x01000100,0x01000001
.double  0x01000100,0x00010001,0x01000100,0x01010001
.double  0x01000100,0x00000101,0x01000100,0x01000101
.double  0x01000100,0x00010101,0x01000100,0x01010101
.double  0x00010100,0x00000000,0x00010100,0x01000000
.double  0x00010100,0x00010000,0x00010100,0x01010000
.double  0x00010100,0x00000100,0x00010100,0x01000100
.double  0x00010100,0x00010100,0x00010100,0x01010100
.double  0x00010100,0x00000001,0x00010100,0x01000001
.double  0x00010100,0x00010001,0x00010100,0x01010001
.double  0x00010100,0x00000101,0x00010100,0x01000101
.double  0x00010100,0x00010101,0x00010100,0x01010101
.double  0x01010100,0x00000000,0x01010100,0x01000000
.double  0x01010100,0x00010000,0x01010100,0x01010000
.double  0x01010100,0x00000100,0x01010100,0x01000100
.double  0x01010100,0x00010100,0x01010100,0x01010100
.double  0x01010100,0x00000001,0x01010100,0x01000001
.double  0x01010100,0x00010001,0x01010100,0x01010001
.double  0x01010100,0x00000101,0x01010100,0x01000101
.double  0x01010100,0x00010101,0x01010100,0x01010101
.double  0x00000001,0x00000000,0x00000001,0x01000000
.double  0x00000001,0x00010000,0x00000001,0x01010000
.double  0x00000001,0x00000100,0x00000001,0x01000100
.double  0x00000001,0x00010100,0x00000001,0x01010100
.double  0x00000001,0x00000001,0x00000001,0x01000001
.double  0x00000001,0x00010001,0x00000001,0x01010001
.double  0x00000001,0x00000101,0x00000001,0x01000101
.double  0x00000001,0x00010101,0x00000001,0x01010101
.double  0x01000001,0x00000000,0x01000001,0x01000000
.double  0x01000001,0x00010000,0x01000001,0x01010000
.double  0x01000001,0x00000100,0x01000001,0x01000100
.double  0x01000001,0x00010100,0x01000001,0x01010100
.double  0x01000001,0x00000001,0x01000001,0x01000001
.double  0x01000001,0x00010001,0x01000001,0x01010001
.double  0x01000001,0x00000101,0x01000001,0x01000101
```

TL/EE/9698–6

FIGURE 2 (Continued)

```
.double  0x01000001,0x00010101,0x01000001,0x01010101
.double  0x00010001,0x00000000,0x00010001,0x01000000
.double  0x00010001,0x00010000,0x00010001,0x01010000
.double  0x00010001,0x00000100,0x00010001,0x01000100
.double  0x00010001,0x00010100,0x00010001,0x01010100
.double  0x00010001,0x00000001,0x00010001,0x01000001
.double  0x00010001,0x00010001,0x00010001,0x01010001
.double  0x00010001,0x00000101,0x00010001,0x01000101
.double  0x00010001,0x00010101,0x00010001,0x01010101
.double  0x01010001,0x00000000,0x01010001,0x01000000
.double  0x01010001,0x00010000,0x01010001,0x01010000
.double  0x01010001,0x00000100,0x01010001,0x01000100
.double  0x01010001,0x00010100,0x01010001,0x01010100
.double  0x01010001,0x00000001,0x01010001,0x01000001
.double  0x01010001,0x00010001,0x01010001,0x01010001
.double  0x01010001,0x00000101,0x01010001,0x01000101
.double  0x01010001,0x00010101,0x01010001,0x01010101
.double  0x00000101,0x00000000,0x00000101,0x01000000
.double  0x00000101,0x00010000,0x00000101,0x01010000
.double  0x00000101,0x00000100,0x00000101,0x01000100
.double  0x00000101,0x00010100,0x00000101,0x01010100
.double  0x00000101,0x00000001,0x00000101,0x01000001
.double  0x00000101,0x00010001,0x00000101,0x01010001
.double  0x00000101,0x00000101,0x00000101,0x01000101
.double  0x00000101,0x00010101,0x00000101,0x01010101
.double  0x01000101,0x00000000,0x01000101,0x01000000
.double  0x01000101,0x00010000,0x01000101,0x01010000
.double  0x01000101,0x00000100,0x01000101,0x01000100
.double  0x01000101,0x00010100,0x01000101,0x01010100
.double  0x01000101,0x00000001,0x01000101,0x01000001
.double  0x01000101,0x00010001,0x01000101,0x01010001
.double  0x01000101,0x00000101,0x01000101,0x01000101
.double  0x01000101,0x00010101,0x01000101,0x01010101
.double  0x00010101,0x00000000,0x00010101,0x01000000
.double  0x00010101,0x00010000,0x00010101,0x01010000
.double  0x00010101,0x00000100,0x00010101,0x01000100
.double  0x00010101,0x00010100,0x00010101,0x01010100
.double  0x00010101,0x00000001,0x00010101,0x01000001
.double  0x00010101,0x00010001,0x00010101,0x01010001
.double  0x00010101,0x00000101,0x00010101,0x01000101
.double  0x00010101,0x00010101,0x00010101,0x01010101
.double  0x01010101,0x00000000,0x01010101,0x01000000
.double  0x01010101,0x00010000,0x01010101,0x01010000
.double  0x01010101,0x00000100,0x01010101,0x01000100
.double  0x01010101,0x00010100,0x01010101,0x01010100
.double  0x01010101,0x00000001,0x01010101,0x01000001
.double  0x01010101,0x00010001,0x01010101,0x01010001
.double  0x01010101,0x00000101,0x01010101,0x01000101
.double  0x01010101,0x00010101,0x01010101,0x01010101
```

TL/EE/9698–7

FIGURE 2 (Continued)

The following is an unrolled version of the rotate image algorithm. For the NS32532, the address computation, currently done with a separate addr instruction, may be done with the ORD instruction. This makes the execution time slightly faster.

```
#
#
#Rotate image emulation code
#
#    Inputs:
#         R0 = Source font address
#         R1 = Source font warp
#         R4 = Rotate table address
#
#    Outputs:
#         R2 = Destination font low 4 bytes (lsb->msb, 0 - 3)
#         R3 = Destination font high 4 bytes (lsb->msb, 4 - 7)
#
ROTIMG:
          movqd     0,r2            #clear destination font
          movd      r2,r3           #clear high bits of dest.
          movd      r2,r5           #clear high bits of temp.
          movb      0(r0),r5        #get a byte of source
          addd      r1,r0           #add source warp
          addd      r2,r2           #shift destination left one bit
          addd      r3,r3           #top 32 bits too
          addr      r4[r5:q],r6     #get pointer to table
          ord       0(r6),r2        #or in low bits
          ord       4(r6),r3        #or in high bits
          movb      0(r0),r5        #get a byte of source
          addd      r1,r0           #add source warp
          addd      r2,r2           #shift destination left one bit
          addd      r3,r3           #top 32 bits too
          addr      r4[r5:q],r6     #get pointer to table
          ord       0(r6),r2        #or in low bits
          ord       4(r6),r3        #or in high bits
          movb      0(r0),r5        #get a byte of source
          addd      r1,r0           #add source warp
          addd      r2,r2           #shift destination left one bit
          addd      r3,r3           #top 32 bits too
          addr      r4[r5:q],r6     #get pointer to table
          ord       0(r6),r2        #or in low bits
          ord       4(r6),r3        #or in high bits
          movb      0(r0),r5        #get a byte of source
          addd      r1,r0           #add source warp
          addd      r2,r2           #shift destination left one bit
          addd      r3,r3           #top 32 bits too
          addr      r4[r5:q],r6     #get pointer to table
          ord       0(r6),r2        #or in low bits
          ord       4(r6),r3        #or in high bits
          movb      0(r0),r5        #get a byte of source
          addd      r1,r0           #add source warp
```

TL/EE/9698-8

FIGURE 3

```
addd    r2,r2           #shift destination left one bit
addd    r3,r3           #top 32 bits too
addr    r4[r5:q],r6     #get pointer to table
ord     0(r6),r2        #or in low bits
ord     4(r6),r3        #or in high bits
movb    0(r0),r5        #get a byte of source
addd    r1,r0           #add source warp
addd    r2,r2           #shift destination left one bit
addd    r3,r3           #top 32 bits too
addr    r4[r5:q],r6     #get pointer to table
ord     0(r6),r2        #or in low bits
ord     4(r6),r3        #or in high bits
movb    0(r0),r5        #get a byte of source
addd    r1,r0           #add source warp
addd    r2,r2           #shift destination left one bit
addd    r3,r3           #top 32 bits too
addr    r4[r5:q],r6     #get pointer to table
ord     0(r6),r2        #or in low bits
ord     4(r6),r3        #or in high bits
movb    0(r0),r5        #get a byte of source
addd    r1,r0           #add source warp
addd    r2,r2           #shift destination left one bit
addd    r3,r3           #top 32 bits too
addr    r4[r5:q],r6     #get pointer to table
ord     0(r6),r2        #or in low bits
ord     4(r6),r3        #or in high bits
ret     $0              #and return
```

TL/EE/9698-9

FIGURE 3 (Continued)

80x86 to Series 32000® Translation; Series 32000 Graphics Note 6

National Semiconductor
Application Note 529
Dave Rand

1.0 INTRODUCTION

This application note discusses the conversion of Intel 8088, 8086, 80188 and 80186 (referred to here as 80x86) source assembly language to Series 32000 source code. As this is not intended to be a tutorial on Series 32000 assembly language, please see the Series 32000 Programmers Reference Manual for more information on instructions and addressing modes.

2.0 DESCRIPTION

The 80x86 model has 6 general purpose registers (AX, BX, CX, DX, SI, DI), each 16 bits wide. 4 of these registers can be further addressed as 8-bit registers (AL, AH, BL, BH, CL, CH, DL, DH). Series 32000 has 8 general purpose registers (R0–R7), each 32 bits wide. Each Series 32000 register may be accessed as an 8-, 16- or 32-bit register. Two special purpose registers on the 80x86, SP and BP, are 16-bit stack and base pointers. These are represented in Series 32000 with the SP and FP registers, each 32-bit.

The 80x86 model is capable of addressing up to 1 Megabyte of memory. Since the 16-bit register pointers are only capable of addressing 64 kbytes, 4 segment registers (CS, DS, ES, SS) are used in combination with the basic registers to point to memory. Series 32000 registers and addressing modes are all full 32-bit, and may point anywhere in the 16 Megabyte (or 4 Gigabyte, depending on processor model) addressing range.

Device ports are given their own 16-bit address on the 80x86, and there is a complement of instructions to handle input and output to these ports. Device ports on Series 32000 are memory mapped, and all instructions are available for port manipulation.

There are 6 addressing modes for data memory on the 80x86: Immediate, Direct, Direct indexed, Implied, Base relative and Stack. There are 9 addressing modes on Series 32000: Register, Immediate, Absolute, Register-relative, Memory space, External, Top-of-stack and Scaled index. Scaled index may be applied to any of the addressing modes (except scaled index) to create more addressing modes. The following figure shows the 80x86 addressing modes, and their Series 32000 counterparts.

Series 32000 assembly code reads left-to-right, meaning source is on the left, destination on the right. As you can see, most of the 80x86 addressing modes fall into the register-relative class of Series 32000. Also note that the ADDW could have been ADDD, performing a 32-bit add instead of only a 16-bit.

Series 32000 also permits memory-to-memory (two address) operation. A common operation like adding two variables is easier in Series 32000. Series 32000 has the same form for all math operations (multiply, divide, subtract), as well as all logical operators.

80x86		Series 32000
ADD AX,1234	Immediate	ADDW $1234,R0
ADD AX,LAB1	Direct	ADDW LAB1,R0
ADD AX,16[SI]	Direct Indexed	ADDW 16(R6),R0
ADD AX,[SI]	Implied	ADDW 0(R6),R0
ADD AX,[BX]	Base Relative	ADDW 0(R1),R0
ADD AX,[BX+SI]	Base Relative Implied	ADDW R1[R6:B],R0
ADD AX,12[BX+SI]	Base Relative Implied Indexed	ADDW 12(R1)[R6:B],R0
ADD AX,4[BP]	Stack (Relative)	ADDW 4(FP),R0
PUSH AX	Stack	MOVW R0,TOS

80x86	Series 32000	
MOV AL,LAB1 ADD LAB2,AL	ADDB LAB1,LAB2	8-Bit Add Operation
MOV AX,LAB3 ADD LAB4,AX	ADDW LAB3,LAB4	16-Bit Add Operation

MOV AX,LAB5L
ADD LAB6L,AX
MOV AX,LAB5H
ADDC LAB6H,AX
	ADDD LAB5,LAB6	32-Bit Add Operation

Most 80x86 instructions have direct Series 32000 equivalents—with a major difference. Most 80x86 instructions affect the flags. Most Series 32000 instructions do not affect the flags in the same manner. For example, the 80x86 ADD instruction affects the Overflow, Carry, Arithmetic, Zero, Sign and Parity flags. The Series 32000 ADD instruction affects the Overflow and Carry flags. Programs that rely on side-effects of instructions which set flags must be changed in order to work correctly on Series 32000.

Table I gives a general guideline of instruction correlation between 80x86 and Series 32000. Many of the common subroutines in 80x86 may be replaced by a single instruction in Series 32000 (for example, 32-bit multiply and divide routines). Many special purpose instructions exist in Series 32000, and these instructions may help to optimize various algorithms.

3.0 IMPLEMENTATION

As an example, we will show some small 80x86 programs which we wish to convert to Series 32000. The first program reads a number of bytes from a port, waiting for status information. Below is the program in 80x86 assembly language:

```
;This program reads count bytes from port ioport, waiting for bit 7 of
;statport to be active (1) before reading each byte.
        xor     bx,bx           ;zero checksum
        mov     cx,count        ;get count of bytes
        mov     es,bufseg       ;get buffer segment
        lea     di,buffer       ;point to buffer offset
l1:     mov     dx,statport     ;get status port address
l2:     in      al,dx           ;read status port
        rcl     al,1            ;move bit 7 to carry
        jnc     l2              ;loop until status available
        mov     dx,ioport       ;point to data port
        in      al,dx           ;read port
        stosb                   ;store byte
        xor     ah,ah           ;zero high part of ax
        add     bx,ax           ;add to checksum
        loop    l1              ;loop for all bytes
        ret
```

TL/EE/9699-1

A direct translation of this program to Series 32000 using Table I, appears below. Note that this program will not work directly, due to the side effect of the rcl instruction being used.

```
#This program reads count bytes from port ioport, waiting for bit 7 of
#statport to be active (1) before reading each byte.
#
# Before optimization

        xord    r1,r1           # zero checksum
        movw    $count,r2       # get count of bytes
        addr    buffer,r5       # point to buffer
l11:    addr    statport,r3     # get status port address
l12:    movb    0(r3),r0        # read status port
        rotb    $1,r0           # move bit 7 to carry <<- does not work
        bcc     l12             # branch if carry clear
        addr    ioport,r3       # point to data port
        movb    0(r3),r0        # read port
        movb    r0,0(r5)        # store byte
        addqd   1,r5
        movzbw  r0,r0           # zero high part of ax
        addw    r0,r1           # add to checksum
        acbw    -1,r2,l11       # loop for all bytes
        ret     $0
```

TL/EE/9699-2

By using some of the special Series 32000 instructions, we can make this program much faster. The ROTB wil not work to test status, so we will replace that with a TBITB instruction. Since TBITB can directly address the port, there is no need to read the status port value at all. We will remove the read status port line, and the register load of r3. Reading the IO port as well can be done directly now, and we use a zero extension to ensure the high bits are cleared in preparation for the checksum addition. Note that it is easy to do a 32-bit checksum instead of only a 16-bit. Below is the 'optimized' code:

```
#This program reads count bytes from port ioport, waiting for bit 7 of
#statport to be active (1) before reading each byte.
#
# After optimization

        xord    r1,r1           # zero checksum
        movw    $count,r2       # get count of bytes
        addr    buffer,r5       # point to buffer
l11:
l12:    tbitb   $7,statport     # is bit 7 of status port valid?
        bfc     l12             # no, loop until it is
        movzbd  ioport,r0       # read io port
        movb    r0,0(r5)        # store in buffer
        addqd   1,r5
        addw    r0,r1           # add to checksum
        acbw    -1,r2,l11       # loop for all bytes
        ret     $0
```

TL/EE/9699-3

A second program shows, in 80x86 assembler, a method to copy and convert a string from mixed case ASCII to all upper case ASCII. This program is shown below:

```
;This program translates a null terminated ASCII string to uppercase
;
        mov     ds,buf1seg      ;point to input segment
        lea     si,buf1         ;point to input string
        mov     es,buf2seg      ;point to output segment
        lea     di,buf2         ;point to output string
        cld                     ;clear direction flag (increasing add)
l1:     lodsb                   ;get a byte
        cmp     al,'a'          ;is the char less than 'a'?
        jb      l2              ;yes, branch out
        cmp     al,'z'          ;is the char greater than 'z'?
        ja      l2              ;yes, branch out
        and     al,5fh          ;and with 5f to make uppercase
l2:     stosb                   ;store the character
        or      al,al           ;is this the last char?
        jnz     l1              ;no, loop for more
        ret                     ;yes, exit
```

TL/EE/9699-4

A direct translation to Series 32000 works fine, as is shown below:

```
#This program translates a null terminate ASCII string to uppercase
#
# Before optimization

        addr    buf1,r4         # point to input string
        addr    buf2,r5         # point to output string
111:    movb    0(r4),r0        # get a byte
        addqd   1,r0
        cmpb    $'a',r0         # is the char less than 'a'?
        blo     112             # yes, branch out
        cmpb    $'z',r0         # is the char greater than 'z'?
        bhi     112             # yes, branch out
```

TL/EE/9699-5

```
        andb    $0x5f,r0        # and with 5f to make uppercase
112:    movb    r0,0(r5)        # store the character
        addqd   1,r5
        cmpqb   0,r0            # is this the last char?
        bne     111             # no, loop for more
        ret     $0
```

TL/EE/9699-6

This program allows us to exploit another Series 32000 instruction, the MOVST (Move and String Translate). With a 256 byte external table, we can translate any byte to any other byte. In this example, we simply use the full range of ASCII values in the translation table, with the lower case entries containing uppercase values.

Watch for other optimization opportunities, especially with multiply and add sequences (the INDEXi instruction could be used), and possible memory to memory sequence changes. When optimizing Series 32000 code, it is important to fully utilize the Complex Instruction Set. Allow the fewest number of instructions possible to do the work. Use the advanced addressing modes where possible. Try to employ larger data types in programs (Series 32000 takes the same number of clocks to add Bytes, Words or Double words).

4.0 CONCLUSION

Series 32000 assembly language offers a much richer complement of instructions when compared to the 80x86 assembly language. Translation from 80x86 to Series 32000 is made much easier by this full instruction set.

```
#This program translates a null terminate ASCII string to uppercase
#
# After optimization

        movqd   -1,r0           # number of bytes in string max.
        addr    buf1,r1         # point to input string
        addr    buf2,r2         # point to output string
        addr    ctable,r3       # address of conversion table
        movqd   0,r4            # match on a zero
        movst   u               # move string, translate, until 0
        movqb   0,0(r2)         # move a zero to output string
        ret     $0
```

TL/EE/9699-7

TABLE I

The following is a conversion table from 80x86 mnemonics to Series 32000. Note that many of the conversions are not exact, as the 80x86 instructions may affect flags that Series 32000 instructions do not. A * marks those instructions that may be affected most by this change in flags. The i in the Series 32000 instructions refers to the size of the data to be operated on. It may be B for Byte, W for Word or D for Double. Most arithmetic instructions also support F for single-precision Floating Point, and L for double-precision Floating-Point.

80x86	Series 32000	Comments
AAA	—	Suggest changing algorithm to use ADDPi
AAD	—	Suggest changing algorithm to use ADDPi/SUBPi
AAM	—	"
AAS	—	Suggest changing algorithm to use SUBPi
ADC	ADDCi	
ADD	ADDi	
AND	ANDi	
BOUND	CHECKi	
CALL	BSR/JSR	
CBW	MOVXBW	You may directly sign-extend data while moving
CLC	BICPSRB $1	Usually not required
CLD	—	Direction encoded within string instructions
CLI	BICPSRW $0x800	Supervisor mode instruction
CMC	—	Usually not required
CMP	CMPi	
CMPS	CMPSi	Many options available
CWD	MOVXWD	You may directly sign-extend data while moving
DAA	—	Suggest changing algorithm to use ADDPi
DAS	—	Suggest changing algorithm to use SUBPi
DEC	ADDQi-1*	Watch for flag usage
DIV	DIVi	Note: Series 32000 uses signed division
ENTER	ENTER[reglist],d	Builds stack frame, saves regs, allocates stack space
ESC	—	Usually used for Floating Point-see Series 32000 FP instructions
HLT	WAIT	
IDIV	DIVi/QUOi	DIVi rounds towards -infinity, QUOi to zero
IMUL	MULi	
IN	—	Series 32000 uses memory-mapped I/O
INC	ADDQi 1*	Watch for flag usage
INS	—	Series 32000 uses memory mapped I/O
INT	SVC	Not exact conversion, but usually used to call O/S
INTO	FLAG	Trap on overflow
IRET	RETI $0	Causes Interrupt Acknowledge cycle
JA/JNBE	BHI	Unsigned comparison
JAE/JNB	BHS	Unsigned comparison
JB/JNAE	BLT	Unsigned comparison
JBE/JNA	BLS	Unsigned comparison
JCXZ	—	Use CMPQi 0, followed by BEQ
JE/JZ	BEQ	Equal comparison
JG/JNLE	BGT	Signed comparison
JGE/JNL	BGE	Signed comparison
JL/JNGE	BLT	Signed comparison
JLE/JNG	BLE	Signed comparison
JMP	BR/JUMP	
JNE/JNZ	BNE	Not Equal comparison
JNO	—	Subroutines should be used for these instructions
JNP	—	as most Series 32000 code will not need these
JNS	—	operations.
JO	—	"
JP	—	"
JPE	—	"
JPO	—	"
JS	—	"
LAHF	—	SPRB UPSR,xxx may be useful
LDS	—	Segment registers not required on Series 32000
LEA	ADDR	
LEAVE	EXIT[reglist]	Restores regs, unallocates frame and stack
LES	—	Segment registers not required
LOCK	—	SBITIi, CBITIi interlocked instructions
LODS	MOVi/ADDQD	MOV instruction followed by address increment
LOOP	ACBi-1	ACBi may use memory or register

TABLE I (Continued)

80x86	Series 32000	Comments
LOOPE	—	BEQ followed by ACBi may be used
LOOPNE	—	BNE followed by ACBi may be used
LOOPNZ	—	BNE followed by ACBi may be used
LOOPZ	—	BEQ followed by ACBi may be used
MOV	MOVi	
MOVS	MOVSi	Many options available
MUL	MULi	Series 32000 uses signed multiplication
NEG	NEGi	Two's complement
NOP	NOP	
NOT	COMi	One's complement
OR	ORi	
OUT	—	Series 32000 uses memory mapped I/O
OUTS	—	Series 32000 uses memory mapped I/O
POP	MOVi TOS,	TOS addressing mode auto increments/decrements SP
POPA	RESTORE [r0,r1 . . r7]	Restores list of registers
POPF	LPRB UPSR,TOS	User mode loads 8 bits, supervisor 16 bits of PSR
PUSH	MOVi xx,TOS	Any data may be moved to TOS
PUSHA	SAVE [r0,r1 . . r7]	Saves list of registers
PUSHF	SPRB UPSR,TOS	User mode stores 8 bits, supervisor 16 bits of PSR
RCL	ROTi*	Does not rotate through carry
RCR	ROTi*	Does not rotate through carry
REP	—	Series 32000 string instructions use 32-bit counts
RET	RET	
ROL	ROTi	
ROR	ROTi	Rotates work in both directions
SAHF	—	LPRB UPSR,xx may be useful
SAL	ASHi	Arithmetic shift
SAR	ASHi	Arithmetic shift works both directions
SBB	SUBCi	
SCAS	SKPSi	Many options available
SHL	LSHi	Logical shift
SHR	LSHi	Logical shift works both directions
STC	BISPSRB $1	
STD	—	Direction is encoded in string instructions
STI	BISPSRW $0x800	Supervisor mode instruction
STOS	MOVi/ADDQD	MOV instruction followed by address increment
SUB	SUBi	
TEST	—	TBITi may be used as a substitute
WAIT	—	
XCHG	—	MOVi x,temp; MOVi y,x; MOVi temp,y
XLAT	MOVi x[R0:b],	Scaled index addressing mode
XOR	XORi	

Bit Mirror Routine; Series 32000® Graphics Note 7

National Semiconductor
Application Note 530
Dave Rand

1.0 INTRODUCTION

The bit mirror routine is designed to reorder the bits in an image. The bits are swapped around a fixed point, that being one half of the size of the data, as is shown for the byte mirror below. These routines can be used for conversion of 68000 based data.

2.0 DESCRIPTION

	Bit Number								Hex Value
	7	6	5	4	3	2	1	0	
Source	1	0	1	1	0	0	1	0	B2
Result of Mirror	0	1	0	0	1	1	0	1	4D

The "mirror", in this case, is between bits 3 and 4.

Several different algorithms are available for the mirror operation. The best algorithm to mirror a byte takes 20 clocks on a NS32016 (about 2.5 clocks per bit), and uses a 256 byte table to do the mirror operation. The table is reproduced at the end of this document. To perform a byte mirror, the following code may be used. The byte to be mirrored is in R0, and the destination is to be R1.

```
MOVB mirtab[r0:b],r1     #Mirror a byte
```

TL/EE/9700-1

An extension of this algorithm is used to mirror larger amounts of data. To mirror a 32-bit block of data from one location to another, the following code may be used. Register R0 points to the source block, register R1 points to the destination. R2 is used as a temporary value.

```
MOVZBD    0(r0),r2              #get first byte
MOVB      mirtab[r2:b],3(r1)    #store in last place
MOVB      1(r0),r2              #get next byte
MOVB      mirtab[r2:b],2(r1)    #store in next place
MOVB      2(r0),r2              #get the third byte
MOVB      mirtab[r2:b],1(r1)    #store in next place
MOVB      3(r0),r2              #get the last byte
MOVB      mirtab[r2:b],0(r1)    #first place
```

TL/EE/9700-2

This code uses 33 bytes of memory, and just 169 clocks to execute. Larger blocks of data can be mirrored with this method as well, with each additional byte taking about 40 clocks.

Registers can also be mirrored with this method, with just a few more instructions. To mirror R0 to R1, for example, the following code could be used. R2 is used as a temporary variable.

```
MOVZBD    r0,r2           #get lsbyte
MOVB      mirtab[r2:b],r1 #mirror the byte
LSHD      $8,r1           #move into higher byte of destination
LSHD      $-8,r0          #and of source
MOVB      r0,r2           #get lsbyte
MOVB      mirtab[r2:b],r1 #mirror the byte
LSHD      $8,r1           #move into higher byte of destination
LSHD      $-8,r0          #and of source
MOVB      r0,r2           #get lsbyte
MOVB      mirtab[r2:b],r1 #mirror the byte
LSHD      $8,r1           #move into higher byte of destination
LSHD      $-8,r0          #and of source
MOVB      r0,r2           #get lsbyte
MOVB      mirtab[r2:b],r1 #mirror the byte
```

TL/EE/9700-3

This code occupies 49 bytes, and executes in 286 clocks on an NS32016.

If space is at a premium, a shorter table may be used, at the expense of time. Each nibble (4 bits) instead of each byte is processed. This means that the table only requires 16 entries. To mirror a byte in R0 to R1, the following code can be used. R2 is used as a temporary variable.

```
MOVB    r0,r2               #get lsbyte
ANDD    $15,r2              #mask to get ls nibble
MOVB    mirtb16[r2:b],r1    #mirror the nibble
LSHD    $4,r1               #high nibble of destination
LSHD    $-4,r0              #and of source
MOVB    r0,r2               #get lsbyte
ANDD    $15,r2              #mask to get ls nibble
ORB     mirtb16[r2:b],r1    #mirror the nibble
```

TL/EE/9700–4

This code requires 32 bytes of memory, and executes in 125 clock cycles on an NS32016. A slightly faster time (100 clocks) may be obtained by adding a second table for the high nibble, and eliminating the LSHD 4,r1 instruction.

TABLES

MIRTAB is a table of all possible mirror values of 8 bits, or 256 bytes. MIRTB16 is a table of all possible mirror values of 4 bits, or 16 bytes. These tables should be aligned for best performance. They may reside in code (PC relative), or data (SB relative) space.

```
mirtab:
        .byte   0x00,0x80,0x40,0xc0,0x20,0xa0,0x60,0xe0,0x10,0x90,0x50
        .byte   0xd0,0x30,0xb0,0x70,0xf0
        .byte   0x08,0x88,0x48,0xc8,0x28,0xa8,0x68,0xe8,0x18,0x98,0x58
        .byte   0xd8,0x38,0xb8,0x78,0xf8
        .byte   0x04,0x84,0x44,0xc4,0x24,0xa4,0x64,0xe4,0x14,0x94,0x54
        .byte   0xd4,0x34,0xb4,0x74,0xf4
        .byte   0x0c,0x8c,0x4c,0xcc,0x2c,0xac,0x6c,0xec,0x1c,0x9c,0x5c
        .byte   0xdc,0x3c,0xbc,0x7c,0xfc
        .byte   0x02,0x82,0x42,0xc2,0x22,0xa2,0x62,0xe2,0x12,0x92,0x52
        .byte   0xd2,0x32,0xb2,0x72,0xf2
        .byte   0x0a,0x8a,0x4a,0xca,0x2a,0xaa,0x6a,0xea,0x1a,0x9a,0x5a
        .byte   0xda,0x3a,0xba,0x7a,0xfa
        .byte   0x06,0x86,0x46,0xc6,0x26,0xa6,0x66,0xe6,0x16,0x96,0x56
        .byte   0xd6,0x36,0xb6,0x76,0xf6
        .byte   0x0e,0x8e,0x4e,0xce,0x2e,0xae,0x6e,0xee,0x1e,0x9e,0x5e
        .byte   0xde,0x3e,0xbe,0x7e,0xfe
        .byte   0x01,0x81,0x41,0xc1,0x21,0xa1,0x61,0xe1,0x11,0x91,0x51
        .byte   0xd1,0x31,0xb1,0x71,0xf1
        .byte   0x09,0x89,0x49,0xc9,0x29,0xa9,0x69,0xe9,0x19,0x99,0x59
        .byte   0xd9,0x39,0xb9,0x79,0xf9
        .byte   0x05,0x85,0x45,0xc5,0x25,0xa5,0x65,0xe5,0x15,0x95,0x55
        .byte   0xd5,0x35,0xb5,0x75,0xf5
        .byte   0x0d,0x8d,0x4d,0xcd,0x2d,0xad,0x6d,0xed,0x1d,0x9d,0x5d
        .byte   0xdd,0x3d,0xbd,0x7d,0xfd
        .byte   0x03,0x83,0x43,0xc3,0x23,0xa3,0x63,0xe3,0x13,0x93,0x53
        .byte   0xd3,0x33,0xb3,0x73,0xf3
        .byte   0x0b,0x8b,0x4b,0xcb,0x2b,0xab,0x6b,0xeb,0x1b,0x9b,0x5b
        .byte   0xdb,0x3b,0xbb,0x7b,0xfb
        .byte   0x07,0x87,0x47,0xc7,0x27,0xa7,0x67,0xe7,0x17,0x97,0x57
        .byte   0xd7,0x37,0xb7,0x77,0xf7
        .byte   0x0f,0x8f,0x4f,0xcf,0x2f,0xaf,0x6f,0xef,0x1f,0x9f,0x5f
        .byte   0xdf,0x3f,0xbf,0x7f,0xff

mirtb16:
        .byte   0x0,0x8,0x4,0xc,0x2,0xa,0x6,0xe,0x1,0x9,0x5
        .byte   0xd,0x3,0xb,0x7,0xf
```

TL/EE/9700–5

Operating Theory of the Series 32000® GNX™ Version 3 Compiler Optimizer

National Semiconductor
Application Note 583
Series 32000 Applications

1.0 INTRODUCTION

The main difference between the GNX-Version 3 compilers and other compilers is the optimizer. Recompiling and optimizing with a GNX-Version 3 compiler will result in a 10% to 200% speedup for most programs, with an average improvement of over 30%. This chapter describes some of the advanced optimization techniques used by the compiler to improve speed or save space. The most important techniques are:

- Value propagation
- Constant folding
- Redundant-assignment elimination
- Partial-redundancy elimination
- Common-subexpression elimination
- Flow optimizations
- Dead-code removal
- Loop-invariant code motion
- Strength reduction
- Induction variable elimination
- Register-allocation by coloring
- Peephole optimizations
- Memory-layout optimizations
- Fixed frame

The following sections describe these techniques in more detail.

2.0 THE OPTIMIZER

The optimizer, shared by all the GNX-Version 3 compilers, is based on advanced optimization theory developed over the past 15 years. Central to the optimizer is an innovative global-data-flow-analysis technique which simplifies the optimizer's implementation. It allows the optimizer to perform some unique optimizations in addition to all the standard optimizations found in other compilers. Optimizations are performed globally on the code of a whole procedure at a time and not just in a local context.

The optimizer is implemented as a multi-step process. Each step performs its particular optimizations and provides new opportunities for the optimizations of the next step.

2.1 STEP ONE

The first step in the optimization process is to read in the source program one procedure at a time and to partition this procedure into basic blocks. A basic block is a straight line sequence of code with a branch only at the entry or exit. Some of the optimizations performed during this step are:

- **Value Propagation**

  Value propagation (or copy propagation) is the attempt to replace a variable with the most recent value that has been assigned to it. This optimization is primarily useful in the special case of constant propagation. It is important because it creates opportunities for other optimizations. Value propagation can be turned off by the /CODE_MOTION optimization flag (-Om on UNIX® systems).

- **Constant Folding**

  If an expression or condition consists of constants only, it is evaluated by the optimizer into one constant, thereby avoiding this computation at run-time. The optimizer, using algebraic properties such as the commutative, associative and distributive law, sometimes rearranges expressions to allow constant folding of part of an expression.

  The GNX-Version 3 C compiler also folds floating-point constant expressions. This feature can be turned off using the /NOFLOAT_FOLD option (-Oc on UNIX systems) of the optimizer.

- **Redundant-Assignment Elimination**

  The optimizer detects and eliminates assignments to variables which are not used later in the program or which are assigned again before being used. This optimization can often be applied as a result of value propagation.

  Value propagation, constant folding, and redundant assignment elimination are illustrated in *Figure 1.*

The program sequence

```
a = 4;
if (a*8 < 0) b = 15;
else b = 20;
...code which uses b but not a...
```

is translated by the GNX-Version 3 C compiler front end into the following intermediate code

```
a ← 4
if (a*8 ≥ 0) goto L1
b ← 15
goto L2
L1: B ← 20
L2: ...
```

which is transformed by "value propagation" into

```
a ← 4
if (4*8 ≥ 0) goto L1
b ← 5
goto L2
L1: b ← 20
L2: ...
```

which after "constant folding" becomes

```
a ← 4
if (true) goto L1
b ← 15
goto L2
L1: b ← 20
L2: ...
```

"dead code removal" results in

```
a ← 4
goto L1
L1: b ← 20
L2: ...
```

which is transformed by another "flow optimization" into

```
a ← 4
b ← 20
```

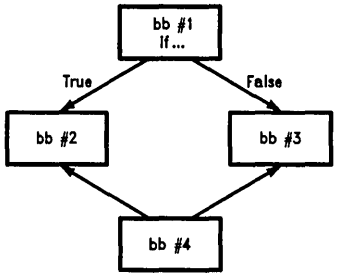
Since there is no further use of a, a ← 4 is a "redundant assignment:"

```
b ← 20
...
```

FIGURE 1. Relationship between Various Optimizations

2.2 STEP TWO

The second step in the optimization process is the construction of the program's "flow graph." This is a graph in which each node represents a basic block. A basic block is a linear segment of code with only one entry point and one exit point. If there is a path in the program that leads from one basic block to another, then an "arrow" is drawn in the graph to represent this path. *Figure 2* illustrates a flow graph, representing an "if-then-else" sequence.



TL/EE/10344–1

FIGURE 2. Flow Graph

During the construction of the flow graph, additional optimizations can be performed:

- **Flow Optimizations**

  Flow optimizations reduce the number of branches performed in the program. One example is to replace a branch whose target is another branch with a direct branch to the ultimate target. This often makes the second branch redundant. At other times, code is reordered to eliminate unnecessary branches. Branches to "return" are replaced by the return-sequence itself.

- **Dead Code Removal**

  Flow optimizations are also designed to help the optimizer discover code which will never actually be executed. Removal of this code, called "dead code removal", results in smaller object programs.

2.3 STEP THREE

Step three of the optimization process is called "global-data-flow-analysis". It identifies desirable global code transformations which speed program execution. Many of these concentrate on speeding up loop execution, since most programs spend 90% or more of their time in loops. Global-data-flow-analysis is the computation of a large number of properties for each expression in the procedure.

Unlike most optimizers, which employ unrelated and separate techniques, the optimizer centers around one innovative technique which involves the recognition of a situation called "partial redundancy". This technique is so powerful that many other optimizations turn out to be special cases. The central idea is that it is wasteful to compute an expression, say a * b, twice on the same path; it is often faster to save the result of the first computation and then replace the fully redundant second computation with the saved value. More common, however, is the case in which an expression is partially redundant; there is one path to an expression, which already contains a computation of that expression, but another path to that same expression does not.

The following optimizations are performed by a common technique:

- **Elimination of Fully Redundant Expressions**

  This optimization is often called "Common Subexpression Elimination". It is relatively simple to avoid the recomputation of fully redundant expressions. The optimizer saves the result of the first computation (usually in a register variable) and uses the saved value in place of the second computation. Performance-conscious programmers sometimes do this themselves, but many cases, such as array index and record number calculations, are recognized only by the optimizer.

- **Partial Redundancy Elimination**

  A partially redundant expression can be eliminated in two steps. First, insert the expression on the paths in which it previously did not occur; this makes the expression fully redundant. Second, save the first computations and use the saved value to replace the redundant computation. An example of this optimization is shown in *Figure 3*.

  Partial redundancy elimination sometimes results in slightly larger code, but execution is not harmed, since all inserted expressions are in parallel and only one is actually executed.

- **Loop Invariant Code Motion**

  If an expression occurs within a loop and its value does not change throughout that loop, it is called "loop invariant". Loop invariant expressions are also partially redundant. This can be understood by realizing that there are two paths into the loop body: one is through the loop entry (the first time the loop is executed), and the other is from the end of the loop, while the exit condition is false. Loop invariant computations are, therefore, removed from the loop in the same way: the expression is first inserted on the entry path to the loop, and then the expression is saved on the entry path in a register, while the redundant computation in the loop is replaced by that register.

- **Strength Reduction**

  This optimization globally replaces complex operations by simpler ones. This is primarily useful for reducing complex array-subscript computations (involving multiplication into simpler additions).

```
for (i = 0; i < 15; i+ = 0)
a [i] = 0;
```

  is transformed into:

```
for (i = 0, p = a; i < 15; i+ = 1, p+ = 4)
    *p = 0;
```

- **Induction Variable Elimination**

  Induction variables are variables that maintain a fixed relation to other variables. The use of such variables can often be replaced by a simple transformation. For instance, the example given for strength reduction can be reduced to the following:

```
for (p = a; p < a + 60; p+= 4)
*p = 0;
```

In the following code, a*b is "partially redundant" (computed twice only if C is true):

```
if (C)
   x = a*b;
else
   b = b + 10
y = a*b;
```

It is first transformed into a "fully redundant" expression

```
if C = 1
   x ← a*b
else
   b ← b + 10
   temp ← a*b
y ← a*b
```

Then, as in the simple case of "redundant expression elimination," this is reduced to

```
if C = 1
   temp ← a*b
   x ← temp
else
   b ← b + 10
   temp ← a*b
y ← temp
```

Now, the expression a*b is computed only once on any path.

FIGURE 3. Example of Partial Redundancy Elimination

2.4 STEP FOUR

The fourth optimization step performed by the optimizer, and possibly the most profitable, is the "register allocation" phase. Register allocation places variables in machine registers instead of main memory. References to a register are always much faster and use less code space than respective memory references.

The algorithm used by the optimizer is called the "coloring algorithm". First, global-flow-analysis is performed to determine the different live ranges of variables within the procedure. A live range is the program path along which a variable has a particular value. Generally, an assignment to a variable starts a new live range; this live range terminates with the last use of that assigned value.

The optimizer subsequently constructs a graph as follows: each node represents a live range; two nodes are connected if there exists a point in the program in which the two live ranges intersect. The allocation of registers to live ranges is now the same as coloring the nodes of the graph so that two connected nodes have different colors. This is a classic problem from graph theory, for which good solutions exist. If there are not enough registers, more frequently used variables have higher priority than less frequently used ones. Loop nesting is taken into account when calculating the frequency of use, meaning that variables used inside of loops have higher priority than those that are not.

Most optimizing compilers attempt register allocation only for true local variables, for which there is no danger of "aliasing." An alias occurs when there are two different ways to access a variable. This can happen when a global variable is passed as reference parameter; the variable can be accessed through its global name, or through the parameter alias. A common case in C is when the address of a variable is assigned to a pointer.

The optimizer takes a more general approach by considering all variables with appropriate data types as candidates for register allocation, including global variables, variables whose addresses have been taken, array elements, and items pointed to by pointers. These special candidates cannot reside in registers across procedure calls and pointer references and, therefore, normally have lower priority than local variables. However, instead of completely disqualifying the special candidates in advance, the decision is made by the coloring algorithm.

Additional important optimizations performed by the register allocator are:

- **Use of Safe and Scratch Registers**

  The Series 32000 machine registers are, by convention, divided into two groups: registers R0 through R2 and F0 through F3, the so-called "scratch" registers which can be used as temporaries but whose values may be changed by a procedure call, and the "safe" registers (R3 through R7 and F4 through F7) which are guaranteed to retain their value across procedure calls. The register allocator spends a special effort to maximize the use of scratch registers, since it is not necessary to save these upon entry or restore them upon exit from the current procedure. The use of scratch registers, therefore, reduces the overhead of procedure calls.

- **Register Parameter Allocation**

  The register allocator attempts to detect routines, whose parameters can be passed in registers. This is possible for static routines only, since by definition all the calls to such routines are visible to the optimizer. Calls to other (externally callable) routines are subject to the standard Series 32000 calling sequence. Passing parameters in registers in another way to reduce the overhead of procedure calls.

2.5 STEP FIVE

The last optimization step consolidates the results of all previous steps by writing out the optimized procedure in intermediate form for the separate code generator. Some reorganizations take place during this step. Local variables which have been allocated in registers are removed from the procedure's activation record (frame), which is reordered to minimize overall frame size.

3.0 THE CODE GENERATOR

The back end (code generator) attempts to match expression trees with optimal code sequences. It applies standard techniques to minimize the use of temporary registers, which are necessary for the computation of the subexpressions of a tree. The main strength of the code generator lies in the number of "peephole optimizations" it performs.

Peephole optimizations are machine-dependent code transformations that are performed by the code generator on small sequences of machine code just before emitting the code. Some of the most important peephole transformations are listed below:

- The code for maintaining the frame of routines which have no local variables, or whose variables are all allocated in registers, is removed.
- Switch statements are optimized into binary search, linear search or table-indexed code (using the Series 32000 CASE instruction), in order to obtain optimal code in each situation.
- The stack and frame areas are always aligned for minimal data fetches.
- Reduction of arithmetic identities, i.e., x*1 = x, x+0 = x, etc.
- Use of the ADDR instruction instead of ADD of three operands.
- Some optimizations performed in the optimizer, such as the application of the distributive law of algebra, i.e., (10+i)*4 = 40+4*i, provide additional opportunities to the code generator to fully exploit the Series 32000's addressing modes.
- Use of ADDR instead of MOVZBD of small constant.
- Strength Reduction Optimizations. Use of MOVD instead of MOVF from memory to memory; use of index addressing mode instead of multiplication by 2, 4 or 8; use of combinations of ADDR instructions or shift and ADD sequences instead of multiplication by other constants up to 200.

- Fixed Frame Optimization. An important contribution of the code generator is its ability to precompute the stack requirements of a procedure in advance. This allows the generation of code which does not use (nor update) the FP (frame pointer), resulting in cheaper calling sequences.

  This optimization is most useful when the procedure contains many procedure calls because it is not necessary to execute code to adjust the stack after every call. Parameters are moved to the pre-allocated space instead of pushing them on to the stack using the top-of-stack addressing mode. Note that when using this optimization, the run-time stack pointer stays the same throughout the procedure, and all references to local variables are relative to it and not the FP. Also note that the evaluation order of parameters is unpredictable because parameters that take more space to evaluate are treated first to save space.

While most optimizations are beneficial for both speed and space, some optimizations favor one over the other. The default setting of the optimizer switch favors speed over space in trade-off situations. The following optimizations are trade-off situations which are affected by an optimization flag.

- Code is not aligned after branches.
- All returns within the code are replaced by a jump to a common return sequence.
- Certain space-expensive peephole transformations are not performed.

4.0 MEMORY LAYOUT OPTIMIZATIONS

The following memory layout optimizations are performed by the GNX-Version 3 C compiler:

- Frame variables that are allocated in registers are removed from the frame.
- Internal, static routines whose parameters are passed in registers have smaller frames.
- The stack alignment is always maintained. Stack parameters are passed in aligned positions.
- Frame variables are allocated in aligned positions. The compiler reorders these variables to save overall frame space.
- Code is aligned after every unconditional jump.

Application Development Using Multiple Programming Languages

National Semiconductor
Application Note 590

INTRODUCTION

National Semiconductor provides optimizing compilers for software development for *Series 32000* based designs. GNX-Version 3 is the name of the software tools family that includes the optimizing compilers. Languages supported in GNX-Version 3 include compilers that support C, Pascal, FORTRAN-77, and Modula-2. Each of the optimizing compilers share a common optimizer and code generator and intermediate representation. This greatly simplifies the process of mixed-language programming, or combining modules written in different high-level languages in the same application. The ability to use mixed-language programming simplifies the porting of pre-existing applications and code reuse.

Mixed-language programs are frequently used for a two reasons. First, one language may be more convenient than another for certain tasks. Second, code sections, already written in another language (e.g., an already existing library function), can be reused by simply making a call to them.

A programmer who wishes to mix several programming languages needs to be aware of subtle differences between the compilation of the various languages. The following sections describe the issues the user needs to be aware of when writing mixed-language programs and then compiling and linking such programs successfully.

WRITING MIXED-LANGUAGE PROGRAMS

The mixed-language programmer should be aware of the following topics:

- Name Sharing—Potential conflicts including permitted name-lengths, legal characters in identifiers, compiler case sensitivity, and high-level to assembly-level name transformations.
- Calling Convention—The way parameters are passed to functions, which registers must be saved, and how values are returned from functions. The application note *Portability Issues and the GNX-Version 3 C Optimizing Compiler* contains a description of parameter passing. This information is also contained in Appendix A of the *GNX-Version 3 compiler reference manuals.*
- Declaration Conventions—The demands that different languages impose when referring to an outside symbol (be it a function or a variable) that is not defined locally in the referring source file. Note that this is also true of references to an outside symbol that is not in the same language as that of the referring source file.

To help the programmer avoid these potential problems, a set of rules for writing mixed-language programs has been devised. Each rule consists of a short mnemonic name (for easy reference), the audience of interest for the rule, and a brief description of the rule.

Table I summarizes all of the rules in the context of each possible cross-language pair.

TABLE I. Cross-Language Pairs

	C	Pascal	FORTRAN 77	Modula-2	Series 32000 Assembly
Series 32000 Assembly	"_" prefix	"_" prefix include ext case sensitivity	"_" prefix "_" suffix ref args case sensitivity	"_" prefix DEF & IMPORT init code	
Modula-2	DEF & IMPORT init code	DEF & IMPORT init code include ext case sensitivity	"_" suffix DEF & IMPORT init code ref args case sensitivity		"_" prefix DEF & IMPORT init code
FORTRAN 77	"_" suffix ref args case sensitivity	"_" suffix include ext ref args		"_" suffix DEF & IMPORT init code ref args case sensitivity	"_" prefix "_" suffix ref args case sensitivity
Pascal	include ext case sensitivity		"_" suffix include ext ref args	DEF & IMPORT init code include ext case sensitivity	"_" prefix include ext case sensitivity
C		include ext case sensitivity	"_" suffix ref args case sensitivity	DEF & IMPORT init code	"_" prefix

RULE 1 case sensitivity

This rule is of interest to every programmer who mixes programming languages.

Modula-2, C, and Series 32000 assembly are case sensitive while FORTRAN 77 and Pascal are not (at least according to the standard). Programmers who share identifiers between these two groups of languages must take this into account. To avoid problems with case sensitivity, the programmer can:

1. Take case to use case-identical identifiers in all sources and compile FORTRAN 77 and Pascal sources using the case-sensitive option (CASE_SENSITIVE on VMS, -d on UNIX).
2. Use only lower-case letters for identifiers which are shared with FORTRAN 77 or Pascal, since the FORTRAN 77 and Pascal compilers fold all identifiers to lower-case if not given the case-sensitive option.

RULE 2 "_" prefix

This rule is of interest to those who mix high-level languages with assembly code.

All compilers map high-level identifier names into assembly symbols by prepending these names with an underscore. This ensures that user-defined names are never identical to assembly reserved words. For example, a high-level symbol NAME, which can be a function name, a procedure name, or a global variable name, generates the assembly symbol _NAME.

Assembly written code which refers to a name defined in any high-level language should, therefore, prepend an underscore to the high-level name. Stated from a high-level language user viewpoint, assembly symbols are not accessible from high-level code unless they start with an underscore.

RULE 3 "_" suffix

This rule is of interest to those who mix FORTRAN 77 with C, Pascal, Modula-2, or assembly code.

The FORTRAN 77 compiler appends an underscore to each high-level identifier name (in addition to the action described in RULE 1). The reason for an appended underscore is to avoid clashes with standard-library functions that are considered part of the language, e.g., the FORTRAN 77 WRITE instruction. For example, a FORTRAN 77 identifier NAME is mapped into the assembly symbol _NAME_.

Therefore, a C, Pascal, Modula-2, or assembly program that refers to a FORTRAN 77 identifier name should append an underscore to that name. Stated from a FORTRAN 77 user viewpoint, it is impossible to refer to an existing C, Pascal, Modula-2, or assembly symbol from FORTRAN 77 unless the symbol terminates with an underscore.

RULE 4 ref args

This rule is of interest to those who mix FORTRAN 77 with other languages.

Any language which passes an argument to a FORTRAN 77 routine must pass its address. This is because a FORTRAN 77 argument is always passed by reference, i.e., a routine written in FORTRAN 77 always expects addresses as arguments.

Routines not written in FORTRAN 77 cannot be called from a FORTRAN 77 program if the called routines expect any of their arguments to be passed by value. Only routines which expect all their arguments to be passed by reference can be called from FORTRAN 77.

Pascal and Modula-2 programs must declare all FORTRAN 77 routine arguments using var. C programs must prepend the address operator & to FORTRAN 77 routine arguments in the call.

The C, Pascal, or Modula-2 programmer who wants to pass an unaddressable expression (such as a constant) to a FORTRAN 77 routine, must assign the expression to a variable and pass the variable, by reference, as the argument.

RULE 5 include ext

This rule is of interest to Pascal programmers who want to share variables between different source files which may or may not be written in Pascal.

Pascal sources which share global variables must define these variables exactly once in an external header (include) file. The external header file has to be included in all Pascal source files which access the shared global variable, and its name must have a .h extension.

RULE 6 DEF and IMPORT

This rule is of interest to those who mix Modula-2 with other languages.

Modula-2 modules which access external symbols must import external symbols. If external symbols are not defined in Modula-2 modules but defined in other languages, the programmer must export these symbols to conform with the strict checks of the Modula-2 compiler.

External symbols can be exported by writing a "dummy" DEFINITION MODULE which exports all of the foreign language symbols, making them available to Modula-2 programs.

This export must be nonqualified to prevent the module name from being prepended to the symbol name.

RULE 7 init code

This rule is of interest to those who mix Modula-2 with other languages.

Modula-2 modules which import from external modules activate the initialization code of the imported modules before they start executing. The initialization code entry-point is identical to the imported module name.

To avoid getting an "Undefined symbol" message from the linker, the programmer should define a possibly empty, initialization function for every imported module. This is in case the implementation part of that module is not written in Modula-2. It should be noted that the initialization code is not necessarily called during run-time. Initialization code is executed if, and only if, the following two conditions hold true:

1. The main program code is written in Modula-2.
2. The Modula-2 routines which are supposed to activate the initialization part are not called indirectly through some non-Modula-2 code.

In addition to these rules, a few points should be noted. First, GNX Version 3 FORTRAN 77 allows identifiers longer than the six character maximum of traditional FORTRAN compilers. Second, the family of GNX Version 3 compilers allows the use of underscores in identifiers. Both of these enhancements simplify name sharing.

IMPORTING ROUTINES AND VARIABLES

The general conventions of all languages must be kept in mixed-language programs. In particular, externals must be declared in those program sections which import them. The following are examples of declarations of external (imported) functions/procedures and external (imported) variables in each language. The examples are in the form:

Caller Language: *external (imported) functions/procedures or external (imported) variables*

```
C: extern int func_( );
   or
   extern int var_name_;
```

Note that the strict reference C model (draft-proposed ANSI C standard) is assumed. If the model is relaxed, then the external declarations are not mandatory.

```
FORTRAN 77: INTEGER func
             or
            COMMON /var_name/ local_name
   Pascal:  function func_: integer;
            external;
            procedure proc_; external;
             or
            #include "var_def.h"
```

where the file var_def.h contains the following declaration:

```
             var
              var_name_: integer;
```

as explained in RULE 5 (include ext).

```
Modula-2:    FROM modula_name IMPORT func_
              or
             FROM module_name IMPORT
             var_name_
Series 32000: .globl _func_
  assembly     or
             .globl _var_name_
```

USING THE ASM KEYWORD

The keyword asm is recognized to enable insertion of assembly instructions directly into the assembly file generated. The syntax of its use is

asm *(constant-string)*;

where *constant-string* is a double-quoted character string.

Asm can be used inside of functions as a statement and out of functions in the scope of global declarations. A newline character will be appended to the given string in the assembly code.

Example: if for the C source:

```
i++;
j+ = 2;
```

the assembly code generated is:

```
addqd $1, _i
addqd $2, _j
```

then the assembly code generated for:

```
i++;
asm ("movd _i, r0");
j+ = 2;
```

will be:

```
addqd $1, _i
movd _i, r0
addqd $2, _j
```

Note: The word asm is a reserved keyword. Using asm as an identifier is a syntax error. Existing programs using such identifiers must be modified.

In support of mixed-language programming, the compiler also recognizes and compiles appropriate files written in other programming languages. Files with a .s suffix are assembly source programs and may be assembled (to produce .o files) and linked. Pascal, FORTRAN 77, and Modula-2 source files are also recognized, and compile appropriately if your system includes the National Semiconductor GNX Version 3 compiler for those languages. The suffixes for these files are listed in Table II.

TABLE II. Filename Conventions

File Name Suffix	File Type
.c	C Source File
.i	Preprocessed C Source File
.f, .for	FORTRAN 77 Source File
.F, .FOR	FORTRAN 77 Source with cpp Directives
.m, .mod	Modula-2 Source File
.M, .MOD	Modula-2 Source with cpp Directives
.def	Modula-2 Definition Module Source File
.DEF	Modula-2 Definition Module Source with cpp Directives
.p, .pas	Pascal Source File
.P, .PAS	Pascal Source with cpp Directives
.s	Assembly Source File
.o	Object Code
.a	Library Archive File

COMPILING MIXED-LANGUAGE PROGRAMS

After writing different program parts in different languages, keeping in mind the rules previously mentioned, the mixed-language programmer must still link and load these parts to make them run successfully. Three points should be mentioned in conjunction with the successful linking and loading of programs. These are as follows:

- External library (standard or nonstandard) routines must be bound with the user-written code that calls them.
- Initialization code which arranges to pass program parameters to the main program and then calls the main program, sometimes has to be bound with user-written code.
- The entry point of the code, i.e., the location where the program starts executing, should be determined.

In some cases, a standard is not so widely accepted as with Modula-2. In these cases, the user must be aware of the libraries that are available and the calling conventions of the main program used by the operating system.

LIBRARIES

Table III lists libraries associated with each compiler. When programming with mixed-languages, the libraries associated with the languages used must be bound with the program during the link phase of compilation.

TABLE III. Compilers and their Associated Libraries

Compiler (Driver) Name	Libraries
cc (Cross nmcc)	libc
f77 (Cross nm77)	libF77, libI77, libm, libc
pc (Cross nmpc)	libpas, libm, libc
m2c (Cross nm2c)	libmod2, libm, libc

INITIALIZATION CODE AND ENTRY-POINTS

Normally, the entry point of the final executable file is called *start*. The code that follows this entry-point is initialization code that prepares the run-time environment and arranges parameters to be passed to the user-written main program. The initialization object file which contains *start* is linked in by default is called *crt0.o*. The *crt0.o* file always calls main.

The assembly-symbol that starts the user main program in the C language is __main (the underscore is prepended by the C compiler) and is called __MAIN____ in Pascal, FORTRAN 77, or Modula-2 programs.

Note that the last three compilers completely ignore the user's main program name. Therefore, in C, the user-written code is called directly from *crt0.o*. In Pascal, FORTRAN 77, and Modula-2, __main is located in the respective standard library which performs additional initializations before calling the user entry-point __MAIN____.

COMPILATION ON UNIX OPERATING SYSTEMS

National Semiconductor's GNX tools (assembler, linker, etc.,) on systems relieve a user's concern about external libraries, initialization code, and entry-points. This is due to the coherency and consistency of the GNX-Version 3 compilers and their integration through the use of a common driver.

When using a GNX Version 3 compiler on a UNIX system, the user does not directly call the compiler front end, optimizer, code generator, assembler or linker. Instead, the calls are indirectly made through the driver program.

The driver program accepts a variable number of filename arguments and options and knows how to identify language-specific options. The driver also identifies the languages in which its filename arguments are written by the names of these arguments. Therefore, the driver can arrange to compile and bind the programs with the needed libraries in order to run the program successfully.

As mentioned earlier, the driver program used by C, Pascal, FORTRAN 77, and Modula-2 programmers is exactly the same program on UNIX systems. The respective driver names are cc, pc, f77, and m2c on native systems such as the SYS32/20 or SYS32/30 and nmcc, nmpc, nf77, and nm2c on cross-support systems such as VAX/VMS or a VAX running Berkley UNIX.

The driver program looks at its own name in order to determine the libraries that are bound with the program. In addition, the driver links additional libraries according to the name extensions of any of its filename arguments. For instance, cc also links libm and libpas when one of the filename arguments is a Pascal source (recognized by the .p extension).

The -v (verbose) option of the driver verbosely outputs all driver actions. With this option, the interested user can track problems that might arise (such as undefined symbols from the linker).

As mentioned in the previous section, different languages use different initialization code that resides in language-specific standard libraries. It is necessary that the correct language initialization code be linked with a mixed-language program. The driver program helps do this, but it needs to know in which language the main program is written.

To ensure that the correct initialization code is linked with a mixed-language program, the user should call the driver that corresponds to the language of the main program module within the mixed-language program.

For example, suppose there are five source modules written in five different languages (c__utils.c written in C, f__utils.f written in FORTRAN 77, p__tuils.p written in Pascal, m__utils.m written in Modula-2, and s__utils.s written in assembly), and there is a sixth module that has already been compiled separately (obj.o, an object module). Assuming there is a main program written in FORTRAN 77, the f77 driver should be used.

f77 main.f c__utils.c f__utils.f p__utils.p m__utils.m s__utils.s obj.o

If the main program is written in C, cc is used, and so on.

COMPILATION ON VMS OPERATING SYSTEMS

When using the GNX tools on VMS systems, the linking phase is separate from the compilation phase; therefore, it demands separate actions from the user.

The interested user should refer to the language tools manuals (assembler, linker, etc.) for a complete description of how to use them on VMS systems.

COMPILING A MIXED-LANGUAGE EXAMPLE

The example listed in Appendix A consists of a number of program modules written in languages different from the main program, which is written in C.

COMPILING THE EXAMPLE ON A UNIX SYSTEM

To compile the program modules on a Berkeley UNIX system, type the command:

```
nmcc c_main.c\
     c_fun.c dmod_fun.def dummy.def
     f77_fun.f\
     imod_fun.m pas_fun.p asm_fun.s
```

This assumes that all the program modules are in the same directory. If the program compiles and links successfully, the result is an executable file that, when run on a Series 32000 CPU, prints the line "Passed OK!!!".

APPENDIX A
PROGRAM MODULE LISTINGS

The different program modules are listed in this section.

c_main.c

```
/*-------------------------------------------------
 * Example of a C program which communicates with C, Pascal,
 * Fortran 77, Modula-2 and Assembly external functions, via
 * direct calls as well as via a global variable.
 * Parameter passing by reference is accomplished by passing the
 * addresses of the characters variables "a', "b', "c', "d' and "e'.
 *-------------------------------------------------*/
char str_[ ] = "Passed OK!!!\n''; /* global ('exported') string*/
main () {
   char a, b, c, d, e;
   int three = 3; /* FORTRAN must get its parameters by reference
                  *So we put this constant into a variable . . .
                  */
   if (c_func (&a, 0)  && /* in C arrays start with 0*/
    pas_func (&b,2)  && /* in Pascal they start at 1*/
    f77_func_(&c,&three)&& /*in f77, at 1*/
    mod_func (&d, 3)  && /* in Modula-2, at 0*/
    asm_func (&e, 4)) /*in assembly, at 0*/
     printf ("%c%c%c%c%c%s'', a, b, c, d, e, str_ +5);
     /*Should print "Passed OK!!!''*/
}
/* dummy initialization function for Modula-2*/
dummy ()
{
}
```

c_fun.c

```
/*
 * Declaration of the public character string 'str[]' and definition
 * of the C function 'c_func()'.
 * Note the appending of an underscore to the external symbol 'str_'
 * which is shared with FORTRAN 77.
 */
 extern char str_[];
int c_func (c_ptr, index)
char  *c_ptr;
int   index;
{
   *c_ptr = str_[index];
   return 1;
}
```

```
          f77_fun.f
C
C  The FORTRAN 77 function:
C
C  All parameters are passed by reference
C  The COMMON statement aliases the external array 'str' as 'text'
C
    LOGICAL FUNCTION f77_func(c, index)
    CHARACTER c
    INTEGER  index
    COMMON /str/text
    CHARACTER text(15)
    c = text(index)
    f77_func = .TRUE.
    RETURN
    END
          dmod_fun.def
DEFINITION MODULE mfunc_module;
  EXPORT mod_func;
  PROCEDURE mod_func(VAR c: CHAR; index: INTEGER): BOOLEAN;
END mfunc_module.
          dummy.def
 (*
  *  This definition module was written in order to 'satisfy' Modula-2
  *  strict conformance checks regarding the foreign language functions
  *  and in order to define the global character array 'str[]'.
  *  The external functions are called from the Modula-2 main program,
  *  so they must be exported from somewhere . . .
  *)
DEFINITION MODULE dummy;
 EXPORT
  str_, c_func, pas_func, f77_func_, asm_func;

  (*external function declarations*)
  PROCEDURE c_func (VAR c: CHAR; index: INTEGER): BOOLEAN;
  PROCEDURE pas_func (VAR c: CHAR; index: INTEGER): BOOLEAN;
  PROCEDURE f77_func (VAR c: CHAR; VAR index: INTEGER): BOOLEAN;
  PROCEDURE asm_func (VAR c: CHAR; index: INTEGER): BOOLEAN;
  VAR
  str_: ARRAY [0. . 14] OF CHAR;
  END dummy.
```

imod_fun.m

```
(*
 * Definition of the Modula-2 function 'mod_func ()'
 *)
IMPLEMENTATION MODULE mfunc_module;
  FROM dummy IMPORT str_;
  PROCEDURE mod_func(VAR c: CHAR; index: INTEGER): BOOLEAN;
  BEGIN
   c: = str_[index];
   RETURN (TRUE);
  END mod_func;
 END mfunc_module.
```

pas_fun.p

```
(*
 * The Pascal function 'pas_func ()'
 *)
(* 'str[] character-array declaration *)
#include 'str_pas.h'';
(* make this function visible to outsiders ('export')*)
function pas_func(var c: char; index: integer): boolean; external;

function pas_func ();
begin
  c: = str_[index];
  pas_func: = TRUE;
end;
```

str_pas.h

```
(* 'str[]' character-array declaration for Pascal*)
var
  str_: packed array [1. . 15] of char;
```

```
        asm_fun.s
#
# The 32000 Assembly Language Function 'asm_func'
#
#  The function includes an artificial use of r7, to demonstrate the
#  need to save it upon entry and restore upon exit, as opposed to
#  r0, r1 and r2; f0, f1, f2 and f3 which can be used freely without
#  saving or restoring. This is according to the Series 32000
#  standard calling convention.
#  The function return value is placed in r0, also according to the
#  standard calling convention.
#
   .globl _str_ #Import the global str[] array.
   .globl _asm_func #Export (make visible) the assembly function.
   .align 4

_asm_func:
   enter  [r7],0     #Set frame, demonstrate saving of r7
   movb  _str_+0(12(fp)),0(8(fp)) # argument_1 ← str[argument_2]
   movqd  $(1), r7      #artificial use of r7
   movd  r7, r0      #return_value ← TRUE
   exit  [r7]      #Unwind frame, restore r7
   ret  $(0)      #Return to caller
```

Portability Issues and the GNX™ Version 3 C Optimizing Compiler

National Semiconductor
Application Note 601
Series 32000 Applications

INTRODUCTION

This application note describes compiler implementation aspects which may differ between those of the GNX-Version 3 C Optimizing compiler and other compilers and which may affect code portability. Portability issues are recognized by the C standard as issues that may differ from one compiler implementation to another.

The GNX-Version 3 C Optimizing Compiler is one of a family of compatible optimizing compilers targeted to the Series 32000® architecture. The compiler fully implements the C Language as defined in *The C Programming Language* by B. Kernighan and D. Ritchie. The C Optimizing Compiler is also compatible with the UNIX® System V Compiler(pcc).

This Application Note contains three sections:

1.0 Implementation Aspects

2.0 Standard Calling Conventions

3.0 Undefined Behavior

1.0 IMPLEMENTATION ASPECTS

This section describes aspects of the implementation of the GNX-Version 3 C compiler of which one should be knowledgeable in order to write portable programs or to port programs written for compilation using other C compilers.

The topics addressed are:

1.1 Memory Representation of Data Types

1.2 External Linkage Considerations

1.3 Data Types and Conversions

1.4 Variable and Structure Memory Alignment

1.5 Functions that Return a Structure

1.6 Mixed-Language Programming

1.7 Order of Evaluation of Parameters

1.8 Order of Allocation of Memory

1.9 Register Variables

1.10 Floating-Point Arithmetic

1.1 MEMORY REPRESENTATION OF DATA TYPES

The representation of the various C types in this compiler are:

C Type	Series 32000 Data Type
int	32-Bit Double-Word
long	32-Bit Double-Word
short	16-Bit Word
char	8-Bit Byte
float	32-Bit Single-Precision Floating-Point
double	64-Bit Double-Precision Floating-Point

- The set of values stored in a **char** object is signed.
- The padding and alignment of members of structures as described in Section 1.4.
- A field of a structure can generally straddle storage unit boundaries.
- While signed bitfields are implemented, it is not recommended to use them since their implementation is slow. Bitfields are not allowed to straddle a double-word boundary.

1.2 EXTERNAL LINKAGE CONSIDERATIONS

- There is no limit to the number of characters in external names.
- Case distinctions are significant in an identifier with external linkage.

1.3 DATA TYPES AND CONVERSIONS

- A right shift of a signed integral type is arithmetic, i.e., the sign is maintained.
- When a negative floating-point number is converted to an integer, it is truncated to the nearest integer that is less than or equal to it in absolute value. The result is returned as a signed integer.
- When a double-precision entity is converted to a single-precision entity, it is converted to the nearest representation that will fit in a **float** with default rounding performed to the nearest value.
- The presence of a **float** operand in an operation not containing double-operands causes a conversion of the other operand to **float** and the use of single-precision arithmetic. If double-operands are present, conversion to double occurs.

1.4 VARIABLE AND STRUCTURE MEMORY ALIGNMENT

The alignment of entities in a program is a trade-off issue. Most Series 32000 CPUs are more efficient when dealing with entities aligned to a double-word boundary. This normally makes it necessary to have some amount of padding added to a program. This padding represents an overhead in storage space.

The GNX-Version 3 C compiler allows the user to tailor the alignment of structures/unions and their members and, independently, the alignment of other variables. Function parameters are always double-word aligned. This allows the calling of functions across modules without dealing with alignment issues.

1.4.1 Alignment of Variables

Extern, static, and **auto** variables are aligned in memory according to their size and the buswidth setting. Table I lists variable size, buswidth, and the alignment determined by these two parameters.

TABLE I. Variable Alignment

Bus Width	Variable Size (Bytes)		
	1	2	≥4
1	byte	byte	byte
2	byte	word	word
4	byte	word	double-word

Variables of size 1 are of the C type **char**, variables of size 2 are of the C type **short**, and variables of size 4 or greater are of the C types **int**, **long**, **float**, and **double** (size 8).

A buswidth setting of 1 means "align to 1 byte". Variables start on a byte boundary, in other words, there is no alignment and no padding. When allocating storage for variables, bytes are allocated sequentially with no padding between bytes.

A buswidth setting of 2 means "align to an even byte." Variables that are larger than 1 byte start on a word boundary. This means that there may be padding of single bytes.

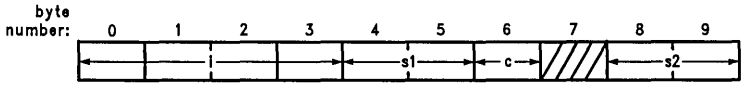
A buswidth setting of 4 means "align to a double-word boundary" (a byte whose address is divisible by four). Variables that are 2 bytes long start on a word boundary; variables that are 4 bytes or larger in size start on a double-word boundary. This means that there may be padding of up to three bytes.

Arrays are aligned as the alignment of their element type. Structures are aligned according to the alignment of the largest structure members. This is affected by the **-J (/ALIGN)** option. See "Structure/Union Alignment" and "Allocation of Bit-Fields" for more details.

Example: The arrangement of

int i; short s1; char c; short s2;

with a buswidth of 2 or 4 is



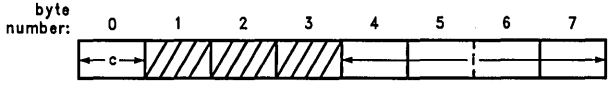
TL/EE/10345-1

Note that to align s2 to a word boundary, padding space of one byte is needed after **c**. This padding does not exist with a buswidth of 1.

Example: The arrangement of

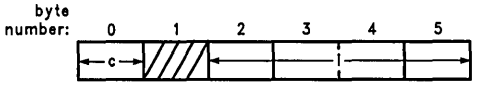
```
char c; int i;
```

with a buswidth of 4 is



TL/EE/10345-2

With a buswidth of 2, the arrangement is



TL/EE/10345-3

With a buswidth of 1, there is no padding.

It is important to note that the order in memory is the same as the declaration order only for **extern** and **static** variables. The optimizer may reorder **auto** variables in order to minimize padding space.

Fastest code is achieved by setting the default alignment to that of the data buswidth of the CPU (4 for all but the NS32008, the NS32CG16, and the NS32016). This can be accomplished by setting the BUS parameter in the target specification file, or by overwriting that file on the command line with the **-KB (/TARGET)** option.

1.4.2 Structure/Union Alignment

Structure members are aligned within the structure, relative to the beginning of the structure, in the same way that variables are aligned in memory. In order to maintain the alignment of the members relative to memory, the structure itself is aligned in memory according to the alignment of its largest members. This alignment may be controlled by putting **-J (/ALIGN)** on the command line.

In addition, the total size of a structure is such that it also ends on an alignment boundary of its largest member. This maintains the alignment of individual members in arrays of structures. This is illustrated in the **FILE struct** example at the end of this section.

For unions, there is no padding. The alignment of the union's largest members determine the alignment of the union itself.

1.4.3 Allocation of Bit-Fields

To understand the way bit-fields are handled, think of the situation where a field is fetched from memory. The number of bits fetched is determined by buswidth. For instance, if a bus is 2-bytes wide, then 2 bytes are fetched, even if only the first few bits are needed. For convenience, the number of bits fetched is called the "fetching unit".

Note that for the purpose of structure member alignment, the align switch value (1 byte, 2 bytes or 4 bytes) is taken as a "virtual buswidth," even if it is different from the actual buswidth.

A complication exists when allocating bit-fields. The complication arises from the fact that different base types for bit-fields (**char short**, and **int**) are supported. The maximum length of a bit-field is the size of its base type; therefore, there may be times when a bit-field is larger than the buswidth. When the size of the base type is larger than the buswidth, the size of the fetching unit is considered to be the base-type size.

The precise rules for determining the start of the fetching unit are quite complicated. In general, it is determined by the current position in the allocation of structure members and by the base-type of the first bit-field in a group of consecutive bit-fields.

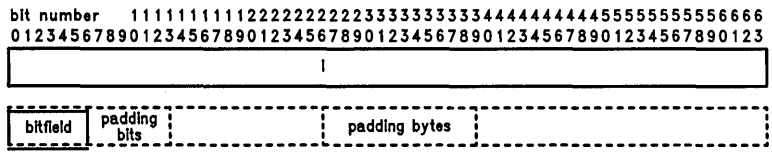
An attempt is made to pack consecutive bit-fields as much as possible, as long as the bit-fields remain in the same fetching unit. As soon as a field "spills over" into the next fetching unit, the alignment is set to the next memory unit (byte, word, or double-word, according to the align switch value and the base type of the field). A hole of padding bits remains, and the beginning of the spill-over field determines the start of a new fetching unit for following bit-fields. Using this method, bit-fields are packed as much as possible while still maintaining the alignment.

If, because of the bit-fields, the structure as a whole does not terminate on a byte boundary, padding bits are added to it to fill up to the end of the last byte it occupies. Additional padding bytes may be needed to fill to the alignment boundary of the largest structure member. This is seen in *Figure 1.* The bit-field does not quite reach the byte boundary; therefore, padding bits are added until the byte boundary is reached. Additional padding bytes are added to fill to the alignment boundary of the double-word structure member. See *Figure 1.*

Example:

```
struct A {
int i;
unsigned bitfield: 4;
} a;
```

The arrangement of **a**'s fields in memory will be:



TL/EE/10345-4

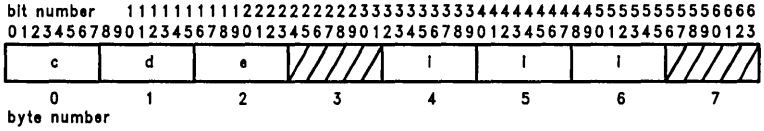
FIGURE 1. Bitfield Padding

Figure 2 is an example of the alignment on bit-fields given the different align switch settings. To summarize, the **-J (/ALIGN)** switch affects:

- the alignment and padding used for structure members and the alignment of variables of the structure type.
- the total storage alocated to a structure by determining if, and how many, padding bytes will be added after its last field.

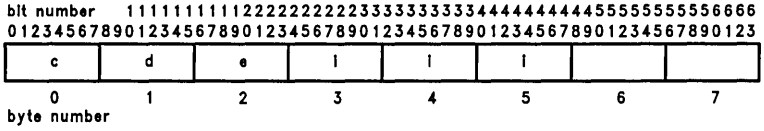
Example:
```
struct X {
    char c,d,e;
    int i: 24;
}
```

ALIGN = 4



TL/EE/10345-5

ALIGN = 2/1



TL/EE/10345-6

FIGURE 2. Alignment on Bitfields

CAUTION

The user must make sure that all parts of the program, including library routines, use the same alignment for the same structures; otherwise, problems result. The following example illustrates this point.

Suppose the example program includes <**stdio.h**>. The file <**stdio.h**> contains the following definitions.

```
extern FILE_iob [_NFILE];
typedef struct {
        int         cnt;
        unsigned char *_ptr;
        unsigned char *_base;
        char        _flag;
        char        _file;
} FILE;
```

Note that FILE has two char members at its end. If align = 4, any variable declared to be of type FILE will have two padding bytes added at its end in order to make it occupy an integral number of double-words. When align = 1 or align = 2, no padding is performed.

If a module using <stdio.h> is compiled with align = 4 and later linked with a module compiled with align = 1 or align = 2 that tries to use iob[n] when n > 0, the result will be wrong. This is because the two modules disagree on the size of the elements in the array. This situation actually does arise if a user module, compiled with align = 1 or align = 2, is linked with the default library libc, which is compiled with align = 4.

The solution to this problem is to make sure all modules are compiled using either the same alignment setting, including all include files and libraries, or a revised header file that has been made insensitive to the setting of the alignment switch. This is performed by including the necessary padding to enforce equal sizes and offsets. If the latter solution is chosen, FILE is revised to look like:

```
typedef struct {
        int         cnt;
        unsigned char *_ptr;
        unsigned char *_base;
        char        _flag;
        char        _file;
        /*padding*/  int:16;
} FILE;
```

No padding is added by the compiler, and the size of the structure is the same for all switch settings.

1.5 FUNCTIONS THAT RETURN A STRUCTURE

In the GNX-Version 3 C compiler, structure returning functions have a hidden argument which is the address of an area the size of the returned structure. This area is allocated by the caller and its address is passed as a first argument to the structure returning function. Structure returning functions are, therefore, re-entrant and interruptible.

Note: At the optimizer's discretion, small structures (less than 5 bytes) may be passed and/or returned in a register.

1.6 MIXED-LANGUAGE PROGRAMMING

Mixed-language programs are frequently used for two reasons. First, one language may be more convenient than another for certain tasks. Second, code sections already written in another language (e.g., an already existing library function) can be reused simply by calling them.

A programmer who wishes to mix several programming languages needs to be aware of subtle differences between the compilation of the various languages. An Application Note is available that describes the issues one needs to be aware of when writing mixed-language programs and compiling and linking such programs successfully.

1.7 ORDER OF EVALUATION OF PARAMETERS

The evaluation order of expressions and actual parameters in theGNX-Version 3 C compiler may differ from those of other compilers. Therefore, programs that rely on a specific order of evaluation may not run correctly when compiled. In particular, the following orders of evaluation are unspecified:

- The order in which expressions are evaluated.
- The order in which function arguments are evaluated.
- The order in which side effects take place. For instance, **a[i++] = i** may be evaluated as

```
a[i] = i;
i++
```

or as

```
t = i;
i++
a[t] = i;
```

1.8 ORDER OF ALLOCATION OF MEMORY

The order of allocation of local variables in memory is compiler-dependent. After the optimizer of the GNX-Version 3 C compiler performs register allocation, it reorders the local variables left in memory. This reordering reduces memory space requirements and minimizes displacement length. User programs that rely on any order of allocation of local variables may not run correctly.

1.9 REGISTER VARIABLES

By default, register variables, as well as other local variables, are equal candidates for register allocation. When given complete freedom, the programmer generally performs a better job of register allocation than when forced to follow the allocation. For programs which make assumptions about variables which reside in specific registers, an optimization flag (**-Ou** or **-O -Fu** on UNIX and **USER_REGISTERS** on VMS™) is available to enforce the **pcc** allocation scheme for register variables of scalar types and of type double.

1.10 FLOATING-POINT ARITHMETIC

The floating-point arithmetic conversion rules of the GNX-Version 3 C compiler differ from most other C compilers.

In an operation not containing double-operands, if one of two operands is of type **float**, the other operand is converted to type **float** and single-precision arithmetic is used. The result of the operation is of type **float**. This behavior differs from previous compilers which perform such operations in double precision.

In old C compilers, the result of float-returning functions was actually returned in double-format and placed in the F0–F1 register pair. When compiled by the GNX-Version 3 C compiler, such functions return the return value result in float format and place the result in the F0 register. Note that assembly programs that interface with float-returning functions may now incorrectly expect a double precision result.

Float parameters, however, are passed as double because the C language semantics do not require type identity between actual and formal parameters. Code is generated in the called function to convert these actual double values back to float if necessary.

Floating-point constants are of type **double**, unless they are typecast to **float** or are suffixed by the letter **f** or **F**. By preference, constants of type **float** should be used in float expressions to avoid the unnecessary casting of other operands to double precision. For example,

```
fmax+ = 17.5f;
```

is more efficient than

```
fmax+ = 17.5;
```

The following examples are of double constants and float constants.

Example:	Double Constants	Float Constants
	14.5 e6	14.5e6f
	14.5	(float) 14.5

2.0 SERIES 32000 STANDARD CALLING CONVENTIONS

The main goal of standard calling conventions is to enable the routines of one program to communicate with different modules, even when written in multiple-programming languages. The standard calling conventions support various special language features (such as the ability to pass a variable number of arguments, which is allowed in C), by using the different calling mechanisms of the Series 32000 architecture. These conventions are employed only to call externally visible routines. Calls to internal routines may employ even faster calling sequences by passing arguments in registers, for instance.

The standard Series 32000 calling conventions are used by the C compiler for calls to external routines of all languages. It is, therefore, unnecessary to use the *fortran* keyword in C programs, (if present, the keyword is ignored). However, local or internal routines (functions which in C are preceded by the static keyword) are called by more efficient calling sequences.

Basically, the calling sequence pushes arguments on top of the stack, executes a call instruction, and then pops the stack while using the fewest possible instructions to execute at the maximum speed. The following sections discuss the various aspects of the Series 32000 standard calling conventions.

2.1 CALLING CONVENTION ELEMENTS

Elements of the standard calling sequence are as follows:

2.1.1 The Argument Stack

Arguments are pushed on the stack from right to left; therefore, the leftmost argument is pushed last. Consequently, the leftmost arguments are always at the same offset from the frame pointer, regardless of how many arguments are actually passed. This allows functions with a variable number of arguments to be used.

Note: This does not imply that the actual parameters are always evaluated from right to left. Programs cannot rely on the order of parameter evaluation.

The run-time stack must be aligned to a full double-word boundary. Argument lists always use a whole number of double-words; pointer and integer values use a double-word (by extension, if necessary), floating-point values use eight bytes and are represented as **long** values; structures (records) use a multiple of double-words.

Note: Stack alignment is maintained by all GNX-Version 3 compilers through aligned allocation and de-allocation of local variables. Interrupt routines and other assembly-written interface routines are advised to maintain this double-word alignment.

The caller routine must pop the arguments off the stack upon return from the called routine.

Note: The compiler uses a more efficient organization of the stack frame if the **FIXED_FRAME (-OF)** optimization is enabled. In that case, programs should not rely on the organization of the stack frame.

2.1.2 Saving Registers

General registers R0, R1, and R2 and floating registers F0, F1, F2, and F3 are temporary or scratch registers whose values may be changed by a called routine. Also included in this list of scratch registers is the long register L1 of the NS32381 FPU. It is not necessary to save these registers on procedure entry or restore them before exit. If the other registers (R3 through R7, F4 through F7, and L3 through L7 of the NS32381) are used, their values should be saved (onto the stack or in temps) by the called routine immediately upon procedure entry and restored just before executing the return instruction. This should be performed because the caller routine may rely on the values in these registers not changing.

Note: Interrupt and trap service routines are required to save/restore all registers that they use.

2.1.3 Return Value

An integer or a pointer value that returns from a function, returns in (part of) register R0.

A long floating-point value that returns from a function, returns in register pair F0-F1. A float-returning function returns the value in register F0.

If a function returns a structure, the calling function passes an additional argument at the beginning of the argument list. This argument points to where the called function returns the structure. The called function copies the structure into the specified location during execution of the return statement. Note that functions that return structures must be correctly declared as such, even if the return value is ignored.

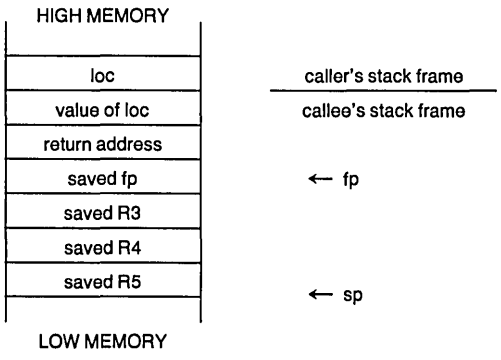
Example:

```
int iglob;
m( )
{
 int loc;
 a = if unc(loc);
}
if unc(pl)
int pl;
}
 int i, j, k;
 j = 0;
 for (i = 1; i ≤ pl; i++)
  j = j + f(i);
 return(j);
}
```

The compiler may generate the following code:

```
_m:
        enter   [ ],4           #Allocate local variable
        movd    -4(fp),tos      #Push argument
        bsr     _if unc
        adjspb  $(-4)           #Pop argument off stack
        movd    r0,_iglob       #Save return value
        exit    [ ]
        ret     $(0)
_ifunc:
        enter   [r3,r4,r5],0    #Save safe registers
        movd    8(fp),r5        #Load argument to temp register
        movqd   $(0),r4         #Initialize j
        cmpqd   $(1),r5
        bgt     .LL1
        movqd   $(1),r3         #Initialize i
.LL2:
        movd    r3,tos          #Push argument
        bsr     _f
        adjspb  $(-4)           #Pop argument off stack
        addd    r0,r4           #Add return value to j
        addqd   $(1),r3         #Increment i
        cmpd    r3,r5
        ble     .LL2
.LL1:
        movd    r4,r0           #Return value
        exit    [r3,r4,r5]      #Restore safe registers
        ret     $(0)
```

After the enter instruction is executed by ifunc( ), the stack will look like this:



3.0 UNDEFINED BEHAVIOR

In the following cases, the behavior of the GNX-Version 3 C compiler is undefined:

- The value of a floating-point or integer constant is not representable.
- An arithmetic conversion produces a result that cannot be represented in the space provided.
- A volatile object is referred to by means of a pointer to a type without the volatile attribute.
- An arithmetic operation is invalid, such as division by 0, or produces a result that cannot be represented in the space provided, such as overflow or underflow.
- A member of a union object is accessed using a member of a different type.
- An object is assigned to an overlapping object.
- The value of a register variable has been changed between a **setjmp** call and a **longjmp** call.

Using the GNX-Version 3 C Optimizing Compiler in the UNIX® Environment

National Semiconductor
Application Note 605

1.0 INTRODUCTION

To optimize the performance of systems built around National's Embedded System Processors™ and Series 32000® microprocessors, National has developed a set of advanced optimizing compilers. Four compilers are available to support the C, Pascal, FORTRAN 77, and Modula 2 languages. They are offered with Release 3.0 of the GENIX™ Native and Cross-Support (GNX™) Language Tools. By generating high-quality code specifically tailored to the Series 32000 architecture, these compilers allow Series 32000 microprocessors to achieve their full performance potential.

National's optimizing compilers use advanced optimization techniques to improve speed or save space. When code size is critical, the compilers can produce code that is more compact than code generated by other compilers. When speed is important, they can produce code that is 30%–200% faster.

Figure 1-1 shows the compilation process performed by National's optimizing compilers. When a program is compiled, the compiler performs syntactic and semantic verification of the source code and then translates it into a unique intermediate language called IR32.

Next, the IR32 code is passed to a dedicated optimizer. The optimizer performs four optimization steps to tailor the code to the processor architecture.

The first step is local optimization. During this step, the IR32 code is partitioned into basic blocks. Each basic block consists of a straight sequence of code. The only branches allowed in a basic block are at the entry or exit of the sequence. Some of the local optimizations performed include constant folding, value propagation, and the elimination of redundant assignments.

The second optimization step is flow optimization. During this step, a flow graph is constructed in which each basic block of code is represented by a node. Optimizations of the flow and elimination of dead code are performed during this step.

The third optimization step is global optimization. During this step, global code transformations are performed to speed program execution. Optimizations performed include loop-invariant code motion and the elimination of fully and partially redundant expressions.

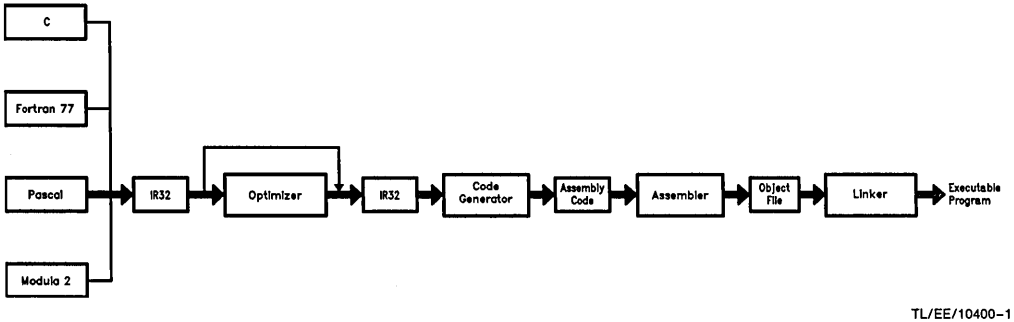
Register allocation is the fourth optimization step performed by the optimizer. During this step, variables are placed in registers instead of main memory. The use of volatile registers and the allocation of register parameters are also optimized.

After the IR32 code has been optimized by the optimizer, it is passed to the code generator. The code generator further optimizes the code by selecting optimal code sequences, performing peephole optimizations, aligning the code and data, and performing frame optimizations. It then translates the optimized IR32 code into assembly code.

Finally, an assembler generates object files from the assembly code, and a linker links the files together for execution.

This application note presents guidelines for using the GNX-Version 3 C Optimizing Compiler. However, much of the information presented here also applies to the optimizing compilers for Pascal, FORTRAN 77, and Modula 2. Topics presented here include:

- Optimization options for UNIX systems.
- UNIX command-line optimization options.
- Porting existing C programs to the GNX-Version 3 C Optimizing Compiler.
- Debugging optimized code.
- Additional techniques to improve code quality.
- Time requirements for compilation.
- Specifying a target machine.



TL/EE/10400–1

FIGURE 1-1. The Compilation Process

2.0 OPTIMIZATION OPTIONS

Table 2-1 lists all of the optimization options for the GNX-Version 3 C Optimizing Compiler. Different combinations of optimization flags can be used to tailor the optimizations for specific applications. For example, some applications must be optimized for speed, while others require smaller code size.

TABLE 2-1. Optimization Options

UNIX	Description
o	Does not invoke the optimizer phase.
c	Does not compute floating-point constant expressions at compile time.
C	Performs floating-point constant folding.
F	Uses fixed frame references, avoids use of the FP register or the ENTER/EXIT instruction.
f	Compiles for debugging: uses slower FP and TOS addressing modes.
I	Applies all optimizations to all variables (including global variables).
i	Compiles system code: assumes that all global and static memory variables and pointer dereferences are volatile.
L	Assumes use of standard run-time library.
l	Assumes that all routines have corrupting side effects.
M	Performs global code motion optimizations.
m	Does not perform global code motion optimizations.
U	Ignores user register declarations.
u	Allocates user-declared register variables in registers as done by pc.
R	Performs the register allocation pass of the optimizer.
r	Does not perform the register allocation pass of the optimizer.
S	Optimizes for speed only.
s	Does not waste space in favor of speed.
1–9	Maximal memory/swap-space available is 1 through 9 Mbytes (default: 4 Mbytes)

3.0 UNIX COMMAND-LINE OPTIONS

Specifying the -O option on the command line enables the optimizer. This results in the fastest possible code based on the default settings listed in Table 2-2. Specifying the optimizer pass is equivalent to entering:

-OCFILMRSU

In special cases, such as when compiling operating-system code, it may be necessary to change the optimization settings from their default values. This can be done by specifying optimization flags. Individual optimization flags can be specified either by using the -F option, or by simply appending them to -O. Table 3-1 suggests situations in which turning off an optimization option may be desirable.

Note that specifying the compiler debug option -g on the command line automatically turns off the optimizer's fixed-frame flag -OF, unless otherwise specified on the command line.

Also note that using the compiler target option -KB1 favors space over speed by saving alignment holes normally produced when the buswidth is the default (4 bytes).

By not specifying the -O option on the command line, the optimizer pass can be omitted. However, even when the optimizer pass is omitted, some optimizations are performed by the code generator. As a result, bypassing the optimizer is equivalent to entering:

-OocfIlmrSu

TABLE 3-1. Reasons to Turn off Optimization Options

Option	Reason for Turning off Option
-Of	To debug the program or to compile nonportable programs that assume knowledge of the runtime stack.
-Oi	To compile system programs, such as device drivers, which contain variables that change or are referenced spontaneously.
-Ol	To compile programs which reimplement standard functions, in a way which does not agree with the optimizer's assumptions (i.e., have side effects).
-Oc	To compile programs whose correct execution depends on the order in which floating-point expressions are evaluated.
-Om	To compile programs which contain huge functions, which are a drain on the system's resources and are time consuming to optimize.
-Ou	To compile programs which rely on the register allocation scheme of pcc.
-Or	To run programs that cease to work when performing register allocation.
-Os	To compile programs which must fit as tightly as possible in memory.
-Oo or use -F*flags* without giving -O	When the optimizer phase is not required and another flag needs to be turned off as well. For instance, -OoF turns fixed frame on without running the optimizer, while -Of turns off fixed frame but runs the optimizer.

4.0 PORTING EXISTING C PROGRAMS

Almost every program that runs when compiled by other C compilers, will compile and run under the GNX-Version 3 C compiler without any changes in the source code. Occasionally, however, a program may operate differently than before. Other programs may work when compiled without the optimizer, but will not work when the code is optimized. Possible causes for these problems are described in the following sections.

4.1 Undetected Program Errors

The single most common reason for a nonfunctioning program is an undetected program error. These errors become apparent when a different compiler is used or when the code is optimized. Many of these errors result from compiler-specific code in non-portable programs. The following lists some of the most common problems:

- Unitialized local variables.

The memory and register allocation algorithms of the GNX-Version 3 C Optimizing Compiler are very different from those of other compilers. As a result, a local variable may end up in a completely different place than expected. Because of this, there is no guarantee that local variables will contain zero when the program is started. Therefore, all local variables should be initialized from within the program.

- Relying on memory allocation.

If two variables are declared in a certain order there is no guarantee that they will actually be allocated in that order. Therefore, a program, which uses address calculations to proceed from one declared variable to another declared variable may not work.

- Failing to declare a function.

A char returning function will return a value in the low-order byte of R0, without affecting the other bytes. A failure to declare that function where it is used may result in an error. For instance, assuming that get_code( ) is defined to return a char, then:

```
main( ) {
        int  i;
        if ((i = get_code( )) = 17)
            do_something( );
}
```

may never execute do_something, even if get_code returns 17. This is because the whole register is compared to 17, not just the low-order byte.

A similar problem exists for functions which return **short** or **float**, or those which return a structure.

4.2 Compiling System Code

System code is distinguished from general "high-level" code by the fact that it is machine-dependent, often contains real-time aspects and interspersed **asm** statements, and is often driven by asynchronous events, such as interrupts. Examples of system code are interrupt routines, device handlers, and kernel code.

To the optimizer, ordinary-looking global variables can actually be semaphores or memory-mapped I/O that can be affected by external events not under the optimizer's control. Even so, it is still possible to optimize such code by taking some precaution and by activating some special optimization flags. Some of these issues are discussed in the following sections.

- Volatile variables.

Volatile variables are variables that may be used or changed by asynchronous events, such as I/O or interrupts. The volatile flag -Oi treats all global variables, static variables, and pointer dereferences as volatile. This means that they are not subject to any optimizations. As a result, the number and nature of memory references to them will not change.

Note: Individual identifiers can be declared as volatile by using the volatile type modifier.

The following examples demonstrate the consequences of volatile variables and pointer dereferences.

Examples: 1. x = 17; x = 18;

If x is volatile, both of the two assignments to x are executed even though the first one seems redundant.

2. x = 9;
   y = x + 1;

If x is volatile, this program segment is not optimized to y = 10.

3. *p = b + c;

If *p is volatile, then this results in

```
movd b, REG
addd c, REG
movd REG, 0(p)
```

and not

```
movd b, 0(p)
addd c, 0(p)
```

The difference stems from the fact that the second sequence, though faster, makes two references to 0(p) when the programmer may have wanted only one.

4.3 Timing Assumptions

Optimizing a program changes the timing of various constructs. In particular, delay-loops may now run faster than before.

4.4 Low-Level Interface

- Relying on register order

A program that relies on the fact that a given register variable resides in a specific register must be compiled with the user-registers flag -Ou turned on. (See Section 6.7.)

- Relying on frame structure.

A program that relies on a specific frame structure must be compiled with the fixed-frame flag -Of turned off. This includes, in particular, programs that use the standard alloca( ) function that allocates space on the user's frame.

Referring to variables on the frame of a different function (such as the caller of this function) by complex pointer arithmetic may also cease to work.

- Using **asm** statements.

The code inserted by **asm** statements may cease to work because the surrounding code produced by the GNX-Version 3 C compiler will normally differ from another compiler's code. (See Section 6.6.)

4.5 Using Non-Standard Library Routines

The GNX-Version 3 C compiler assumes by default that all the C standard mathematical library routines listed in Table 4-1 are available as a standard run-time library. These library routines have absolutely no access to global variables. Therefore, calls to these routines are specially recognized and marked as calls that do not disturb optimizations

of global variables. This is normally a safe assumption since it is unusual for a program to redefine (and thereby hide) these standard routines. In addition, the functions *abs, fabs,* and *ffabs* actually compile into in-line code and do not generate a procedure call at all.

The compiler generates a warning message whenever it compiles a program which does redefine one of these routines. In this case, the user must decide whether the redefined behavior of the routine is consistent with the assumption of the optimizer that it will not affect the optimization of global variables. If it does affect global-variable optimizations, the user has the choice of:

- renaming the redefined routine (so that calls to it are not specially recognized), or
- using the no-standard-libraries flag -O -Fl to turn off the recognition of all library routines.

TABLE 4-1. Recognized Library Routines

abs	erf	fceil	fhypot	fsinh	jn	sqrt
acos	erfc	fcos	flog	fsqrt	ldexp	tan
asin	exp	fcosh	flog10	ftan	log	tanh
atan	fabs	ferf	fmod	ftanh	log10	y0
atan2	facos	ferfc	fmodf	gamma	modf	y1
cabs	fasin	fexp	fpow	hypot	pow	yn
ceil	fatan	ffabs	frexp	j0	sin	
cos	fatan2	ffmod	fsin	j1	sinh	
cosh	fcabs	ffmodf				

4.6 Reliance on Naive Algebraic Relations

The optimizer performs floating-point constant folding. That is, it rearranges expressions to evaluate constant subexpressions at compile time. As a result, some naive algebraic expressions are folded away.

```
Example:  do {
                  a = a*2;
          }
          while ((a + 1.0) - 1.0 == a);
  is optimized to
          do {
                  a = a*2;
          }
          while (1);
  which was not the programmer's intention.
```

To maintain the program and keep the programmer's original intention, the programmer should use the nofloat-fold flag -Oc to suppress the folding optimization.

5.0 DEBUGGING OF OPTIMIZED CODE

Most of the time, the user should not need to debug an optimized program. The majority of all bugs can be found before optimization is turned on. However, there are some very rare bugs which make their appearance only when the optimizer is introduced. These bugs are difficult to find without a debugger.

The problem is that code motion optimizations and register allocation make most of the symbolic debugging information generated by the compiler obsolete. With this in mind, special care must be used when reviewing assembly code generated by the compiler. The following "rules of thumb" can be employed when using symbolic debug information together with the optimizer:

- Line number information is correct, but the code performed at the specified lines may be different from non-optimized code. This is a result of various code motion optimizations, such as moving loop invariant expressions out of loops.
- Symbolic information for global variables is normally correct, since global variables are rarely put in registers. In particular, if a global variable is not referenced within the current procedure, the value in memory is valid and the symbolic information is correct.
- Symbolic information for parameters is correct except in the following two cases:
  1. When a parameter is allocated a register and there is an assignment to that parameter, the symbolic information is incorrect.
  2. When a parameter of a local procedure is passed in a register as a result of an optimization, the symbolic information is incorrect. In this case, the symbolic information of all other paramaters is incorrect because their offset within the procedure's frame has been changed.
- Symbolic information of local variables is likely to be incorrect because most of the local variables are put in registers; the rest of the local variables are reordered into new frame locations.
- Note that if symbolic information is requested, then slightly different code is generated. This happens because the fixed-frame flag -Of is automatically disabled when the debug qualifier -g is used. Specifically, the ENTER instruction is always generated at the entry of procedures, and frame variables are referenced by FP-relative rather than SP-relative addressing mode. Without disabling this flag, symbolic debugging is almost impossible.

It is helpful to have an assembly listing of the program in question which has been compiled with the -S and the -n qualifiers. Such a listing contains comments from the optimizer regarding its actions.

6.0 ADDITIONAL GUIDELINES FOR IMPROVING CODE QUALITY

The following programming guidelines take advantage of the GNX-Version 3 C compiler optimizations to further improve the quality of compiled code.

6.1 Static Functions

It is not only good software engineering practice, but also good optimization practice to declare all functions not called from outside the file as "static". This allows the optimizer to use a more efficient internal calling sequence to call such routines. This internal calling sequence uses the BSR instruction instead of the JSR or CXP instruction and also passes parameters in registers rather than on the stack.

Note: If a program consists of a single file, and compilation and linking is indicated in one step, then all functions within that file are automatically considered as static by the compiler.

6.2 Integer Variables

Many operators, including index calculations, are defined in C to operate on integers and imply a conversion when given

non-integer operands. Therefore, to avoid frequent run-time conversions from **char** or **short** to **int,** integer variables should be defined as type **int** and not **short** or **char.** This is particularly important for integer variables that serve as array indices.

6.3 Local Variables

Since local variables have a better chance of being placed in registers, they should be used as much as possible, particularly when they are employed as loop counters or array indices.

6.4 Floating-Point Computations

In programs which do not require double-precision floating-point computations, a significant run-time improvement can be achieved by using the following guidelines:

- All functions should be defined as returning type **float,** not **double.**
- All constants should be defined to be **float** using the f suffix or cast expressions explicitly to float.
- The single-precision version of the standard floating-point routines should be used. For example, ffabs( ) should be used instead of abs( ), fsin( ) instead of sin( ), etc.

6.5 Using Pointers

6.5.1 Terminology

The following terms are used throughout this section.

- Potential definition
  A statement potentially defines a memory location if the execution of the statement may change the contents of that memory location.

  Example: A call to a function potentially defines all global variables because their values may change during the execution of that function. Imagine the following code fragment:

```
extern int *p, *q:
        •
        •
*p = 8;
        •
        •
```

  The assignment statement potentially defines the memory location *q because q may point to the same memory location as p. The location *p is defined (i.e., given a new value) by the assignment. Location *q may be changed; therefore, it has the potential definition.
- Potential use
  A statement "potentially uses" a memory location if it may reference (read from) that memory location.
- Address taken variable
  A vaiable is considered "address taken" if the address operator (&) is applied to it within the file or if the variable is a global variable that is visible by other files.
- Voltatile/nonvolatile registers
  By convention, the registers are divided into volatile registers (registers R0 through R2 and F0 through F3) and nonvolatile registers (registers R3 through R7 and F4 through F7). Volatile registers may be changed by a procedure call, whereas nonvolatile registers are guaranteed to retain their value across procedure calls. Therefore, all nonvolatile registers used within a procedure must be saved at the entry and restored at the exit of that procedure.

6.5.2 Potential Difference Assumptions

The optimizer does not keep track of the contents of pointers. Therefore, it cannot tell, for any given location in the program, where each pointer is pointing. Since a pointer can point to any memory location, the optimizer makes the following assumptions concerning pointer usage:

1. Every assignment to a pointer dereference (the location pointed to by a pointer) potentially defines all other pointer dereferences and all address-taken variables.
2. Every use of a pointer dereference (i.e., a value read through a pointer) potentially uses all other pointer dereferences and all address-taken variables. This is because any accessible memory location is potentially read.
3. Every function call potentially defines and potentially uses all pointer dereferences, all address taken-variables, and all global variables. Therefore, using pointers, the function's code may read and/or write any accessible memory location. Of course, any global variable may be used and/or changed.

When working with pointers, these assumptions should be considered. For example, using arrays is preferable to using pointers. The following example illustrates this point. Assume a is an array of char and p is a pointer to char. The two program segments perform the same function.

```
Example:  program segment 1

for (i = 0 ; i != 10 ; i++) {
        a[i] = global_var;
        a[i+1] = global_var + 1;
}

        program segment 2

for (p = &a[0] ; p != &a[10] ; P++) {
        *p = global_var;
        *(p+1) = global_var + 1;
}
```

In program segment 1, global_var can be put in a register. In program segment 2, however, p may point to global_var. The first statement (*p = global_var) potentially defines global_var; therefore, it cannot be put in a register.

6.5.3 Common Subexpressions

Another aspect of this same issue is that of common subexpressions. The optimizer normally recognizes multiple

uses of the same expression and saves that expression in a temporary variable (usually a register). This cannot be performed when worst-case assumptions are made about potential definition of expressions (as described above). Expressions that contain pointer dereferences or global variables are vulnerable. Therefore, if many uses of the same expression span across procedure calls, it is advisable to save them in local variables. Consider the example:

```
foo1(p → x);
foo2(p → x);
```

Here, the expression p → x cannot be recognized by the optimizer as a common subexpression because foo1( ) may change its value. In this case, the following hand optimization may help:

```
t = p → x;   /* t is local, therefore */
foo1(t);     /*not potentially defined by foo1( ) */
foo2(t);     /*so its value is still valid for foo2( ) */
```

The programmer can make this optimization by using the knowledge that p → x is not changed by foo1( ). The optimizer cannot do the same because it assumes the worst case.

6.6 asm Statements

The keyword **asm** is recognized to enable insertion of assembly instructions directly into the assembly file generated. The syntax for its use is:

```
asm(constant-string);
```

where constant-string is a double-quoted character string.

Extreme care should be taken if **asm** statements are used. The following guidelines should be observed:

- The optimizer is not aware of the contents of an **asm** statement. Therefore, it assumes that an **asm** statement potentially defines and potentially uses all of the variables (including local variables). This means that no common subexpressions can be recognized across an **asm** statement.
- In order to allow an **asm** statement to use a specific register (e.g., asm ("save [r0,r1,r2]");), the optimizer deallocates all the registers.
- The compiler usually generates code which differs from the code generated by other compilers. This applies particularly to allocation of local variables and parameters of static procedures.
- The code surrounding the **asm** statement may change as a result of changes in other parts of the procedure.
- An **asm** statement that contains a branch instruction or a branch target (label) may cause the optimizer to generate wrong code.

Note: For these reasons, looking at the generated assembly code is strongly recommended before and after inserting **asm** statements into a program.

6.7 Register Allocation

The C language is unique in that it allows the programmer to specify (or rather, recommend) that some variables be allocated to machine registers. The optimizer normally ignores these recommendations, since in most cases the optimizer's own register allocation algorithms are as good as or superior to the programmer's recommendations. There are several reasons for this:

- The user can use a register for one variable only. The optimizer, however, allocates a register along live ranges of variables, making it possible for several variables with non-conflicting live ranges to use the same register.
- The user can declare as a register only local variables whose addresses are not taken; whereas, the optimizer allocates global variables as well as variables whose addresses are taken (where possible).
- The user can allocate variables in safe registers only. Therefore, every register used must be saved/restored at the entry/exit of the procedure. The optimizer allocates variables that do not live across procedure calls in unsafe registers. Therefore, these registers need not be saved/restored.
- Because of code motion optimizations, the number of references of variables may be changed. Therefore, the choice of register variables may not be optimal. This is illustrated in the following example:

```
Example:  int j;
          register int i;
          i = j;
          if (i == 3 || i == 4 || i == 5)
```

In this example, undesired effects result if optimized with the user-registers flag -Ou. The reason is that j is copy-propagated and replaces all occurences of i. As a result, i occupies a register but is not used. If the ordinary register allocation of the optimizer is not invoked, or if there are no registers left, j will be placed in memory.

6.8 setjmp( )

Calls to setjmp( ) are specially recognized by the compiler. Procedures that contain calls to setjmp( ) are only partially optimized because procedure calls may end up in a call to longjmp( ). Code motion optimizations are performed only within linear code sequences (those sequences not containing branches or branch targets). Register allocation is limited to optimizer-generated temporary variables, register-declared variables, and variables whose live ranges do not contain function calls.

6.9 Optimizing for Space

The default behavior of the GNX-Version 3 C compiler is to optimize for optimal speed. However, there are several things that can be done to improve code density:

- Optimize with the no-speed-over-space flag -Os turned on.
- Squeeze the data area by using -KB1 for smaller alignment between variables.
- Squeeze all record definitions by using the -J1 switch.

7.0 COMPILATION TIME REQUIREMENTS

Using the optimizer slows down the compilation process. Therefore, it is recommended that the optimizer be used only on final production versions of a program. The amounts of resources (time and memory) vary strongly from program to program and actually depend on the size of the routines in the compiled program file. The larger a routine, the more time and memory needed to optimize it. This behavior is

more or less quadratic. That is, the optimizer needs about four times the resources to optimize a routine of 1000 lines than to optimize a routine of 500 lines.

If time or memory requirements are unacceptable and routines cannot be reduced to a reasonable size of about 500 lines, it is possible to turn off some optimizations using the no-code-motion -Om and/or the no-register-allocation -Or flags.

On UNIX host systems, an optimization flag is available to set an upper limit on the memory requirements of the optimizer to a certain number of megabytes. This can be useful on host systems with a limited swap-space configuration. If necessary, the optimizer then skips certain optimizations on huge routines only, in order to stay under the chosen limit. In such cases, an appropriate message is given. This flag is only necessary when compiling modules with extremely large procedures (over 500 lines in a single procedure), a case when the optimizer may need a larger swap space than the one currently available. For example, the option:

-O2

limits the optimizer to 2 megabytes of swap space.

An alternate method for setting an upper limit on memory requirements is to use the environment variable AVAIL_SWAP. This sets the maximum swap space requirement of the optimizer in megabyte units. This environment variable should be set to the number of megabytes to be used. The user can choose from 1 Mbyte to 16 Mbytes. If the user's choice is outside of these parameters, the default value of 4 Mbytes is chosen. For example,

```
setenv AVAIL_SWAP 2
```

makes 2 Mbytes of swap space the default. This can be overridden using the -0 number option previously described.

8.0 TARGET MACHINE SPECIFICATION

The GNX-Version 3 C Optimizing Compiler provides a way to tune the code for a specific target machine by specifying its CPU, FPU, and buswidth. The values for the CPU and FPU can either be the complete device name (e.g., NS32332 or NS32081) or the last three digits of the device name (e.g., 332 or 081). The buswidth is specified in bytes. This tuning is performed by specifying compiler target option -K on the command line. Table 8-1 lists the flags and the possible settings.

Example: The following example specifies an NS32332 CPU, an NS32081 FPU, and a buswidth of 4 bytes.

```
cc -KC332 -KF081 -KB4 temp.c
```

or for cross-support,

```
nmcc -KC332 -KF081 -KB4 temp.c
```

TABLE 8-1. Target Selection Parameters

CPU (C)	FPU (F)	Buswidth (B)
[NS32]008	[NS32]081	1
[NS32]016	[NS32]381	2
[NS32]cg16	[NS32]580	4
[NS32]032		
[NS32]332		
[NS32]532		

Using the GNX-Version 3 C Optimizing Compiler in the VMS Environment

National Semiconductor
Application Note 606

1.0 INTRODUCTION

To optimize the performance of systems built around National's Embedded System Processor™ and Series 32000® microprocessors, National has developed a set of advanced optimizing compilers. Four compilers are available to support the C, Pascal, FORTRAN 77, and Modula 2 high-level languages. They are offered with Release 3.0 of the GENIX™ Native and Cross-Support (GNX™) Language Tools. By generating high-quality code specifically tailored to the Series 32000 architecture, these compilers allow Series 32000 microprocessors to achieve their full performance potential.

National's optimizing compilers use advanced optimization techniques to improve speed or save space. When code size is critical, the compilers can produce code that is more compact than code generated by other compilers. When speed is important, they can produce code that is 30%–200% faster.

Figure 1-1 shows the compilation process performed by National's optimizing compilers. When a program is compiled, the compiler performs syntactic and semantic verification of the source code and then translates it into a unique intermediate language called IR32.

Next, the IR32 code is passed to a dedicated optimizer. The optimizer performs four optimization steps to tailor the code to the processor architecture.

The first step is local optimization. During this step, the IR32 code is partitioned into basic blocks. Each basic block consists of a straight sequence of code. The only branches allowed in a basic block are at the entry or exit of the sequence. Some of the local optimizations performed include constant folding, value propagation, and the elimination of redundant assignments.

The second optimization step is flow optimization. During this step, a flow graph is constructed in which each basic block of code is represented by a node. Optimizations of the flow and elimination of dead code are performed during this step.

The third optimization step is global optimization. During this step, global code transformations are performed to speed program execution. Optimizations performed include loop-invariant code motion and the elimination of fully and partially redundant expressions.

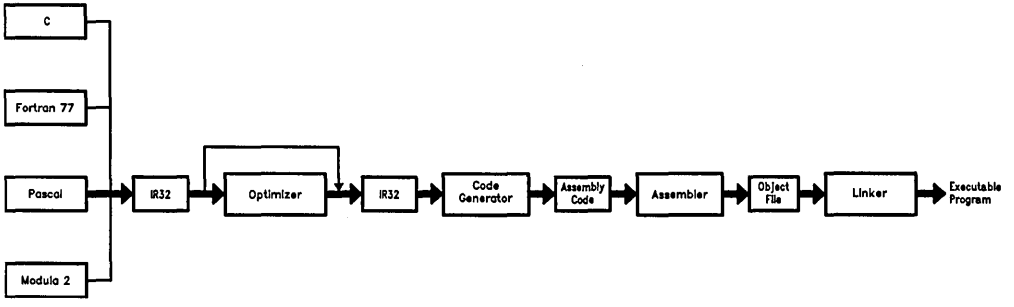
Register allocation is the fourth optimization step performed by the optimizer. During this step, variables are placed in registers instead of main memory. The use of volatile registers and the allocation of register parameters are also optimized.

After the IR32 code has been optimized by the optimizer, it is passed to the code generator. The code generator further optimizes the code by selecting optimal code sequences, performing peephole optimizations, aligning the code and data, and performing frame optimizations. It then translates the optimized IR32 code into assembly code.

Finally, an assembler generates object files from the assembly code, and a linker links the files together for execution.

This application note presents guidelines for using the GNX-Version 3 C Optimizing Compiler. However, much of the information presented here also applies to the optimizing compilers for Pascal, FORTRAN 77, and Modula 2. Topics presented here include:

- Optimization options for VMS systems.
- VMS command-line optimization options.
- Porting existing C programs to the GNX-Version 3 C Optimizing Compiler.
- Debugging optimized code.
- Additional techniques to improve code quality.
- Time requirements for compilation.
- Specifying a target machine.



TL/EE/10401–1

FIGURE 1-1. The Compilation Process

2.0 OPTIMIZATION OPTIONS

Table 2-1 lists all of the optimization options for the GNX-Version 3 C Optimizing Compiler. Different combinations of optimization flags can be used to tailor the optimizations for specific applications. For example, some applications must be optimized for speed, while others require smaller code size.

3.0 VMS COMMAND-LINE OPTIONS

The fastest possible code is generated by specifying /OPTIMIZE on the command line. This is equivalent to entering:

/OPTIMIZE = (FIXED_FRAME, CODE_MOTION,
REGISTER_ALLOCATION, FLOAT_FOLD,
SPEED_OVER_SPACE, NOVOLATILE,
STANDARD_LIBRARIES, NOUSER_REGISTERS)

In special cases, such as when compiling operating-system code, it may be necessary to change some of the optimization flags from their default settings. Table 3-1 suggests situations in which turning off an optimization option may be desirable.

Note that specifying the compiler debug option (/DEBUG) on the command line automatically turns off the optimizer fixed-frame option (/FIXED_FRAME) unless otherwise specified by the user.

Also note that using the compiler option /TARGET = (BUSWIDTH=1) favors space over speed by saving alignment holes normally produced when the buswidth is the default (n = 4).

Even when the optimizer pass is omitted, some optimizations are performed by the code generator. Therefore, specifying /NOOPTIMIZE (the default for this qualifier) is equivalent to entering:

/OPTIMIZE=(NOOPT, NOFIXED_FRAME,
NOCODE_MOTION, NOREGISTER_ALLOCATION,
NOFLOAT_FOLD, SPEED_OVER_SPACE,
NOVOLATILE, NOSTANDARD_LIBRARIES,
USER_REGISTERS)

4.0 PORTING EXISTING C PROGRAMS

Almost every program that runs when compiled by other C compilers, will compile and run under the GNX-Version 3 C compiler without any changes in the source code. Occasionally, however, a program may operate differently than

TABLE 2-1. Optimization Options

VMS	Description
NOOPT	does not invoke the optimizer phase.
NOFLOAT_FOLD	does not compute floating-point constant expressions at compile time.
FLOAT_FOLD	performs floating-point constant folding.
FIXED_FRAME	uses fixed frame references, avoids use of the FP register or the ENTER/EXIT instruction.
NOFIXED_FRAME	compiles for debugging: uses slower FP and TOS addressing modes.
NOVOLATILE	applies all optimizations to all variables (including global variables).
VOLATILE	compiles system code: assumes that all global and static memory variables and pointer dereferences are volatile.
STANDARD_LIBRARIES	assumes use of standard run-time library.
NO STANDARD_LIBRARIES	assumes that all routines have corrupting side effects.
CODE_MOTION	performs global code motion optimizations.
NOCODE_MOTION	does not perform global code motion optimizations.
NOUSER_REGISTERS	ignores user register declarations.
USER_REGISTERS	allocates user-declared register variables in registers as done by pc.
REGISTER_ALLOCATION	performs the register allocation pass of the optimizer.
NOREGISTER_ALLOCATION	does not perform the register allocation pass of the optimizer.
SPEED_OVER_SPACE	optimizes for speed only.
NOSPEED_OVER_SPACE	does not waste space in favor of speed.

TABLE 3-1. Reasons to Turn Off Optimization Options

VMS	Description
NOFIXED_FRAME	to debug the program or to compile non-portable programs that assume knowledge of the run-time stack.
VOLATILE	to compile system programs, such as device drivers, which contain variables that change or are referenced spontaneously.
NO_STANDARD_LIBRARIES	to compile programs which reimplement standard functions, in a way which does not agree with the optimizers assumptions (*i.e.*, have side effects).
NOFLOAT_FOLD	to compile programs whose correct execution depends on the order in which floating-point expressions are evaluated.
NOCODE_MOTION	to compile programs which contain huge functions, which are a drain on the system's resources and are time consuming to optimize.
USER_REGISTERS	to compile programs which rely on the register allocation scheme of pcc.
NOREGISTER_ALLOCATION	to run programs that cease to work when performing register allocation.
NOSPEED_OVER_SPACE	to compile programs which must fit as tightly as possible in memory.
NOOPT	when the optimizer phase is not required and another flag needs to be turned off as well.

before. Other programs may work when compiled without the optimizer, but will not work when the code is optimized. Possible causes for these problems are described in the following sections.

4.1 Undetected Program Errors

The single most common reason for a nonfunctioning program is an undetected program error. These errors become apparent when a different compiler is used or when the code is optimized. Many of these errors result from compiler-specific code in non-portable programs. The following lists some of the most common problems:

- Uninitialized local variables.

The memory and register allocation algorithms of the GNX-Version 3 C Optimizing Compiler are very different from those of other compilers. As a result, a local variable may end up in a completely different place than expected. Because of this, there is no guarantee that local variables will contain zero when the program is started. Therefore, all local variables should be initialized from within the program.

- Relying on memory allocation.

If two variables are declared in a certain order there is no guarantee that they will actually be allocated in that order. Therefore, a program, which uses address calculations to proceed from one declared variable to another declared variable may not work.

- Failing to declare a function.

A char returning function will return a value in the low-order byte of R0, without affecting the other bytes. A failure to declare that function where it is used may result in an error. For instance, assuming that get_code ( ) is defined to return a char, then:

```
main( ) {
        int  i;
        if ((i = get_code( )) = 17)
            do_something( );
}
```

may never execute do_something, even if get_code returns 17. This is because the whole register is compared to 17, not just the low-order byte.

A similar problem exists for functions which return **short** or **float**, or those which return a structure.

4.2 Compiling System Code

System code is distinguished from general "high-level" code by the fact that it is machine-dependent, often contains real-time aspects and interspersed **asm** statements, and is often driven by asynchronous events, such as interrupts. Examples of system code are interrupt routines, device handlers, and kernel code.

To the optimizer, ordinary-looking global variables can actually be semaphores or memory-mapped I/O that can be affected by external events not under the optimizer's control. Even so, it is still possible to optimize such code by taking some precaution and by activating some special optimization flags. Some of these issues are discussed in the following sections.

- Volatile variables

Volatile variables are variables that may be used or changed by asynchronous events, such as I/O or interrupts. The /VOLATILE flag treats all global variables, static variables, and pointer dereferences as volatile. This means that they are not subject to any optimizations. As a result, the number and nature of memory references to them will not change. (Note: Individual identifiers can be declared as volatile by using the volatile type modifier.) The following examples demonstrate the consequences of volatile variables and pointer dereferences.

Examples: 1. x = 17; x = 18;

If x is volatile, both of the two assignments to x are executed even though the first one seems redundant.

2. x = 9;
   y = x + 1;

If x is volatile, this program segment is not optimized to y = 10.

3. *p = b + c;

If *p is volatile, then this results in

```
movd  b, REG
addd  c, REG
movd  REG, 0(p)
```

and not

```
movd  b, 0(p)
addd  c, 0(p)
```

The difference stems from the fact that the second sequence, though faster, makes two references to 0(p) when the programmer may have wanted only one.

4.3 Timing Assumptions

Optimizing a program changes the timing of various constructs. In particular, delay-loops may now run faster than before.

4.4 Low-Level Interface

- Relying on register order

A program that relies on the fact that a given register variable resides in a specific register must be compiled with the /USER_REGISTERS flag turned on. (See section 6.7.)

- Relying on frame structure

A program that relies on a specific frame structure must be compiled with the /FIXED_FRAME flag turned off. This includes, in particular, programs that use the standard alloca( ) function that allocates space on the user's frame.

Referring to variables on the frame of a different function (such as the caller of this function) by complex pointer arithmetic may also cease to work.

- Using asm statements

The code inserted by asm statements may cease to work because the surrounding code produced by the GNX-Version 3 C compiler will normally differ from another compiler's code. (See section 6.6.)

4.5 Using Non-Standard Library Routines

The GNX-Version 3 C compiler assumes by default that all the C standard mathematical library routines listed in Table 4-1 are available as a standard run-time library. These library routines have absolutely no access to global variables. Therefore, calls to these routines are specially recognized and marked as calls that do not disturb optimizations of global variables. This is normally a safe assumption since it is unusual for a program to redefine (and thereby hide) these standard routines. In addition, the functions *abs*, *fabs*, and *ffabs* actually compile into in-line code and do not generate a procedure call at all.

The compiler generates a warning message whenever it compiles a program which does redefine one of these routines. In this case, the user must decide whether the redefined behavior of the routine is consistent with the assumption of the optimizer that it will not affect the optimization of global variables. If it does affect global-variable optimizations, the user has the choice of:

- renaming the redefined routine (so that calls to it are not specially recognized), or
- using the /NOSTANDARD_LIBRARY flag to turn off the recognition of all library routines.

TABLE 4-1. Recognized Library Routines

abs	erf	fceil	fhypot	fsinh	jn	sqrt
acos	erfc	fcos	flog	fsqrt	ldexp	tan
asin	exp	fcosh	flog10	ftan	log	tanh
atan	fabs	ferf	fmod	ftanh	log10	y0
atan2	facos	ferfc	fmodf	gamma	modf	y1
cabs	fasin	fexp	fpow	hypot	pow	yn
ceil	fatan	ffabs	frexp	j0	sin	
cos	fatan2	ffmod	fsin	j1	sinh	
cosh	fcabs	ffmodf				

4.6 Reliance on Naive Algebraic Relations

The optimizer performs floating-point constant folding. That is, it rearranges expressions to evaluate constant subexpressions at compile time. As a result, some naive algebraic expressions are folded away.

```
Example:  do {
                    a = a*2;
          }
          while ((a + 1.0) - 1.0 = a);
  is optimized to
          do {
                    a = a*2;
          }
          while (1);
  which was not the programmer's intention.
```

To maintain the program and keep the programmer's original intention, the programmer should use the /NOFLOAT_FOLD flag to suppress the folding optimization.

5.0 DEBUGGING OF OPTIMIZED CODE

Most of the time, the user should not need to debug an optimized program. The majority of all bugs can be found before optimization is turned on. However, there are some very rare bugs which make their appearance only when the optimizer is introduced. These bugs are difficult to find without a debugger.

The problem is that code motion optimizations and register allocation make most of the symbolic debugging information generated by the compiler obsolete. With this in mind, special care must be used when reviewing assembly code generated by the compiler. The following "rules of thumb" can be employed when using symbolic debug information together with the optimizer:

- Line number information is correct, but the code performed at the specified lines may be different from non-optimized code. This is a result of various code motion optimizations, such as moving loop invariant expressions out of loops.
- Symbolic information for global variables is normally correct, since global variables are rarely put in registers. In particular, if a global variable is not referenced within the current procedure, the value in memory is valid and the symbolic information is correct.

- Symbolic information for parameters is correct except in the following two cases:
  1. When a parameter is allocated a register and there is an assignment to that parameter, the symbolic information is incorrect.
  2. When a parameter of a local procedure is passed in a register as a result of an optimization, the symbolic information is incorrect. In this case, the symbolic information of all other parameters is incorrect because their offset within the procedure's frame has been changed.
- Symbolic information of local variables is likely to be incorrect because most of the local variables are put in registers; the rest of the local variables are reordered into new frame locations.
- Note that if symbolic information is requested, then slightly different code is generated. This happens because the /FIXED_FRAME optimizing flag is automatically disabled when the /DEBUG qualifier is used. Specifically, the ENTER instruction is always generated at the entry of procedures, and frame variables are referenced by FP-relative rather than SP-relative addressing mode. Without disabling this flag, symbolic debugging is almost impossible.

It is helpful to have an assembly listing of the program in question which has been compiled with the /ASM and the /ANNOTATE qualifiers. Such a listing contains comments from the optimizer regarding its actions.

6.0 ADDITIONAL GUIDELINES FOR IMPROVING CODE QUALITY

The following programming guidelines take advantage of the GNX-Version 3 C compiler optimizations to further improve the quality of compiled code.

6.1 Static Functions

It is not only good software engineering practice, but also good optimization practice to declare all functions not called from outside the file as "static." This allows the optimizer to use a more efficient internal calling sequence to call such routines. This internal calling sequence uses the JSR instruction instead of the HSR or CXP instruction and also passes parameters in registers rather than on the stack.

Note: If a program consists of a single file, and compilation and linking is indicated in one step, then all functions within that file are automatically considered as static by the compiler.

6.2 Integer Variables

Many operators, including index calculations, are defined in C to operate on integers and imply a conversion when given non-integer operands. Therefore, to avoid frequent run-time conversions from **char** or **short** to **int**, integer variables should be defined as type int and not short or char. This is particularly important for integer variables that serve as array indices.

6.3 Local Variables

Since local variables have a better chance of being placed in registers, they should be used as much as possible, particularly when they are employed as loop counters or array indices.

6.4 Floating-Point Computations

In programs which do not require double-precision floating-point computations, a significant run-time improvement can be achieved by using the following guidelines:

- All functions should be defined as returning type, **float** not **double**.
- All constants should be defined to be **float** using the f suffix or cast expressions explicitly to **float**.
- The single-precision version of the standard floating-point routines should be used. For example, ffabs( ) should be used instead of abs( ), fsin( ) instead of sin( ), etc.

6.5 Using Pointers

6.5.1 Terminology

The following terms are used throughout this section.

- Potential definition

A statement potentially defines a memory location if the execution of the statement may change the contents of that memory location.

Example: A call to a function potentially defines all global variables because their values may change during the execution of that function. Imagine the following code fragment:

```
extern int *p, *q;
:
*p = 8;
:
```

The assignment statement potentially defines the memory location *q because q may point to the same memory location as p. The location *p is defined (i.e., given a new value) by the assignment. Location *q may be changed; therefore, it has the potential definition.

- Potential use.

A statement "potentially uses" a memory location if it may reference (read from) that memory location.

- Address taken variable.

A variable is considered "address taken" if the address operator (&) is applied to it within the file or if the variable is a global variable that is visible by other files.

- Volatile/nonvolatile registers.

By convention, the registers are divided into volatile registers (registers R0 through R2 and F0 through F3) and nonvolatile registers (registers R3 through R7 and F4 through F7). Volatile registers may be changed by a procedure call, whereas nonvolatile registers are guaranteed to retain their value across procedure calls. Therefore, all nonvolatile registers used within a procedure must be saved at the entry and restored at the exit of that procedure.

6.5.2 Potential Difference Assumptions

The optimizer does not keep track of the contents of pointers. Therefore, it cannot tell, for any given location in the program, where each pointer is pointing. Since a pointer can point to any memory location, the optimizer makes the following assumptions concerning pointer usage:

1. Every assignment to a pointer dereference (the location pointed to by a pointer) potentially defines all other pointer dereferences and all address-taken variables.
2. Every use of a pointer dereference (i.e., a value read through a pointer) potentially uses all other pointer dereferences and all address-taken variables. This is because any accessible memory location is potentially read.
3. Every function call potentially defines and potentially uses all pointer dereferences, all address taken-variables, and all global variables. Therefore, using pointers, the function's code may read and/or write any accessible memory location. Of course, any global variable may be used and/or changed.

When working with pointers, these assumptions should be considered. For example, using arrays is preferable to using pointers. The following example illustrates this point. Assume a is an array of char and p is a pointer to char. The two program segments perform the same function.

```
Example:  program segment 1
          for (i = 0; i != 10; i++){
               a[i] = global_var;
               a[i+1] = global_var + 1;
          }
          program segment 2
          for (p = &a[0]; p != &a[10]; p++){
               *p = global_var;
               *(p+1) = global_var + 1;
          }
```

In program segment 1, global_var can be put in a register. In program segment 2, however, p may point to global_var. The first statement (*p = global_var) potentially defines global_var; therefore, it cannot be put in a register.

6.5.3 Common Subexpressions

Another aspect of this same issue is that of common subexpressions. The optimizer normally recognizes multiple uses of the same expression and saves that expression in a temporary variable (usually a register). This cannot be performed when worst-case assumptions are made about potential definition of expressions (as described above). Expressions that contain pointer dereferences or global variables are vulnerable. Therefore, if many uses of the same expression span across procedure calls, it is advisable to save them in local variables. Consider the example:

```
foo1(p → x);
foo2(p → x);
```

Here, the expression p → x cannot be recognized by the optimizer as a common subexpression because foo1( ) may change its value. In this case, the following hand optimization may help:

```
t = p → x; /* t is local, therefore */
foo1(t);       /* not potentially defined by foo1( ) */
foo2(t);       /* so its value is still valid for foo2( ) */
```

The programmer can make this optimization by using the knowledge that p → x is not changed by foo1( ). The optimizer cannot do the same because it assumes the worst case.

6.6 asm Statements

The keyword **asm** is recognized to enable insertion of assembly instructions directly into the assembly file generated. The syntax for its use is:

asm (constant-string);

where constant-string is a double-quoted character string.

Extreme care should be taken if **asm** statements are used. The following guidelines should be observed:

- The optimizer is not aware of the contents of an **asm** statement. Therefore, it assumes that an **asm** statement potentially defines and potentially uses all of the variables (including local variables). This means that no common subexpressions can be recognized across an **asm** statement.
- In order to allow an **asm** statement to use a specific register (e.g., **asm** ("save [r0, r1, r2]");), the optimizer de-allocates all the registers.
- The compiler usually generates code which differs from the code generated by other compilers. This applies particularly to allocation of local variables and parameters of static procedures.
- The code surrounding the **asm** statement may change as a result of changes in other parts of the procedure.
- An **asm** statement that contains a branch instruction or a branch target (label) may cause the optimizer to generate wrong code.

Note: For these reasons, looking at the generated assembly code is strongly recommended before and after inserting **asm** statements into a program.

6.7 Register Allocation

The C language is unique in that it allows the programmer to specify (or rather, recommend) that some variables be allocated to machine registers. The optimizer normally ignores these recommendations, since in most cases the optimizer's own register allocation algorithms are as good as or superior to the programmer's recommendations. There are several reasons for this:

- The user can use a register for one variable only. The optimizer, however, allocates a register along live ranges of variables, making it possible for several variables with non-conflicting live ranges to use the same register.
- The user can declare as a register only local variables whose addresses are not taken; whereas, the optimizer allocates global variables as well as variables whose addresses are taken (where possible).
- The user can allocate variables in safe registers only. Therefore, every register used has to be saved/restored at the entry/exit of the procedure. The optimizer allocates variables that do not live across procedure calls in unsafe registers. Therefore, these registers need not be saved/restored.
- Because of code motion optimizations, the number of references of variables may be changed. Therefore, the choice of register variables may not be optimal. This is illustrated in the following example:

```
Example:  int j;
          register int i;
          i = j;
          if (i == 3 || i == 4 || i == 5)
```

In this example, undesired effects result if optimized with the /USER_REGISTERS flag. The reason is that j is copy-propagated and replaces all occurrences of i. As a result, i occupies a register but is not used. If the ordinary register allocation of the optimizer is not invoked, or if there are no registers left, j will be placed in memory.

6.8 setjmp( )

Calls to setjmp( ) are specially recognized by the compiler. Procedures that contain calls to setjmp( ) are only partially optimized because procedure calls may end up in a call to longjmp( ). Code motion optimizations are performed only within linear code sequences (those sequences not containing branches or branch targets). Register allocation is limited to optimizer-generated temporary variables, register-declared variables, and variables whose live ranges do not contain function calls.

6.9 Optimizing for Space

The default behavior of the GNX-Version 3 C compiler optimizes for optimal speed. However, there are several things that can be done to improve code density:

- Optimize with the /NOSPEED_OVER_SPACE turned on.
- Squeeze the data area by using /TARGET = (BUS = 1) for smaller alignment between variables.
- Squeeze all record definitions by using the /ALIGN = 1 switch.

7.0 COMPILATION TIME REQUIREMENTS

Using the optimizer slows down the compilation process. Therefore, it is recommended that the optimizer be used only on final production versions of a program. The amounts of resources (time and memory) vary strongly from program to program and actually depend on the size of the routines in the compiled program file. The larger a routine, the more time and memory needed to optimize it. This behavior is more or less quadratic. That is, the optimizer needs about four times the resources to optimize a routine of 1000 lines than to optimize a routine of 500 lines.

If time or memory requirements are unacceptable and routines cannot be reduced to a reasonable size of about 500 lines, it is possible to turn off some optimizations using the /NOCODE_MOTION and/or the /NOREGISTER_ALLOCATION flags.

8.0 TARGET MACHINE SPECIFICATION

The GNX-Version 3 C Optimizing Compiler provides a way to tune the code for a specific target machine by specifying its CPU, FPU, and buswidth. The values for the CPU and FPU can either be the complete device name (e.g., NS32332 or NS32081) or the last three digits of the device name (e.g., 332 or 081). The buswidth is specified in bytes. This tuning is performed by specifying /TARGET on the command line. Table 8-1 lists the flags and the possible settings.

Example: The following example specifies an NS32332 CPU, an NS32081 FPU, and a buswidth of 4 bytes.

NMCC /TARGET = (CPU = 332, FPU = 081, BUS = 4) TEMP.C

TABLE 8-1. Target Selection Parameters

CPU (C)	FPU (F)	Buswidth (B)
[NS32]008	[NS32]081	1
[NS32]016	[NS32]381	2
[NS32]cg16	[NS32]580	4
[NS32]032		
[NS32]332		
[NS32]532		

Section 7

NSC800

Section 7 Contents

National Semiconductor

microCMOS

NSC800™ High-Performance Low-Power CMOS Microprocessor

General Description

The NSC800 is an 8-bit CMOS microprocessor that functions as the central processing unit (CPU) in National Semiconductor's NSC800 microcomputer family. National's microCMOS technology used to fabricate this device provides system designers with performance equivalent to comparable NMOS products, but with the low power advantage of CMOS. Some of the many system functions incorporated on the device, are vectored priority interrupts, refresh control, power-save feature and interrupt acknowledge. The NSC800 is available in dual-in-line and surface mounted chip carrier packages.

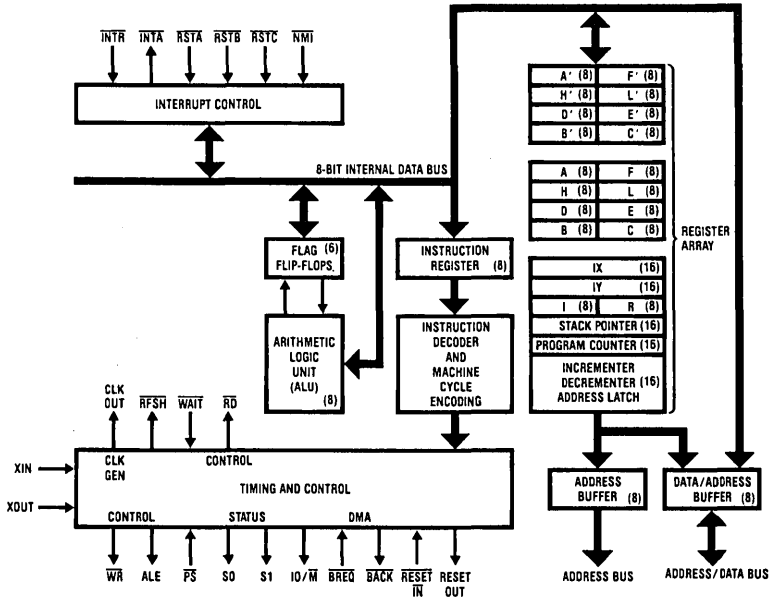
The system designer can choose not only from the dedicated CMOS peripherals that allow direct interfacing to the NSC800 but from the full line of National's CMOS products to allow a low-power system solution. The dedicated peripherals include NSC810A RAM I/O Timer, NSC858 UART, and NSC831 I/O.

All devices are available in commercial, industrial and military temperature ranges along with two added reliability flows. The first is an extended burn in test and the second is the military class C screening in accordance with Method 5004 of MIL-STD-883.

Features

- Fully compatible with Z80® instruction set:
  Powerful set of 158 instructions
  10 addressing modes
  22 internal registers
- Low power: 50 mW at 5V V_{CC}
- Unique power-save feature
- Multiplexed bus structure
- Schmitt trigger input on reset
- On-chip bus controller and clock generator
- Variable power supply 2.4V−6.0V
- On-chip 8-bit dynamic RAM refresh circuitry
- Speed: 1.0 μs instruction cycle at 4.0 MHz
  NSC800-4 4.0 MHz
  NSC800-35 3.5 MHz
  NSC800-3 2.5 MHz
  NSC800-1 1.0 MHz
- Capable of addressing 64k bytes of memory and 256 I/O devices
- Five interrupt request lines on-chip

Block Diagram



TL/C/5171–73

Table of Contents

1.0 Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.3V to V_{CC} +0.3V
Maximum V_{CC}	7V
Power Dissipation	1W
Lead Temp. (Soldering, 10 seconds)	300°C

2.0 Operating Conditions

NSC800-1	→ T_A = 0°C to +70°C
	T_A = −40°C to +85°C
NSC800-3	→ T_A = 0°C to +70°C
	T_A = −40°C to +85°C
	T_A = −55°C to +125°C
NSC800-35/883C	→ T_A = −55°C to +125°C
NSC800-4	→ T_A = 0°C to +70°C
	T_A = −40°C to +85°C
NSC800-4MIL	→ T_A = −55°C to +90°C

3.0 DC Electrical Characteristics V_{CC} = 5V ±10%, GND = 0V, unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Logical 1 Input Voltage		0.8 V_{CC}		V_{CC}	V
V_{IL}	Logical 0 Input Voltage		0		0.2 V_{CC}	V
V_{HY}	Hysteresis at RESET IN input	V_{CC} = 5V	0.25	0.5		V
V_{OH1}	Logical 1 Output Voltage	I_{OUT} = −1.0 mA	2.4			V
V_{OH2}	Logical 1 Output Voltage	I_{OUT} = −10 μA	V_{CC} −0.5			V
V_{OL1}	Logical 0 Output Voltage	I_{OUT} = 2 mA	0		0.4	V
V_{OL2}	Logical 0 Output Voltage	I_{OUT} = 10 μA	0		0.1	V
I_{IL}	Input Leakage Current	0 ≤ V_{IN} ≤ V_{CC}	−10.0		10.0	μA
I_{OL}	Output Leakage Current	0 ≤ V_{IN} ≤ V_{CC}	−10.0		10.0	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, $f_{(XIN)}$ = 2 MHz, T_A = 25°C		8	11	mA
I_{CC}	Active Supply Current	I_{OUT} = 0, $f_{(XIN)}$ = 5 MHz, T_A = 25°C		10	15	mA
I_{CC}	Active Supply Current	I_{OUT} = 0, $f_{(XIN)}$ = 7 MHz, T_A = 25°C		15	21	mA
I_{CC}	Active Supply Current	I_{OUT} = 0, $f_{(XIN)}$ = 8 MHz, T_A = 25°C		15	21	mA
I_Q	Quiescent Current	I_{OUT} = 0, $\overline{PS}$ = 0, V_{IN} = 0 or V_{IN} = V_{CC} $f_{(XIN)}$ = 0 MHz, T_A = 25°C, X_{IN} = 0, CLK = 1		2	5	mA
I_{PS}	Power-Save Current	I_{OUT} = 0, $\overline{PS}$ = 0, V_{IN} = 0 or V_{IN} = V_{CC} $f_{(XIN)}$ = 5.0 MHz , T_A = 25°		5	7	mA
C_{IN}	Input Capacitance			6	10	pF
C_{OUT}	Output Capacitance			8	12	pF
V_{CC}	Power Supply Voltage	(Note 2)	2.4	5	6	V

Note 1: Absolute Maximum Ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended and should be limited to those conditions specified under DC Electrical Characteristics.

Note 2: CPU operation at lower voltages will reduce the maximum operating speed. Operation at voltages other than 5V ±10% is guaranteed by design, not tested.

4.0 AC Electrical Characteristics V_{CC} = 5V ±10%, GND = 0V, unless otherwise specified

Symbol	Parameter	NSC800-1		NSC800-3		NSC800-35		NSC800-4		Units	Notes
		Min	Max	Min	Max	Min	Max	Min	Max		
t_X	Period at XIN and XOUT Pins	500	3333	200	3333	142	3333	125	3333	ns	
T	Period at Clock Output ($=2\,t_X$)	1000	6667	400	6667	284	6667	250	6667	ns	
t_R	Clock Rise Time		110		110		90		80	ns	Measured from 10%–90% of signal
t_F	Clock Fall Time		70		60		55		50	ns	Measured from 10%–90% of signal
t_L	Clock Low Time	435		150		90		80		ns	50% duty cycle, square wave input on XIN
t_H	Clock High Time	450		145		85		75		ns	50% duty cycle, square wave input on XIN
$t_{ACC(OP)}$	ALE to Valid Data		1340		490		340		300	ns	Add t for each $\overline{WAIT}$ STATE
$t_{ACC(MR)}$	ALE to Valid Data		1875		620		405		360	ns	Add t for each $\overline{WAIT}$ STATE
t_{AFR}	AD(0–7) Float after $\overline{RD}$ Falling		0		0		0		0	ns	
t_{BABE}	$\overline{BACK}$ Rising to Bus Enable		1000		400		300		250	ns	
t_{BABF}	$\overline{BACK}$ Falling to Bus Float		50		50		50		50	ns	
t_{BACL}	$\overline{BACK}$ Fall to CLK Falling	425		125		60		55		ns	
t_{BRH}	$\overline{BREQ}$ Hold Time	0		0		0		0		ns	
t_{BRS}	$\overline{BREQ}$ Set-Up Time	100		50		50		45		ns	
t_{CAF}	Clock Falling ALE Falling	0	70	0	65	0	60	0	55	ns	
t_{CAR}	Clock Rising to ALE Rising	0	100	0	100	0	90	0	80	ns	
t_{CRD}	Clock Rising to Read Rising		100		90		90		80	ns	
t_{CRF}	Clock Rising to Refresh Falling		80		70		70		65	ns	
t_{DAI}	ALE Falling to $\overline{INTA}$ Falling	445		160		95		85		ns	
t_{DAR}	ALE Falling to $\overline{RD}$ Falling	400	575	160	250	100	180	90	160	ns	
t_{DAW}	ALE Falling to $\overline{WR}$ Falling	900	1010	350	420	225	300	200	265	ns	
$t_{D(BACK)1}$	ALE Falling to $\overline{BACK}$ Falling	2460		975		635		560		ns	Add t for each $\overline{WAIT}$ state Add t for opcode fetch cycles
$t_{D(BACK)2}$	$\overline{BREQ}$ Rising to $\overline{BACK}$ Rising	500	1610	200	700	140	540	125	475	ns	
$t_{D(I)}$	ALE Falling to $\overline{INTR}$, $\overline{NMI}$, $\overline{RSTA\text{-}C}$, $\overline{PS}$, $\overline{BREQ}$, Inputs Valid		1360		475		284		250	ns	Add t for each $\overline{WAIT}$ state Add t for opcode fetch cycles
t_{DPA}	Rising $\overline{PS}$ to Falling ALE	500	1685	200	760	140	580	125	510	ns	See *Figure 14* also
$t_{D(WAIT)}$	ALE Falling to $\overline{WAIT}$ Input Valid		550		250		170		125	ns	

OP— Opcode Fetch
MR— Memory Read

4.0 AC Electrical Characteristics V_{CC} = 5V ±10%, GND = 0V, unless otherwise specified (Continued)

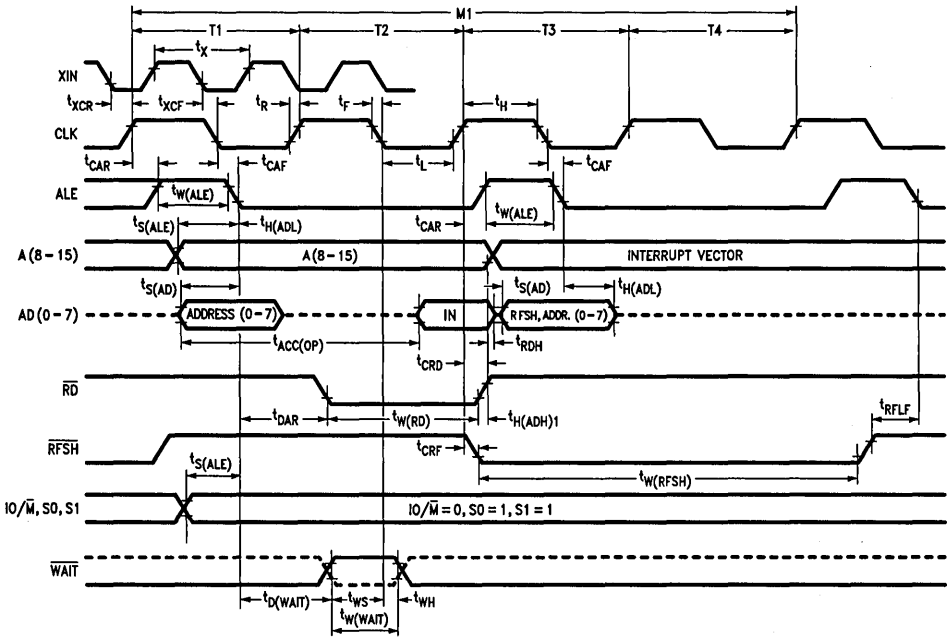
Symbol	Parameter	NSC800-1		NSC800-3		NSC800-35		NSC800-4		Units	Notes
		Min	Max	Min	Max	Min	Max	Min	Max		
$T_{H(ADH)1}$	A(8–15) Hold Time During Opcode Fetch	0		0		0		0		ns	
$T_{H(ADH)2}$	A(8–15) Hold Time During Memory or IO, $\overline{RD}$ and $\overline{WR}$	400		100		85		60		ns	
$T_{H(ADL)}$	AD(0–7) Hold Time	100		60		35		30		ns	
$T_{H(WD)}$	Write Data Hold Time	400		100		85		75		ns	
t_{INH}	Interrupt Hold Time	0		0		0		0		ns	
t_{INS}	Interrupt Set-Up Time	100		50		50		45		ns	
t_{NMI}	Width of NMI Input	50		30		25		20		ns	
t_{RDH}	Data Hold after Read	0		0		0		0		ns	
t_{RFLF}	$\overline{RFSH}$ Rising to ALE Falling	60		50		45		40		ns	
$t_{RL(MR)}$	$\overline{RD}$ Rising to ALE Rising (Memory Read)	390		100		50		45		ns	
$t_{S(AD)}$	AD(0–7) Set-Up Time	300		45		45		40		ns	
$t_{S(ALE)}$	A(8–15), SO, SI, IO/$\overline{M}$ Set-Up Time	350		70		55		50		ns	
$t_{S(WD)}$	Write Data Set-Up Time	385		75		35		30		ns	
$t_{W(ALE)}$	ALE Width	430		130		115		100		ns	
t_{WH}	$\overline{WAIT}$ Hold Time	0		0		0		0		ns	
$t_{W(I)}$	Width of $\overline{INTR}$, $\overline{RSTA-C}$, $\overline{PS}$, $\overline{BREQ}$	500		200		140		125		ns	
$t_{W(INTA)}$	$\overline{INTA}$ Strobe Width	1000		400		225		200		ns	Add two t states for first $\overline{INTA}$ of each interrupt response string Add t for each $\overline{WAIT}$ state
t_{WL}	$\overline{WR}$ Rising to ALE Rising	450		130		70		70		ns	
$t_{W(RD)}$	Read Strobe Width During Opcode Fetch	960		360		210		185		ns	Add t for each $\overline{WAIT}$ State Add t/2 for Memory Read Cycles
$t_{W(RFSH)}$	Refresh Strobe Width	1925		725		450		395		ns	
t_{WS}	$\overline{WAIT}$ Set-Up Time	100		70		60		55		ns	
$t_{W(WAIT)}$	$\overline{WAIT}$ Input Width	550		250		195		175		ns	
$t_{W(WR)}$	Write Strobe Width	985		370		250		220		ns	Add t for each $\overline{WAIT}$ state
t_{XCF}	XIN to Clock Falling	25	100	15	85	5	90	5	80	ns	
t_{XCR}	XIN to Clock Rising	25	85	15	85	5	90	5	80	ns	

Note 1: Test conditions: t = 1000 ns for NSC800-1, 400 ns for NSC800, 285 ns for NSC800-35, 250 ns for NSC800-4.

Note 2: Output timings are measured with a purely capacitive load of 100 pF.

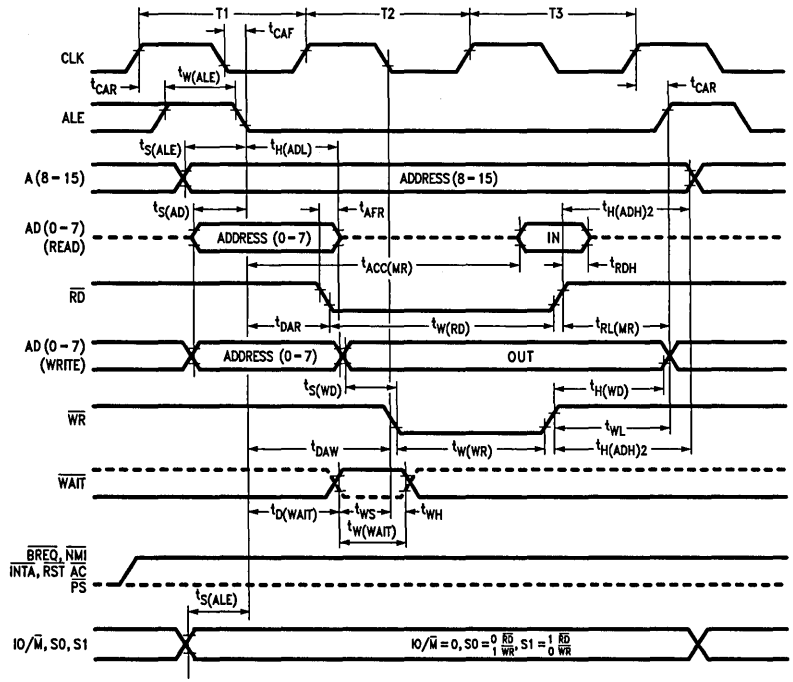
5.0 Timing Waveforms

Opcode Fetch Cycle



TL/C/5171-3

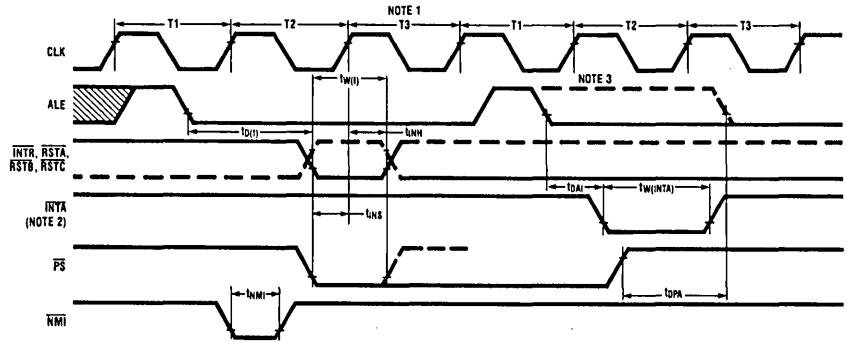
Memory Read and Write Cycle



TL/C/5171-4

5.0 Timing Waveforms (Continued)

Interrupt—Power-Save Cycle



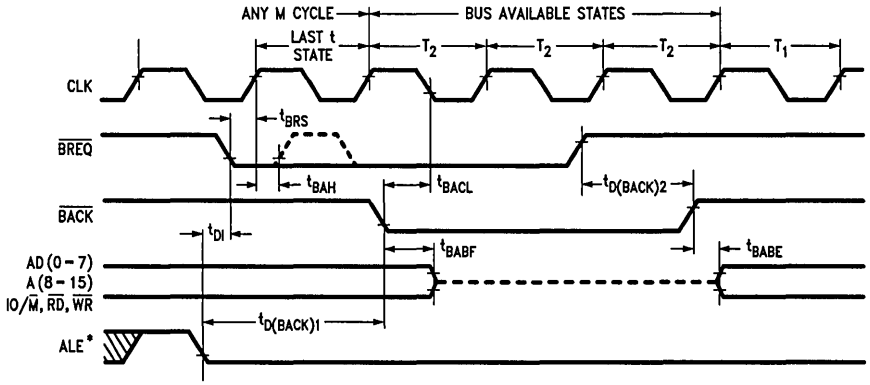
TL/C/5171-5

Note 1: This t state is the last t state of the last M cycle of any instruction.

Note 2: Response to INTR input.

Note 3: Response to PS input.

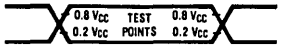
Bus Acknowledge Cycle



TL/C/5171-6

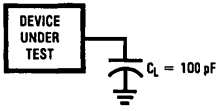
*Waveform not drawn to proportion. Use only for specifying test points.

AC Testing Input/Output Waveform



TL/C/5171-7

AC Testing Load Circuit



TL/C/5171-8

NSC800 HARDWARE

6.0 Pin Descriptions

6.1 INPUT SIGNALS

Reset Input ($\overline{\text{RESET IN}}$): Active low. Sets A (8–15) and AD (0–7) to TRI-STATE® (high impedance). Clears the contents of PC, I and R registers, disables interrupts, and activates reset out.

Bus Request ($\overline{\text{BREQ}}$): Active low. Used when another device requests the system bus. The NSC800 recognizes $\overline{\text{BREQ}}$ at the end of the current machine cycle, and sets A(8–15), AD(0–7), IO/$\overline{\text{M}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ to the high impedance state. $\overline{\text{RFSH}}$ is high during a bus request cycle. The CPU acknowledges the bus request via the $\overline{\text{BACK}}$ output signal.

Non-Maskable Interrupt ($\overline{\text{NMI}}$): Active low. The non-maskable interrupt, generated by the peripheral device(s), is the highest priority interrupt. The edge sensitive interrupt requires only a pulse to set an internal flip-flop which generates the internal interrupt request. The $\overline{\text{NMI}}$ flip-flop is monitored on the same clock edge as the other interrupts. It must also meet the minimum set-up time spec for the interrupt to be accepted in the current machine instruction. When the processor accepts the interrupt the flip-flop resets automatically. Interrupt execution is independent of the interrupt enable flip-flop. $\overline{\text{NMI}}$ execution results in saving the PC on the stack and automatic branching to restart address X'0066 in memory.

Restart Interrupts, A, B, C ($\overline{\text{RSTA}}$, $\overline{\text{RSTB}}$, $\overline{\text{RSTC}}$): Active low level sensitive. The CPU recognizes restarts generated by the peripherals at the end of the current instruction, if their respective interrupt enable and master enable bits are set. Execution is identical to $\overline{\text{NMI}}$ except the interrupts vector to the following restart addresses:

Name	Restart Address (X')
$\overline{\text{NMI}}$	0066
$\overline{\text{RSTA}}$	003C
$\overline{\text{RSTB}}$	0034
$\overline{\text{RSTC}}$	002C
$\overline{\text{INTR}}$ (Mode 1)	0038

The order of priority is fixed. The list above starts with the highest priority.

Interrupt Request ($\overline{\text{INTR}}$): Active low, level sensitive. The CPU recognizes an interrupt request at the end of the current instruction provided that the interrupt enable and master interrupt enable bits are set. $\overline{\text{INTR}}$ is the lowest priority interrupt. Program control selects one of three response modes which determines the method of servicing $\overline{\text{INTR}}$ in conjunction with $\overline{\text{INTA}}$. See Interrupt Control.

Wait (WAIT): Active low. When set low during $\overline{\text{RD}}$, $\overline{\text{WR}}$ or $\overline{\text{INTA}}$ machine cycles (during the $\overline{\text{WR}}$ machine cycle, wait must be valid prior to write going active) the CPU extends its machine cycle in increments of t (wait) states. The wait machine cycle continues until the $\overline{\text{WAIT}}$ input returns high.

The wait strobe input will be accepted only during machine cycles that have $\overline{\text{RD}}$, $\overline{\text{WR}}$ or $\overline{\text{INTA}}$ strobes and during the machine cycle immediately after an interrupt has been accepted by the CPU. The later cycle has its RD strobe suppressed but it will still accept the wait.

Power-Save ($\overline{\text{PS}}$): Active low. $\overline{\text{PS}}$ is sampled during the last t state of the current instruction cycle. When $\overline{\text{PS}}$ is low, the CPU stops executing at the end of current instruction and keeps itself in the low-power mode. Normal operation resumes when $\overline{\text{PS}}$ returns high (see Power Save Feature description).

CRYSTAL (X_{IN}, X_{OUT}): X_{IN} can be used as an external clock input. A crystal can be connected across X_{IN} and X_{OUT} to provide a source for the system clock.

6.2 OUTPUT SIGNALS

Bus Acknowledge ($\overline{\text{BACK}}$): Active low. $\overline{\text{BACK}}$ indicates to the bus requesting device that the CPU bus and its control signals are in the TRI-STATE mode. The requesting device then commands the bus and its control signals.

Address Bits 8–15 [A(8–15)]: Active high. These are the most significant 8 bits of the memory address during a memory instruction. During an I/O instruction, the port address on the lower 8 address bits gets duplicated onto A(8–15). During a BREQ/BACK cycle, the A(8–15) bus is in the TRI-STATE mode.

Reset Out (RESET OUT): Active high. When RESET OUT is high, it indicates the CPU is being reset. This signal is normally used to reset the peripheral devices.

Input/Output/Memory (IO/$\overline{\text{M}}$): An active high on the IO/$\overline{\text{M}}$ output signifies that the current machine cycle is an input/output cycle. An active low on the IO/$\overline{\text{M}}$ output signifies that the current machine cycle is a memory cycle. It is TRI-STATE during $\overline{\text{BREQ}}$/$\overline{\text{BACK}}$ cycles.

Refresh ($\overline{\text{RFSH}}$): Active low. The refresh output indicates that the dynamic RAM refresh cycle is in progress. $\overline{\text{RFSH}}$ goes low during T3 and T4 states of all M1 cycles. During the refresh cycle, AD(0–7) has the refresh address and A(8–15) indicates the interrupt vector register data. $\overline{\text{RFSH}}$ is high during $\overline{\text{BREQ}}$/$\overline{\text{BACK}}$ cycles.

Address Latch Enable (ALE): Active high. ALE is active only during the T1 state of any M cycle and also T3 state of the M1 cycle. The high to low transition of ALE indicates that a valid memory, I/O or refresh address is available on the AD(0–7) lines.

Read Strobe ($\overline{\text{RD}}$): Active low. The CPU receives data via the AD(0–7) lines on the trailing edge of the $\overline{\text{RD}}$ strobe. The $\overline{\text{RD}}$ line is in the TRI-STATE mode during $\overline{\text{BREQ}}$/$\overline{\text{BACK}}$ cycles.

Write Strobe ($\overline{\text{WR}}$): Active low. The CPU sends data via the AD(0–7) lines while the $\overline{\text{WR}}$ strobe is low. The $\overline{\text{WR}}$ line is in the TRI-STATE mode during $\overline{\text{BREQ}}$/$\overline{\text{BACK}}$ cycles.

Clock (CLK): CLK is the output provided for use as a system clock. The CLK output is a square wave at one half the input frequency.

Interrupt Acknowledge ($\overline{\text{INTA}}$): Active low. This signal strobes the interrupt response vector from the interrupting peripheral devices onto the AD(0–7) lines. $\overline{\text{INTA}}$ is active during the M1 cycle immediately following the t state where the CPU recognized the $\overline{\text{INTR}}$ interrupt request.

Two of the three interrupt request modes use $\overline{\text{INTA}}$. In mode 0 one to four $\overline{\text{INTA}}$ signals strobe a one to four byte instruction onto the AD(0–7) lines. In mode 2 one $\overline{\text{INTA}}$ signal strobes the lower byte of an interrupt response vector onto the bus. In mode 1, $\overline{\text{INTA}}$ is inactive and the CPU response to $\overline{\text{INTR}}$ is the same as for an NMI or restart interrupt.

6.0 Pin Descriptions (Continued)

Status (SO, S1): Bus status outputs provide encoded information regarding the current M cycle as follows:

Machine Cycle	Status			Control	
	S0	S1	IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$
Opcode Fetch	1	1	0	0	1
Memory Read	0	1	0	0	1
Memory Write	1	0	0	1	0
I/O Read	0	1	1	0	1
I/O Write	1	0	1	1	0
Halt*	0	0	0	0	1
Internal Operation*	0	1	0	1	1
Acknowledge of Int**	1	1	0	1	1

*ALE is not suppressed in this cycle.

**This is the cycle that occurs immediately after the CPU accepts an interrupt ($\overline{RSTA}$, $\overline{RSTB}$, $\overline{RSTC}$, $\overline{INTR}$, $\overline{NMI}$).

Note 1: During halt, CPU continues to do dummy opcode fetch from location following the halt instruction with a halt status. This is so CPU can continue to do its dynamic RAM refresh.

Note 2: No early status is provided for interrupt or hardware restarts.

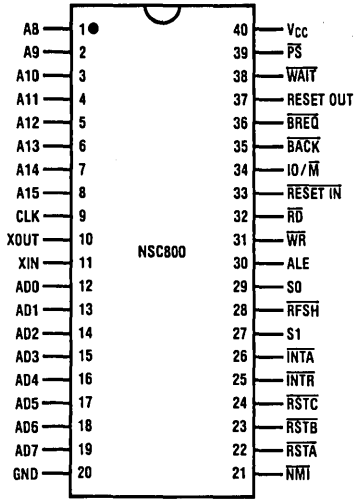
6.3 INPUT/OUTPUT SIGNALS

Multiplexed Address/Data [AD(0–7)]: Active high

At $\overline{RD}$ Time:	Input data to CPU.
At $\overline{WR}$ Time:	Output data from CPU.
At Falling Edge of ALE Time:	Least significant byte of address during memory reference cycle. 8-bit port address during I/O reference cycle.
During $\overline{BREQ}$/ $\overline{BACK}$ Cycle:	High impedance.

7.0 Connection Diagrams

Dual-In-Line Package

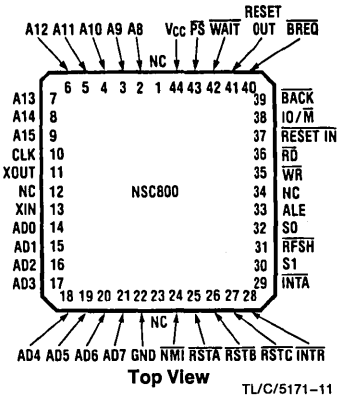


Top View

TL/C/5171–10

Order Number NSC800D or N
See NS Package D40C or N40A

Chip Carrier Package



Top View

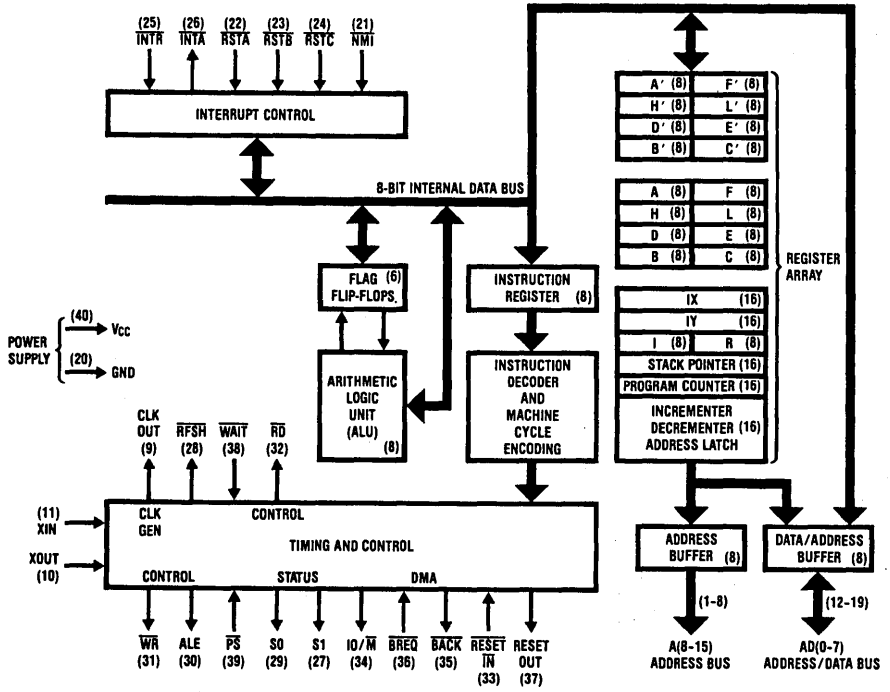
TL/C/5171–11

Order Number NSC800E or V
See NS Package E44B or V44A

8.0 Functional Description

This section reviews the CPU architecture shown below, focusing on the functional aspects from a hardware perspective, including timing details.

As illustrated in *Figure 1*, the NSC800 is an 8-bit parallel device. The major functional blocks are: the ALU, register array, interrupt control, timing and control logic. These areas are connected via the 8-bit internal data bus. Detailed descriptions of these blocks ae provided in the following sections.



TL/C/5171–9

Note: Applicable pinout for 40-pin dual-in-line package within parentheses

FIGURE 1. NSC800 CPU Functional Block Diagram

8.0 Functional Description (Continued)

8.1 REGISTER ARRAY

The NSC800 register array is divided into two parts: the dedicated registers and the working registers, as shown in *Figure 2*.

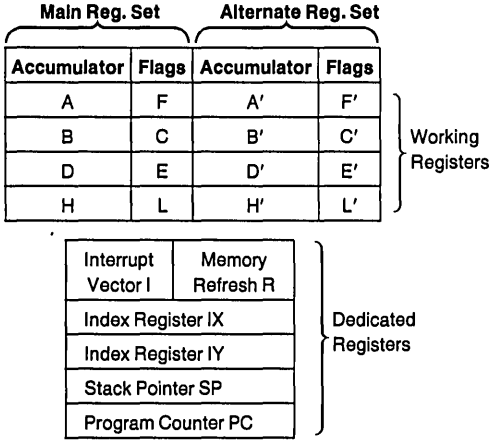


FIGURE 2. NSC800 Register Array

8.2 DEDICATED REGISTERS

There are 6 dedicated registers in the NSC800: two 8-bit and four 16-bit registers (see *Figure 3*).

Although their contents are under program control, the program has no control over their operational functions, unlike the CPU working registers. The function of each dedicated register is described as follows:

CPU Dedicated Registers

Program Counter PC	(16)
Stack Pointer SP	(16)
Index Register IX	(16)
Index Register IY	(16)
Interrupt Vector Register I	(8)
Memory Refresh Register R	(8)

FIGURE 3. Dedicated Registers

8.2.1 Program Counter (PC)

The program counter contains the 16-bit address of the current instruction being fetched from memory. The PC increments after its contents have been transferred to the address lines. When a program jump occurs, the PC receives the new address which overrides the incrementer.

There are many conditional and unconditional jumps, calls, and return instructions in the NSC800's instruction repertoire that allow easy manipulation of this register in controlling the program execution (i.e. JP NZ nn, JR Zd2, CALL NC, nn).

8.2.2 Stack Pointer (SP)

The 16-bit stack pointer contains the address of the current top of stack that is located in external system RAM. The stack is organized in a last-in, first-out (LIFO) structure. The pointer decrements before data is pushed onto the stack, and increments after data is popped from the stack.

Various operations store or retrieve, data on the stack. This, along with the usage of subroutine calls and interrupts, allows simple implementation of subroutine and interrupt nesting as well as alleviating many problems of data manipulation.

8.2.3 Index Register (IX and IY)

The NSC800 contains two index registers to hold independent, 16-bit base addresses used in the indexed addressing mode. In this mode, an index register, either IX or IY, contains a base address of an area in memory making it a pointer for data tables.

In all instructions employing indexed modes of operation, another byte acts as a signed two's complement displacement. This addressing mode enables easy data table manipulations.

8.2.4 Interrupt Register (I)

When the NSC800 provides a Mode 2 response to $\overline{\text{INTR}}$, the action taken is an indirect call to the memory location containing the service routine address. The pointer to the address of the service routine is formed by two bytes, the high-byte is from the I Register and the low-byte is from the interrupting peripheral. The peripheral always provides an even address for the lower byte (LSB=0). When the processor receives the lower byte from the peripheral it concatenates it in the following manner:

I Register	External byte	
8 bits		0

↑

The LSB of the external byte must be zero.

FIGURE 4a. Interrupt Register

The even memory location contains the low-order byte, the next consecutive location contains the high-order byte of the pointer to the beginning address of the interrupt service routine.

8.2.5 Refresh Register (R)

For systems that use dynamic memories rather than static RAM's, the NSC800 provides an integral 8-bit memory refresh counter. The contents of the register are incremented after each opcode fetch and are sent out on the lower portion of the address bus, along with a refresh control signal. This provides a totally transparent refresh cycle and does not slow down CPU operation.

The program can read and write to the R register, although this is usually done only for test purposes.

8.0 Functional Description (Continued)

8.3 CPU WORKING AND ALTERNATE REGISTER SETS

8.3.1 CPU Working Registers

The portion of the register array shown in *Figure 4b* represents the CPU working registers. These sixteen 8-bit registers are general-purpose registers because they perform a multitude of functions, depending on the instruction being executed. They are grouped together also due to the types of instructions that use them, particularly alternate set operations.

The F (flag) register is a special-purpose register because its contents are more a result of machine status rather than program data. The F register is included because of its interaction with the A register, and its manipulations in the alternate register set operations.

8.3.2 Alternate Registers

The NSC800 registers designated as CPU working registers have one common feature: the existence of a duplicate register in an alternate register set. This architectural concept simplifies programming during operations such as interrupt response, when the machine status represented by the contents of the registers must be saved.

The alternate register concept makes one set of registers available to the programmer at any given time. Two instructions (EX AF, A'F' and EXX), exchange the current working set of registers with their alternate set. One exchange between the A and F registers and their respective duplicates (A' and F') saves the primary status information contained in the accumulator and the flag register. The second exchange instruction performs the exchange between the remaining registers, B, C, D, E, H, and L, and their respective alternates B', C', D', E', H', and L'. This essentially saves the contents of the original complement of registers while providing the programmer with a usable alternate set.

CPU Main Working Register Set

Accumulator A	(8)	Flags F	(8)
Register B	(8)	Register C	(8)
Register D	(8)	Register E	(8)
Register H	(8)	Register L	(8)

CPU Alternate Working Register Set

Accumulator A'	(8)	Flags F'	(8)
Register B'	(8)	Register C'	(8)
Register D'	(8)	Register E'	(8)
Register H'	(8)	Register L'	(8)

FIGURE 4b. CPU Working and Alternate Registers

8.4 REGISTER FUNCTIONS

8.4.1 Accumulator (A Register)

The A register serves as a source or destination register for data manipulation instructions. In addition, it serves as the accumulator for the results of 8-bit arithmetic and logic operations.

The A register also has a special status in some types of operations; that is, certain addressing modes are reserved for the A register only, although the function is available for all the other registers. For example, any register can be loaded by immediate, register indirect, or indexed addressing modes. The A register, however, can also be loaded via an additional register indirect addressing.

Another special feature of the A register is that it produces more efficient memory coding than equivalent instruction functions directed to other registers. Any register can be rotated; however, while it requires a two-byte instruction to normally rotate any register, a single-byte instruction is available for rotating the contents of the accumulator (A register).

8.4.2 F Register - Flags

The NSC800 flag register consists of six status bits that contain information regarding the results of previous CPU operations. The register can be read by pushing the contents onto the stack and then reading it, however, it cannot be written to. It is classified as a register because of its affiliation with the accumulator and the existence of a duplicate register for use in exchange instructions with the accumulator.

Of the six flags shown in *Figure 5*, only four can be directly tested by the programmer via conditional jump, call, and return instructions. They are the Sign (S), Zero (Z), Parity/Overflow (P/V), and Carry (C) flags. The Half Carry (H) and Add/Subtract (N) flags are used for internal operations related to BCD arithmetic.

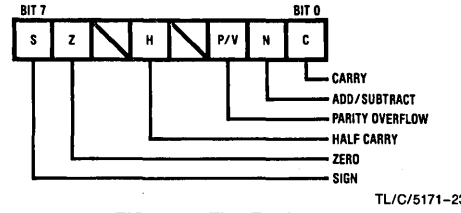


FIGURE 5. Flag Register

8.0 Functional Description (Continued)

8.4.3 Carry (C)

A carry from the highest order bit of the accumulator during an add instruction, or a borrow generated during a subtraction instruction sets the carry flag. Specific shift and rotate instructions also affect this bit.

Two specific instructions in the NSC800 instruction repertoire set (SCF) or complement (CCF) the carry flag.

Other operations that affect the C flag are as follows:

- Adds
- Subtracts
- Logic Operations (always resets C flag)
- Rotate Accumulator
- Rotate and Shifts
- Decimal Adjust
- Negation of Accumulator

Other operations do not affect the C flag.

8.4.4 Adds/Subtract (N)

This flag is used in conjunction with the H flag to ensure that the proper BCD correction algorithm is used during the decimal adjust instruction (DAA). The correction algorithm depends on whether an add or subtract was previously done with BCD operands.

The operations that set the N flag are:

- Subtractions
- Decrements (8-bit)
- Complementing of the Accumulator
- Block I/O
- Block Searches
- Negation of the Accumulator

The operations that reset the N flag are:

- Adds
- Increments
- Logic Operations
- Rotates
- Set and Complement Carry
- Input Register Indirect
- Block Transfers
- Load of the I or R Registers
- Bit Tests

Other operations do not affect the N flag.

8.4.5 Parity/Overflow (P/V)

The Parity/Overflow flag is a dual-purpose flag that indicates results of logic and arithmetic operations. In logic operations, the P/V flag indicates the parity of the result; the flag is set (high) if the result is even, reset (low) if the result is odd. In arithmetic operations, it represents an overflow condition when the result, interpreted as signed two's complement arithmetic, is out of range for the eight-bit accumulator (i.e. −128 to +127).

The following operations affect the P/V flag according to the parity of the result of the operation:

- Logic Operations
- Rotate and Shift
- Rotate Digits
- Decimal Adjust
- Input Register Indirect

The following operations affect the P/V flag according to the overflow result of the operation.

- Adds (16 bit with carry, 8-bit with/without carry)
- Subtracts (16 bit with carry, 8-bit with/without carry)
- Increments and Decrements
- Negation of Accumulator

The P/V flag has no significance immediately after the following operations.

- Block I/O
- Bit Tests

In block transfers and compares, the P/V flag indicates the status of the BC register, always ending in the reset state after an auto repeat of a block move. Other operations do not affect the P/V flag.

8.4.6 Half Carry (H)

This flag indicates a BCD carry, or borrow, result from the low-order four bits of operation. It can be used to correct the results of a previously packed decimal add, or subtract, operation by use of the Decimal Adjust Instruction (DAA).

The following operations affect the H flag:

- Adds (8-bit)
- Subtracts (8-bit)
- Increments and Decrements
- Decimal Adjust
- Negation of Accumulator
- Always Set by: Logic AND
  Complement Accumulator
  Bit Testing
- Always Reset By: Logic OR's and XOR's
  Rotates and Shifts
  Set Carry
  Input Register Indirect
  Block Transfers
  Loads of I and R Registers

The H flag has no significance immediately after the following operations.

- 16-bit Adds with/without carry
- 16-Bit Subtracts with carry
- Complement of the carry
- Block I/O
- Block Searches

Other operations do not affect the H flag.

8.0 Functional Description (Continued)

8.4.7 Zero Flag (Z)

Loading a zero in the accumulator or when a zero results from an operation sets the zero flag.

The following operations affect the zero flag.

- Adds (16-bit with carry, 8-bit with/without carry)
- Subtracts (16-bit with carry, 8-bit with/without carry)
- Logic Operations
- Increments and Decrements
- Rotate and Shifts
- Rotate Digits
- Decimal Adjust
- Input Register Indirect
- Block I/O (always set after auto repeat block I/O)
- Block Searches
- Load of I and R Registers
- Bit Tests
- Negation of Accumulator

The Z flag has no signficance immediately after the following operations:

- Block Transfers

Other operations do not affect the zero flag.

8.4.8 Sign Flag (S)

The sign flag stores the state of bit 7 (the most-significant bit and sign bit) of the accumulator following an arithmetic operation. This flag is of use when dealing with signed numbers.

The sign flag is affected by the following operation according to the result:

- Adds (16-bit with carry, 8-bit with/without carry)
- Subtracts (16-bit with carry, 8-bit with/without carry)
- Logic Operations
- Increments and Decrements
- Rotate and Shifts
- Rotate Digits
- Decimal Adjust
- Input Register Indirect
- Block Search
- Load of I and R Registers
- Negation of Accumulator

The S flag has no significance immediately after the following operations:

- Block I/O
- Block Transfers
- Bit Tests

Other operations do not affect the sign bit.

8.4.9 Additional General-Purpose Registers

The other general-purpose registers are the B, C, D, E, H and L registers and their alternate register set, B', C', D', E', H' and L'. The general-purpose registers can be used interchangeably.

In addition, the B and C registers perform special functions in the NSC800 expanded I/O capabilities, particularly block I/O operations. In these functions, the C register can address I/O ports; the B register provides a counter function when used in the register indirect address mode.

When used with the special condition jump instruction (DJNZ) the B register again provides the counter function.

8.4.10 Alternate Configurations

The six 8-bit general purpose registers (B,C,D,E,H,L) will combine to form three 16-bit registers. This occurs by concatenating the B and C registers to form the BC register, the D and E registers form the DE register, and the H and L registers form the HL register.

Having these 16-bit registers allows 16-bit data handling, thereby expanding the number of 16-bit registers available for memory addressing modes. The HL register typically provides the pointer address for use in register indirect addressing of the memory.

The DE register provides a second memory pointer register for the NSC800's powerful block transfer operations. The BC register also provides an assist to the block transfer operations by acting as a byte-counter for these operations.

8.5 ARITHMETIC-LOGIC UNIT (ALU)

The arithmetic, logic and rotate instructions are performed by the ALU. The ALU internally communicates with the registers and data buffer on the 8-bit internal data bus.

8.6 INSTRUCTION REGISTER AND DECODER

During an opcode fetch, the first byte of an instruction is transferred from the data buffer (i.e. its on the internal data bus) to the instruction register. The instruction register feeds the instruction decoder, which gated by timing signals, generates the control signals that read or write data from or to the registers, control the ALU and provide all required external control signals.

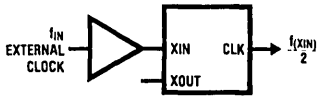
9.0 Timing and Control

9.1 INTERNAL CLOCK GENERATOR

An inverter oscillator contained on the NSC800 chip provides all necessary timing signals. The chip operation frequency is equal to one half of the frequency of this oscillator.

The oscillator frequency can be controlled by one of the following methods:

1. Leaving the X_{OUT} pin unterminated and driving the X_{IN} pin with an externally generated clock as shown in *Figure 6*. When driving X_{IN} with a square wave, the minimum duty cycle is 30% high.

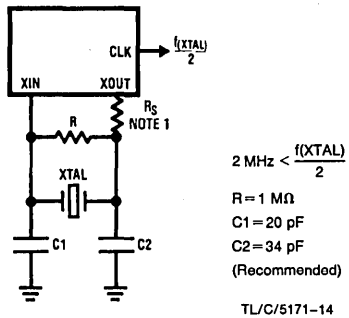


TL/C/5171–13

FIGURE 6. Use of External Clock

2. Connecting a crystal with the proper biasing network between X_{IN} and X_{OUT} as shown in *Figure 7*. Recommended crystal is a parallel resonance AT cut crystal.

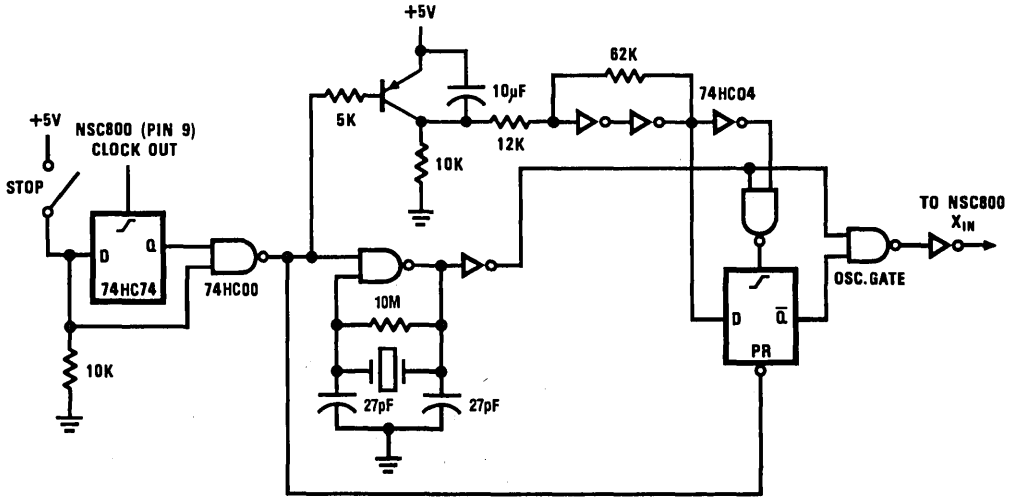
Note 1: If the crystal frequency is between 1 MHz and 2 MHz a series resistor, R_S, (470Ω to 1500Ω) should be connected between X_{OUT} and R, XTAL and C_Z. Additionally, the capacitance of C1 and C2 should be increased by 2 to 3 times the recommended value. For crystal frequencies less than 1 MHz higher values of C1 and C2 may be required. Crystal parameters will also affect the capacitive loading requirements.



TL/C/5171–14

FIGURE 7. Use Of Crystal

The CPU has a minimum clock frequency input (@ X_{IN}) of 300 kHz, which results in 150 kHz system clock speed. All registers internal to the chip are static, however there is dynamic logic which limits the minimum clock speed. The input clock can be stopped without fear of losing any data or damaging the part. You stop it in the phase of the clock that has X_{IN} low and CLK OUT high. When restarting the CPU, precautions must be taken so that the input clock meets these minimum specification. Once started, the CPU will continue operation from the same location at which it was stopped. During DC operation of the CPU, typical current drain will be 2 mA. This current drain can be reduced by placing the CPU in a wait state during an opcode fetch cycle then stopping the clock. For clock stop circuit, see *Figure 8*.



TL/C/5171–36

FIGURE 8. Clock Stop Circuit

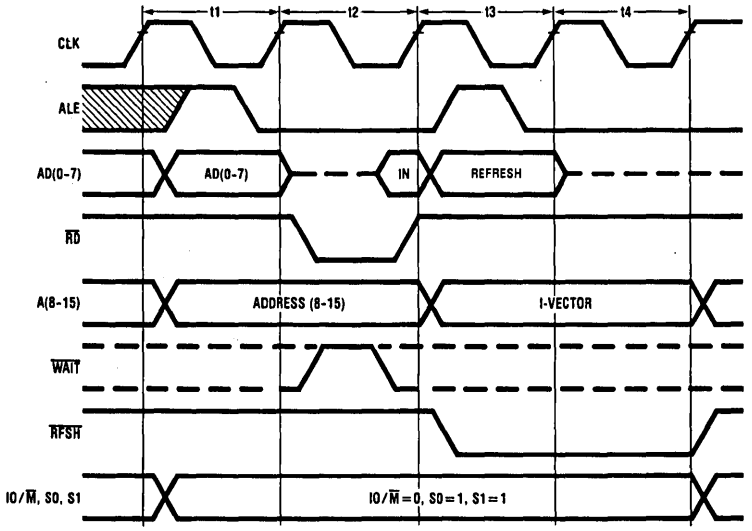
9.0 Timing and Control (Continued)

9.2 CPU TIMING

The NSC800 uses a multiplexed bus for data and addresses. The 16-bit address bus is divided into a high-order 8-bit address bus that handles bits 8–15 of the address, and a low-order 8-bit multiplexed address/data bus that handles bits 0–7 of the address and bits 0–7 of the data. Strobe outputs from the NSC800 (ALE, $\overline{RD}$ and $\overline{WR}$) indicate when a valid address or data is present on the bus. IO/$\overline{M}$ indicates whether the ensuing cycle accesses memory or I/O.

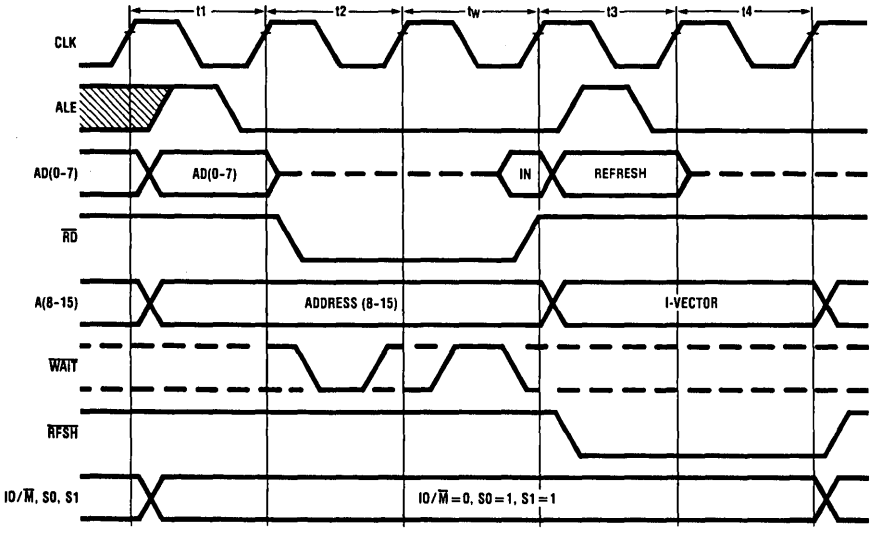
During an input or output instruction, the CPU duplicates the lower half of the address [AD(0–7)] onto the upper address bus [A(8–15)]. The eight bits of address will stay on A(8–15) for the entire machine cycle and can be used for chip selection directly.

Figure 9 illustrates the timing relationship for opcode fetch cycles with and without a wait state.



TL/C/5171–15

FIGURE 9a. Opcode Fetch Cycles without $\overline{WAIT}$ States

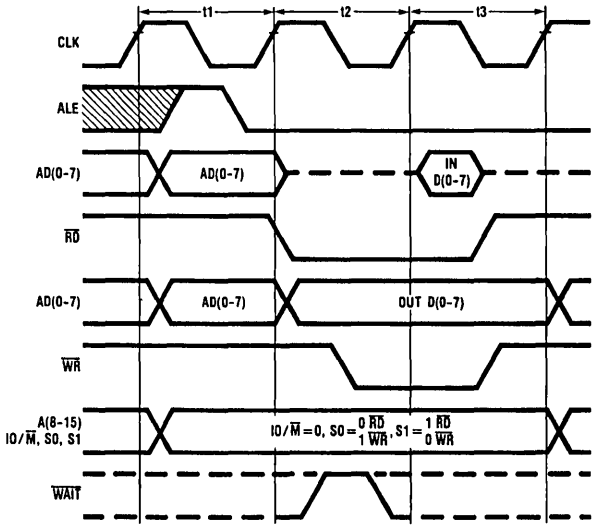


TL/C/5171–16

FIGURE 9b. Opcode Fetch Cycles with $\overline{WAIT}$ States

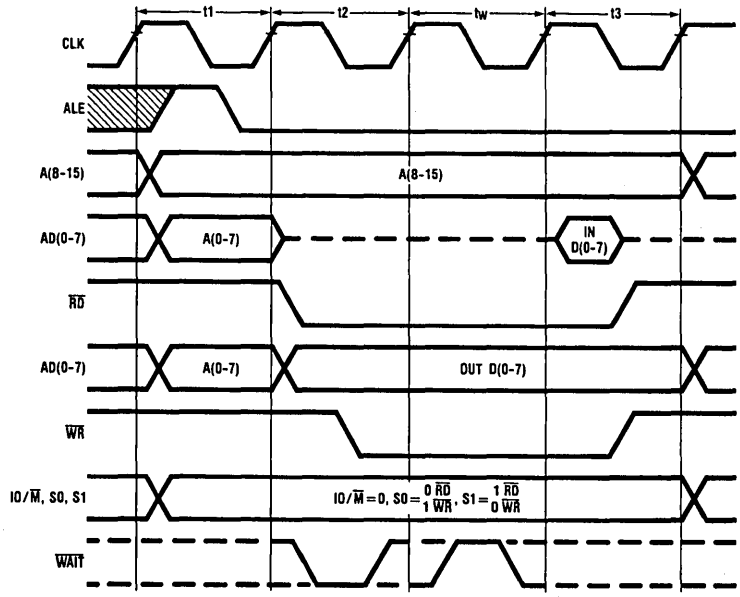
9.0 Timing and Control (Continued)

During the opcode fetch, the CPU places the contents of the PC on the address bus. The falling edge of ALE indicates a valid address on the AD(0–7) lines. The $\overline{WAIT}$ input is sampled during t_2 and if active causes the NSC800 to insert a wait state (t_W). WAIT is sampled again during t_W so that when it goes inactive, the CPU continues its opcode fetch by latching in the data on the rising edge of $\overline{RD}$ from the AD(0–7) lines. During t_3, $\overline{RFSH}$ goes active and AD(0–7) has the dynamic RAM refresh address from register R and A(8–15) the interrupt vector from register I.



TL/C/5171–17

FIGURE 10a. Memory Read/Write Cycles without $\overline{WAIT}$ States



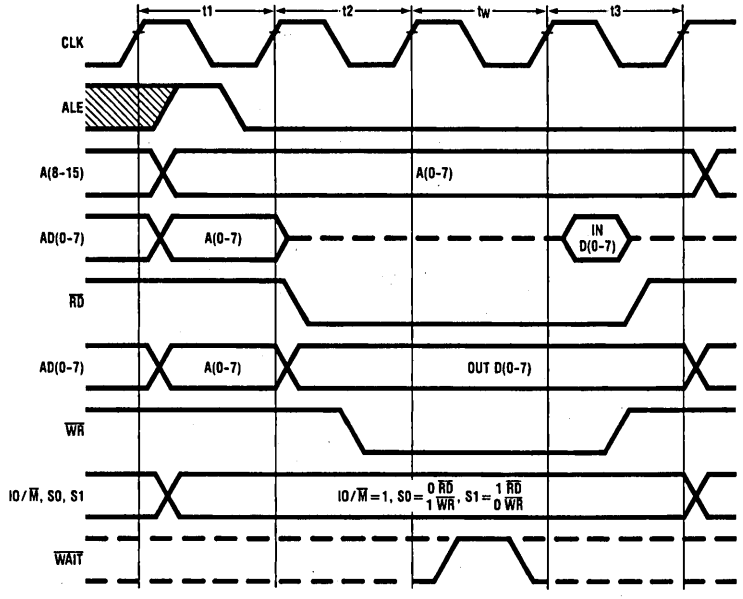
TL/C/5171–18

FIGURE 10b. Memory Read and Write with $\overline{WAIT}$ States

9.0 Timing and Control (Continued)

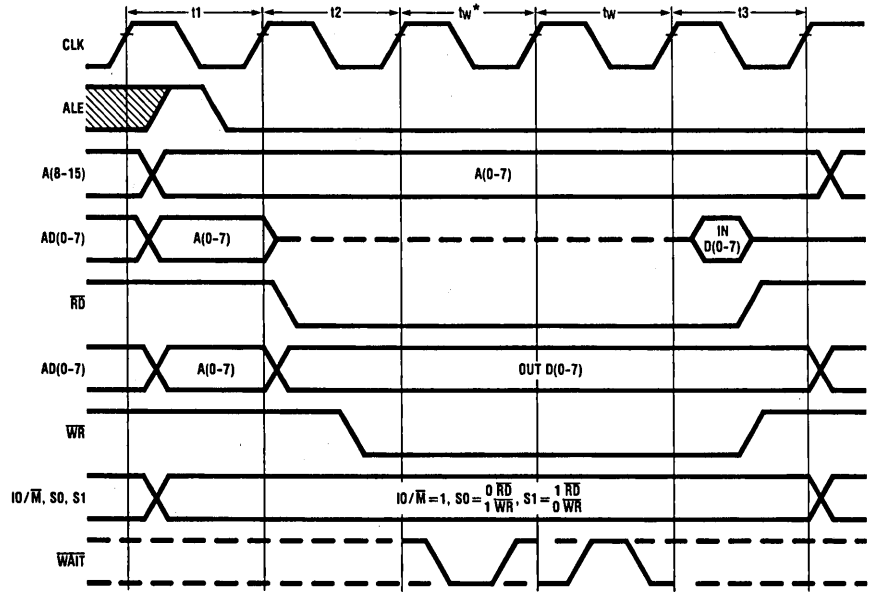
Figure 10 shows the timing for memory read (other than opcode fetchs) and write cycles with and without a wait state. The $\overline{RD}$ stobe is widened by $\frac{t}{2}$ (half the machine state) for memory reads so that the actual latching of the input data occurs later.

Figure 11 shows the timing for input and output cycles with and without wait states. The CPU automatically inserts one wait state into each I/O instruction to allow sufficient time for an I/O port to decode the address.



TL/C/5171-19

FIGURE 11a. Input and Output Cycles without $\overline{WAIT}$ States



TL/C/5171-20

*$\overline{WAIT}$ state automatically inserted during IO operation.

FIGURE 11b. Input and Output Cycles with $\overline{WAIT}$ States

9.0 Timing and Control (Continued)

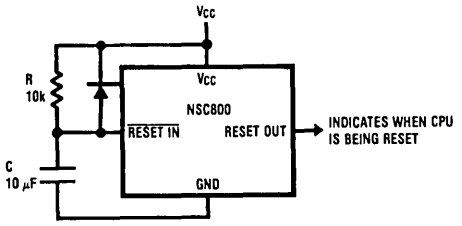
9.3 INITIALIZATION

$\overline{\text{RESET IN}}$ initializes the NSC800; RESET OUT initializes the peripheral components. The Schmitt trigger at the $\overline{\text{RESET IN}}$ input facilitates using an R-C network reset scheme during power up (see *Figure 12*).

To ensure proper power-up conditions for the NSC800, the following power-up and initialization procedure is recommended:

1. Apply power (V_{CC} and GND) and set $\overline{\text{RESET IN}}$ active (low). Allow sufficient time (approximately 30 ms if a crystal is used) for the oscillator and internal clocks to stabilize. $\overline{\text{RESET IN}}$ must remain low for at least 3t state (CLK) times. RESET OUT goes high as soon as the active $\overline{\text{RESET IN}}$ signal is clocked into the first flip-flop after the on-chip Schmitt trigger. RESET OUT signal is available to reset the peripherals.
2. Set $\overline{\text{RESET IN}}$ high. RESET OUT then goes low as the inactive $\overline{\text{RESET IN}}$ signal is clocked into the first flip-flop after the on-chip Schmitt trigger. Following this the CPU initiates the first opcode fetch cycle.

Note: The NSC800 initialization includes: Clear PC to X'0000 (the first opcode fetch, therefore, is from memory location X'0000). Clear registers I (Interrupt Vector Base) and R (Refresh Counter) to X'00. Clear interrupt control register bits IEA, IEB and IEC. The interrupt control bit IEI is set to 1 to maintain INS8080A/Z80A compatibility (see INTERRUPTS for more details). The CPU disables maskable interrupts and enters $\overline{\text{INTR}}$ Mode 0. While $\overline{\text{RESET IN}}$ is active (low), the A(8–15) and AD(0–7) lines go to high impedance (TRI-STATE) and all CPU strobes go to the inactive state (see *Figure 13*).

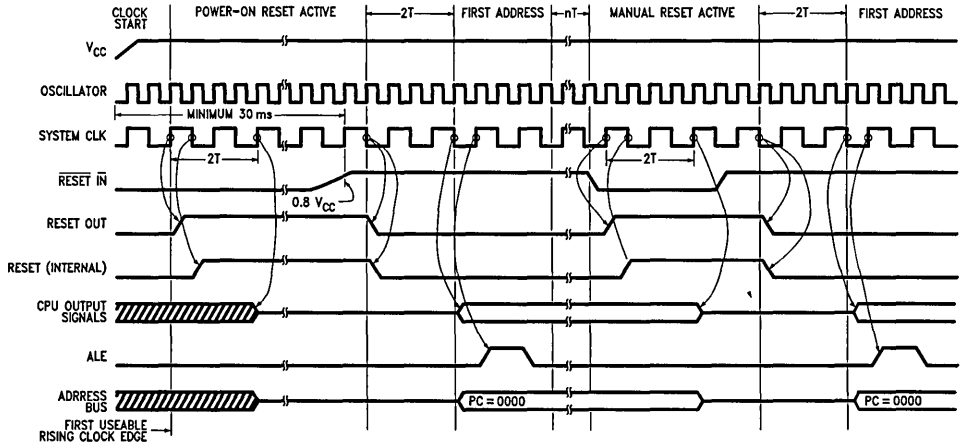


TL/C/5171–21

FIGURE 12. Power-On Reset

9.4 POWER-SAVE FEATURE

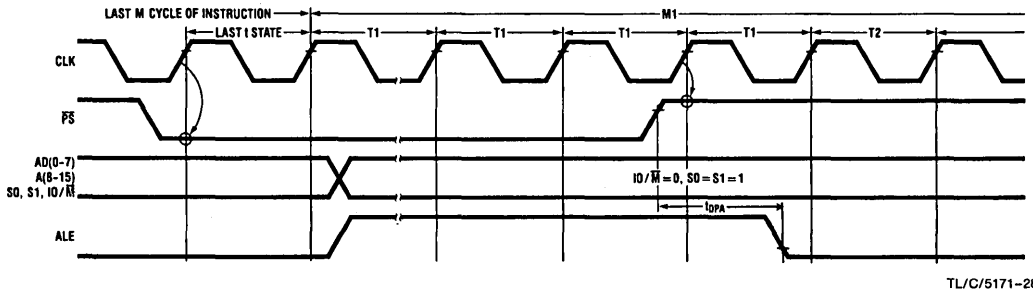
The NSC800 provides a unique power-save mode by the means of the $\overline{\text{PS}}$ pin. $\overline{\text{PS}}$ input is sampled at the last t state of the last M cycle of an instruction. After recognizing an active (low) level on $\overline{\text{PS}}$, The NSC800 stops its internal clocks, thereby reducing its power dissipation to one half of operating power, yet maintaining all register values and internal control status. The NSC800 keeps its oscillator running, and makes the CLK signal available to the system. When in power-save the ALE strobe will be stopped high and the address lines [AD(0–7), A(8–15)] will indicate the next machine address. When $\overline{\text{PS}}$ returns high, the opcode fetch (or M1 cycle) of the CPU begins in a normal manner. Note this M1 cycle could also be an interrupt acknowledge cycle if the NSC800 was interrupted simultaneously with $\overline{\text{PS}}$ (i.e. $\overline{\text{PS}}$ has priority over a simultaneously occurring interrupt). However, interrupts are not accepted during power save. *Figure 14* illustrates the power save timing.



TL/C/5171–74

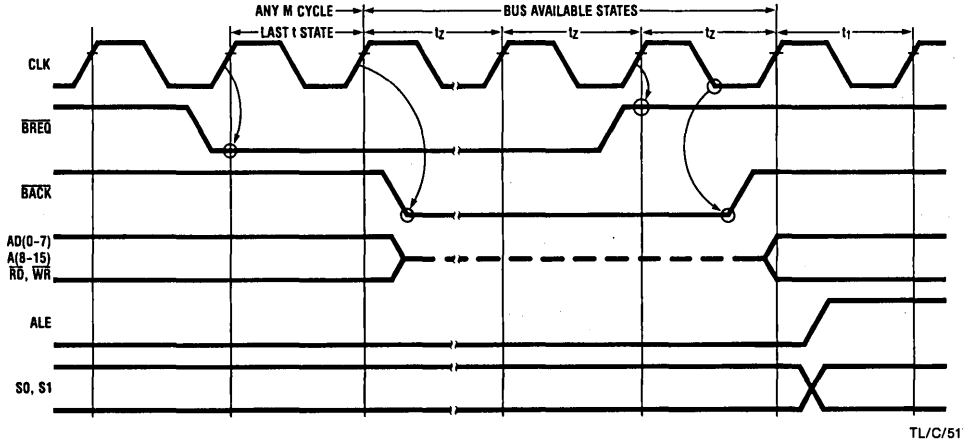
FIGURE 13. NSC800 Signals During Power-On and Manual Reset

9.0 Timing and Control (Continued)



TL/C/5171-28

FIGURE 14. NSC800 Power-Save



TL/C/5171-22

*S0, S1 during $\overline{BREQ}$ will indicate same machine cycle as during the cycle when $\overline{BREQ}$ was accepted.

t_Z = time states during which bus and control signals are in high impedance mode.

FIGURE 15. Bus Acknowledge Cycle

In the event $\overline{BREQ}$ is asserted (low) at the end of an instruction cycle and $\overline{PS}$ is active simultaneously, the following occurs:

1. The NSC800 will go into $\overline{BACK}$ cycle.
2. Upon completion of $\overline{BACK}$ cycle if $\overline{PS}$ is still active the CPU will go into power-save mode.

9.5 BUS ACCESS CONTROL

Figure 15 illustrates bus access control in the NSC800. The external device controller produces an active $\overline{BREQ}$ signal that requests the bus. When the CPU responds with $\overline{BACK}$ then the bus and related control strobes go to high impedance (TRI-STATE) and the $\overline{RFSH}$ signal remains high. It should be noted that (1) $\overline{BREQ}$ is sampled at the last t state of any M machine cycle only. (2) The NSC800 will not acknowledge any interrupt/restart requests, and will not perform any dynamic RAM refresh functions until after $\overline{BREQ}$ input signal is inactive high. (3) $\overline{BREQ}$ signal has priority over all interrupt request signals, should $\overline{BREQ}$ and interrupt request become active simultaneously. Therefore, interrupts latched at the end of the instruction cycle will be serviced after a simultaneously occurring $\overline{BREQ}$. $\overline{NMI}$ is latched during an active $\overline{BREQ}$.

9.6 INTERRUPT CONTROL

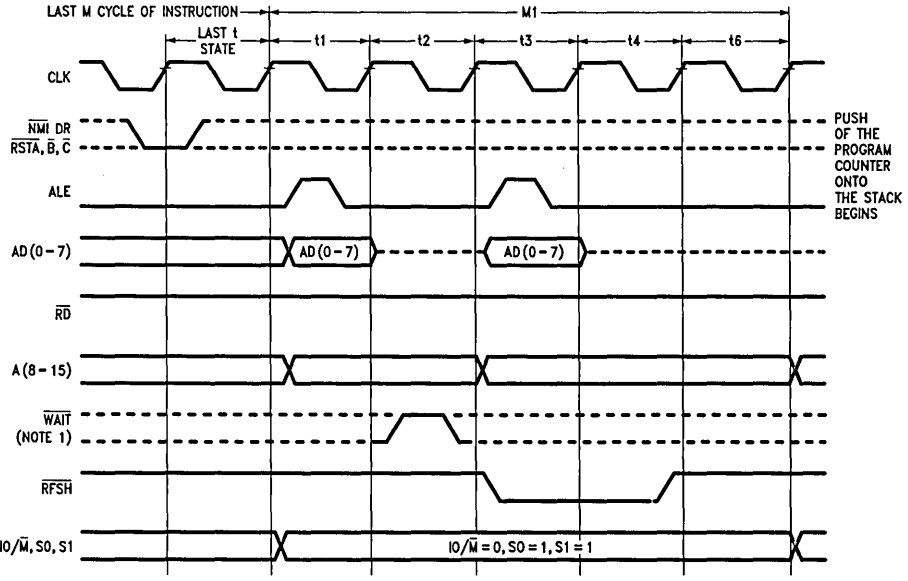
The NSC800 has five interrupt/restart inputs, four are maskable ($\overline{RSTA}$, $\overline{RSTB}$, $\overline{RSTC}$, and $\overline{INTR}$) and one is non-maskable ($\overline{NMI}$). $\overline{NMI}$ has the highest priority of all interrupts; the user cannot disable $\overline{NMI}$. After recognizing an active input on $\overline{NMI}$, the CPU stops before the next instruction, pushes the PC onto the stack, and jumps to address X'0066, where the user's interrupt service routine is located (i.e., restart to memory location X'0066). $\overline{NMI}$ is intended for interrupts requiring immediate attention, such as power-down, control panel, etc.

$\overline{RSTA}$, $\overline{RSTB}$ and $\overline{RSTC}$ are restart inputs, which, if enabled, execute a restart to memory location X'003C, X'0034, and X'002C, respectively. Note that the CPU response to the $\overline{NMI}$ and $\overline{RST}$ ($\overline{A}$, $\overline{B}$, $\overline{C}$) request input is basically identical, except for the restored memory location. Unlike $\overline{NMI}$, however, restart request inputs must be enabled.

Figure 16 illustrates $\overline{NMI}$ and $\overline{RST}$ interrupt machine cycles. M1 cycle will be a dummy opcode fetch cycle followed by M2 and M3 which are stack push operations. The following instruction then starts from the interrupts restart location.

Note: $\overline{RD}$ does *not* go low during this dummy opcode fetch. A unique indication of INTA can be decoded using 2 ALEs and $\overline{RD}$.

9.0 Timing and Control (Continued)



TL/C/5171-24

Note 1: This is the only machine cycle that does not have an $\overline{RD}$, $\overline{WR}$, or $\overline{INTA}$ strobe but will accept a wait strobe.

FIGURE 16. Non-Maskable and Restart Interrupt Machine Cycle

The NSC800 also provides one more general purpose interrupt request input, $\overline{INTR}$. When enabled, the CPU responds to $\overline{INTR}$ in one of the three modes defined by instruction IM0, IM1, and IM2 for modes 0, 1, and 2, respectively. Following reset, the CPU automatically enables mode 0.

Interrupt ($\overline{INTR}$) Mode 0: The CPU responds to an interrupt request by providing an INTA (interrupt acknowledge) strobe, which can be used to gate an instruction from a peripheral onto the data bus. The CPU inserts two wait states during the first $\overline{INTA}$ cycle to allow the interrupting device (or its controller) ample time to gate the instruction and determine external priorities (*Figure 18*). This can be any instruction from one to four bytes. The most popular instruction is one-byte call (restart instruction) or a three-byte call (CALL NN instruction). If it is a three-byte call, the CPU issues a total of three $\overline{INTA}$ strobes. The last two (which do not include wait states) read NN.

Note: If the instruction stored in the ICU doesn't require the PC to be pushed onto the stack (eq. JP nn), then the PC will not be pushed.

Interrupt ($\overline{INTR}$) Mode 1: Similar to restart interrupts except the restart location is X'0038 (*Figure 18*).

Interrupt ($\overline{INTR}$) Mode 2: With this mode, the programmer maintains a table that contains the 16-bit starting address of every interrupt service routine. This table can be located anywhere in memory. When the CPU accepts a Mode 2 interrupt (*Figure 17*), it forms a 16-bit pointer to obtain the desired interrupt service routine starting address from the table. The upper 8 bits of this pointer are from the contents of the I register. The lower 8 bits of the pointer are supplied by the interrupting device with the LSB forced to zero. The programmer must load the interrupt vector prior to the interrupt occurring. The CPU uses the pointer to get the two adjacent bytes from the interrupt service routine starting address table to complete 16-bit service routine starting address. The first byte of each entry in the table is the least significant (low-order) portion of the address. The programmer must obviously fill this table with the desired addresses before any interrupts are to be accepted.

Note that the programmer can change this table at any time to allow peripherals to be serviced by different service routines. Once the interrupting device supplies the lower portion of the pointer, the CPU automatically pushes the program counter onto the stack, obtains the starting address from the table and does a jump to this address.

The interrupts have fixed priorities built into the NSC800 as:

$\overline{NMI}$	0066	(Highest Priority)
$\overline{RSTA}$	003C	
$\overline{RSTB}$	0034	
$\overline{RSTC}$	002C	
$\overline{INTR}$	0038	(Lowest Priority)

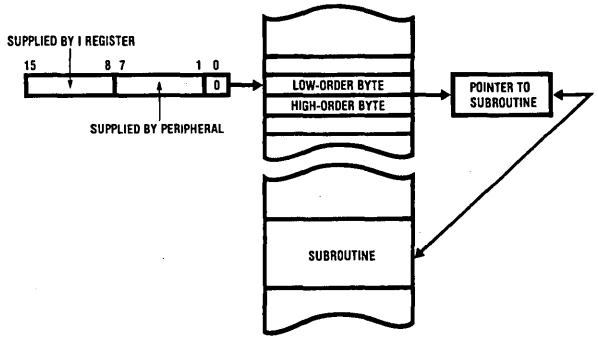
Interrupt Enable, Interrupt Disable. The NSC800 has two types of interrupt inputs, a non-maskable interrupt and four software maskable interrupts. The non-maskable interrupt ($\overline{NMI}$) cannot be disabled by the programmer and will be accepted whenever a peripheral device requests an interrupt. The $\overline{NMI}$ is usually reserved for important functions that must be serviced when they occur, such as imminent power failure. The programmer can selectively enable or disable maskable interrupts ($\overline{INT}$, $\overline{RSTA}$, $\overline{RSTB}$ and $\overline{RSTC}$). This selectivity allows the programmer to disable the maskable interrupts during periods when timing constraints don't allow program interruption.

There are two interrupt enable flip-flops (IFF_1 and IFF_2) on the NSC800. Two instructions control these flip-flops. Enable Interrupt (EI) and Disable Interrupt (DI). The state of IFF_1 determines the enabling or disabling of the maskable interrupts, while IFF_2 is used as a temporary storage location for the state of IFF_1.

9.0 Timing and Control (Continued)

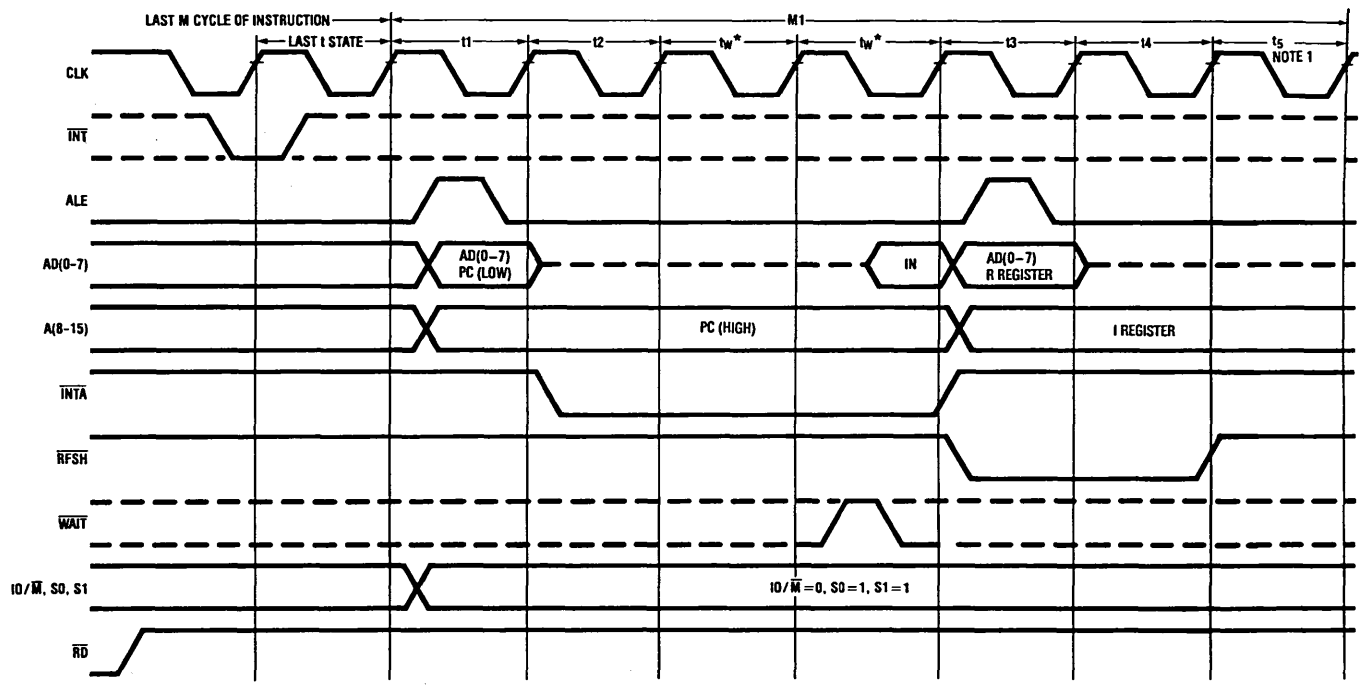
A reset to the CPU will force both IFF_1 and IFF_2 to the reset state disabling maskable interrupts. They can be enabled by an EI instruction at any time by the programmer. When an EI instruction is executed, any pending interrupt requests will not be accepted until after the instruction following EI has been executed. This single instruction delay is necessary in situations where the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instruction sets both IFF_1 and IFF_2 to the enable state. When the CPU accepts an interrupt, both IFF_1 and IFF_2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all the previous cases, IFF_1 and IFF_2 are always equal.

The function of IFF_2 is to retain the status of IFF_1 when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF_1 is reset to prevent further interrupts until reenabled by the programmer. Thus, after a non-maskable interrupt has been accepted, maskable interrupts are disabled but the previous state of IFF_1 is saved by IFF_2



TL/C/5171–27

FIGURE 17. Interrupt Mode 2



TL/C/5171-25

*t_W is the CPU generated WAIT state in response to an interrupt request.

Note 1: t5 will only occur in mode 1 and mode 2. During t5 the stack pointer is decremented.

Note 2: A jump to the appropriate address occurs here in mode 1 and mode 2. The CPU continues gathering data from the interrupting peripheral in mode 0 for a total of 2–4 machine cycles. In mode 0 cycles M2–M4 have only 1 wait state.

FIGURE 18. Interrupt Acknowledge Machine Cycle

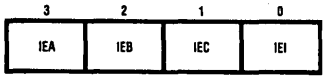
9.0 Timing and Control (Continued)

so that the complete state of the CPU just prior to the non-maskable interrupt may be restored. The method of restoring the status of IFF_1 is through the execution of a Return Non-Maskable Interrupt (RETN) instruction. Since this instruction indicates that the non-maskable interrupt service routine is completed, the contents of IFF_2 are now copied back into IFF_1, so that the status of IFF_1 just prior to the acceptance of the non-maskable interrupt will be automatically restored.

Figure 19 depicts the status of the flip flops during a sample series of interrupt instructions.

Interrupt Control Register. The interrupt control register (ICR) is a 4-bit, write only register that provides the programmer with a second level of maskable control over the four maskable interrupt inputs.

The ICR is internal to the NSC800 CPU, but is addressed through the I/O space at I/O address port X'BB. Each bit in the register controls a mask bit dedicated to each maskable interrupt, $\overline{RSTA}$, $\overline{RSTB}$, $\overline{RSTC}$ and $\overline{INTR}$. For an interrupt request to be accepted on any of these inputs, the corresponding mask bit in the ICR must be set (= 1) and IFF_1 and IFF_2 must be set. This provides the programmer with control over individual interrupt inputs rather than just a system wide enable or disable.



TL/C/5171–26

Bit	Name	Function
0	IEI	Interrupt Enable for $\overline{INTR}$
1	IEC	Interrupt Enable for $\overline{RSTC}$
2	IEB	Interrupt Enable for $\overline{RSTB}$
3	IEA	Interrupt Enable for $\overline{RSTA}$

For example: In order to enable $\overline{RSTB}$, CPU interrupts must be enabled and IEB must be set.

At reset, IEI bit is set and other mask bits IEA, IEB, IEC are cleared. This maintains the software compatibility between NSC800 and Z80A.

Execution of an I/O block move instruction will not affect the state of the interrupt control bits. The only two instructions that will modify this write only register are OUT (C), r and OUT (N), A.

Operation	IFF_1	IFF_2	Comment
Initialize	0	0	Interrupt Disabled
•			
•			
•			
EI	1	1	Interrupt Enabled after next instruction
•			
•			
•			
$\overline{INTR}$	0	0	Interrupt Disable and $\overline{INTR}$ Being Serviced
•			
•			
•			
EI	1	1	Interrupt Enabled after next instruction
RET	1	1	Interrupt Enabled
•			
•			
•			
$\overline{NMI}$	0	1	Interrupt Disabled
•			
•			
•			
RETN	1	1	Interrupt Enabled
•			
$\overline{INTR}$	0	0	Interrupt Disabled
•			
•			
•			
$\overline{NMI}$	0	0	Interrupt Disabled and $\overline{NMI}$ Being Serviced
•			
•			
•			
RETN	0	0	Interrupt Disabled and $\overline{INTR}$ Being Serviced
•			
•			
•			
EI	1	1	Interrupt Enabled after next instruction
RET	1	1	Interrupt Enabled
•			
•			
•			

FIGURE 19. IFF_1 and IFF_2 States Immediately after the Operation has been Completed

NSC800 SOFTWARE

10.0 Introduction

This chapter provides the reader with a detailed description of the NSC800 software. Each NSC800 instruction is described in terms of opcode, function, flags affected, timing, and addressing mode.

11.0 Addressing Modes

The following sections describe the addressing modes supported by the NSC800. Note that particular addressing modes are often restricted to certain types of instructions. Examples of instructions used in the particular addressing modes follow each mode description.

The 10 addressing modes and 158 instructions provide a flexible and powerful instruction set.

11.1 REGISTER

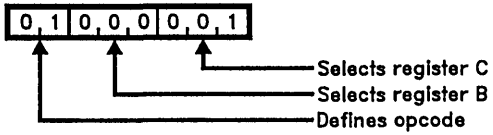
The most basic addressing mode is that which addresses data in the various CPU registers. In these cases, bits in the opcode select specific registers that are to be addressed by the instruction.

Example:

Instruction: Load register B from register C

Mnemonic: LD B,C

Opcode:



TL/C/5171-50

In this instruction, both the B and C registers are addressed by opcode bits.

11.2 IMPLIED

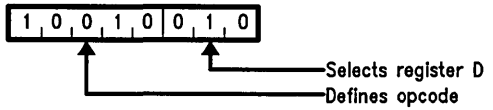
The implied addressing mode is an extension to the register addressing mode. In this mode, a specific register, the accumulator, is used in the execution of the instruction. In particular, arithmetic operations employ implied addressing, since the A register is assumed to be the destination register for the result without being specifically referenced in the opcode.

Example:

Instruction: Subtract the contents of register D from the Accumulator (A register)

Mnemonic: SUB D

Opcode:



TL/C/5171-51

In this instruction, the D register is addressed with register addressing, while the use of the A register is implied by the opcode.

11.3 IMMEDIATE

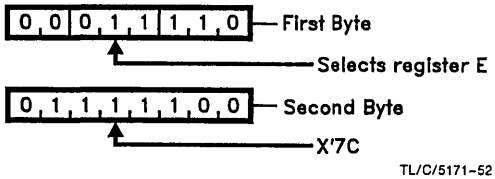
The most straightforward way of introducing data to the CPU registers is via immediate addressing, where the data is contained in an additional byte of multi-byte instructions.

Example:

Instruction: Load the E register with the constant value X'7C.

Mnemonic: LD E,X'7C

Opcode:



TL/C/5171-52

In this instruction, the E register is addressed with register addressing, while the constant X'7C is immediate data in the second byte of the instruction.

11.4 IMMEDIATE EXTENDED

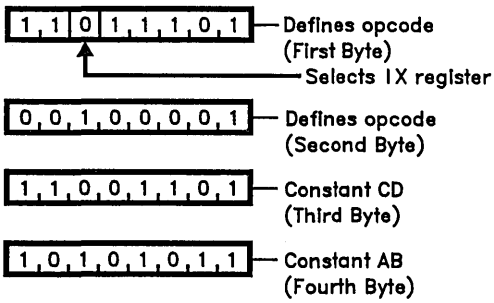
As immediate addressing allows 8 bits of data to be supplied by the operand, immediate extended addressing allows 16 bits of data to be supplied by the operand. These are in two additional bytes of the instruction.

Example:

Instruction: Load the 16-bit IX register with the constant value X'ABCD.

Mnemonic: LD IX,X'ABCD

Opcode:



TL/C/5171-53

In this instruction, register addressing selects the IX register, while the 16-bit quanity X'ABCD is immediate data supplied as immediate extended format.

11.0 Addressing Modes (Continued)

11.5 DIRECT ADDRESSING

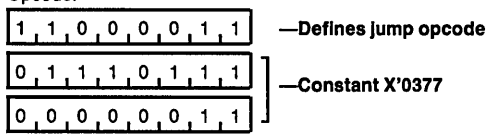
Direct addressing is the most straightforward way of addressing supplies a location in the memory space. Direct addressing, 16-bits of memory address information in two bytes of data as part of the instruction. The memory address could be either data, source of destination, or a location for program execution, as in program control instructions.

Example:

Instruction: Jump to location X'0377

Mnemonic: JP X'0377

Opcode:



This instruction loads the Program Counter (PC) is loaded with the constant in the second and third bytes of the instruction. The program counter contents are transferred via direct addressing.

11.6 REGISTER INDIRECT

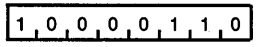
Next to direct addressing, register indirect addressing provides the second most straightforward means of addressing memory. In register indirect addressing, a specified register pair contains the address of the desired memory location. The instruction references the register pair and the register contents define the memory location of the operand.

Example:

Instruction: Add the contents of memory location X'0254 to the A register. The HL register contains X'0254.

Mnemonic: ADD A,(HL)

Opcode



This instruction uses implied addressing of the A and HL registers and register indirect addressing to access the data pointed to by the HL register.

11.7 INDEXED

The most flexible mode of memory addressing is the indexed mode. This is similar to the register indirect mode of addressing because one of the two index registers (IX or IY) contains the base memory address. In addition, a byte of data included in the instruction acts as a displacement to the address in the index register.

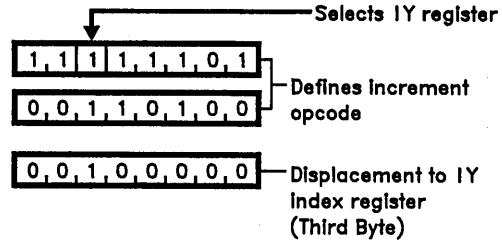
Indexed addressing is particularly useful in dealing with lists of data.

Example:

Instruction: Increment the data in memory location X'1020. The IY register contains X'1000.

Mnemonic: INC (IY+X'20)

Opcode:



TL/C/5171-54

The indexed addressing mode uses the contents of index registers IX or IY along with the displacement to form a pointer to memory.

11.8 RELATIVE

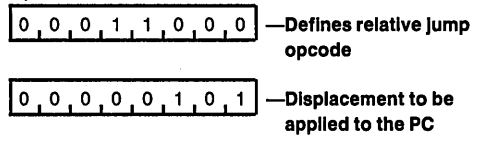
Certain instructions allow memory locations to be addressed as a position relative to the PC register. These instructions allow jumps to memory locations which are offsets around the program counter. The offset, together with the current program location, is determined through a displacement byte included in the instruction. The formation of this displacement byte is explained more fully in the "Instructions Set" section.

Example:

Instruction: Jump to a memory location 7 bytes beyond the current location.

Mnemonic: JR $+7

Opcode:



The program will continue at a location seven locations past the current PC.

11.0 Addressing Modes (Continued)

11.9 MODIFIED PAGE ZERO

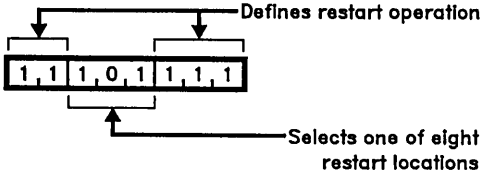
A subset of NSC800 instructions (the Restart instructions) provides a code-efficient single-byte instruction that allows CALLs to be performed to any one of eight dedicated locations in page zero (locations X'0000 to X'00FF). Normally, a CALL is a 3-byte instruction employing direct memory addressing.

Example:

Instruction: Perform a restart call to location X'0028.

Mnemonic: RST X'28

Opcode:



TL/C/5171–55

p	00H	08H	10H	18H	20H	28H	30H	38H
t	000	001	010	011	100	101	110	111

Program execution continues at location X'0028 after execution of a single-byte call employing modified page zero addressing.

11.10 BIT

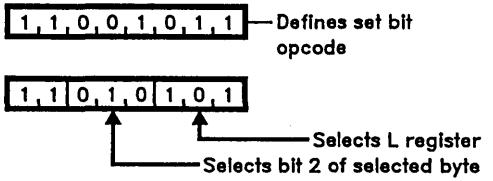
The NSC800 allows setting, resetting, and testing of individual bits in registers and memory data bytes.

Example:

Operation: Set bit 2 in the L register

Mnemonic: SET 2,L

Opcode:



TL/C/5171–56

Bit addressing allows the selection of bit 2 in the L register selected by register addressing.

12.0 Instruction Set

This section details the entire NSC800 instruction set in terms of

- Opcode
- Instruction
- Function
- Timing
- Addressing Mode

The instructions are grouped in order under the following functional headings:

- 8-Bit Loads
- 16-Bit Loads
- 8-Bit Arithmetic
- 16-Bit Arithmetic
- Bit Set, Reset, and Test
- Rotate and Shift
- Exchanges
- Memory Block Moves and Searches
- Input/Output
- CPU Control
- Program Control

12.1 Instruction Set Index

Alphabetical Assembly Mnemonic	Operation
ADC A,m_1	Add, with carry, memory location contents to Accumulator
ADC A,n	Add, with carry, immediate data n to Accumulator
ADC A,r	Add, with carry, register r contents to Accumulator
ADC HL,pp	Add, with carry, register pair pp to HL
ADD A,m_1	Add memory location contents to Accumulator
ADD A,n	Add immediate data n to Accumulator
ADD A,r	Add register r contents to Accumulator
ADD HL,pp	Add register pair pp to HL
ADD IX,pp	Add register pair pp to IX
ADD IY,pp	Add register pair pp to IY
ADD ss,pp	Add register pair pp to contents of register pair ss
AND m_1	Logical 'AND' memory contents to Accumulator
AND n	Logical 'AND' immediate data to Accumulator
AND r	Logical 'AND' register r contents to Accumulator
BIT b,m_1	Test bit b of location m_1
BIT b,r	Test bit b of register r
CALL cc,nn	Call subroutine at location nn if condition cc is true
CALL nn	Unconditional call to subroutine at location nn
CCF	Complement carry flag
CP m_1	Compare memory contents with Accumulator
CP n	Compare immediate data n with Accumulator
CP r	Compare register r to contents with Accumulator
CPD	Compare location (HL) and Accumulator, decrement HL and BC
CPDR	Compare location (HL) and Accumulator, decrement HL and BC; repeat until BC = 0
CPI	Compare location (HL) and Accumulator, increment HL, decrement BC
CPIR	Compare location (HL) and Accumulator, increment HL, decrement BC; repeat until BC = 0
CPL	Complement Accumulator (1's complement)
DAA	Decimal adjust Accumulator
DEC m_1	Decrement data in memory location m_1
DEC r	Decrement register r contents
DEC rr	Decrement register pair rr contents

12.1 Instruction Set Index (Continued)

Alphabetical Assembly Mnemonic	Operation
DI	Disable interrupts
DJNZ,d	Decrement B and jump relative B ≠ 0
EI	Enable interrupts
EX (SP),ss	Exchange the location (SP) with register ss
EX AF,A'F'	Exchange the contents of AF and A'F'
EX DE,HL	Exchange the contents of DE and HL
EXX	Exchange the contents of BC, DE and HL with the contents of B'C, D'E' and H'L', respectively
HALT	Halt (wait for interrupt or reset)
IM 0	Set interrupt mode 0
IM 1	Set interrupt mode 1
IM 2	Set interrupt mode 2
IN A,(n)	Load Accumulator with input from device (n)
IN r,(C)	Load register r with input from device (C)
INC m_1	Increment data in memory location m_1
INC r	Increment register r
INC rr	Increment contents of register pair rr
IND	Load location (HL) with input from port (C), decrement HL and B
INDR	Load location (HL) with input from port (C), decrement HL and B; repeat until B = 0
INI	Load location (HL) with input from port (C), increment HL, decrement B
INIR	Load location (HL) with input from port (C), increment HL, decrement B; repeat until B = 0
JP cc,nn	Jump to location nn, if condition cc is true
JP nn	Unconditional jump to location nn
JP (ss)	Unconditional jump to location (ss)
JR d	Unconditional jump relative to PC + d
JR kk,d	Jump relative to PC + d, if kk true
LD A,I	Load Accumulator with register I contents
LD A,m_2	Load Accumulator from location m_2
LD A,R	Load Accumulator with register R contents
LD I,A	Load register I with Accumulator contents
LD m_1,n	Load memory with immediate data n
LD m_1,r	Load memory from register r
LD m_2,A	Load memory from Accumulator
LD (nn),rr	Load memory location nn with register pair rr
LD r,m_1	Load register r from memory
LD r,n	Load register with immediate data n
LD R,A	Load register R from Accumulator
LD r_d,r_s	Load destination register r_d from source register r_s
LD rr,(nn)	Load register pair rr from memory location nn
LD rr,nn	Load register pair rr with immediate data nn
LD SP,ss	Load SP from register pair ss
LDD	Load location (DE) with location (HL), decrement DE, HL and BC
LDDR	Load location (DE) with location (HL), decrement DE, HL and BC; repeat until BC = 0
LDI	Load location (DE) with location (HL), increment DE and HL, decrement BC
LDIR	Load location (DE) with location (HL), increment DE and HL, decrement BC; repeat until BC = 0
NEG	Negate Accumulator (2's complement)
NOP	No operation

12.1 Instruction Set Index (Continued)

Alphabetical Assembly Mnemonic	Operation
OR m_1	Logical 'OR' of memory location contents and accumulator
OR n	Logical 'OR' of immediate data n and Accumulator
OR r	Logical 'OR' of register r and Accumulator
OTDR	Load output port (C) with location (HL), decrement HL and B; repeat until B = 0
OTIR	Load output port (C) with location (HL), increment HL, decrement B; repeat until B = 0
OUT (C),r	Load output port (C) with register r
OUT (n),A	Load output port (n) with Accumulator
OUTD	Load output port (C) with location (HL), decrement HL and B
OUTI	Load output port (C) with location (HL), increment HL, decrement B
POP qq	Load register pair qq with top of stack
PUSH qq	Load top of stack with register pair qq
RES b,m_1	Reset bit b of memory location m_1
RES b,r	Reset bit b of register r
RET	Unconditional return from subroutine
RET cc	Return from subroutine, if cc true
RETI	Unconditional return from interrupt
RETN	Unconditional return from non-maskable interrupt
RL m_1	Rotate memory contents left through carry
RL r	Rotate register r left through carry
RLA	Rotate Accumulator left through carry
RLC m_1	Rotate memory contents left circular
RLC r	Rotate register r left circular
RLCA	Rotate Accumulator left circular
RLD	Rotate digit left and right between Accumulator and memory (HL)
RR m_1	Rotate memory contents right through carry
RR r	Rotate register r right through carry
RRA	Rotate Accumulator right through carry
RRC m_1	Rotate memory contents right circular
RRC r	Rotate register r right circular
RRCA	Rotate Accumulator right circular
RRD	Rotate digit right and left between Accumulator and memory (HL)
RST P	Restart to location P
SBC A,m_1	Subtract, with carry, memory contents from Accumulator
SBC A,n	Subtract, with carry, immediate data n from Accumulator
SBC A,r	Subtract, with carry, register r from Accumulator
SBC HL,pp	Subtract, with carry, register pair pp from HL
SCF	Set carry flag
SET b,m_1	Set bit b in memory location m_1 contents
SET b,r	Set bit b in register r
SLA m_1	Shift memory contents left, arithmetic
SLA r	Shift register r left, arithmetic
SRA m_1	Shift memory contents right, arithmetic
SRA r	Shift register r right, arithmetic
SRL m_1	Shift memory contents right, logical
SRL r	Shift register r right, logical
SUB m_1	Subtract memory contents from Accumulator
SUB n	Subtract immediate data n from Accumulator
SUB r	Subtract register r from Accumulator
XOR m_1	Exclusive 'OR' memory contents and Accumulator
XOR n	Exclusive 'OR' immediate data n and Accumulator
XOR r	Exclusive 'OR' register r and Accumulator

12.0 Instruction Set (Continued)

12.2 INSTRUCTION SET MNEMONIC NOTATION

In the following instruction set listing, the notations used are shown below.

b: Designates one bit in a register or memory location. Bit address mode uses this indicator.

cc: Designates condition codes used in conditional Jumps, Calls, and Return instruction; may be:

NZ = Non-Zero (Z flag = 0)

Z = Zero (Z flag = 1)

NC = Non-Carry (C flag = 0)

C = Carry (C flag = 1)

PO = Parity Odd or No Overflow (P/V = 0)

PE = Parity Even or Overflow (P/V = 1)

P = Positive (S = 0)

M = Negative (S = 1)

d: Designates an 8-bit signed complement displacement. Relative or indexed address modes use this indicator.

kk: Subset of cc condition codes used in conjunction with conditional relative jumps; may be NZ, Z, NC or C.

m_1: Designates (HL), (IX + d) or (IY + d). Register indirect or indexed address modes use this indicator.

m_2: Designates (BC), (DE) or (nn). Register indirect or direct address modes use this indicator.

n: Any 8-bit binary number.

nn: Any 16-bit binary number.

p: Designates restart vectors and may be the hex values 0, 8, 10, 18, 20, 28, 30 or 38. Restart instructions employing the modified page zero addressing mode use this indicator.

pp: Designates the BC, DE, SP or any 16-bit register used as a destination operand in 16-bit arithmetic operations employing the register address mode.

qq: Designates BC, DE, HL, A, F, IX, or IY during operations employing register address mode.

r: Designates A, B, C, D, E, H or L. Register addressing modes use this indicator.

rr: Designates BC, DE, HL, SP, IX or IY. Register addressing modes use this indicator.

ss: Designates HL, IX or IY. Register addressing modes use this indicator.

X_L: Subscript L indicates the lower-order byte of a 16-bit register.

X_H: Subscript H indicates the high-order byte of a 16-bit register.

( ): parentheses indicate the contents are considered a pointer address to a memory or I/O location.

12.3 ASSEMBLED OBJECT CODE NOTATION

Register Codes:

r	Register	rp	Register	rs	Register
000	B	00	BC	00	BC
001	C	01	DE	01	DE
010	D	10	HL	10	HL
011	E	11	SP	11	AF
100	H	**pp**	**Register**	**qq**	**Register**
101	L	00	BC	00	BC
111	A	01	DE	01	DE
		10	IX	10	HL
		11	SP	11	AF

Conditions Codes:

cc	Mnemonic	True Flag Condition
000	NZ	Z = 0
001	Z	Z = 1
010	NC	C = 0
011	C	C = 1
100	PO	P/V = 0
101	PE	P/V = 1
110	P	S = 0
111	M	S = 1
kk	**Mnemonic**	**True Flag Condition**
00	NZ	Z = 0
01	Z	Z = 1
10	NC	C = 0
11	C	C = 1

Restart Addresses:

t	T
000	X'00
001	X'08
010	X'10
011	X'18
100	X'20
101	X'28
110	X'30
111	X'38

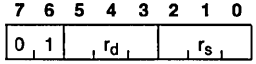
12.4 8-Bit Loads

REGISTER TO REGISTER

LD r_d, r_s

Load register r_d with r_s:

$r_d \leftarrow r_s$ No flags affected



Timing: M cycles — 1
T states — 4

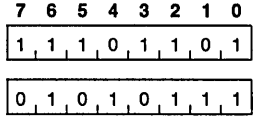
Addressing Mode: Register

LD A, I

Load Accumulator with the contents of the I register.

$A \leftarrow I$

S: Set if negative result
Z: Set if zero result
H: Reset
P/V: Set according to IFF_2 (zero if interrupt occurs during operation)
N: Reset
C: Not affected



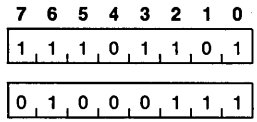
Timing: M cycles — 2
T states — 9 (4, 5)

Addressing Mode: Register

LD I, A

Load Interrupt vector register (I) with the contents of A.

$I \leftarrow A$ No flags affected



Timing: M cycles — 2
T states — 9 (4, 5)

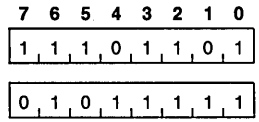
Addressing Mode: Register

LD A, R

Load Accumulator with contents of R register.

$A \leftarrow R$

S: Set if negative result
Z: Set if zero result
H: Reset
P/V: Set according to IFF_2 (zero if interrupt occurs during operation)
N: Reset
C: Not affected



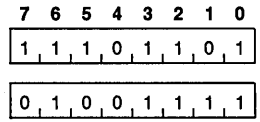
Timing: M cycles — 2
T states — 9 (4, 5)

Addressing Mode: Register

LD R, A

Load Refresh register (R) with contents of the Accumulator.

$R \leftarrow A$ No flags affected



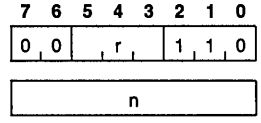
Timing: M cycles — 2
T states — 9 (4, 5)

Addressing Mode: Register

LD r, n

Load register r with immediate data n.

$r \leftarrow n$ No flags affected



Timing: M cycles — 2
T states — 7 (4, 3)

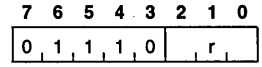
Addressing Mode: Source — Immediate
Destination — Register

REGISTER TO MEMORY

LD m_1, r

Load memory from reigster r.

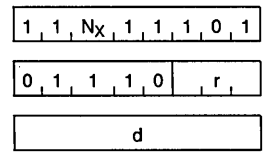
$m_1 \leftarrow r$ No flags affected



LD (HL), r

Timing: M cycles — 2
T states — 7 (4,3)

Addressing Mode: Source — Register
Destination — Register Indirect



LD (IX+d), r(for $N_X=0$)
LD (IY+d), r(for $N_X=1$)

Timing: M cycles — 2
T states — 19 (4, 4, 3, 5, 3)

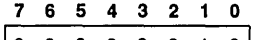
Addressing Mode: Source — Register
Destination — Indexed

12.4 8-Bit Loads (Continued)

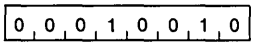
LD m_2, A

Load memory from the Accumulator.

$m_2 \leftarrow A$ No flags affected



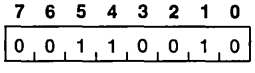
LD (BC), A



LD (DE), A

Timing: M cycles — 2
T states — 7 (4, 3)

Addressing Mode: Source — Register (Implied)
Destination — Register Indirect



LD (nn), A

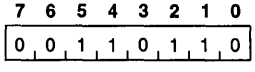
Timing: M cycles — 4
T states — 3 (4, 3, 3, 3)

Addressing Mode: Source — Register (Implied)
Destination — Direct

LD m_1, n

Load memory with immediate data.

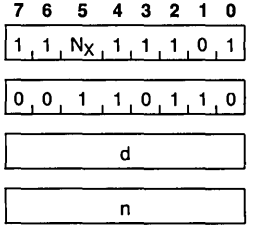
$m_1 \leftarrow n$ No flags affected



LD(HL), n

Timing: M cycles—3
T states—10 (4, 3, 3)

Addressing Mode: Source—Immediate
Destination—Register Indirect



LD (IX + d), n(for $N_X = 0$)
LD (IY + d), n(for $N_X = 1$)

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)

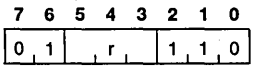
Addressing Mode: Source—Immediate
Destination—Indexed

MEMORY TO REGISTER

LD r, m_1

Load register r from memory location m_1.

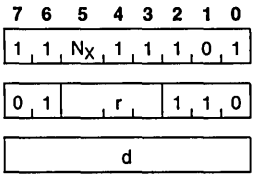
$r \leftarrow m_1$ No flags affected



LD R, (HL)

Timing: M cycles—2
T states—7 (4, 3)

Addressing Mode: Source—Register Indirect
Destination—Register



LD r, (IX + d) (for $N_X = 0$)
LD r, (IY + d) (for $N_X = 1$)

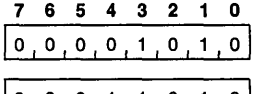
Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)

Addressing Mode: Source—Indexed
Destination—Register

LD A, m_2

Load the Accumulator from memory location m_2.

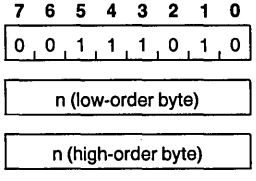
$A \leftarrow m_2$ No flags affected



LD A, (BC)
LD A, (DE)

Timing: M cycles—2
T states—7 (4, 3)

Addressing Mode: Source—Register Indirect
Destination—Register (Implied)



LD A, (nn)

Timing: M cycles—4
T states—13 (4, 3, 3, 3)

Addressing Mode: Source—Immediate Extended
Destination—Register (Implied)

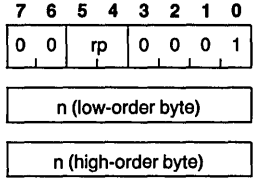
12.5 16-Bit Loads

REGISTER TO REGISTER

LD rr, nn

Load 16-bit register pair with immediate data.

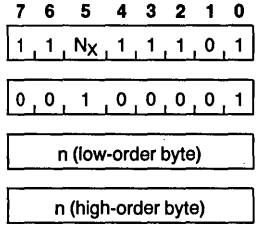
rr, ← nn — No flags affected



LD BC, nn
LD DE, nn
LD HL, nn
LD SP, nn

Timing: M cycles—3
T states—10 (4, 3, 3)

Addressing Mode: Source—Immediate Extended
Destination—Register



LD IX, nn (for $N_X = 0$)
LD IY, nn (for $N_X = 1$)

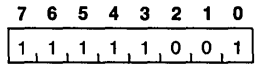
Timing: M cycles—4
T states—14 (4, 4, 3, 3)

Addressing Mode: Source—Immediate Extended
Destination—Register

LD SP, ss

Load the SP from 16-bit register ss.

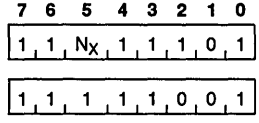
SP ← ss — No flags affected



LD SP, HL

Timing: M cycles—1
T states—6

Addressing Mode: Source—Register
Destination—Register (Implied)



LD SP, IX (for $N_X = 0$)
LD SP, IY (for $N_X = 1$)

Timing: M cycles—2
T states—10 (4, 6)

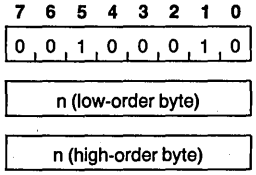
Addressing Mode: Source—Register
Destination—Register (Implied)

REGISTER TO MEMORY

LD (nn), rr

Load memory location nn with contents of 16-bit register, rr.

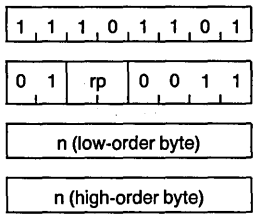
(nn) ← rr_L — No flags affected
(nn + 1) ← rr_H



LD (nn), HL
(note an alternate opcode below)

Timing: M cycles—5
T states—16 (4, 3, 3, 3, 3)

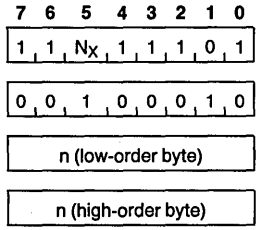
Addressing Mode: Source—Register
Destination—Direct



LD (nn), BC
LD (nn), DE
LD (nn), HL
LD (nn), SP

Timing: M cycles—6
T states—20 (4, 4, 3, 3, 3, 3)

Addressing Mode: Source—Register
Destination—Direct



LD (nn), IX (for $N_X = 0$)
LD (nn) IY (for $N_X = 1$)

Timing: M cycles—6
T states—20 (4, 4, 3, 3, 3, 3)

Addressing Mode: Source—Register
Destination—Direct

12.5 16-Bit Loads (Continued)

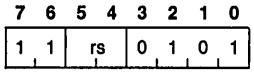
PUSH qq

Push the contents of register pair qq onto the memory stack.

$(SP - 1) \leftarrow qq_H$ No flags affected

$(SP - 2) \leftarrow qq_L$

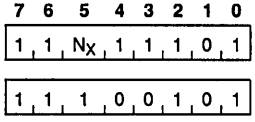
$SP \leftarrow SP - 2$



PUSH BC
PUSH DE
PUSH HL
PUSH AF

Timing: M cycles—3
T states—11 (5, 3, 3)

Addressing Mode: Source—Register
Destination—Register Indirect (Stack)



PUSH IX (for $N_X = 0$)
PUSH IY (for $N_X = 1$)

Timing: M cycles—3
T states—15 (4, 5, 3, 3)

Addressing Mode: Source—Register
Destination—Register Indirect (Stack)

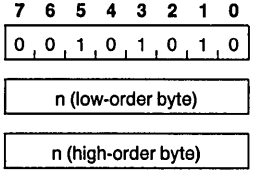
MEMORY TO REGISTER

LD rr, (nn)

Load 16-bit register from memory location nn.

$rr_L \leftarrow (nn)$ No flags affected

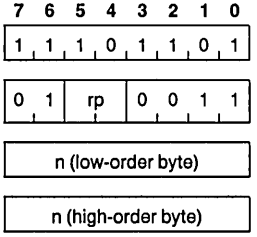
$rr_H \leftarrow (nn + 1)$



LD HL, (nn)
(note an alternate opcode below)

Timing: M cycles—5
T states—16 (4, 3, 3, 3, 3)

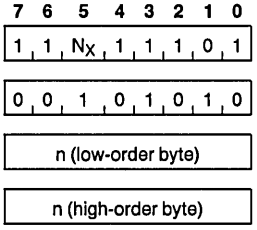
Addressing Mode: Source—Direct
Destination—Register



LD BC, (nn)
LD DE, (nn)
LD HL, (nn)
LD SP, (nn)

Timing: M cycles—6
T states—20 (4, 4, 3, 3, 3, 3)

Addressing Mode: Source—Direct
Destination—Register



LD IX, (nn)(for $N_X = 0$)
LD IY, (nn) (for $N_X = 1$)

Timing: M cycles—6
T states—20 (4, 4, 3, 3, 3, 3)

Addressing Mode: Source—Direct
Destination—Register

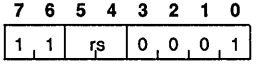
POP qq

Pop the contents of the memory stack to register qq.

$qq_L \leftarrow (SP)$ No flags affected

$qq_H \leftarrow (SP + 1)$

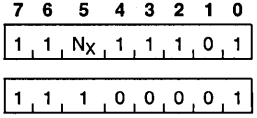
$SP \leftarrow SP + 2$



POP BC
POP DE
POP HL
POP AF

Timing: M cycles—3
T states—10 (4, 3, 3)

Addressing Mode: Source—Register Indirect (Stack)
Destination—Register



POP IX (for $N_X = 0$)
POP IY (for $N_X = 1$)

Timing: M cycles—4
T states—14 (4, 4, 3, 3)

Addressing Mode: Source—Register Indirect (Stack)
Destination—Register

12.6 8-Bit Arithmetic

REGISTER ADDRESSING ARITHMETIC

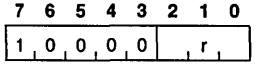
Op	C Before DAA	Hex Value In Upper Digit (Bits 7-4)	H Before DAA	Hex Value In Lower Digit (Bits 3-0)	Number Added To Byte	C After DAA
	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
ADD	0	A-F	0	0-9	60	1
ADC	0	9-F	0	A-F	66	1
INC	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
SUB	0	0-9	0	0-9	00	0
SBC	0	0-8	1	6-F	FA	0
DEC	1	7-F	0	0-9	A0	1
NEG	1	6-F	1	6-F	9A	1

ADD A, r

Add contents of register r to the Accumulator.

A ← A+r

S: Set if negative result
Z: Set if zero result
H: Set if carry from bit 3
P/V: Set according to overflow condition
N: Reset
C: Set if carry from bit 7



Timing: M cycles—1
T states—4

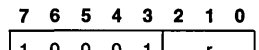
Addressing Mode: Source—Register
Destination—Implied

ADC A, r

Add contents of register r, plus the carry flag, to the Accumulator.

A ← A + r + CY

S: Set if negative result
Z: Set if zero result
H: Set if carry from bit 3
P/V: Set if result exceeds 2's complement range
N: Reset
C: Set if carry from bit 7



Timing: M cycles—1
T states—4

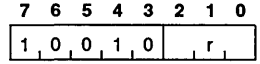
Addressing Mode: Source—Register
Destination—Implied

SUB r

Subtract the contents of register r from the Accumulator.

A ← A − r

S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow



Timing: M cycles—1
T states—4

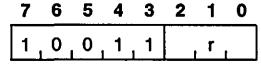
Addressing Mode: Source—Register
Destination—Implied

SBC A, r

Subtract contents of register r and the carry bit C from the Accumulator.

A ← A − r − CY

S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow



Timing: M cycles—1
T states—4

Addressing Mode: Source—Register
Destination—Implied

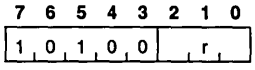
AND r

Logically AND the contents of the r register and the Accumulator.

A ← A ∧ r

S: Set if result is negative
Z: Set if result is zero
H: Set
P/V: Set if result parity is even
N: Reset
C: Reset

12.6 8-Bit Arithmetic (Continued)



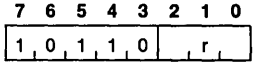
Timing: M cycles—1
T states—4
Addressing Mode: Source—Register
Destination—Implied

OR r

Logically OR the contents of the r register and the Accumulator.

A ← A ∨ r

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Reset



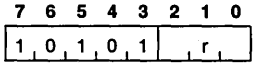
Timing: M cycles—1
T states—4
Addressing Mode: Source—Register
Destination—Implied

XOR r

Logically exclusively OR the contents of the r register with the Accumulator.

A ← A ⊕ r

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Reset



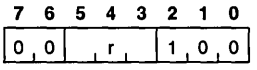
Timing: M cycles—1
T states—4
Addressing Mode: Source—Register
Destination—Implied

INC r

Increment register r.

r ← r + 1

S: Set if result is negative
Z: Set if result is zero
H: Set if carry from bit 3
P/V: Set only if r was X'7F before operation
N: Reset
C: N/A



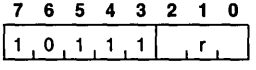
Timing: M cycles—1
T states—4
Addressing Mode: Source—Register
Destination—Register

CP r

Compare the contents of register r with the Accumulator and set the flags accordingly.

A − r

S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow



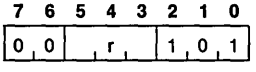
Timing: M cycles—1
T states—4
Addressing Mode: Source—Register
Destination—Implied

DEC r

Decrement the contents of register r.

r ← r − 1

S: Set if result is negative
Z: Set if result is zero
H: Set according to a borrow from bit 4
P/V: Set only if r was X'80 prior to operation
N: Set
C: N/A



Timing: M cycles—1
T states—4
Addressing Mode: Source—Register
Destination—Register

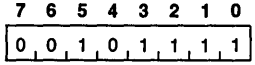
CPL

Complement the Accumulator (1's complement).

A ← $\overline{A}$

S: N/A
Z: N/A
H: Set
P/V: N/A
N: Set
C: N/A

12.6 8-Bit Arithmetic (Continued)



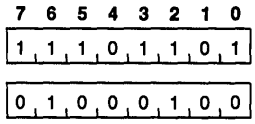
Timing: M cycles—1
T states—4
Addressing Mode: Implied

NEG

Negate the Accumulator (2's complement).

A ← 0 − A

S: Set if result is negative
Z: Set if result is zero
H: Set according to borrow from bit 4
P/V: Set only if Accumulator was X'80 prior to operation
N: Set
C: Set only if Accumulator was not X'00 prior to operation



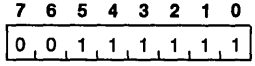
Timing: M cycles—2
T states—8 (4, 4)
Addressing Mode: Implied

CCF

Complement the carry flag.

CY ← $\overline{CY}$

S: N/A
Z: N/A
H: Previous carry
P/V: N/A
N: Reset
C: Complement of previous carry



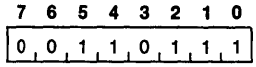
Timing: M cycles—1
T states—4
Addressing Mode: Implied

SCF

Set the carry flag.

CY ← 1

S: N/A
Z: N/A
H: Reset
P/V: N/A
N: Reset
C: Set



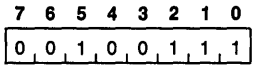
Timing: M cycles—1
T states—4
Addressing Mode: Implied

DAA

Adjust the Accumulator for BCD addition and subtraction operations. To be executed after BCD data has been operated upon the standard binary ADD, ADC, INC, SUB, SBC, DEC or NEG instructions (see "Register Addressing Arithmetic" table).

———

S: Set according to bit 7 of result
Z: Set if result is zero
H: Set according to instructions
P/V: Set according to parity of result
N: N/A
C: Set according to instructions



Timing: M cycles—1
T states—4
Addressing Mode: Implied

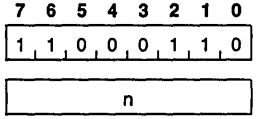
IMMEDIATELY ADDRESSED ARITHMETIC

ADD A, n

Add the immediate data n to the Accumulator.

A ← A + n

S: Set if result is negative
Z: Set if result is zero
H: Set if carry from bit 3
P/V: Set if result exceeds 8-bit 2's complement range
N: Reset
C: Set if carry from bit 7



Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Immediate
Destination—Implied

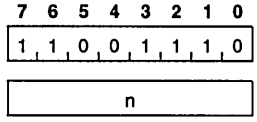
ADC A, n

Add, with carry, the immediate data n and the Accumulator.

A ← A + n + CY

S: Set if result is negative
Z: Set if result is zero
H: Set if carry from bit 3
P/V: Set if result exceeds 8-bit 2's complement range
N: Reset
C: Set according to carry from bit 7

12.6 8-Bit Arithmetic (Continued)



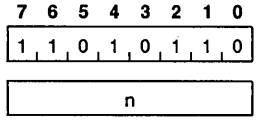
Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Immediate
Destination—Implied

SUB n

Subtract the immediate data n from the Accumulator.

A ← A − n

S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow condition



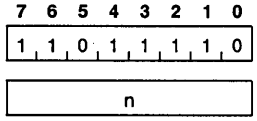
Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Immediate
Destination—Implied

SBC A, n

Subtract, with carry, the immediate data n from the Accumulator.

A ← A − n − CY

S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow condition



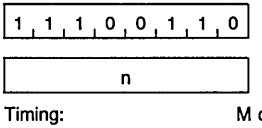
Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Immediate
Destination—Implied

AND n

The immediate data n is logically AND'ed to the Accumulator.

A ← A ∧ n

S: Set if result is negative
Z: Set if result is zero
H: Set
P/V: Set if result parity is even
N: Reset
C: Reset



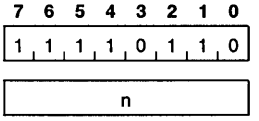
Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Immediate
Destination—Implied

OR n

The immediate data n is logically OR'ed to the contents of the Accumulator.

A ← A ∨ s

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Reset



Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Immediate
Destination—Implied

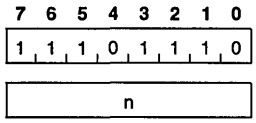
XOR n

The immediate data n is exclusively OR'ed with the Accumulator.

A ← A ⊕ n

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Reset

12.6 8-Bit Arithmetic (Continued)



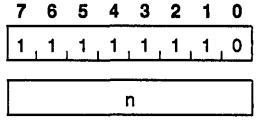
Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Immediate
Destination—Implied

CP n

Compare the immediate data n with the contents of the Accumulator via subtraction and return the appropriate flags. The contents of the Accumulator are not affected.

A − n

S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow condition



Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Immediate

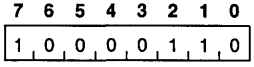
MEMORY ADDRESSED ARITHMETIC

ADD A, m1

Add the contents of the memory location m_1 to the Accumulator.

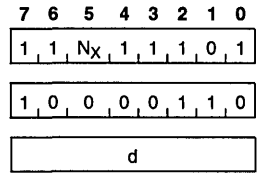
A ← A + m_1

S: Set if result is negative
Z: Set if result is zero
H: Set if carry from bit 3
P/V: Set if result exceeds 8-bit 2's complement range
N: Reset
C: Set according to carry from bit 7



ADD A, (HL)

Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Register Indirect
Destination—Implied



ADD A, (IX + d) (for N_X=0)
ADD A, (IY + d) (for N_X=1)

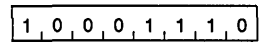
Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)
Addressing Mode: Source—Indexed
Destination—Implied

ADC A, m_1

Add the contents of the memory location m_1 plus the carry to the Accumulator.

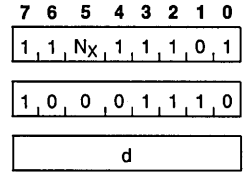
A ← A + m_1 + CY

S: Set if result is negative
Z: Set if result is zero
H: Set if carry from bit 3
P/V: Set if result exceeds 8-bit 2's complement range
N: Reset
C: Set according to carry from bit 7



ADC A, (HL)

Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Register Indirect
Destination—Implied



ADC A, (IX + d) (for N_X=0)
ADC A, (IY + d) (for N_X=1)

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)
Addressing Mode: Source—Indexed
Destination—Implied

SUB m_1

Subtract the contents of memory location m_1 from the Accumulator.

A ← A − m_1

S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow condition

12.6 8-Bit Arithmetic (Continued)

7	6	5	4	3	2	1	0	
1	0	0	1	0	1	1	0	SUB (HL)

Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Register Indirect
Destination—Implied

7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	SUB (IX + d) (for $N_X=0$) SUB (IY + d) (for $N_X=1$)
1	0	0	1	0	1	1	0	
d								

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)
Addressing Mode: Source—Indexed
Destination—Implied

SBC A, m_1

Subtract, with carry, the contents of memory location m_1 from the Accumulator.

$A \leftarrow A - m_1 - CY$

S: Set if result is negative
Z: Set if result is zero
H: Set if carry from bit 3
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow condition

7	6	5	4	3	2	1	0	
1	0	0	1	1	1	1	0	SBC A, (HL)

Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Register Indirect
Destination—Implied

7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	SBC A, (IX + d) (for $N_X=0$) SBC A, (IY + d) (for $N_X=1$)
1	0	0	1	1	1	1	0	
d								

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)
Addressing Mode: Source—Indexed
Destination—Implied

AND m_1

The data in memory location m_1 is logically AND'ed to the Accumulator.

$A \leftarrow A \wedge m_1$

S: Set if result is negative
Z: Set if result is zero
H: Set
P/V: Set if result parity is even
N: Reset
C: Reset

7	6	5	4	3	2	1	0	
1	0	1	0	0	1	1	0	AND (HL)

Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Register Indirect
Destination—Implied

7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	AND (IX + d) (for $N_X=0$) AND (IY + d) (for $N_X=1$)
1	0	1	0	0	1	1	0	
d								

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)
Addressing Mode: Source—Indexed
Destination—Implied

OR m_1

The data in memory location m_1 is logically OR'ed with the Accumulator.

$A \leftarrow A \vee m_1$

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Reset

7	6	5	4	3	2	1	0	
1	0	1	1	0	1	1	0	OR (HL)

Timing: M cycles—2
T states—7 (4, 3)
Addressing Mode: Source—Register Indexed
Destination—Implied

7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	OR (IX + d) (for $N_X=0$) OR (IY + d) (for $N_X=1$)
1	0	1	1	0	1	1	0	
d								

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)
Addressing Mode: Source—Indexed
Destination—Implied

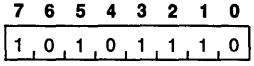
12.6 8-Bit Arithmetic (Continued)

XOR m_1

The data in memory location m_1 is exclusively OR'ed with the data in the Accumulator.

$A \leftarrow A \oplus m_1$

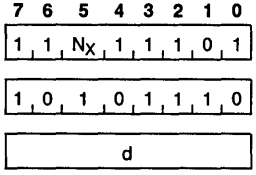
S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Reset



XOR (HL)

Timing: M cycles—2
T states—7 (4, 3)

Addressing Mode: Source—Register Indexed
Destination—Implied



XOR (IX + d) (for $N_X = 0$)
XOR (IY + d) (for $N_X = 1$)

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)

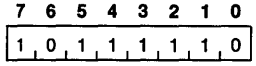
Addressing Mode: Source—Indexed
Destination—Implied

CP m_1

Compare the data in memory location m_1 with the data in the Accumulator via subtraction.

$A - m_1$

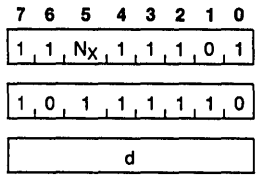
S: Set if result is negative
Z: Set if result is zero
H: Set if borrow from bit 4
P/V: Set if result exceeds 8-bit 2's complement range
N: Set
C: Set according to borrow condition



CP (HL)

Timing: M cycles—2
T states—7 (4, 3)

Addressing Mode: Source—Register Indirect
Destination—Implied



CP (IX + d) (for $N_X = 0$)
CP (IY + d) (for $N_X = 1$)

Timing: M cycles—5
T states—19 (4, 4, 3, 5, 3)

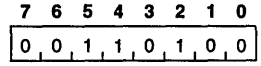
Addressing Mode: Source—Indexed
Destination—Implied

INC m_1

Increment data in memory location m_1.

$m_1 \leftarrow m_1 + 1$

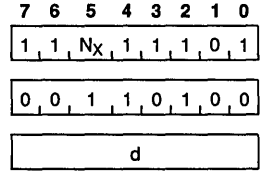
S: Set if result is negative
Z: Set if result is zero
H: Set according to carry from bit 3
P/V: Set if data was X'7F before operation
N: Reset
C: N/A



INC (HL)

Timing: M cycles—3
T states—11 (4, 4, 3)

Addressing Mode: Source—Register Indexed
Destination—Register Indexed



INC (IX + d) (for $N_X = 0$)
INC (IY + d) (for $N_X = 1$)

Timing: M cycles—6
T states—23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Source—Indexed
Destination—Indexed

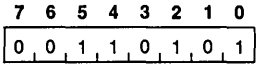
DEC m_1

Decrement data in memory location m_1.

$m_1 \leftarrow m_1 - 1$

S: Set if result is negative
Z: Set if result is zero
H: Set according to borrow from bit 4
P/V: Set only if m_1 was X'80 before operation
N: Set
C: N/A

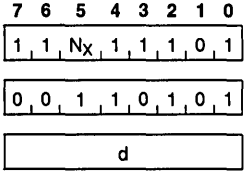
12.6 8-Bit Arithmetic (Continued)



DEC (HL)

Timing: M cycles — 3
T states — 11 (4, 4, 3)

Addressing Mode: Source — Register Indexed
Destination — Register Indexed



DEC (IX + d) (for $N_X = 0$)
DEC (IY + d) (for $N_X = 1$)

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Source — Indexed
Destination — Indexed

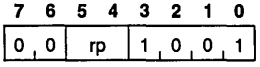
12.7 16-Bit Arithmetic

ADD ss, pp

Add the contents of the 16-bit register rp or pp to the contents of the 16-bit register ss.

ss ← ss + rp
or
ss ← ss + pp

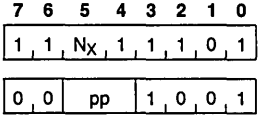
S: N/A
Z: N/A
H: Set if carry from bit 11
P/V: N/A
N: Reset
C: Set if carry from bit 15



ADD HL, rp

Timing: M cycles — 3
T states — 11 (4, 4, 3)

Addressing Mode: Source — Register
Destination — Register



ADD IX, pp (for $N_X = 0$)
ADD IY, pp (for $N_X = 1$)

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

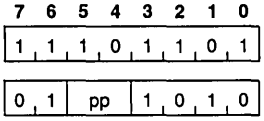
Addressing Mode: Source — Register
Destination — Register

ADC HL, pp

The contents of the 16-bit register pp are added, with the carry bit, to the HL register.

HL ← HL + pp + CY

S: Set if result is negative
Z: Set if result is zero
H: Set according to carry out of bit 11
P/V: Set if result exceeds 16-bit 2's complement range
N: Reset
C: Set if carry out of bit 15



Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

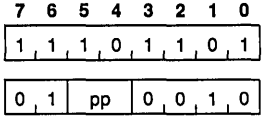
Addressing Mode: Source — Register
Destination — Register

SBC HL, pp

Subtract, with carry, the contents of the 16-bit pp register from the 16-bit HL register.

HL ← HL − pp − CY

S: Set if result is negative
Z: Set if result is zero
H: Set according to borrow from bit 12
P/V: Set if result exceeds 16-bit 2's complement range
N: Set
C: Set according to borrow condition



Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

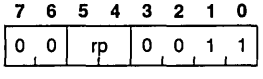
Addressing Mode: Source — Register
Destination — Register

INC rr

Increment the contents of the 16-bit register rr.

rr ← rr + 1

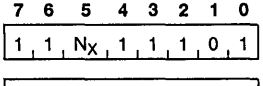
No flags affected



INC BC
INC DE
INC HL
INC SP

Timing: M cycles — 1
T states — 6

Addressing Mode: Register



INC IX (for $N_X = 0$)
INC IY (for $N_X = 1$)

Timing: M cycles — 2
T states — 10 (4, 6)

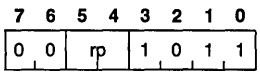
Addressing Mode: Register

12.7 16-Bit Arithmetic (Continued)

DEC rr

Decrement the contents of the 16-bit register rr.

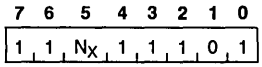
rr ← rr − 1 No flags affected



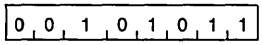
DEC BC
DEC DE
DEC HL
DEC SP

Timing: M cycles — 1
T states — 6

Addressing Mode: Register



DEC IX (for $N_X=0$)
DEC IY (for $N_X=1$)



Timing: M cycles — 2
T states — 10 (4, 6)

Addressing Mode: Register

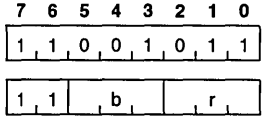
12.8 Bit Set, Reset, and Test

REGISTER

SET b, r

Bit b in register r is set.

R_b ← 1 No flags affected



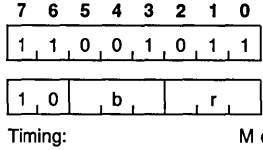
Timing: M cycles — 2
T states — 8 (4, 4)

Addressing Mode: Bit/Register

RES b, r

Bit b in register r is reset.

r_b ← 0 No flags affected



Timing: M cycles — 2
T states — 8 (4, 4)

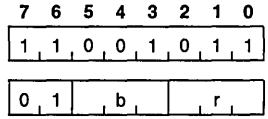
Addressing Mode: Bit/Register

BIT b, r

Bit b in register r is tested with the result put in the Z flag.

$Z \leftarrow \overline{r_b}$

S: Undefined
Z: Inverse of tested bit
H: Set
P/V: Undefined
N: Reset
C: N/A



Timing: M cycles — 2
T states — 8 (4, 4)

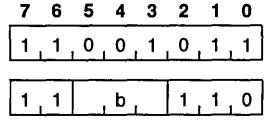
Addressing Mode: Bit/Register

MEMORY

SET b, m_1

Bit b in memory location m_1 is set.

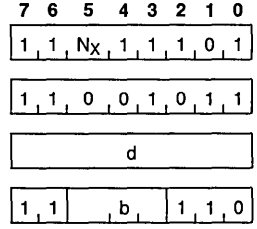
m_{1b} ← 1 No flags affected



SET b, (HL)

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Bit/Register Indirect



SET b, (IX+d) (for $N_X=0$)
SET b, (IY+d) (for $N_X=1$)

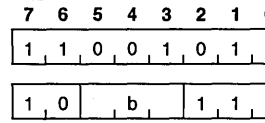
Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Bit/Indexed

RES b, m_1

Bit b in memory location m_1 is reset.

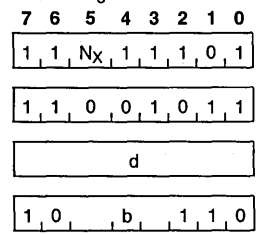
m_{1b} ← 0 No flags affected



RES b, (HL)

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Bit/Register Indirect



RES b, (IX+d) (for $N_X=0$)
RES b, (IY+d) (for $N_X=1$)

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Bit/Indexed

12.8 Bit Set, Reset, and Test (Continued)

BIT B, m_1

Bit b in memory location m_1 is tested via the Z flag.

$Z \leftarrow \overline{m_{1b}}$

S: Undefined
Z: Inverse of tested bit
H: Set
P/V: Undefined
N: Reset
C: N/A

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	BIT b, (HL)
0	1	b			1	1	0	

Timing: M cycles — 3
T states — 12 (4, 4, 4)

Addressing Mode: Bit/Register Indirect

7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	BIT b, (IX+d) (for $N_X=0$) / BIT b, (IY+d) (for $N_X=1$)
1	1	0	0	1	0	1	1	
d								
0	1	b			1	1	0	

Timing: M cycles — 5
T states — 20 (4, 4, 3, 5, 4)

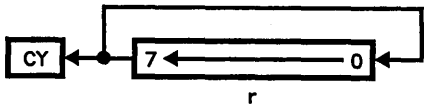
Addressing Mode: Bit/Indexed

12.9 Rotate and Shift

REGISTER

RLC r

Rotate register r left circular.



TL/C/5171-57

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 7 of r

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	RLC r
0	0	0	0	0		r		(Note alternate for A register below)

Timing: M cycles — 2
T states — 8 (4, 4)

Addressing Mode: Register

7	6	5	4	3	2	1	0	
0	0	0	0	0	1	1	1	RLCA

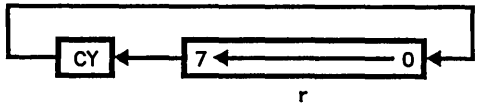
Timing: M cycles — 1
T states — 4

Addressing Mode: Implied

(Note RLCA does not affect S, Z, or P/V flags.)

RL r

Rotate register r left through carry.



TL/C/5171-58

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 7 of r

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	RL r
0	0	0	1	0		r		(Note alternate for A register below)

Timing: M cycles — 2
T states — 8 (4, 4)

Addressing Mode: Register

7	6	5	4	3	2	1	0	
0	0	0	1	0	1	1	1	RLA

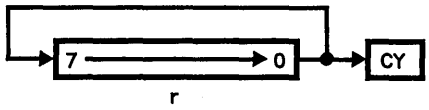
Timing: M cycles — 1
T states — 4

Addressing Mode: Implied

(Note RLA does not affect S, Z, or P/V flags.)

RRC r

Rotate register r right circular.



TL/C/5171-59

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of r

12.9 Rotate and Shift (Continued)

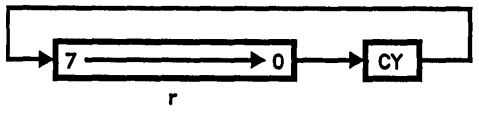
7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	RRC r
0	0	0	0	1	r			(Note alternate for A register below)

Timing: M cycles — 2
T states — 8 (4, 4)
Addressing Mode: Register

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	1	1	RRCA

Timing: M cycles — 1
T states — 4
Addressing Mode: Implied
(Note RRCA does not affect S, Z, or P/V flags.)

RR r
Rotate register r right through carry.



TL/C/5171-60

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of r

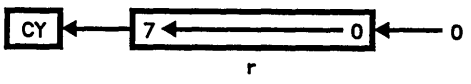
7	6	5	4	3	2	1	0	
1	1	0	0	1	0	0	1	RR r
0	0	0	1	1	r			(Note alternate for A register below)

Timing: M cycles — 2
T states — 8 (4, 4)
Addressing Mode: Register

7	6	5	4	3	2	1	0	
0	0	0	1	1	1	1	1	RRA

Timing: M cycles — 1
T states — 4
Addressing Mode: Implied
(Note RRA does not affect S, Z, or P/V flags.)

SLA r
Shift register r left arithmetic.



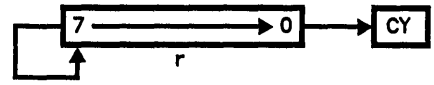
TL/C/5171-61

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 7 of r

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	
0	0	1	0	0	r			

Timing: M cycles — 2
T states — 8 (4, 4)
Addressing Mode: Register

SRA r
Shift register r right arithmetic.



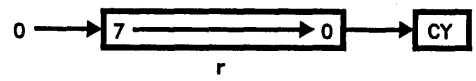
TL/C/5171-62

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of r

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	
0	0	1	0	1	r			

Timing: M cycles — 2
T states — 8 (4, 4)
Addressing Mode: Register

SRL r
Shift register r right logical.



TL/C/5171-63

S: Reset
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of r

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	
0	0	1	1	1	r			

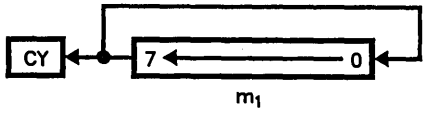
Timing: M cycles — 2
T states — 8 (4, 4)
Addressing Mode: Register

12.9 Rotate and Shift (Continued)

MEMORY

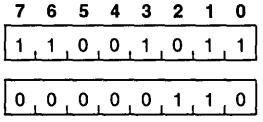
RLC m_1

Rotate date in memory location m_1 left circular.



TL/C/5171-64

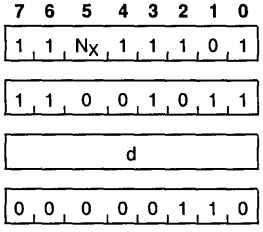
S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 7 of m_1



RLC (HL)

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Register indirect



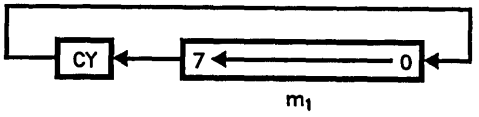
RLC (IX+d) (for $N_X=0$)
RLC (IY+d) (for $N_X=1$)

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Indexed

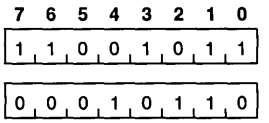
RL m_1

Rotate the data in memory location m_1 left though carry.



TL/C/5171-65

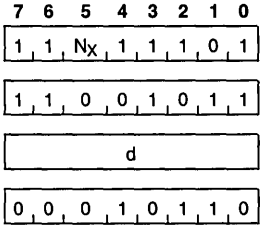
S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 7 of m_1



RL (HL)

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Register Indirect



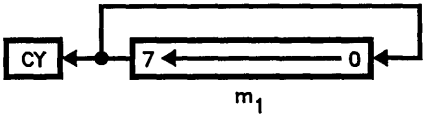
RL (IX+d) (for $N_X=0$)
RL (IY+d) (for NX=1)

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Indexed

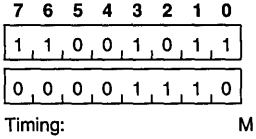
RRC m_1

Rotate the data in memory location m_1 right circular.



TL/C/5171-66

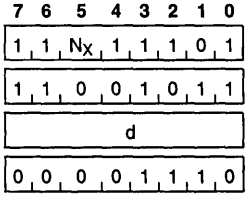
S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of m_1



RRC (HL)

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Register Indirect



RRC (IX + d) (for $N_X = 0$)
RRC (IY + d) (for $N_X = 1$)

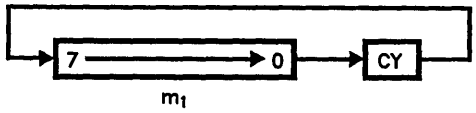
Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Indexed

12.9 Rotate and Shift (Continued)

RR m_1

Rotate the data in memory location m_1 right through the carry.



TL/C/5171-67

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of m_1

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	RR (HL)
0	0	0	1	1	1	1	0	

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Register Indirect

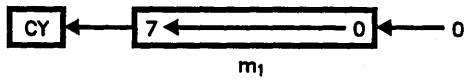
7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	RR (IX + d) (for $N_X = 0$)
1	1	0	0	1	0	1	1	RR (IY + d) (for $N_X = 1$)
d								
0	0	0	1	1	1	1	0	

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Indexed

SLA m_1

Shift the data in memory location m_1 left arithmetic.



TL/C/5171-68

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 7 of m_1

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	SLA (HL)
0	0	1	0	0	1	1	0	

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Register Indirect

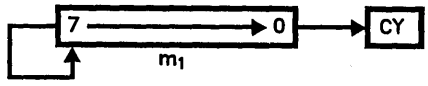
7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	SLA (IX + d) (for $N_X = 0$)
1	1	0	0	1	0	1	1	SLA (IY + d) (for $N_X = 1$)
d								
0	0	1	0	0	1	1	0	

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Indexed

SRA m_1

Shift the data in memory location m_1 right arithmetic.



TL/C/5171-69

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of m_1

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	SRA (HL)
0	0	1	0	1	1	1	0	

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)

Addressing Mode: Register Indirect

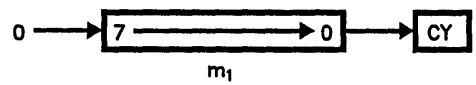
7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	SRA (IX + d) (for $N_X = 0$)
1	1	0	0	1	0	1	1	SRA (IY + d) (for $N_X = 1$)
d								
0	0	1	0	1	1	1	0	

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)

Addressing Mode: Indexed

SRL m_1

Shift right logical the data in memory location m_1.



TL/C/5171-70

S: Reset
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: Set according to bit 0 of m_1

12.9 Rotate and Shift (Continued)

7	6	5	4	3	2	1	0	
1	1	0	0	1	0	1	1	SRL (HL)
0	0	1	1	1	1	1	0	

Timing: M cycles — 4
T states — 15 (4, 4, 4, 3)
Addressing Mode: Register Indirect

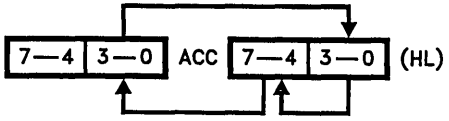
7	6	5	4	3	2	1	0	
1	1	N_X	1	1	1	0	1	SRL (IX + d) (for N_X = 0)
1	1	0	0	1	0	1	1	SRL (IY + d) (for N_X = 1)
d								
0	0	1	1	1	1	1	0	

Timing: M cycles — 6
T states — 23 (4, 4, 3, 5, 4, 3)
Addressing Mode: Indexed

REGISTER/MEMORY

RLD

Rotate digit left and right between the Accumulator and memory (HL).



TL/C/5171-71

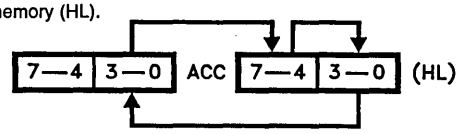
S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: N/A

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1
0	1	1	0	1	1	1	1

Timing: M cycles — 5
T states — 18 (4, 4, 3, 4, 3)
Addressing Mode: Implied/Register Indirect

RRD

Rotate digit right and left between the Accumulator and memory (HL).



TL/C/5171-72

S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: N/A

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1
0	1	1	0	0	1	1	1

Timing: M cycles — 5
T states — 18 (4, 4, 3, 4, 3)
Addressing Mode: Implied/Register Indirect

12.10 Exchanges

REGISTER/REGISTER

EX DE, HL

Exchange the contents of the 16-bit register pairs DE and HL.

DE ⟷ HL No flags affected

7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	1

Timing: M cycles — 1
T states — 4
Addressing Mode: Register

EX AF, A'F'

The contents of the Accumulator and flag register are exchanged with their corresponding alternate registers, that is A and F are exchanged with A' and F'.

A ⟷ A' No flags affected
F ⟷ F'

7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0

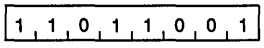
Timing: M cycles — 1
T states — 4
Addressing Mode: Register

12.10 Exchanges (Continued)

EXX

Exchange the contents of the BC, DE, and HL registers with their corresponding alternate register.

BC ⟷ B'C' — No flags affected
DE ⟷ D'E'
HL ⟷ H'L'



Timing: M cycles — 1
T states — 4

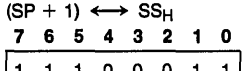
Addressing Mode: Implied

REGISTER/MEMORY

EX (SP), ss

Exchange the two bytes at the top of the external memory stack with the 16-bit register ss.

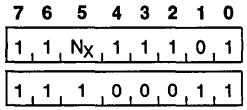
(SP) ⟷ SS_L — No flags affected
(SP + 1) ⟷ SS_H



EX (SP), HL

Timing: M cycles — 5
T states — 19 (4, 3, 4, 3, 5)

Addressing Mode: Register/Register Indirect



EX (SP), IX (for $N_X = 0$)
EX (SP),IY (for $N_X = 1$)

Timing: M cycles — 6
T states — 23 (4, 4, 3, 4, 3, 5)

Addressing Mode: Register/Register Indirect

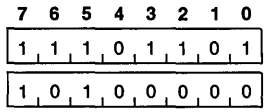
12.11 Memory Block Moves and Searches

SINGLE OPERATIONS

LDI

Move data from memory location (HL) to memory location (DE), increment memory pointers, and decrement byte counter BC.

(DE) ← (HL) — S: N/A
DE ← DE + 1 — Z: N/A
HL ← HL + 1 — H: Reset
BC ← BC − 1 — P/V: Set if BC −1 ≠0, otherwise reset
N: Reset
C: N/A



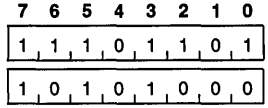
Timing: M cycles — 4
T states — 16 (4, 4, 3, 5)

Addressing Mode: Register Indirect

LDD

Move data from memory location (HL) to memory location (DE), and decrement memory pointer and byte counter BC.

(DE) ← (HL) — S: N/A
DE ← DE − 1 — Z: N/A
HL ← HL − 1 — H: Reset
BC ← BC − 1 — P/V: Set if BC −1 ≠0, otherwise reset
N: Reset
C: N/A



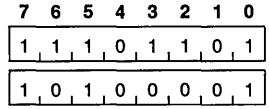
Timing: M cycles — 4
T states — 16 (4, 4, 3, 5)

Addressing Mode: Register Indirect

CPI

Compare data in memory location (HL) to the Accumulator, increment the memory pointer, and decrement the byte counter. The Z flag is set if the comparison is equal.

A − (HL) — S: Set if result of comparison subtract is negative
HL ← HL + 1 — Z: Set if result of comparison is zero
BC ← BC − 1 — H: Set according to borrow from bit 4
Z ← 1 if A = (HL) — P/V: Set if BC − 1≠ 0, otherwise reset
N: Set
C: N/A



Timing: M cycles — 4
T states — 16 (4, 4, 3, 5)

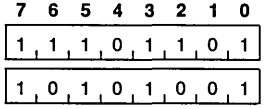
Addressing Mode: Register Indirect

CPD

Compare data in memory location (HL) to the Accumulator, and decrement the memory pointer and byte counter. The Z flag is set if the comparison is equal.

A − (HL) — S: Set if result is negative
HL ← HL − 1 — Z: Set if result of comparison is zero
BC ← BC − 1 — H: Set according to borrow from bit 4
Z ← 1 if A = (HL) — P/V: Set if BC − 1≠ 0, otherwise reset
N: Set
C: N/A

12.11 Memory Block Moves and Searches (Continued)



Timing: M cycles — 4
T states — 16 (4, 4, 3, 5)

Addressing Mode: Register Indirect

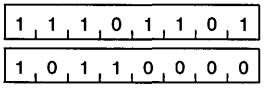
REPEAT OPERATIONS

LDIR

Move data from memory location (HL) to memory location (DE), increment memory pointers, decrement byte counter BC, and repeat until BC = 0.

(DE) ← (HL)
DE ← DE + 1
HL ← HL + 1
BC ← BC − 1
Repeat until
BC = 0

S: N/A
Z: N/A
H: Reset
P/V: Reset
N: Reset
C: N/A



Timing: For BC≠0 M cycles — 5
T states — 21 (4, 4, 3, 5, 5)
For BC=0 M cycles — 4
T states — 16 (4, 4, 3, 5)

Addressing Mode: Register Indirect

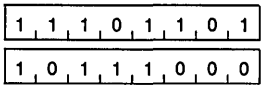
(Note that each repeat is accomplished by a decrement of the BC, so that refresh, etc. continues for each cycle.)

LDDR

Move data from memory location (HL) to memory location (DE), decrement memory pointers and byte counter BC, and repeat until BC = 0.

(DE) ← (HL)
DE ← DE − 1
HL ← HL − 1
BC ← BC − 1
Repeat until
BC = 0

S: N/A
Z: N/A
H: Reset
P/V: Reset
N: Reset
C: N/A



Timing: For BC≠0 M cycles — 5
T states — 21 (4, 4, 3, 5, 5)
For BC=0 M cycles — 4
T states — 16 (4, 4, 3, 5)

Addressing Mode: Register Indirect

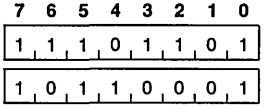
(Note that each repeat is accomplished by a decrement of the BC, so that refresh, etc. continues for each cycle.)

CPIR

Compare data in memory location (HL) to the Accumulator, increment the memory, decrement the byte counter BC, and repeat until BC = 0 or (HL) equals A.

A − (HL)
HL ← HL + 1
BC ← BC − 1
Repeat until BC = 0
or A = (HL)

S: Set if sign of subtraction performed for comparison is negative
Z: Set if A = (HL), otherwise reset
H: Set according to borrow from bit 4
P/V: Set if BC − 1 ≠ 0, otherwise reset
N: Set
C: N/A



Timing: For BC ≠ 0 M cycles — 5
T states — 21 (4, 4, 3, 5, 5)
For BC = 0 M cycles — 4
T states — 16 (4, 4, 3, 5)

Addressing Mode: Register Indirect

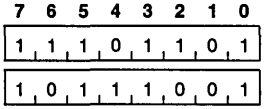
(Note that each repeat is accomplished by a decrement of the PC, so that refresh, etc. continues for each cycle.)

CPDR

Compare data in memory location (HL) to the contents of the Accumulator, decrement the memory pointer and byte counter BC, and repeat until BC = 0, or until (HL) equals the Accumulator.

A − (HL)
HL ← HL − 1
BC ← BC − 1
Repeat until BC = 0
or A = (HL)

S: Set if sign of subtraction performed for comparison is negative
Z: Set according to equality of A and (HL), set if true
H: Set according to borrow from bit 4
P/V: Set if BC − 1 ≠ 0, otherwise reset
N: Set
C: N/A



Timing: For BC ≠ 0 M cycles — 5
T states — 21 (4, 4, 3, 5, 5)
For BC = 0 M cycles — 4
T states — 16 (4, 4, 3, 5)

Addressing Mode: Register Indirect

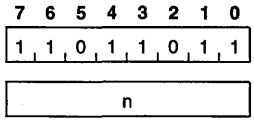
(Note that each repeat is accomplished by a decrement of the BC, so that refresh, etc. continues for each cycle.)

12.12 Input/Output

IN A, (n)

Input data to the Accumulator from the I/O device at address N.

A ← (n) — No flags affected



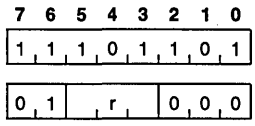
Timing: M cycles — 3
T states — 11 (4, 3, 4)

Addressing Mode: Source — Direct
Destination — Register

IN r, (C)

Input data to register r from the I/O device addressed by the contents of register C. If r=110 only flags are affected.

r ← (C)
S: Set if result is negative
Z: Set if result is zero
H: Reset
P/V: Set if result parity is even
N: Reset
C: N/A



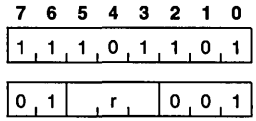
Timing: M cycles — 3
T states — 12 (4, 4, 4)

Addressing Mode: Source — Register Indirect
Destination — Register

OUT (C), r

Output register r to the I/O device addressed by the contents of register C.

(C) ← r — No flags affected



Timing: M cycles — 3
T states — 12 (4, 4, 4)

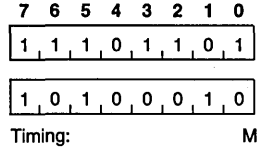
Addressing Mode: Source — Register
Destination — Register Indirect

INI

Input data from the I/O device addressed by the contents of register C to the memory location pointed to by the contents of the HL register. The HL pointer is incremented and the byte counter B is decremented.

(HL) ← (C)
B ← B − 1
HL ← HL + 1

S: Undefined
Z: Set if B−1=0, otherwise reset
H: Undefined
P/V: Undefined
N: Set
C: N/A



Timing: M cycles — 4
T states — 16 (4, 5, 3, 4)

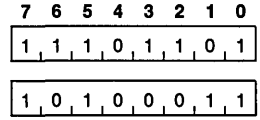
Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

OUTI

Output data from memory location (HL) to the I/O device at port address (C), increment the memory pointer, and decrement the byte counter B.

(C) ← (HL)
B ← B − 1
HL ← HL + 1

S: Undefined
Z: Set if B−1=0, otherwise reset
H: Undefined
P/V: Undefined
N: Set
C: N/A



Timing: M cycles — 4
T states — 16 (4, 5, 3, 4)

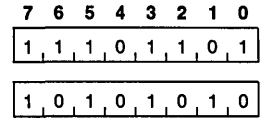
Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

IND

Input data from I/O device at port address (C) to memory location (HL), and decrement HL memory pointer and byte counter B.

(HL) ← (C)
HL ← HL − 1
B ← B − 1

S: Undefined
Z: Set if B−1=0, otherwise reset
H: Undefined
P/V: Undefined
N: Set
C: N/A



Timing: M cycles — 4
T states — 16 (4, 5, 3, 4)

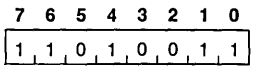
Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

12.12 Input/Output (Continued)

OUT (n), A

Output the Accumulator to the I/O device at address n.

(n) ← A — No flags affected



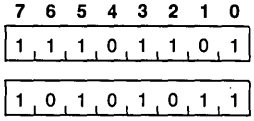
Timing: M cycles — 3
T states — 11 (4, 3, 4)

Addressing Mode: Source — Register
Destination — Direct

OUTD

Data is output from memory location (HL) to the I/O device at port address (C), and the HL memory pointer and byte counter B are decremented.

(C) ← (HL) — S: Undefined
B ← B − 1 — Z: Set if B−1=0, otherwise reset
HL ← HL − 1 — H: Undefined
P/V: Undefined
N: Set
C: N/A



Timing: M cycles — 4
T states — 16 (4, 5, 3, 4)

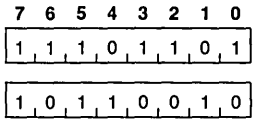
Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

INIR

Data is input from the I/O device at port address (C) to memory location (HL), the HL memory pointer is incremented, and the byte counter B is decremented. The cycle is repeated until B = 0.

(Note that B is tested for zero after it is decremented. By loading B initially with zero, 256 data transfers will take place.)

(HL) ← (C) — S: Undefined
HL ← HL + 1 — Z: Set
B ← B − 1 — H: Undefined
Repeat until B = 0 — P/V: Undefined
N: Set
C: N/A



Timing: For B ≠ 0 — M cycles — 5
T states — 21 (4, 5, 3, 4, 5)
For B = 0 — M cycles — 4
T states — 16 (4, 5, 3, 4)

Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

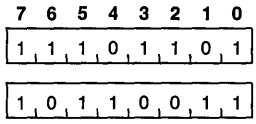
(Note that at the end of each data transfer cycle, interrupts may be recognized and two refresh cycles will be performed.)

OTIR

Data is output to the I/O device at port address (C) from memory location (HL), the HL memory pointer is incremented, and the byte counter B is decremented. The cycles are repeated until B = 0.

(Note that B is tested for zero after it is decremented. By loading B initially with zero, 256 data transfers will take place.)

(C) ← (HL) — S: Undefined
HL ← HL + 1 — H: Undefined
B ← B − 1 — Z: Set
Repeat until B = 0 — P/V: Undefined
N: Set
C: N/A



Timing: For B ≠ 0 — M cycles — 5
T states — 21 (4, 5, 3, 4, 5)
For B = 0 — M cycles — 4
T states — 16 (4, 5, 3, 4)

Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

(Note that at the end of each data transfer cycle, interrupts may be recognized and two refresh cycles will be performed.)

12.12 Input/Output (Continued)

INDR

Data is input from the I/O device at address (C) to memory location (HL), then the HL memory pointer is byte counter B are decremented. The cycle is repeated until B = 0.

(Note that B is tested for zero after it is decremented. By loading B initially with zero, 256 data transfers will take place.)

(HL) ← (C) S: Undefined
HL ← HL − 1 Z: Set
B ← B − 1 H: Undefined
Repeat until B = 0 P/V: Undefined
N: Set
C: N/A

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1

1	0	1	1	0	0	1	0

Timing: For B ≠ 0 M cycles — 5
T states — 21 (4, 5, 3, 4, 5)
For B = 0 M cycles — 4
T states — 16 (4, 5, 3, 4)
Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

(Note that after each data transfer cycle, interrupts may be recognized and two refresh cycles are performed.)

OTDR

Data is output from memory location (HL) to the I/O device at port address (C), then the HL memory pointer and byte counter B are decremented. The cycle is repeated until B = 0.

(Note that B is tested for zero after it is decremented. By loading B initially with zero, 256 data transfers will take place.)

(C) ← (HL) S: Undefined
HL ← HL − 1 Z: Set
B ← B − 1 H: Undefined
Repeat until B = 0 P/V: Undefined
N: Set
C: N/A

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1

1	0	1	1	1	0	1	1

Timing: For B ≠ 0 M cycles — 5
T states — 21 (4, 5, 3, 4, 5)
For B = 0 M cycles — 4
T states — 16 (4, 5, 3, 4)
Addressing Mode: Implied/Source — Register Indirect
Destination — Register Indirect

(Note that after each data transfer cycle the NSC800 will accept interrupts and perform two refresh cycles.)

12.13 CPU Control

NOP

The CPU performs no operation.

— — — No flags affected

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

Timing: M cycles — 1
T states — 4
Addressing Mode: N/A

HALT

The CPU halts execution of the program. Dummy op-code fetches are performed from the next memory location to keep the refresh circuits active until the CPU is interrupted or reset from the halted state.

— — — No flags affected

7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0

Timing: M cycles — 1
T states — 4
Addressing Mode: N/A

DI

Disable system level interrupts.

IFF_1 ← 0 No flags affected
IFF_2 ← 0

7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1

Timing: M cycles — 1
T states — 4
Addressing Mode: N/A

EI

The system level interrupts are enabled. During execution of this instruction, and the next one, the maskable interrupts will be disabled.

IFF_1 ← 1 No flags affected
IFF_2 ← 1

7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1

Timing: M cycles — 1
T states — 4
Addressing Mode: N/A

IM 0

The CPU is placed in interrupt mode 0.

— — — No flags affected

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1

0	1	0	0	0	1	1	0

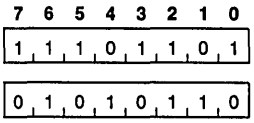
Timing: M cycles — 2
T states — 8 (4, 4)
Addressing Mode: N/A

12.13 CPU Control (Continued)

IM 1

The CPU is placed in interrupt mode 1.

— — — No flags affected



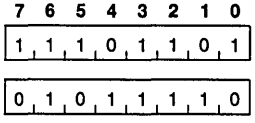
Timing: M cycles — 2
T states — 8 (4, 4)

Addressing Mode: N/A

IM 2

The CPU is placed in interrupt mode 2.

— — — No flags affected



Timing: M cycles — 2
T states — 8 (4, 4)

Addressing Mode: N/A

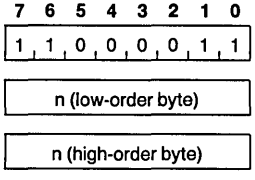
12.14 Program Control

JUMPS

JP nn

Unconditional jump to program location nn.

PC ← nn No flags affected



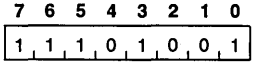
Timing: M cycles — 3
T states — 10 (4, 3, 3)

Addressing Mode: Direct

JP (ss)

Unconditional jump to program location pointed to by register ss.

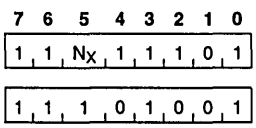
PC ← ss No flags affected



JP (HL)

Timing: M cycles — 1
T states — 4

Addressing Mode: Register Indirect



JP (IX) (for $N_X = 0$)
JP (IY) (for $N_X = 1$)

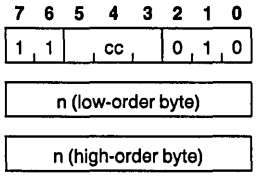
Timing: M cycles — 2
T states — 8 (4, 4)

Addressing Mode: Register Indirect

JP cc, nn

Conditionally jump to program location nn based on testable flag states.

If cc true, No flags affected
PC ← nn,
otherwise continue



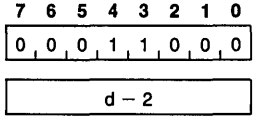
Timing: M cycles — 3
T states — 10 (4, 3, 3)

Addressing Mode: Direct

JR d

Unconditional jump to program location calculated with respect to the program counter and the displacement d.

PC ← PC + d No flags affected



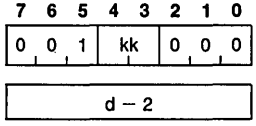
Timing: M cycles — 3
T states — 12 (4, 3, 5)

Addressing Mode: PC Relative

JR kk, d

Conditionally jump to program location calculated with respect to the program counter and the displacement d, based on limited testable flag states.

If kk true, No flags affected
PC ← PC + d,
otherwise continue



Timing: if kk met (true) M cycles — 3
T states — 12 (4, 3, 5)
if kk not met (not true) M cycles — 2
T states — 7 (4, 3)

Addressing Mode: PC Relative

12.14 Program Control (Continued)

DJNZ d

Decrement the B register and conditionally jump to program location calculated with respect to the program counter and the displacement d, based on the contents of the B register.

B ← B − 1 — No flags affected

If B = 0 continue,

else PC ← PC + d

7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0

d − 2

Timing: If B ≠ 0 — M cycles — 3; T states — 13 (5, 3, 5)

If B = 0 — M cycles — 2; T states — 8 (5, 3)

Addressing Mode: PC Relative

CALLS

CALL nn

Unconditional call to subroutine at location nn.

(SP − 1) ← PC_H — No flags affected

(SP − 2) ← PC_L

SP ← SP − 2

PC ← nn

7	6	5	4	3	2	1	0
1	1	0	0	1	1	0	1

n (low-order byte)

n (high-order byte)

Timing: M Cycles — 5; T states — 17 (4, 3, 4, 3, 3)

Addressing Mode: Direct

CALL cc, nn

Conditional call to subroutine at location nn based on testable flag stages.

If cc true, — No flags affected

(SP − 1) ← PC_H

(SP − 2) ← PC_L

SP ← SP − 2

PC ← nn,

else continue

7	6	5	4	3	2	1	0
1	1		cc		1	0	0

n (low-order byte)

n (high-order byte)

Timing: If cc true — M cycles — 5; T states 17 (4, 3, 4, 3, 3)

If cc not true — M cycles — 3; T states — 10 (4, 3, 3)

Addressing Mode: Direct

RETURNS

RET

Unconditional return from subroutine or other return to program location pointed to by the top of the stack.

PC_L ← (SP) — No flags affected

PC_H ← (SP + 1)

SP ← SP + 2

7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	1

Timing: M cycles — 3; T states — 10 (4, 3, 3)

Addressing Mode: Register Indirect

RET cc

Conditional return from subroutine or other return to program location pointed to by the top of the stack.

If cc true, — No flags affected

PC_L ← (SP)

PC_H ← (SP + 1)

SP ← SP + 2,

else continue

7	6	5	4	3	2	1	0
1	1		cc		0	0	0

Timing: If cc true — M cycles — 3; T states — 11 (5, 3, 3)

If cc not true — M cycles — 1; T states — 5

Addressing Mode: Register Indirect

RETI

Unconditional return from interrupt handling subroutine. Functionally identical to RET instruction. Unique opcode allows monitoring by external hardware.

PC_L ← (SP) — No flags affected

PC_H ← (SP + 1)

SP ← SP + 2

7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1

0	1	0	0	1	1	0	1

Timing: M cycles — 4; T states — 14 (4, 4, 3, 3)

Addressing Mode: Register Indirect

12.14 Program Control (Continued)

RETN

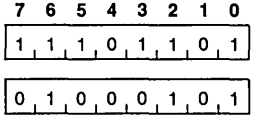
Unconditional return from non-maskable interrupt handling subroutine. Functionally similar to RET instruction, except interrupt enable state is restored to that prior to non-maskable interrupt.

$PC_L \leftarrow (SP)$ No flags affected

$PC_H \leftarrow (SP + 1)$

$SP \leftarrow SP + 2$

$IFF_1 \leftarrow IFF_2$



Timing: M cycles — 4
T states — 14 (4, 4, 3, 3)

Addressing Mode: Register Indirect

RESTARTS

RST P

The present contents of the PC are pushed onto the memory stack and the PC is loaded with dedicated program locations as determined by the specific restart executed.

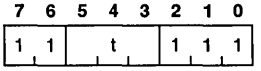
$(SP - 1) \leftarrow PC_H$ No flags affected

$(SP - 2) \leftarrow PC_L$

$SP \leftarrow SP - 2$

$PC_H \leftarrow 0$

$PC_L \leftarrow P$



Timing: M cycles — 3
T states — 11 (5, 3, 3)

Addressing Mode: Modified Page Zero

p	00H	08H	10H	18H	20H	28H	30H	38H
t	000	001	010	011	100	101	110	111

12.15 Instruction Set: Alphabetical Order

ADC	A, (HL)	8E	BIT	0, B	CB 40
ADC	A, (IX+d)	DD 8Ed	BIT	0, C	CB 41
ADC	A, (IY+d)	FD 8Ed	BIT	0, D	CB 42
ADC	A, A	8F	BIT	0, E	CB 43
ADC	A, B	88	BIT	0, H	CB 44
ADC	A, C	89	BIT	0, L	CB 45
ADC	A, D	8A	BIT	1, (HL)	CB 4E
ADC	A, E	8B	BIT	1, (IX+d)	DD CBd4E
ADC	A, H	8C	BIT	1, (IY+d)	FD CBd4E
ADC	A, L	8D	BIT	1, A	CB 4F
ADC	A, n	CE n	BIT	1, B	CB 48
ADC	HL, BC	ED 4A	BIT	1, C	CB 49
ADC	HL, DE	ED 5A	BIT	1, D	CB 4A
ADC	HL, HL	ED 6A	BIT	1, E	CB 4B
ADC	HL, SP	ED 7A	BIT	1, H	CB 4C
ADD	A, (HL)	86	BIT	1, L	CB 4D
ADD	A, (IX+d)	DD 86d	BIT	2, (HL)	CB 56
ADD	A, (IY+d)	FD 86d	BIT	2, (IX+d)	DD CBd56
ADD	A, A	87	BIT	2, (IY+d)	FD CBd56
ADD	A, B	80	BIT	2, A	CB 57
ADD	A, C	81	BIT	2, B	CB 50
ADD	A, D	82	BIT	2, C	CB 51
ADD	A, E	83	BIT	2, D	CB 52
ADD	A, H	84	BIT	2, E	CB 53
ADD	A, L	85	BIT	2, H	CB 54
ADD	A, n	C6 n	BIT	2, L	CB 55
ADD	HL, BC	09	BIT	3, (HL)	CB 5E
ADD	HL, DE	19	BIT	3, (IX+d)	DD CBd5E
ADD	HL, HL	29	BIT	3, (IY+d)	FD CBd5E
ADD	HL, SP	39	BIT	3, A	CB 5F
ADD	IX, BC	DD 09	BIT	3, B	CB 58
ADD	IX, DE	DD 19	BIT	3, C	CB 59
ADD	IX, IX	DD 29	BIT	3, D	CB 5A
ADD	IX, SP	DD 39	BIT	3, E	CB 5B
ADD	IY, BC	FD 09	BIT	3, H	CB 5C
ADD	IY, DE	FD 19	BIT	3, L	CB 5D
ADD	IY, IY	FD 29	BIT	4, (HL)	CB 66
ADD	IY, SP	FD 39	BIT	4, (IX+d)	DD CBd66
AND	(HL)	A6	BIT	4, (IY+d)	FD CBd66
AND	(IX+d)	DD A6d	BIT	4, A	CB 67
AND	(IY+d)	FD A6d	BIT	4, B	CB 60
AND	A	A7	BIT	4, C	CB 61
AND	B	A0	BIT	4, D	CB 62
AND	C	A1	BIT	4, E	CB 63
AND	D	A2	BIT	4, H	CB 64
AND	E	A3	BIT	4, L	CB 65
AND	H	A4	BIT	5, (HL)	CB 6E
AND	L	A5	BIT	5, (IX+d)	DD CBd6E
AND	n	E6 n	BIT	5, (IY+d)	FD CBd6E
BIT	0, (HL)	CB 46	BIT	5, A	CB 6F
BIT	0, (IX+d)	DD CBd46	BIT	5, B	CB 68
BIT	0, (IY+d)	FD CBd46	BIT	5, C	CB 69
BIT	0, A	CB 47	BIT	5, D	CB 6A

(nn) = address of memory location
nn = Data (16 bit)
n = Data (8 bit)
d = signed displacement
d2 = d−2

12.15 Instruction Set: Alphabetical Order (Continued)

BIT	5, E	CB 6B
BIT	5, H	CB 6C
BIT	5, L	CB 6D
BIT	6, (HL)	CB 76
BIT	6, (IX+d)	DD CBd76
BIT	6, (IY+d)	FD CBd76
BIT	6, A	CB 77
BIT	6, B	CB 70
BIT	6, C	CB 71
BIT	6, D	CB 72
BIT	6, E	CB 73
BIT	6, H	CB 74
BIT	6, L	CB 75
BIT	7, (HL)	CB 7E
BIT	7, (IX+d)	DD CBd7E
BIT	7, (IY+d)	FD CBd7E
BIT	7, A	CB 7F
BIT	7, B	CB 78
BIT	7, C	CB 79
BIT	7, D	CB 7A
BIT	7, E	CB 7B
BIT	7, H	CB 7C
BIT	7, L	CB 7D
CALL	C, nn	DCnn
CALL	M, nn	FCnn
CALL	NC, nn	D4nn
CALL	nn	CDnn
CALL	NZ, nn	C4nn
CALL	P, nn	F4nn
CALL	PE, nn	ECnn
CALL	PO, nn	E4nn
CALL	Z, nn	CCnn
CCF		3F
CP	(HL)	BE
CP	(IX+d)	DD BEd
CP	(IY+d)	FD BEd
CP	A	BF
CP	B	B8
CP	C	B9
CP	D	BA
CP	E	BB
CP	H	BC
CP	L	BD
CP	n	FE n
CPD		ED A9
CPDR		ED B9
CPI		ED A1
CPIR		ED B1
CPL		2F
DAA		27
DEC	(HL)	35
DEC	(IX+d)	DD 35d
DEC	(IY+d)	FD 35d
DEC	A	3D
DEC	B	05
DEC	BC	0B
DEC	C	0D
DEC	D	15
DEC	DE	1B
DEC	E	1D
DEC	H	25
DEC	HL	2B
DEC	IX	DD 2B
DEC	IY	FD 2B
DEC	L	2D
DEC	SP	3B
DI		F3
DJNZ	d2	10 d2
EI		FB
EX	(SP), HL	E3
EX	(SP), IX	DD E3
EX	(SP), IY	FD E3
EX	AF, A'F'	08
EX	DE, HL	EB
EXX		D9
HALT		76
IM	0	ED 46
IM	1	ED 56
IM	2	ED 5E
IN	A, (C)	ED78
IN	A, (n)	DB n
IN	B, (C)	ED 40
IN	C, (C)	ED 48
IN	D, (C)	ED 50
IN	E, (C)	ED 58
IN	H, (C)	ED 60
IN	L, (C)	ED 68
INC	(HL)	34
INC	(IX+d)	DD 34d
INC	(IY+d)	FD 34d
INC	A	3C
INC	B	04
INC	BC	03
INC	C	0C
INC	D	14
INC	DE	13
INC	E	1C
INC	H	24
INC	HL	23
INC	IX	DD 23
INC	IY	FD 23
INC	L	2C
INC	SP	33
IND		ED AA
INDR		ED BA
INI		ED A2

(nn) = Address of memory location
nn = Data (16 bit)
n = Data (8 bit)
d = signed displacement
d2 = d − 2

12.15 Instruction Set: Alphabetical Order (Continued)

INIR		ED B2
JP	(HL)	E9
JP	(IX)	DD E9
JP	(IY)	FD E9
JP	C, nn	DAnn
JP	M, nn	FAnn
JP	NC, nn	D2nn
JP	nn	C3nn
JP	NZ, nn	C2nn
JP	P, nn	F2nn
JP	PE, nn	EAnn
JP	PO, nn	E2nn
JP	Z, nn	CAnn
JR	C, d2	38 d2
JR	d2	18 d2
JR	NC, d2	30 d2
JR	NZ, d2	20 d2
JR	Z, d2	28 d2
LD	(BC), A	02
LD	(DE), A	12
LD	(HL), A	77
LD	(HL), B	70
LD	(HL), C	71
LD	(HL), D	72
LD	(HL), E	73
LD	(HL), H	74
LD	(HL), L	75
LD	(HL), n	36 n
LD	(IX+d), A	DD 77d
LD	(IX+d), B	DD 70d
LD	(IX+d), C	DD 71d
LD	(IX+d), D	DD 72d
LD	(IX+d), E	DD 73d
LD	(IX+d), H	DD 74d
LD	(IX+d), L	DD 75d
LD	(IX+d), n	DD 36dn
LD	(IY+d), A	FD 77d
LD	(IY+d), B	FD 70d
LD	(IY+d), C	FD 71d
LD	(IY+d), D	FD 72d
LD	(IY+d), E	FD 73d
LD	(IY+d), H	FD 74d
LD	(IY+d), L	FD 75d
LD	(IY+d), n	FD 36dn
LD	(nn), A	32nn
LD	(nn), BC	ED 43nn
LD	(nn), DE	ED 53nn
LD	(nn), HL	22nn
LD	(nn), IX	DD 22nn
LD	(nn), IY	FD 22nn
LD	(nn), SP	ED 73nn
LD	A, (BC)	0A
LD	A, (DE)	1A
LD	A, (HL)	7E
LD	A, (IX+d)	DD 7Ed
LD	A, (IY+d)	FD 7Ed
LD	A, (nn)	3Ann
LD	A, A	7F
LD	A, B	78
LD	A, C	79
LD	A, D	7A
LD	A, E	7B
LD	A, H	7C
LD	A, I	ED 57
LD	A, L	7D
LD	A, n	3E n
LD	B, (HL)	46
LD	B, (IX+d)	DD 46d
LD	B, (IY+d)	FD 46d
LD	B, A	47
LD	B, B	40
LD	B, C	41
LD	B, D	42
LD	B, E	43
LD	B, H	44
LD	B, L	45
LD	B, n	06 n
LD	BC, (nn)	ED 4B
LD	BC, nn	01nn
LD	C, (HL)	4E
LD	C, (IX+d)	DD 4Ed
LD	C, (IY+d)	FD 4Ed
LD	C, A	4F
LD	C, B	48
LD	C, C	49
LD	C, D	4A
LD	C, E	4B
LD	C, H	4C
LD	C, L	4D
LD	C, n	0E n
LD	D, (HL)	56
LD	D, (IX+d)	DD 56d
LD	D, (IY+d)	FD 56d
LD	D, A	57
LD	D, B	50
LD	D, C	51
LD	D, D	52
LD	D, E	53
LD	D, H	54
LD	D, L	55
LD	D, n	16 n
LD	DE, (nn)	ED 5Bnn
LD	DE, nn	11nn
LD	E, (HL)	5E
LD	E, (IX+d)	DD 5Ed
LD	E, (IY+d)	FD 5Ed

(nn) = Address of memory location
nn = Data (16 bit)
n = Data (8 bit)
d = signed displacement
d2 = d−2

12.15 Instruction Set: Alphabetical Order (Continued)

LD	E, A	5F
LD	E, B	58
LD	E, C	59
LD	E, D	5A
LD	E, E	5B
LD	E, H	5C
LD	E, L	5D
LD	E, n	1E n
LD	H, (HL)	66
LD	H, (IX+d)	DD 66d
LD	H, (IY+d)	FD 66d
LD	H, A	67
LD	H, B	60
LD	H, C	61
LD	H, D	62
LD	H, E	63
LD	H, H	64
LD	H, L	65
LD	H, n	26 n
LD	HL, (nn)	2Ann
LD	HL, nn	21nn
LD	I, A	ED 47
LD	IX, (nn)	DD 2Ann
LD	IX, nn	DD 21nn
LD	IY, (nn)	FD 2Ann
LD	IY, nn	FD 21nn
LD	L, (HL)	6E
LD	L, (IX+d)	DD 6Ed
LD	L, (IY+d)	FD 6Ed
LD	L, A	6F
LD	L, B	68
LD	L, C	69
LD	L, D	6A
LD	L, E	6B
LD	L, H	6C
LD	L, L	6D
LD	L, n	2E n
LD	SP, (nn)	ED 7Bnn
LD	SP, HL	F9
LD	SP, IX	DD F9
LD	SP, IY	FD F9
LD	SP, nn	31nn
LDD		ED A8
LDDR		ED B8
LDI		ED A0
LDIR		ED B0
NEG		ED n
NOP		00
OR	(HL)	B6
OR	(IX+d)	DD B6d
OR	(IY+d)	FD B6d
OR	A	B7
OR	B	B0
OR	C	B1
OR	D	B2
OR	E	B3
OR	H	B4
OR	L	B5
OR	n	F6 n
OTDR		ED BB
OTIR		ED B3
OUT	(C), A	ED 79
OUT	(C), B	ED 41
OUT	(C), C	ED 49
OUT	(C), D	ED 51
OUT	(C), E	ED 59
OUT	(C), H	ED 61
OUT	(C), L	ED 69
OUT	n, A	D3 n
OUTD		ED AB
OUTI		ED A3
POP	AF	F1
POP	BC	C1
POP	DE	D1
POP	HL	E1
POP	IX	DD E1
POP	IY	FD E1
PUSH	AF	F5
PUSH	BC	C5
PUSH	DE	D5
PUSH	HL	E5
PUSH	IX	DD E5
PUSH	IY	FD E5
RES	0, (HL)	CB 86
RES	0, (IX+d)	DD CBd86
RES	0, (IY+d)	FD CBd86
RES	0, A	CB 87
RES	0, B	CB 80
RES	0, C	CB 81
RES	0, D	CB 82
RES	0, E	CB 83
RES	0, H	CB 84
RES	0, L	CB 85
RES	1, (HL)	CB 8E
RES	1, (IX+d)	DD CBd8E
RES	1, (IY+d)	FD CBd8E
RES	1, A	CB 8F
RES	1, B	CB 88
RES	1, C	CB 89
RES	1, D	CB 8A
RES	1, E	CB 8B
RES	1, H	CB 8C
RES	1, L	CB 8D
RES	2, (HL)	CB 96
RES	2, (IX+d)	DD CBd96
RES	2, (IY+d)	FD CBd96

(nn) = Address of memory location
nn = Data (16 bit)
n = Data (8 bit)
d = signed displacement
d2 = d−2

12.15 Instruction Set: Alphabetical Order (Continued)

RES	2, A	CB 97
RES	2, B	CB 90
RES	2, C	CB 91
RES	2, D	CB 92
RES	2, E	CB 93
RES	2, H	CB 94
RES	2, L	CB 95
RES	3, (HL)	CB 9E
RES	3, (IX+d)	DD CBd9E
RES	3, (IY+d)	FD CBd9E
RES	3, A	CB 9F
RES	3, B	CB 98
RES	3, C	CB 99
RES	3, D	CB 9A
RES	3, E	CB 9B
RES	3, H	CB 9C
RES	3, L	CB 9D
RES	4, (HL)	CB A6
RES	4, (IX+d)	DD CBdA6
RES	4, (IY+d)	FD CBdA6
RES	4, A	CB A7
RES	4, B	CB A0
RES	4, C	CB A1
RES	4, D	CB A2
RES	4, E	CB A3
RES	4, H	CB A4
RES	4, L	CB A5
RES	5, (HL)	CB AE
RES	5, (IX+d)	DD CBdAE
RES	5, (IY+d)	FD CBdAE
RES	5, A	CB AF
RES	5, B	CB A8
RES	5, C	CB A9
RES	5, D	CB AA
RES	5, E	CB AB
RES	5, H	CB AC
RES	5, L	CB AD
RES	6, (HL)	CB B6
RES	6, (IX+d)	DD CBdB6
RES	6, (IY+d)	FD CBdB6
RES	6, A	CB B7
RES	6, B	CB B0
RES	6, C	CB B1
RES	6, D	CB B2
RES	6, E	CB B3
RES	6, H	CB B4
RES	6, L	CB B5
RES	7, (HL)	CB BE
RES	7, (IX+d)	DD CBdBE
RES	7, (IY+d)	FD CBdBE
RES	7, A	CB BF
RES	7, B	CB B8
RES	7, C	CB B9
RES	7, D	CB BA
RES	7, E	CB BB
RES	7, H	CB BC
RES	7, L	CB BD
RET		C9
RET	C	D8
RET	M	F8
RET	NC	D0
RET	NZ	C0
RET	P	F0
RET	PE	E8
RET	PO	E0
RET	Z	C8
RETI		ED 4D
RETN		ED 45
RL	(HL)	CB 16
RL	(IX+d)	DD CBd16
RL	(IY+d)	FD CBd16
RL	A	CB 17
RL	B	CB 10
RL	C	CB 11
RL	D	CB 12
RL	E	CB 13
RL	H	CB 14
RL	L	CB 15
RLA		17
RLC	(HL)	CB 06
RLC	(IX+d)	DD CBd06
RLC	(IY+d)	FD CBd06
RLC	A	CB 07
RLC	B	CB 00
RLC	C	CB 01
RLC	D	CB 02
RLC	E	CB 03
RLC	H	CB 04
RLC	L	CB 05
RLCA		07
RLD		ED 6F
RR	(HL)	CB 1E
RR	(IX+d)	DD CBd1E
RR	(IY+d)	FD CBd1E
RR	A	CB 1F
RR	B	CB 18
RR	C	CB 19
RR	D	CB 1A
RR	E	CB 1B
RR	H	CB 1C
RR	L	CB 1D
RRA		1F
RRC	(HL)	CB OE
RRC	(IX+d)	DD CBd0E
RRC	(IY+d)	FD CBd0E
RRC	A	CB 0F

(nn) = Address of memory location
nn = Data (16 bit)
n = Data (8 bit)
d = signed displacement
d2 = d−2

12.15 Instruction Set: Alphabetical Order (Continued)

RRC	B	CB 08
RRC	C	CB 09
RRC	D	CB 0A
RRC	E	CB 0B
RRC	H	CB 0C
RRC	L	CB 0D
RRCA		0F
RRD		ED 67
RST	0	C7
RST	08H	CF
RST	10H	D7
RST	18H	DF
RST	20H	E7
RST	28H	EF
RST	30H	F7
RST	38H	FF
SBC	A, (HL)	9E
SBC	A, (IX+d)	DD 9Ed
SBC	A, (IY+d)	FD 9Ed
SBC	A, A	9F
SBC	A, B	98
SBC	A, C	99
SBC	A, D	9A
SBC	A, E	9B
SBC	A, H	9C
SBC	A, L	9D
SBC	A, n	DE n
SBC	HL, BC	ED 42
SBC	HL, DE	ED 52
SBC	HL, HL	ED 62
SBC	HL, SP	ED 72
SCF		37
SET	0, (HL)	CB C6
SET	0, (IX+d)	DD CBdC6
SET	0, (IY+d)	FD CBdC6
SET	0, A	CB C7
SET	0, B	CB C0
SET	0, C	CB C1
SET	0, D	CB C2
SET	0, E	CB C3
SET	0, H	CB C4
SET	0, L	CB C5
SET	1, (HL)	CB CE
SET	1, (IX+d)	DD CBdCE
SET	1, (IY+d)	FD CBdCE
SET	1, A	CB CF
SET	1, B	CB C8
SET	1, C	CB C9
SET	1, D	CB CA
SET	1, E	CB CB
SET	1, H	CB CC
SET	1, L	CB CD
SET	2, (HL)	CB D6
SET	2, (IX+d)	DD CBdD6
SET	2, (IY+d)	FD CBdD6
SET	2, A	CB D7
SET	2, B	CB D0
SET	2, C	CB D1
SET	2, D	CB D2
SET	2, E	CB D3
SET	2, H	CB D4
SET	2, L	CB D5
SET	3, (HL)	CB DE
SET	3, (IX+d)	DD CBdDE
SET	3, (IY+d)	FD CBdDE
SET	3, A	CB DF
SET	3, B	CB D8
SET	3, C	CB D9
SET	3, D	CB DA
SET	3, E	CB DB
SET	3, H	CB DC
SET	3, L	CB DD
SET	4, (HL)	CB E6
SET	4, (IX+d)	DD CBdE6
SET	4, (IY+d)	FD CBdE6
SET	4, A	CB E7
SET	4, B	CB E0
SET	4, C	CB E1
SET	4, D	CB E2
SET	4, E	CB E3
SET	4, H	CB E4
SET	4, L	CB E5
SET	5, (HL)	CB EE
SET	5, (IX+d)	DD CBdEE
SET	5, (IY+d)	FD CBdEE
SET	5, A	CB EF
SET	5, B	CB E8
SET	5, C	CB E9
SET	5, D	CB EA
SET	5, E	CB EB
SET	5, H	CB EC
SET	5, L	CB ED
SET	6, (HL)	CB F6
SET	6, (IX+d)	DD CBdF6
SET	6, (IY+d)	FD CBdF6
SET	6, A	CB F7
SET	6, B	CB F0
SET	6, C	CB F1
SET	6, D	CB F2
SET	6, E	CB F3
SET	6, H	CB F4
SET	6, L	CB F5
SET	7, (HL)	CB FE
SET	7, (IX+d)	DD CBdFE
SET	7, (IY+d)	FD CBdFE
SET	7, A	CB FF

(nn) = Address of memory location
nn = Data (16 bit)
n = Data (8 bit)
d = displacement
d2 = d−2

12.15 Instruction Set: Alphabetical Order (Continued)

SET	7, B	CB F8	SRL	A	CB 3F
SET	7, C	CB F9	SRL	B	CB 38
SET	7, D	CB FA	SRL	C	CB 39
SET	7, E	CB FB	SRL	D	CB 3A
SET	7, H	CB FC	SRL	E	CB 3B
SET	7, L	CB FD	SRL	H	CB 3C
SLA	(HL)	CB 26	SRL	L	CB 3D
SLA	(IX+d)	DD CBd26	SUB	(HL)	96
SLA	(IY+d)	FD CBd26	SUB	(IX+d)	DD 96d
SLA	A	CB 27	SUB	(IY+d)	FD 96d
SLA	B	CB 20	SUB	A	97
SLA	C	CB 21	SUB	B	90
SLA	D	CB 22	SUB	C	91
SLA	E	CB 23	SUB	D	92
SLA	H	CB 24	SUB	E	93
SLA	L	CB 25	SUB	H	94
SRA	(HL)	CB 2E	SUB	L	95
SRA	(IX+d)	DD CBd2E	SUB	n	D6 n
SRA	(IY+d)	FD CBd2E	XOR	(HL)	AE
SRA	A	CB 2F	XOR	(IX+d)	DD AEd
SRA	B	CB 28	XOR	(IY+d)	FD AEd
SRA	C	CB 29	XOR	A	AF
SRA	D	CB 2A	XOR	B	A8
SRA	E	CB 2B	XOR	C	A9
SRA	H	CB 2C	XOR	D	AA
SRA	L	CB 2D	XOR	E	AB
SRL	(HL)	CB 3E	XOR	H	AC
SRL	(IX+d)	DD CBd3E	XOR	L	AD
SRL	(IY+d)	FD CBd3E	XOR	n	EE n

12.16 Instruction Set: Numerical Order

Op Code	Mnemonic	Op Code	Mnemonic	Op Code	Mnemonic
00	NOP	15	DEC D	2Ann	LD HL,(nn)
01nn	LD BC,nn	16n	LD D,n	2B	DEC HL
02	LD (BC),A	17	RLA	2C	INC L
03	INC BC	18d2	JR d2	2D	DEC L
04	INC B	19	ADD HL,DE	2En	LD L,n
05	DEC B	1A	LD A,(DE)	2F	CPL
06n	LD B,n	1B	DEC DE	30d2	JR NC,d2
07	RLCA	1C	INC E	31nn	LD SP,nn
08	EX AF,A'F'	1D	DEC E	32nn	LD (nn),A
09	ADD HL,BC	1En	LD E,n	33	INC SP
0A	LD A,(BC)	1F	RRA	34	INC (HL)
0B	DEC BC	20d2	JR NZ,d2	35	DEC (HL)
0C	INC C	21nn	LD HL,nn	36n	LD (HL),n
0D	DEC C	22nn	LD (nn),HL	37	SCF
0En	LD C,n	23	INC HL	38	JR C,d2
0F	RRCA	24	INC H	39	ADD HL,SP
10d2	DJNZ d2	25	DEC H	3Ann	LD A,(nn)
11nn	LD DE,nn	26n	LD H, n	3B	DEC SP
12	LD (DE),A	27	DAA	3C	INC A
13	INC DE	28d2	JR Z,d2	3D	DEC A
14	INC D	29	ADD HL,HL	3En	LD A,n

(nn) = Address of memory location
nn = Data (16 bit)
n = Data (8 bit)
d = displacement
d2 = d − 2

12.16 Instruction Set: Numerical Order (Continued)

Op Code	Mnemonic	Op Code	Mnemonic	Op Code	Mnemonic
3F	CCF	74	LD (HL),H	A9	XOR C
40	LD B,B	75	LD (HL),L	AA	XOR D
41	LD B,C	76	HALT	AB	XOR E
42	LD B,D	77	LD (HL),A	AC	XOR H
43	LD B,E	78	LD A,B	AD	XOR L
44	LD B,H	79	LD A,C	AE	XOR (HL)
45	LD B,L	7A	LD A,D	AF	XOR A
46	LD B,(HL)	7B	LD A,E	B0	OR B
47	LD B,A	7C	LD A,H	B1	OR C
48	LD C,B	7D	LD A,L	B2	OR D
49	LD C,C	7E	LD A,(HL)	B3	OR E
4A	LD C,D	7F	LD A,A	B4	OR H
4B	LD C,E	80	ADD A,B	B5	OR L
4C	LD C,H	81	ADD A,C	B6	OR (HL)
4D	LD C,L	82	ADD A,D	B7	OR A
4E	LD C,(HL)	83	ADD A,E	B8	CP B
4F	LD C,A	84	ADD A,H	B9	CP C
50	LD D,B	85	ADD A,L	BA	CP D
51	LD D,C	86	ADD A,(HL)	BB	CP E
52	LD D,D	87	ADD A,A	BC	CP H
53	LD D,E	88	ADC A,B	BD	CP L
54	LD D,H	89	ADC A,C	BE	CP (HL)
55	LD D,L	8A	ADC A,D	BF	CP A
56	LD D,(HL)	8B	ADC A,E	C0	RET NZ
57	LD D,A	8C	ADC A,H	C1	POP BC
58	LD E,B	8D	ADC A,L	C2nn	JP NZ,nn
59	LD E,C	8E	ADC A,(HL)	C3nn	JP nn
5A	LD E,D	8F	ADC A,A	C4nn	CALL NZ,nn
5B	LD E,E	90	SUB B	C5	PUSH BC
5C	LD E,H	91	SUB C	C6n	ADD A,n
5D	LD E,L	92	SUB D	C7	RST 0
5E	LD E,(HL)	93	SUB E	C8	RET Z
5F	LD E,A	94	SUB H	C9	RET
60	LD H,B	95	SUB L	CAnn	JP Z,nn
61	LD H,C	96	SUB (HL)	CB00	RLC B
62	LD H,D	97	SUB A	CB01	RLC C
63	LD H,E	98	SBC A,B	CB02	RLC D
64	LD H,H	99	SBC A,C	CB03	RLC E
65	LD H,L	9A	SBC A,D	CB04	RLC H
66	LD H,(HL)	9B	SBC A,E	CB05	RLC L
67	LD H,A	9C	SBC A,H	CB06	RLC (HL)
68	LD L,B	9D	SBC A,L	CB07	RLC A
69	LD L,C	9E	SBC A,(HL)	CB08	RRC B
6A	LD L,D	9F	SBC A,A	CB09	RRC C
6B	LD L,E	A0	AND B	CB0A	RRC D
6C	LD L,H	A1	AND C	CB0B	RRC E
6D	LD L,L	A2	AND D	CB0C	RRC H
6E	LD L,(HL)	A3	AND E	CB0D	RRC L
6F	LD L,A	A4	AND H	CB0E	RRC (HL)
70	LD (HL),B	A5	AND L	CB0F	RRC A
71	LD (HL),C	A6	AND (HL)	CB10	RL B
72	LD (HL),D	A7	AND A	CB11	RL C
73	LD (HL),E	A8	XOR B	CB12	RL D

(nn) = Address of memory location d = displacement
nn = Data (16 bit) d2 = d − 2
n = Data (8-bit)

12.16 Instruction Set: Numerical Order (Continued)

Op Code	Mnemonic	Op Code	Mnemonic	Op Code	Mnemonic
CB13	RL E	CB4F	BIT 1,A	CB83	RES 0,E
CB14	RL H	CB50	BIT 2,B	CB84	RES 0,H
CB15	RL L	CB51	BIT 2,C	CB85	RES 0,L
CB16	RL (HL)	CB52	BIT 2,D	CB86	RES 0,(HL)
CB17	RL A	CB53	BIT 2,E	CB87	RES 0,A
CB18	RR B	CB54	BIT 2,H	CB88	RES 1,B
CB19	RR C	CB55	BIT 2,L	CB89	RES 1,C
CB1A	RR D	CB56	BIT 2,(HL)	CB8A	RES 1,D
CB1B	RR E	CB57	BIT 2,A	CB8B	RES 1,E
CB1C	RR H	CB58	BIT 3,B	CB8C	RES 1,H
CB1D	RR L	CB59	BIT 3,C	CB8D	RES 1,L
CB1E	RR (HL)	CB5A	BIT 3,D	CB8E	RES 1,(HL)
CB1F	RR A	CB5B	BIT 3,E	CB8F	RES 1,A
CB20	SLA B	CB5C	BIT 3,H	CB90	RES 2,B
CB21	SLA C	CB5D	BIT 3,L	CB91	RES 2,C
CB22	SLA D	CB5E	BIT 3,(HL)	CB92	RES 2,D
CB23	SLA E	CB5F	BIT 3,A	CB93	RES 2,E
CB24	SLA H	CB60	BIT 4,B	CB94	RES 2,H
CB25	SLA L	CB61	BIT 4,C	CB95	RES 2,L
CB26	SLA (HL)	CB62	BIT 4,D	CB96	RES 2,(HL)
CB27	SLA A	CB63	BIT 4,E	CB97	RES 2,A
CB28	SRA B	CB64	BIT 4,H	CB98	RES 3,B
CB29	SRA C	CB65	BIT 4,L	CB99	RES 3,C
CB2A	SRA D	CB66	BIT 4,(HL)	CB9A	RES 3,D
CB2B	SRA E	CB67	BIT 4,A	CB9B	RES 3,E
CB2C	SRA H	CB68	BIT 5,B	CB9C	RES 3,H
CB2D	SRA L	CB69	BIT 5,C	CB9D	RES 3,L
CB2E	SRA (HL)	CB6A	BIT 5,D	CB9E	RES 3,(HL)
CB2F	SRA A	CB6B	BIT 5,E	CB9F	RES 3,A
CB38	SRL B	CB6C	BIT 5,H	CBA0	RES 4,B
CB39	SRL C	CB6D	BIT 5,L	CBA1	RES 4,C
CB3A	SRL D	CB6E	BIT 5,(HL)	CBA2	RES 4,D
CB3B	SRL E	CB6F	BIT 5,A	CBA3	RES 4,E
CB3C	SRL H	CB70	BIT 6,B	CBA4	RES 4,H
CB3D	SRL L	CB71	BIT 6,C	CBA5	RES 4,L
CB3E	SRL (HL)	CB72	BIT 6,D	CBA6	RES 4,(HL)
CB3F	SRL A	CB73	BIT 6,E	CBA7	RES 4,A
CB40	BIT 0,B	CB74	BIT 6,H	CBA8	RES 5,B
CB41	BIT 0,C	CB75	BIT 6,L	CBA9	RES 5,C
CB42	BIT 0,D	CB76	BIT 6,(HL)	CBAA	RES 5,D
CB43	BIT 0,E	CB77	BIT 6,A	CBAB	RES 5,E
CB44	BIT 0,H	CB78	BIT 7,B	CBAC	RES 5,H
CB45	BIT 0,L	CB79	BIT 7,C	CBAD	RES 5,L
CB46	BIT 0,(HL)	CB7A	BIT 7,D	CBAE	RES 5,(HL)
CB47	BIT 0,A	CB7B	BIT 7,E	CBAF	RES 5,A
CB48	BIT 1,B	CB7C	BIT 7,H	CBB0	RES 6,B
CB49	BIT 1,C	CB7D	BIT 7,L	CBB1	RES 6,C
CB4A	BIT 1,D	CB7E	BIT 7,(HL)	CBB2	RES 6,D
CB4B	BIT 1,E	CB7F	BIT 7,A	CBB3	RES 6,E
CB4C	BIT 1,H	CB80	RES 0,B	CBB4	RES 6,H
CB4D	BIT 1,L	CB81	RES 0,C	CBB5	RES 6,L
CB4E	BIT 1,(HL)	CB82	RES 0,D	CBB6	RES 6,(HL)

(nn) = Address of memory location d = displacement
nn = Data (16 bit) d2 = d − 2
n = Data (8-bit)

12.16 Instruction Set: Numerical Order (Continued)

Op Code	Mnemonic
CBB7	RES 6,A
CBB8	RES 7,B
CBB9	RES 7,C
CBBA	RES 7,D
CBBB	RES 7,E
CBBC	RES 7,H
CBBD	RES 7,L
CBBE	RES 7,(HL)
CBBF	RES 7,A
CBC0	SET 0,B
CBC1	SET 0,C
CBC2	SET 0,D
CBC3	SET 0,E
CBC4	SET 0,H
CBC5	SET 0,L
CBC6	SET 0,(HL)
CBC7	SET 0,A
CBC8	SET 1,B
CBC9	SET 1,C
CBCA	SET 1,D
CBCB	SET 1,E
CBCC	SET 1,H
CBCD	SET 1,L
CBCE	SET 1,(HL)
CBCF	SET 1,A
CBD0	SET 2,B
CBD1	SET 2,C
CBD2	SET 2,D
CBD3	SET 2,E
CBD4	SET 2,H
CBD5	SET 2,L
CBD6	SET 2,(HL)
CBD7	SET 2,A
CBD8	SET 3,B
CBD9	SET 3,C
CBDA	SET 3,D
CBDB	SET 3,E
CBDC	SET 3,H
CBDD	SET 3,L
CBDE	SET 3,(HL)
CBDF	SET 3,A
CBE0	SET 4,B
CBE1	SET 4,C
CBE2	SET 4,D
CBE3	SET 4,E
CBE4	SET 4,H
CBE5	SET 4,L
CBE6	SET 4,(HL)
CBE7	SET 4,A
CBE8	SET 5,B
CBE9	SET 5,C
CBEA	SET 5,D
CBEB	SET 5,E
CBEC	SET 5,H
CBED	SET 5,L
CBEE	SET 5,(HL)
CBEF	SET 5,A
CBF0	SET 6,B
CBF1	SET 6,C
CBF2	SET 6,D
CBF3	SET 6,E
CBF4	SET 6,H
CBF5	SET 6,L
CBF6	SET 6,(HL)
CBF7	SET 6,A
CBF8	SET 7,B
CBF9	SET 7,C
CBFA	SET 7,D
CBFB	SET 7,E
CBFC	SET 7,H
CBFD	SET 7,L
CBFE	SET 7,(HL)
CBFF	SET 7,A
CCnn	CALL Z,nn
CDnn	CALL nn
CEn	ADC A,n
CF	RST 8
D0	RET NC
D1	POP DE
D2nn	JP NC,nn
D3n	OUT (n),A
D4nn	CALL NC,nn
D5	PUSH DE
D6n	SUB n
D7	RST 10H
D8	RET C
D9	EXX
DAnn	JP,C,nn
DBn	IN A,(n)
DCnn	CALL C,nn
DD09	ADD IX,BC
DD19	ADD IX,DE
DD21nn	LD IX,nn
DD22nn	LD (nn),IX
DD23	INC IX
DD29	ADD IX,IX
DD2Ann	LD IX,(nn)
DD2B	DEC IX
DD34d	INC (IX+d)
DD35d	DEC (IX+d)
DD36dn	LD (IX+d),n
DD39	ADD IX,SP
DD46d	LD B,(IX+d)
DD4Ed	LD C,(IX+d)
DD56d	LD D,(IX+d)
DD5Ed	LD E,(IX+d)
DD66d	LD H,(IX+d)
DD6Ed	LD L,(IX+d)
DD70d	LD (IX+d),B
DD71d	LD (IX+d),C
DD72d	LD (IX+d),D
DD73d	LD (IX+d),E
DD74d	LD (IX+d),H
DD75d	LD (IX+d),L
DD77d	LD (IX+d),A
DD7Ed	LD A,(IX+d)
DD86d	ADD A,(IX+d)
DD8Ed	ADC A,(IX+d)
DD96d	SUB (IX+d)
DD9Ed	SBC A,(IX+d)
DDA6d	AND (IX+d)
DDAEd	XOR (IX+d)
DDB6d	OR (IX+d)
DDBEd	CP (IX+d)
DDCBd06	RLC (IX+d)
DDCBd0E	RRC (IX+d)
DDCBd16	RL (IX+d)
DDCBd1E	RR (IX+d)
DDCBd26	SLA (IX+d)
DDCBd2E	SRA (IX+d)
DDCBd3E	SRL (IX+d)
DDCBd46	BIT 0,(IX+d)
DDCBd4E	BIT 1,(IX+d)
DDCBd56	BIT 2,(IX+d)
DDCBd5E	BIT 3,(IX+d)
DDCBd66	BIT 4,(IX+d)
DDCBd6E	BIT 5,(IX+d)
DDCBd76	BIT 6,(IX+d)
DDCBd7E	BIT 7,(IX+d)
DDCBd86	RES 0,(IX+d)
DDCBd8E	RES 1,(IX+d)
DDCBd96	RES 2,(IX+d)
DDCBd9E	RES 3,(IX+d)
DDCBdA6	RES 4,(IX+d)
DDCBdAE	RES 5,(IX+d)
DDCBdB6	RES 6,(IX+d)
DDCBdBE	RES 7,(IX+d)
DDCBdC6	SET 0,(IX+d)
DDCBdCE	SET 1,(IX+d)
DDCBdD6	SET 2,(IX+d)
DDCBdDE	SET 3,(IX+d)
DDCBdE6	SET 4,(IX+d)
DDCBdEE	SET 5,(IX+d)
DDCBdF6	SET 6,(IX+d)
DDCBdFE	SET 7,(IX+d)
DDE1	POP IX
DDE3	EX (SP),IX
DDE5	PUSH IX
DDE9	JP (IX)

(nn) = Address of memory location
nn = Data (16 bit)
n = Data (8-bit)
d = displacement
d2 = d−2

12.16 Instruction Set: Numerical Order (Continued)

Op Code	Mnemonic
DDF9	LD SP,IX
DEn	SCB A,n
DF	RST 18H
E0	RET PO
E1	POP HL
E2nn	JP PO,nn
E3	EX (SP),HL
E4nn	CALL PO,nn
E5	PUSH HL
E6n	AND n
E7	RST 20H
E8	RET PE
E9	JP (HL)
EAnn	JP PE,nn
EB	EX DE,HL
ECnn	CALL PE,nn
ED40	IN B,(C)
ED41	OUT (C),B
ED42	SBC HL,BC
ED43nn	LD (nn),BC
ED44	NEG
ED45	RETN
ED46	IM 0
ED47	LD I,A
ED48	IN C,(C)
ED49	OUT (C),C
ED4A	ADC HL,BC
ED4Bnn	LD BC,(nn)
ED4D	RETI
ED50	IN D,(C)
ED51	OUT (C),D
ED52	SBC HL,DE
ED53nn	LD (nn),DE
ED56	IM 1
ED57	LD A,I
ED58	IN E,(C)
ED59	OUT (C), E
ED5A	ADC HL,DE
ED5Bnn	LD DE,(nn)
ED5E	IM 2
ED60	IN H,(C)
ED61	OUT (C),H
ED62	SBC HL,HL
ED67	RRD
ED68	IN L,(C)
ED69	OUT (C),L
ED6A	ADC HL,HL
ED6F	RLD
ED72	SBC HL,SP
ED73nn	LD (nn),SP
ED78	IN A,(C)
ED79	OUT (C),A
ED7A	ADC HL,SP
ED7Bnn	LD SP,(nn)
EDA0	LDI
EDA1	CPI
EDA2	INI
EDA3	OUTI
EDA8	LDD
EDA9	CPD
EDAA	IND
EDAB	OUTD
EDB0	LDIR
EDB1	CPIR
EDB2	INIR
EDB3	OTIR
EDB8	LDDR
EDB9	CPDR
EDBA	INDR
EDBB	OTDR
EEn	XOR n
EF	RST 28H
F0	RET P
F1	POP AF
F2nn	JP P,nn
F3	DI
F4nn	CALL P,nn
F5	PUSH AF
F6n	OR n
F7	RST 30H
F8	RET M
F9	LD SP,HL
FAnn	JP M,nn
FB	EI
FCnn	CALL M,nn
FD09	ADD IY,BC
FD19	ADD IY,DE
FD21nn	LD IY,nn
FD22nn	LD (nn),IY
FD23	INC IY
FD29	ADD IY,IY
FD2Ann	LD IY,(nn)
FD2B	DEC IY
FD34d	INC (IY+d)
FD35d	DEC (IY+d)
FD36dn	LD (IY+d),n
FD39	ADD IY,SP
FD46d	LD B,(IY+d)
FD4Ed	LD C,(IY+d)
FD56d	LD D,(IY+d)
FD5Ed	LD E,(IY+d)
FD66d	LD H,(IY+d)
FD6Ed	LD L,(IY+d)
FD70d	LD (IY+d),B
FD71d	LD (IY+d),C
FD72d	LD (IY+d),D
FD73d	LD (IY+d),E
FD74d	LD (IY+d),H
FD75d	LD (IY+d),L
FD77d	LD (IY+d),A
FD7Ed	LD A,(IY+d)
FD86d	ADD A,(IY+d)
FD8Ed	ADC A,(IY+d)
FD96d	SUB (IY+d)
FD9Ed	SBC A,(IY+d)
FDA6d	AND (IY+d)
FDAEd	XOR (IY+d)
FDB6d	OR (IY+d)
FDBEd	CP (IY+d)
FDE1	POP IY
FDE3	EX (SP), IY
FDE5	PUSH IY
FDE9	JP (IY)
FDF9	LD SP,IY
FDCBd06	RLC (IY+d)
FDCBd0E	RRC (IY+d)
FDCBd16	RL (IY+d)
FDCBd1E	RR (IY+d)
FDCBd26	SLA (IY+d)
FDCBd2E	SRA (IY+d)
FDCBd3E	SRL (IY+d)
FDCBd46	BIT 0,(IY+d)
FDCBd4E	BIT 1,(IY+d)
FDCBd56	BIT 2,(IY+d)
FDCBd5E	BIT 3,(IY+d)
FDCBd66	BIT 4,(IY+d)
FDCBd6E	BIT 5,(IY+d)
FDCBd76	BIT 6,(IY+d)
FDCBd7E	BIT 7,(IY+d)
FDCBd86	RES 0,(IY+d)
FDCBd8E	RES 1,(IY+d)
FDCBd96	RES 2,(IY+d)
FDCBd9E	RES 3,(IY+d)
FDCBdA6	RES 4,(IY+d)
FDCBdAE	RES 5,(IY+d)
FDCBdB6	RES 6,(IY+d)
FDCBdBE	RES 7,(IY+d)
FDCBdC6	SET 0,(IY+d)
FDCBdCE	SET 1,(IY+d)
FDCBdD6	SET 2,(IY+d)
FDCBdDE	SET 3,(IY+d)
FDCBdE6	SET 4,(IY+d)
FDCBdEE	SET 5,(IY+d)
FDCBdF6	SET 6,(IY+d)
FDCBdFE	SET 7,(IY+d)
FEn	CP n
FF	RST 38H

(nn) = Address of memory location d = displacement
nn = Data (16 bit) d2 = d−2
n = Data (8-bit)

13.0 Data Acquisition System

A natural application for the NSC800 is one that requires remote operation. Since power consumption is low if the system consists of only CMOS components, the entire package can conceivably operate from only a battery power source. In the application described herein, the only source of power will be from a battery pack composed of a stacked array of NiCad batteries (see *Figure 20*).

The application is that of a remote data acquisition system. Extensive use is made of some of the other LSI CMOS components manufactured by National: notably the ADC0816 and MM58167. The ADC0816 is a 16-channel analog-to-digital converter which operates from a 5V source. The MM58167 is a microprocessor-compatible real-time clock (RTC). The schematic for this system is shown in *Figure 20*. All the necessary features of the system are contained in six integrated circuits: NSC800, NSC810A, NSC831, HN6136P, ADC0816, and MM58167. Some other small scale integration CMOS components are used for normal interface requirements. To reduce component count, linear selection techniques are used to generate chip selects for the NSC810A and NSC831. Included also is a current loop communication link to enable the remote system to transfer data collected to a host system.

In order to keep component count low and maximize effectiveness, many of the features of the NSC800 family have been utilized. The RAM section of the NSC810A is used as a data buffer to store intermediate measurements and as scratch pad memory for calculations. Both timers contained in the NSC810A are used to produce the clocks required by the A/D converter and the RTC. The Power-Save feature of the NSC800 makes it possible to reduce system power consumption when it is not necessary to collect any data. One of the analog input channels of the A/D is connected to the battery pack to enable the CPU to monitor its own voltage supply and notify the host that a battery change is needed.

In operation, the NSC800 makes readings on various input conditions through the ADC0816. The type of devices connected to the A/D input depends on the nature of the remote environment. For example, the duties of the remote system might be to monitor temperature variations in a large building. In this case, the analog inputs would be connected to temperature transducers. If the system is situated in a process control environment, it might be monitoring fluid flow, temperatures, fluid levels, etc. In either case, operation would be necessary even if a power failure occurred, thus the need for battery operation or at least battery backup. At some fixed times or at some particular time durations, the system takes readings by selecting one of the analog input channels, commands the A/D to perform a conversion, reads the data, and then formats it for transmission; or, the system checks the readings against set points and transmits a warning if the set points are exceeded. With the addition of the RTC, the host need not command the remote system to take these readings each time it is necessary. The NSC800 could simply set up the RTC to interrupt it at a previously defined time and when the interrupt occurs, make the readings. The resultant values could be stored in the NSC810A for later correlation. In the example of temperature monitoring in a building, it might be desired to know the high and low temperatures for a 12-hour period. After compiling the information, the system could dump the data to the host over the communications link. Note from the schematic that the current for the communication link is supplied by the host to remove the constant current drain from the battery supply.

The required clocks for the two peripheral devices are generated by the two timers in the NSC810A. Through the use of various divisors, the master clock generated by the NSC800 is divided down to produce the clocks. Four examples are shown in the table following *Figure 20*.

All the crystal frequencies are standard frequencies. The various divisors listed are selected to produce, from the master clock frequency of the NSC800, an exact 32,768 Hz clock for the MM58167 and a clock within the operating range of the A/D converter.

The MM58167 is a programmable real-time clock that is microprocessor compatible. Its data format is BCD. It allows the system to program its interrupt register to produce an interrupt output either on a time of day match (which includes the day of the week, the date and month) and/or every month, week, day, hour, minute, second, or tenth of a second. With this capability added to the system, precise time of day measurements are possible without having the CPU do timekeeping. The interrupt output can be connected, through the use of one port bit of the NSC810A, to put the CPU in the power-save mode and reenable it at a preset time. The interrupt output is also connected to one of the hardware restart inputs ($\overline{RSTB}$) to enable time duration measurements. This power-down mode of operation would not be possible if the NSC800 had the duties of timekeep-

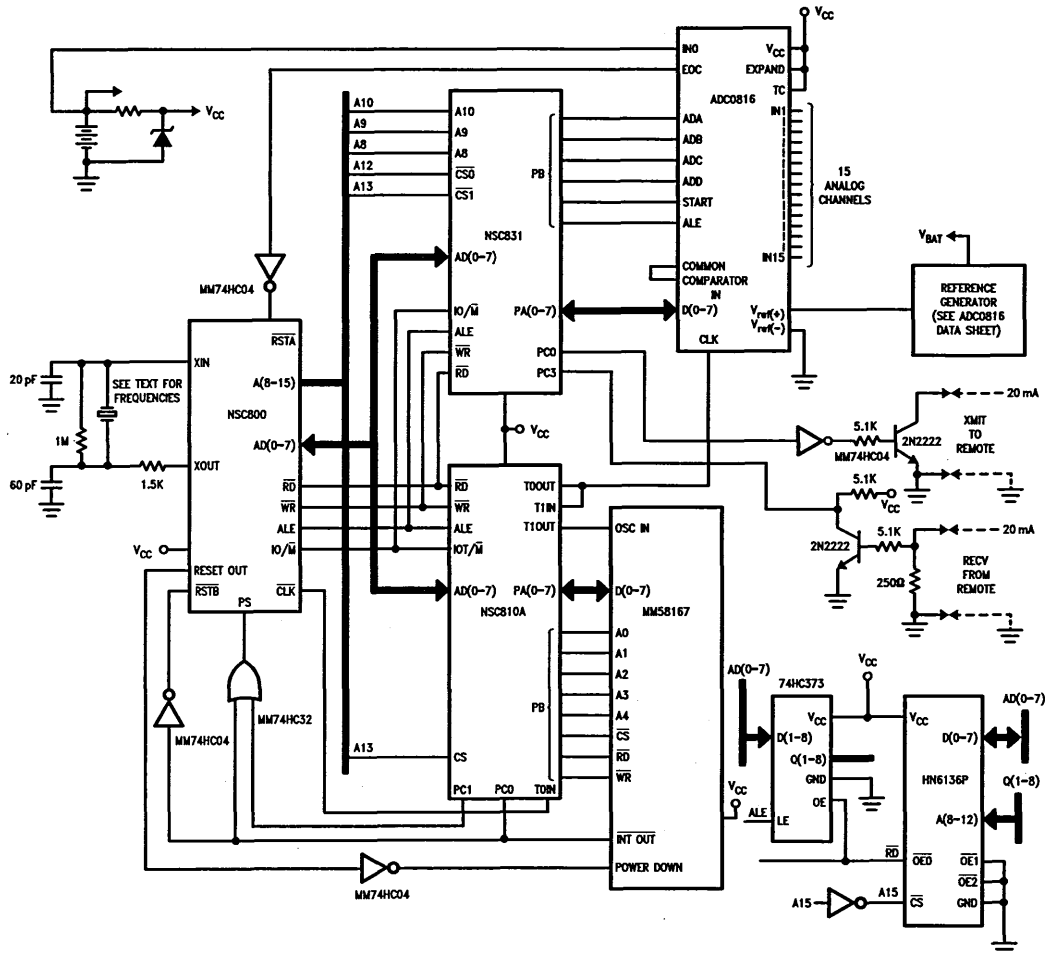


FIGURE 20. Remote Data Acquisition

13.0 Data Acquisition System (Continued)

ing. When in the power-save mode, the system power requirements are decreased by about 50%, thus extending battery life.

Communication with the peripheral devices (MM58167 and ADC0816) is accomplished through the I/O ports of the NSC810A and NSC831. The peripheral devices are not connected to the bus of the NSC800 as they are not directly compatible with a multiplexed bus structure. Therefore, additional components would be required to place them on the microprocessor bus. Writing data into the MM58167 is performed by first putting the desired data on Port A, followed by selecting the address of the internal register and applying the chip select through the use of Port B. A bit set and clear operation is performed to emulate a pulse on the bit of Port B connected to the $\overline{WR}$ input of the MM58167. For a read operation, the same sequence of operations is performed except that Port A is set for the input mode of operation and the $\overline{RD}$ line is pulsed. Similar techniques are used to read converted data from the A/D converter. When a conversion is desired, the CPU selects a channel and commands the ADC0816 to start a conversion. When the conversion is complete, the converter will produce an End-of-Conversion signal which is connected to the $\overline{RSTA}$ interrupt input of the NSC800.

When operating, the system shown consumes about 125 mw. When in the power-save mode, power consumption is decreased to about 70 mw. If, as is likely, the system is in the power-save mode most of the time, battery life can be quite long depending on the amp-hour rating of the batteries incorporated into the system. For example, if the battery pack is rated at 5 amp-hours, the system should be able to operate for about 400-500 hours before a battery charge or change is required.

As shown in the schematic (refer to *Figure 20*), analog input IN0 is connected to the battery source. In this way, the CPU can monitor its own power source and notify the host that it needs a battery replacement or charge. Since the battery source shown is a stacked array of 7 NiCads producing 8.4V, the converter input is connected in the middle so that it can take a reading on two or three of the cells. Since NiCad batteries have a relatively constant voltage output until very nearly discharged, the CPU can sense that the "knee" of the discharge curve has been reached and notify the host.

Typical Timer Output Frequencies

Crystal Frequency	CPU Clock Output	Timer 0 Output	Timer 1 Output
2.097152 MHz	1.048576 MHz	262.144 kHz divisor = 4	32.768 kHz divisor = 8
3.276800 MHz	1.638400 MHz	327.680 kHz divisor = 5	32.768 kHz divisor = 10
4.194304 MHz	2.097152 MHz	262.144 kHz divisor = 8	32.768 kHz divisor = 8
4.915200 MHz	2.457600 MHz	491.520 kHz divisor = 5	32.768 kHz divisor = 15

14.0 NSC800M/883B MIL-STD-833 Class C Screening

National Semiconductor offers the NSC800D and NSC800E with full class B screening per MIL-STD-883 for Military/ Aerospace programs requiring high reliability. In addition, this screening is available for all of the key NSC800 peripheral devices.

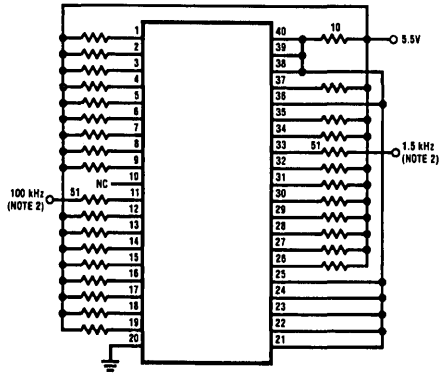
Electrical testing is performed in accordance with RESTS800X, which tests or guarantees all of the electrical performance characteristics of the NSC800 data sheet. A copy of the current revision of RETS800X is available upon request.

100% Screening Flow

Test	MIL-STD-883 Method/Condition	Requirement
Internal Visual	2010B	100%
Stabilization Bake	1008 C 24 Hrs. @ +150°C	100%
Temperature Cycling	1010 C 10 Cycles −65°C/+150°C	100%
Constant Acceleration	2001 E 30,000 G's, Y1 Axis	100%
Fine Leak	1014 A or B	100%
Gross Leak	1014C	100%
Burn-In	1015 160 Hrs. @ +125°C (using burn-in circuits shown below)	100%
Final Electrical	+25°C DC per RETS800X	100%
PDA	10% Max	
	+125°C AC and DC per RETS800X	100%
	−55°C AC and DC per RETS800X	100%
	+25°C AC per RETS800X	100%
QA Acceptance	5005	Sample Per
Quality Conformance		Method 5005
External Visual	2009	100%

15.0 Burn-In Circuits

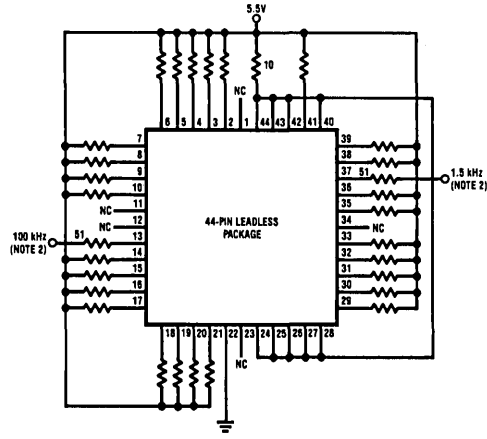
5240HR
NSC800D/883B (Dual-In-Line)



TL/C/5171-32

Top View

5241HR
NSC800E/883B (Leadless Chip Carrier)



TL/C/5171-33

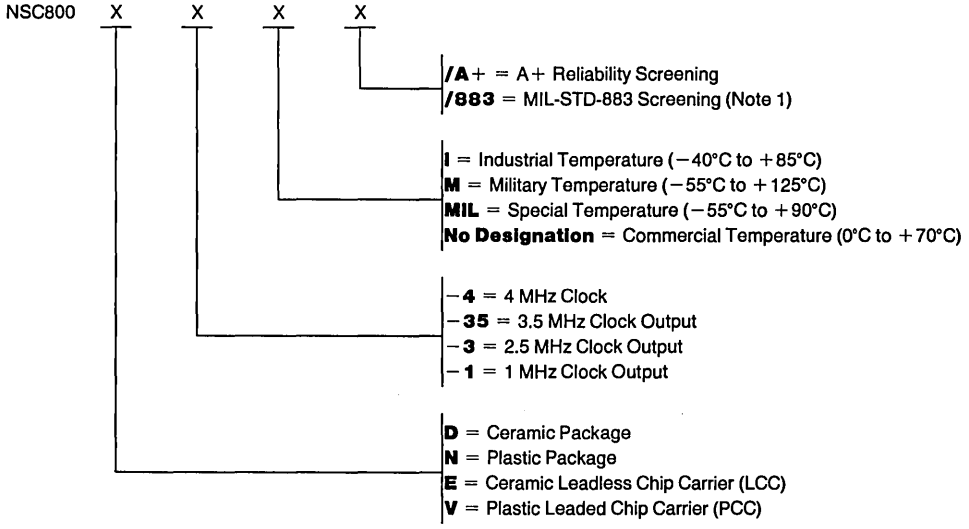
All resistors 2.7 kΩ unless marked otherwise.

Note 1: All resistors are ¼W ± 5% unless otherwise specified.

Note 2: All clocks 0V to 3V, 50% duty cycle, in phase with < 1 μs rise and fall time.

Note 3: Device to be cooled down under power after burn-in.

16.0 Ordering Information



Note 1: Do not specify a temperature option; all parts are screened to military temperature.

17.0 Reliability Information

Gate Count 2750

Transistor Count 11,000

National Semiconductor

NSC810A RAM-I/O-Timer

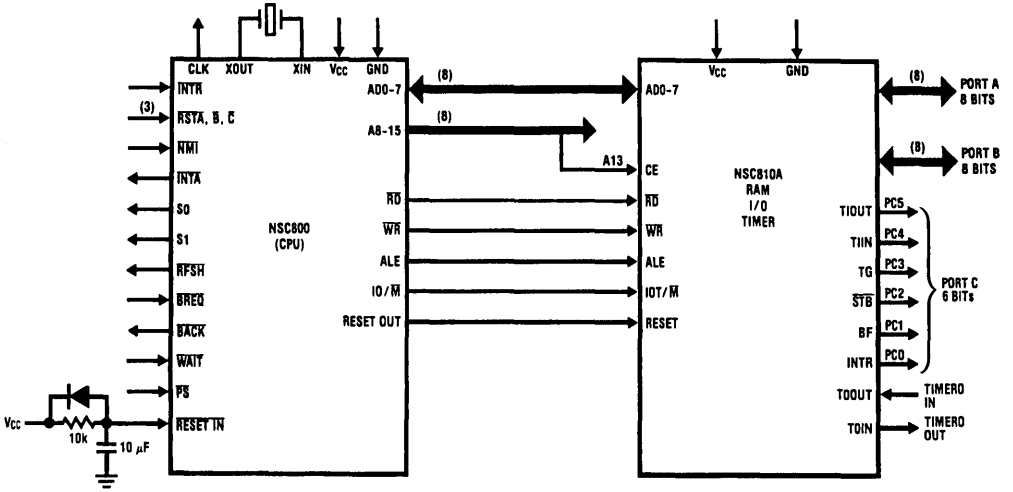
General Description

The NSC810A, the luxury model of our NSC800™ peripheral line, sports triple ported I/O, dual 16-bit timers and a 1024-bit static storage area. The three ports can be combined for a total of 22 general purpose I/O lines. In addition, port A has several strobed mode operations. Note the single instruction I/O bit operations for quick and efficient data handling from the ports. The timers feature 6 modes of operation and prescalers for those complicated timing tasks. The NSC810A comes in two models: the Dual-In-Line (DIP) and the surface mount chip carrier (LCC). It also comes in three exciting temperature ranges (Commercial, Industrial, and Military) and two reliability flows (extended burn-in and military class B in accordance with Method 5004 of MIL-STD-883). This is brought to you through the microCMOS silicon gate technology of National Semiconductor.

Features

- Three programmable I/O ports
- Dual 16-bit programmable counter/timers
- 2.4V–6.0V power supply
- Very low power consumption
- Fully static operation
- Single-instruction I/O bit operations
- Timer operation—DC to 5 MHz
- Bus compatible with NSC800™ family
- Speed: compatible with NSC800
  NSC810A-4 → NSC800-4 @ 4.0 MHz
  NSC810A-3 → NSC800 @ 2.5 MHz
  NSC810A-1 → NSC800-1 @ 1.0 MHz

NSC810A Connection Diagram



TL/C/5517–1

Table of Contents

1.0 Absolute Maximum Ratings

(Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Storage Temperature Range	−65°C to +150°C
Voltage at Any Pin with Respect to Ground	−0.3V to V_{CC} + 0.3V
V_{CC}	7V
Power Dissipation	1W
Lead Temperature (Soldering, 10 seconds)	300°C

2.0 Operating Conditions

V_{CC} = 5V ± 10%

NSC810A-1 → 0°C to +70°C
−40°C to +85°C

NSC810A-3 → 0°C to +70°C
−40°C to +85°C
−55°C to +125°C

NSC810A-4 → 0°C to +70°C
−40°C to +85°C
−55°C to +125°C

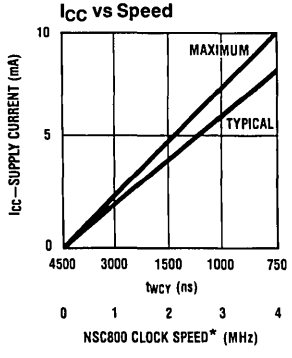
3.0 DC Electrical Characteristics

V_{CC} = 5V ±10%, GND = 0V, unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Logical 1 Input Voltage		0.8 V_{CC}		V_{CC}	V
V_{IL}	Logical 0 Input Voltage		0		0.2 V_{CC}	V
V_{OH}	Logical 1 Output Voltage	I_{OH} = −1.0 mA I_{OUT} = −10 μA	2.4 V_{CC}−0.5			V V
V_{OL}	Logical 0 Output Voltage	I_{OL} = 2 mA I_{OUT} = 10 μA	0 0		0.4 0.1	V V
I_{IL}	Input Leakage Current	0 ≤ V_{IN} ≤ V_{CC}	−10.0		10.0	μA
I_{OL}	Output Leakage Current	0 ≤ V_{IN} ≤ V_{CC}	−10.0		10.0	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, Timer = Mode 1, T0IN = T1IN = 2.5 Mhz, t_{WCY} = 750 ns, T_A = 25°C		8	10	mA
I_Q	Quiescent Current	No Input Switching, T_A = 25°C, RESET = 0, IO/$\overline{M}$ = 1, $\overline{RD}$ = 1, $\overline{WR}$ = 1, ALE = 1, V_{IN} = V_{CC}, t_{IN} = 0 Hz, t_{OUT} = 0		10	100	μA
C_{IN}	Input Capacitance			4	7	pF
C_{OUT}	Output Capacitance			6	10	pF
V_{CC}	Power Supply Voltage	(Note 2)	2.4	5	6	V
V_{DRV}	Data Retention Voltage		1.8			V

Note 1: Absolute maximum ratings are those values beyond which the safety of the device cannot be guaranteed. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under DC Electrical Characteristics.

Note 2: Operation at lower power supply voltages will reduce the maximum operating speed. Operation at voltages other than 5V ±10% is guaranteed by design, not tested.



TL/C/5517-2

*When NSC810A is used with NSC800

4.0 AC Electrical Characteristics $V_{CC}=5V \pm 10\%$, GND = 0V

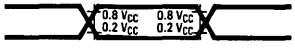
Symbol	Parameter	Conditions	NSC810A-1		NSC810A-3		NSC810-4		Units
			Min	Max	Min	Max	Min	Max	
t_{ACC}	Access Time from ALE	C_L = 150 pF		1000		400		300	ns
t_{AH}	AD0-7, CE, IOT/$\overline{M}$ Hold Time		100		60		30		ns
t_{ALE}	ALE Strobe Width (High)		200		125		100		ns
t_{ARW}	ALE to $\overline{RD}$ or $\overline{WR}$ Strobe		150		120		75		ns
t_{AS}	AD0-7, CE, IOT/$\overline{M}$ Set-Up Time		100		45		25		ns
t_{DH}	Data Hold Time		150		90		40		ns
t_{DO}	Port Data Output Valid			350		310		300	ns
t_{DS}	Data Set-Up Time		100		80		50		ns
t_{PE}	Peripheral Bus Enable			320		200		200	ns
t_{PH}	Peripheral Data Hold Time		150		125		100		ns
t_{PS}	Peripheral Data Set-Up Time		100		75		50		ns
t_{PZ}	Peripheral Bus Disable (TRI-STATE®)			150		150		150	ns
t_{RB}	$\overline{RD}$ to BF Invalid			300		300		300	ns
t_{RD}	Read Strobe Width		400		320		185		ns
t_{RDD}	Data Bus Disable		0	100	0	100	0	75	ns
t_{RI}	$\overline{RD}$ to $\overline{INTR}$ Output			320		320		300	ns
t_{RWA}	$\overline{RD}$ or $\overline{WR}$ to Next ALE		125		100		75		ns
t_{SB}	$\overline{STB}$ to BF Valid			300		300		300	ns
t_{SH}	Peripheral Data Hold with Respect to $\overline{STB}$		150		125		100		ns
t_{SI}	$\overline{STB}$ to $\overline{INTR}$ Output			300		300		300	ns
t_{SS}	Peripheral Data Set-Up with Respect to $\overline{STB}$		100		75		50		ns
t_{SW}	$\overline{STB}$ Width		400		320		220		ns
t_{WB}	$\overline{WR}$ to BF Output			340		340		300	ns
t_{WI}	$\overline{WR}$ to $\overline{INTR}$ Output			320		320		300	ns
t_{WR}	$\overline{WR}$ Strobe Width		400		320		220		ns
t_{WCY}	Width of Machine Cycle		3000		1200		750		ns

Note: Test conditions: t_{WCY} = 3000 ns for NSC810A-1, 1200 ns for NSC810A-3, 750 ns for NSC810A-4

5.0 Timer AC Electrical Characteristics

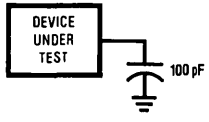
Symbol	Parameter	Conditions	Min	Typ	Max	Units
F_C	Clock Frequency		DC		2.5	MHz
F_{CP}	Clock Frequency	Prescale Selected	DC		5.0	MHz
t_{CW}	Clock Pulse Width		150			ns
t_{CWP}	Clock Pulse Width	Prescale Selected	75			ns
t_{GS}	Gate Set-Up Time	With Respect to Negative Clock Edge	100			ns
t_{GH}	Gate Hold Time	With Respect to Negative Clock Edge	250			ns
t_{CO}	Clock to Output Delay	C_L = 100 pF			350	ns

AC TESTING INPUT/OUTPUT WAVEFORM



TL/C/5517-3

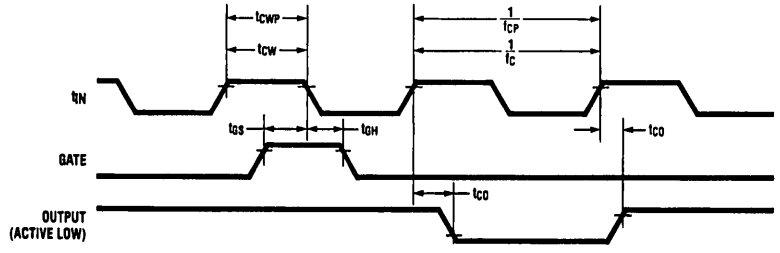
AC TESTING LOAD CIRCUIT



TL/C/5517-4

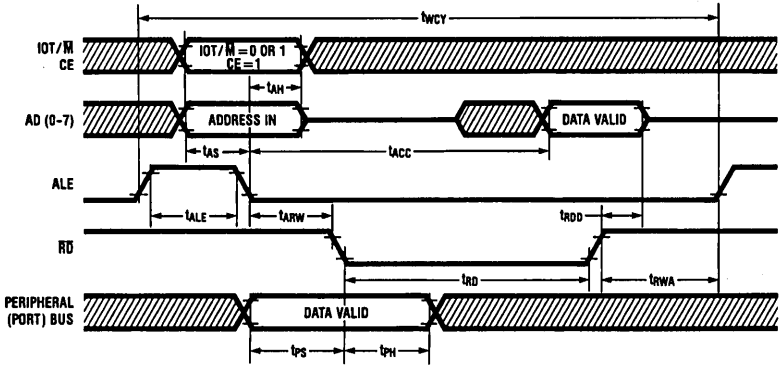
6.0 Timing Waveforms

Timer Waveforms



TL/C/5517–5

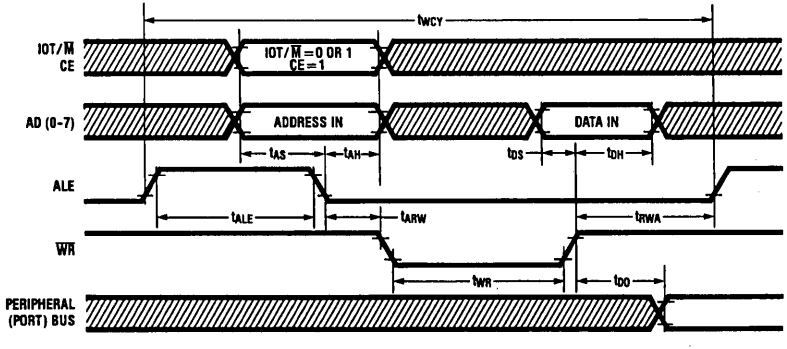
Read Cycle (Read from RAM, Port or Timer)



TL/C/5517–6

Note: Diagonal lines indicate interval of invalid data.

Write Cycle (Write to RAM, Port or Timer)

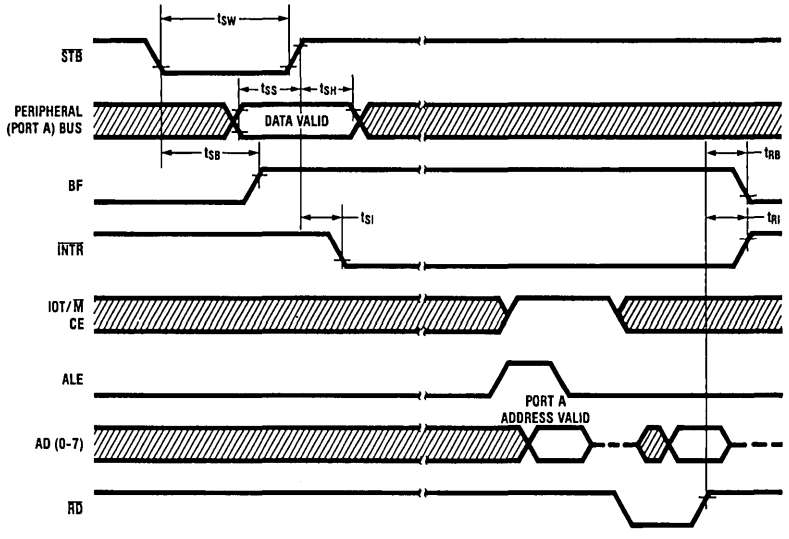


TL/C/5517–7

Note: Diagonal lines indicate interval of invalid data.

6.0 Timing Waveforms (Continued)

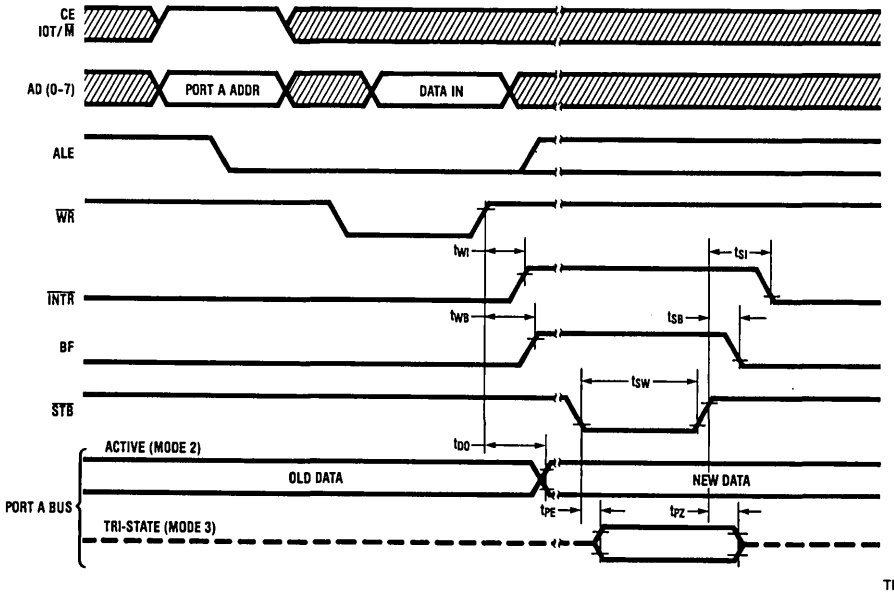
Strobed Mode Input



TL/C/5517–8

Note: Diagonal lines indicate interval of invalid data.

Strobed Mode Output



TL/C/5517–9

Note: Diagonal lines indicate interval of invalid data.

7.0 Pin Descriptions

The function and mnemonic for the NSC810A signals are described below:

7.1 INPUT SIGNALS

Reset (RESET): RESET is an active-high input that resets all registers to 0 (low). The RAM contents remain unaltered.

Input/Output Timer or RAM Select (IOT/$\overline{M}$): IOT/$\overline{M}$ is an I/O memory select input line. A logic 1 (high) input selects the I/O-timer portion of the chip; a logic 0 (low) input selects the RAM portion of the chip. IOT/$\overline{M}$ is latched at the falling edge of ALE.

Chip Enable (CE): CE is an active-high input that allows access to the NSC810A. CE is latched at the falling edge of ALE.

Read ($\overline{RD}$): The $\overline{RD}$ is an active-low input that enables a read operation of the RAM or I/O-timer location.

Write ($\overline{WR}$): The $\overline{WR}$ is an active-low input that enables a write operation to RAM or I/O-timer locations.

Address Latch Enable (ALE): The falling edge of the ALE input latches AD0–AD7, CE and IOT/$\overline{M}$ inputs to form the address for RAM, I/O or timer.

Timer 0 Input (T0IN): T0IN is the clock input for timer 0.

7.2 OUTPUT SIGNALS

Timer 0 Output (T0OUT): T0OUT is the programmable output of timer 0. After reset, T0OUT is set high.

7.3 POWER SUPPLY SIGNALS

Positive DC Voltage (V_{CC}): V_{CC} is the 5V supply pin.

Ground (GND): Ground reference pin.

7.4 INPUT/OUTPUT SIGNALS

Address/Data Bus (AD0–AD7): The multiplexed bidirectional address/data bus; AD0–AD7 pins, are in the high impedance state when the NSC810A is not selected. AD0–AD7 will latch address inputs at the falling edge of ALE. The address will designate a location in RAM, I/O or timer. $\overline{WR}$ input enables 8-bit data to be written into the addressed location. $\overline{RD}$ input enables 8-bit data to be read from the addressed location. The $\overline{RD}$ or $\overline{WR}$ inputs occur while ALE is low.

Port A, 0–7 (PA0–PA7): Port A is an 8-bit basic mode input/output port, also capable of strobed mode I/O utilizing three control signals from port C. Strobed mode of operation on port A has three different modes; strobed input, strobed output with active peripheral bus, strobed output with TRI-STATE peripheral bus.

Port B, 0–7 (PB0–PB7): Port B is an 8-bit basic mode input/output port.

Port C, 0–5 (PC0–PC5): Port C is a 6-bit basic mode I/O port. Each pin has a programmable second function, as follows:

PC0/$\overline{INTR}$: $\overline{INTR}$ is an active-low, strobed mode interrupt request to the Central Processor Unit (CPU).

PC1/BF: BF is an active-high, strobed mode, buffer full output to peripheral devices.

PC2/$\overline{STB}$: $\overline{STB}$ is an active-low, strobed mode input from peripheral devices.

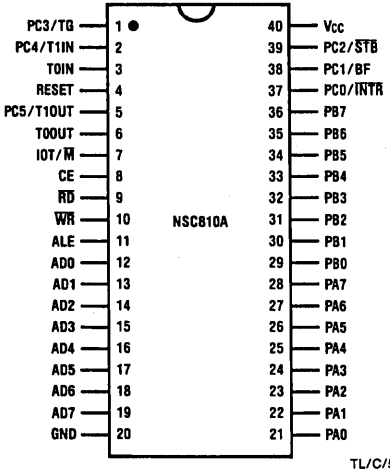
PC3/TG: TG is the timer gating signal.

PC4/T1IN: T1IN is the clock input for timer 1.

PC5/T1OUT: T1OUT is the programmable output of timer 1.

8.0 Connection Diagrams

Dual-In-Line Package

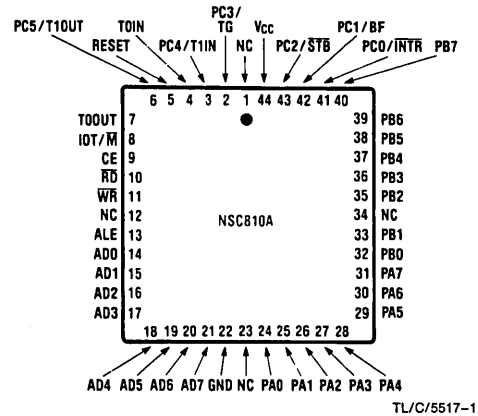


TL/C/5517–10

Top View

Order Number NSC810AD or NSC810AN
See NS Package Number D40C or N40A

Chip Carrier



TL/C/5517–11

Top View

NC = no connect

Order Number NSC810AE or NSC810AV
See NS Package Number E44B or V44A

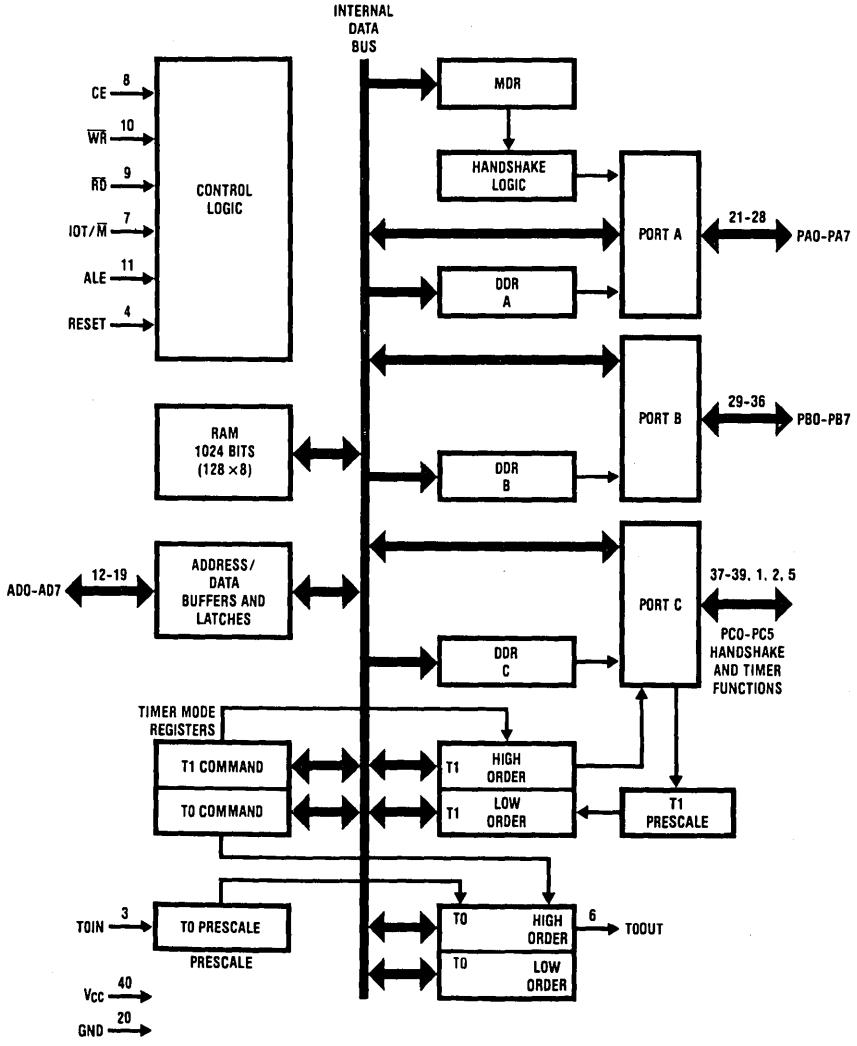
9.0 Functional Description

Figure 1 is a detailed block diagram of the NSC810A. The functional description that follows describes the RAM, I/O and TIMER sections.

9.1 RANDOM ACCESS MEMORY (RAM)

The memory portion of the RAM-I/O-timer is accessed by a 7-bit address input to pins AD0 through AD6. The IOT/$\overline{M}$ input must be low (RAM select) and the CE input must be high at the falling edge of ALE to address the RAM. Address bit AD7 is a "don't care" for RAM addressing. Timing for RAM read and write operations is shown in the timing diagrams. The RAM is 128 x 8.

9.2 DETAILED BLOCK DIAGRAM



TL/C/5517–12

FIGURE 1

9.0 Functional Description (Continued)

9.3 I/O PORTS

The three I/O ports, labeled A, B, and C, can be programmed to be almost any combination of Input and Output bits. Ports A and B are configured as 8 bits wide, while port C is 6 bits. There are four different modes of operation for the ports. Three of the modes are for timed transfer of data between the peripheral and the NSC810A, this is called strobed I/O. The fourth mode is for direct transfer without handshaking with the peripheral.

The NSC810A can be programmed to operate in four different modes. One of these modes (Basic I/O) allows direct transfer of I/O data without any handshaking between the NSC810A and the peripheral. The other three modes (Strobed I/O) provide for timed transfers of I/O data with handshaking between the NSC810A and the peripheral.

The determination of the mode, data direction and data is done by five registers which are, handily, under program control. The Mode Definition Register (MDR), oddly enough, determines which mode the device will operate in, while the Data Direction Register (DDR) establishes the direction of the data transfer. The Data register contains the data that is being sent or has been received. The other two registers (bit-set, bit-clear) allow the individual bits in the data register to be set or cleared without affecting the other bits. Each port has its own set of these registers, except the MDR which affects ports A and C only.

In the strobed I/O modes, port C bits 0, 1 and 2 function as INTR (for the processor), BF, and STB respectively.

9.3.1 Registers

As can be seen in Table I, all the registers affecting I/O transfer are grouped at the lower address locations, this allows quicker handling and more maneuverability in tight data transfers. Also note in Table I that the NSC810A uses 23 I/O addresses out of a block of 26. The upper three bits of the address are determined by the chip enable address.

- **Mode Definition Register (MDR)**

As noted above this register defines the operating mode for ports A and C (port B is always in the basic I/O mode). The upper 3 bits of port C will also be in the basic I/O mode even when the lower 3 bits are being used for handshaking.

The four modes are as follows:

Mode 0—Basic I/O (Input or Output)

Mode 1—Strobed Mode Input

Mode 2—Strobed Mode Output (Active Peripheral Bus)

Mode 3—Strobed Mode Output (TRI-STATE Peripheral Bus)

The address assignment of the MDR is xxx00111 as shown in Table I. Table II specifies the data that must be loaded into the MDR to select the mode.

- **Data Direction Registers (DDR)**

Each port has a DDR that determines whether an individual port bit will be an input or an output. This can be considered the traffic light for the transfer of data between the CPU and the peripheral. Each port bit has a corresponding bit in this register. If the DDR bit is set (1) the port bit is an output; if it is cleared (0) the port bit is an input. The DDR bits cannot be written to individually. The register as a whole must be set to be consistent with all desired port bit directions.

TABLE I. I/O and Timer Address Designations

8-Bit Address Field Bits								Designation I/O Port, Timer, etc.	R (Read) W (Write)
7	6	5	4	3	2	1	0		
x	x	x	0	0	0	0	0	Port A (Data)	R/W
x	x	x	0	0	0	0	1	Port B (Data)	R/W
x	x	x	0	0	0	1	0	Port C (Data)	R/W
x	x	x	0	0	0	1	1	Not Used	**
x	x	x	0	0	1	0	0	DDR - Port A	W
x	x	x	0	0	1	0	1	DDR - Port B	W
x	x	x	0	0	1	1	0	DDR - Port C	W
x	x	x	0	0	1	1	1	Mode Definition Reg.	W
x	x	x	0	1	0	0	0	Port A - Bit-Clear	W
x	x	x	0	1	0	0	1	Port B - Bit-Clear	W
x	x	x	0	1	0	1	0	Port C - Bit-Clear	W
x	x	x	0	1	0	1	1	Not Used	**
x	x	x	0	1	1	0	0	Port A - Bit-Set	W
x	x	x	0	1	1	0	1	Port B - Bit-Set	W
x	x	x	0	1	1	1	0	Port C - Bit-Set	W
x	x	x	0	1	1	1	1	Not Used	**
x	x	x	1	0	0	0	0	Timer 0 (LB)	*
x	x	x	1	0	0	0	1	Timer 0 (HB)	*
x	x	x	1	0	0	1	0	Timer 1 (LB)	*
x	x	x	1	0	0	1	1	Timer 1 (HB)	*
x	x	x	1	0	1	0	0	STOP Timer 0	W
x	x	x	1	0	1	0	1	START Timer 0	W
x	x	x	1	0	1	1	0	STOP Timer 1	W
x	x	x	1	0	1	1	1	START Timer 1	W
x	x	x	1	1	0	0	0	Timer 0 Mode	R/W
x	x	x	1	1	0	0	1	Timer 1 Mode	R/W
x	x	x	1	1	0	1	0	Not Used	**
x	x	x	1	1	0	1	1	Not Used	**
x	x	x	1	1	1	0	0	Not Used	**
x	x	x	1	1	1	0	1	Not Used	**
x	x	x	1	1	1	1	0	Not Used	**
x	x	x	1	1	1	1	1	Not Used	**

x = don't care

LB = low-order byte

HB = high-order byte

* A write accesses the modulus register, a read the read buffer.

** A read from an unused location reads invalid data, a write does not affect any operation of NSC810A.

TABLE II. Mode Definition Register Bit Assignments

Mode	Bit 7	6	5	4	3	2	1	0
0	x	x	x	x	x	x	x	0
1	x	x	x	x	x	x	0	1
2	x	x	x	x	x	0	1	1
3	x	x	x	x	x	1	1	1

9.0 Functional Description (Continued)

Any write or read to the port bits contradicting the direction established by the DDR will not affect the port bits output or input. However, a write to a port bit, defined as an input, will modify the output latch and a read to a port bit, defined as an output, will read this output latch. See *Figure 2*.

• Data Registers

These registers contain the actual data being transferred between the CPU and the peripheral. In Basic I/O, data presented by the peripheral (read cycle) will be latched on the falling edge of $\overline{RD}$. Data presented by the CPU (write cycle) will be valid after the rising edge of $\overline{WR}$ (see AC characteristics for exact timing).

During Strobed I/O, data presented by the peripheral must be valid on the rising edge of $\overline{STB}$. Data received by the peripheral will be valid on the rising edge of $\overline{STB}$. Data latched by the port on the rising edge of $\overline{STB}$ will be preserved until the next CPU read or $\overline{STB}$ signal.

• Bit Set-Clear Registers

The I/O features of the RAM-I/O-timer allow modification of a single bit or several bits of a port with the Bit-Set and Bit-Clear commands. The address selected indicates whether a Bit-Set or Clear will take place. The incoming data on the address/data bus is latched at the trailing edge of the $\overline{WR}$ strobe and is treated as a mask. All bits containing 1s will cause the indicated operation to be performed on the corresponding port bit. All bits of the mask with 0s cause the corresponding port bits to remain unchanged. Three sample operations are shown in Table III using port B as an example.

TABLE III. Bit-Set and Clear Examples

Operation Port B	Set B7	Clear B2 and B0	Set B4, B3 and B1
Address	xxx01101	xxx01001	xxx01101
Data	10000000	00000101	00011010
Port Pins			
Prior State	00001111	10001111	10001010
Next State	10001111	10001010	10011010

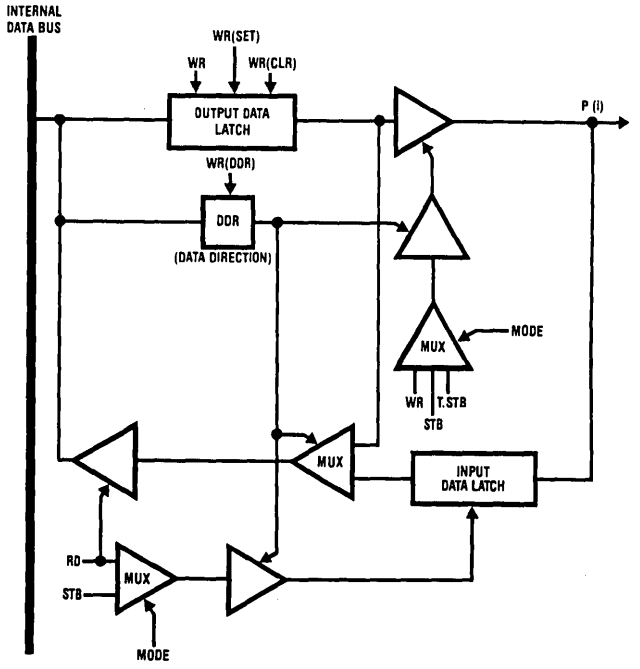
9.3.2 Modes

Two data transfer modes are implemented: Basic I/O and Strobed I/O. Strobed I/O can be further subdivided into three categories: Strobed Input, Strobed Output (active peripheral bus) and Strobed Output (TRI-STATE peripheral bus). The following descriptions detail the functions of these categories.

• Basic I/O

Basic I/O mode uses the $\overline{RD}$ and $\overline{WR}$ CPU bus signals to latch data at the peripheral bus. This mode is the permanent mode of operation for ports B and C. Port A is in this mode if the MDR is set to mode 0. Read and write byte operations and bit operations can be done in Basic I/O. Timing for these modes is shown in the AC Characteristics Table and described with the data register definitions.

When the NSC810A is reset, all registers are cleared to zero. This results in the basic mode of operation being selected, all port bits are made inputs and the output latch for each port bit is cleared to zero. The NSC810A, at this point, can read data from any peripheral port without further setup. If outputs are desired, the CPU merely has to program the appropriate DDR and then send data to the data ports.



TL/C/5517–13

FIGURE 2

9.0 Functional Description (Continued)

• Strobed I/O

Strobed I/O Mode uses the $\overline{STB}$, BF and $\overline{INTR}$ signals to latch the data and indicate that new data is available for transfer. Port A is used for the transfer of data when in any of the Strobed modes. Port B can still be used for Basic I/O and the lower 3-bits of port C are now the three handshake signals for Strobed I/O. Timing for this mode is shown in the AC Characteristic Tables.

Initializing the NSC810A for Strobed I/O Mode is done by loading the data shown in Table IV into the specified register. The registers should be loaded in the order (left to right) that they appear in Table IV.

TABLE IV. Mode Definition Register Configurations

Mode	MDR	DDR Port A	DDR Port C	Port C Output Latch
Basic I/O	xxxxxxx0	Port bit directions are determined by the bits of each port's DDR		
Strobed Input	xxxxxx01	00000000	xxx011	xxx1xx
Strobed Output (Active)	xxxxx011	11111111	xxx011	xxx1xx
Strobed Output (TRI-STATE)	xxxxx111	11111111	xxx011	xxx1xx

• Strobed Input (Mode 1)

During strobed input operations, an external device can load data into port A with the $\overline{STB}$ signal. Data is input to the PA0–7 input latches on the leading (negative) edge of $\overline{STB}$, causing BF to go high (true). On the trailing (positive) edge of $\overline{STB}$ the data is latched and the interrupt signal, $\overline{INTR}$, becomes valid indicating to the CPU that new data is available. $\overline{INTR}$ becomes valid only if the interrupt is enabled, that is the output data latch for PC2 is set to 1.

When the CPU reads port A, address x'00, the trailing edge of the $\overline{RD}$ strobe causes BF and $\overline{INTR}$ to become inactive, indicating that the strobed input cycle has been completed.

• Strobed Output—Active (Mode 2)

During strobed output operations, an external device can read data from port A using the $\overline{STB}$ signal. Data is initially loaded into port A by the CPU writing to I/O address x'00. On the trailing edge of $\overline{WR}$, $\overline{INTR}$ is set inactive and BF becomes valid indicating new data is available for the external device. When the external device is ready to accept the data in port A it pulses the $\overline{STB}$ signal. The rising edge of $\overline{STB}$ resets BF and activates the $\overline{INTR}$ signal. $\overline{INTR}$ becomes valid only if the interrupt is enabled, that is the output latch for PC2 is set to 1. $\overline{INTR}$ in this mode indicates a condition that requires CPU intervention (the output of the next byte of data).

• Strobed Output—TRI-STATE (Mode 3)

The Strobed Output TRI-STATE Mode and the Strobed Output active (peripheral) bus mode function in a similar manner with one exception. The exception is that the data signals on PA0–7 assume the high impedance state at all times except when accessed by the $\overline{STB}$ signal. Strobed Mode 3 is identical to Strobed Mode 2, except as indicated above.

Example Mode 1 (Strobed Input):

Action Taken	$\overline{INTR}$	BF	Results of Action
INITIALIZATION			
Reset NSC810A	H	L	Basic input mode all ports.
Load 01'H into MDR	H	L	Strobed input mode entered; no byte loads to port C after this step; bit-set and clear commands to $\overline{INTR}$ and BF no longer work.
Load 00'H into DDR A	H	L	Sets data direction register for port A to input; data from port A peripheral bus is available to the CPU if the $\overline{STB}$ signal is used, other handshake signals aren't initialized, yet.
Load 03'H into DDR C	H	L	Sets data direction register of port C; buffer full signal works after this step and it is unaffected by the bit-set and clear registers.
Load 04'H into Port C Bit-Set Register	H	L	Sets output latch (PC2) to enable $\overline{INTR}$; $\overline{INTR}$ will latch active whenever $\overline{STB}$ goes low; $\overline{INTR}$ can be disabled by a bit-clear to PC2.*
OPERATION			
$\overline{STB}$ pulses low	L	H	Data on peripheral bus is latched into port A; $\overline{INTR}$ is cleared by a CPU read of port A or a bit-clear of $\overline{STB}$.
CPU reads Port A	H	L	CPU gets data from port A; $\overline{INTR}$ is cleared; peripheral is signalled to send next byte via an inactive BF signal. Repeat last two steps until EOT at which time CPU sends bit-clear to the output latch (PC2).

* Port C can be read by the CPU at anytime, allowing polled operation instead of interrupt driven operation.

9.0 Functional Description (Continued)

Example Mode 2 (Strobed Output—active peripheral bus):

Action Taken	$\overline{INTR}$	BF	Results of Action
INITIALIZE			
Reset NSC810A	H	L	basic input mode all ports.
Load 03'H into MDR	H	L	strobed output mode entered; no byte loads to port C after this step; bit-set and clear commands to $\overline{INTR}$ and BF no longer work.
Load FF'H into DDR A	H	L	Sets data direction register for port A to output; data from port A is available to the peripheral if the $\overline{STB}$ signal is used other handshake signals aren't initialized, yet.
Load 03'H into DDR C	H	L	Sets data direction register of port C; buffer full signal works after this step and it is unaffected by the bit-set and clear registers
Load 04'H into Port C Bit-Set Register	L	L	Sets output latch (PC2) to enable $\overline{INTR}$; active $\overline{INTR}$ indicates that CPU should send data; $\overline{INTR}$ becomes inactive whenever the CPU loads port A; $\overline{INTR}$ can be disabled by a bit-clear to $\overline{STB}$.*
OPERATION			
CPU writes to Port A	H	H	Data on CPU bus is latched into port A; $\overline{INTR}$ is set by the CPU write to port A; active
$\overline{STB}$ pulses low	L	L	BF indicates to peripheral that data is valid; Peripheral gets data from port A; $\overline{INTR}$ is reset active; The active $\overline{INTR}$ signals the CPU to send the next byte. Repeat last two steps until EOT at which time CPU sends bit-clear to the output latch (PC2).

*Port C can be read by the CPU at any time, allowing polled operation instead of interrupt driven operation.

In addition to its timing function, $\overline{STB}$ enables port A outputs to active logic levels. This Mode 3 operation allows other data sources, in addition to the NSC810A, to access the peripheral bus.

• Handshaking Signals

In the Strobed mode of operation, the lower 3-bits of port C transmit/receive the handshake signals (PC0 = $\overline{INTR}$, PC1 = BF, PC2 = $\overline{STB}$).

$\overline{INTR}$ (Strobe Mode Interrupt) is an active-low interrupt from the NSC810A to the CPU. In strobed input mode, the CPU reads the valid data at port A to clear the interrupt. In strobed output mode, the CPU clears the interrupt by writing data to port A.

The $\overline{INTR}$ output can be enabled or disabled, thus giving it the ability to control strobed data transfer. It is enabled or disabled, respectively, by setting or clearing bit 2 of the port C output data latch ($\overline{STB}$).

PC2 is always an input during strobed mode of operation, its output data latch is not needed. Therefore, during strobed mode of operation it is internally gated with the interrupt signal to generate the $\overline{INTR}$ output. Reset clears this bit to zero, so it must be set to one to enable the $\overline{INTR}$ pin for strobed operation.

Once the strobed mode of operation is programmed, the only way to change the output data latch of PC2 is by using the Bit-Set and Clear registers. The port C byte write command will not alter the output data latch of PC2 during the strobed mode of operation.

$\overline{STB}$ (Strobe) is an active low input from the peripheral device, signalling a data transfer. The NSC810A latches data on the rising edge of $\overline{STB}$ if the port bit is an input and the peripheral should latch data on the rising edge of $\overline{STB}$ if the port bit is an output.

BF (Buffer Full) is a high active output from the NSC810A. For input port bits, it indicates that new data has been received from the peripheral. For output port bits, it indicates that new data is available for the peripheral.

Note: In either input or output mode the BF may be cleared by rewriting the MDR.

9.4 TIMERS

The NSC810A has two timers. These are independently programmable, 16-bit binary down-counters. Full count is reached at n + 1, where n is the count loaded into the modulus registers. Timer outputs provide six distinct modes of operation and allow the CPU to check the present count at anytime. Each timer has an independent clock input and output. Start and stop words from the CPU can individually start and stop the timers in any of the modes. A common gate signal can start and stop both timers in three of the six modes. Timer 0 has three possible input clock prescalers ÷1, ÷2 and ÷64. Timer 1 has two possible input clock prescalers ÷1 and ÷2.

Primary components of one timer are shown in *Figure 3.* The timer mode register is a read/write register providing

9.0 Functional Description (Continued)

the primary characterization of the timer output. The start/stop logic and prescaler block divides the clock input by the prescale factor, passing the output (INTCLK) to the binary down-counter. This block also gates the clock input signal (TIN) with the timer gate signal (TG). The timer block loads the modulus from the modulus register and uses (INTCLK) to count to zero. It loads the current count into the read buffer block where the CPU can access it at anytime. This timer block also indicates to the output control logic when the modulus is loaded (or reloaded) and when the count reaches 0. The output control logic block drives the output pins according to the timer mode register and the timer block. The output of the timer block (*Figure 3*) (terminal count) is related to the input TIN by:

$$\text{terminal count} = \frac{\text{TIN}}{p[2(m+1)]}$$

where:

TIN = the input frequency

p = the programmed prescale

m = the modulus

This relationship can be seen directly (TOUT) in Mode 5 (square wave) as it is not masked by the subsequent output logic.

9.4.1 Registers

There are five control registers for each timer. These are shown in the second group of Table I. They determine all timer functions and outputs.

• Modulus Registers and Read Buffer

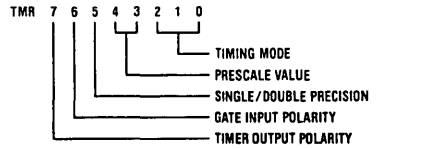
There are two modulus registers per timer (low byte, high byte). These are write only registers, and the two 8-bit values loaded by the CPU are combined into a 16-bit modulus for the timer's down counter.

When the CPU reads from the modulus register addresses, it actually accesses the read buffers. These contain the low and high byte of the decremented modulus. This count is constantly updated by the timer block on the falling edge of INTCLK and can be read without stopping the timers (see single/double precision).

• Timer Mode Register

The timer mode register determines the operating configuration and the active input and output signal levels. Each timer has its own timer mode register, allowing independent operation.

The timer mode register (TMR) may be written or read at any time; however, to assure accurate timing it is important to modify the mode only when the timer is stopped (see Timer Programming). The timer mode is selected from one of six modes by TMR bits 0, 1, and 2 (see Table V). Bits 3 and 4 select the prescale value if the prescaler is to be used. Bits 5, 6 and 7 select the modulus width (8- or 16-bits), gate input polarity, and timer output polarity (active-high or low), respectively. The bit functions of the TMR are illustrated in *Figure 4*.

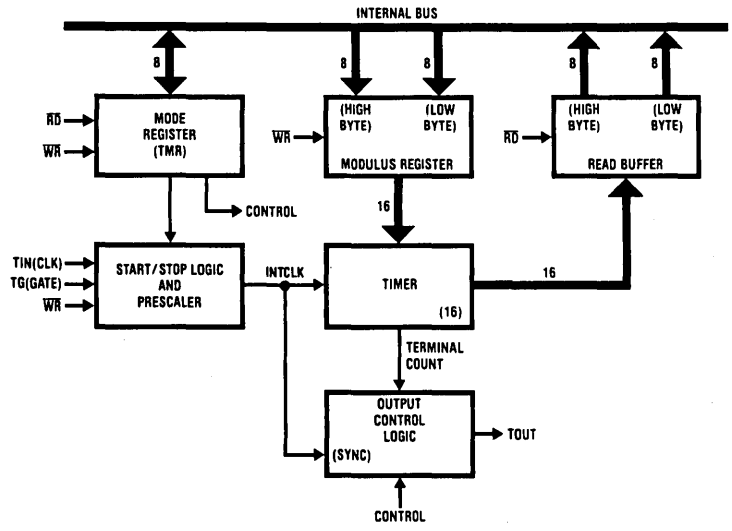


TL/C/5517–15

FIGURE 4. Timer Mode Register

TABLE V. Mode Selection

Bit 2	1	0	–	Timer Function
0	0	0	–	Timer Stopped and Reset
0	0	1	–	Event Counter
0	1	0	–	Event Timer (Stopwatch)
0	1	1	–	Event Timer (Resetting)
1	0	0	–	One Shot
1	0	1	–	Square Wave
1	1	0	–	Pulse Generator
1	1	1	–	Timer Stopped and Reset



TL/C/5517–14

FIGURE 3. Timer Internal Block Diagram (One of Two Timers)

9.0 Functional Description (Continued)

— Timer Prescaler

There is a prescale function associated with each timer. It serves as an additional divisor to lengthen the counts for each timer circuit. The value of the divisor is fixed and selectable in each TMR, as shown below.

	Bits		
TMR0	4	3	Prescale
	0	0	÷1
	0	1	÷2
	1	1	÷64

The ÷64 is not available on timer 1; TMR1 bit 4 is a "don't care."

	Bits		
TMR1	4	3	Prescale
	x	0	÷1
	x	1	÷2

The timer prescale divides the input clock (TIN) and provides the output (INTCLK) to the drive the timer block (*Figure 3*).

— Single/Double Precision

Bit 5 of the TMR determines whether a single or double byte can be accurately read from the read buffer. This option does not affect the use of the modulus registers by the timer block (i.e., the modulus used is always a double byte regardless of the precision mode selected).

The read buffer keeps track of the count and is constantly being updated by the timer block. In order to allow the CPU to read the read buffer, the NSC810A must discontinue updates to this buffer during the read. The precision bit determines whether one or two bytes in the read buffer will be frozen during the read process. In double precision mode, the NSC810A freezes high and low bytes in the read buffer for two consecutive read cycles. In the single precision mode, the NSC810A freezes the read buffer for only one read cycle. Read accesses should be done as follows.

When the TMR bit 5 is:

0— (double byte) read or write the low byte first, then the high byte to maintain proper read/write communications.

1— (single byte) In this mode either the high or low byte of the count can be read at any given instant but not both bytes consecutively. Always write the low byte first, then the high byte to load the modulus.

The following example illustrates this point. If the read buffer had a value of 0200 when the low byte was read and the down-counter decremented to 01FF before the high byte was read, then in the double precision mode the CPU would have read 00 and 02, respectively. In the single precision mode the CPU would have read 00 and 01.

NOTE: In the double precision mode, the high byte should be read immediately after the low byte. Do not access any other registers or unused address locations between the reads.

— Gate Input Polarity

In modes 2, 3 and 4, the TG input is the common hardware control for starting and stopping the timers.

The polarity of the gate input may be selected by the contents of bit 6 of the TMR. If bit 6 equals 0, the gate signal will be active-high or positive edge for mode 4; if bit 6 equals 1, the gate polarity will be active-low or negative edge for mode 4. Modes 2 and 3 are level sensitive. Mode 4 is edge sensitive.

— Timer Output Polarity

Like the gating function, the polarity of the output signal is programmable via bit 7 of the TMR. A zero will cause an active-low output; a one will generate an active-high output.

The output for T1 is multiplexed with port C, bit 5. (Similarly T1IN is multiplexed with port C, bit 4.) When any timer mode other than 0 or 7 is specified for T1, or when mode 2, mode 3, or mode 4 is specified for T0, the three port C pins, bit 3, bit 4, and bit 5, become TG, T1IN and T1OUT, respectively.

• Start and Stop Registers

This is the software start and stop for the timers. There is one start and one stop register for each timer. Writing any data to the start register of a timer starts that timer or transfers start and stop control to TG (in the gated modes 2, 3 and 4). Writing any data to the stop register stops the timer and removes start and stop control from TG (in the gated modes 2, 3 and 4). Restarting the timers causes the modulus to be reloaded for all gated timer modes (2, 3 and 4).

During software restarts of the timers (write to the STOP register and then to the START register) the modulus will be reloaded only if the internal clock signal (INTCLK) is in the high level or makes at least one transition to the high level between the time that the STOP and START registers are written. If INTCLK doesn't meet one of these criteria then the modulus will not be reloaded and the timer will continue to count down from where it was stopped.*

Since it is difficult, if not impossible, to know the level of INTCLK in non-gated modes the recommended practice for restart operation is to reload the modulus after stopping the timer using the 4 step programming procedure in the Timer Programming section of this datasheet. In gated modes INTCLK always stops high.

***NOTE:** INTCLK is coupled via the prescaler to TIN and reacts to the TIN clock input regardless of whether the timer is started or stopped.

— Start/Stop Timing

Figure 5 shows the relationships between the $\overline{WR}$ signal (start register), TIN and INTCLK for both the non-gated and gated modes. The TG signal is only sampled during the positive half of the TIN cycle. This means that when the gated modes are used the internal clock (INTCLK) is never stopped in the low state. Hence, when TG goes active high INTCLK is restarted on the next high-to-low transition of TIN. When TG goes inactive low INTCLK will stop as soon as TIN is high.

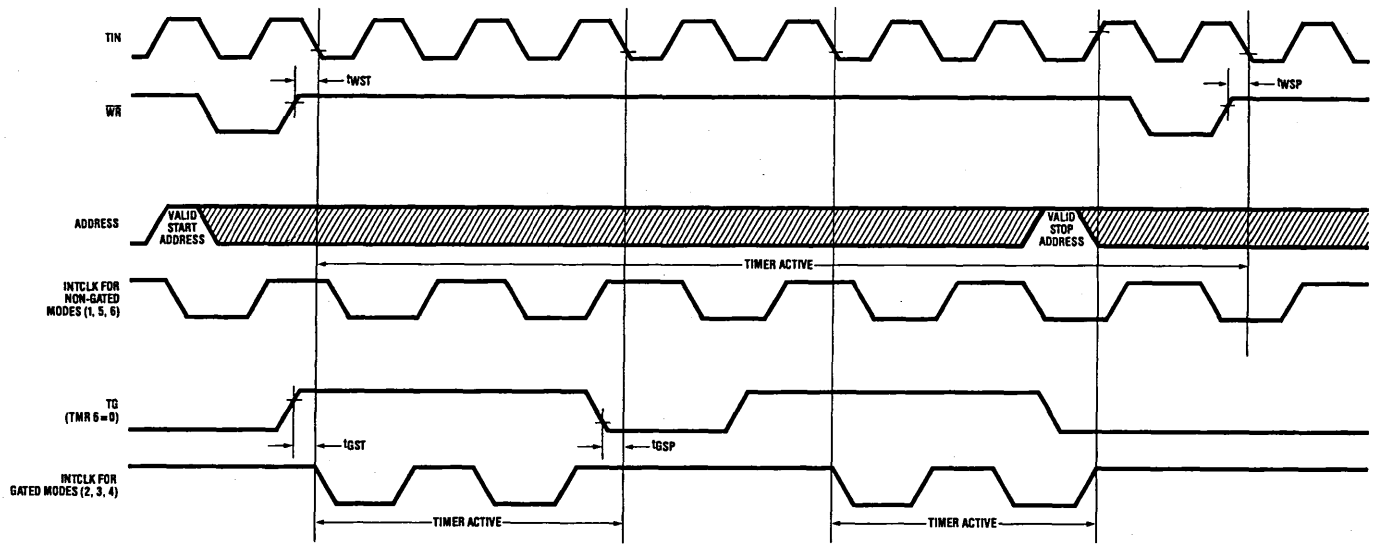
9.4.2 Timer Pins

TIN, TOUT, and TG

Timer 0 has dedicated pins for its clock, T0IN, and its output, T0OUT. Timer 1 must borrow its input and output pins from port C. This is accomplished by writing to the TMR for timer 1. If mode 1, 2, 3, 4, 5 or 6 is specified in TMR1, the pins from port C (PC3, PC4 and PC5) are automatically made available to the timer(s) for gating (TG), T1IN and T1OUT, respectively. These pins are also taken from port C any time timer 0 is in mode 2, 3, 4, so that it has a TG pin. In order to change pins PC3, PC4 and PC5 back to their original configuration as Basic I/O, the timer mode registers must be reset by selecting mode 0 or 7.

TG (PC3), the timer gate, is used for hardware control to start/stop (or trigger) the timers. The timer gate may be used individually by either timer or simultaneously by both timers.

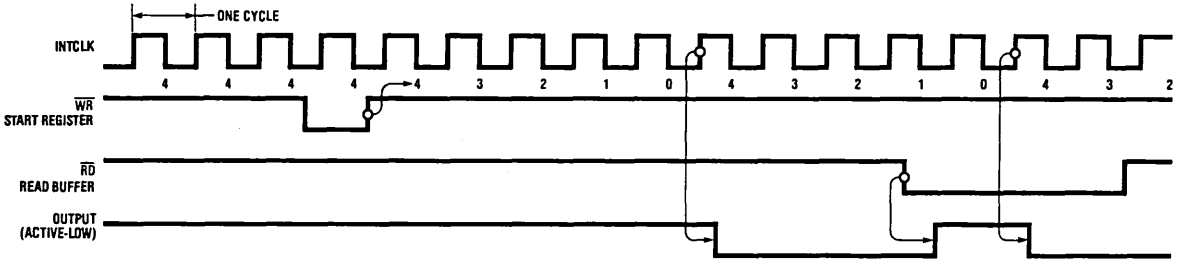
For modes 2 and 3, the timer starts on the gate-active transition assuming the start address was previously written. If



TL/C/5517–16

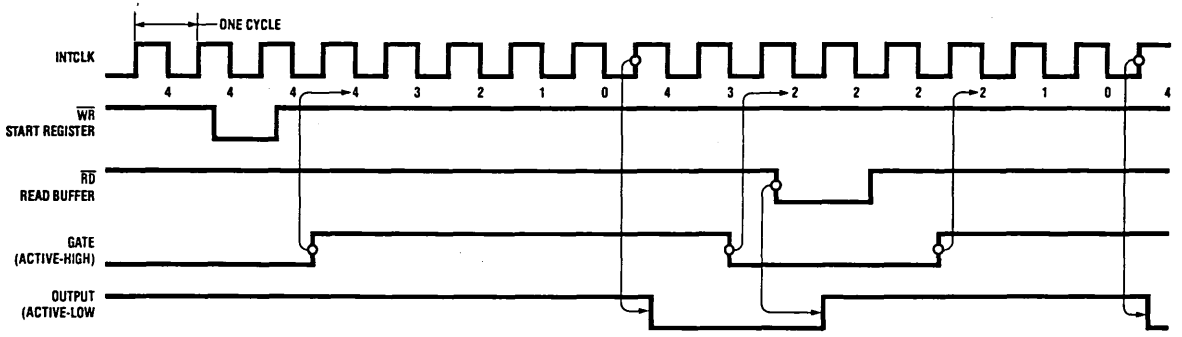
FIGURE 5. Start/Stop Timing

Note: Diagonal lines indicate interval of invalid data.
For mode 4 (one shot), only start-timing applies.
t_{WST}—$\overline{WR}$ set-up for starting timer 150 ns.
t_{WSP}—$\overline{WR}$ set-up for stopping timer 150 ns.
t_{GST}—TG (gate) set-up for starting timer 100 ns.
t_{GSP}—TG (gate) set-up for stopping timer 100 ns.



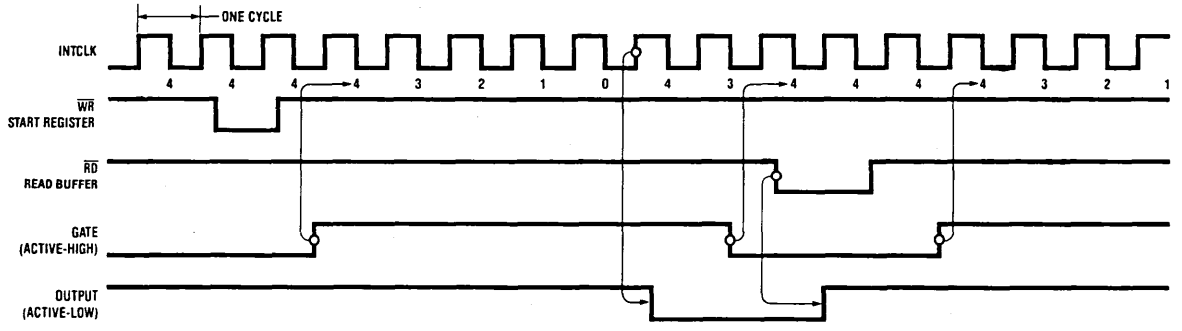
TL/C/5517–17

FIGURE 6a. Event Counter Mode (Mode 1)



TL/C/5517–18

FIGURE 6b. Accumulative Timer (Mode 2)



TL/C/5517–19

FIGURE 6c. Restartable Timing

9.0 Functional Description (Continued)

TABLE VI. Timer Programming Selection Example

Mode Register Bit (TMR) 7	6	5	4	3	2	1	0	Timer Output Polarity Active L/H	Timer Gate Polarity Active L/H	Mode Description Single/Double Precision S/D	Prescale Value	Timing Mode	Port C DDR 543210					
TIMER 0																		
x	x	x	x	x	0	0	0	x	x	x	x	0	x	x	x	x	x	x
0	x	0	0	0	0	0	1	L	x	D	÷1	1	x	x	x	x	x	x
1	x	0	1	1	1	1	0	H	x	D	÷64	6	x	x	x	x	x	x
1	0	0	0	1	1	0	0	H	H	D	÷2	4	1	0	0	x	x	x
0	1	1	0	0	0	1	0	L	L	S	÷1	2	1	0	0	x	x	x
TIMER 1																		
x	x	x	x	x	1	1	1	x	x	x	x	7	x	x	x	x	x	x
0	x	0	x	0	0	0	1	L	x	D	÷1	1	1	0	0	x	x	x
1	0	1	x	1	1	0	1	H	H	S	÷2	5	1	0	0	x	x	x
0	1	0	x	0	0	1	1	L	L	D	÷1	3	1	0	0	x	x	x

the timer gate makes an active transition prior to a write to the start register's address, the trailing edge of the $\overline{WR}$ strobe starts the timer. However, for mode 4 the timer always waits for an active gate edge following a write to the start address before it begins counting.

The DDR for port C must be programmed with the correct I/O direction for TG, T1IN and T1OUT of timer 1. See Table VI for programming examples.

9.4.3 Timer Modes

The low-order three bits (bits 0, 1, 2) of the timer mode registers (TMR) define the mode of operation for the timers. Each TMR may be written to, or read from, at any time. However, to ensure accurate timing, it is important to modify the mode of the timer only when the timer is stopped. Inputs of 000 or 111 define a NOP (no operation) mode. In either of these modes (0 or 7) the timer is stopped, INTCLK is high, and the output is inactive. Inputs of 001 through 110 will select one of six distinct timer functions.

In the explanations that follow, assume that the modulus register for the timer was loaded with the appropriate value (0004) by writing to the low and high bytes of each timer modulus register. Assume also, that the prescale is ÷1.

- **Event Counter (mode 1 TMR bits = 001)**

In this non-gated mode the count is decremented for each clock period (INTCLK) input to the timer block (see *Figure 6a*). When the count reaches zero, the output goes valid and remains valid, until the read buffer is read by the CPU or the timer stop register is written.

At the terminal count (0) the modulus is reloaded into the timer block and the count continues even when the output is valid. This mode can be used to cause periodic interrupts to the CPU.

- **Accumulative Timer (mode 2, TMR bits = 010)**

In this gated mode, the counter will decrement only when the gate input is active (see *Figure 6b*). If the gate becomes inactive, the counter will hold at its present value and continue to decrement when the gate again becomes active. When the count decrements to zero, the output becomes valid and remains valid until the count is read by the CPU or the timer is stopped.

At the terminal count the timer is reloaded and the count continues as long as the gate is active.

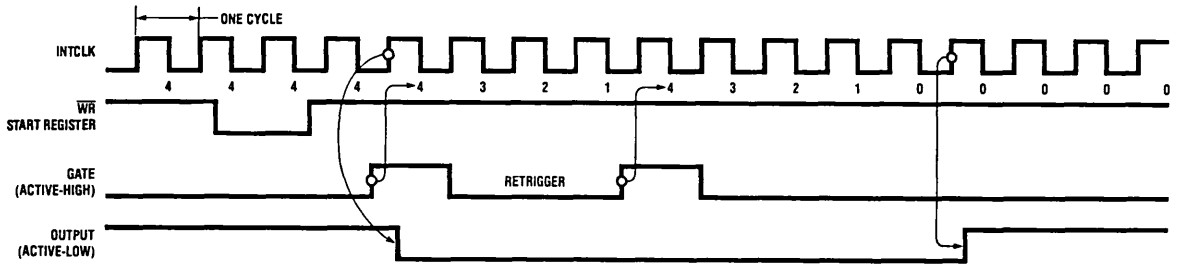
This mode can be used to time processor independent events and to interrupt the CPU when they occur. The prescale and modulus need to be longer than the expected event duration and the gate should go inactive at the event, to preserve the read buffer count for the CPU.

- **Restartable Timer (mode 3, TMR bits = 011)**

In this gated mode, the counter will decrement only when the gate input is active. If the gate becomes inactive, the counter will reload the modulus and hold this value until the gate again becomes active (see *Figure 6c*). If the timer is read when the gate is inactive, you will always read the value the timer has counted down to, not the value the timer has been reloaded with.

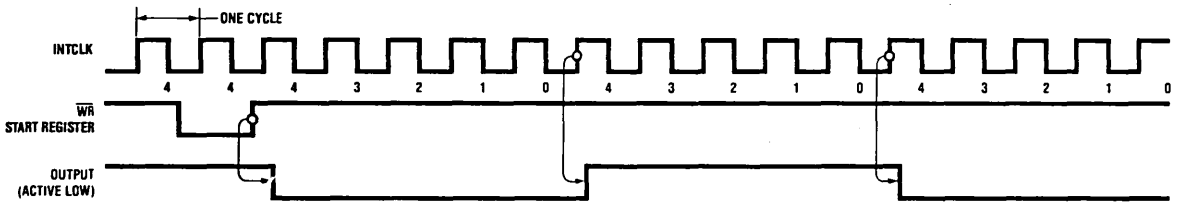
At terminal count the output becomes valid and the timer is reloaded. The timer will continue to run as normal, the only difference is the output is valid. The output remains valid until the count is read by the CPU or the timer stop register is written.

NOTE: The gate inactive time must be longer than the high time of the internal clock (INTCLK) on the chip. Therefore, with ÷64 prescale selected the gate inactive time must be 33 input clocks or greater.



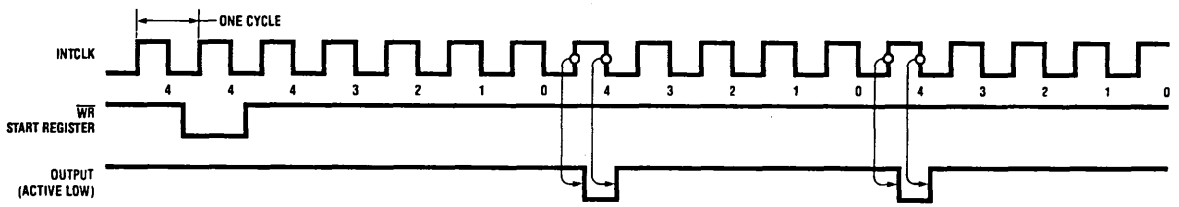
TL/C/5517–20

FIGURE 6d. One Shot (Mode 4)



TL/C/5517–21

FIGURE 6e. Square Wave (Mode 5)



TL/C/5517–22

FIGURE 6f. Pulse Generator (Mode 6)

9.0 Functional Description (Continued)

• One Shot Mode (mode 4, TMR bits = 100)

In this gated mode, the timer holds the modulus count until the active gate edge (see *Figure 6d*). The output immediately becomes valid and remains valid as the counter decrements. The gating signal may go inactive without affecting the count. If TG (the gate) becomes inactive and returns active prior to the terminal count, the modulus will be reloaded, retriggering the one shot period. When the timer reaches the terminal count, the output becomes inactive (see NOTE). The gate, in this mode, is edge sensitive; the active edge is defined by the TMR.

NOTE: The one shot cannot be retriggered during its last internal count (INTCLK) regardless of prescaler selected. Therefore, using the divide by 1 prescaler, it cannot be retriggered during the last clock (TIN), using the divide by 2 prescaler during the last two clocks (TIN) and using the divide by 64 prescaler during the last 64 clocks (TIN).

• Square Wave Mode (mode 5, TMR bits = 101)

In this non-gated mode, the output will go active as soon as the timer is started. The counter decrements for each clock period (INTCLK) and complements its output when zero is reached (see *Figure 6e*). The modulus is then reloaded and counting continues. Assuming a regular clock input, the output will then be a square wave with a period equal to twice the prescale value times the value loaded into the modulus +1 (see equation Timer section intro.). Therefore, varying the modulus will vary the period of the square wave.

• Pulse Generator (mode 6, TMR bits = 110)

In this non-gated mode, the counter decrements for each period of INTCLK (see *Figure 6f*). When the terminal count is reached the output becomes valid for ½ of the TIN clock width for a prescale of ÷1, for one full TIN clock width for a prescale of ÷2 and for 32 TIN clock widths for a prescale of ÷64. The modulus is then reloaded and the sequence is repeated. Varying the prescale and modulus varies the frequency of the pulse.

9.4.4 Timer Programming

The following is the proper sequence to program the timer and should always be used:

1. Write timer mode register selecting mode 0 or 7. This stops the timer, resets the prescaler, and sets internal clock high.
2. Write timer mode register again, this time loading it for your requirements.
3. Write the modulus values, low byte first, high byte second.
4. Start the timers.

The timer read buffer is only updated when the internal timer clock (INTCLK) makes a negative-going transition. Therefore, enough input clock cycles (TIN) must occur to cause a transition of INTCLK given the programmed pre-scaler. After the first transition, the new modulus will be loaded into the read buffer and it can then be read by the CPU.

To guarantee the integrity of the data during a read operation, updates to the timer read buffer are blocked out. If an update is blocked out due to a read, the read buffer will not be updated until the next active transition of INTCLK. Thus, it would appear as if a count was skipped between reads. For example, if the output latches were FF when a block out (read) occurred, the next update could occur at FD, thereby giving an appearance that the count FE was skipped. In actuality the correct number of clocks has occurred for the read buffer to hold FD.

Writing the modulus value when the timer is running does not update the timer immediately. The new value written will get into the timer when the timer reaches its terminal count and reloads its value. If the timer is stopped and a modulus is written the new modulus value will get into the timer when the internal clock is high during the modulus write or on the next low to high internal clock transition. The next time the timer reaches its terminal count it will load the new modulus into the timer. One way to guarantee the new modulus will get into the timer is to follow steps 1 through 4. Although this procedure guarantees that the data will get into the timer you will not be able to read it back until you get a negative-going transition on the internal clock.

Rewriting modulus does not reset the prescaler. The only way to reset the prescaler is to write the mode register and have the internal clock signal be high for some period between the write of the mode register and the start of the timer. Once again, steps 1 through 4 will reset the prescaler.

10.0 NSC810A/883 MIL-STD-883 Class B Screening

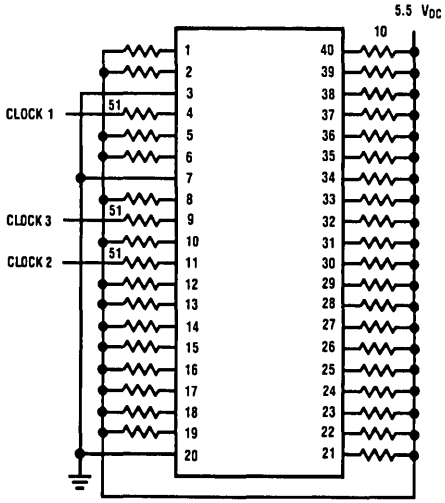
National Semiconductor offers the NSC810AD and NSC810AE with full class B screening per MIL-STD-883 for Military/Aerospace programs requiring high reliability. In addition, this screening is available for all of the key NSC800 peripheral devices.

Electrical testing is performed in accordance with RETS810AX, which tests or guarantees all of the electrical performance characteristics of the NSC810A data sheet. A copy of the current revision of RETS810AX is available upon request. The following table is the MIL-STD-883 flow as of the date of publication.

Test	MIL-STD-883 Method/Condition	Requirement
Internal Visual	2010 B	100%
Stabilization Bake	1008 C 24 Hrs. @ +150°C	100%
Temperature Cycling	1010 C 10 Cycles −65°C/ +150°C	100%
Constant Acceleration	2001 E 30,000 G's, Y1 Axis	100%
Fine Leak	1014 A or B	100%
Gross Leak	1014 C	100%
Burn-In	1015 160 Hrs. @ +125°C (using burn-in circuits shown below)	100%
Final Electrical	+25°C DC per RETS810AX	100%
PDA	5% Max	
	+125°C AC and DC per RETS810AX	100%
	−55°C AC and DC per RETS810AX	100%
	+25°C AC per RETS810AX	100%
QA Acceptance	5005	Sample per
Quality Conformance	5056	Method 5005
External Visual	2009	100%

11.0 Burn-In Circuit

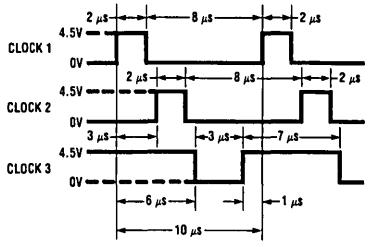
5242HR
NSC810AD/883B (Dual-In-Line)



TL/C/5517–23

12.0 Timing Diagram

Input Clocks



TL/C/5517–24

Note 1: All resistors ±5%, ¼ watt unless otherwise designated, 125°C operating life circuit.

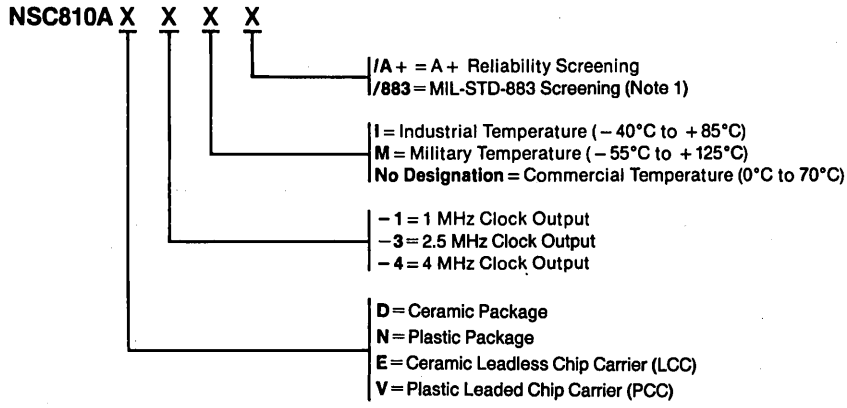
Note 2: E package burn-in circuit 5244HR is functionally identical to the D package.

Note 3: All resistors 2.7 kΩ unless marked otherwise.

Note 4: All clocks 0V to 4.5V.

Note 5: Device to be cooled down under power after burn-in.

13.0 Ordering Information



TL/C/5517–25

Note 1: Do not specify a temperature option; all parts are screened to military temperature.

14.0 Reliability Information

Gate Count	4000
Transistor Count	14,000

National Semiconductor

microCMOS

NSC831 Parallel I/O

General Description

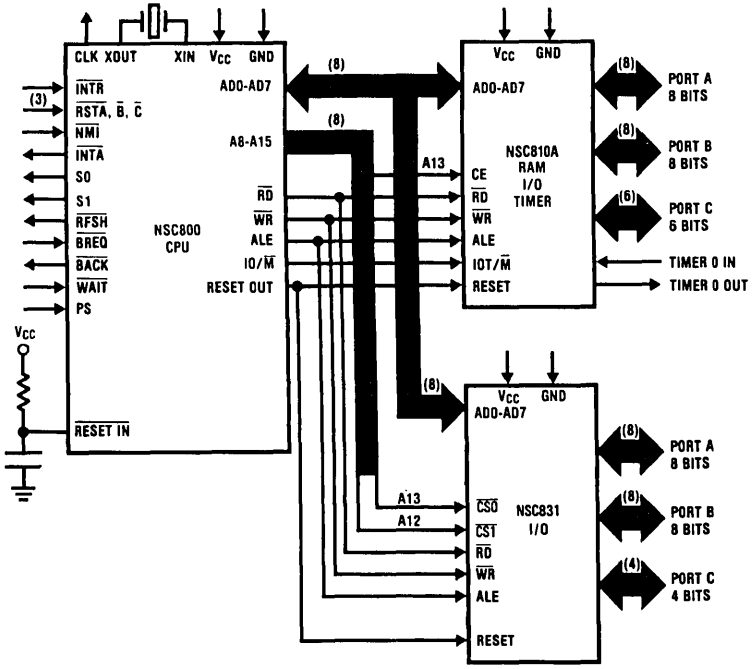
The NSC831 is an I/O device which is fabricated using microCMOS silicon gate technology, functioning as an input/output peripheral interface device. It consists of 20 programmable input/output bits arranged as three separate ports, with each bit individually definable as an input or output. The port bits can be set or cleared individually and can be written to or read from in bytes. Several types of strobed mode operations are available through Port A.

For military applications the NSC831 is available with class B screening in accordance with methods 5004 of MIL-STD-883.

Features

- Three programmable I/O ports
- Single 5V Power Supply
- Very low power consumption
- Fully static operation
- Single-instruction I/O bit operations
- Directly compatible with NSC800 family
- Strobed modes available on Port A

Microcomputer Family Block Diagram



TL/C/5594–1

Table of Contents

1.0 Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Storage Temperature Range	−65°C to +150°C
Voltage at Any Pin With Respect to Ground	−0.3V to V_{CC} + 0.3V
V_{CC}	7V
Lead Temp. (Soldering, 10 seconds)	300°C
Power Dissipation	1W

Note: *Absolute maximum ratings are those values beyond which the safety of the device cannot be guaranteed. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under DC Electrical Characteristics.*

2.0 Operating Range V_{CC} = 5V ±10%

NSC831-1: 0°C to +70°C
−40°C to +85°C

NSC831-3: −40°C to +85°C
−55°C to +125°C

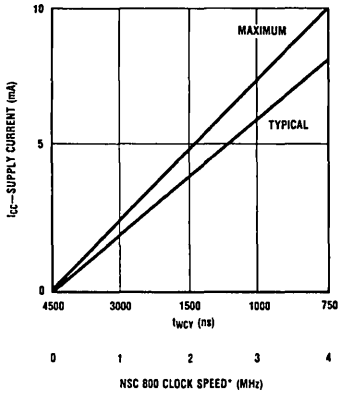
NSC831-4: 0°C to +70°C
−40°C to +85°C
−55°C to +125°C

3.0 DC Electrical Characteristics V_{CC} = 5V ±10%, GND = 0V, unless otherwise specified

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
V_{IH}	Logical 1 Input Voltage		0.8 V_{CC}		V_{CC}	V
V_{IL}	Logical 0 Input Voltage		0		0.2 V_{CC}	V
V_{OH}	Logical 1 Output Voltage	I_{OH} = −1.0 mA	2.4			V
		I_{OUT} = −10 μA	4.0V			V
V_{OL}	Logical 0 Output Voltage	I_{OL} = 2 mA	0		0.4	V
		I_{OUT} = 10 μA	0		0.1	V
I_{IL}	Input Leakage Current	0 ≤ V_{IN} ≤ V_{CC}	−10.0		10.0	μA
I_{OL}	Output Leakage Current	0 ≤ V_{IN} ≤ V_{CC}	−10.0		10.0	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, t_{WCY} = 750 ns		15	20	mA
I_Q	Quiescent Current	RESET = 0, $\overline{RD}$ = 1, $\overline{WR}$ = 1, CE = 1, AD0–7 = 0, ALE = 1, V_{IN} = 0, or V_{IN} = V_{CC} V_{CC} = 5.5V, GND = 0V, PA0–7 = 1, PB0–7 = 1, PC0–7 = 1 No Input Switching, T_A = 25°C		10	100	μA
C_{IN}	Input Capacitance			4	7	pF
C_{OUT}	Output Capacitance			6	10	pF
V_{CC}	Power Supply Voltage	(Note 1)	2.4	5	6	V

Note 1: Operation at lower power supply voltages will reduce the maximum operating speed. Operation at voltages other than 5V ±10% is guaranteed by design, not tested.

I_{CC} vs. SPEED



TL/C/5594–2

*When NSC831 is used with NSC800

4.0 AC Electrical Characteristics V_{CC} = 5V ±10%, GND = 0V

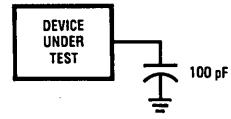
Symbol	Parameter	Test Conditions	NSC831-1		NSC831-3		NSC831-4		Units
			Min	Max	Min	Max	Min	Max	
t_{ACC}	Access Time from ALE	C_L = 150 pF		1000		400		250	ns
t_{AH}	AD0–AD7, CE, IO/$\overline{M}$ Hold Time		100		60		30		ns
t_{ALE}	ALE Strobe Width (High)		200		130		90		ns
t_{ARW}	ALE to $\overline{RD}$ or $\overline{WR}$ Strobe		150		120		120		ns
t_{AS}	AD0–AD7, CE, IO/$\overline{M}$ Setup Time		100		45		40		ns
t_{DH}	Data Hold Time		150		90		40		ns
t_{DO}	Port Data Output Valid			350		320		300	ns
t_{DS}	Data Setup Time		100		80		50		ns
t_{PE}	Peripheral Bus Enable			320		200		200	ns
t_{PH}	Peripheral Data Hold Time		150		125		100		ns
t_{PS}	Peripheral Data Setup Time		100		75		50		ns
t_{PZ}	Peripheral Bus Disable (TRI-STATE®)			150		150		150	ns
t_{RB}	$\overline{RD}$ to BF Output			300		300		300	ns
t_{RD}	Read Strobe Width		400		320		220		ns
t_{RDD}	Data Bus Disable		0	100	0	85	0	85	ns
t_{RI}	$\overline{RD}$ to $\overline{INTR}$ Output			320		300		300	ns
t_{RWA}	$\overline{RD}$ or $\overline{WR}$ to Next ALE		125		100		80		ns
t_{SB}	$\overline{STB}$ to BF Valid			300		300		300	ns
t_{SH}	Peripheral Data Hold With Respect to $\overline{STB}$		150		125		100		ns
t_{SI}	$\overline{STB}$ to INTR Output			300		300		300	ns
t_{SS}	Peripheral Data Setup With Respect to $\overline{STB}$		100		75		50		ns
t_{SW}	$\overline{STB}$ Width		400		320		220		ns
t_{WB}	$\overline{WR}$ to BF Output			340		300		300	ns
t_{WI}	$\overline{WR}$ to $\overline{INTR}$ Output			320		300		300	ns
t_{WR}	$\overline{WR}$ Strobe Width		400		320		220		ns
t_{WCY}	Width of Machine Cycle		3000		1200		750		ns

Note: Test conditions: t_{WCY} = 3000 ns for NSC831-1, 1200 ns for NSC831-3, 750 ns for NSC831-4

AC TESTING INPUT/OUTPUT WAVEFORM

TL/C/5594–3

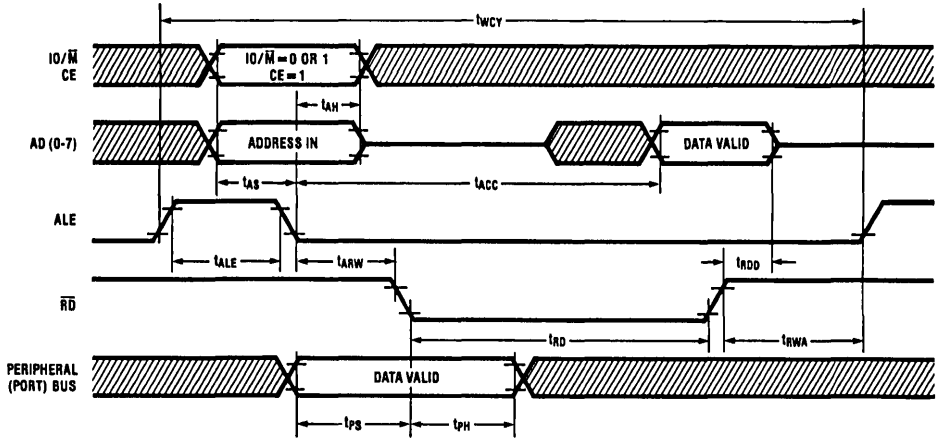
AC TESTING LOAD CIRCUIT



TL/C/5594–4

5.0 Timing Waveforms

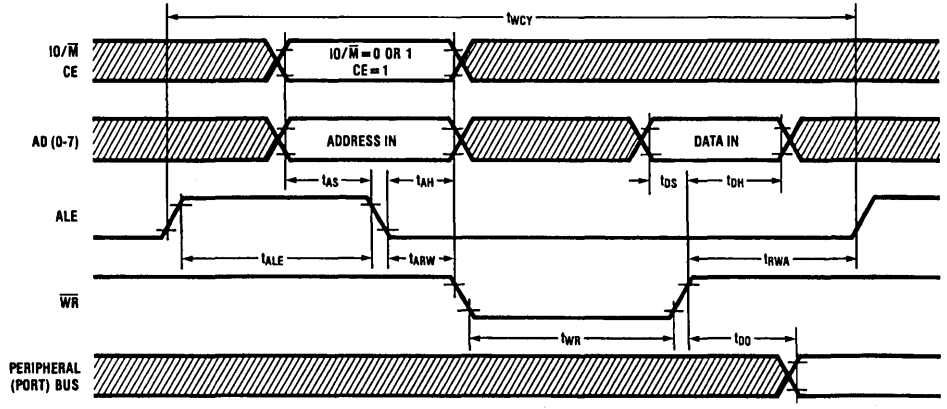
Read Cycle (Read from Port)



TL/C/5594–5

Note: Diagonal lines indicate interval of invalid data.

Write Cycle (Write to Port)

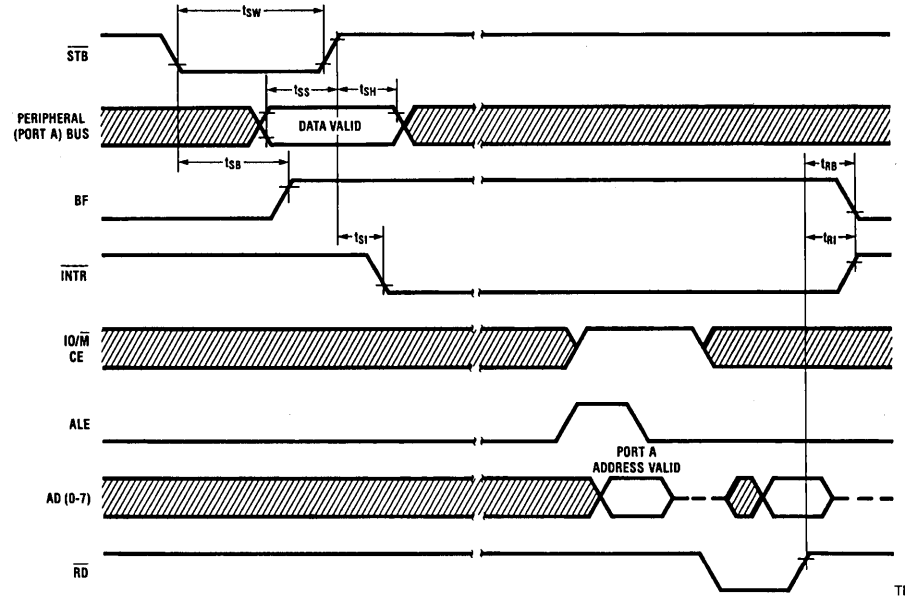


TL/C/5594–6

Note: Diagonal lines indicate interval of invalid data.

5.0 Timing Waveforms (Continued)

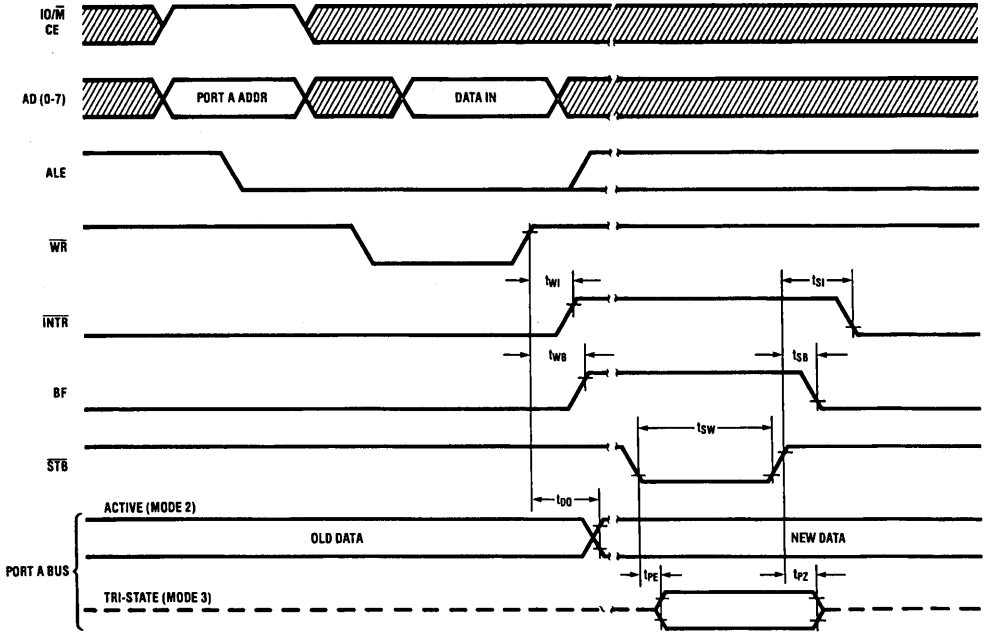
Strobed Mode Input



TL/C/5594–7

Note: Diagonal lines indicate interval of invalid data.

Strobed Mode Output



TL/C/5594–8

Note: Diagonal lines indicate interval of invalid data.

6.0 Pin Descriptions

The following describes the function of all NSC831 input/output pins. Some of these descriptions reference internal circuits.

6.1 INPUT SIGNALS

Master Reset (RESET): An active-high input on the RESET pin initializes the chip causing the three I/O ports (A, B and C) to revert to the input mode. The three ports, the three data direction registers and the mode definition register are reset to low (0).

Chip Enable ($\overline{CE_0}$, $\overline{CE_1}$): The CE inputs must be active at the falling edge of ALE. At ALE time, the CE inputs are latched to provide access to the NSC831.

Read ($\overline{RD}$): when the $\overline{RD}$ input is an active low, data is read from the AD0–AD7 bus.

Write ($\overline{WR}$): When the CE inputs are active an active low $\overline{WR}$ input causes the selected output port to be written with the data from the AD0–AD7 bus.

Address Latch Enable (ALE): The trailing edge (high to low transition) of the ALE input signal latches the address/data present on the AD0–AD7 bus, plus the input control signals on $\overline{CE_0}$ and $\overline{CE_1}$.

Power (V_{CC}): 5V power supply.

Ground (V_{SS}): Ground reference.

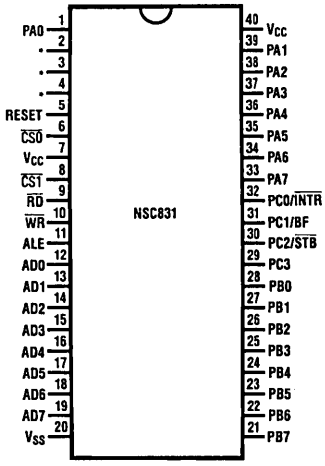
6.2 INPUT/OUTPUT SIGNALS

Bidirectional Address/Data Bus AD0–AD7: The lower 8 bits of the I/O address are applied to these pins, and latched by the trailing edge of ALE. During read operations, 8 bits are present on these pins, and are read when $\overline{RD}$ is low. During an I/O write cycle, Port A, B, or C is written with the data present on this bus at the trailing edge of the $\overline{WR}$ strobe.

Ports A, B, C (PA0–PA7, PB0–PB7, PC0–PC3): These are general purpose I/O pins. Their input/output direction is determined by the contents of the Data Direction Register (DDRs).

7.0 Connection Diagrams

Dual-In-Line Package



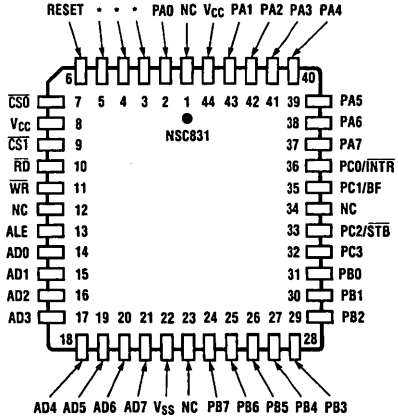
TL/C/5594-9

Top View

*Tie pins 2, 3, and 4 to either V_{CC} or V_{SS}.

Order Number NSC831D or N
See NS Package Number D40C or N40A

Leadless Chip Carrier



NC = NO CONNECT

TL/C/5594-10

Top View

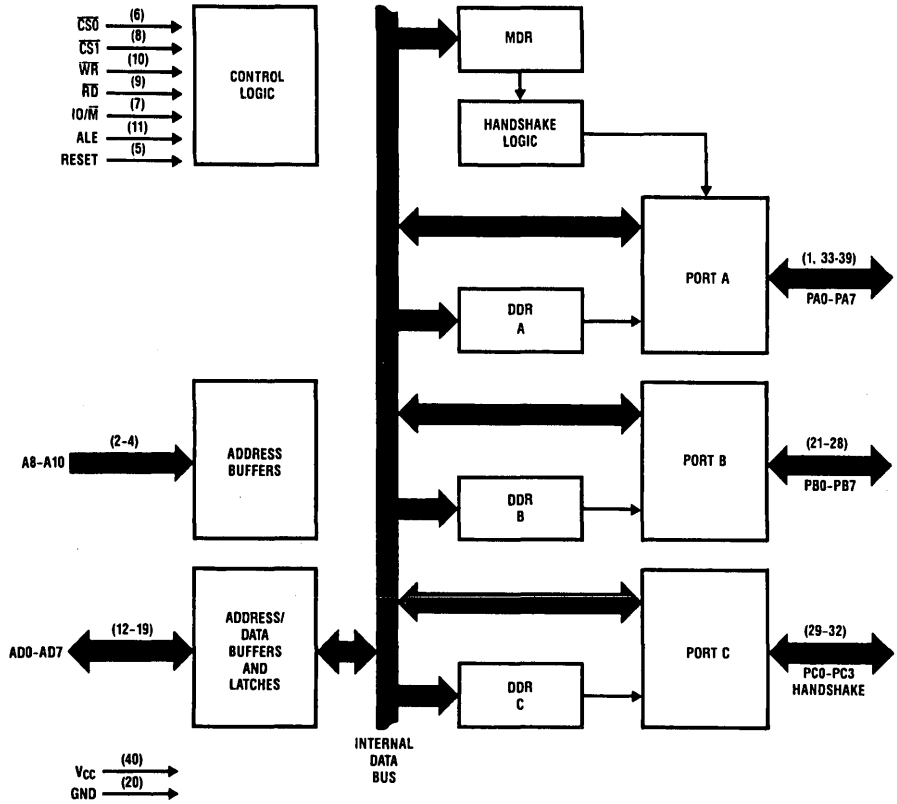
Order Number NSC831E
See NS Package Number E44A

8.0 Functional Description

Refer to *Figure 1* for a detailed block diagram of the NSC831, while reading the following paragraphs.

Input/Output (I/O): The I/O of the NSC831 contains three sets called Ports. There are two ports (A and B) which contain 8 bits each and one port (Port C) which has 4 bits. Any bit or combination of bits in a port may be addressed with Set or Clear commands. A port can also be addressed as an 8-bit word (4 bits for Port C). When reading Port C, bits 4–7 will be read as ones. All ports share common functions of Read, Write, Bit-Set and Bit-Clear. Additionally, Port A is programmable for strobed (handshake mode input or output. Port C has a programmable second function for each bit associated with strobed modes. Table I defines the address location of the ports and control registers.

8.1 BLOCK DIAGRAM



TL/C/5594–11

Note: Applicable pinout for 40 pin dual-in-line package within parentheses.

FIGURE 1

8.0 Functional Description (Continued)

8.2 I/O PORTS

There are three I/O ports (labeled A, B and C) on the NSC831. Ports A and B are 8-bits wide; port C is 4-bits wide. These ports transfer data between the CPU bus and the peripheral bus and vice versa. The way in which these transfers are handled depends upon the currently programmed operating mode.

The NSC831 can be programmed to operate in four different modes. One of these modes (Basic I/O) allows direct transfer of I/O data without any handshaking between the NSC831 and the peripheral. The other three modes (Strobed I/O) provide for timed transfers of I/O data with handshaking between the NSC831 and the peripheral.

Determination of the NSC831 port's mode, data direction and data is done by five registers which are under program control. The Mode Definition Register determines in which of the four I/O modes the chip will operate. Another register (Data Direction Register) establishes the data direction for each bit in that port. The Data Register holds data to be transferred or that which was received. The final two registers per port allow individual data register bits to be cleared (Bit-Clear Register) or data register bits to be set (Bit-Set Register).

Operation during Strobed I/O utilizes two of the port C pins for handshaking and one port C pin to interrupt the CPU.

8.3 REGISTERS

As indicated in the overview, programmable registers control the flow of data through the ports. Table I shows the registers of the NSC831. All registers affecting I/O transfers are in the first grouping of this table.

• Mode Definition Register (MDR)

The MDR determines the operating mode for port A and whether or not the lower 3-bits of port C will be used for handshaking (Strobed I/O). Port B always transfers data via the Basic I/O mode, regardless of how the MDR is programmed.

The four modes are as follows:

Mode 0—Basic I/O (Input or Output)

Mode 1—Strobed Mode Input

Mode 2—Strobed Mode Output (Active Peripheral Bus)

Mode 3—Strobed Mode Output (TRI-STATE Peripheral Bus)

The address assignment of the MDR is xxx00111 as shown in Table I. The upper 3 "don't care" bits are determined by the users decode logic (chip enable address). Table II specifies the data that must be loaded into the MDR to select the mode.

• Data Direction Registers (DDR)

Each port has a DDR that determines whether an individual port bit will be an input or an output. If DDR for the port bit is set to a 1, then that port bit is an output. If its DDR is reset to a 0, then it is an input. The DDR bits cannot be individually written to; the entire DDR register is affected by a write to the DDR. Thus, all data bits written must be consistent for all desired port bit directions.

TABLE I. I/O and Timer Address Designations

8-Bit Address Field Bits								Designation I/O Port, Timer, etc.	R (Read) W (Write)
7	6	5	4	3	2	1	0		
x	x	x	x	0	0	0	0	Port A (Data)	R/W
x	x	x	x	0	0	0	1	Port B (Data)	R/W
x	x	x	x	0	0	1	0	Port C (Data)	R/W
x	x	x	x	0	0	1	1	Not Used	**
x	x	x	x	0	1	0	0	DDR - Port A	W
x	x	x	x	0	1	0	1	DDR - Port B	W
x	x	x	x	0	1	1	0	DDR - Port C	W
x	x	x	x	0	1	1	1	Mode Definition Reg.	W
x	x	x	x	1	0	0	0	Port A - Bit-Clear	W
x	x	x	x	1	0	0	1	Port B - Bit-Clear	W
x	x	x	x	1	0	1	0	Port C - Bit-Clear	W
x	x	x	x	1	0	1	1	Not Used	**
x	x	x	x	1	1	0	0	Port A - Bit-Set	W
x	x	x	x	1	1	0	1	Port B - Bit-Set	W
x	x	x	x	1	1	1	0	Port C - Bit-Set	W
x	x	x	x	1	1	1	1	Not Used	**

x = don't care

LB = low-order byte

HB = high-order byte

* A write accesses the modulus register, a read the read buffer.

** A read from an unused location reads invalid data, a write does not affect any operation of NSC831.

TABLE II. Mode Definition Register Bit Assignments

Mode	Bit							
	7	6	5	4	3	2	1	0
0	x	x	x	x	x	x	x	0
1	x	x	x	x	x	x	0	1
2	x	x	x	x	x	0	1	1
3	x	x	x	x	x	1	1	1

7

8.0 Functional Description (Continued)

Any write or read to the port bits contradicting the direction established by the DDR will not affect the port bits output or input. However, a write to a port bit, defined as an input, will modify the output latch and a read to a port bit, defined as an output, will read this output latch. See *Figure 2*.

• Data Registers

These registers contain the actual data being transferred between the CPU and the peripheral. In Basic I/O, data presented by the peripheral (read cycle) will be latched on the falling edge of $\overline{RD}$. Data presented by the CPU (write cycle) will be valid after the rising edge of $\overline{WR}$ (see AC characteristics for exact timing).

During Strobed I/O, data presented by the peripheral must be valid on the rising edge of $\overline{STB}$. Data received by the peripheral will be valid on the rising edge of $\overline{STB}$. Data latched by the port on the rising edge of $\overline{STB}$ will be preserved until the next CPU read or $\overline{STB}$ signal.

• Bit Set-Clear Registers

The I/O features of the RAM-I/O-timer allow modification of a single bit or several bits of a port with the Bit-Set and Bit-Clear commands. The address selected indicates whether a Bit-Set or Clear will take place. The incoming data on the address/data bus is latched at the trailing edge of the $\overline{WR}$ strobe and is treated as a mask. All bits containing 1s will cause the indicated operation to be performed on the corresponding port bit. All bits of the mask with 0s cause the corresponding port bits to remain unchanged. Three sample operations are shown in Table III using port B as an example.

TABLE III. Bit-Set and Clear Examples

Operation Port B	Set B7	Clear B2 and B0	Set B4, B3 and B1
Address	xxx01101	xxx01001	xxx01101
Data	10000000	00000101	00011010
Port Pins			
Prior State	00001111	10001111	10001010
Next State	10001111	10001010	10011010

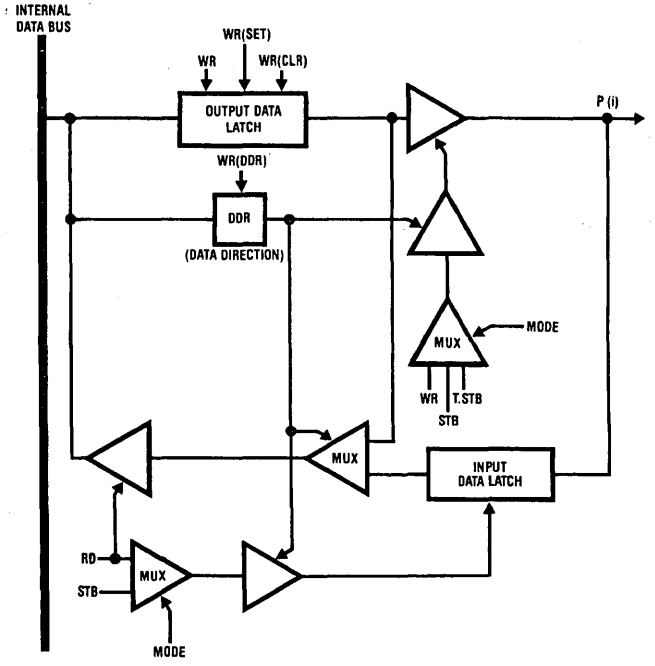
8.4 MODES

Two data transfer modes are implemented: Basic I/O and Strobed I/O. Strobed I/O can be further subdivided into three categories: Strobed Input, Strobed Output (active peripheral bus) and Strobed Output (TRI-STATE peripheral bus). The following descriptions detail the functions of these categories.

• Basic I/O

Basic I/O mode uses the $\overline{RD}$ and $\overline{WR}$ CPU bus signals to latch data at the peripheral bus. This mode is the permanent mode of operation for ports B and C. Port A is in this mode if the MDR is set to mode 0. Read and write byte operations and bit operations can be done in Basic I/O. Timing for these modes is shown in the AC Characteristics Table and described with the data register definitions.

When the NSC831 is reset, all registers are cleared to zero. This results in the basic mode of operation being selected, all port bits are made inputs and the output latch for each port bit is cleared to zero. The NSC831, at this point, can read data from any peripheral port without further set-up. If outputs are desired, the CPU merely has to program the appropriate DDR and then send data to the data ports.



TL/C/5594–12

FIGURE 2

8.0 Functional Description (Continued)

• Strobed I/O

Strobed I/O Mode uses the $\overline{STB}$, BF and $\overline{INTR}$ signals to latch the data and indicate that new data is available for transfer. Port A is used for the transfer of data when in any of the Strobed modes. Port B can still be used for Basic I/O and the lower 3-bits of port C are now the three handshake signals for Strobed I/O. Timing for this mode is shown in the AC Characteristic Tables.

Initializing the NSC831 for Strobed I/O Mode is done by loading the data shown in Table IV into the specified register. The registers should be loaded in the order (left to right) that they appear in Table IV.

TABLE IV. Mode Definition Register Configurations

Mode	MDR	DDR Port A	DDR Port C	Port C Output Latch
Basic I/O	xxxxxxx0	Port bit directions are determined by the bits of each port's DDR		
Strobed Input	xxxxxx01	00000000	xxx011	xxx1xx
Strobed Output (Active)	xxxxx011	11111111	xxx011	xxx1xx
Strobed Output (TRI-STATE)	xxxxx111	11111111	xxx011	xxx1xx

• Strobed Input (Mode 1)

During strobed input operations, an external device can load data into port A with the $\overline{STB}$ signal. Data is input to the PA0–7 input latches on the leading (negative) edge of $\overline{STB}$, causing BF to go high (true). On the trailing (positive) edge of $\overline{STB}$ the data is latched and the interrupt signal, $\overline{INTR}$, becomes valid indicating to the CPU that new data is available. $\overline{INTR}$ becomes valid only if the interrupt is enabled, that is the output data latch for PC2 is set to 1.

When the CPU reads port A, address x'00, the trailing edge of the $\overline{RD}$ strobe causes BF and $\overline{INTR}$ to become inactive, indicating that the strobed input cycle has been completed.

• Strobed Output—Active (Mode 2)

During strobed output operations, an external device can read data from port A using the $\overline{STB}$ signal. Data is initially loaded into port A by the CPU writing to I/O address x'00. On the trailing edge of $\overline{WR}$, $\overline{INTR}$ is set inactive and BF becomes valid indicating new data is available for the external device. When the external device is ready to accept the data in port A it pulses the $\overline{STB}$ signal. The rising edge of $\overline{STB}$ resets BF and activates the $\overline{INTR}$ signal. $\overline{INTR}$ becomes valid only if the interrupt is enabled, that is the output latch for PC2 is set to 1. $\overline{INTR}$ in this mode indicates a condition that requires CPU intervention (the output of the next byte of data).

• Strobed Output—TRI-STATE (Mode 3)

The Strobed Output TRI-STATE Mode and the Strobed Output active (peripheral) bus mode function in a similar manner with one exception. The exception is that the data signals on PA0–7 assume the high impedance state at all times except when accessed by the $\overline{STB}$ signal. Thus, in addition to its timing function, $\overline{STB}$ enables port A outputs to active logic levels. This Mode 3 operation allows other data sources, in addition to the NSC831, to access the peripheral bus. Strobed Mode 3 is identical to Strobed Mode 2, except as indicated above.

Example Mode 1 (Strobed Input):

Action Taken	$\overline{INTR}$	BF	Results of Action
INITIALIZATION			
Reset NSC831	H	L	Basic input mode all ports.
Load 01'H into MDR	H	L	Strobed input mode entered; no byte loads to port C after this step; bit-set and clear commands to $\overline{INTR}$ and BF no longer work.
Load 00'H into DDR A	H	L	Sets data direction register for port A to input; data from port A peripheral bus is available to the CPU if the $\overline{STB}$ signal is used, other handshake signals aren't initialized, yet.
Load 03'H into DDR C	H	L	Sets data direction register of port C; buffer full signal works after this step and it is unaffected by the bit-set and clear registers.
Load 04'H into Port C Bit-Set Register	H	L	Sets output latch (PC2) to enable $\overline{INTR}$; $\overline{INTR}$ will latch active whenever $\overline{STB}$ goes low; $\overline{INTR}$ can be disabled by a bit-clear to PC2.*
OPERATION			
$\overline{STB}$ pulses low	L	H	Data on peripheral bus is latched into port A; $\overline{INTR}$ is cleared by a CPU read of port A or a bit-clear of $\overline{STB}$.
CPU reads Port A	H	L	CPU gets data from port A; $\overline{INTR}$ is cleared; peripheral is signalled to send next byte via an inactive BF signal. Repeat last two steps until EOT at which time CPU sends bit-clear to the output latch (PC2).

*Port C can be read by the CPU at anytime, allowing polled operation instead of interrupt driven operation.

8.0 Functional Description (Continued)

Example Mode 2 (Strobed Output—active peripheral bus):

Action Taken	$\overline{INTR}$	BF	Results of Action
INITIALIZE			
Reset NSC831	H	L	Basic input mode all ports.
Load 03'H into MDR	H	L	Strobed output mode entered; no byte loads to port C after this step; bit-set and clear commands to $\overline{INTR}$ and BF no longer work.
Load FF'H into DDR A	H	L	Sets data direction register for port A to output; data from port A is available to the peripheral if the $\overline{STB}$ signal is used other handshake signals aren't initialized, yet.
Load 03'H into DDR C	H	L	Sets data direction register of port C; buffer full signal works after this step and it is unaffected by the bit-set and clear registers
Load 04'H into Port C Bit-Set Register	L	L	Sets output latch (PC2) to enable $\overline{INTR}$; active $\overline{INTR}$ indicates that CPU should send data; $\overline{INTR}$ becomes inactive whenever the CPU loads port A; $\overline{INTR}$ can be disabled by a bit-clear to $\overline{STB}$.*
OPERATION			
CPU writes to Port A	H	H	Data on CPU bus is latched into port A; $\overline{INTR}$ is set by the CPU write to port A; active BF indicates to peripheral that data is valid; Peripheral gets data from port A; $\overline{INTR}$ is reset active; The active $\overline{INTR}$ signals the CPU to send the next byte. Repeat last two steps until EOT at which time CPU sends bit-clear to the output latch (PC2).
$\overline{STB}$ pulses low	L	L	

*Port C can be read by the CPU at any time, allowing polled operation instead of interrupt driven operation.

• Handshaking Signals

In the Strobed mode of operation, the lower 3-bits of port C transmit/receive the handshake signals (PC0 = $\overline{INTR}$, PC1 = BF, PC2 = $\overline{STB}$).

$\overline{INTR}$ (Strobe Mode Interrupt) is an active-low interrupt from the NSC831 to the CPU. In strobed input mode, the CPU reads the valid data at port A to clear the interrupt. In strobed output mode, the CPU clears the interrupt by writing data to port A.

The $\overline{INTR}$ output can be enabled or disabled, thus giving it the ability to control strobed data transfer. It is enabled or disabled, respectively, by setting or clearing bit 2 of the port C output data latch ($\overline{STB}$).

PC2 is always an input during strobed mode of operation, its output data latch is not needed. Therefore, during strobed mode of operation it is internally gated with the interrupt signal to generate the $\overline{INTR}$ output. Reset clears this bit to zero, so it must be set to one to enable the $\overline{INTR}$ pin for strobed operation.

Once the strobed mode of operation is programmed, the only way to change the output data latch of PC2 is by using the Bit-Set and Clear registers. The port C byte write command will not alter the output data latch of PC2 during the strobed mode of operation.

$\overline{STB}$ (Strobe) is an active low input from the peripheral device, signalling a data transfer. The NSC831 latches data on the rising edge of $\overline{STB}$ if the port bit is an input and the peripheral should latch data on the rising edge of $\overline{STB}$ if the port bit is an output.

BF (Buffer Full) is a high active output from the NSC831. For input port bits, it indicates that new data has been received from the peripheral. For output port bits, it indicates that new data is available for the peripheral.

Note: In either input or output mode the BF may be cleared by rewriting the MDR.

9.0 NSC831/883B MIL-STD-883 Class B Screening

National Semiconductor offers the NSC831D and NSC831E with full class B screening per MIL-STD-883 for Military/ Aerospace programs requiring high reliability. In addition, this screening is available for all of the key NSC800 peripheral devices.

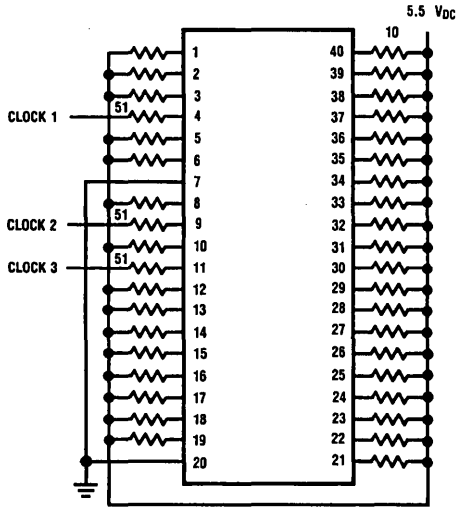
Electrical testing is performed in accordance with RETS831X, which tests or guarantees all of the electrical performance characteristics of the NSC831 data sheet. A copy of the current revision of RETS831X is available upon request. The following table is the MIL-STD-883 flow as of the date of publication.

100% Screening Flow

Test	MIL-STD-883 Method/Condition	Requirement
Internal Visual	2010 B	100%
Stabilization Bake	1008C 24 Hrs. @ +150°C	100%
Temperature Cycling	1010C 10 Cycles −65°C/ +150°C	100%
Constant Acceleration	2001E 30,000 Gs, Y1 Axis	100%
Fine Leak	1014 A or B	100%
Gross Leak	1014C	100%
Burn-In	1015 160 Hrs. @ +125°C (using burn-in circuits shown below)	100%
Final Electrical	+25°C DC per RETS831X	100%
PDA	5% Max	
	+125°C AC and DC per RETS831X	100%
	−55°C AC and DC per RETS831X	100%
	+25°C AC per RETS831X	100%
QA Acceptance	5005	Sample per
Quality Conformance		Method 5005
External Visual	2009	100%

10.0 Burn-In Circuit

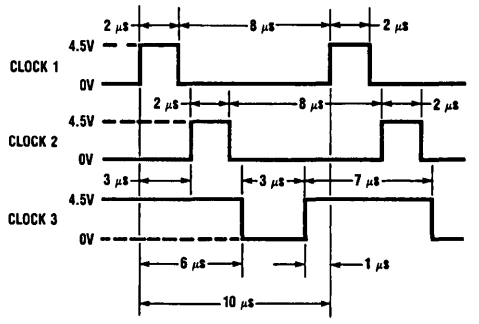
5242HR
NSC831AD/883B (Dual-In-Line)



TL/C/5594–13

11.0 Timing Diagram

Input Clocks



TL/C/5594–14

Note 1: All resistors ±5%, ¼ watt unless otherwise designated, 125°C operating life circuit.

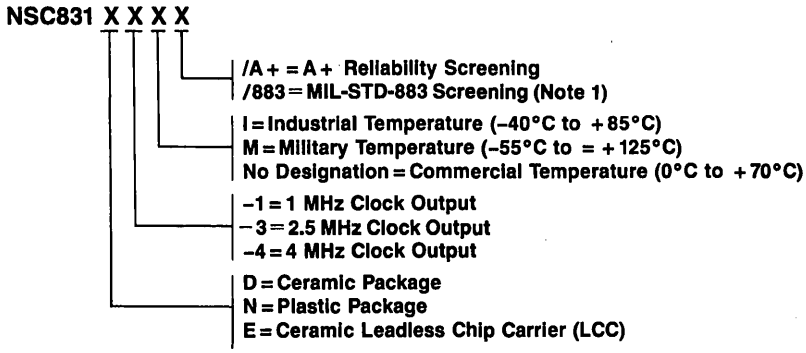
Note 2: E package burn-in circuit 5244HR is functionally identical to the D package.

Note 3: All resistors 2.7 kΩ unless marked otherwise.

Note 4: All clocks 0V to 4.5V.

Note 5: Device to be cooled down under power after burn-in.

12.0 Ordering Information



TL/C/5594–15

Note 1: Do not specify a temperature option: all parts are screened to military temperature.

13.0 Reliability Information (NSC831)

Gate Count 1900
Transistor Count 7400

National Semiconductor

microCMOS

NSC858 Universal Asynchronous Receiver/Transmitter

General Description

The NSC858 is a CMOS programmable Universal Asynchronous Receiver/Transmitter (UART). It has an on chip programmable baud rate generator. The UART, which is fabricated using microCMOS silicon gate technology, functions as a serial receiver/transmitter interface for your microcomputer system.

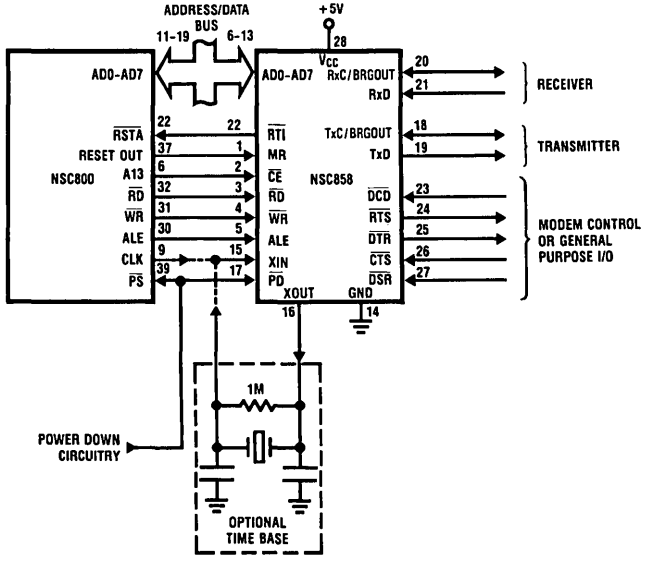
The transmitter converts parallel data from the CPU to serial form and shifts it out in the standard asynchronous communication data format. Appropriate start, parity, and stop bits are added to the outgoing serial stream. Incoming serial data is converted to parallel form by the receiver. The receiver checks incoming data for errors (parity, overrun, framing or break interrupt) and then converts it from serial to parallel for transfer to the CPU. Five pins on the chip are available for modem control functions or general purpose I/O.

The NSC858 has a programmable baud generator that is capable of dividing the timing reference clock input by divisors of 1 to ($2^{16}-1$), and producing a 1X, 16X, 32X, 64X clock for driving the transmitter and/or receiver logic. Both the transmitter and receiver can either be driven by an external clock or the internal baud rate generator. The NSC858 has an interrupt system that can be tailored to the user's requirements. In addition to the CMOS power consumption levels there are hardware and software power down modes which further reduce power consumption levels.

Features

- Maximum baud rate 256k BPS (16X), 1M BPS (1X)
- Programmable baud rate generator
- Double buffered receiver and transmitter
- Independently configured receiver and transmitter
  - 5-, 6-, 7-, 8-bit characters
  - Odd, even, force high, force low, or no parity
  - 1, 1½, 2 stop bits
- Five bits modem I/O or general purpose I/O (3 input, 2 output)
- Programmable auto enables for $\overline{CTS}$ and $\overline{DCD}$
- Local and remote loopback diagnostics
- False start bit detection
- Break condition detection and generation
- Program polled, or interrupt driven operation
  - 8 maskable status conditions for receiver and transmitter interrupt
  - 4 maskable status conditions for modem interrupt
- Variable power supply (2.4V–6.0V)
- Low power consumption with software and hardware power down modes
- 8-bit multiplexed address/data bus directly compatible with NSC800™

System Configuration



TL/C/5593–1

7

Table of Contents

1.0 Absolute Maximum Ratings

(Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.3V to V_{CC} +0.3V
Maximum V_{CC}	7V
Power Dissipation	1W
Lead Temp. (Soldering, 10 seconds)	300°C

2.0 Operating Conditions V_{CC} = 5V ± 10%

Ambient Temperature	
Industrial	−40°C to +85°C
Commercial	0°C to +70°C

3.0 DC Electrical Characteristics V_{CC} = 5V ± 10%, GND = 0V, unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Logical 1 Input Voltage		0.8 V_{CC}		V_{CC}	V
V_{IL}	Logical 0 Input Voltage		0		0.2 V_{CC}	V
V_{HY}	Hysteresis at RESET IN Input	V_{CC} = 5V	0.25	0.5		V
V_{OH1}	Logical 1 Output Voltage	I_{OUT} = −1.0 mA	2.4			V
V_{OH2}	Logical 1 Output Voltage	I_{OUT} = −10 μA	V_{CC} − 0.5			V
V_{OL1}	Logical 0 Output Voltage	I_{OL} = 2 mA except X_{OUT}	0		0.4	V
V_{OL2}	Logical 0 Output Voltage	I_{OUT} = 10 μA	0		0.1	V
I_{IL}	Input Leakage Current	$0 \le V_{IN} \le V_{CC}$	−10.0		10.0	μA
I_{OL}	Output Leakage Current	$0 \le V_{IN} \le V_{CC}$	−10.0		10.0	μA
I_{CC}	Active Supply Current	T_A = 25°C		2	10	mA
I_{HPD}	Current Hardware Power Down	Pin $\overline{PD}$ = 0, No Resistive Output Loads, V_{IN} = 0V or V_{IN} = V_{CC}, T_A = 25°C		100		μA
I_{SPD}	Current Software Power Down	Power Down Reg Bit 0 = 1, No Resistive Output Loads, V_{IN} = 0V or V_{IN} = V_{CC}, T_A = 25°C		300		μA
C_{IN}	Input Capacitance			6	10	pF
C_{OUT}	Output Capacitance			8	12	pF
V_{CC}	Power Supply Voltage	(Note 2)	2.4	5	6	V

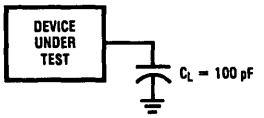
Note 1: Absolute Maximum Ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended and should be limited to those conditions specified under DC Electrical Characteristics.

Note 2: Operation at lower power supply voltages will reduce the maximum operating speed. Operation at voltages other than 5V ±10% is guaranteed by design, not tested.

AC Testing Input/Output Waveform

TL/C/5593-2

AC Testing Load Circuit



TL/C/5593-3

4.0 AC Electrical Characteristics V_{CC} = 5V ± 10%, GND = 0V, C_L = 100 pF

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
BUS						
t_{AS}	Address 0–7 Set-Up Time		40			ns
t_{AH}	Address 0–7 Hold Time		30			ns
t_{ALE}	ALE Strobe Width (High)		100			ns
t_{ARW}	ALE to Read or Write Strobe		75			ns
t_{CRW}	Chip Enable to Read or Write		110			ns
t_{RD}	Read Strobe Width		250			ns
t_{DDR}	Data Delay from Read			180	200	ns
t_{RDD}	Data Bus Disable				75	ns
t_{CH}	Chip Enable Hold After Read or Write		60			ns
t_{RWA}	Read or Write to Next ALE		45			ns
t_{WR}	Write Strobe Width		200	250		ns
t_{DS}	Data Set-Up Time		100			ns
t_{DH}	Data Hold Time		75			ns
MODEM						
t_{MD}	$\overline{WR}$ Command Reg. to Modem Outputs Delay			180		ns
t_{SIM}	Delay to Set Interrupt from Modem Input			200		ns
t_{RIM}	Delay to Reset Modem Status Interrupt from $\overline{RD}$			240		ns
t_{SMI}	$\overline{WR}$ to Status Mask Reg., Delay to $\overline{RTI}$				230	ns
POWER DOWN						
t_{PCS}	Power Down to All Clocks Stopped			1	2	$t_{BIT}+t_{XC}$
t_{PCR}	Power Down Removed to Clocks Running			1	2	$t_{BIT}+t_{XC}$
t_{PXS}	Power Down Removed to XTAL Oscillator Stable	When Using On Chip Inverter for Oscillator Circuit		100		ms
t_{PSE}	Power Down Set-Up to $\overline{RD}$ or $\overline{WR}$ Edge		160	260		ns
t_{EPI}	$\overline{WR}$ or $\overline{RD}$ Edge Following $\overline{PD}$ to Internal Signals	Enable or Disable		100		ns
BAUD GENERATOR						
t_{XH}	XTAL In High		100			ns
t_{XL}	XTAL In Low		100			ns
f_{BRC}	Baud Rate Clock Input Frequency				4.1	MHz
t_{BD1}	Baud Out Delay ÷ 1			160		ns
t_{BD2}	Baud Out Delay ÷ 2			200		ns
t_{BD3}	Baud Out Delay ÷ 3			200		ns
t_{BDN}	Baud Out Delay ÷ N > 3			200		ns
t_{XC}	Baud Clock Cycle	$t_{XC} = \frac{1}{f_{BRC}}$	243			ns

4.0 AC Electrical Characteristics (Continued)

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
TRANSMITTER						
t_{TCD}	TxD Delay from $\overline{TxC}$	External Clock		275		ns
		Internal Clock		140		ns
t_{TXC}	Cycle Time $\overline{TxC}$	16X, 32X, 64X Clock Factor	243			ns
		1X Clock Factor	1000			ns
t_{TCH}	$\overline{TxC}$ High		100			ns
t_{TCL}	$\overline{TxC}$ Low		100			ns
t_{HRI}	$\overline{WR}$ TxHR to Reset TxBE $\overline{RTI}$			260		ns
t_{HTS}	$\overline{WR}$ TxHR to TxD Start		2	3	4	t_{BIT}
t_{TSI}	Skew Start Bit to $\overline{RTI}$		−100	+20	+120	ns
t_{ETS}	Enable Tx to Start Bit		3	4	5	t_{BIT}
t_{BIT}[1]	One Bit Time	1X	1000			ns
		16X	3.88			μs
		32X	7.77			μs
		64X	15.55			μs
RECEIVER						
t_{RS}	RxD Set-Up	1X Clock Factor		160		ns
t_{RH}	RxD Hold	1X Clock Factor		100		ns
t_{RXC}	Cycle time $\overline{RxC}$	16X, 32X, 64X Clock Factor	243			ns
		1X Clock Factor	1000			ns
t_{RCH}	$\overline{RxC}$ High		100			ns
t_{RCL}	$\overline{RxC}$ Low		100			ns
t_{RRI}	$\overline{RD}$ to Reset $\overline{RTI}$			300		ns
t_{BIT}[1]	One Bit Time	1X	1000			ns
		16X	3.88			μs
		32X	7.77			μs
		64X	15.55			μs
t_{ERS}	Enable Rx to Correctly Detect Start Bit	All Clock Factors	2	3	4	t_{RXC}
t_{RNO}	Read RxHR Before Next Data; No OE		240			ns
t_{BI}	$\overline{RxC}$, Break to $\overline{RTI}$			340		
t_{REI}	Receiver Error Int			½ Clock Factor		t_{RXC}
t_{RDI}	Receiver Ready Int			$t_{REI}+1$		t_{RXC}
t_{RSI}	$\overline{RxC}$ to $\overline{RTI}$			300		ns
RESET TIMING						
t_{MR}	MR Pulse Width			100		ns
t_{RA}	MR to ALE if Valid $\overline{WR}$ or $\overline{RD}$ Cycle			100		ns

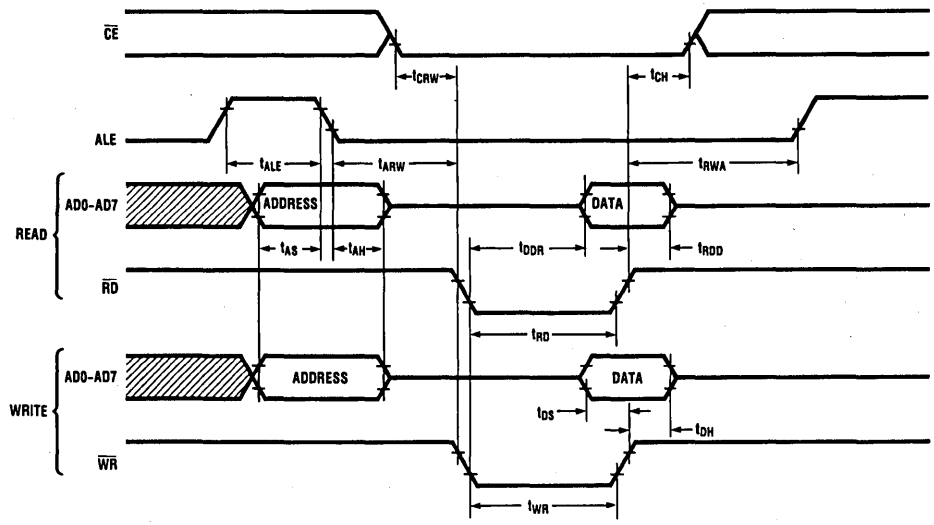
Note 1: $t_{BIT} = t_{TXC} \times$ Clock Factor (1, 16, 32, 64), transmitter

$t_{BIT} = t_{RXC} \times$ Clock Factor (1, 16, 32, 64), receiver

$$t_{BIT} = \frac{1}{\text{Baud Rate}}$$

5.0 Timing Waveforms

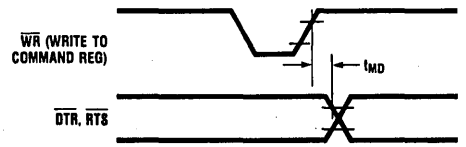
Read and Write Cycles



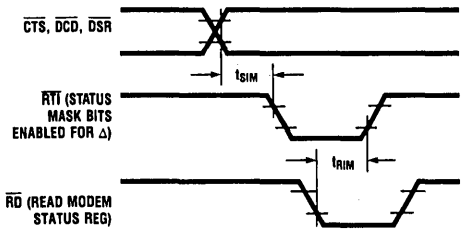
TL/C/5593-4

Note: The internal write is made inactive by either the next ALE or $\overline{CE}$ going invalid

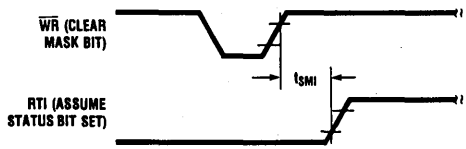
Modem Timing



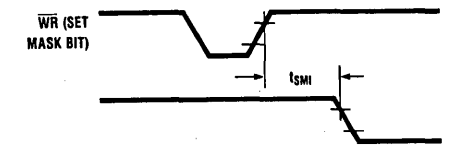
TL/C/5593-5



TL/C/5593-6



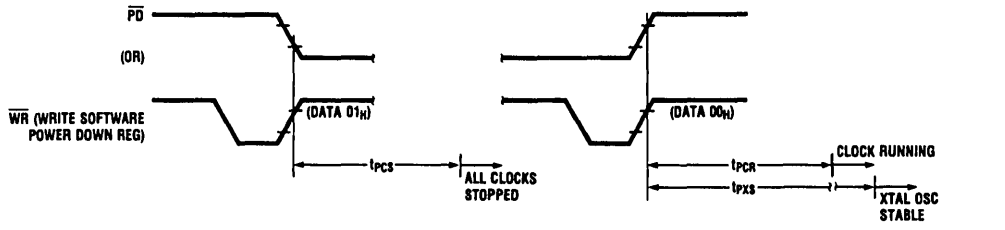
TL/C/5593-7



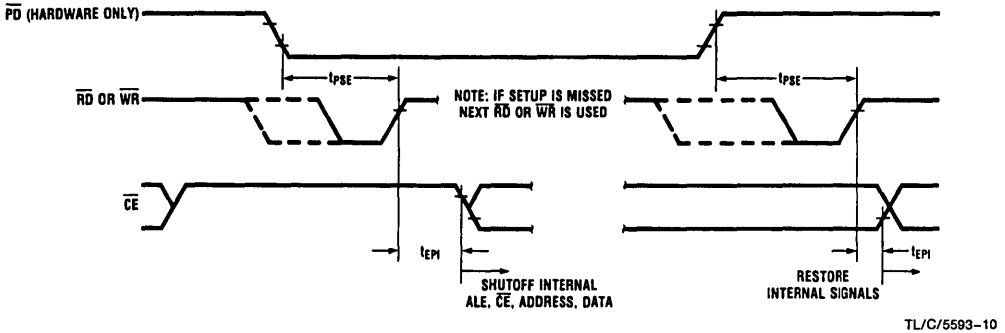
TL/C/5593-8

5.0 Timing Waveforms (Continued)

Power Down Timing

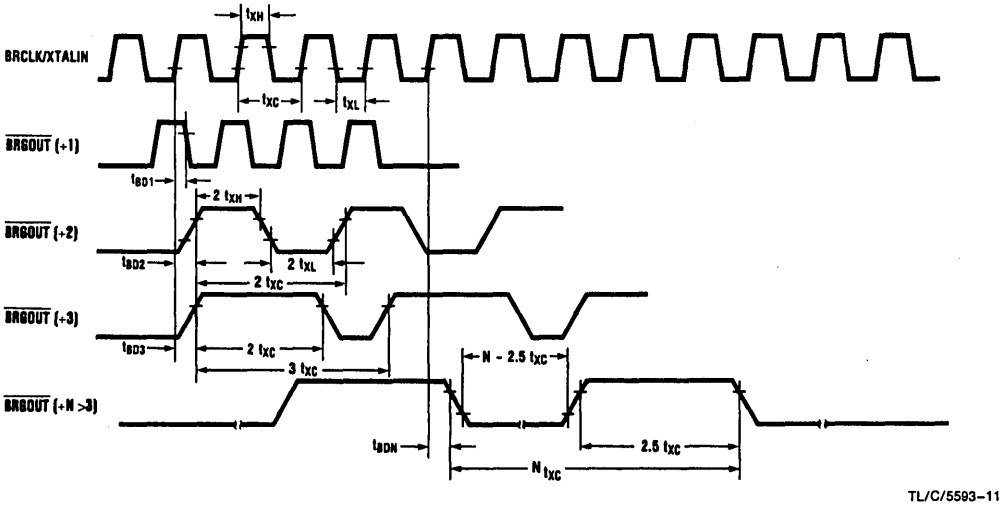


TL/C/5593–9



TL/C/5593–10

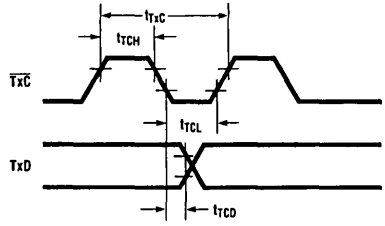
Baud Out Timing



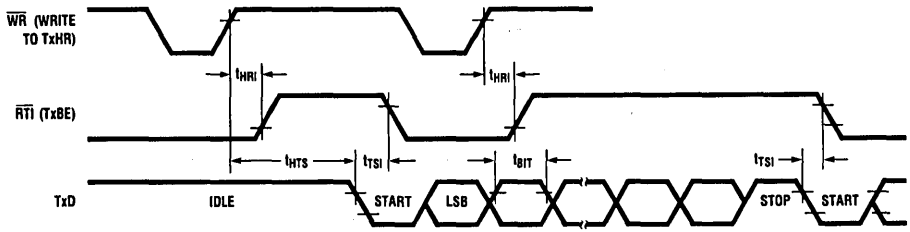
TL/C/5593–11

5.0 Timing Waveforms (Continued)

Transmitter Timing



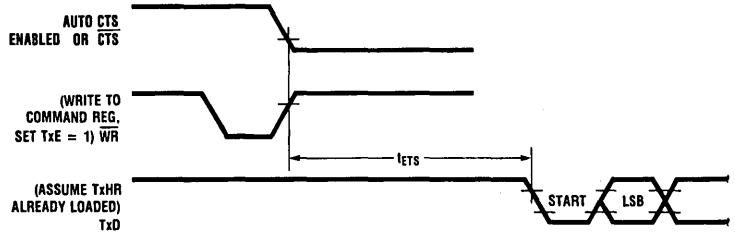
TL/C/5593–12



$$t_{BIT} = \frac{1}{\text{BAUD RATE}} = t_{TxC} \times \text{CLOCK FACTOR (1, 16, 32, 64)}$$

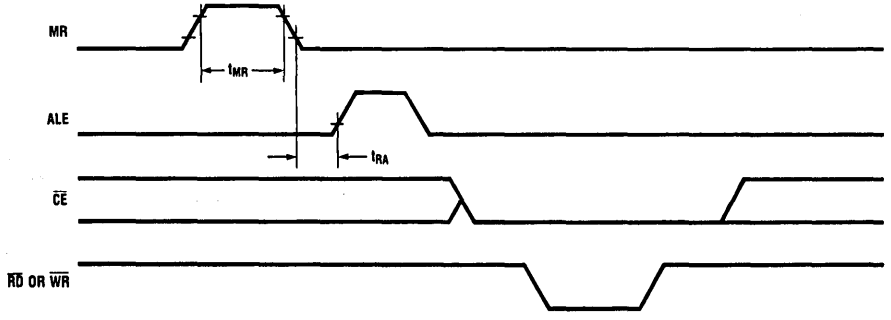
TL/C/5593–13

Note: The AC Timing Spec for $\overline{RTI}$ due to TXU or TBK will be published in the next data sheet.



TL/C/5593–14

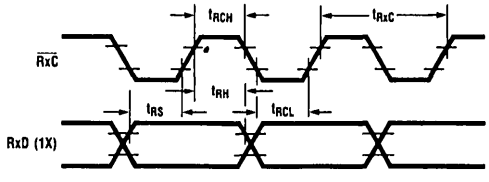
Reset Timing



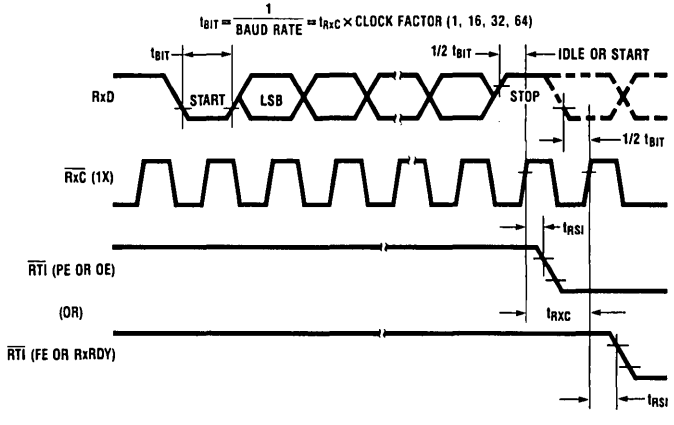
TL/C/5593–15

5.0 Timing Waveforms (Continued)

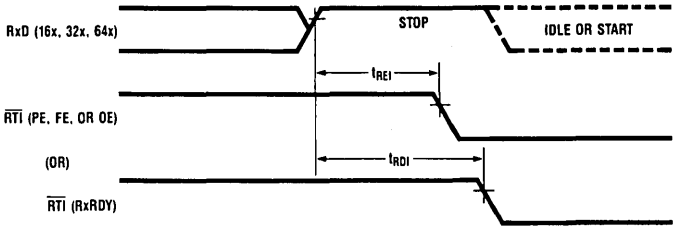
Receiver Timing



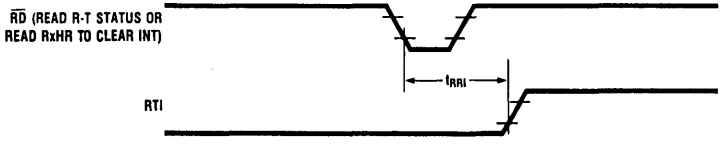
TL/C/5593-16



TL/C/5593-17



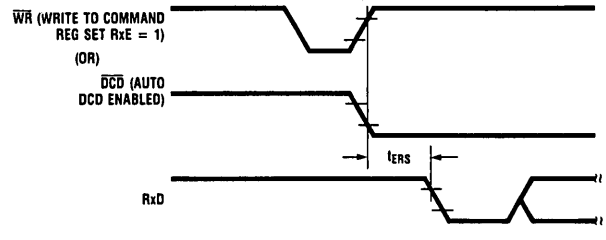
TL/C/5593-18



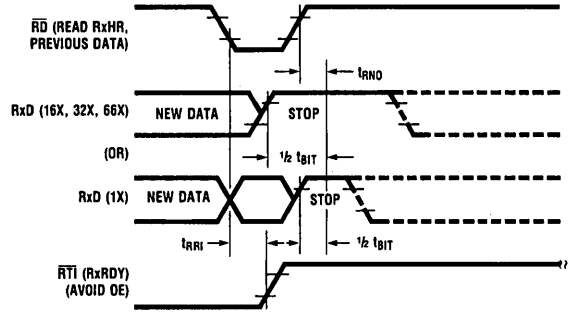
TL/C/5593-19

5.0 Timing Waveforms (Continued)

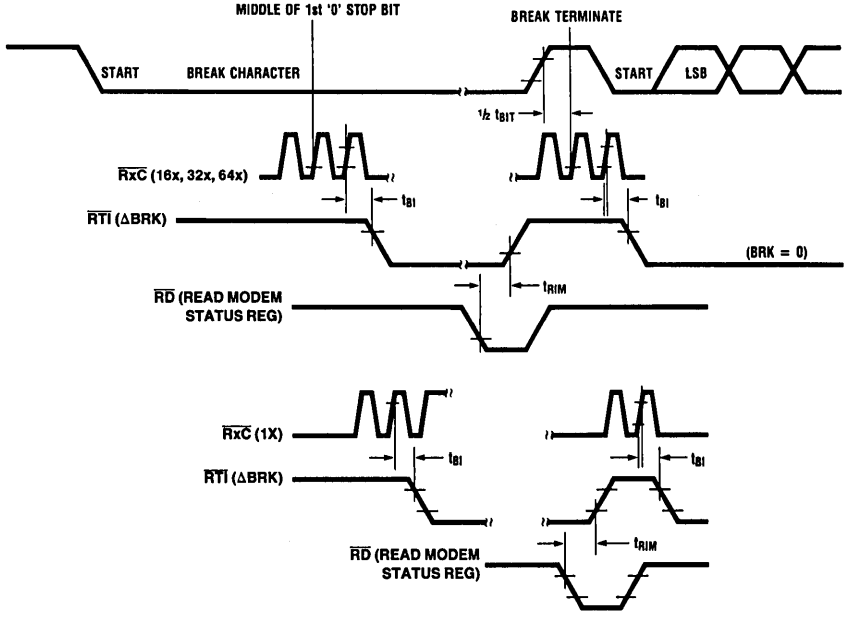
Receiver Timing (Continued)



TL/C/5593–20



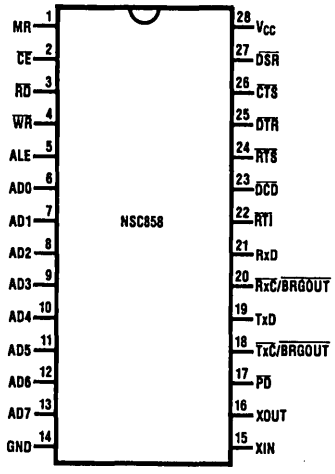
TL/C/5593–21



TL/C/5593–22

6.0 Connection Diagrams

Dual-In-Line Package

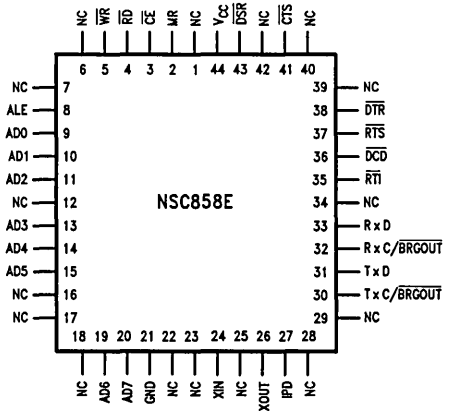


TL/C/5593-23

Top View

Order Number NSC858D or N
See NS Package D28C or N28B

Leadless Chip Carrier

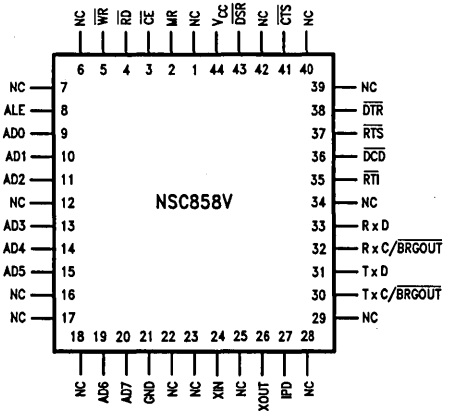


TL/C/5593-24

Top View

Order Number NSC858E
See NS Package Number E44A

Plastic Chip Carrier



TL/C/5593-25

Top View

Order Number NSC858V
See NS Package Number V44A

7.0 Pin Descriptions

7.1 INPUT SIGNALS

Master Reset (MR): active high, Pin 1. This Schmitt trigger input has a 0.5V typical hysteresis. When high, the following registers are cleared: receiver mode, transmitter mode, global mode, R-T status (except for TxBE which is set to one), R-T status mask, modem mask, command (which disables receiver "Rx" and the transmitter "Tx"), power down, and receiver holding. In the modem status register, ΔCTS, ΔDCD, ΔDSR, BRK and ΔBRK are cleared.

Chip Enable ($\overline{CE}$): active low, Pin 2. Chip enable must be low during a valid read or write pulse in order to select the device. Chip enable is not latched.

Read ($\overline{RD}$): active low, Pin 3. While the chip is enabled the CPU latches data from the selected register on the rising edge of $\overline{RD}$.

Write ($\overline{WR}$): active low, Pin 4. While the chip is enabled it latches data from the CPU on the rising edge of $\overline{WR}$.

Address Latch Enable (ALE): negative edge sensitive, Pin 5. The negative edge (high to low) of ALE latches the address for the register select during a read or write operation.

7.0 Pin Descriptions (Continued)

Power Down ($\overline{PD}$): active low, Pin 17. When active it disables all internal clocks, shuts off the oscillator, clears RxE, TxE, and break control bits in the command register. All other registers retain their data. Unlike software power down, $\overline{PD}$ also disables the internal ALE, $\overline{CE}$, $\overline{RD}$, $\overline{WR}$, address and data paths for minimum power consumption. Registers cannot be accessed in hardware power down; they may be in software power down.

Receiver Data (RxD): Pin 21. This accepts serial data input from the communications link (peripheral device, modem, or data set). Serial data is received least significant bit (LSB) first. "Mark" is high (1), "space" is low (0).

Data Carrier Detect ($\overline{DCD}$): active low, Pin 23. Can be used as a modem or general purpose input. When this modem input is low it indicates that the data carrier has been detected by the modem or data set. The $\overline{DCD}$ signal is a modem control function input whose complement value can be tested by the CPU by reading bit 5 (DCD) of the modem status register. Bit 1 (ΔDCD) of the modem status register indicated whether the $\overline{DCD}$ input has changed state since the previous reading of the modem status register. $\overline{DCD}$ can also be programmed to become an auto enable for the receiver.

NOTE: Whenever the DCD bit of the modem status register changes state, an interrupt is generated if the ΔDCD mask and the DSCHG mask bits are set.

Clear to Send ($\overline{CTS}$): active low, Pin 26. Can be used as a modem or a general purpose input. The $\overline{CTS}$ inputs complement can be tested by the CPU by reading bit 4 (CTS) of the modem status register. Bit 0 (ΔCTS) of the modem status register indicates whether the $\overline{CTS}$ input has changed state since the previous reading of the modem status register. $\overline{CTS}$ can be programmed to automatically enable the transmitter. Note: Whenever the CTS bit of the modem status register changes state, an interrupt is generated if the ΔCTS mask and the DSCHG mask bits are set.

Data Set Ready ($\overline{DSR}$): active low, Pin 27. Can be used as a modem or a general purpose input. When this modem input is low it indicates that the modem or data set is ready to establish the communication link and transfer data with the NSC858. The $\overline{DSR}$ is a modem-control function input whose complement value can be tested by the CPU by reading bit 6 (DSR) of the modem status register. Bit 2 (ΔDSR) of the modem status register indicates whether the ($\overline{DSR}$) input has changed state since the previous reading of the modem status register.

NOTE: Whenever the DSR bit of the modem status register changes state, an interrupt is generated if ΔDSR mask and the DSCHG mask bits are set.

Power (V_{CC}): Pin 28. +5V supply.

Ground (GND): Pin 14. Ground (0V) supply.

7.2 OUTPUT SIGNALS

Transmit Data (TxD): Pin 19: Composite serial data output to the communication link (peripheral, modem or data set) least significant bit first. The TxD signal is set to the marking (logic 1) state upon a master reset. In hardware or software power down this pin will always be a one.

Receiver-Transmitter Interrupt ($\overline{RTI}$): active low, Pin 22. Goes low when any R-T status register bit and its corresponding mask bit are set. This bit can change states during either hardware or software power down due to a change in modem status information.

Request to Send ($\overline{RTS}$): active low, Pin 24. Can be used as a modem or a general purpose output. When this modem output is low it informs the modem or data set that the NSC858 is ready to transmit data. The $\overline{RTS}$ output or general purpose output signal can be set to an active low by programming bit 6 of the command register with a 1. The $\overline{RTS}$ signal is set high upon a master reset operation. During remote loopback $\overline{RTS}$ signal reflects the complement of bit 6 of the command register. During local loopback the $\overline{RTS}$ signal is forced to its inactive state (high). $\overline{RTS}$ cannot change states during hardware power down; it can during software power down.

Data Terminal Ready ($\overline{DTR}$): active low, Pin 25. Can be used as a modem or general purpose output. When this modem output is low it informs the modem or data set that the NSC858 is ready to communicate. The $\overline{DTR}$ output or the general purpose output signal can be set to an active low by programming bit 7 of the command register with a 1. The $\overline{DTR}$ signal is set high upon a master reset operation. During remote loopback $\overline{DTR}$ signal reflects the complement of bit 7 of the command register. During local loopback the $\overline{DTR}$ signal is forced to its inactive state (high). $\overline{DTR}$ signal cannot change state during hardware power down; it can during software power down.

7.3 INPUT/OUTPUT SIGNALS

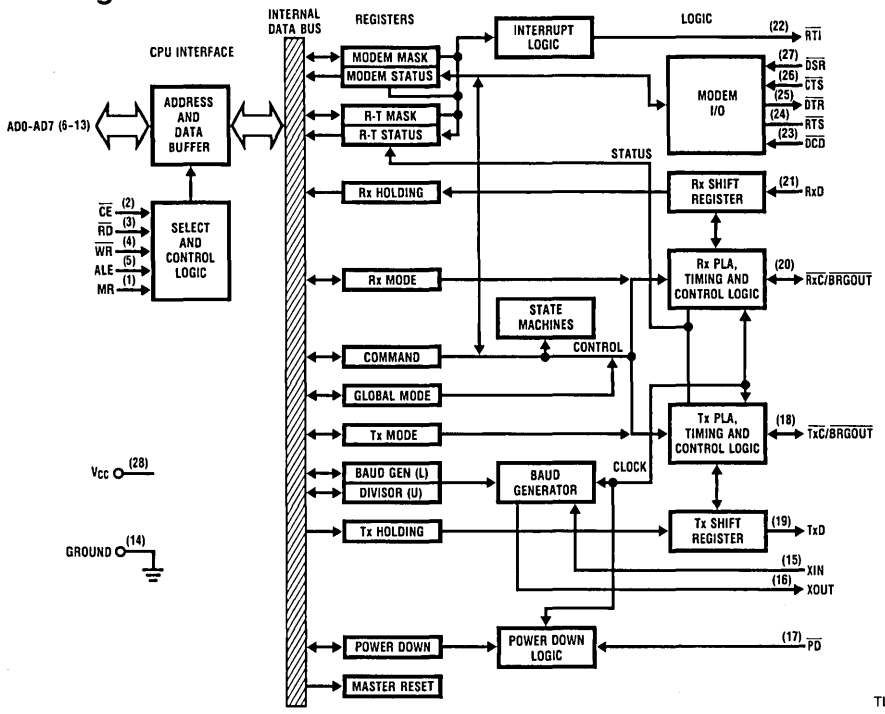
Address/Data Bus (AD0–AD7): Pins 6–13. The multiplexed bidirectional address/data bus, AD0–AD7 pins, are in the high impedance state when the NSC858 is not selected or whenever it is in hardware power down. AD0–AD3 are latched on the trailing edge of ALE, providing the four address inputs. The rising edge of the $\overline{WR}$ input enables 8 bits to be written in, through AD0–AD7, to the addressed register. $\overline{RD}$ input enables 8 bits to be read from a register out through AD0–AD7.

Transmitter Clock/Baud Rate Generator Output ($\overline{TxC}$/$\overline{BRGOUT}$): Pin 18. If the transmitter is programmed for an external clock, $\overline{TxC}$ is an input. If the transmitter is programmed for an internal clock, then the Baud Rate Generator is used for the transmitter, and it is output at $\overline{TxC}$/$\overline{BRGOUT}$. In either case, $\overline{TxC}$/$\overline{BRGOUT}$ signal is running at 1X, 16X, 32X, 64X the data rate, as selected by the clock factor. If this pin is used as an output it will be set to a zero (0) in both hardware and software power down.

Receiver Clock/Baud Rate Generator Output ($\overline{RxC}$/$\overline{BRGOUT}$): Pin 20. If the receiver is programmed for an external clock, RxC is an input. If the receiver is programmed for an internal clock, the Baud Rate Generator is used for the receiver, and it is output at $\overline{RxC}$/$\overline{BRGOUT}$. In either case, $\overline{RxC}$/$\overline{BRGOUT}$ signal is running at 1X, 16X, 32X, 64X, the data rate as selected by the clock factor. If this pin is programmed as an output it will be set to one (1) in both hardware and software power down.

Crystal (XIN, XOUT): Pins 15, 16. These two pins connect the main timing reference. A crystal network can be connected across these two pins, or a square wave can be driven into XIN with XOUT left floating. In hardware and software power down XOUT is set to a 1. Ground XIN when using both RxC and TxC to supply external clocks to the UART.

8.0 Block Diagram



TL/C/5593-26

FIGURE 1. NSC858 Functional Block Diagram

9.0 Registers

The system programmer may access control of any of the NSC858 registers summarized in Table I via the CPU. These 8-bit registers are used to control NSC858 operation and to transmit and receive data.

TABLE I. Register Address Designations

Address				Register	Read/ Write
A_3	A_2	A_1	A_0		
0	0	0	0	Rx Holding	R
0	0	0	0	Tx Holding	W
0	0	0	1	Receiver Mode	R/W
0	0	1	0	Transmitter Mode	R/W
0	0	1	1	Global Mode	R/W
0	1	0	0	Command	R/W
0	1	0	1	Baud Rate Generator Divisor Latch (Lower)	R/W
0	1	1	0	Baud Rate Generator Divisor Latch (Upper)	R/W
0	1	1	1	R-T Status Mask	R/W
1	0	0	0	R-T Status	R
1	0	0	1	Modem Status Mask	R/W
1	0	1	0	Modem Status	R
1	0	1	1	Power Down	R/W
1	1	0	0	Master Reset	W

Note: Offset address OD, OE, OF are unused.

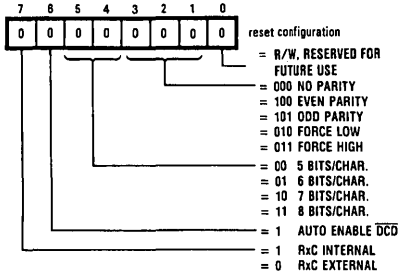
9.1 RECEIVER AND TRANSMITTER HOLDING REGISTER

A read to offset location 00 will access the Receiver holding register; a write will access the Transmitter holding register.

9.2 RECEIVER MODE REGISTER

The system programmer specifies the data format of the receiver (which may differ from the transmitter) by programming the Receiver mode register at offset location "01." This read/write register programs the parity, bits/character, auto enable option, and clock source. When bit 6 of this register is set high the receiver will be enabled any time the $\overline{DCD}$ signal input is low (provided CR0 = 1). When bit 7 is set to a "1" the receiver clock source is the internal baud rate generator and $\overline{RxC}$ is then an output. After reset this register is set to "00."

TABLE II. Receiver Mode Register (Address "01") (Bits RMO-7)



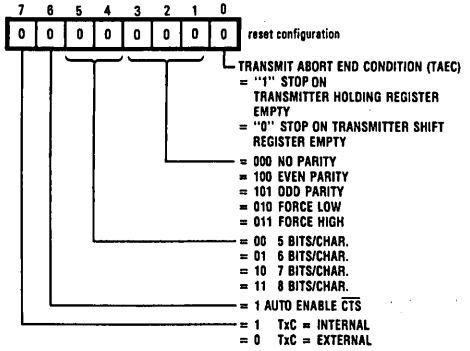
TL/C/5593-27

9.0 Registers (Continued)

9.3 TRANSMITTER MODE REGISTER

The system programmer specifies the data format of the transmitter (which may differ from the receiver) by programming the transmitter mode register at offset location "02."

TABLE III. Transmit Mode Register (Address "02") (Bits TM0–7)



TL/C/5593–28

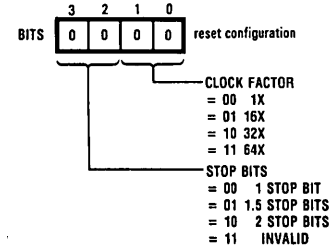
The transmitter mode register is similar in operation to the receiver mode register except for the addition of the Transmit Abort End Condition (TAEC). If this bit is set to a one when a request to disable the transmitter or send a break is pending then the data in the shift register and holding register will be transmitted prior to such action occurring. If TAEC equals 0 then the action will take place after the shift register has been emptied. When bit 6 of this register is set high the transmitter will be enabled any time the $\overline{CTS}$ signal is low (provided CR1 = 1). When bit 7 is set to a "1" the transmitter clock source is the internal baud rate generator, and $\overline{TxC}$ is then an output. After reset this register is set to "00."

9.4 GLOBAL MODE REGISTER

This register is used to program the number of stop bits and the clock factor for both the receiver and transmitter. Only the lower four bits of this register are used, the upper four can be programmed as don't cares and they will be read back as zeros. Programming the number of stop bits is for the transmitter only; the receiver always checks for one stop bit. If a 1X clock factor with 1.5 stop bits is selected for the transmitter the number of stop bits will default to 1. After reset this register is set to "00."

Note: Selecting the 1x clock requires that the clock signal be sent or received along with the data.

TABLE IV. Global Mode Register (Address "03") (Bits GM0–3)



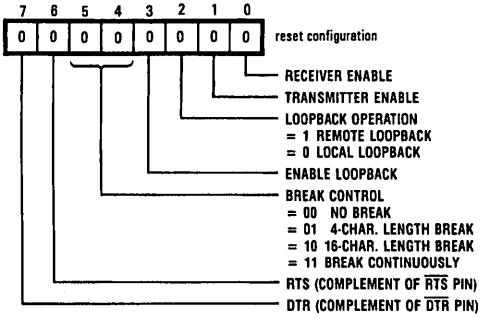
TL/C/5593–29

Bits 4–7 are don't care, read as 0s.

9.5 COMMAND REGISTER

The Command register is an eight bit read/write register which is accessed at offset location "04." After reset the command register equals "00."

TABLE V. Command Register (Address "04") (Bits CR0–7)



TL/C/5593–30

Bit 0: Receive Enable, when set to a one the receiver is enabled. If auto enable for the receiver has been programmed then in addition to CR0 = 1, the $\overline{DCD}$ input must be low to enable receiver.

Bit 1: Transmitter Enable, when set to a one the transmitter is enabled. If auto enable for the transmitter is programmed then in addition to CR1 = 1, the $\overline{CTS}$ input must be low to enable transmitter.

Bit 2: A zero selects local loopback and a one selects remote loopback.

Bit 3: A one enables either of the diagnostic modes selected in bit 2 of the command register.

Bits 4 and 5: Bits 4 and 5 of the command register are used to program the length of a transmitted break condition. A continuous break must be terminated by the CPU, but the 4 and 16 character length breaks are self clearing. (At the beginning of the last break character bits 4 and 5 will automatically be reset to 0.) Break commands affect the status of bit 6 (TBK) of the R-T Status register (see R-T Status register). Break control bits are cleared by software or hardware power down.

Bits 6 and 7: These two bits control the status of the output pins $\overline{RTS}$ (pin 24) and $\overline{DTR}$ (pin 25) respectively. They may be used as modem control functions or be used as general purpose outputs. The output pins will always reflect the complement of the register bits.

9.6 R-T STATUS REGISTER

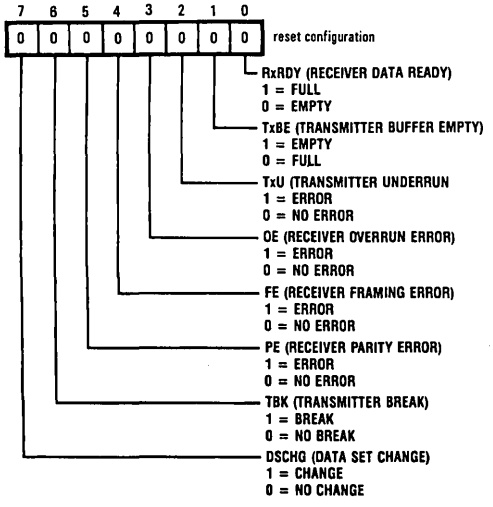
This 8-bit register contains status information of the NSC858 and therefore is a read only register at offset location "08." Each bit in this register can generate an interrupt ($\overline{RTI}$). If any bit goes active high and its associated mask bit is set then the $\overline{RTI}$ will go low. $\overline{RTI}$ will be cleared when all unmasked R-T Status bits are cleared. Bits 0 and 1, receiver ready and transmitter empty are cleared by reading the receiver holding register or writing the transmitter holding register respectively. Bits 2 through 5, transmit underrun, receiver overrun, framing error, parity error are cleared by reading the R-T Status register. Bit two, transmitter underrun will occur when both the transmit holding register and the transmit shift register are empty.

9.0 Registers (Continued)

Bit three, overrun error, will occur when the CPU does not read a character before the next one becomes available. The OE bit informs the programmer or CPU that RXHR data has been overrun or overwritten. The byte in the shift register is always transferred to the holding register, even after an overrun occurs. If an OE occurs, it is standard protocol to request a re-transmission of that block of data. A read of RXHR, when a subsequent read of R-T status shows that no OE is present, indicates current receiver data is available. Bit four, framing error, occurs when a valid stop bit is not detected. Bit 5 is set when a parity error is detected. Bits three, four and five are affected by the receiver only.

Bit 6, Transmit Break (TBK) is set at the beginning of each break character during a break continuously command, or at the beginning of the final break character in a 4 or 16 character programmed break length. It is cleared by reading the R-T Status register. Bit 7, Data Set Change (DSCHG) will be set whenever any of the bits 0–3 of the Modem Status register and their associated mask bit are set. Data Set Change bit is cleared by reading the Modem Status register or is masked off by writing "0" to all modem register bits. After reset the R-T Status register equals '02', i.e. all bits except TxBE are reset to zero.

TABLE VI. R-T Status Register (Address "08") (Bits SR0–7)



TL/C/5593–31

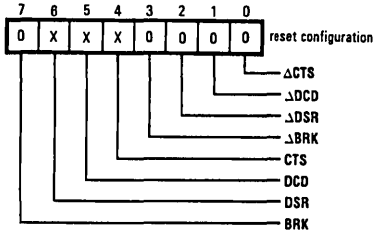
9.7 R-T STATUS MASK REGISTER (SM0–7)

This register is used in conjunction with the R-T Status register to enable or disable conditional interrupts A one in any bit unmasks its associated bit in the R-T Status register, and allows it to generate an interrupt out through $\overline{RTI}$. The mask affects only the interrupt and not the R-T Status bits. This eight bit register is both read and writable at offset location "07." After reset it is set to "0" which disables all interrupts. Each bit in the R-T Status mask register is associated with that bit in the R-T Status register (e.g., SM0 is SR0's mask).

9.8 MODEM STATUS

This eight bit read only register which is addressed at offset location "0A" contains modem or general purpose input and receiver break information.

TABLE VII. Modem Status Register (Address "0A") (Bits MS0–7)



TL/C/5593–32

Each of the four status signals in this register also have an associated delta bit in this register. Each delta bit (bits MS0–3) will be set when its corresponding bit changes states. These four delta bits are cleared when the Modem Status register is read. If any of these four delta bits and associated mask bits are set they will force DSCHG (bit 7) of the R-T Status register high. Bits 4–6, CTS, DCD, DSR can be used as modem signals or general purpose inputs. In either case the value in the register represents the complements of the input pins $\overline{CTS}$ (pin 26), $\overline{DCD}$ (pin 23), and $\overline{DSR}$ (Pin 27). Bit 7 (BRK) when set to a one indicates that the receiver has detected a break condition. It is cleared when break terminates. After reset ΔCTS, ΔDCD, ΔDSR, ΔBRK and BRK are cleared.

9.9 MODEM MASK REGISTER (MM0–3)

This 4-bit read/write register, which is addressed at offset location "09," contains mask bits for the four delta bits of the Modem Status register (MS0–3). A one ("1") in any of three bits and a one in the associated delta bit of the Modem Status register will set the DSCHG bit of the R-T Status register. Modem Mask bit 0 is associated with Modem Status bit 0, etc. The four (4) most significant bits of this register will read as zeros. After reset the register equals '00'.

9.10 POWER DOWN REGISTER (PD0)

This one bit register can both be read and written at offset location "0B." When bit zero is set to a one the NSC858 will be put into software power down. This disables the receiver and transmitter clocks, shuts off the baud rate generator and crystal oscillator, and clears the RxE, TxE, and break control bits in the command register. Registers on chip can still be accessed by the CPU during software power down. Bits 1 through 7 will always read as 0.

9.11 MASTER RESET REGISTER

This write only register is addressed at offset location "0C." When writing to this register the data can be any value (don't cares). Resetting the NSC858 by way of the reset register is functionally identical to resetting it by the MR pin.

9.12 BAUD RATE GENERATOR DIVISOR LATCH

These two 8-bit read/write registers which are accessed at offset locations "05" (lower) and "06" (upper) are used to program the baud rate divisor. These registers are not affected by the reset function and are powered up in a random state.

10.0 Functional Description

10.1 PROGRAMMABLE BAUD GENERATOR

The NSC858 contains a programmable Baud Generator that is capable of taking any clock input (DC to 4.1 MHz) and dividing it by any divisor from 1 to ($2^{16}-1$). The output frequency of the Baud Generator (available at $\overline{TxC}/\overline{BRGOUT}$ or $\overline{RxC}/\overline{BRGOUT}$, if internal $\overline{TxC}$ or $\overline{RxC}$ is selected) is equal to the clock factor (1X, 16X, 32X, 64X) times the baud rate. The divisor number is determined by the following equation:

$$\text{divisor \#} = \frac{\text{Frequency Input } (f_{BRC})}{[\text{Baud Rate} \times \text{Clock Factor (1, 16, 32, 64)}]}$$

Two 8-bit latches store the divisor in a 16-bit binary format. These Divisor Latches must be loaded during initialization in order to ensure desired operation of the Baud Generator. Upon loading either of the Divisor Latches, a 16-bit Baud counter is immediately loaded. This prevents long counts on initial load.

Tables VIII and IX illustrate the use of the Baud Generator with crystal frequencies of 1.8432 MHz and 3.072 MHz respectively. For baud rates of 38400 and below, the error obtained is minimal. The accuracy of the desired baud rate is dependent on the crystal frequency chosen.

TABLE VIII. Baud Rates Using 1.8432 MHz Crystal

Desired Baud Rate	Divisor Used To Generate 16 x Clock	Percent Error Difference Between Desired and Actual
50	2304	—
75	1536	—
110	1047	0.026
134.5	857	0.058
150	768	—
300	384	—
600	192	—
1200	96	—
1800	64	—
2000	58	0.69
2400	48	—
3600	32	—
4800	24	—
7200	16	—
9600	12	—
19200	6	—
38400	3	—
56000	2	2.86

TABLE IX. Baud Rates Using 3.072 MHz Crystal

Desired Baud Rate	Divisor Used To Generate 16 x Clock	Percent Error Difference Between Desired and Actual
50	3840	—
75	2560	—
110	1745	0.026
134.5	1428	0.034
150	1280	—
300	640	—
600	320	—
1200	160	—
1800	107	0.317
2000	96	—
2400	80	—
3600	53	0.628
4800	40	—
7200	27	1.23
9600	20	—
19200	10	—
38400	5	—

10.2 RECEIVER AND TRANSMITTER OPERATION

The NSC858 transmits and receives data in an asynchronous communications mode. The CPU must set up the appropriate mode of operation, number of bits per character, parity, number of stop bits, etc. Separate mode registers exist for the independent specification of receiver and transmitter operation. These independent specifications include parity, character length, and internal or external clock source. Only the Global Mode Register, which controls the number of stop bits and the clock factor, exercises common control over the receiver and transmitter (receiver looks for only one stop bit).

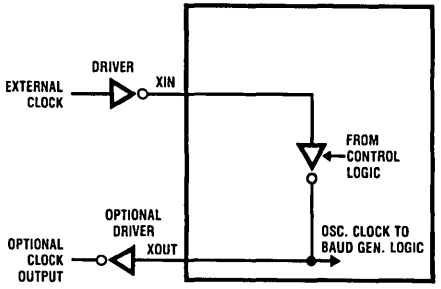
10.3 TRANSMITTER OPERATION

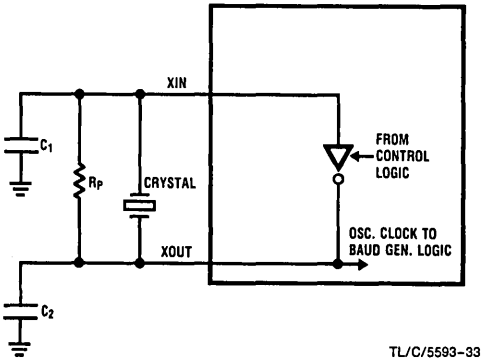
The Transmitter Holding register is loaded by the CPU. To enable the transmitter, TxE must be set in the Command register. $\overline{CTS}$ must be low if the auto enable is set in the Tx Mode register. The Transmitter Holding register is then parallel loaded into the Transmitter Shift register, and the start bit, parity bit and the specified number of stop bits are inserted. This serialized data is available at the TxD output pad, and changes on the rising edge of $\overline{TxC}$, or equivalently the falling edge of $\overline{TxC}$. The TxD output remains in a mark ("1") condition when no data is being transmitted, with the exception of sending a break ("0").

A break condition is initiated by writing either a continuous or specified length break request to the Command Register. A finite break specification of either 4 or 16 character lengths can be extended by re-writing the break command before the specified break length is completed. Each break character is transmitted as a start bit, logical zero data, logical zero parity (if specified) and logical zero stop bit(s). The number of data and stop bits, plus the presence of a parity bit are determined by the Transmitter and Global Mode registers. Thus, the total number of (all zero) bits in a break character is the same as that for data. The break is terminated by writing "00" to the Break Control bits in the Command Register. The Set Break bits in the Command register are always reset to "00" after the termination of the specified break transmission or if the transmitter is disabled during a break transmission. The TxD output will always return to a mark condition for at least one bit time before transmitting a character after a break condition. Data in the Transmitter Holding register, whether loaded before (on TAEC=0) or during the break will be transmitted after the break is terminated.

10.0 Functional Description (Continued)

10.4 TYPICAL CLOCK CIRCUITS

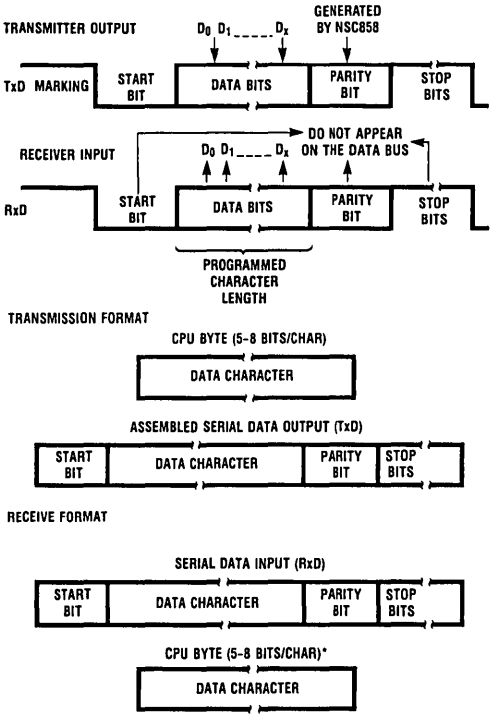




TL/C/5593-33

CRYSTAL	R_P	C_1	C_2
3.1 MHz	1 MΩ	10–30 pF	40–60 pF
1.8 MHz	1 MΩ	10–30 pF	40–60 pF

FIGURE 2. Typical Crystal Oscillator Network



TL/C/5593-34

Note: If character length is defined as 5, 6 or 7 bits, the unused bits are set to "0").

FIGURE 3

10.5 RECEIVER OPERATION

The NSC858 receives serial data on the RxD input. To enable the receiver, $\overline{DCD}$ must be low if the $\overline{DCD}$ Auto Enable bit in the Receiver Mode register is set ("1"). RxE must be set in the Command register. RxD is sampled on the falling edge of RxC or equivalently on the rising edge of $\overline{RxC}$. If a high ("1") to low ("0") transition of RxD is detected, RxD is sampled again, for all except the 1X clock factor, at ½ of a bit time later. If RxD is still low, then a valid start bit has been received and character assembly proceeds. If RxD has returned high, then a valid start bit has not been received, and the search for a valid start bit continues. When a character has been assembled in the Receiver Shift Register and transferred to the Receiver Holding Register, the RxRDY bit (and any error bits that may have occurred) in the R-T Status register will be set and $\overline{RTI}$ will go low (if the proper mask bits are set). After the CPU reads the Receiver Holding register, the RxRDY will go low and the $\overline{RTI}$ will go inactive ("1").

The receiver will detect a break condition on RxD if an all zero character with zero parity bit (if parity is specified) and a zero stop bit is received. For the break condition to terminate, RxD must be high for one half a bit time. If a break condition is detected, bits 3 and 7 in the Modem Status register (ΔBRK and BRK respectively) will be set. Bit 3 (ΔBRK) will then cause bit 7 (DSCHG) in the R-T Status register to be set which in turn forces $\overline{RTI}$ to an asserted state ("0"). These interrupts will occur only if the appropriate mask bits are set for the registers in question.

When the 1x clock factor is selected:

The RxC pin on the NSC858 should be connected to the clock signal of the incoming data stream and bit 7 of the receiver mode register should be cleared to A0.

The TxC output of the NSC858 does not have to be sent to the remote receiver unless the receiver is using a 1x clock factor.

10.6 PROGRAMMING THE NSC858

There are two distinct steps in programming the 858. During initialization, the modes, clocks, masks and commands are set up. Then, in operation, Modem I/O takes place, status is monitored, the receiver and transmitter are run as needed.

To initialize the 858, first pulse the MR line or write to the Master Reset register. Then, write to the following registers in any order, except for enabling the Rx and Tx, which must

10.0 Functional Description (Continued)

be at the end of the set up procedure. The Global, Receiver and Transmitter Mode registers determine the modes for the Rx and Tx. These latter two registers often will have the same data byte written to them, but are kept independent for flexibility. If the mode registers indicate that the receiver and/or the transmitter use an internal clock, then data (determined by the crystal frequency and desired bit time and clock factor) should be written to the upper and lower Baud Rate Generator Divisor Latches. The Modem Status Mask register enables Data Set change in R-T Status. If interrupts are required, the R-T Status Mask register allows $\overline{RTI}$ to occur. Write to the Command register to enable the receiver and/or transmitter only when all else is set up.

In operation, the 858 can transmit, receive and handle I/O simultaneously. Modem outputs are written to at the Command register, while the inputs are read at the Modem Status register. Data flow and errors are read at the R-T Status register. When serial data has been shifted in and assembled, the receiver is ready, and the word can be read at the Rx Holding register. When the transmitter buffer is empty, the Tx Holding register can be written to, and the word will be shifted out as serial asynchronous data.

Once the 858 is running, several options may be exercised. Masks can be changed at any time. The Rx and Tx are disabled or enabled, as needed, by writing to the Command register, or toggling the auto enable modem inputs (if used). Both the Rx and Tx should be disabled before either altering any mode or engaging a loopback diagnostic, and they can be re-enabled then or at a later time. Power down is allowed at any time except during loopback, although data may be lost if PD occurs in the middle of a word.

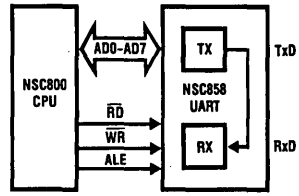
Thus, software for the NSC858 is of two types. The initialization routine is performed once. The operation routines, usually incorporating polling or interrupts, are then run continuously or on demand, depending upon the system or application.

10.7 DIAGNOSTIC CAPABILITIES

The NSC858 offers both remote and local loopback diagnostic capabilities. These features are selected through the Command register.

Local Loopback Mode (see *Figure 4*)

1. The transmitter output is internally connected to the receiver input.
2. $\overline{DTR}$ is internally connected to $\overline{DCD}$, and $\overline{RTS}$ is internally connected to $\overline{CTS}$.
3. $\overline{TxC}$ is internally connected to $\overline{RxC}$.
4. The DSR is internally held low (inactive).
5. The TxD, $\overline{DTR}$ and $\overline{RTS}$ outputs are held high.
6. The $\overline{CTS}$, $\overline{DCD}$, $\overline{DSR}$ and $\overline{RxD}$ inputs are ignored.
7. Except as noted, all other Status, Mode and Command Register bits and interrupts retain their functions and settings.



TL/C/5593–35

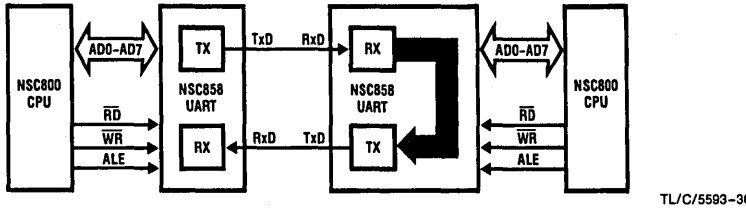
FIGURE 4. Local Loopback

Remote Loopback Mode (see *Figure 5*)

1. The contents of the Receiver Holding Register, when RxRDY = 1 indicates it is full, are transferred to the Transmitter Holding register, when TxBE = 1 indicates it is empty. After this action, both RxRDY and TxBE are cleared.
2. $\overline{RxC}$ is connected internally to $\overline{TxC}$.
3. Setting the Remote Loopback Mode places all receiver and transmitter flags under control of the remote loopback sequencer. RxRDY and TxBE can be monitored to follow automatic remote loopback data flow, while OE and TxU can indicate system problems.
4. The CPU can read the Receiver Holding register if desired, but this is not necessary. The CPU cannot load the Transmitter Holding Register.
5. Modem Status, all Mode and Command register bits retain their functions and interrupts are generated.

Under certain conditions entering the remote loopback mode causes a character in the receiver or transmitter holding registers to be sent, even though, the transmitter is disabled.

1. If the UART enters the remote loopback mode immediately after receiving a break character in the normal receive mode, it will then automatically transmit that character.
2. If the UART enters the remote loopback mode before the CPU has read the latest character in the receiver holding register, it will then automatically transmit that character.
3. If the UART enters the remote loopback mode before the last character written to the transmitter holding register is transmitted, then it will automatically transmit this character.

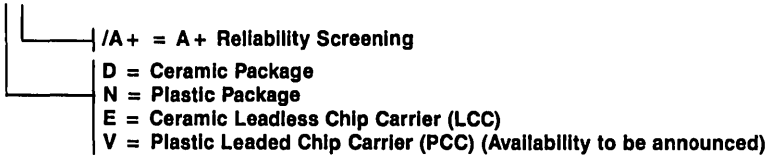


TL/C/5593–36

FIGURE 5. Remote Loopback

11.0 Ordering Information

NSC858XX



TL/C/5593–37

12.0 Reliability Information

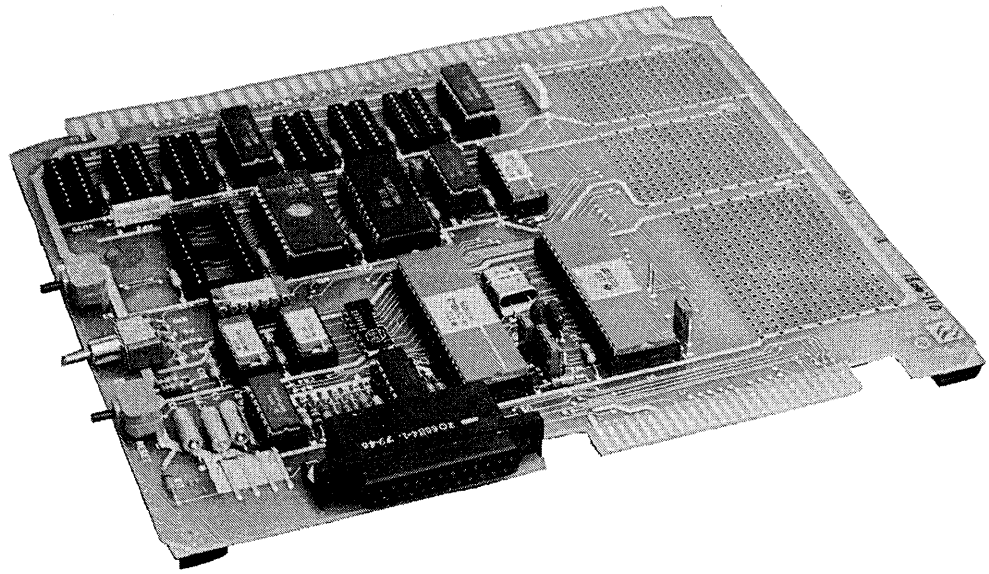
Gate Count 4280

Transistor Count 8450

National Semiconductor

microCMOS

NSC888
NSC800™ Evaluation Board



TL/C/8533–1

- **NSC800 8-Bit microCMOS CPU**
- **Executes Z80® Instruction Set**
- **20 programmable parallel I/O lines**
- **Two 16-Bit programmable counters/timers**
- **Powerful 2k x 8 monitor program**
- **Five levels of vectored prioritized interrupts**
- **RS232 Interface**
- **1k x 8 microCMOS RAM with sockets for up to 4k x 8 RAM**
- **Socket for additional 2k x 8, 2716 compatible memory component**
- **Wire wrap area**
- **Edge connectors for system expansion**
- **Single-step operation mode**
- **Fully assembled and tested**

Product Overview

The NSC888 is a self-contained microprocessor board which enables the user to quickly evaluate the performance and features of the NSC800 product family. This fully assembled, tested board requires only the addition of a ±5V supply and an RS232 interface cable to the user's terminal to begin NSC800 evaluation.

A powerful system monitor is provided on the board which controls serial communications via the RS232 port. The monitor also includes command functions to load, execute and debug NSC800 programs.

The board includes an NSC800 CPU plus RAM, EPROM, I/O, Timers and interface components yet draws only 30 mA from the +5V supply and 3 mA from the −5V supply.

Although designed primarily as an assessment vehicle, the NSC888 can be readily programmed and adapted to a variety of uses. Wire wrap area is provided on-board for the user to build up additional circuitry or interfaces, thus tailoring this high-performance, low-power microprocessor board to meet individual needs.

Functional Description

Figure 1 and *Figure 2* provide information on the organization of the NSC888 board. Please refer to these figures for the following discussion.

Central Processor

The powerful NSC800 is the central processor for the NSC888. It provides bus control, clock generation and extensive interrupt capability. Featuring a multiplicity of programmable registers and sophisticated addressing modes, the NSC800 executes the Z80 instruction set.

Memory

- 128 bytes of RAM are provided by the NC810A RAM-I/O-Timer and are used by the monitor program for the system stack.
- 1024 bytes of RAM are provided by two 1k x 4 NMC6514's. Sockets are provided for six additional NMC6514's, for a total of 4k bytes of RAM.
- A 2k byte EPROM system monitor is provided on-board which includes facilities to load, execute and debug a users program.
- An additional EPROM socket is also on-board which accepts a 2k byte 2716 compatible memory component.

Input/Output

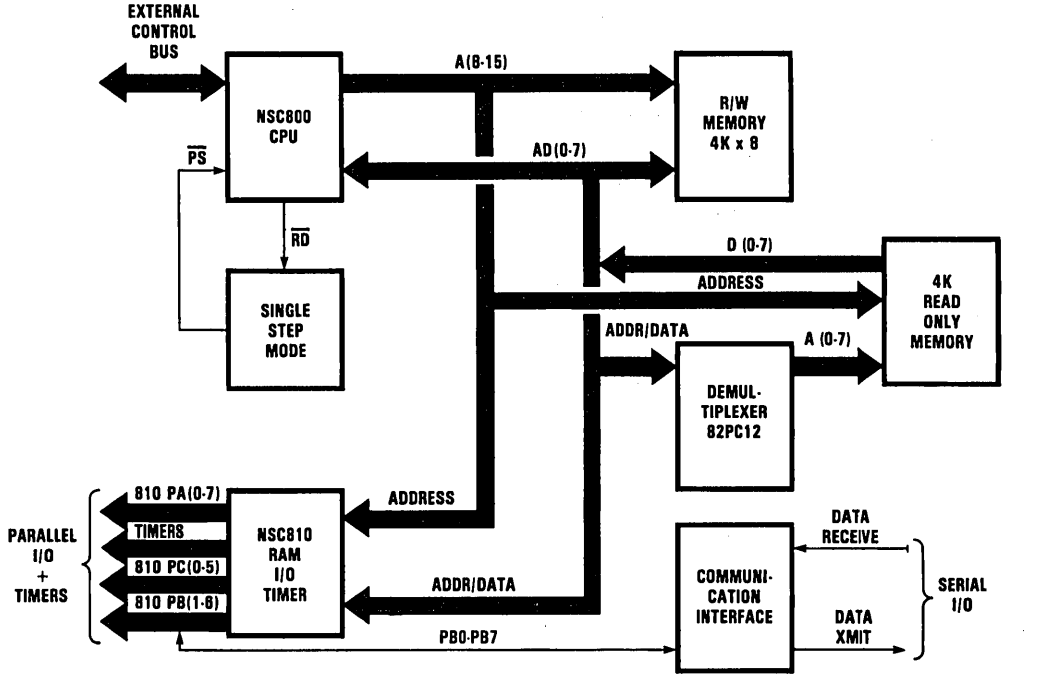
- **Parallel I/O**

  The NSC888 provides 20 programmable parallel I/O lines implemented using the I/O ports of the NSC810A RAM-I/O-Timer. The port bits may be individually defined as input or output, and can also be written to or read from in bytes. The I/O lines are conveniently brought to a 50 contact edge connector for user interface.

- **Serial I/O**

  An RS232 connector and accompanying support circuitry are provided on-board. Two I/O lines from the NSC810A RAM-I/O-Timer are used for the serial communications function, which is controlled exclusively by software. The baud rate is determined upon system initialization by the character bit rate from the users terminal. The maximum baud rate is 2400 baud.

Block Diagram



TL/C/8533-2

FIGURE 1

Functional Description (Continued)

Timers

The NSC888 provides two fully programmable binary 16-bit counters/timers utilizing the NSC810A RAM-I/O-Timer. These signals are also brought to the parallel I/O connector. Each timer may operate in any of six different modes:

- Event Counter
- Accumulative Timer
- Restartable Timer
- One Shot
- Square Wave
- Pulse Generator

Connectors

- **Parallel I/O**

  The parallel I/O lines and timer lines from the NSC810A RAM-I/O-Timer, plus interrupt lines from the CPU are brought to this 50 contact edge connector.

- **System Bus**

  All NSC800 CPU lines except XIN are brought to this 86 contact edge connector. In addition, the −5V line is also brought to the system bus connector.

- **RS232**

  This connector is provided for system interface to the users terminal.

Interrupts

The NSC888 utilizes the powerful interrupt processing capability of the NSC800 CPU. Interrupts are routed via a jumper matrix to the five interrupt inputs of the NSC800. Each input, which may be from the NSC810A I/O ports, NSC810A timers or off board via the system bus connector, generates a unique memory address (see Table I). All interrupts with the exception of NMI can be masked via software. Interrupt lines are also brought to the parallel I/O connector.

TABLE I.

Interrupt Input	Memory Address	Type	Priority
NMI	0066H	Non-maskable	Highest
RSTA	003CH	Maskable	
RSTB	0034H	Maskable	
RSTC	002CH	Maskable	
INTR	0038H*	Maskable	Lowest

*mode 1

NSC888 Firmware

The NSC888 system monitor is provided by a preprogrammed EPROM. This comprehensive monitor includes facilities to load, execute and debug programs. The monitor allows the user to examine and modify any RAM memory location or CPU register. It permits the insertion of break points to facilitate debugging. Programs can be executed starting at any location.

The commands supported by the NSC888 system monitor are as follows:

- B - Select a new baud rate
- D - Display memory
- F - Fill memory between ranges
- G - Execute program with break points
- H - Hexadecimal math routine
- J - Non-destructive memory test
- K - Store 16-bit value in memory
- M - Move a block of data
- P - Put ASCII characters in memory
- Q - Query I/O ports
- S - Substitute and/or examine memory
- T - Type memory contents in ASCII
- V - Verify two blocks of data
- X - Examine or modify CPU registers
- Y - Memory search for string

These commands are fully explained in the NSC888 Hardware/Software Users Manual.

Single Step/Power Save

The NSC888 provides a unique single-step mode, utilizing the Power Save input of the NSC800 CPU. This input, when activated, reduces CPU power consumption from 50 mW to only 25 mW. It also allows the user to single-step through a program, checking and modifying code. This function is controlled via a switch on the board.

Specifications

Microprocessor

CPU—	NSC800
Data Word—	8 bits
Instruction Word—	8, 16, 24, 32 bits
Cycle Time—	2.00 μs (minimum instruction time)
System Clock—	2.00 MHz
Registers—	14 general purpose (8-bit) 2 index registers (16-bit) 1 stack pointer (16-bit) 1 program counter (16-bit)
Number of Instructions—	158
Address Capability—	64k bytes

Memory

RAM—	1152 bytes on-board plus sockets for an additional 3k bytes
ROM/EPROM—	Sockets for 4k bytes on-board
Access Time—	625 ns for opcode fetch 875 ns for memory read

Specifications (Continued)

Connectors

System Bus	86-pin double-sided card cage edge connector on 0.156 inch centers
Parallel I/O	50-pin double-sided edge connector on 0.1 inch centers Recommended mating connector: 3M 3415-0001 AMP 2-86792-3
Serial I/O	Standard RS232 connector
Power	+5V 30 mA (27C16 EPROM monitor) or 90 mA (2716 EPROM monitor) −5V 3 mA

Physical

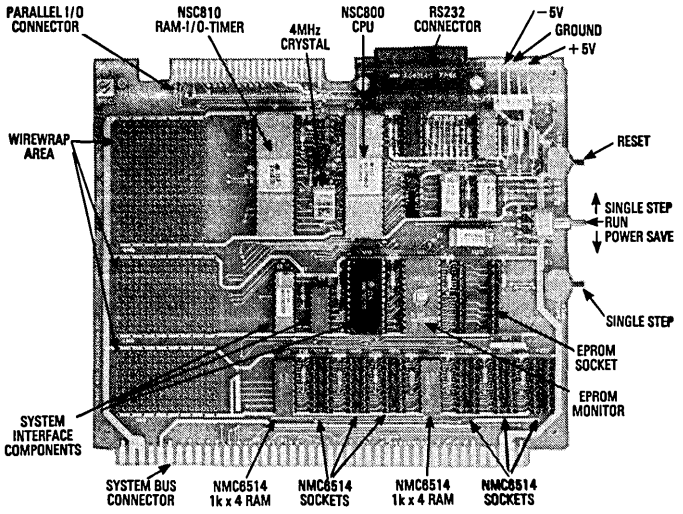
Height	6.75 (17.15 cm)
Width	7.85 (19.94 cm)

Order Information

NSC888	Includes CPU, 1152 bytes of RAM, sockets for additional 3k bytes of RAM, 2k byte monitor with additional socket for 2k byte ROM/EPROM, 20 I/O lines, RS232 interface, wire wrap area.

Documentation

The NSC888 Hardware/Software Users Manual and NSC800 Microprocessor Family Handbook are shipped with the NSC888 Evaluation Board



TL/C/8533-3

FIGURE 2. NSC888 Evaluation Board

Comparison Study NSC800 vs. 8085/80C85 Z80®/Z80 CMOS

Introduction

The NSC800 is an 8-bit parallel processor with a Z80 compatible instruction set manufactured using National's micro-CMOS process. This process combines the speed of silicon gate NMOS with the low power inherent to CMOS.

The NSC800 has a 16-bit address bus which consists of the upper eight address bits (A8–A15) and the lower eight address bits (AD0–AD7). Address bits A0–A7 are time multiplexed on the 8-bit bidirectional address/data bus (AD0–AD7).

There are several advantages to using a multiplexed address/data bus. Multiplexing frees pins on the CPU and peripheral packages for other purposes, such as status outputs, DMA control lines, and multiple interrupts. This can reduce system component count. Fewer bus signal lines are required for device interconnections in most applications (16 lines for multiplexed bus systems vs. 24 lines for non-multiplexed systems). This reduces PC board complexity.

Peripherals of the NSC800 Family include:

NSC810A RAM I/O Timer

NSC831 I/O

NSC858 UART

In addition to the above parts, a complete family of low power speed compatible logic and interface parts is also available.

NSC800 vs. 8085

In terms of bus structure, the NSC800 is similar to the 8085. Both processors utilize a multiplexed bus and timing relationships are approximately the same. The 8085 does not guarantee that output data on AD0–AD7 are valid on both the leading and trailing edges of $\overline{WR}$. For the NSC800, data are valid on both the leading and trailing edges of $\overline{WR}$.

Both the NSC800 and the 8085 use ALE, S0, S1, and IO/$\overline{M}$ to indicate status. The lower eight address bits are guaranteed to be valid on the data bus at the trailing edge (high to low transition) of ALE (Address Latch Enable). This signal is used by the external system components to separate the address and data buses. When the only components utilized in the system are members of the NSC800 family (which contain on-chip demultiplexers), ALE needs only to be connected to the enable inputs. If non-NSC800 family components are used, ALE can be used to enable an 8-bit latch to perform the function of bus separation.

Decoding status bits S0 and S1, in conjunction with IO/$\overline{M}$, notifies the external system of the type of the ensuing M cycle. TABLE I shows a truth table of the encoded information. During a halt status the NSC800 will continue to refresh dynamic RAM.

TABLE I.
Machine Cycle Status - NSC800 and 8085

S0	S1	IO/M	Status
1	0	0	Memory Write
0	1	0	Memory Read
1	0	1	I/O Write
0	1	1	I/O Read
1	1	0	Opcode Fetch
0	1	0	Bus Idle*
0	0	0	Halt

*ALE not suppressed during Bus Idle

Direct Memory Access (DMA) control signals $\overline{BREQ}$ and $\overline{BACK}$ of the NSC800 perform the same functions as HOLD and HLDA on the 8085. The NSC800 allows simple wire ORing by using active low states for the DMA control signals. An active low on the $\overline{BREQ}$ (Bus Request) line, tested during the last T state of the current M cycle, initiates a DMA condition. The NSC800 will then respond with an active low $\overline{BACK}$ (Bus Acknowledge) signal causing the address, data and control buses (TRI-STATE® circuits) to go to the high impedance state, and notifies the interrupting device that the system bus is available for use. There is a difference in the timing relationship between these functions for the two processors. The 8085 responds with HLDA, one-half T state after it recognizes HOLD. The NSC800 responds with $\overline{BACK}$, one T state after it recognizes $\overline{BREQ}$.

During Input/Output cycles for peripherals, the NSC800 automatically inserts one wait state. This reduces the external hardware required for slow peripherals. The 8085 does not insert its own wait state during these I/O cycles. When they are needed, the 8085 user must design his system to contain the additional hardware required to do the wait state insertion. When more than one wait state is required, additional wait states can be added to the I/O cycles in a similar way on both the NSC800 and the 8085. On the NSC800, this is accomplished by bringing the $\overline{WAIT}$ control signal active low during T2 of an I/O or memory cycle. The 8085 is controlled in the same way through the use of the READY line.

The NSC800 instruction set is Z80 compatible and more powerful than the 8085's. The NSC800 does not support the RIM and SIM instructions of the 8085 (RIM and SIM can be emulated with I/O instructions), but has an improved instruction set for enhanced system performance. The NSC800 has two functions, $\overline{RFSH}$ and PS, instead of the two serial I/O lines SOD and SID. $\overline{RFSH}$ (Refresh) is a status signal which indicates that an eight bit refresh address is present on the address/data bus (AD0–AD7). The refresh address occurs during T3 of each M1 (opcode fetch) cycle. The internal refresh counter is incremented after

each instruction cycle. This counter output can be employed by the user's dynamic RAM refresh circuits. The $\overline{PS}$ (Power Save) control input, when active, causes the CPU to stop all internal clocks at the end of the current instruction, which reduces power consumption. The on-chip oscillator and CLK remain active for any required external timing. The NSC800 leaves all buses unchanged during this time, which has the effect of reducing power consumption on other CMOS parts in the system since the buses are not changing states. All internal registers and status conditions are maintained, and when $\overline{PS}$ subsequently goes high, the opcode fetch cycle begins in a normal fashion.

TABLE II indicates the major differences between the NSC800 and the 8085 presented in tabular form for quick reference.

TABLE II.
NSC800 vs. 8085/80C85 Comparison

Item	NSC800	8085	80C85
Power Consumption	50 mW @ 5V	850 mW @ 5V	50 mW @ 5V
Bus Drive Capacity	1 std. TTL (100 pF)	1 std. TTL (100 pF)	1 std. TTL (150 pF)
Dynamic RAM Refresh Counter	Yes, 8-bit	No	No
Automatic WAIT State on I/O	Yes	No	No
Number of instruction types	158	80	80
Number of Programmer Accessible Registers	22	10	10
Block I/O and Search	Yes	No	No

NSC800 vs. Z80/Z80 CMOS

The NSC800 contains the same complement of internal registers as the Z80 and maintains instruction set and opcode compatibility.

Machine cycle timing for the standard speed version of the NSC800 compares directly with the Z80. Although the software execution speeds are comparable, the NSC800 offers architectural advantages.

The bus structures of the NSC800 and the Z80 are quite different. The NSC800 uses a multiplexed address/data bus. The Z80 has separate address and data buses. As stated earlier, the separate bus structure requires additional signal lines for interconnection and gives up some package pins which could be used for other purposes.

The main differences between the NSC800 and the Z80, in addition to the bus structures, are the refresh counter, on-chip clock generation, and the interrupt capability.

1. The NSC800 contains an 8-bit refresh counter as opposed to a 7-bit refresh counter in the Z80. (This enables refresh of a 64K dynamic RAM system memory). The refresh timing of the NSC800 is functionally identical to that of the Z80.
2. The on-chip clock generation reduces the system component count. In place of an external clock generator chip, the NSC800 needs only a crystal or RC circuit to produce the system clock.
3. The NSC800 provides three interrupts that are not available on the Z80: $\overline{RSTA}$, $\overline{RSTB}$, $\overline{RSTC}$. This gives the NSC800 five levels of vectored, prioritized interrupts with no external logic. The general purpose interrupt ($\overline{INTR}$) and Non-maskable Interrupt ($\overline{NMI}$) are identical to the Z80. $\overline{INTR}$ has the same three modes of operation in both processors: Modes 0, 1, and 2. Upon initialization, the NSC800 is in mode 0 to maintain 8080 code compatibility. $\overline{NMI}$, when active, causes a restart to location X'66 as is the case with the Z80. Being a non-maskable interrupt, $\overline{NMI}$ cannot be disabled. The additional interrupts $\overline{RSTA}$, $\overline{RSTB}$, and $\overline{RSTC}$ cause restarts to locations X'3C, X'34, and X'2C respectively. The priority levels of the five interrupts are: $\overline{NMI}$ (highest), $\overline{RSTA}$, $\overline{RSTB}$, $\overline{RSTC}$, and $\overline{INTR}$ (lowest). For the NSC800, Interrupt acknowledge ($\overline{INTA}$) is provided on a dedicated output pin and need not be decoded externally, as is the case with the Z80. With the status outputs (S0, S1, IO/$\overline{M}$), early read/write information is obtainable. This is impossible to derive from the Z80.

   Refer to TABLE III for comparison of the major differences between the NSC800 and the Z80.

TABLE III.
NSC800 vs. Z80/Z80 CMOS Comparison

Item	NSC800	Z80	Z80 CMOS
Power Consumption	50 mW @ 5V	750 mW @ 5V	75 mW @ 5V
Instruction Execution (Minimum)	1 μs	1 μs	1 μs
On-Chip Clock Generator	Yes	No	No
Number of On-Chip Vectored Interrupts	5	2	2
Early Read/Write Status	Yes	No	No
Dynamic RAM Refresh Counter	Yes, 8-bit	Yes, 7-bit	Yes, 7-bit

NSC800 Family Devices (microCMOS)

MM82PC08 8-Bit Bidirectional Transceiver

MM82PC12 Input/Output Port

Note: The above devices are pin for pin and function compatible with the standard TTL, CMOS or NMOS versions currently available.

SUMMARY

National's NSC800 has a Z80 compatible instruction set, which is more powerful than the 8085. NSC800 external hardware requirements are less because of on-chip automatic wait state insertion, clock generation and five levels of vectored prioritized interrupts.

The 8085 and the NSC800 have similar bus structures, and timing. The key advantages of the NSC800 over the 8085 are the larger instruction set, more registers accessible to programmers, low power consumption, and a dynamic RAM refresh counter.

The main advantages of the NSC800 compared to the Z80 are the multiplexed address/data bus, an 8-bit refresh counter for dynamic RAMs, on-chip clock generation, and five interrupts. The speed of the NSC800 and Z80 is the same but, the NSC800 has very low power consumption.

Software Comparison NSC800 vs. 8085, Z80®

Introduction

The NSC800 is an 8-bit parallel microprocessor fabricated using National's microCMOS process. This process allows fabrication of a microprocessor family that has the performance of silicon gate NMOS along with the low power inherent to CMOS. The NSC800 instruction set is a superset of the 8080's instruction set. It comprises over 900 operation codes falling into 158 instruction types. The instruction categories are:

- Load and Exchange
- Arithmetic and Logic
- Rotate and Shift
- Jump and Call
- Input/Output
- Bit manipulation (set, test, reset)
- Block Transfer and Search
- CPU control

The load instructions allow the movement of data into and out of the CPU, between internal registers, plus the capability to load immediate data into internal registers. The exchange instructions allow swapping of data between two registers.

The arithmetic and logic instructions operate on the data in the accumulator (primary working register) and in the other registers. Status flags are set or reset depending on the result of the particular operation executed. This group includes 8-bit and 16-bit operations.

The rotate and shift instructions allow any register or memory location to be rotated or shifted, left or right, with or without carry. These can be either an arithmetic or logic type.

The jump and call group includes several different types: one byte calls, two byte relative jumps, conditional branching, and three byte calls and jumps, which can reach any location in memory. Calls push the current contents of the Program Counter onto the stack before branching to the new program address to facilitate subroutine execution.

Input/Output instructions allow communications between the NSC800 and external peripheral devices. There are 255 (location X'BB is used for an interrupt mask) unique peripheral I/O locations available to the NSC800. I/O instructions can move data between any memory location or internal register and any I/O location. There are also block I/O instructions which allow moving data blocks of up to 256 bytes directly from memory to any peripheral location or from any peripheral location to a block of memory.

Bit manipulation instructions can set, test or reset any bit in the accumulator, any general purpose register or any memory location.

The block transfer instructions allow a single instruction to move any size block of memory to any other location in memory. Through the use of the block search instructions, any size block of memory can be searched for a particular byte of data.

Finally, the CPU control group allows user control over the various modes of CPU operation, such as enabling and disabling interrupts or setting modes of interrupt response.

The following sections will compare the instruction set of the NSC800 with those of the 8085 and the Z80.

NSC800 vs. 8085

The 8085 instruction set consists of 246 op codes falling into 80 instruction types. With the exception of RIM and SIM, the NSC800 is instruction and op code compatible with the 8085. The RIM and SIM instructions are not supported because the NSC800 does not have the SID and SOD serial I/O lines. The interrupt mask on the NSC800 is accessible by writing the mask word to I/O location X'BB. The bit positions for the interrupt enables are shown below:

Location X'BB Bit Assignments

Bit	Interrupt Enable for
7	N/A
6	N/A
5	N/A
4	N/A
3	$\overline{\text{RSTA}}$
2	$\overline{\text{RSTB}}$
1	$\overline{\text{RSTC}}$
0	$\overline{\text{INTR}}$

N/A = not used: a don't care bit.

As an example, to enable interrupts on the $\overline{\text{RSTA}}$ input, a logic '1' is written into bit 3 of I/O location X'BB. If the master interrupt enable has been set by executing the Enable Interrupt (EI) instruction, interrupts will now be accepted on $\overline{\text{RSTA}}$ only.

Other than the method of enabling and disabling individual interrupts and the RIM and SIM instructions themselves, the NSC800 instruction set is a superset of the 8085's instruction set.

The following benchmark demonstrates the code reduction and throughtput improvement obtained by using one of the special NSC800 instructions over the same function implemented with the limited 8085 instruction set. The function is to move a 512-byte block of data from one section of memory to another.

8085

Bytes		Mnemonics		Cycles
3		LXI	H,SOURCE	10
3		LXI	D,DEST	10
3		LXI	B,COUNT	10
1	LOOP:	MOV	A,M	7
1		STAX	D	7
1		INX	H	6
1		INX	D	6
1		DCX	B	6
1		MOV	A,C	4
1		ORA	B	4
3		JNZ	LOOP	10
Total: 19				Total: 80

NSC800

Bytes	Mnemonics		Cycles
3	LD	HL,SOURCE	10
3	LD	DE,DEST	10
3	LD	BC,COUNT	10
2	LDIR		21
Total: 11			Total: 51

The use of the LDIR instruction of the NSC800 results in a 47.5% increase in throughput and a 42% decrease in the number of bytes required to implement the function when compared with the 8085 implementation. The time required to make the move is approximately 2.69 ms for the NSC800 and approximately 5.12 ms for the 8085. Note that even though the 8085 runs at a faster cycle time (200 ns vs. 250 ns), the improved instruction set of the NSC800 produces an increase in system performance.

The NSC800 includes all 8085 flags plus some additional flags. The flag formats for the NSC800 and 8085 are:

NSC800 Flags (Z80 Flags)

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

8085 Flags

7	6	5	4	3	2	1	0
S	Z	X	AC	X	P	X	CY

The differences between the flag registers on the NSC800 and the 8085 are identified below:

1. Bit position D1 (additional on the NSC800) contains an add/subtract flag that is used internally for proper operation of BCD instructions.

2. In the NSC800, the P/V flag will not match the 8085's P flag after an 8-bit arithmetic operation, since it acts as an overflow bit for the NSC800, but acts as a parity bit for these operations in the 8085.

3. Bit position D2 (changed for the NSC800) is a dual purpose flag; it indicates the parity of the result in the accumulator when logical operations are performed and also represents overflow when signed two's complement arithmetic operations are performed. An overflow occurs when the result of a two's complement operation within the accumulator is out of range.

4. For general Compare operations, the NSC800 uses the P/V flag as an overflow bit, while the 8085 uses the P flag for parity.

5. The H flag (bit position D4) on the NSC800 is functionally the same as the auxiliary carry on the 8085.

6. For Double Precision Addition, the NSC800 leaves the H flag undefined, while the 8085 does not affect the AC flag for this operation (DAD).

7. For Rotate operations, the NSC800 resets the H flag, while the 8085 leaves the AC flag unaffected for these operations.

8. When Complementing the Accumulator, the NSC800 sets the H flag (H = 1), while the 8085 leaves the AC flag unaffected.

9. When Complementing Carry, the NSC800 leaves the H flag undefined, while the 8085 leaves the AC flag unaffected.

10. When Setting the Carry, the NSC800 clears the H flag (H = 0), while the 8085 leaves the AC flag unaffected.

NSC800 vs. Z80

The instruction set and op codes of the NSC800 are identical to those of the Z80. Software written for the Z80 will run on the NSC800 without change, unless I/O location X'BB is used. Another location should be assigned since location X'BB is an on-chip write-only register used for the interrupt mask. Since the NSC800 executes code at the same cycle time as the Z80, any software timing loops will also remain the same, and no change is necessary. The NSC800 expanded interrupt capability is transparent to the user unless specifically evoked by the user software.

The NSC800 has 8-bit refresh rather than the 7-bit refresh scheme of the Z80. Therefore, the state of the 8th bit will be indeterminate since it is part of the R Register and so included in refresh operations.

The status flags on the NSC800 are identical to those on the Z80. There is no difference between the positions of the individual bits in the flag register, nor in the manner in which the flags are set or reset due to an arithmetic or logical operation. Testing of the flags is also the same.

NSC800™ Applications System: ROM Monitor and System Board

National Semiconductor
Application Note 612
Louis W. Shay

ABSTRACT

This document describes a NSC800-based microcomputer system and ROM monitor software that can be tailored to fit a variety of applications, and can be used with the IBM PC and several off-the-shelf NSC800 development products currently available for fast software development. Included are system schematics, a system user's manual and a list of some vendors of NSC800/Z80 development products. Additional documentation and program listings are available through National's Dial-A-Helper on-line information system. (See Appendix F).

SECTION 1.0 OVERVIEW

1.1 Introduction

The NSC800 Applications System is a general-purpose, 8-bit microcomputer and ROM monitor program, MON800, that can easily be configured and reconfigured to fit a wide variety of applications. The main purpose of the NSC800 Applications System is to allow the system designer to quickly and efficiently develop application programs using the IBM PC and available off-the-shelf development tools for the NSC800 microprocessor.

The MON800 monitor program allows the user to perform such tasks as program downloads, program editing, program execution, defining breakpoints, register manipulation, and on-line assembly. Also included are several handy monitor I/O calls and math routines that can be called from user programs. MON800 is a powerful debugging tool that allows the programmer to develop NSC800 code on any PC or other host system, download the code to the NSC800 system, and debug it using MON800's command set.

The NSC800 Applications System can run stand-alone, and needs only the addition of a suitable power supply and RS-232 compatible terminal to operate. The design of the system is extremely simple, and the parts count is low. The NSC800 series peripheral devices provide the system with such functions as parallel I/O, RAM, programmable timers, and serial I/O. Most of the system signals are accessible to the user by means of wire-wrap jumper blocks. Thus, addition of other types of devices is simple. With these headers, the system interrupts, timers, and ports can be configured and reconfigured to fit various applications. The core design, however, remains consistent, and it can serve as a core design for more specific applications. The NSC800 Applications System provides a powerful solution for a multitude of educational, industrial, and communications needs.

1.2 Features

- Fully compatible with the Z80 instruction set and architecture
  - — 158 instructions
  - — 22 internal registers
  - — 10 addressing modes
- Fabricated in National's microCMOS technology. NSC800 family devices have a very low power consumption. The NSC800 also has a unique power-save feature.
- Multiplexed bus structure
- Five prioritized interrupts on-chip, with support for addition of off-chip interrupt control circuitry
- Operation at speeds from 150 kHz to 4 MHz
- Six programmable, parallel I/O ports (up to 44 lines)
- Four 16-bit, programmable timers, each having six possible modes of operation
- Two programmable Universal Asynchronous Receiver/Transmitters (UARTs) for serial I/O, with RS232 standard CMOS line drivers
- 8 kbytes of static RAM for user programs, expandable to 40 kbytes
- ROM monitor program, MON800
  - — 18 commands including a file downloader, memory manipulation, program execution with up to five breakpoints, CPU register manipulation, and more
  - — Monitor service routines available to user programs that perform various I/O and math functions
  - — Source code is provided. The monitor may be modified to fit specific applications.

1.3 Setup and Operation

Once the NSC800 Applications System has been assembled using the schematics provided, it requires only the addition of a suitable power supply and VT100-type terminal to operate. The system runs stand-alone, and needs no other host computer or software to operate. However, it is possible to use a PC as the terminal device by means of the PC's serial port and a VT100 terminal emulation program such as KERMIT. In this way the PC can be used to assemble and link programs, download them to the NSC800 target system, then run the program under MON800's control. The terminal attaches to connector J1, which is the main serial I/O port. This connector is RS-232 and DCE configured. The baud rate is controlled by the setting of the DIP switch on the system board. Switch settings are listed in Appendix B.

Power supply requirements are as follows:

- +5.0V ±5%
- +12.0V ±10%
- −12.0V ±10%

Power is applied to connector J6. Connector pin assignments are listed in Appendix A.

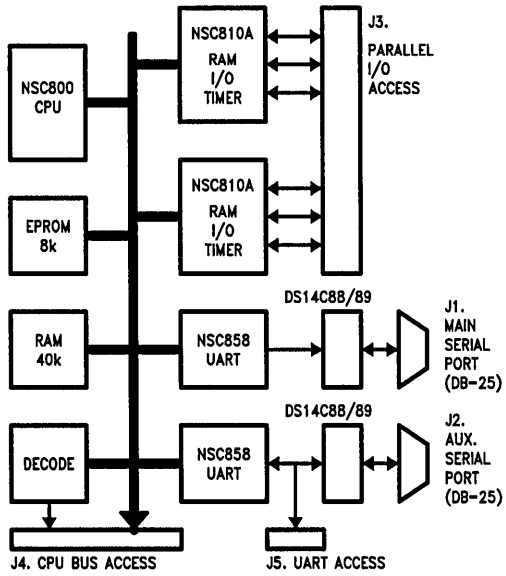
On power-up, the monitor will output a sign-on message to the terminal on the main I/O port and prompt for a command. If the message does not appear, then verify the circuit connections, power supply, switch settings, and terminal setup and power the board on again.

1.4 Document Organization

The remainder of this document describes the hardware and software of the NSC800 Applications System. Section 2 describes the system hardware and architecture. Section 3 describes the MON800 program operation and command set. Appendices A, B and C detail system connector pinouts, and switch AND jumper configuration options. Appendix D describes the Intel Hex file format used by MON800 when downloading files from a remote host. Appendix E is an example for using the MS-KERMIT program (Columbia University) to interface the NSC800 Applications System to the IBM PC/AT. Appendix F is a MON800 program listing. Appendix G contains system schematics, suggested board layout, and a parts list. Appendix H is a list of vendors that offer support products for the NSC800.

SECTION 2. HARDWARE DESCRIPTION

Figure 1 is a block diagram of the NSC800 Applications System. Following are descriptions of each element in the system. For detailed descriptions of NSC800 series components, refer to the **Series 32000 Microprocessors Databook** (1987).



TL/C/10435–1

FIGURE 1. NSC800 Applications System Block Diagram

2.1 CPU

The NSC800 microprocessor is the heart of the NSC800 Applications System. The NSC800 is completely code-compatible with the Z80 microprocessor, and will run programs written for the Z80. The external hardware is different, however. The NSC800 uses a multiplexed address/data bus. There are five prioritized hardware interrupts, including one non-maskable interrupt, and one that supports the addition of an off-chip interrupt controller circuit. All necessary bus timing and control signals are also generated on-chip, including DMA support and DRAM refresh functions. The CPU can address 64 kbytes of memory and 256 I/O ports. Operating speeds can be as low as 150 kHz for low-power applications, or as fast as 4 MHz. The NSC800 also has a unique power-save feature, that allows a remote source to place the NSC800 in a minimum power state, or "sleep mode".

2.2 RAM-I/O-Timers

The NSC810A RAM-I/O-Timer modules integrate several system functions onto one chip. The chip features three programmable parallel I/O ports. The port lines are bi-directional and individually controlled. One of the ports supports strobed-mode I/O, interrupt-mode operation, and can be placed in TRI-STATE® mode.

The chip also contains two 16-bit programmable timers. Each timer has six modes of operation, including pulsed output, square wave output, and gated modes that allow the timers to be started and stopped via an external signal.

128 bytes of RAM are on-chip for storage of system data. The RAM can retain data at voltages below 2V. The RAM is suitable for battery-backed storage of critical system data.

2.3 Serial I/O

Serial I/O is implemented with the NSC858 UART. These devices feature programmable baud rates to 256k baud, independent transmitter and receiver functions, modem control functions, polled or interrupt-mode operation, and software or hardware power-down options.

The E.I.A. line drivers used for the RS232 interface are National's DS14C88 and DS14C89 CMOS line driver and receiver chips. For more information on these devices, refer to National's **Interface Databook** (1986).

There are two serial ports in the system. One port is dedicated to communication with the host terminal. This is the main serial port. It is permanently configured as a DCE port. The second UART is the auxiliary port, and can be used for target applications.

2.4 Switches and Jumper Options

There are two push switches, S1 and S2, that can be used to generate a hardware reset or non-maskable interrupt, respectively. The 8-position DIP switch on the board is read at reset time, and that value controls the main serial channel baud rate, data bits, etc. The DIP switch can also be read by user programs.

Pin jumpers are provided to allow configuration of hardware signals. W1 and W2 select external input or switch options for RESET and NMI signals. W3 and W4 control serial I/O port options. Jumper settings are listed in Appendix C.

2.5 Hardware Interface

There are six connectors on the NSC800 Applications System board. J1 and J2 are the main and auxiliary serial port connectors. These connectors are standard DB-25 connectors and the pinouts are as per the RS232 standard. J3 is the parallel port header. J4 is the CPU bus access header, from which most of the system signals such as address, data, and control signals can be accessed. J5 is the UART access header. Auxiliary port I/O signals are accessed here. J6 is the power connector. All connector pin functions are listed in Appendix A.

Configuring the system is very simple because of the wire-wrap headers J3, J4, and J5. For example, if the designer wishes to use the CLK signal from the NSC800 to drive Timer #1 of one of the NSC810A chips, all that needs to be done is to run a wire from the CLK pin on J4 to the T1IN line of J3. Now, if the designer wants to use the timer output to drive the $\overline{\text{RSTA}}$ line of the NSC800, just connect the two signals together using another wire. To reconfigure the system to fit another application, remove the wires and start over.

2.6 System Architecture

Tables 2.1 and 2.2 list the memory and I/O space configuration for the NSC800 Applications System. MON800 uses 256 bytes of RAM in address range FF00H to FFFFH for storage of monitor data. Detailed usage of this reserved space is described in Section 3.

TABLE 2.1. Memory Configuration

Hex Address	Function
0000–1FFF	MON800 EPROM (NMC27C64 8k x 8)
2000–207F	NSC810A #1 RAM (128 Bytes)
2080–3FFF	Invalid—Do Not Use
4000–407F	NSC810A #2 RAM (128 Bytes)
4080–5FFF	Invalid—Do Not Use
6000–7FFF	User RAM #5 (8k)—Decode Provided
8000–9FFF	User RAM #4 (8k)—Decode Provided
A000–BFFF	User RAM #3 (8k)—Decode Provided
C000–DFFF	User RAM #2 (8k)—Decode Provided
E000–FFEF	User RAM #1
FF00–FFFF	Reserved for MON800 Use

TABLE 2.1. I/O Configuration

Hex Address	Function
00–1F	8-Bit DIP Switch Input Latch
20–3F	NSC810A #1 Ports & Timers
40–5F	NSC810A #1 Ports & Timers
60–7F	User I/O—Decode Provided
80–9F	User I/O—Decode Provided
A0–BF	User I/O—Decode Provided ** Do Not Use I/O Address BBh **
C0–CF	NSC858 Auxiliary Serial Port
D0–DF	Invalid—Do Not Use
E0–EF	NSC858 Main Serial Port
F0–FF	Invalid—Do Not Use

SECTION 3. MON800 MONITOR

3.1 Overview: MON800

The primary purpose of the NSC800 Applications System and MON800 firmware is to provide the system designer with an efficient and cost-effective way to develop applications using the NSC800 series devices. Keeping this in mind, the system was designed to be used with a variety of hardware and software packages currently on the shelf. The IBM PC can function as a complete software development station using available software. Assembled and linked code can be downloaded from the PC to the NSC800 system in Intel Hex file format using DOS commands and the PC's serial port. Other software can be used to emulate a VT100 terminal through the PC's serial port so that commands can be issued to the monitor from the PC. One such package is the KERMIT shareware program. An example for interfacing the NSC800 Applications System to the IBM PC is given in Appendix E.

The rest of this section describes the organization and operation of the MON800 program, system interrupts, command set, service calls, and examples for each.

MON800's command set is very useful for debugging user programs. A command summary is given below.

1. Memory manipulation commands
   - — Assemble an NSC800 instruction and place it in memory
   - — Examine/Change a single location, or several
   - — Display a block of memory
   - — Move a block of memory
   - — Fill a block of memory
2. Register manipulation commands
   - — Display all user CPU registers
   - — Examine/modify one register, or several
3. Program execution commands
   - — Execute user program from address xxxx
   - — Set a breakpoint at address xxxx (up to five total)
   - — List active breakpoints
   - — Kill a breakpoint (1 or all)
4. I/O commands
   - — Input byte from a specified port
   - — Output a byte to a specified port
5. Miscellaneous Commands
   - — Download a file from the main or auxiliary port
   - — Verify the success of the download operation
   - — Configure the auxiliary port
   - — Show a list of commands
   - — Calculate an address offset: offset = yyyy − xxxx
   - — Convert Hex-to-Decimal, Decimal-to-Hex

In addition to the command set are several monitor service routines that can be called from user programs. These routines include:

- — Text string output
- — Text string input
- — Output a hex input byte as two ASCII characters
- — Output a hex byte to the serial port
- — Input to the accumulator from the serial port
- — Convert a Hex nibble to an ASCII byte
- — Convert an ASCII byte to a hex nibble
- — 16-bit unsigned comparison
- — 8-bit unsigned multiply
- — 16-bit unsigned multiply

Note: I/O calls use the main serial channel.

3.2 MON800 Organization and Operation

For detailed monitor functions and structure, refer to the MON800 listing file that is available on Dial-A-Helper. (See Appendix F). The MON800 program is organized in the following sections:

A. System reset/initialization routine

B. Monitor command interpreter loop "GETCOM"
   - — Individual command programs

C. Monitor Restart/Interrupt Routines

D. Monitor Subroutines

E. Data Tables

The MON800 program uses RAM during its operation. 256 bytes are reserved at addresses FF00h to FFFFh. User programs should not use this area, and the monitor will not allow the user to write these locations when in monitor command mode. A memory map of the monitor's RAM space is listed in Table 3.1.

TABLE 3.1. MON800 RAM Usage

Hex Address	Function
FF00	Default User Stack Base (Initial Value)
FF00–FF60	Monitor Stack Area (Base = FF60)
FF64–FF7D	User CPU Register Set Storage
FF7E–FF96	MON800 Flags, Break Addresses, etc.
FF97–FFB7	Interrupt Vector Table (11 Vectors)
FFB8–FFFF	I/O Buffer, 72 Bytes

3.2.1 System Initialization

Upon power-up or reset, the system will initialize itself. The first thing it does is initialize its own RAM data. The stack pointer is loaded, user breakpoints are cleared, status flags are initialized, the Interrupt Vector Table is loaded with MON800's default vector set, and the user CPU registers are set for a default reentry to the monitor command loop. The serial ports are then initialized. A sign-on message is sent to the main serial port when the initialization is complete.

3.2.2 The "GETCOM" Loop

Once the system is initialized, control passes to the command interpreter loop, labeled "GETCOM" on the list file. This routine sends a prompt to the terminal and waits for input. When the carriage return is typed, the routine looks in the I/O buffer for the first non-space character. This character must be a capital A–Z, or an "INVALID COMMAND" message will display. If the first letter is valid, the routine then passes control to the command routine specified. Some of the letters A–Z have no function, and these will also result in an error message. When the command routine has completed execution, control passes back to "GETCOM". This is the starting point for all monitor operations.

3.2.3 Interrupts and Restarts

When the NSC800 encounters an interrupt or restart from either the hardware inputs or by a "RST xx" instruction, the current Program Counter (PC) register value is pushed onto the stack, and the PC is loaded with a predefined value, or vector, that is the entry point for the corresponding interrupt request. Since these hard vectors are in ROM at the start, MON800 routes interrupt requests through a second set of vectors located in MON800's reserved RAM space. These are soft vectors, and can be modified by user programs.

The Interrupt Vector Table is located in RAM at addresses FF97h to FFB7h. At reset or power-on, this table is loaded with MON800's default interrupt vector set. The table has eleven entries, one for each interrupt source except reset. The vector entry is simply a "JP xxxx" instruction that points to a service routine in MON800 ROM. The user can change the jump address to another location, most likely the entry point to a user's own interrupt service routine. One way to do this is to use a block move instruction to load a user's vector set into the table. Table 3.2 lists the Interrupt Vector Entries and default functions.

Note that some of the default functions are critical for proper monitor operation. For example, breakpoints use the RST 30 vector. If it is changed, breakpoints will no longer function.

TABLE 3.2. MON800 Interrupt Vector Table

Hex Address	Interrupt	Default Function
FF97	RST 08	Re-enter the Monitor
FF9A	RST 10	Monitor Service Call
FF9D	RST 18	No Function
FFA0	RST 20	No Function
FFA3	RST 28	No Function
FFA6	$\overline{\text{RSTC}}$	No Function
FFA9	RST 30	Monitor Breakpoint Trap
FFAC	$\overline{\text{RSTB}}$	No Function
FFAF	RST 38, $\overline{\text{INTR}}$	No Function
FFB2	$\overline{\text{RSTA}}$	No Function
FFB5	$\overline{\text{NMI}}$	Re-enter the Monitor

Example: Modifying an interrupt vector.

The vector is an opcode for a "JP xxxx" Instruction. Its format is "C3 aabb", where C3 is the opcode, aa is the low byte of the address, and bb is the high byte. If the vector to be modified is RST 20, the new address should be loaded at address FFA1. This can be done with the following code:

```
LD HL, U_NTRY      ;GET THE USER ENTRY POINT ADDRESS xxxx
LD (FFA1H), HL     ; MODIFY THE VECTOR
```

Note: Use of breakpoints in interrupt routines may produce unpredictable results if the interrupts are still active when the monitor is re-entered. The monitor breakpoints were not intended for real-time use. Rather, use the register manipulation commands to load CPU registers with simulated data values, and use the breakpoints to debug the logical flow of the code before running it in real-time.

3.3 MON800 Command Descriptions and Examples

The following sections describe the individual MON800 commands and their usage. All MON800 commands are identified by the capital letters A–Z. All alphabetic entries must be in upper case. Each term in the command line should be separated by at least one space.

3.3.1 A—Assemble NSC800 Instruction

Syntax: A xxxx Where: xxxx = load address

Description

This command invokes a line assembler routine which prompts the user for a NSC800 instruction in mnemonic form, assembles the instruction, and loads the opcode at address xxxx. The prompt issued by the line assembler is the address at which the opcode will be loaded at. If there are no assembly errors, the routine will prompt for another instruction at the next sequential address. To exit the routine, type the return key <CR> twice.

Example:

```
>A 8000<CR>
8000:LD C,B<CR>
8001:ADD A,(IX+6)<CR>
8004:CP 055<CR>
8006:JP M,D1F2<CR>
8009:SET 7,(HL)<CR>
800B:<CR>
>
```

Assembler Rules

1. All alphabetic text must be capitals.
2. All numeric entries are interpreted as hex values.
3. Immediate values must be preceded by a zero if the operand can also be a register. For example, the assembler will not know the difference between "CP B" and "CP B7". To be correct, use "CP 0B7".
4. Blank characters are allowed before the mnemonic and after, but not in the operand string or after.

3.3.2 B—Set Monitor Breakpoint

Syntax: B xxxx

Description

This command will cause a monitor breakpoint to be placed at address xxxx. The address must be in hexidecimal format. When the CPU fetches an opcode from this address, the current CPU state is saved in monitor RAM space and control of the system is returned to the monitor.

Example:

```
> B 4000
```

This will cause a breakpoint to be placed at hex address 4000.

Restrictions and Considerations

1. The break address must be in valid RAM space.
2. A maximum of five active breakpoints are allowed at any one time.
3. The address must be the location of the first opcode byte of an instruction. Breakpoints at any other locations will not be recognized and will result in a F7H byte being inserted into the instruction stream at that location.
4. Upon recognition of a breakpoint, the value of the PC register will be pushed onto the user's stack. The stack pointer will be restored once the user PC value has been removed by the monitor. If user programs modify the SP register, care should be taken to insure that the SP is in valid RAM space at the time of the break.
5. When a breakpoint is encountered, system interrupts are not disabled. Use caution when using breakpoints in an interrupt-driven system.

3.3.3 C—Convert Hex to Decimal/Decimal to Hex

Syntax: C <D or H> [value]

Description

This command will convert a hex value (0000–FFFF) to decimal, or a decimal value (0–65535) to hex. "D" specifies decimal output, in which case the input, [value], should be hex. "H" specifies hex output, in which case [value] should be decimal. If nothing is specified for [value], the monitor will prompt for input.

Examples:

1. `> C D 10`
2. `> C D`
   `INPUT HEX VALUE: 10`

Both (1) and (2) result in the output: =00016

3. `> C H 16`
4. `> C H`
   `INPUT DECIMAL VALUE: 16`

Both (3) and (4) result in the output: =00010

3.3.4 D—Download Intel Hex File

Syntax: D <1 or 2>

Description

This command will cause the monitor to download an Intel Hex file via the main serial port, specified by "1", or via the auxiliary serial port. When the command is issued, the monitor will "listen" to the specified port for the incoming file. The file must then be manually transmitted from the remote system. There is no handshaking between the NSC800 system and the remote system.

When the file is completely received, the monitor verifies the checksum and will output a status message indicating success or failure to the main port. If there is a break in transmission, or invalid characters are received, the download is aborted and a message issued. The status of the download operation can also be viewed by the "V" command.

Example:

`> D 2`

This will cause the monitor to listen to the auxiliary serial port for incoming data. The ASCII character ":" signifies the start of a record. The monitor will ignore all input unitl it gets ":". When all records are received, including the end-of-file record, the checksum is verified. If the checksum is good, the message "DOWNLOAD SUCCESSFUL" will be output to the main port. The download will be aborted if errors occur. A list of possible failure messages follows:

"FAILED—NON-HEX CHAR" —indicates an illegal character found.

"FAILED—BAD LOAD ADDRESS" —data address is not in user RAM space.

"FAILED—CHECKSUM ERROR" —indicates a checksum error.

"FAILED—BAD RECORD TYPE" —only record types 0 and 1 are allowed.

"FAILED—VERIFY FAILED" —data load to RAM failed.

For a description of the Intel Hex file format, see Appendix D.

Usage Considerations

Downloads to the main serial channel may require more steps to complete the operation. There will be four main steps in the process.

Step 1. With the RS232 terminal at the main port, issue the command "D 1".

Step 2. Remove the terminal from the main port, and attach the device which will be sending the file. An alternative to this would be for the remote system, say a PC, to use a terminal emulation program to complete Step 1, then switch from the terminal emulator to DOS or some other mode that allows the file to be transmitted via the PC's serial port. (See Appendix E for an example of how the KERMIT program is used to communicate with MON800 from an IBM PC/AT.)

Step 3. Send the file from the remote device. On a PC/AT the DOS command.

`> copy file.hex com1:`

can be used, provided that the port called com1: is attached to the NSC800 system at the main port.

Step 4. When the file has been sent, reattach the terminal to the main port and use the "V" command to check the status of the download.

3.3.5 E—Examine/Modify a Single Memory Location

Syntax: E xxxx

Description
This command allows the user to examine or change a memory location specified by hex address xxxx.

Example:

```
> E 2000
<ESC> TO EXIT
2000 5F - 00 <CR>
2001 5F - <CR>
2002 5F - 00 <CR>
2003 5F - <ESC>
>
```

This entry will cause the monitor to display the contents of address 2000H. The monitor then prompts the user for input with "-". The contents can be changed by entering the new data and hitting return <CR>. The monitor then displays the next address and prompts again. If the user types <CR> without entering any data, the monitor will move to the next location without changing the contents of the current location. Hitting the escape <ESC> key causes an immediate exit from the routine. The above sequence will affect the memory in the following way.

Address	Old Data	New Data
2000	5F	00
2001	5F	5F
2002	5F	00
2003	5F	5F

Restrictions
This command may only be used to modify address locations that are in user RAM space. Attempts to write reserved RAM locations are not allowed.

3.3.6 F—Fill Memory Block

Syntax: F xxxx yyyy zz

Description
This command will cause hex memory locations xxxx to yyyy to be filled with hex data zz.

Example:

```
> F 2000 20A0 5A
```

The result of this command will be the hex data value 5A being written to memory addresses 2000 through 20A0.

Restrictions

1. Both start and end addresses must be in user RAM.
2. The block end address, yyyy, must be the same or higher than the block start address, xxxx.
3. Attempts to fill reserved RAM locations are not allowed.

3.3.7 G—Go. Begin User Program Execution

Syntax: G [xxxx]

Description
The "G" command will load the NSC800 CPU registers from reserved RAM locations where the user's CPU state has been saved, and begin execution. A hex start address, xxxx, may be specified, and program execution will begin at that location. If no start address is specified, the PC value that was saved at the time the last breakpoint was encountered will be used.

Examples:

1. `> G 2000`

This will cause the CPU to execute code at address 2000.

2. `> G`

This will cause the CPU to execute code at the current user program's PC location that was saved at the time of the last breakpoint. If no breakpoint was encountered prior to this, the monitor will be re-entered by default.

3.3.8 H—Help. Display List of Commands

Syntax: H

Description
This command will cause a command menu to be displayed on the terminal.

3.3.9 I—Input from I/O Port

Syntax: I xx

Description
The monitor will read the I/O port location specified by hex value xx, and display its contents to the screen.

3.3.10 J—Calculate Jump Offset

Syntax: J xxxx yyyy

Description

This command provides the user with a quick way of determining the hex offset between two hex addresses xxxx and yyyy. The monitor calculates the difference yyyy − xxxx and displays the result to the screen. The hex values are treated as unsigned integers. This is especially handy when computing negative offsets.

Example:

```
> J AAAB AAAA
```

The monitor will display the result FFFF, which is an offset of negative one.

3.3.11 K—Kill, or Delete, Monitor Breakpoint(s)

Syntax: K [xxxx]

Description

This command will delete a breakpoint at hex location xxxx. If no address is specified, all breakpoints are deleted.

3.3.12 L—List Monitor Breakpoint(s)

Syntax: L

Description

Using this command, the user can view the current breakpoint addresses.

Example:

If breakpoints exist at locations 2000, 2002, and 2004, then the following sequence of commands will produce the following output.

```
>L
2000 20002 20004
>K 20002
>L
2000 2004
>B 2002
>L
2000 2002 2004
>K
>L


>
```

This shows how the B, K, and L commands can be used to set, delete, and list monitor breakpoints.

3.3.13 M—Display Memory Block

Syntax: M xxxx yyyy

Description

The contents of the memory locations between hex addresses xxxx and yyyy can be displayed to the screen using this command. Addresses that are specified are rounded to 16-byte blocks when displayed.

Example:

```
> M 4005 4015
```

This entry will produce the following output:

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    ASCII
4000 00 FF 00 FA 00 55 03 02 01 0C C0 BB FF 55 66 77 .....U.......Uf.
4010 00 00 00 00 00 CC DD DC BB 00 55 AA FF D3 F6 75 ..........U.....
>
```

3.3.14 O—Output Byte to I/O Port

Syntax: 0 xx dd

Description

This command will cause the hex data byte dd to be output to I/O port location xx (hex).

Example:

```
> 0 21 55
```

This will cause the data 55H to be written to I/O port location 21H.

3.3.15 P—Configure Auxiliary Port

Syntax: P modulus mode stop

Description

This command can be used to initialize the NSC858 UART for the auxiliary serial port in the following way.

modulus —is a 16-bit hex value that will be loaded into the baud rate divisor latches of the NSC858 UART.

mode —is an 8-bit hex value that will be written to both TxMODE and RxMODE registers of the NSC858 UART.

stop —a 2-bit hex value indicating the number of stop bits to use. 0 = 1 stop bit, 1 = 1.5 stop bits, 2 = 2 stop bits.

Example:

```
> P  0C B8  2
```

The NSC858 registers will be loaded as follows:

Register Name	I/O Address	Contents	Comments
Receiver Mode	21H	B8H	RxC Int., 8 Data, Even Pty.
Transmitter Mode	22H	B8H	TxC Int., 8 Data, Even Pty.
Global Mode	23H	09H	2 Stop, 16X Clock Factor
Command	24H	C3H	* Default *
Baud Rate Divisor (LSB)	25H	0CH	9600 Baud (1.84 MHz)
Baud Rate Divisor (MSB)	26H	00H	

This command provides the user with a quick way to configure the auxiliary serial port.

3.3.16 R—Examine/Modify User CPU Registers

Syntax: R [specifier]

Where: [specifier] = A, B, C, D, E, F, H, L, A', B', C', D', E', F', H', L', I, IX, IY, SP or PC.

Description

The user CPU registers can be examined or modified using this command. An individual register may be specified with an optional specifier term. If no specifier is given, the monitor will display all user CPU register values. To modify a particular register's contents, a specifier must be given.

Examples:

1.

```
> R
A  F  B  C  D  E  H  L  A'  F' B'  C'  D'  E'  H'  L'  I  IX  IY  SP  PC
00 0C 55 23 45 66 77 A4 D2 00 00 00 00 00 00 00 45 1234 5678 ABCD 2000
>
```

2.

```
> R A
<ESC> TO EXIT
A  00 - 33 <CR>
F  0C - <CR>
B  55 - <ESC>
>
```

In Example 1, all register values, with the exception of the R register, will be displayed. When a specifier is given as in (2), a register can be modified by typing in the new value and hitting carriage return, <CR>. A return by itself will display the next register without modifying the previous one. Typing escape, <ESC>, will exit the program immediately.

3.3.17 T—Move a Block of Memory

Syntax: T xxxx yyyy zzzz
Where: xxxx = block start address
yyyy = block end address
zzzz = destination address

Description
This command copies the contents of memory between and including start address xxxx and end address yyyy to the destination address zzzz. The source block can be anywhere in the memory space, including ROM.

Rules

1. The block end address yyyy must be equal to or larger than the block start address xxxx. If not, the message "ILLEGAL ADDRESS" will display.
2. There must be enough room in the user RAM area to store the destination block, or the message "TOO LARGE" will display.

3.3.18 V—Verify Download Status

Syntax: V

Description
This allows the user to verify the status of the most recent file download operation. It is useful when using the main serial port to download files. When a download to the main port is complete, or an abort has occurred, a status message is sent back to the main port. If the terminal is not ready to receive this message, the user can verify the status by this command, which will repeat the status message one time for every download.

3.4 MON800 Service Calls

MON800 includes a set of handy utility routines that can be called by user programs. These include terminal I/O routines, data conversion routines, and math routines. To call a service routine from a user program, the "A" register is loaded with the call number, the necessary registers are loaded with appropriate input values, and the "RST 10H" instruction is issued. Note that the default RAM vector for RST 10 (addresses E003-4) must not be altered, or the service call routine will not be entered correctly. If the call number is illegal, an error message is displayed and control returns to the monitor. The monitor service routines are described in the remainder of this section.

3.4.1 Text String Output Utility

Inputs: A = 00
HL = Address of first byte of ASCII string
Outputs: None

Description
This routine will output an ASCII character string to the terminal via the main serial channel. The HL register contains the address of the first byte in the ASCII string (lowest address). The string should terminate with NULL (00) byte. The routine will output characters to the main port until a 00 byte is encountered in the data string.

3.4.2 Text String Input Utility

Inputs: A = 01
HL = Starting address of input buffer space
BC = Max. buffer size, in bytes
Outputs: BC = Max_size—Bytes_buffered

Description
This routine allows text to be input from the terminal via the main serial channel to a buffer area in RAM. The HL register specifies the starting address of the input buffer. The BC register specifies the maximum size, in bytes, of this buffer. The routine will ignore input after the maximum size has been reached. The program returns to the caller when a carriage return (0DH) is detected. HL is unaffected. BC will equal the input value minus the number of bytes buffered.

3.4.3 Output ASCII Hex Byte

Inputs: A = 02
B = Output byte
Outputs: None

Description
The contents of register B are converted to two ASCII bytes which represent the hex byte in that register. The ASCII characters are then sent to the main serial port. The contents of the B register are unaffected.

3.4.4 Output Hex Register Contents

Inputs: A = 03
B = Output byte
Outputs: None

Description
The binary contents of register B are sent directly to the main serial port. The contents of B are not affected.

3.4.5 Input Hex Byte to Accumulator A

Inputs: A = 04
Outputs: A = Input byte

Description
This routine polls the main serial channel for input. The first byte received is loaded into register A and is returned to the caller.

3.4.6 Convert Hex Nibble in Register B to ASCII Byte in A

Inputs: A = 05
B = Hex Nibble Input (least significant nibble)
Outputs: A = ASCII byte output

Description
This program will convert the binary value of the least significant nibble of register B to a ASCII hex character 0–F. The ASCII byte is returned in A. B is unchanged.

3.4.7 Convert ASCII Byte in B to Hex Nibble in A

Inputs: A = 06
B = ASCII input
Outputs: A = 0x, where x = hex nibble
= FF, if B = non-hex ASCII character

Description
The contents of register B will be converted to a four-bit hex representation. This nibble is returned in A. If the ASCII input is a character other than the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E or F (no lower case letters), the A register will be loaded with FFH.

3.4.8 16-Bit Unsigned Compare HL–DE

Inputs: A = 07
HL = 16-bit input value
DE = 16-bit input value
Outputs: A = 00, if HL > DE
A = F0, if HL = DE
A = FF, if HL < DE

Description
This program performs a non-destructive, unsigned comparison of the 16-bit values in the HL and DE registers. The result of the compare operation is returned in register A.

3.4.9 8-Bit Unsigned Multiply: HL = D * E

Inputs: A = 08
D = 8-bit multiplier
E = 8-bit multiplicand
Outputs: HL = 16-bit product

Description
This program multiplies the contents of the D and E registers and places the result in the HL register. The contents of D are destroyed.

Note: The execution speed of this program is affected by the service call decoder execution. To multiply more effectively, use the "MULT08" subroutine found in the MON800 program listing. This code segment can be included in the user program to increase execution speed.

3.4.10 16-Bit Unsigned Multiply

Inputs: A = 09
HL = 16-bit multiplicand
DE = 16-bit multiplier
Outputs: IX = 32-bit product (lower half)
IY = 32-bit product (upper half)

Description
The contents of HL and DE are multiplied, and the result is placed in the IX and IY registers. The contents of HL, DE and BC are destroyed.

Note: The execution speed of this routine is affected by the service call decoder program. To multiply more efficiently, place the "MULT16" code segment from the MON800 listing in the user program. This will greatly increase the execution speed.

APPENDIX A. CONNECTOR PIN DESCRIPTIONS

The following tables list pin descriptions and pin assignments for the interface connectors/headers in the NSC800 Applications System. Refer to the kit schematic and the NSC800 series datasheets for detailed operation of each signal.

Signal Access Headers
Tables A-2, A-3 and A-4 show the individual pin functions of the interface headers J3, J4 and J5, respectively. The signal mnemonics listed correspond to signal names in the NSC800 Applications System schematic. For specific functions of each signal, refer to the schematic and data sheets.

TABLE A-1. Pin Assignments and Signal Directions for Serial Connectors J1 and J21 (E.I.A. Standard RS232C)

Connector	Pin	Mnemonic	Description	Direction
J1 (DCE)	1	PG	Protective Ground	Ground
J2 (DCE)	2	RXD	Received Data	Out
	3	TXD	Transmitted Data	In
	4	CTS	Clear to Send	Out
	5	RTS	Request to Send	In
	6	DTR	Data Terminal Ready	In
	7	SG	Signal Ground	Ground
	8–19	—	Not Used	
	20	DSR	Data Set Ready	Out
	21–25	—	Not Used	
J2 (DTE)	1	PG	Protective Ground	Ground
	2	TXD	Transmitted Data	Out
	3	RXD	Received Data	In
	4	RTS	Request to Send	Out
	5	CTS	Clear to Send	In
	6	DSR	Data Set Ready	In
	7	SG	Signal Ground	Ground
	8	DCD	Data Carrier Detect	In
	9–19	—	Not Used	
	20	DTR	Data Terminal Ready	Out
	21–25	—	Not Used	

TABLE A-2. Parallel I/O—Timer Access Header—J3

Pin	Mnemonic	Pin	Mnemonic
A1	1PA0	B1	2PA0
A2	1PA1	B2	2PA1
A3	1PA2	B3	2PA2
A4	1PA3	B4	2PA3
A5	1PA4	B5	2PA4
A6	1PA5	B6	2PA5
A7	1PA6	B7	2PA6
A8	1PA7	B8	2PA7
A9	1PB0	B9	2PB0
A10	1PB1	B10	2PB1
A11	1PB2	B11	2PB2
A12	1PB3	B12	2PB3
A13	1PB4	B13	2PB4
A14	1PB5	B14	2PB5
A15	1PB6	B15	2PB6
A16	1PB7	B16	2PB7
A17	1PC0	B17	2PC0
A18	1PC1	B18	2PC1
A19	1PC2	B19	2PC2
A20	1PC3	B20	2PC3
A21	1PC4	B21	2PC4
A22	1PC5	B22	2PC5
A23	1T0IN	B23	2T0IN
A24	1T0OUT	B24	2T0OUT

TABLE A-3. CPU Bus Signal Access Header—J4

Pin	Mnemonic	Pin	Mnemonic
A1	AD0	B1	A0
A2	AD1	B2	A1
A3	AD2	B3	A2
A4	AD3	B4	A3
A5	AD4	B5	A4
A6	AD5	B6	A5
A7	AD6	B7	A6
A8	AD7	B8	A7
A9	ALE	B9	A8
A10	$\overline{RD}$	B10	A9
A11	$\overline{WR}$	B11	A10
A12	IO/$\overline{M}$	B12	A11
A13	RES OUT	B13	A12
A14	$\overline{PS}$	B14	A13
A15	$\overline{WAIT}$	B15	A14
A16	$\overline{BREQ}$	B16	A15
A17	$\overline{INTR}$	B17	$\overline{INTA}$
A18	$\overline{RSTC}$	B18	$\overline{RFSH}$
A19	$\overline{RSTB}$	B19	S0
A20	$\overline{RSTA}$	B20	S1
A21	$\overline{BACK}$	B21	CLK
A22	$\overline{RAM5}$	B22	$\overline{IO1}$
A23	$\overline{RAM4}$	B23	$\overline{IO2}$
A24	$\overline{RAM3}$	B24	$\overline{IO3}$
A25	$\overline{RAM2}$	B25	Not Used
A26	$\overline{XRES\ IN}$	B26	$\overline{XNMI}$

TABLE A-4. UART Signal Access Header—J5

Pin	Mnemonic	Pin	Mnemonic
A1	$\overline{1RTI}$	B1	2RXD
A2	$\overline{1RXC}$	B2	2TXD
A3	$\overline{1TXC}$	B3	$\overline{2CTS}$
A4	$\overline{2RTI}$	B4	$\overline{2RTS}$
A5	$\overline{2RXC}$	B5	$\overline{2DSR}$
A6	$\overline{2TXC}$	B6	$\overline{2DTR}$
A7	Not Used	B7	$\overline{2DCD}$

Note: Do not drive UART input lines B1, B3, B5 or B7 unless the DS14C89 receiver IC (U5) has been removed or disconnected.

APPENDIX B. SERIAL PORT CONFIGURATION SETTINGS

Table B-1 lists the switch settings that control the serial channel initialization at power-up or reset. DS3 is read by the NSC800 via Port A of NSC810A # (U2). The switch's data is gated onto the Port A data bus by setting Port C, bit 1 (PC1) of the same NSC810A to a low level (logic 0). Thus, the switch may also be accessed by user programs, and unused switch positions S5–S8 may have custom functions assigned to them. In order to use Port A for any other purpose, the programmer must be sure to set PC1 to a high (logic 1) level). For applications that use the strobed I/O function of the NSC810A, use NSC810A #2 (U3) for this purpose.

TABLE B-1. Serial Channel Initialization Settings—DS3

Function	S1	S2	S3	S4	S5	S6	S7	S8
Baud = 1200	x	x	Off	Off	x	x	x	x
Baud = 2400	x	x	Off	On	x	x	x	x
Baud = 4800	x	x	On	Off	x	x	x	x
Baud = 9600	x	x	On	On	x	x	x	x
Data Bits = 7	x	On	x	x	x	x	x	x
Data Bits = 8	x	Off	x	x	x	x	x	x
Stop Bits = 1	On	x	x	x	x	x	x	x
Stop Bits = 2	Off	x	x	x	x	x	x	x

APPENDIX C. JUMPER OPTIONS AND SETTINGS

W1—Reset Input	On Board Switch (S1) :	1 to 2
	Remote Input (J4-A26):	2 to 3
W1—/NMI Input	On Board Switch (S2) :	1 to 2
	Remote Input (J4-B26):	2 to 3
Main Serial Channel—RS232		
W3—RS232 Chassis Ground (J1-Pin1).	Isolation:	Open
	System Ground:	1 to 2
Auxiliary Serial Channel—RS232 Interface Only		
W4.1—RS232 Chassis Ground (J2-Pin1).	Isolation:	Open
	System Ground:	1 to 2

Aux. Port Configuration Settings	DTE	DCE
W4.2—TXD/RXD	1 to 2	1 to 3
	3 to 4	2 to 4
W4.3—CTS/RTS	1 to 2	1 to 3
	3 to 4	2 to 4
W4.4—DSR/DTR	1 to 2	1 to 3
	3 to 4	2 to 4
W4.5—DCD	1 to 2	1 to 3
		2 to 4

APPENDIX D. INTEL HEX FILE FORMAT

This section describes the file transfer format that is used when downloading linked executable code modules to the NSC800 Applications System using the MON800 "D" command. The program information is contained in groups of ASCII characters called load-records. An Intel load-record has the following general format:

: nn aaaa tt dd...dd cc

where:

: is the start-of-record mark (hex 3A).

nn is the record length field. Two ASCII characters represent the number of data bytes (in hex) that are in the load-record. A zero value here indicates an end-of-file record.

aaaa is the load address field. Four ASCII characters represent the starting hexadecimal load address for the data in the record. The data is loaded in successive addresses.

tt is the record type field. Two ASCII characters represent the record type: 00 = data record, 01 = end-of-file record, 02 = extended address record, 03 = start address record. Only record types 0 and 1 are accepted by MON800.

dd . . . dd is the data field. Each byte of data in the record is represented by two ASCII characters that indicate its hex value.

cc is the checksum field. The checksum is calculated by taking the hexadecimal sum of the fields nn, aaaa, tt, dd . . . dd and cc. The final sum, taken modulo 2, should be zero. Thus, cc is the negative sum of the hex bytes in the record.

Example. Here is an example data load-record followed by an end-of-file record.

```
:0C20300004422CFED430622ED430422CF32
:00000001FF
```

APPENDIX E. EXAMPLE INTERFACE: IBM AT-NSC800 APPLICATIONS SYSTEM

This is an example of how off-the-shelf software can be used to create a direct interface from the IBM PC/AT to the NSC800 Applications System. The interface is simple, fairly easy to use, and will allow the PC to function as a software development tool for the NSC800. The example setup is as follows:

- IBM PC/AT with serial port
- NSC800 Applications System microcomputer and power supply
- MS DOS version 3.0 or higher
- KERMIT serial I/O program (Columbia University, public domain software)
- NSC800 or Z80 cross-assembler and linker package, C cross-compiler, etc. (Intel Hex output is a must)

Operation

1. The NSC800 system is connected to the AT serial port through its main serial channel. The AT port is assumed to be DTE. The critical signals needed are Transmitted Data (pin 2), Received data (pin 3), and ground (pin 7).
2. NSC800 programs can be coded, assembled, and linked using the PC-resident software. For this example, a program called test800.asm is created using a text editor. Its assembled and linked Intel hex output file is called test800.hex. This is the program to be downloaded to the NSC800 system for debugging.
3. The NSC800 system is configured to meet the RS232 requirements of the PC's serial port.
4. On the PC, the KERMIT program is run. KERMIT will emulate a VT100 compatible terminal at the PC comm port. The NSC800 system is powered on, and a MON800 sign-on message should appear on the PC screen.
5. MON800 commands can now be issued to the NSC800 system directly from the PC. To download the example program, test800.hex, the MON800 command "D 1" is given.
6. To download a file from the PC, the KERMIT program is exited to DOS, and the DOS command

   "copy \<pathname>\test800.hex com1: "

   is given. The file should be sent to the serial port.
7. After the file is sent, the KERMIT program is again run so that MON800 commands can be issued to the system. The "V" command is used to find out any error messages associated with the file download. A "M xxxx yyyy" command will display the program information that was loaded into RAM. Once the file is in RAM, MON800 can be used to debug the user program.

APPENDIX F. MON800 PROGRAM LISTING

A complete assembler list file of the MON800 program, MON800.LST includes four files.
They are:

MON800A.ASM — ASCII source code for MON800 rev A.

MON800A.OBJ — Relocatable object code (binary)

MON800A.HEX — Linked executable code module (Intel hex format)

MON800A.LST — Assembler list file

The user may make changes to the ASCII source code to customize MON800 to fit individual system requirements. System address assignments and constants are configured by means of the equate statements at the beginning of the assembly program.

The MON800 program occupies approximately 8k bytes of ROM, and, if unmodified, this program will fit in any 8k by 8 EPROM such as National's NMC27C64Q-250 CMOS EPROM. National Semiconductor does not guarantee this software. All program changes are made at the user's own risk.

The code described in this App Note is available on Dial-A-Helper.

Dial-A-Helper is a service provided by the Microcontroller Applications Group. The Dial-A-Helper system provides access to an automated information storage and retrieval system that may be accessed over standard dial-up telephone lines 24 hours a day. The system capabilities include a MESSAGE SECTION (electronic mail) for communicating to and from the Microcontroller Applications Group and a FILE SECTION mode that can be used to search out and retrieve application data about NSC Microcontrollers. The minimum

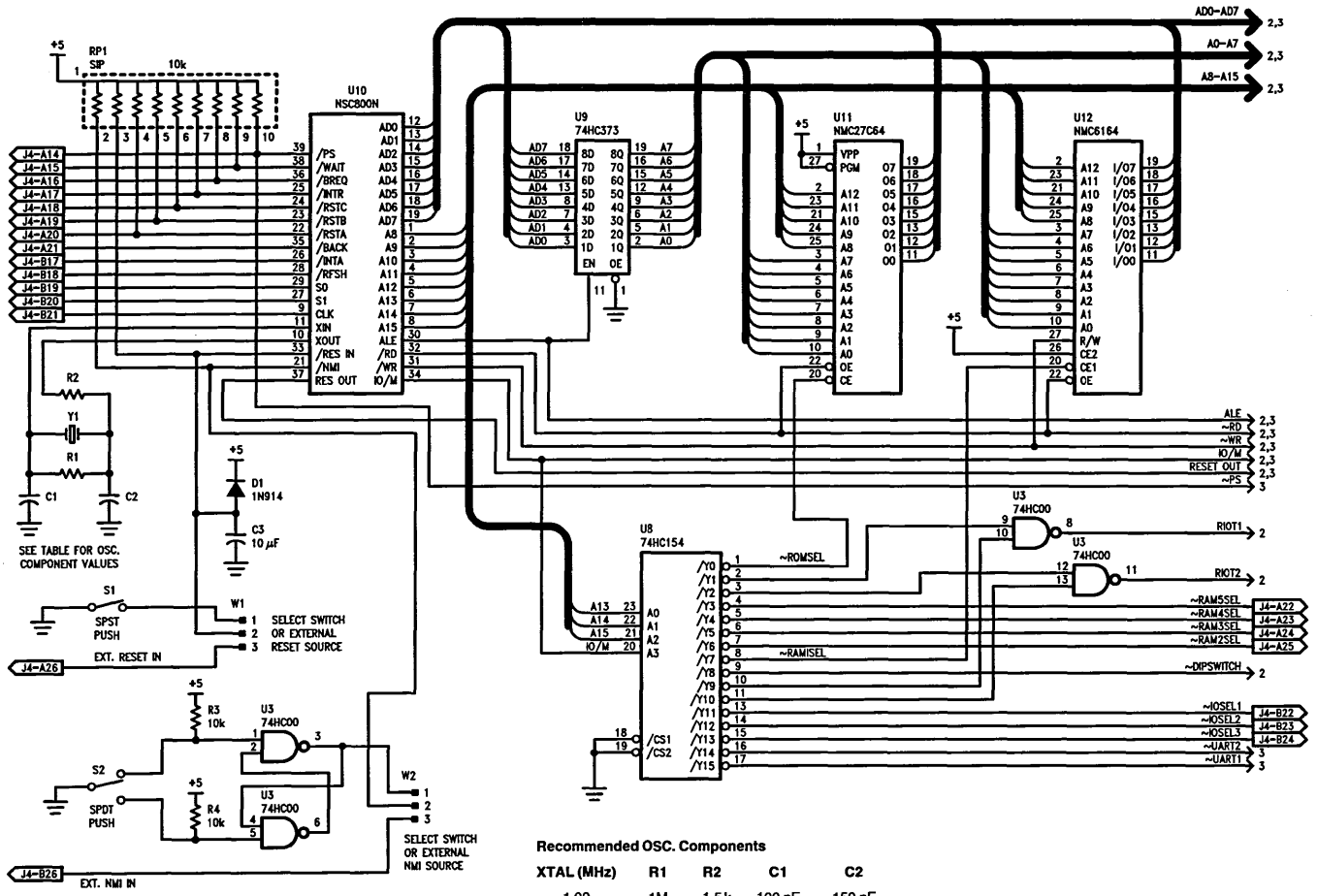
system requirement is a dumb terminal, 300 or 1200 baud modem, and a telephone. With a communications package and a PC, the code detailed in this App Note can be down loaded from the FILE SECTION to disk for later use. The Dial-A-Helper telephone lines are:

Modem (408) 739-1162
Voice (408) 721-7264

For Additional Information, Please Contact the Factory

APPENDIX G. NSC800 APPLICATIONS SYSTEM SCHEMATICS

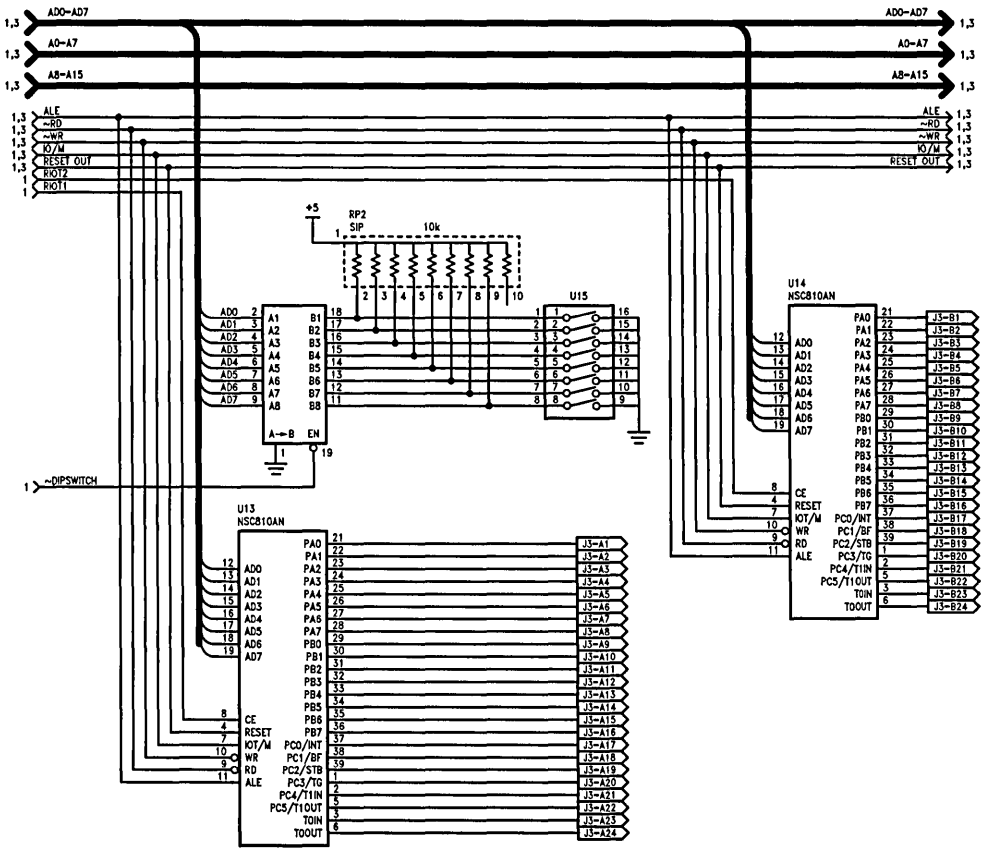
The schematics for the NSC800 are provided on the following pages. Also included is a suggested chip layout for a wire-wrapped or PC board and a component list.



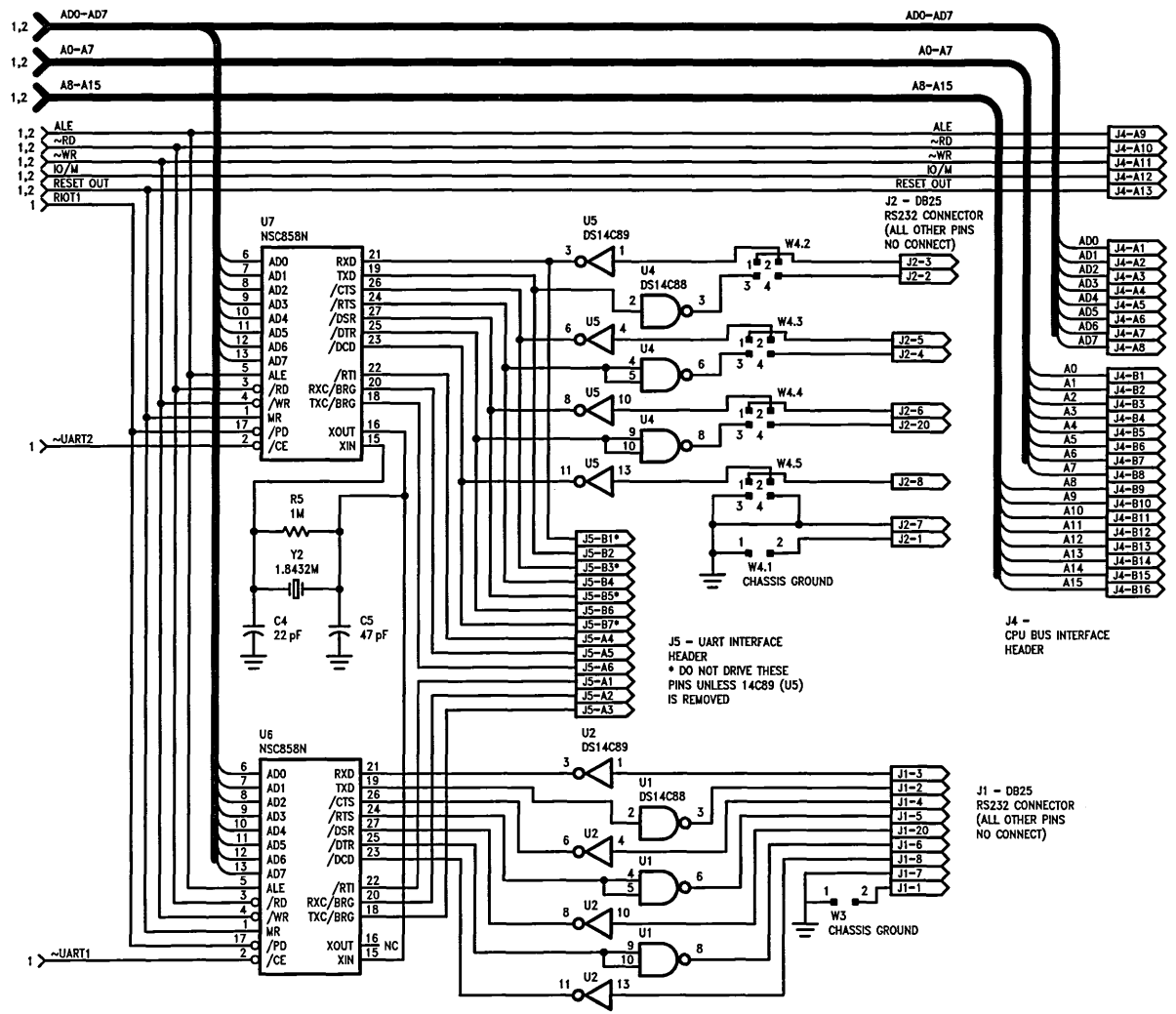
Recommended OSC. Components

XTAL (MHz)	R1	R2	C1	C2
1.00	1M	1.5k	100 pF	150 pF
2.00	1M	470	68 pF	100 pF
4.00	1M	0	33 pF	47 pF
8.00	1M	0	22 pF	33 pF

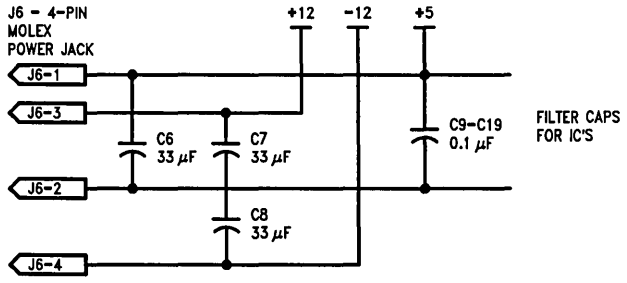
TL/C/10435–2


AD0–AD7
A0–A7
A8–A15
ALE
~RD
~WR
IO/M
RESET OUT
RIOT2
RIOT1
+5
RP2
SIP
10k
U15
U14
NSC810AN
U13
NSC810AN
~DIPSWITCH
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
CE
RESET
IOT/M
WR
RD
ALE
PC0/INT
PC1/BF
PC2/STB
PC3/TG
PC4/T1IN
PC5/T1OUT
TOIN
TOOUT

TL/C/10435–3


AD0–AD7
A0–A7
A8–A15
ALE
~RD
~WR
IO/M
RESET OUT
RIOT1
~UART2
~UART1
U7
NSC858N
U6
NSC858N
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
ALE
/RD
/WR
MR
/PD
/CE
RXD
TXD
/CTS
/RTS
/DSR
/DTR
/DCD
/RTI
RXC/BRG
TXC/BRG
XOUT
XIN
NC
R5
1M
Y2
1.8432M
C4
22 pF
C5
47 pF
U5
DS14C89
U2
DS14C89
U4
DS14C88
U1
DS14C88
W4.2
W4.3
W4.4
W4.5
W4.1
CHASSIS GROUND
W3
CHASSIS GROUND
J2 – DB25
RS232 CONNECTOR
(ALL OTHER PINS
NO CONNECT)
J1 – DB25
RS232 CONNECTOR
(ALL OTHER PINS
NO CONNECT)
J5 – UART INTERFACE
HEADER
* DO NOT DRIVE THESE
PINS UNLESS 14C89 (U5)
IS REMOVED
J4 –
CPU BUS INTERFACE
HEADER

TL/C/10435–4



TL/C/10435-5

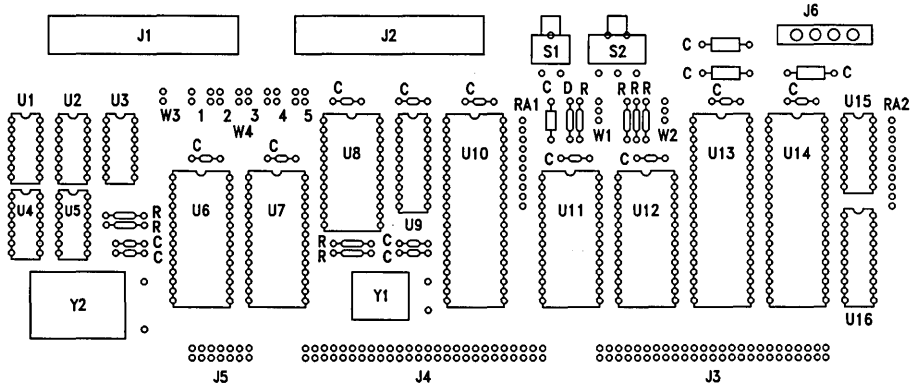
Place filter caps near IC's

U3	U6	U7	U8	U9	U10	U11	U12	U13	U14	U16
C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19

IC Power & Ground Connections

IC	+5	+12	−12	GND	No Connect
U1		14	1	7	
U2	14			7	2,5,9,12
U3	14			7	
U4		14	1	7	
U5	14			7	2,5,9,12
U6	28			14	
U7	28			14	
U8	24			12	
U9	20			10	
U10	40			20	
U11	28			14	
U12	28			14	
U13	40			20	
U14	40			20	
U16	20			10	

NSC800 Designer Kit Suggested Chip Layout for Wire-Wrapped or PC Board



TL/C/10435-6

Wire-Wrap Area

NSC800 Designer Kit: Component List

Description	Ref.	Qty.
1. IC'S		
NSC800N-4	U1	1
NSC810AN-4	U8, U9	2
NSC858N	U5, 6	2
MM74HC154N	U2	1
MM74HC373N	U3	1
MM74HC245N	U7	1
MM74HC00N	U11	1
DS14C88N	U12, U14	2
DS14C89AN	U13, U15	2
NMC27C64Q250	U4	1
NMC6164AN-70	U10	1
2. DISCREET COMPONENTS		
10 kΩ 5%	R3–R6	4
10k SIP (9 Resistors)	RP1, RP2	2
1 MΩ 5%	R1, R7	2
1N914 Diode	D1	1
33 μF	C1–C3	3
0.01 μF	C4–C15	12
10 μF	C18	1
22 pF	C19	1
47 pF	C20	1
1.8432 MHz Crystal—		
AT cut, Parallel	Y2	1
3. MISCELLANEOUS HARDWARE		
DB25—Female Connector	J1, J2	2
Wire-Wrap Headers:		
2 x 26 Pin	J3	1
2 x 25 Pin	J4	1
2 x 7 Pin	J5	1
MOLEX 4-Pin SIP (Male)	J6	1
8-Position DIP Switch	DS1	1
Push Switch (Momentary)	S1	1
SPDT (On-Momentary)	S2	1

4. NSC800 OSCILLATOR NETWORK: Component Values for Various Crystal Speeds. Y1 = AT Cut, Parallel Resonant Crystal.

Y1 (MHz)	R1	R2	C1	C2
8.00	1M	0	22 pF	33 pF
4.00	1M	0	33 pF	47 pF
2.00	1M	1.5k	68 pF	100 pF
1.00	1M	1.5k	100 pF	150 pF

APPENDIX H. DEVELOPMENT SUPPORT PRODUCTS FOR THE NSC800

The following is a list of vendors who offer products that support hardware and software development for NSC800 microprocessors. The companies, products, and approximate price (where available) for each are listed.

Note to Vendors: If your company offers a related product that is not included on this list, let us tell our customers about your products. Send your company name, phone number, product information, and price ranges to the following address:

NSC800 Applications Engineering
Mail Stop E2-55
National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090

Company	Product	Approximate Price
Applied Microsystems Corporation (800) 426-3925	NS800 Emulator EM-800	
	—16k RAM stand-alone	$2995
	—64k RAM + PC driver	$3995
Avocet Systems, Inc. (207) 236-8227	Z80 "C" cross-compiler	
	—MS-DOS host	$895
	Z80 "C" native compiler	$295
	Z80 macro assembler	
	—MS-DOS host	$349
Digital Research, Inc. (408) 649-3896	Z80 macro assembler	
	—MS-DOS host	$200
Ecosoft, Inc (317) 255-6476	Z80 "C" compiler	
	—Native (CP/M host)	$100
Huntsville Microsystems, Inc. (205) 881-6000	NSC800 emulator IDP-800	
	—Emulator, power supply,	
	x-assembler (MS-DOS)	$3195
Manx Software Systems (800) 221-0440	Aztec C cross-compiler	
	—MS-DOS	$750
	—VAX/ULTRIX	$3000
	—CP/M-80	$349
Microcomputer Tools (415) 825-4200	Z80 macro-assembler	
	—MS-DOS host	$150
Northwest Instrument Systems, Inc. (800) 547-4445	Microtek MICE 2+ emulators	
	64k RAM + PC-based	
	debugger software	$5950
Orion Instruments (415) 361-8883	Unilab 8620 Analyzer/	$3500 to
	Emulator (PC-based)	$5000
Softech Microsystems (718) 851-3100	"C" cross-compiler	
	—MS-DOS host	$300
	Z80 macro assembler	
	—MS-DOS host	$80
2500 A.D. Software, Inc. (719) 395-8683	Z80 C x-compiler and	
	macro assembler (MS-DOS)	$500
	Z80 macro assembler only	$200

NSC800 Applications System: NS16550A UART 8237A DMA Controller Interface

National Semiconductor
Application Note 613
Greg DeJager

INTRODUCTION

This document describes a system which utilizes the DMA control signals and FIFOs of the NS16550A UART in conjunction with the 8237A DMA Controller and the NSC800 CPU. Included is an operation overview section and descriptions of hardware requirements, software used and system timing considerations.

OPERATION OVERVIEW

The system used is an NSC800 Application System with all software written in NSC800 assembly code. DMA request signals from the 16550A to an 8237A DMA controller cause direct data transfers to be made from on board RAM to the 16550A transmitter FIFO or from the receiver FIFO to RAM. Simultaneous memory and I/O read/write signals from the 8237A produce single cycle data transfers between the UART and memory. This results in high speed file upload and download operations independent of the system CPU.

HARDWARE REQUIREMENTS

The system requires an NSC800 based board running a ROM based MON800 (Version 1.0) monitor program, at least 8K RAM, two RS-232 serial ports (one controlled by an NSC858 UART and the other by an NS16550A), an NSC810A RAM-I/O-Timer, an 8237A DMA controller and various interface logic components.

DMA Request/Bus Access Control

The $\overline{\text{RXRDY}}$ and $\overline{\text{TXRDY}}$ DMA signals from the 16550A are connected to 8237A device request inputs DREQ0 and DREQ2 and indicate a full receiver FIFO and empty transmitter FIFO respectively. Upon receiving a device request, the 8237A asserts a hold request signal which is connected to the NSC800's bus request input. After completing execution of its current instruction, the CPU will TRI-STATE® its buses and asserts a bus acknowledge signal. The 8237 receives this signal in its hold acknowledge input and immediately performs its programmed DMA operation.

Read/Write/Chip Select

An LS257 Data Selector generates separate memory and I/O read and write signals from the CPU read and write outputs. AEN, a signal from the 8237A asserted during DMA cycles, is connected to the output enable pin of the LS257. Thus during DMA cycles, the LS257 will TRI-STATE its outputs and all memory and I/O read/write strobes are generated by the 8237A. The chip select signal for the 16550A is produced by ORing 8237A device acknowledge signals (DACK0 and DACK2) with the output of the system address decoder. This allows selection of the UART by the CPU during normal bus cycles and by the 8237A during DMA cycles. The chip selects for the 8237A and system RAM are generated by the system address decoder. Note that there are two 8k x 8 RAM chips in the system schematics. The extra RAM was used during software development, it is not necessary in a minimal system.

Address/Data Buses

The 8237A provides a low address bus which is common with the system's lower address bus. The 8237A also has an 8-bit multiplexed bus which outputs upper address bits and data. This bus is common with the system data bus. An LS373 is used to latch the upper address byte onto the system upper address bus. The AEN signal is used to enable the latch and disable the system's LS373 during DMA cycles.

During DMA cycles, the 8237A outputs a 16-bit memory address but no I/O address. Register and FIFO addresses for the 16550A are generated instead by an LS157 multiplexer which outputs the receiver or transmitter FIFO address (000) during DMA cycles and system address signals A2–A0 during CPU bus cycles.

Hardware Interrupts

NSC800 interrupts $\overline{\text{RSTA}}$ and $\overline{\text{RSTB}}$ are used to facilitate and terminate the file upload and download processes. The hardware connections and purpose of the interrupts is explained in the SOFTWARE DESCRIPTION AND DMA OPERATIONS section of this document.

SOFTWARE DESCRIPTION AND DMA OPERATIONS

The two programs included in this package, DMAWR.ASM and DMARD.ASM, are NSC800 assembly listings for file download and upload operations respectively. The following includes a description of serial port initialization and full descriptions of DMAWR.ASM and DMARD.ASM programs and their associated interrupt service routines.

Serial Port Initialization

Both programs initialize the NS16550A to 9600 baud, 8 bits, 1 stop, no parity. Modem control outputs $\overline{\text{RTS}}$ and $\overline{\text{DSR}}$ are asserted to support communication with a terminal. FIFO's are enabled and programmed for DMA mode 1. In DMAWR.ASM, the receiver FIFO interrupt trigger level is set to 14 bytes in order to maximize the efficiency of the download operation.

DMAWR.ASM Description

DMAWR.ASM performs a file download operation by transferring bytes received in the serial port to RAM. After initializing the system, the CPU enters a NOP loop and, unless servicing interrupts, remains there until the file transfer is complete.

The program first prompts the user for a destination address for the incoming file and programs the 8237A with this 16-bit value. The 8237A is then programmed to accept a device service request ($\overline{\text{RXRDY}}$) through channel 0 and to perform I/O read/memory write transfers in the demand mode. In this mode, the controller executes single byte transfers as long as the device request remains asserted. In this system, the controller begins reading bytes from the FIFO when the trigger level is reached ($\overline{\text{RXRDY}}$ goes active) and stops reading when the FIFO empties ($\overline{\text{RXRDY}}$ goes inactive).

Throughout the file transfer process, the DMA controller automatically empties the receiver FIFO into RAM each time the trigger level is reached.

The end of a file is indicated to the CPU by an NSC810 timer which produces a timeout signal which interrupts the CPU through $\overline{RSTB}$ if no characters have been received for a 2.5 second period of time. The 810A is programmed as a restartable timer with the greatest possible timeout period (maximum prescaler (64) and modulus (FFFFH). The timer counts down only when $\overline{RXRDY}$ is inactive and resets itself each time $\overline{RXRDY}$ becomes active. Thus the timer output is asserted if $\overline{RXRDY}$ remains inactive (no characters received into serial port) for more than 2.5 seconds.

DMAWR.ASM must be started before any characters arrive in the serial port. Since the user must start transmission of a file on some other machine, there is an indeterminable amount of time before the first characters are received. The $\overline{RXRDY}$ signal remains inactive during this time and until the first 14 characters arrive and $\overline{RXRDY}$ is asserted, the 810A will interrupt the CPU every 2.5 seconds.

$\overline{RSTB}$ Service Routine

The interrupt service routine handles this "false" interrupt by first checking if any characters have been received yet. If not, the CPU reinitializes the timer and the interrupt and exits the routine. The maximum possible timeout period of 2.5 seconds is used to minimize these initial false interrupts. When a "true" interrupt has been received (bytes had been received but none have arrived during the last 2.5 seconds signifying the end of the file), the interrupt service routine stops the timer, outputs a termination message and exits to the MON800 monitor.

DMARD.ASM Description

DMARD.ASM performs a file upload operation by transferring bytes from RAM to the serial port transmit FIFO. After initializing the system, the CPU enters a NOP loop and, unless servicing interrupts, remains there until the file transfer is complete.

The program first prompts the user for the starting and ending address of the data to be transmitted. The 8237A 16-bit address register is programmed with the starting address. The total number of bytes in the file is also calculated. This number is used by the $\overline{RSTA}$ interrupt service routine. The 8237A is then set up to accept a device service request ($\overline{TXRDY}$) through channel 2 and to perform memory read/I/O write transfers in the block mode. In this mode, a device request causes the controller to transfer a block of data with the block size being programmed into its transfer count register. With exception of the first and last block transfer of the file being uploaded, a block size of 16 is programmed into the count register so that an active $\overline{TXRDY}$ signal (empty FIFO) will result in the 8237A filling the Transmit FIFO. The $\overline{RSTA}$ Service Routine section describes how the count for each block is determined and how the end of the file is recognized.

It is necessary to use block mode because the NS16550A does not deassert $\overline{TXRDY}$ quickly enough to stop a demand mode transfer when the transmitter FIFO becomes full. $\overline{TXRDY}$ goes inactive upon receiving the *trailing* edge of the write pulse of the byte which fills the FIFO. This is to late stop the controller from performing one additional transfer which will overflow the FIFO. This is different from $\overline{RXRDY}$ which goes inactive upon receiving the *leading* edge of the final read pulse allowing enough time to prevent another transfer. Because the 8237A must have its transfer count register reinitialized after each block transfer, the CPU must be interrupted each time the transmit FIFO is refilled. This is done by latching the End of Process (EOP) signal into the $\overline{RSTA}$ interrupt of the NSC800. The EOP is generated by the 8237A at the end of a block transfer.

$\overline{RSTA}$ Service Routine

The $\overline{RSTA}$ interrupt service routine performs three operations: calculation of the size of the next block to be transferred (16 or less), programming the 8237A transfer count register with this value and reenabling the $\overline{RSTA}$ interrupt. The routine maintains a count of the number of bytes of the file already transferred. It uses this number along with the total number of bytes in the file (calculated in the main program) to determine the number of remaining bytes. If greater or equal to 16, a block size of 16 is programmed into the 8237A transfer count register. If there is less than 16 bytes left, the number remaining is programmed into the count register. When there are 0 bytes left, the upload operation is complete and the program exits to the MON800 monitor.

Note: The main program initially programs the count register to 1. This is done because it is assumed that a file contains at least one byte but that the file could be less than 16 bytes. Thus upon starting the file transfer, a one byte block is transferred and, if there are at least 16 more bytes in the file, the $\overline{RSTA}$ service routine programs the next transfer to be 16 bytes.

TIMING CONSIDERATIONS AND OPERATING SPEED

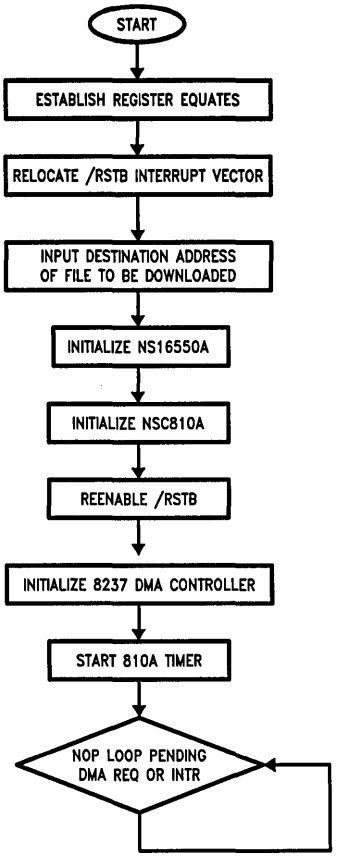
DMARD.ASM

The 8237A used in this application has a maximum operating frequency of 5 MHz (8237A-5). The 8237A in normal timing mode can transfer characters from 120 ns RAM to the 16550A FIFO at this maximum speed. The 8237A can also operate in a compressed timing mode in which bytes are transferred to sequential addresses in two clock periods instead of three. This is done by shortening the read pulse from two clocks to one. However, at 5 MHz the access time for the RAM is too long to meet the data set up time (t_{dS}) for writing to the 16550A. The maximum operating frequency which meets RAM read/16550A write timing specs is calculated to be 3.57 MHz. At this lower frequency, character transfers still occur faster in the compressed mode: 560 ns per character (two clocks at 3.57 MHz) for compressed mode versus 600 ns (three clocks at 5 MHz) for normal timing. Since the maximum speed of the NSC800 CPU used is 4 MHz, the system was not tested at 5 MHz. It was tested at 4 MHz, however, and both normal and compressed timing modes did run properly, even though data set up time for the 16550A was not met in compressed mode.

DMAWR.ASM

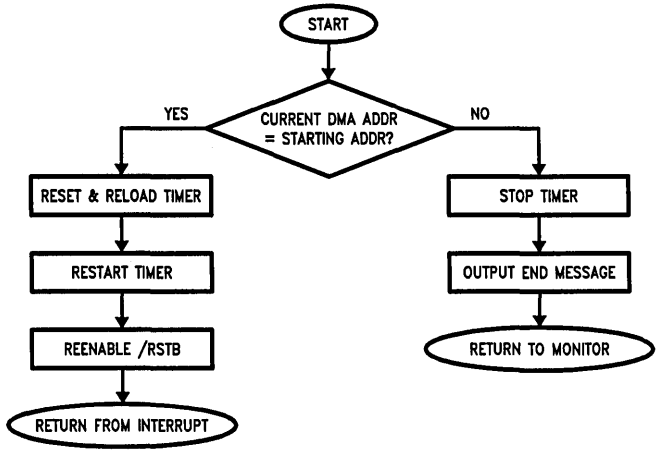
When operating in the demand mode, the 8237A requires that its device request input, $\overline{RXRDY}$ in this case, be deasserted before the final clock period of the last character's transfer cycle. However, when the 8237A reads the last character in the receiver FIFO, there is a delay before the 16550A deasserts $\overline{RXRDY}$. This creates the bottleneck for the 16550A to memory transfer process. The maximum operating frequency which will allow $\overline{RXRDY}$ to be deasserted in time is calculated to be 3.125 MHz. However, the system was found to run at 4 MHz without failure as the $\overline{RXRDY}$ signal was found to go inactive sooner than is specified.

DMAWR.ASM Flowchart



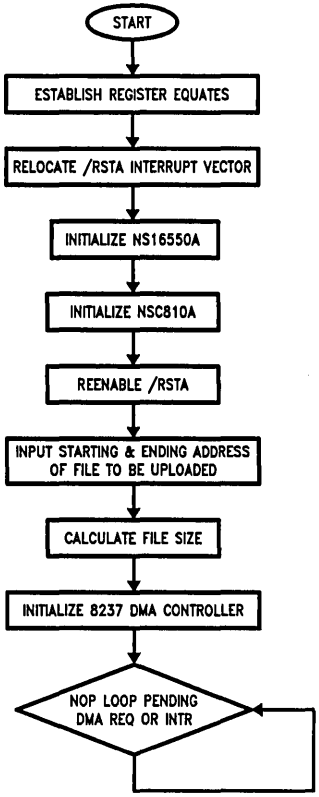
TL/C/10436-1

$\overline{\text{RSTB}}$ Service Routine Flowchart



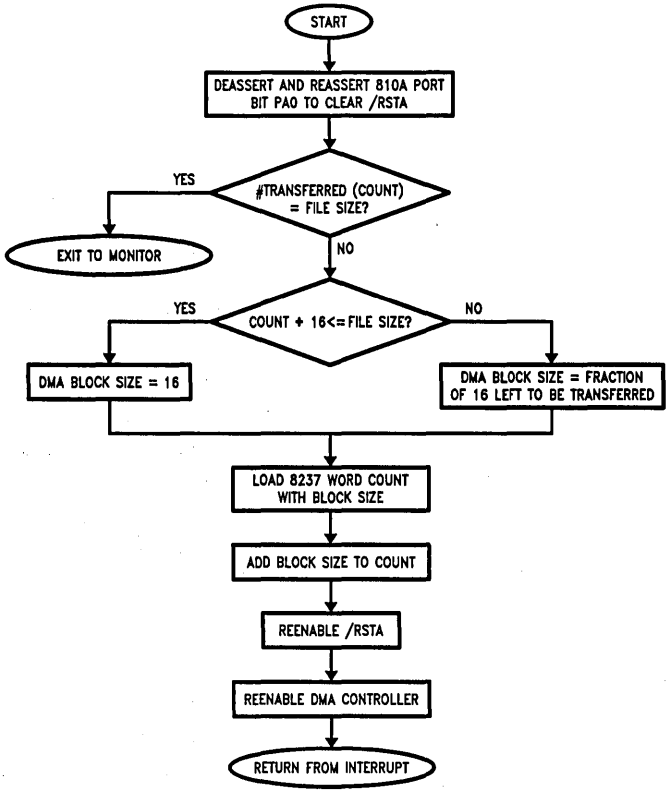
TL/C/10436-2

DMARD. ASM Flowchart

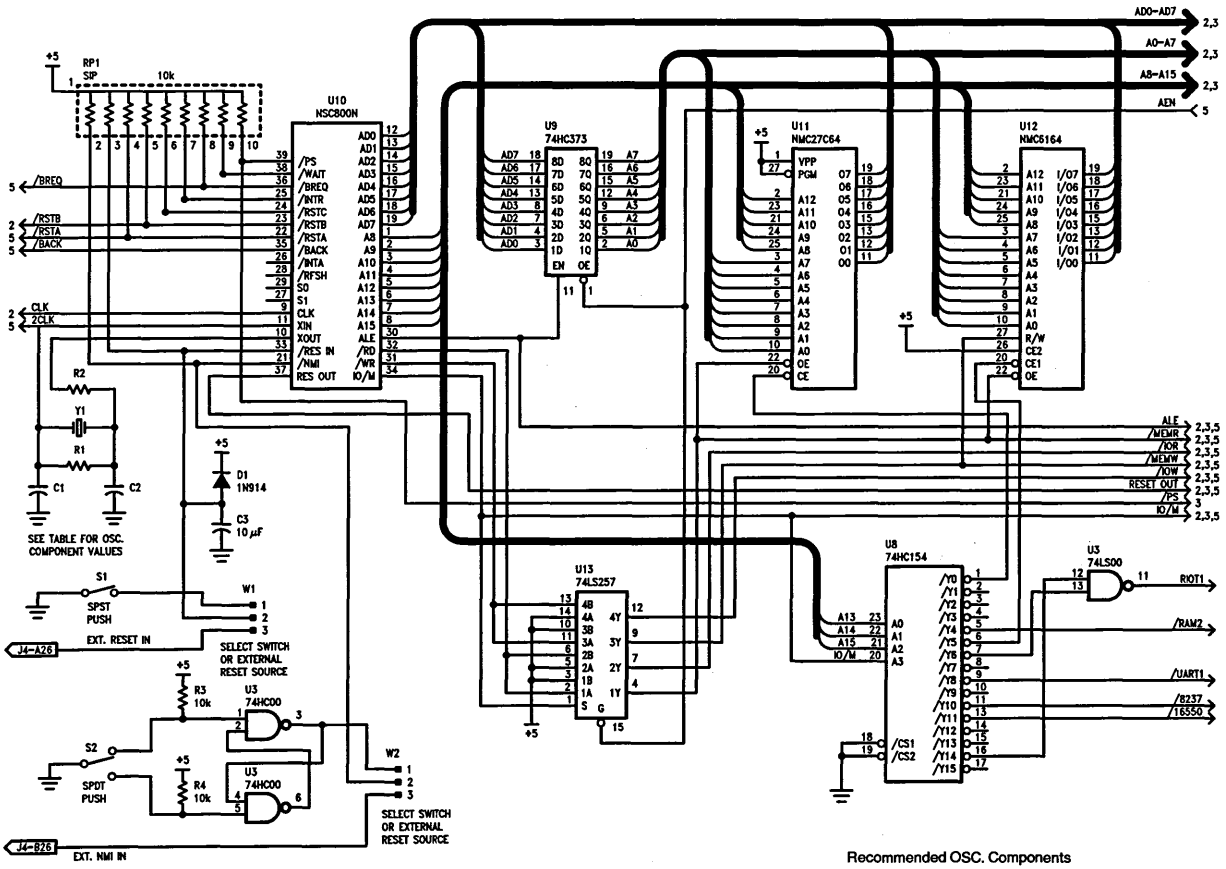


TL/C/10436–3

$\overline{\text{RSTA}}$ Service Routine Flowchart



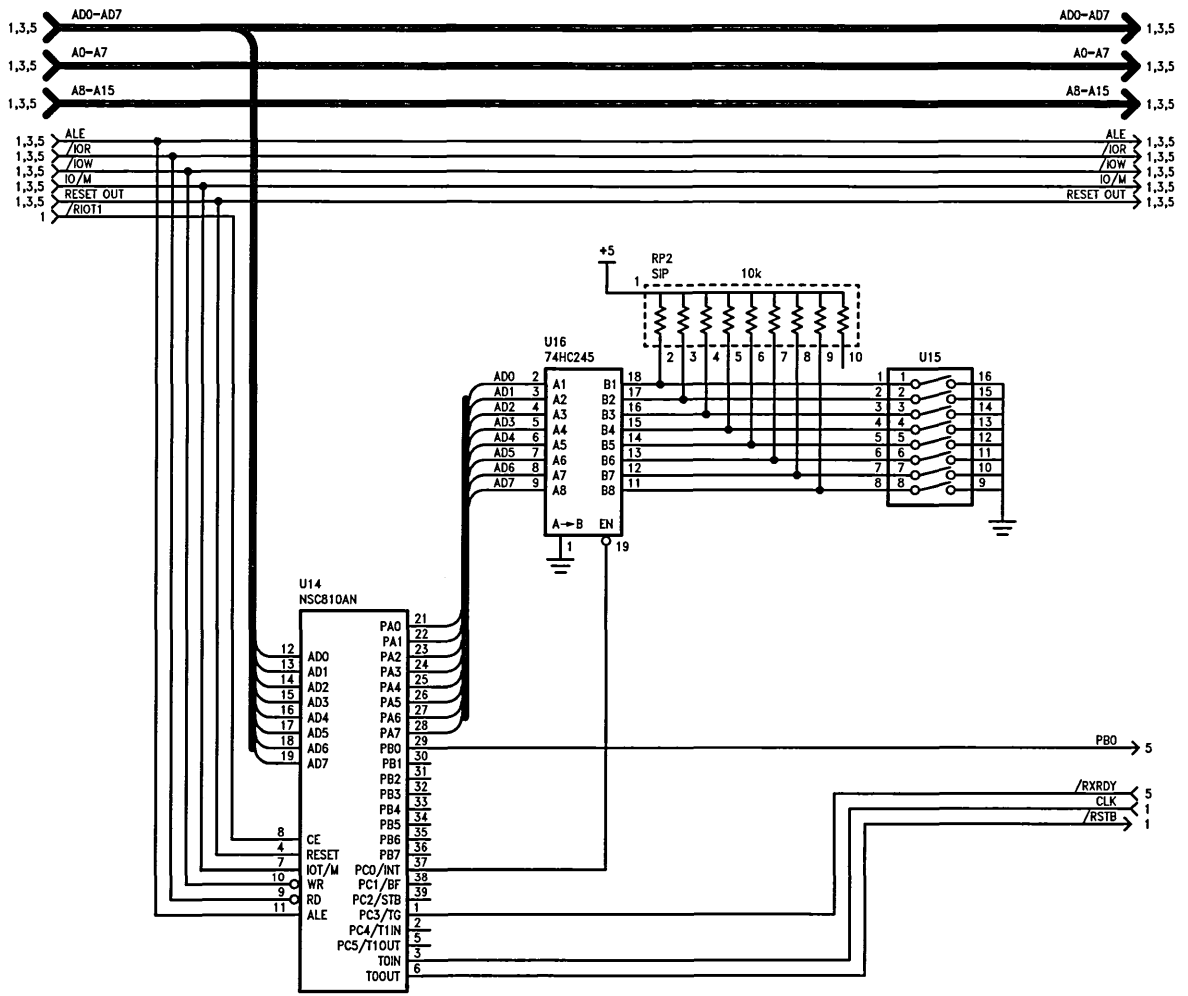
TL/C/10436–4

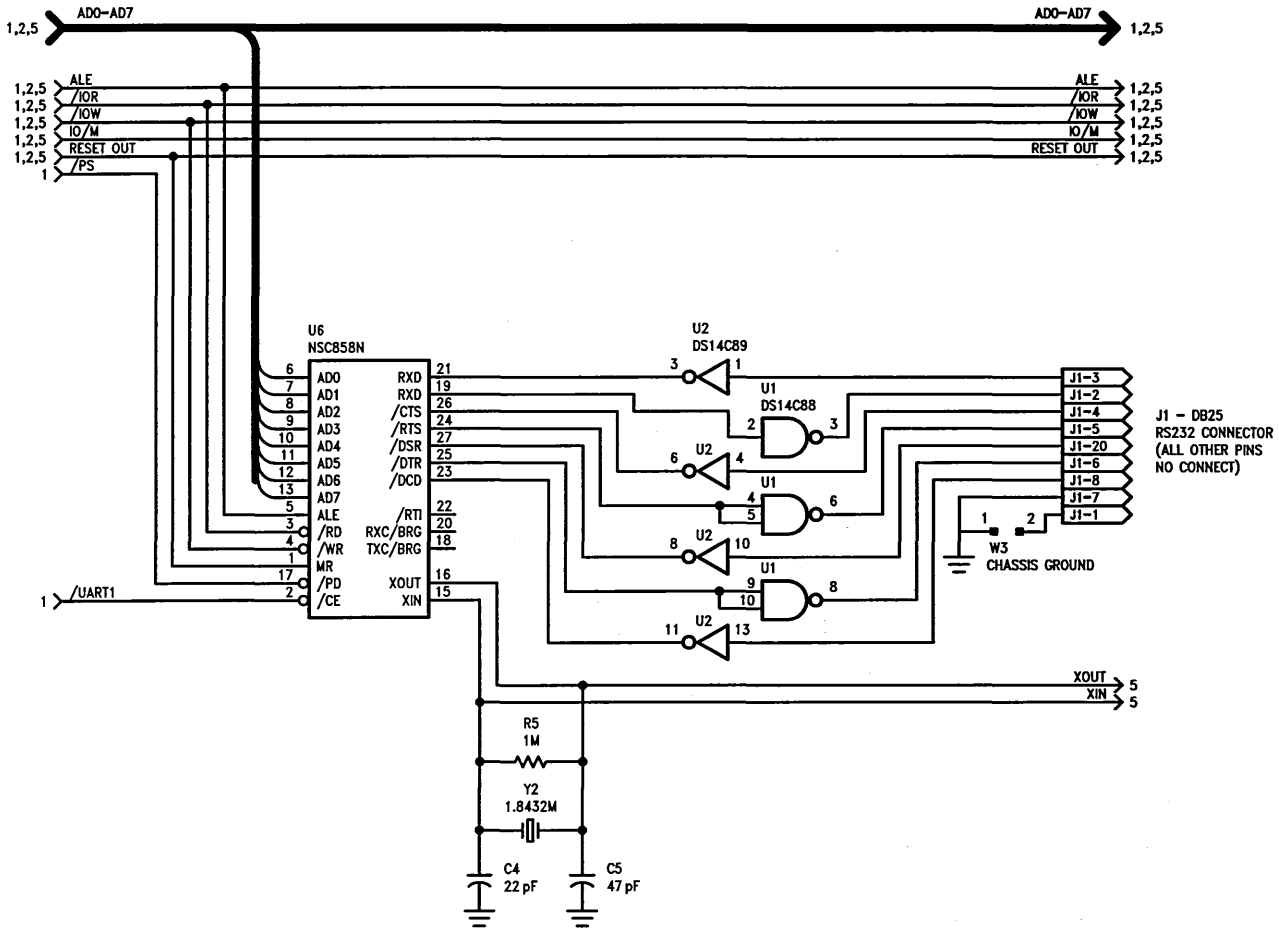


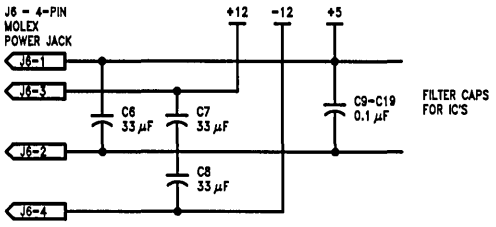
Recommended OSC. Components

XTAL (MHz)	R1	R2	C1	C2
1.00	1M	1.5k	100 pF	150 pF
2.00	1M	470	68 pF	100 pF
4.00	1M	0	33 pF	47 pF
8.00	1M	0	22 pF	33 pF

TL/C/10436–5


AD0–AD7
A0–A7
A8–A15
ALE
/IOR
/IOW
IO/M
RESET OUT
/RIOT1
+5
RP2
SIP
10k
U16
74HC245
U15
U14
NSC810AN
PB0
/RXRDY
CLK
/RSTB
TL/C/10436-6


AD0–AD7
1,2,5
AD0–AD7
1,2,5
ALE
/IOR
/IOW
IO/M
RESET OUT
/PS
1
/UART1
U6
NSC858N
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
ALE
/RD
/WR
MR
/PD
/CE
RXD
RXD
/CTS
/RTS
/DSR
/DTR
/DCD
/RTI
RXC/BRG
TXC/BRG
XOUT
XIN
U2
DS14C89
U1
DS14C88
J1-3
J1-2
J1-4
J1-5
J1-20
J1-6
J1-8
J1-7
J1-1
J1 – DB25
RS232 CONNECTOR
(ALL OTHER PINS
NO CONNECT)
W3
CHASSIS GROUND
XOUT
XIN
R5
1M
Y2
1.8432M
C4
22 pF
C5
47 pF
TL/C/10436–7

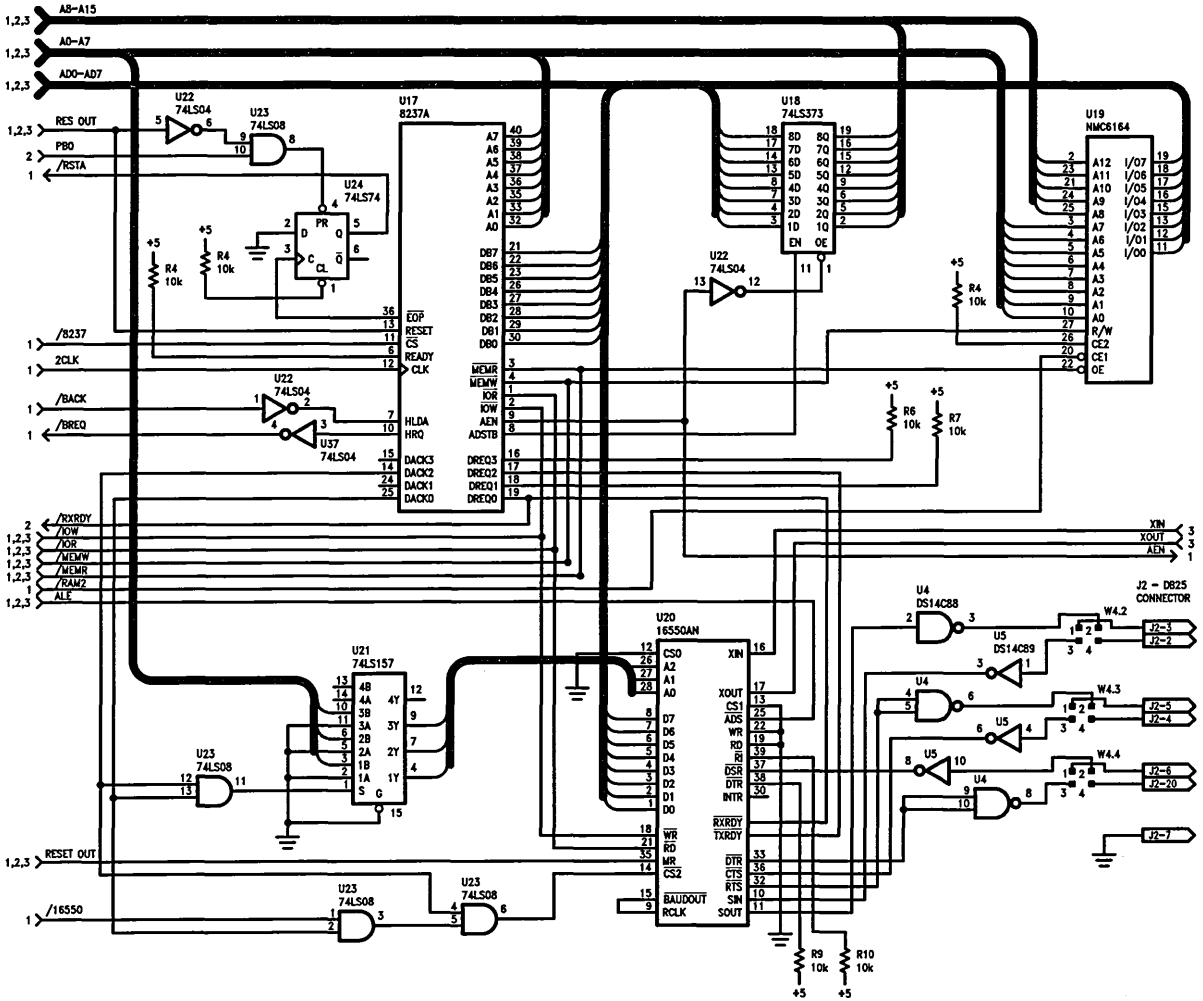


TL/C/10436–8

Place Filter Caps Near IC's
U3 U6 U7 U8 U9 U10 U11 U12 U13 U14 U16 U17 U19 U20
C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20 C21 C22

IC Power and Ground Connections

IC	+5	+12	−12	GND	No Connect
U1		14	1	7	
U2	14			7	2, 5, 9, 12
U3	14			7	
U4		14	1	7	
U5	14			7	2, 5, 9, 12
U6	28			14	
U8	24			12	
U9	20			10	
U10	40			20	
U11	28			14	
U12	28			14	
U13	16			8	
U14	40			20	
U16	20			10	
U17	31			20	
U18	20			10	
U19	28			14	
U20	40			20	
U21	16			8	
U22	14			7	
U23	14			7	


A8–A15
A0–A7
AD0–AD7
RES OUT
PB0
/RSTA
U22 74LS04
U23 74LS08
U24 74LS74
U17 8237A
U18 74LS373
U19 NMC6164
R4 10k
/8237
2CLK
/BACK
/BREQ
U37 74LS04
R6 10k
R7 10k
/RXRDY
/IOW
/IOR
/MEMW
/MEMR
/RAM2
ALE
XIN
XOUT
AEN
U21 74LS157
U20 16550AN
U4 DS14C88
U5 DS14C89
J2 – DB25 CONNECTOR
W4.2
W4.3
W4.4
RESET OUT
/16550
R9 10k
R10 10k

TL/C/10436-9

```
; DMARD - MEMORY TO SERIAL PORT UPLOAD ROUTINE FOR NSC800 DESIGNER KIT
; LAST REVISION: 12/1/88  BY G.D.
;
; THIS SOFTWARE UTILIZES AN INTERFACE BETWEEN THE NSC800 MICROPROCESSOR,
; 8237 DMA CONTROLLER, NS16550A UART AND RAM MEMORY.  THE USER IS PROMPTED
; FOR THE STARTING AND ENDING ADDRESS OF THE MEMORY BLOCK TO BE UPLOADED.
; A TRANSMITTER FIFO EMPTY INDICATION (/TXRDY) FROM THE 16550 UART CAUSES
; THE 8237 TO ASSUME CONTROL OF THE BUS AND TRANSFER DATA FROM RAM TO THE
; 16550 TRANSMITTER FIFO.  THE TRANSFERS CONTINUE UNTIL THE USER ENTERED
; ENDING ADDRESS IS REACHED.  THE SOFTWARE INCLUDES INITIALIZATION ROUTINES
; FOR THE 16550, 8237, NSC810 PARALLEL PORT AND NSC800 INTERRUPTS.  AN
; INTERRUPT SERVICE ROUTINE RELOADS THE 8237 AFTER EACH FILL OF THE 16550
; FIFO.  THE CPU PERFORMS A NOP LOOP BETWEEN DMA ACCESSES AND INTERRUPT
; SERVICES.

; HARDWARE EQUATES
UART    EQU     60H     ;PORT ADDRESS OF 16550 UART
DLL     EQU     UART+0  ;BAUD DIVISOR LSB
DLM     EQU     UART+1  ;BAUD DIVISOR MSB
FCR     EQU     UART+2  ;FIFO CONTROL
LCR     EQU     UART+3  ;LINE CONTROL
MCR     EQU     UART+4  ;MODE CONTROL

DMA     EQU     40H     ;PORT ADDRESS OF 8237 DMA CONTROLLER
CH2ADDR EQU     DMA+4   ;CHANNEL 2 STARTING MEMORY ADDRESS
CH2WRD  EQU     DMA+5   ;CHANNEL 2 CURRENT WORD
COMM    EQU     DMA+8   ;COMMAND
MASK    EQU     DMA+10  ;CHANNEL MASK
MODE    EQU     DMA+11  ;MODE

PORT    EQU     C0H     ;PORT ADDRESS OF 810A PARALLEL PORT
MDR     EQU     PORT+7  ;PORT MODE
PBDDR   EQU     PORT+5  ;PORT B DIRECTION
PBCLR   EQU     PORT+9  ;CLEAR PORT BITS
PBSET   EQU     PORT+13 ;SET PORT BITS

        ORG 00H

;***** HARDWARE RESTART A (/RSTA) SERVICE ROUTINE VECTOR RELOCATION *******
        LD A,C3H        ;OPCODE FOR 'JP' INSTRUCTION
        LD (BF9BH),A    ;PLACE OPCODE AT MONITOR LOCATION
        LD HL,BC00H     ;ADDRESS FOR NEW ROUTINE
        LD (BF9CH),HL   ;LOAD NEW ADDRESS

;********************** 16550 INITIALIZATION ***************************
        LD A,80H
        OUT (LCR),A     ;DLAB
        LD A,0CH
        OUT (DLL),A     ;9600 BAUD
        LD A,00H
        OUT (DLM),A
        LD A,03H
        OUT (LCR),A     ;8,1,N
        LD A,01
        OUT (FCR),A     ;ENABLE FIFOS
        LD A,03H
        OUT (FCR),A     ;RESET FIFOS
        LD A,09H
        OUT (FCR),A     ;SET DMA MODE TO 1
        LD A,03H
        OUT (MCR),A     ;SET /RTS AND /DSR ACTIVE (FOR USE BY TERMINAL)
```

TL/C/10436–10

```
;************* 810A INITIALIZATION FOR INTERRUPT LATCH CONTROL *************
        LD A,0
        OUT (MDR),A      ;PORT B - MODE 0 (BASIC I/O)
        LD A,1
        OUT (PBDDR),A    ;SET PB0 DIRECTION AS OUTPUT
        OUT (PBSET),A    ;SET PB0 HIGH (/RSTA INACTIVE)

;****************** INITIALIZE CPU INTERRUPT /RSTA ************************
        EI               ;ENABLE CPU INTERRUPTS
        LD A,08H
        OUT (BBH),A      ;UNMASK /RSTA INTERRUPT

;******* INPUT STARTING AND ENDING ADDRESS OF BLOCK TO BE UPLOADED ********
        LD HL,MESG0      ;PROMPT FOR STARTING ADDRESS
        CALL GETWORD     ;RETURNS STARTING ADDRESS IN DE
        PUSH DE
        LD HL,MESG1      ;PROMPT FOR ENDING ADDRESS
        CALL GETWORD     ;RETURNS ENDING ADDRESS
        PUSH DE
        POP HL           ;HL CONTAINS ENDING ADDRESS
        POP DE           ;DE CONTAINS STARTING ADDRESS

        LD A,E
        OUT (CH2ADDR),A ;LOAD 8237 WITH LSB OF STARTING ADDRESS
        LD A,D
        OUT (CH2ADDR),A ;LOAD MSB OF STARTING ADDRESS

        XOR A            ;CLEAR CARRY
        SBC HL,DE        ;SUBTR. ADDRESSES TO GET # OF BYTES TO BE TRANS
        PUSH HL
        POP DE
        INC DE           ;DE REGISTER PAIR CONTAINS BLOCK SIZE

;************************** INITIALIZE 8237 ******************************
        LD A,48H
        OUT (COMM),A     ;COMMAND REG: MEM TO MEM DISABLE, CONTROLLER ENABLE,
                         ;COMPRESSED TIMING, FIXED PRIORITY, LATE WRITE,
                         ;ACTIVE LOW DREQ, ACTIVE LOW DACK
        LD A,8AH
        OUT (MODE),A     ;MODE REG: CH2, READ TRANSFERS (MEM TO PORT),
                         ;NO AUTOINITIALIZATION, INC ADDRESS, BLOCK MODE

        LD A,00H         ;LSB OF BYTE COUNT OF 8237
        OUT (CH2WRD),A   ;START WITH 1 BYTE BLOCK
        OUT (CH2WRD),A   ;MSB
        LD H,0
        LD L,1           ;CURRENT COUNT = 1
        LD B,00
        LD C,10H         ;BC = 16 (CONSTANT)
        LD A,02H
        OUT (MASK),A     ;UNMASK CHANNEL 2


;************************ NOP WAIT LOOP FOR CPU ******************************
WAIT    NOP              ;NOP LOOP WAITING FOR DMA ACCESSES AND INTERRUPTS
        JR WAIT

;********************** PROCEDURE GETWORD ************************************
GETWORD LD A,0           ;ROUTINE INPUTS WORD FROM PORT
        RST 10H          ;OUTPUT PROMPT FOR ADDRESS
        CALL GETBYT
        LD D,C           ;MSB OF ADDRESS

        CALL GETBYT
        LD E,C           ;LSB OF ADDRESS
```

TL/C/10436–11

```
        RET

;************************ PROCEDURE GETBYTE *****************************
GETBYT  LD C,0          ;ROUTINE TO INPUT BYTE FROM PORT
        CALL INNIB
        SLA A
        SLA A
        SLA A
        SLA A
        LD C,A          ;MOST SIGNIFICANT NIBBLE
        CALL INNIB
        OR C            ;LEAST SIGNIFICANT NIBBLE
        LD C,A
        RET

;************************ PROCEDURE INNIB *******************************
INNIB   LD A,04H        ;ROUTINE TO INPUT, ECHO AND CONVERT NIBBLE
        RST 10H         ;INPUT HEX BYTE FROM PORT
        LD B,A
        LD A,3
        RST 10H         ;ECHO BYTE TO SCREEN
        LD A,06H
        RST 10H         ;CONVERT TO HEX NIBBLE (ACC = 0x)
        RET


;************* /RSTA INTERRUPT SERVICE ROUTINE ****************
        ORG 100H        ;START OF INTERRUPT ROUTINE

        LD A,1
        OUT (PBCLR),A   ;CLEAR PB0 TO SET /RSTA LATCH OUTPUT
        OUT (PBSET),A   ;SET PB0 AGAIN
        LD A,07H
        RST 10H         ;16 BIT COMPARE OF HL (# TRANSFERRED)
                        ;AND DE (TOTAL TO BE TRANSF)
        CP F0H          ;COMPARE ROUTINE RETURNS ACC=F0H IF HL=DE
        JP Z,DONE

        PUSH HL         ;SAVE COUNT
        ADD HL,BC       ;ADD 16 TO COUNT
        LD A,07H
        RST 10H         ;16 BIT COMPARE ROUTINE
        CP FFH
        JP Z,NEED_16    ;COUNT+16 IS LESS THAN TOTAL SO SEND BLOCK OF 16
                        ;ELSE DETERMINE FRACTION OF 16 TO BE SENT
        XOR A           ;CLEAR CARRY
        SBC HL,DE       ;LOAD 8237 WITH BYTE COUNT REMAINING (COUNT < 16)
        LD B,L
        LD A,10H
        SUB A,B
        DEC A           ;BYTE COUNT=16-(HL-DE)-1 (COUNT MUST BE # TRANSF.- 1)
        JP CONT

NEED_16 LD A,0FH
CONT    POP HL
        LD C,A          ;LD SIZE OF BLOCK TO BE TRANSMITTED INTO BC
        INC C
        LD B,0
        ADC HL,BC       ;ADD BLOCK SIZE TO TOTAL
        OUT (CH2WRD),A  ;LSB
        LD A,0
        OUT (CH2WRD),A  ;MSB

        EI              ;REENABLE CPU INTERRUPTS
        LD A,02H
        OUT (MASK),A    ;UNMASK CHANNEL 2
        RETI

DONE    RST 08H         ;EXIT MONITOR

MESG0   DB 13,10,'ENTER STARTING ADDR OF MEMORY BLOCK TO BE UPLOADED:',13,10,0
MESG1   DB 13,10,'ENTER ENDING ADDR OF MEMORY BLOCK:',13,10,0
```

TL/C/10436–12

```
        TITLE DMA WRITE TRANSFER
        LIST ON
        PL 58

;*****************************************************************************
;DMAWR - SERIAL PORT TO MEMORY DOWNLOAD ROUTINE FOR NSC800 DESIGNER KIT       *
;LAST REVISION: 12/2/88  BY G.D.                                              *
;                                                                            *
;THIS SOFTWARE UTILIZES AN INTERFACE BETWEEN THE NSC800 MICROPROCESSOR,       *
;8237 DMA CONTROLLER, NS16550A UART, NSC810A TIMER AND RAM MEMORY.  DATA      *
;RECEIVED BY THE 16550 UART FILLS ITS INTERNAL FIFO UNTIL A PROGRAMMED TRIGGER*
;LEVEL IS REACHED.  THIS ACTIVATES /RXRDY, A DMA REQUEST SIGNAL TO THE 8237.  *
;THE 8237 THEN ASSUMES CONTROL OF THE SYSTEM BUS AND EMPTIES THE FIFO BY      *
;TRANSFERRING DATA DIRECTLY TO RAM.  THE DOWNLOAD PROCESS IS TERMINATED BY A  *
;TIMEOUT SIGNAL FROM THE NSC810A.                                             *
;*****************************************************************************

; HARDWARE EQUATES
UART    EQU     60H     ;PORT ADDRESS OF 16550 UART
DLL     EQU     UART+0  ;BAUD DIVISOR LSB
DLM     EQU     UART+1  ;BAUD DIVISOR MSB
FCR     EQU     UART+2  ;FIFO CONTROL
LCR     EQU     UART+3  ;LINE CONTROL
MCR     EQU     UART+4  ;MODEM CONTROL

DMA     EQU     40H     ;PORT ADDRESS OF 8237 DMA CONTROLLER
CH0ADDR EQU     DMA+0   ;CHANNEL 0 STARTING MEMORY ADDRESS
CH0WRD  EQU     DMA+1   ;CHANNEL 0 CURRENT WORD COUNT
COMM    EQU     DMA+8   ;COMMAND REGISTER
MASK    EQU     DMA+10  ;CHANNEL MASK REGISTER
MODE    EQU     DMA+11  ;CONTROLLER MODE REGISTER

TIMR    EQU     C0H     ;PORT ADDRESS OF 810A TIMER
LMOD    EQU     TIMR+16 ;LSB OF MODULUS
HMOD    EQU     TIMR+17 ;MSB OF MODULUS
TMRMOD  EQU     TIMR+24 ;TIMER 0 MODE
START   EQU     TIMR+21 ;TIMER 0 START
STOP    EQU     TIMR+20 ;TIMER 0 STOP
DDRC    EQU     TIMR+6  ;PORT C DIRECTION REGISTER


        ORG 00H

;******** HARDWARE RESTART B (/RSTB) SERVICE ROUTINE VECTOR RELOCATION *******
        LD A,C3H        ;OPCODE FOR 'JP' INSTRUCTION
        LD (BF95H),A    ;PLACE OPCODE IN MONITOR JUMP TABLE
        LD HL,BE00H     ;ADDRESS FOR NEW ROUTINE
        LD (BF96H),HL   ;LOAD NEW ADDRESS

;***************** INPUT DOWNLOAD DESTINATION ADDRESS ************************
        LD HL,MESG0     ;PROMPT FOR ADDRESS
        LD A,0
        RST 10H         ;SERVICE ROUTINE TO OUTPUT STRING
        CALL GETBYT
        LD D,C          ;MSB OF ADDRESS
        CALL GETBYT
        LD E,C          ;LSB OF ADDRESS
        LD (ADDR),DE    ;STORE STARTING ADDRESS
        LD HL,MESG1     ;OUTPUT STATUS MESSAGE
        LD A,0
        RST 10H

;******************* INITIALIZATION OF 16550 UART *****************************
        LD A,80H
        OUT (LCR),A     ;DLAB
        LD A,0CH
```

TL/C/10436–13

```
        OUT (DLL),A     ;9600 BAUD
        LD A,00H
        OUT (DLM),A
        LD A,03H
        OUT (LCR),A     ;8,1,N
        LD A,01H
        OUT (FCR),A     ;ENABLE FIFOS
        LD A,03H
        OUT (FCR),A     ;RESET FIFOS
        LD A,C9H
        OUT (FCR),A     ;SET TRIGGER LEVEL TO 14, SET DMA MODE TO 1
        LD A,03H
        OUT (MCR),A     ;SET /RTS AND /DSR ACTIVE (FOR USE BY TERMINAL)

;******************** INITIALIZATION OF 810A TIMER ***************************
        LD A,0
        OUT (DDRC),A    ;SET PORT C (TIMER GATE PIN) TO BE INPUTS
        OUT (TMRMOD),A  ;TIMER 0 RESET
        LD A,1BH
        OUT (TMRMOD),A  ;MODE: RESTARTABLE TIMER
        LD A,FFH
        OUT (LMOD),A    ;LSB OF MODULUS
        OUT (HMOD),A    ;MSB OF MODULUS

;******************** INITIALIZATION OF CPU INTERRUPTS ***********************
        EI              ;ENABLE CPU INTERRUPTS
        LD A,04H
        OUT (BBH),A     ;UNMASK /RSTB INTERRUPT

;************************ INITIALIZATION OF 8237 ******************************
        LD IX,ADDR
        LD A,(IX+0)
        OUT (CH0ADDR),A ;LSB OF STARTING MEMORY ADDRESS
        LD A,(IX+1)
        OUT (CH0ADDR),A ;MSB OF STARTING MEMORY ADDRESS
        LD A,FFH
        OUT (CH0WRD),A  ;LSB OF WORD COUNT
                        ;MAXIMUM # OF TRANSFERS = 64k
        LD A,FFH
        OUT (CH0WRD),A  ;MSB OF WORD COUNT
        LD A,40H
        OUT (COMM),A    ;COMMAND REG: MEM TO MEM DISABLE, CONTROLLER ENABLE,
                        ;NORMAL TIMING, FIXED PRIORITY, LATE WRITE, ACTIVE
                        ;LOW DREQ, ACTIVE LOW DACK
        LD A,04H
        OUT (MODE),A    ;MODE REG: CH0, WRITE TRANSFERS (PORT TO MEM),
                        ;NO AUTOINITIALIZATION, INC ADDRESS, DEMAND MODE

        LD A,00H
        OUT (MASK),A    ;UNMASK CHANNEL 0
        OUT (START),A   ;START 810 TIMER

;***************************** CPU WAIT LOOP **********************************
WAIT    NOP             ;ALLOW PORT TO MEMORY TRANSFERS UNTIL 810A TIMEOUT
        JR WAIT
;******************************************************************************


;************************** PROCEDURE GETBYT **********************************
GETBYT  LD C,0
        CALL INNIB      ;INPUTS ASCII BYTE FROM PORT AND CONVERTS TO HEX NIB
        SLA A
        SLA A
        SLA A
        SLA A
        LD C,A          ;MOST SIGNIFICANT NIBBLE
```

TL/C/10436–14

```
        CALL INNIB
        OR C
        LD C,A          ;RETURN BYTE IN REG C
        RET

;************************ PROCEDURE INNIB **********************************
INNIB   LD A,04H
        RST 10H         ;INPUT ASCII BYTE FROM SERIAL PORT
        LD B,A
        LD A,3
        RST 10H         ;ECHO BYTE TO SCREEN
        LD A,06H
        RST 10H         ;CONVERT ASCII BYTE TO HEX NIBBLE (ACC = 0x)
        RET



        ORG 0100H
;******************** /RSTB INTERRUPT SERVICE ROUTINE ************************
        IN A,(CH0ADDR)  ;READ LSB OF DMA CURRENT MEMORY ADDRESS
        LD C,A
        IN A,(CH0ADDR)  ;READ MSB OF CURRENT ADDRESS
        LD B,A
        LD IX,ADDR
        LD A,(IX)       ;GET LSB OF STARTING DMA ADDRESS
        CP C
        JP NZ,END       ;BYTES WERE TRANSFERRED BEFORE TIMEOUT SO STOP
        LD A,(IX+1)     ;GET MSB OF STARTING ADDRESS
        CP B
        JP NZ,END       ;BYTES WERE TRANSFERRED SO STOP
        LD A,0          ;ELSE RESTART TIMER
        OUT (TMRMOD),A  ;RESET
        LD A,1BH
        OUT (TMRMOD),A  ;LOAD MODE
        LD A,FFH
        OUT (LMOD),A    ;LSB OF MODULUS
        OUT (HMOD),A    ;MSB OF MODULUS
        OUT (START),A   ;RESTART TIMER
        EI              ;REENABLE INTERRUPT
        RETI

END     OUT (STOP),A    ;VALID TIMEOUT SO STOP TIMER AND SET OUTPUT INACTIVE
        LD HL,MESG2     ;VALID TIMEOUT SO OUTPUT MESG AND STOP
        LD A,0
        RST 10H         ;OUTPUT TERMINATION MESSAGE
        RST 08H         ;RETURN TO DEBUGGER

ADDR    DW      00H
MESG0   DB 13,10,'ENTER 16 BIT DESTINATION ADDRESS:',0
MESG1   DB 13,10,'DOWNLOADING...',13,10,0
MESG2   DB 13,10,'PORT RECEIVER TIMEOUT.  DOWNLOAD COMPLETE.',13,10,0
```

TL/C/10436–15

The source code for DMARD, DMAWR and the monitor program (MON800 V1.ASM) are available on Dial-A-Helper. The files are located in the \F1100\NSC800 directory.

Dial-A-Helper is a service provided by the Microcontroller Applications Group. The Dial-A-Helper system provides access to an automated information storage and retrieval system that may be accessed over standard dial-up telephone lines 24 hours a day. The system capabilities include a MESSAGE SECTION (electronic mail) for communicating to and from the Microcontroller Applications Group and a FILE SECTION mode that can be used to search out and retrieve application data about NSC Microcontrollers. The minimum system requirement is a dumb terminal, 300 or 1200 baud modem, and a telephone.

With a communications package and a PC, the code detailed in this App Note can be down loaded from the FILE SECTION to disk for later use. The Dial-A-Helper telephone lines are:

Modem (408) 739-1162
Voice (408) 721-5582

For Additional Information, Please Contact Factory

Section 8

Physical Dimensions/ Appendices

Section 8 Contents

Glossary

In our efforts to be concise and precise, we often invent new words or acronyms to use as shorthand representations of "things" that require much longer names if the jargon is not used. Being humans, we then become very impressed with our ability to exclude those not in "the know" and another "in" group is formed. This glossary has been developed to help bridge this language gap. We know it will help. We hope you will use it.

Abort—The first step of recovery when an instruction or its operand(s) is not available in main memory. An Abort is initiated by the Memory Management Unit (MMU) and handled by the CPU.

Absolute Address—An address that is permanently assigned to a fixed location in main memory. In assembly code, a pattern of characters that identifies a fixed storage location.

Access Time—The time interval between when a request for information is made and the instant this information is available.

Access Class—The five Series 32000 access classes are memory read, memory write, memory read-modify-write, memory address, and register address. The access class informs the Series 32000 CPU how to interpret a reference to a general operand. Each instruction assigns an access class to each of it two operands, which in turn fully defines the action of any addressing mode in referencing that operand.

Accumulator—A register which stores the result of an ALU operation.

Ada—A high level language designed for the Department of Defense. It gives preference to full English words. It is meant to be the standard military language.

Address—An expression, usually numerical, which designates a specific location in a storage or memory device.

Address-Data Register—A register which may contain either address or data, sometimes referred to as a general-purpose register.

Address Strobe—Control signal used to tell external devices when the address is valid on the external address bus.

Address Translation—The process by which a logical address emanating from the CPU is transformed into a physical address to main memory. This is performed by the Memory Management Unit (MMU) in Series 32000 systems. Logical address to Physical address mapping is established by the operating system when it brings pages into main memory.

Addressing Mode—The manner in which an operand is accessed. Series 32000 CPUs have nine addressing modes: Register, Register Relative, Memory Relative, Immediate, Absolute, External, Top-of Stack, Memory Space, and Scaled Indexing.

Algorithm—A set of procedures to which a given result is obtained.

Alignment—The issue of whether an instruction must begin on a byte, double byte, or quad byte address boundary.

ALU—Arithmetic Logic Unit. A computational subsystem which performs the arithmetic and logical operations of a digital system.

Array—A structured data type consisting of a number of elements, all of the same data type, such that each data element can be individually identified by an integer index. Arrays represent a basic storage data type used in all high-level languages.

ASCII—(*American National Standard Code for Information Interchange,* 1968). This standard code uses a character set generally coded as 7-bit characters (8-bits when using parity check). Originally defined to allow human readable information to be passed to a terminal, it is used for information interchange among data processing systems, communication systems, and associated equipment. The ASCII set consists of alphabetic, numeric, and control characters. Synonymous with USASCII.

Assemble—To prepare a machine language program (also called machine code or object code) from a symbolic language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses. Machine code is a series of ones and zeros which a computer "understands".

Assembler—This program changes the programmer's source program (written in English assembly language and understandable to the programmer) to the 1's and 0's that the machine "understands". In particular, the Assembler converts assembly language to machine code. This machine code output is called the OBJECT file.

Assembly Language—A step up in the language chain. This is a set of instructions which is made up of alpha numeric characters which, with study, are understandable to the programmer. Different type of machines have different assembly languages, so the assembly language programmer must learn a different set of instructions each time s/he changes machine.

Associative Cache—A dual storage area where each data entry has an associated "tag" entry. The tags are simultaneously compared to the input value (a logical address) in the case of the MMU, and if a matching tag is found, the associated data entry is output. An associative cache is present within the MMU in Series 32000 systems to provide logical-to-physical address translation.

Asynchronous Device—A device in which the speed of operation is not related to any frequency in the system to which it is connected.

BASIC—This acronym stands for Beginner's All-purpose Symbolic Instruction Code. BASIC is one of the most "English like" of the high level languages and is usually the first programming language learned.

Baud Rate—Data transfer rate. For most serial transmission protocols, this is synonymous with bits-per-second (bps).

BCD—Binary Coded Decimal. A binary numbering system for coding decimal numbers. A 4-bit grouping provides a binary value range from 0000 to 1001, and codes the decimal digits "0" through "9". To count to 9 requires a single 4-bit grouping; to count to 99 takes two groupings of 4 bits; to count to 999 takes three groupings of 4 bits, etc.

Benchmark—In terms of computers, this refers to a software program designed to perform some task which will demonstrate the relative processing speed of one computer versus another.

Glossary (Continued)

Bit—An abbreviation of "binary digit". It is a unit of information represented by either a one or a zero.

Bit Field—A group of bits addressable as a single entity. A bit field is fully specified by the location of its least significant bit and its length in bits. In Series 32000 systems, bit fields may be from one to 32 bits in length.

Branch—A nonsequential flow in a software instruction stream.

Breakpoint—A place in a routine specified by an instruction, instruction digit, or other condition, where the software program flow will be interrupted by external intervention or by a monitor routine.

Buffer—An isolating circuit used to avoid reaction of a driven circuit on the corresponding driver circuit. Buffers also supply increased current drive capacity.

Bus—A group of conductors used for transmitting signals or power.

Bus Cycle—The time necessary to complete one transfer of information requiring the use of external address, data and control buses.

Byte—Eight bits.

Byte Enable—$\overline{BE0}$ to $\overline{BE3}$. CPU control signals which activate memory banks, each bank providing one byte of data per address.

C—A highly structured high level language developed by Bell Laboratories to optimize the size and efficiency of the program. This language has gained much popularity because it allows the programmer to get close to the hardware (low level) as well as being a high level language. Before C, the programmer who had to address the hardware had to use assembly language or machine code.

Cache—See Associative Cache.

Cache Hit—In the MMU, logical-to-physical address translation takes place via the associative cache. For this to happen, the addressed page must be resident in physical memory such that a logical address tag is present in the MMU's translation cache.

Cache Miss—When a logical address is presented to the MMU, and no physical address translation entry is found in the MMU's associative cache.

Cascaded—Stringing together of units to expand the operation of the unit. Interrupt Control Units present in a Series 32000 system which are in addition the Master ICU are referred to as "cascaded" ICUs; i.e., interrupts cascade from a second-level ICU through the master ICU to the CPU.

Clock—A device that generates a periodic signal used for synchronization.

Clock Cycle—After making a low-to-high transition, the clock will have completed one cycle when it is about to make another low-to-high transition. This time is equal to 1/f where f = the clock frequency.

COBOL—This acronym stands for "Common Business Oriented Language". It is a language especially good for bookkeeping and accounting.

COFF-COMMON OBJECT FILE FORMAT is a standard way of constructing files developed by AT&T for the express purpose of making all files similar. This will help reduce the situation where large files developed by one organization won't run on another organization's equipment simply because the software interfaces are different. It provides a great potential for savings in both time and money.

Compile—To take a program written in a High-Level Language such as C, Pascal, or FORTRAN and convert it into an object-code format which can be loaded into a computer's main memory. During compilation, symbolic HLL statements, called source code, are converted into one or more machine instructions which the CPU "understands". A compiler also calls the assemble function.

Compiler—The program that converts from Source to Machine Code. The conversion is from a particular high level language to machine code. For example, the C compiler will convert a C source program written by a programmer to machine code. This machine code output is in the same format as that of the assembler and is also called an OBJECT file.

CPU—Central Processing Unit. The portion of a computer system that contains the arithmetic logic unit, register file, and other control oriented subsystems. It performs arithmetic operations, controls instruction processing, and provides timing signals and other housekeeping operations.

Cross Support—The alternative to using a "Native" development like SYS32 to develop your programs is to use Cross Support software. "Native" means that the CPU in the development system is the same as the CPU in the system being developed. Cross support software is all of the necessary programs for development that operate on one CPU, but generate code for another CPU. Use of the VAX to generate Series 32000 code is a good example of cross support.

Demand-Paged Virtual Memory—A virtual memory method in which memory is divided into blocks of equal size which are referred to as pages. These pages are then moved back and forth between main memory and secondary storage as required by the CPU. Demand paging reduces the problem of memory fragmentation which results in unused memory space.

Dispatch Table—In Series 32000 systems, this is an area of memory which contains interrupt descriptors for all possible hardware interrupts and software traps. The interrupt descriptor directs the CPU to the module descriptor for the procedure which is designed to handle that particular interrupt.

Displacement—A numerical offset from a known point of reference. Displacements are used in programming to facilitate position independent code, such that a given program can be loaded anywhere in memory. In Series 32000 processors, a displacement is contained in the instruction itself.

Glossary (Continued)

DMA—Direct Memory Access. A method that uses a small processor (DMA Controller) whose sole task is that of controlling input-output or data movement. With DMA, data is moved into or out of the system without CPU intervention once the DMA controller has been initialized by the CPU and activated.

Double-Precision—With reference to 32000 floating-point arithmetic, a double-precision number has a 52-bit fraction field, 11-bit exponent field and a sign bit (64-bits total).

Double Word—Two words, i.e., 32 bits.

Editor—A program which allows a person to write and modify text. This program can be as complicated as the situation requires, from the very simple line editor to the most complicated word processor. Letters, numbers and unprintable control characters are stored in memory so that they can be recalled for modification or printing. The programmer uses this device to enter the program into the computer. At this stage, the program is recognizable to both the programmer and the computer as lines of English text. This English version of the program is known as the SOURCE.

Emulate—To imitate one system with another, such that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system.

Exception—An occurrence which must be resolved through CPU intervention. An exception results in the suspension of normal program flow. In Series 32000 systems, exceptions occur as a result of a hardware reset, interrupt or software traps. Execution of floating-point instructions may also result in occurrences which must be resolved through CPU intervention.

Exponent—In scientific notation, a numeral that indicates the power to which the base is raised.

EXEC2—NSC's Real Time Executive for Series 32000.

FIFO—First-in first-out. A FIFO device is one from which data can be read out only in the same order as it was entered, but not necessarily at the same rate.

Floating-Point—A method by which computers deal with numbers having a fractional component. In general, it pertains to a system in which the location of the decimal/binary point does not remain fixed with respect to one end of numerical expressions, but is regularly recalculated. The location of the point is usually given by expressing a power of the base.

FORTRAN—A high level language written for the scientific community. It makes heavy use of algebraic expressions and arithmetic statements.

FP—Frame Pointer. CPU register which points to a dynamically allocated data area created at the beginning of a procedure by the ENTER instruction.

FPU—Floating-Point Unit is a slave processor in Series 32000 systems which implements in hardware all calculations needed to support floating-point arithmetic, which otherwise would have to be implemented in software. The NS32081 FPU provides high-speed floating point instructions for single (32-bit) and double (64-bit) precision. Supports IEEE standard for binary floating point arithmetic. Compatible with NS32032, NS32C032, NS32016, NS32C016 and NS32008 CPUs.

Fragmented—The term used to describe the presence of small, unused blocks of memory. The problem is especially common in segmented memory systems, and results in inefficient use of memory storage.

Frame—A block of memory on the stack that provides local storage for parameters in the current procedure.

GENIX—The NSC version of the UNIX operating system, ported to work with the Series 32000. It also has all of the necessary utilities added so that program development can be accomplished.

Hardware—Physical equipment, e.g., mechanical, magnetic, electrical, or electronic devices, as opposed to the software programs or method in which the hardware is used.

High Level Languages—These are languages which are not dependent on the type of computer on which they run. A program written in a high level language will generally run on any computer for which there is a compiler for that language. This feature makes high level languages "Portable", i.e., the same program will run on many different types of computers. A HLL requires a compiler or interpreter that translates each HLL statement into a series of machine language instructions for a particular machine.

ICU—Interrupt Control Unit. A memory-mapped microprocessor support chip in Series 32000 systems which handles external interrupts as well as additional software traps. The ICU provides a vector to the CPU to identify the servicing software procedure.

Indexing—In computers, a method of address modification that is by means of index registers.

Index Register—A register whose contents may be added to or subtracted from the operand address.

Indirect Addressing—Programming method where the initial address is the storage location of a word which is the actual address. This indirect address is the location of the data to be operated upon.

Instruction—A statement that specifies an operation and the values or locations of its operands, i.e., it tells the CPU what to do and to what.

Instruction Cycle—The period of time during which a programmed system executes a particular instruction.

Instruction Fetch—The action of accessing the next instruction from memory, often overlapped by its partial execution.

Instruction Queue—With Series 32000 CPUs, this is a small area of RAM organized as a FIFO buffer which stores prefetched instructions until the CPU is ready to execute them.

Interpreter—A program which translates HLL statements into machine instructions at run time, i.e., while the program is executing, and is co-resident with the user program.

Glossary (Continued)

Interrupt—To signal the CPU to stop a software program in such a way that it can be resumed and branch to another section of code. Interrupts can be caused by events external or internal to the CPU, and by either software or hardware.

INTBASE—Interrupt Base Register. In the Series 32000, a 32-bit CPU register which holds the address of the dispatch table containing addresses for interrupts and traps.

ISE—In-System Emulator. A computer system which imitates the operation of another in terms of software execution. In microprocessor system development, the ISE takes the place of the microprocessor by means of a connector at the end of an umbilical cable. Not only does the ISE perform all the functions of the microprocessor, but it also allows the engineer to debug his system by setting breakpoints on various conditions, permits tracing of program flow, and provides substitution memory which may be used in place of actual target system memory.

ISV—Independent Software Vendor. A vendor, independent from National Semiconductor, who ports or develops software for Series 32000 components. They in turn sell this software to our customers who are designing Series 32000 based products.

Kernel—This is the name given to the core of the operating system. Other programs are added to the kernel to provide the features of the operating system. The kernel provides control and synchronization.

Language—A set of characters and symbols and the rules for using them. In our context, it is the "English like" format of the instructions which are understood by both the programmer and the computer.

Library—High level languages as well as assembly language contain many routines which are used over and over again. To prevent the programmer from having to write the routine every time it is needed, these routines are stored in libraries to be referenced each time they are needed. These libraries are also OBJECT files.

Linear Address Space—An address space where addresses start at location zero and proceed in a linear fashion (i.e., with no holes or breaks) to the upper limit imposed by the total number of bits in a logical address.

Link Base—In the Series 32000, Module Descriptor entry which points to a table in memory containing entries which reference variables or entry points in Modules external to the one presently executing.

Linker—Large programs are generally broken down to component parts and farmed out to several programmers. Each one of these parts is called a MODULE. Each programmer will develop the module using either high level or assembly language, then "assemble" assembly language modules or "compile" high level language modules. A programmer tells the linker how to connect these modules to make the program run. The linker makes these connections, resolves all questions about data needed by one module, but contained in another, finds all library routines, and cleans up any other loose ends. The output from the linker is called BINARY file and is the file that will run on the computer.

Logical Address Space—The range of addresses which a programmer can assign in a software program. This range is determined by the length of the computer's address registers.

LSB—Least Significant Bit. The bit in a string of bits representing the lowest value.

Machine Code—The code that a computer recognizes. Specifies internal register files and operations that directly control the computer's internal hardware.

Machine Language—The ones and zeros which are "understood" by the machine. This is often called "Binary Code." The programmer must be able to understand the bit patterns to be able to decipher the language. Each machine has a unique machine language.

Main Memory—The program and data storage area in a computer system which is physically addressed by the microprocessor or MMU address lines.

Mantissa—In a floating-point number, this is the fractional component.

Mapping—The process whereby the operating system assigns physical addresses in main memory to the logical addresses assigned by the software.

Memory-Mapped—Referring to peripheral hardware devices which are addressed as if they were part of the computer's memory space. They are accessed in the same manner as main memory, i.e., through memory read/write operations.

Microcode—A sequence of primitive instructions that control the internal hardware of a computer. Their execution is initiated by the decoding of a software instruction. Microcode is maintained in special storage and often used in place of hardwired logic.

Microcomputer—A computer system whose Central Processing Unit is a Microprocessor. Generally refers to a board-level product.

Minicomputer—A "box-level" computer with system capabilities generally between that of a microcomputer and a mainframe.

MMU—Memory Management Unit. This is a slave processor in Series 32000 which aids in the implementation of demand-paged virtual memory. It provides logical to physical address translation and initiates an instruction abort to the CPU when a desired memory location is not in main memory.

MOD—Mod Register. In the Series 32000, a 16-bit CPU register which holds the address of the Module Descriptor of the currently executing software module.

Module—An independent subprogram that performs a specific function and is usually part of a task, i.e., part of a larger program.

Module Descriptor—In the Series 32000, a set of four 32-bit entries found in main memory. Three are currently defined and point to the static data area, link table, and first instruction of the module it describes. The fourth is reserved.

Glossary (Continued)

Modularity—A software concept which provides a means of overcoming natural human limitations for dealing with programming complexity by specifying the subdivision of large and complex programming tasks into smaller and simpler subprograms, or modules, each of which performs some well-defined portion of the complete processing task.

MSB—Most Significant Bit. The bit in a string of bits representing the highest value.

NET—Short for NETWORK and describes a number of computers connected to each other via phone or high speed links. A net is convenient for exchanging common information in the form of "mail" as well as for data exchange.

NMI—Nonmaskable Interrupt. A hardware interrupt which cannot be disabled by software. It is generally the highest priority interrupt.

Object Code—Output from a compiler or assembler which is itself executable machine code (or is suitable for processing to produce executable machine code).

Operand—In a computer, a datum which is processed by the CPU. It is referenced by the address part of an instruction.

Operating System—A collection of integrated service routines used by the computer to control the sequence of programs. The operating system consists of software which controls the execution of computer programs and which may provide storage assignment, input/output control, scheduling, data management, accounting, debugging, editing, and related services. Their sophistication varies from small monitor systems, like those used on boards, to the large, complex systems used on main frames.

Operating System Mode—In this mode, the CPU can execute all instructions in the instruction set, access all bits in the Processor Status Register, and access any memory location available to the processor.

Operator—In the description of an instruction, it is the action to be performed on operands.

Page Fault—A hardware generated trap used to tell the operating system to bring the missing page in from secondary storage.

Page Swap—The exchange of a page of software in secondary storage with another page located in main memory. The operating system supervises this operation, which is executed by the CPU and involves external devices such as disk and DMA controllers.

Page Table—A 1K-byte area in main memory containing 256 entries which describe the location and attributes of all pointer tables, i.e., a list of pointer table addresses.

Peripheral—A device which is part of the computer system and operates under the supervision of the CPU. Peripheral devices are often physically separated from the CPU.

Pascal—A high level language designed originally to teach structured programming. It has become popular in the software community and has been expanded to be a versatile language in industry.

Physical Address—The address presented to main memory, either by the CPU or MMU.

Pointer Table—A 512-byte page located either in main memory or secondary storage containing 128 entries. Each entry describes an individual page of the software program. Each page of the software program may reside in main memory or in secondary storage.

Pop—To read a datum from the top of a stack.

PORT—To port an operating system is to cause that particular operating system to operate with a defined hardware package. GENIX is the NSC version of UNIX which has been ported to SYS32. The operating system for other Series 32000 based systems will differ in some degree from SYS32 and the NSC GENIX binary will not operate. It is now necessary to modify GENIX to fit the situation caused by the new hardware. The GENIX SOURCE is used because this is the program that is most readily understood by the programmer. The source is changed, compiled, and linked to get a new binary for that particular machine.

Primitive Data Type—A data type which can be directly manipulated by the hardware. With Series 32000, these are integers, floating-point numbers, Booleans, BCD digits, and bit fields.

Procedure—A subprogram which performs a particular function required by a module, i.e., by a larger program; an ordered set of instructions that have a general or frequent use.

Process—A task.

Program Base—Module Descriptor entry which points to the first instruction in the module being described.

Program Counter—CPU register which specifies the logical address of the currently executing instruction.

Protection—The process of restricting a software program's access to certain portions of memory using hardware mechanisms. Typically done at the operating system and page level.

PSR—Processor Status Register. A 16-bit register on Series 32000 CPU's which contains bits used by the software to make decisions and determine program flow.

Push—to write a datum to the top of a stack.

Quad word—Four words, i.e., 64 bits.

Queue—A First-In-First-Out data storage area, in which the data may be removed at a rate different from that at which it was stored.

Real Time—The actual time in human terms, related to a process. In a UNIX system, real time is total elapsed time, CPU time is the percent of time a process is actually in the CPU. Sys time is the time spent in system mode, and user time is the time spent in user mode.

Glossary (Continued)

Real Time Operating Systems—An operating system which operates with a known and predictable response time limit, so that it can control a physical event.

Record—A structured data type with multiple elements, each of which may be of a different data type, e.g., strings, arrays, bytes, etc.

Register—A temporary storage location, usually in the CPU, which holds digital data.

Relative Address—The number that specifies the difference between the base address and the absolute address.

Relocatable—In reference to software programs, this is code which can be loaded into any location in main memory without affecting the operation of the program.

Return Address—The address to which a subroutine call, interrupt or trap subroutine will return after it is finished executing.

Routine—A procedure.

Royalty—Royalty is money paid to the inventor for each item of product sold. A good analogy to use is the music business. Any time a song is used, the songwriter is paid a royalty. Think of UNIX as a song and GENIX or SYSTEM V as special arrangements. For each shipment of GENIX or SYSTEM V, the customer pays a royalty to NSC who, in turn, pays a royalty to AT&T.

SB—In the Series 32000 Static Base Register. Points to the start of the static data area for the currently executing module.

Secondary Storage—This is generally slow-access, nonvolatile memory such as a hard-disk which is used to store the pages of software programs not currently needed by the CPU.

Segmented Address Space—Term used to describe the division of allocatable memory space into blocks of segments of variable size.

Setup Time—The minimum amount of time that data must be present at an input to ensure data acceptance when the device is clocked.

Slave Processor—A processor which cooperates with the main microprocessor in executing certain instructions from the instruction stream. A slave processor generally accelerates certain functions which increases overall system throughput. Examples of slave processors are the FPU and MMU of Series 32000.

Software—Programs or data structures that execute instructions or cause instructions to be executed and that will cause the computer to do work.

Software License—NSC does not sell software. Rather, we license the right to use our software. A software license is required for all Series 32000 software. We use the license to protect NSC's interests and to assist in honoring our commitment to AT&T. The license is also the vehicle which we use to track customers so that updates can be issued in a timely manner.

Software Q/A—It is the charter of the Quality Assurance people to ensure that when a software product reaches the customer that it is "bug" free. In the real world, it is impossible to test every combination of functions, so some bugs do get through. The Q/A engineer develops test programs which rigorously test the product prior to its introduction to the market place.

SP1—In the Series 32000, User Stack Pointer. Points to the top of the User Stack and is selected for all stack operations while in User Mode.

SP0—In the Series 32000, Interrupt Stack Pointer. Points to the top of the interrupt stack. It is used by the operating system whenever an interrupt or trap occurs.

Stack—A one-dimensional data structure in which values are entered and removed one datum at a time from a location called the Top-of-Stack. To the programmer, it appears as a block of memory and a variable called the Stack Pointer (which points to the top of the stack).

Stack Pointer—CPU register which points to the top of a stack.

Static Base Register—A 32-bit CPU register which points to the beginning of the static data area for the currently executing module.

String—An array of integers, all of the same length. The integers may be bytes, words, or double words. The integers may be interpreted in various ways (see ASCII).

Subroutine—A self-contained program which is part of a procedure.

Symmetry—A computer architecture is said to be symmetrical when any instruction can specify any operand length (byte, word or double word) and make use of any address-data register or memory location while using any addressing mode.

Synchronous—Refers to two or more things made to happen in a system at the same time, by means of a common clock signal.

Tag—A label appended to some data entry used in a look-up process whereby the desired datum can be identified by its tag.

Task—The highest-level subdivision of a user software program. The largest program entity that a computer's hardware directly deals with.

TCU—Timing Control Unit. A device used to provide system clocks, bus control signals and bus cycle extension capability for Series 32000.

Trap—An internally generated interrupt request caused as a direct and immediate result of the encounter of an event.

T-State—One clock period. If the system clock frequency is 10 MHz, one T-State will take 100 ns to complete. Operations internal and external to the CPU are synchronized to the beginning and middle of the T-States. There are four T-States in a normal Series 32000 CPU bus cycle.

Glossary (Continued)

UNIX™—An operating system developed at Bell Laboratories in the early 1970s. Software programs that run under UNIX are written in the high-level language C, making them highly portable. UNIX systems do not distinguish user programs from operating system programs in either capability or usage, and they allow users to route the output of one program directly into the input of another. This operating is unique and is becoming very popular in the microcomputer world.

USENET—A net to which UNIX systems in the United States connect. Some systems in Europe and Australia also use this net for the purpose of passing information.

User—A software program. The total set of tasks (instructions) that accomplish a desired result. Tasks are managed by the operating system.

User Mode—Machine state in which the executing procedure has limited use of the instruction set and limited access to memory and the PSR.

uucp—Software which allows UNIX computers to pass information to other UNIX systems.

Variable—A parameter that can assume any of a given set of values.

Vector—Byte provided by the ICU (Interrupt Control Unit) which tells the CPU where within the Descriptor table the descriptor is located for the interrupt it has just requested.

Virtual Address—Address generated by the user to the available address space which is translated by the computer and operating system to a physical address of available memory.

Virtual Memory—The storage space that may be regarded as addressable main storage by the system. The operating system maps Virtual addresses into physical (main memory) addresses. The size of virtual memory is limited by the method of memory management employed and by the amount of secondary storage available, not by the actual number of main storage locations, so that the user does not have to worry about real memory size or allocation.

VMS—This is the operating system designed by Digital Equipment Corporation for their VAX series of computers. The original Series 32000 software was developed on a VAX which was being controlled by the VMS Operating System.

Wait-State—An additional clock period added to a CPU memory cycle which gives an external memory device additional time to provide the CPU with data. Also used by bus arbitration circuitry to hold the CPU in an idle state until access to a shared resource is gained.

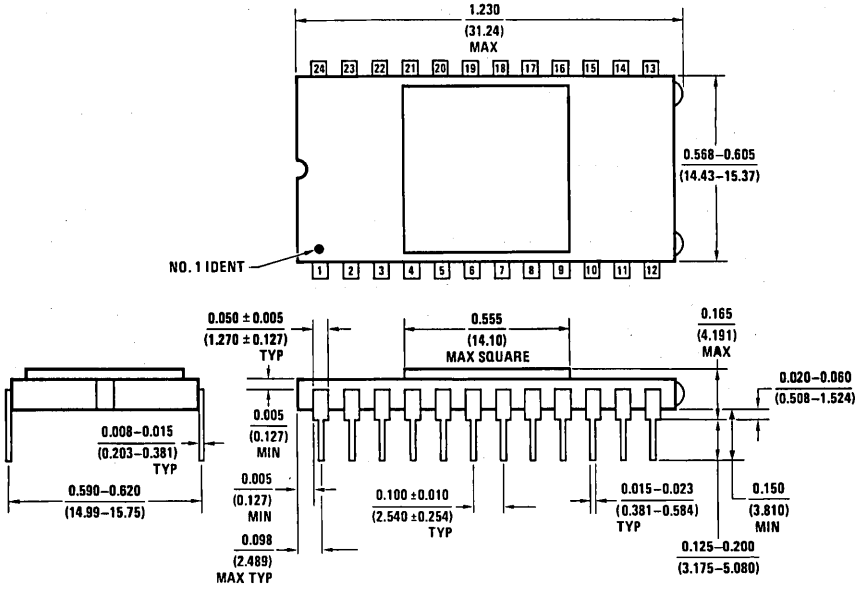
Winchester—Small, hard-disk media commonly found in personal computers.

Word—A character string or bit string considered as the primary data entity. For historical reasons, a word is a group of 16 bits in Series 32000 systems.

National Semiconductor

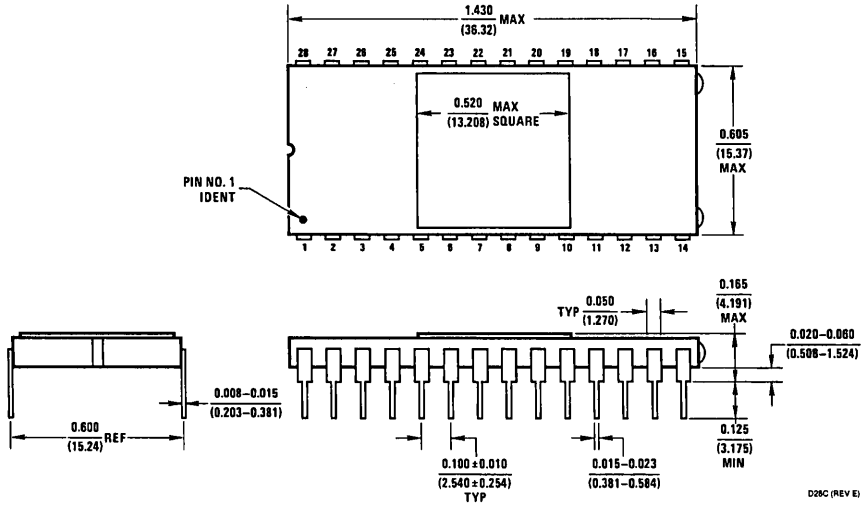
All dimensions are in inches (millimeters)

24 Lead Hermetic Dual-In-Line Package (D)
NS Package Number D24C



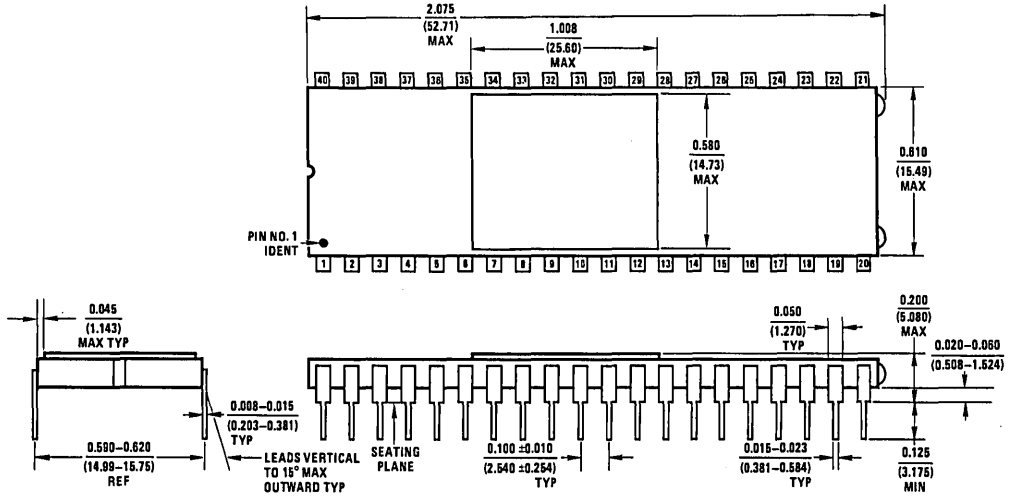
D24C (REV G)

28 Lead Hermetic Dual-In-Line Package (D)
NS Package Number D28C

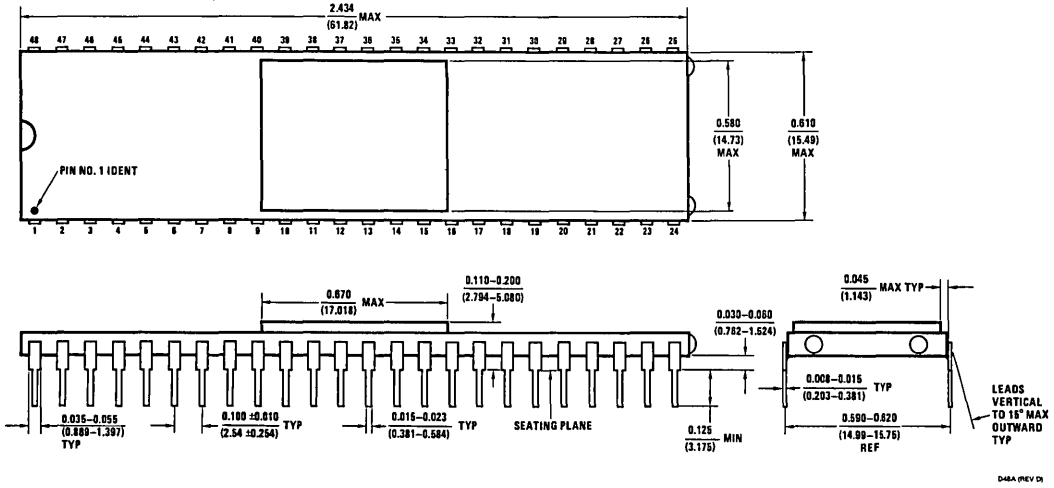


D28C (REV E)

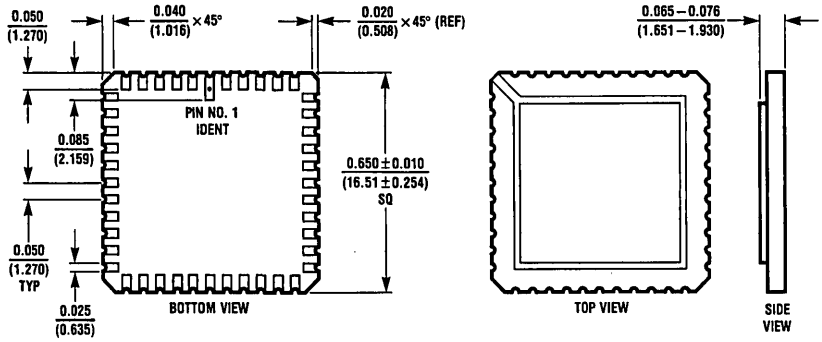
40 Lead Hermetic Dual-In-Line Package (D)
NS Package Number D40C



48 Lead Hermetic Dual-In-Line Package (D)
NS Package Number D48A

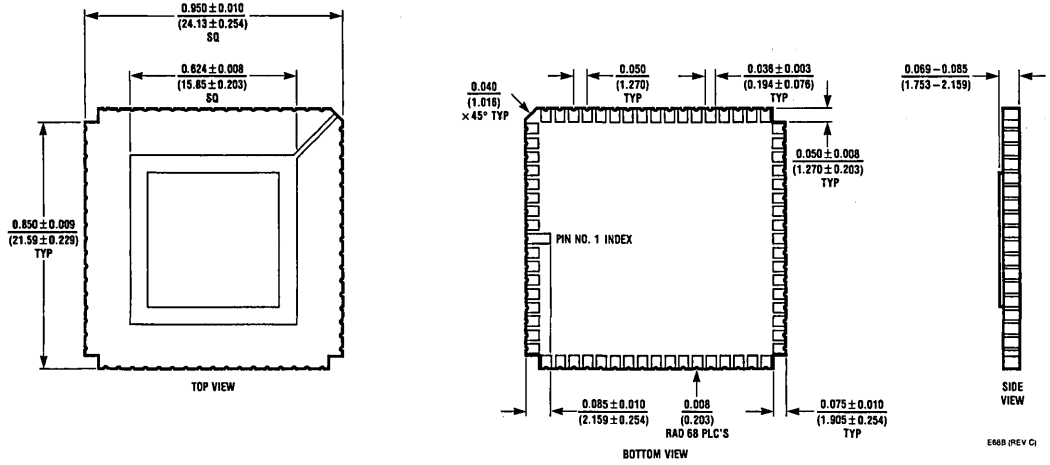


44 Leadless Chip Carrier, Type C
NS Package Number E44A



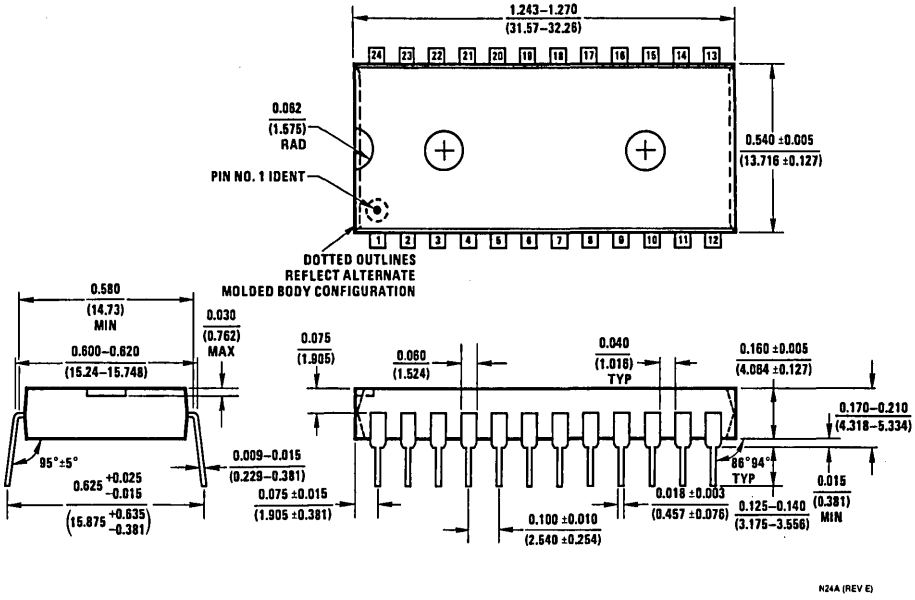
E44A (REV C)

68 Leadless Chip Carrier, Type B
NS Package Number E68B



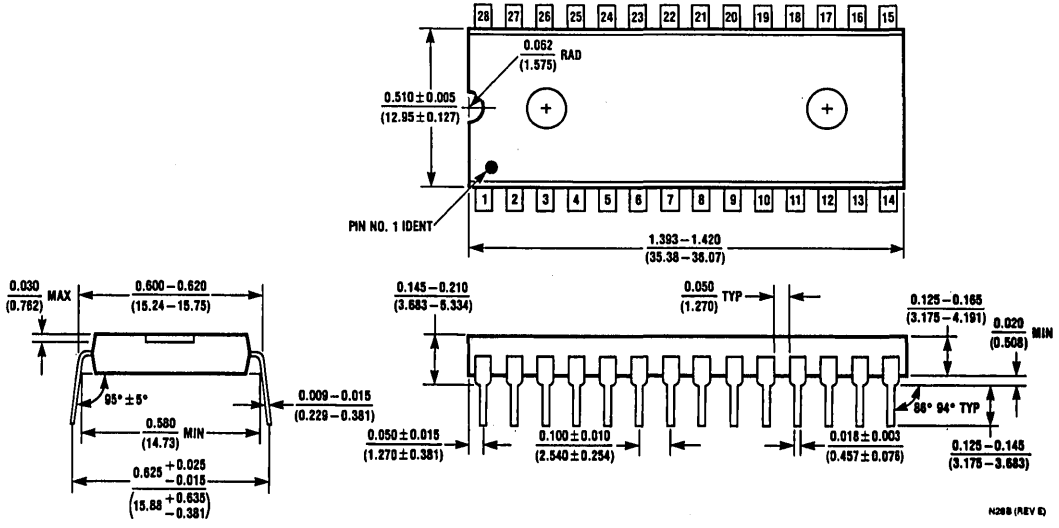
E68B (REV C)

24 Lead Molded Dual-In-Line Package (N)
NS Package Number N24A



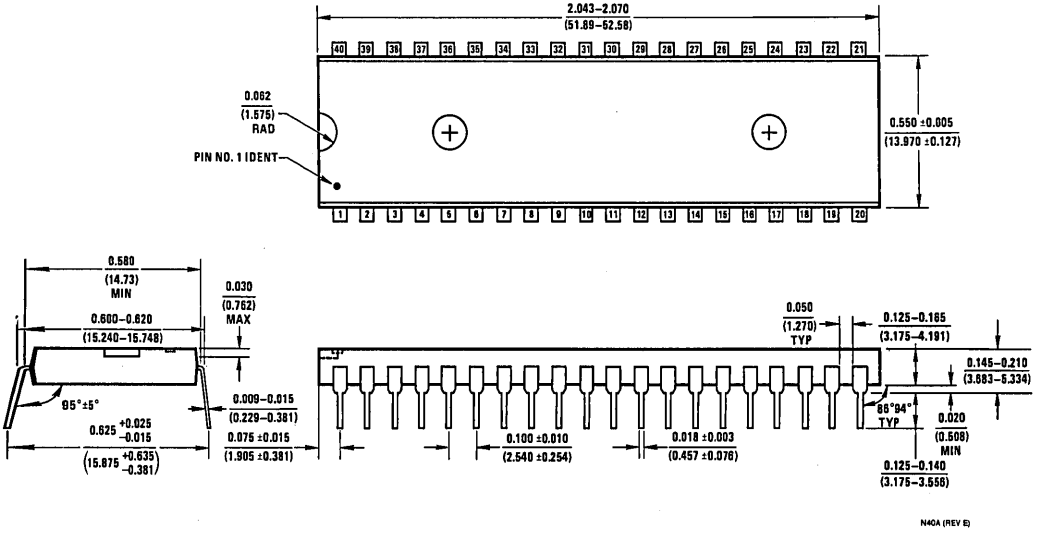
N24A (REV E)

28 Lead Molded Dual-In-Line Package (N)
NS Package Number N28B

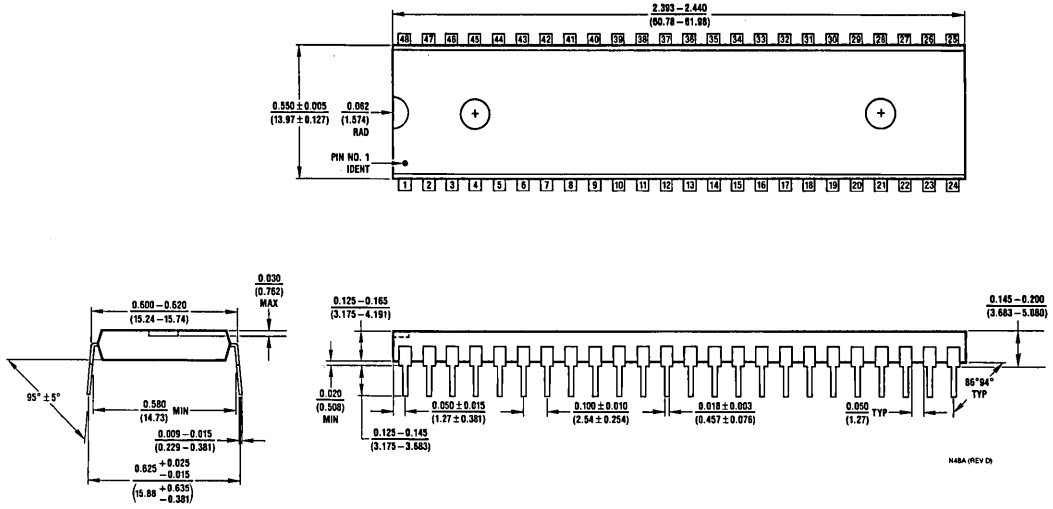


N28B (REV E)

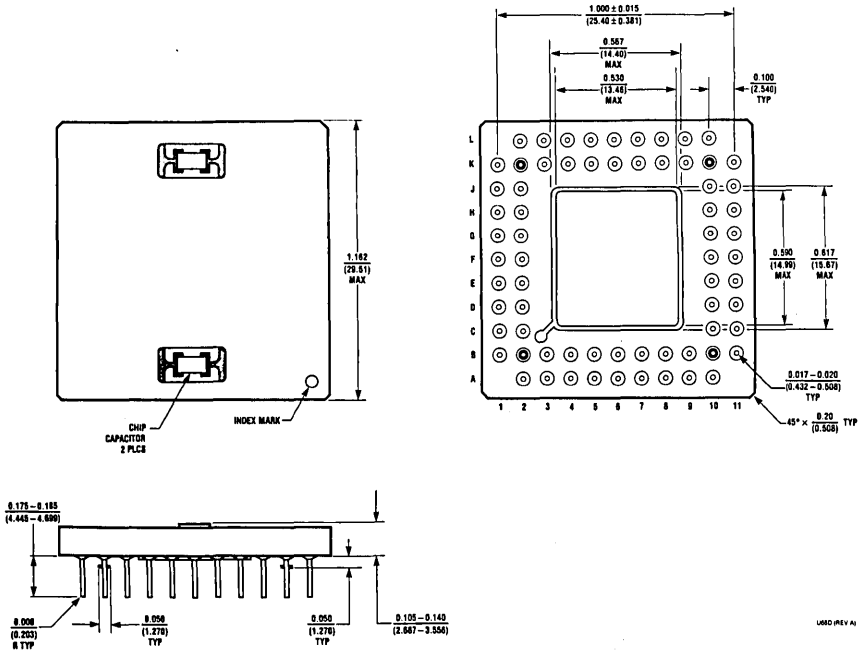
40 Lead Molded Dual-In-Line Package (N)
NS Package Number N40A



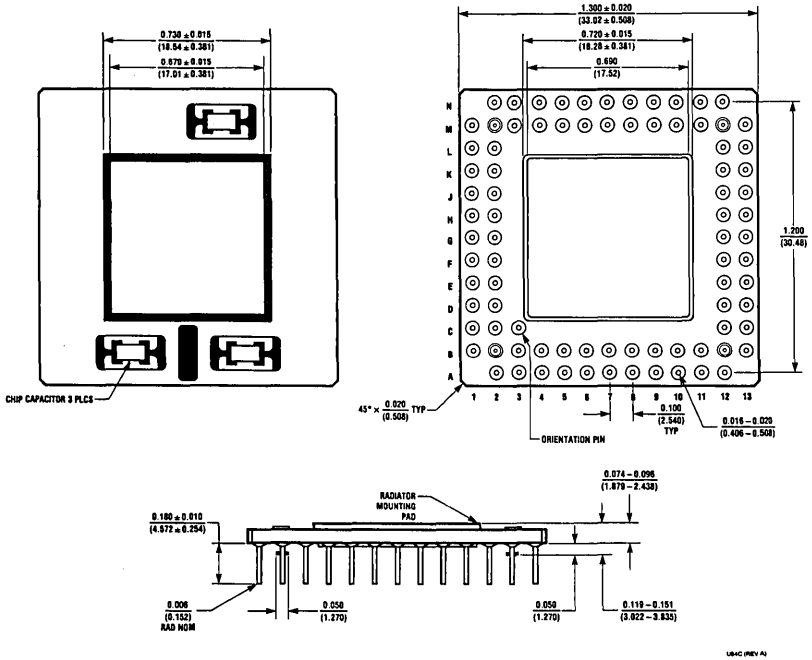
48 Lead Molded Dual-In-Line Package (N)
NS Package Number N48A



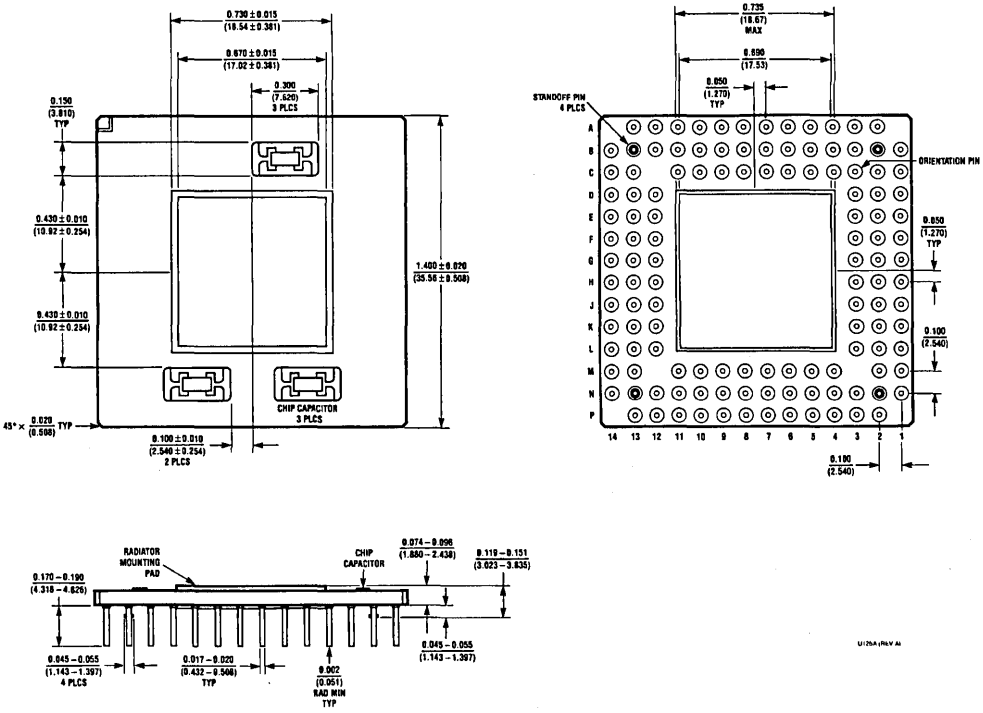
68 Pin Grid Array, Cavity Down
NS Package Number U68D



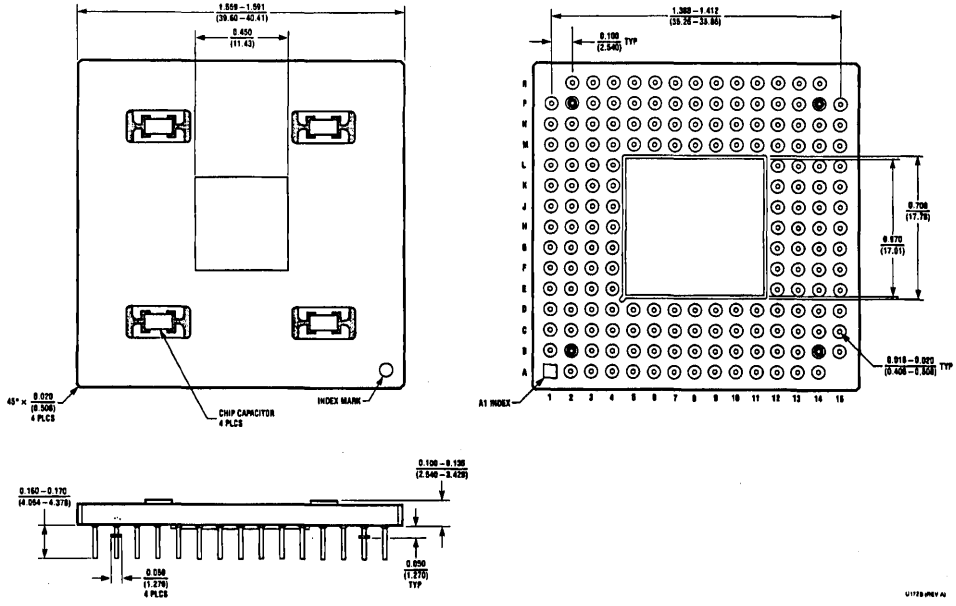
84 Pin Grid Array, Ceramic, Cavity Down
NS Package Number U84C



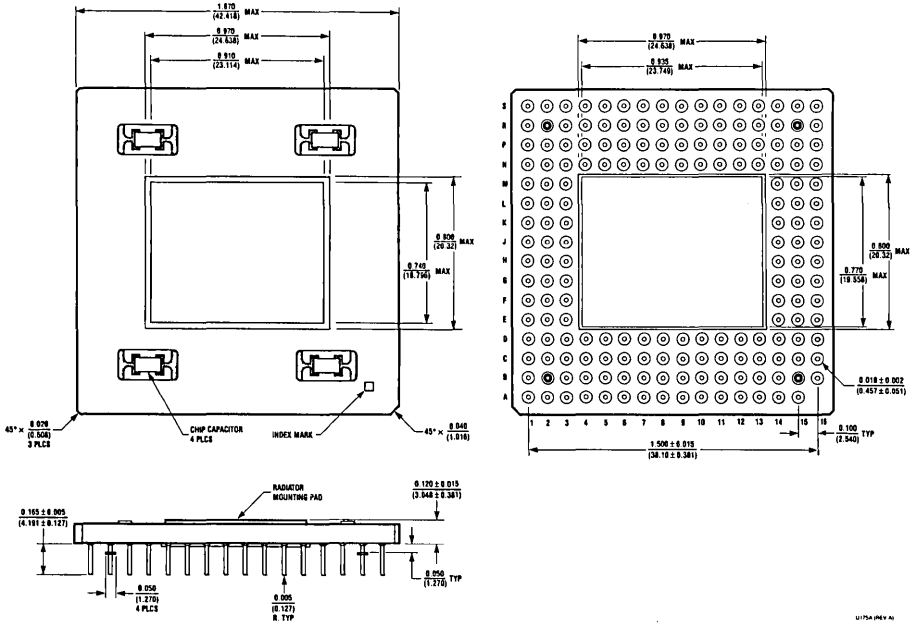
125 Pin Grid Array, Cavity Up
NS Package Number U125A



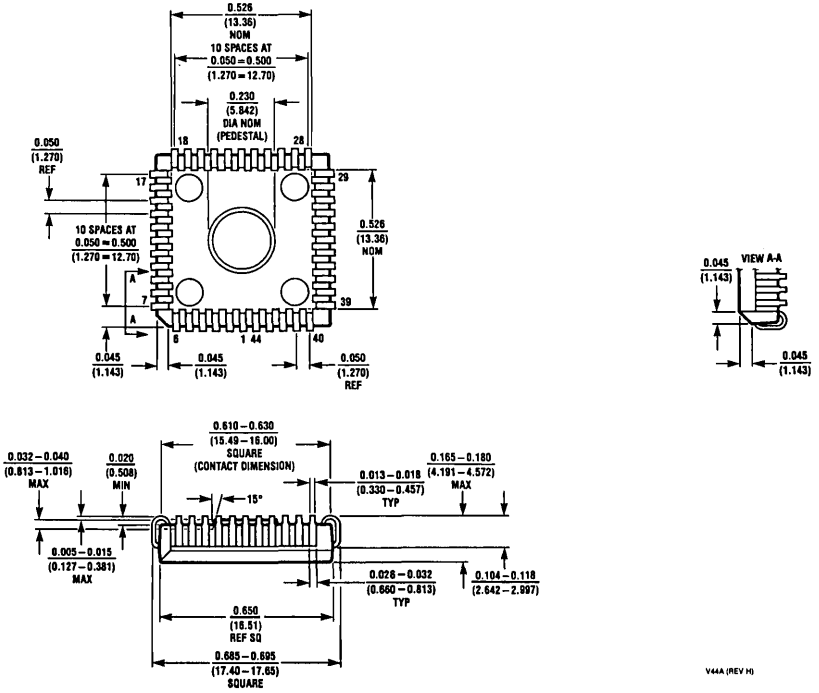
172 Pin Grid Array, Cavity Down
NS Package Number U172B



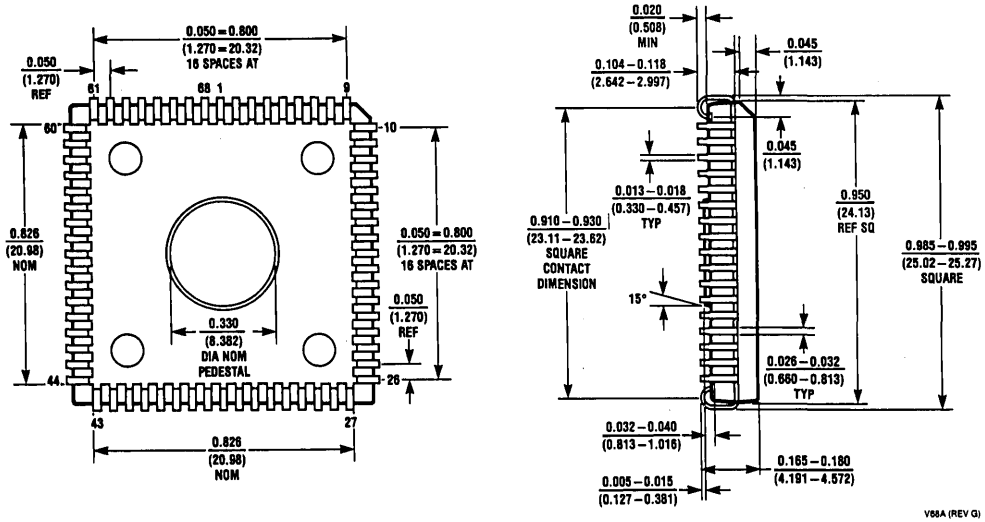
175 Pin Grid Array, Cavity Down (Type A)
NS Package Number U175A



44 Lead Plastic Chip Carrier (V)
NS Package Number V44A



68 Lead Plastic Chip Carrier (V)
NS Package Number V68A



National Semiconductor

Bookshelf of Technical Support Information

National Semiconductor Corporation recognizes the need to keep you informed about the availability of current technical literature.

This bookshelf is a compilation of books that are currently available. The listing that follows shows the publication year and section contents for each book.

Please contact your local National sales office for possible complimentary copies. A listing of sales offices follows this bookshelf.

We are interested in your comments on our technical literature and your suggestions for improvement.

Please send them to:

Technical Communications Dept. M/S 16300
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090

ALS/AS LOGIC DATABOOK—1987

Introduction to Bipolar Logic • Advanced Low Power Schottky • Advanced Schottky

ASIC DESIGN MANUAL/GATE ARRAYS & STANDARD CELLS—1987

SSI/MSI Functions • Peripheral Functions • LSI/VLSI Functions • Design Guidelines • Packaging

CMOS LOGIC DATABOOK—1988

CMOS AC Switching Test Circuits and Timing Waveforms • CMOS Application Notes • MM54HC/MM74HC MM54HCT/MM74HCT • CD4XXX • MM54CXXX/MM74CXXX • Surface Mount

DATA ACQUISITION LINEAR DEVICES—1989

Active Filters • Analog Switches/Multiplexers • Analog-to-Digital Converters • Digital-to-Analog Converters Sample and Hold • Temperature Sensors • Voltage Regulators • Surface Mount

DATA COMMUNICATION/LAN/UART DATABOOK—1989

LAN IEEE 802.3 • High Speed Serial/IBM Data Communications • ISDN Components • UARTs Modems • Transmission Line Drivers/Receivers

DISCRETE SEMICONDUCTOR PRODUCTS DATABOOK—1989

Selection Guide and Cross Reference Guides • Diodes • Bipolar NPN Transistors Bipolar PNP Transistors • JFET Transistors • Surface Mount Products • Pro-Electron Series Consumer Series • Power Components • Transistor Datasheets • Process Characteristics

DRAM MANAGEMENT HANDBOOK—1989

Dynamic Memory Control • Error Detection and Correction • Microprocessor Applications for the DP8408A/09A/17/18/19/28/29 • Microprocessor Applications for the DP8420A/21A/22A Microprocessor Applications for the NS32CG821

EMBEDDED SYSTEM PROCESSOR DATABOOK—1989

Embedded System Processor Overview • Central Processing Units • Slave Processors • Peripherals Development Systems and Software Tools

F100K DATABOOK—1989

Family Overview • F100K Datasheets • 11C Datasheets • 10K and 100K Memory Datasheets Design Guide • Circuit Basics • Logic Design • Transmission Line Concepts • System Considerations Power Distribution and Thermal Considerations • Testing Techniques • Quality Assurance and Reliability

FACT™ ADVANCED CMOS LOGIC DATABOOK—1989

Description and Family Characteristics • Ratings, Specifications and Waveforms
Design Considerations • 54AC/74ACXXX • 54ACT/74ACTXXX

FAST® ADVANCED SCHOTTKY TTL LOGIC DATABOOK—Rev. 1—1988

Circuit Characteristics • Ratings, Specifications and Waveforms • Design Considerations • 54F/74FXXX

FAST® APPLICATIONS HANDBOOK—REPRINT

Reprint of 1987 Fairchild FAST Applications Handbook
Contains application information on the FAST family: Introduction • Multiplexers • Decoders • Encoders
Operators • FIFOs • Counters • TTL Small Scale Integration • Line Driving and System Design
FAST Characteristics and Testing • Packaging Characteristics • Index

GENERAL PURPOSE LINEAR DEVICES DATABOOK—1989

Continuous Voltage Regulators • Switching Voltage Regulators • Operational Amplifiers • Buffers • Voltage Comparators
Instrumentation Amplifiers • Surface Mount

GRAPHICS HANDBOOK—1989

Advanced Graphics Chipset • DP8500 Development Tools • Application Notes

INTERFACE DATABOOK—1988

Transmission Line Drivers/Receivers • Bus Transceivers • Peripheral Power Drivers • Display Drivers
Memory Support • Microprocessor Support • Level Translators and Buffers • Frequency Synthesis • Hi-Rel Interface

LINEAR APPLICATIONS HANDBOOK—1986

The purpose of this handbook is to provide a fully indexed and cross-referenced collection of linear integrated circuit applications using both monolithic and hybrid circuits from National Semiconductor.

Individual application notes are normally written to explain the operation and use of one particular device or to detail various methods of accomplishing a given function. The organization of this handbook takes advantage of this innate coherence by keeping each application note intact, arranging them in numerical order, and providing a detailed Subject Index.

LS/S/TTL DATABOOK—1989

Contains former Fairchild Products
Introduction to Bipolar Logic • Low Power Schottky • Schottky • TTL • TTL—Low Power

MASS STORAGE HANDBOOK—1989

Rigid Disk Pulse Detectors • Rigid Disk Data Separators/Synchronizers and ENDECs
Rigid Disk Data Controller • SCSI Bus Interface Circuits • Floppy Disk Controllers • Disk Drive Interface Circuits
Rigid Disk Preamplifiers and Servo Control Circuits • Rigid Disk Microcontroller Circuits • Disk Interface Design Guide

MEMORY DATABOOK—1988

PROMs, EPROMs, EEPROMs • Flash EPROMs and EEPROMs • TTL I/O SRAMs
ECL I/O SRAMs • ECL I/O Memory Modules

MICROCONTROLLER DATABOOK—1989

COP400 Family • COP800 Family • COPS Applications • HPC Family • HPC Applications
MICROWIRE and MICROWIRE/PLUS Peripherals • Microcontroller Development Tools

MICROPROCESSOR DATABOOK—1989

Series 32000 Overview • Central Processing Units • Slave Processors • Peripherals
Development Systems and Software Tools • Application Notes • NSC800 Family

PROGRAMMABLE LOGIC DATABOOK & DESIGN MANUAL—1989

Product Line Overview • Datasheets • Designing with PLDs • PLD Design Methodology • PLD Design Development Tools
Fabrication of Programmable Logic • Application Examples

REAL TIME CLOCK HANDBOOK—1989

Real Time Clocks and Timer Clock Peripherals • Application Notes

RELIABILITY HANDBOOK—1986

Reliability and the Die • Internal Construction • Finished Package • MIL-STD-883 • MIL-M-38510
The Specification Development Process • Reliability and the Hybrid Device • VLSI/VHSIC Devices
Radiation Environment • Electrostatic Discharge • Discrete Device • Standardization
Quality Assurance and Reliability Engineering • Reliability and Documentation • Commercial Grade Device
European Reliability Programs • Reliability and the Cost of Semiconductor Ownership
Reliability Testing at National Semiconductor • The Total Military/Aerospace Standardization Program
883B/RETS™ Products • MILS/RETS™ Products • 883/RETS™ Hybrids • MIL-M-38510 Class B Products
Radiation Hardened Technology • Wafer Fabrication • Semiconductor Assembly and Packaging
Semiconductor Packages • Glossary of Terms • Key Government Agencies • AN/ Numbers and Acronyms
Bibliography • MIL-M-38510 and DESC Drawing Cross Listing

SPECIAL PURPOSE LINEAR DEVICES DATABOOK—1989

Audio Circuits • Radio Circuits • Video Circuits • Motion Control Circuits • Special Function Circuits
Surface Mount

TELECOMMUNICATIONS—1987

Line Card Components • Integrated Services Digital Network Components • Modems
Analog Telephone Components • Application Notes

NOTES

NATIONAL SEMICONDUCTOR CORPORATION DISTRIBUTORS

ALABAMA
Huntsville
Arrow Electronics
(205) 837-6955
Bell Industries
(205) 837-1074
Hamilton/Avnet
(205) 837-7210
Pioneer Technology
(205) 837-9300

ARIZONA
Chandler
Hamilton/Avnet
(602) 231-5100
Phoenix
Arrow Electronics
(602) 437-0750
Tempe
Anthem Electronics
(602) 966-6600
Bell Industries
(602) 966-7800

CALIFORNIA
Agora Hills
Zeus Components
(818) 889-3838
Anaheim
Time Electronics
(714) 934-0911
Chatsworth
Anthem Electronics
(818) 700-1000
Arrow Electronics
(818) 701-7500
Hamilton Electro Sales
(818) 700-6500
Time Electronics
(818) 998-7200
Costa Mesa
Avnet Electronics
(714) 754-6050
Hamilton Electro Sales
(714) 641-4159
Garden Grove
Bell Industries
(714) 895-7801
Gardena
Bell Industries
(213) 515-1800
Hamilton/Avnet
(213) 217-6751
Irvine
Anthem Electronics
(714) 768-4444
Ontario
Hamilton/Avnet
(714) 989-4602
Rocklin
Anthem Electronics
(916) 624-9744
Bell Industries
(916) 652-0414
Sacramento
Hamilton/Avnet
(916) 925-2216
San Diego
Anthem Electronics
(619) 453-9005
Arrow Electronics
(619) 565-4800
Hamilton/Avnet
(619) 571-7510
Time Electronics
(619) 586-1331
San Jose
Anthem Electronics
(408) 453-1200
Pioneer Technology
(408) 954-9100
Zeus Components
(408) 998-5121
Sunnyvale
Arrow Electronics
(408) 745-6600
Bell Industries
(408) 734-8570
Hamilton/Avnet
(408) 743-3355
Time Electronics
(408) 734-9888
Thousand Oaks
Bell Industries
(805) 499-6821
Torrance
Time Electronics
(213) 320-0880
Tustin
Arrow Electronics
(714) 838-5422
Yorba Linda
Zeus Components
(714) 921-9000

COLORADO
Englewood
Anthem Electronics
(303) 790-4500
Arrow Electronics
(303) 790-4444
Hamilton/Avnet
(303) 799-7800
Wheatridge
Bell Industries
(303) 424-1985

CONNECTICUT
Cheshire
Time Electronics
(203) 271-3200
Danbury
Hamilton/Avnet
(203) 797-2800
Meriden
Anthem Electronics
(203) 237-2282
Norwalk
Pioneer Standard
(203) 853-1515
Wallingford
Arrow Electronics
(203) 265-7741

FLORIDA
Altamonte Springs
Bell Industries
(407) 339-0078
Pioneer Technology
(407) 834-9090
Clearwater
Pioneer Technology
(813) 536-0445
Deerfield Beach
Arrow Electronics
(305) 429-8200
Bell Industries
(305) 421-1997
Pioneer Technology
(305) 428-8877
Fort Lauderdale
Hamilton/Avnet
(305) 971-2900
Lake Mary
Arrow Electronics
(407) 333-9300
Largo
Bell Industries
(813) 541-4434
Oviedo
Zeus Components
(407) 365-3000
St. Petersburg
Hamilton/Avnet
(813) 576-3930
Winter Park
Hamilton/Avnet
(407) 628-3888

GEORGIA
Norcross
Arrow Electronics
(404) 449-8252
Bell Industries
(404) 662-0923
Hamilton/Avnet
(404) 447-7500
Pioneer Technology
(404) 448-1711

ILLINOIS
Addison
Pioneer Electronics
(312) 437-9680
Bensenville
Hamilton/Avnet
(312) 860-7780
Elk Grove Village
Anthem Electronics
(312) 640-6066
Bell Industries
(312) 640-1910
Itasca
Arrow Electronics
(312) 250-0500
Urbana
Bell Industries
(217) 328-1077
Wood Dale
Time Electronics
(312) 350-0610

INDIANA
Carmel
Hamilton/Avnet
(317) 844-9333
Fort Wayne
Bell Industries
(219) 423-3422
Indianapolis
Advent Electronics Inc.
(317) 872-4910
Arrow Electronics
(317) 243-9353
Bell Industries
(317) 634-8200
Pioneer Standard
(317) 849-7300

IOWA
Cedar Rapids
Advent Electronics
(319) 363-0221
Arrow Electronics
(319) 395-7230
Bell Industries
(319) 395-0730
Hamilton/Avnet
(319) 362-4757

KANSAS
Lenexa
Arrow Electronics
(913) 541-9542
Hamilton/Avnet
(913) 888-8900
Pioneer Standard
(913) 492-0500

MARYLAND
Columbia
Anthem Electronics
(301) 995-6640
Arrow Electronics
(301) 995-0003
Hamilton/Avnet
(301) 995-3500
Time Electronics
(301) 964-3090
Zeus Components
(301) 997-1118
Gaithersburg
Pioneer Technology
(301) 921-0660

MASSACHUSETTS
Andover
Bell Industries
(508) 474-8880
Lexington
Pioneer Standard
(617) 861-9200
Zeus Components
(617) 863-8800
Norwood
Gerber Electronics
(617) 769-6000
Peabody
Hamilton/Avnet
(508) 531-7430
Time Electronics
(508) 532-6200
Wilmington
Anthem Electronics
(508) 657-5170
Arrow Electronics
(508) 658-0900

MICHIGAN
Ann Arbor
Arrow Electronics
(313) 971-8220
Bell Industries
(313) 971-9093
Grand Rapids
Arrow Electronics
(616) 243-0912
Hamilton/Avnet
(616) 243-8805
Pioneer Standard
(616) 698-1800
Livonia
Pioneer Standard
(313) 525-1800
Novi
Hamilton/Avnet
(313) 347-4720
Wyoming
R. M. Electronics, Inc.
(616) 531-9300

MINNESOTA
Eden Prairie
Anthem Electronics
(612) 944-5454
Pioneer Standard
(612) 944-3355
Edina
Arrow Electronics
(612) 830-1800
Minnetonka
Hamilton/Avnet
(612) 932-0600

MISSOURI
Chesterfield
Hamilton/Avnet
(314) 537-1600
St. Louis
Arrow Electronics
(314) 567-6888
Time Electronics
(314) 391-6444

NEW HAMPSHIRE
Hudson
Bell Industries
(603) 882-1133
Manchester
Arrow Electronics
(603) 668-6968
Hamilton/Avnet
(603) 624-9400

NATIONAL SEMICONDUCTOR CORPORATION DISTRIBUTORS (Continued)

NEW JERSEY
Cherry Hill
Hamilton/Avnet
(609) 424-0100
Fairfield
Anthem Electronics
(201) 227-7960
Hamilton/Avnet
(201) 575-3390
Marlton
Arrow Electronics
(609) 596-8000
Parsippany
Arrow Electronics
(201) 538-0900
Pine Brook
Nu Horizons Electronics
(201) 882-8300
Pioneer Standard
(201) 575-3510
Time Electronics
(201) 882-4611

NEW MEXICO
Albuquerque
Alliance Electronics Inc.
(505) 292-3360
Arrow Electronics
(505) 243-4566
Bell Industries
(505) 292-2700
Hamilton/Avnet
(505) 765-1500

NEW YORK
Amityville
Nu Horizons Electronics
(516) 226-6000
Binghamton
Pioneer
(607) 722-9300
Buffalo
Summit Electronics
(716) 887-2800
Fairport
Pioneer Standard
(716) 381-7070
Time Electronics
(716) 383-8853
Hauppauge
Anthem Electronics
(516) 273-1660
Arrow Electronics
(516) 231-1000
Hamilton/Avnet
(516) 434-7413
Time Electronics
(516) 273-0100
Port Chester
Zeus Components
(914) 937-7400
Rochester
Arrow Electronics
(716) 427-0300
Hamilton/Avnet
(716) 475-9130
Summit Electronics
(716) 334-8110
Ronkonkoma
Zeus Components
(516) 737-4500
Syracuse
Hamilton/Avnet
(315) 437-2641
Time Electronics
(315) 432-0355
Westbury
Hamilton/Avnet Export Div.
(516) 997-6868
Woodbury
Pioneer Electronics
(516) 921-8700

NORTH CAROLINA
Charlotte
Pioneer Technology
(704) 527-8188
Time Electronics
(704) 522-7600
Durham
Pioneer Technology
(919) 544-5400
Raleigh
Arrow Electronics
(919) 876-3132
Hamilton/Avnet
(919) 878-0810
Winston-Salem
Arrow Electronics
(919) 725-8711

OHIO
Centerville
Arrow Electronics
(513) 435-5563
Bell Industries
(513) 435-8660
Bell Industries-Military
(513) 434-8231
Cleveland
Pioneer
(216) 587-3600
Dayton
Hamilton/Avnet
(513) 439-6700
Pioneer Standard
(513) 236-9900
Zeus Components
(914) 937-7400
Solon
Arrow Electronics
(216) 248-3990
Hamilton/Avnet
(216) 831-3500
Westerville
Hamilton/Avnet
(614) 882-7004

OKLAHOMA
Tulsa
Arrow Electronics
(918) 252-7537
Hamilton/Avnet
(918) 252-7297
Radio Inc.
(918) 587-9123

OREGON
Beaverton
Almac-Stroum Electronics
(503) 629-8090
Anthem Electronics
(503) 643-1114
Arrow Electronics
(503) 645-6456
Hamilton/Avnet
(503) 627-0201
Lake Oswego
Bell Industries
(503) 635-6500

PENNSYLVANIA
Horsham
Anthem Electronics
(215) 443-5150
Pioneer Technology
(215) 674-4000
King of Prussia
Time Electronics
(215) 337-0900
Monroeville
Arrow Electronics
(412) 856-7000
Pittsburgh
Hamilton/Avnet
(412) 281-4150
Pioneer
(412) 782-2300

TEXAS
Austin
Arrow Electronics
(512) 835-4180
Hamilton/Avnet
(512) 837-8911
Pioneer Standard
(512) 835-4000
Time Electronics
(512) 399-3051
Carrollton
Arrow Electronics
(214) 380-6464
Time Electronics
(214) 241-7441
Dallas
Hamilton/Avnet
(214) 404-9906
Pioneer Standard
(214) 386-7300
Houston
Arrow Electronics
(713) 530-4700
Pioneer Standard
(713) 988-5555
Richardson
Anthem Electronics
(214) 238-7100
Zeus Components
(214) 783-7010
Stafford
Hamilton/Avnet
(713) 240-7733

UTAH
Midvale
Bell Industries
(801) 255-9611
Salt Lake City
Anthem Electronics
(801) 973-8555
Arrow Electronics
(801) 973-6913
Hamilton/Avnet
(801) 972-4300
West Valley
Time Electronics
(801) 973-8181

WASHINGTON
Bellevue
Almac-Stroum Electronics
(206) 643-9992
Bothell
Anthem Electronics
(206) 483-1700
Kent
Arrow Electronics
(206) 575-4420
Redmond
Hamilton/Avnet
(206) 881-6697

WISCONSIN
Brookfield
Arrow Electronics
(414) 792-0150
Mequon
Taylor Electric
(414) 241-4321
Waukesha
Bell Industries
(414) 547-8879
Hamilton/Avnet
(414) 784-4516

CANADA
WESTERN PROVINCES
Burnaby
Hamilton/Avnet
(604) 437-6667
Semad Electronics
(604) 420-9889
Calgary
Hamilton/Avnet
(403) 250-9380
Semad Electronics
(403) 252-5664
Zentronics
(403) 272-1021
Edmonton
Zentronics
(403) 468-9306
Richmond
Zentronics
(604) 273-5575
Saskatoon
Zentronics
(306) 955-2207
Winnipeg
Zentronics
(204) 694-1957

EASTERN PROVINCES
Brampton
Zentronics
(416) 451-9600
Mississauga
Hamilton/Avnet
(416) 677-7432
Nepean
Hamilton/Avnet
(613) 226-1700
Zentronics
(613) 226-8840
Ottawa
Semad Electronics
(613) 727-8325
Pointe Claire
Semad Electronics
(514) 694-0860
St. Laurent
Hamilton/Avnet
(514) 335-1000
Zentronics
(514) 737-9700
Willowdale
ElectroSonic Inc.
(416) 494-1666

SALES OFFICES

ALABAMA
Huntsville
(205) 721-9367

ARIZONA
Tempe
(602) 966-4563

CALIFORNIA
Inglewood
(213) 645-4226
Roseville
(916) 786-5577
San Diego
(619) 587-0666
Santa Clara
(408) 562-5900
Tustin
(714) 259-8880
Woodland Hills
(818) 888-2602

COLORADO
Boulder
(303) 440-3400
Colorado Springs
(303) 578-3319
Englewood
(303) 790-8090

CONNECTICUT
Hamden
(203) 288-1560

FLORIDA
Boca Raton
(407) 997-8133
Orlando
(305) 629-1720
St. Petersburg
(813) 577-1380

GEORGIA
Norcross
(404) 441-2740

ILLINOIS
Schaumburg
(312) 397-8777

INDIANA
Carmel
(317) 843-7160
Fort Wayne
(219) 484-0722

IOWA
Cedar Rapids
(319) 395-0090

KANSAS
Overland Park
(913) 451-4402

MARYLAND
Hanover
(301) 796-8900

MASSACHUSETTS
Burlington
(617) 273-3170

MICHIGAN
Grand Rapids
(616) 940-0588
W. Bloomfield
(313) 855-0166

MINNESOTA
Bloomington
(612) 854-8200

NEW JERSEY
Paramus
(201) 599-0955

NEW MEXICO
Albuquerque
(505) 884-5601

NEW YORK
Fairport
(716) 223-7700
Liverpool
(315) 451-9091
Melville
(516) 351-1000
Wappinger Falls
(914) 298-0680

NORTH CAROLINA
Cary
(919) 481-4311

OHIO
Dayton
(513) 435-6886
Dublin
(614) 766-3679
Independence
(216) 524-5577

ONTARIO
Mississauga
(416) 678-2920
Nepean
(613) 596-0411

OREGON
Portland
(503) 639-5442

PENNSYLVANIA
Horsham
(215) 672-6767

PUERTO RICO
Rio Piedras
(809) 758-9211

QUEBEC
Lachine
(514) 636-8525

TEXAS
Austin
(512) 346-3990
Houston
(713) 771-3547
Richardson
(214) 234-3811

UTAH
Salt Lake City
(801) 322-4747

WASHINGTON
Bellevue
(206) 453-9944

WISCONSIN
Brookfield
(414) 782-1818

National Semiconductor

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: (408) 721-5000
TWX: (910) 339-9240

SALES OFFICES (Continued)

INTERNATIONAL OFFICES

Electronica NSC de Mexico SA
Juventino Rosas No. 118-2
Col Guadalupe Inn
Mexico, 01020 D.F. Mexico
Tel: 52-5-524-9402

National Semicondutores Do Brasil Ltda.
Av. Brig. Faria Lima, 1383
6.0 Andor-Conj. 62
01451 Sao Paulo, SP, Brasil
Tel: (55/11) 212-5066
Fax: (55/11) 211-1181 NSBR BR

National Semiconductor GmbH
Industriestrasse 10
D-8080 Furstenfeldbruck
West Germany
Tel: (0-81-41) 103-0
Telex: 527-649

National Semiconductor (UK) Ltd.
The Maple, Kembrey Park
Swindon, Wiltshire SN2 6UT
United Kingdom
Tel: (07-93) 61-41-41
Telex: 444-674
Fax: (07-93) 69-75-22

National Semiconductor Benelux
Vorstlaan 100
B-1170 Brussels
Belgium
Tel: (02) 6-61-06-80
Telex: 61007

National Semiconductor (UK) Ltd.
Ringager 4A, 3
DK-2605 Brondby
Denmark
Tel: (02) 43-32-11
Telex: 15-179
Fax: (02) 43-31-11

National Semiconductor S.A.
Centre d'Affaires-La Boursidiere
Bâtiment Champagne, B.P. 90
Route Nationale 186
F-92357 Le Plessis Robinson
France
Tel: (1) 40-94-88-88
Telex: 631065
Fax: (1) 40-94-88-11

National Semiconductor (UK) Ltd.
Unit 2A
Clonskeagh Square
Clonskeagh Road
Dublin 14
Tel: (01) 69-55-89
Telex: 91047
Fax: (01) 69-55-89

National Semiconductor S.p.A.
Strada 7, Palazzo R/3
20089 Rozzano
Milanofiori
Italy
Tel: (02) 8242046/7/8/9

National Semiconductor S.p.A.
Via del Cararaggio, 107
00147 Rome
Italy
Tel: (06) 5-13-48-80
Fax: (06) 5-13-79-47

National Semiconductor (UK) Ltd.
P.O. Box 29
N-1321 Stabekk
Norway
Tel: (2) 12-53-70
Fax: (2) 12-53-75

National Semiconductor AB
Box 2016
Stensatravagen 13
S-12702 Skarholmen
Sweden
Tel: (08) 970190
Telex: 10731

National Semiconductor
Calle Agustin de Foxa, 27
28036 Madrid
Spain
Tel: (01) 733-2958
Telex: 46133

National Semiconductor Switzerland
Alte Winterthurerstrasse 53
Postfach 567
Ch-8304 Wallisellen-Zurich
Switzerland
Tel: (01) 830-2727
Telex: 828-444

National Semiconductor
Kauppakartanonkatu 7 A22
SF-00930 Helsinki
Finland
Tel: (90) 33-80-33
Telex: 126116

National Semiconductor
Postbus 90
1380 AB Weesp
The Netherlands
Tel: (0-29-40) 3-04-48
Telex: 10-956
Fax: (0-29-40) 3-04-30

National Semiconductor Japan Ltd.
Sanseido Bldg. 5F
4-15 Nishi Shinjuku
Shinjuku-ku
Tokyo 160 Japan
Tel: 3-299-7001
Fax: 3-299-7000

National Semiconductor Hong Kong Ltd.
Suite 513, 5th Floor,
Chinachem Golden Plaza,
77 Mody Road, Tsimshatsui East,
Kowloon, Hong Kong
Tel: 3-7231290
Telex: 52996 NSSEA HX
Fax: 3-3112536

National Semiconductor (Australia) PTY, Ltd.
1st Floor, 441 St. Kilda Rd.
Melbourne, 3004
Victory, Australia
Tel: (03) 267-5000
Fax: 61-3-2677458

National Semiconductor (PTE), Ltd.
200 Cantonment Road 13-01
Southpoint
Singapore 0208
Tel: 2252226
Telex: RS 33877

National Semiconductor (Far East) Ltd.
Taiwan Branch
P.O. Box 68-332 Taipei
7th Floor, Nan Shan Life Bldg.
302 Min Chuan East Road,
Taipei, Taiwan R.O.C.
Tel: (86) 02-501-7227
Telex: 22837 NSTW
Cable: NSTW TAIPEI

National Semiconductor (Far East) Ltd.
Korea Branch
13th Floor, Dai Han Life Insurance
63 Building,
60, Yoido-dong, Youngdeungpo-ku,
Seoul, Korea 150-763
Tel: (02) 784-8051/3, 785-0696/8
Telex: 24942 NSPKLO
Fax: (02) 784-8054

 TL2649 RRD/RRD55M089/Printed in U.S.A.

National Semiconductor

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: (408) 721-5000
TWX: (910) 339-9240

SALES OFFICES (Continued)

INTERNATIONAL OFFICES

Electronica NSC de Mexico SA
Juventino Rosas No. 118-2
Col Guadalupe Inn
Mexico, 01020 D.F. Mexico
Tel: 52-5-524-9402

National Semicondutores Do Brasil Ltda.
Av. Brig. Faria Lima, 1383
6.0 Andor-Conj. 62
01451 Sao Paulo, SP, Brasil
Tel: (55/11) 212-5066
Fax: (55/11) 211-1181 NSBR BR

National Semiconductor GmbH
Industriestrasse 10
D-8080 Furstenfeldbruck
West Germany
Tel: (0-81-41) 103-0
Telex: 527-649

National Semiconductor (UK) Ltd.
The Maple, Kembrey Park
Swindon, Wiltshire SN2 6UT
United Kingdom
Tel: (07-93) 61-41-41
Telex: 444-674
Fax: (07-93) 69-75-22

National Semiconductor Benelux
Vorstlaan 100
B-1170 Brussels
Belgium
Tel: (02) 6-61-06-80
Telex: 61007

National Semiconductor (UK) Ltd.
Ringager 4A, 3
DK-2605 Brondby
Denmark
Tel: (02) 43-32-11
Telex: 15-179
Fax: (02) 43-31-11

National Semiconductor S.A.
Centre d'Affaires-La Boursidiere
Bâtiment Champagne, B.P. 90
Route Nationale 186
F-92357 Le Plessis Robinson
France
Tel: (1) 40-94-88-88
Telex: 631065
Fax: (1) 40-94-88-11

National Semiconductor (UK) Ltd.
Unit 2A
Clonskeagh Square
Clonskeagh Road
Dublin 14
Tel: (01) 69-55-89
Telex: 91047
Fax: (01) 69-55-89

National Semiconductor S.p.A.
Strada 7, Palazzo R/3
20089 Rozzano
Milanofiori
Italy
Tel: (02) 8242046/7/8/9

National Semiconductor S.p.A.
Via del Cararaggio, 107
00147 Rome
Italy
Tel: (06) 5-13-48-80
Fax: (06) 5-13-79-47

National Semiconductor (UK) Ltd.
P.O. Box 29
N-1321 Stabekk
Norway
Tel: (2) 12-53-70
Fax: (2) 12-53-75

National Semiconductor AB
Box 2016
Stensatravagen 13
S-12702 Skarholmen
Sweden
Tel: (08) 970190
Telex: 10731

National Semiconductor
Calle Agustin de Foxa, 27
28036 Madrid
Spain
Tel: (01) 733-2958
Telex: 46133

National Semiconductor Switzerland
Alte Winterthurerstrasse 53
Postfach 567
Ch-8304 Wallisellen-Zurich
Switzerland
Tel: (01) 830-2727
Telex: 828-444

National Semiconductor
Kauppakartanonkatu 7 A22
SF-00930 Helsinki
Finland
Tel: (90) 33-80-33
Telex: 126116

National Semiconductor
Postbus 90
1380 AB Weesp
The Netherlands
Tel: (0-29-40) 3-04-48
Telex: 10-956
Fax: (0-29-40) 3-04-30

National Semiconductor Japan Ltd.
Sanseido Bldg. 5F
4-15 Nishi Shinjuku
Shinjuku-ku
Tokyo 160 Japan
Tel: 3-299-7001
Fax: 3-299-7000

National Semiconductor Hong Kong Ltd.
Suite 513, 5th Floor,
Chinachem Golden Plaza,
77 Mody Road, Tsimshatsui East,
Kowloon, Hong Kong
Tel: 3-7231290
Telex: 52996 NSSEA HX
Fax: 3-3112536

National Semiconductor (Australia) PTY, Ltd.
1st Floor, 441 St. Kilda Rd.
Melbourne, 3004
Victory, Australia
Tel: (03) 267-5000
Fax: 61-3-2677458

National Semiconductor (PTE), Ltd.
200 Cantonment Road 13-01
Southpoint
Singapore 0208
Tel: 2252226
Telex: RS 33877

National Semiconductor (Far East) Ltd.
Taiwan Branch
P.O. Box 68-332 Taipei
7th Floor, Nan Shan Life Bldg.
302 Min Chuan East Road,
Taipei, Taiwan R.O.C.
Tel: (86) 02-501-7227
Telex: 22837 NSTW
Cable: NSTW TAIPEI

National Semiconductor (Far East) Ltd.
Korea Branch
13th Floor, Dai Han Life Insurance
63 Building,
60, Yoido-dong, Youngdeungpo-ku,
Seoul, Korea 150-763
Tel: (02) 784-8051/3, 785-0696/8
Telex: 24942 NSPKLO
Fax: (02) 784-8054

 TL2649 RRD/RRD55M089/Printed in U.S.A.