National Semiconductor

400016

Embedded System Processor Databook

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.

Charlie Sporck

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 produisànt 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.

Charlie Sporck

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

Embedded System Processor

Databook

1989 Edition

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.

Table of Contents

Alpha-Numeric Index

Section 1

Embedded System Processor Overview

Section 1 Contents

National Semiconductor

Introduction

National's Embedded System Processor™ family offers the most complete solution to your 32-bit embedded processor 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 Embedded System Processors to effectively meet the needs of an embedded system designer.

This Databook presents technical descriptions of our 32-bit Embedded System Processors, 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 Embedded System Processor.

When we at National Semiconductor began designing the Embedded System Processor family, we decided to support an architecture that addressed the needs of embedded design. We chose to take the time to design it properly so that optimal system cost/performance, high system integration, and total system solutions were addressed. Working from the top down, we analyzed the issues and anticipated the embedded computing needs. The result is an advanced and efficient family of Embedded System Processors.

Software productivity has become a major issue in embedded system product development. In embedded systems this issue centers around the capability of the processor to maximize the utility of software relative to shorter development cycles, under the constraints of lower cost and higher performance.

In short, the degree to which the processor 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. National Semiconductor's Embedded System Processor family combines 32-bit performance with efficient management of a large address space. It facilitates high-level language program development and efficient instruction execution. Floating-point is integrated into the architecture.

But we didn't stop there. Advanced architecture isn't enough. Our total product system solution approach includes the hardware, software, and development support products necessary for your design. The evaluation board, in-system emulator, software development tools, and third party software are available now for your evaluation and development.

The Embedded System Processor 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 National's Embedded System Processors™

Some of the features that set the Embedded System Processor family apart as the best choice for 32-bit designs are as follows:

FAMILY OF EMBEDDED SYSTEM PROCESSORS

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

CLEANEST 32-BIT OPTIMIZED ARCHITECTURE

The Embedded System Processor 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

Embedded System Processor 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

National offers a complete set of floating-point solutions. This includes the NS32081 Floating-Point Unit, and the NS32381 Floating-Point Unit. 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.

HIGH-LEVEL LANGUAGE SUPPORT

National's Embedded System Processor 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. Also, modular programming is supported by special hardware registers, software instructions, an external addressing mode, and architecturally supported link tables.

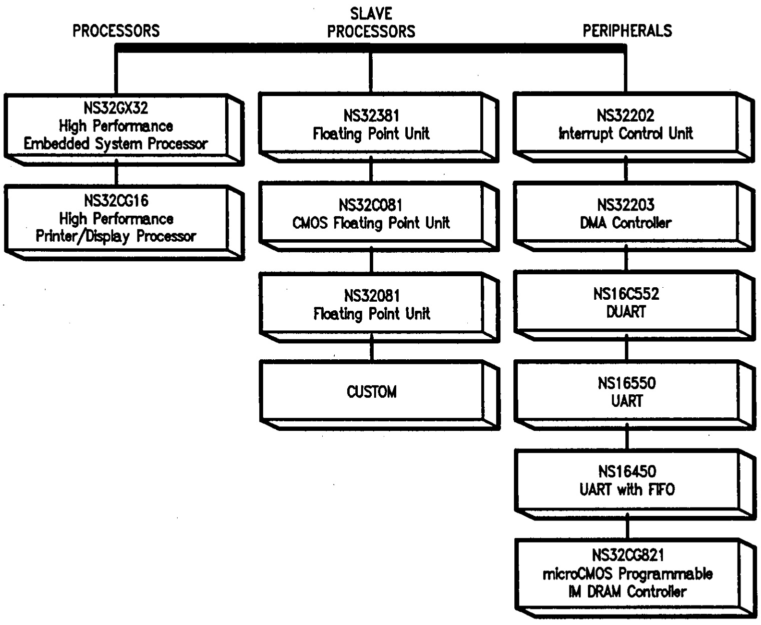
National Semiconductor

Component Descriptions

Device	Description	Bus Width			Process	Package Type
		Internal	External			
			Address	Data		
CENTRAL PROCESSING UNITS (CPU's)						
NS32GX32	High-Performance 32-Bit Embedded System Processor	32	32	32	M²CMOS	175-pin PGA
NS32CG16	High-Performance Printer/Display Processor	32	24	16	CMOS	68-pin PCC
SLAVE PROCESSORS						
NS32081	Floating-Point Unit	64	—	16	XMOS	24-pin DIP Dual-In-Line Package
NS32381	Floating-Point Unit	64	—	16	CMOS	68-pin PGA
PERIPHERALS						
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

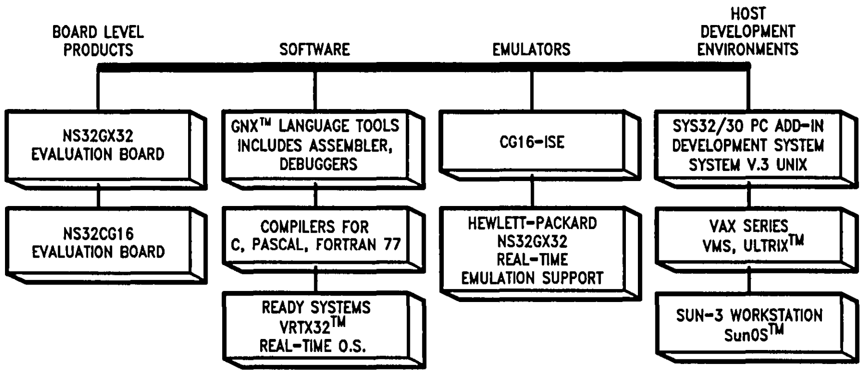
Hardware Chart


PROCESSORS
SLAVE PROCESSORS
PERIPHERALS
NS32GX32 High Performance Embedded System Processor
NS32CG16 High Performance Printer/Display Processor
NS32381 Floating Point Unit
NS32C081 CMOS Floating Point Unit
NS32081 Floating Point Unit
CUSTOM
NS32202 Interrupt Control Unit
NS32203 DMA Controller
NS16C552 DUART
NS16550 UART
NS16450 UART with FIFO
NS32CG821 microCMOS Programmable IM DRAM Controller

TL/XX/0164-1

National Semiconductor

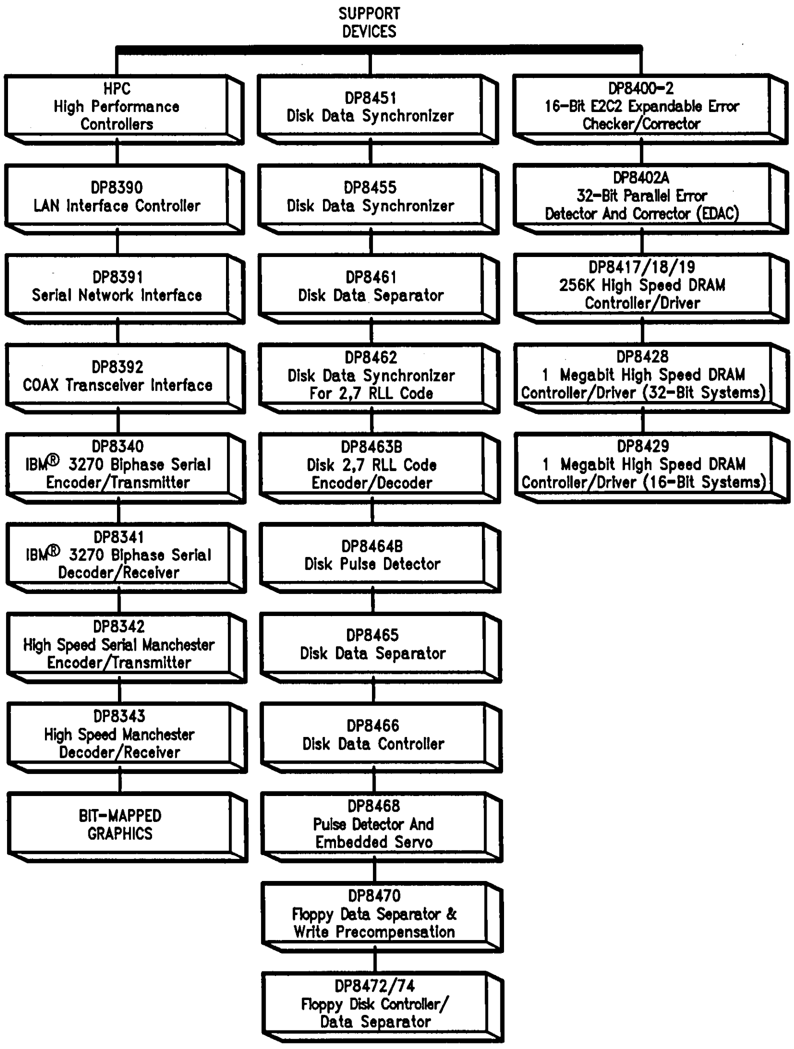
Systems and Software Chart


BOARD LEVEL PRODUCTS
SOFTWARE
EMULATORS
HOST DEVELOPMENT ENVIRONMENTS
NS32GX32 EVALUATION BOARD
NS32CG16 EVALUATION BOARD
GNX™ LANGUAGE TOOLS INCLUDES ASSEMBLER, DEBUGGERS
COMPILERS FOR C, PASCAL, FORTRAN 77
READY SYSTEMS VRTX32™ REAL-TIME O.S.
CG16-ISE
HEWLETT-PACKARD NS32GX32 REAL-TIME EMULATION SUPPORT
SYS32/30 PC ADD-IN DEVELOPMENT SYSTEM SYSTEM V.3 UNIX
VAX SERIES VMS, ULTRIX™
SUN-3 WORKSTATION SunOS™

TL/XX/0165-1

National Semiconductor

Support Devices Chart


SUPPORT DEVICES
HPC High Performance Controllers
DP8390 LAN Interface Controller
DP8391 Serial Network Interface
DP8392 COAX Transceiver Interface
DP8340 IBM® 3270 Biphase Serial Encoder/Transmitter
DP8341 IBM® 3270 Biphase Serial Decoder/Receiver
DP8342 High Speed Serial Manchester Encoder/Transmitter
DP8343 High Speed Manchester Decoder/Receiver
BIT-MAPPED GRAPHICS
DP8451 Disk Data Synchronizer
DP8455 Disk Data Synchronizer
DP8461 Disk Data Separator
DP8462 Disk Data Synchronizer For 2,7 RLL Code
DP8463B Disk 2,7 RLL Code Encoder/Decoder
DP8464B Disk Pulse Detector
DP8465 Disk Data Separator
DP8466 Disk Data Controller
DP8468 Pulse Detector And Embedded Servo
DP8470 Floppy Data Separator & Write Precompensation
DP8472/74 Floppy Disk Controller/ Data Separator
DP8400-2 16-Bit E2C2 Expandable Error Checker/Corrector
DP8402A 32-Bit Parallel Error Detector And Corrector (EDAC)
DP8417/18/19 256K High Speed DRAM Controller/Driver
DP8428 1 Megabit High Speed DRAM Controller/Driver (32-Bit Systems)
DP8429 1 Megabit High Speed DRAM Controller/Driver (16-Bit Systems)

TL/XX/0166-1

Section 2

CPU—Central Processing Units

Section 2 Contents

National Semiconductor

PRELIMINARY

NS32GX32-20/NS32GX32-25/NS32GX32-30 High-Performance 32-Bit Embedded System Processor

General Description

The NS32GX32 is a high-performance 32-bit embedded system processor in the Series 32000® family. It is software compatible with the previous microprocessors in the family but with a greatly enhanced internal implementation.

The NS32GX32 integrates more than 320,000 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 peak performance of 15 million instructions per second.

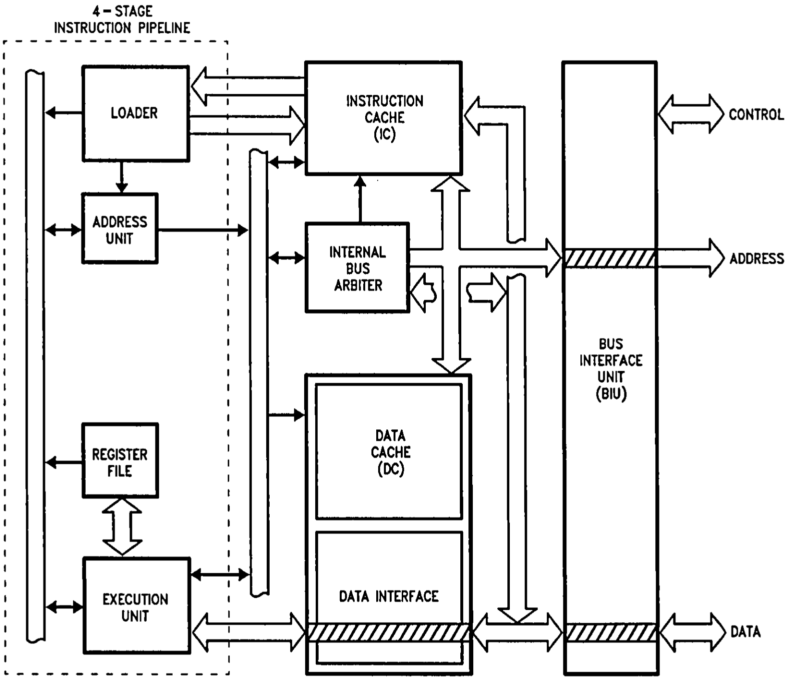
The high-performance specifications are the result of a four-stage instruction pipeline, on-chip instruction and data caches, and a significantly increased clock frequency. In addition, the system interface provides optimal support for applications spanning a wide range, from low-cost, real-time controllers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

In addition to generally improved performance, the NS32GX32 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
- 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
- Floating-point support via the NS32381
- 1.25 μm double-metal CMOS technology
- 175-pin PGA package

Block Diagram



TL/EE/10253-1

FIGURE 1

Table of Contents

Table of Contents (Continued)

List of Illustrations

List of Illustrations (Continued)

List of Tables

1.0 Product Introduction

The NS32GX32 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 NS32GX32 is fully software compatible will all the other Series 32000 CPUs. The architectural features of the Series 32000 family and particularly the NS32GX32 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.

Large, Uniform Addressing. The NS32GX32 has 32-bit address pointers that can address up to 4 gigabytes without requiring any segmentation.

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 NS32GX32 CPU has 21 internal registers grouped according to functions as follows: 8 general purpose, 7 address, 1 processor status, 1 configuration, 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 NS32GX32 internal registers.

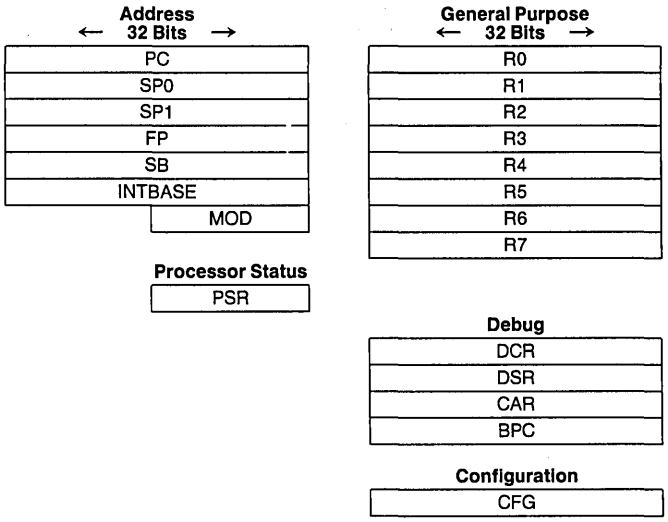


FIGURE 2-1. NS32GX32 Internal Registers

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 NS32GX32 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 NS32GX32 supports only the 32-bit slave protocol and no memory management: consequently these bits are forced to 1.

When the CFG register is loaded using the LPRi instruction, bit 2 and bits 13 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 12 are forced to 0. Bit 2 must be set 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.

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).

31	12				8	7							0
Reserved	LIC	IC	LDC	DC	DE	1	1	1	1	C	Res	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 Debug Registers

The NS32GX32 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 2-4;* 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
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 NS32GX32 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 2-5;* 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: If an address compare is detected for a read and write for the same instruction, 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.

15 8	7							0
Reserved	CAE	CRD	CWR	Res	CBE3	CBE2	CBE1	CBE0

31 24	23							16
Reserved	DEN	SD	UD	PCE	TR	BCP	SI	Res

FIGURE 2-4. Debug Condition Register (DCR)

31			28	27 0
RD	BPC	BEX	BCA	Reserved

FIGURE 2-5. 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 NS32GX32 implements full 32-bit addresses. This allows the CPU to access up to 4 Gbytes of 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.

7 0
A

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.

15 8	7 0
A+1	A
MSB	LSB

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.

31 24	23 16	15 8	7 0
A+3	A+2	A+1	A
MSB			LSB

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 2-6 shows the NS32GX32 address mapping.

The NS32GX32 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 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 memory (addresses between FF000000 (hex) and FF7FFFFF (hex), inclusive), that is dedicated for memory-mapped I/O. This is because the NS32GX32 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 NS32GX32 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.

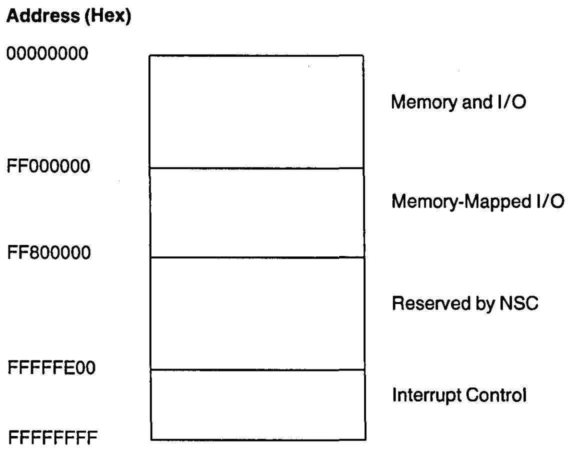


FIGURE 2-6. NS32GX32 Address Mapping

2.0 Architectural Description (Continued)

2.3 MODULAR SOFTWARE SUPPORT

The NS32GX32 provides special support for software modules and modular programs.

Each module in a NS32GX32 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 NS32GX32 uses a module table in memory and two registers in the CPU.

The Module Table is located within the first 64 kbytes of 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:

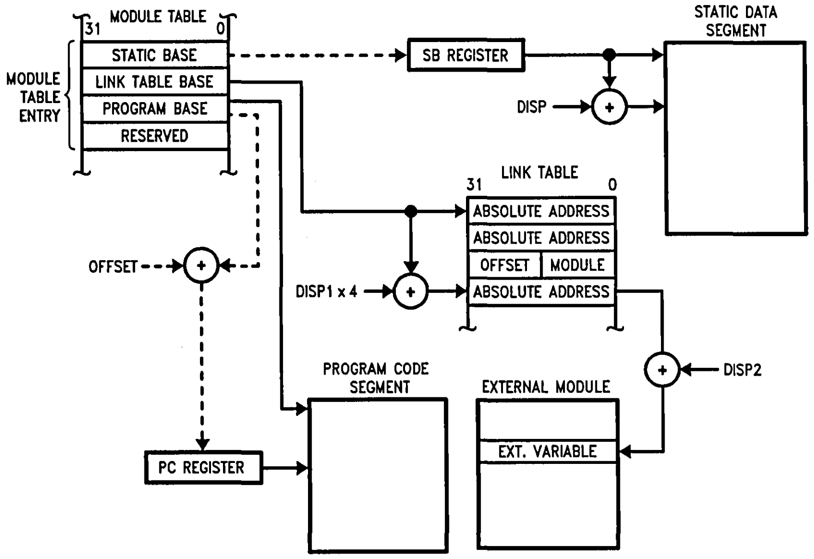
- The Static Base entry contains the address of the beginning of the module's static data segment.
- The Link Table Base points to the beginning of the module's Link Table.
- The Program Base is the address of the beginning of the code and constant data for the module.
- A fourth entry is currently unused 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 NS32GX32 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 2-7* shows a typical NS32GX32 run-time environment.

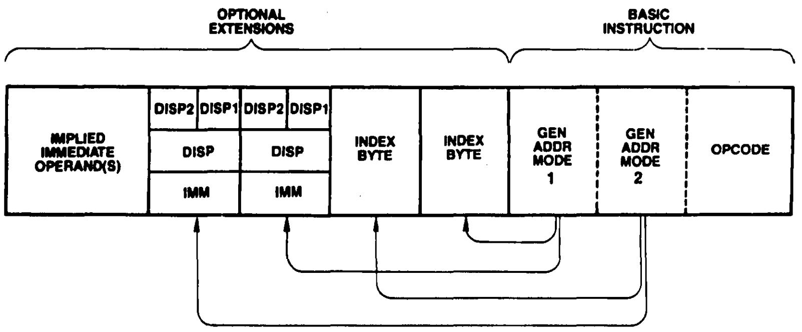


TL/EE/10253-2

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

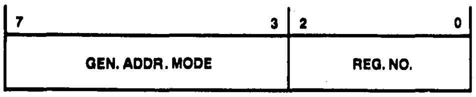
FIGURE 2-7. NS32GX32 Run-Time Environment

2.0 Architectural Description (Continued)



TL/EE/10253-5

FIGURE 2-8. General Instruction Format



TL/EE/10253-6

FIGURE 2-9. Index Byte Format

2.4 INSTRUCTION SET

2.4.1 General Instruction Format

Figure 2-8 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 2-9.*

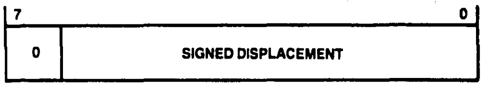
Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Disp/Imm field may contain one or two displacements, or one immediate value. The size of a Displacement field is encoded with the top bits of that field, as shown in *Figure 2-10*, 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 2.4.3).

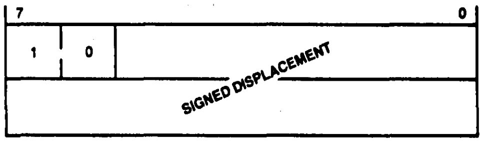
2.4.2 Addressing Modes

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."

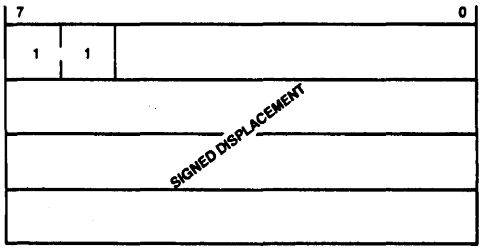
Byte Displacement: Range −64 to +63



Word Displacement: Range −8192 to +8191



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



TL/EE/10253-7

FIGURE 2-10. 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)

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.

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.

2.4.3 Instruction Set Summary

Table 2-3 presents a brief description of the NS32GX32 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.

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. NS32GX32 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. NS32GX32 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. NS32GX32 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. NS32GX32 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.
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

2.0 Architectural Description (Continued)

TABLE 2-3. NS32GX32 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.
14	CINV	options,gen	Cache Invalidate. (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)

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 NS32GX32 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 NS32GX32 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*.

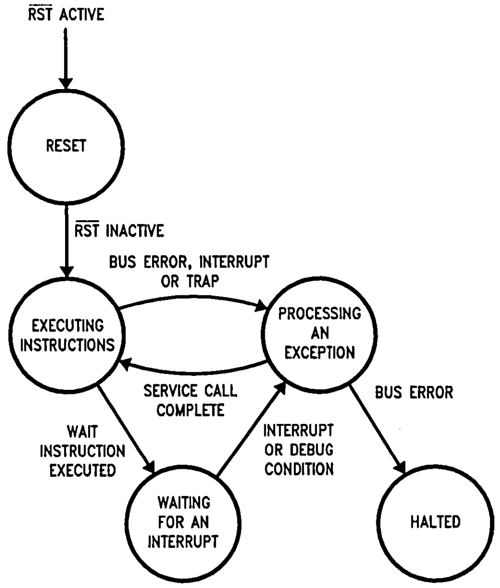
Whenever the $\overline{\text{RST}}$ signal is asserted, the CPU enters the reset state. The CPU remains in the reset state until the $\overline{\text{RST}}$ signal is driven inactive, at which time it enters the Executing-Instructions state. In the Reset state the contents of certain registers are initialized.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



$\overline{\text{RST}}$ ACTIVE
RESET
$\overline{\text{RST}}$ INACTIVE
BUS ERROR, INTERRUPT OR TRAP
EXECUTING INSTRUCTIONS
PROCESSING AN EXCEPTION
SERVICE CALL COMPLETE
BUS ERROR
WAIT INSTRUCTION EXECUTED
INTERRUPT OR DEBUG CONDITION
WAITING FOR AN INTERRUPT
HALTED

TL/EE/10253-8

FIGURE 3-1. Operating States

The CPU enters the Halted state when a bus error 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 NS32GX32 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 (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 NS32GX32 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 is the contents of the PC. 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

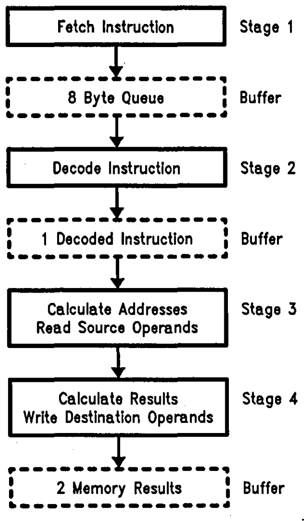
When a restartable bus error, interrupt, or debug condition is recognized during execution of a string instruction, the instruction is said to be partially completed. A partially completed instruction has not completed, but all other instructions executed since the last exception occurred have been completed. 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 NS32GX32 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/10253-9

FIGURE 3-2. NS32GX32 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)

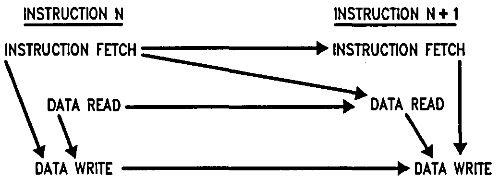
The order of memory references performed by the CPU may also differ from that related to a strictly sequential instruction execution. In fact, when an instruction is being executed, some of the source operands may be read from memory before the instruction is completely fetched. For example, the CPU 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 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/10253-10

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).

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 NS32GX32 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 NS32GX32 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 NS32GX32 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 NS32GX32 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 NS32GX32 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 NS32GX32 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 memory. The NS32GX32 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.

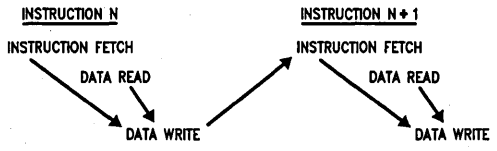
For example, when a new value is loaded into the PSR by executing an LPRW instruction, the pipeline is flushed and a serializing operation takes place. This is necessary since the privilege level might have changed and the instructions following the LPRW instruction must be fetched again with the new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

The CPU serializes instruction execution after executing one of the following instructions: BICPSRW, BISPSRW, BPT, CINV, DIA, FLAG (trap taken), 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/10253-11

FIGURE 3-4. Memory References after Serialization

3.1.4 Slave Processor Instructions

The NS32GX32 recognizes two groups of instructions being executable by external slave processors:

- Floating Point Instructions
- Custom Slave Instructions

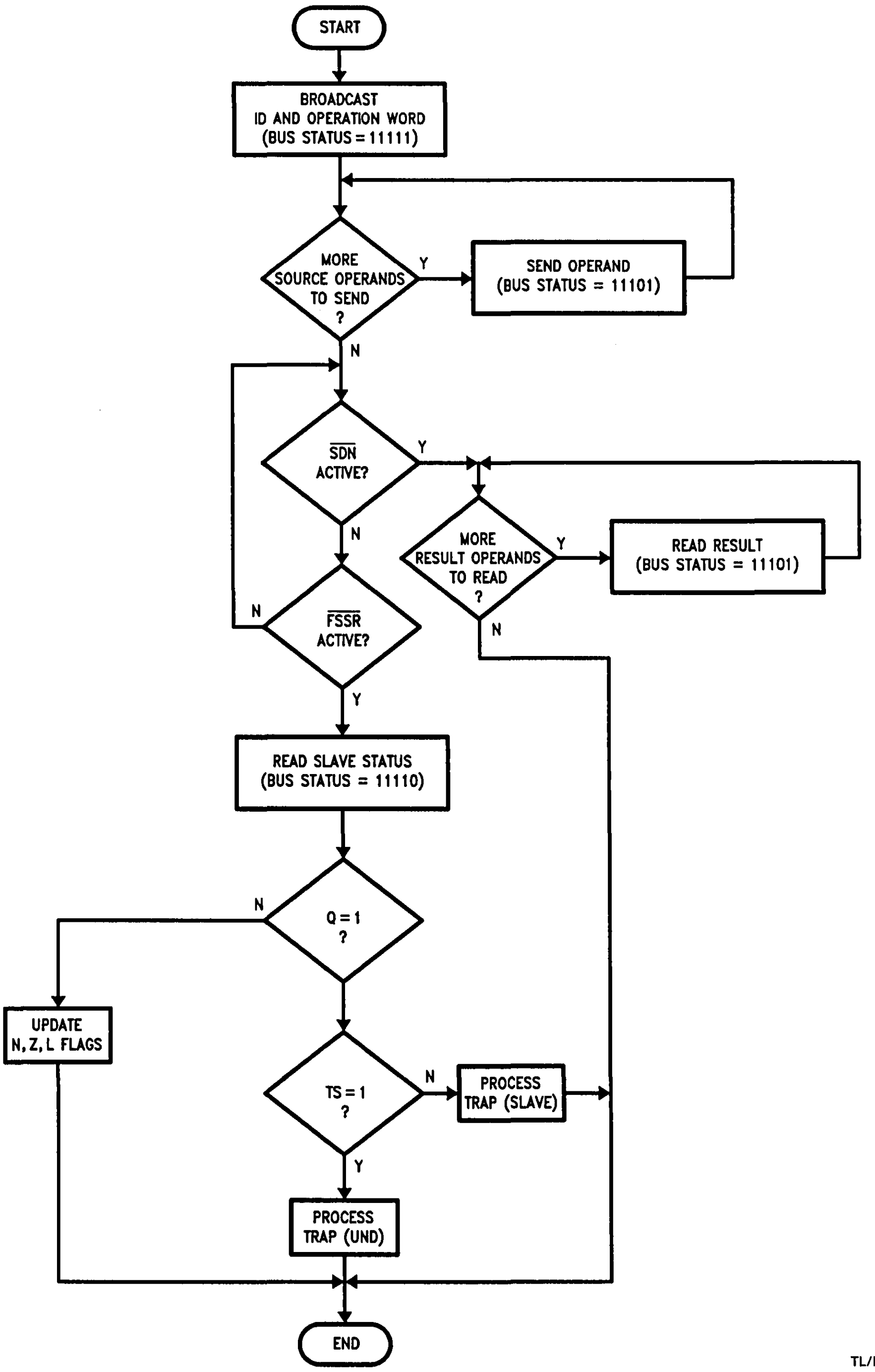
Each Slave Instruction Set is enabled by a bit in the Configuration Register (Section 2.1.4). Any Slave Instruction which does not have its corresponding Configuration Register bit set will trap as undefined, without any Slave Processor communication attempted by the CPU. This allows software simulation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

3.1.4.1 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


START
BROADCAST ID AND OPERATION WORD (BUS STATUS = 11111)
MORE SOURCE OPERANDS TO SEND ?
Y
N
SEND OPERAND (BUS STATUS = 11101)
SDN ACTIVE?
Y
N
MORE RESULT OPERANDS TO READ ?
Y
N
READ RESULT (BUS STATUS = 11101)
FSSR ACTIVE?
N
Y
READ SLAVE STATUS (BUS STATUS = 11110)
Q = 1 ?
N
UPDATE N, Z, L FLAGS
TS = 1 ?
N
Y
PROCESS TRAP (SLAVE)
PROCESS TRAP (UND)
END

TL/EE/10253–12

FIGURE 3-5. 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 NS32GX32 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 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 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.

3.0 Functional Description (Continued)

3.1.4.2 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.

3.1.4.3 Custom Slave Instructions

Provided in the NS32GX32 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

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.

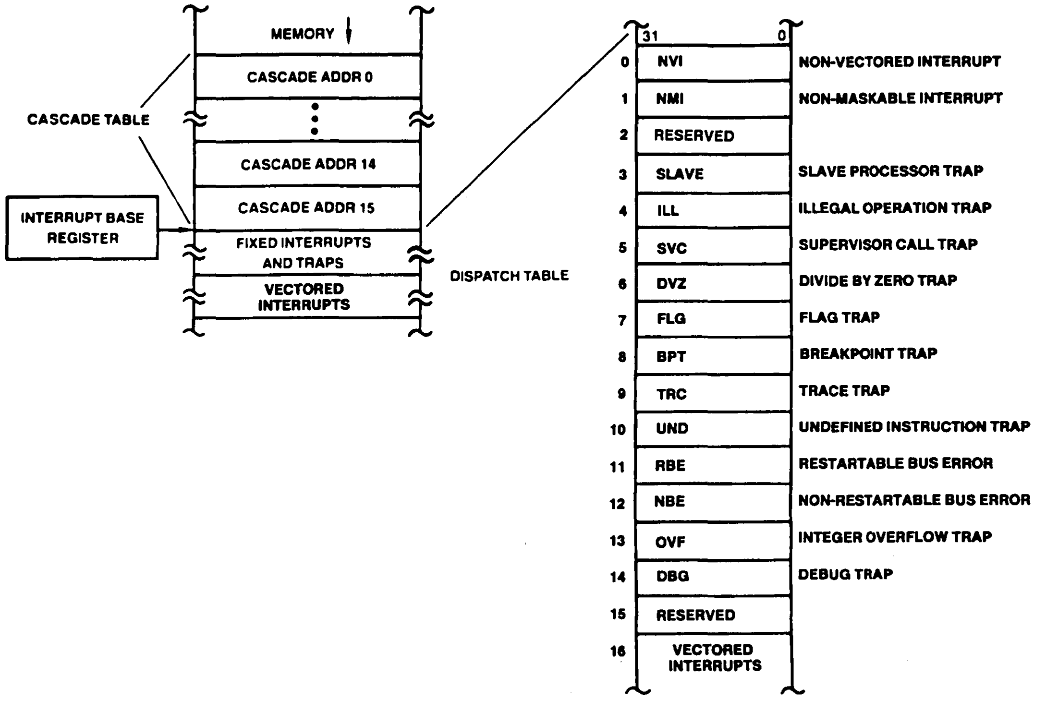
Direct-Exception Mode Disabled

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 3-8* and *3-9*. 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

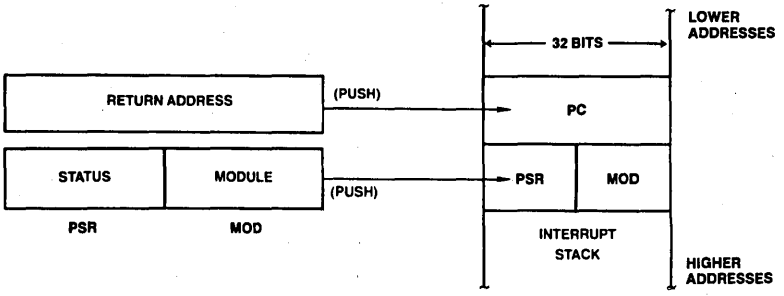
The Direct-Exception mode is enabled when the DE bit in the CFG register is set to 1. In this case the CPU first pushes the saved PSR copy along with the contents of the PC register on the Interrupt Stack. The word stored on the Interrupt Stack between the saved PSR and PC register is reserved for future 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 3-10* provides a pictorial of the acknowledge sequence. It is to be noted that while the


MEMORY
CASCADE ADDR 0
CASCADE TABLE
CASCADE ADDR 14
CASCADE ADDR 15
INTERRUPT BASE REGISTER
FIXED INTERRUPTS AND TRAPS
VECTORED INTERRUPTS
DISPATCH TABLE
31
0
0 NVI NON-VECTORED INTERRUPT
1 NMI NON-MASKABLE INTERRUPT
2 RESERVED
3 SLAVE SLAVE PROCESSOR TRAP
4 ILL ILLEGAL OPERATION TRAP
5 SVC SUPERVISOR CALL TRAP
6 DVZ DIVIDE BY ZERO TRAP
7 FLG FLAG TRAP
8 BPT BREAKPOINT TRAP
9 TRC TRACE TRAP
10 UND UNDEFINED INSTRUCTION TRAP
11 RBE RESTARTABLE BUS ERROR
12 NBE NON-RESTARTABLE BUS ERROR
13 OVF INTEGER OVERFLOW TRAP
14 DBG DEBUG TRAP
15 RESERVED
16 VECTORED INTERRUPTS

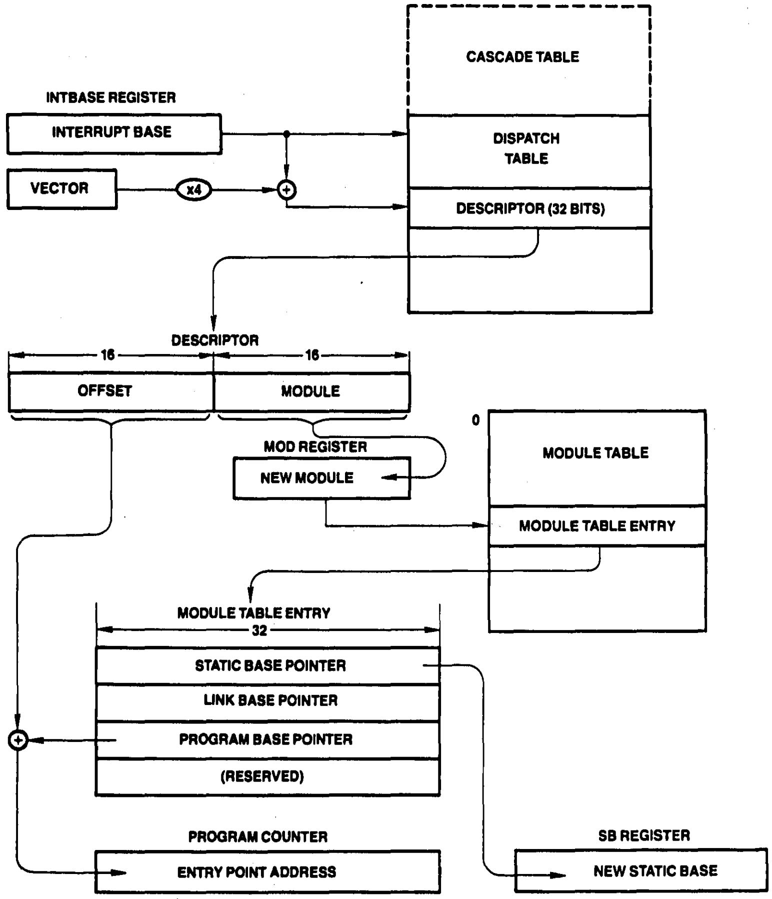
TL/EE/10253-13

FIGURE 3-8. Interrupt Dispatch Table

3.0 Functional Description (Continued)



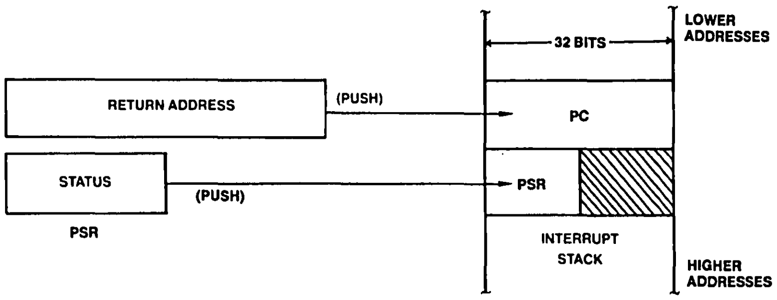
TL/EE/10253-14



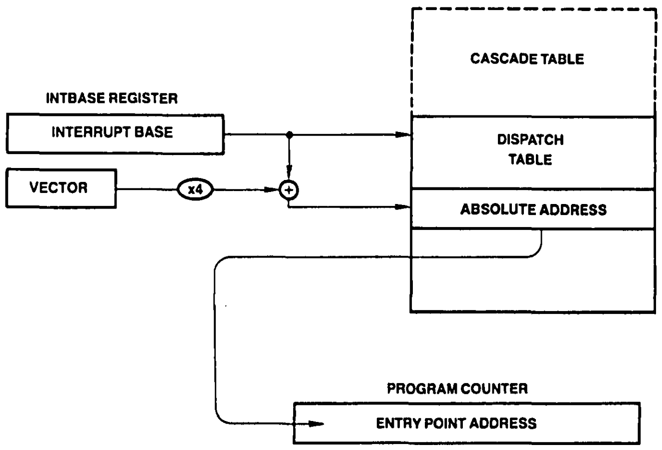
TL/EE/10253-15

FIGURE 3-9. Exception Acknowledge Sequence. Direct-Exception Mode Disabled.

3.0 Functional Description (Continued)



TL/EE/10253-16



TL/EE/10253-17

FIGURE 3-10. 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

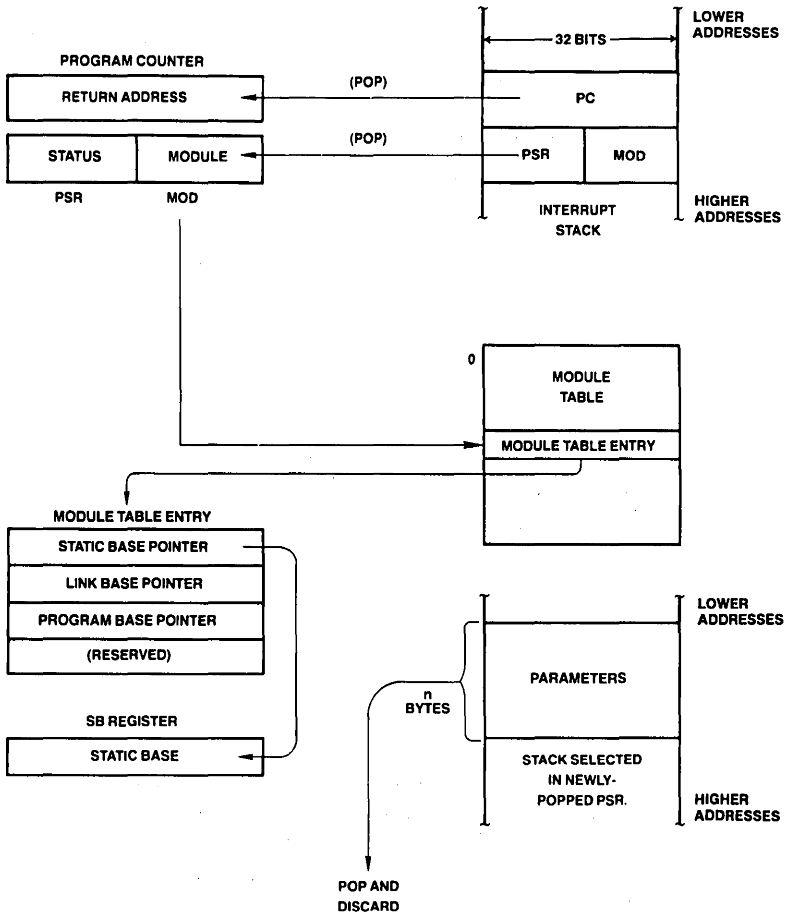
To return control to an interrupted program, one of two instructions can be used: RETT (Return from Trap) and RETI (Return from Interrupt).

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 3-11* and *3-12* show the RETT and RETI instruction flows when the Direct-Exception mode is disabled.

3.0 Functional Description (Continued)



TL/EE/10253-18

FIGURE 3-11. Return from Trap (RETT n) Instruction Flow. Direct-Exception Mode Disabled.

3.2.3 Maskable Interrupts

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

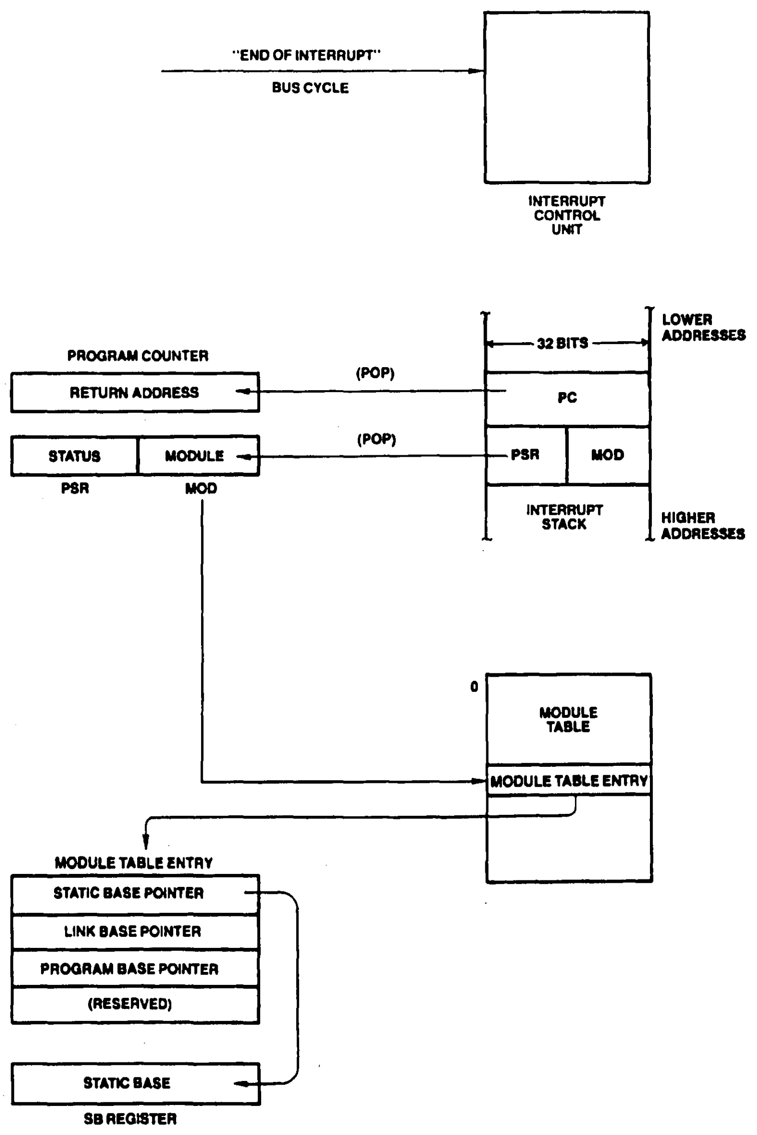
3.2.3.1 Non-Vectored Mode

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.

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 $\overline{\text{INT}}$ 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



TL/EE/10253-19

FIGURE 3-12. 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 $\overline{\text{INT}}$ 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 $\overline{\text{INT}}$ 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 $\overline{\text{NMI}}$ 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 (DBG).

There are 10 trap conditions recognized by the NS32GX32 as described below.

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 reserved general adressing mode encoding (10011) is used.
5. Immediate addressing mode is used for an operand that has access class different from read.

3.0 Functional Description (Continued)

6. Scaled Indexing is used and the basemode is also Scaled Indexing.
7. 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 $\overline{\text{DBG}}$. 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 privileged custom instruction while in User-Mode and the C-bit in the CFG register is 0, then Trap (UND) occurs.

Note 3: 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 4: If an undefined instruction or illegal operation is detected, then no data references are performed for the instruction.

Note 5: 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 $\overline{\text{BER}}$ 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. 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.

In this case, any results that have not yet been written to memory are discarded, and any pending traps other than Trap (DBG) for external condition, are eliminated. The PC value saved on the stack is undefined.

The NS32GX32 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 $\overline{\text{BER}}$ signal during a bus cycle to prefetch an instruction that is not executed because the previous instruction caused a trap.

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 NS32GX32 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 instruction to invalidate the on-chip caches. 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 3-13 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.

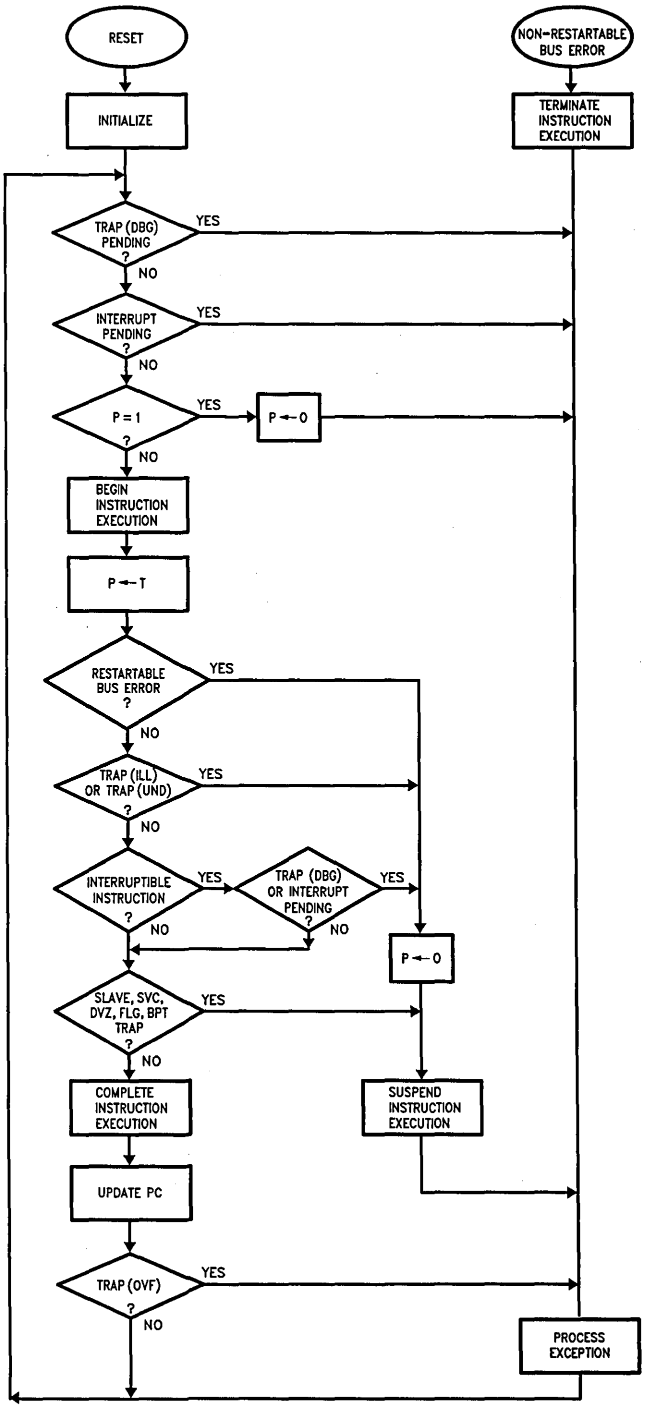
Before executing an instruction, the CPU checks for pending Trap (DBG), interrupts, and Trap (TRC), in that order. If a Trap (DBG) is pending, then the CPU processes that exception, otherwise the CPU checks for pending interrupts. At this point, the CPU responds to any pending interrupt requests; nonmaskable 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 three exceptions:

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

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)


RESET
INITIALIZE
NON-RESTARTABLE BUS ERROR
TERMINATE INSTRUCTION EXECUTION
TRAP (DBG) PENDING ?
YES
NO
INTERRUPT PENDING ?
YES
NO
P = 1 ?
YES
P ← 0
NO
BEGIN INSTRUCTION EXECUTION
P ← T
RESTARTABLE BUS ERROR ?
YES
NO
TRAP (ILL) OR TRAP (UND) ?
YES
NO
INTERRUPTIBLE INSTRUCTION ?
YES
NO
TRAP (DBG) OR INTERRUPT PENDING ?
YES
NO
P ← 0
SLAVE, SVC, DVZ, FLG, BPT TRAP ?
YES
NO
COMPLETE INSTRUCTION EXECUTION
SUSPEND INSTRUCTION EXECUTION
UPDATE PC
TRAP (OVF) ?
YES
NO
PROCESS EXCEPTION

TL/EE/10253-20

FIGURE 3-13. 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 3-14*.

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 $\overline{\text{NMI}}$ pin receives a falling edge, or the $\overline{\text{INT}}$ 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" $\geq$ 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 3-14.*

3.2.8.2 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 11.
5. Set "Return Address" to the address of the first byte of the suspended instruction.
6. Perform Service (Vector, Return Address), *Figure 3-14.*

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 3-14.*

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 3-14.*

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.

3.0 Functional Description (Continued)

3. Set "Return Address" to the address of the next instruction.
4. Perform Service (Vector, Return Address), *Figure 3-14*.

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 3-14*.

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 3-14*.

3.3 DEBUGGING SUPPORT

The NS32GX32 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	TVUSPI
Interrupt	Before Instruction	None/P*	TVUSPI
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 3-14. 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.

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 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 NS32GX32 provides two on-chip caches: the Instruction Cache (IC) and the Data Cache (DC).

These are used to hold the contents of frequently used memory locations.

The IC and DC can be individually enabled by setting appropriate bits in the CFG Register (See Section 2.1.4).

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 3-15*.

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.

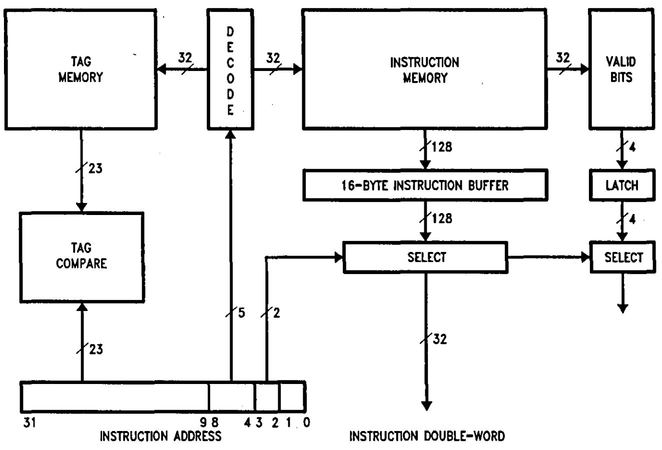
Each block contains a 23-bit tag, which holds the most-significant bits of the physical address for the locations stored in the block, along with 4 double-words and 4 validity bits (one for each double-word).

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.

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/10253-21

FIGURE 3-15. 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. Refer to Section 3.4.3 for details. Clearing the IC bit in the CFG Register also invalidates the instruction cache. Refer to Section C.2 for information on loading the CFG register.

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 3-16.*

Each of the 32 sets has 2 cache blocks. Each block contains a 23-bit tag, which holds the most-significant bits of the 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.
- The reference is not an interlocked read resulting from executing a CBITI or SBITI instruction.

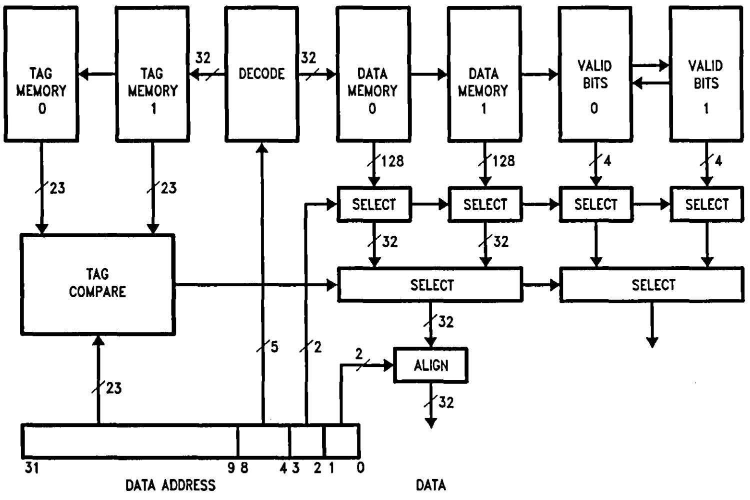
If the DC is disabled, the CPU bypasses it during the data read and its contents are not affected. The data is read directly from external memory. The DC is also bypassed for Interrupt-Acknowledge and End-of-Interrupt bus cycles.

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 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/10253-22

FIGURE 3-16. 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.

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. Clearing the DC bit in the CFG Register also invalidates the data cache. Refer to Section C.2 for information on loading the CFG register.

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 NS32GX32 provides means for maintaining coherence between the on-chip caches and external memory. The CINV instruction can be executed to invalidate 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.

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.

3.5 SYSTEM INTERFACE

This section provides general information on the NS32GX32 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.

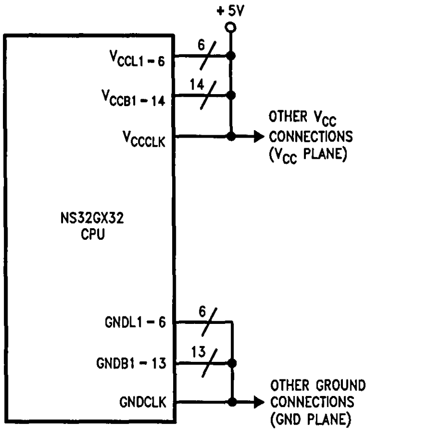
3.0 Functional Description (Continued)

3.5.1 Power and Grounding

The NS32GX32 requires a single 5-volt power supply, applied on 21 pins. The logic voltage pins (VCCL1 to VCCL6) supply 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 NS32GX32 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 3-17*.


+5V
V_{CCL1-6}
6
$V_{CCB1-14}$
14
V_{CCCLK}
OTHER V_{CC} CONNECTIONS (V_{CC} PLANE)
NS32GX32 CPU
GNDL1-6
6
GNDB1-13
13
GNDCLK
OTHER GROUND CONNECTIONS (GND PLANE)

TL/EE/10253-24

FIGURE 3-17. Power and Ground Connections

3.5.2 Clocking

The NS32GX32 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 3-18*, 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.

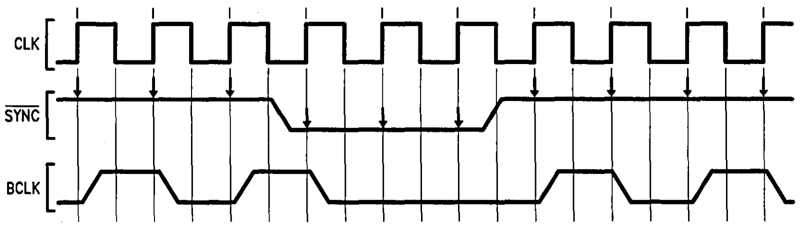
Every rising edge of BCLK defines a transition in the timing state ("T-State") of the CPU.

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 NS32GX32. 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.



TL/EE/10253-25

FIGURE 3-18. Bus Clock Synchronization

3.0 Functional Description (Continued)

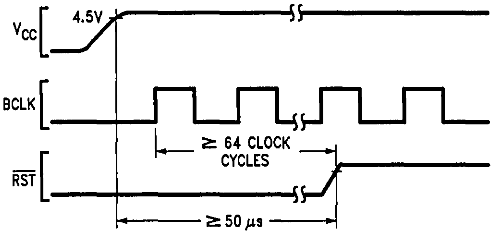
The CPU stores the PC contents in the R0 Register and the PSR contents in the least-significant word of R1, leaving the most-significant word undefined. The PC is then cleared to 0 and so are all the implemented bits in the PSR, MSR, MCR and CFG registers. The DEN-bit in the DCR Register is also cleared 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 3-19* and *3-20*.

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.

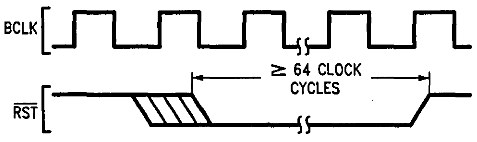
Note 1: If $\overline{HOLD}$ is active at the time $\overline{RST}$ is deasserted, the CPU acknowledges $\overline{HOLD}$ before performing any bus cycle.

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/10253-26

FIGURE 3-19. Power-On Reset Requirements



TL/EE/10253-27

FIGURE 3-20. General Reset Timing

3.5.4 Bus Cycles

The NS32GX32 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 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 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 a 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.

3.0 Functional Description (Continued)

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 $\overline{\text{FSSR}}$ 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 3-21* for a read cycle, and *Figure 3-22* 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 $\overline{\text{RDY}}$ 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 $\overline{\text{ADS}}$ 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 ($\overline{\text{CONF}}$) is asserted in the middle of T1. Note that due to a bus cycle cancellation, the $\overline{\text{BMT}}$ 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 Cache hit occurs. In such a case, the $\overline{\text{CONF}}$ signal remains High and the $\overline{\text{BMT}}$ signal is driven High; the T1-state is followed by another T1-state or an idle T-state.

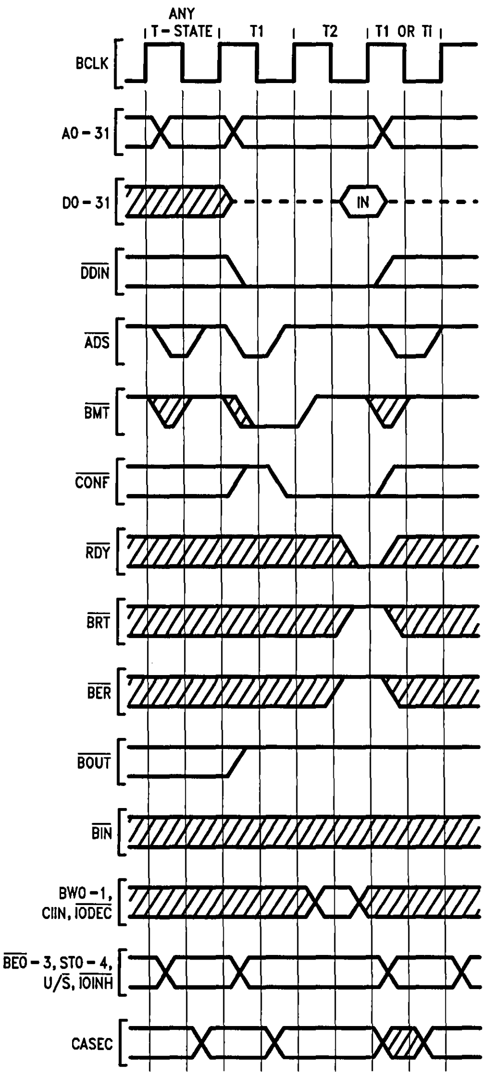
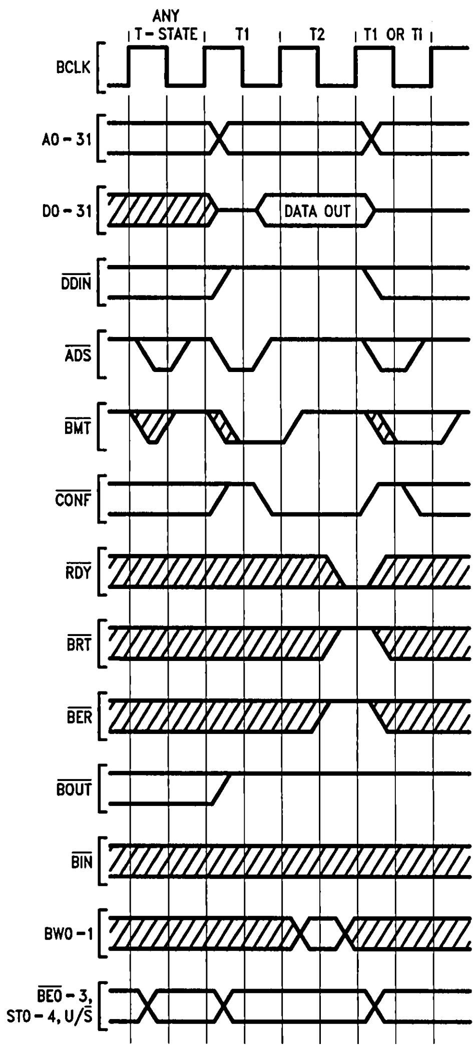


FIGURE 3-21. Basic Read Cycle

3.0 Functional Description (Continued)



TL/EE/10253-29

FIGURE 3-22. Write Cycle

3.5.4.3 Burst Cycles

The NS32GX32 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 3-23*. 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 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 (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 (BIN). BIN is sampled by the CPU in the middle of T2 on the falling edge of BCLK. If the memory does not allow burst (BIN high), the cycle will terminate at the end of T2 and BOUT will go inactive immediately. If the memory allows burst (BIN low), and the CPU has not deasserted BOUT, the second part of the Burst cycle will be performed and 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 (BE0-3) are activated.

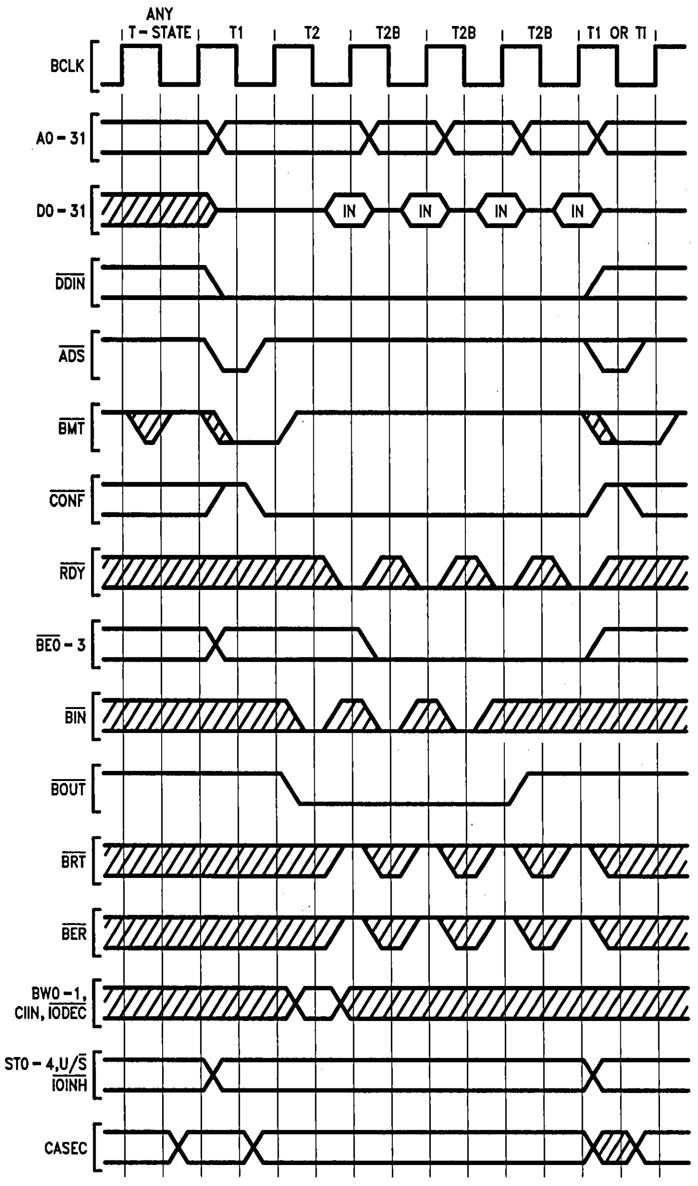
As shown in *Figures 3-23* and *4-8* (in Section 4), the CPU samples RDY at the end of each nibble. It extends the access time for the burst transfer if RDY is inactive.

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 both of the following conditions are met.
   - The data cache is enabled and not locked. (DC = 1 and LDC = 0 in the CFG register.)
   - 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.



TL/EE/10253-30

FIGURE 3-23. Burst Read Cycles

3.0 Functional Description (Continued)

4. The $\overline{\text{BIN}}$ signal is deasserted.
5. $\overline{\text{BRT}}$ is asserted to signal a bus retry.
6. $\overline{\text{IODEC}}$ 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 $\overline{\text{IODEC}}$ 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 $\overline{\text{BER}}$ 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 NS32GX32 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 $\overline{\text{RDY}}$ line is sampled by the CPU. If $\overline{\text{RDY}}$ is active, then the transfer cycle will be completed. If $\overline{\text{RDY}}$ 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 $\overline{\text{BIN}}$, $\overline{\text{BER}}$, $\overline{\text{BRT}}$, BW0–1, CIIN and $\overline{\text{IODEC}}$.

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

Figure 3-24 illustrates a normal read cycle with wait states added through the $\overline{\text{RDY}}$ pin.

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

3.5.4.5 Interlocked Bus Cycles

The NS32GX32 supports indivisible read-modify-write transactions by asserting the $\overline{\text{ILO}}$ 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 $\overline{\text{ILO}}$ 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.

The $\overline{\text{ILO}}$ 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 $\overline{\text{ILO}}$ is deasserted before the next bus cycle begins.

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 3-21*; 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 NS32GX32 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 3-25 shows the timing for a slave write bus cycle. The CPU asserts $\overline{\text{SPC}}$ 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 $\overline{\text{FSSR}}$ signal is asserted. *Figure 3-26* shows the timing for a slave read bus cycle.

During T1 and T2 the CPU drives the status lines and asserts $\overline{\text{SPC}}$. 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.

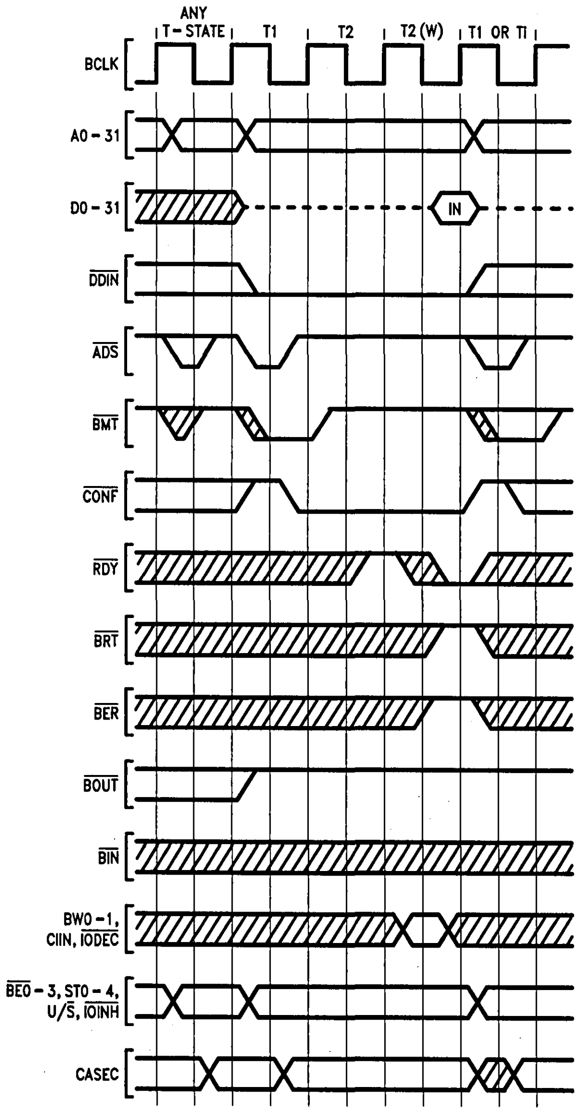
Slave processor data transfers are always 32 bits wide. If the operand is a single byte, then it is transferred on D0 through D7. If it is a word, then it is transferred on D0 through D15.

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 $\overline{\text{BE0–3}}$ are undefined while the input signals BW0–1 and $\overline{\text{RDY}}$ are ignored.

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

3.0 Functional Description (Continued)



TL/EE/10253-31

3-24. Cycle Extension of a Basic Read Cycle

3.0 Functional Description (Continued)

TABLE 3-4. Interrupt Sequences

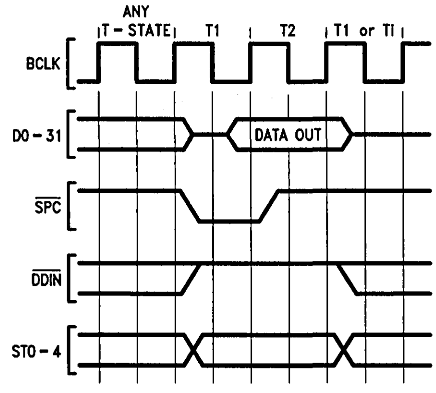
Cycle	Status	Address	$\overline{DDIN}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	Data Bus			
								Byte 3	Byte 2	Byte 1	Byte 0
A. Non-Maskable Interrupt Control Sequences											
Interrupt Acknowledge											
1	00100	$FFFFFF00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
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

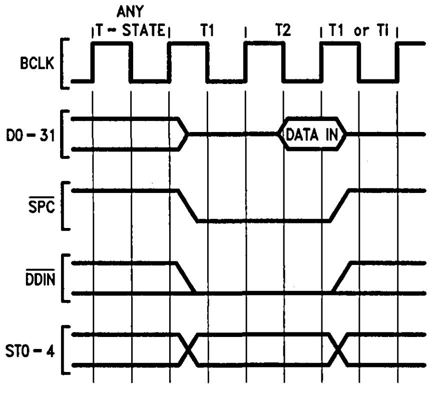
2

3.0 Functional Description (Continued)



TL/EE/10253-32

FIGURE 3-25. Slave Processor Write Cycle



TL/EE/10253-33

FIGURE 3-26. Slave Processor Read Cycle

3.5.5 Bus Exceptions

The NS32GX32 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 3-27 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.

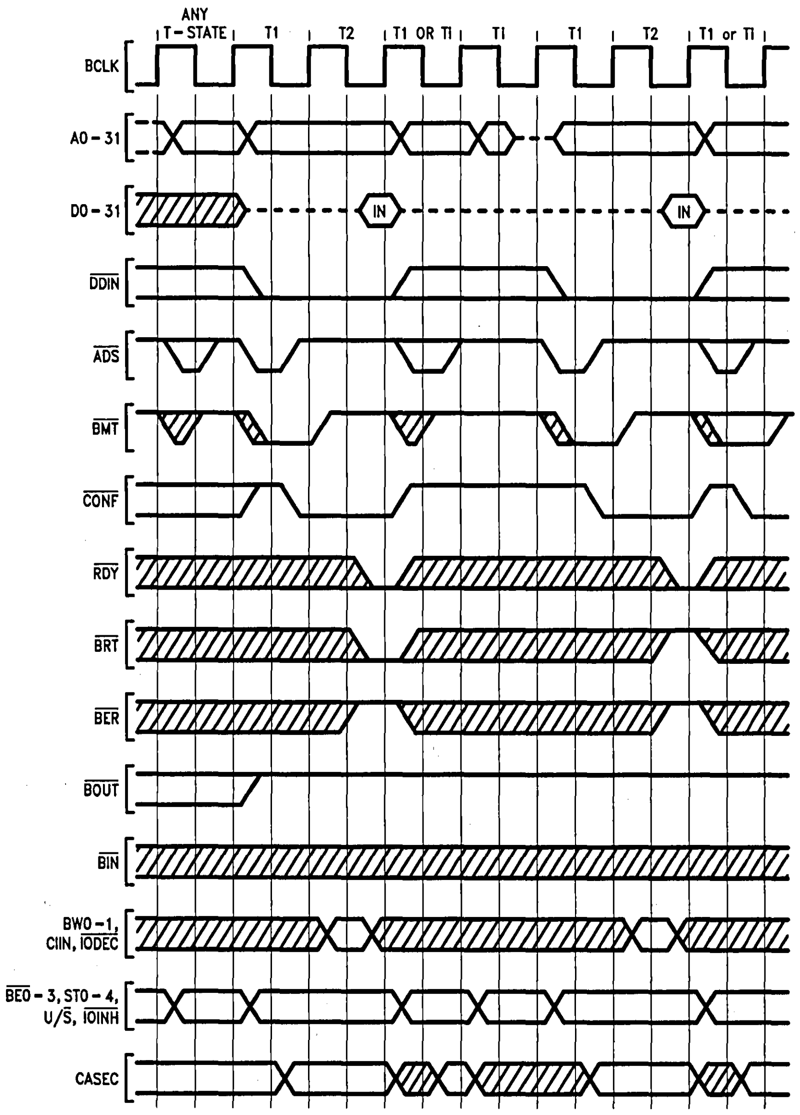
Note 3: If $\overline{BER}$ is asserted during a bus cycle of a multi-cycle data transfer, the CPU completes the entire transfer normally, but the data will be ignored. The CPU also ignores any subsequent assertion of $\overline{BER}$ during the same data transfer.

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 NS32GX32 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.



TL/EE/10253-34

FIGURE 3-27. 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 is always 32 bits during slave cycles (See Section 3.5.4.7). An important feature of the NS32GX32 is that it does not impose any restrictions on the data alignment, regardless of the bus width.

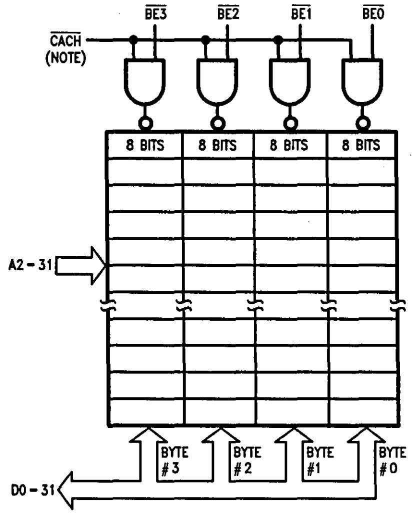
Bus accesses are performed in double-word units. Accesses of data operands that cross double-word boundaries are decomposed into two or more aligned double-word accesses.

The CPU provides four byte enable signals (BE0-3) which facilitate individual byte accessing on either a 32-bit or a 16-bit bus.

Figures 3-28 and *3-29* 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 BE0-3 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 NS32GX32 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 BE0-3 signals are activated accordingly.

The 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 BIN signal is ignored and 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.



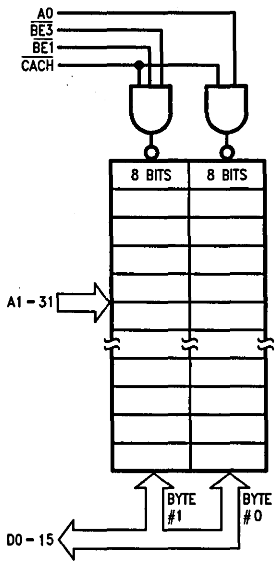
TL/EE/10253-35

FIGURE 3-28. Basic Interface for 32-Bit Memories

Note: The 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 NS32GX32 ignores the BIN signal for 8-bit and 16-bit bus widths, it is recommended that 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/10253-36

FIGURE 3-29. 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 (BE0-3) for both sequential and non-sequential fetches. 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 BOUT and 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{\mathrm{BE}}$0–3 all active.
- Two burst cycles are then performed and addresses 8 and 12 are output while $\overline{\mathrm{BE}}$0–3 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{\mathrm{BE}}$0–3 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{\mathrm{BE}}$0–3 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{\mathrm{BE}}$0–3 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{\mathrm{BE}}$0–3 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{\mathrm{BE}}$0–3	Data Bus			
1	11	B0	—	—	—	A	L L L L	B0	C/I	C/I	C/I
2	10	B1	B0	—	—	A	L L L L	B1	B0	C/I	C/I
3	01	B2	B1	B0	—	A	L L L L	B2	B1	B0	C/I
4	00	B3	B2	B1	B0	A	L L L L	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{\mathrm{BE}}$0–3	Data Bus			
1	11	B0	—	—	—	A	L L L L	—	—	B0	C/I
						*A − 3	L L L L	—	—	C	C
2	10	B1	B0	—	—	A	L L L L	—	—	B1	B0
						*A − 2	L L L L	—	—	C	C
3	01	B2	B1	B0	—	A	L L L L	—	—	B0	C/I
						A + 1	L L L L	—	—	B2	B1
4	00	B3	B2	B1	B0	A	L L L L	—	—	B1	B0
						A + 2	L L L L	—	—	B3	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	L L L L	—	—	—	B0
						*A − 3	L L L L	—	—	—	C
						*A − 2	L L L L	—	—	—	C
						*A − 1	L L L L	—	—	—	C
2	10	B1	B0	—	—	A	L L L L	—	—	—	B0
						A + 1	L L L L	—	—	—	B1
						*A − 2	L L L L	—	—	—	C
						*A − 1	L L L L	—	—	—	C
3	01	B2	B1	B0	—	A	L L L L	—	—	—	B0
						A + 1	L L L L	—	—	—	B1
						A + 2	L L L L	—	—	—	B2
						*A − 1	L L L L	—	—	—	C
4	00	B3	B2	B1	B0	A	L L L L	—	—	—	B0
						A + 1	L L L L	—	—	—	B1
						A + 2	L L L L	—	—	—	B2
						A + 3	L L L L	—	—	—	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	HHHL	HHHL	—	—	C/I	B0
						*A + 2	LLLL		—	—	C	C
1	01	—	—	B0	—	A	HHLH	HHLH	—	—	B0	C/I
						*A + 1	LLLL		—	—	C	C
1	10	—	B0	—	—	A	HLHH	HLHH	—	—	C/I	B0
						*A − 2	LLLL		—	—	C	C
1	11	B0	—	—	—	A	LHHH	LHHH	—	—	B0	C/I
						*A − 3	LLLL		—	—	C	C
2	00	—	—	B1	B0	A	HHLL	HHLL	—	—	B1	B0
						*A + 2	LLLL		—	—	C	C
2	01	—	B1	B0	—	A	HLLH	HLLH	—	—	B0	C/I
						A + 1	LLLL	HLHH	—	—	C/I	B1
2	10	B1	B0	—	—	A	LLHH	LLHH	—	—	B1	B0
						*A − 2	LLLL		—	—	C	C
3	00	—	B2	B1	B0	A	HLLL	HLLL	—	—	B1	B0
						A + 2	LLLL	HLHH	—	—	C/I	B2
3	01	B2	B1	B0	—	A	LLLH	LLLH	—	—	B0	C/I
						A + 1	LLLL	LLHH	—	—	B2	B1
4	00	B3	B2	B1	B0	A	LLLL	LLLL	—	—	B1	B0
						A + 2	LLLL	LLHH	—	—	B3	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 $\overline{BE}1$ and $\overline{BE}2$ active.
- A write at address 6 is then performed with $\overline{BE}2$ 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 $\overline{BE}3$ active.

3.5.7 Bus Access Control

The NS32GX32 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 3-30* 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 3-29*). 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

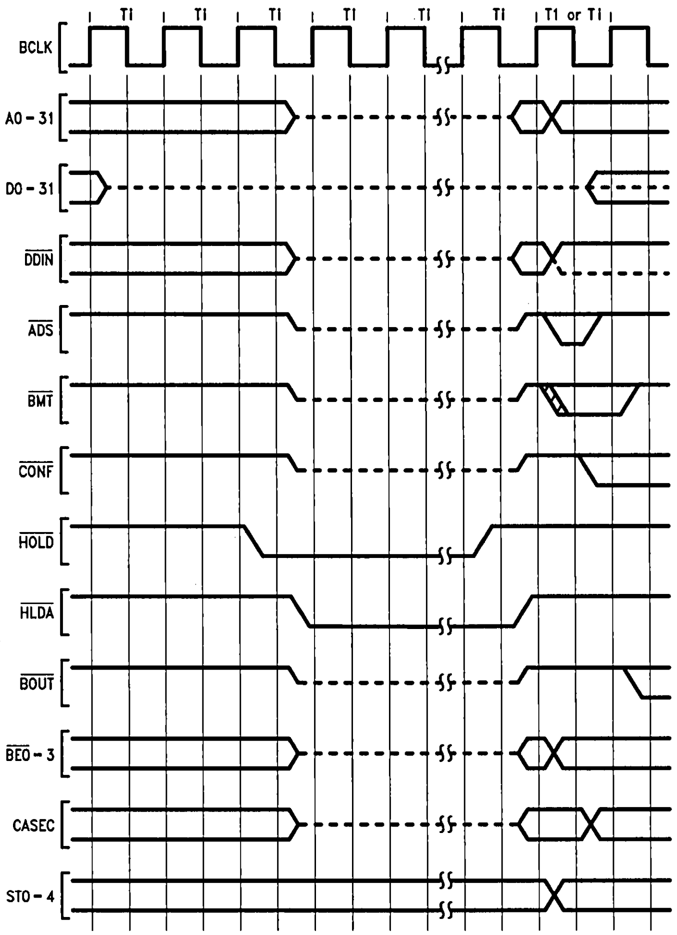
TABLE 3-12. Data Writes to a 16-Bit Bus

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	HLLH	•	B1	B0	B0
						A + 1	HLHH	•	•	•	B1
2	10	B1	B0	—	—	A	LLHH	B1	B0	B1	B0
3	00	—	B2	B1	B0	A	HLLL	•	B2	B1	B0
						A + 2	HLHH	•	•	•	B2
3	01	B2	B1	B0	—	A	LLLH	B2	B1	B0	B0
						A + 1	LLHH	•	•	B2	B1
4	00	B3	B2	B1	B0	A	LLLL	B3	B2	B1	B0
						A + 2	LLHH	•	•	B3	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	HHLL	•	•	B1	B0
						A + 1	HHLH	•	•	•	B1
2	01	—	B1	B0	—	A	HLLH	•	B1	B0	B0
						A + 1	HLHH	•	•	•	B1
2	10	B1	B0	—	—	A	LLHH	B1	B0	B1	B0
						A + 1	LHHH	•	•	•	B1
3	00	—	B2	B1	B0	A	HLLL	•	B2	B1	B0
						A + 1	HLLH	•	•	•	B1
						A + 2	HLHH	•	•	•	B2
3	01	B2	B1	B0	—	A	LLLH	B2	B1	B0	B0
						A + 1	LLHH	•	•	•	B1
						A + 2	LHHH	•	•	•	B2
4	00	B3	B2	B1	B0	A	LLLL	B3	B2	B1	B0
						A + 1	LLLH	•	•	•	B1
						A + 2	LLHH	•	•	•	B2
						A + 3	LHHH	•	•	•	B3



TL/EE/10253-37

FIGURE 3-30. 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 ($\overline{\text{BRT}}$) 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 NS32GX32 due to its internal pipelined implementation. Two special signals are provided for this purpose: $\overline{\text{IOINH}}$ and $\overline{\text{IODEC}}$. The CPU asserts $\overline{\text{IOINH}}$ 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 $\overline{\text{IODEC}}$ to indicate to the CPU that an I/O device has been selected. $\overline{\text{IODEC}}$ is sampled by the CPU in the middle of state T2. If the cycle is extended, then the CPU uses the $\overline{\text{IODEC}}$ value sampled during the last wait state. If a bus error or a bus retry occurs, the sampled $\overline{\text{IODEC}}$ value is ignored. $\overline{\text{IODEC}}$ must be kept high during burst transfer cycles.

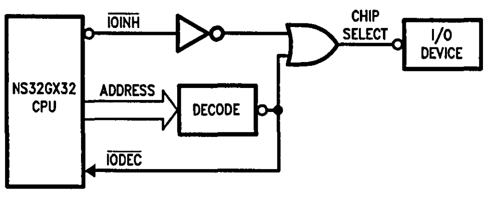
When $\overline{\text{IODEC}}$ is active during a bus cycle for which $\overline{\text{IOINH}}$ is asserted, the CPU discards the data and applies the special handling required for I/O devices. *Figure 3-31* shows a possible implementation of an I/O device interface where the address mapping of the I/O devices is fixed.

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

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

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

3.0 Functional Description (Continued)



TL/EE/10253-38

FIGURE 3-31. 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. $\overline{\text{INT}}$ and $\overline{\text{NMI}}$ are for maskable and non-maskable interrupts respectively. $\overline{\text{DBG}}$ is used for requesting an external debug trap.

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

$\overline{\text{NMI}}$ 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.

$\overline{\text{INT}}$ is level-sensitive and, as such, once asserted, it must be kept asserted until it is acknowledged.

The $\overline{\text{DBG}}$ signal, like $\overline{\text{NMI}}$, is edge-sensitive; it differs from $\overline{\text{NMI}}$ in that the CPU samples it on each rising edge of BCLK. $\overline{\text{DBG}}$ 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 $\overline{\text{DBG}}$ 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 $\overline{\text{NMI}}$ 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 Internal Status

The NS32GX32 provides information on the system interface concerning its internal activity.

The U/$\overline{\text{S}}$ signal will indicate the state of the U bit in the PSR except in the following cases:

While executing a MOVUS instruction it will be '1' during the source read.

While executing a MOVSU instruction it will be '1' during the destination write.

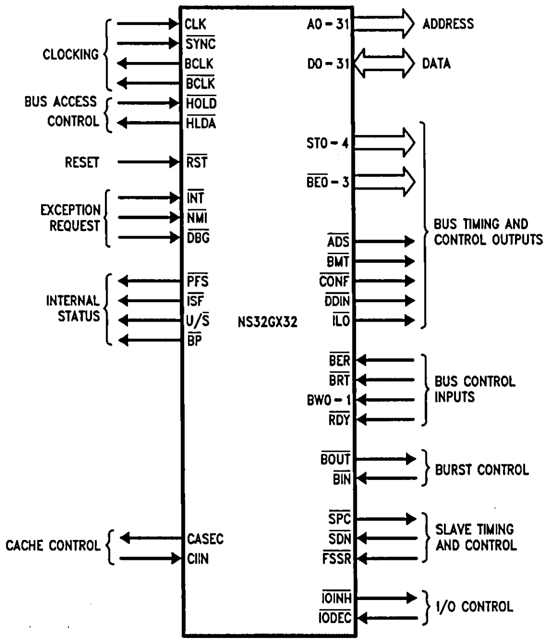
The $\overline{\text{PFS}}$ signal is asserted for one BCLK cycle when the CPU begins executing a new instruction. The $\overline{\text{ISF}}$ signal is driven High along with $\overline{\text{PFS}}$ if the new instruction does not follow the previous instruction in sequence. More specifically, $\overline{\text{ISF}}$ is High along with $\overline{\text{PFS}}$ 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 $\overline{\text{BP}}$ signal is asserted for one BCLK cycle when an address-compare or PC-match condition is detected. If the $\overline{\text{BP}}$ signal is asserted one BCLK cycle after $\overline{\text{PFS}}$, it indicates that an address-compare debug condition has been detected. If $\overline{\text{BP}}$ is asserted at any other time, it indicates that a PC-Match debug condition has been detected.

While executing a CINV instruction, the CPU displays the operation code and source operand using slave processor write bus cycles.

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/10253-39

FIGURE 4-1. NS32GX32 Interface Signals

4.1 NS32GX32 PIN DESCRIPTIONS

Descriptions of the NS32GX32 pins are given in the following sections.

Included are also references to portions of the functional description, Section 3.

Figure 4-1 shows the NS32GX32 interface signals grouped according to related functions.

Note: An asterisk next to the signal name indicates a TRI-STATE condition for that signal when $\overline{\text{HOLD}}$ 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.

$\overline{\text{SYNC}}$ — **Synchronize.**

When $\overline{\text{SYNC}}$ is active, BCLK will stop toggling. This signal can be used to synchronize two or more CPUs (Section 3.5.2).

$\overline{\text{HOLD}}$ — **Hold Request.**

When active, causes the CPU to release the bus for DMA or multiprocessing purposes (Section 3.5.7).

Note:

If the $\overline{\text{HOLD}}$ 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 $\overline{\text{HLDA}}$ latency. This is to avoid speed degradations in cases of heavy $\overline{\text{HOLD}}$ activity (i.e. DMA controller cycles interleaved with CPU cycles).

$\overline{\text{RST}}$ — **Reset.**

When $\overline{\text{RST}}$ is active, the CPU is initialized to a known state (Section 3.5.3).

$\overline{\text{INT}}$ — **Interrupt.**

A low level on this signal requests a maskable interrupt (Section 3.5.9).

$\overline{\text{NMI}}$ — **Nonmaskable Interrupt.**

A High-to-Low transition of this signal requests a nonmaskable interrupt (Section 3.5.9).

2

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).
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 3.5.10).
$\overline{ISF}$	**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 (Section 3.5.10).
$\overline{BP}$	**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.
$\overline{IOINH}$	**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).
$\overline{DDIN}$	***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).

4.0 Device Specifications (Continued)

$\overline{BMT}$	***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 • • • 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

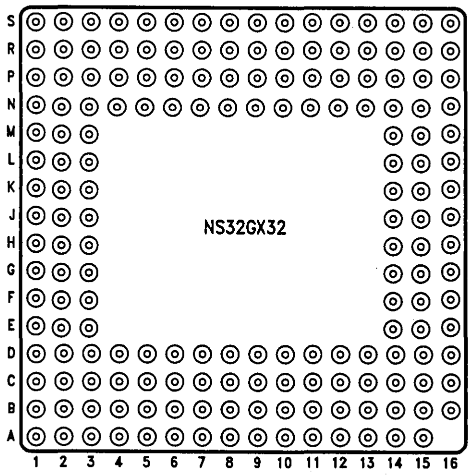
Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS NS32GX32-20, 25: T_{CASE} = 0° to +95°C, V_{CC} = 5V ±10%, GND = 0V
NS32GX32-30: 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			0.45	V
	$\overline{CONF}$, $\overline{BMT}$	I_{OL} = 6 mA			0.45	V
	BCLK, $\overline{BCLK}$	I_{OL} = 16 mA			0.45	V
	All Other Outputs	I_{OL} = 2 mA			0.45	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		700 @ 30 MHz 600 @ 25 MHz 470 @ 20 MHz	800 @ 30 MHz 700 @ 25 MHz 575 @ 20 MHz	mA

4.0 Device Specifications (Continued)

Connection Diagram



TL/EE/10253-40

Bottom View

FIGURE 4-2. 175-Pin PGA Package

NS32GX32 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
Reserved (2)	A11	$\overline{SYNC}$	C10	$\overline{CASEC}$	F2
Reserved (1)	A12	Reserved (2)	C11	Reserved	F3
Reserved (2)	A13	Reserved (2)	C12	D21	F14
Reserved (2)	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
Reserved (2)	B10	BCLK	D8	VCCB13	H14
Reserved (2)	B11	GNDCLK	D9	D15	H15
Reserved (2)	B12	CLK	D10	D14	H16
Reserved (2)	B13	Reserved (2)	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	Reserved	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.

Note 2: This pin should be connected to logical high.
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.

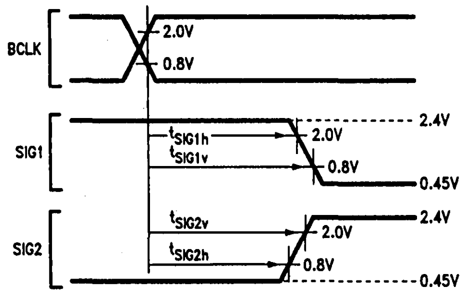


FIGURE 4-3. Output Signals Specification Standard

ABBREVIATIONS:

L.E.—leading edge	R.E.—rising edge
T.E.—training edge	F.E.—falling edge

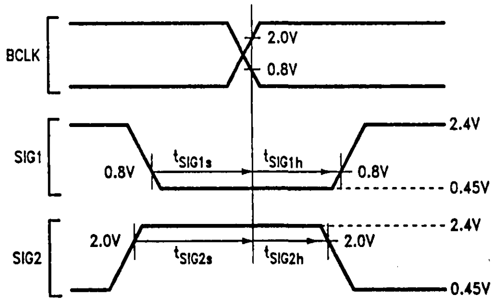


FIGURE 4-4. Input Signals Specification Standard

4.0 Device Specifications (Continued)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32GX32-20, NS32GX32-25, NS32GX32-30

- 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 $\overline{\text{BCLK}}$ is also assumed.
- The output to input timings (e.g., Address to $\overline{\text{RDY}}$, Address to $\overline{\text{BER}}$, etc.) are at least 2 ns better than the worst case values calculated from the output valid and input setup times relative to BCLK or $\overline{\text{BCLK}}$.

Name	Figure	Description	Reference/Conditions	NS32GX32-20		NS32GX32-25		NS32GX32-30		Units
				Min	Max	Min	Max	Min	Max	
t_{BC_p}	4-24	Bus Clock Period	R.E., BCLK to Next R.E., BCLK	50	100	40	100	33.3	100	ns
t_{BC_h}	4-24	BCLK High Time	At 2.0V on BCLK (Both Edges)	20		16		13		
t_{BC_l}	4-24	BCLK Low Time	At 0.8V on BCLK (Both Edges)	20		16		13		
t_{BC_r} (Note 1)	4-24	BCLK Rise Time	0.8V to 2.0V on R.E., BCLK		5		4		3	ns
t_{BC_f} (Note 1)	4-24	BCLK Fall Time	2.0V to 0.8V on F.E., BCLK		5		4		3	ns
t_{NBC_h}	4-24	$\overline{\text{BCLK}}$ High Time	At 2.0V on $\overline{\text{BCLK}}$ (Both Edges)	20		16		13		
t_{NBC_l}	4-24	$\overline{\text{BCLK}}$ Low Time	At 0.8V on $\overline{\text{BCLK}}$ (Both Edges)	20		16		13		
t_{NBC_r} (Note 1)	4-24	$\overline{\text{BCLK}}$ Rise Time	0.8V to 2.0V on R.E., $\overline{\text{BCLK}}$		5		4		3	ns
t_{NBC_f} (Note 1)	4-24	$\overline{\text{BCLK}}$ Fall Time	2.0V to 0.8V on F.E., $\overline{\text{BCLK}}$		5		4		3	ns
$t_{CBC_{dr}}$	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
$t_{CBC_{df}}$	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
$t_{CNBC_{dr}}$	4-24	CLK to $\overline{\text{BCLK}}$ R.E. Delay	2.0V on R.E., CLK to 0.8V on R.E., $\overline{\text{BCLK}}$		20		17		15	ns
$t_{CNBC_{df}}$	4-24	CLK to $\overline{\text{BCLK}}$ F.E. Delay	2.0V on R.E., CLK to 0.8V on F.E., $\overline{\text{BCLK}}$		20		17		15	ns
$t_{BCNBC_{rf}}$ (Note 1)	4-24	Bus Clocks Skew	2.0V on R.E., BCLK to 0.8V on F.E., $\overline{\text{BCLK}}$	−2	+2	−2	+2	−2	+2	ns
$t_{BCNBC_{fr}}$ (Note 1)	4-24	Bus Clocks Skew	0.8V on F.E., BCLK to 2.0V on R.E., $\overline{\text{BCLK}}$	−2	+2	−2	+2	−2	+2	ns
t_{A_v}	4-5, 4-6	Address Bits 0-31 Valid	After R.E., BCLK T1		11		9		8	ns
t_{A_h}	4-5, 4-6	Address Bits 0-31 Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{A_f}	4-11, 4-12	Address Bits 0-31 Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{A_{nf}}$	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, NS32GX32-20, NS32GX32-25, NS32GX32-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32GX32-20		NS32GX32-25		NS32GX32-30		Units
				Min	Max	Min	Max	Min	Max	
t_{AB_v}	4–8	Address Bits A2, A3 Valid (Burst Cycle)	After R.E., BCLK T2B		11		9		8	ns
t_{AB_h}	4–8	Address Bits A2, A3 Hold (Burst Cycle)	After R.E., BCLK T2B	0		0		0		ns
t_{DO_v}	4–6, 4–15	Data Out Valid	After R.E., BCLK T1		0.5 t_{BC_p} + 13		0.5 t_{BC_p} + 12		0.5 t_{BC_p} + 11	ns
t_{DO_h}	4–6, 4–15	Data Out Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
$t_{DO_{spc}}$	4–15	Data Out Setup (Slave Write)	Before $\overline{\text{SPC}}$ T.E.	8		6		5		ns
t_{DO_f}	4–7	Data Bus Floating	After R.E., BCLK T1 or Ti		21		17		13	ns
$t_{DO_{nf}}$	4–7	Data Bus Not Floating	After F.E., BCLK T1	0		0		0		ns
t_{BMT_v}	4–5, 4–7	$\overline{\text{BMT}}$ Signal Valid	After R.E., BCLK T1		32		27		23	ns
t_{BMT_h}	4–5, 4–7	$\overline{\text{BMT}}$ Signal Hold	After R.E., BCLK T2	0		0		0		ns
t_{BMT_f}	4–11, 4–12	$\overline{\text{BMT}}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{BMT_{nf}}$	4–11, 4–12	$\overline{\text{BMT}}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{CONF_a}	4–5, 4–8	$\overline{\text{CONF}}$ Signal Active	After R.E., BCLK T1	0.5 t_{BC_p}	0.5 t_{BC_p} + 11	0.5 t_{BC_p}	0.5 t_{BC_p} + 9	0.5 t_{BC_p}	0.5 t_{BC_p} + 8	ns
$t_{CONF_{ia}}$	4–5, 4–8	$\overline{\text{CONF}}$ Signal Inactive	After R.E., BCLK T1 or Ti		11		9		8	ns
t_{CONF_f}	4–11, 4–12	$\overline{\text{CONF}}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{CONF_{nf}}$	4–11, 4–12	$\overline{\text{CONF}}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{ADS_a}	4–5, 4–8	$\overline{\text{ADS}}$ Signal Active	After R.E., BCLK T1		11		9		8	ns
$t_{ADS_{ia}}$	4–5, 4–8	$\overline{\text{ADS}}$ Signal Inactive	After F.E., BCLK T1		11		9		8	ns
t_{ADS_w}	4–6	$\overline{\text{ADS}}$ Pulse Width	At 0.8V (Both Edges)	15		12		9		ns
t_{ADS_f}	4–11, 4–12	$\overline{\text{ADS}}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{ADS_{nf}}$	4–11, 4–12	$\overline{\text{ADS}}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{BE_v}	4–6, 4–8	$\overline{\text{BE}_n}$ Signals Valid	After R.E., BCLK T1		11		9		8	ns
t_{BE_h}	4–6, 4–8	$\overline{\text{BE}_n}$ Signals Hold	After R.E., BCLK T1, Ti or T2B	0		0		0		ns
t_{BE_f}	4–11, 4–12	$\overline{\text{BE}_n}$ Signals Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{BE_{nf}}$	4–11, 4–12	$\overline{\text{BE}_n}$ Signals Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{DDIN_v}	4–5, 4–6	$\overline{\text{DDIN}}$ Signal Valid	After R.E., BCLK T1		11		9		8	ns
t_{DDIN_h}	4–5, 4–6	$\overline{\text{DDIN}}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{DDIN_f}	4–11, 4–12	$\overline{\text{DDIN}}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{DDIN_{nf}}$	4–11, 4–12	$\overline{\text{DDIN}}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{SPC_a}	4–14, 4–15	$\overline{\text{SPC}}$ 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, NS32GX32-20, NS32GX32-25, NS32GX32-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32GX32-20		NS32GX32-25		NS32GX32-30		Units
				Min	Max	Min	Max	Min	Max	
t_{SPCia}	4-14, 4-15	$\overline{\text{SPC}}$ Signal Inactive	After R.E., BCLK Ti, T1 or T2		19		15		12	ns
t_{DDSPC} (Note 1)	4-14	$\overline{\text{DDIN}}$ Valid to $\overline{\text{SPC}}$ Active	Before $\overline{\text{SPC}}$ L.E.	0		0		0		ns
t_{HLDAa}	4-12, 4-13	$\overline{\text{HLDA}}$ Signal Active	After F.E., BCLK Ti		15		11		10	ns
t_{HLDAia}	4-12	$\overline{\text{HLDA}}$ Signal Inactive	After F.E., BCLK Ti		15		11		10	ns
t_{STv}	4-5, 4-14	Status (ST0-4) Valid	After R.E., BCLK T1		11		9		8	ns
t_{STh}	4-5, 4-14	Status (ST0-4) Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{BOUTa}	4-8, 4-9	$\overline{\text{BOUT}}$ Signal Active	After R.E., BCLK T2		15		12		11	ns
t_{BOUTia}	4-8, 4-9	$\overline{\text{BOUT}}$ Signal Inactive	After R.E., BCLK Last T2B, T1 or Ti		15		12		11	ns
t_{BOUTf}	4-11, 4-12	$\overline{\text{BOUT}}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
t_{BOUTnf}	4-11, 4-12	$\overline{\text{BOUT}}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{ILOa}	4-7	Interlock Signal Active	After F.E., BCLK Ti		11		9		8	ns
t_{ILOia}	4-7	Interlock Signal Inactive	After F.E., BCLK Ti		11		9		8	ns
t_{PFSa}	4-21	$\overline{\text{PFS}}$ Signal Active	After F.E., BCLK		15		11		10	ns
t_{PFSia}	4-21	$\overline{\text{PFS}}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{ISFa}	4-22	$\overline{\text{ISF}}$ Signal Active	After F.E., BCLK		15		11		10	ns
t_{ISFia}	4-22	$\overline{\text{ISF}}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{BPa}	4-23	$\overline{\text{BP}}$ Signal Active	After F.E., BCLK		15		11		10	ns
t_{BPia}	4-23	$\overline{\text{BP}}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{USv}	4-5	U/$\overline{\text{S}}$ Signal Valid	After R.E., BCLK T1		11		9		8	ns
t_{USh}	4-5	U/$\overline{\text{S}}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{CASv}	4-5	$\overline{\text{CASEC}}$ Signal Valid	After F.E., BCLK T1		15		11		10	ns
t_{CASh}	4-5	$\overline{\text{CASEC}}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{CASf}	4-11, 4-12	$\overline{\text{CASEC}}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
t_{CASnf}	4-11, 4-12	$\overline{\text{CASEC}}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{IOIv}	4-5	$\overline{\text{IOINH}}$ Signal Valid	After R.E., BCLK T1		15		11		10	ns
t_{IOIh}	4-5	$\overline{\text{IOINH}}$ Signal Hold	After R.E., BCLK T1 or 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)

4.4.2.2 Input Signal Requirements: NS32GX32-20, NS32GX32-25, NS32GX32-30

Name	Figure	Description	Reference/Conditions	NS32GX32-20		NS32GX32-25		NS32GX32-30		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} (Note 1)	4–24	CLK Rise Time	0.8V to 2.0V on R.E., CLK		5		4		3	ns
t_{Cf} (Note 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	13		11		9		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	$\overline{\text{RDY}}$ Setup Time	Before R.E., BCLK T2(W), T1 or Ti	22		18		15		ns
t_{RDYh}	4–5	$\overline{\text{RDY}}$ 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)	21		17		14		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	$\overline{\text{HOLD}}$ Setup Time	Before F.E., BCLK	21		17		14		ns
t_{HOLDh}	4–12	$\overline{\text{HOLD}}$ Hold Time	After F.E., BCLK	1		1		1		ns
t_{BINs}	4–8	$\overline{\text{BIN}}$ Setup Time	Before F.E., BCLK T2 or T2(W)	21		17		14		ns
t_{BINh}	4–8	$\overline{\text{BIN}}$ Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{BERs}	4–6, 4–8	$\overline{\text{BER}}$ Setup Time	Before R.E., BCLK T1 or Ti	21		17		14		ns
t_{BERh}	4–6, 4–8	$\overline{\text{BER}}$ Hold Time	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{BRTs}	4–6, 4–8	$\overline{\text{BRT}}$ Setup Time	Before R.E., BCLK T1 or Ti	21		17		14		ns
t_{BRTh}	4–6, 4–8	$\overline{\text{BRT}}$ Hold Time	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{IODs}	4–5	$\overline{\text{IODEC}}$ Setup Time	Before F.E., BCLK T2 or T2(W)	21		17		14		ns
t_{IODh}	4–5	$\overline{\text{IODEC}}$ Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{PWR} (Note 1)	4–26	Power Stable to R.E. of $\overline{\text{RST}}$	After VCC Reaches 4.5V	50		40		30		µs
t_{RSTs}	4–27	$\overline{\text{RST}}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{RSTw}	4–27	$\overline{\text{RST}}$ Pulse Width	At 0.8V (Both Edges)	64		64		64		t_{BCp}

Note 1: Due to tester conditions, this parameter is not 100% tested.

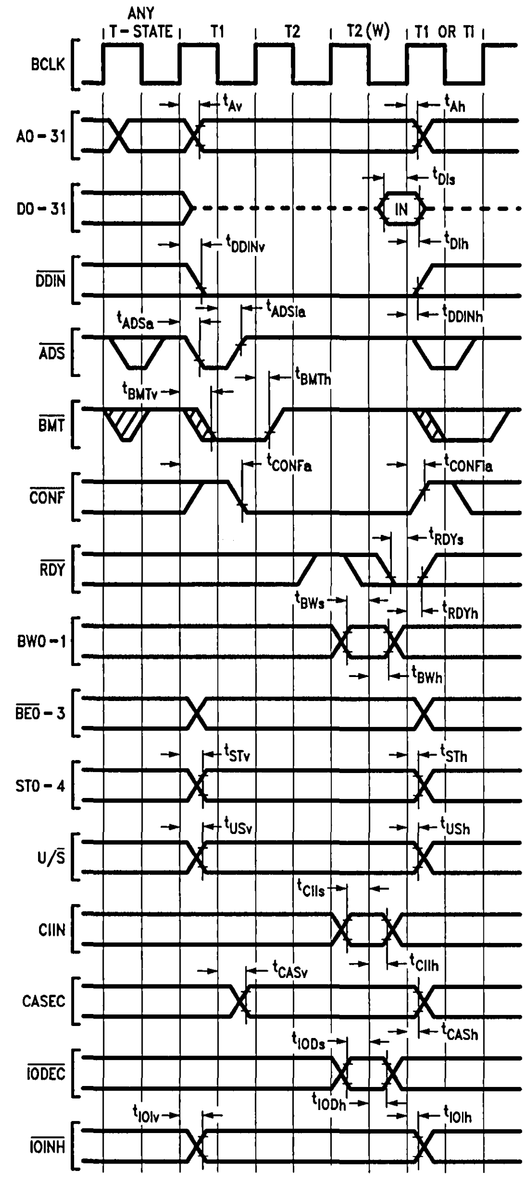
4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32GX32-20, NS32GX32-25, NS32GX32-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32GX32-20		NS32GX32-25		NS32GX32-30		Units
				Min	Max	Min	Max	Min	Max	
t_{CII_s}	4-5	$\overline{\text{CIIN}}$ Setup Time	Before F.E., BCLK T2	21		17		14		ns
t_{CII_h}	4-5	$\overline{\text{CIIN}}$ Hold Time	After F.E., BCLK T2	1		1		1		ns
t_{INT_s}	4-19	$\overline{\text{INT}}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{INT_h}	4-19	$\overline{\text{INT}}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{NMI_s}	4-19	$\overline{\text{NMI}}$ Setup Time	Before R.E., BCLK	20		17		16		ns
t_{NMI_h}	4-19	$\overline{\text{NMI}}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{SD_s}	4-16	$\overline{\text{SDN}}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{SD_h}	4-16	$\overline{\text{SDN}}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{FSSR_s}	4-17	$\overline{\text{FSSR}}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{FSSR_h}	4-17	$\overline{\text{FSSR}}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{SYNC_s}	4-25	$\overline{\text{SYNC}}$ Setup Time	Before R.E., CLK	10		8		7		ns
t_{SYNC_h}	4-25	$\overline{\text{SYNC}}$ Hold Time	After R.E., CLK	1		1		1		ns
t_{DBG_s}	4-20	$\overline{\text{DBG}}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{DBG_h}	4-20	$\overline{\text{DBG}}$ Hold Time	After R.E., BCLK	1		1		1		ns

4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams

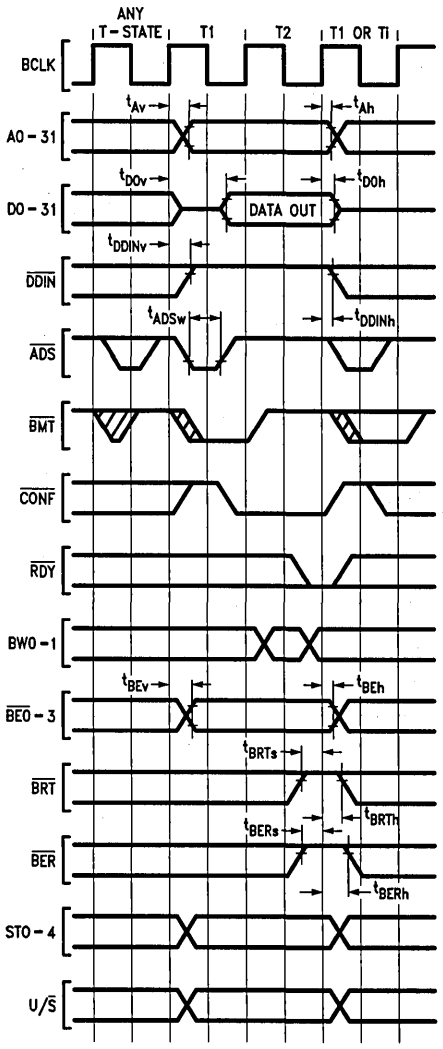


TL/EE/10253-43

FIGURE 4-5. Basic Read Cycle Timing

2

4.0 Device Specifications (Continued)

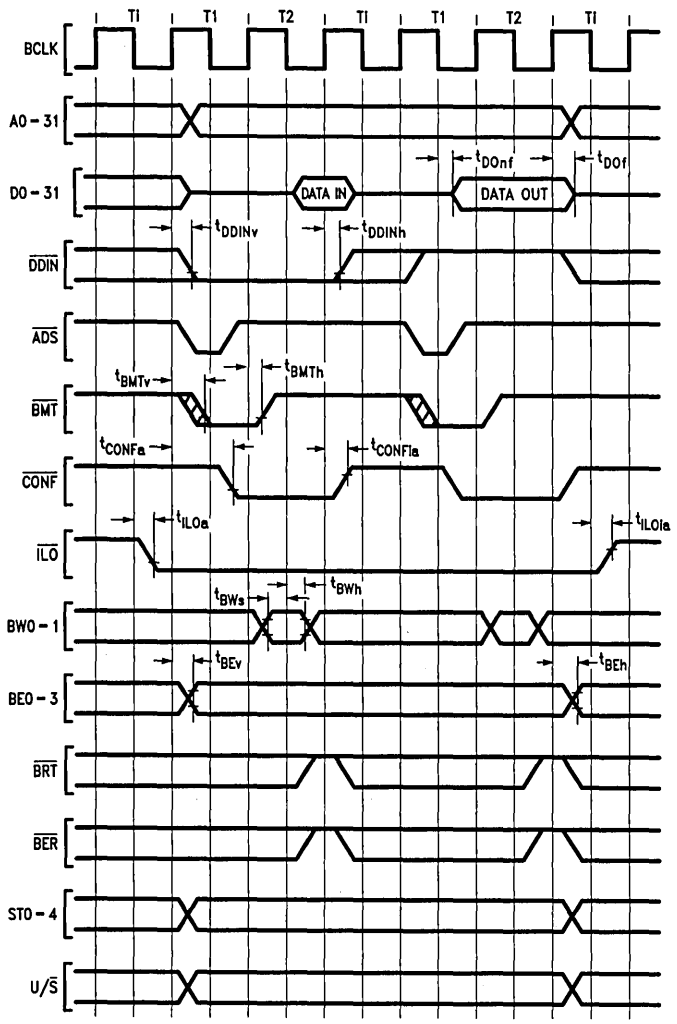


TL/EE/10253-44

Note: An Idle State is always inserted before a Write Cycle when the Write immediately follows a confirmed Read Cycle. A0–31, DDIN, BE0–3, ST0–4 remain unchanged during this idle state.

FIGURE 4-6. Write Cycle Timing

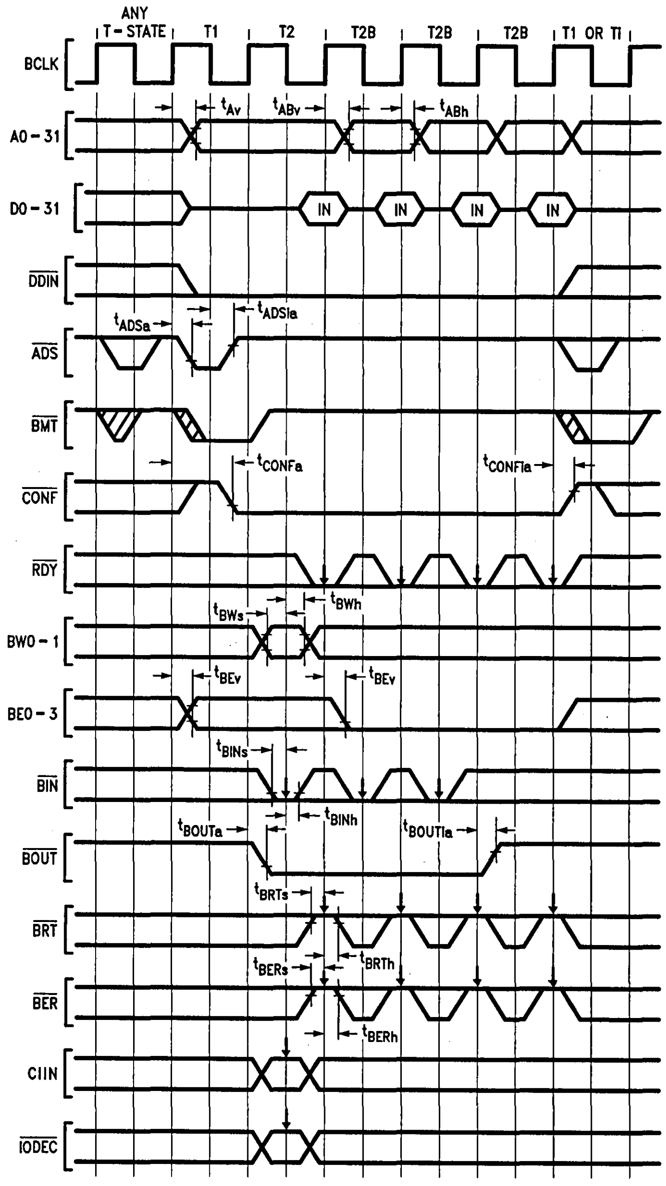
4.0 Device Specifications (Continued)



TL/EE/10253–45

FIGURE 4-7. Interlocked Read and Write Cycles

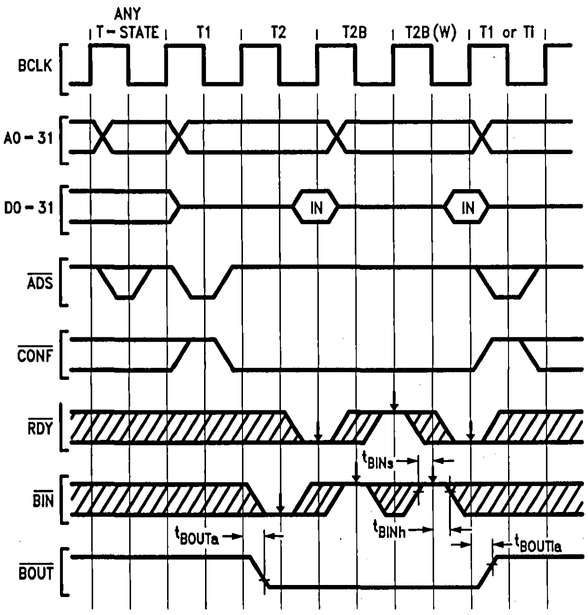
4.0 Device Specifications (Continued)



TL/EE/10253-46

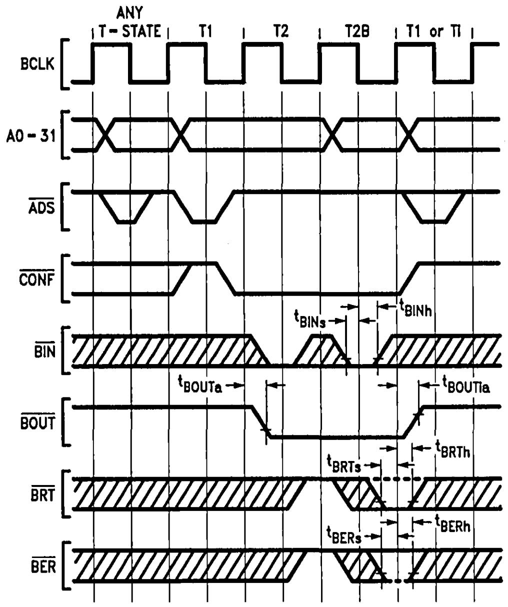
FIGURE 4-8. Burst Read Cycles

4.0 Device Specifications (Continued)



TL/EE/10253-47

FIGURE 4-9. External Termination of Burst Cycles

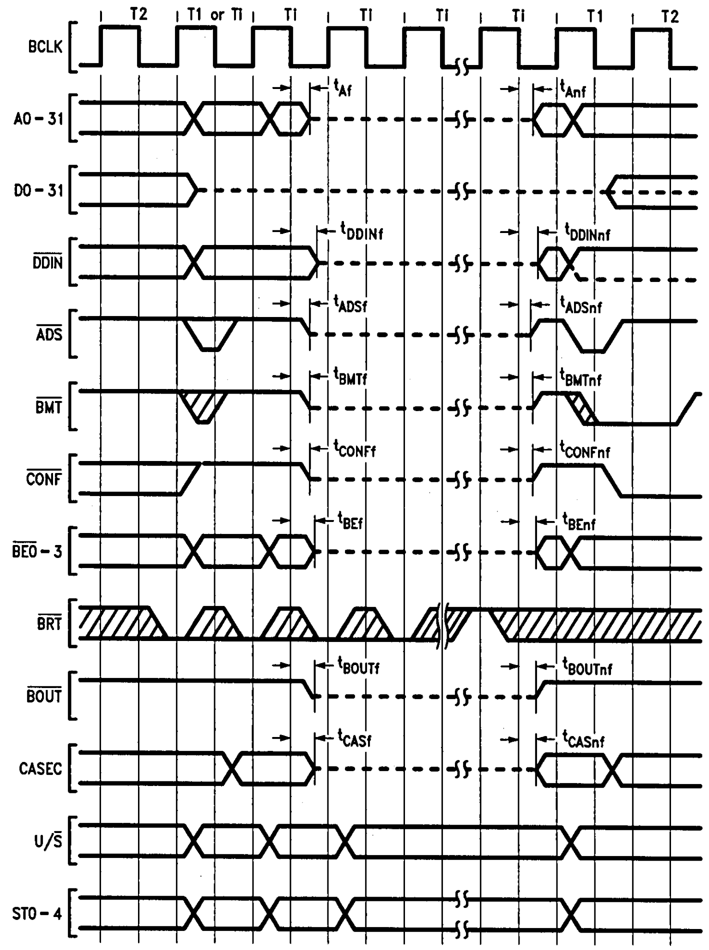


TL/EE/10253-48

FIGURE 4-10. Bus Error or Retry During Burst Cycles

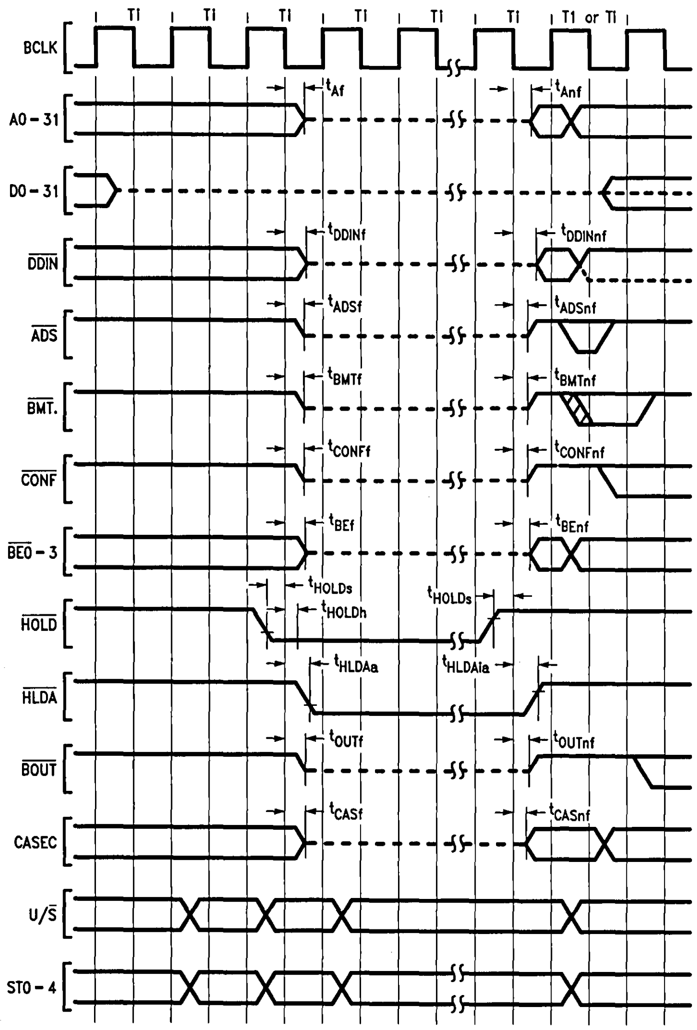
Note: Two idle state are always inserted by the CPU following the assertion of $\overline{\text{BRT}}$.

4.0 Device Specifications (Continued)



TL/EE/10253–49

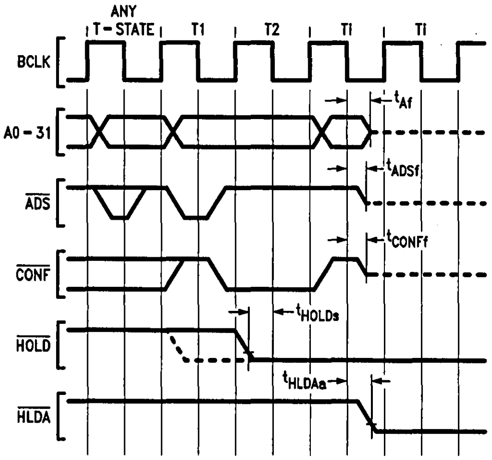
FIGURE 4-11. Extended Retry Timing



TL/EE/10253–50

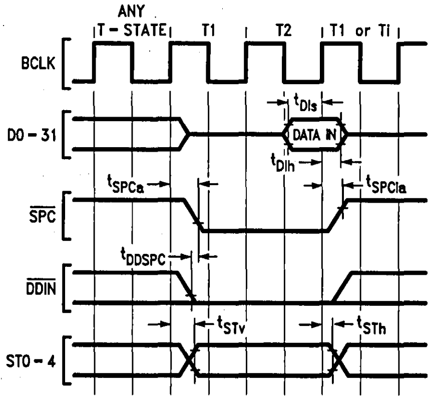
FIGURE 4-12. Hold Timing (Bus Initially Idle)

4.0 Device Specifications (Continued)



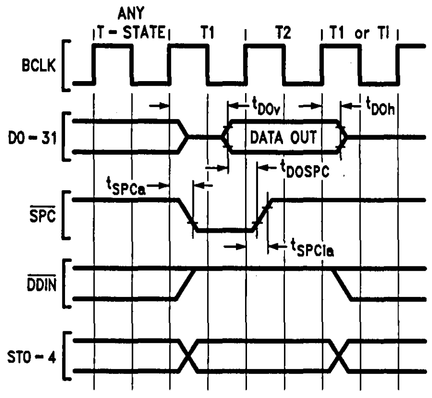
TL/EE/10253-51

FIGURE 4-13. HOLD Acknowledge Timing (Bus Initially Not Idle)



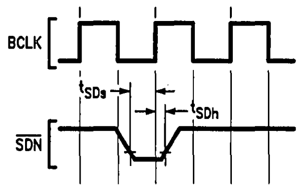
TL/EE/10253-52

FIGURE 4-14. Slave Processor Read Timing



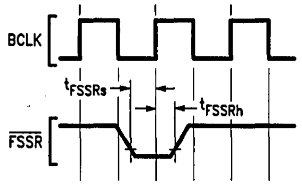
TL/EE/10253-53

FIGURE 4-15. Slave Processor Write Timing



TL/EE/10253-54

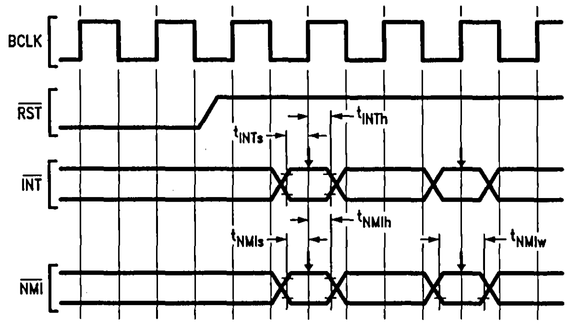
FIGURE 4-16. Slave Processor Done



TL/EE/10253-55

FIGURE 4-17. FSSR Signal Timing

4.0 Device Specifications (Continued)

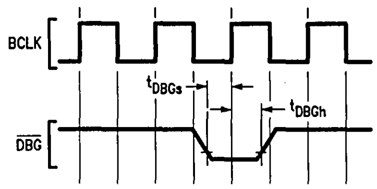


TL/EE/10253-57

FIGURE 4-18. $\overline{\text{INT}}$ and $\overline{\text{NMI}}$ Signals Sampling

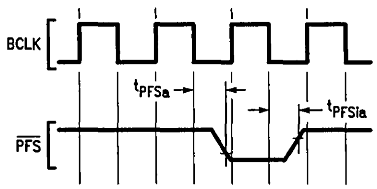
Note 1: $\overline{\text{INT}}$ and $\overline{\text{NMI}}$ are sampled on every other rising edge of BCLK, starting with the second rising edge of BCLK after $\overline{\text{RST}}$ goes high.

Note 2: $\overline{\text{INT}}$ is level sensitive, and once asserted, it should not be deasserted until it is acknowledged.



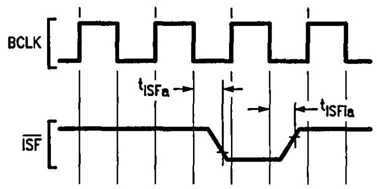
TL/EE/10253-58

FIGURE 4-19. Debug Trap Request



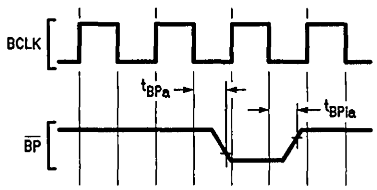
TL/EE/10253-59

FIGURE 4-20. $\overline{\text{PFS}}$ Signal Timing



TL/EE/10253-60

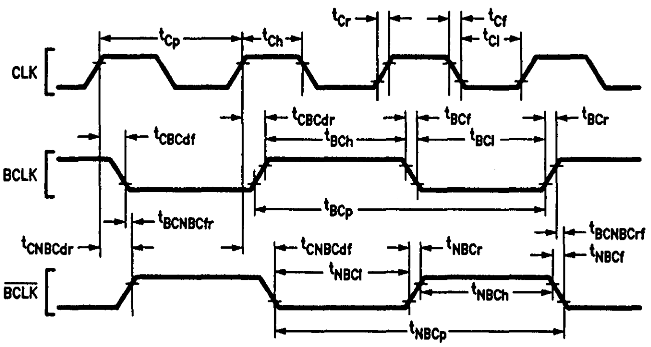
FIGURE 4-21. $\overline{\text{ISF}}$ Signal Timing



TL/EE/10253-61

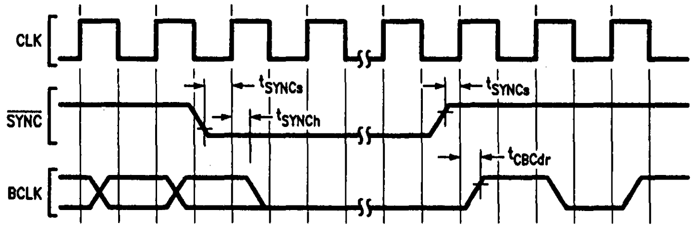
FIGURE 4-22. Break Point Signal Timing

4.0 Device Specifications (Continued)



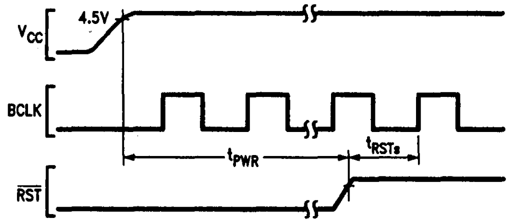
TL/EE/10253-62

FIGURE 4-23. Clock Waveforms



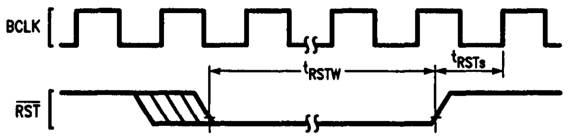
TL/EE/10253-63

FIGURE 4-24. Bus Clock Synchronization



TL/EE/10253-64

FIGURE 4-25. Power-On Reset



TL/EE/10253-65

FIGURE 4-26. Non-Power-On Reset

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 = US
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	Res	F	I

Note: Reserved bit must be set to 0 when executing SETCFG.

7	0
cond	1 0 1 0

Format 0

Bcond (BR)

7	0
op	0 0 1 0

Format 1

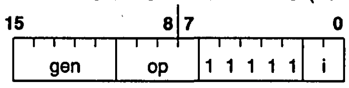
BSR	-0000	ENTER	-1000
RET	-0001	EXIT	-1001
CXP	-0010	NOP	-1010
RXP	-0011	WAIT	-1011
RETT	-0100	DIA	-1100
RETI	-0101	FLAG	-1101
SAVE	-0110	SVC	-1110
RESTORE	-0111	BPT	-1111

15	8 | 7			0
gen	short	op	1 1	i

Format 2

ADDQ	-000	ACB	-100
CMPQ	-001	MOVQ	-101
SPR	-010	LPR	-110
Scond	-011		

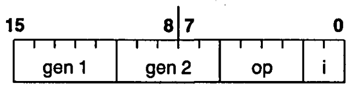
Appendix A: Instruction Formats (Continued)



Format 3

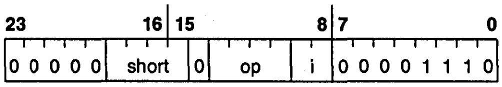
CXPD	-0000	ADJSP	-1010
BICPSR	-0010	JSR	-1100
JUMP	-0100	CASE	-1110
BISPSR	-0110		

Trap (UND) on XXX1, 1000



Format 4

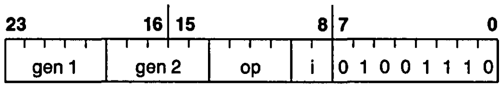
ADD	-0000	SUB	-1000
CMP	-0001	ADDR	-1001
BIC	-0010	AND	-1010
ADDC	-0100	SUBC	-1100
MOV	-0101	TBIT	-1101
OR	-0110	XOR	-1110



Format 5

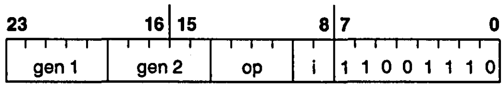
MOVS	-0000	SETCFG	-0010
CMPS	-0001	SKPS	-0011

Trap (UND) on 1XXX, 01XX



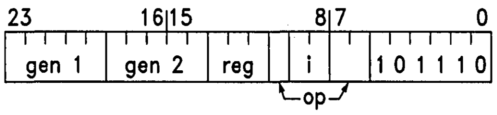
Format 6

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

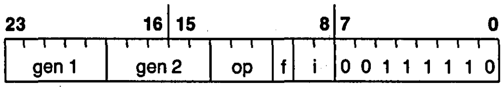
MOVM	-0000	MUL	-1000
CMPM	-0001	MEI	-1001
INSS	-0010	Trap (UND)	-1010
EXTS	-0011	DEI	-1011
MOVXBW	-0100	QUO	-1100
MOVZBW	-0101	REM	-1101
MOVZiD	-0110	MOD	-1110
MOVXiD	-0111	DIV	-1111



TL/EE/10253-66

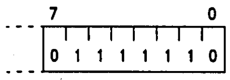
Format 8

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

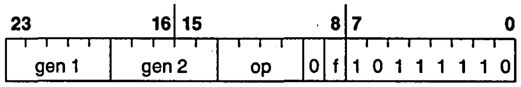
MOVif	-000	ROUND	-100
LFSR	-001	TRUNC	-101
MOVLF	-010	SFSR	-110
MOVFL	-011	FLOOR	-111



TL/EE/10253-67

Format 10

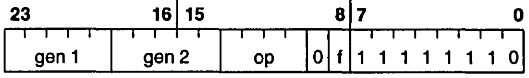
Trap (UND) Always



Format 11

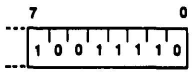
ADDf	-0000	DIVf	-1000
MOVf	-0001	Note 1	-1001
CMPf	-0010	Note 3	-1010
Note 3	-0011	Note 1	-1011
SUBf	-0100	MULf	-1100
NEGf	-0101	ABSf	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111

Appendix A: Instruction Formats (Continued)



Format 12

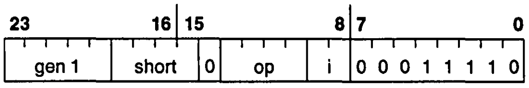
Note 2	-0000	Note 2	-1000
Note 1	-0001	Note 1	-1001
POLYf	-0010	Note 3	-1010
DOTf	-0011	Note 1	-1011
SCALBf	-0100	Note 2	-1100
LOGBf	-0101	Note 1	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111



TL/EE/10253-68

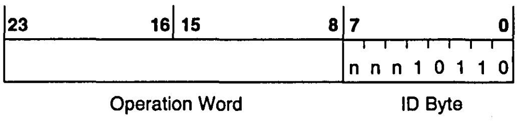
Format 13

Trap (UND) Always



Format 14

CINV −1001
Trap (UND) on 00XX, 01XX, 1000, 101X, 11XX

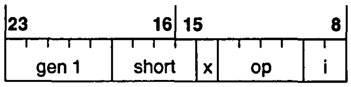


Format 15

(Custom Slave)

nnn **Operation Word Format**

000

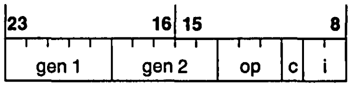


Format 15.0

LCR	-0010
SCR	-0011

Trap (UND) on all others

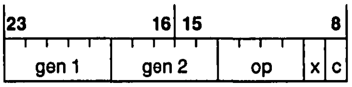
001



Format 15.1

CCV3	-000	CCV2	-100
LCSR	-001	CCV1	-101
CCV5	-010	SCSR	-110
CCV4	-011	CCV0	-111

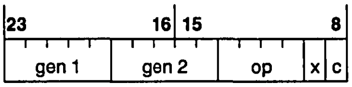
101



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

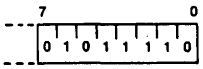
111



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

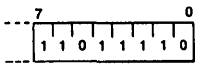
If nnn = 010, 011, 100, 110 then Trap (UND) Always.



TL/EE/10253-69

Format 16

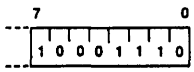
Trap (UND) Always



TL/EE/10253-70

Format 17

Trap (UND) Always

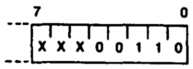


TL/EE/10253-71

Appendix A: Instruction Formats (Continued)

Format 18

Trap (UND) Always



TL/EE/10253-72

Format 19

Trap (UND) Always

Implied Immediate Encodings:

7							0
r7	r6	r5	r4	r3	r2	r1	r0

Register Mark, Appended to SAVE, ENTER

7							0
r0	r1	r2	r3	r4	r5	r6	r7

Register Mark, Appended to RESTORE, EXIT

7							0
offset			length - 1				

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 NS32GX32 is compatible with the Series 32000 architecture implemented by the NS32532, 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 NS32GX32. 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 NS32GX32.

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 NS32GX32 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 NS32GX32 may differ from those of the NS32032.
3. The program does not depend on the use of a Memory Management Unit (MMU).
4. The program does not depend on the detection of bus errors according to the implementation of the NS32332. For example, the NS32GX32 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 NS32GX32 and will result in a Trap (UND) when their execution is attempted.

B.2 ARCHITECTURE EXTENSIONS

The NS32GX32 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).

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 NS32GX32 due to effects of the Instruction Cache and branch prediction. In order to execute self-modifying code on the NS32GX32 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 NS32GX32 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 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.
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. 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 NS32GX32.

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.

C.2 INSTRUCTION DEFINITIONS

This section provides a description of the operations and encodings of the new NS32GX32 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.

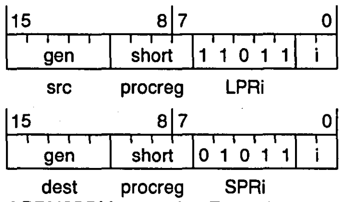


FIGURE C-1. LPRi/SPRi Instruction Formats

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)

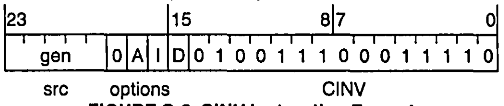


FIGURE C-2. CINV Instruction Format

Appendix D. Instruction Execution Times

The NS32GX32 achieves its performance by using an advanced implementation incorporating a 4-stage Instruction Pipeline, an Instruction Cache and a Data Cache into a single integrated circuit.

As a consequence of this advanced implementation, the performance evaluation for the NS32GX32 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 NS32GX32 by calculating various timing parameters for an instruction sequence. Due to the high degree of parallelism in the NS32GX32, the evaluation techniques presented here include some simplifications and approximations.

D.1 INTERNAL ORGANIZATION AND INSTRUCTION EXECUTION

The NS32GX32 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, memory-to-register instructions and register-to-memory. 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.
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 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.

Appendix D. Instruction Execution Times (Continued)

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. 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 NS32GX32 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. 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. The delay is 3 cycles when, as in the following example, the base register is modified by the immediately preceding instruction.

```
  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
```

Appendix D. Instruction Execution Times (Continued)

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.

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 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.

Appendix D. Instruction Execution Times (Continued)

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 (Section D.2.2). T_{anp} 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.
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 $\longrightarrow$ 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.

$T_e = \text{func}(T_{au}(n), T_{eu}(n-1), T_{flt}(n-1), \text{Break}(n-1)) + T_{eu}(n) + T_{flt}(n)$

Appendix D. Instruction Execution Times (Continued)

func provides the amount of processing time in the Address Unit that cannot be hidden. Its definition is given below.

0	if $T_{au}(n) \leq (T_{eu}(n-1) + T_{flt}(n-1))$ 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 $(T_{au}(n) + K) \leq 0$ AND Break $(n-1)$

K is the value associated with Break $(n-1)$.

T_{flt} only applies to floating-point instructions and is always 0 for other instructions. It is evaluated as follows:

$$T_{flt} = t_{tcs} + T_{tsc} + T_{fpu}$$

T_{fpu} is the execution time in the Floating-Point Unit.

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 2 If Branch Taken
        movd     r3,r1      ; 2 cycles
        addqd    $(-2),r1   ; 2 cycles
        bsr      _fib       ; 3 cycles
        movd     r0,r4      ; 2 cycles + 4 Cycles due to RET
        movd     r3,r1      ; 2 cycles
        addqd    $(-1),r1   ; 2 cycles
        bsr      _fib       ; 3 cycles
        addd     r4,r0      ; 2 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 NS32GX32 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 datasheet.

D.5.5.1 Basic 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}	$2 \rightarrow 3$ 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	$2 \rightarrow 3$	2	
BSR	$2 \rightarrow 3$	$3 + T_{ad}$	
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.

Mnemonic	T_{eu}	T_{au}	Notes
CINV	10	$2 + T_{ad}$	Wait for pending writes. Break 5
CMPi CMPMi	2 $6 + 8 * n$	$2 + T_{ad}$	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	$(30 \rightarrow 40) + 4 * i$	$2 + T_{ad}$	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
FFSi	$11 + 3 * i$	$2 + T_{ad}$	i = number of bytes

Appendix D. Instruction Execution Times (Continued)

D.5.5.1 Basic Instructions (Continued)

Mnemonic	T_{eu}	T_{au}	Notes
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> If <M> then 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
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

Mnemonic	T_{eu}	T_{au}	Notes
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
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 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
SPRi	5 3	2 + T_{ad} 2 + T_{ad}	CPU Reg = PSR, CAR CPU Reg = all others

Mnemonic	T_{eu}	T_{au}	Notes
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> If <M> then break 0
WAIT	3	2	Wait for pending writes. Wait for interrupt
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}	Notes
MOVf, NEGf,	2	2 + T_{anp}	2	1	<FF>
ABSf, LOGBf	4 + 3 * I	2 + T_{anp} + T_{ad}	2	1	<MF>
	6 + 3 * I	2 + T_{anp}	2	1	<IF>
	6 + 3 * I	2 + T_{anp}	2	1	<TF>
	11 + 4 * I	2 + T_{anp} + T_{ad}	2	3 + 2 * I	<FM> Break − (1 + I)
	13 + 7 * I	2 + T_{anp} + T_{ad}	2	3 + 2 * I	<MM>, <IM> Break − (1 + I)
ADDf, SUBf,	2	2 + T_{anp}	2	1	<FF>
MULf, DIVf,	4 + 3 * I	2 + T_{anp}	2	1	<MF>
SCALBf	6 + 3 * I	2 + T_{anp}	2	1	<IF>
	6 + 3 * I	2 + T_{anp}	2	1	<TF>
	17 + 7 * I	2 + T_{anp} + T_{ad}	2	3 + 2 * I	<FM> Break − (1 + I)
	19 + 10 * I	2 + T_{anp} + T_{ad}	2	3 + 2 * I	<MM>, <IM> Break − (1 + I)
ROUNDfi, TRUNCfi,	11	2 + T_{anp}	2	3 + 2 * I	<FR> Break − 1
FLOORfi	11 + 4 * I	2 + T_{anp} + T_{ad}	2	3 + 2 * I	<FM> Break − (1 + I)
	13	2 + T_{anp} + T_{ad}	2	3 + 2 * I	<MR>, <IR> Break − 1
	13 + 7 * I	2 + T_{anp} + T_{ad}	2	3 + 2 * I	<MM>, <IM> Break − (1 + I)
CMPf	18	2 + T_{anp}	2		<FF>
	20 + 3 * I	2 + T_{anp} + T_{ad}	2		<MF>
	23 + 3 * I	2 + T_{anp} + T_{ad}	2		<FM>
	25 + 6 * I	2 + T_{anp} + T_{ad}	2		<MM>, <IM>, <MI>, <II> Break 3
POLYf, DOTf	2	2 + T_{anp}	2	1	<FF>
	4 + 3 * I	2 + T_{anp} + T_{ad}	2	1	<MF>
	6 + 3 * I	2 + T_{anp}	2	1	<IF>, <TF>
	11 + 4 * I	2 + T_{anp} + T_{ad}	2	1	<FM> Break − (1 + I)
	13 + 7 * I	2 + T_{anp} + T_{ad}	2	1	<MM>, <MI>, <IM>, <II> Break − (1 + I)
MOVif	6	2 + T_{anp}	2	1	<RF>
	13	2 + T_{anp} + T_{ad}	2		<RM> Break − 1
	6 + 3 * I	2 + T_{anp} + T_{ad}	2	1	<MF>, <IF>, <TF>
	13 + 7 * I	2 + T_{anp} + T_{ad}	2		<MM>, <IM> Break − (1 + I)
LFSR	6	2 + T_{anp}	2	1	<R>
	6 + 3 * I	2 + T_{anp} + T_{ad}	2	1	<M>
	6 + 3 * I	2 + T_{anp}	2	1	<I>
	6 + 3 * I	2 + T_{anp}	2	1	<T>
SFSR	11	2 + T_{anp} + T_{ad}	2	3	Break − 1
MOVFL	4	2 + T_{anp}	2	1	<FF>
	6	2 + T_{anp} + T_{ad}	2	1	<MF>, <IF>, <TF>
	15	2 + T_{anp} + T_{ad}	2		<FM> Break 0
	17	2 + T_{anp} + T_{ad}	2		<MM>, <IM> Break 0
MOVLF	4	2 + T_{anp}	2	1	<FF>
	9	2 + T_{anp} + T_{ad}	2	1	<MF>, <IF>, <TF>
	15	2 + T_{anp} + T_{ad}	2		<FM> Break 0
	20	2 + T_{anp} + T_{ad}	2		<MM>, <IM> Break 0

National Semiconductor

PRELIMINARY

NS32CG16-10/NS32CG16-15 High-Performance Printer/Display Processor

General Description

The NS32CG16 is a 32-bit microprocessor in the Series 32000® family that provides special features for graphics applications. It is specifically designed to support page oriented printing technologies such as Laser, LCS, LED, Ion-Deposition and InkJet.

The NS32CG16 provides a 16 Mbyte linear address space and a 16-bit external data bus. It also has a 32-bit ALU, an eight-byte prefetch queue, and a slave processor interface.

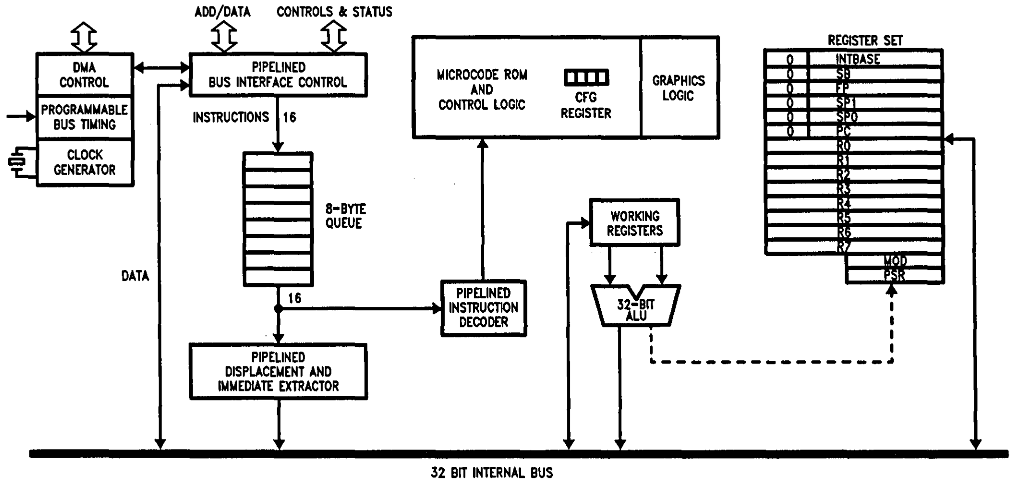
The capabilities of the NS32CG16 can be expanded by using an external floating point unit which interfaces to the NS32CG16 as a slave processor. This combination provides optimal support for outline character fonts.

The NS32CG16's highly efficient architecture, in addition to the built-in capabilities for supporting BITBLT (BIT-aligned BLock Transfer) operations and other special graphics functions, make the device the ideal choice to handle a variety of page description languages such as Postscript™ and PCL™.

Features

- Software compatible with the Series 32000 family
- 32-bit architecture and implementation
- 16 Mbyte linear address space
- Special support for imaging applications such as printers, faxes and scanners
  - — 18 graphics instructions
  - — Binary compression/expansion capability for font storage using RLL encoding
  - — Pattern magnification for Epson and HP LaserJet™ emulations
  - — 6 BITBLT instructions on chip
  - — Interface to an external BITBLT processing unit for very fast BITBLT operations (optional)
- Floating point support via the NS32081 or the NS32381 for outline fonts, scaling and rotation
- On-chip clock generator
- Optimal interface to large memory arrays via the DP84xx family of DRAM controllers
- Power save mode
- High-speed CMOS technology
- 68-pin plastic PCC package

Block Diagram



TL/EE/9424–1

1.0 Product Introduction

The NS32CG16 is a high speed CMOS microprocessor in the Series 32000 family. It is software compatible with all the other CPUs in the family. The device incorporates all of the Series 32000 advanced architectural features, with the exception of the virtual memory capability.

Brief descriptions of the NS32CG16 features that are shared with other members of the family are provided 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.

Large, Uniform Addressing. The NS32CG16 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

Table of Contents

List of Illustrations

List of Illustrations (Continued)

List of Tables

1.0 Product Information (Continued)

1.1 NS32CG16 SPECIAL FEATURES

In addition to the above Series 32000 features, the NS32CG16 provides features that make the device extremely attractive for a wide range of applications where graphics support, low chip count, and low power consumption are required.

The most relevant of these features are the graphics support capabilities, that can be used in applications such as printers, CRT terminals, and other varieties of display systems, where text and graphics are to be handled.

Graphics support is provided by eighteen instructions that allow operations such as BITBLT, data compression/expansion, fills, and line drawing, to be performed very efficiently. In addition, the device can be easily interfaced to an external BITBLT Processing Unit (BPU) for high BITBLT performance.

The NS32CG16 allows systems to be built with a relatively small amount of random logic. The bus is highly optimized to allow simple interfacing to a large variety of DRAMs and peripheral devices. All the relevant bus access signals and clock signals are generated on-chip. The cycle extension logic is also incorporated on-chip.

The device is fabricated in a low-power, double-poly, single metal, CMOS technology. It also includes a power-save feature that allows the clock to be slowed down under software control, thus minimizing the power consumption. This feature can be used in those applications where power saving during periods of low performance demand is highly desirable.

The bus characteristics and the power save feature are described in the "Functional Description" section. A general overview of BITBLT operations and a description of the graphics support instructions is provided in Section 2.4. Details on all the NS32CG16 instructions can be found in the NS32CG16 Printer/Display Processor Programmer's Reference Supplement and the related NS32CG16 supplement.

Below is a summary of the instructions that are directly applicable to graphics along with their intended use.

Instruction	Application
BBAND BBOR BBFOR BBXOR BBSTOD BITWT EXTBLT	The BitBlt group of instructions provide a method of quickly imaging characters, creating patterns, windowing and other block oriented effects.
MOVMP	Move Multiple Pattern is a very fast instruction for clearing memory and drawing patterns and lines.
TBITS	Test Bit String will measure the length of 1's or 0's in an image, supporting many data compression methods (RLL), TBITS may also be used to test for boundaries of images.
SBITS	Set Bit String is a very fast instruction for filling objects, outline characters and drawing horizontal lines. The TBITS and SBITS instructions support Group 3 and Group 4 CCITT communications (FAX).
SBITPS	Set Bit Perpendicular String is a very fast instruction for drawing vertical, horizontal and 45° lines. In printing applications SBITS and SBITPS may be used to express portrait and landscape respectively from the same compressed font data. The size of the character may be scaled as it is drawn.
SBIT CBIT TBIT IBIT	The Bit group of instructions enable single pixels anywhere in memory to be set, cleared, tested or inverted.
INDEX	The INDEX instruction combines a multiply-add sequence into a single instruction. This provides a fast translation of an X-Y address to a pixel relative address.

2.0 Architectural Description

2.1 REGISTER SET

The NS32CG16 CPU has 17 internal registers grouped according to functions as follows: 8 general purpose, 7 address, 1 processor status and 1 configuration. *Figure 2-1* shows the NS32CG16 internal registers.

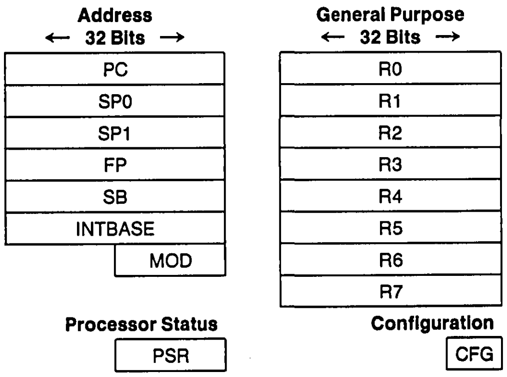


FIGURE 2-1. NS32CG16 Internal Registers

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

2.0 Architectural Description (Continued)

an operand that is 8 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. Except for the MOD register that is 16 bits wide, all the others are 32 bits. In the NS32CG16 only the lower 24 bits are implemented in the six 32-bit address registers. The top 8 bits are always zero. A description of the address registers 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.

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.

15							8	7							0
B				I	P	S	U	N	Z	F	J	K	L	T	C

FIGURE 2-2. Processor Status Register (PSR)

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 3.7.6).

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.

K Reserved for use by the CPU.

J Reserved for use by the CPU.

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).

P The P bit prevents a TRC trap from occurring more than once for an instruction (Section 3.7.6). 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.

2.0 Architectural Description (Continued)

B Reserved for use by the CPU. This bit is set to 1 during the execution of the EXTBLT instruction and causes the $\overline{BPU}$ signal to become active. Upon reset, B is set to zero and the $\overline{BPU}$ signal is set high.

Note 1: When an interrupt is acknowledged, the B, I, P, S and U bits are set to zero and the $\overline{BPU}$ signal is set high. A return from interrupt will restore the original values from the copy of the PSR register saved in the interrupt stack.

Note 2: If BITBLT (BB) instructions are executed in an interrupt routine, the PSR bits J and K must be cleared first.

2.1.4 Configuration Register

The Configuration Register (CFG) is 8 bits wide, of which four bits are implemented. The implemented bits are used to declare the presence of certain external devices and to select the clock scaling factor. CFG is programmed by the SETCFG instruction. The format of CFG is shown in *Figure 2-3*. The various control bits are described below.

7							0
				C	M	F	I

FIGURE 2-3. Configuration Register (CFG)

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 Clock scaling. This bit is used in conjuction with the C bit to select the clock scaling factor.

C Clock scaling. Same as the M bit above. Refer to Section 3.2.1 on "Power Save Mode" for details.

2.2 MEMORY ORGANIZATION

The main memory of the NS32CG16 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.

7   0
A

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.

15 8	7 0
A+1	A
MSB	LSB

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.

31 24	23 16	15 8	7 0
A+3	A+2	A+1	A
MSB			LSB

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.

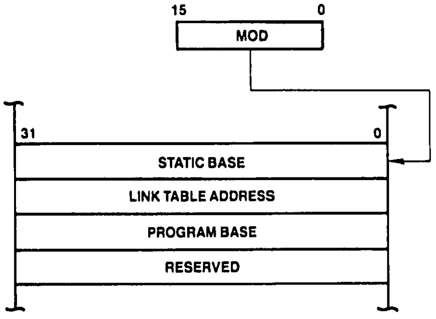
2.2.1 Dedicated Tables

Two of the NS32CG16 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 NS32CG16. 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/9424-2

FIGURE 2-4. Module Descriptor Format

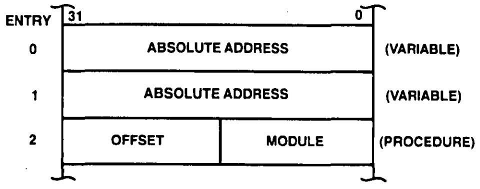
2.0 Architectural Description (Continued)

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/9424-3

FIGURE 2-5. A Sample Link Table

2.3 INSTRUCTION SET

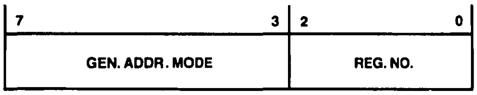
2.3.1 General Instruction Format

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 addressing modes. Each Disp/Imm field may contain 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 2.2).

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 2.3.3).



TL/EE/9424-5

FIGURE 2-7. Index Byte Format

2.3.2 Addressing Modes

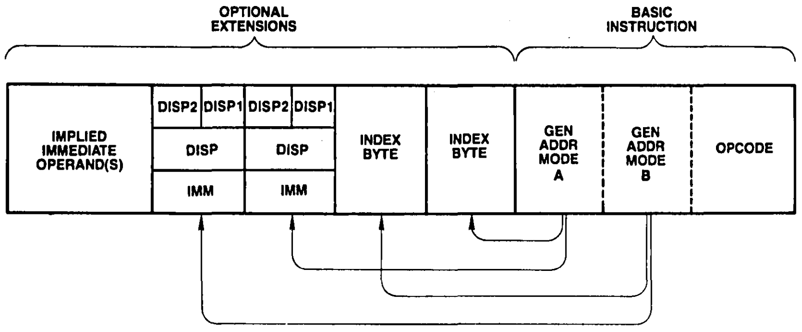
The NS32CG16 CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."

Addressing modes in the NS32CG16 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.

NS32CG16 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.



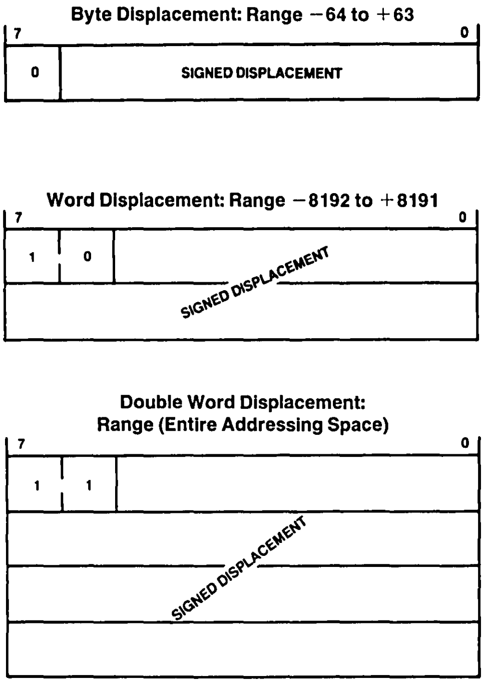
TL/EE/9424-4

FIGURE 2-6. General Instruction Format

2.0 Architectural Description (Continued)

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.


Byte Displacement: Range −64 to +63
7
0
0
SIGNED DISPLACEMENT
Word Displacement: Range −8192 to +8191
7
0
1
0
SIGNED DISPLACEMENT
Double Word Displacement: Range (Entire Addressing Space)
7
0
1
1
SIGNED DISPLACEMENT

TL/EE/9424-6

FIGURE 2-8. Displacement Encodings

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 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.

In addition to the general modes, Register-Indirect with auto-increment/decrement and warps or pitch are available on several of the graphics instructions.

2.0 Architectural Description (Continued)

TABLE 2-1. NS32CG16 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)

2.3.3 Instruction Set Summary

Table 2-2 presents a brief description of the NS32CG16 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 and the NS32CG16 Printer/Display Processor Programmer's Reference.

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: SP, SB, FP, INTBASE, MOD, PSR, US (bottom 8 PSR bits).

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

TABLE 2-2. NS32CG16 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. NS32CG16 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.

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. NS32CG16 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 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.

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. NS32CG16 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.
12	POLYf	gen,gen	Polynomial Step.
12	DOTf	gen,gen	Dot Product.
12	SCALBf	gen,gen	Binary Scale.
12	LOGBf	gen,gen	Binary Log.

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.

GRAPHICS

Format	Operation	Operands	Description
5	BBOR	options*	Bit-aligned block transfer 'OR'.
5	BBAND	options	Bit-aligned block transfer 'AND'.
5	BBFOR		Bit-aligned block transfer fast 'OR'.
5	BBXOR	options	Bit-aligned block transfer 'XOR'.
5	BBSTOD	options	Bit-aligned block source to destination.
5	BITWT		Bit-aligned word transfer.
5	EXTBLT	options	External bit-aligned block transfer.
5	MOVMPi		Move multiple pattern.
5	TBITS	options	Test bit string.
5	SBITS		Set bit string.
5	SBITPS		Set bit perpendicular string.

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.

***Note:** Options are controlled by fields of the instruction, PSR status bits, or dedicated register values.

2.0 Architectural Description (Continued)

2.4 GRAPHICS SUPPORT

The following sections provide a brief description of the NS32CG16 graphics support capabilities. Basic discussions on frame buffer addressing and BITBLT operations are also provided. More detailed information on the NS32CG16 graphics support instructions can be found in the NS32CG16 Printer/Display Processor Programmer's Reference.

2.4.1 Frame Buffer Addressing

There are two basic addressing schemes for referencing pixels within the frame buffer: Linear and Cartesian (or x-y). Linear addressing associates a single number to each pixel representing the physical address of the corresponding bit in memory. Cartesian addressing associates two numbers to each pixel representing the x and y coordinates of the pixel relative to a point in the Cartesian space taken as the origin. The Cartesian space is generally defined as having the origin in the upper left. A movement to the right increases the x coordinate; a movement downward increases the y coordinate.

The correspondence between the location of a pixel in the Cartesian space and the physical (BIT) address in memory is shown in *Figure 2-9.* The origin of the Cartesian space (x=0, y=0) corresponds to the bit address 'ORG'. Incrementing the x coordinate increments the bit address by one. Incrementing the y coordinate increments the bit address by an amount representing the warp (or pitch) of the Cartesian space. Thus, the linear address of a pixel at location (x, y) in the Cartesian space can be found by the following expression.

$$ADDR = ORG + y * WARP + x$$

Warp is the distance (in bits) in the physical memory space between two vertically adjacent bits in the Cartesian space.

Example 1 below shows two NS32CG16 instruction sequences to set a single pixel given the x and y coordinates. Example 2 shows how to create a fat pixel by setting four adjacent bits in the Cartesian space.

Example 1: Set pixel at location (x, y)

Setup: R0 x coordinate
R1 y coordinate

Instruction Sequence 1:

```
MULD   WARP, R1             ; Y*WARP
ADDD   R0, R1               ; + X = BIT OFFSET
SBITD  R1, ORG              ; SET PIXEL
```

Instruction Sequence 2:

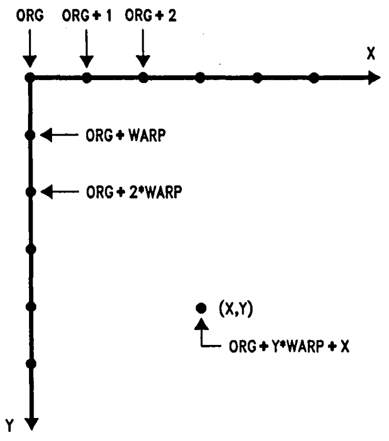
```
INDEXD R1, (WARP-1), R0     ; Y*WARP + X
SBITD  R1, ORG              ; SET PIXEL
```

Example 2: Create fat pixel by setting bits at locations (x, y), (x+1, y), (x, y+1) and (x+1, y+1).

Setup: R0 x coordinate
R1 y coordinate

Instruction Sequence:

```
INDEXD  R1, (WARP-1), R0    ; BIT ADDRESS
SBITD   41, ORG             ; SET FIRST PIXEL
ADDQD   1, R1               ; (X+1, Y)
SBITD   R1, ORG             ; SECOND PIXEL
ADDD    (WARP-1), R1        ; (X, Y+1)
SBITD   R1, ORG             ; THIRD PIXEL
ADDQD   1, R1               ; (X+1, Y+1)
SBITD   R1, ORG             ; LAST PIXEL
```



TL/EE/9424-61

FIGURE 2-9. Correspondence between Linear and Cartesian Addressing

2.4.2 BITBLT Fundamentals

BITBLT, BIT-aligned BLock Transfer, is a general operator that provides a mechanism to move an arbitrary size rectangle of an image from one part of the frame buffer to another. During the data transfer process a bitwise logical operation can be performed between the source and the destination data. BITBLT is also called Raster-Op: operations on rasters. It defines two rectangular areas, source and destination, and performs a logical operation (e.g., AND, OR, XOR) between these two areas and stores the result back to the destination. It can be expressed in simple notation as:

Source op Destination → Destination
op: AND, OR, XOR, etc.

2.0 Architectural Description (Continued)

2.4.2.1 Frame Buffer Architecture

There are two basic types of frame buffer architectures: plane-oriented or pixel-oriented. BITBLT takes advantage of the plane-oriented frame buffer architecture's attribute of multiple, adjacent pixels-per-word, facilitating the movement of large blocks of data. The source and destination starting addresses are expressed as pixel addresses. The width and height of the block to be moved are expressed in terms of pixels and scan lines. The source block may start and end at any bit position of any word, and the same applies for the destination block.

2.4.2.2 Bit Alignment

Before a logical operation can be performed between the source and the destination data, the source data must first be bit aligned to the destination data. In *Figure 2-10*, the source data needs to be shifted three bits to the right in order to align the first pixel (i.e., the pixel at the top left corner) in the source data block to the first pixel in the destination data block.

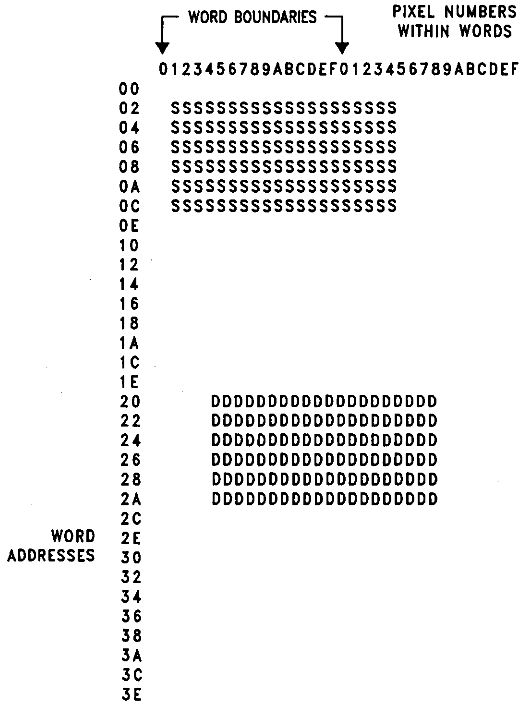
2.4.2.3 Block Boundaries and Destination Masks

Each BITBLT destination scan line may start and end at any bit position in any data word. The neighboring bits (bits sharing the same word address with any words in the destination data block, but not a part of the BITBLT rectangle) of the BITBLT destination scan line must remain unchanged after the BITBLT operation.

Due to the plane-oriented frame buffer architecture, all memory operations must be word-aligned. In order to preserve the neighboring bits surrounding the BITBLT destination block, both a left mask and a right mask are needed for all the leftmost and all the rightmost data words of the destination block. The left mask and the right mask both remain the same during a BITBLT operation.

The following example illustrates the bit alignment requirements. In this example, the memory data path is 16 bits wide. *Figure 2-10* shows a 32 pixel by 32 scan line frame buffer which is organized as a long bit stream which wraps around every two words (32 bits). The origin (top left corner) of the frame buffer starts from the lowest word in memory (word address 00 (hex)).

Each word in the memory contains 16 bits, D0–D15. The least significant bit of a memory word, D0, is defined as the first displayed pixel in a word. In this example, BITBLT addresses are expressed as pixel addresses relative to the origin of the frame buffer. The source block starting address is 021 (hex) (the second pixel in the third word). The destination block starting address is 204 (hex) (the fifth pixel in the 33rd word). The block width is 13 (hex), and the height is 06 (hex) (corresponding to 6 scan lines). The shift value is 3.

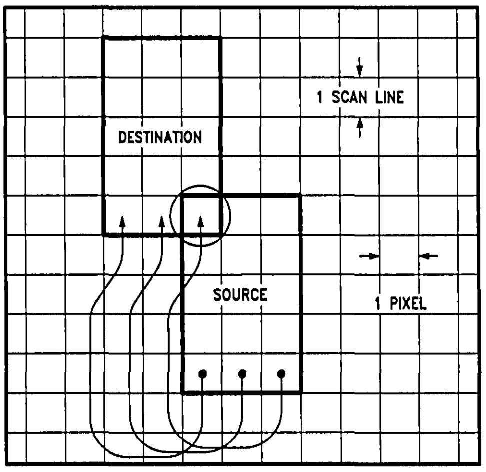


```
                          0123456789ABCDEF0123456789ABCDEF
                     00
                     02    SSSSSSSSSSSSSSSSSSSS
                     04    SSSSSSSSSSSSSSSSSSSS
                     06    SSSSSSSSSSSSSSSSSSSS
                     08    SSSSSSSSSSSSSSSSSSSS
                     0A    SSSSSSSSSSSSSSSSSSSS
                     0C    SSSSSSSSSSSSSSSSSSSS
                     0E
                     10
                     12
                     14
                     16
                     18
                     1A
                     1C
                     1E
                     20       DDDDDDDDDDDDDDDDDDDD
                     22       DDDDDDDDDDDDDDDDDDDD
                     24       DDDDDDDDDDDDDDDDDDDD
                     26       DDDDDDDDDDDDDDDDDDDD
                     28       DDDDDDDDDDDDDDDDDDDD
                     2A       DDDDDDDDDDDDDDDDDDDD
                     2C
              WORD   2E
         ADDRESSES   30
                     32
                     34
                     36
                     38
                     3A
                     3C
                     3E
```

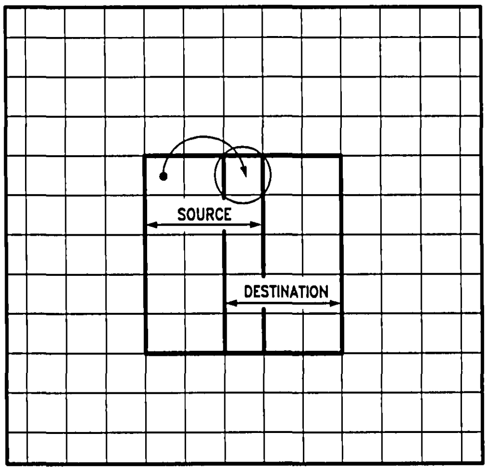
TL/EE/9424-62

FIGURE 2-10. 32-Pixel by 32-Scan Line Frame Buffer

2.0 Architectural Description (Continued)



(a)



(b)

FIGURE 2-11. Overlapping BITBLT Blocks

The left mask and the right mask are 0000,1111,1111,1111 and 1111,1111,0000,0000 respectively.

Note 1: Zeros in either the left mask or the right mask indicate the destination bits which will not be modified.

Note 2: The BB(function) and EXTBLT instructions use different set up parameters, and techniques.

2.4.2.2 BITBLT Directions

A BITBLT operation moves a rectangular block of data in a frame buffer. The operation itself can be considered as a subroutine with two nested loops. The loops are preceeded by setup operations. In the outer loop the source and destination starting addresses are calculated, and the test for completion is performed. In the inner loop the actual data movement for a single scan line takes place. The length of the inner loop is the number of (aligned) words spanned by each scan line. The length of the outer loop is equal to the height (number of scan lines) of the block to be moved. A skeleton of the subroutine representing the BITBLT operation follows.

```
BITBLT:      calculate BITBLT setup parameters;
             (once per BITBLT operation).
             such as
             width, height
             bit misalignment (shift number)
             left, right masks
             horizontal, vertical directions
             etc
             •
             •
OUTERLOOP:   calculate source, dest addresses;
             (once per scanline).
INNERLOOP:   move data, (logical operation) and incre-
             ment addresses;
             (once per word).
UNTIL        done horizontally
UNTIL        done vertically
RETURN       (from BITBLT).
```

Note: In the NS32CG16 only the setup operations must be done by the programmer. The inner and outer loops are automatically executed by the BITBLT instructions.

Each loop can be executed in one of two directions: the inner loop from left to right or right to left, the outer loop from top to bottom (down) or bottom to top (up).

The ability to move data starting from any corner of the BITBLT rectangle is necessary to avoid destroying the BITBLT source data as a result of destination writes when the source and destination are overlapped (i.e., when they share pixels). This situation is routinely encountered while panning or scrolling.

A determination of the correct execution directions of the BITBLT must be performed whenever the source and destination rectangles overlap. Any overlap will result in the destruction of source data (from a destination write) if the correct vertical direction is not used. Horizontal BITBLT direction is of concern only in certain cases of overlap, as will be explained below.

Figures 2-11(a) and *(b)* illustrate two cases of overlap. Here, the BITBLT rectangles are three pixels wide by five scan lines high; they overlap by a single pixel in *(a)* and a single column of pixels in *(b)*. For purposes of illustration, the BITBLT is assumed to be carried out pixel-by-pixel. This convention does not affect the conclusions.

In *Figure 2-11(a)*, if the BITBLT is performed in the UP direction (bottom-to-top) one of the transfers of the bottom scan line of the source will write to the circled pixel of the destination. Due to the overlap, this pixel is also part of the uppermost scan line of the source rectangle. Thus, data needed later is destroyed. Therefore, this BITBLT must be performed in the DOWN direction. Another example of this oc-

2.0 Architectural Description (Continued)

curs any time the screen is moved in a purely vertical direction, as in scrolling text. It should be noted that, in both of these cases, the choice of horizontal BITBLT direction may be made arbitrarily.

Figure 2-11(b) demonstrates a case in which the horizontal BITBLT direction may not be chosen arbitrarily. This is an instance of purely horizontal movement of data (panning). Because the movement from source to destination involves data within the same scan line, the incorrect direction of movement will overwrite data which will be needed later. In this example, the correct direction is from right to left.

2.4.2.5 BITBLT Variations

The 'classical' definition of BITBLT, as described in "Smalltalk-80 The Language and its Implementation", by Adele Goldberg and David Robson, provides for three operands: source, destination and mask/texture. This third operand is commonly used in monochrome systems to incorporate a stipple pattern into an area. These stipple patterns provide the appearance of multiple shades of gray in single-bit-per-pixel systems, in a manner similar to the 'halftone' process used in printing.

Texture op1 Source op2 Destination → Destination

While the NS32CG16 and the external BPU (if used) are essentially two-operand devices, three-operand BITBLT operations can be implemented quite flexibly and efficiently by performing the two operations serially.

2.4.3 GRAPHICS SUPPORT INSTRUCTIONS

The NS32CG16 provides eleven instructions for supporting graphics oriented applications. These instructions are divided into three groups according to the operations they perform. General descriptions for each of them and the related formats are provided in the following sections.

2.4.3.1 BITBLT (BIT-aligned BLock Transfer)

This group includes seven instructions. They are used to move characters and objects into the frame buffer which will be printed or displayed. One of the instructions works in conjunction with an external BITBLT Processing Unit (BPU) to maximize performance. The other six are executed by the NS32CG16.

BIT-aligned BLock Transfer

Syntax: BB(function) Options

Setup:	R0	base address, source data
	R1	base address, destination data
	R2	shift value
	R3	height (in lines)
	R4	first mask
	R5	second mask
	R6	source warp (adjusted)
	R7	destination warp (adjusted)
	0(SP)	width (in words)
Function:	AND, OR, XOR, FOR, STOD	
Options:	IA	Increasing Address (default option). When IA is selected, scan lines are transferred in the increasing BIT/BYTE order.
	DA	Decreasing Address.
	S	True Source (default option).
	−S	Inverted Source.

These five instructions perform standard BITBLT operations between source and destination blocks. The operations available include the following:

BBAND:	src	AND	dst
	−src	AND	dst
BBOR:	src	OR	dst
	−src	OR	dst
BBXOR:	src	XOR	dst
	−src	XOR	dst
BBFOR:	src	OR	dst
BBSTOD:	src	TO	dst
	−src	TO	dst

'src' and '−src' stand for 'True Source' and 'Inverted Source' respectively; 'dst' stands for 'Destination'.

Note 1: For speed reasons, the BB instructions require the masks to be specified with respect to the source block. In *Figure 2-10* masking was defined relative to the destination block.

Note 2: The options −S and DA are not available for the BBFOR instruction.

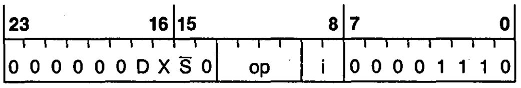
Note 3: BBFOR performs the same operation as BBOR with IA and S options.

Note 4: IA and DA are mutually exclusive and so are S and −S.

Note 5: The width is defined as the number of words of source data to read.

Note 6: An odd number of bytes can be specified for the source warp. However, word alignment of source scan lines will result in faster execution.

The horizontal and vertical directions of the BITBLT operations performed by the above instructions, with the exception of BBFOR, are both programmable. The horizontal direction is controlled by the IA and DA options. The vertical direction is controlled by the sign of the source and destination warps. *Figure 2-12* and Table 2-3 show the format of the BB instructions and the encodings for the 'op' and 'i' fields.



- D is set when the DA option is selected
- S̄ is set when the −S option is selected
- X is set for BBAND, and it is clear for all other BB instructions

FIGURE 2-12. BB Instructions Format

TABLE 2-3. 'op' and 'i' Field Encodings

Instruction	Options	'op' Field	'i' Field
BBAND	Yes	1010	11
BBOR	Yes	0110	01
BBXOR	Yes	1110	01
BBFOR	No	1100	01
BBSTOD	Yes	0100	01

BIT-aligned Word Transfer

Syntax: BITWT

Setup:	R0	Base address, source word
	R1	Base address, destination double word
	R2	Shift value

The BITWT instruction performs a fast logical OR operation between a source word and a destination double word, stores the result into the destination double word and increments registers R0 and R1 by two. Before performing the OR operation, the source word is shifted left (i.e., in the direction of increasing bit numbers) by the value in register R2.

2.0 Architectural Description (Continued)

This instruction can be used within the inner loop of a block OR operation. Its use assumes that the source data is 'clean' and does not need masking. The BITWT format is shown in *Figure 2-13.*

23							16	15							8	7							0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0

FIGURE 2-13. BITWT Instruction Format

External BITBLT

Syntax: EXTBLT

Setup:	R0	base addresses, source data
	R1	base address, destination data
	R2	width (in bytes)
	R3	height (in lines)
	R4	horizontal increment/decrement
	R5	temporary register (current width)
	R6	source warp (adjusted)
	R7	destination warp (adjusted)

Note 1: R0 and R1 are updated after execution to point to the last source and destination addresses plus related warps. R2, R3 and R5 will be modified. R4, R6, and R7 are returned unchanged.

Note 2: Source and destination pointers should point to word-aligned operands to maximize speed and minimize external interface logic.

This instruction performs an entire BITBLT operation in conjunction with an external BITBLT Processing Unit (BPU). The external BPU Control Register should be loaded by the software before the instruction is executed (refer to the DP8510 or DP8511 data sheets for more information on the BPU). The NS32CG16 generates a series of source read, destination read and destination write bus cycles until the entire data block has been transferred. The BITBLT operation can be performed in either horizontal direction. As controlled by the sign of the contents of register R4.

Depending on the relative alignment of the source and destination blocks, an extra source read may be required at the beginning of each scan line, to load the pipeline register in the external BPU. The L bit in the PSR register determines whether the extra source read is performed. If L is 1, no extra read is performed. The instructions CMPQB 2,1 or CMPQB 1,2 could be executed to provide the right setting for the L bit just before executing EXTBLT. *Figure 2-14* shows the EXTBLT format. The bus activity for a simple BITBLT operation is shown in *Figure 2-19.*

23								15							8	7							0
0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	1	1	1	0

FIGURE 2-14. EXTBLT Instruction Format

B.3.2 Pattern Fill

Only one instruction is in this group. It is usually used for clearing RAM and drawing patterns and lines.

Move Multiple Pattern

Syntax: MOVMPi

Setup:	R0	base address of the destination
	R1	pointer increment (in bytes)
	R2	number of pattern moves
	R3	source pattern

Note: R1 and R3 are not modified by the instruction. R2 will always be returned as zero. R0 is modified to reflect the last address into which a pattern was written.

This instruction stores the pattern in register R3 into the destination area whose address is in register R0. The pattern count is specified in register R2. After each store operation the destination address is changed by the contents of register R1. This allows the pattern to be stored in rows, in columns, and in any direction, depending on the value and sign of R1. The MOVMPi instruction format is shown in *Figure 2-15.*

23								15							8	7							0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	i	0	0	0	0	1	1	1	0

FIGURE 2-15. MOVMPi Instruction Format

B.3.3 Data Compression, Expansion and Magnify

The three instructions in this group can be used to compress data and restore data from compression. A compressed character set may require from 30% to 50% less memory space for its storage.

The compression ratio possible can be 50:1 or higher depending on the data and algorithm used. TBITS can also be used to find boundaries of an object. As a character is needed, the data is expanded and stored in a RAM buffer. The expand instructions (SBITS, SBITPS) can also function as line drawing instructions.

Test Bit String

Syntax: TBITS option

Setup:	R0	base address, source (byte address)
	R1	starting source bit offset
	R2	destination run length limited code
	R3	maximum value run length limit
	R4	maximum source bit offset
Option:	1	count set bits until a clear bit is found
	0	count clear bits until a set bit is found

Note: R0, R3 and R4 are not modified by the instruction execution. R1 reflects the new bit offset. R2 holds the result.

This instruction starts at the base address, adds a bit offset, and tests the bit for clear if "option" = 0 (and for set if "option" = 1). If clear (or set), the instruction increments to the next higher bit and tests for clear (or set). This testing for clear proceeds through memory until a set bit is found or until the maximum source bit offset or maximum run length value is reached. The total number of clear bits is stored in the destination as a run length value.

When TBITS finds a set bit and terminates, the bit offset is adjusted to reflect the current bit address. Offset is then ready for the next TBITS instruction with "option" = 0. After the instruction is executed, the F flag is set to the value of the bit previous to the bit currently being pointed to (i.e., the value of the bit on which the instruction completed execution). In the case of a starting bit offset exceeding the maximum bit offset (R1 ≥ R4), the F flag is set if the option was 1 and clear if the option was 0. The L flag is set when the desired bit is found, or if the run length equalled the maximum run length value and the bit was not found. It is cleared otherwise. *Figure 2-16* shows the TBITS instruction format.

23								15							8	7							0
0	0	0	0	0	0	0	0	S	0	1	0	0	1	1	1	0	0	0	0	1	1	1	0

• S is set for 'TBITS 1' and clear for 'TBITS 0'.

FIGURE 2-16. TBITS Instruction Format

2.0 Architectural Description (Continued)

Set Bit String

Syntax: SBITS

Setup:	R0	base address of the destination
	R1	starting bit offset (signed)
	R2	number of bits to set (unsigned)
	R3	address of string look-up table

Note: When the instruction terminates, the registers are returned unchanged.

SBITS sets a number of contiguous bits in memory to 1, and is typically used for data expansion operations. The instruction draws the number of ones specified by the value in R2, starting at the bit address provided by registers R0 and R1. In order to maximize speed and allow drawing of patterned lines, an external 1k byte lookup table is used. The lookup table is specified in the NS32CG16 Printer/Display Processor Programmer's Reference Supplement.

When SBITS begins executing, it compares the value in R2 with 25. If the value in R2 is less than or equal to 25, the F flag is cleared and the appropriate number of bits are set in memory. If R2 is greater than 25, the F flag is set and no other action is performed. This allows the software to use a faster algorithm to set longer strings of bits. *Figure 2-17* shows the SBITS instruction format.

```
23             15            8 7             0
0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 1 0
```

FIGURE 2-17. SBITS Instruction Format

Set BIT Perpendicular String

Syntax: SBITPS

Setup:	R0	base address, destination (byte address)
	R1	starting bit offset
	R2	number of bits to set
	R3	destination warp (signed value, in bits)

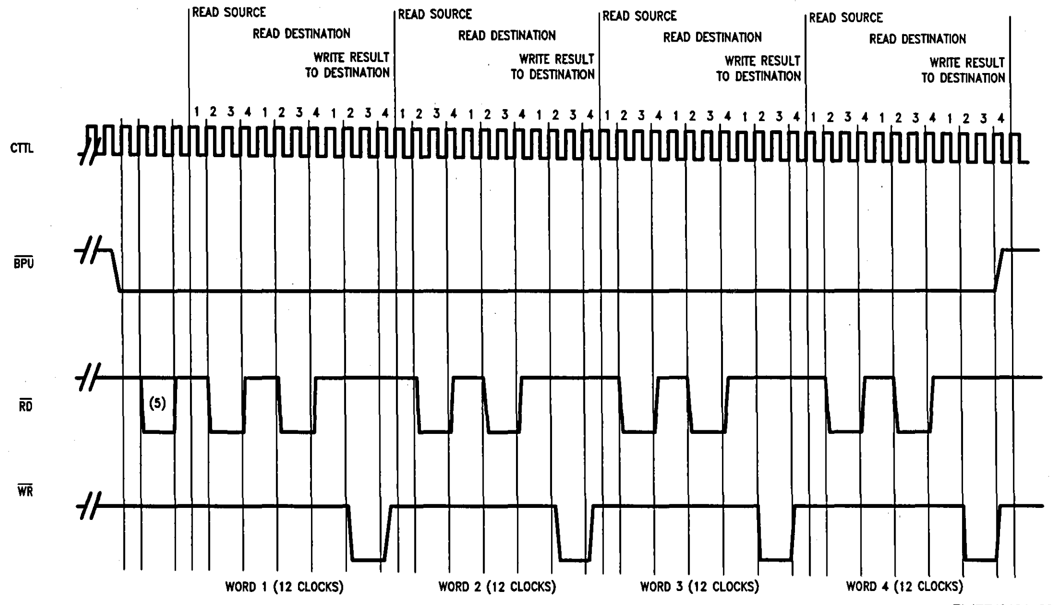
Note: When the instruction terminates, the R0 and R3 registers are returned unchanged. R1 becomes the final bit offset. R2 is zero.

The SBITPS can be used to set a string of bits in any direction. This allows a font to be expanded with a 90 or 270 degree rotation, as may be required in a printer application. SBITPS sets a string of bits starting at the bit address specified in registers R0 and R1. The number of bits in the string is specified in R2. After the first bit is set, the destination warp is added to the bit address and the next bit is set. The process is repeated until all the bits have been set. A negative raster warp offset value leads to a 90 degree rotation. A positive raster warp value leads to a 270 degree rotation. If the R3 value is = (space warp +1 or −1), then the result is a 45 degree line. If the R3 value is +1 or −1, a horizontal line results.

SBITS and SBITPS allow expansion on any 90 degree angle, giving portrait, landscape and mirror images from one font. *Figure 2-18* shows the SBITPS instruction format.

```
23             15            8 7             0
0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 0
```

FIGURE 2-18. SBITPS Instruction Format



TL/EE/9424-66

FIGURE 2-19. Bus Activity for a Simple BITBLT Operation

Note 1: This example is for a block 4 words wide and 1 line high.

Note 2: The sequence is common with all logical operations of the DP8510/DP8511 BPU.

Note 3: Mask values, shift values and number of bit planes do not affect the performance.

Note 4: Zero wait states are assumed throughout the BITBLT operation.

Note 5: The extra read is performed when the BPU pipeline register needs to be preloaded.

2.0 Architectural Description (Continued)

B.3.3.1 Magnifying Compressed Data

Restoring data is just one application of the SBITS and SBITPS instructions. Multiplying the "length" operand used by the SBITS and SBITPS instructions causes the resulting pattern to be wider, or a multiple of "length".

As the pattern of data is expanded, it can be magnified by 2x, 3x, 4x, . . . , 10x and so on. This creates several sizes of the same style of character, or changes the size of a logo. A magnify in both dimensions X and Y can be accomplished by drawing a single line, then using the MOVS (Move String) or the BB instructions to duplicate the line, maintaining an equal aspect ratio.

More information on this subject is provided in the NS32CG16 Printer/Display Processor Programmer's Reference Supplement.

3.0 Functional Description

3.1 POWER AND GROUNDING

The NS32CG16 requires a single 5-Volt power supply, applied on 5 pins. The logic voltage pin (V_{CCL}) supplies the power to the on-chip logic. The buffer voltage pins VCCCTTL, VCCFCLK, VCCAD, and VCCIO supply the power to the on-chip output drivers.

Grounding connections are made on 6 pins. The Logic Ground Pin (VSSL) provides the ground connection to the on-chip logic. The buffer ground pins VSSFCLK, VSSNTSO, VSSHAD, VSSLAD, VSSIO are the ground pins for the on-chip output drivers.

For optimal noise immunity, the power and ground pins should be connected to V_{CC} and ground planes respectively. If V_{CC} and ground planes are not used, single conductors should be run directly from each V_{CC} pin to a power point, and from each GND pin to a ground point. Daisy-chained connections should be avoided.

Decoupling capacitors should also be used to keep the noise level to a minimum. Standard 0.1 μF ceramic capacitors can be used for this purpose. In addition, a 1.0 μF tantalum capacitor should be connected between V_{CCL} and ground. They should attach to V_{CC}, V_{SS} pairs as close as possible to the NS32CG16.

During prototype using wire-wrap or similar methods, the capacitors should be soldered directly to the power pins of the NS32CG16 socket, or as close as possible, with very short leads.

Recommended bypass for production in printed circuit boards:

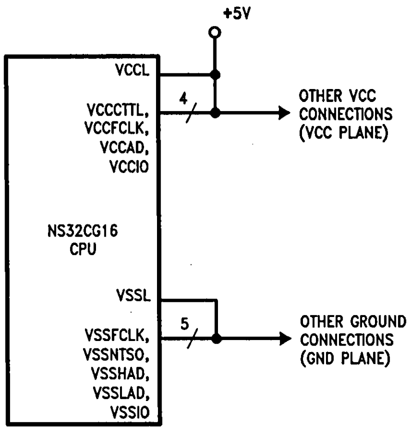
+5	Ground	Capacitors
VCCL	VSSL	0.1 μF Disk Ceramic 1.0 μF Tantulum
VCCIO	VSSIO	0.1 μF
VCCCTTL	VSSNTSO	0.1 μF
VCCAD	VSSLAD	0.1 μF
VCCAD	VSSHAD	None
VCCFCLK	VSSFCLK	0.1 μF

VCCL-VSSL bypass requires a very short lead length and low inductance on the 0.1 μF capacitor.

Design Notes

When constructing a board using high frequency clocks with multiple lines switching, special care should be taken to avoid resonances on signal lines. A separate power and ground layer is recommended. This is true when designing boards for the NS32CG16. Switching times of under 5 ns on some lines are possible. Resonant frequencies should be maintained well above the 200 MHz frequency range on signal paths by keeping traces short and inductance low. Loading capacitance at the end of a transmission line contributes to the resonant frequency and should be minimized if possible. Capacitors should be located as close as possible across each power and ground pair near the NS32CG16.

Power and ground connections are shown in *Figure 3-1.*



TL/EE/9424-7

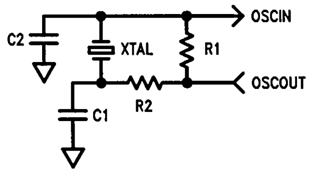
FIGURE 3-1. Power and Ground Connections

3.2 CLOCKING

The NS32CG16 provides an internal oscillator that interacts with an external clock source through two signals; OSCIN and OSCOUT.

Either an external single-phase clock signal or a crystal can be used as the clock source. If a single-phase clock source is used, only the connection on OSCIN is required; OSCOUT should be left unconnected or loaded with no more than 5 pF of stray capacitance. The voltage level requirements specified in Section 4.3 must also be met for proper operation.

When operation with a crystal is desired, a fundamental mode crystal should be used. In this case, special care should be taken to minimize stray capacitances and inductances, especially when operating at a crystal frequency of 30 MHz. The crystal, as well as the external RC components, should be placed in close proximity to the OSCIN and OSCOUT pins to keep the printed circuit trace lengths to an absolute minimum. *Figure 3-2* shows the external crystal interconnections. Table 3-1 provides the crystal characteristics and the values of the RC components, including stray capacitance, required for various frequencies.



TL/EE/9424-8

FIGURE 3-2. Crystal Interconnections

3.0 Functional Description (Continued)

TABLE 3-1. External Oscillator Specifications

Crystal Characteristics

Type	At-Cut
Tolerance	0.005% at 25°C
Stability	0.01% from 0°C to 70°C
Resonance	Fundamental (parallel)
Capacitance	20 pF
Maximum Series Resistance	50Ω

R and C Values

Frequency (MHz)	R1 (kΩ)	R2 (Ω)	C1 (pF)	C2 (pF)
12	470	120	20	20
16	360	100	20	20
20	270	75	20	20
25	220	68	20	20
30	180	51	20	20

3.2.1 Power Save Mode

The NS32CG16 provides a power save feature that can be used to significantly reduce the power consumption at times when the computational demand decreases. The device uses the clock signal at the OSCIN pin to derive the internal clock as well as the external signals PHI1, PHI2, CTTL and FCLK. The frequency of all these clock signals is affected by the clock scaling factor. Scaling factors of 1, 2, 4 or 8 can be selected by properly setting the C and M bits in the CFG register. The power save mode should not be used to reduce the clock frequency below the minimum frequency required by the CPU.

Upon reset, both C and M are set to zero, thus maximum clock rate is selected.

Due to the fact that the C and M bits are programmed by the SETCFG instruction, the power save feature can only be controlled by programs running in supervisor mode.

The following table shows the C and M bit settings for the various scaling factors, and the resulting supply current for a crystal frequency of 30 MHz.

Clock Scaling Factor vs Supply Current

C	M	Scaling Factor	CPU Clock Frequency	Typical I_{CC} at +5V
0	0	1	15 MHz	140 mA
0	1	2	7.5 MHz	76 mA
1	0	4	3.75 MHz	42 mA
1	1	8	1.88 MHz	25 mA

3.3 RESETTING

The $\overline{\text{RSTI}}$ input pin is used to reset the NS32CG16. The CPU samples $\overline{\text{RSTI}}$ on the falling edge of CTTL.

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 interrupts and traps are eliminated. The internal latch for the edge-sensitive $\overline{\text{NMI}}$ signal is cleared.

On application of power, $\overline{\text{RSTI}}$ 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 CTTL cycles. See *Figures 3-3* and *3-4*.

While in the Reset state, the CPU drives the signals $\overline{\text{ADS}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{DBE}}$, $\overline{\text{TSO}}$, $\overline{\text{BPU}}$, and $\overline{\text{DDIN}}$ inactive. AD0–AD15, A16–A23 and $\overline{\text{SPC}}$ are floated, and the state of all other output signals is undefined.

The internal CPU clock, PHI1, PHI2 and CTTL all run at half the frequency of the signal on the OSCIN pin. FCLK runs at the same frequency of OSCIN.

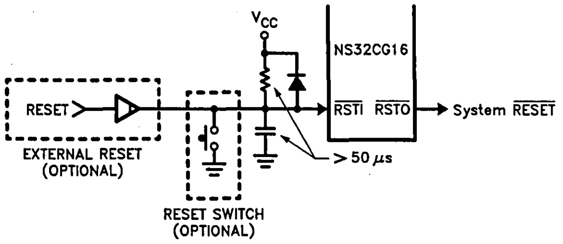
The $\overline{\text{HOLD}}$ signal must be kept inactive. After the $\overline{\text{RSTI}}$ signal is driven high, the CPU will stay in the reset condition for approximately 8 clock cycles and then it will begin execution at address 0.

The PSR is reset to 0. The CFG C and M bits are reset to 0. $\overline{\text{NMI}}$ is enabled to allow Non-Maskable Interrupts. The following conditions are present after reset due to the PSR being reset to 0:

Tracing is disabled.
Supervisor mode is enabled.
Supervisor stack space is used when the TOS addressing mode is indicated.
No trace traps are pending.
Only $\overline{\text{NMI}}$ is enabled. $\overline{\text{INT}}$ is not enabled.
$\overline{\text{BPU}}$ is inactive high.
The Clock Scaling Factor is set to 1, refer to Section 3.2.1.

Note that vector/non-vectored interrupts have not been selected. While interrupts are disabled, a SETCFG [I] instruction must be executed to declare the presence of the NS32202 if vectored interrupts are desired. If non-vectored interrupts are required, a SETCFG without the [I] must be executed.

The presence/absence of the NS32081 or NS32381 has also not been declared. If there is a Floating Point Unit, a SETCFG [F] instruction must be executed. If there is no floating point unit, a SETCFG without the [F] must be executed.



V_{CC}
NS32CG16
RESET
$\overline{\text{RSTI}}$ $\overline{\text{RSTO}}$
System $\overline{\text{RESET}}$
> 50 μs
EXTERNAL RESET (OPTIONAL)
RESET SWITCH (OPTIONAL)

TL/EE/9424-67

FIGURE 3-2a. Recommended Reset Connections

3.0 Functional Description (Continued)

In general, a SETCFG instruction must be executed in the reset routine, in order to properly configure the CPU. The options should be combined, and executed in a single instruction. For example, to declare vectored interrupts, a Floating Point unit installed, and full CPU clock rate, execute a SETCFG [F, I] instruction. To declare non-vectored interrupts, no FPU, and full CPU clock rate, execute a SETCFG [ ] instruction.

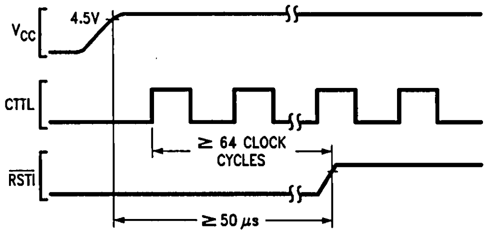


FIGURE 3-3. Power-On Reset Requirements

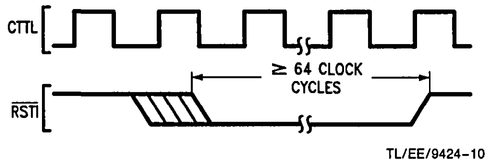


FIGURE 3-4. General Reset Timing

3.4 BUS CYCLES

The CPU will perform a bus cycle for one of the following reasons:

1) To write or read data, to or from memory or peripheral devices. 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 3.4.7).

3.4.1 Bus Status

The NS32CG16 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.

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 an interrupt request. See Section 3.4.6.
0101 —	Interrupt Acknowledge, Cascaded. The CPU is reading an interrupt vector to acknowledge a maskable interrupt request from a Cascaded Interrupt Control Unit.
0110 —	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.
0111 —	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.
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, 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. The write cycle of RMW will have a "write" status.
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 after the Slave Processor has signalled completion of an instruction.
1111 —	Broadcast Slave ID. The CPU is initiating the execution of a Slave Processor instruction by transferring the first byte of the instruction, which represents the slave processor indentification.

3.0 Functional Description (Continued)

3.4.2 Basic Read and Write Cycles

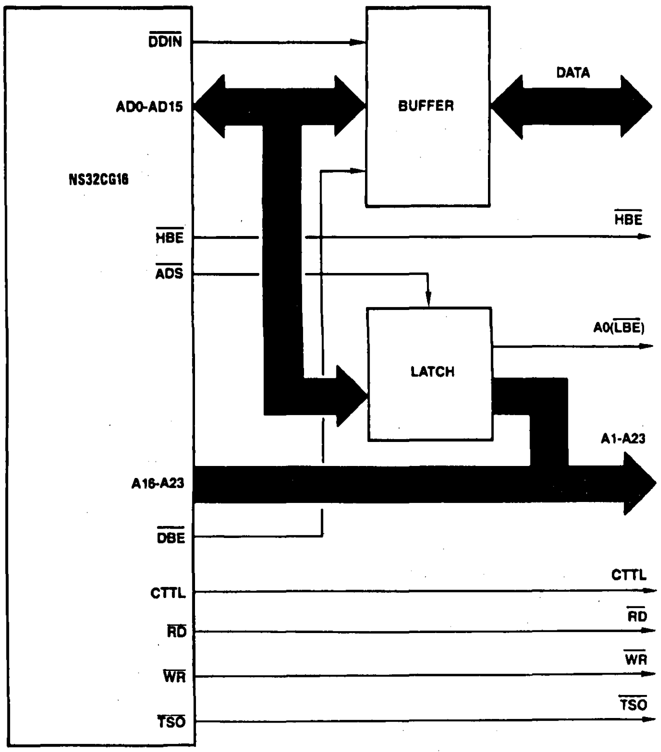
The sequence of events occurring during a CPU access to either memory or peripheral device is shown in *Figure 3-6* for a read cycle, and *Figure 3-7* for a write cycle.

The cases shown assume that the selected memory or peripheral device is capable of communicating with the CPU at full speed. If not, then cycle extension may be requested through $\overline{\text{CWAIT}}$ and/or $\overline{\text{WAIT}}$1-2.

A full-speed bus cycle is performed in four cycles of the CTTL clock signal, labeled T1 through T4. Clock cycles not associated with a bus cycle are designated Ti (for "Idle").

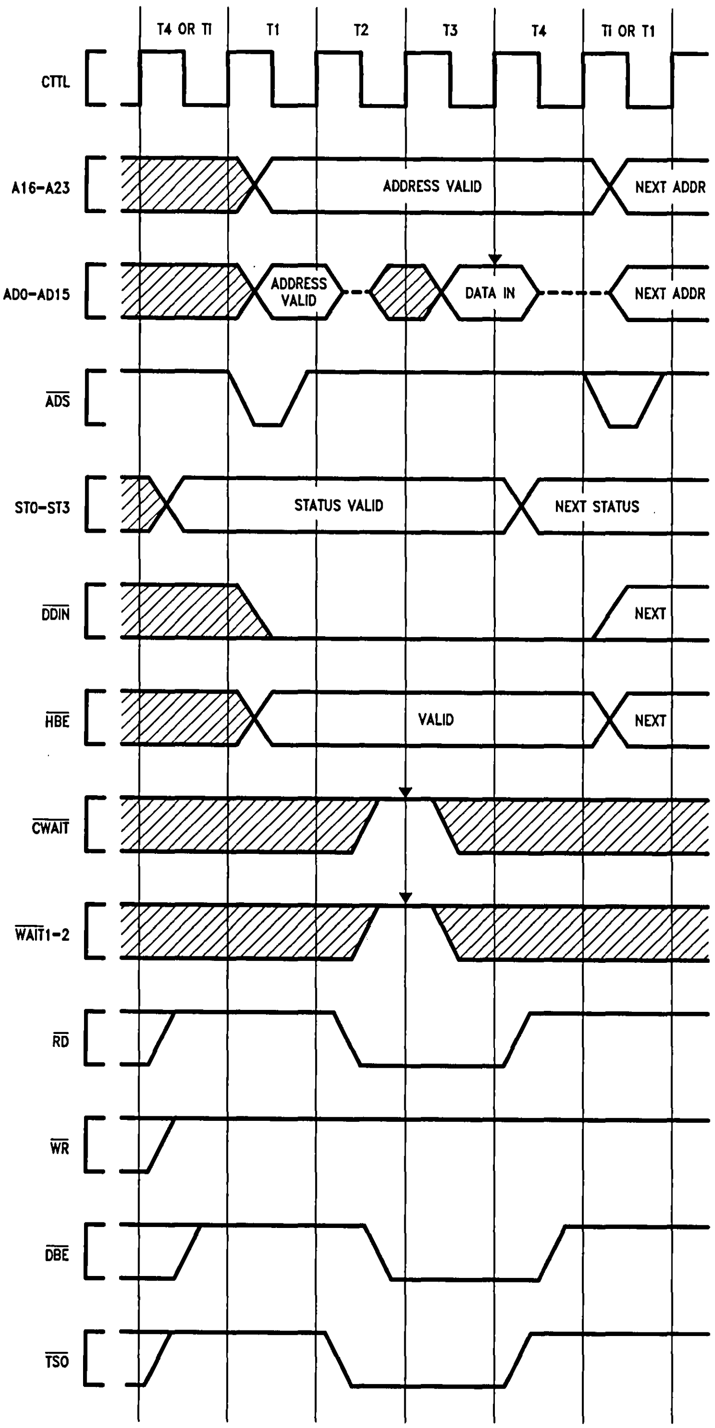
During T1, the CPU applies an address on pins AD0–AD15 and A16–A23. It also provides a low-going pulse on the $\overline{\text{ADS}}$ pin, which serves the dual purpose of informing external circuitry that a bus cycle is starting and of providing control to an external latch for demultiplexing Address bits 0–15 from the AD0–AD15 pins. See *Figure 3-5*. During this time also the status signals $\overline{\text{DDIN}}$, indicating the direction of the transfer, and $\overline{\text{HBE}}$, 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.



TL/EE/9424-11

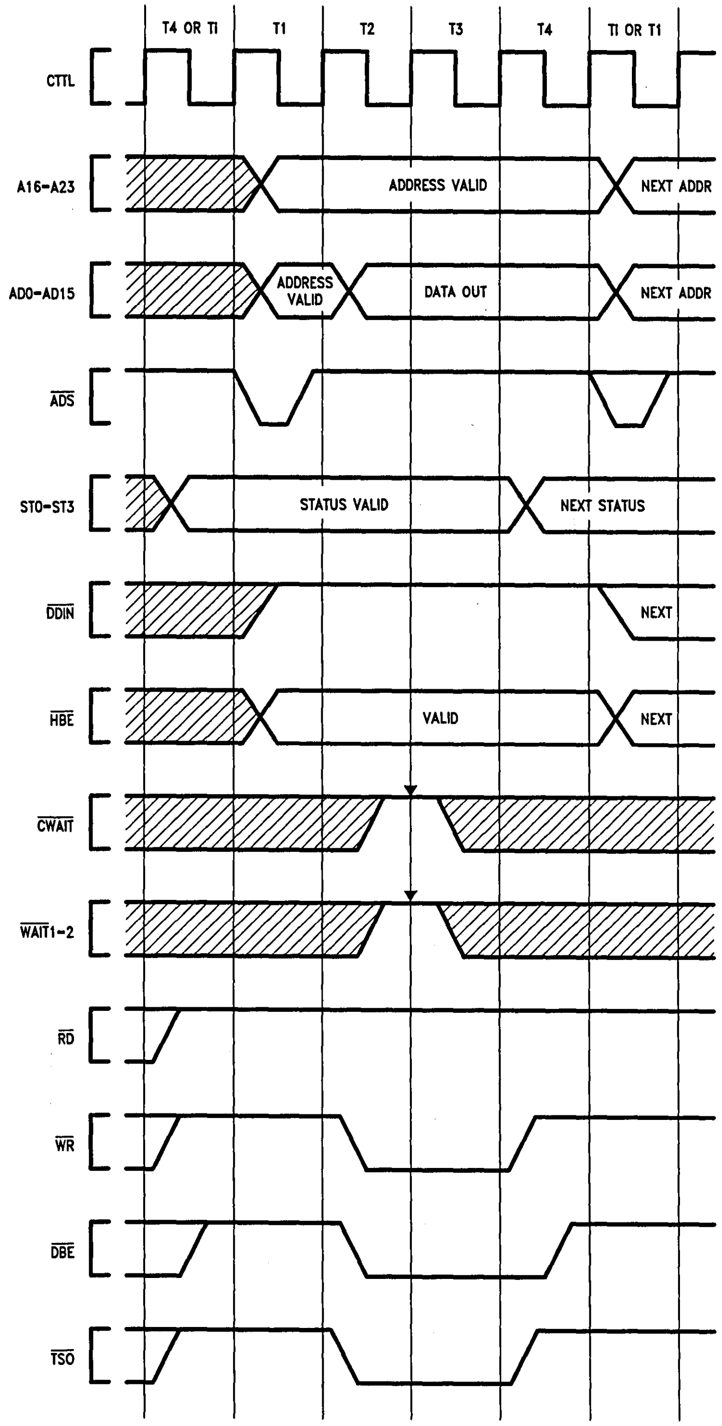
FIGURE 3-5. Bus Connections



TL/EE/9424–12

FIGURE 3-6. Read Cycle Timing

3.0 Functional Description (Continued)



TL/EE/9424-13

FIGURE 3-7. Write Cycle Timing

3.0 Functional Description (Continued)

At this time the signals $\overline{\text{TSO}}$ (Timing State Output), $\overline{\text{DBE}}$ (Data Buffer Enable) and either $\overline{\text{RD}}$ (Read Strobe) or $\overline{\text{WR}}$ (Write Strobe) will also be activated.

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 rising edge of CTTL, the $\overline{\text{CWAIT}}$ and $\overline{\text{WAIT}}$1-2 signals are sampled to determine whether the bus cycle will be extended. See Section 3.4.3.

If the CPU is performing a read cycle, the data bus (AD0-AD15) is sampled at the beginning of T4 on the rising edge of CTTL. Data must, however, be held a little longer to meet the data hold time requirements. The $\overline{\text{RD}}$ signal is guaranteed not to go inactive before this time, so its rising edge can be safely used to disable the device providing the input data.

The T4 state finishes the bus cycle. At the beginning of T4, the $\overline{\text{RD}}$ or $\overline{\text{WR}}$, and $\overline{\text{TSO}}$ signals go inactive, and on the falling edge of CTTL, $\overline{\text{DBE}}$ 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).

3.4.3 Cycle Extension

To allow sufficient access time for any speed of memory or peripheral device, the NS32CG16 provides for extension of a bus cycle. Any type of bus cycle except a Slave Processor cycle can be extended.

In *Figures 3-6* and *3-7*, note that during T3 all bus control signals from the CPU are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the $\overline{\text{WAIT}}$1-2 and $\overline{\text{CWAIT}}$ input signals.

At the end of state T2, on the rising edge of CTTL, $\overline{\text{WAIT}}$1-2 and $\overline{\text{CWAIT}}$ are sampled.

If any of these signals are active, the bus cycle will be extended by at least one clock cycle. Thus, one or more additional T3 state (also called wait state) will be inserted after the next T-State. Any combination of the above signals can be activated at one time. However, the $\overline{\text{WAIT}}$1-2 inputs are only sampled by the CPU at the end of state T2. They are ignored at all other times.

The $\overline{\text{WAIT}}$1-2 inputs are binary weighted, and can be used to insert up to 3 wait states, according to the following table.

$\overline{\text{WAIT}}$2	$\overline{\text{WAIT}}$1	Number of Wait States
HIGH	HIGH	0
HIGH	LOW	1
LOW	HIGH	2
LOW	LOW	3

$\overline{\text{CWAIT}}$ causes wait states to be inserted continuously as long as it is sampled active. It is normally used when the number of wait states to be inserted in the CPU bus cycle is not known in advance.

The following sequence shows the CPU response to the $\overline{\text{WAIT}}$1-2 and $\overline{\text{CWAIT}}$ inputs.

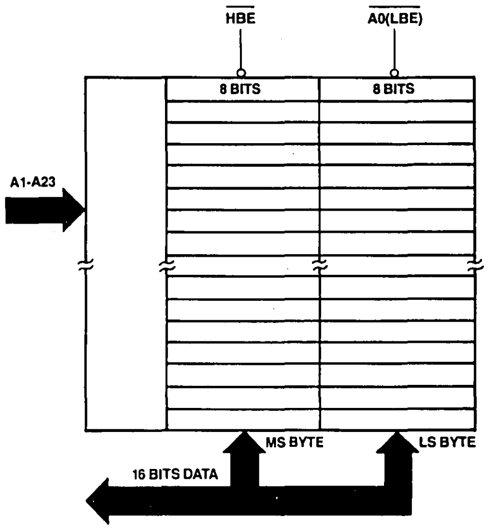
1. Start bus cycle.
2. Sample $\overline{\text{WAIT}}$1-2 and $\overline{\text{CWAIT}}$ at the end of state T2.
3. If the $\overline{\text{WAIT}}$1-2 inputs are both inactive, then go to step 6.
4. Insert the number of wait states selected by $\overline{\text{WAIT}}$1-2.
5. Sample $\overline{\text{CWAIT}}$ again.
6. If $\overline{\text{CWAIT}}$ is not active, then go to step 8.
7. Insert one wait state and then go to step 5.
8. Complete bus cycle.

Figure 3-8 shows a bus cycle extended by three wait states, two of which are due to $\overline{\text{WAIT}}$2, and one is due to $\overline{\text{CWAIT}}$.

3.4.4 Data Access Sequences

The 24-bit address provided by the NS32CG16 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 NS32CG16 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 NS32CG16 provides a special control signal, High Byte Enable ($\overline{\text{HBE}}$), 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 $\overline{\text{HBE}}$ is low. See *Figure 3-9*.



TL/EE/9424-15

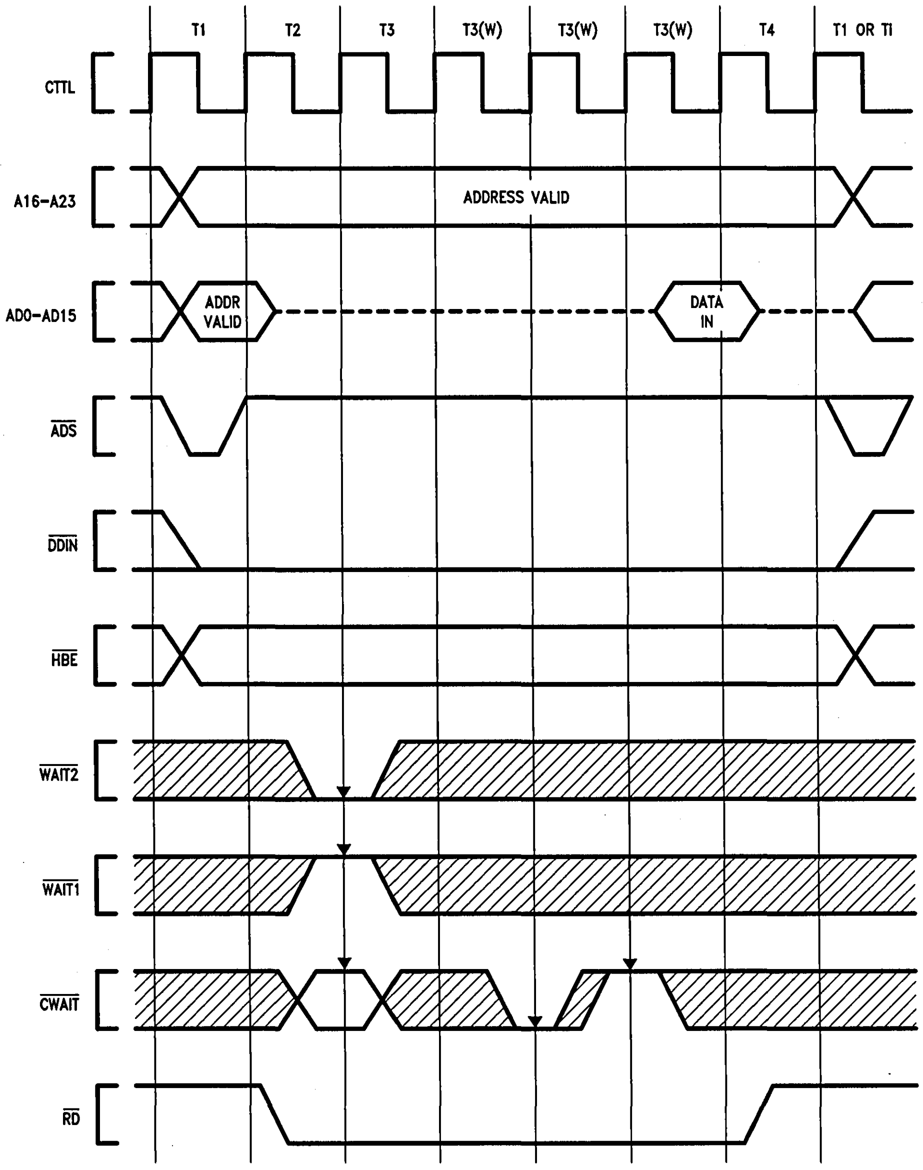
FIGURE 3-9. Memory Interface

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 3-2 gives the state of A0 and $\overline{\text{HBE}}$ for each category.

TABLE 3-2. Bus Cycle Categories

Category	$\overline{\text{HBE}}$	A0
Even Byte	1	0
Odd Byte	0	1
Even Word	0	0

3.0 Functional Description (Continued)



TL/EE/9424–14

FIGURE 3-8. Cycle Extension of a Read Cycle

3.0 Functional Description (Continued)

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 3-3 lists the bus cycle performed for each situation. For the timing of A0 and $\overline{\text{HBE}}$, see Section 3.4.2.

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 Multiply Accesses

The Multiply Extended Integer (MEI) instruction 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.

3.4.5 Instruction Fetches

Instructions for the NS32CG16 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 3.4.1).

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 3-2).

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.

3.4.6 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.

Table 3-4 shows the Interrupt Control sequences associated with each interrupt and with the return from its service routine. For full details of the NS32CG16 interrupt structure, see Section 3.8.

3.0 Functional Description (Continued)

TABLE 3-3. Access Sequences

A. Odd Word Access Sequence

BYTE 1	BYTE 0	
		← A

Cycle	Type	Address	$\overline{\text{HBE}}$	A0	High Bus	Low Bus
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

BYTE 3	BYTE 2	BYTE 1	BYTE 0	
				← A

Cycle	Type	Address	$\overline{\text{HBE}}$	A0	High Bus	Low Bus
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

BYTE 3	BYTE 2	BYTE 1	BYTE 0	
				← A

Cycle	Type	Address	$\overline{\text{HBE}}$	A0	High Bus	Low Bus
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

BYTE 7	BYTE 6	BYTE 5	BYTE 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0	
								← A

Cycle	Type	Address	$\overline{\text{HBE}}$	A0	High Bus	Low Bus
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.

Cycle	Type	Address	$\overline{\text{HBE}}$	A0	High Bus	Low Bus
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

BYTE 7	BYTE 6	BYTE 5	BYTE 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0	
								← A

Cycle	Type	Address	$\overline{\text{HBE}}$	A0	High Bus	Low Bus
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.

Cycle	Type	Address	$\overline{\text{HBE}}$	A0	High Bus	Low Bus
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)

TABLE 3-4. Interrupt Sequences

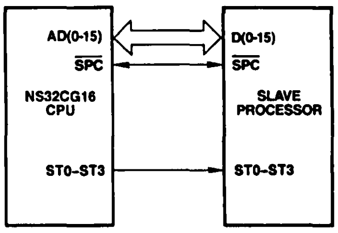
Cycle	Status	Address	$\overline{DDIN}$	$\overline{HBE}$	A0	High Bus	Low Bus
A. Non-Maskable Interrupt Control Sequence							
Interrupt Acknowledge							
1	0100	$FFFF00_{16}$	0	1	0	Don't Care	Don't Care
Interrupt Return							
None: Performed through Return from Trap (RETT) instruction.							
B. Non-Vectored Interrupt Control Sequence							
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 Sequence: 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 Sequence: 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)

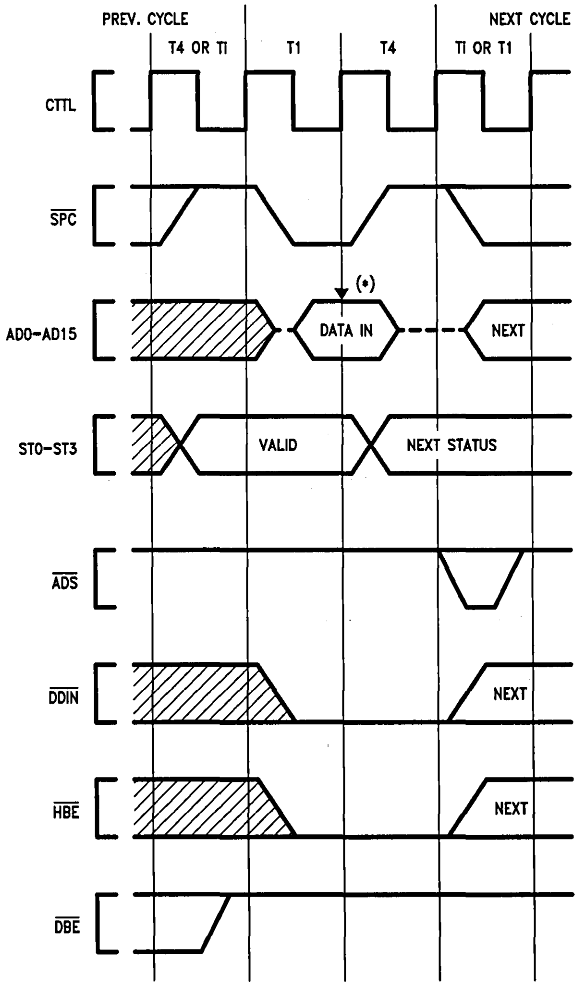
3.4.7 Slave Processor Communication

The $\overline{SPC}$ pin is used as the data strobe for Slave Processor transfers. In a Slave Processor bus cycle, data is transferred on the Data Bus (AD0–AD15), and the status lines ST0–ST3 are monitored by the 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 3.8 for full protocol sequences.



TL/EE/9424-16

FIGURE 3-10. Slave Processor Connections



TL/EE/9424-17

***Note:** CPU samples Data Bus here.

FIGURE 3-11. Slave Processor Read Cycle

3.0 Functional Description (Continued)

3.4.7.1 Slave Processor Bus Cycles

A Slave Processor bus cycle always takes exactly two clock cycles, labeled T1 and T4 (see *Figures 3-11* and *3-12*). 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.

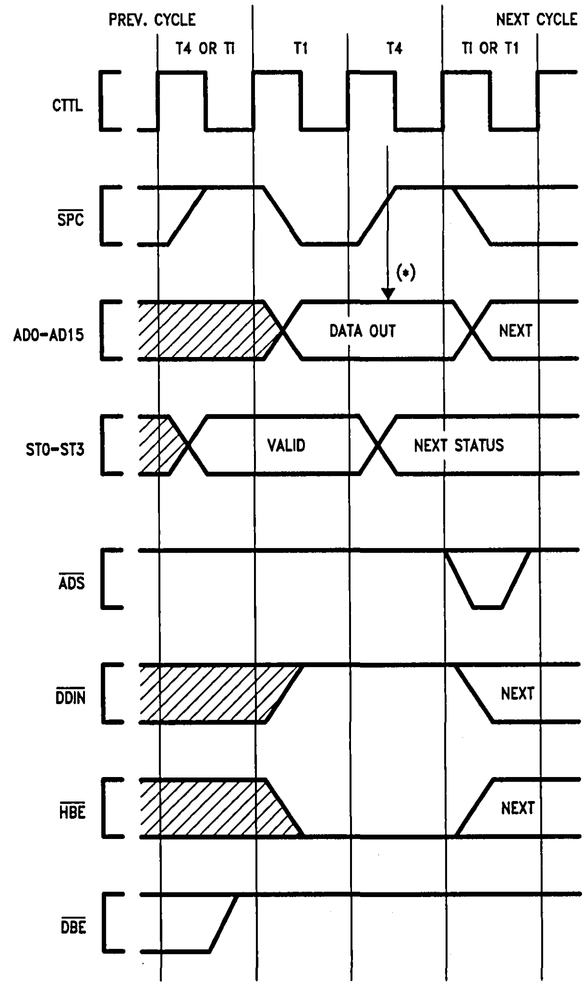
The CPU does not pulse the Address Strobe ($\overline{ADS}$), and no bus signals are generated. The direction of a transfer is determined by the sequence ("protocol") established by the instruction under execution; but the CPU indicates the direction on the $\overline{DDIN}$ pin for hardware debugging purposes.

3.4.7.2 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.

3.5 BUS ACCESS CONTROL

The NS32CG16 CPU has the capability of relinquishing its access to the bus upon request from a DMA controller or another CPU. This capability is implemented on the $\overline{HOLD}$ (Hold Request) and $\overline{HLDA}$ (Hold Acknowledge) pins. By as-



TL/EE/9424–18

***Note:** Slave Processor samples Data Bus here.

FIGURE 3-12. Slave Processor Write Cycle

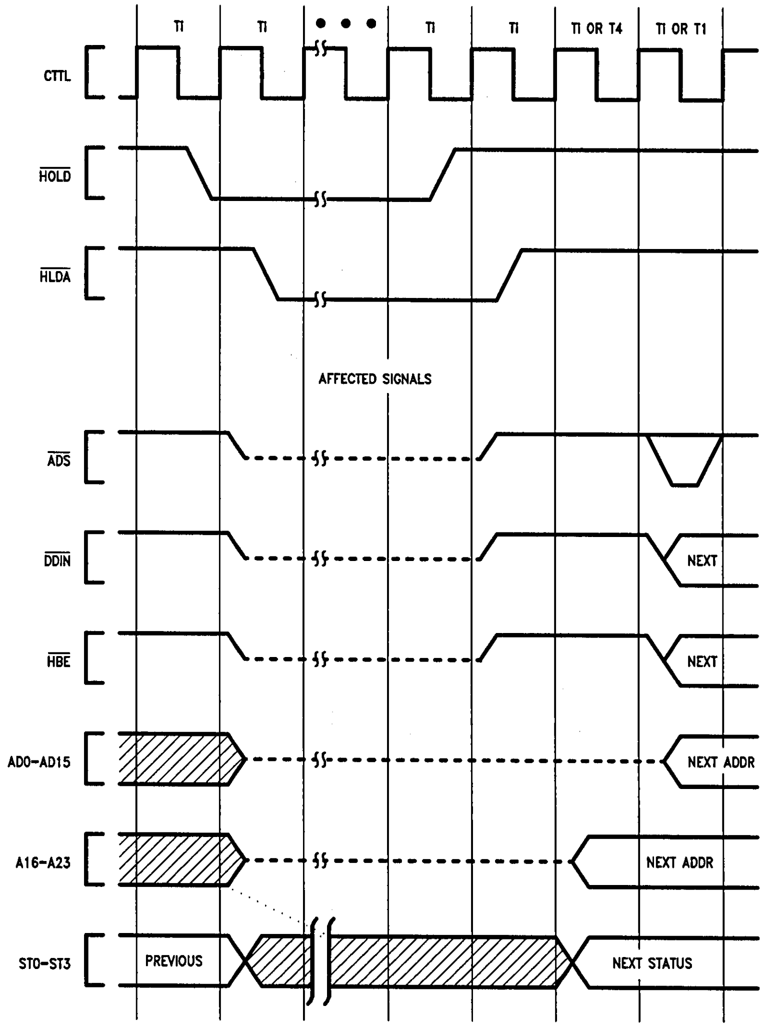
3.0 Functional Description (Continued)

serting $\overline{\text{HOLD}}$ low, an external device requests access to the bus. On receipt of $\overline{\text{HLDA}}$ from the CPU, the device may perform bus cycles, as the CPU at this point has set AD0–AD15, A16–A23 and $\overline{\text{HBE}}$ to the TRI-STATE® condition and has switched $\overline{\text{ADS}}$ and $\overline{\text{DDIN}}$ to the input mode. The CPU now monitors $\overline{\text{ADS}}$ and $\overline{\text{DDIN}}$ from the external device to generate the relevant strobe signals (i.e., $\overline{\text{TSO}}$, $\overline{\text{DBE}}$, $\overline{\text{RD}}$ or $\overline{\text{WR}}$). To return control of the bus to the CPU, the device sets $\overline{\text{HOLD}}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{\text{HLDA}}$ inactive.

How quickly the CPU releases the bus depends on whether it is idle on the bus at the time the $\overline{\text{HOLD}}$ request is made, as the CPU must always complete the current bus cycle. *Figure 3-13* shows the timing sequence when the CPU is idle. In this case, the CPU grants the bus during the immediately following clock cycle. *Figure 3-14* shows the sequence if the CPU is using the bus at the time that the $\overline{\text{HOLD}}$ 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.

Note 1: During DMA cycles the $\overline{\text{WAIT}}$1–2 signals should be kept inactive, unless they are also monitored by the DMA controller. If wait states are required, $\overline{\text{CWAIT}}$ should be used.

Note 2: The logic value of the status pins, ST0–ST3, is undefined during DMA activity.



TL/EE/9424-19

FIGURE 3-13. $\overline{\text{HOLD}}$ Timing, Bus Initially Idle

3.0 Functional Description (Continued)

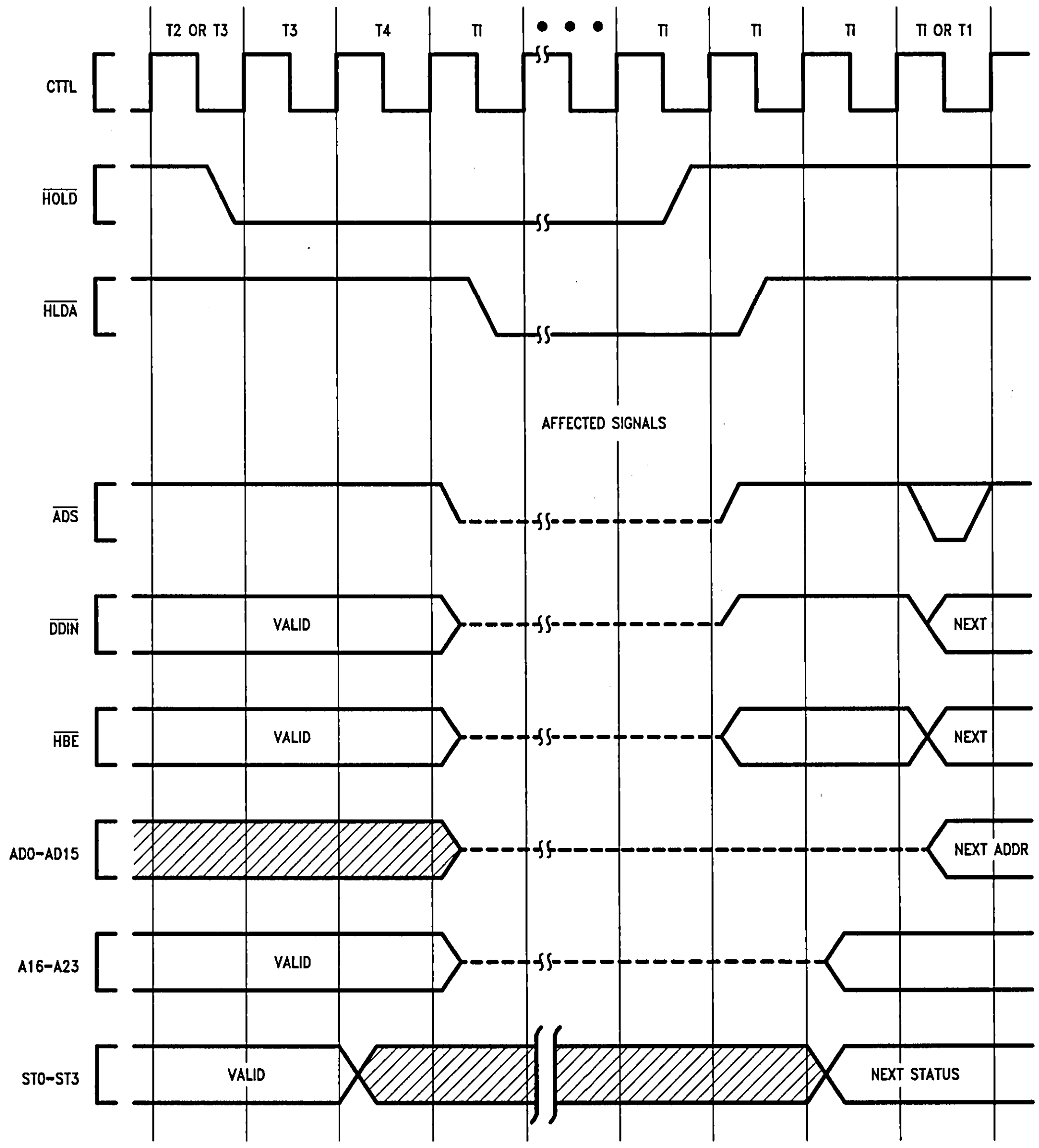


FIGURE 3-14. HOLD Timing, Bus Initially Not Idle

3.6 INSTRUCTION STATUS

In addition to the four bits of Bus Cycle status (ST0–ST3), the NS32CG16 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.

PFS (Program Flow Status) is pulsed low as each instruction begins execution. It is intended for debugging purposes.

U/S originates from the U bit of the Processor Status Register, and indicates whether the CPU is currently running in User or Supervisor mode. Although it is not synchronous to bus cycles, there are guarantees on its validity during any given bus cycle. See the Timing Specifications in Section 4.

3.0 Functional Description (Continued)

$\overline{\text{ILO}}$ (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. $\overline{\text{ILO}}$ is guaranteed to be active during the operand accesses performed by the interlocked instructions.

Note: The acknowledge of $\overline{\text{HOLD}}$ is on a cycle by cycle basis. Therefore, it is possible to have $\overline{\text{HLDA}}$ active when an interlocked operation is in progress. In this case, $\overline{\text{ILO}}$ remains low and the interlocked instruction continues only after $\overline{\text{HOLD}}$ is de-asserted.

3.7 EXCEPTION PROCESSING

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

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).

When an exception is recognized, the CPU saves the PC, PSR and 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 for details on the reset operation.

3.7.1 Exception Acknowledge Sequence

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

1) Adjustment of Registers.

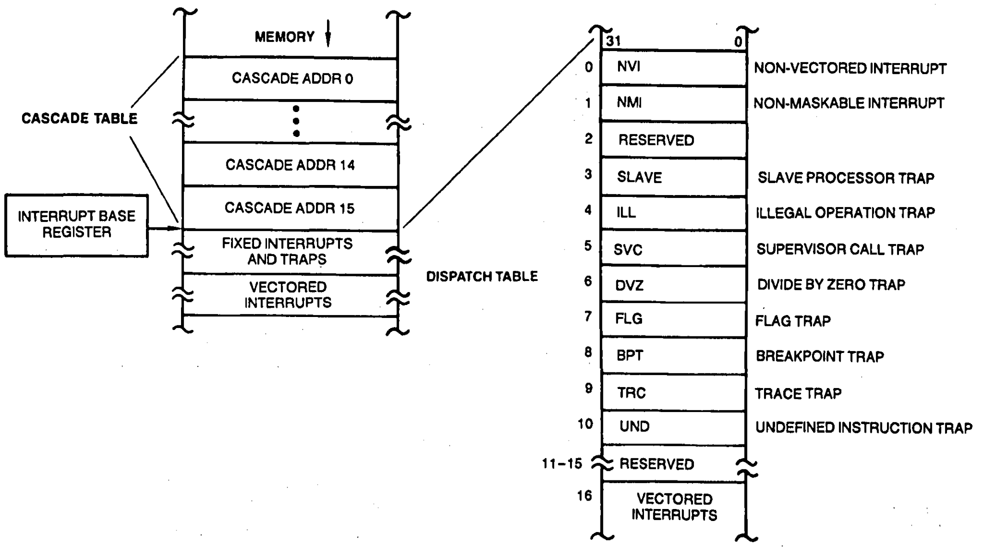
   Depending on the source of the exception, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR) and the currently-selected Stack Pointer (SP). A copy of the PSR is made, and the PSR is then set to reflect Supervisor Mode and selection of the Interrupt Stack.

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 3-15.* 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.


MEMORY
CASCADE ADDR 0
CASCADE TABLE
CASCADE ADDR 14
CASCADE ADDR 15
INTERRUPT BASE REGISTER
FIXED INTERRUPTS AND TRAPS
VECTORED INTERRUPTS
DISPATCH TABLE
31
0
0 NVI NON-VECTORED INTERRUPT
1 NMI NON-MASKABLE INTERRUPT
2 RESERVED
3 SLAVE SLAVE PROCESSOR TRAP
4 ILL ILLEGAL OPERATION TRAP
5 SVC SUPERVISOR CALL TRAP
6 DVZ DIVIDE BY ZERO TRAP
7 FLG FLAG TRAP
8 BPT BREAKPOINT TRAP
9 TRC TRACE TRAP
10 UND UNDEFINED INSTRUCTION TRAP
11-15 RESERVED
16 VECTORED INTERRUPTS

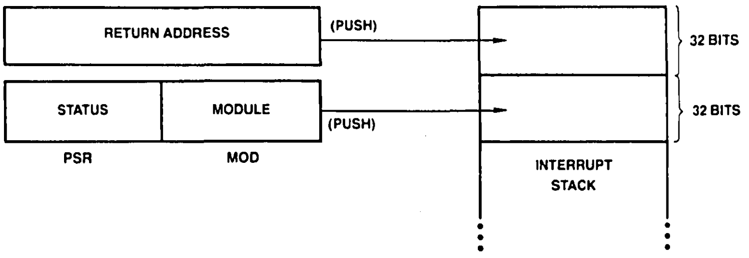
TL/EE/9424-21

FIGURE 3-15. Interrupt Dispatch and Cascade Tables

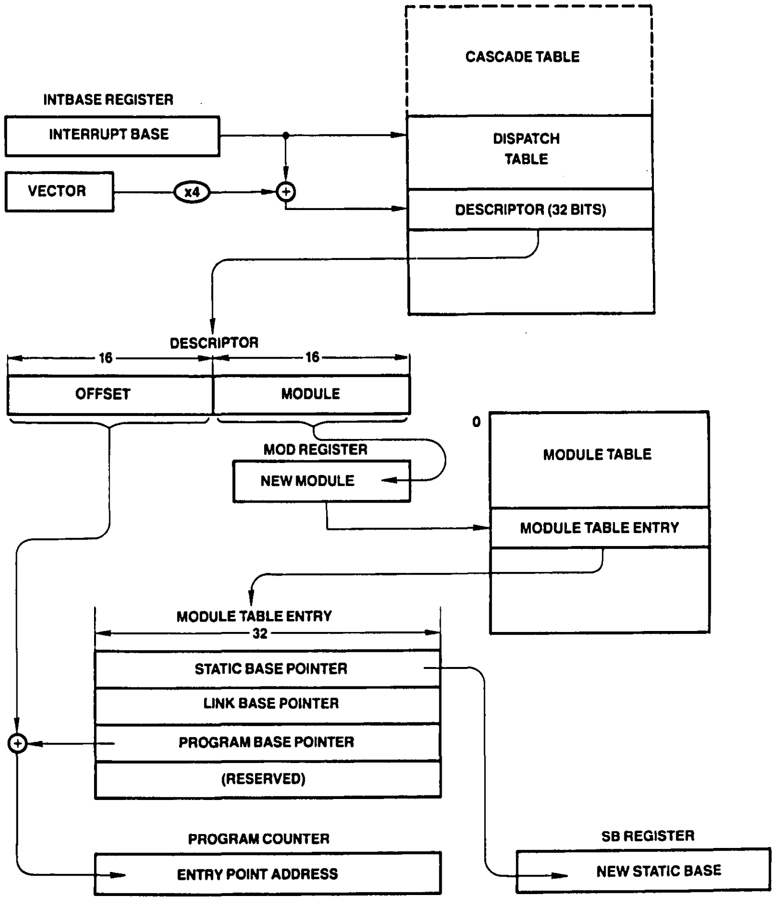
3.0 Functional Description (Continued)

This process is illustrated in *Figure 3-16*, from the viewpoint of the programmer.

Details on the sequences of events in processing interrupts and traps are given in the following sections.



TL/EE/9424-22



TL/EE/9424-23

FIGURE 3-16. Exception Acknowledge Sequence

3.0 Functional Description (Continued)

3.7.2 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 or a non-maskable interrupt 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 PSR, MOD, PC and SB registers to their previous contents.

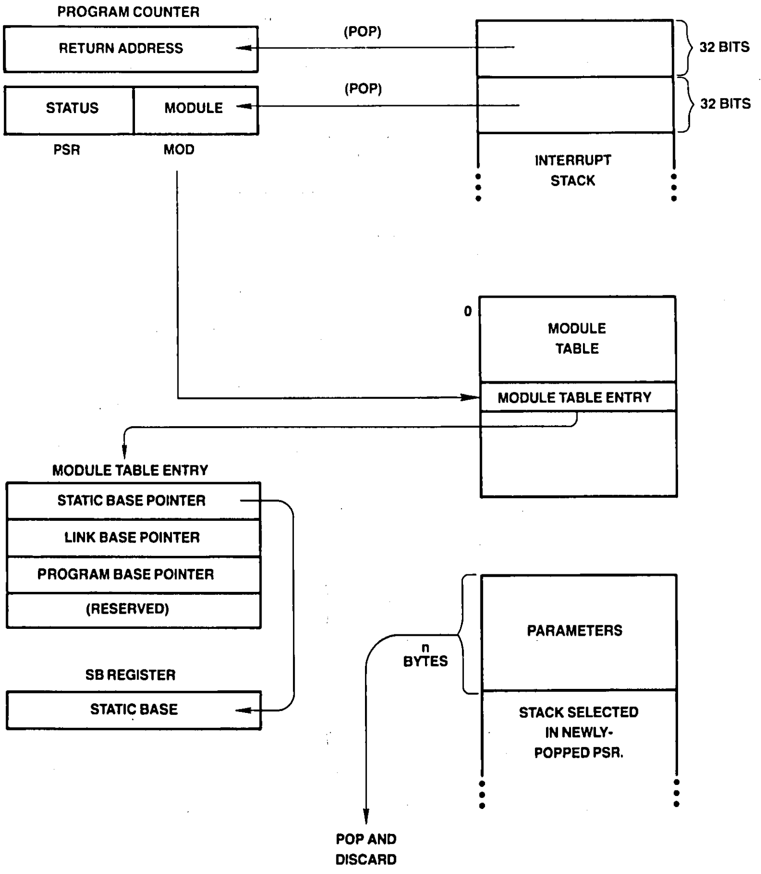
3.7.3 Maskable Interrupts

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 $\overline{\text{INT}}$ or $\overline{\text{NMI}}$ 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).

3.7.3.1 Non-Vectored Mode

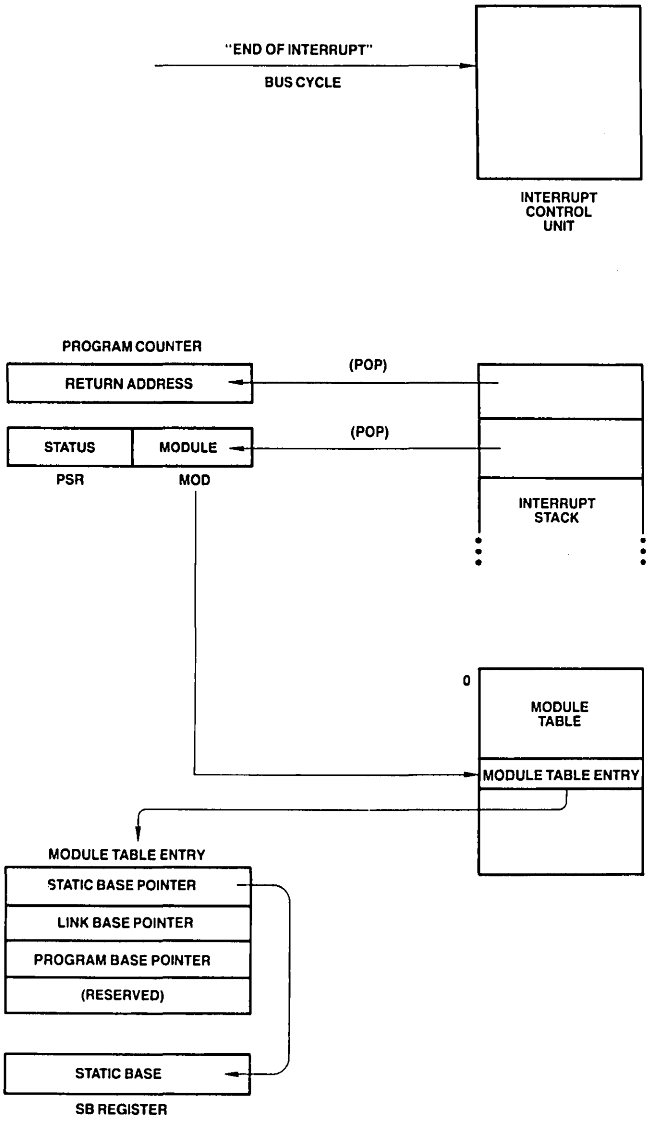
In the Non-Vectored mode, an interrupt request on the $\overline{\text{INT}}$ pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization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



TL/EE/9424-24

FIGURE 3-17. Return from Trap (RETT n) Instruction Flow

3.0 Functional Description (Continued)



TL/EE/9424-25

FIGURE 3-18. Return from Interrupt (RETI) Instruction Flow

3.7.3.2 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 $\overline{\text{INT}}$ pin, the CPU performs an "Interrupt Acknowledge, Master" bus cycle 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.

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).

3.7.3.3 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

3.0 Functional Description (Continued)

Unit (ICU) to transparently support cascading. *Figure 3-20* 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 $\overline{\text{INT}}$ pin.

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

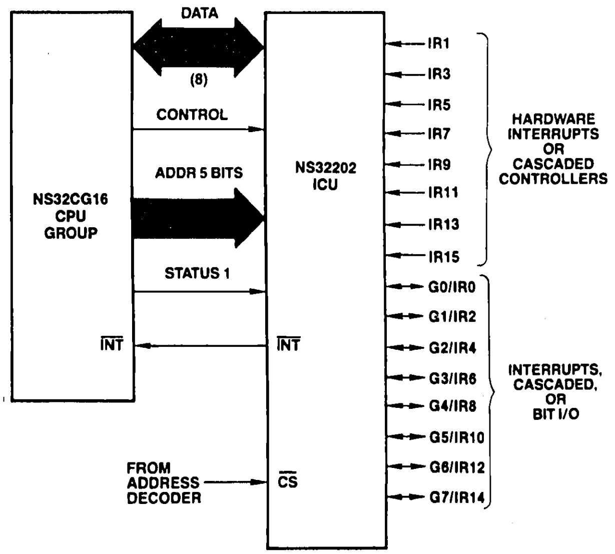
1) For each Cascaded ICU in the system, the Mater ICU must be informed of the line number (0 to 15) on which it receives the cascaded requests.
2) A Cascade Table must be established in memory. The Cascade Table is located in a NEGATIVE direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

Figure 3-15 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 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, informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

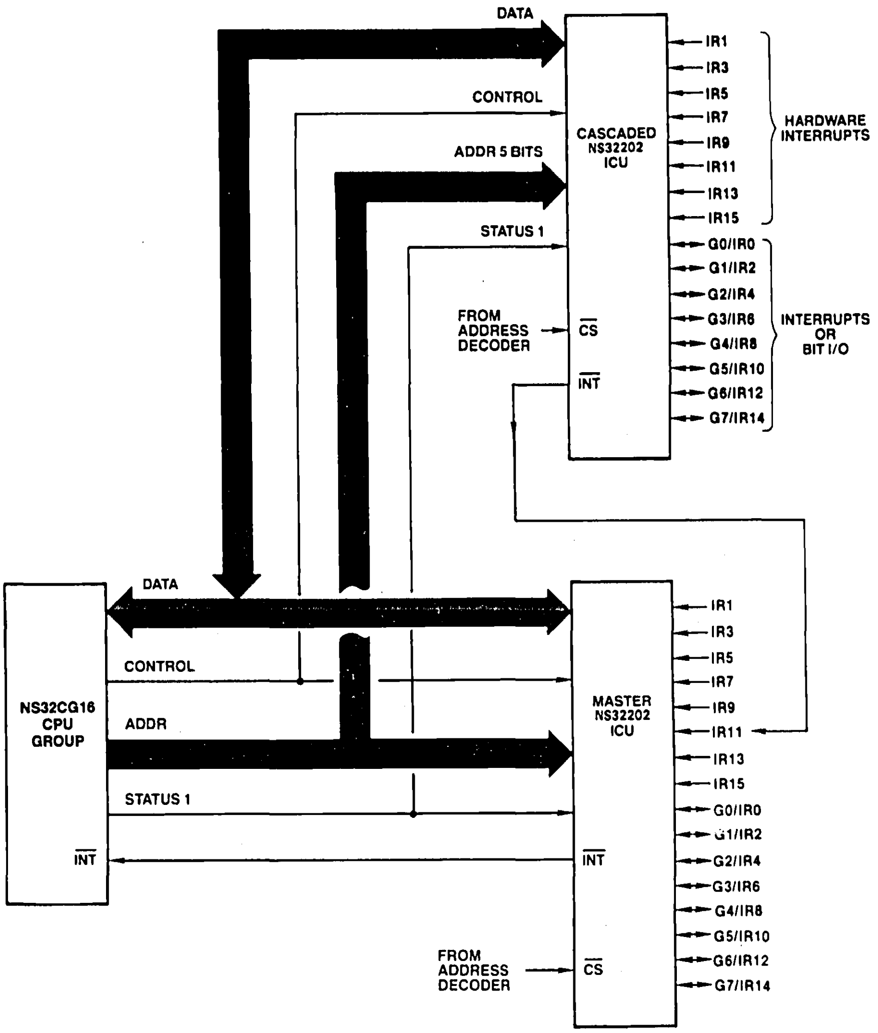
Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the Interrupt Mask Register of the Interrupt Controller. However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the $\overline{\text{INT}}$ 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/9424-26

FIGURE 3-19. Interrupt Control Unit Connections (16 Levels)

3.0 Functional Description (Continued)



TL/EE/9424-27

FIGURE 3-20. Cascaded Interrupt Control Unit Connections

3.7.4 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 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 3.7.7.1.

3.7.5 Traps

Traps are processing exceptions that are generated as direct results 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 NS32CG16 CPU are:

Trap (SLAVE): An exceptional condition was detected by the Floating Point Unit during the execution of a Slave Instruction. This trap is requested via the Status Word returned as part of the Slave Processor Protocol (Section 3.8.1).

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 Section 3.7.6.

Trap (UND): An undefined opcode was encountered by the CPU.

3.7.6 Instruction Tracing

Instruction tracing is a 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.

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.

While debugging the NS32CG16 instructions which have interior loops (BBOR, BBXOR, BBAND, BBFOR, EXTBLT, MOVMP, SBITPS, TBITS), special care must be taken with the single-step trap. If an interrupt occurs during a single-step of one of the graphics instructions, the interrupt will be serviced. Upon return from the interrupt service routine, the new NS32CG16 instruction will not be re-entered, due to a single-step trap. Both the NMI and INT interrupts will cause this behavior. Another single-step operation (S command in DBG16/MONCG) will resume from where the instruction was interrupted. There are no side effects from this early termination, and the instruction will complete normally.

For all other Series 32000 instructions, a single-step operation will complete the entire instruction before trapping back to the debugger. On the instructions mentioned above, several single-step commands may be required to complete the instruction, ONLY when interrupts are occurring.

There are some methods to give the appearance of single-stepping for these NS32CG16 instructions.

1. MON16/MONCG monitors the return from single-step trap vector, PC value. If the PC has not changed since the last single-step command was issued, the single-step operation is repeated. It is also advisable to ensure that one of the NS32CG16 instructions is being single-stepped, by inspecting the first byte of the address pointed to by the PC register. If it is 0x0E, then the instruction is an NS32CG16-specific instruction.
2. A breakpoint following the instruction would also trap after the instruction had completed.

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.

3.7.7 Priority Among Exceptions

The NS32CG16 CPU internally prioritizes simultaneous interrupt and trap requests as follows:

1) Traps other than Trace	(Highest priority)
2) Non-Maskable Interrupt	
3) Maskable Interrupts	
4) Trace Trap	(Lowest priority)

3.7.8 Exception Acknowledge Sequences: Detail Flow

For purposes of the following detailed discussion of interrupt and trap acknowledge sequences, a single sequence called "Service" is defined in *Figure 3-21*. 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.

3.7.8.1 Maskable/Non-Maskable Interrupt Sequence

This sequence is performed by the CPU when the $\overline{\text{NMI}}$ pin receives a falling edge, or the $\overline{\text{INT}}$ 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, or Graphics instructions which have interior loops (BBOR, BBXOR, BBAND, BBFOR, EXTBLT, MOVMP, SBITPS, TBITS), at the next interruptible point during its execution. The graphics instructions are interruptible.

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 3.4.1). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.

3.0 Functional Description (Continued)

4. If the interrupt is Non-Vectored:
   a. Read a byte from address $FFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Section 3.4.1). 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 3.4.1).
6. If "Byte" $\geq$ 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 3.4.1).
8. Push the PSR copy (from Step 2) onto the Interrupt Stack as a 16-bit value.
9. Perform Service (Vector, Return Address), *Figure 3-21.*

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 temporary 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 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.
8) Copy temporary MOD Register to MOD Register.

FIGURE 3-21. Service Sequence
Invoked during All Interrupt/Trap Sequences

3.7.8.2 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 3-21.*

3.7.8.3 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 3-21.*

3.8 SLAVE PROCESSOR INSTRUCTIONS

The NS32CG16 supports only one group of instructions, the floating point instruction set, as being executable by a slave processor. The floating point instruction set is validated by the F bit in the CFG register.

If a floating-point instruction is encountered and the F bit in the CFG register is not set, a Trap(UND) will result, without any slave processor communication attempted by the CPU. This allows software emulation in case an external floating point unit (FPU) is not used.

3.8.1 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 3-22.* While applying Status Code 1111 (Broadcast ID, Section 3.4.1), 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 3.4.1). 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.

3.0 Functional Description (Continued)

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 3.4.1).

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 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 3-22. Slave Processor Protocol

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.

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).

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). This word has the format shown in *Figure 3-23*. 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).

3.8.2 Floating Point Instructions

Table 3-5 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.

TABLE 3-5. 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

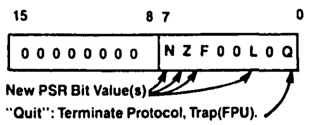
Note:
D = Double Word
i = integer size (B,W,D) specified in mnemonic.
f = Floating Point type (F,L) specified in mnemonic.
N/A = Not Applicable to this instruction.

3.0 Functional Description (Continued)

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 3-23)*.



TL/EE/9424-28

FIGURE 3-23. 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.

4.0 Device Specifications

4.1 NS32CG16 PIN DESCRIPTIONS

The following is a brief description of all NS32CG16 pins. The descriptions reference portions of the Functional Description, Section 3.

Unless otherwise indicated, reserved pins should be left open.

Note: An asterisk next to the signal name indicates a TRI-STATE condition for that signal during $\overline{\text{HOLD}}$ acknowledge.

4.1.1 Supplies

V_{CCL}	**Logic Power.** +5V positive supply for on-chip logic.
VCCCTTL, VCCFCLK, VCCAD, VCCIO	**Buffers Power.** +5V positive supplies for on-chip output buffers.
VSSL	**Logic Ground.** Ground reference for on-chip logic.
VSSFCLK, VSSNTSC, VSSHAD, VSSLAD, VSSIO	**Buffers Ground.** Ground reference for on-chip output buffers.

4.1.2 Input Signals

$\overline{\text{RSTI}}$	**Reset Input.** Schmitt triggered, asynchronous signal used to generate a CPU reset. See Section 3.3. **Note:** The reset signal is a true asynchronous input. Therefore, no external synchronizing circuit is needed. When $\overline{\text{RSTI}}$ changes right before the falling edge of CTTL, and meets the specified set-up time, it will be recognized on that falling edge. Otherwise it will be recognized on the falling edge of CTTL in the following clock cycle.
$\overline{\text{HOLD}}$	**Hold Request.** When active, causes the CPU to release the bus for DMA or multiprocessing purposes. See Section 3.5. **Note:** If the $\overline{\text{HOLD}}$ 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 $\overline{\text{HLDA}}$ latency. This is to avoid speed degradations in cases of heavy $\overline{\text{HOLD}}$ activity (i.e., DMA controller cycles interleaved with CPU cycles).
$\overline{\text{INT}}$	**Interrupt.** A low level on this pin requests a maskable interrupt. $\overline{\text{INT}}$ must be kept asserted until the interrupt is acknowledged.
$\overline{\text{NMI}}$	**Non-Maskable Interrupt.** A High-to-Low transition on this signal requests a non-maskable interrupt
$\overline{\text{CWAIT}}$	**Continuous Wait.** Causes the CPU to insert continuous wait states if sampled low at the end of T2 and each following T-State. See Section 3.4.3.
$\overline{\text{WAIT}}$1–2	**Two-Bit Wait State Inputs.** These inputs, collectively called $\overline{\text{WAIT}}$1–2, allow from zero to three wait states to be specified. They are binary weighted. See Section 3.4.3. **Note:** During a DMA cycle, $\overline{\text{WAIT}}$1–2 should be kept inactive unless they are also monitored by the DMA Controller. Wait states, in this case, should be generated through $\overline{\text{CWAIT}}$.
OSCIN	**Crystal/External Clock Input.** Input from a crystal or an external clock source. See Section 3.2.

4.1.3 Output Signals

A16–A23	***High-Order Address Bits.** These are the most significant 8 bits of the memory address bus.
$\overline{\text{HBE}}$	***High Byte Enable.** Status signal used to enable data transfers on the most significant byte of the data bus.

4.0 Device Specifications (Continued)

ST0–3 **Status.**
Bus cycle status code; ST0 is the 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.

U/$\overline{\text{S}}$ **User/Supervisor.**
User or Supervisor Mode status. High indicates User Mode; low indicates Supervisor Mode.

$\overline{\text{ILO}}$ **Interlocked Operation.**
When active, indicates that an interlocked operation is being executed.

$\overline{\text{HLDA}}$ **Hold Acknowledge.**
Activated by the CPU in response to the $\overline{\text{HOLD}}$ input to indicate that the CPU has released the bus.

$\overline{\text{PFS}}$ **Program Flow Status.**
A pulse on this signal indicates the beginning of execution of an instruction.

$\overline{\text{BPU}}$ **BPU Cycle.**
This signal is activated during a bus cycle to enable an external BITBLT processing unit. The EXTBLT instruction activates this signal.*

$\overline{\text{RSTO}}$ **Reset Output.**
This signal becomes active when $\overline{\text{RSTI}}$ is low, initiating a system reset.

$\overline{\text{RD}}$ **Read Strobe.**
Activated during CPU or DMAC read cycles to enable reading of data from memory or peripherals. See Section 3.4.2.

$\overline{\text{WR}}$ **Write Strobe.**
Activated during CPU or DMAC write cycles to enable writing of data to memory or peripherals.

***Note:** $\overline{\text{BPU}}$ is low (Active) only during bus cycles involving pre-fetching instructions and execution of EXTBLT operands. It is recommended that $\overline{\text{BPU}}$, $\overline{\text{ADS}}$ and status lines (ST0–ST3) be used to qualify BPU bus cycles. If a DMA circuit exists in the system, the $\overline{\text{HLDA}}$ signal should be used to further qualify BPU cycles. $\overline{\text{BPU}}$ may become active during T4 of a non-BPU bus cycle, and may become inactive during T4 of a BPU bus cycle. $\overline{\text{BPU}}$ must be qualified by $\overline{\text{ADS}}$ and status lines (ST0–ST3) to be used as an external gating signal.

$\overline{\text{TSO}}$ **Timing State Output.**
The falling edge of $\overline{\text{TSO}}$ identifies the beginning of state T2 of a bus cycle. The rising edge identifies the beginning of state T4.

$\overline{\text{DBE}}$ **Data Buffers Enable.**
Used to control external data buffers. It is active when the data buffers are to be enabled.

OSCOUT **Crystal Output.**
This line is used as the return path for the crystal (if used). When an external clock source is used, OSCOUT should be left unconnected or loaded with no more than 5 pF of stray capacitance.

FCLK **Fast Clock.**
This clock is derived from the clock waveform on OSCIN. Its frequency is either the same as OSCIN or is lower, depending upon the scale factor programmed into the CFG register. See Section 3.2.1.

PHI1, PHI2 **Two-Phase Clock.**
These outputs provide a two-phase clock with frequency half that of FCLK. They can be used to clock the DP8510/DP8511 BPU. The trace lengths of PHI1 and PHI2 should be shorter than 4 inches (10 centimeters) when connected to the BPU.

CTTL **System Clock.**
This clock is similar to PHI1 but has a much higher driving capability. The skew between its rising edge and PHI1 rising edge is kept to a minimum.

4.1.4 Input-Output Signals

AD0–15 ***Address/Data Bus.**
Multiplexed Address/Data information. Bit 0 is the least significant bit of each.

$\overline{\text{SPC}}$ **Slave Processor Control.**
Used by the CPU as the data strobe output for slave processor transfers; used by a slave processor to acknowledge completion of a slave instruction. See Section 3.4.7.1.

$\overline{\text{DDIN}}$ ***Data Direction.**
Status signal indicating the direction of the data transfer during a bus cycle. During $\overline{\text{HOLD}}$ acknowledge this signal becomes an input and determines the activation of $\overline{\text{RD}}$ or $\overline{\text{WR}}$.

$\overline{\text{ADS}}$ ***Address Strobe**
Controls address latches; signals the beginning of a bus cycle. During $\overline{\text{HOLD}}$ acknowledge this signal becomes an input and the CPU monitors it to detect the beginning of a DMA cycle and generate the relevant strobe signals. When a DMA is used, $\overline{\text{ADS}}$ should be pulled up to V_{CC} through a 10 kΩ resistor.

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

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	(Note 4)	2.0		V_{CC} + 0.5	V
V_{IL}	Low Level Input Voltage	(Note 3)	−0.5		0.8	V
V_{T+}	$\overline{\text{RSTI}}$ Rising Threshold Voltage	V_{CC} = 5.0V (Note 5)	2.5		3.5	V
V_{HYS}	$\overline{\text{RSTI}}$ Hysteresis Voltage	V_{CC} = 5.0V (Note 5)	0.8		1.8	V
V_{XL}	OSCIN Input Low Voltage				0.5	V
V_{XH}	OSCIN Input High Voltage		4.5			V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA (Note 6)	2.4			V
V_{OL}	Low Level Output Voltage	I_{OL} = 4 mA (Note 6)			0.45	V
I_{ILS}	$\overline{\text{SPC}}$ Input Current (low)	V_{IN} = 0.4V, $\overline{\text{SPC}}$ in Input Mode	0.05		1.0	mA
I_I	Input Load Current	$0 \le V_{IN} \le V_{CC}$, All Inputs except $\overline{\text{SPC}}$	−20		20	μA
I_L	Leakage Current Output and I/O Pins in TRI-STATE Input Mode	$0.4 \le V_{OUT} \le V_{CC}$	−20		20	μA
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C (Note 2)		140	200	mA
V_{PH}	PHI1, 2 High Level Output Voltage	I_{OH} = −400 μA	0.9 V_{CC}			V
V_{PL}	PHI1, 2 Low Level Output Voltage	I_{OL} = 4 mA			0.1 V_{CC}	V

Note 1: Care should be taken by designers to provide a minimum inductance path between the V_{SS} pins and system ground in order to minimize noise.

Note 2: I_{CC} is affected by the clock scaling factor selected by the C and M bits in the CFG register, see Section 3.2.1.

Note 3: $V_{IL\ min}$—in the range of −0.5V to −1.5V, the pulse must be ≤ 20 ns, and the period between pulses ≥ 120 ns.

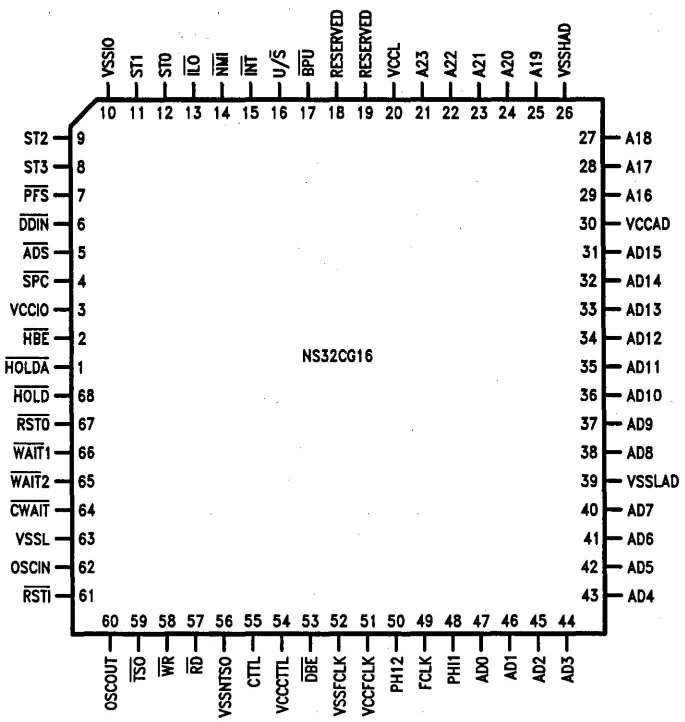
Note 4: $V_{IH\ max}$—in the range of V_{CC} + 0.5V to V_{CC} + 2.0V, the pulse must be ≤ 25 ns, and the period between pulses ≥ 120 ns.

Note 5: Not 100% tested.

Note 6: All outputs except PHI1 and PHI2.

4.0 Device Specifications (Continued)

68-Pin PCC Package



TL/EE/9424-29

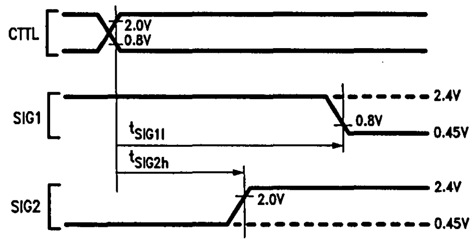
Bottom View

FIGURE 4-1. Connection Diagram

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 rising or falling edges of CTTL when the capacitive loading of CTTL is 100 pF, unless specifically stated otherwise. The timing specifications refer to 0.8 or 2.0V on the TTL output and input signals as illustrated in *Figures 4-2* and *4-3* unless specifically stated otherwise.



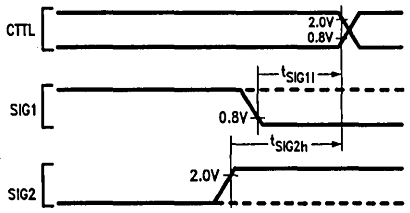
TL/EE/9424-30

FIGURE 4-2. Timing Specification Standard (TTL Output Signals)

4.0 Device Specifications (Continued)

ABBREVIATIONS:

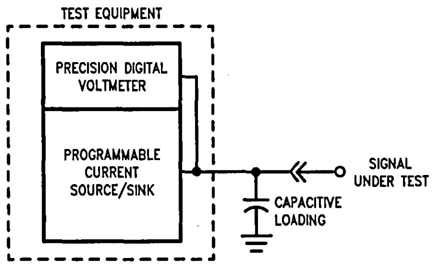
L.E. — leading edge	R.E. — rising edge
T.E. — trailing edge	F.E. — falling edge



TL/EE/9424-31

FIGURE 4-3. Timing Specification Standard (TTL Input Signals)

4.4.2 DEVICE TESTING



TL/EE/9424-65

FIGURE 4.4. Test Loading Configuration

TABLE 4-1. Test Loading Characteristics

Signal Name	Capacitive Loading	High Level Output Voltage ($I_{OH} = -400\ \mu A$)	Low Level Output Voltage ($I_{OL} = 4\ mA$)	Input Load Current ($0 \le V_{IN} \le V_{CC}$)	High Level Input Voltage	Low Level Input Voltage
$\overline{HBE}$, ST0–3, U/$\overline{S}$, $\overline{ILO}$, $\overline{HLDA}$, $\overline{PFS}$, $\overline{BPU}$, $\overline{RSTO}$, $\overline{RD}$, $\overline{WR}$, $\overline{TSO}$, $\overline{DBE}$, FCLK, $\overline{DDIN}$, $\overline{ADS}$	50 pF	$2.0V \le V_{OH} \le V_{CC}+0.5V$	$-0.5V \le V_{OL} \le 0.8V$	$-20\ \mu A \le I_I \le 20\ \mu A$	$2.0V \le V_{IH} \le V_{CC}+0.5V$	$-0.5V \le V_{IL} \le 0.45V$
$\overline{RSTI}$, $\overline{HOLD}$, $\overline{INT}$, $\overline{NMI}$, $\overline{CWAIT}$, $\overline{WAIT}$1–2	50 pF			$-20\ \mu A \le I_I \le 20\ \mu A$	$2.0V \le V_{IH} \le V_{CC}+0.5V$	$-0.5V \le V_{IL} \le 0.8V$
OSCIN	50 pF			$-20\ \mu A \le I_I \le 20\ \mu A$	$4.5V \le V_{IH} \le V_{CC}+0.5V$	$-0.5V \le V_{IL} \le 0.5V$
AD0–15, A16–23, CTTL	100 pF	$2.0V \le V_{OH} \le V_{CC}+0.5V$	$-0.5V \le V_{OL} \le 0.8V$	$-20\ \mu A \le I_I \le 20\ \mu A$	$2.4V \le V_{IH} \le V_{CC}+0.5V$	$-0.5V \le V_{IL} \le 0.45V$
PHI1, PHI2	30 pF	(Note 2)	(Note 2)			
$\overline{SPC}$	30 pF	$2.0V \le V_{OH} \le V_{CC}+0.5V$	$-0.5V \le V_{OL} \le 0.8V$	$50\ \mu A \le I_I \le 1.0\ mA$	$2.0V \le V_{IH} \le V_{CC}+0.5V$	$-0.5V \le V_{IL} \le 0.4V$
OSCOUT (Note 1)	see Table 3-1	$2.0V \le V_{OH} \le V_{CC}+0.5V$	$-0.5V \le V_{OL} \le 0.8V$			

Note 1: The maximum capacitive loading of OSCOUT is given in Table 3-1 when the NS32CG16's oscillator is driven with a crystal. If a single phase clock source is used, OSCOUT should be left unconnected or loaded with no more than 5 pF of stray capacitance.

Note 2: As stated in Table 4.4.3.

4.0 Device Specifications (Continued)

4.4.3 Timing Tables

4.4.3.1 Output Signals: Internal Propagation Delays, NS32CG16-10 and NS32CG16-15

Name	Figure	Description	Reference/Conditions	NS32CG16-10		NS32CG16-15 (Note 3)		Units
				Min	Max	Min	Max	
t_{CTp}	4-20	CTTL Clock Period	R.E., CTTL to Next R.E., CTTL	100	1000	66	1000	ns
t_{CTh}	4-20	CTTL High Time At 1.5V (Both Edges) (see Note 1)	25 pF–100 pF Capacitive Load	0.40	0.57	0.46	0.58	t_{CTp}
t_{CTl}	4-20	CTTL Low Time	At 0.8V 25 pF–100 pF Capacitive Load	0.42	0.56	0.40	0.53	t_{CTp}
t_{CTr}	4-20	CTTL Rise Time	0.8V to 2.0V V_{CC} on R.E., CTTL	0	8	0	6	ns
t_{CTf}	4-20	CTTL Fall Time	2.0V to 0.8V V_{CC} on F.E., CTTL	0	8	0	6	ns
$t_{CLw(1,2)}$	4-20	PHI1, PHI2 Pulse Width	At 2.0V on PHI1, PHI2 (Both Edges)	0.35	0.55	0.32	0.53	t_{CTp}
t_{CLh}	4-20	Clock High Time	At 90% V_{CC} on PHI1, PHI2 (Both Edges)	0.22	0.50	0.28	0.50	t_{CTp}
$t_{nOVL(1,2)}$	4-20	PHI1, PHI2, Non-Overlap Time	At 50% V_{CC} on PHI1, PHI2	2		2		ns
t_{XFr}	4-20	OSCIN to FCLK R.E. Delay	80% V_{CC} on R.E., OSCIN to R.E., FCLK	2	29	2	25	ns
t_{FCr}	4-20	FCLK to CTTL R.E. Delay	R.E., FCLK to R.E., CTTL	−2	10	−2	10	ns
t_{FCf}	4-20	FCLK to CTTL F.E. Delay	R.E., FCLK to F.E., CTTL	−2	10	−2	10	ns
t_{PCr}	4-20	CTTL and PHI1 Skew	R.E., CTTL to R.E., PHI1	−4	4	−4	4	ns
t_{ALv}	4-5	Address Bits 0–15 Valid	after R.E., CTTL T1		40	4	30	ns
t_{ALh}	4-5	Address Bits 0–15 Hold	after R.E., CTTL T2	5		5		ns
t_{AHv}	4-5	Address Bits 16–23 Valid	after R.E., CTTL T1		40	0	30	ns
t_{AHh}	4-5	Address Bits 16–23 Hold	after R.E., CTTL Next T1 or Ti	0		0		ns
t_{ALfr}	4-5	Address Bits 0–15 floating (during read)	after R.E., CTTL T2	5	38	5	28	ns
t_{ALnfr}	4-5	AD0–AD15 Floating (Note 2)		4	36	4	26	ns

Note 1: Device testing is performed using the Test Loading Characteristics in Table 4.1. Additional timing data for CTTL with various capacitive loads is not 100% tested.

Note 2: t_{ALnfr} is address bits 0–15 floating or not active after R.E. CTTL T1. This is only valid if the previous CPU cycle was a read *(Figure 4.5)*. A previous write may have "data" active into T1 of the next cycle which then becomes "address" during T1.

Note 3: 15 MHz specifications are only guaranteed when t_{CTp} = 66 ns.

4.0 Device Specifications (Continued)

4.4.3.1 Output Signals: Internal Propagation Delays, NS32CG16-10 and NS32CG16-15 (Continued)

Name	Figure	Description	Reference/Conditions	NS32CG16-10		NS32CG16-15		Units
				Min	Max	Min	Max	
t_{ALf}	4-7	AD0–AD15 Floating (Caused by $\overline{\text{HOLD}}$)	after R.E., CTTL Ti		25		18	ns
t_{AHf}	4-7	A16–A23 Floating	after R.E., CTTL Ti		25		18	ns
t_{ALnf}	4-5, 4-8	Address Bits 0–15 Not Floating	after R.E., CTTL T1	4	36	4	26	ns
t_{AHnf}	4-8	Address Bits 16–23 Not Floating	after R.E., CTTL T4	4	36	4	26	ns
t_{Dv}	4-6, 4-10	Data Valid (Write Cycle)	after R.E., CTTL T2 or T1		50		38	ns
t_{Dh}	4-6, 4-10	Data Hold	after R.E., CTTL Next T1 or Ti	0		0		ns
t_{ADSa}	4-5	$\overline{\text{ADS}}$ Signal Active	after R.E., CTTL T1	5	35	5	26	ns
t_{ADSia}	4-5	$\overline{\text{ADS}}$ Signal Inactive	after F.E., CTTL T1	5	35	5	25	ns
t_{ADSw}	4-6	$\overline{\text{ADS}}$ Pulse Width	at 15% V_{CC} (Both Edges)	30		25		ns
t_{ADSf}	4-7	$\overline{\text{ADS}}$ Floating	after R.E., CTTL Ti		55		40	ns
t_{ADSr}	4-8	$\overline{\text{ADS}}$ Return from Floating	after R.E., CTTL Ti		55		40	ns
t_{ALADSs}	4-6	Address Bits 0–15 Setup	before $\overline{\text{ADS}}$ T.E.	25		20		ns
t_{AHADSs}	4-6	Address Bits 16–23 Setup	before $\overline{\text{ADS}}$ T.E.	25		20		ns
t_{ALADSh}	4-5	Address Bits 0–15 Hold	after $\overline{\text{ADS}}$ T.E.	12		12		ns
t_{HBEv}	4-5	$\overline{\text{HBE}}$ Signal Valid	after R.E., CTTL T1		60		38	ns
t_{HBEh}	4-5	$\overline{\text{HBE}}$ Signal Hold	after R.E., CTTL Next T1 or Ti	0		0		ns
t_{HBEf}	4-7	$\overline{\text{HBE}}$ Signal Floating	after R.E., CTTL Ti		55		40	ns
t_{HBEr}	4-8	$\overline{\text{HBE}}$ Return from Floating	after R.E., CTTL Ti		55		40	ns
t_{DDINv}	4-5	$\overline{\text{DDIN}}$ Signal Valid	after R.E., CTTL T1		65		38	ns
t_{DDINh}	4-5	$\overline{\text{DDIN}}$ Signal Hold	after R.E., CTTL Next T1 or Ti	0		0		ns
t_{DDINf}	4-7	$\overline{\text{DDIN}}$ Floating	after R.E., CTTL Ti		55		40	ns
t_{DDINr}	4-8	$\overline{\text{DDIN}}$ Return from Floating	after R.E., CTTL Ti		55		40	ns
t_{SPCa}	4-10	$\overline{\text{SPC}}$ Output Active	after R.E., CTTL T1		35	5	26	ns
t_{SPCia}	4-10	$\overline{\text{SPC}}$ Output Inactive	after R.E., CTTL T4		35	5	26	ns
t_{SPCnf}	4-12	$\overline{\text{SPC}}$ Output Non-Forcing (Note 2)	after F.E., CTTL T4		$t_{CTp} + 10$		$t_{CTp} + 8$	ns
t_{HLDAa}	4-7	$\overline{\text{HLDA}}$ Signal Active	after R.E., CTTL Ti		50		26	ns
t_{HLDAia}	4-8	$\overline{\text{HLDA}}$ Signal Inactive	after R.E., CTTL Ti		50		26	ns
t_{STv}	4-5	Status ST0–ST3 Valid	after R.E., CTTL T4 (before T1, see Note 1)		45		38	ns
t_{STh}	4-5	Status ST0–ST3 Hold	after R.E., CTTL T4	0		0		ns
t_{BPUv}	4-5	$\overline{\text{BPU}}$ Signal Valid	after R.E., CTTL T4		45		30	ns
t_{BPUh}	4-5	$\overline{\text{BPU}}$ Signal Hold	after R.E., CTTL T4	5		5		ns

Note 1: 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 ...".

Note 2: If the CPU is connected directly to the FPU and the CTTL loading is not violated, the CPU and FPU will function correctly together. The CPU and FPU connect directly without buffers. They should be located less than 4 inches (10 centimeters) apart. t_{SPCa} and t_{SPCia} will track each other on all CPU's and therefore it is not possible to have a minimum t_{SPCia} and a maximum t_{SPCa} value. The pulse width minimum, t_{SPCw}, of the FPU will not be violated by the NS32CG16 when connected directly to the FPU.

4.0 Device Specifications (Continued)

4.4.3.1 Output Signals: Internal Propagation Delays, NS32CG16-10 and NS32CG16-15 (Continued)

Name	Figure	Description	Reference/Conditions	NS32CG16-10		NS32CG16-15		Units
				Min	Max	Min	Max	
t_{TSOa}	4-5	$\overline{TSO}$ Signal Active	after R.E., CTTL T2		15	2	12	ns
t_{TSOia}	4-5	$\overline{TSO}$ Signal Inactive	after R.E., CTTL T4		15	0	10	ns
t_{RDa}	4-5	$\overline{RD}$ Signal Active	after R.E., CTTL T2		20		15	ns
t_{RDia}	4-5	$\overline{RD}$ Signal Inactive	after R.E., CTTL T4		20	0	15	ns
t_{WRa}	4-6	$\overline{WR}$ Signal Active	after R.E., CTTL T2		20		15	ns
t_{WRia}	4-6	$\overline{WR}$ Signal Inactive	after R.E., CTTL T4		20	0	15	ns
$t_{DBEa(R)}$	4-5	$\overline{DBE}$ Active (Read Cycle)	after F.E., CTTL T2		21		15	ns
$t_{DBEa(W)}$	4-6	$\overline{DBE}$ Active (Write Cycle)	after R.E., CTTL T2		28		15	ns
t_{DBEia}	4-5, 4-6	$\overline{DBE}$ Inactive	after F.E., CTTL T4		23		15	ns
t_{USv}	4-5	U/$\overline{S}$ Signal Valid	after R.E., CTTL T4		40		30	ns
t_{USh}	4-5	U/$\overline{S}$ Signal Hold	after R.E., CTTL T4	5		5		ns
t_{PFSa}	4-13	$\overline{PFS}$ Signal Active	after F.E., CTTL		50		38	ns
t_{PFSia}	4-13	$\overline{PFS}$ Signal Inactive	after F.E., CTTL		50		38	ns
t_{PFSw}	4-13	$\overline{PFS}$ Pulse Width	at 15% V_{CC} (Both Edges)	70		45		ns
t_{NSPF}	4-16	Nonsequential Fetch to Next $\overline{PFS}$ Clock Cycle	after R.E., CTTL T1	4		4		t_{CTp}
t_{PFNS}	4-15	$\overline{PFS}$ Clock Cycle to Next Nonsequential Fetch	before R.E., CTTL T1	4		4		t_{CTp}
t_{LXPF}	4-14	Last Operand Transfer of an Instruction to Next $\overline{PFS}$ Clock Cycle	before R.E., CTTL T1 of First Bus Cycle of Transfer	0		0		t_{CTp}
t_{ILOs}	4-17	$\overline{ILO}$ Signal Setup	before R.E., CTTL T1 of First Interlocked Read Cycle	30		30		ns
t_{ILOh}	4-18	$\overline{ILO}$ Signal Hold	after R.E., CTTL T3 of Last Interlocked Write Cycle	5		5		ns
t_{ILOa}	4-19	$\overline{ILO}$ Signal Active	after R.E., CTTL		55		35	ns
t_{ILOia}	4-19	$\overline{ILO}$ Signal Inactive	after R.E., CTTL		55		35	ns
t_{RSTOa}	4-22	$\overline{RSTO}$ Signal Active	after R.E., CTTL		21		15	ns
t_{RSTOia}	4-22	$\overline{RSTO}$ Signal Inactive	after R.E., CTTL		21		15	ns
t_{RTOI}	4-22	Reset to Idle	after F.E. of RSTO		10		10	t_{CTp}
t_{RTOF}	4-22	Reset to Fetch	after R.E. of RSTO		8		8	t_{CTp}

4.0 Device Specifications (Continued)

4.4.3.2 Input Signal Requirements: NS32CG16-10 and NS32CG16-15

Name	Figure	Description	Reference/Conditions	NS32CG16-10		NS32CG16-15		Units
				Min	Max	Min	Max	
t_{Xp}	4-20	OSCIN Clock Period	R.E., OSCIN to Next R.E., OSCIN	50	500	33	500	ns
t_{Xh}	4-20	OSCIN High Time (External Clock)	at 4.2V (Both Edges)	16		11		ns
t_{Xl}	4-20	OSCIN Low Time	at 1.0V (Both Edges)	16		11		ns
t_{DIs}	4-5, 4-11	Data In Setup	before R.E., CTTL T4	18		15		ns
t_{DIh}	4-5, 4-11	Data In Hold (see Note 1)	after R.E., CTTL T4	7		7		ns
t_{CWs}	4-5, 4-6	$\overline{\text{CWAIT}}$ Signal Setup	before R.E., CTTL T3 or T3(w)	20		20		ns
t_{CWh}	4-5, 4-6	$\overline{\text{CWAIT}}$ Signal Hold	after R.E., CTTL T3 or T3(w)	5		5		ns
t_{Ws}	4-5, 4-6	$\overline{\text{WAIT}}$n Signals Setup	before R.E., CTTL T3 or T3(w)	20		20		ns
t_{Wh}	4-5, 4-6	$\overline{\text{WAIT}}$n Signals Hold	after R.E., CTTL T3 or T3(w)	5		5		ns
t_{HLDs}	4-7, 4-8	$\overline{\text{HOLD}}$ Setup Time	before R.E., CTTL TX2 or Ti	30		22		ns
t_{HLDh}	4-7, 4-8	$\overline{\text{HOLD}}$ Hold Time	after R.E., CTTL Ti	0		0		ns
t_{PWR}	4-21	Power Stable to $\overline{\text{RSTI}}$ R.E.	after V_{CC} Reaches 4.5V	50		33		μs
t_{RSTs}	4-21, 4-22	$\overline{\text{RSTI}}$ Signal Setup	before F.E., CTTL	20		20		ns
t_{RSTw}	4-22	$\overline{\text{RSTI}}$ Pulse Width	at 0.8V (Both Edges)	64		64		t_{CTp}
t_{SPCh}	4-12	SPC Hold Time (see Note 3)	after R.E., CTTL	0		0		ns
t_{INTh}	4-23	$\overline{\text{INT}}$ Signal Hold	after Interrupt Acknowledge		8		8	t_{CTp}
t_{NMIw}	4-24	$\overline{\text{NMI}}$ Pulse Width	at 0.8V (Both Edges)	70		50		ns
t_{SPCd}	4-12	$\overline{\text{SPC}}$ Pulse Delay from Slave	after F.E., CTTL T4	2		2		t_{CTp}
t_{SPCs}	4-12	$\overline{\text{SPC}}$ Input Setup	before F.E., CTTL	37		30		ns
t_{ADSs}	4-9	$\overline{\text{ADS}}$ Input Setup	before F.E., CTTL	15		10		ns
t_{ADSh}	4-9	$\overline{\text{ADS}}$ Input Hold (see Note 2)	after F.E., CTTL T1	10		10		ns
t_{DDINs}	4-9	$\overline{\text{DDIN}}$ Input Setup	before F.E., CTTL	15		10		ns
t_{DDINh}	4-9	$\overline{\text{DDIN}}$ Input Hold	after R.E., CTTL T4	7		5		ns

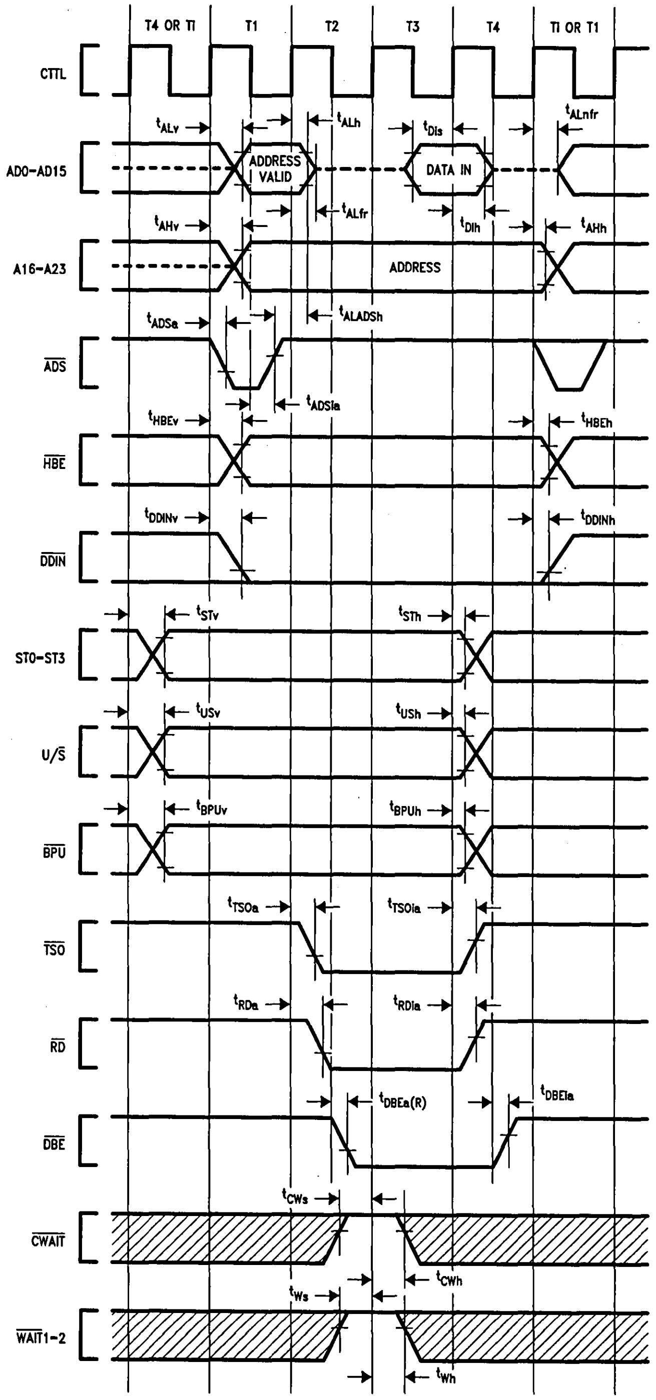
Note 1: t_{DIh} is always less than or equal to t_{RDia}.

Note 2: $\overline{\text{ADS}}$ must be deasserted before state T4 of the DMA controller cycle.

Note 3: Not tested, guaranteed by design.

4.0 Device Specifications (Continued)

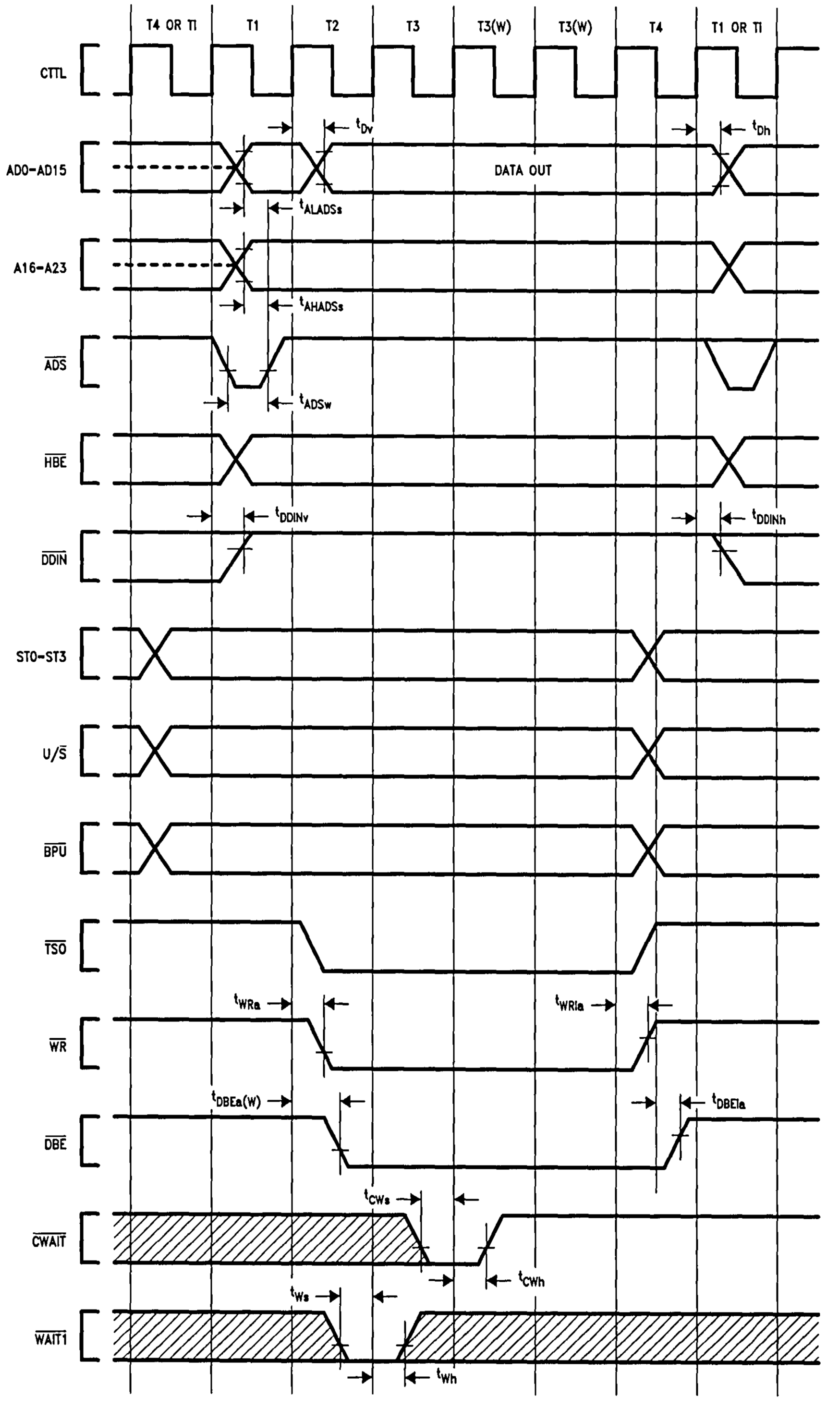
4.4.4 TIMING DIAGRAMS



TL/EE/9424–32

FIGURE 4-5. Read Cycle

4.0 Device Specifications (Continued)



TL/EE/9424–33

FIGURE 4-6. Write Cycle

4.0 Device Specifications (Continued)

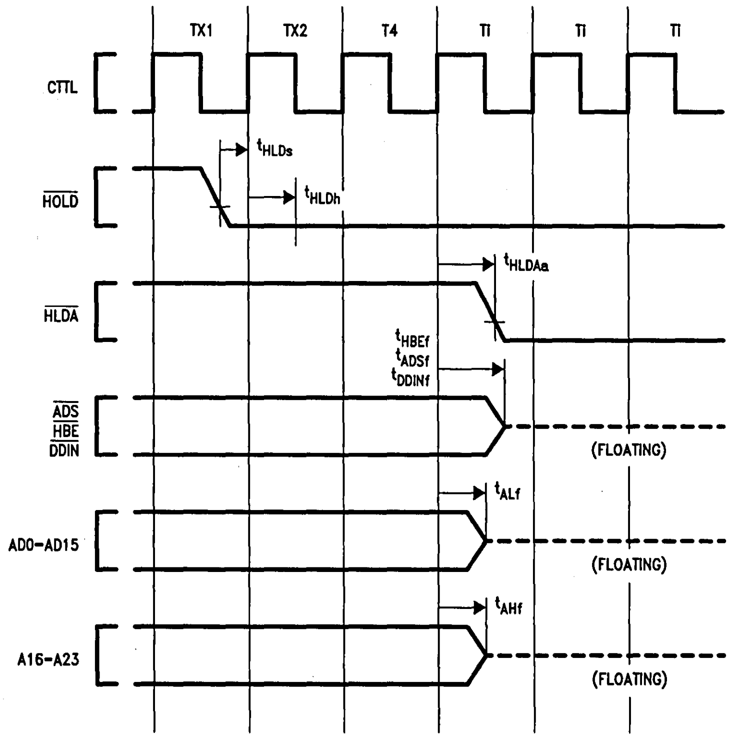
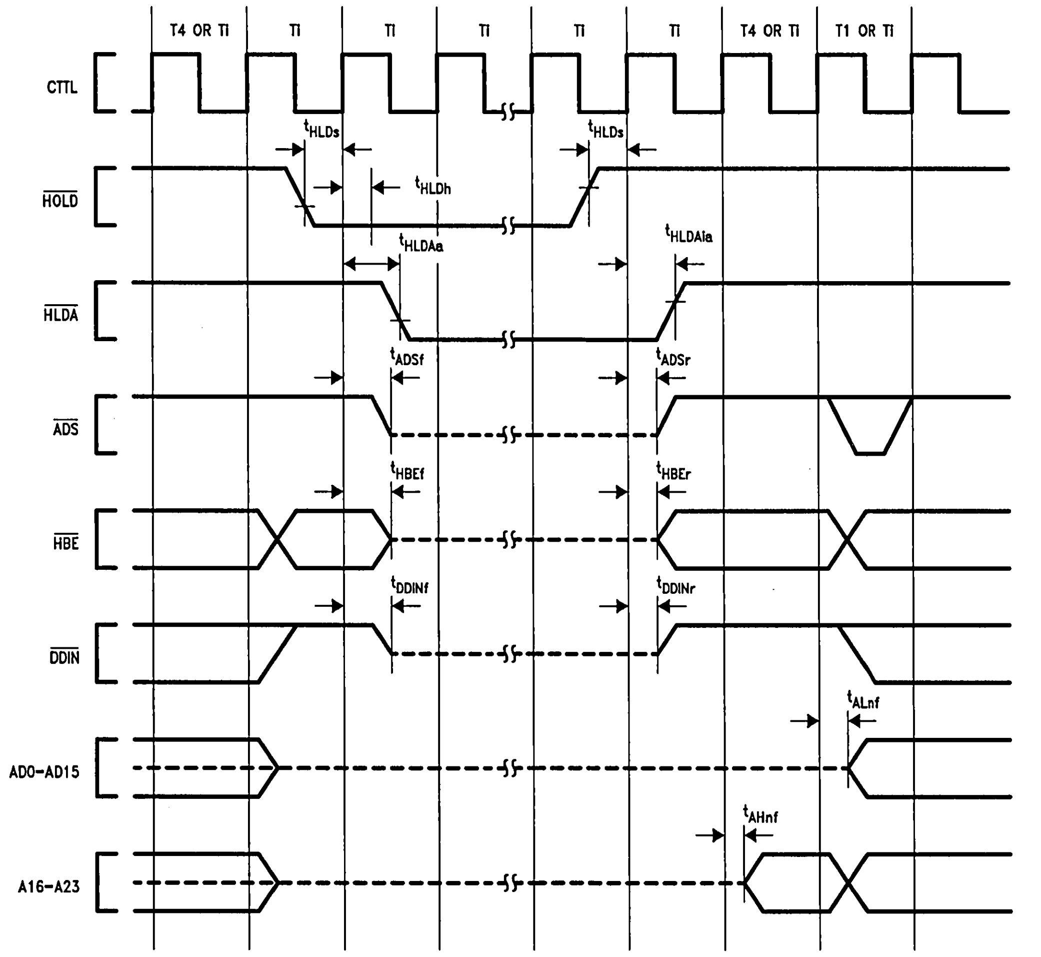


FIGURE 4-7. $\overline{\text{HOLD}}$ Acknowledge Timing (Bus Initially Not Idle)

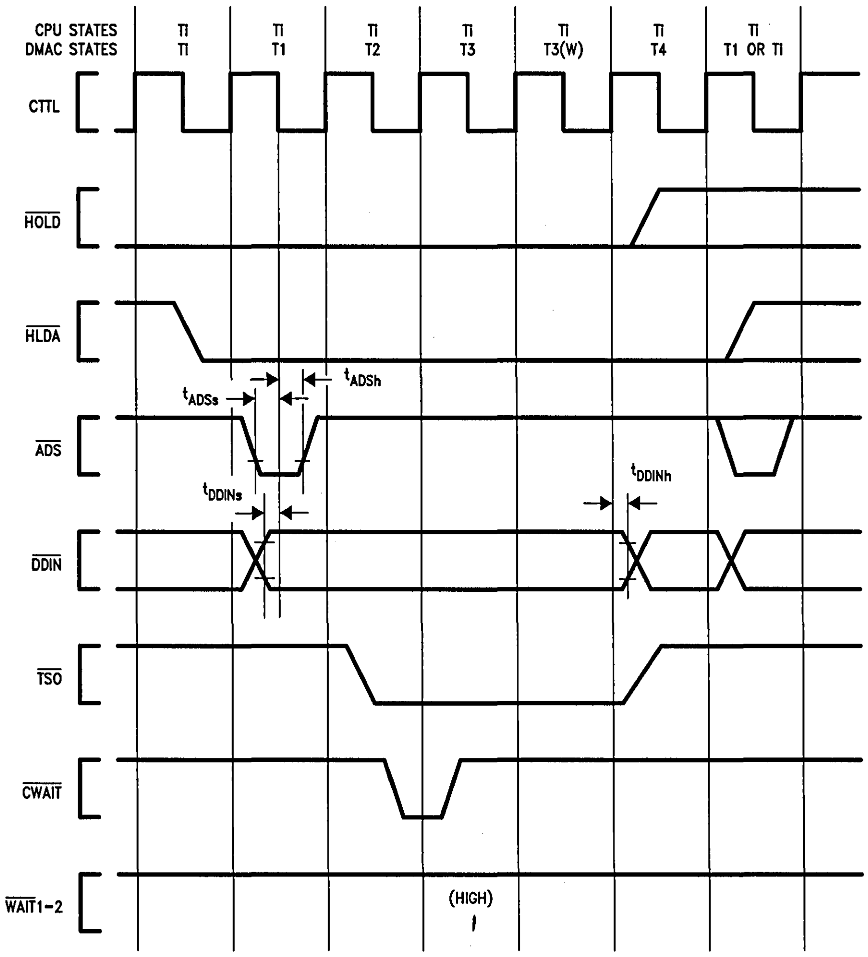
Note: When the bus is not idle, $\overline{\text{HOLD}}$ must be asserted before the rising edge of CTTL of the timing state that precedes state T4 in order for the request to be acknowledged.



TL/EE/9424–35

FIGURE 4-8. $\overline{\text{HOLD}}$ Timing (Bus Initially Idle)

4.0 Device Specifications (Continued)



TL/EE/9424-36

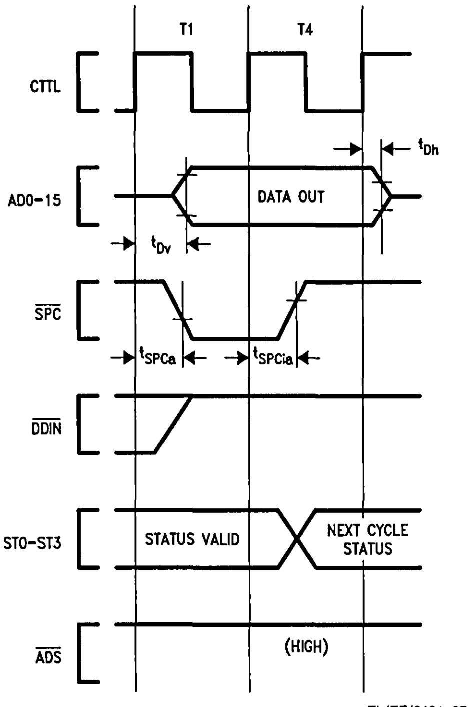
FIGURE 4-9. DMAC Initiated Bus Cycle

Note 1: $\overline{\text{ADS}}$ must be deactivated before state T4 of the DMA controller cycle.

Note 2: During a DMA cycle $\overline{\text{WAIT}}$1-2 must be kept inactive unless they are monitored by the DMA Controller. A DMA cycle is similar to a CPU cycle. The NS32CG16 generates $\overline{\text{TSO}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\overline{\text{DBE}}$. The DMAC drives the address/data lines $\overline{\text{HBE}}$, $\overline{\text{ADS}}$ and $\overline{\text{DDIN}}$.

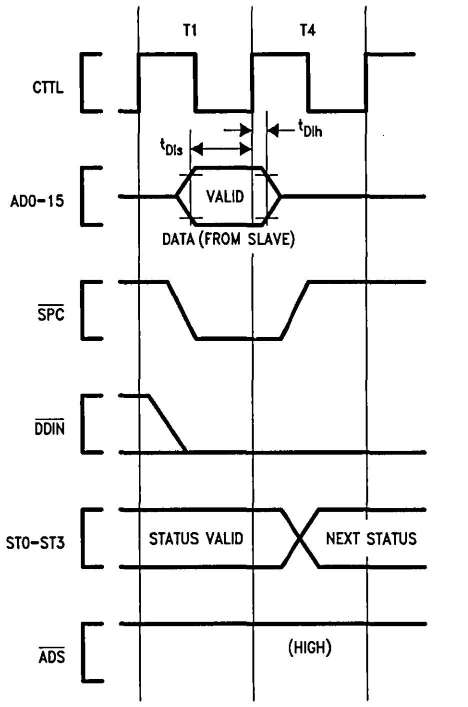
Note 3: During a DMA cycle, if the $\overline{\text{ADS}}$ signal is pulsed in order to initiate a bus cycle, the $\overline{\text{HOLD}}$ signal must remain asserted until state T4 of the DMAC cycle.

4.0 Device Specifications (Continued)



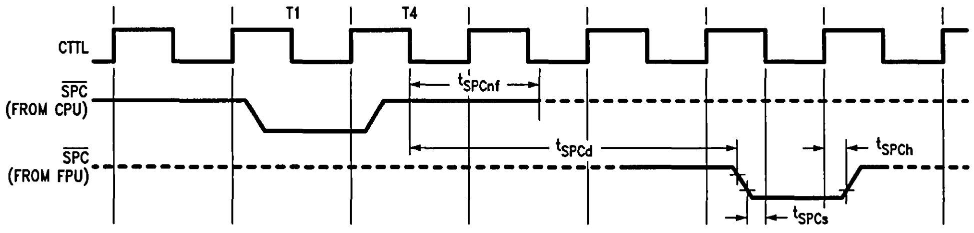
TL/EE/9424–37

FIGURE 4-10. Slave Processor Write Timing



TL/EE/9424–38

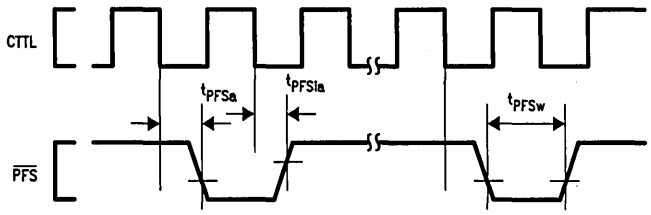
FIGURE 4-11. Slave Processor Read Timing



TL/EE/9424–39

FIGURE 4-12. $\overline{SPC}$ Timing

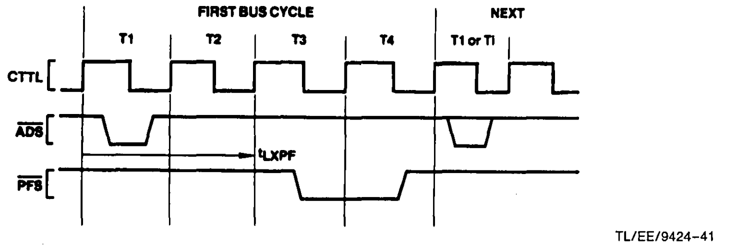
After transferring the last operand to the FPU, the CPU turns OFF the output driver and holds $\overline{SPC}$ high with an internal 5 kΩ pullup.



TL/EE/9424–40

FIGURE 4-13. Relationship of $\overline{PFS}$ to Clock Cycles

4.0 Device Specifications (Continued)



Note: In a transfer of a Read-Modify-Write type operand, this is the Read transfer, displaying RMW Status (Code 1011).

FIGURE 4-14. Relationship Between Last Data Transfer of an Instruction and $\overline{PFS}$ Pulse of Next Instruction

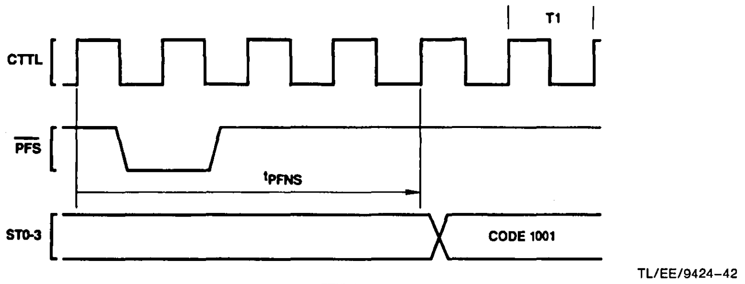


FIGURE 4-15. Guaranteed Delay, $\overline{PFS}$ to Non-Sequential Fetch

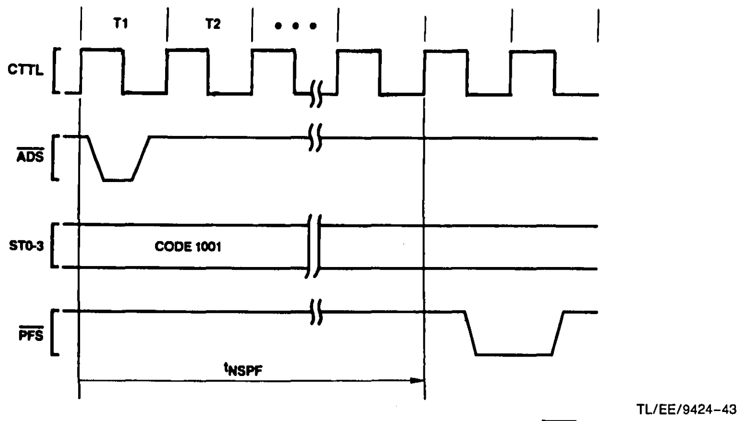


FIGURE 4-16. Guaranteed Delay, Non-Sequential Fetch to $\overline{PFS}$

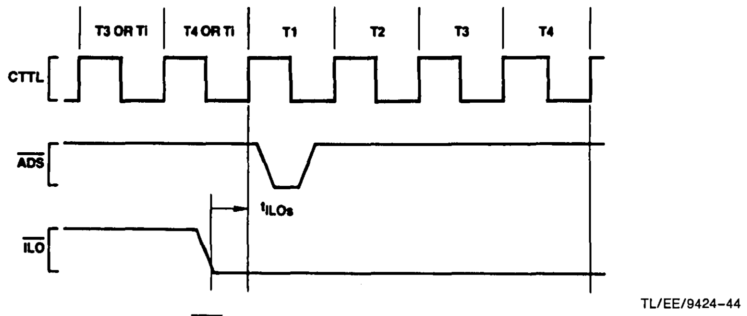
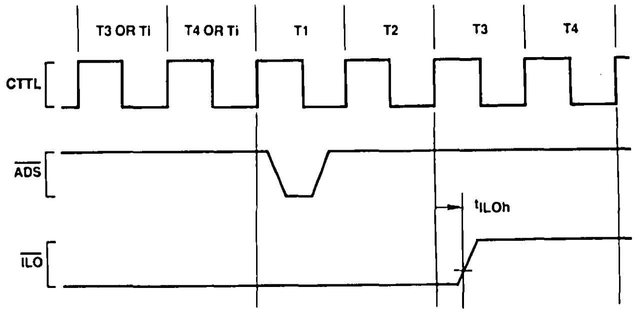


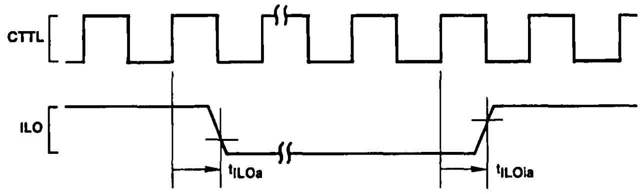
FIGURE 4-17. Relationship of $\overline{ILO}$ to First Operand Cycle of an Interlocked Instruction

4.0 Device Specifications (Continued)



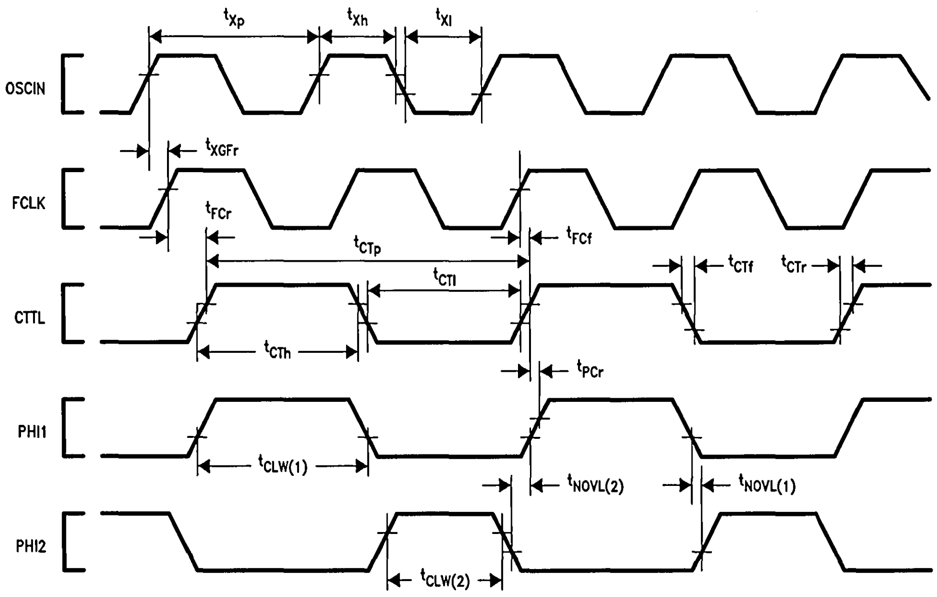
TL/EE/9424-45

FIGURE 4-18. Relationship of $\overline{\text{ILO}}$ to Last Operand Cycle of an Interlocked Instruction



TL/EE/9424-46

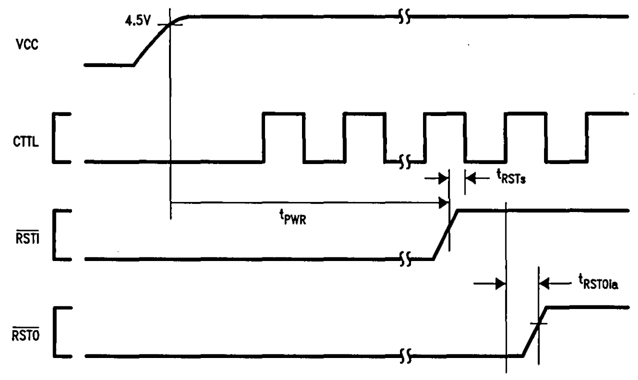
FIGURE 4-19. Relationship of $\overline{\text{ILO}}$ to Any Clock Cycle



TL/EE/9424-47

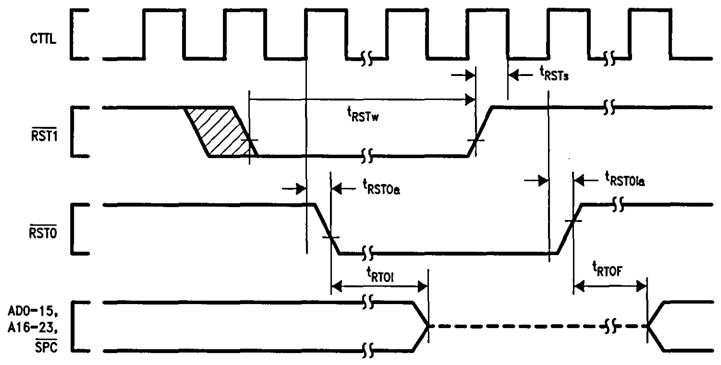
FIGURE 4-20. Clock Waveforms

4.0 Device Specifications (Continued)



TL/EE/9424–48

FIGURE 4-21. Power-On Reset

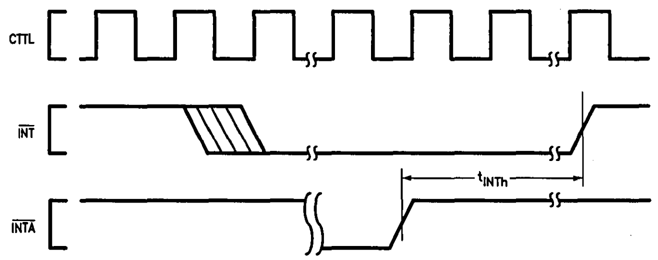


TL/EE/9424–49

FIGURE 4-22. Non-Power-On Reset

Note 1: During Reset the $\overline{\text{HOLD}}$ signal must be kept high.

Note 2: After $\overline{\text{RSTI}}$ is deasserted the first bus cycle will be an instruction fetch at address zero.

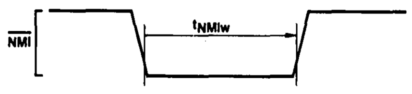


TL/EE/9424–50

FIGURE 4-23. $\overline{\text{INT}}$ Interrupt Signal Detection

Note 1: Once $\overline{\text{INT}}$ is asserted, it must remain asserted until it is acknowledged.

Note 2: $\overline{\text{INTA}}$ is the Interrupt Acknowledge bus cycle (not a CPU signal). Refer to Section 3.4.1 and Table 3.4.



TL/EE/9424–51

FIGURE 4-24. $\overline{\text{NMI}}$ 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)

op = Operation Code
Valid encodings shown with each format.

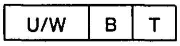
gen, gen 1, gen 2 = General Addressing Mode Field
See Sec. 2.3.2 for encodings.

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)

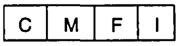
short = Short Immediate value. May contain
quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
cond: Condition Code (above), in Scond.
areg: CPU Dedicated Register, in LPR, SPR.
0000 = UPSR
0001 − 0111 = (Reserved)
1000 = FP
1001 = SP
1010 = SB
1011 = (Reserved)
1100 = (Reserved)
1101 = PSR
1110 = INTBASE
1111 = MOD

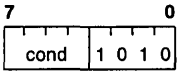
Options: in String Instructions



T = Translated
B = Backward
U/W = 00: None
01: While Match
11: Until Match

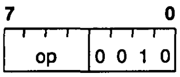
Configuration bits in SETCFG instruction:





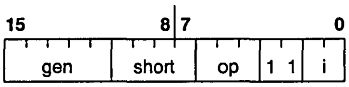
Format 0

Bcond (BR)



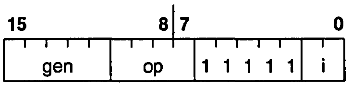
Format 1

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

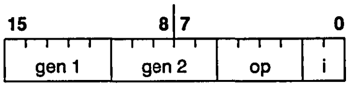
ADDQ	−000	ACB	−100
CMPQ	−001	MOVQ	−101
SPR	−010	LPR	−110
Scond	−011		



Format 3

CXPD	−0000	ADJSP	−1010
BICPSR	−0010	JSR	−1100
JUMP	−0100	CASE	−1110
BISPSR	−0110		

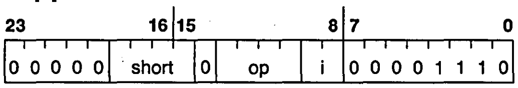
Trap (UND) on XXX1, 1000



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

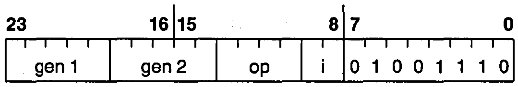
Appendix A: Instruction Formats (Continued)



Format 5

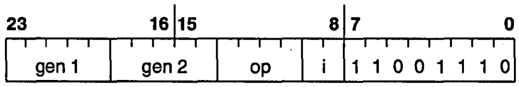
MOVS	−0000	BITWT	−1000
CMPS	−0001	TBITS	−1001
SETCFG	−0010	BBAND	−1010
SKPS	−0011	SBITPS	−1011
BBSTOD	−0100	BBFOR	−1100
EXTBLT	−0101	SBITS	−1101
BBOR	−0110	BBXOR	−1110
MOVMP	−0111		

No Operation on 1111



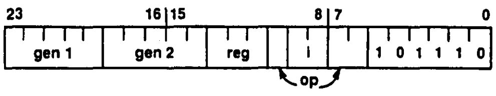
Format 6

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

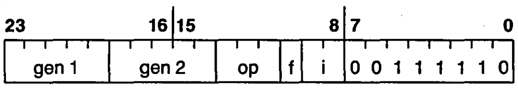


TL/EE/9424-52

Format 8

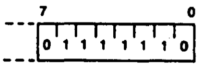
EXT	−0 00	INDEX	−1 00
CVTP	−0 01	FFS	−1 01
INS	−0 10		
CHECK	−0 11		

Trap (UND) on −1 10 and −1 11



Format 9

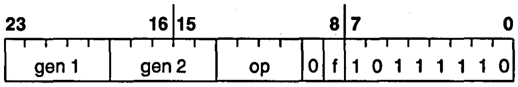
MOVif	−000	ROUND	−100
LFSR	−001	TRUNC	−101
MOVLF	−010	SFSR	−110
MOVFL	−011	FLOOR	−111



TL/EE/9424-53

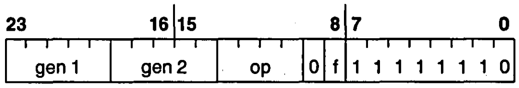
Format 10

Trap (UND) Always



Format 11

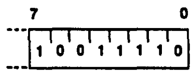
ADDf	−0000	DIVf	−1000
MOVf	−0001	(Note 1)	−1001
CMPf	−0010	Trap (UND)	−1010
(Note 3)	−0011	Trap (UND)	−1011
SUBf	−0100	MULf	−1100
NEGf	−0101	ABSf	−1101
Trap (UND)	−0110	Trap (UND)	−1110
Trap (UND)	−0111	Trap (UND)	−1111



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

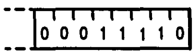
*Instructions with Format 12 are available only when the NS32381 is used.



TL/EE/9424-54

Format 13

Trap (UND) Always

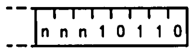


TL/EE/9424-55

Appendix A: Instruction Formats (Continued)

Format 14

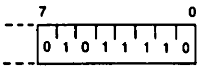
Trap (UND) Always



TL/EE/9424-56

Format 15

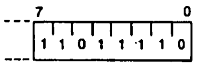
Trap (UND) Always



TL/EE/9424-57

Format 16

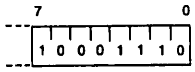
Trap (UND) Always



TL/EE/9424-58

Format 17

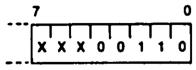
Trap (UND) Always



TL/EE/9424-59

Format 18

Trap (UND) Always

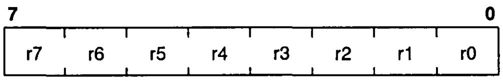


TL/EE/9424-60

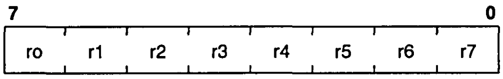
Format 19

Trap (UND) Always

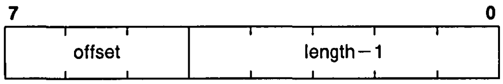
Implied Immediate Encodings:



Register Mask, appended to SAVE, ENTER



Register Mask, appended to RESTORE, EXIT



Offset/Length Modifier appended to INSS, EXTS

Note 1: Opcode not defined; CPU treats like MOVf. First operand has access class of read; second operand has access class of write; f-field selects 32-bit or 64-bit data.

Note 2: Opcode not defined; CPU treats like ADDf. First operand has access class of read; second operand has access class of read-modify-write. f-field selects 32-bit or 64-bit data.

Note 3: Opcode not defined; CPU treats like CMPf. First operand has access class of read; second operand has access class of read. f-field selects 32-bit or 64-bit data.

Section 3

Slave Processors

Section 3 Contents

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.

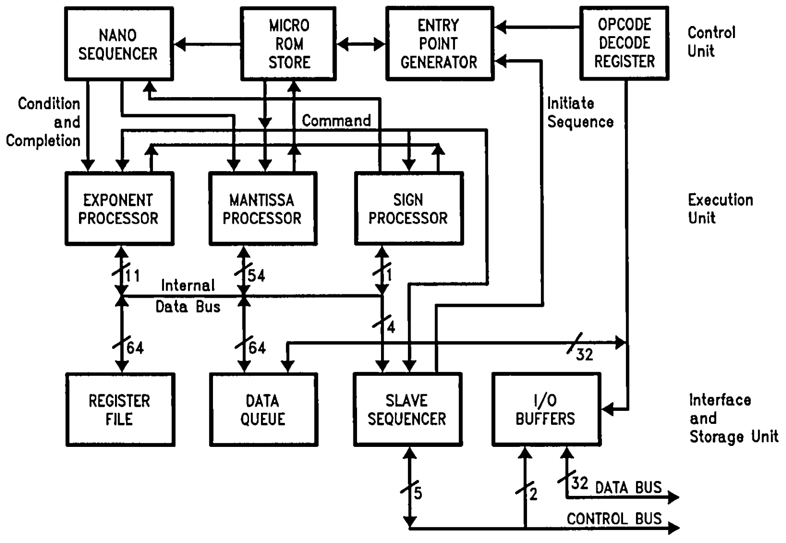
The CPU and NS32381 FPU form a tightly coupled computer cluster, which appears to the user as a single processing unit. The CPU and FPU communication is handled automatically, and is user transparent.

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	$1.5 \times 2^{-1} = 0.75$
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 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.

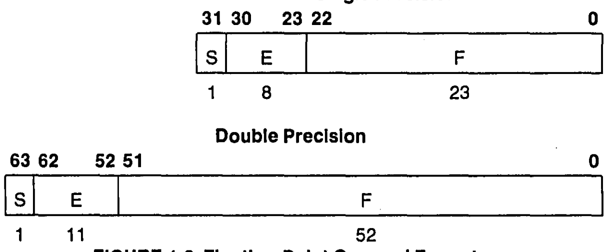


FIGURE 1-2. Floating-Point Operand Formats

1.0 Product Introduction (Continued)

TABLE 1-3. Normalized Number Ranges

	Single Precision	Double Precision
Most Positive	$2^{127} \times (2 - 2^{-23})$	$2^{1023} \times (2 - 2^{-52})$
	$= 3.40282346 \times 10^{38}$	$= 1.7976931348623157 \times 10^{308}$
Least Positive	2^{-126}	2^{-1022}
	$= 1.17549436 \times 10^{-38}$	$= 2.2250738585072014 \times 10^{-308}$
Least Negative	$-(2^{-126})$	$-(2^{-1022})$
	$= -1.17549436 \times 10^{-38}$	$= -2.2250738585072014 \times 10^{-308}$
Most Negative	$-2^{127} \times (2 - 2^{-23})$	$-2^{1023} \times (2 - 2^{-52})$
	$= -3.40282346 \times 10^{38}$	$= -1.7976931348623157 \times 10^{308}$

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.

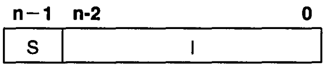


FIGURE 1-3. Integer Format

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.

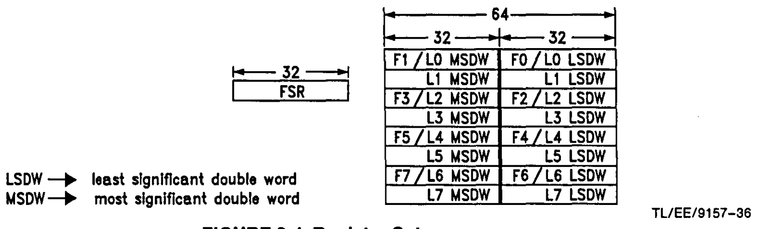


FIGURE 2-1. Register Set

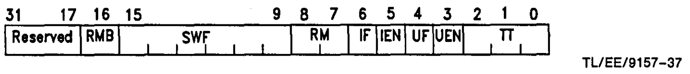


FIGURE 2-2. The Floating-Point Status Register

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)

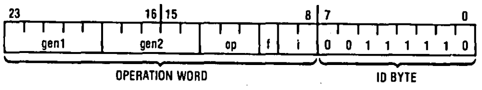
2.2 INSTRUCTION SET

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.

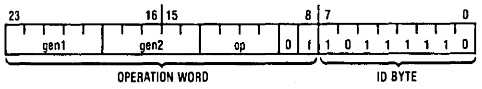
Certain notations in the following instruction description tables serve to relate the assembly language form of each instruction to its binary format in *Figure 2-3*.

Format 9



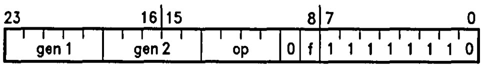
TL/EE/9157-5

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 $\overline{\text{SPC}}$ line for one clock cycle, pulsing the $\overline{\text{SDN332}}$ line for two and a half clock cycles and pulsing the $\overline{\text{FSSR}}$ line for one clock cycle. (The user will connect the correct lines according to the CPU being used).

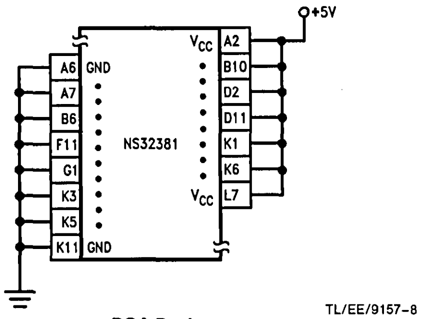
In addition, the FPU sets the Q bit in the status word register. The CPU responds by reading the status word register (refer to Section 3.6.1 for its format) while applying status h'E (transferring status word) on the status lines. A trapped instruction returns no result (even if the destination is FPU 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

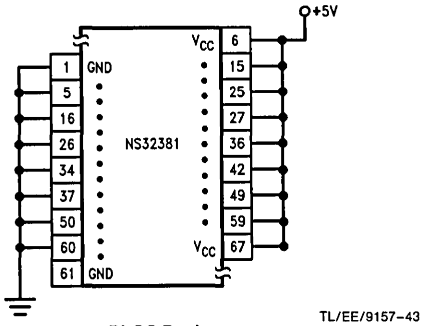
3.1 POWER AND GROUNDING

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.*



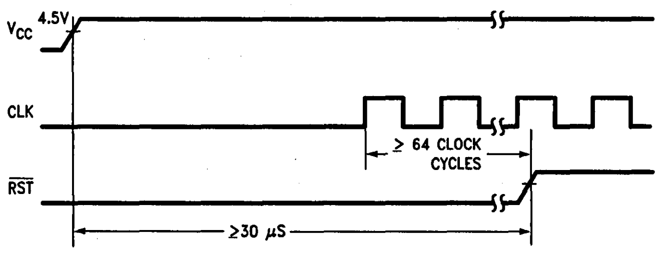
PGA Package



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.

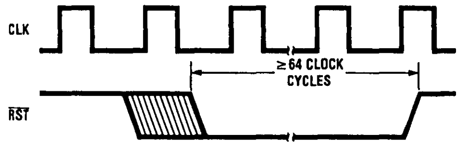
3.3 CLOCKING

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.

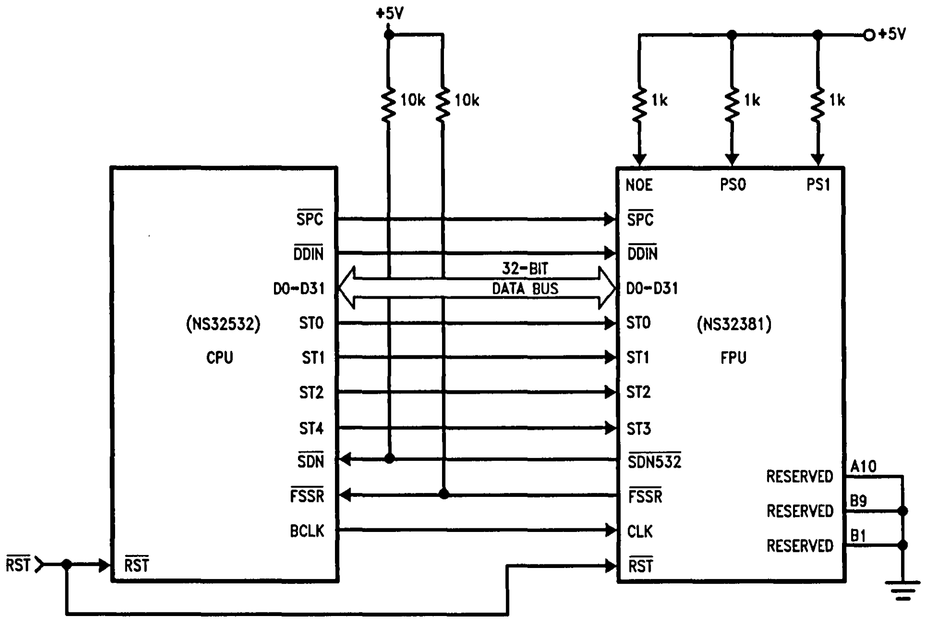
The direction of the transfer and the role of the bidirectional $\overline{SPC}$ line are determined by the instruction protocol being performed. $\overline{SPC}$ is always driven by the CPU during slave processor bus cycles. Protocol sequences for each instruction are given in Section 3.6.

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.

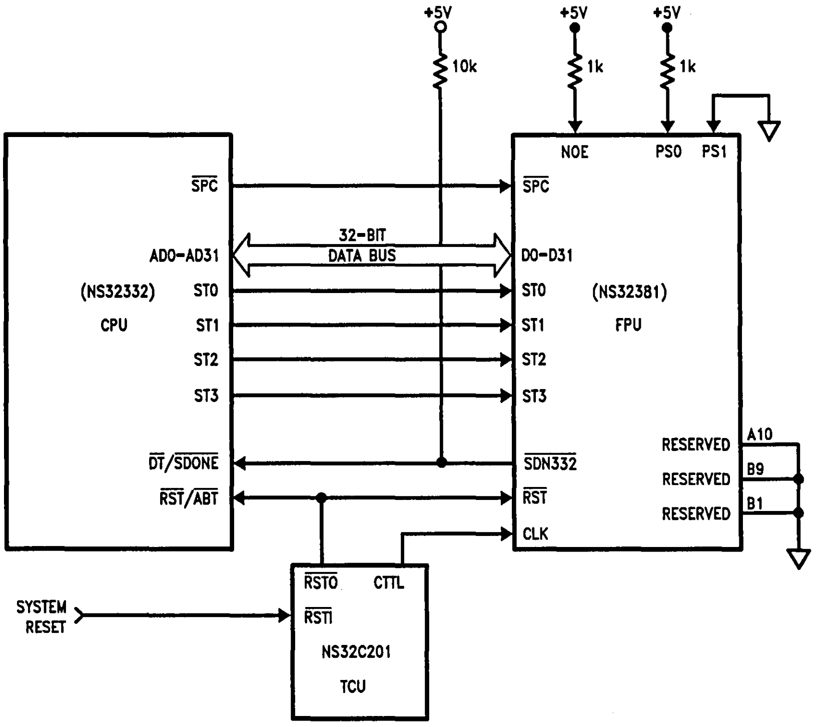
For the 32-Bit Slave Protocol a 4-byte operand is transferred on the entire data bus in a single bus cycle and an 8-byte operand is transferred in two consecutive bus cycles with the most significant byte transferred on data bits (D0–D7). The complete operand transfer of bytes B0–B7 where B0 is the least 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

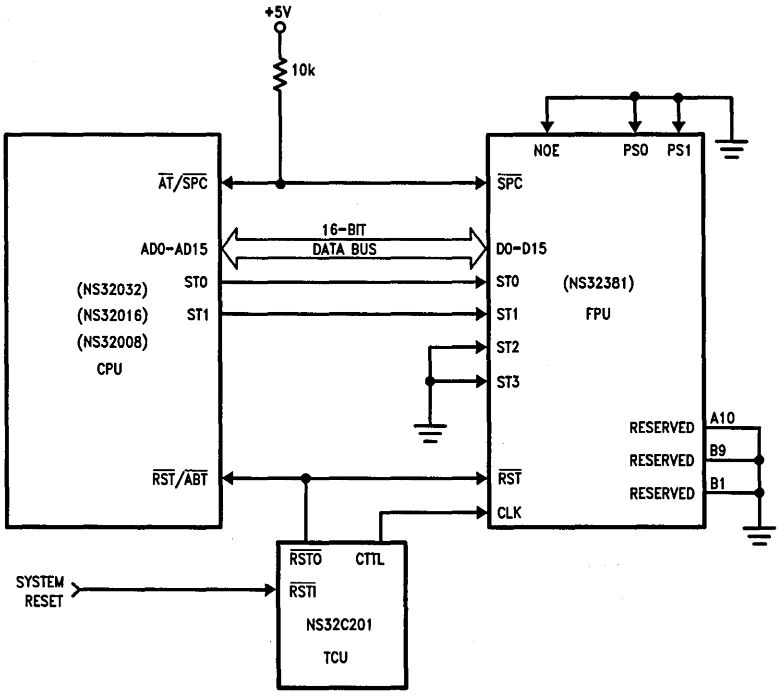
FIGURE 3-4a. System Connection Diagram with the NS32532 CPU



TL/EE/9157–39

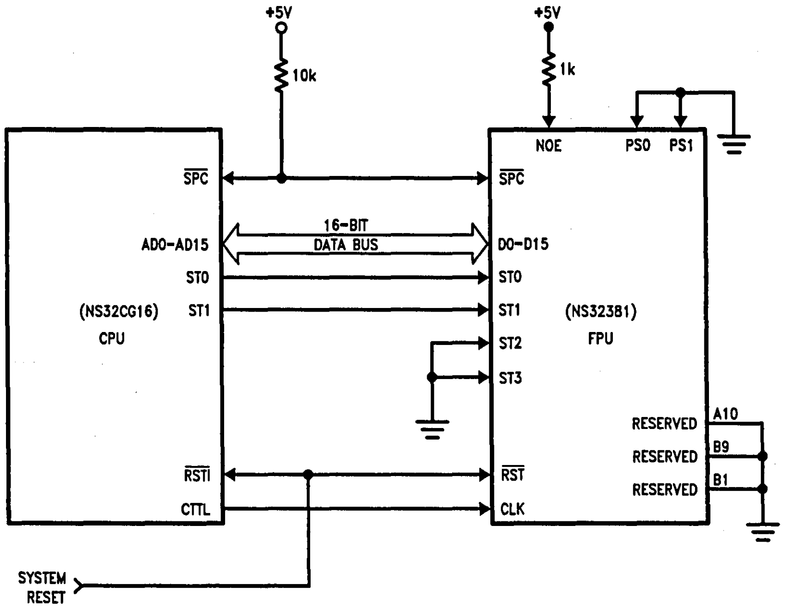
FIGURE 3-4b. System Connection Diagram with the NS32332 CPU

3.0 Functional Description (Continued)



TL/EE/9157-40

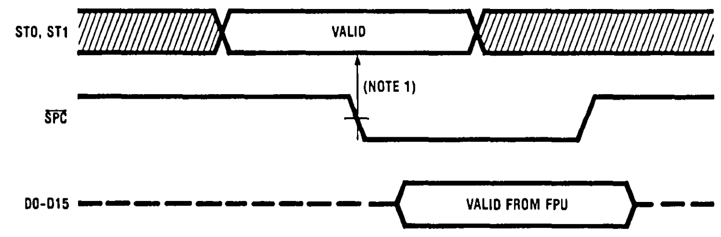
FIGURE 3-4c. System Connection Diagram with the NS32008, NS32016 or NS32032 CPU



TL/EE/9157-41

FIGURE 3-4d. System Connection Diagram with the NS32CG16 CPU

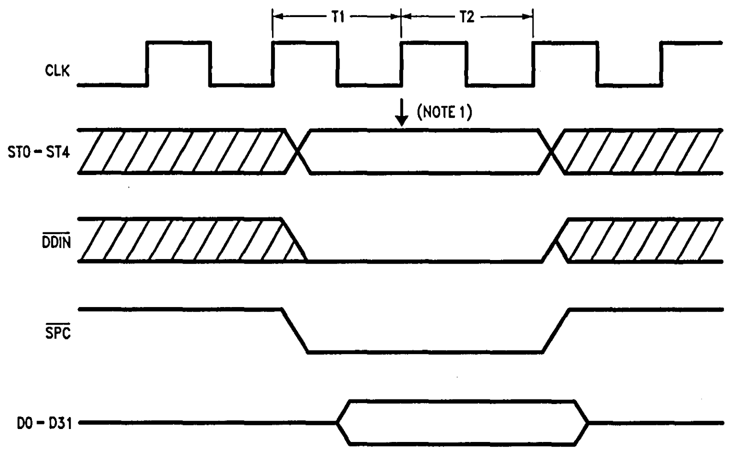
3.0 Functional Description (Continued)



TL/EE/9157-12

Note 1: FPU samples CPU status here.

FIGURE 3-5. Slave Processor Read Cycle (NS32008, NS32016, NS32032 and NS32332 CPUs)

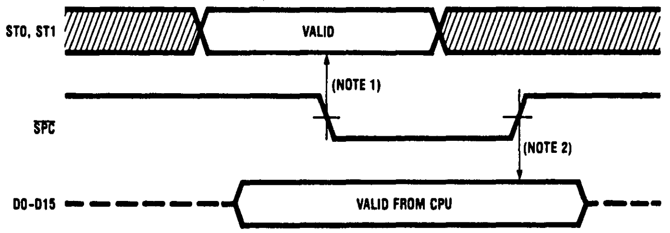


TL/EE/9157-13

Note 1: FPU samples CPU status here.

FIGURE 3-6. Slave Processor Read Cycle (NS32532 CPU)

3.0 Functional Description (Continued)

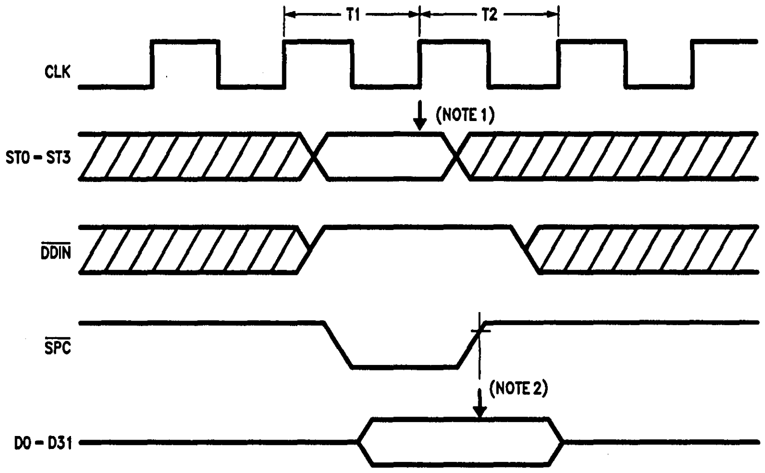


TL/EE/9157-14

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)

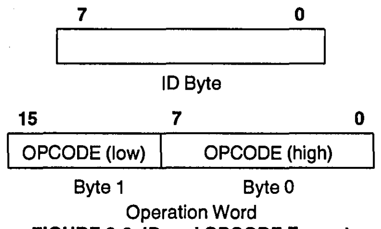


FIGURE 3-9. ID and OPCODE Format 16-Bit Slave Protocol

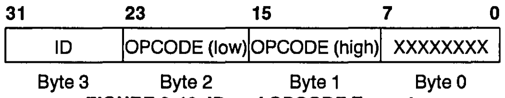


FIGURE 3-10. ID and OPCODE Format 32-Bit Slave Protocol

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.

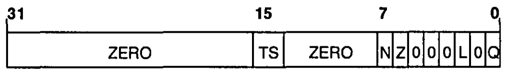
For the 32-bit Slave Protocol, upon completion of the instruction, the FPU will signal the CPU by pulsing either $\overline{SDNXXX}$ or $\overline{FSSR}$ (Force Slave Status Read).

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.

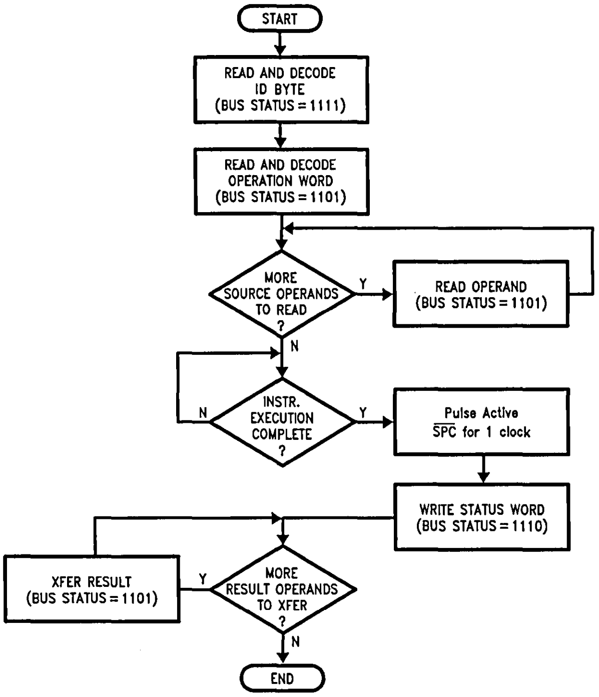
In all cases for both the 16-Bit and 32-Bit Slave Protocols the CPU will use $\overline{SPC}$ to read the Status Word from the FPU, while applying status code (1110). This word has the format shown in *Figure 3-11*. If the Q bit ("Quit", Bit 0) is set, this indicates that an error (TRAP) has been detected 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).



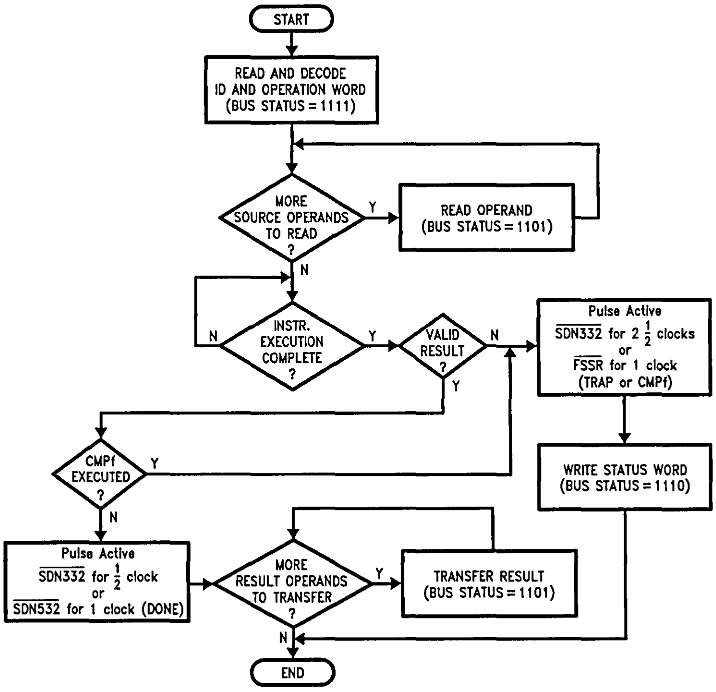
Bit		Description
(0)	Q:	Set to "1" if an FPU TRAP (error) occurred. Cleared to '0" by a valid CMPf.
(2)	L:	Cleared to "0" by the FPU.
(6)	Z:	Set to "1" if the second operand is equal to the first operand. Otherwise it is cleared to "0".
(7)	N:	Set to "1" if the second operand is less than the first operand. 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


START
READ AND DECODE ID BYTE (BUS STATUS = 1111)
READ AND DECODE OPERATION WORD (BUS STATUS = 1101)
MORE SOURCE OPERANDS TO READ ?
Y
READ OPERAND (BUS STATUS = 1101)
N
INSTR. EXECUTION COMPLETE ?
N
Y
Pulse Active $\overline{SPC}$ for 1 clock
WRITE STATUS WORD (BUS STATUS = 1110)
XFER RESULT (BUS STATUS = 1101)
Y
MORE RESULT OPERANDS TO XFER ?
N
END

TL/EE/9157-16

FIGURE 3-12. 16-Bit General Slave Instruction Protocol: FPU Actions


START
READ AND DECODE ID AND OPERATION WORD (BUS STATUS = 1111)
MORE SOURCE OPERANDS TO READ ?
Y
READ OPERAND (BUS STATUS = 1101)
N
INSTR. EXECUTION COMPLETE ?
N
Y
VALID RESULT ?
N
Y
Pulse Active $\overline{SDN332}$ for $2\frac{1}{2}$ clocks or $\overline{FSSR}$ for 1 clock (TRAP or CMPf)
WRITE STATUS WORD (BUS STATUS = 1110)
CMPf EXECUTED ?
Y
N
Pulse Active $\overline{SDN332}$ for $\frac{1}{2}$ clock or $\overline{SDN532}$ for 1 clock (DONE)
MORE RESULT OPERANDS TO TRANSFER ?
Y
TRANSFER RESULT (BUS STATUS = 1101)
N
END

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.

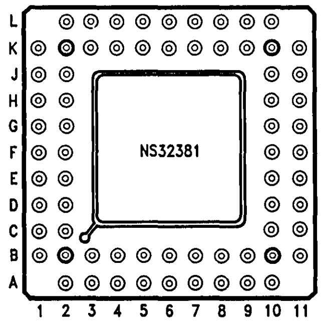
4.1.4 Input/Output Signals

*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
$\overline{\text{RST}}$	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
$\overline{\text{SDN332}}$	H11
D7	J1
D23	J2
$\overline{\text{SPC}}$	J10
$\overline{\text{SDN532}}$	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
$\overline{\text{DDIN}}$	L6
V_{CC}	L7
ST2	L8
ST3	L9
$\overline{\text{FSSR}}$	L10

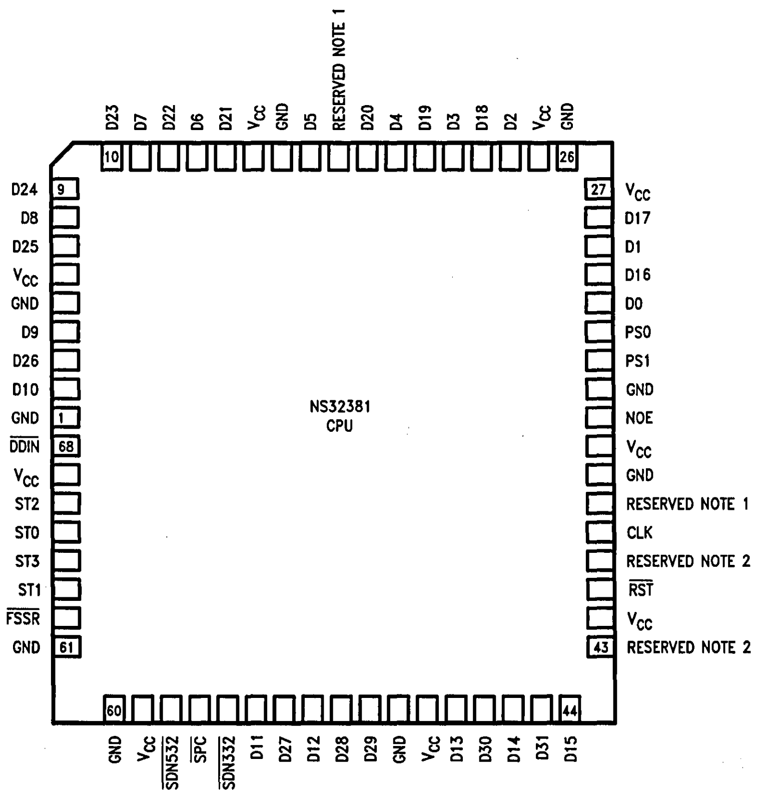
Note 1: CMOS input; never float.

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*	$0 \le V_{IN} \le V_{CC}$	−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	$0 \le V_{IN} \le V_{CC}$	−100		100	μA
I_L	Leakage Current (Output and I/O Pins in TRI-STATE®/Input Mode)	$0.4 \le V_{OUT} \le 2.4V$	−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.

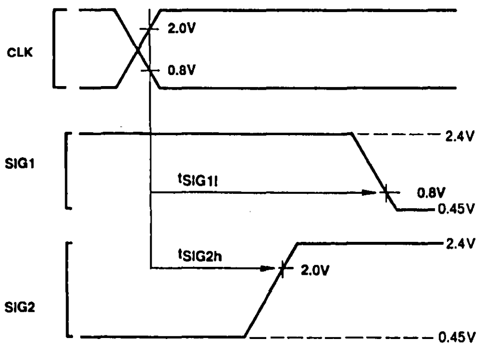


FIGURE 4-3. Timing Specification Standard (Signal Valid after Clock Edge)

ABBREVIATIONS

L.E. — Leading Edge	R.E. — Rising Edge
T.E. — Trailing Edge	F.E. — Falling Edge

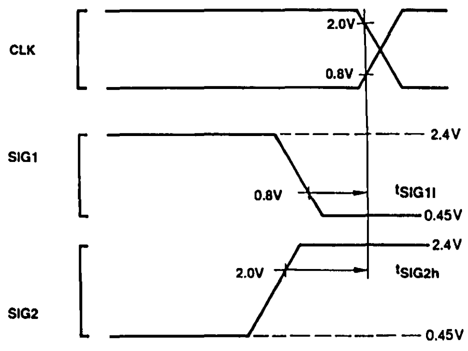


FIGURE 4-4. Timing Specification Standard (Signal Valid before Clock Edge)

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	$\overline{\text{SPC}}$ 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	$\overline{\text{SPC}}$ Output Active	After CLK R.E.		17		17		15	ns
$t_{SPCF_{ia}}$	4-18	$\overline{\text{SPC}}$ Output Inactive	After CLK R.E.		38		33		25	ns
t_{SPCF_f}(1)	4-18	$\overline{\text{SPC}}$ 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 $\overline{\text{SPC}}$ L.E.		30				18	ns
t_{Df}(1)	4-8	D0–D15 Floating	After $\overline{\text{SPC}}$ 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 $\overline{\text{SPC}}$ L.E.; 75 pF Cap. Loading		25	ns
t_{Dh}	4-10	Data Hold	After $\overline{\text{SPC}}$ T.E.	8		ns
t_{Df}(1)	4-10	Data Floating	After $\overline{\text{SPC}}$ 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_{Dv}	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_{SDa}	4-16	Slave Done Active	After CLK R.E.		35		25		20	ns
t_{SDh}	4-16	Slave Done Hold	After CLK R.E.	2	33	2	25	2	20	ns
t_{SDf}(1)	4-16	Slave Done Floating	After CLK R. E.		30		30		30	ns
t_{FSSRa}	4-17	Forced Slave Status Read Active	After CLK R.E.		35		25		20	ns
t_{FSSRh}	4-17	Forced Slave Status Read Hold	After CLK R.E.	2	33	2	25	2	20	ns
t_{FSSRf}(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_{RSTw}	4-6	Reset Pulse Width	At 0.8V (Both Edges)	64		64		64		64		t_{CLKp}
t_{RSTs}	4-7	Reset Setup Time	Before CLK R.E.	10		14		12		11		ns
t_{RSTh}	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_{Ss}	4-8	Status (ST0–ST1) Setup	Before $\overline{SPC}$ L.E.	20		20		15		ns
t_{Sh}	4-8	Status (ST0–ST1) Hold	After $\overline{SPC}$ L.E.	20		20		17		ns
t_{Ds}	4-9	Data Setup (D0–D15)	Before $\overline{SPC}$ T.E.	25		20		15		ns
t_{Dh}	4-9	Data Hold (D0–D15)	After $\overline{SPC}$ T.E.	20		20		15		ns
t_{SPCw}	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_{STs}	4-11	Status Setup	Before $\overline{SPC}$ L.E.	20		ns
t_{STh}	4-11	Status Hold	After $\overline{SPC}$ L.E.	20		ns
t_{Ds}	4-11	Data Setup	Before $\overline{SPC}$ T.E.	20		ns
t_{Dh}	4-11	Data Hold	After $\overline{SPC}$ T.E.	20		ns
t_{SPCw}	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_{STs}	4-15	Status Setup	Before CLK (T2) R.E.	25		20		20		ns
t_{STh}	4-15	Status Hold	After CLK (T2) R.E.	20		10		10		ns
t_{DDINs}	4-15	Data Direction In Setup	Before $\overline{SPC}$ L.E.	0		0		0		ns
t_{DDINh}	4-15	Data Direction In Hold	After $\overline{SPC}$ T.E.	10		10		10		ns
t_{Ds}	4-15	Data Setup	Before $\overline{SPC}$ T.E.	6		6		4		ns
t_{Dh}	4-15	Data Hold	After $\overline{SPC}$ T.E.	20		10		10		ns
t_{SPCs}	4-14, 15	$\overline{SPC}$ Setup	Before CLK R.E.	20		20		20		ns
t_{SPCh}	4-14, 15	$\overline{SPC}$ Hold	After CLK R.E.	0		0		0		ns

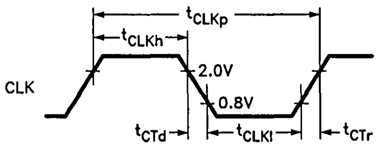
4.4.2.9 Clocking Requirements with all CPUs

Symbol	Figure	Description	Reference/ Conditions	NS32381								Units
				15		20		25		30		
				Min	Max	Min	Max	Min	Max	Min	Max	
t_{CLKh}	4-4	Clock High Time	At 2.0 V (Both Edges)	25	1000	20	1000	16	1000	13	1000	ns
t_{CLKl}	4-4	Clock Low Time	At 0.8V (Both Edges)	25	DC	20	DC	16	DC	13	DC	ns
t_{CTr}(1)	4-4	Clock Rise Time	Between 0.8V and 2.0V		7		5		4		3	ns
t_{CTd}(1)	4-4	Clock Fall Time	Between 2.0V and 0.8V		7		5		4		3	ns
t_{CLKp}	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.

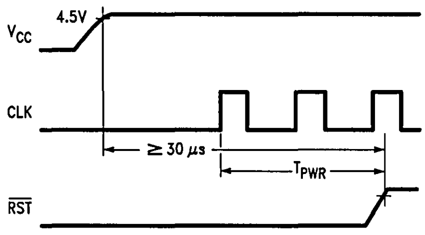
4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams



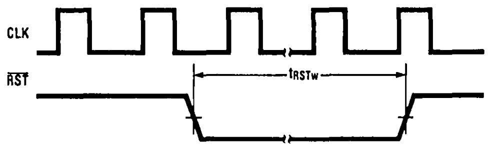
TL/EE/9157-21

FIGURE 4-5. Clock Timing



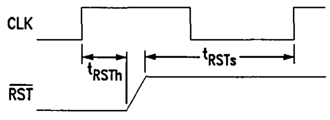
TL/EE/9157-22

FIGURE 4-6. Power-On Reset



TL/EE/9157-23

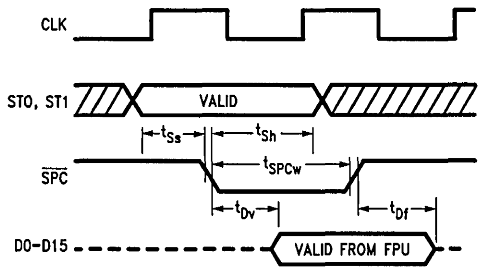
FIGURE 4-7. Non-Power-On Reset



TL/EE/9157-24

FIGURE 4-8. $\overline{RST}$ Release Timing

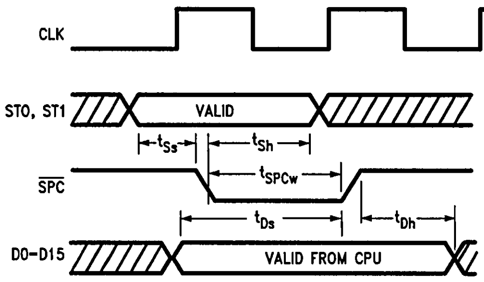
Note: The rising edge of $\overline{RST}$ must occur while CLK is high, as shown.



TL/EE/9157-25

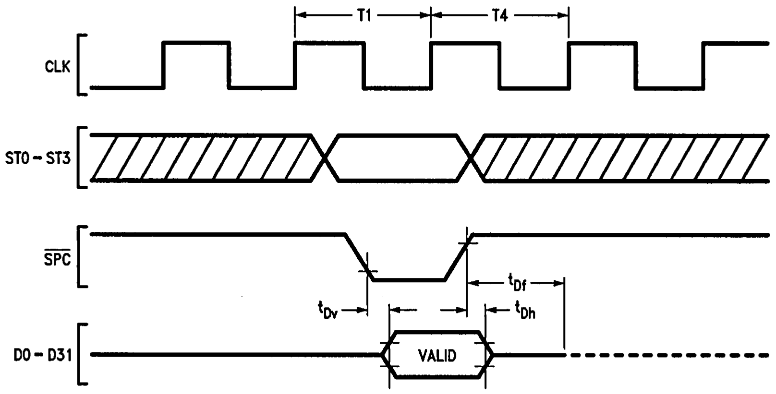
FIGURE 4-9. Read Cycle from FPU (NS32008, NS32016, NS32032 CPUs)

4.0 Device Specifications (Continued)



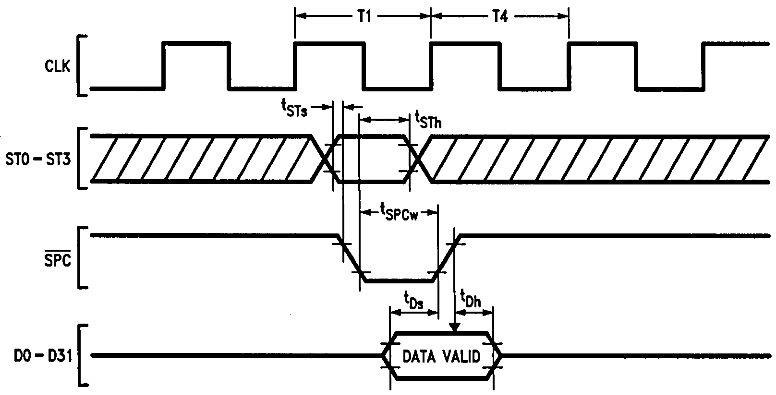
TL/EE/9157-26

FIGURE 4-10. Write Cycle to FPU (NS32008, NS32016, NS32032 CPUs)



TL/EE/9157-27

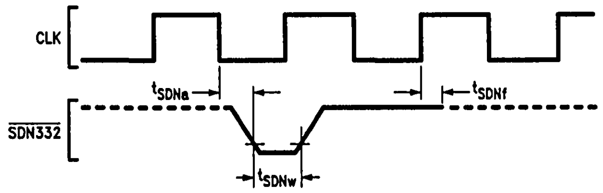
FIGURE 4-11. Read Cycle from FPU (NS32332 CPU)



TL/EE/9157-28

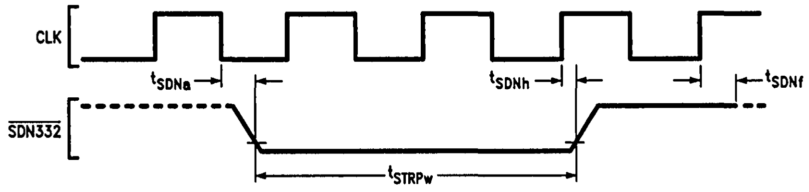
FIGURE 4-12. Write Cycle to FPU (NS32332 CPU)

4.0 Device Specifications (Continued)



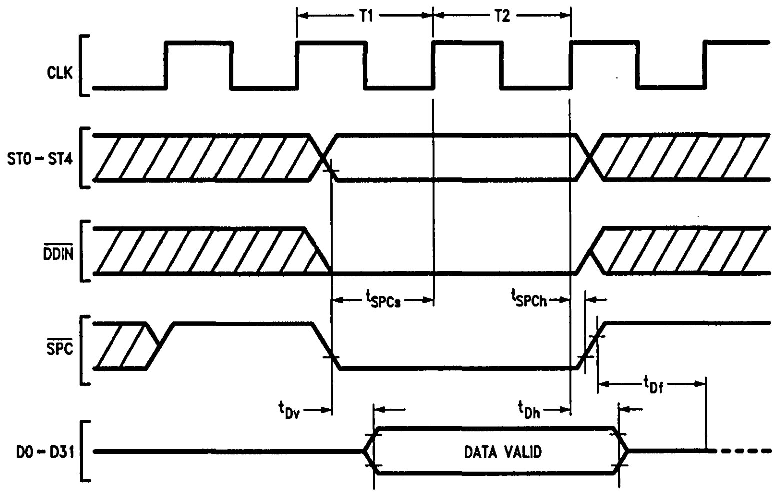
TL/EE/9157-29

FIGURE 4-13. $\overline{SDN332}$ Timing (NS32332 CPU)



TL/EE/9157-30

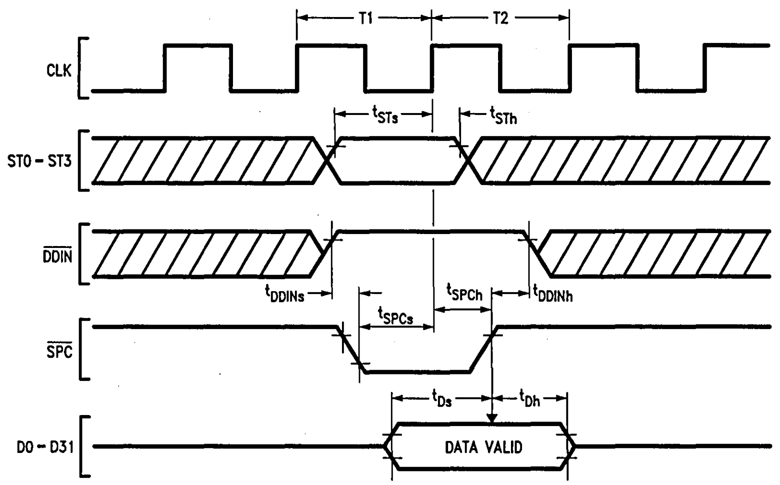
FIGURE 4-14. $\overline{SDN332}$ (TRAP) Timing (NS32332 CPU)



TL/EE/9157-31

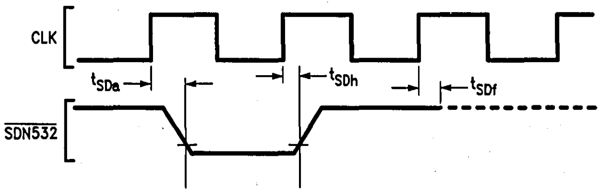
FIGURE 4-15. Read Cycle from FPU (NS32532 CPU)

4.0 Device Specifications (Continued)



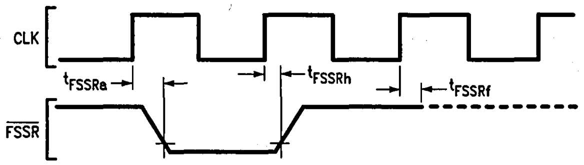
TL/EE/9157–32

FIGURE 4-16. Write Cycle to FPU (NS32532 CPU)



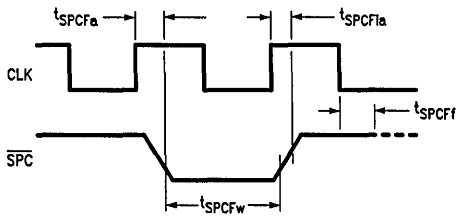
TL/EE/9157–33

FIGURE 4-17. $\overline{SDN532}$ Timing (NS32532 CPU)



TL/EE/9157–34

FIGURE 4-18. $\overline{FSSR}$ Timing (NS32532 CPU)



TL/EE/9157–35

FIGURE 4-19. $\overline{SPC}$ 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

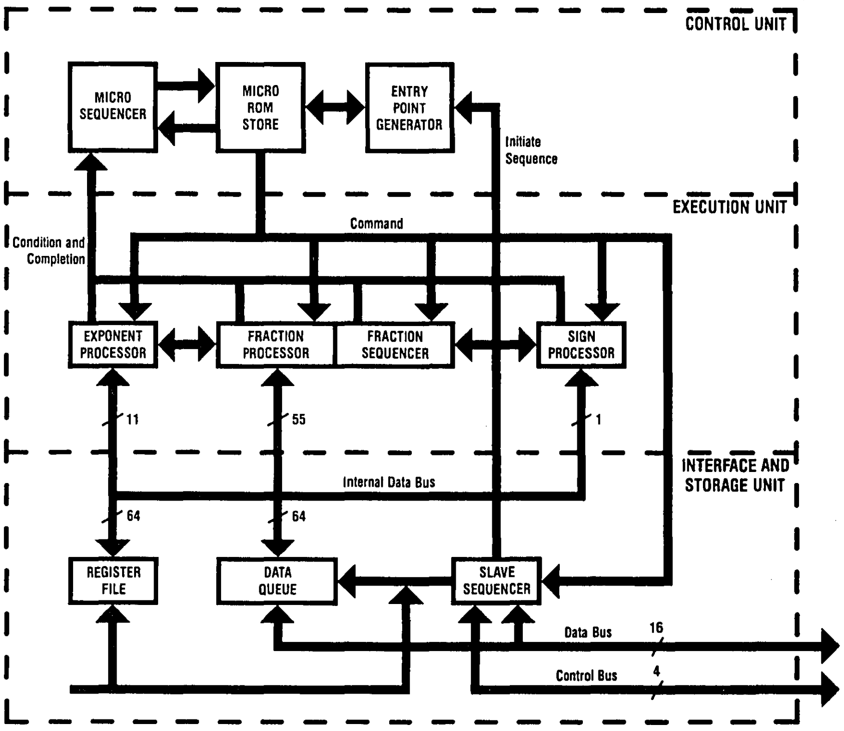
General Description

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 $1.0 \le x \le 2.0$.

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:

$$(-1)^S \times 2^{(E-Bias)} \times (1 + F)$$

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.

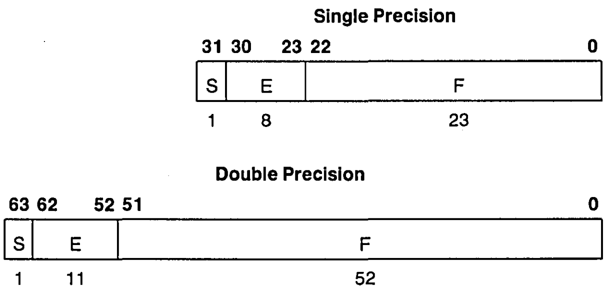


FIGURE 1-1. Floating-Point Operand Formats

1.0 Product Introduction (Continued)

TABLE 1-3. Normalized Number Ranges

	Single Precision	Double Precision
Most Positive	$2^{127} \times (2-2^{-23})$ $= 3.40282346 \times 10^{38}$	$2^{1023} \times (2-2^{-52})$ $= 1.7976931348623157 \times 10^{308}$
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.

1.1.4 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

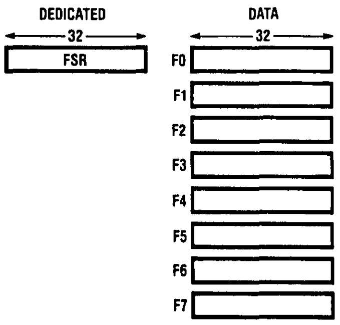
The NS32081 FPU does not directly access memory. However, it is cooperatively involved in the execution of a set of two-address instructions with its Series 32000 Family CPU. The CPU determines the representation of operands in memory.

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

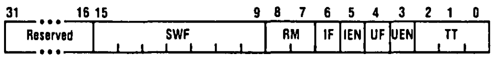
FIGURE 2-1. Register Set

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.

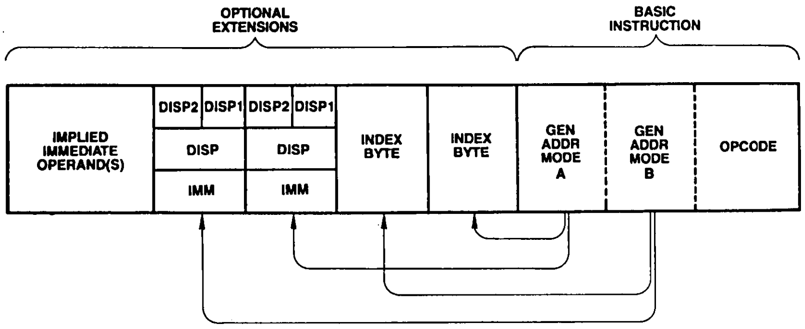
2.1.2.3 FSR Software Field (SWF)

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

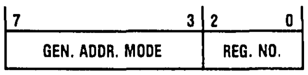
The Series 32000 Family CPUs generally access an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."

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.

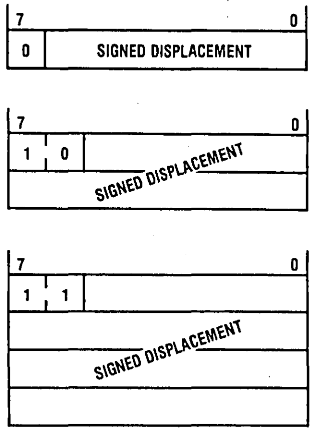
Absolute: The address of the operand is specified by a Displacement field in the instruction.

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)		

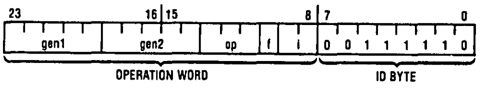
2.0 Architectural Description (Continued)

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.

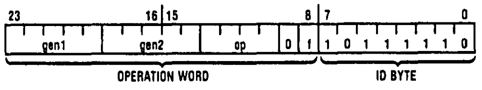
Certain notations in the following instruction description tables serve to relate the assembly language form of each instruction to its binary format in *Figure 2-6*.

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.

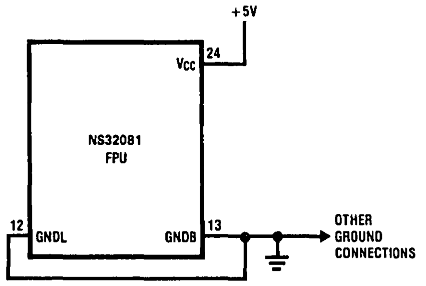
A trapped floating-point instruction returns no result, and does not affect the CPU Processor Status Register (PSR). The FPU displays the reason for the trap in the Trap Type (TT) field of the FSR (Section 2.1.2.2).

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

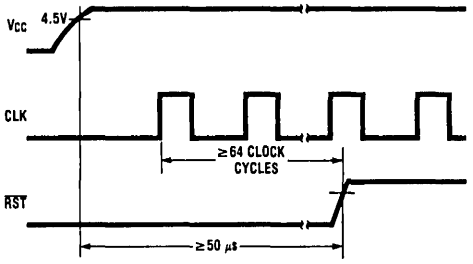
3.2 CLOCKING

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

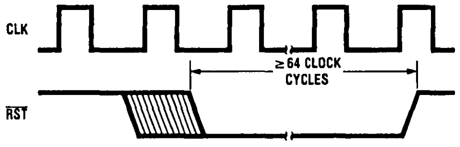
The $\overline{RST}$ pin serves as a reset for on-chip logic. The FPU may be reset at any time by pulling the $\overline{RST}$ pin low for at least 64 clock cycles. Upon detecting a reset, the FPU terminates instruction processing, resets its internal logic, and clears the FSR to all zeroes.

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

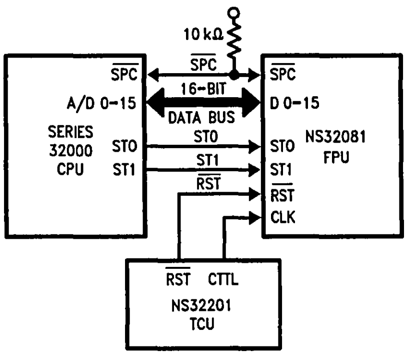


TL/EE/5234-15

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:

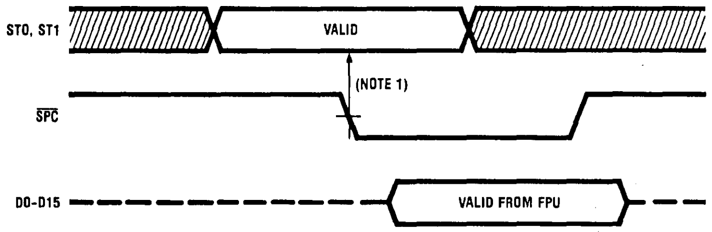
1) It identifies the instruction to the CPU as being a Slave Processor instruction.

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.

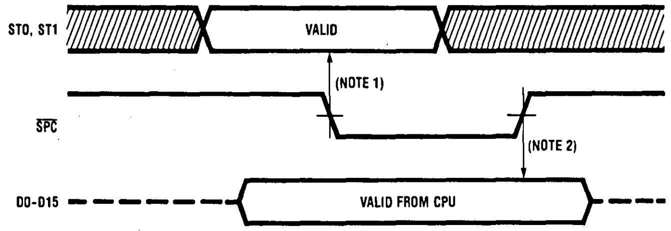
The CPU next sends the Operation Word while applying Status Code 01 (Transfer Slave Operand, Table 3-1). Upon receiving it, the FPU decodes it, and at this point both the CPU and the FPU are aware of the number of operands to be transferred and their sizes. The Operation Word is swapped on the Data Bus; that is, bits 0–7 appear on pins D8–D15, and bits 8–15 appear on pins D0–D7.



TL/EE/5234-16

Note 1: FPU samples CPU status here.

FIGURE 3-5. Slave Processor Read Cycle



TL/EE/5234-17

Note 1: FPU samples CPU status here.

Note 2: FPU samples data bus here.

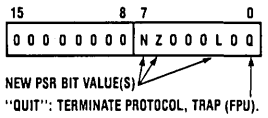
FIGURE 3-6. Slave Processor Write Cycle

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.

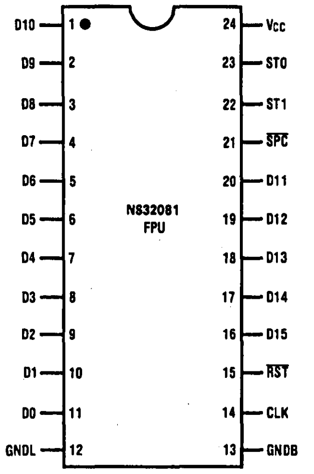
N/A = Not Applicable to this instruction.

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	$I_{OH} = -400\ \mu A$	2.4			V
V_{OL}	LOW Level Output Voltage	$I_{OL} = 4$ mA			0.45	V
I_I	Input Load Current	$0 \leq V_{IN} \leq V_{CC}$	−10.0		10.0	μA
I_L	Leakage Current Output and I/O Pins in TRI-STATE/Input Mode	$0.45 \leq V_{IN} \leq 2.4V$	−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)

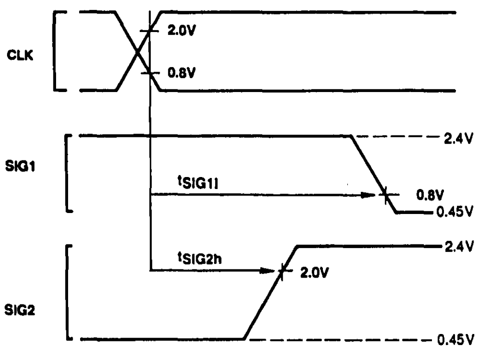
4.4 SWITCHING CHARACTERISTICS

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.

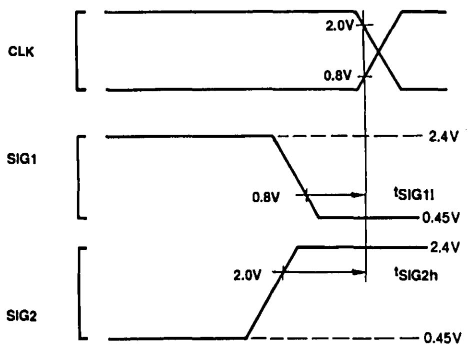
ABBREVIATIONS

L.E. — Leading Edge	R.E. — Rising Edge
T.E. — Trailing 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

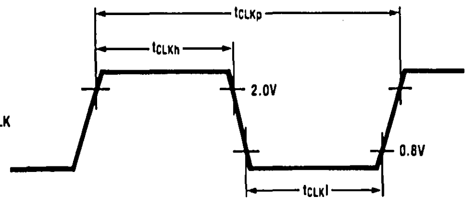
Name	Figure	Description	Reference/ Conditions	Min	Max	Min	Max	Units
t_{PWR}	4-5	Power Stable to $\overline{RST}$ R.E.	After V_{CC} Reaches 4.5V	50		50		µs
t_{RSTw}	4-6	$\overline{RST}$ Pulse Width	At 0.8V (Both Edges)	64		64		t_{CLKp}
t_{Ss}	4-7	Status (ST0–ST1) Setup	Before $\overline{SPC}$ L.E.	50		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

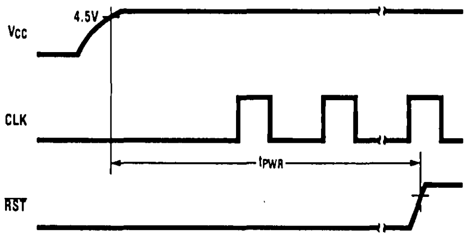
4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams



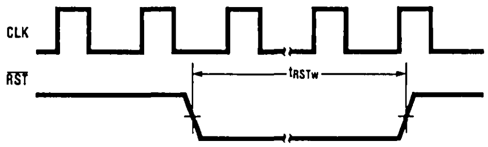
TL/EE/5234-19

FIGURE 4-4. Clock Timing



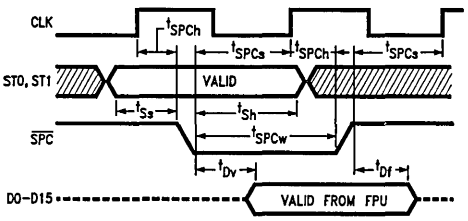
TL/EE/5234-20

FIGURE 4-5. Power-On Reset



TL/EE/5234-21

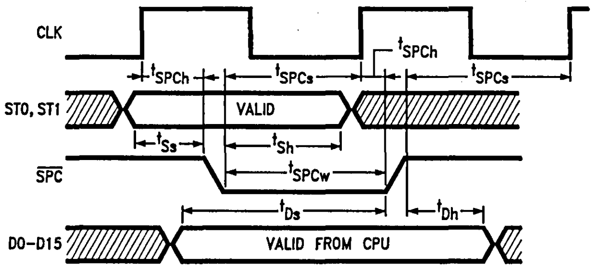
FIGURE 4-6. Non-Power-On Reset



TL/EE/5234-22

FIGURE 4-7. Read Cycle from FPU

Note: $\overline{\text{SPC}}$ pulse must be (nominally) 1 clock wide when writing into FPU.

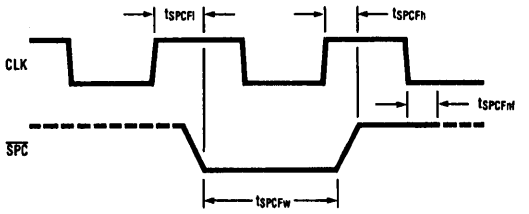


TL/EE/5234-23

FIGURE 4-8. Write Cycle to FPU

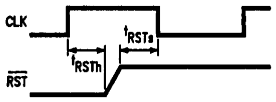
Note: $\overline{\text{SPC}}$ pulse may also be 2 clocks wide, but its edges must meet the t_{SPCs} and t_{SPCh} requirements with respect to CLK.

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.

Section 4

Peripherals

Section 4 Contents

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.

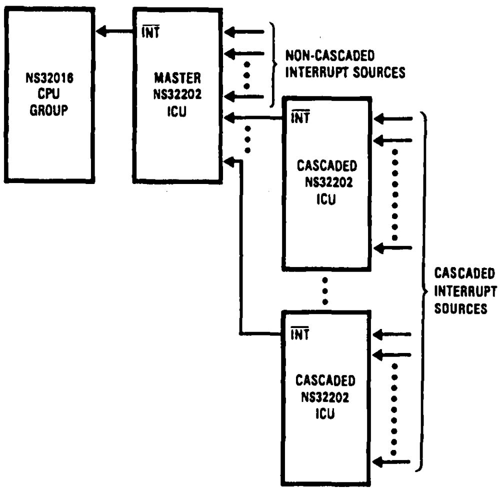
The NS32202 can operate in either of two data bus modes: 16-bit or 8-bit. In the 16-bit mode, eight hardware and eight software interrupt positions are available. In the 8-bit mode, 16 hardware interrupt positions are available, 8 of which can be used as software interrupts. In this mode, up to 16 additional 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.

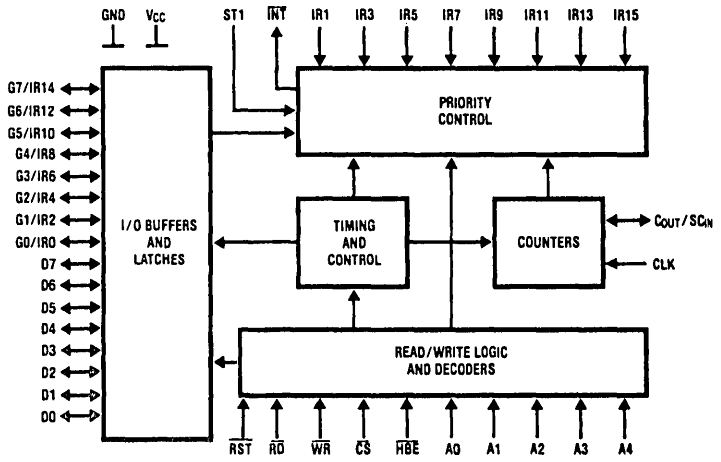
1.4 PRIORITY CONTROL

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 $\overline{\text{RST}}$ pin. At reset, most internal ICU registers are affected, and the ICU becomes inactive.

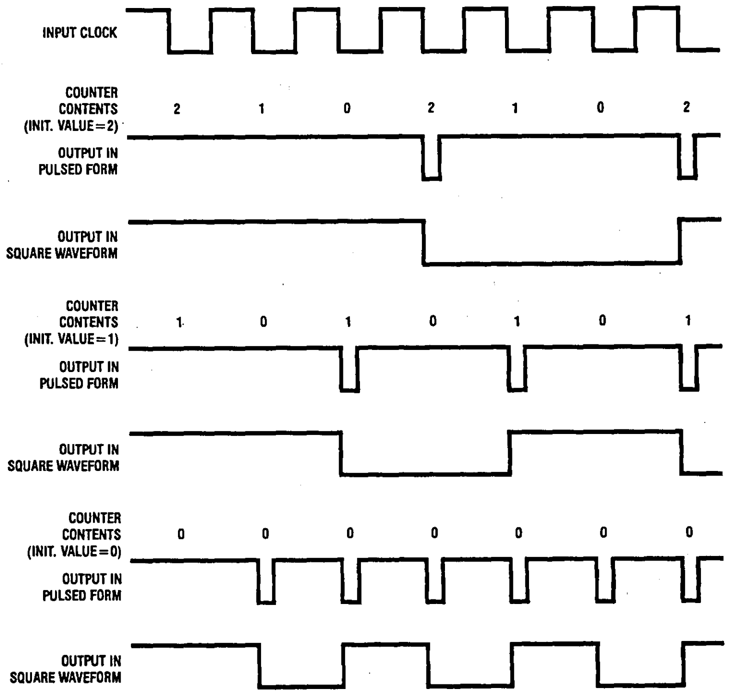
2.2 INITIALIZATION

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


INPUT CLOCK
COUNTER CONTENTS (INIT. VALUE=2)
2 1 0 2 1 0 2
OUTPUT IN PULSED FORM
OUTPUT IN SQUARE WAVEFORM
COUNTER CONTENTS (INIT. VALUE=1)
1 0 1 0 1 0 1
OUTPUT IN PULSED FORM
OUTPUT IN SQUARE WAVEFORM
COUNTER CONTENTS (INIT. VALUE=0)
0 0 0 0 0 0 0
OUTPUT IN PULSED FORM
OUTPUT IN SQUARE WAVEFORM

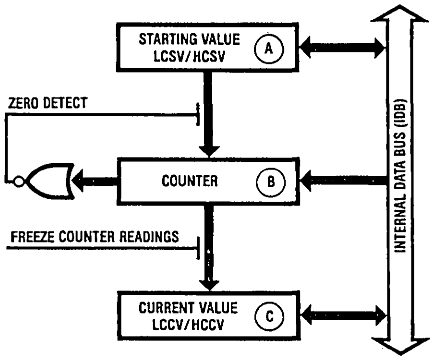
TL/EE/5117-4

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 ($\overline{\text{INT}}$) 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.

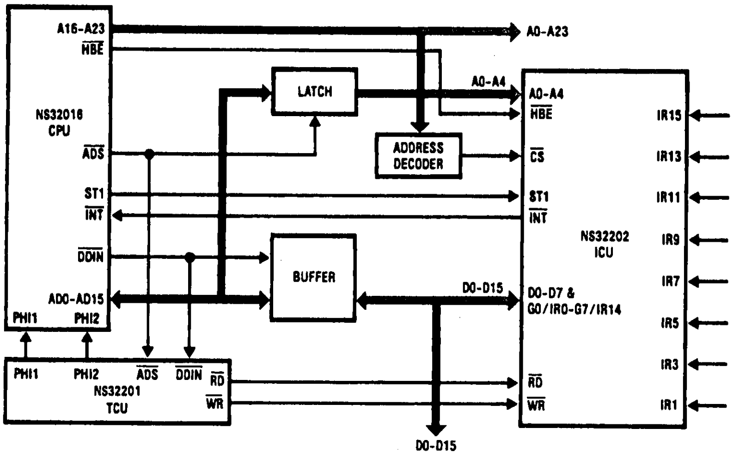


TL/EE/5117-5

BASIC OPERATIONS:	
WRITING TO LCSV/HCSV	$\boxed{A} \leftarrow$ (IDB)
READING LCSV/HCSV	$\boxed{A} \rightarrow$ (IDB)
WRITING TO LCCV/HCCV	$\boxed{B} \leftarrow$ (IDB)
(only possible when counters are halted)	$\boxed{C} \leftarrow$ (IDB)
READING LCCV/HCCV	$\boxed{C} \rightarrow$ (IDB)
(only possible when counter readings are frozen)	
COUNTER COUNTS AND READINGS ARE NOT FROZEN	$\boxed{C} \leftarrow \boxed{B}$
COUNTER RELOADS STARTING VALUE	$\boxed{B} \leftarrow \boxed{A}$
(occurs on the clock cycle following the one in which it reaches zero)	

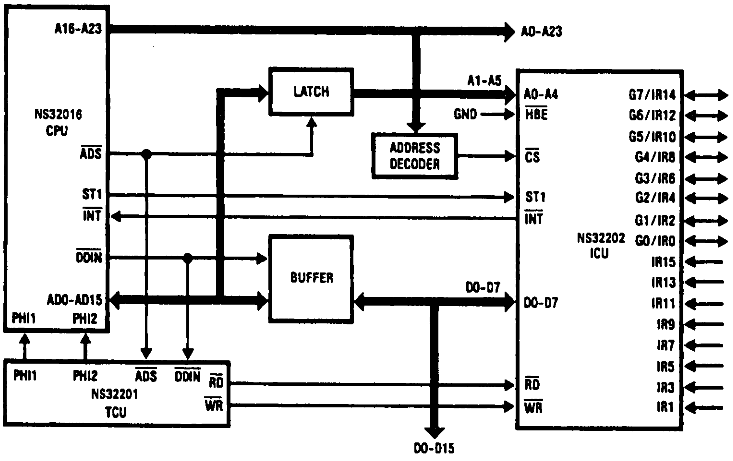
FIGURE 1-3. Counter Configuration and Basic Operations

2.0 Functional Description (Continued)



TL/EE/5117-6

FIGURE 2-1. Interrupt Control Unit Connections in 16-Bit Bus Mode

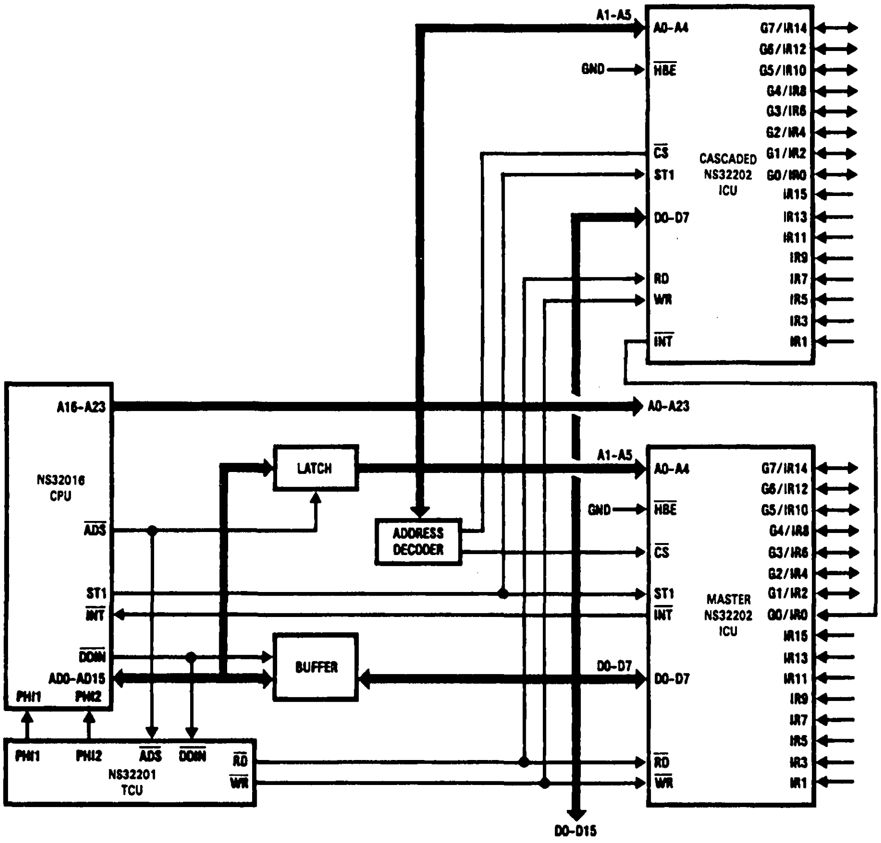


TL/EE/5117-7

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

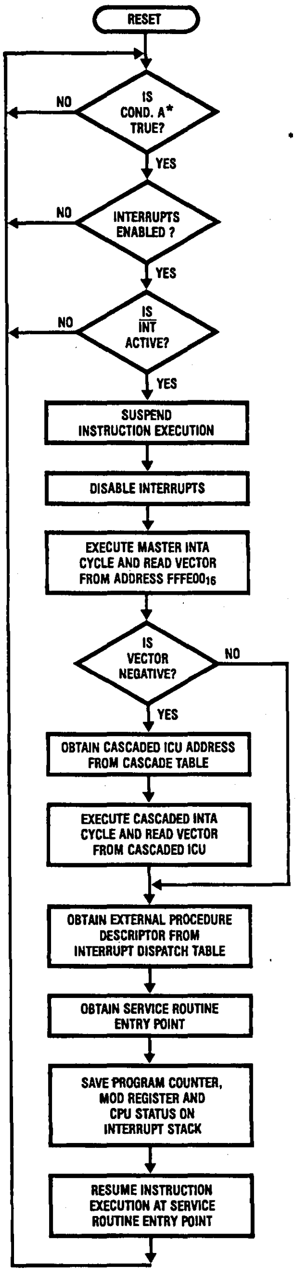
2.0 Functional Description (Continued)



TL/EE/5117-8

FIGURE 2-3. Cascaded Interrupt Control Unit Connections In 8-Bit Bus Mode

2.0 Functional Description (Continued)


RESET
IS COND. A* TRUE?
NO
YES
INTERRUPTS ENABLED ?
NO
YES
IS $\overline{\text{INT}}$ ACTIVE?
NO
YES
SUSPEND INSTRUCTION EXECUTION
DISABLE INTERRUPTS
EXECUTE MASTER INTA CYCLE AND READ VECTOR FROM ADDRESS FFFE00₁₆
IS VECTOR NEGATIVE?
NO
YES
OBTAIN CASCADED ICU ADDRESS FROM CASCADE TABLE
EXECUTE CASCADED INTA CYCLE AND READ VECTOR FROM CASCADED ICU
OBTAIN EXTERNAL PROCEDURE DESCRIPTOR FROM INTERRUPT DISPATCH TABLE
OBTAIN SERVICE ROUTINE ENTRY POINT
SAVE PROGRAM COUNTER, MOD REGISTER AND CPU STATUS ON INTERRUPT STACK
RESUME INSTRUCTION EXECUTION AT SERVICE ROUTINE ENTRY POINT

* 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.

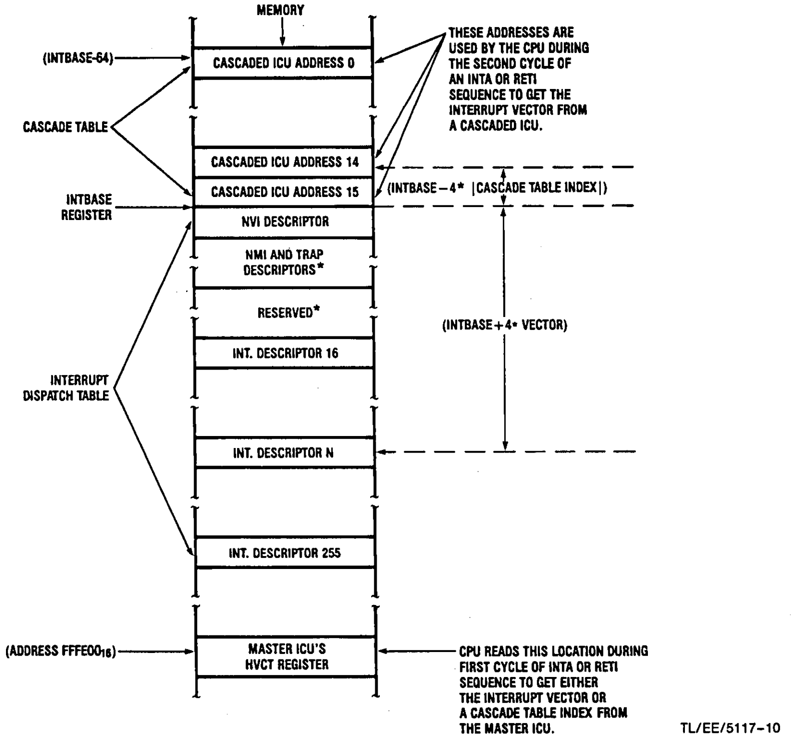
2.3.1 Non-Cascaded Operation. Whenever an interrupt request from a peripheral device is issued directly to the master ICU, a non-cascaded interrupt request to the CPU results. In a system using a single NS32202, up to 16 interrupt requests can be prioritized. Upon receipt of an interrupt request on the $\overline{INT}$ 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.

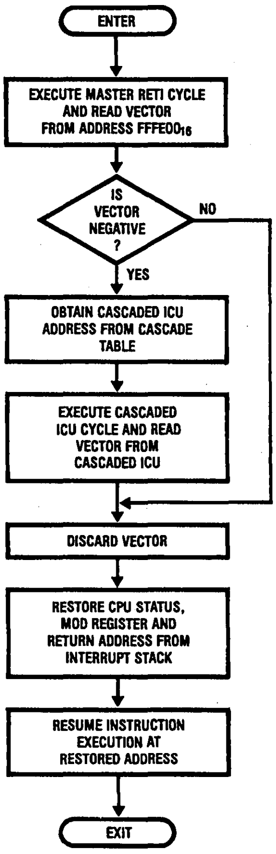
(*)Note: The CPU status corresponding to both, master interrupt acknowledge and return from interrupt bus cycles, as well as address bit A8, could be used to generate the chip select ($\overline{CS}$) signal for accessing the master ICU during one of the above cycles. In this case the master ICU can 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.


MEMORY
(INTBASE-64)
CASCADED ICU ADDRESS 0
CASCADE TABLE
CASCADED ICU ADDRESS 14
CASCADED ICU ADDRESS 15
INTBASE REGISTER
NVI DESCRIPTOR
NMI AND TRAP DESCRIPTORS*
RESERVED*
INT. DESCRIPTOR 16
INTERRUPT DISPATCH TABLE
INT. DESCRIPTOR N
INT. DESCRIPTOR 255
(ADDRESS FFFE00₁₆)
MASTER ICU'S HVCT REGISTER
THESE ADDRESSES ARE USED BY THE CPU DURING THE SECOND CYCLE OF AN INTA OR RETI SEQUENCE TO GET THE INTERRUPT VECTOR FROM A CASCADED ICU.
(INTBASE−4* |CASCADE TABLE INDEX|)
(INTBASE+4• VECTOR)
CPU READS THIS LOCATION DURING FIRST CYCLE OF INTA OR RETI SEQUENCE TO GET EITHER THE INTERRUPT VECTOR OR A CASCADE TABLE INDEX FROM THE MASTER ICU.
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)


ENTER
EXECUTE MASTER RETI CYCLE AND READ VECTOR FROM ADDRESS FFFE00₁₆
IS VECTOR NEGATIVE ?
NO
YES
OBTAIN CASCADED ICU ADDRESS FROM CASCADE TABLE
EXECUTE CASCADED ICU CYCLE AND READ VECTOR FROM CASCADED ICU
DISCARD VECTOR
RESTORE CPU STATUS, MOD REGISTER AND RETURN ADDRESS FROM INTERRUPT STACK
RESUME INSTRUCTION EXECUTION AT RESTORED ADDRESS
EXIT

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.

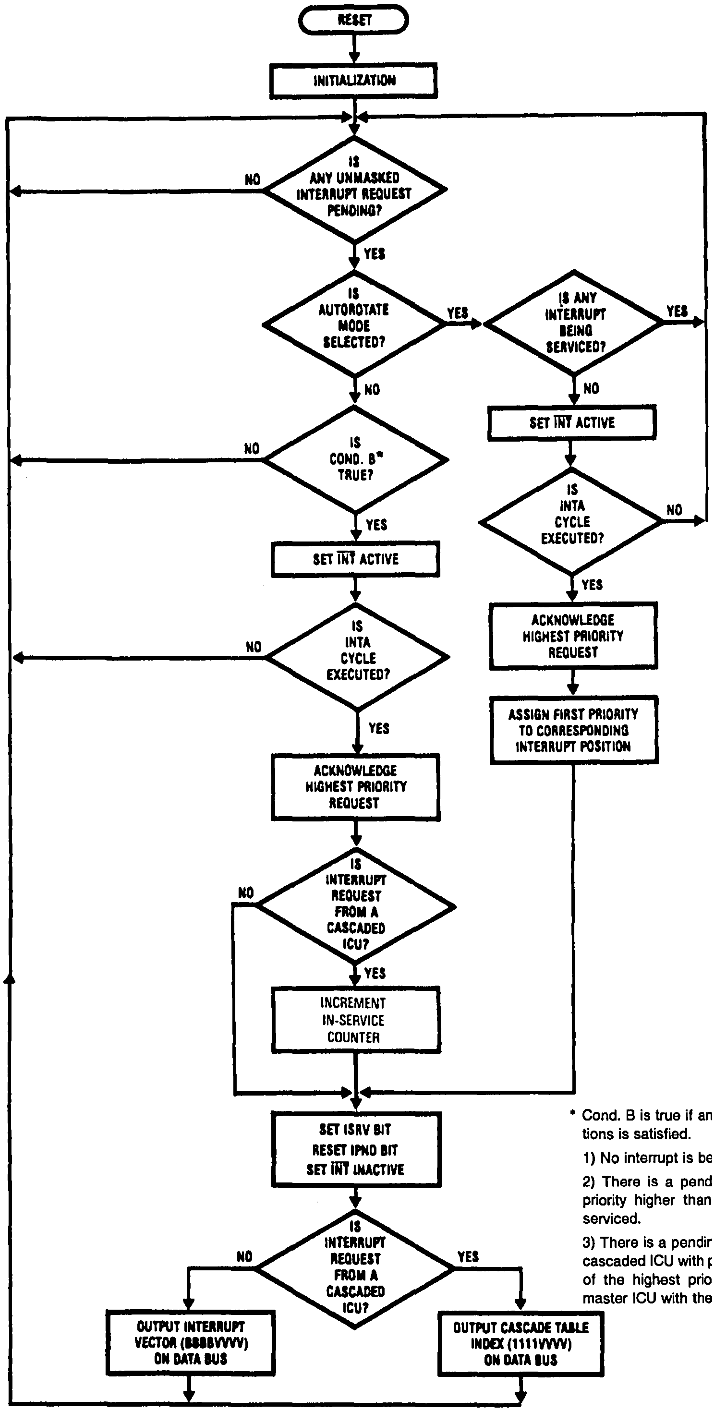
Once the ICU is initialized, it is enabled to accept interrupts. If an active interrupt is not masked, and has a higher priority than any interrupt currently being serviced, the ICU activates its Interrupt Output ($\overline{\text{INT}}$). *Figure 2-7* is a flowchart showing the ICU interrupt acknowledge sequence.

The CPU responds to the active $\overline{\text{INT}}$ 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 $\overline{\text{INT}}$ 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 $\overline{\text{INT}}$ 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)


RESET
INITIALIZATION
IS ANY UNMASKED INTERRUPT REQUEST PENDING?
NO
YES
IS AUTOROTATE MODE SELECTED?
YES
NO
IS ANY INTERRUPT BEING SERVICED?
YES
NO
SET INT ACTIVE
IS INTA CYCLE EXECUTED?
NO
YES
ACKNOWLEDGE HIGHEST PRIORITY REQUEST
ASSIGN FIRST PRIORITY TO CORRESPONDING INTERRUPT POSITION
IS COND. B* TRUE?
NO
YES
SET INT ACTIVE
IS INTA CYCLE EXECUTED?
NO
YES
ACKNOWLEDGE HIGHEST PRIORITY REQUEST
IS INTERRUPT REQUEST FROM A CASCADED ICU?
NO
YES
INCREMENT IN-SERVICE COUNTER
SET ISRV BIT
RESET IPND BIT
SET INT INACTIVE
IS INTERRUPT REQUEST FROM A CASCADED ICU?
NO
YES
OUTPUT INTERRUPT VECTOR (SSSSVVVV) ON DATA BUS
OUTPUT CASCADE TABLE INDEX (1111VVVV) ON DATA BUS

* Cond. B is true if any one of the following conditions is satisfied.

1) No interrupt is being serviced

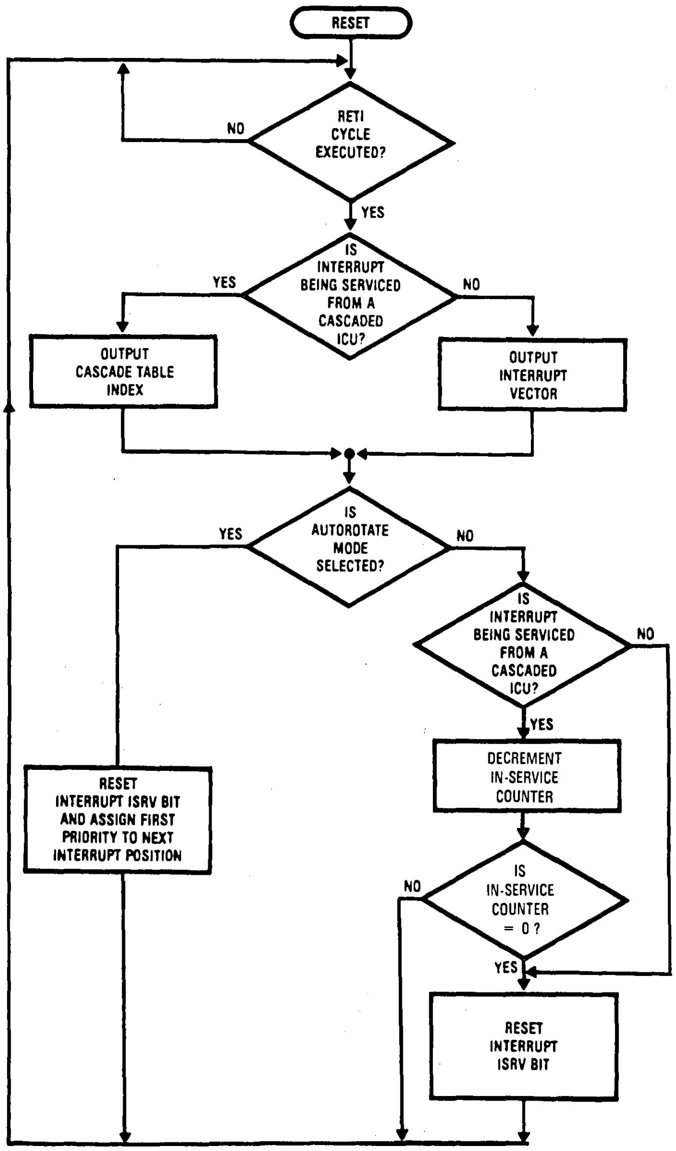
2) There is a pending unmasked interrupt with priority higher than that of the interrupt being serviced.

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)


RESET
RETI CYCLE EXECUTED?
NO
YES
IS INTERRUPT BEING SERVICED FROM A CASCADED ICU?
YES
NO
OUTPUT CASCADE TABLE INDEX
OUTPUT INTERRUPT VECTOR
IS AUTOROTATE MODE SELECTED?
YES
NO
IS INTERRUPT BEING SERVICED FROM A CASCADED ICU?
NO
YES
DECREMENT IN-SERVICE COUNTER
RESET INTERRUPT ISRV BIT AND ASSIGN FIRST PRIORITY TO NEXT INTERRUPT POSITION
IS IN-SERVICE COUNTER = 0 ?
NO
YES
RESET INTERRUPT ISRV BIT

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.

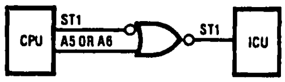
3.2 SVCT — SOFTWARE VECTOR REGISTER (R1)

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 $\overline{\text{INT}}$ 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 $\overline{\text{INT}}$ 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 GO/IRO, . . . ,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)

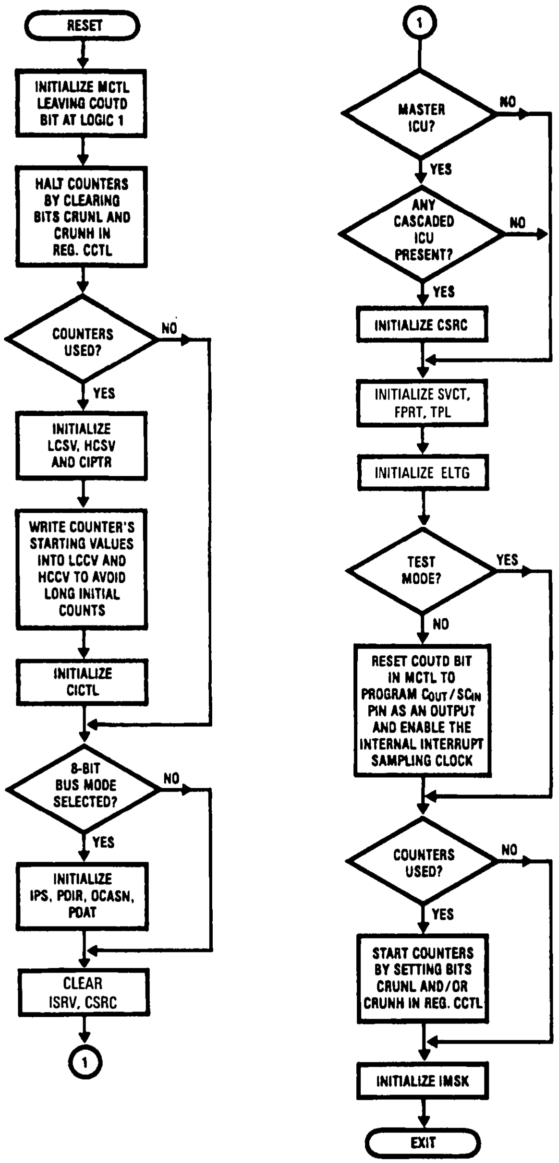
The LCSV and HCSV registers store the start values for the L-Counter and H-Counter, respectively. Each time a counter reaches zero, the start value is automatically reloaded from either LCSV or HCSV, one clock cycle after zero count is reached. Loading LCSV or HCSV from the CPU must be synchronized to 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)


RESET
INITIALIZE MCTL LEAVING COUTD BIT AT LOGIC 1
HALT COUNTERS BY CLEARING BITS CRUNL AND CRUNH IN REG. CCTL
COUNTERS USED?
NO
YES
INITIALIZE LCSV, HCSV AND CIPTR
WRITE COUNTER'S STARTING VALUES INTO LCCV AND HCCV TO AVOID LONG INITIAL COUNTS
INITIALIZE CICTL
8-BIT BUS MODE SELECTED?
NO
YES
INITIALIZE IPS, PDIR, OCASN, PDAT
CLEAR ISRV, CSRC
1
1
MASTER ICU?
NO
YES
ANY CASCADED ICU PRESENT?
NO
YES
INITIALIZE CSRC
INITIALIZE SVCT, FPRT, TPL
INITIALIZE ELTG
TEST MODE?
YES
NO
RESET COUTD BIT IN MCTL TO PROGRAM COUT/SCIN PIN AS AN OUTPUT AND ENABLE THE INTERNAL INTERRUPT SAMPLING CLOCK
COUNTERS USED?
NO
YES
START COUNTERS BY SETTING BITS CRUNL AND/OR CRUNH IN REG. CCTL
INITIALIZE IMSK
EXIT

TL/EE/5117-15

FIGURE 3-3. Recommended ICU's Initialization Sequence

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

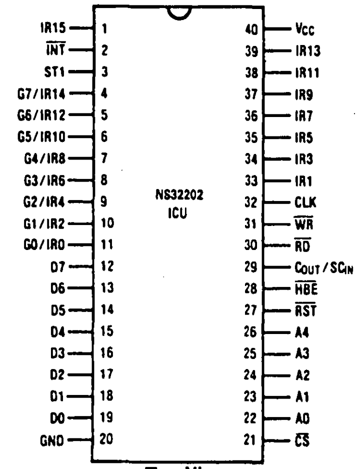
Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS

$T_A = 0°$ to 70°C, $V_{CC} = +5V \pm 5\%$, 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	$I_{OH} = -400\ \mu A$	2.4			V
I_L	Leakage Current (Output and I/O Pins in TRI-STATE/Input mode)	$0.4 \leq V_{IN} \leq V_{CC}$	−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



Top View

TL/EE/5117-3

Order Number NS32202D-6, NS32202D-10

See NS Package Number D40C

FIGURE 4-1

4.0 Device Specifications (Continued)

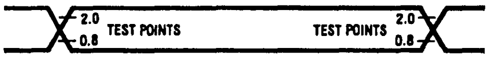
4.4 SWITCHING CHARACTERISTICS

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

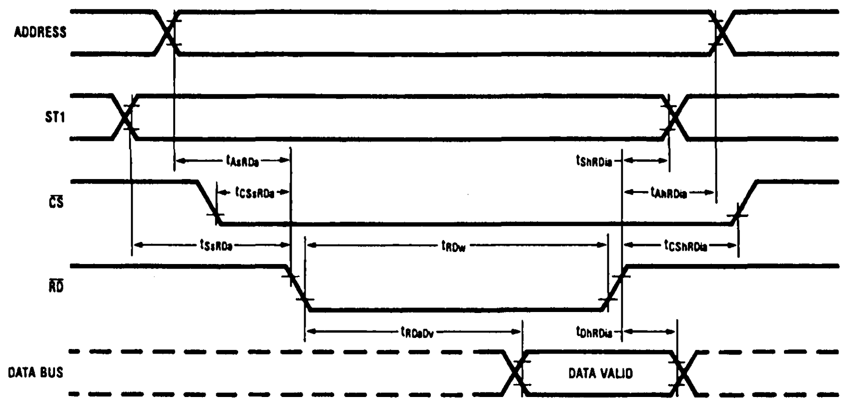
Symbol	Figure	Description	Reference/Conditions	NS32202-10		Units
				Min	Max	
READ CYCLE						
t_{AhRDia}	4-3	Address Hold Time	After $\overline{RD}$ T.E.	10		ns
t_{AsRDa}	4-3	Address Setup Time	Before $\overline{RD}$ L.E.	35		ns
$t_{CShRDia}$	4-3	$\overline{CS}$ Hold Time	After $\overline{RD}$ T.E.	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_{IRld}	4-5	$\overline{\text{INT}}$ 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}		$\overline{\text{RST}}$ Pulse Width	At 0.8V (Both Edges)	400		ns

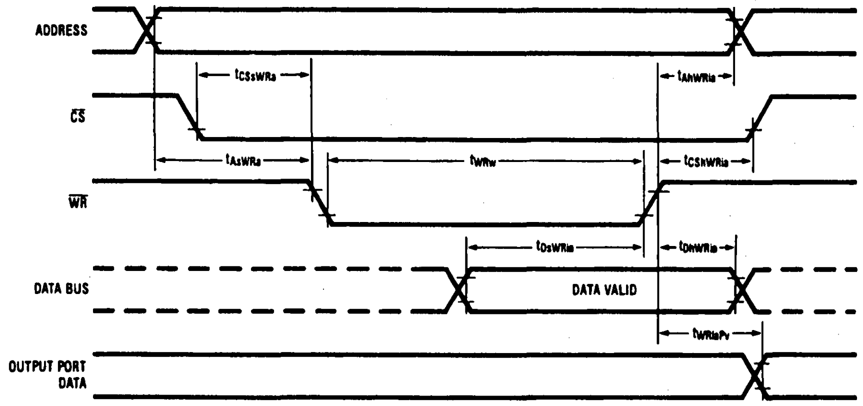
4.4.1.2 Timing Diagrams



TL/EE/5117-17

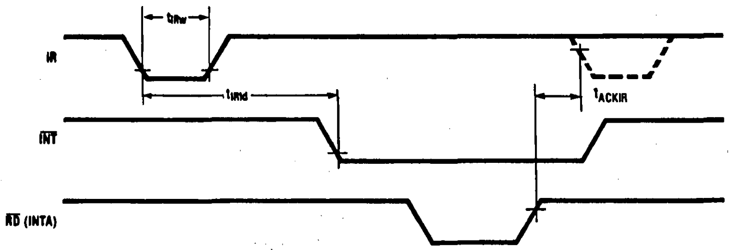
FIGURE 4-3. READ/INTA Cycle

4.0 Device Specifications (Continued)



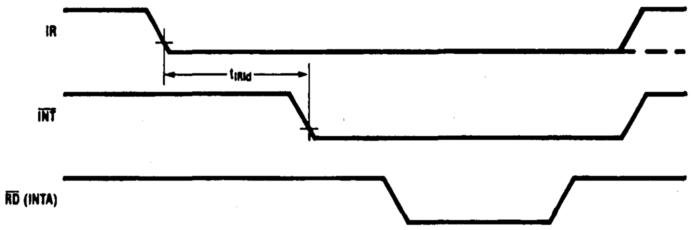
TL/EE/5117-18

FIGURE 4-4. Write Cycle



TL/EE/5117-19

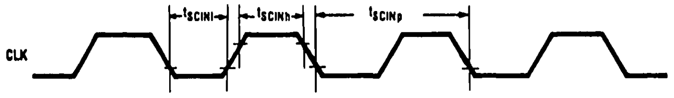
FIGURE 4-5. Interrupt Timing in Edge Triggering Mode



TL/EE/5117-20

FIGURE 4-6. Interrupt Timing in Level Triggering Mode

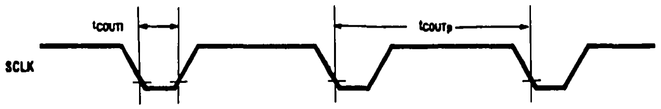
4.0 Device Specifications (Continued)



TL/EE/5117-21

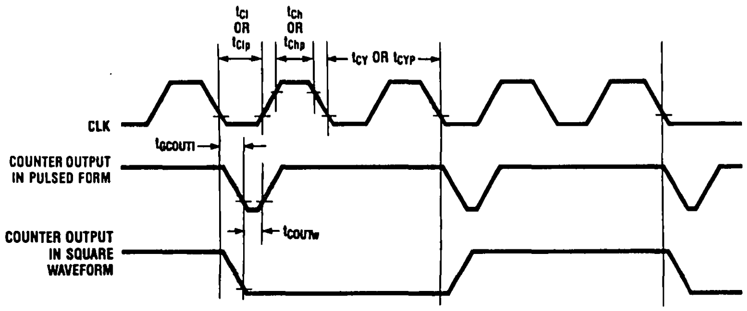
Note: Interrupts are sampled on the rising edge of CLK.

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

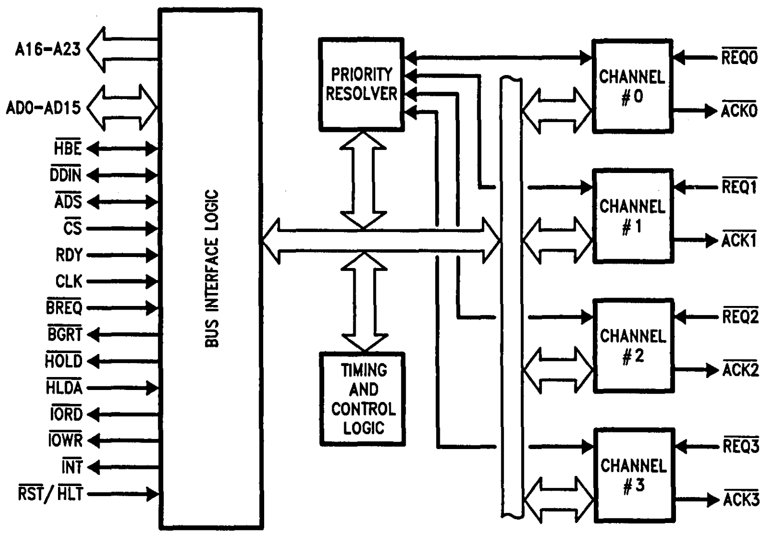
General Description

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.

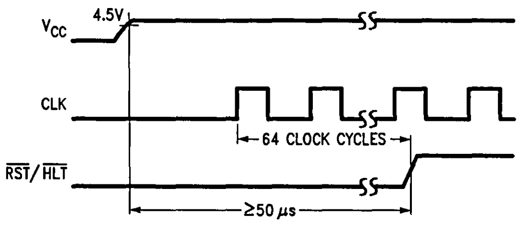
Timing and Control Logic. This block generates all the sequencing and control signals necessary for the operation of the DMAC.

Priority Resolver. This block resolves contentions among channels requesting service simultaneously.

2.0 Functional Description

2.1 RESETTING

The $\overline{\text{RST}}/\overline{\text{HLT}}$ line serves both as a reset input for the on-chip logic and as a DMAC HALT input. Resetting is accomplished by pulling $\overline{\text{RST}}/\overline{\text{HLT}}$ 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, $\overline{\text{RST}}/\overline{\text{HLT}}$ 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 $\overline{\text{RST}}/\overline{\text{HLT}}$ 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)

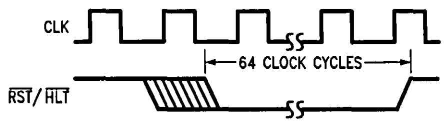


FIGURE 2-2. General Reset Timing

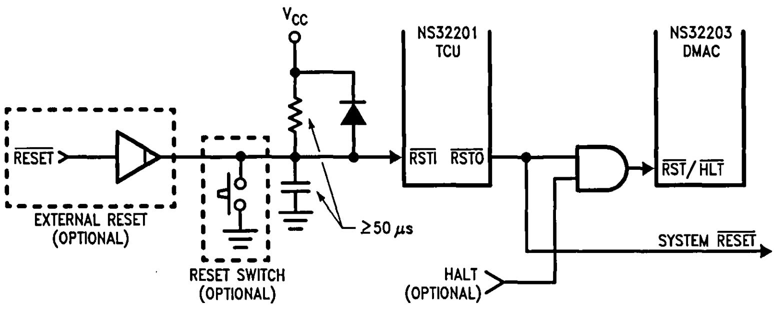


FIGURE 2-3. Recommended Reset Connections

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:

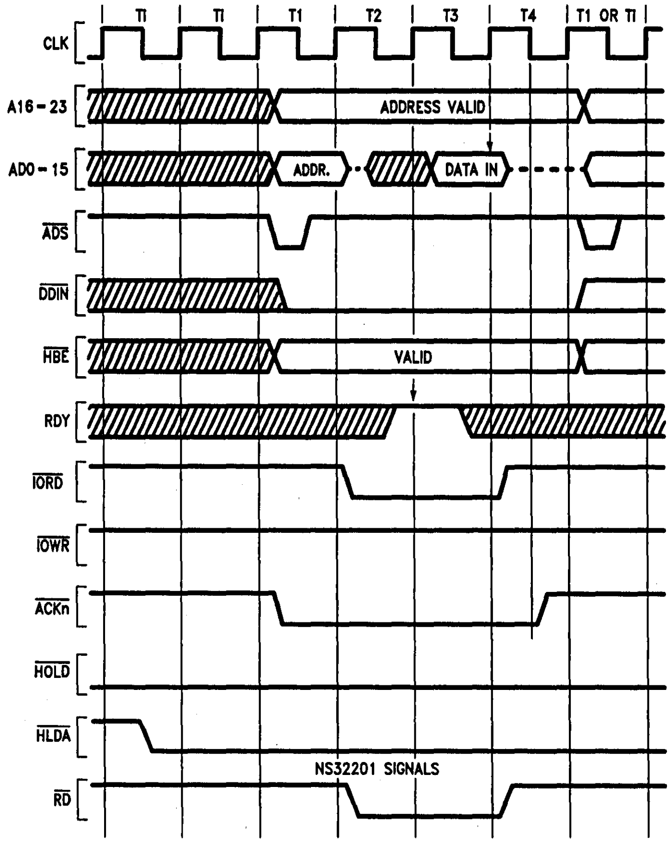
1) Acquires control of the bus
2) Acknowledge the requesting I/O device which is connected to the highest priority channel.
3) Starts executing data transfer cycles according to the values stored into the control registers of the channel being serviced.
4) Terminates data transfers and relinquishes control 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.

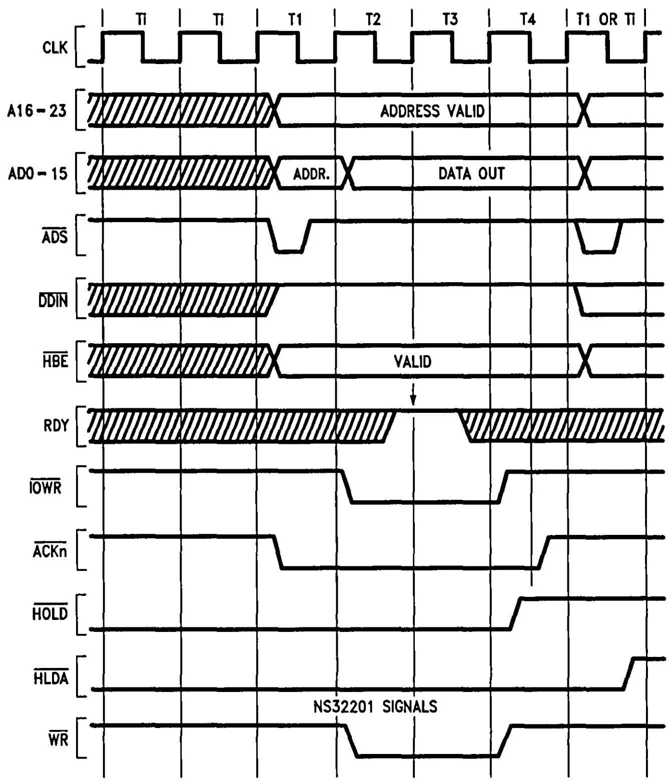
2.0 Functional Description (Continued)



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, $\overline{\text{HOLD}}$ is released at the end of the transfer operation.

2.0 Functional Description (Continued)

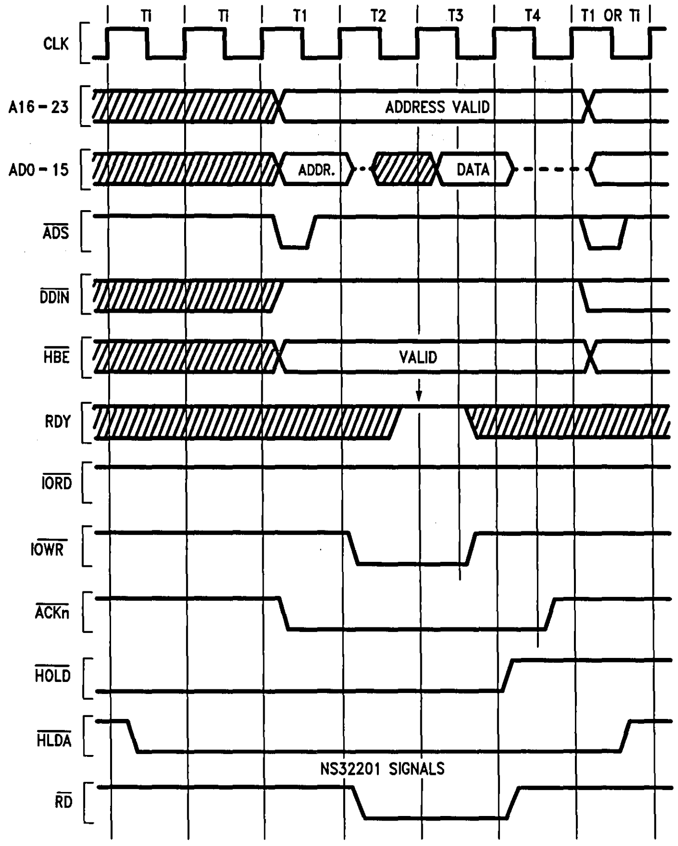
2.2.2 Direct (Flyby) Data Transfers

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.



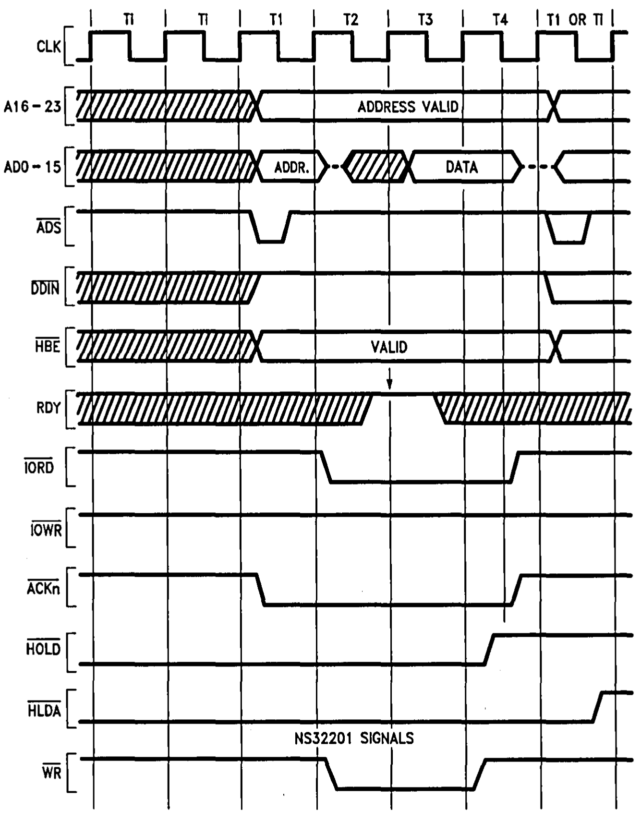
TL/EE/8701-7

FIGURE 2-6. Direct Memory-To-I/O Data Transfer (Single Transfer Mode)

2.0 Functional Description (Continued)

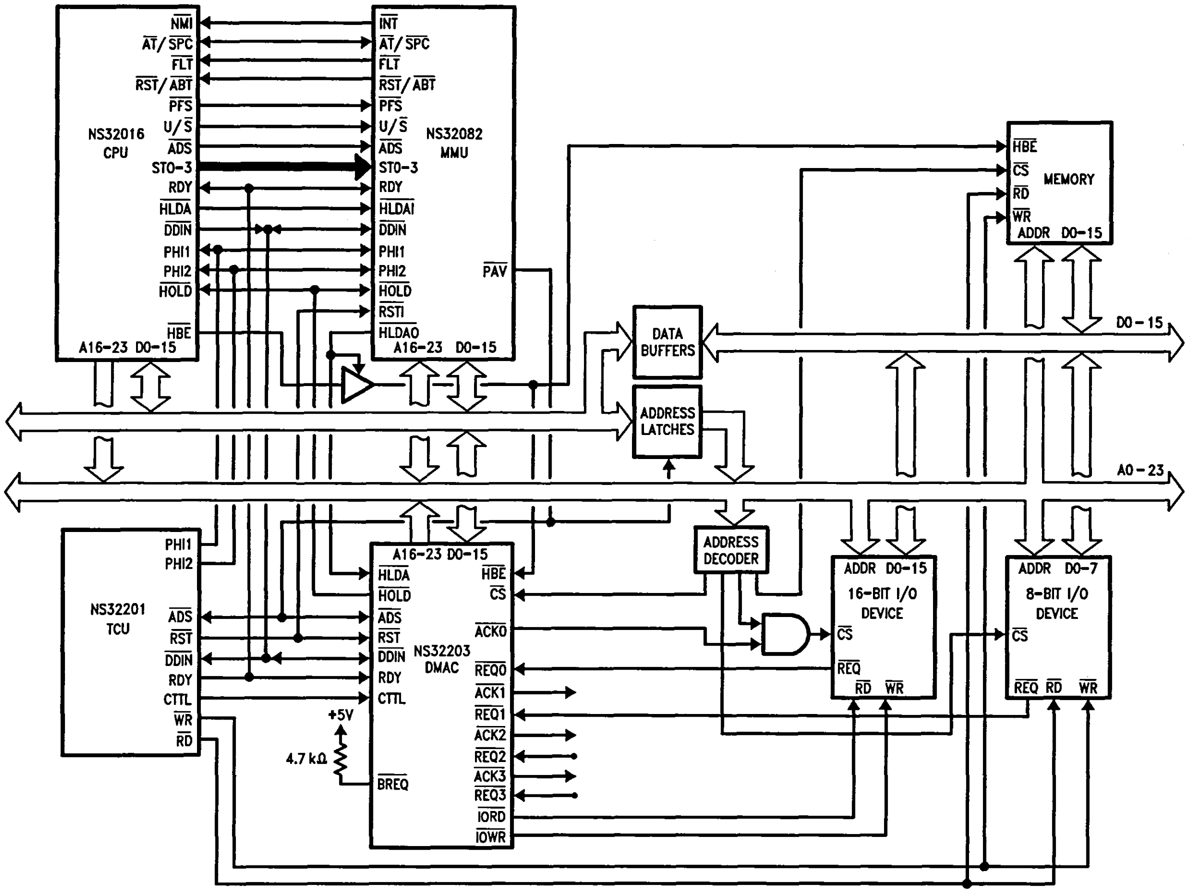
2.3 LOCAL CONFIGURATION

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.

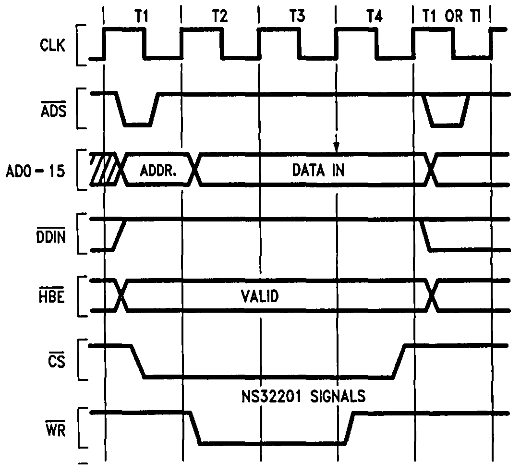
2.0 Functional Description (Continued)

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 $\overline{\text{BREQ}}$ signal. As the dedicated bus becomes idle, the DMAC pulls off the bus and acknowledges the CPU request by activating $\overline{\text{BGRT}}$. This output is also used as a control signal for the interconnection logic of the two buses.

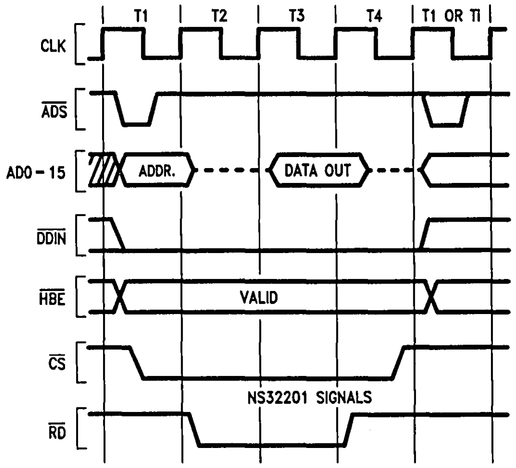
The CPU can either be interrupted by $\overline{\text{BGRT}}$ or it can poll $\overline{\text{BGRT}}$ to determine when the dedicated bus can be accessed. The DMAC, in turn, before accessing the CPU bus, has to gain control of it. This is accomplished through the usual request-acknowledge mechanism performed by means of the $\overline{\text{HOLD}}$ and $\overline{\text{HLDA}}$ 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 $\overline{\text{RWEN}}$/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 $\overline{\text{HOLD}}$ 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 $\overline{\text{IORD}}$ and $\overline{\text{ACKn}}$; the destination device is accessed by $\overline{\text{WR}}$ (from the NS32201) and $\overline{\text{CS}}$ (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 $\overline{\text{HBE}}$/ 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 $\overline{\text{REQn}}$ 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 $\overline{\text{INT}}$ 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 $\overline{\text{HOLD}}/\overline{\text{HLDA}}$ 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 $\longrightarrow$	ch.0	ch.3
2	ch.1	ch.1 ch.1 $\longrightarrow$	fixed priority
1	ACK $\longrightarrow$ 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 $\overline{\text{INT}}$ signal is deasserted unless another interrupt from a lower priority channel is pending. In this case the $\overline{\text{INT}}$ signal is deasserted when the acknowledge cycle for the second interrupt is performed.

For this reason, if the $\overline{\text{INT}}$ 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 $\overline{\text{RST}}/\overline{\text{HLT}}$ 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

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.

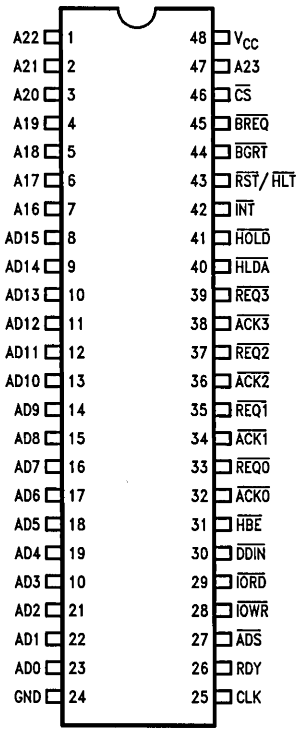
Note: A direct data transfer can be stopped by writing zeroes into the LNGT register. The number of bytes transferred can be determined in this case, from the value of either the SRC or the DST register.

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 $+70°C$, $V_{CC} = 5V \pm 5\%$, 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_{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_I	Input Load Current	$0 < V_{IN} \leq V_{CC}$	−20		20	μA
I_L	Leakage Current Output and I/O Pins in TRI-STATE/Input Mode	$0.4 \leq V_{IN} \leq V_{CC}$	−20		20	μA
I_{CC}	Active Supply Current	$I_{OUT} = 0, T_A = 25°C$		180	300	mA

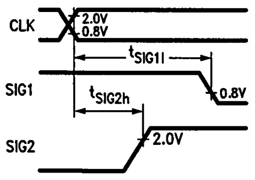
4.4 SWITCHING CHARACTERISTICS

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.

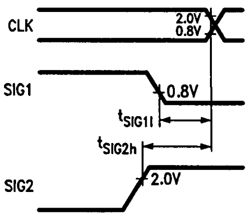
ABBREVIATIONS:

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

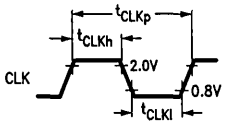
4.0 Device Specifications (Continued)

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

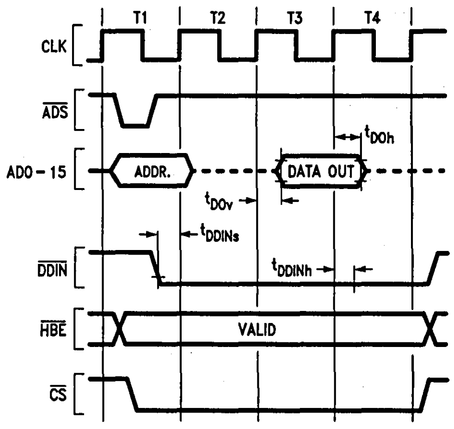
4.4.3 Timing Diagrams



TL/EE/8701–17

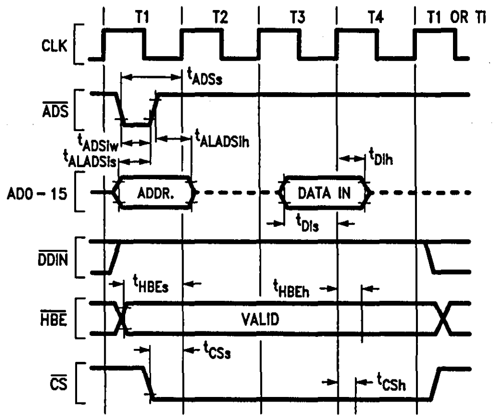
FIGURE 4-4. Clock Timing

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)

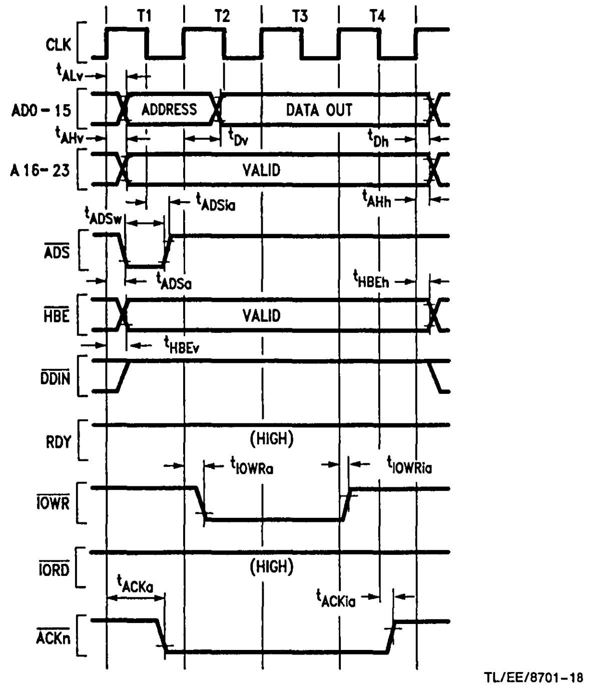


FIGURE 4-7. Indirect Write Cycle

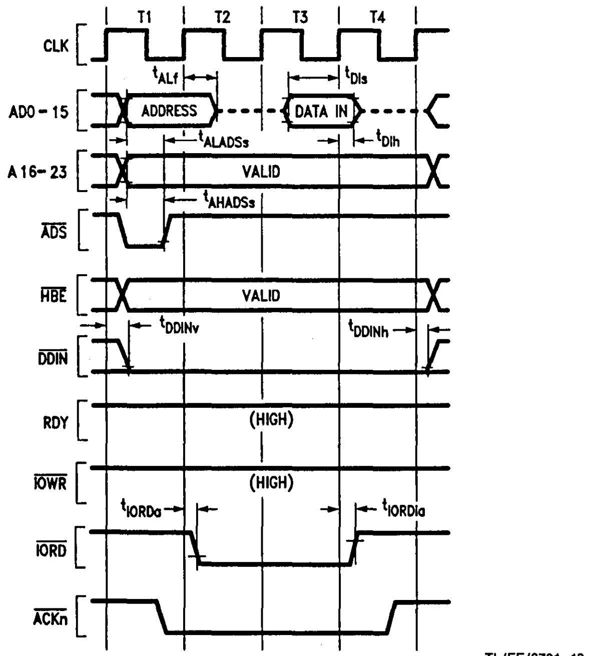
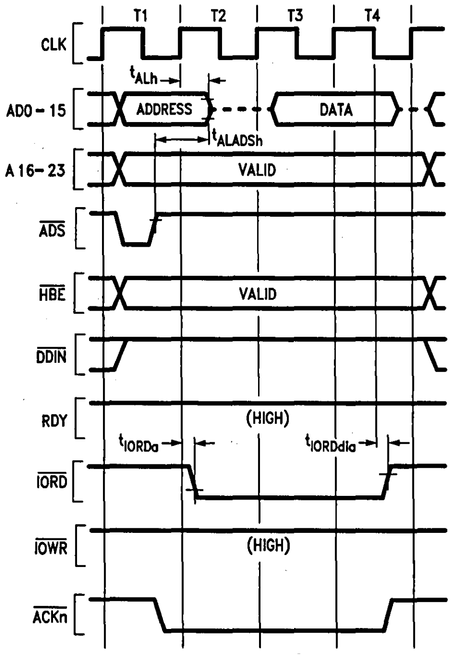


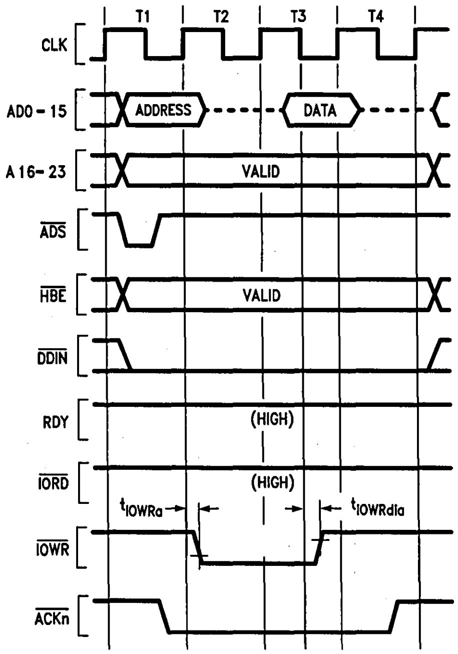
FIGURE 4-8. Indirect Read Cycle

4.0 Device Specifications (Continued)



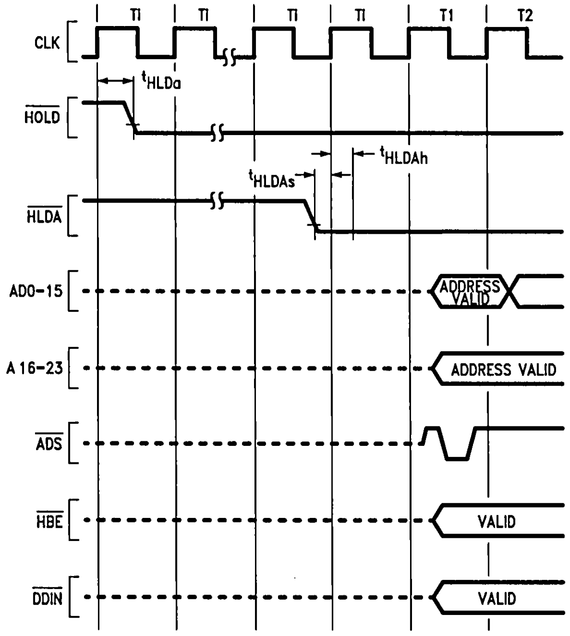
TL/EE/8701-20

FIGURE 4-9. Direct I/O to Memory Transfer



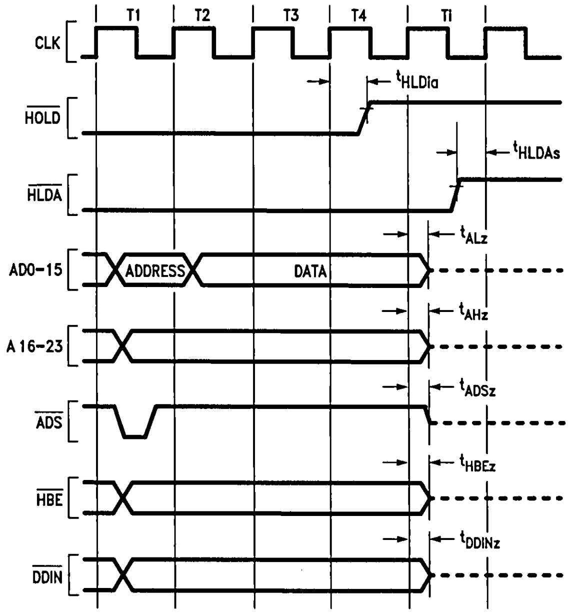
TL/EE/8701-21

FIGURE 4-10. Direct Memory to I/O Transfer



TL/EE/8701-22

FIGURE 4-11. $\overline{HOLD}/\overline{HOLDA}$ Sequence Start



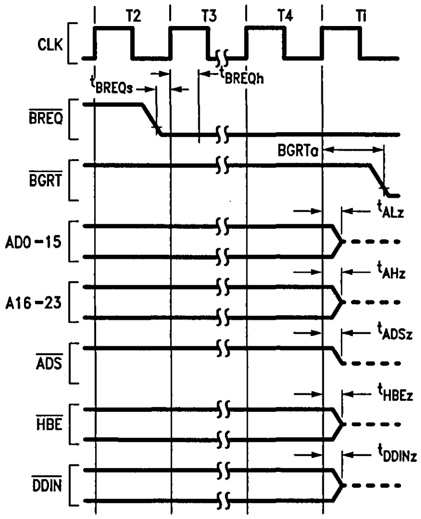
TL/EE/8701-23

FIGURE 4-12. $\overline{HOLD}/\overline{HOLDA}$ Sequence End

Note 1: DMAC in local configuration.

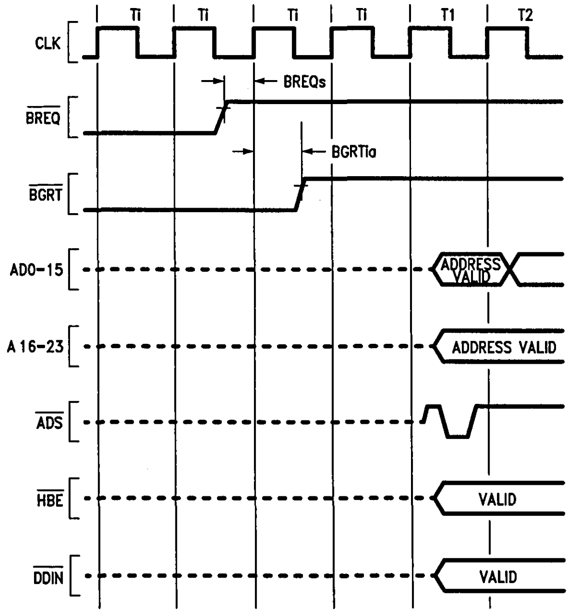
Note 2: The $\overline{HOLD}/\overline{HOLDA}$ sequence shown above is related to the single transfer mode. In burst transfer mode $\overline{HOLD}$ is deactivated two cycles later.

4.0 Device Specifications (Continued)



TL/EE/8701-24

FIGURE 4-13. Bus Request/Grant Sequence Start



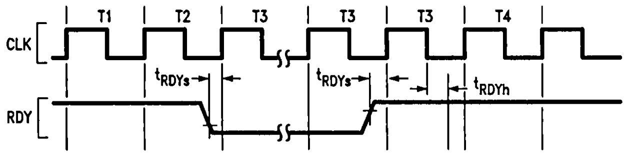
TL/EE/8701-25

FIGURE 4-14. Bus Request/Grant Sequence End

Note 1: DMAC in remote configuration.

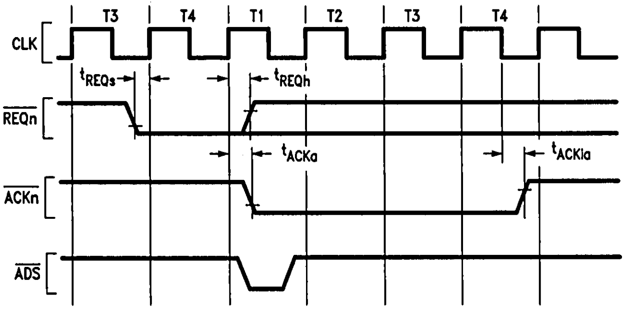
Note 2: If $\overline{\text{BREQ}}$ is asserted in the middle of a DMAC transfer, the transfer will always be completed.

4.0 Device Specifications (Continued)



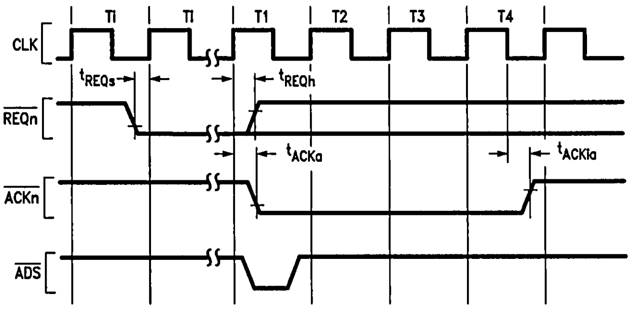
TL/EE/8701-26

FIGURE 4-15. Ready Sampling



TL/EE/8701-27

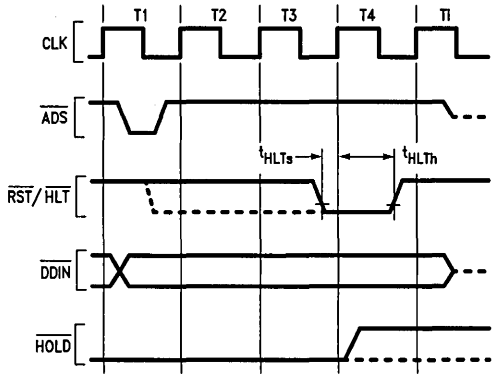
FIGURE 4-16. $\overline{REQn}$/$\overline{ACKn}$ Sequence (DMAC Initially Not Idle)



TL/EE/8701-28

FIGURE 4-17. $\overline{REQn}$/$\overline{ACKn}$ Sequence (DMAC Initially Idle)

4.0 Device Specifications (Continued)

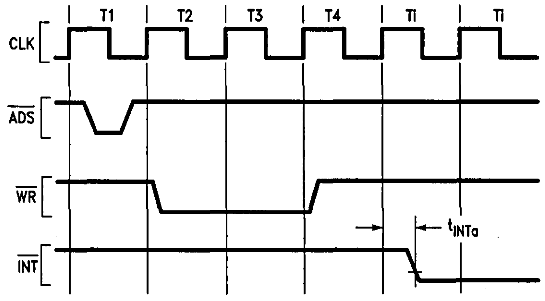


TL/EE/8701-29

FIGURE 4-18. Halted Cycle

Note 1: Halt may occur in previous T-States. It must be applied for 1 or 2 clock cycles.

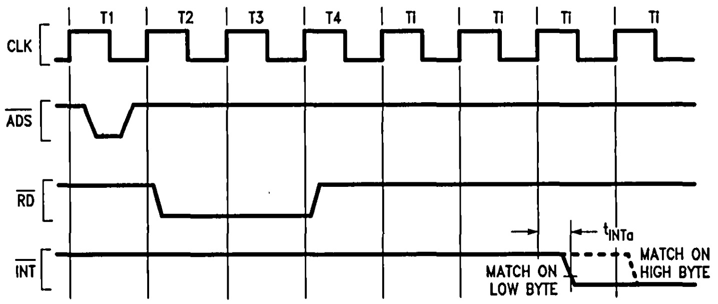
Note 2: If $\overline{BREQ}$ is asserted in the middle of a DMAC transfer, the transfer will always be completed.



TL/EE/8701-30

FIGURE 4-19. Interrupt on Transfer Complete

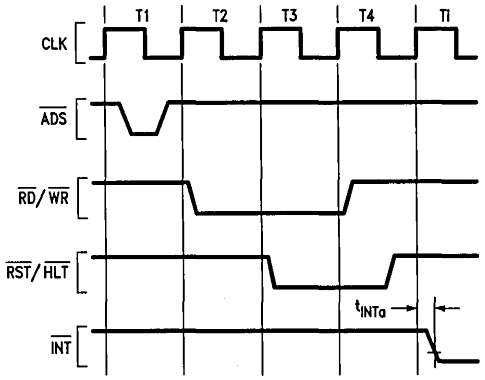
4.0 Device Specifications (Continued)



TL/EE/8701-31

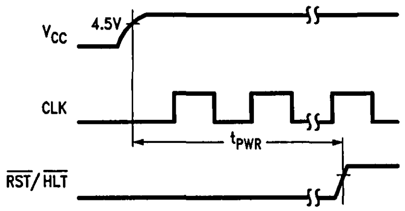
FIGURE 4-20. Interrupt on Match/No Match

Note: If inclusive search is specified a write cycle is performed before $\overline{INT}$ is activated.



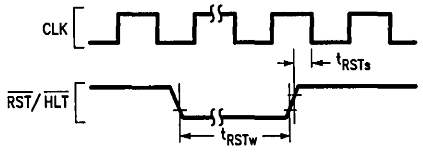
TL/EE/8701-32

FIGURE 4-21. Interrupt on Halt



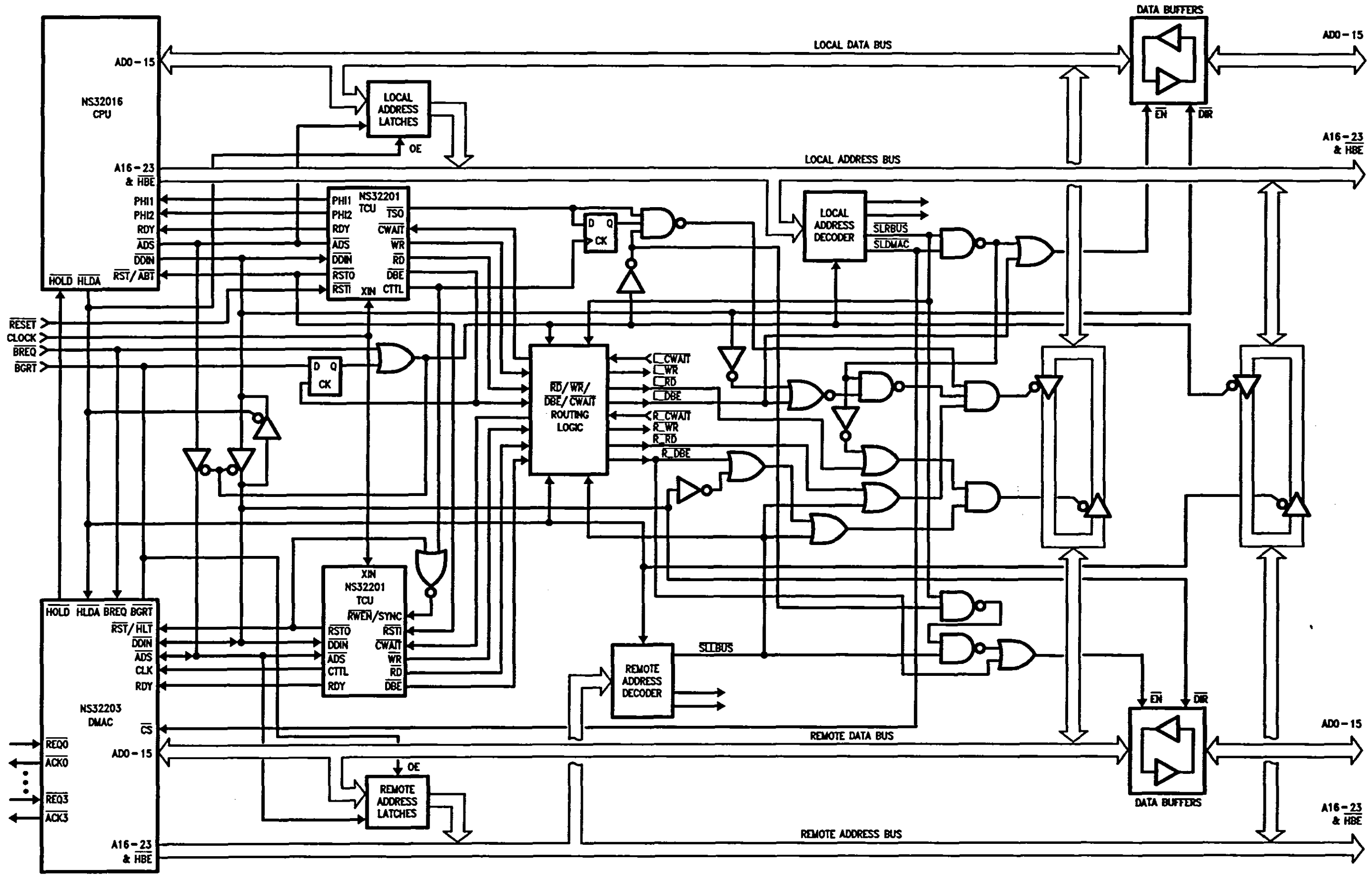
TL/EE/8701-33

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.

National Semiconductor

PRELIMINARY

NS32CG821 microCMOS Programmable 1M Dynamic RAM Controller/Driver

General Description

The NS32CG821 dynamic RAM controller provides a low cost, single chip interface between dynamic RAM and the NS32CG16. The NS32CG821 generates all the required access control signal timing for DRAMs. An on-chip refresh request clock is used to automatically refresh the DRAM array. Refreshes and accesses are arbitrated on chip. If necessary, a $\overline{\text{WAIT}}$ output inserts wait states into memory access cycles, including burst mode accesses. $\overline{\text{RAS}}$ low time during refreshes and $\overline{\text{RAS}}$ precharge time after refreshes and back to back accesses are guaranteed through the insertion of wait states. Separate on-chip precharge counters for each $\overline{\text{RAS}}$ output can be used for memory interleaving to avoid delayed back to back accesses because of precharge.

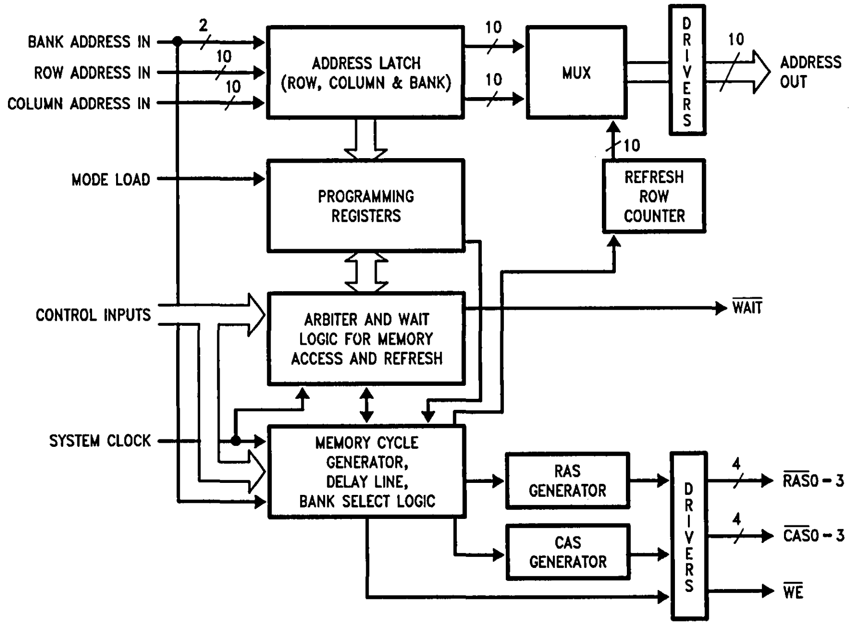
Features

- Allows zero wait state operation
- On chip high precision delay line to guarantee critical DRAM access timing parameters
- microCMOS process for low power
- High capacitance drivers for $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, $\overline{\text{WE}}$ and DRAM address on chip
- On chip support for page and static column DRAMs
- Byte enable signals on chip allow byte writing with no external logic
- Selection of controller speeds: 20 MHz and 25 MHz
- On board access refresh arbitration logic
- Direct interface to the NS32CG16 microprocessor
- 4 $\overline{\text{RAS}}$ and 4 $\overline{\text{CAS}}$ drivers (the $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ configuration is programmable)

Control	# of Pins (PLCC)	# of Address Outputs	Largest DRAM Possible	Direct Drive Memory Capacity
NS32CG821	68	10	1 Mbit	8 Mbytes

Block Diagram

NS32CG821 DRAM Controller



TL/F/10126-1

FIGURE 1

4

National Semiconductor

PRELIMINARY

HPC16083/HPC26083/HPC36083/HPC46083/ HPC16003/HPC26003/HPC36003/HPC46003 High-Performance microControllers

General Description

The HPC16083 and HPC16003 are members of the HPC™ family of High Performance microControllers. Each member of the family has the same core CPU with a unique memory and I/O configuration to suit specific applications. The HPC16083 has 8k bytes of on-chip ROM. The HPC16003 has no on-chip ROM and is intended for use with external direct memory. Each part is fabricated in National's advanced microCMOS technology. This process combined with an advanced architecture provides fast, flexible I/O control, efficient data manipulation, and high speed computation.

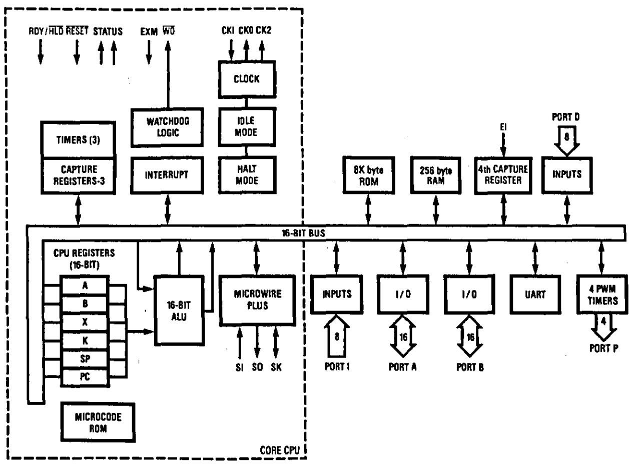
The HPC devices are complete microcomputers on a single chip. All system timing, internal logic, ROM, RAM, and I/O are provided on the chip to produce a cost effective solution for high performance applications. On-chip functions such as UART, up to eight 16-bit timers with 4 input capture registers, vectored interrupts, WATCHDOG™ logic and MICROWIRE/PLUS™ provide a high level of system integration. The ability to address up to 64k bytes of external memory enables the HPC to be used in powerful applications typically performed by microprocessors and expensive peripheral chips. The term "HPC16083" is used throughout this datasheet to refer to the HPC16083 and HPC16003 devices unless otherwise specified.

The microCMOS process results in very low current drain and enables the user to select the optimum speed/power product for his system. The IDLE and HALT modes provide further current savings. The HPC is available in 68-pin PLCC, LCC, LDCC, PGA and 84-Pin TapePak® packages.

Features

- HPC family—core features:
  - 16-bit architecture, both byte and word
  - 16-bit data bus, ALU, and registers
  - 64k bytes of external direct memory addressing
  - FAST—200 ns for fastest instruction when using 20.0 MHz clock, 134 ns at 30 MHz
  - High code efficiency—most instructions are single byte
  - 16 x 16 multiply and 32 x 16 divide
  - Eight vectored interrupt sources
  - Four 16-bit timer/counters with 4 synchronous outputs and WATCHDOG logic
  - MICROWIRE/PLUS serial I/O interface
  - CMOS—very low power with two power save modes: IDLE and HALT
- UART—full duplex, programmable baud rate
- Four additional 16-bit timer/counters with pulse width modulated outputs
- Four input capture registers
- 52 general purpose I/O lines (memory mapped)
- 8k bytes of ROM, 256 bytes of RAM on chip
- ROMless version available (HPC16003)
- Commercial (0°C to +70°C), industrial (−40°C to +85°C), automotive (−40°C to +105°C) and military (−55°C to +125°C) temperature ranges

Block Diagram (HPC16083 with 8k ROM shown)



TL/DD/8801-1

National Semiconductor

DP8510 BITBLT Processing Unit

General Description

The DP8510 BITBLT Processing Unit (BPU) is a high-performance microCMOS device designed for use in raster graphics applications. It implements, in high-speed pipelined logic, the data operations which are fundamental to BITBLT (BIT boundary Block Transfer) graphics: shifting, masking and bitwise logic operations. Under control of external hardware such as a state machine or a general-purpose microprocessor, it provides all necessary data path operations, easing the implementation of a wide variety of BITBLT systems. A number of input pins control the proper data flow in the BPU. A simple handshake scheme is used to interface the CPU, the BPU and the memory system.

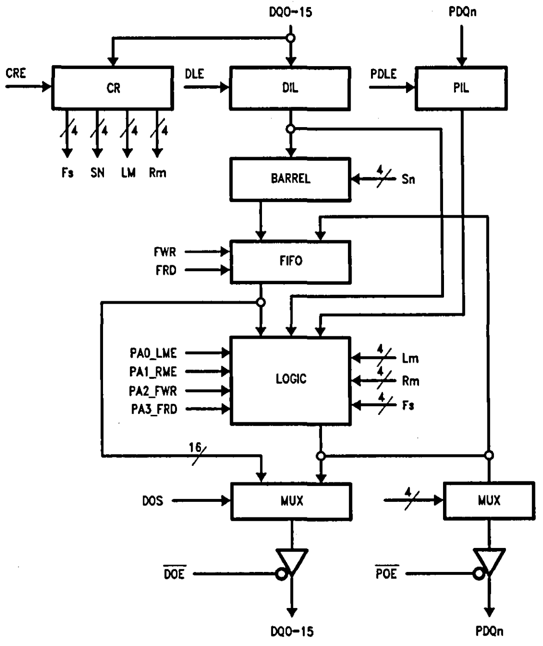
The BPU has two modes, BITBLT and line drawing. The mode is set by the $\overline{B}$/L pin. The line-drawing mode can be treated as a special case BITBLT with height and width equal to one.

In order to perform a BITBLT operation, the BPU's control register must first be loaded with four parameters: the shift number, left and right masks and the function select code, a total of 16 bits. BITBLT can then proceed, as directed by an external processor or state machine. It is the responsibility of the controller to generate appropriate addresses for the BITBLT, to interface with the frame buffer's memory control circuitry, and to control the BPU itself.

Features

- Supports all 16 classical BITBLT functions
- Pipelined data input for high system throughput
- Flexible architecture allows BPU to be used with a state machine or processor
- Multiple BPUs can be used for multiple bitplane/color applications
- Line drawing support
- Compatible with static or dynamic RAMs, including Video DRAMs
- Compatible with page mode, nibble mode and static column RAMs
- 32-bit to 16-bit barrel shifter
- 16-bit data port
- 16-word FIFO
- 16-bit logic operations
- 20 MHz operation
- Single +5 volt supply
- All inputs and outputs TTL-compatible
- Packaged in a 44-pin PCC (commercial) or 44-pin PGA (MIL)
- Single-bit pixel I/O port
- A member of National's Advanced Graphics Chip Set
- microCMOS technology

Block Diagram



TL/F/8672–22

National Semiconductor

DP8511 BITBLT Processing Unit (BPU)

General Description

The DP8511 BITBLT Processing Unit (BPU), a member of National Semiconductor's Advanced Graphics Chip Set (AGCS), is a high performance microCMOS device intended for use in raster graphics applications. Specifically designed to complement the DP8500 Raster Graphics Processor (RGP), the BPU performs data operations that are elementary to BITBLT (BIT boundary Block Transfer) graphics: Shift, mask, and bitwise logical manipulation of memory. Under the control of the RGP, the BPU performs the necessary BITBLT data path operations at pipelined hardware speeds. A simple set of control lines interfaces the BPU to the RGP and to the system memory.

The BPU has two modes of operation: BITBLT and Line Drawing. BITBLT performs shift and logical operations on blocks of 16-bit data words. Line drawing performs similar operations on single-bit pixel data by utilizing a single bit pixel port (PDn). This port allows data read and read-modify-write operations on single pixels across a number of bitplanes, giving access to pixel depth. The BPU provides both pixel level processing commonly used in image processing applications and extremely fast planar operations used most frequently in color graphics.

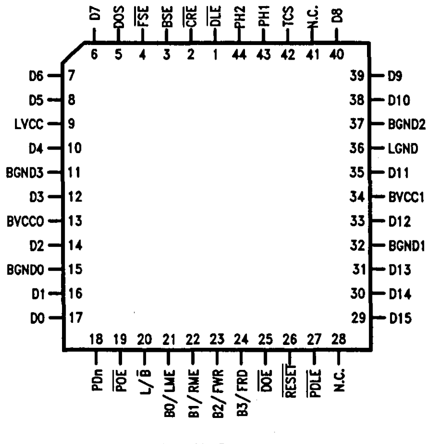
The BPU's operation is controlled by the values loaded to the Control Register (CR) and the Function Select Register (FSR). This dual register configuration of the DP8511 allows for high throughput in multi-plane systems that incorporate a BPU per plane. This performance advantage is achieved by allowing the flexibility of changing the FSR's contents independent of the CR, so that multiple bitplanes can be updated simultaneously while each BPU performs different logical operations on its own destination data.

Features

- Interfaces directly to the DP8500 Raster Graphics Processor or any general purpose controller
- 20 MHz operation
- Supports all 16 classical BITBLT functions
- Pipelined data input for high system throughput
- Provides performance independent of the number of bitplanes
- Line Drawing support
- Compatible with static, dynamic RAMs, and Video RAMs
- Compatible with page mode, nibble mode and static column RAMs
- 32-bit to 16-bit barrel shifter
- 16-bit data port, single bit pixel port
- 16-word FIFO
- 16-bit logic operations
- Single +5V supply
- All inputs and outputs TTL compatible
- 2 micron microCMOS technology
- Packaged in a 44-pin PCC (commercial) or 44-pin PGA (MIL)

Connection Diagrams

44-Pin Plastic Chip Carrier (PCC)



TL/F/9337-1

N.C. = No Connection

Top View

Order Number DP8511V
See NS Package Number V44A

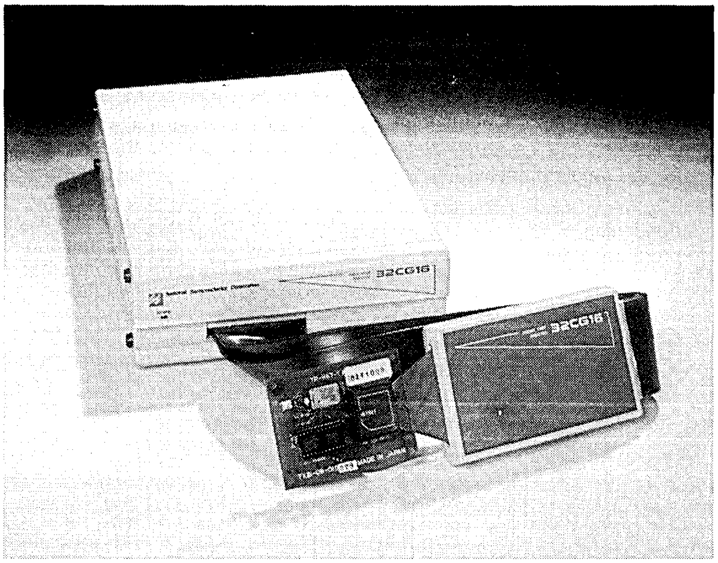
Section 5

Development Systems and Software Tools

Section 5 Contents

National Semiconductor

NS32CG16 ISE Development Tool



TL/EE/10334-4

- NS32CG16 emulator for software and hardware development and debugging
- 512 kbytes of mappable memory for emulation
- 15 MHz, 0 wait state access to emulation memory
- Sixteen definable events—match on address and data, no match on address and data and match on status conditions (address fetch, data read/write, slave cycle and interrupt acknowledge)
- Thirty-six software breakpoints using NS32CG16's BPT instruction
- Two hardware breakpoints based on events
- 2k deep, event triggered, real time, trace display in mnemonic and machine formats
- Execution time measurement with 1 μs resolution
- On screen menu for command selection
- FPU (Floating Point Unit) and BPU (Bit Aligned Block Transfer Processing Unit) support
- Software support via GNX™ tools
- Includes PC interface board and cable

1.0 Product Overview

The NS32CG16 ISE is a full featured emulator for the development of NS32CG16 based systems. The emulator works with SYS32/20 and SYS32/30 hosts. Up to 512 kbytes of memory may be mapped onto the target, allowing users to download their software into mapped (or emulation) memory. The emulator supports single stepping, 36 software breakpoints and 2 hardware breakpoints based on any of sixteen pre-defined events. Events may be defined as match on address & data, no match on address & data and match on status conditions (address fetch, data read/write, slave cycle and interrupt acknowledge). A 2k deep real time trace may be triggered by any of the sixteen pre-defined events and displayed in mnemonic or machine formats. The emulator supports execution time measurement with a resolution of 1 μs.

1.0. Product Overview (Continued)

The emulator connects to a high speed parallel interface board on the development system host. The emulator connects to the target system via a probe unit and target cable. An IC plug at the end of the target cable fits into the CPU socket on the target board. The probe unit contains an NS32CG16 microprocessor for emulation.

The emulator software resides in a DOS environment on the host. The emulator runs from a DOS environment on the host. An on-screen menu enables command selection.

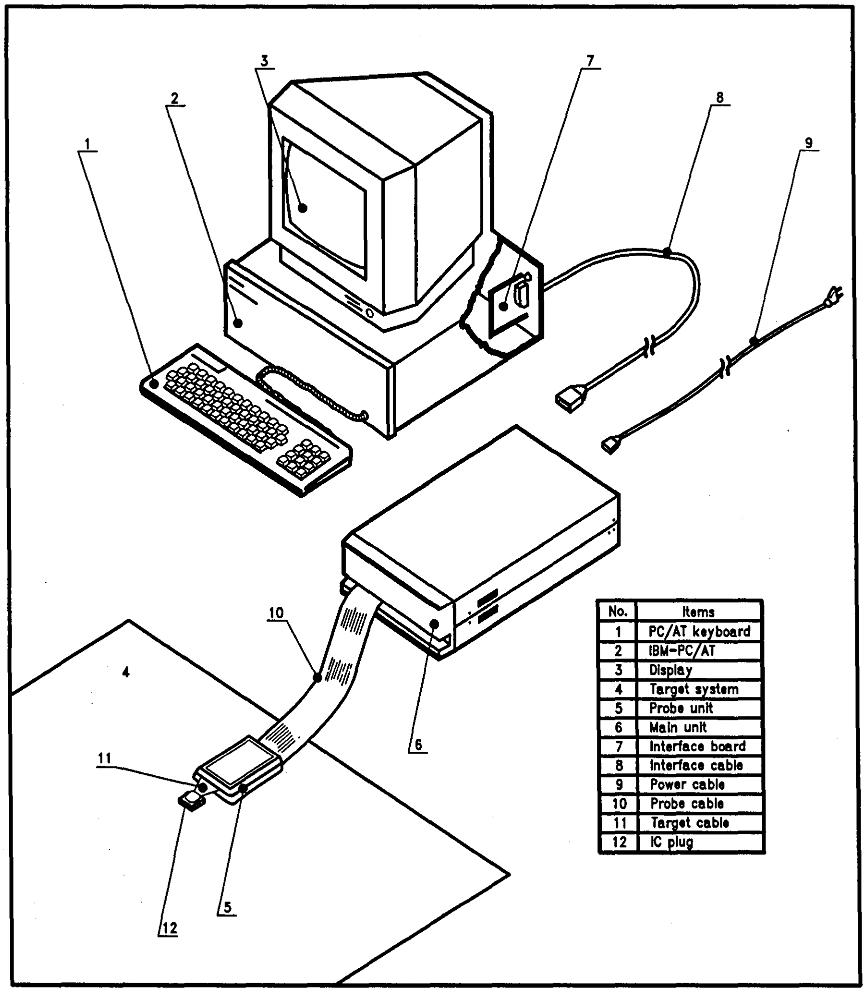
Commands supported by the emulator include:

Program down-loading
Assembly language debugging
Symbolic access to program variables
Modification of CPU registers and Memory locations
FPU and BPU slave processor support
Single stepping and software breakpoints
Trace display
On-screen command prompting facility

Full software support is provided by National's GNX tools in the UNIX® environment of the SYS32/20 or SYS32/30 host. The object files produced by the compilation (or assembly) and linking process in the UNIX environment may be converted into DOS-format files and loaded into the emulator.

2.0 Description of Features

The NS32CG16 ISE consists of a main emulator unit, a probe unit with target cable and IC plug, an interface cable and PC interface board that resides on the host. *Figure 1* shows a pictorial view of the emulator.



No.	Items
1	PC/AT keyboard
2	IBM-PC/AT
3	Display
4	Target system
5	Probe unit
6	Main unit
7	Interface board
8	Interface cable
9	Power cable
10	Probe cable
11	Target cable
12	IC plug

TL/EE/10334-1

FIGURE 1. NS32CG16 Emulator System

2.0 Description of Features (Continued)

2.1 NS32CG16 ISE System Configuration

Figure 2 shows the NS32CG16 ISE system configuration.

2.2 Description of the System

The development system consists of the SYS32/20 or SYS32/30 host computer with the emulator interface board, the emulator and probe units and the IC plug (located at the end of the target cable) which fits into the CPU socket on the target board. The emulator SCSI interface board enables high speed parallel communication between the emulator and the host. The probe unit contains an NS32CG16 microprocessor for emulation.

The emulator unit consists of Controller, Memory, Trace and Breakpoint and Probe Interface boards. The Controller board communicates with the SCSI interface on the host and with all other boards in the emulator unit. The Probe Interface board communicates with the probe unit.

The Memory Board provides 512 kbytes of emulation memory, with 0-wait state access at 15 MHz. Sixteen memory partitions may be mapped in 4 kbyte blocks with write protection capability. 4 kbytes of the available memory is used by the emulator's monitor, and the remaining memory may be used for emulation.

The Trace and Breakpoint board supports trace and breakpoint capabilities. The 2k deep trace of address, data and status may be displayed in mnemonic or machine formats, and may be triggered by any of 16 predefined events. Two hardware breakpoints (based on any of the 16 predefined events) and 36 software breakpoints are supported.

Execution time measurement is accomplished with a resolution of 1 μs, and may be measured between two instruction execution addresses or between the occurrence of any two of the 16 predefined events.

Sixteen events may be defined based on the following:

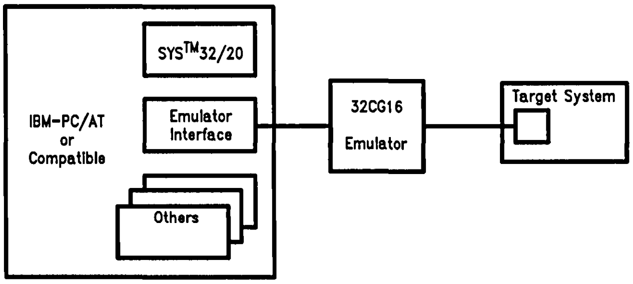
match on address and data
no match on address and data
match on status conditions (address fetch, data read/write, slave cycle and interrupt acknowledge)

In specifying the formats for the address and data, for example, any combination of 0s, 1s or Xs (don't cares) may be used. For example FFX0 or XXFF (in hexadecimal) are valid formats for specifying address and data.

All symbolic information in the source program is retained during debugging.

The emulator software resides in a DOS environment on the system host. The emulator runs from the DOS environment and may be invoked from the DOS directory in which the emulator software resides and commands may then be issued to control the operating mode of the emulator. An on-screen menu enables selection of commands with prompting facility. Commands are provided to download, execute and debug programs. The command structure supports symbolic access to program variables.

Software support is provided by National's GNX tools in the UNIX environment on the SYS32 host. A user program may be edited, compiled and linked in this environment to obtain an executable object file. The object file may then be converted into DOS-format and copied into the DOS environment, by using the **udcp** (UNIX to DOS copy) utility in the UNIX environment. This resulting DOS-format file may be directly loaded into emulation memory by emulator commands. The **ducp** (DOS to UNIX copy) utility may be used (in the UNIX environment) to convert files in the DOS-format (in the DOS environment) to UNIX-format (in the UNIX environment). Both **udcp** and **ducp** also support conversion of ASCII files.



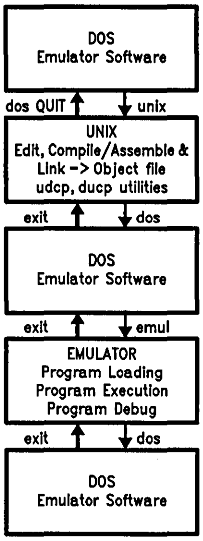
TL/EE/10334-2

FIGURE 2. NS32CG16 ISE System Configuration

2.0 Description of Features (Continued)

2.3 The Development Process

Figure 3 shows the development process in the different environments.



TL/EE/10334-3

FIGURE 3. The Development Process

2.4 Command Summary

The following is a summary of the commands supported by the emulator.

CONFIGURATION COMMANDS

Mapping address Thru address Rom|RAm|TArget|Locked

Maps 4 kbyte memory blocks in the specified address range as ROM, RAM, Target or Locked memory space.

MOnitor address

This command maps a single 4 kbyte memory block at specified address for use by the monitor.

Interrupt Enable|Disable Nmi|Int

Enables or Disables the selected interrupt NMI or INT.

DMa Enable|Disable

Enables or Disables DMA transfers when the CPU is not accessing the bus.

Break Enable|Disable Monitor|Rom write

Enables or Disables a break in program execution when an access to Monitor address space or a write to the ROM address space occurs.

Load Coff|Sform file Offset

Loads a specified file in COFF or Motorola S formats into memory at a specified offset from address 0.

Store file From address Thru address

Stores the program data in the specified address range in memory into the specified file in Motorola S format.

Clear

Clears all the symbols used in the program.

2.0 Description of Features (Continued)

2.4 Command Summary (Continued)

The following is a summary of the commands supported by the emulator.

DISPLAY COMMANDS

Display Configuration
Displays the current configuration of the emulator.

Display Register Format General|Single|Double
Displays CPU registers in the specified format.

Display Memory address Format Byte|Word|Dword|Qword|Mnemonic|Single|Double
Displays memory contents starting at specified address in the specified format.

Display Trace Trigger|TOp|Bottom|line Mnemonic|MAchine
Displays results of the trace with the specified display position and display format.
The display position may be specified at the Trigger point or the top of the trace or the bottom of the trace or a specified line number on the trace.
The display format may be specified to be in mnemonic or machine formats.

Display SWbreak
Displays all the software breakpoints.

Display Event
Displays all the pre-defined events.

DATA MANIPULATION COMMANDS

Register Format General|Single|Double
Specifies the display and change formats for register commands.

MOdify reg To data
Modifies the specified register to the specified data.

Memory address Format Byte|Word|Dword|Qword|Mnemonic|Single|DOuble
Specifies the display and change formats for memory commands.

MOdify address Thru address To data
Modifies the memory locations in the specified address range to the specified data.

EVENT SETUP COMMANDS

Event
Initiates the event definition process.

Add Address =|# address Data =|# data Status Off|Fetch|Data|DRead|DWrite|Intack|Slave
Adds an event with specified address match or nomatch, with specified data match or nomatch, and specified status conditions.

Replace number Address =|# address Data =|# data Status Off|Fetch|Data|DRead|DWrite|Intack|Slave
Replaces the event with the specified event number with the new event defined with the specified address match or nomatch, with specified data match or nomatch, and specified status conditions.

DELete All|number
Deletes all currently defined events or the event with the specified event number.

2.0 Description of Features (Continued)

2.4 Command Summary (Continued)

The following is a summary of the commands supported by the emulator.

SOFTWARE BREAKPOINT COMMANDS

SWbreak
Initiates the setup of software breakpoints.

Add address
Adds a software breakpoint at specified address.

Replace number To address
Replaces the breakpoint address of a pre-defined breakpoint (referenced by the specified number) with the new specified address.

DELete All|number
Deletes all the pre-defined software breakpoints or the pre-defined breakpoint with the specified number.

Set Enable|Disable All|number
Enables or disables the state of all pre-defined software breakpoints or the pre-defined software breakpoint (referenced by the specified number).

PROGRAM EXECUTION COMMANDS

RESet
Resets the CPU.

Go From address Until address1|Event# Or address2|Event# Times number
Executes program from specified address until a match occurs on the specified address (address1) or on the specified event (hardware breakpoint #1), or until a match occurs on the specified address (address2) or on the specified event (hardware breakpoint #2). A specified number of times a specified match occurs may also be used to control program execution. If the hardware breakpoint conditions are omitted, then program execution breaks on the software breakpoints that may be set and enabled.

Step From address
Executes one instruction from the specified address.

Trace From address Trigger address1|Event# Or address2|Event#
Enables the trace from the specified address, with the trigger points being defined by address1 or a specified event or by address2 and a specified event.

MEAsure From address Start address1|Event# End address2|Event#
Enables program execution from specified address with execution time being measured from specified start address1 or event until the specified end address2 or event.

Quit
Forces a break in program execution and stops the CPU.

EMULATOR CONTROL COMMANDS

CANcel
Resets the emulator to its initial state at start-up.

EXIT
Exits from the emulator environment to the DOS environment.

DOS
Suspends temporarily to the DOS environment from the emulator environment.

MAcro file
Executes command lines stored in the specified macro file in text format.

3.0 Specifications

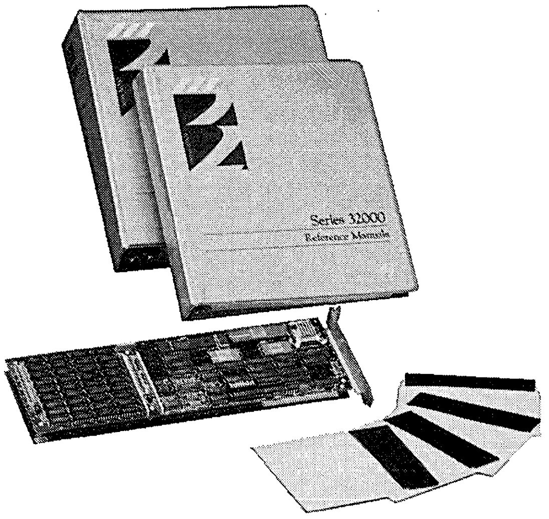
Environment	The NS32CG16 ISE is designed to operate in a laboratory environment. The emulator unit may be mounted horizontally (flat) or vertically.
Temperature	Operative: +15°C to +50°C Storage: −40 °C to +60°C
Humidity	10% to 90% relative, non-condensing
Altitude	Operative 15000 feet
Power Requirements	NS32CG16 ISE requires a standard AC power outlet (125V AC).

4.0 Ordering Information

NSS-ISE-CG16 NS32CG16 Emulator.

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.

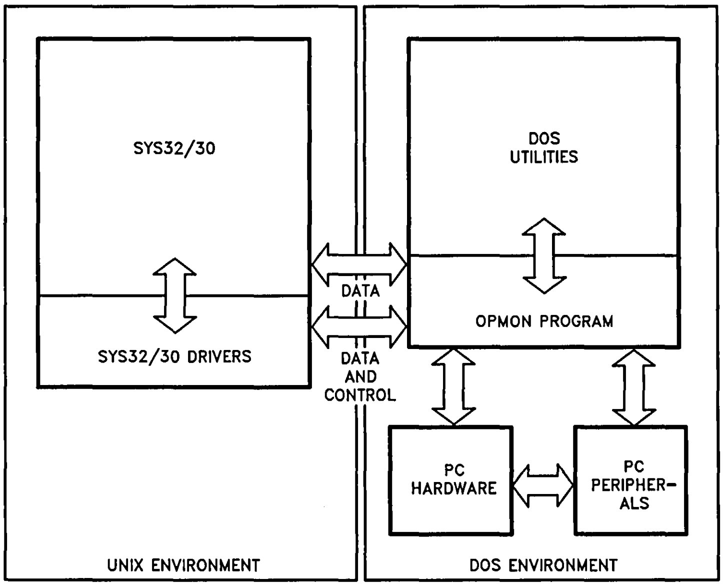
This processor board forms the high-performance center of the computer system with the host PC/AT processor. Peripherals are under the control of the PC/AT's microprocessor and are located either on the PC/AT motherboard or on other boards in the PC/AT chassis. The PC/AT handles all direct access to 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

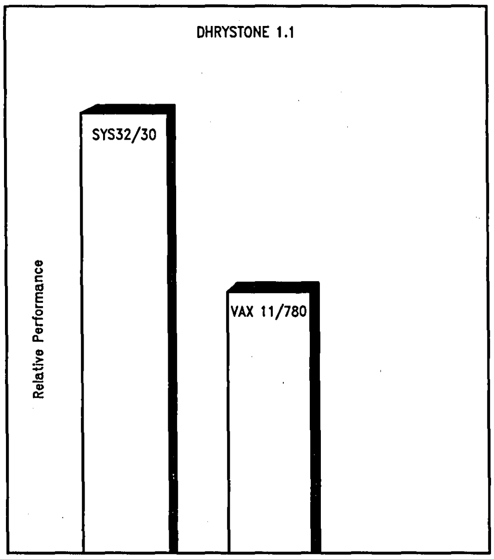
The NS32332 CPU and associated devices operating at 15 MHz provide computing power greater than that of a VAX 11/780. Sustained performance for the NS32332 device cluster is 2–3 VAX MIPS (Million Instructions Per Second). An example of relative performance using the widely recognized Dhrystone benchmark 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

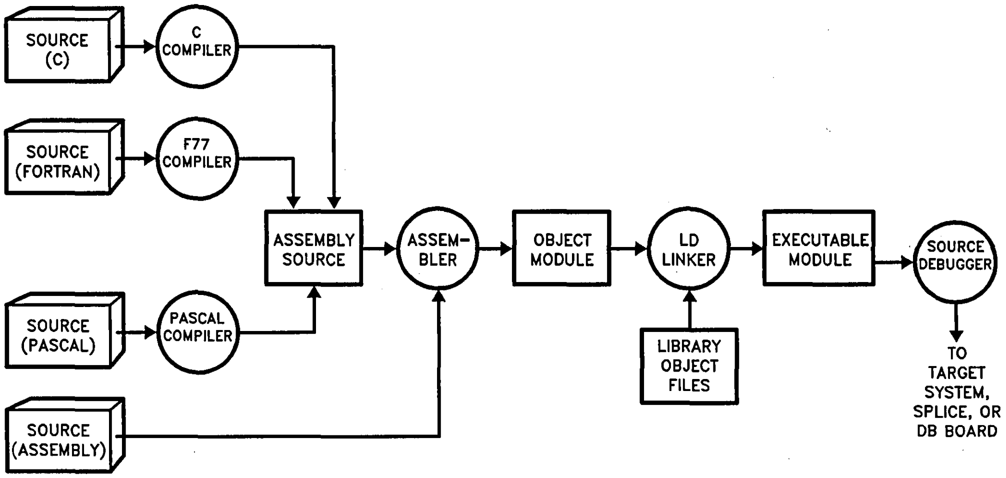
The SYS32/30 with the PC/AT operates as a local host computer system for integrating application software into target prototype boards containing Series 32000 components. Programs can be written in assembly language or in a higher level language. Optional compilers are available for C, FORTRAN 77, and 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.

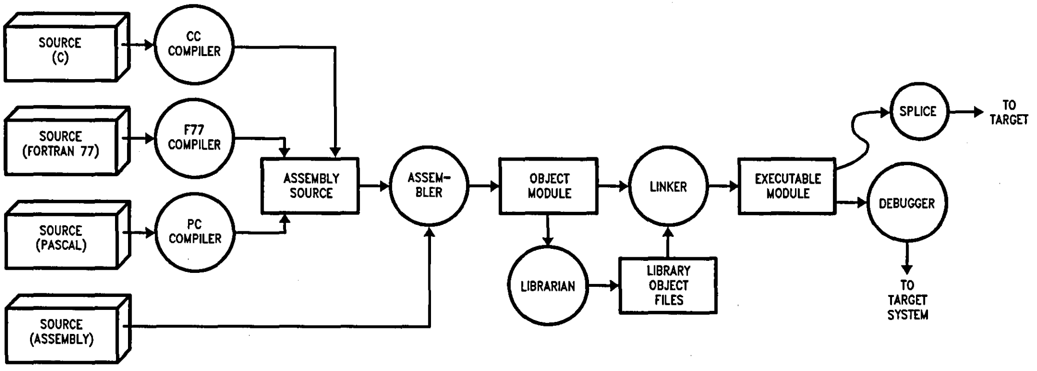
Optional Software Packages

(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-

Supported Development Hosts (Continued)

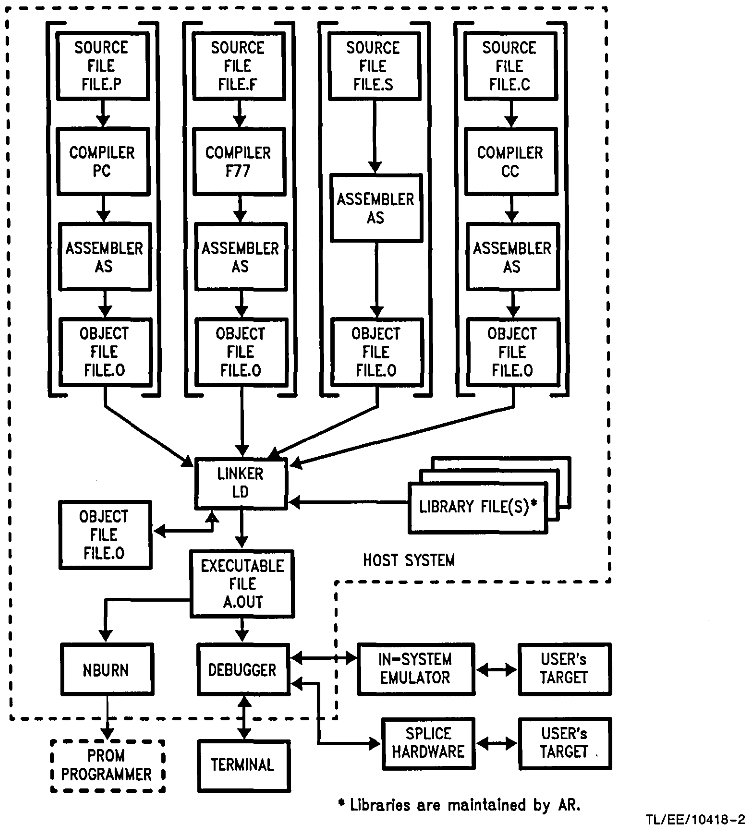


FIGURE 1. Sample Development Process

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

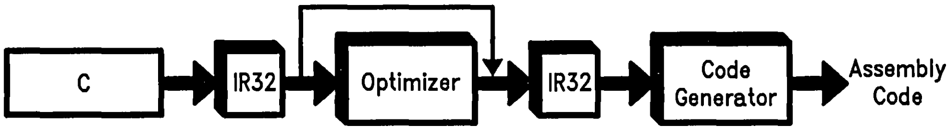
GNX-Version 3 Pascal Optimizing Compiler

SYS32/20 PC-Add-In Development Package

SYS32/30 PC-Add-In Development Package

SPLICE Development Tool

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 System 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:"

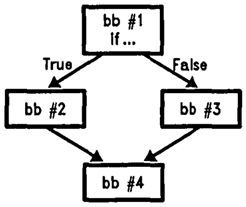
```
        b ← 20
        ...
```

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	

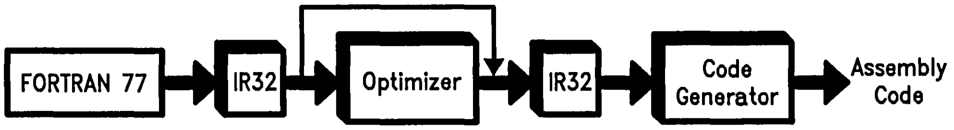
GNX-Version 3 Assembler and Cross-Development tools (required for use with the Optimizing C Compiler):

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:"

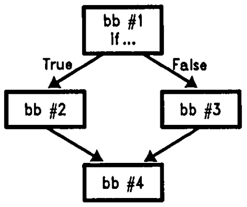
```
        b ← 20
        . . .
```

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: NSW-F77-3-BHAF3	**VAX/ULTRIX (UNIX bsd):** NSW-F77-3-BRVX
SYS32/30: NSW-F77-3-BHBF3	**Micro VAX/VMS:** NSW-F77-3-BCVM
VAX/VMS: NSW-F77-3-BRVM	**Micro VAX/ULTRIX:** NSW-F77-3-BCVX

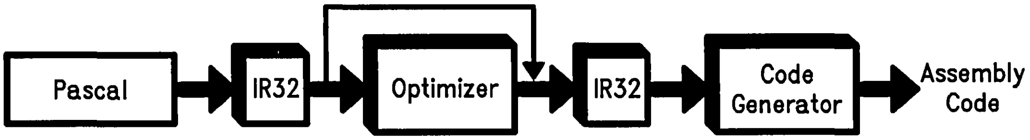
GNX-Version 3 Assembler and Cross-development tools (required for use with the Optimizing FORTRAN 77 Compiler):

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

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:

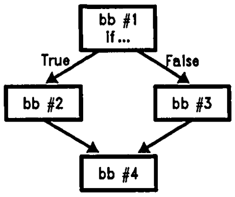
- **Value Propagation**—replacing variables with their most recent values
- **Constant Folding**—evaluating expressions that consist solely of constants
- **Redundant Assignment Elimination**—eliminating assignments that are not used or that are reassigned prior to use

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
Physical Dimensions/ Appendices

Section 6 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.

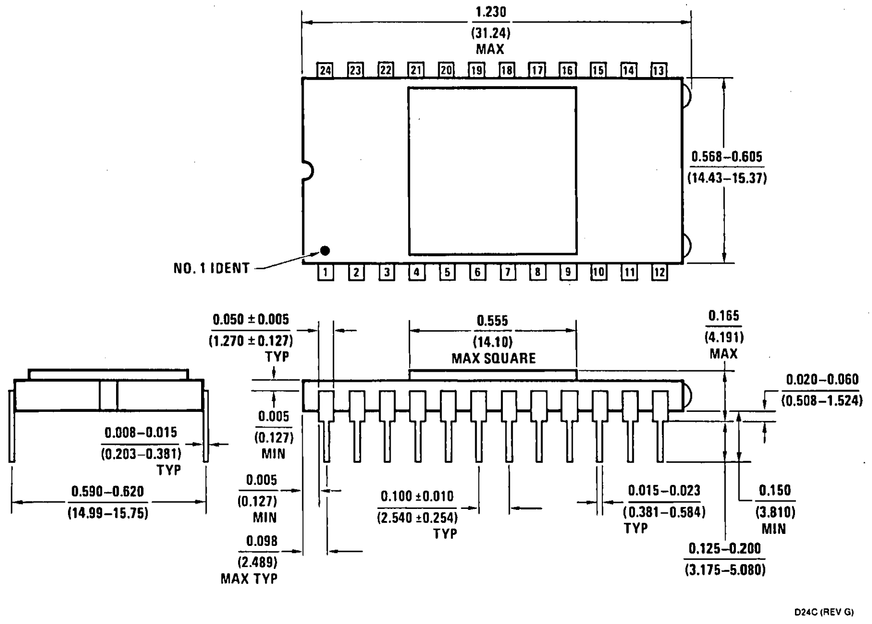
Winchester—Small, hard-disk media commonly found in personal computers.

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.

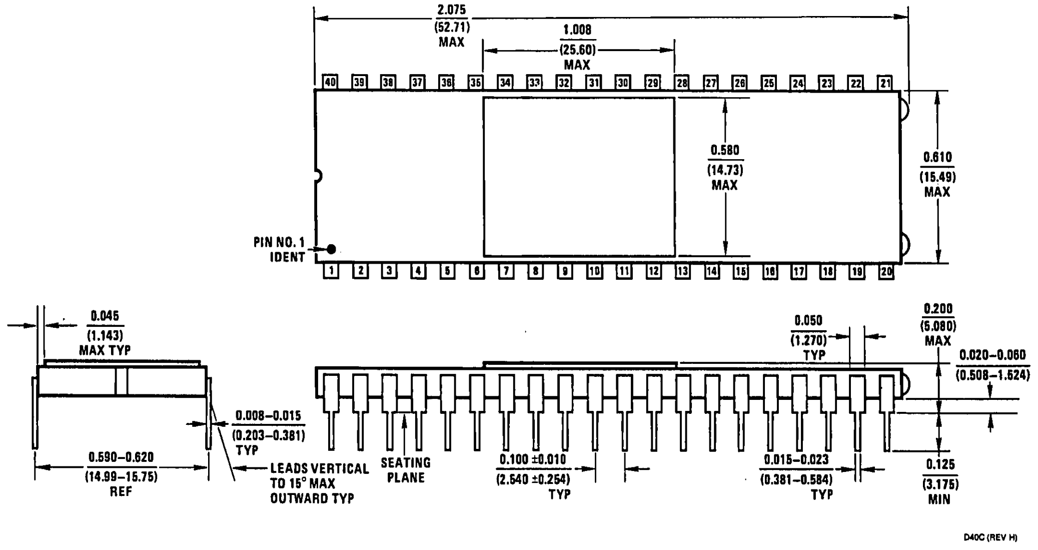
National Semiconductor

All dimensions are in inches (millimeters)

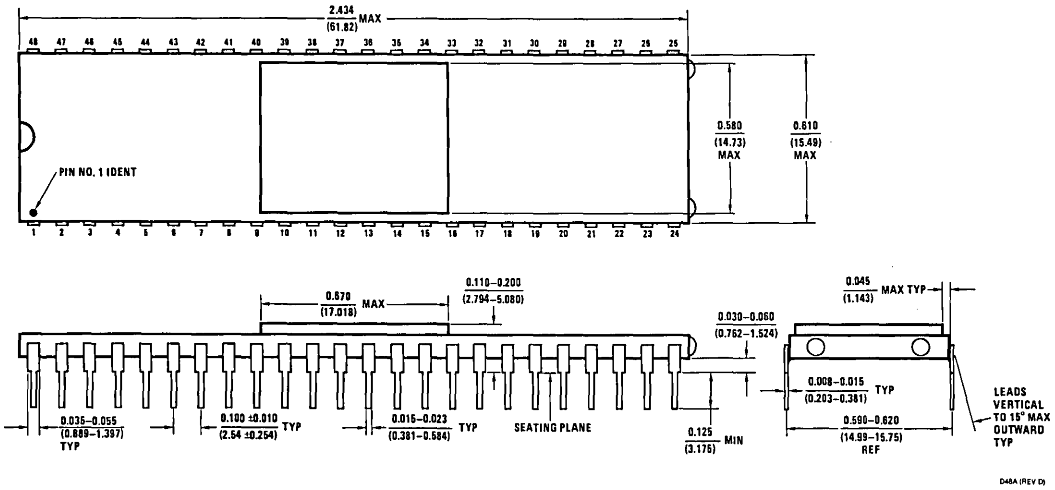
24 Lead Hermetic Dual-In-Line Package (D) NS Package Number D24C



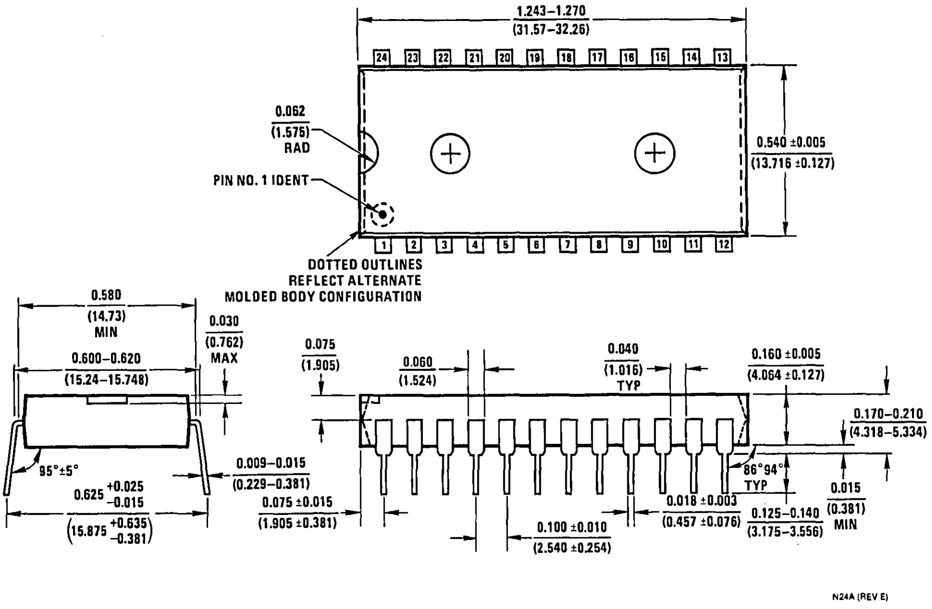
40 Lead Hermetic Dual-In-Line Package (D) NS Package Number D40C



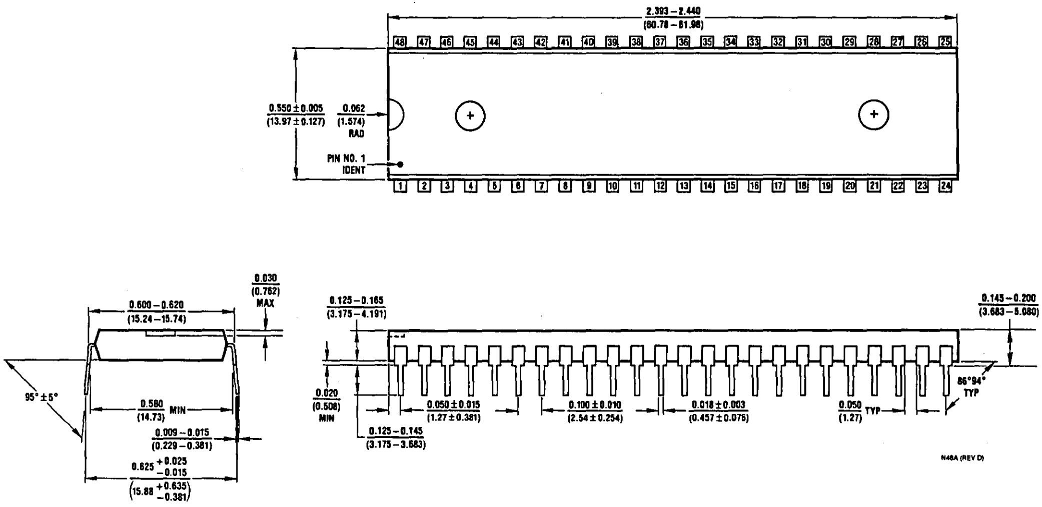
48 Lead Hermetic Dual-In-Line Package (D) NS Package Number D48A



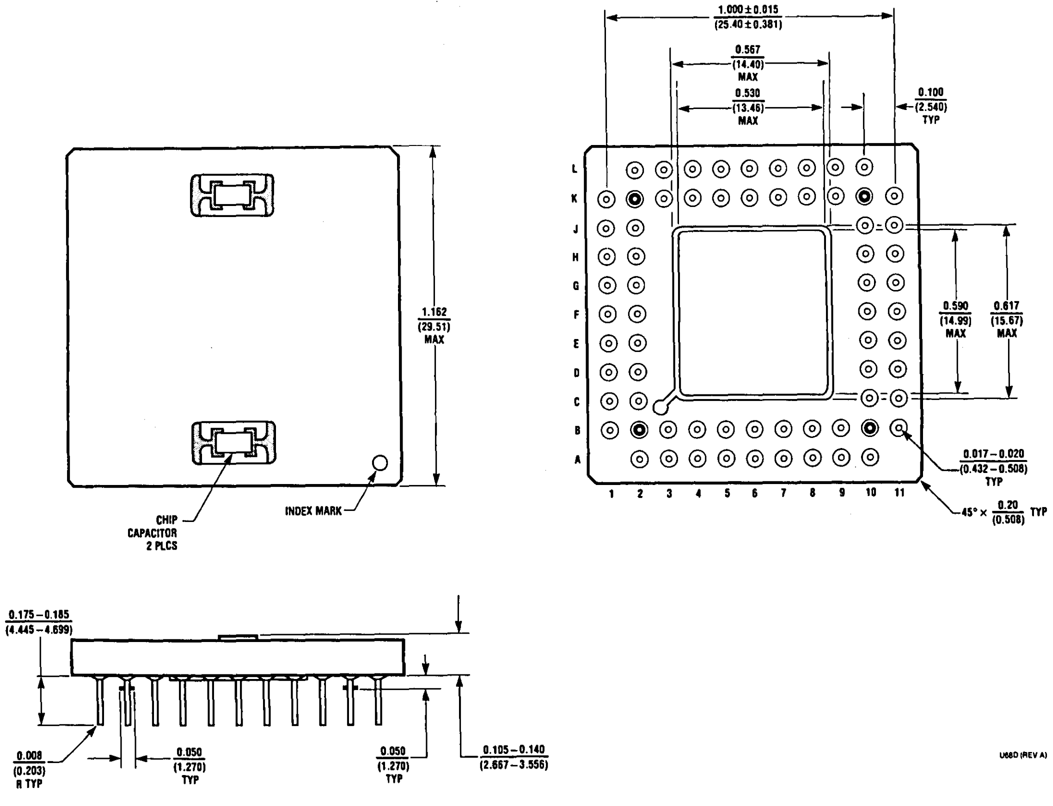
24 Lead Molded Dual-In-Line Package (N) NS Package Number N24A



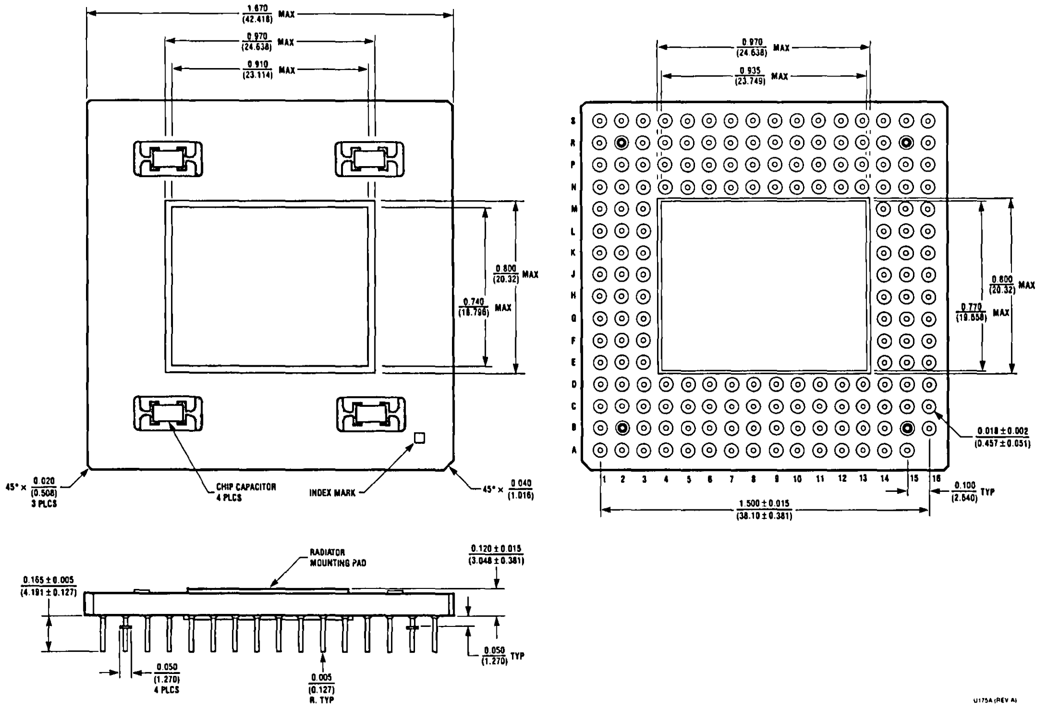
48 Lead Molded Dual-In-Line Package (N) NS Package Number N48A



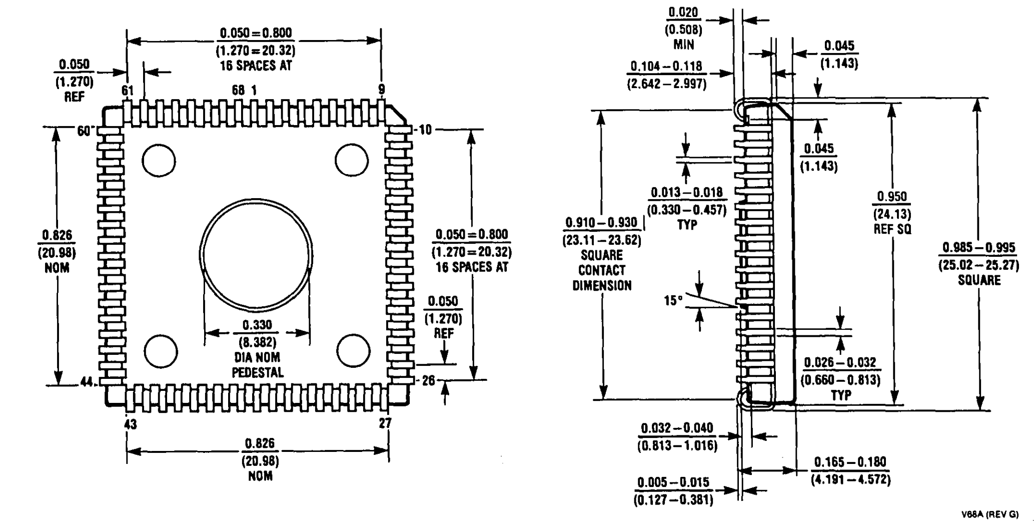
68 Pin Grid Array, Cavity Down NS Package Number U68D



175 Pin Grid Array, Cavity Down (Type A) NS Package Number U175A



68 Lead Plastic Chip Carrier NS Package Number V68A



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 Semiconductor 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) [illegible] 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) 614141
Telex: 444674
Fax: (07-93) 69-75-22

National Semiconductor Benelux
Vorstlaan 100
B-1170 Brussels
Belgium
Tel: (02) 6725360
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 Boursidière
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) 692955
Telex: 91047
Fax: (01) 692956

National Semiconductor S.p.A.
Strada 7, Palazzo R/3
20089 Rozzano
Milanofiori
Italy
Tel: (02) 8242046/7/8/9

National Semiconductor AS
[illegible] 107
[illegible] Oslo
Norway
Tel: (02) [illegible]
Fax: (02) [illegible]

National Semiconductor (UK) Ltd.
P.O. Box 56
[illegible]
Finland
Tel: (0) [illegible]
Fax: (0) [illegible]

National Semiconductor AB
Box 2016
Stensätravägen 13
S-12702 Skärholmen
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-Zürich
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: (0-2940) 3-04-48
Telex: 10-956
Fax: (0-2940) 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: K24942 NSRKLO
Fax: (02) 784-8054

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