

HONEYWELL BULL BILLERICA	SPECIFICATION NUMBER 60165905	DISTRIBUTION CODE C40C	SHEETS 1/xx	REV 1
CUSTOM AND SPECIAL PRODUCTS	TITLE:  PRODUCT TECHNICAL DESCRIPTION  16-BIT CUSTOM PROCESSOR			
PREPARED BY R Lemay				
APPROVED BY				

REVISION	AUTHORITY	DATE	SIGNATURE	SHEETS AFFECTED
A	Draft	01JUL82	R. Lemay	all
0	Ultimate	01JUL83		all
1	Ultimate	01JAN88		all

This revision is issued to reflect changes attendant in the redesign of the Custom Processor. The new functionality is summarized in Appendix A.

HONEYWELL BULL CONFIDENTIAL AND PROPRIETARY

TABLE OF CONTENTS

This document and the information contained herein are confidential to and the joint property of the Ultimate Corporation and of Honeywell Bull Corporation printed for the sole purpose of conducting their businesses. This document, any copy thereof and the information contained herein shall be maintained in strictest confidence; shall not be copied in whole or in part except as authorized by the employee's manager, and shall not be disclosed or distributed (a) to persons who are not Honeywell Bull or Ultimate employees, or (b) to Honeywell Bull or Ultimate employees for whom such information is not necessary in connection with the execution of their assigned duties. Upon request, or when the employee in possession of this document no longer has need for the document for authorized Honeywell Bull or Ultimate purposes, this document and any copies thereof shall be returned to the employee's manager. Deviation from this policy may only be authorized by a Honeywell Bull or Ultimate Vice President.

## TABLE OF CONTENTS

## SECTION ONE

1.0 INTRODUCTION	1-1
1.1 Purpose	1-1
1.2 Scope	1-1
1.3 Disclaimer	1-1
1.4 References	1-2

## SECTION TWO

2.0 OPERATIONAL OVERVIEW	2-1
2.1 System environment	2-1
2.2 Introductory description	2-3
2.2.1 the ralu	2-3
2.2.2 the aram	2-3
2.2.3 the dbus	2-3
2.2.4 the address bus	2-4
2.2.5 the zbus	2-4
2.2.6 the system bus	2-4
2.2.7 flags	2-4
2.2.8 the shifter	2-4
2.2.9 indicators	2-4
2.2.10 op registers	2-4
2.2.11 other registers	2-5
2.2.12 the clock	2-5
2.2.13 the stack	2-5
2.2.14 the next address generator	2-5
2.2.15 availability	2-5

## SECTION THREE

3.0 HARDWARE DESCRIPTION	3-1
3.1 Register file and alu	3-2
3.1.1 the 2901's	3-2
3.1.2 ralu addressing and control	3-2
3.1.3 ralu support logic	3-2
3.2 Auxiliary Random Access Memory	3-3
3.2.1 aram read or write	3-3
3.2.2 aram addressing	3-3
3.3 The D Bus	3-4
3.3.1 dbus byte x	3-4
3.3.2 dbus byte y	3-5
3.3.3 dbus byte z	3-6

3.4	The Address Bus	3-6
3.4.1	address bus data flow	3-7
3.4.2	address bus enables	3-7
3.5	The Z Bus	3-8
3.5.1	zbus sources	3-8
3.5.2	zbus enables	3-8
3.6	The System Bus	3-9
3.6.1	megabus cache addressing	3-9
3.6.2	megabus cache data storage	3-9
3.6.3	megabus cache procedure storage	3-9
3.6.3.1	the procedure bus	3-10
3.6.3.2	the procedure bus latches	3-10
3.6.4	megabus control circuitry	3-10
3.6.4.1	writes	3-11
3.6.4.2	reads	3-12
3.6.4.3	reads	3-10
3.6.4.3.1	cache/local memory reads	3-12
3.6.4.3.2	MEGABUS reads	3-12
3.6.5	megabus response circuits	3-13
3.7	Temporary and Permanent Flags	3-14
3.7.1	temporary flags	3-14
3.7.2	permanent flags	3-15
3.8	Nibble Shifter	3-15
3.8.1	shifter data flow	3-16
3.8.2	shifter control	3-16
3.9	Arithmetic and Miscellaneous Indicators	3-16
3.9.1	arithmetic indicators	3-16
3.9.2	miscellaneous indicators	3-17
3.9.2.1	scram indicator	3-17
3.9.2.2	difbuf indicator	3-17
3.10	The OP Register	3-17
3.10.1	the abcd registers	3-17
3.10.2	the opcode register	3-13
3.11	Loading Various Registers	3-17
3.11.1	loading the H register	3-18
3.11.2	loading the output register	3-18
3.11.3	loading adra and adrb	3-18
3.11.4	loading adrp and pctr	3-19
3.12	The Four Speed Clock	3-20
3.12.1	the basic clock	3-20
3.12.2	the gear shifter	3-20
3.12.3	clock stalls	3-21

3.13	The Return Stack	3-23
3.13.1	the return registers	3-23
3.13.2	the stack pointer	3-23
3.14	The Next Address Generator	3-23
3.14.1	bank selection	3-24
3.14.2	else-bank-next-address generation	3-25
3.14.3	if-bank-next-address generation	3-25
3.14.3.1	if bank address bits 01-09	3-25
3.14.3.2	if bank address bits 10-13	3-26
3.15	Availability Circuits	3-27
3.15.1	error detection circuits	3-27
3.15.1.1	procedure parity	3-28
3.15.1.2	data parity	3-28
3.15.1.3	procedure red	3-28
3.15.1.4	data red	3-28
3.15.1.5	procedure and data uar	3-29
3.15.1.6	procedure page cross detection	3-29
3.15.1.7	control store parity	3-29
3.15.2	parity generation circuits	3-29
3.15.3	verifying the integrity circuits	3-30
3.15.4	branch to zero	3-30
3.16	Alterable Firmware Array	3-30
3.16.1	loading the alterable firmware array	3-30
3.16.2	enabling the alterable firmware array	3-31

**SECTION FOUR**

4.0	FIRMWARE DESCRIPTION	4-1
4.1	2901 Control	4-3
4.2	Aram Control	4-6
4.2.1	rm	4-6
4.2.2	rw	4-6
4.3	D Bus Control	4-6
4.3.1	full literal	4-6
4.3.2	broadside	4-6
4.3.3	mixes	4-7
4.4	Address Bus Control	4-7
4.5	Z Bus Control	4-7
4.6	Megabus/Cache Control	4-8
4.7	Flag Control	4-10
4.7.1	permanent flags	4-11
4.7.2	temporary flags	4-11
4.8	Nibble Shifter Control	4-11

4.9 Indicator Control	4-11
4.9.1 arithmetic indicators	4-11
4.9.2 miscellaneous indicators	4-12
4.10 ABCD and OP Control	4-12
4.11 Load Controls	4-12
4.11.1 loading adra, adrb, or adrp/pctr	4-13
4.11.2 loading outr and/or shrg	4-13
4.12 Clock Control	4-13
4.13 Stack Control	4-13
4.14 Next Address Control	4-14
4.15 Availability	4-17

**SECTION FIVE**

5.0 THE NANOSECONDS	5-1
5.1 The Symbology	5-1
5.2 The Choices	5-2
5.3 Putting It Together	5-2

**APPENDIX A**

Enhancements	A-1
--------------	-----

LIST OF FIGURES

FIGURE 2-1A	SYSTEM ENVIRONMENT WITH CACHE	2-2
FIGURE 2-1B	SYSTEM ENVIRONMENT WITHOUT CACHE	2-2
FIGURE 2-2	PROCESSOR BLOCK DIAGRAM	2-6
FIGURE 3-1	IF AND ELSE BANK ADDRESS GENERATORS	3-20
FIGURE 4-1	FIRMWARE WORD FORMAT	4-2
FIGURE 5-1	PERT CHART	5-4

LIST OF TABLES

TABLE 3-1	RALU CARRY NETWORK	3-2
TABLE 3-2	ARAM ADDRESS SELECTOR	3-3
TABLE 3-3	DBUS BYTE X	3-4
TABLE 3-4	DBUS BYTE Y	3-5
TABLE 3-5	DBUS BYTE Z	3-6
TABLE 3-6	ADDRESS BUS	3-7
TABLE 3-7	ADDRESS REGISTER ENABLES	3-7
TABLE 3-8	ZBUS SOURCES	3-8
TABLE 3-9	ZBUS ENABLES	3-8
TABLE 3-10	SIDE-EFFECTS OF TEMPORARY FLAGS	3-15
TABLE 3-11	SIDE-EFFECTS OF PERMANENT FLAGS	3-15
TABLE 3-12	CONFIGURATION REGISTER INPUTS	3-19
TABLE 3-13	CONFIGURATION REGISTER OUTPUTS	3-19
TABLE 3-14	CLOCK SPEEDS	3-20
TABLE 3-15	STALL CONDITIONS	3-22
TABLE 3-16	ELSE BANK SOURCES	3-25
TABLE 3-17	IF BANK SOURCES	3-25
TABLE 3-18	IF BANK ADDRESS BITS 01-09	3-26
TABLE 3-19	IF BANK ADDRESS BITS 10-13 (PART 1)	3-26
TABLE 3-19	IF BANK ADDRESS BITS 10-13 (PART 2)	3-27
TABLE 3-20	CYCLE MODULATION OF FWLOAD	3-31

LIST OF TABLES (continued)

TABLE 4-1	THE FOURTEEN FIRMWARE ZONES	4-1
TABLE 4-2	2901 CONTROL FIELDS (PART 1)	4-3
TABLE 4-2	2901 CONTROL FIELDS (PART 2)	4-4
TABLE 4-2	2901 CONTROL FIELDS (PART 3)	4-5
TABLE 4-2	2901 CONTROL FIELDS (PART 4)	4-6
TABLE 4-3	DBUS LITERALS	4-7
TABLE 4-4	DBUS BROADSIDES	4-7
TABLE 4-5	DBUS MIXES	4-7
TABLE 4-6	ZBUS MICROS	4-8
TABLE 4-7	MEGABUS AND CACHE/LOCAL MEMORY (PART 1)	4-9
TABLE 4-7	MEGABUS AND CACHE/LOCAL MEMORY (PART 2)	4-10
TABLE 4-7	MEGABUS AND CACHE/LOCAL MEMORY (PART 3)	4-10
TABLE 4-7	MEGABUS AND CACHE/LOCAL MEMORY (PART 4)	4-11
TABLE 4-7	MEGABUS AND CACHE/LOCAL MEMORY (PART 5)	4-11
TABLE 4-7	MEGABUS AND CACHE/LOCAL MEMORY (PART 6)	4-12
TABLE 4-8	ARITHMETIC INDICATORS (PART 1)	4-13
TABLE 4-8	ARITHMETIC INDICATORS (PART 2)	4-14
TABLE 4-9	ABCD and OP MICROS	4-14
TABLE 4-10	LOAD MICROS (PART 1) ADDRESS REGISTERS	4-15
TABLE 4-10	LOAD MICROS (PART 2) OUTR SHRG and TIMER	4-15
TABLE 4-10	LOAD MICROS (PART 3) CNFG	4-15
TABLE 4-11	CONFIGURATION REGISTER	4-16
TABLE 4-12	CLOCK SPEED MICROS	4-16
TABLE 4-13	PUSH MICRO	4-17
TABLE 4-14	NEXT ADDRESS CONTROL FIELDS (PART 1)	4-17
TABLE 4-14	NEXT ADDRESS CONTROL FIELDS (PART 2)	4-18
TABLE 4-15	NEXT ADDRESS MECHANISMS (PART 1)	4-18
TABLE 4-15	NEXT ADDRESS MECHANISMS (PART 2)	4-19
TABLE 4-15	NEXT ADDRESS MECHANISMS (PART 3)	4-20
TABLE 4-16	FRAMIT ACTIONS	4-22



## INTRODUCTION

The "sixteen-bit" Custom Processor is a nine megahertz, twenty-four-bit wide, microprogrammable MEGABUS connected firmware engine driven by a ninety-six-bit wide control store word and having a blank identity.

### 1.1 PURPOSE

This technical description may be useful to those who wish to provide the Custom Processor with a new incarnation but it is specifically directed toward the test technician who needs to understand its inner workings. Those who attempt personalization of the Custom Processor are referred to the Specification of the 16-bit Custom Processor. For board testing or firmware checkout, Custom and Special Products offers a Firmware Development Facility which greatly simplifies the task.

### 1.2 SCOPE

This document is intended for the test technician. It (particularly section 3) describes the operation of the sixteen-bit Custom Processor at the level sufficient to perform component level fault isolation.

In addition to this section, this document contains four other sections.

Section two describes the system environment, an exposure of the Custom Processor capabilities and a brief discussion of each of its major areas.

Section three is a detailed description of each of the fifteen hardware areas.

Section four is a detailed description of each of the fifteen firmware areas.

Section five is a discussion of internal speed considerations.

### 1.3 DISCLAIMER

The firmware dictionary serves as the specification for the Custom Processor. The firmware dictionary shall govern in any disagreement between it and this technical description.

1.4 REFERENCES

In order to code firmware to execute on the CUP16, the following additional documents may prove useful:

- CUP16 logic block diagrams
  - for the mother board . . . . . 60156205
  - for the daughter board . . . . . 60156210
- Other related documents are:
  - 16-Bit Custom Processor Specification . . . . . 60165904
  - CUP16 Test procedures . . . . . 71220271
  - RTL6 assembly language manual . . . . . LDA-021

## OPERATIONAL OVERVIEW

This section describes the system environment into which the sixteen-bit Custom Processor may be connected. The section also gives a broad-brush description of the Custom Processor's inner workings.

### 2.1 System Environment

The sixteen-bit Custom Processor is comprised of a mother/daughter board pair which connects to the MEGABUS backplane (see figure 2-1). The MEGABUS backplane is a means of interconnecting system elements including peripheral controllers, communication controllers, processors, and memories. Dialogue among system elements is transacted on the MEGABUS backplane. Many kinds of transactions are allowed but two major categories are of interest, "writes" which transpire in one MEGABUS cycle, and "reads" which transpire in two MEGABUS cycles. The circuitry which supports the MEGABUS backplane is evenly distributed among all system elements such that each element performs its share of the priority resolution (tie-breaking); each element is responsible for determining that it is the intended recipient of a transaction; each element is responsible for remembering the identity of the requestor of a read type operation so that the response may be sent to its proper destination.

The Custom Processor is a system element which possesses a sixteen-bit-wide MEGABUS data interface but possesses a twenty-four-bit-wide MEGABUS address interface and internally contains a nine megahertz, twenty-four-bit-wide microprocessor driven by a ninety-six-bit-wide firmware word. Memory and I/O dialogue may emanate from the Custom Processor. Controllers may be directed to treat the Custom Processor as the recipient of their interrupts. By virtue of the 24-bit internal structure and the 24-bit MEGABUS address interface, the Custom Processor has access to all 16 megabytes of main memory. This allows the Custom Processor to access (and hide) memory not addressable by some level-6 processors.

An optional Cache or Local Memory may be connected to the Custom Processor in order to minimize MEGABUS traffic generated by the CUP16's voracious appetite. Applications which include the CUP16 are mostly configurations of cooperating heterogeneous system elements each dedicated to a specific function (e.g., the CUP16 is assigned to process the application, the Level-6 cpu is assigned to manage the data base).

The system configuration should include at least one Level 6 processor, if for no other reason than to allow test software to be loaded and executed. This is the most effective way of allowing the user to verify the integrity of all device and communication controllers.

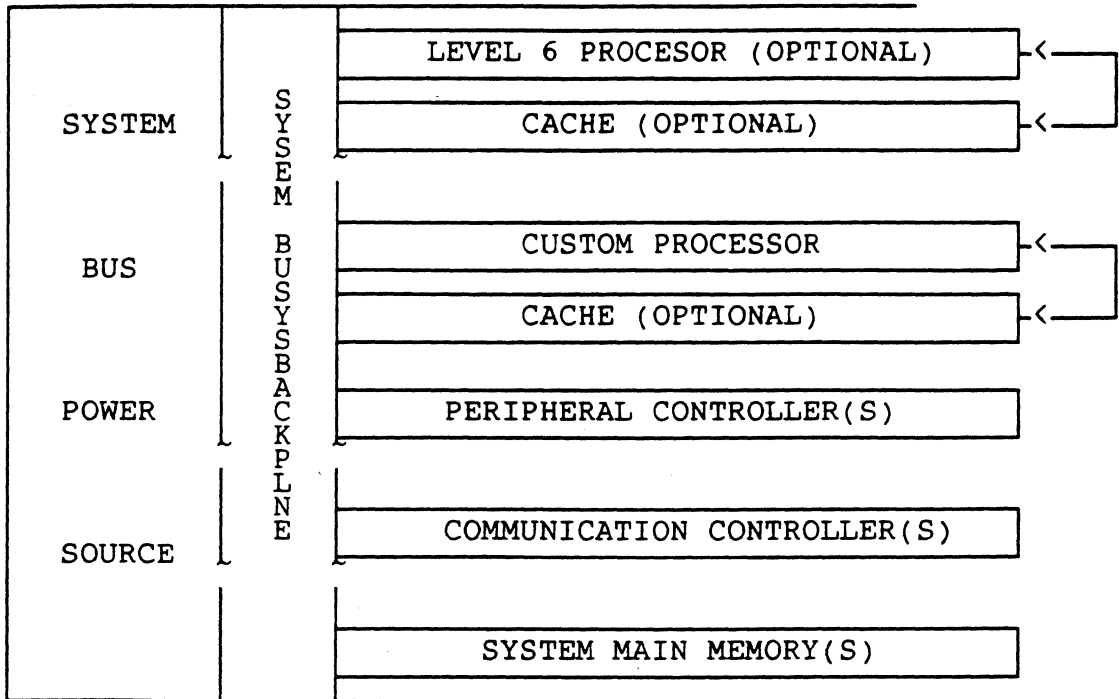


FIGURE 2-1A SYSTEM ENVIRONMENT WITH CACHE

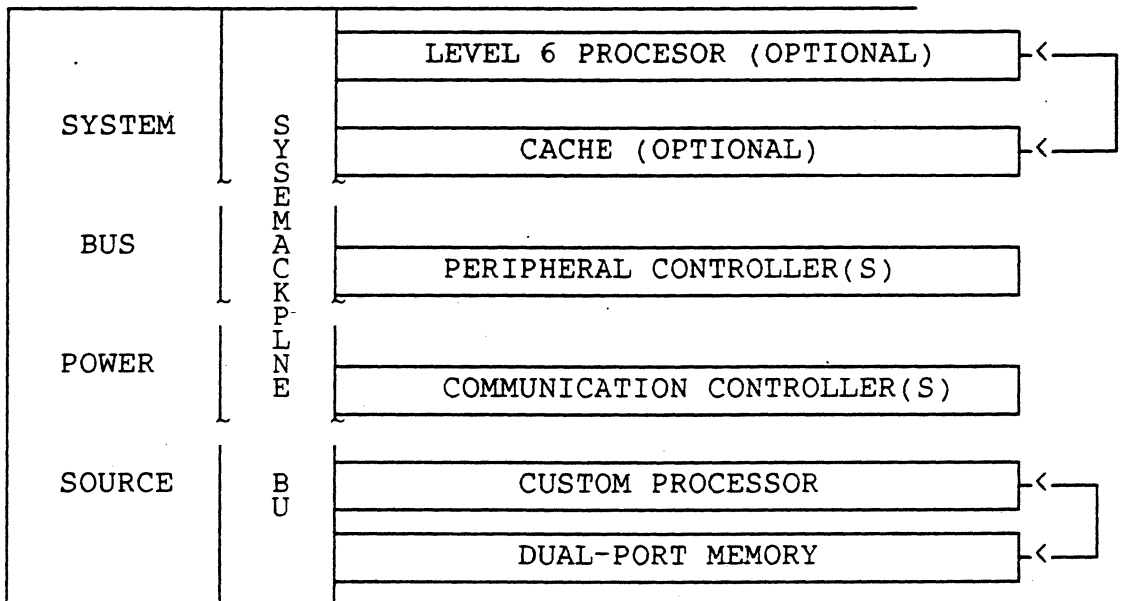


FIGURE 2-1B SYSTEM ENVIRONMENT WITH LOCAL MEMORY

## 2.2 INTRODUCTORY DESCRIPTION

The major block diagram of figure 2-2 is a representation of the Custom Processor. This section is divided into fifteen paragraphs each of which discusses the topic at a first level. The discussions in sections three and four are similarly subdivided allowing the reader quick reference to greater hardware detail (section three) or greater firmware detail (section four). The fifteen sections are listed below, each with its associated block diagram identifier:

- I. the ralu
- II. the aram
- III. the d bus
- IV. the address bus
- V. the z bus
- VI. the system bus
- VII. flags
- VIII. shifter
- IX. indicators
- X. op code registers
- XI. other registers
- XII. the clock
- XIII. the stack
- XIV. the next address generator
- XV. availability

### 2.2.1 the ralu (see block diagram identifier I)

The register file and alu is comprised of six 2901 bit slice chips constituting a twenty-four-bit alu, a sixteen-location dual-ported random access memory and a bit shifter (24 or 48 bits wide). Operations inside the alu occur at nearly a nine megahertz rate. The ralu receives external twenty-four-bit data from the dbus and transmits twenty-four-bit results to the zbus.

### 2.2.2 the aram (see block diagram identifier II)

The auxiliary random access memory is a single-ported 16-location memory. Each location contains twenty-four bits. The memory may be addressed in any one of four ways. Its data is read onto the dbus. Data written into the aram is taken from the zbus.

### 2.2.3 the dbus (see block diagram identifier III)

The dbus is the place where all roads lead. It is a twenty-four bit bus which receives literal data, or data from the address bus or the pbus or numerous other secondary sources. It may deliver its wealth to the ralu or to the output register. It has byte partitioning capabilities; i.e., it may take combinations of literal byte(s), SHRG byte(s) and shifted zbus byte(s). As an example, the dbus may receive an all one's literal for its eight most significant bits, the middle byte of SHRG for its eight middle bits, and any byte of the zbus for its eight least significant bits.

**2.2.4 the address bus (see block diagram identifier IV)**

The address bus provides addresses sent to the system bus. It receives data from four sources; the three local bus address registers (ADRA, ADRB, and ADRP), and a register which captures the output of the MEGABUS address receivers (RUPT). It may deliver the content of any one of these four address registers to the dbus.

**2.2.5 the zbus (see block diagram identifier V)**

If the dbus is where all roads lead, the zbus is quite uninspiring. Indeed, it is a journey person bus, capable of receiving data from the outside world (ie, the memory subsystem) and receiving ralu revelations and sending all this to the dbus via the nibble shifter and/or to the ARAM and/or to any one of the address registers.

**2.2.6 the system bus (see block diagram identifier VI)**

The system bus is the area of the processor responsible for communicating with non-memory system elements; e.g., a level-6 processor or a level-6 controller and with locally connected or MEGABUS connected memories. The system bus area contains, along with the interface circuits required to carry on a dialogue with other system elements, an eight-byte look-ahead procedure buffer and two two-byte data buffers. The two data buffers receive information from the system bus (memory or I/O data) to be deposited onto the zbus. The look-ahead procedure buffer is organized to supply one byte to the op register and/or the dbus and to replenish itself automatically as bytes are consumed.

**2.2.7 flags (see block diagram identifier VII)**

The Custom Processor contains sixteen firmware settable and testable flops. Some have hardware dedicated functions (e.g., processor off-line); others have more sophisticated firmware sequence control characteristics (e.g., they may participate in sixteen-way "splatters").

**2.2.8 shifter (see block diagram identifier VIII)**

The shifter is a nibble rotator connected between the zbus and the dbus. The H register (SHRG) also receives the shifter output. All six possible 4-bit shift distances are supported; e.g., shift left four nibbles, shift right two nibbles are two equal operations.

**2.2.9 indicators (see block diagram identifier IX)**

Indicators are storage elements which remember some property of the results obtained in one firmware step so that they may affect the firmware sequence later on. There are arithmetic indicators like "zero", "carry" and "overflow"; there are more specialized indicators like the frame bound detector. Many have the ability to participate in "splatters" which permit up to 17-way firmware branches.

**2.2.10 op registers (see block diagram identifier X)**

The op registers permit capturing interesting nibbles of the procedure stream for future reference. The storage mechanisms involved are a register called OPCOD, and four registers called RAA, RAB, RAC, and RAD. OP captures eight bits of the procedure for sixteen-way "splatters". RAA, RAB, and RAC each capture four bits of the procedure bus, while RAD is an incrementing register which is loaded from the nibble shifter.

**2.2.11 other registers (see block diagram identifier XI)**

Other registers provide strategically located information storage. Address registers for procedure and data fetches are loaded from the zbus and communicate their content to the system bus. A register (PCTR) is provided to keep track of the address of the next procedure byte. The output data register is loaded from the dbus and communicates its content to the system bus. The H register, one of the byte partitionable sources to the dbus, captures the output of the nibble shifter.

**2.2.12 clock (see block diagram identifier XII)**

The clock has a maximum frequency of nine megahertz. It is an asynchronous mechanism whose speed for each step is selected by the firmware assembler. The clock is structured to wait before starting the next step (stall) if the coder wishes, implicitly or explicitly, to postpone its start until an external event occurs (such as receiving previously requested memory subsystem data).

**2.2.13 the stack (see block diagram identifier XIII)**

The return stack is a mechanism which simplifies the use of subroutines. It contains two levels. Absolute addresses are "pushed" onto the stack. Unconditional, conditional and alternate (masked) returns are provided.

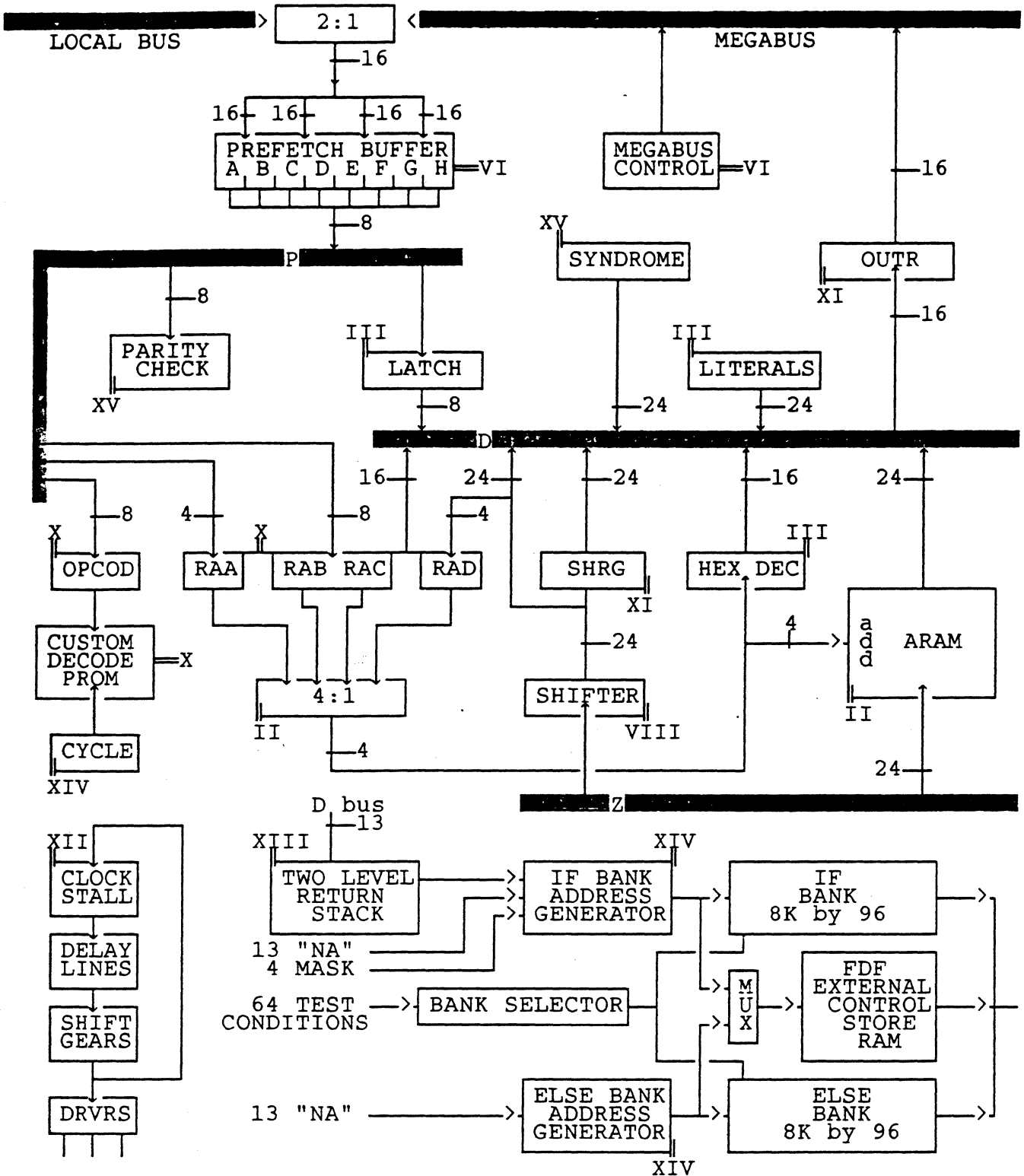
**2.2.14 the next address generator (see block diagram identifier XIV)**

The next address generator is a particularly flexible element in that it eliminates the need for numeric sequentiality in the execution of firmware steps. The next address may be any one of the 16384 locations provided. The destination may be specified as a GO-TO or it may be chosen from a pair of addresses dependent upon one of sixty-four test conditions. It may instead be chosen among sixteen locations dependent upon some group of four indicators (five groups are provided) or the destination may be a seventeenth location dependent upon one of the sixty-four test conditions. Then again, it may be chosen from among one of 16 locations dependent upon a byte of the procedure stream (OP) via a table look-up mechanism containing eight look-up tables. Or, it may be a subroutine return. Four types of returns are provided: an unconditional return, or one which returns if a specified one of the sixty-four test conditions is true, or one which returns to an alternate return location as a function of subroutine processing discoveries, or one which conditionally performs an alternate return.

**2.2.15 availability**

The Custom Processor is possessed of data parity checking circuits, data uncorrectable memory edac error detectors, firmware parity error detectors, and detectors for references to unavailable system resources. Parity bits accompany data sent from the Custom Processor to other system elements allowing them the opportunity of verifying the integrity of the received information.

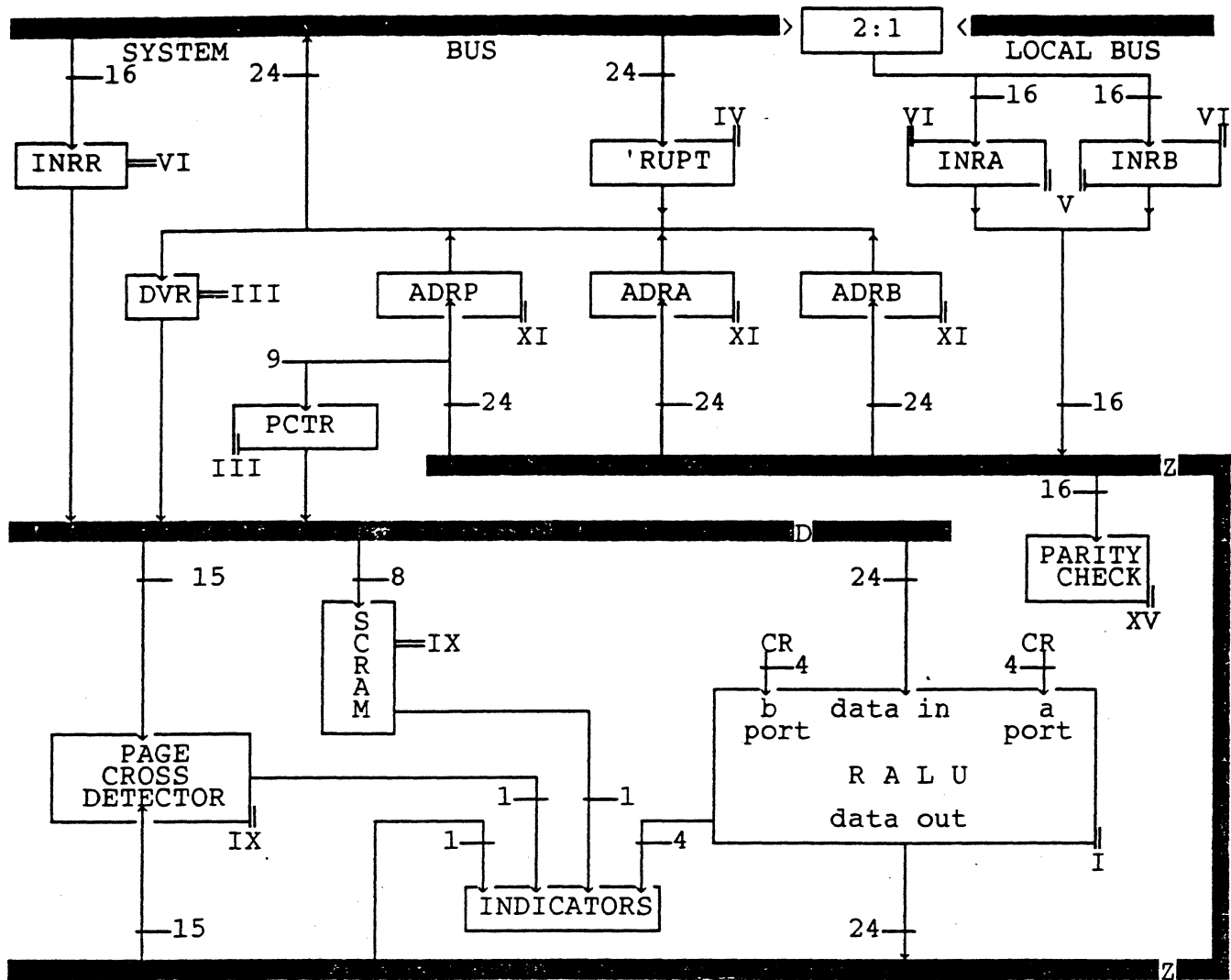
All Custom Processors include an imbedded comprehensive self-test firmware routine called the quality logic test (QLT) which, at every system initialization, exercises all processor hardware elements verifying their specified operation. This firmware routine may optionally include a thorough memory array test.



11AUG 87

BLOCK DIAGRAM OF THE





- CONTROL REGISTER FOR:
- clocks
  - next address
  - system bus
  - bus D and Z
  - address registers
  - RALU
  - ARAM
  - SHRG/shifter
  - flags
  - indicators

LEGEND	
┆──┆	data width
══	text references

FIGURE 2-2

16-BIT CUSTOM PROCESSOR

## HARDWARE DESCRIPTION

This section describes hardware entities at a level of detail sufficient for comprehension if the reader has a set of Custom Processor logic block diagrams (lbd) and if the reader has experience in the interpretation of logic diagrams. Lbd page references are suffixed with m for mother board or d for daughter board. Reference is also made to the block diagram of 2-2.

The hardware discussion is subdivided into sixteen zones. Each of these zones is a separable entity that has one or more firmware fields dedicated to its control as shall be seen in section four.

The fifteen sections and the block diagram identifier are:

- I. Register file and alu (2901's)
- II. Auxiliary random access memory (ARAM)
- III. D bus
- IV. Address bus
- V. Z bus
- VI. System bus
- VII. Temporary and permanent flags
- VIII. Nibble shifter
- IX. Arithmetic and miscellaneous indicators
- X. OP registers
- XI. Loading of H and other registers
- XII. Four-speed clock
- XIII. Return stack
- XIV. Next address generation
- XV. Availability
- XVI. Alterable Firmware Array

NOTE: In the logic descriptions that follow, formal signal names are capitalized and take one of three forms:

- o six characters; e.g., OLDMTY
- o nine characters with a neutral polarity indicator; e.g., NAIB06.EX
- o concatenations; e.g., BUSP00-07 (the pbus)  
PTAKE4/2/1 (the take counter)

Polarity indicators are avoided. If a description claims signal ABCDEF to be on, left unstated is that ABCDEF+ (if any) is "high" and ABCDEF- (if any) is "low".

3.1 Register file and alu (block diagram identifier I)

The ralu is the resource which performs twenty-four-bit arithmetic and logic operations. It resides between the dbus, from which it receives operands, and the zbus to which it delivers results.

3.1.1 The 2901's (see page 12d)

The ralu is comprised of six 2901's. These six chips constitute:

1. a dual-ported 16-location by 24-bit register file
2. a 24-bit arithmetic and logic unit
3. a 24-bit q register
4. a shifting element capable of shifting the alu output one bit left or right.
5. a shifting element capable of shifting the concatenation of the alu output and the q register one bit left or right.
6. a zero detector, an overflow detector, a sign detector and a carry detector.

3.1.2 Ralu addressing and control (see lbd pages 3d, 4d and 5d)

The raw control register provides the 2901's almost all of their control inputs:

- A port address           CRAA(00-03)
- B port address           CRAB(00-03)
- source select            CRAS(00-02)
- function select          CRAF(00-02)
- destination select      CRAD(00-02)

3.1.3 Ralu support logic (see lbd page 12d)

The carry look-ahead network is comprised of two 74S182 carry look-ahead chips connected in a conventional manner. First, the carry into the least significant 2901 (AUC032) is derived by decoding bits three and four of the AF field to produce four cases as shown in Table 31-1.

TABLE 3-1  
RALU CARRY NETWORK

CRAF03	CRAF04	AUC032
0	0	cause a carry into alu unconditionally
0	1	cause a carry into the alu if the carry indicator is off
1	0	cause a carry into the alu if the carry indicator is on
1	1	do not cause a carry into the alu

Next, the carry into the second, third and fourth 2901's (AUCO28, AUCO24, and AUCO20) is from the first 74S182. Last, the carry into the fifth (AUCO16) and sixth (AUCO12) 2901's (the most significant stages) are generated by the second 74S182 which receives inputs from the original carry-in, the propagate/generate outputs of the most significant two 2901's and the propagate/generate outputs of the other 74S182 carry look-ahead chip.

The zero detector of the 2901 is an open-collector output. Six 2901 zero-detect outputs are connected together in an open-collector network "terminated" by a 430-ohm resistor.

The bit-shift pins of the 2901's are connected in a conventional manner, where AUSR07 allows the value of temp flag 1 to be injected into alu right shifts and AUSQ31 allows the value of temp flag 1 to be injected into q register left shifts.

**3.2 Auxiliary Random Access Memory - ARAM (block diagram indentifier II)**

The Auxiliary Random Access Memory is comprised of six 74S189's. Each chip contains 16 locations of four-bit-wide static ram. The ARAM receives a four-bit address from a four-way selector called RAMAD. In any step, the ARAM may be read onto the dbus or written from the zbus but not both.

**3.2.1 ARAM read or write (see lbd page 16d)**

when written, the ARAM captures 24 bits from the zbus into the addressed location. When read, 24 bits of data are placed upon the dbus. ARAMEN is the chip enable. It comes on when reading the ARAM (CRRMRD) or when writing the ARAM (ARAMWR). The d input of the flop CRRMRD is a predecode of the output of the control store array, so that CRRMRD can be valid as early in the cycle as possible, affording the fastest access of the ARAM onto the dbus. ARAMWR is timed to occur during the second half of a step creating a "write pulse" which copies the zbus data into the addressed ARAM location through the end of the step. By the time the copying ends, the zbus has settled into validity.

**3.2.2 ARAM addressing (see page 15d)**

The two-bit firmware field RM provides control for a four-to-one selection of ARAM addressing sources. Two 74S153's perform the selection resulting in RAMAD(0-3). Table 3-2 shows what address results from each selection.

TABLE 3-2  
ARAM ADDRESS SELECTOR

CRRM(00,01)	RAMAD(0-3)
0	RAA00(0-3)
1	RAB00(0-3)
2	RAC00(0-3)
3	RAD00(0-3)

CRRM(00,01) are the control register flops which store the two-bit firmware field specified to control the ARAM addressing source. RAMAD(0-3) is the four-bit ARAM address. RAA00(0-3), RAB00(0-3), RAC00(0-3), and RAD00(0-3) are the four OP registers. A, B, and C store appropriate procedure nibbles and D stores a non-procedural nibble from the four least-significant bits of the nibble shifter.

3.3 The D Bus (block diagram identifier III)

The dbus is a major node in the 16-bit Custom Processor. It is one of the 24-bit buses and is the most prolific. Data may be deposited upon it from a wide variety of sources. Such data may then be made available for computation in the ralu, or placed into the output register, or used to address the stop code ram. The dbus is structured as three eight-bit buses, allowing up to three dbus sources to be combined (see section 4.3 for a complete list of the combinations and permutations).

3.3.1 dbus byte x - bits 08-15 (see lbd pages 13d, 16d, 17d, 31m, and 35m)

Byte x may receive from six mutually exclusive sources which are listed in Table 3-3 along with the signal that enables each source.

TABLE 3-3  
DBUS BYTE X

source	enable signal
1. the address bus (ADRS08-15)	ADRS2D
2. the h register (SHRG08-15)	SRLF2D
3. the nibble shifter (SHFT08-15)	SHLF2D
4. the ARAM (ARAM08-15)	ARAMEN
5. literal data (8*CRDB02)	LVL2D
6. the syndrome register (SYND08-15)	SYND2D

Source #1 is a collector of four sources, three of which store 24-bit addresses for MEGABUS references, and the other stores the MEGABUS address receivers when the CUP16 is the recipient of an unsolicited MEGABUS transaction (i.e., interrupt).

Source #2 is the register which is parked between the nibble shifter and the dbus i.e., it receives the output of the nibble shifter and can place the captured value onto the dbus in some subsequent step.

Source #3 is the nibble shifter capable of rotating the zbus value into the H register, onto the dbus, or both.

Source #4 is the ARAM placing its 24-bit "readout" of the addressed location onto the dbus.

Source #5 provides the dbus with 16-bit literals and a byte of "fill" (eight one's or eight zeros).

Source #6 deposits the syndrome register onto the dbus. The syndrome register captures the reason code for any hardware detected error.

Three of the sources are partitioned on byte boundaries; the shifter, the H register, or the literal may be selected for each dbus byte independently.

3.3.2 dbus byte y - bits 16-23 see lbd pages 13d, 15d, 18d, 19d  
31m, and 35m)

Byte y may receive from ten mutually exclusive sources which are listed in Table 3-4 along with the signal that enables each source.

TABLE 3-4  
DBUS BYTE Y

source	enable signal
1. the address bus (ADRS16-23)	ADRS2D
2. the h register (SHRG16-23)	SRMD2D
3. the nibble shifter (SHFT16-23)	SHMD2D
4. the ARAM (ARAM16-23)	ARAMEN
5. literal data (CRDC16-23)	CRDB00
6. the syndrome register (SYND16-23)	SYND2D.03
7. the ABCD register (RAA, RAB for byte y)	ABCD2D
8. the hex decoder (HEXD00-07)	HEXD2D
9. the interrupt register (INRR16-23)	INRR2D
10. the procedure counter (PCTR16-23)	PCTR2D

Sources #1 through #6 are equivalent to the six sources on byte x.

Source #7 allows access to the sixteen bits of the ABCD register.

Source #8 is one byte of the hex decoder, a mechanism which decodes, with a couple of 74S138's, the four bits of the ARAM address (RAMAD0-3) and emits (in conjunction with the z byte) a one in a field of fifteen zeros.

Source #9 allows the interrupt data register to be placed upon the dbus. The interrupt data register captures the MEGABUS data receivers when an interrupt is accepted by the Custom Processor.

Source #10 for this byte deposits seven zeroes and the most significant bit of the procedure counter, a register which tracks the address of the procedure byte next to be "taken" from the prefetch buffer. PCTR is accurate modulo 512.

3.3.3 dbus byte z - bits 24-31 (see lbd pages 13d, 15d, 20d, 21d, 31m, and 35m)

Byte z may receive from eleven mutually exclusive sources which are listed in Table 3-5 along with the signal that enables each source.

TABLE 3-5  
DBUS BYTE Z

source	enable signal
1. the address bus (ADRS24-31)	ADRS2D
2. the h register (SHRG24-31)	SRRT2D
3. the nibble shifter (SHFT24-31)	SHRT2D
4. the ARAM (ARAM24-31)	ARAMEN
5. literal data (CRDC24-31)	CRDB01
6. the syndrome register (SYND24-31)	SYND2D
7. the ABCD register (RAC, RAD for byte z)	ABCD2D
8. the hex decoder (HEXD08-15)	HEXD2D
9. the interrupt register (INRR24-31)	INRR2D
10. the procedure counter (PCTR24-31)	PCTR2D
11. the pbus (BUSP00-07)	PBUS2D

Sources #1 through #10 are equivalent to those on byte y.

Source #11 allows access to bytes of the procedure stream for computational purposes. Bytes and/or nibbles of the procedure stream are also accessible to the register ABCD and to the eight-bit register OPCOD.

3.4 The Address Bus (block diagram identifier IV)

The address bus is a collector of address sources. Its primary purpose is to supply twenty-four bit byte addresses to the MEGABUS system bus or to the Cache/Local Memory interfaces. A secondary purpose is to deliver the address bus's selected source to the dbus and another purpose is to deliver to the dbus the function code from an interrupting MEGABUS channel. Registers ADRA, ADRB, and ADRP contain addresses. The mechanism is optimized for memory references in that ADRA and ADRB access data whereas ADRP accesses procedure. Memory references may be either single or double pull, may read "lock" or "unlock", may write word or byte, or may write "unlock". ADRA and ADRB may also be loaded with channel-number/function-code information so that they may be used to communicate with other system elements. The CUP16 may, for instance, interrupt another processor, read a disk controller's status, or instruct a communication controller to perform a DMA transfer.

3.4.1 address bus data flow (ADRS08-31) (see lbd pages 29m and 30m)

The address bus may receive from four mutually exclusive sources which are listed in Table 3-6 along with the signal which enables each source.

TABLE 3-6  
ADDRESS BUS

source	enable signal
1. address register A (ADRA08-31)	ADRAEN
2. address register B (ADRB08-31)	ADRBEN
3. address register P (ADRP08-31)	ADRPEN
4. the interrupt register (RUPT08-31)	RUPTEN

Regarding sources #1, #2, and #3, each address register is loaded from the zbus and may participate in a read/write transaction by supplying the twenty-four address bits.

Source #4 captures the CUP16's channel number and the function code when another MEGABUS system element directs an unsolicited MEGABUS transaction toward the CUP16. By virtue of this data path (which allows the dbus to receive this information via the address bus) and the MEGABUS data lines from INRR, the microcoder may determine the nature of the message that the interruptor is attempting to transmit (by examining RUPT for the function and INRR for the accompanying data).

3.4.2 address bus enables (see lbd pages 18m, 29m and 33m)

Two bits of the "MEGABUS" control field (CSMG68,69) determine which of the four address-bus sources will be chosen. These two control store array bits are predecoded by a 74S139 forming the four select signals for the next firmware step. Before these signals are used, they must pass another test; as they arrive on the input of a 74AS1843 latch, the latch's gate input (MYDCNP) decides, based upon the current state of the MEGABUS interface, whether to pass these select signals through or ignore them. If either interface is currently "busy", then these select signals are ignored; if both interfaces are not busy, then these select signals proceed to their destination and cause the address bus to receive the appropriate address register. This mechanism guards against changing the address value presented to the MEGABUS or Cache/Local Memory address drivers while the MEGABUS transaction is still in process. The two-bit field results in one of four selections as listed in Table 3-7.

TABLE 3-7  
ADDRESS REGISTER ENABLES

CSMG68-69	predecode	"FR"input	select signal
00	CERPTE	CRRPTE	RUPTEN
01	CEADPE	CRADPE	ADRPEN
10	CEADAE	CRADAE	ADRAEN
11	CEADBE	CRADBE	ADRBEN



3.5 The Z Bus (block diagram identifier V)

The zbus is the bus where the ralu may deposit its computations, is the bus upon which external data arrives (from inra/b) and is the bus which feeds the nibble shifter.

3.5.1 zbus sources (see lbd pages 16d-21d, and 22m)

The zbus is twenty-four bits wide and receives data from two twenty-four-bit-wide sources and two sixteen-bit-wide sources. Along with the ralu (24 bits) and the input data registers (16 bits), the zbus may also receive from an external 24-bit-wide connection (i.e., the firmware development facility). The four sources are listed in Table 3-8 along with their respective enable signals.

TABLE 3-8  
ZBUS SOURCES

source	enable signal
1. the inra (INRA16-31)	CRIA2Z
2. the inrb (INRB16-31)	CRIB2Z
3. the ralu (ALUY08-15) the ralu (ALUY16-31)	CRTB2Z CRAY2Z
4. the fdf (BUSZ08-31.ex)	CRTB2Z

3.5.2 zbus enables (see lbd page 9d)

To achieve the earliest zbus "valid", the two control store array bits of the ZB field are predecoded by a 74S139 into four early enable signals which are then captured in four control register flops. The two-bit ZB field (CSZB48,49), the zbus source, the early enable and the final enable are shown in Table 3-9.

TABLE 3-9  
ZBUS ENABLES

CSZB(48,49)	source	early enable	final enable
00	ralu	CEAY2Z	CRAY2Z
01	fdf	CETB2Z	CRTB2Z
10	inra	CEIA2Z	CRIA2Z
11	inrb	CEIB2Z	CRIB2Z

When the sixteen bits of inra or inrb are placed upon the zbus as a result of code two or three, they are accompanied by eight bits of the alu. Also in codes two or three, a parity check is performed to verify the integrity of the data received from the Cache/Local Memory or MEGABUS receivers (see 3.15).

### 3.6 The System Bus and Local Bus Interfaces (block diagram identifier VI)

This interface hardware can be divided into four sections:

- o An addressing mechanism
- o A data storage mechanism
- o A procedure storage mechanism and
- o Control circuitry

The MG field is responsible for managing the resources represented by the above list which contains three address registers, 64 bits of procedure stream storage, and 32 bits of data storage.

#### 3.6.1 MEGABUS and Cache/Local Memory addressing (see lbd pages 29m and 30m)

Addresses from the processor to the MEGABUS (or Cache/Local Memory) may emanate from one of four sources: adra, adrb, adrp or rupt. Three of the four registers are comprised of 74F374's whose outputs are tied together to form the address bus in conjunction with a set of 74F241 drivers allowing the fourth register (adrp) to join in. This forms a four-to-one mux selected by ADRAEN, ADRBEN, ADRPEN and RUPTEN. The resultant address bus is connected to the address transmitters (26S10's) providing the address for all MEGABUS transactions including the non-memory-reference variety. This address is also sent to the Cache/Local Memory. This same resultant address bus is connected to the dbus by 74F241 drivers enabled by the signal ADRS2D.

#### 3.6.2 MEGABUS and Cache/Local Memory data storage (see lbd pages 1m and 22m)

Data from the MEGABUS or from the Cache/Local Memory is captured in either inra or inrb. A group of four 74F257's select between Cache/Local Memory and MEGABUS data as a function of CADONG. Cache/Local Memory data is selected only if the outstanding data reference is being processed by the Cache/Local Memory. The data on the output of the selector is captured by inra if the MEGABUS or Cache/Local Memory reference in process was requested for inra as signified by INRACK making a "leading edge". If INRBCK makes a leading edge, then the data is destined for inrb. Both inra and inrb deliver their data to the zbus as described in 3.5.

#### 3.6.3 MEGABUS and Cache/Local Memory procedure storage (see lbd pages 26m and 27m)

Procedure arrives in sixteen-bit waves in the same manner as data arrives; namely, a selection of Cache/Local Memory versus MEGABUS is performed by four 74F257's controlled by CADONG. The selector output (DATA00-15) is directed into the sixty-four-bit procedure buffer as a function of a two-bit counter (PRASK0, PRASK1) which is responsible for remembering where the next sixteen-bits of procedure are to be placed. In the beginning, PRASK0 and PRASK1 are off. When the first sixteen bits arrive, PRASK1 sets, clocking DATA(00-07) into INRPA(0-7) and DATA(08-15) into INRPB(0-7). When the next sixteen bits of procedure arrive, PRASK0 sets, clocking DATA(00-07) into INRPC(0-7) and DATA(08-15) into INRPD(0-7). When the third sixteen bits of procedure arrive, PRASK1 goes off, clocking DATA(00-07) into INRPE(0-7) and DATA(08-15) into INRPF(0-7). When the fourth sixteen-bit procedure word arrives, PRASK0 goes off, clocking DATA(00-07) into INRPG(0-7) and DATA(08-15) into INRPH(0-7). Now

PRASK0 and PRASK1 are back to their original state (both off) such that when the fifth sixteen-bit word arrives, INRPA and INRPB each receive a new byte of procedure. As long as procedure continues to be fetched from consecutive memory locations, the eight-byte buffer is treated as a circular storage mechanism.

### 3.6.3.1 The procedure bus

In order to dispense one byte at a time, the eight 74F374 chips which form this eight-byte storage register have their outputs connected together to form an eight-bit bus. The output of this network is called BUSP00-07. A three-bit counter, called the take counter, keeps track of which byte is next for delivery onto BUSP00-07. When PTAKE4/2/1=0, the first 74S374 chip (INRPA0-7) is enabled onto BUSP00-07, when PTAKE4/2/1=1, then INRPB0-7 is enabled onto BUSP00-07, ..., and when PTAKE4/2/1=7, then INRPH0-7 is enabled onto BUSP00-07.

Decoding the take counter is accomplished with a 74F138. The take counter itself is comprised of a 74S163 counter chip which, with the help of CDPTAK, increments by zero or one each firmware step, recording the removal of procedure bytes. Each time the PTAKE counter increments, it signifies the removal of a procedure byte and therefore a one-byte vacancy.

### 3.6.3.2 The procedure bus latches (see lbd pages 20d and 36m)

The procedure bus generally heads in four directions: the dbus, the OP registers (RAA, RAB, and RAC), the OPCOD register, and a next-address splatter input. The dbus and the splatter input require the pbus information to be valid until the end of the firmware step. Each has a latch dedicated to that purpose. The pbus to dbus latch is BUSPD(0-7) and the splatter latch is BUSPS(0-3).

### 3.6.4 Interface control circuitry (see lbd pages 9m, 12m, 18m, and 33m)

Sometime during the middle of each firmware step, the address of the next firmware step is determined by mechanisms described in 3.14. Sometime near the end of each firmware step, the output of the control-store array (the next firmware word) becomes valid. The MG field emanates from the control-store array at bits 67-71 (CSMG67-71) and is captured in five control-register flops (CRMG00-04). Processor timing is such that CSMG67-71 are valid at least 15ns before the end of each step allowing certain decisions to be made before the next step actually begins. Two decisions are made which are related:

1. Does the action in the upcoming step require that a request to the MEGABUS or Cache/Local Memory be initiated?
2. Does the action in the upcoming step require that the present step be delayed in terminating?

If, for any reason, a transaction between the processor and an external system element is in process, the signal MGBUSY is on. As an illustration of the two decisions above, suppose a firmware sequence is encountered which calls for two memory writes in two consecutive firmware steps. As the first step nears completion, CSMG67-71 alerts the request logic contained in the PAL (BUSNOW, LOCNOW) that the next step will initiate (another) transaction but, since only one transaction may proceed at a time and since the first memory write is still in process, this step must be delayed in terminating (stalled). Thus, MGBUSY and CSMG67,68 will stall the clock until the first write is complete (see 3.12 for more details about clock stalls). MYASKK/CAHREQ get a chance at the beginning of every step and are, of

course, structured to assume that whatever stall was required, has occurred and has been released. When a Local Memory is connected to the local port (rather than a Cache), writes may be directed to it rather than the MEGABUS. When the first write was sent to the Local Memory, MGBUSY remains true until CAHREQ relaxes whereas if the first write was directed at the MEGABUS, MGBUSY remains true until MYASKK relaxes.

#### 3.6.4.1 Writes via the MEGABUS

When performing a memory write or an I/O "output", the sequence is as follows:

1. MYASKK comes on at the beginning of any step which initiates a memory write.
2. as soon as the MEGABUS tie-breaker becomes not busy (BSBUSY), MYREQT comes on.
3. the CUP16 will now participate in the next tie-break. Typically, about 75ns are required to "fetch" the bus. If the tie-break was triggered by a lower priority MEGABUS element, the CUP16 could win in 24ns! Because the CUP16 is designed for the typical case, its address and data (and parity) signals are not ready before the 70ns point in the initiating step. When MEGABUS access is granted to the CUP16, all of the inputs to MYDCNS will be satisfied save one (MYDCNP) which will activate only when the internal information is valid.
4. MYDCNN sets enabling the (26S10) address and data transmitters.
5. The MEGABUS element to which this transaction is directed, signals BSACKR, acknowledging the responsibility for completing the request (e.g., writing the memory location).
6. BSACKR clears MYASKK (MYDCNN and MYREQT as well) which causes MGBUSY to go off, releasing a stall which may have been waiting for the termination of the write.

Note the distinction among REQNOW, MYASKK/MYREQT. REQNOW is synchronized to the CUP clock and is on during MEGABUS (and Cache/Local Memory initiation steps) only. MYASKK and MYREQT are synchronized to the MEGABUS timing; MYASKK clears at the trailing edge of the MYDCNN which clears MYREQT signifying that the transaction has been accepted by the other party.

Once any request is initiated, the firmware sequence is permitted to go on its merry way, ignoring the MEGABUS and Cache/Local Memory interfaces until such time as a resynchronization point is encountered (e.g., a data stall, a new MEGABUS or Cache/Local Memory transaction initiation). During this interval, the transaction is remembered in three storage elements:

1. the control information is in the "fred" register (FRSHBC, FRLOCK, FRMREF, FRWRIT and FRDBPL).
2. the twenty-four-bit address is "frozen" on the address bus by ADRPEN, ADRAEN, ADRBEN, and RUPTEN.
3. the data to be written (if any) is in outr (OUTR00-15).

### 3.6.4.2 Writes to the Local Memory

When performing a memory write to the Local Memory, the sequence is as follows:

1. LOCNOW signifies that a Cache/Local Memory operation is to be initiated (i.e., the Local Memory is connected and On-line) and FWRITE signifies that the operation is to be a write.
2. CAHREQ signals the request to the Local Memory. FRBDWR sends OUTR to the Local Memory's data lines.
3. The Local Memory captures the entire transaction at its first idle moment and signals CYCADN which clears CAHREQ signifying the completion of the write operation.

### 3.6.4.3 Reads

Reads are also complicated by the existence of the Cache/Local Memory. If the Cache/Local Memory is not present or is off-line (CACHON is off), all reads are directed to the MEGABUS. If the Cache/Local Memory is present and on-line, then memory reads which do not attempt a "lock" (FRLOCK) are directed through the Cache/Local Memory. All other reads (locks or I/O) are directed through the MEGABUS path. For any read, REQNOW causes SETDCP which, with the help of a PAL decoding the MG and FL fields, emits five control signals captured by the "fred" register and emits three signals which signify what kind of information is being read. CPDUES means procedure is being requested, CDDUES means that data is being requested and CADUES means that the data being requested is to be placed into inra (when it arrives). The arrival of information from either source is signalled by MYSHRP for procedure and MYSHRD for data.

#### 3.6.4.3.1 Cache/Local Memory reads

LOCNOW emanates from a PAL and with proper timing, causes CAHREQ. Implied by LOCNOW are both the firmware bit signifying a Cache/Local Memory request (CSMG73) and the Cache/Local Memory On-Line signal. The Cache/Local Memory request is not allowed to be true for read/lock or I/O read operations. When the Cache/Local Memory receives CAHREQ, it accesses the location specified by the address bus and returns sixteen bits (CADP00-15) and a data strobe (CYCADN). The Cache/Local Memory does not "double-pull" (FRDBPL is ignored), does not participate in lock transactions (FRLOCK and FRSHBC are off), and only performs memory references (FRMREF is on).

#### 3.6.4.3.2 MEGABUS reads

BUSNOW emanates from a PAL, sets MYASKK which causes MYREQT and later, after MYDCNP has waited for the dust to settle, provides MYDCNN; the appropriate "due" flop is set; all as previously described.

When a double-pull procedure read is requested, the memory may or may not send two transfers; if BSDBPL accompanies the first sixteen-bits, the memory promises to send another sixteen-bits. PRCDUE clears at the arrival of the last sixteen-bits.

When a double-pull data read is requested, DATDUE clears similarly when the last sixteen-bits arrive distinguishing the two cases as above. Note that when DATDUE is on and FRADUE is off, it means that sixteen bits are due for delivery into inrb.

If the reference does not succeed, either because it encountered a time-out or because the reference was nak'ed (BSNAKR), the due flops are cleared by DUECLR.

### 3.6.5 MEGABUS response circuits (see lbd page 8m)

When MEGABUS cycles are directed to the CUP, channel recognition logic activates the response circuitry. The response circuitry generates MYACKR or MYNAKR and sends the response to the MEGABUS element which initiated the transaction. The response circuitry also creates timing signals which allow the capturing of information pertinent to the transaction. A PAL receives information about the MEGABUS cycle as described below and creates signals captured in flops "at the middle of the MEGABUS cycle:

1. The channel recognition logic comprises a 74F521 (MYCHAN.0A) eight-bit comparator, two 74S86's (MYCHAN.0B,0C) one-bit comparators, and a 74S260 (MYCHAN.00) which makes the total determination that the channel number associated with a MEGABUS cycle matches the channel-number switches (MYCH00-09).
2. Integrity information (BSAPOK and BSDPOK) which, with the help of 74AS280 parity checkers signify that the MEGABUS address, data and command leads agree with the transmitted parity bits.
3. "State Machine" information indicating whether "Second-Half data is expected (PRCDUE, DATDUE, FRADUE), whether the CUP is now processing an interrupt (RPSYNC), whether the CUP is fully ON-LINE (P6SYNC) and whether the CUP is operating in "Force-Error" mode (FRCERR).
4. A stimulus which indicates that the Function-Code 01 feature is enabled and that a function code 01 bus cycle addressed to the CUP is now occurring having a "one" in its most-significant data bit (BSDT00).

The eight outputs of this PAL provide the D input to flops each of which is responsible for an aspect of either providing a proper MEGABUS response or of capturing data into an appropriate storage register:

MBSHRP	activates MYSHRP when procedure is arriving via the MEGABUS (LMSHRP activates MYSHRP when procedure is arriving from the Local Memory)
MBSHRD	activates MYSHRD when data is arriving via the MEGABUS (LMSHRD activates MYSHRD when data is arriving from the Local Memory)
MBINRA	activates INRACK clocking 16 data bits into INRA when that data is arriving via the MEGABUS (LMINRA activates INRACK when the data is arriving from the Local Memory)
MBINRB	activates INRBCK clocking 16 data bits into INRB when that data is arriving via the MEGABUS (LMINRB activates INRBCK when the data is arriving from the Local Memory)
MBACKR	activates MYACKR when the CUP acknowledges a MEGABUS cycle whether receiving requested data or receiving an (unsolicited) interrupt cycle
MBNAKR	activates MYNAKR when the CUP refuses an interrupt MEGABUS cycle (because it is already processing one). TONAKR activates MYNAKR when the CUP is Channel zero and a bus cycle has not otherwise receive a response for 5 microseconds.
MBINTR	activates MYINTR when the CUP accepts an interrupt bus cycle. MBINTR captures into ADRx the MEGABUS address leads associated with the interrupt cycle and captures into INRX the data lead associated with the interrupt cycle.
MBINIT	activates INITFF when a function-code 01 is accepted. INITFF causes the CUP to enter its QLT firmware sequence.

### 3.7 Temporary and Permanent Flags (block diagram identifier VII)

There are sixteen flags organized as two groups of eight. All sixteen are controlled by the FL firmware field.

#### 3.7.1 temporary flags (see lbd page 15d)

The temporary flags are controlled by the FL field (CRFL00-05). Seventeen of the sixty-four possible codes are devoted to the temporary flags; one to clear them all (and perform other functions), eight to set one of them at a time and eight to clear one of them at a time. The flops themselves are a 74LS259. One 74S138 output (GPINIT) determines when a broadside clear occurs, and a 74S10 (FLGTCK) determines when a one-bit change occurs. The polarity of the change is determined by the data input CRFL02, which flop is to change is determined by the select inputs CRFL03-05. The eight outputs are synchronized by a 74S374 and sent to eight test-condition inputs (see 3.14).

Some temporary flags have side-effects as shown in Table 3-10.

TABLE 3-10  
SIDE-EFFECTS OF TEMPORARY FLAGS

FLAGT0	16-way splatter (see 3.14)
FLAGT1	16-way splatter and shift-end-effect control
FLAGT2	16-way splatter
FLAGT3	16-way splatter
FLAGT4	undedicated
FLAGT5	undedicated
FLAGT6	data input to SCRAM
FLAGT7	undedicated

3.7.2 permanent flags (see lbd page 23m)

The permanent flags are also controlled by the FL field (CRFL00-05). Sixteen of the possible sixty-four codes are devoted to the permanent flags; eight to set one of them at a time, and eight to clear one of them at a time. A broadside clear occurs at master clear. The flops themselves are a 74S259. A 74S10 (FLGPCK) determines when a one-bit change occurs. The polarity of the change is determined mostly by CRFL02 overridden by the general error detector SYRCLK. Which flop is to change is determined by the select inputs CRFL03-05. The eight outputs are synchronized by a 74S374 and sent to eight test conditions (see 3.14). Permanent flag #5 controls how the CUP reacts to hardware detected errors; if FLGEP5 is off, the error reason code is captured in the syndrome and an unscheduled branch to firmware location zero occurs; if FLGEP5 is on, only the reason code is captured. Permanent flag #7 (FLAGP7) is one of the "break" (COFEBK) stimuli. Other permanent flags also have side-effects as shown in Table 3-11.

TABLE 3-11  
SIDE-EFFECTS OF PERMANENT FLAGS

FLAGP0	on line
FLAGP1	QLT failed (if FLAGP0=0)
FLAGP2	generate even parity to OUTR etc (if FLAGP0=0)
FLAGP3	undedicated
FLAGP4	undedicated
FLAGP5	inhibit disaster
FLAGP6	accept MEGABUS interrupts
FLAGP7	firmware "interrupter"

3.8 Nibble Shifter (block diagram identifier VIII)

The nibble shifter is the only connection between the zbus and the dbus and is therefore in a critical path. The shifter is really a six-nibble rotator; there is no distinction, for instance between shifting one nibble left and shifting five nibbles right.



### 3.8.1 shifter data flow (see lbd pages 13d and 14d)

The shifter is implemented with twelve 25S10's connected as three parallel groups of four. The first group is enabled when the shift distance prescribed by the SD field (CRSD00-02) is either zero, one, two, or three; the second group is enabled when the shift distance is either zero, one, four, or five; and the third group is enabled when the shift distance is two, three, four, or five. In each step, two of the three groups are enabled, producing twenty-four outputs.

The 25S10 chips whose output names are SHFT00.SA, SHFT04.SA, SHFT08.SA and SHFT12.SA reveals first that these chips are enabled for shift distances of 0, 1, 2, or 3 and that, for a distance of zero, each chip internally connects its four outputs to its lowest four inputs; for a shift distance of one, the four outputs are connected to the next higher four inputs; for a shift distance of two, to the next higher four inputs; and for a shift distance of three, to the top four data inputs.

The second group of four shifter chips (SHFTxx.SB) operates like the first, but is enabled for shift distances of zero, one, four, or five and thus their data inputs are biased eight bit positions.

The third group of four shifter chips operates like the other two, but is enabled for shift distances of two, three, four, or five and thus their data inputs are biased sixteen bit positions.

### 3.8.2 shifter control (see lbd pages 5d and 13d)

The three control register flops CRSD(00-02) are decoded by SHGCEN to enable the third group of shifter chips. The first and second group enables require no decoding.

## 3.9 Arithmetic and Miscellaneous Indicators (block diagram identifier IX)

There are six arithmetic indicators and two miscellaneous indicators. The intent of these indicators is to allow the coder to remember some characteristic(s) of some data for the purpose of affecting the addressing sequence of a subsequent firmware routine.

### 3.9.1 arithmetic indicators (see lbd pages 11d and 12d)

Five of the six arithmetic indicators derive their inputs from the output of the alu. The sixth samples the least-significant bit of the zbus (BUSZ31).

The overflow indicator (INOVFL) captures whether the carry into alu bit 08 was not equal to the carry out of alu bit 08 (AUOVFL).

The carry indicator (INCARY) captures whether a carry occurred out of alu bit 08 (AUCO08).

The sign indicator (INSIGN) captures whether the alu's most-significant bit was on (AUSIGN).

The zero indicator (INZERO) captures whether the alu output was zero (AUZERO).

Double zero (INZRDB) is derived from the alu being zero in this step and the zero indicator being previously on.

The indicators are captured at the behest of the FL field where either of two 74S138 outputs (I1BCLD or INDLD1) loads the six arithmetic indicators.

### 3.9.2 miscellaneous indicators (see lbd pages 11d and 15d)

There are two miscellaneous indicators. They are captured in a 74S174 by INCLK2 derived from a 74S138 output with pulse shaping.

#### 3.9.2.1 scram indicator

The stop code ram is a 256-location by one-bit-wide memory, addressed from byte y of the dbus. It is written from FLAGT6 under command of CRAMIT, a write pulse created from another output of the 74S138 properly timed by MCLKVL. The output data from the scram is exclusive-ored with FLAGT6 forming STOPCD, which is routed to the indicator INSCRM.

#### 3.9.2.2 difbuf indicator

This mechanism compares fifteen bits of the dbus with fifteen bits of the zbus, utilizing fifteen 74S86 exclusive-or circuits, three 74S260 circuits, and one 74S10.

### 3.10 The OP Register (block diagram identifier X)

Certain bytes and/or nibbles of the procedure stream must be stored for future reference. The three 4-bit wide registers RAA00(0-3), RAB00(0-3), RAC00(0-3), and the eight-bit wide register OPCOD(0-7) provide said storage. These four registers are loaded from the pbus (latch). The FL field controls the loading of RAA, RAB, RAC (and RAD), whereas the OP field (a dedicated bit) controls the loading of OPCOD.

#### 3.10.1 The ABCD registers (see lbd page 15d)

There is a 2918 for each of RAA (RAA000-3), RAB (RAB000-3), and RAC (RAC000-3). The associated register RAD (RAD000-3) is an incrementing counter useful in controlling loops. RAA, RAB, and RAC are loaded from the procedure bus in the middle of the firmware step, where RAA receives the least significant nibble (BUSP04-07) when the signal GPCLER occurs (generally at the beginning of instruction processing). RAB and RAC are also loaded from the procedure bus, where RAB receives the more significant nibble (BUSP00-03) and RAC receives the less significant nibble (BUSP04-07), both when the signal RABCK occurs. Thus three nibbles from the instruction stream may be saved for future reference, a four-bit counter can be loaded (RAD), and each may be used to address the ARAM.

#### 3.10.2 the OPCOD register (see lbd page 24d)

The OPCOD register captures the pbus byte (BUSP00-07) in the middle of the firmware step when the load signal OPCDCK occurs. The eight bits of OPCOD and three bits of the cycle counter (CYCLE4/2/1) are inputs to the "custom decode" table-look-up mechanism. Eight tables are provided, one for each value of CYCLE4/2/1. More detail on the use of the mechanism is provided in 3.14.

### 3.11 Loading Various Registers (block diagram identifier XI)

Many firmware fields have no other purpose than to control the loading of various registers. These fields are SR, LD, and OP. The FL field has a secondary justification for existence by providing the load controls to certain registers such as the indicators, ABCD, and the flags. The MG field has a secondary purpose in loading the Configuration Register (CNFG).

#### 3.11.1 loading the H register (see lbd pages 10d, 17d, 20d, 22d, 22d, 26d)

H, the SHRG register, is loaded at the behest of a dedicated control store bit (CSSR20) with its attendant control-register flop (CRSR00) pulse-formed in SHRGCK.

**3.11.2 loading the output register (see lbd pages 18m, 24m and 25m)**

The output register (OUTR00-15) is loaded at the behest of a control store bit (CSLD64) with its attendant control-register flop (CRLD00) pulse-formed in OUTRCK. Note that OTR is sent to the MEGABUS data transmitters for "writes" only; for reads, a pair of 74F373s contribute the CUP channel number which is sent instead.

**3.11.3 loading adra and adrb (see lbd pages 18m and 29m)**

The two remaining control-store bits in the LD field (CSLD65,66) and their cr flops (CRLD01,02) are provided, in part, to control the loading of these two address registers. Each address register is 24 bits long and comprises three 74S374 chips. Address registers adra (ADRA08-31) and adrb (ADRB08-31) are loaded from the zbus. The load control for adra (ADRACK) is a 74S10 circuit which loads adra if the two LD bits have a value of 01. Adrb is loaded by ADRBCK if the two LD bits have a value of 11.

**3.11.4 loading adrp and pctr (see lbd pages 18m and 28m)**

Adrp is a twenty-four-bit register which is part of the procedure prefetch mechanism along with other elements described in 3.6. These elements, the take counter (PTAKE4/2/1), the prefetch buffer, and the prefetch control logic automatically replenish the procedure buffer as bytes are consumed. Adrp remembers the memory address where the next procedure word is to be read, and pctr remembers the low order nine bits of the address of the next byte in the buffer to be "taken". Notification that the procedure stream crossed a 512-byte or an 8192-byte boundary is available (see 3.15).

This partitioning is carried into the loading of adrp if the switch APLONG is off. CRLD00-2 control which portion of adrp is to be loaded. A code of one generates PLLoad and PULoad which loads all of adrp (and pctr) if APLONG is on; otherwise, a code of one generates PLLoad only and loads the nine least significant bits of adrp (and pctr). A code of five generates PULoad and loads the fifteen most significant bits of adrp (and also loads outR).

Adrp is positioned in three 74AS869 counter chips so that it can increment by two bytes each time a procedure word is received from the memory or Cache/Local Memory.

The register PCTR(23-30) mimics adrp's loading actions for its corresponding bits. Its bit justification in a 74A869 chip, supplemented by the least-significant bit of the take counter (PTAKE1), permit incrementing by one when procedure bytes are removed from the prefetch buffer. When pctr is delivered to the dbus, PTAKE1 becomes the units position of pctr.

**3.11.4 Loading the Accounting Timer (see lbd page 27m)**

The Accounting Register is a 24-bit counter loaded from the Zbus when the FL field = 05. Once loaded, the counter increments for each firmware step executed. When it reaches zero, an interrupt is created. The interrupt is cleared when a test condition = 25 is issued.

**3.11.5 Loading the Configuration Register (see lbd page 23m)**

The Configuration Register is an eight-bit register which controls the behavior of the Custom Processor. Three of the eight Configuration Register bits can be loaded and all eight bits can be tested. The entire Configuration Register is implemented in a PAL. Its control over CUP behavior is explained in Tables 3-12 and 3-13.

TABLE 3-12  
CONFIGURATION REGISTER INPUTS

INPUT	DESCRIPTION
CNFGCK	Affects certain bits of the CUP Configuration Register as follows: COMMAND PARITY is enabled if BUSZ(28) = 1 INITFC is enabled if BUSZ(30) = 1
BUSZ	To modify the state of the alterable configuration bits.
ADRP23 ADRP19	To allow the procedure page-cross detector to signal the 512 or 8192 byte boundary (see ADRPWP).
CRMG	To allow the three test conditions CNFIGA,B,C to be able to sample the other Configuration Register bits.
MYMCLR	To set the initial state of each Configuration Register output.

TABLE 3-13  
CONFIGURATION REGISTER OUTPUTS

OUTPUT	DESCRIPTION
ACTREN	Enables the Accounting Timer.
ADRPWP	Determines whether an error (ADRPOV) will be detected when ADRP traverses the 512 byte boundary (ADRP23) or the 8192 byte boundary (ADRP19).
APLONG	The output which puts ADRP in an unpartitioned mode. When APLONG is true ADRP, when incrementing propagates the carry through the upper 74AS869's and also causes loads of ADRP to affect all 24 bits.
CMDPEN	In addition to address and data parity, a parity bit on the command leads accompanies all MEGABUS transfers. CMDPEN allows the command parity generator output to be transmitted on the MEGABUS and also enables the command parity checker such that a bus cycle with bad command parity is ignored regardless of its other saving graces.
CNFIGA CNFIGB CNFIGC	These three outputs are strictly test conditions and traditionally become the three-least-significant bits of the reply to a function code 26 inquiry (Who are you?).
CSTEAL	Enables the mechanism which allows the CUP to behave as a low priority MEGABUS requestor even when plugged into a high priority slot. CSTEAL is an input to the PAL which generates MYREQS and PRIREQ which initiate the MEGABUS request and energizes the MEGABUS request tie-breaker respectively.
INITFC	Enables the mechanism which responds to a special MEGABUS cycle (function code 01) which (re)initiates the QLT regardless of the current Custom Processor state. See 3.6.5.
NEWPEN	Enable address parity on the 16-least significant bits.

3.12 The Four-Speed Clock (block diagram identifier XII)

The Custom Processor thrives in an asynchronous world. The Custom Processor clock is an asynchronous mechanism. The clock has two orthogonal properties:

1. the clock allows the duration of each firmware step to be any one of four lengths as a function of the combination of micros coded in that step.
2. the clock allows each firmware step to delay its completion until some external event occurs.

3.12.1 the basic clock (see lbd pages 12m and 13m)

The basic clock has three delay lines, three delay-line drivers, one 74S32, one 74S64 and two switch banks for adjustment purposes. Two of the three delay lines are connected in parallel. When MCX000 occurs because all of its inputs are high (take it on faith), a positive to negative edge travels down the 100ns delay line MCX010 through MCX100. At the same time, MCX000 creates an edge of the opposite polarity (negative to positive) traveling down the delay line MCPW05 through MCPW50. The outputs of this latter delay line are switch selected to create the width of MCLOCK. This is accomplished by connecting the output of the switch bank (MCKPWA) through a 74S64 (MCSTLA) to the original delay line driver (MCX000) making one of its inputs low which achieves a pulse width of about 44ns. Having established that a "negative" forty-four-nanosecond pulse is wending its way down the MCX010-100 delay line, it is now appropriate to pursue what happens to cause the cycle to complete (and start over). The earliest connected tap on the MCX010-100 delay line is buffered by a 74S32 and, in the simplest case (i.e., the fastest clock speed), causes the delay line driver MCY000 to go high. The delay line MCY010-030 is now propagating a "positive" forty-four nanosecond pulse. The output of the switch bank to which this delay line is connected (THEEND) has the responsibility for terminating the cycle. That is accomplished by first sustaining the effect which MCKPWA achieved upon MCSTLA. When MCKPWA rose, MCX000 went high and MCLOCK went low signifying the "middle" of the cycle and completing MCKPWA's contribution. Just before this midpoint, THEEND is allowed to take over the responsibility of keeping MCLOCK low. The stage is now set so that when THEEND gets exhausted, the cycle ends and, of course, a new one begins.

3.12.2 the gear shifter (see lbd page 13)

The clock speed for each firmware step is selected when the firmware is assembled. The assembler (or an agent thereof) selects, as a function of the particular combination of micros in every step, a two-bit clock speed code for the CK field. Listed in Table 3-14 are the four speeds and their definition.

TABLE 3-14  
CLOCK SPEEDS

CK	definition
00	very fast (vf)
01	half fast (hf)
10	half long (hl)
11	very long (vl)

Control-store bits 80 and 81 (CSCK80, 81) are sampled by two 74S374 flops which are decoded into three clock-speed-enable signals. On the delay line driver MCY000, one "diode" is devoted to each of four delay-line taps as selected by the clock enable signals. For a very fast speed, no enables are active and the operation is as described above (i.e., the vf tap is always enabled). For a half-fast step, CKHFEN causes MCLKHF to sample another delay-line tap which has the effect of extending the on time of MCY000; that is, it goes positive at the same time as in a vf step but lasts longer and makes THEEND last longer. It was established previously that the positive to negative transition of THEEND ends the firmware step, so now, because of CKHFEN, the MCLOCK low time increases.

When a half-long step is prescribed, both CKHFEN and CKHLEN activate causing three of the four inputs of the delay line driver MCY000 to get into the act. The additional contributor, MCLKHL, stretches the high time of THEEND even further and increases the delay to the completion of the step.

Finally, when a very-long step is required, all inputs to MCY000 get a turn because all three enable outputs (CKHFEN, CKHLEN, and CKVLEN) are active, and the high time of THEEND stretches even further, creating the longest programable firmware step available.

### 3.12.3 clock stalls (see lbd page 12m)

Three 74S64's provide the inputs for stalling the clock from "external" stimuli. The basic clock operation uses two of the twelve inputs; that is, the MCKPWA input controls the clock first-phase pulse width and the THEEND input controls the second-phase pulse width. The other active input gates provide stalls for the conditions listed below. Each gate must be off during the first phase of the step. During the second phase of the step, these gates are ignored until THEEND relaxes (the gear shifter time has elapsed). Then these gates come into play, each having the power to postpone the completion of this step.

TABLE 3-15  
STALL CONDITIONS

STALL GATE	EXPLANATION
CSMG67& CSMG68& MGBUSY	stall WHEN the next firmware step is to unconditionally initiate a MEGABUS or Cache/Local Memory request IF previously initiated MEGABUS or Cache/Local Memory activity is still in progress.
ACKNAK& ACKSTL	stall WHEN an explicitly coded stall micro requires that the completion of this step be postponed until it is known whether the previous request was acknowledged IF the ACK/NAK has not arrived.
CSMG67& BLOTHR& MGBUSY	stall WHEN the next firmware step is to conditionally initiate a MEGABUS or Cache/Local Memory request and the condition (below-threshold) is satisfied IF the previously initiated MEGABUS or Cache/Local Memory activity is still in progress.
DSASTF& DSA200	stall WHEN a hardware error has been detected until such time as the reason code (syndrome) has been captured and the next address generator has generated and accessed control store location zero.
BSYSTL& MGBUSY	stall WHEN an explicitly coded stall micro (or load of ADRP lower) requires that the completion of this step be postponed until all MEGABUS or Cache/Local Memory activity has subsided IF MEGABUS or Cache/Local Memory activity is still in progress.
TBSTOP	stall if and while the firmware development facility says so.
CSMG67& OLDDUE& PEMPTY	stall WHEN the next firmware step will examine byte(s) of the procedure buffer IF the procedure buffer is empty.
POWRON& MYMCLR	stall during the master clear pulse stimulated by power coming on.
FRADUE& AFLSTL	stall WHEN an explicitly coded stall micro requires that the completion of this firmware step be postponed until data requested for delivery into inra arrives IF that data has not arrived.

### 3.13 The Return Stack (block diagram identifier XIII)

The return stack is a last-in-first-out (lifo) mechanism for storing and retrieving firmware addresses. It can store two addresses. A firmware address is written into the stack when a push micro is executed. An address is read from the stack when a return micro is "successfully" executed. An unconditional return is always successful; i.e., it uses and discards the stack's "top" entry. A conditional return is successful only if the test condition is met; if the condition is not met, the stack is undisturbed. A push is performed in preparation for calling a subroutine and a return is the mechanism used to exit a subroutine. The return stack hardware comprises:

1. two return registers
2. an alternate action top-of-stack pointer

#### 3.13.1 the return registers (see lbd page 22d)

The two return registers each comprise two 74S374 chips which provide storage for fourteen-bit firmware addresses. The outputs of the two registers are bused together and selected by the tri-state enables RTNAEN and RTNBEN. The enables are mutually exclusive and the active one signifies which of the two return registers is currently the stack top; i.e., the address where the next successful return will go. The flop STKTPA is the top of stack pointer and is the only difference between the inputs of the two enable signals. Likewise, the two load signals, one of which happens at each "push", differ only in the polarity of the STKTPA input but the push overlays the stack "bottom". By the beginning of the next firmware step, the pushed address, by virtue of STKTGL, is the stack top.

#### 3.13.2 the stack pointer (see lbd page 22d)

STKTPA is an alternate action top-of-stack pointer which toggles either when a successful return occurs or when a push is performed. STKTGL drives both the J and K inputs of a 74S112, toggling STKTPA if RETURN and TCTRUE (a successful return) or CRPS00 (a push). The toggle signal is constructed as a latch in order to overcome a peculiarity of the firmware development facility when it injects a derailing address (in a firmware debug environment). STKTPA is so constructed that the toggling takes effect at the beginning of the next firmware step.

### 3.14 The Next Address Generator (block diagram identifier XIV)

The next-address generator determines what the address of the next firmware location will be. One next address is as attainable as any other next address because the Custom Processor has no concept of address sequentiality. This flexibility is provided by devoting five firmware fields (27 control-store bits) to the task of deciding what control-store location shall next be executed. The "control-store" is comprised of two arrays; a PROM array and a RAM array. The PROM array is physically implemented as two 8192-location memories; the lower addresses access the else bank and the higher addresses access the if bank. The RAM array is physically one 16384-location memory. Figure 3-1 diagrams the if and else next address generators.

Of the five fields involved in this activity, the BR field is the major determining factor distinguishing "go-to's" from "returns" from "splatters" and deciding how the MK field will be applied. The TC field specifies which one of the 64 "conditions" will be tested; when the condition is met, the if bank is enabled, and when the condition is not met, the else bank is enabled.



3.14.1 bank selection (see lbd pages 23m and 15d)

The TC field (CRTC00-05) can specify one of sixty-four conditions to be tested in order to decide which PROM bank will be selected. Eight 74S151's, two 74S251's and a couple of 74S00 circuits (TCNLEN, TCIVEN) accept the sixty-four test conditions, the six bits of the TC field, and one bit of the next address field (CRNA00) reducing all this into one signal called TCTRUE, utilizing a tri-state network of TCTRUE.NL and TCTRUE.IV. The high order next-address bit specifies the test polarity, which is why the firmware dictionary has twice as many (128) tests. The signal TCTRUE is fed to four 74S140 drivers for distribution purposes. The drivers are disabled if either the firmware development facility is supplying the writable control store output (TBRMEN) or if the RAM array is enabled (RAMENB)..When RAMENB is true, a 2to1 mux which supplies RAM addressing selects the if bank address if TCTRUE=1 (and selects the else bank address if TCTRUE=0).

RAMENB is controlled by the "flag field" micro FLEQ07 which sets RAMENB when, in the same firmware step, the OP register is being loaded (CRXX00) such that RAM execution takes command (at the end of the next firmware step). If micro FLEQ07 occurs when the OP register is not being loaded, then RAMENB clears and the PROMs become enabled (at the end of the next firmware step).

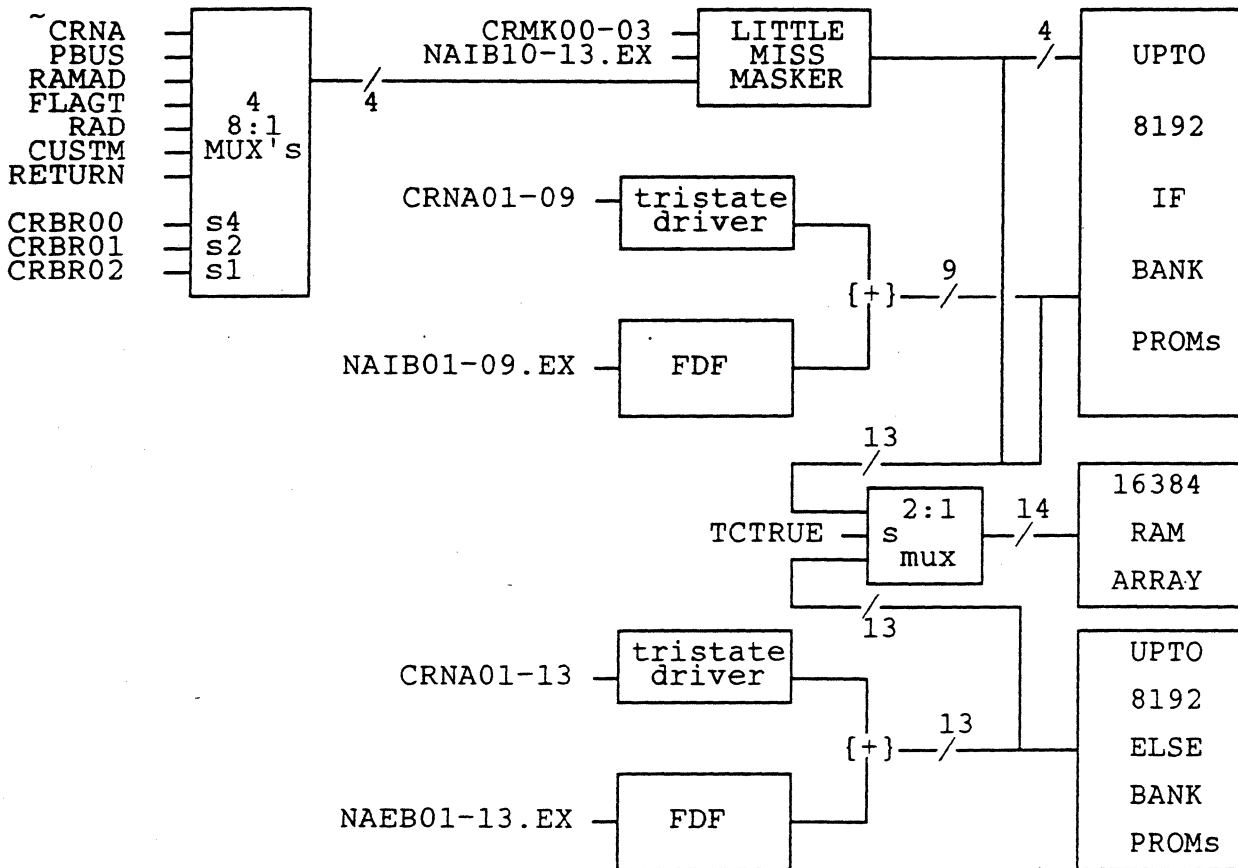


FIGURE 3-1  
IF AND ELSE BANK ADDRESS GENERATORS

## 3.14.2 else-bank next-address generation (see lbd pages 14m, 20m, 21m, and 1d)

The else bank may receive an address onto its tri-state address bus from one of two sources, which are listed in Table 3-16 along with the BR code(s) that enable(s) each source. Note that the determination of which bank shall be selected is the responsibility of the TC field. Note also that CSNA82, through CRNA00 (the most significant bit of na) determines the polarity of the test condition. In firmware lingo, this bit is referred to as the BI field.

TABLE 3-16  
ELSE BANK SOURCES

else bank source	BR code
the next address field (CRNA01-13)	all
the firmware development facility (NAEB01-13.EX)	all

Source #1 is the register which stores the thirteen control-store bits CSNA83-95, connected through tri-state drivers to the else-bank address bus (NAEB01-13). The drivers are enabled if the FDF is not injecting an address.

Source #2 is the input which the firmware development facility employs when it forces the Custom Processor to start execution at some arbitrary address. The signal TBNASB imports the fdf-specified address and disables the other tri-state source to the else bank (and if bank) address bus.

## 3.14.3 if-bank next-address generation (see lbd pages 9m, 14m, 22d and 24d)

The if bank may receive an address upon its address bus (NAIB01-13) from any one of four sources which are listed in Table 3-17 along with the BR code(s) that enable(s) each source.

TABLE 3-17  
IF BANK SOURCES

if-bank source address	BR code(s)
the next-address field (CRNA01-13)	0
the 16-way masked splatter (splata-d)	1,2,3,4,5,6
the return stack (RTNA01-13 or RTNB01-13)	7
the firmware development facility (NAIB01-13.EX)	all

The if-bank address generator is divided into two partitions: that circuitry which develops bits 1 through 9, and that circuitry which develops bits 10 through 13.

## 3.14.3.1 if-bank address bits 01-09 (see lbd page 22d)

The nine most-significant bits of the if-bank address bus (NAIB01-09) receive their inputs from one of three sources as shown in Table 3-18 with the BR code(s) that enable(s) each source.

TABLE 3-18  
IF BANK ADDRESS BITS 01-09

source for if-bank address bits 01-09	BR code(s)
the next-address field (CRNA01-09) via 74S240 circuits (CRNA01-09.11) enabled by a 74S32 (NANLEN).	0,1,2,3,4,5,6
the return stack (RTNA01-09 or RTNB01-09) via a pair of 74S10 circuits (RTNAEN and RTNBEN).	7
the firmware development facility (NAIB01-09.ex) via TBNASB. Note that the drivers for this source are in the fdf.	all

3.14.3.2 if-bank address bits 10-13 (see lbd page 14m)

The four least-significant bits of the if-bank address bus receive four types of inputs as shown in Table 3-19, along with the BR code(s) that enable(s) each input.

TABLE 3-19  
IF BANK ADDRESS BITS 10-13 (PART 1)

source for if-bank address bits 10-13	BR code
the next-address field (CRNA10-13) Two elements are used to allow the four least-significant bits of the if/else alternatives in two-way branches to differ. These two elements are a 74S64 chip per bit and a 74S151 chip per bit (i.e., eight total chips). The three-bit BR code selects among the eight inputs of each 74S151. With a BR code of zero, crna is selected and sent to each 74S64 anded with the mask (CRMK00-03). Another gate of each 74S64 ands mask with crna. The effect of these gates is to allow the assembler, by controlling the mask field, to generate any four-bit value for the if bank as a function of the raw else-bank value.	0
the 16-way masked splatters The 74S151's are selected by the three bits of the BR code to choose among six types of sixteen-way splatters. The outputs of the selectors (SPLATA, B, C, D) are anded with the mask bits on the 74S64's. Another gate on the 74S64 allows the crna bits through if the mask is off. In this manner, any combination of the four selector outputs may participate in the splatter. These splatters are:	1,2,3,4,5,6

TABLE 3-19  
IF BANK ADDRESS BITS 10-13 (PART 2)

source for if-bank address bits 10-13	BR code
a. four bits of the pbus (BUSP00-03)	1
b. four bit ARAM address (RAMAD0-3)	2
c. four temporary flags (FLAGT0-3)	3
d. the RAD register (RAD000-3)	4
e. four "arithmetic" indicators	5
f. outputs of the custom decode prom (CUSTM0-3)	6
the return stack (RTRN10-13)	7
The BR code selects input #7 on each 74S151 (RTRN10-13) whose output is anded with the mask bit on each 74S64. Another gate on each 74S64 injects a zero bit if the mask is off. This mechanism provides alternate returns.	
The firmware development facility (NAIB10-13.EX) via TBNASB which, after disabling all other inputs, routes four bits through the 74S64's.	all

### 3.15 Availability Circuits (block diagram identifier XV)

The availability circuits are of three types:

1. those that detect errors,
2. those that generate check information so that other system elements might detect errors, and
3. those that verify the integrity of the other two.

#### 3.15.1 error-detection circuits

Error detection requires that data, transmitted around the system, be accompanied by redundant information like parity or an error detection and correction code (EDAC). Another mechanism for error detection is to recognize that a totally unexpected event occurred, such as receiving no response when attempting to communicate with another system element.

The Custom Processor's areas of error detection are listed below. Subsequent paragraphs will discuss each area in further detail:

1. procedure parity
2. data parity
3. procedure red (uncorrectable edac error)
4. data red (uncorrectable edac error)
5. procedure uar (unavailable resource)
6. data uar (unavailable resource)
7. control store parity

### 3.15.1.1 procedure parity (see lbd page 36m)

When procedure is delivered from main memory, 16 information bits are accompanied by 2 parity bits. These parity bits are captured by the edges of PRASK0 and PRASK1 in registers parallel to those which capture the procedure itself. When PRASK1 goes on, DATAP0 and DATAP8 are captured in a 2918 chip corresponding to prefetch buffers A and B (INABP0 and INABP8). The arrival of the next 16 bits causes PRASK0 to come on, capturing DATAP0 and DATAP8 into a 2918 corresponding to prefetch buffers C and D (INCDP0 and INCDP8). Similarly, INEFP0/INEFP8 receive the MEGABUS and Cache/Local Memory parity bits when PRASK1 goes off and INGHP0/INGHP8 get loaded when PRASK0 goes off. The trick is to select the parity bit which correspond to the procedure byte now on the pbus. The tri-state outputs of these four 2918's are bused together to create a four-to-one selector; i.e., tri-state enable PTAKAB is on when either prefetch buffer A or B is next to be taken; PTAKCD is on when bytes C or D are next to be taken and so on for PTAKEF and PTAKGH. The selection is now down to two bits (BUSPP0 or BUSPP8) which each feed a 74S51 to perform the last selection level as a function of the least-significant bit of the take counter. The output of this last selection level (BUSPPB) is delivered to the 82S62 which monitors BUSP(0-7), enabled when procedure is sampled. The result of the parity check (PERKEY) is latched at mid-cycle (PERKED), qualified by the error blocking signal (PRLKFR) and sent to the general error collector (DSASTR) and to the syndrome register.

### 3.15.1.2 data parity (see lbd page 24m)

When inra or inrb captures the MEGABUS and Cache/Local Memory data, its load signal (INRACK or INRBCK) also captures, in a parallel (2918) register, the corresponding parity bits. INRAP0 and INRAP8 contain the inra parity bits and INRBP0 and INRBP8 contain the inrb parity bits. The outputs of the 2918's are bused together to form a two-to-one selection, enabled by CRIA2Z or CRIB2Z, the inra/inrb selection signals. When inra or inrb is placed upon the zbus, the outputs of the selector (BUSZP0 and BUSZP8) each feed an 82S62 parity checker whose outputs (DTPER0 and DTPER8), qualified by the error inhibitor (NOFALT), create the data parity error signal PERDTR which is sent to the general error collector DSASTR and to the syndrome register.

### 3.15.1.3 procedure red (see lbd page 36m)

When procedure is delivered from the memory subsystem, 16 information bits are accompanied by a "red" indicator (DATARD). This indicator, accusing the 16 bits of containing an uncorrectable edac error, is captured in four 2918 sections, just like the procedure parity bits. Again, a four-to-one selector is formed whose output (BUSPRD), after a mid-cycle latch, is qualified by the error inhibit signal (NOFALT) to create REDPRC which is sent to the general error collector DSASTR and to the syndrome register.

### 3.15.1.4 data red (see lbd page 34m)

When data is delivered from MEGABUS and Cache/Local Memory, 16 information bits are accompanied by a "red" indicator (DATARD). This indicator, accusing the 16 bits of containing an uncorrectable edac error, is captured in a 2918 section for each of inra and inrb, just like the data parity bits. Again, a two-to-one selector is formed whose output (BUSZRD) is qualified by the error inhibitor (NOFALT) and sent to the general error collector DSASTR and to the syndrome register.

## 3.15.1.5 procedure and data uar (see lbd page 9m and 36m)

When a memory reference is made to an uninstalled memory location, the signal TIMOUT will result approximately three microseconds from the initiation of the reference, if the CUP is not channel zero (MASTER) or five microseconds from read initiation, if the CUP is indeed MASTER. TIMOUT is sent to the general error collector DSASTR and to the syndrome register.

## 3.15.1.6 Procedure page-cross detection (see lbd page 36m)

The signal ADRPOV monitors bit 23 of adrp, detecting ADRP23 going off, which signifies a transition from 511 to 512. If adrp lower is not being loaded (PLLOAD), then ADRPOV sets and acts as a time bomb awaiting the next attempt to load a procedure buffer from the MEGABUS and Cache/Local Memory. 2918 sections INABUR, INCDUR, INEFUR, and INGHUR create BUSPUR and, if an attempt is made to remove a byte of procedure labeled "UR" when APLONG is off, then the general error collector and the syndrome register are notified via UARPOB, a mid-cycle latch, and UARPRC.

## 3.15.1.7 control-store parity (see lbd pages 18m, 19m, 20m, 21m, 3d, 4d, 5d, 6d, 7d, 8d, 9d, and 10d)

The 96-bit control-store array contains one parity bit in each thirty-two. The parity bits are CSOP24, CSOP56, and CSOP72. In each thirty-two-bit group, the control-register (cr) outputs are tested for correct parity. In each group, 74AS280 parity checkers detect whether the thirty-one "data" bits agree with the one parity bit. Eleven parity checker chips are gainfully employed in this endeavor.

Parity chip EP0008 checks parity on the first nine control-store bits, EP0917 checks parity on the next nine, EP1826 generates parity for the next eight (including the parity bit for this thirty-two-bit group) and finally EP0031 checks parity for the last five and gathers the outputs of the other three chips to form an error signal which is sent to the general error collector (DSASTR) and to the syndrome register.

In the second group of 32 control-store bits, EP3240 checks parity on the first nine bits, EP4149 checks parity on the second nine bits, EP5058 checks parity for the next nine bits (including the parity bit for this group) and EP3263 checks parity for the last five and gathers the outputs of the other three chips to form an error signal which is sent to the general error collector (DSASTR) and to the syndrome register.

In the third group of 32 control-store bits, EP6472 checks parity on the first nine bits, including the parity bit for this group, EP7381 gets the next nine, EP8290 gets the next nine, and EP6495 gathers the remaining five in addition to accepting the output of the previous checkers and is sent to the general error collector (DSASTR) and to the syndrome register.

## 3.15.2 parity generation circuits (see lbd pages 24m and 25m)

When a MEGABUS transaction is initiated, two data parity bits, three address parity bits and, if CMDPEN is true, a command parity bit must be generated. The twenty-four bits of the address bus are sent to three 74AS280 parity generators resulting in MYAP00, MYAP08 and MYAP16 then to the address parity transmitters BSAP00, BSAP08 and BSAP16. The data parity bits are generated by two 74AS280 parity generators which receive a tri-state bus MYDT(00-15) which performs a two-to-one selection of outr (OUTR00-15) versus the channel number generator (CHNL00-15). When memory writes are initiated, outr is enabled to the mydt bus and when memory reads are initiated, chnl is enabled.

### 3.15.3 verifying the integrity circuits

When parity is generated to accompany the data loaded into outr, the hardware provides a method of generating even rather than the conventional odd parity. This can be accomplished only if the processor is off-line (FLAGP0 is off) and if FLAGP2 is on (FRCERR). This mechanism, in conjunction with the WRAP micros, allows a firmware routine (presumably the quality logic test) to verify the integrity circuits by transferring data with both even and odd parity from outr to inra, from outr to inrb, from outr to pa and from outr to pe. This same mechanism also allows even address parity to be generated by ADPGEN for the eight-most-significant address bits.

The micro LD-SYND is available to verify the integrity of the syndrome register by causing a simulated error, stimulating the general error collector (DSASTR) and sampling the presumably quiescent error sources into the syndrome register.

### 3.15.4 branch to zero (see lbd 25m)

When the 74S133 (DSASTR) is stimulated, it sets SYRCLK at the start of the next firmware step. The leading edge of SYRCLK samples all error sources into the syndrome register (SYND08-31). During the first step following the detection of an error, the control flop FLAGP5 controls whether the flop DSASTF will set leading toward the hardware "interrupt". If FLAGP5 is off, DSASTF starts a chain of events which will cause location zero to be the next executed. Immediately, a clock stall is instituted while the derail sequence takes hold. The delay-line driver DSA000 starts an edge down the delay line DSA200. For 200 nanoseconds, HOLDIT clears the next-address control-register flops (CRNA00-13). The test-condition muxes are disabled, forcing an else-bank enable. The next-address generator emits zero and the control-store array is accessed. The clock stall (DSASTF and DSA200), which has been active through this entire sequence, is allowed to relax and the content of location zero is executed next.

### 3.16 ALTERABLE FIRMWARE ARRAY (see lbd pages 16m, 17m)

The alterable firmware array is comprised of twenty-four 16K x 4 high speed RAM chips whose data outputs participate in the control store tri-state network (CSxx00-95). The signal RAMENB determines whether firmware execution is to proceed from the PROM array or the RAM array.

#### 3.16.1 Loading the alterable firmware array

The RAM array may only be written when execution is proceeding from the PROM array. When FL field code 01 occurs, one of eight actions occur as a function of the value contained in CYCLE(4,2,1) as shown in Table 3-20.

If FL=01:

- o For each firmware location to be written, a ninety-six bit Firmware Write Register is loaded in six sixteen-bit portions for CYCLE(4,2,1) decodes of 2 through 7. Four portions (bits 00-63) are on the daughterboard and two portions (bits 64-95) are on the motherboard.
- o The Firmware Address Register is loaded when CYCLE(4,2,1) decode is 1. This register is duplicated; i.e., one "copy" on the motherboard and one copy on the daughterboard.
- o The 96-bit Firmware Write Register is copied into the RAMS when CYCLE(4,2,1) decode is 0.

The daughterboard generates most of the control signals to perform these activities:

- o WRTFW(0-5) are the six Firmware Write Register load signals
- o WRTFWA is the Firmware Address Register load signal
- o FWLOAD is the firmware write signal pulse-shaped to WRTFW

TABLE 3-20  
CYCLE MODULATION OF FWLOAD

CYCLE 4 2 1	DEFINITION
0 0 0	Write all 96 bits of Firmware Write Register into the firmware RAM array at the location specified by the Firmware Address Register
0 0 1	Load the Firmware Address Register from BUSD(18-31)
0 1 0	Load Firmware Write Register bits 00-15 from BUSZ(16-31)
0 1 1	Load Firmware Write Register bits 16-31 from BUSZ(16-31)
1 0 0	Load Firmware Write Register bits 32-47 from BUSZ(16-31)
1 0 1	Load Firmware Write Register bits 48-63 from BUSZ(16-31)
1 1 0	Load Firmware Write Register bits 64-79 from BUSZ(16-31)
1 1 1	Load Firmware Write Register bits 80-95 from BUSZ(16-31)

### 3.16.2 Enabling a Firmware Array

RAMENB determines whether the RAMs or the PROMs are to be accessed from the "next address". Power-up or Master Clear causes RAMENB to be false so that execution begins out of PROMs. When the FL field emits micro 07, RAMENB becomes true (at the beginning of the next firmware step) but the CR register has already been loaded with the output of the next-addressed PROM array location. As a result, the "switch" from one array to the other takes place with a one firmware step delay. When the FL field emits micro 06, RAMENB becomes false causing execution to revert to the PROM array, again with a one firmware step delayed effectivity.



FIRMWARE DESCRIPTION

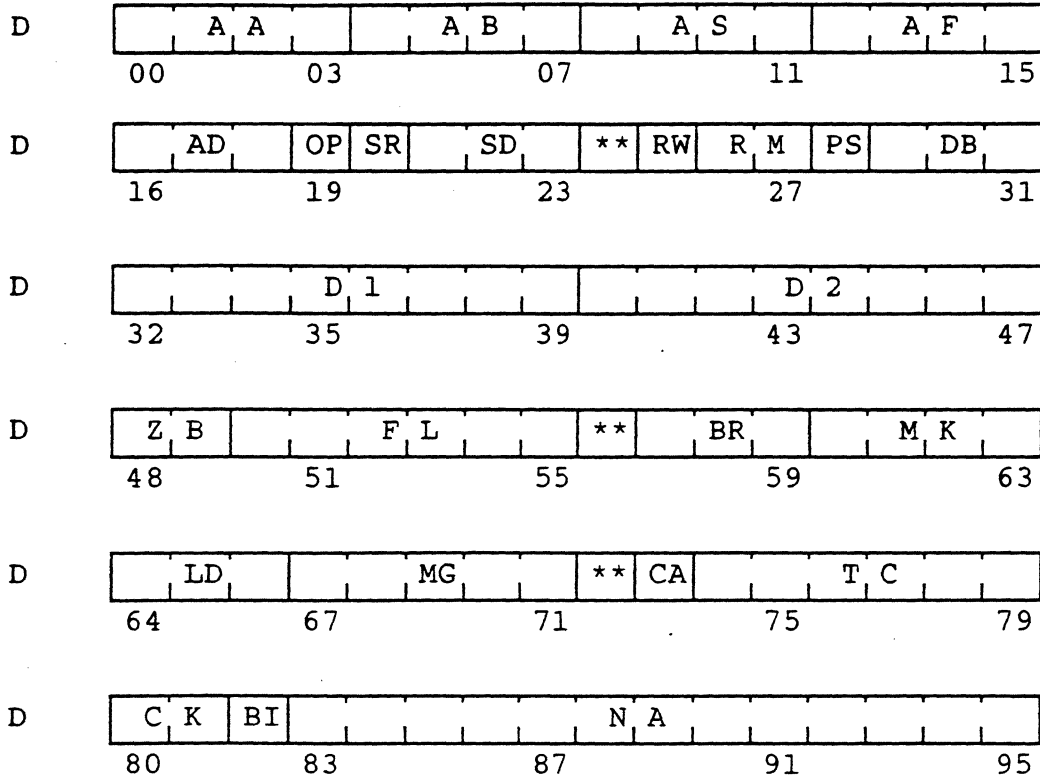
This section provides a detailed description of the Custom Processor firmware structure.

The Custom Processor divides its firmware word into fifteen zones and twenty-five fields (see table 4-1 and figure 4-1).

TABLE 4-1  
THE FOURTEEN FIRMWARE ZONES

I. 2901 Control	The five fields which control the 2901 are AA, AB, AS, AF, and AD.
II. ARAM Control	The two fields which control the ARAM are RM and RW.
III. D Bus Control	The three fields which control the D bus are DB, D1, and D2.
IV. Address Bus Control	The fields which controls the Address bus are MG and FL.
V. Z Bus Control	The field which controls the Z bus is ZB.
VI. MEGABUS & Cache /Local Memory	The field which controls the MEGABUS and Cache/Local Memory is MG.
VII. Flag Control	The field which controls the Temporary flags and the Permanent Flags is FL.
VIII. Nibble Shifter Control	The field which controls the nibble shifter is SD.
IX. Indicator Control	The field which controls the loading of both the arithmetic and miscellaneous indicators is FL.
X. Op Register Control	The fields which controls the loading and modification of the OP register are OP and FL and TC.
XI. Load Control	The fields which control the loading of other major registers are LD, SR, and FL.
XII. Clock Control	The field which controls the clock speed is CK.
XIII. Stack Control	The field that determines whether a return address is pushed onto the stack is PS (DB, D1, D2 specify the address).
XIV. Next Address Control	The five fields which control the value of the next firmware address are BR, MK, TC, BI, and NA.
XV. Alterable firmware control	The FL field control both writing the alterable firmware and selecting whether the RAMs or PROMs shall be enabled.

A field is a group of control store bits devoted to the control of a hardware entity. In general, a one-bit field has one possible micro, a two-bit field has three possible micros, a three-bit field has seven possible micros etc. In any firmware step, each field may contain only one micro. The design attempts to collect those micros which are naturally mutually exclusive into the same field; e.g., in some step, the coder may wish the alu to add or he may wish it to subtract but certainly not both. The firmware word format is depicted in figure 4-1.



where \*\* = firmware parity checks

FIGURE 4-1 FIRMWARE WORD FORMAT

NOTE: In the discussion that follows, micro names and firmware field references are capitalized. Capitalization is otherwise avoided.

The firmware is organized into fourteen zones where the micros available in these zones are described one field at a time. Some fields contain no micros but serve merely as arguments for other micros. Some fields control more than one hardware element. The coder may not recognize that even though each hardware element appears to have a selection of micros, only one micro from each field is allowed in each step. The fourteen zones are listed below with a brief description of the related hardware each zone controls.

4.1 2901 CONTROL

Five fields control the 2901 register file and alu as shown in Table 4-2.

TABLE 4-2  
2901 CONTROL FIELDS (PART 1)

FIELD	DESCRIPTION																		
AA	provides the four bits which select the register file location available at the A port. The coder specifies the A-port address as an argument of an AS-field and/or AD field micro (see below).																		
AB	provides the four bits which select the register file location available at the B port. When the alu output is to be retained, the AB field selects the register file location where the alu output is stored. The coder specifies the B-port address as an argument of an AS field and/or AD field micro (see below).																		
AS	<p>selects two operands (r/s) for manipulation by the alu from among five sources: the a port, the b port, the q register, the dbus and zero. The available micros are:</p> <table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;">micro</th> <th style="text-align: left;">description</th> </tr> </thead> <tbody> <tr> <td>R:A'S:Q(AA)</td> <td>the r input to the alu receives location AA of the register file. the s input of the alu receives the q register.</td> </tr> <tr> <td>R:A'S:B(AA,AB)</td> <td>the r input of the alu receives location AA of the register file. the s input of the alu receives location AB of the register file.</td> </tr> <tr> <td>R:0'S:Q</td> <td>the r input of the alu receives zero. the s input of the alu receives the q register.</td> </tr> <tr> <td>R:0'S:B(AB)</td> <td>the r input of the alu receives zero. the s input of the alu receives location AB of the register file.</td> </tr> <tr> <td>R:0'S:A(AA)</td> <td>the r input of the alu receives zero. the s input of the alu receives location AA of the register file.</td> </tr> <tr> <td>R:D'S:A(AA)</td> <td>the r input of the alu receives the dbus. The s input of the alu receives location AA of the register file.</td> </tr> <tr> <td>R:D'S:Q</td> <td>the r input of the alu receives the dbus. The s input of the alu receives the q register.</td> </tr> <tr> <td>R:D'S:0</td> <td>the r input of the alu receives the dbus. The s input of the alu receives zero.</td> </tr> </tbody> </table>	micro	description	R:A'S:Q(AA)	the r input to the alu receives location AA of the register file. the s input of the alu receives the q register.	R:A'S:B(AA,AB)	the r input of the alu receives location AA of the register file. the s input of the alu receives location AB of the register file.	R:0'S:Q	the r input of the alu receives zero. the s input of the alu receives the q register.	R:0'S:B(AB)	the r input of the alu receives zero. the s input of the alu receives location AB of the register file.	R:0'S:A(AA)	the r input of the alu receives zero. the s input of the alu receives location AA of the register file.	R:D'S:A(AA)	the r input of the alu receives the dbus. The s input of the alu receives location AA of the register file.	R:D'S:Q	the r input of the alu receives the dbus. The s input of the alu receives the q register.	R:D'S:0	the r input of the alu receives the dbus. The s input of the alu receives zero.
micro	description																		
R:A'S:Q(AA)	the r input to the alu receives location AA of the register file. the s input of the alu receives the q register.																		
R:A'S:B(AA,AB)	the r input of the alu receives location AA of the register file. the s input of the alu receives location AB of the register file.																		
R:0'S:Q	the r input of the alu receives zero. the s input of the alu receives the q register.																		
R:0'S:B(AB)	the r input of the alu receives zero. the s input of the alu receives location AB of the register file.																		
R:0'S:A(AA)	the r input of the alu receives zero. the s input of the alu receives location AA of the register file.																		
R:D'S:A(AA)	the r input of the alu receives the dbus. The s input of the alu receives location AA of the register file.																		
R:D'S:Q	the r input of the alu receives the dbus. The s input of the alu receives the q register.																		
R:D'S:0	the r input of the alu receives the dbus. The s input of the alu receives zero.																		

TABLE 4-2  
2901 CONTROL FIELDS (PART 2)

FIELD	DESCRIPTION
AF	<p>determines what manipulation shall be performed on the two operands provided by the AS field. The available micros are:</p> <p>micro            the alu output receives:</p> <p>F:ADD1            the s input plus the r input plus one</p> <p>F:ADDC            the s input plus the r input [plus one if the carry indicator (ind1) is on]</p> <p>F:ADDC'           the s input plus the r input [plus one if the carry indicator (ind1) is off]</p> <p>F:ADD             the s input plus the r input</p> <p>F:S-R             the s input plus not the r input plus one</p> <p>F:S-R-C'          the s input plus not the r input [plus one if the carry indicator (ind1) is on]</p> <p>F:S-R-C           the s input plus not the r input [plus one if the carry indicator (ind1) is off]</p> <p>F:S-R-1           the s input plus not the r input</p> <p>F:R-S             the r input plus not the s input plus one</p> <p>F:R-S-C'          the r input plus not the s input [plus one if the carry indicator (ind1) is on]</p> <p>F:R-S-C           the r input plus not the s input [plus one if the carry indicator (ind1) is off]</p> <p>F:R-S-1           the r input plus not the s input</p> <p>F:OR              the s input inclusive ored with the r input</p> <p>F:SR              the s input anded with the r input</p> <p>F:SR'             the s input anded with not the r input</p> <p>F:XOR             the s input exclusive ored with the r input</p> <p>F:XNOR            the s input exclusive ored with not the r input</p>

TABLE 4-2  
2901 CONTROL FIELDS (PART 3)

FIELD	DESCRIPTION																
AD	<p>determines where the alu output is to be retained within the 2901. The choices are the register file, the q register, or neither. The AD field also allows any register file location to be presented to the zbus port. The AD field also controls the single-bit shifter. The 24-bit output of the alu may be shifted one bit left or right and stored in the register file at location AB. The 48-bit output of the alu and the q register may be shifted one bit left or right and stored in the register file and q. The available micros are:</p> <table border="0"> <thead> <tr> <th data-bbox="370 621 467 646">micro</th> <th data-bbox="695 621 906 646">description</th> </tr> </thead> <tbody> <tr> <td data-bbox="331 667 467 693">Y:F'Q:F</td> <td data-bbox="617 667 1354 743">the q register receives the alu output. the zbus control field may also select the alu output.</td> </tr> <tr> <td data-bbox="331 764 389 789">Y:F</td> <td data-bbox="617 764 1396 819">the zbus control field may select the alu output.</td> </tr> <tr> <td data-bbox="331 835 594 861">Y:A'B:F(AA,AB)</td> <td data-bbox="617 835 1419 932">the alu output is written at location AB in the register file. the zbus control field may select the data from the A port of the register file.</td> </tr> <tr> <td data-bbox="331 953 539 978">Y:F'B:F(AB)</td> <td data-bbox="617 953 1419 1050">the alu output is written at location AB in the register file. the zbus control field may also select the alu output.</td> </tr> <tr> <td data-bbox="331 1071 607 1096">Y:F'BQ:FQSR(AB)</td> <td data-bbox="617 1071 1419 1289">the 48-bit concatenation of the alu output and the q register are shifted right one bit position. The value of FLAGT1 (early) is shifted into the msb position. The 24 next most-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.</td> </tr> <tr> <td data-bbox="331 1310 607 1335">Y:F'BQ:FQSL(AB)</td> <td data-bbox="617 1310 1419 1549">the 48-bit concatenation of the alu output and the q register are shifted left one bit position. The value of FLAGT1 (early) is shifted into the least-significant bit position. the 24 least-significant bits are written into the q register. The 24 next least-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.</td> </tr> <tr> <td data-bbox="331 1570 571 1596">Y:F'B:FSR(AB)</td> <td data-bbox="617 1570 1419 1789">the alu output is shifted right one bit position. The value of FLAGT1 (early) is shifted into the most-significant bit position. the 24 most-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.</td> </tr> </tbody> </table>	micro	description	Y:F'Q:F	the q register receives the alu output. the zbus control field may also select the alu output.	Y:F	the zbus control field may select the alu output.	Y:A'B:F(AA,AB)	the alu output is written at location AB in the register file. the zbus control field may select the data from the A port of the register file.	Y:F'B:F(AB)	the alu output is written at location AB in the register file. the zbus control field may also select the alu output.	Y:F'BQ:FQSR(AB)	the 48-bit concatenation of the alu output and the q register are shifted right one bit position. The value of FLAGT1 (early) is shifted into the msb position. The 24 next most-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.	Y:F'BQ:FQSL(AB)	the 48-bit concatenation of the alu output and the q register are shifted left one bit position. The value of FLAGT1 (early) is shifted into the least-significant bit position. the 24 least-significant bits are written into the q register. The 24 next least-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.	Y:F'B:FSR(AB)	the alu output is shifted right one bit position. The value of FLAGT1 (early) is shifted into the most-significant bit position. the 24 most-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.
micro	description																
Y:F'Q:F	the q register receives the alu output. the zbus control field may also select the alu output.																
Y:F	the zbus control field may select the alu output.																
Y:A'B:F(AA,AB)	the alu output is written at location AB in the register file. the zbus control field may select the data from the A port of the register file.																
Y:F'B:F(AB)	the alu output is written at location AB in the register file. the zbus control field may also select the alu output.																
Y:F'BQ:FQSR(AB)	the 48-bit concatenation of the alu output and the q register are shifted right one bit position. The value of FLAGT1 (early) is shifted into the msb position. The 24 next most-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.																
Y:F'BQ:FQSL(AB)	the 48-bit concatenation of the alu output and the q register are shifted left one bit position. The value of FLAGT1 (early) is shifted into the least-significant bit position. the 24 least-significant bits are written into the q register. The 24 next least-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.																
Y:F'B:FSR(AB)	the alu output is shifted right one bit position. The value of FLAGT1 (early) is shifted into the most-significant bit position. the 24 most-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.																

TABLE 4-2  
2901 CONTROL FIELDS (PART 4)

FIELD	DESCRIPTION
AD (cont)	Y:F'B:FSL(AB) the alu output is shifted left one bit position. The most-significant bit of q (Q08) is shifted into the least-significant bit position. the 24 least-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the unshifted alu output.

## 4.2 ARAM CONTROL

Two fields control the aram. The RM field determines the aram address and the RW field decides when the aram is written.

### 4.2.1 RM

RM determines which of the four sources of aram addressing is to be used. The four sources are registers RAA, RAB, RAC and RAD. Micros are not directly associated with the RM field; instead, the RM field is an argument of other micros. These micros are:

D:HEX(RM) which emits the hex-decoder to the dbus. The value of RM determines whether the content of RAA, RAB, RAC, or RAD will be decoded.

D:REG(RM) reads the aram location specified by RM onto the dbus.

REG:Z(RM) writes the zbus data into the aram location specified by RM.

### 4.2.2 RW

RW is a one-bit field which determines whether the zbus data is to be copied into the selected aram location and supports the micro:

REG:Z(RM)

## 4.3 D-BUS CONTROL

The dbus has many sources. The three fields discussed in this paragraph determine which source or combination of sources is allowed to the dbus. The dbus is subdivided into three bytes named x, y, and z. The DB field specifies how the other two fields are to be interpreted. In general, the four-weight-bit of DB specifies whether the Y byte will receive an eight-bit constant; the two-weight-bit specifies whether the Z byte will receive an eight-bit constant; and the one-weight-bit of DB is a fill bit which is replicated where necessary to create 00 or FF bytes.

### 4.3.1 full literal

When DB=00X, DB(02), D1 and D2 provide a seventeen bit literal as shown in Table 4-3.

TABLE 4-3  
DBUS LITERALS

micro	byte x	byte y	byte z
D:LIT(DL)	8*[DL msb]	next 8 bits	last 8 bits

4.3.2 broadsides

Some of the dbus sources are considered integral entities and are sourced to the dbus with the shown in Table 4-4.

TABLE 4-4  
DBUS BROADSIDES

micro	byte x	byte y	byte z
D:ADRS	ADRS(08-15)	ADRS(16-23)	ADRS(24-31)
D:RAA-D	zeros	RAA,RAB	RAC,RAD
D:HEX(RM)	zeros	HEXD(00-07)	HEXD(08-15)
D:INRX	zeros	INRX(00-07)	INRX(08-15)
D:PCTR	zeros	0000000,PCTR23	PCTR(24-31)
D:PROC	zeros	zeros	PBUS(00-07)
D:REG(RM)	ARAM(08-15)	ARAM(16-23)	ARAM(24-31)
D:REG0	ARAM(08-15)	ARAM(16-23)	ARAM(24-31)
D:SHRG	SHRG(08-15)	SHRG(16-23)	SHRG(24-31)
D:SYND	SYND(08-15)	SYND(16-23)	SYND(24-31)
D:TIMER	TIMER(08-15)	TIMER(16-23)	TIMER(24-31)
D:ZSH(SD)	ZBUS(08-31) ROTATED BY THE DISTANCE SD		

4.3.3 mixes

Certain sources may, on byte boundaries, be mixed onto the dbus. These sources are SHRG, a literal byte, a fill byte, and the zbus rotated via the shifter. Micros of the form D:: require each byte to be specified in a language where K is a constant byte specified in the argument (D1) or (D2), 00 is a fill byte of all zeros, FF is a fill byte of all ones, S is a byte of SHRG, and Z is a byte of the zbus whose rotation is specified by the argument (SD). A few examples are shown in Table 4-5.

TABLE 4-5  
DBUS MIXES

micro	byte x	byte y	byte z
D::SSK(D2)	SHRG(08-15)	SHRG(16-31)	constant D2
D::FFZZ(SD)	Ones	ZBUS ROTATED BY SD	
D::ZSS(SD)	ZBUS shifted	SHRG(16-23)	SHRG(24-31)
D::00SZ	zeros	SHRG(16-23)	ZBUS shifted

See the firmware dictionary for a complete list.

4.4 ADDRESS BUS CONTROL

The address bus is the mechanism which provides channel#/function code information to the MEGABUS for programmed I/O dialogue and provides byte addresses to the MEGABUS and Cache/Local Memory interfaces for memory references. Both the MEGABUS and the Cache/Local Memory interfaces are completely asynchronous. One of four sources may control the address bus; namely, either ADRA, ADRB, ADRP or RUPT. When an external reference is initiated, the firmware step which specifies the initiation is responsible for selecting one of the four sources for the address bus. The hardware maintains that selection, ignoring any attempted selection(s) by subsequent steps until the initiated dialogue has concluded. It is therefore good coding practice to include a stall prior to any step in which the micro D:ADRS appears. It is also, of course, necessary to specify the ADRS source (e.g., ADRS:B) in the step which initiates the request. Although D:ADRS is operable, it is usually more performant to "save" a copy of the address register in a more accessible place (aram or ralu location), if one is available.

ADRX is a register which may only be loaded from the MEGABUS address receivers. It has some value in self-testing (i.e., QLT) but is of primary benefit when an unsolicited MEGABUS transfer is directed toward the CUP. In this instance, ADRX(16-25) contain the CUP's channel number (not terribly interesting), but ADRX(26-31) contain a function code, which may be used as a vector into an array of interrupt handlers (e.g., function code 20 could mean "suspend processing"; function code 32 could mean "dispatch").

4.5 Z BUS CONTROL

The zbus may receive from any one of three sources. Two sources are inra and inrb which each may receive 16 bits of data from the outside world. Two flavors of inra and inrb sourcing are provided. One flavor places the ralu's most significant byte onto zbus byte x with inra/b on bytes y and z, while the other flavor leaves zbus byte x dangling in the breeze. In either flavor, each byte of inra/b is validated against its accompanying parity bit which was also received from the outside world. Another zbus source comes from an external element. This interface is intended for testing purposes. The micros in this field are shown in Table 4-6.

TABLE 4-6  
ZBUS MICROS

micro	byte x	byte y	byte z
Z:Y-INRA	ALUY(08-15)	INRA(00-07)	INRA(08-15)
Z:Y-INRB	ALUY(08-15)	INRB(00-07)	INRB(08-15)
Z:Y	ALUY(08-15)	ALUY(16-23)	ALUY(24-31)
Z:INRA	unspecified	INRA(00-07)	INRA(08-15)
Z:INRB	unspecified	INRB(00-07)	INRB(08-15)

4.6 MEGABUS AND CACHE/LOCAL MEMORY CONTROL

The MG field controls the MEGABUS and Cache/Local Memory interfaces. Thus, the MG field initiates data reads and writes, I/O reads and writes, and procedure reads. Data and I/O reads and writes are explicit while procedure reads are implied by other micros, also in the MG field, which cause procedure bytes to be consumed, with an incrementation of PCTR. If all this sounds complicated, it is.



The initiation, monitoring, and consummation of MEGABUS and Cache/Local Memory activities requires reasonable care in the use of resources which may still be committed to a previous request. When an interlock is required, the CUP will stall its clock automatically. It recognizes that a stall is required as follows:

1. Stall unless and until at least one byte is available in the prefetch buffer before entering any step which will consume a procedure byte; i.e., before entering a step which contains a PTAKE micro.
2. Stall unless and until the MEGABUS and Cache/Local Memory interfaces are quiescent before entering a step which will initiate a new MEGABUS and Cache/Local Memory request i.e., before entering a step which contains a PREFETCH, a read (RD..), or a write (WR..) micro.
3. Stall unless and until the MEGABUS and Cache/Local Memory interfaces are quiescent before entering a step which will consume a procedure byte (a step which contains a PTAKE micro), if and only if the procedure buffer has at least four "empty" byte positions. (This is an obscure instance of condition #2, above.)

Other stalls must be stated explicitly as follows:

1. Stall unless and until the MEGABUS and Cache/Local Memory interfaces are quiescent before leaving a step which contains a STALL:BUSY micro.
2. Stall unless and until a positive acknowledge (ACK) or negative acknowledge (NAK) has been received from the most recent MEGABUS and Cache/Local Memory request in any step which contains a STALL:ACK micro.
3. Stall unless and until inra has received the requested data before leaving a step containing a STALL:INRA micro.

The MG field supplies the following MEGABUS and Cache/Local Memory procedure related micros as shown in Table 4-7 Part 1.

TABLE 4-7  
MEGABUS AND CACHE/LOCAL MEMORY MICROS (PART 1)

micro	description
PREFETCH	stall before entering this step until all external request activity has concluded; then initiate a procedure read with the address in adrp. When the MEGABUS or Cache/Local Memory responds, place the first two bytes into prefetch buffers a and b, and the second two bytes into procedure buffers c and d. Add one to adrp for each byte that arrives.
PTAKE	if the procedure buffer has space for four bytes, stall before entering this step and initiate a four byte procedure read; add one to adrp for each each byte that arrives; in any case, consume a procedure byte (by adding one to pctr) and verify parity of the procedure byte.

The MG field supplies two memory and Cache/Local Memory non-procedure read micros as shown in Table 4-7 Part 2.

TABLE 4-7  
MEGABUS AND CACHE/LOCAL MEMORY MICROS (PART 2)

micro	description
RD-MEM-WORD (FLM, MCA)	stall before entering this firmware step until all external request activity has concluded; then initiate an external request, to read two bytes at memory addresses adra/b-adra/b+1 (as a function of ADRS:A or ADRS:B). Inra/b(00-07) will receive the first byte and inra/b(08-15) will receive the second byte.
RD-MEM-DBLW (FLM, MCA)	stall before entering this step until all external request activity has subsided; then initiate an external request to read four bytes at adra/b, adra/b+1, adra/b+2, and adra/b+3. Inra(00-07) will receive the first byte, inra(08-15) will receive the second byte, inrb(00-07) will receive the third byte, and inrb(08-15) will receive the fourth byte.

The MCA argument permits the reads to bypass the Cache/Local Memory; i.e., initiate the read on the MEGABUS even if the Cache/Local Memory is present and on-line. For example:

RD-MEM-WORD ( ,NO-CACHE)

As shown in Table 4-7 Part 3, the FLM argument permits other variations of the read micros.

TABLE 4-7  
MEGABUS AND CACHE/LOCAL MEMORY MICROS (PART 3)

micro	description
RD-MEM..(LOCK)	stall before entering this step until all external request activity has subsided; then initiate a read-and-lock-memory request on the MEGABUS and arm the UNLOCK indicator. NOTE 1: ACK must be tested before attempting to remove data from inra/b. NOTE 2: If ack is not received, the memory module was previously locked, indicating the resource was seized by another requester. It is recommended that the UNLOCK indicator be used to determine when to make another attempt. NOTE 3: Any successful LOCK must be followed by either a RD-MEM..UNLOCK or a WR-MEM..UNLOCK micro; otherwise the memory module will remain seized indefinitely.
RD-MEM..(UNLOCK)	stall before entering this step until all external request activity has subsided; then initiate a read-and-unlock memory request on the MEGABUS and arm the SEMA4 indicator. The memory module is released. NOTE : If this action is part of a Read-Lock/Read-Unlock sequence signifying that a semaphore was "busy", it is recommended that the SEMA4 indicator be used to determine when to make another attempt.

As shown in Table 4-7 Part 4, the MG field provides two micros for programed I/O.

TABLE 4-7  
MEGABUS AND CACHE/LOCAL MEMORY MICROS (PART 4)

micro	description
RD-NON-MEM	stall before entering this step until all external request activity has subsided; then initiate a MEGABUS I/O read , using adra/b(16-25) as the channel number and adra/b(26-31) as the function code. If the channel acknowledges the request, the data will be placed into inra/b(0-15). Note: ACK must be tested before attempting to remove the data from inra/b.
WR-NON-MEM(MOP)	stall before entering this step until all external request activity has subsided; then initiate a MEGABUS request using adra/b(16-25) as the channel number and adra/b(26-31) as the function code. Send outr(0-15) to the specified channel. The argument MOP has three non-testing flavors. REPLY is reserved for the CUP "interrupt handler(s)" responding to an inquiry by sending a "second-half bus cycle" to the requestor. The other flavors (CMND,RUPT) differ only at the documentation level, where CMND is directed toward a controller, while RUPT is directed toward a processor.
WR-LM-CNFG	stall before entering this step until all external request activity has subsided; then initiate a request via the Local Memory interface. Send OUTR(00-15) to the Local Memory Configuration Register. See ???for definition.

As shown in Table 4-7 Part 5, the MG field provides two memory write micros.

TABLE 4-7  
MEGABUS MICROS (PART 5)

micro	description
WR-MEM-BYTE	stall before entering this step until all external request activity has subsided; then initiate a memory byte write request to the MEGABUS or to the Local Memory If adra/b is even, then outr(0-7) is written; if adra/b is odd, then outr(8-15) is written.
WR-MEM-WORD (FLM)	stall before entering this step until all external request activity has subsided; then initiate a memory word write request to the MEGABUS or to the Local Memory Outr(0-15) is written and adra/b(31) is assumed to be zero!

As shown in Table 4-7 Part 6, the argument FLM, allows two useful variations of the word write micro.

TABLE 4-7  
MEGABUS AND CACHE/LOCAL MEMORY MICROS (PART 6)

micro	description
WR-MEM-WORD (LOCK)	stall before entering this step until all external request activity has subsided; then initiate a memory-write-lock request on the MEGABUS. If the memory module was not locked, then outr(0-15) is written at adra/b, (adra/b(31) is assumed to be zero). NOTE 1: If the memory module was previously locked, no data is written. The coder must test ACK in order to determine which result occurred. NOTE 2: If ack is not received, the memory module was previously locked, indicating the resource was seized by another requester. It is recommended that the UNLOCK indicator be used to determine when to make another attempt. NOTE 3: Any successful LOCK must be followed by either a RD-MEM..UNLOCK or a WR-MEM..UNLOCK micro; otherwise the memory module will remain seized indefinitely.
WR-MEM-WORD (UNLOCK)	stall before entering this step until all external request activity has subsided; then initiate a write-and-unlock memory request on the MEGABUS. Outr(0-15) is written at adra/b (adra/b(31) is assumed to be zero). The memory module is released.

4.7 FLAG CONTROL

There are two groups of flags. Each group has eight elements for a total of sixteen flags. They are: the eight permanent flags and the eight temporary flags.

4.7.1 Permanent flags

Sixteen micros are dedicated to the eight permanent flags. Most permanent flags are dedicated to hardware entities. FLAGP0 is dedicated to testing. FLAGP1 and P2 are dedicated to testing only when FLAGP0 is off. FLAGP5 allows the microcoder to ignore hardware detected faults (e.g., memory parity). Micros which control the permanent flags are mutually exclusive with micros of the form IND:..(loading the indicators) or LD-RA..(loading RAA, RAB, RAC, and RAD) or changing the temporary flags.

An example of a permanent flag micro:

FLAGP3:1 turns permanent flag #3 on.

4.7.2 Temporary flags

Seventeen micros control the temporary flags. Sixteen of them set or clear one flag while the seventeenth clears all eight temporary flags. FLAGT1 has hidden effect in that it participates in controlling the shift end-effects of the ralu. FLAGT6 serves as the data written into the scram and inverts the polarity of the data read from the scram on its way to ind7. Micros which control the temporary flags are mutually exclusive with micros of the form IND:..(loading the indicators) or LD-RA..(loading RAA, RAB, RAC, and RAD) or changing the permanent flags.

An example of a temporary flag micro:

FLAGT4:0 turns temporary flag #4 off.

**4.8 Nibble-shifter control**

The field which controls the nibble shifter is SD. The nibble shifter rotates the 24 bits of the zbus from zero to five places (any multiple of four bit positions) and makes the result available to the dbus, RAD, and/or SHR. The notion of left/right is provided in the dictionary for convenience. The SD field has no micros but instead provides an argument for those micros which allow RAD, SHR, or the dbus to receive the zbus. As an example:

```
SHR:Z(L8)      SHR(08-15) receives ZBUS(16-23)
                SHR(16-23) receives ZBUS(24-31) and
                SHR(24-31) receives ZBUS(08-15)
```

**4.9 Indicator control**

The indicators are stored in two groups: the arithmetic indicators and the miscellaneous indicators. The general structure encourages the coder to manipulate data in one step, storing the result of the manipulation in indicators; and then, in a subsequent step, test the appropriate indicator(s) via either IF... or BR... micros.

**4.9.1 Arithmetic indicators**

There are six arithmetic indicators called ind(0-5). For the sake of performance, the coder is encouraged to store results of computations in indicators for subsequent testing rather than performing dynamic tests. As shown in Table 4-8, two of the arithmetic indicators (ind0,1) are useful only in add/subtract type computations, while the other four are of more general utility.

TABLE 4-8  
ARITHMETIC INDICATORS (PART 1)

INDICATOR	PURPOSE
overflow (ind0)	detects that the carry into alu bit 8 is not equal to the carry out of alu bit 8.
carry (ind1)	detects that a carry out of alu bit 8 occurred.
sign (ind2)	detects that alu bit 8 is on.
zero (ind3)	detects that alu bits 8 through 31 are zero.
dblzero (ind4)	detects that alu bits 8 through 31 are zero and that ind3 is on.
odd (ind5)	detects that zbus bit 31 is on.

Overflow (ind0) lives up to its name only in two's complement arithmetic subtract operations.

**4.9.2 Miscellaneous indicators**

The two miscellaneous indicators are stored in one "register", but may be thought of separately. The first indicator (ind6) is used to detect frame-bound crossings where a frame is a 512-byte contiguous block of memory, starting at any multiple of 512 bytes. The second indicator (ind7), which stores the output of the stop-code ram, is the simpler of the pair. Note that the stop-code ram is addressed by dbus(16-23).

TABLE 4-8  
ARITHMETIC INDICATORS (PART 2)

INDICATOR	PURPOSE
ind6	frame-bound indicator #6 becomes true if dbus bits 08 through 22 do not match zbus bits 08 through 22.
ind7	the stop-code ram indicator stores the output of the stop-code ram (exclusive ored with FLAGT6), as addressed by dbus(16-23).

4.10 ABCD AND OP CONTROL

The ABCD register is a sixteen-bit register intended as long-term storage for interesting nibbles of the procedure stream. Since each of the four nibbles of the ABCD register may be used to address the ARAM and since RAD may also be incremented and tested for its extreme value (all ones), ABCD is an important resource whose use must be carefully planned. The eight-bit OP register and its attendant custom decode prom (see 4.14) provide a powerful table look-up mechanism which allows quick decoding of procedure stream information. The cycle counter is used with OP and the custom decode prom to provide multiple tables (see 4.14). The micros which load or change these resources are shown in Table 4-9.

TABLE 4-9  
ABCD and OP MICROS

micro	description
LD-OP	The OPCOD register receives, at mid-cycle, the pbus; i.e., the byte whose address is in pctr.
LD-CYCLE(AA)	The cycle counter receives, at mid-cycle, the four-bit value of the argument (AA). Note that (AA) determines the address of the a port of the register file and opportunities for conflict with micros of the form R:.. or Y:A'B:F(AA,AB), abound.
INT	RAA(0-3) receives pbus(4-7) and RAD(0-3) receives zeros and FLAGT(0-7) receive zeros.
LD-RAB'C	RAB(0-3) receives pbus(0-3) and RAC(0-3) receives pbus(4-7).
LD-RAB'C'IND	RAB(0-3) receives pbus(0-3) and RAC(0-3) receives pbus(4-7) and the arithmetic indicators receive their alu inputs
LD-RAD(SD)	RAD(0-3) receives shifter(28-31) from zbus rotated by SD.
INC-RAD	RAD(0-3) receives RAD(0-3) plus one.

Note (in the dictionary) that all of the micros which affect ABCD emit from the FL field and are therefore mutually exclusive and that the FL field also controls the indicators and the permanent and temporary flags.

4.11 LOAD CONTROLS

Many registers are found lurking on the ends of buses just waiting for an opportunity to snarf up the data thereupon. These registers are adra, adrb, adrp/pctr, cnfg, outr, shrg, and timer.

4.11.1 Loading adra, adrb or adrp/pctr

Adra may be loaded from the zbus. Adrb may be loaded from the zbus. Adrp may be loaded from the zbus three different ways. Micros which load adrp also load pctr. The applicable micros are shown in Table 4-10.

TABLE 4-10  
LOAD MICROS (PART 1) ADDRESS REGISTERS

micro	description
ADRA:Z	adra(08-31) receives zbus(08-31).
ADRB:Z	adrb(08-31) receives zbus(08-31).
ADRP:Z	adrp(08-31) receives zbus(08-31). (switch APLONG=1) pctr(23-31) receives zbus(23-31)
ADRP:Z	adrp(08-22) receives zbus(08-22)
ADRP:Z	adrp(23-31) receives zbus(23-31) (switch APLONG=0) pctr(23-31) receives zbus(23-31)

4.11.2 Loading registers outr, shrg, and timer)

The micros which load these three registers are shown in Table 4-10 Part 2.

TABLE 4-10  
LOAD MICROS (PART 2) OTR SHRG and TIMER

micro	description
OUTR:D	outr(00-15) receives dbus(16-31)
SHRG:Z(SD)	shrg(08-31) receives zbus(08-31) rotated by SD
TIMER:Z	timer(08-31) receives zbus(08-31)

4.11.3 Loading the Configuration Register (cnfg)

The micro which loads the Configuration register is shown in Table 4-10 Part 3. The eight-bit Configuration Register controls the CUP behavior in certain optional or product dependent ways. All eight bits can be tested. The definition of each bit is given in Table 4-11.

TABLE 4-10  
LOAD MICROS (PART 3) CNFG

micro	description
CNFG:Z	cmdpar receives zbus(28) fcode1 receives zbus(30)

TABLE 4-11  
CONFIGURATION REGISTER

CNFG OUTPUT	DEFINITION
APLONG	If false, ADRP is load and carry is partitioned 15/9 bits and procedure related page faults are not allowed.
APWRAP	Selects 512 or 8192 page size thereby "configures" the procedure-page-fault error detector.
CMDPAR	In addition to address and data parity, a parity bit on the command leads accompanies all MEGABUS transfers.
CNFIGA CNFIGB CNFIGC	Determines the three-least-significant bits of the "Who are you" reply (to function code 26). Can be used by the microcoder to distinguish among product models.
CSTEAL	A mechanism which allows the CUP to behave as a low priority MEGABUS requestor even when plugged into a high priority slot.
FCODE1	A mechanism which responds to special MEGABUS cycle (function code 01) which (re)initiates the QLT regardless of the current Custom Processor state.

4.12 CLOCK CONTROL

A preprocessor to the firmware assembler (see Section Five) determines what speed each firmware step should be, by examining the microcode source. The microcoder seldom has reason to explicitly state the clock speed. For completeness, the clock micros are listed in Table 4-12.

TABLE 4-12  
CLOCK SPEED MICROS

micro	description
C(VF)	very fast clock - 105 nanoseconds
C(HF)	half fast clock - 125 nanoseconds
C(HL)	half long clock - 145 nanoseconds
C(VL)	very long clock - 175 nanoseconds

4.13 STACK CONTROL

The return stack may contain two return addresses. The return stack operates as a last-in-first-out (LIFO) mechanism, allowing for two levels of nesting. The coder may push a firmware address onto the return stack by issuing a PUSH micro with appropriate arguments. The pushed address will occupy whichever return register is not the current stack top. By the beginning of the next firmware step, the address just pushed becomes the stack top; i.e., the stack entry to which the next successful "return" will go. An unconditional return is always successful. A conditional return is successful if the condition is met. The choice as to which return register contains the top of stack is determined by a toggling type mechanism. A toggle occurs each time a push is performed and each time a successful return is performed. An example is shown in Table 4-13.



TABLE 4-13  
PUSH MICRO

micro	description
PUSH(\$SHOVE)	return stack "bottom" receives the literal address of \$SHOVE. Note that during this step, the dbus is occupied with the aforementioned literal.

See 4.14 for a description of unconditional, conditional and alternate returns.

4.14 NEXT-ADDRESS CONTROL

Five fields control which of the 16384 firmware locations is next to be executed. Before each field is discussed, it is first necessary to expose some general information.

Firmware steps are identified by a 14-bit "control store address" (CSA) and optionally a mnemonic label. In the discussion that follows, "CSA" is used instead of "CSA/LABEL" but it should be understood that restrictions applicable to a CSA (e.g., must be an IF-bank address) are equally applicable to the label.

The 16384 location firmware array is divided into two banks. The first 8192 locations, 0000 through 1FFF (hex), constitute the ELSE bank. The last 8192 locations, 2000 through 3FFF (hex), constitute the IF bank.

Firmware sequencing in the Custom Processor is never implicitly nor arithmetically determined. Every step specifies its successor or choice of successors. There are numerous ways in which the coder may specify what firmware location is next to be executed and they are best exposed by explaining each of the five fields which control firmware sequencing as shown in Table 4-14.

TABLE 4-14  
NEXT ADDRESS CONTROL FIELDS (PART 1)

FIELD	PURPOSE
TC	<p>test condition - a six-bit field which allows the coder to specify that the next step shall be a location in the IF bank or a location in the ELSE bank as determined by the result of testing one of sixty-four unary indicators. (e.g., IF-I-ZRO(\$FAIR,\$RAIN) In this example the addresses of \$FAIR and \$RAIN are required to have the following relationship:</p> <ol style="list-style-type: none"> <li>1. \$FAIR must be in the IF bank</li> <li>2. \$RAIN must be in the ELSE bank</li> <li>3. Their addresses may not be different except in the most-significant bit (from 1 and 2 above) and the four least-significant bits. (i.e., if \$FAIR is in a particular "block" of sixteen locations in the IF bank, then \$RAIN must be in the corresponding sixteen-location block in the ELSE bank.</li> </ol> <p>The result of all this is that if the arithmetic indicator I-ZRO was on, \$FAIR will next be executed; if the arithmetic indicator I-ZRO was off, \$RAIN will next be executed.</p>

TABLE 4-14  
NEXT ADDRESS CONTROL FIELDS (PART 2)

FIELD	PURPOSE
BI	branch invert - a one-bit field which controls the polarity of the test condition.
NA	next address - a thirteen-bit field which, in its simplest form, specifies which of the 8192 locations of the bank specified by BI is next to be executed, as in the micro GOTO(\$THEDEVIL).
MK	mask - a four-bit field which serves many purposes. One of the purposes is best described here while the others can wait for the BR field:  When the coder has specified a simple choice between IF and ELSE, MK identifies which of the four least-significant bits of the alternative addresses differ. For instance if, in the previous example \$FAIR had been allocated CSA 2340 and \$RAIN had been allocated CSA 034F, the assembler (not the coder) would set the value of the MK field to F indicating that all four least-significant bits of the alternative addresses differ and the hardware does the rest.
BR	branch - a three-bit field which determines how the succeeding firmware step shall be chosen. It permits the coder a veritable plethora of mechanisms for deciding the value of the next CSA as shown in Table 4-13.

TABLE 4-15  
NEXT ADDRESS MECHANISMS (PART 1)

MECHANISM	DESCRIPTION
GO-TO(\$SNOW)	specifies an unconditional successor
IF-Z24(2345#,0342#)	specifies a two-way choice as a function of the value of zbus bit 24. If Z24 is on, then location 2345 is executed next, otherwise location 342 is executed next.
RETURN	specifies that the succeeding step is that CSA at the top of the return stack (i.e., an unconditional return).
RETURN'(9)	is a "masked" type return which specifies that the succeeding step is that CSA at the top of the return stack but with the two weight bit and the four weight bit set equal to zero (masked by FFF9). Here the MK field contains the four-least-significant bits of the mask and the hardware sets the other mask bits to ones.

TABLE 4-15  
NEXT ADDRESS MECHANISMS (PART 2)

MECHANISM	DESCRIPTION
IF-ACK RETURN(\$DONT)	<p>specifies that the succeeding step is that CSA at the top of the return stack only if the ACK indicator is on. If the ACK indicator is off, the succeeding step is \$DONT. Note that \$DONT must be in the ELSE bank, thus CSA's pushed onto the stack are IF bank addresses. Note also that a conditional masked return of the form: IF-ACK RETURN'(A,\$DONT) is also possible.</p>
BR-ARITH(C,3210#)	<p>specifies that the succeeding step is to be one of four as a function of the condition of two arithmetic indicators; namely, I-SGN and I-ZRO. If they are both off, location 3210 will be next executed; if I-SGN is off and I-ZRO is on, location 3214 will be next; if I-SGN is on and I-ZRO is off, location 3218 will be next; if they are both on, not possible except in examples, location 321C will be next. Note that this mechanism permits the examination of two other arithmetic indicators. By specifying a mask with a two-weight bit, CARRY08 (the dynamic alu carry) may be examined, and by specifying a mask with a one-weight bit, the bit shifting out of Q31 may be examined. Note also that a conditional version, of the form: IF-Z'Z BR-ARITH(4,\$CLOUDY) is also provided.</p>
BR-FLAGS(F,3450#)	<p>specifies that the succeeding step is to be one of sixteen in the block of if-bank locations starting at 3450 as a function of four temporary flags (to, t1, t2, and t3). Note that a conditional version of the form: IF-I-SCR BR-FLAGS(6,\$BANG) is also provided.</p>
BR-P0(D,2AB0#)	<p>specifies that the succeeding step is to be one of eight in the block of addresses starting at location 2AB0 as a function of pbus bits 0, 1, and 3. Note that a conditional version of the form: IF-FLAGP1 BR-P0(E,\$WET) is also provided.</p>

TABLE 4-15  
NEXT ADDRESS MECHANISMS (PART 3)

MECHANISM	DESCRIPTION
BR-RAMAD(5,\$SLEET)	specifies that the succeeding step is to be one of four in the IF-bank block in which \$SLEET resides, as a function of ramad1 and ramad3. Note that a conditional version of the form: IF-NOT-RAD:F BR-RAMAD(A,\$SLUSH) is also provided.
BR-OPERAND(F,\$SUNNY)	specifies that the succeeding step is to be one of eight in the IF-bank block in which \$SUNNY resides as a function of register RAA(1-3). Note that a conditional version of the form: IF-RUPT BR-OPERAND(F,\$SMILE) is also provided.
BR-DECODE(F,\$BEGIN)	specifies that the succeeding step is to be one of 16 IF-bank locations in the \$BEGIN block. The cycle counter participates by supplying its low-order three bits to a table look-up mechanism which contains eight tables. The value from the cycle-counter selected table becomes the output of the opcode decode unit. This splatter mechanism is intended to permit 8 different interpretations of procedure stream bytes (e.g., based upon position).

#### 4.15 AVAILABILITY

The Custom Processor has many availability features which insure that processing proceeds error free. Some of these features test parity of incoming data, some check integrity data delivered to the Custom Processor from other system elements, some test the validity of the control-store array, and some generate parity to accompany output data, so that other system elements may verify that the data sent arrived intact. The list of availability features follows:

1. data and procedure parity checks - whenever information is sourced from an input data register (e.g., Z:INRA) or from a procedure buffer (e.g., PTAKE), parity is checked on the appropriate bytes.
2. data and procedure edac errors - whenever information is sourced from an input data register (e.g., Z:INRB) or from a procedure buffer (e.g., D:PROC), any uncorrectable memory error associated with the information is detected.
3. data and procedure unavailable resource checks - whenever any attempt is made to communicate with another system element (i.e., a memory, a controller, or a processor), circuitry detects the lack of a response and signals an unavailable resource error.

4. control-store array checks - a parity bit is imbedded in each 32 bits of control-store data. During each firmware step executed, three separate parity checks are performed to validate the integrity of the control-store read-out.

If any of the above conditions occur, the syndrome register samples its inputs (reloads) and, if FLAGP5 is off, the \$SDSASTR routine at location zero is executed. The \$SDSASTR routine differentiates these entries from "master clear". If a master clear is detected, the QLT is entered; otherwise, the \$SDSASTR routine exits to \$SYSERR. The routine \$SYSERR is supplied by the specific firmware application. When the CUP is a co-processor, the \$SYSERR routine notifies the "other" processor that a hardware detected failure has occurred and that remedial action is required. Note that among other things, an out of range memory address can result in an invocation of \$SYSERR which might be the result of a "program" in a stage of development. Thus \$SYSERR may not be catastrophic. Further, before or instead of notifying the KERNEL processor, certain error conditions can be retried if the firmware is properly structured for such an eventuality.

Other features are intended to improve the chances that the correct unit will be identified when a failure is detected:

1. parity generation - parity generation for the control-store array is performed by the assembler. Parity generation for data destined for other system elements via outr is performed when outr is sent to the MEGABUS or to the Local Memory. In order to verify the integrity of the parity generation and checking circuits, a mechanism is available in test mode for generating both even and odd parity.
2. self testing - as part of every system initialization, the Custom Processor executes a sequence of firmware routines whose purpose is to detect any hardware fault, either in the processor or in any cup-dedicated memory subsystem. This routine utilizes the integrity features mentioned above and takes advantage of the syndrome register which captures the reason code for any hardware-detected error.

#### 4.15 ALTERABLE FIRMWARE ARRAY

The RAM array provides loadable firmware space. Once the RAM array is loaded, firmware execution may proceed from either the PROM array or the RAM array.

The RAM array is an independent 16k x 96 bit memory; i.e., the total available firmware space is 32k x 96. RAM execution and PROM execution run equally fast; i.e., a very-fast (VF) step takes 110ns. The CUP enables the PROM array upon being initialized; i.e., Power-on, Master Clear, or Function 01.

The CUP need contain only a minimum PROM space (e.g., 4K) since when initialized (forcing PROM execution), the Custom Processor executes the self-test routine (from PROM space) ending in Test Idle. A Test-Idle PROM routine responds to a RAM-Load command (function code), which causes a portion of memory to be copied to RAM space.

The RAM array may then be enabled. While executing in RAM space, a micro may be issued to return execution to PROM space.

4.15.1 Loading the RAM array

The RAM array is loaded when a Function Code 29 is received. The accompanying address locates a structure containing two addresses and a range as shown in Figure 4-3.

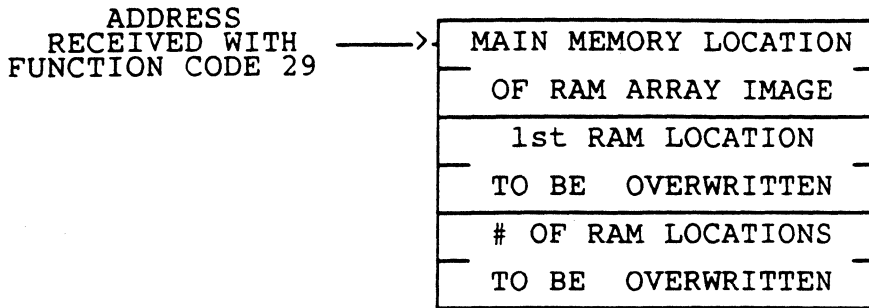


FIGURE 4-2  
STRUCTURE "PASSED" WITH FUNCTION 29

The RAM array may only be written when execution is proceeding from the PROM array. One micro (FRAMIT) causes three types of actions to occur as a function of the value contained in CYCLE(4,2,1) as shown below:

- o For each firmware location to be written, a ninety-six bit Firmware Write Register (FWR) is loaded in six sixteen-bit portions for CYCLE(4,2,1) decodes of 2 through 7.
- o The Firmware Address Register (FAR) is loaded when CYCLE(4,2,1) decode is 1.
- o The 96-bit Firmware Write Register is copied into the selected RAM location when CYCLE(4,2,1) decode is 0.

Table 4-16 describes each of the eight combinations of CYCLE(4,2,1) when the FRAMIT micro is issued.

TABLE 4-16  
FRAMIT ACTIONS

CYCLE 8 4 2 1	DEFINITION
X 0 0 0	Write all 96 bits of Firmware Write Register into the firmware RAM array at the location specified by the Firmware Address Register
X 0 0 1	Load the Firmware Address Register from BUSD(18-31)
X 0 1 0	Load Firmware Write Register bits 00-15 from BUSZ(16-31)
X 0 1 1	Load Firmware Write Register bits 16-31 from BUSZ(16-31)
X 1 0 0	Load Firmware Write Register bits 32-47 from BUSZ(16-31)
X 1 0 1	Load Firmware Write Register bits 48-63 from BUSZ(16-31)
X 1 1 0	Load Firmware Write Register bits 64-79 from BUSZ(16-31)
X 1 1 1	Load Firmware Write Register bits 80-95 from BUSZ(16-31)

#### 4.16.2 Array Jumps

Two micros are provided in order to switch firmware execution from one array to another. A switch from PROM execution to RAM execution is accomplished by issuing the micro ENBRAM and a switch from RAM execution to PROM execution is accomplished by ENPROM.

The micros ENBRAM and ENPROM which can jump from one array to another have three special properties:

1. After Power-On, Master Clear or Function Code 01, execution always proceeds from PROMs.
2. They perform no action if firmware execution is already in process from the array to which the jump was intended.
3. They take effect after the completion of the next firmware step.

## THE NANOSECONDS

The assembler rejects micro-op combinations which would result in a clock speed outside the range of the longest clock setting. The purpose of this section is to help explain how the "gear" is selected.

The diagram of 5-1 is a representation of the custom processor oriented toward the subject of nanoseconds. In essence, the assembler contains a model which depicts the information of figure 5-1 along with a definition of each clock speed (i.e., 110ns, 120ns, 140ns and 170ns).

The subject shall be approached as follows:

- first, a detailed explanation of the symbols used in figure 5-1;
- second, an explanation of why some paths have multiple symbols and, in general, how the reader handles choices;
- and last, an example of how one converts a set of micros into a number of nanoseconds.

### 5.1 the symbology

Clearly, for performance reasons, it is desirable for every step to be a VF step. Figure 5-1 illustrates those paths which might prevent the desirable but does not illustrate those paths which cannot impact the clock. An example of this point is shown in the table labeled "next address". Notice that only three of the sixty-four test conditions are listed. This is because each of the others will succeed at performing its appointed function (bank selection) in no more than 110ns.

Numbers in squares or non-squares are nanoseconds and abide by the following rules:

1. squares contain the number of nanoseconds from the beginning of the firmware step until the data, at the point where the square is located, is valid. This number means something only if the micro combination in the step being analyzed requires this data path, and can be ignored otherwise.
2. non-squares (mostly circles) contain the number of additive nanoseconds the data must pay in order to propagate through.



5.2 the choices

Choices are essentially of two types:

1. the square  
the nanosecond value from the beginning of the firmware step may or may not be determined at any arbitrary point in the diagram.

A simple example of this is the zbus. Since data comes to the zbus from multiple sources, one may have to backtrack as a function of the micro combination being analyzed. The number inside a square is the number of nanoseconds from the beginning of the firmware step. In the case of the zbus, if the micro Z:INRA were coded, then the zbus is valid at nanosecond 35, but if the micro Z:Y is coded, then one must backtrack, examining other micros to determine when the alu will emit information. If the alu is not receiving from the dbus, then the zbus validity can be determined by analyzing only the micros relating to the alu (i.e., R:..., F:..., and Y:....). On the other hand, if the alu is receiving from the dbus, one must backtrack further.

2. the non-square (circle)  
the number in a circle is the number of nanoseconds which must be added to the current total in order to pass this point if, of course, the micros coded require this path.
3. the non-square (non-circle)  
these symbols generally contain conditions. A simple example is the path through the alu oval. If the alu is required by the F:... micro for an arithmetic operation, then the data must pay 67.5ns to pass; otherwise the cost is only 25ns.

5.3 putting it together

In the step:

```
R:A'S:B(0,0)
F:ADD
IND-AR
```

The first micro specifies the two alu sources to be the two ports of the register file. The second micro specifies that the alu shall add. The third micro specifies that the arithmetic indicators shall be loaded.

In figure 5-1, the rectangle labeled "ram" is signified to have its output valid at nanosecond 24 (from the beginning of the step). From there, the alu is encountered where, because an add is to be performed, the 24 must be increased by 67.5ns. Leaving the alu and heading for the indicators, it is necessary to select another addend which, since the alu operated arithmetically, is 15ns. The subtotal (24+67.5+15) is 106.5ns which, to arrive at the total, must be increased by 10% to account for non-silicon delays (i.e., media). This yields 117.15 which requires a clock setting of 120ns; namely, an HF "box".

In a more complicated step, where there are multiple end points, the length of the step is determined by the longest path. Each path must be analyzed as shown below:

MICRO		CUMUL
RD-MEM-DBLW	0.0	
R:A'S:B(2,3) F:S-R Y:A'B:F(2,3)		to register file 91.5*
Z:Y		to zbus 52.0
IND-AR	106.5*	
REG:Z(RD)	77.0*	
ADRA:Z	57.0*	
SHRG:Z(L4)	69.0*	
D:FFZZ(L4)		to dbus 73.0
OUTR:D	78.0*	
IF Z24 BR-ARITH(2,\$GWHIZ)		bank select 119.5*
		if bank address 151.0*

\*=end point

In this step, the memory reference initiation micro (RD-MEM-DBLW) has no impact on the length of the step (even though it would have stretched the previous step if megabus/cache activity were still in progress).

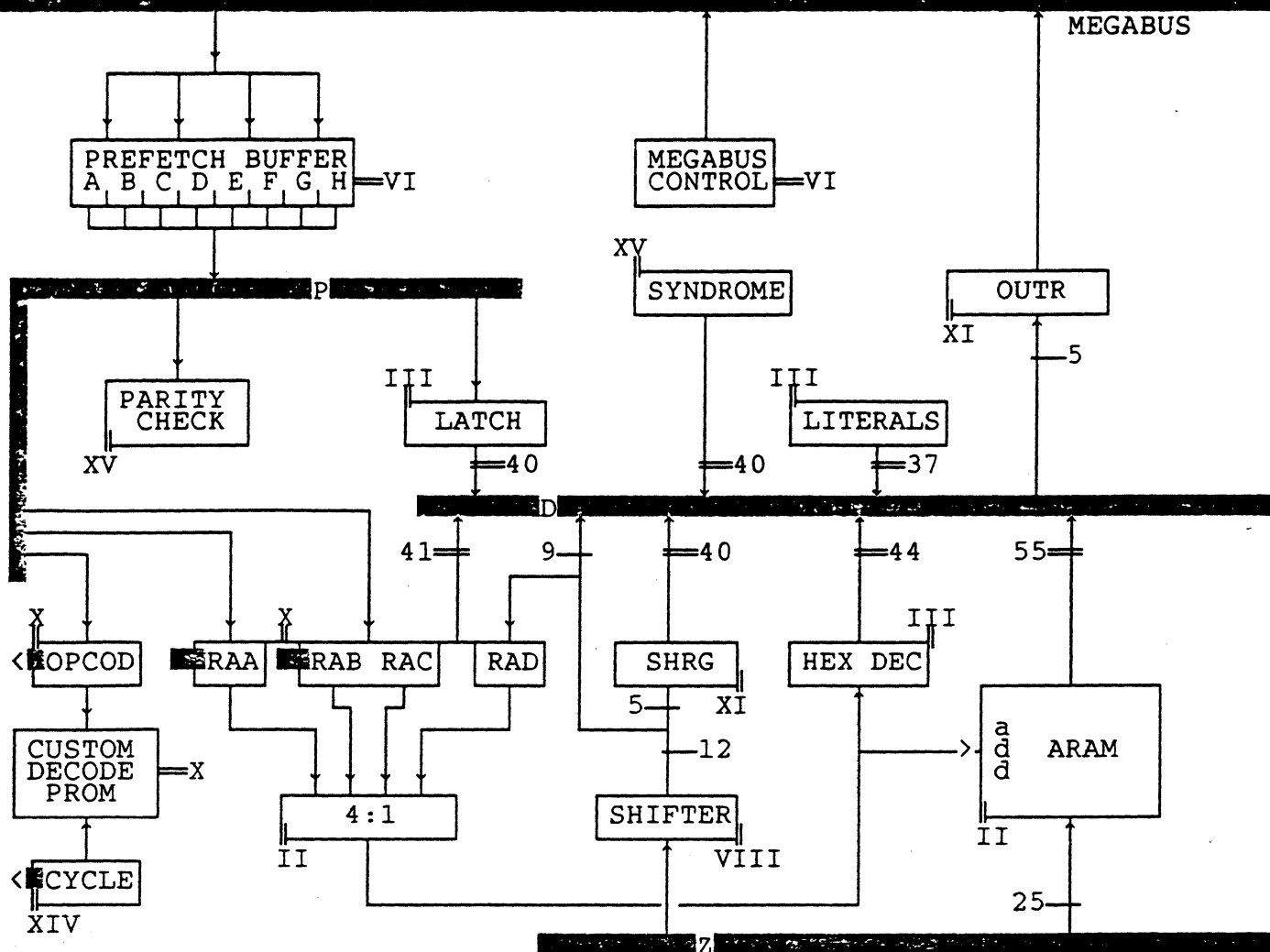
The R and S sources for the alu are F2 and F3 respectively and the alu is directed to subtract, placing the difference back into F3. For this activity, the register file is read out, arriving at the inputs of the "adder" at ns24, getting themselves subtracted and returned to the register file in 67.5ns, attaining this end point in 91.5ns(24.0+67.5). The micro IND-AR will retain certain properties of the difference at a nominal cost of 15ns; the register file contributes its 24.0, the alu contributes its 67.5 and the indicator mechanism contributes 15.0 because the alu is performing an arithmetic operation (as opposed to a logical operation; e.g., XOR). The total (24.0+67.5+15.0) is 106.5ns to attain this end point.


In the meantime, the register file A port is sent to the zbus arriving there at ns52. The zbus data is first sent to the ARAM, attaining this end point at ns77 (52.0+25.0); then gets shifted left one nibble and plopped into SHRG, attaining that end point at ns69.0 (52.0+12.0+5.0).

The shifted zbus also heads straight for the dbus arriving there at ns73 while the eight fill bits on the upper byte of the dbus were emitted from the literal generator at ns37, and have been waiting there patiently ever since (i.e., not in the critical path). The 73ns dbus data is sent to OUTR and arrives at that end point at ns78 (52.0+12.0+9.0+5.0).

The next-address computation involves both a bank select (IF...) and a low speed IF bank splatter (the alu carry out). The bank select starts at the zbus (ns52.0), then 5.5ns for the "if" and 62.0ns to access the control store proms yielding 119.5. The splatter branch yields 89.0 to determine the alu output carry and 62.0 to access the prom array yielding 151.0 ns.

Clearly, the splatter contributor is the longest path which, after adding 10% for "media", results in a total of 166.1ns, or a VL "box".



 denotes a register which clocks at midcycle. The register cannot participate in a next address decision during a step in which its contents is changing.

NEXT ADDRESS	
MICRO	ADDED ns
IF Z	+5.5
BR-OP(8)	+7.0
BR-ARITH(2)	+89 or D+65
BR-ARITH(1)	+60.5

28SEP87

16-BIT CUSTOM PROCESSOR