THE CUSTOM PROCESSOR SERIES

TECHNICAL DESCRIPTION OF THE 32-BIT CUSTOM CENTRAL PROCESSOR

TABLE OF CONTENTS

Throughout the text of this document, the term "custom processor" shall be interchangeble with the term "Ultimate Processor".

TABLE OF CONTENTS

S E C T I O N III

INTRODUCTION

The thirty-two-bit custom processor is a ten megahertz, thirty-two-bit wide, microprogramable firmware engine driven by a one-hundred and twenty-eight bit wide control store word and having a blank identity.

1.1 PURPOSE

This tchnical description imparts information which is necessary for any who wish to provide the custom processor with a new incarnation. Those who attempt personalization of the custom processor need be capable of writing and testing microcode. For testing microcode, Custom and Special Products offers a Firmware Development Facility which greatly simplifies the task (see appendix A).

1.2 SCOPE

This document is intended for the prospective microprogrammer. It describes the operation of the 32-bit custom processor at the level of an experienced coder. Others, such as test technicians, might also find the information in section three useful.

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

Section II describes the system environment, an exposure of the custom processor capabilities and a brief discussion of each of its major areas.

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

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

Section V is a discussion of internal speed considerations.

Appendix A contains a description of the Firmware Development Facility which is available for the checkout of firmware written for the Custom Processor.

Appendix B contains a description of the Quality Logic Test firmware which is available for linking into the custom firmware load and is recommended as a means of verifying the integrity of the Custom Processor at every system initialization.

Appendix C is a description of the Test and Verification routine available with every incarnation of the Custom Processor.

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 CUP32, the following additional documents may prove useful:

CUP32 dictionary

CUP32 logic block diagrams

for the mother board ..60160259

for the daughter board 60160249

RTL6 assembly language manual

document #LDA-021

OPERATIONAL OVERVIEW

This section describes the system environment into which the 32-bit Custom Processor may be connected. The section also gives a first-level description of the Custom Processor's inner workings.

2.1 SYSTEM ENVIRONMENT

The 32 bit Custom Processor is housed in a central subsystem chassis (see figure 2-1). This chassis contains an interconnect backplane called the Local Bus which allows communication among elements within the central subsystem chassis. A minimum of two elements must be installed into the central subsystem chassis: namely, a 32-bit Custom Processor and a tri-port memory. One port of the tri-port memory connects to a Custom Processor on the Local Bus, another port connects via a Custom MEGABUS Adapter board to a 32-bit MEGABUS System Bus, and a third port is available to connect to a second Custom Processor on the Local Bus. The Custom MEGABUS Adapter board physically plugs into the 32-bit System Bus and a set of cables connect it to the tri-port memory. The central subsystem chassis is designed to accommodate a total of two 32-bit Custom Processors and two tri-port memories. In configurations where two tri-port memories are installed, each tri-port memory is connected via cables to its dedicated Custom MEGABUS Adapter board. The Custom MEGABUS Adapter board provides a Custom Processor access to System Bus elements including peripheral controllers, communication controllers, central memories, and other processing elements (e.g., a Series 6 CPU).

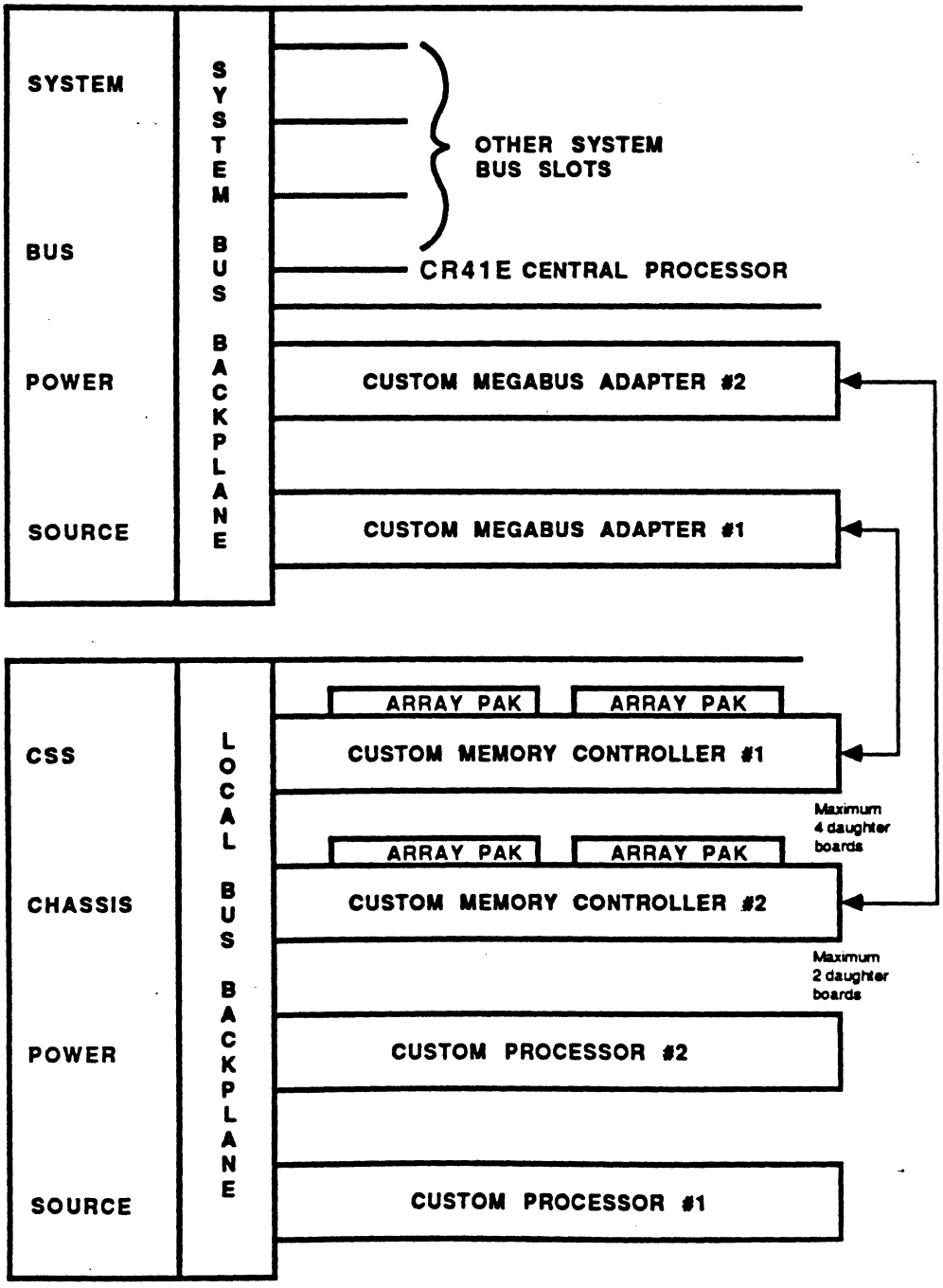
Each tri-port memory may be configured to contain either two-megabytes or four-megabytes of memory. When more than one tri-port memory is installed, the total address space is contiguous. Figure 2-2 illustrates possible memory configurations which are supported. Note that it is possible to connect another complete central subsystem chassis to the System Bus for a maximum of four 32-bit Custom Processors and four tri-port memories.

The 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.

Architecturally, the system is strategized to minimize System Bus memory traffic in support of a processor's instruction stream. In these systems memory traffic on the System Bus will service direct memory access(DMA) almost exclusively. The only other System Bus traffic will be that programmed I/O dialogue required to institute and control DMA and that interprocessor mailbox dialogue necessary for sharing system resources.

ULTIMATE

CONFIGURATION



SYSTEM ENVIRONMENT

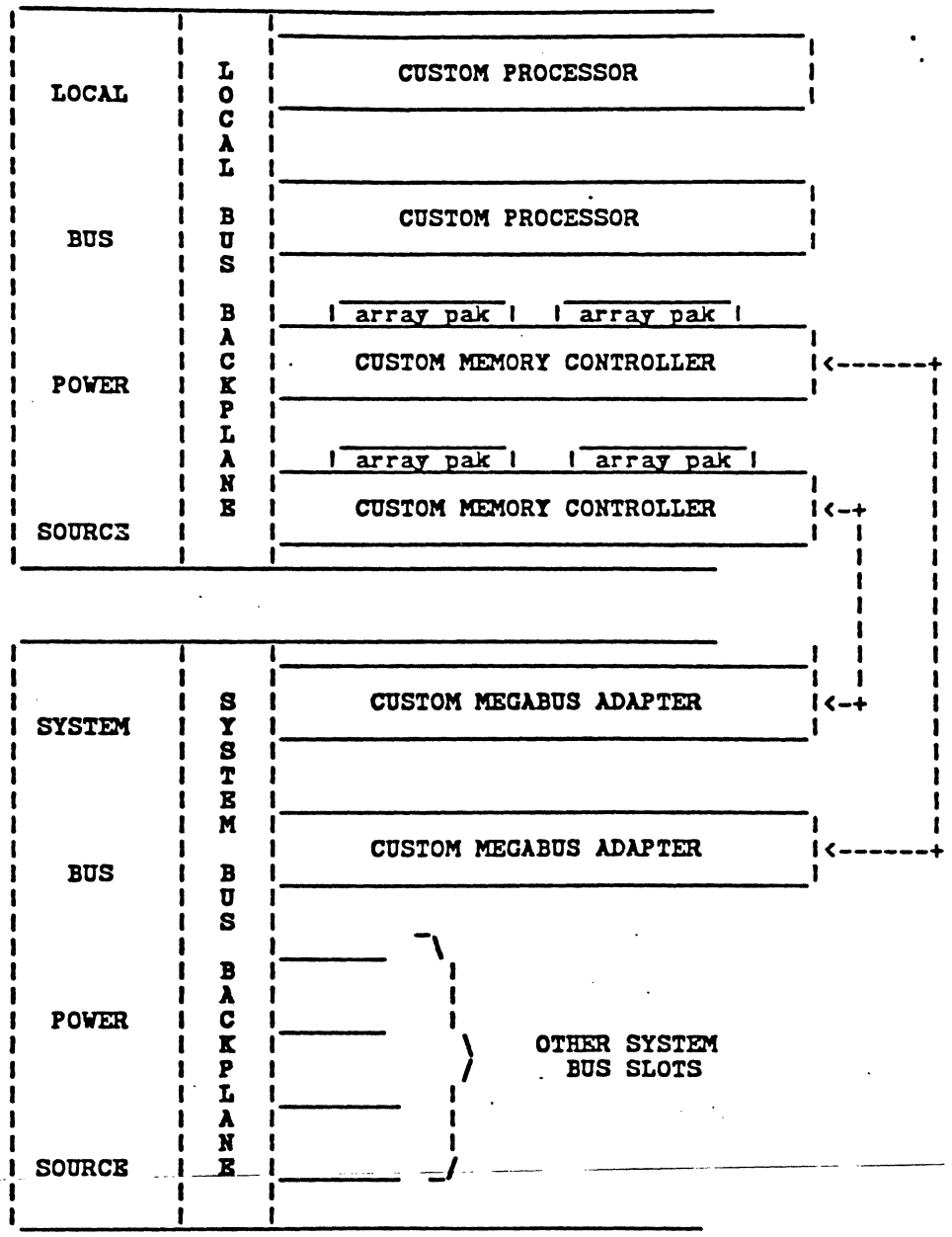


FIGURE 2-1

SYSTEM ENVIRONMENT

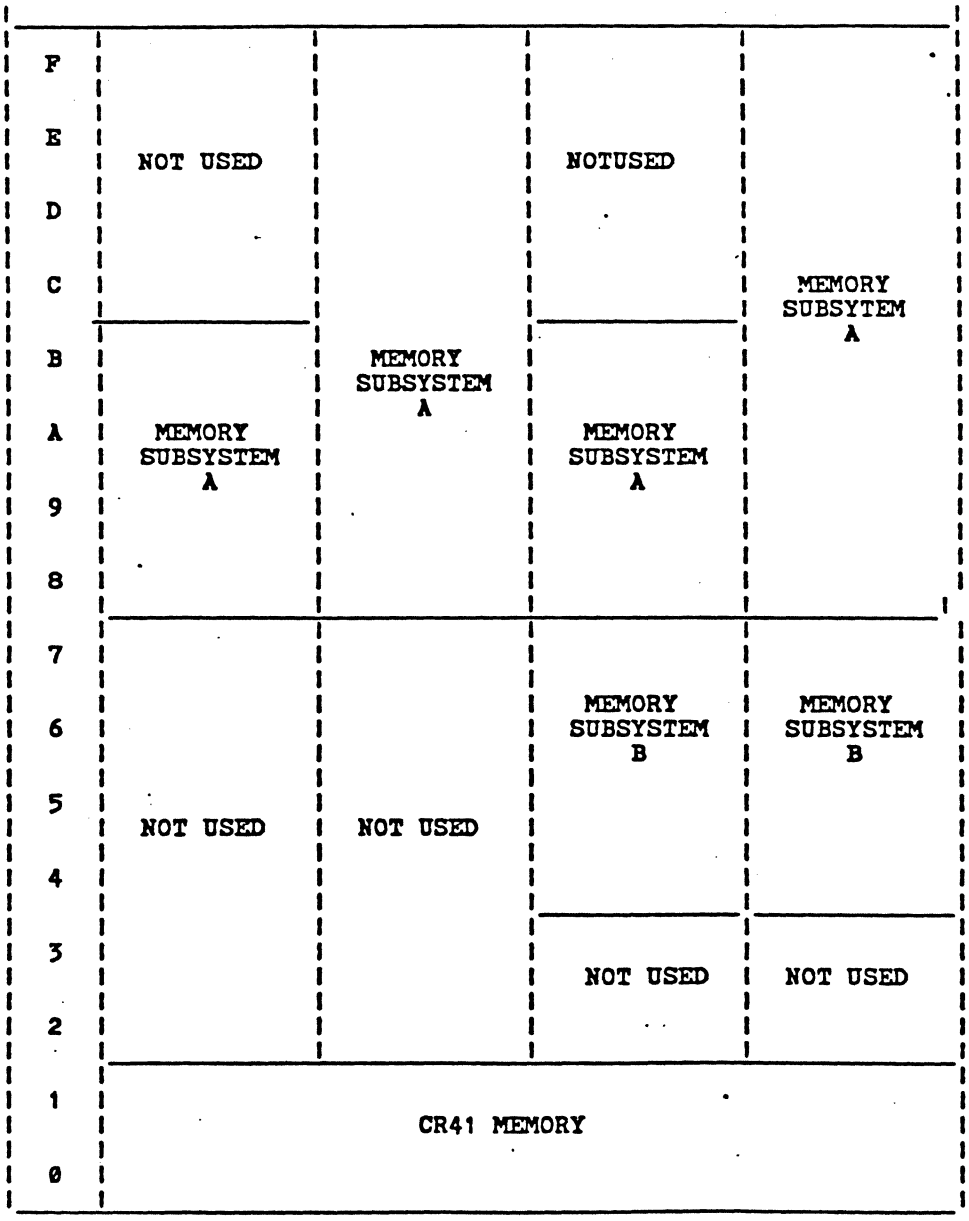


FIGURE 2-2A

MEMORY CONFIGURATIONS FOR THE 24-BIT ADDRESS SPACE

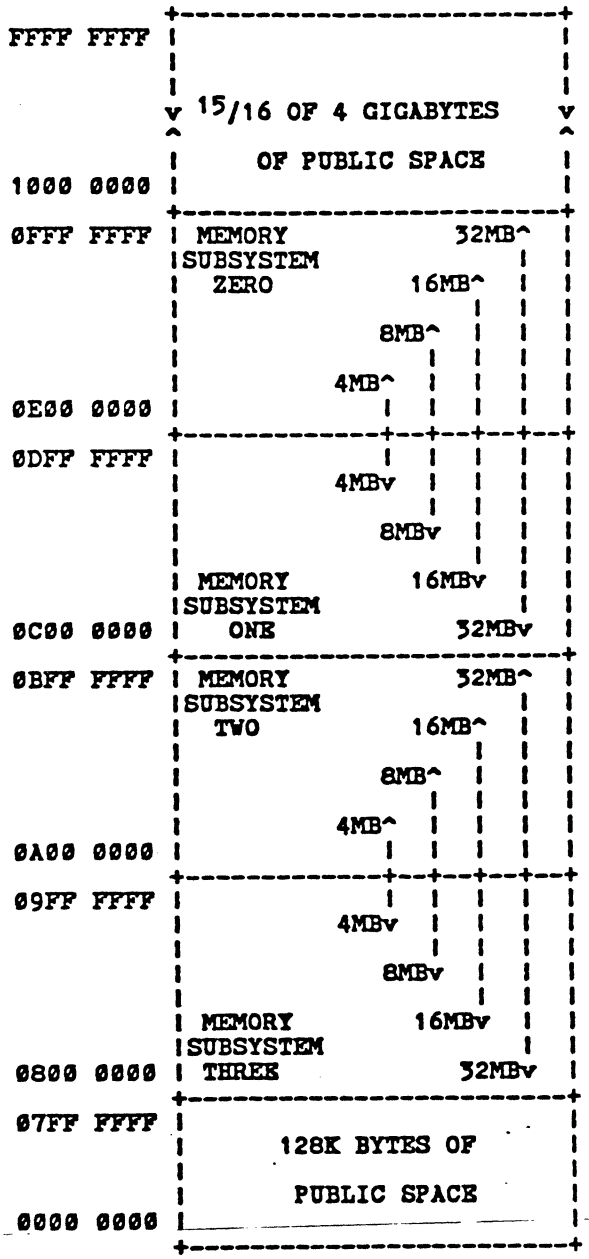


FIGURE 2-3B

MEMORY ADDRESS ALLOCATION FOR 32-BIT ADDRESS SPACE

2.2 INTRODUCTORY DESCRIPTION

The major block diagram of figure 2-3 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 their associated block diagram identifier:

I. the ralu

II. the aram

III. the d bus

IV. the s bus

V. the z bus

VI. the Local 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 eight 2901 bit slice chips constituting a thirty-two-bit alu, a sixteen-location dual-ported random- access memory and a bit shifter (32 or 64 bits wide). Operations inside the alu occur at nearly a ten megahertz rate. The ralu receives external thirty-two-bit data from the dbus and transmits thirty-two-bit results to the zbus.

2.2.2 the aram (see block diagram identifier II)

The auxiliary random access memory is a single-ported 4096-location memory. Each location contains thirty-two bits. The memory may be addressed in any one of seven different ways. Its data is read onto the sbus. Data written into the aram is taken from the sbus. Writing of the aram is byte partitioned; that is, any combination of 1, 2, 3, or 4 bytes may be "copied" from the sbus into the addressed aram location.

2.2.3 the dbus (see block diagram identifier III)

The dbus is the place where all roads lead. It is a thirty-two bit bus which receives data from the sbus, the zbus, the pbus and numerous other secondary sources. It may deliver its wealth to the ralu, the output register, and to the zbus. It has byte partitioning capabilities; e.g., it may take a byte of the aram for its upper eight bits, an h register byte for its next eight bits, a literal for its next eight bits, and a bunch of zeros for its last eight bits.

2.2.4 the sbus (see block diagram identifier IV)

The sbus receives and provides aram data. It receives and provides data for the three Local Bus address registers. It has the unique capability of receiving and reflecting two of its sources during the same firmware step by virtue of a time-multiplexing mechanism.

2.2.5 the zbus (see block diagram identifier V)

If the dbus is where all roads lead and the sbus is hermaphroditic, the zbus is totally colorless. Indeed, it is a journeyperson bus, capable of receiving data from the outside world (the memory subsystem), receiving ralu revelations and sending all this to the v register, and/or to the dbus, and/or to the sbus via the nibble shifter.

2.2.6 the Local Bus (see block diagram identifier VI)

The Local Bus is the area of the processor responsible for communicating with the memory subsystem and through it, via the System Bus, to all other system elements; e.g., a Series 6 processor or a Series 6 controller. The Local Bus area contains, along with the interface circuits required to carry on a dialogue upon the Local Bus, an eight-byte look-ahead procedure buffer and two four-byte data buffers. The two data buffers receive information from memory to be deposited onto the zbus. The look-ahead procedure buffer is organized to supply one or two bytes to the op register and/or the dbus and automatically replentish the prefectch buffer as bytes are consumed.

2.2.7 flags (see block diagram identifier VII)

The Custom Processor contains twenty-four 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 sbus. All eight possible 4 bit shift distances are supported e.g., shift left four nibbles, shift right four nibbles and "word swap" are three 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" and "carry" which are provided at any byte partition and there are more specialized indicators like the leading-zero detector. Many have the ability to participate in "splatters" which permits 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 may also receive nibbles from the sbus; they contribute to aram addressing, they may be incremented or decremented, they may be tested for crossing the zero boundary, and they may participate in 17 way branches.

2.2.11 other registers (see block diagram identifier XI)

Many other registers provide strategically located storage for information. Address registers for data and procedure fetches, which can be incremented or decremented, are loaded from the sbus and communicate their content to the memory subsystem. The output data register is loaded from the dbus and communicates its content to the memory subsytem. The v register is connected bidirectionally to the zbus and is a convenient stopover for zbus data. The h register, one of the byte partitionable sources to the dbus, and the accounting timer are also loaded from the sbus.

2.2.12 clock (see block diagram identifier XII)

The clock has a maximum frequency of nearly ten 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 seventeen levels. Absolute or relative addresses may be "pushed" onto the stack. Unconditional, conditional and 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 seventy-two test conditions. It may instead be chosen among sixteen locations dependent upon some group of four indicators (six groups are provided) or the destination may be a seventeenth location dependent upon one of the seventy-two test conditions. Then again, it may be chosen among one of 256 locations dependent upon a byte of the procedure stream via a table look-up mechanism containing sixteen look-up tables. Or, it may be a subroutine return. Three types of returns are provided: an unconditional return or one which returns depending on one of the seventy-two test conditions or one which returns to an alternate return location as a function of subroutine processing discoveries.

2.2.15 availability

The Custom Processor is possessed of data parity checking circuits, data uncorrectable memory edac error detectors, and firmware parity error detectors. It can also detect references to unavailable system resources, and inappropriate return stack references. 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 and memory subsystem hardware elements verifying their specified operation. This firmware routine includes a thorough memory array test.

major block diagram goes here

HARDWARE DESCRIPTION

This section describes hardware entities at a level of detail sufficient for comprehension if the reader has a set of custom processor schematics (LBD's) 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 2-3.

The hardware discussion will be subdivided into fifteen 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 iv.

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. S bus

V. Z bus

VI. Local bus

VII. Temporary, permanent and control flags

VIII. Nibble shifter

IX. Arithmetic and miscellaneous indicators

X. OP register and OP register multiplexers

XI. Load of H, V and other registers

XII. Four-speed clock

XIII. Return stack

XIV. Next address generation

XV. Availability

In the text which follows, capitalization is used sparingly so that all signal names and micro names may stand out. Signal name polarities are avoided wherever possible; instead the terms "on" and "off" are used. When, for instance, REQNOW is "on", REQNOW+ is high and REQNOW- is low. This applies even when the signal exists in only one polarity (e.g., "MCX000 is on" means MCX000- is low and would also mean MCX000+ is high were there such a signal. A neutral polarity indicator "." is used for those signal names which achieve uniqueness only in their eighth and ninth characters (e.g., NAEB13.EX).

3.1 Register file and alu (block diagram identifier I)

The ralu is the resource which performs thirty-two-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 15d)

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

1. a dual-ported 16-location by 32-bit register file
2. a 32-bit arithmetic and logic unit
3. a 32-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 page 12d)

The carry look-ahead network is comprised of 74S182's connected in a somewhat unconventional 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:

AF3	AF4	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 2901 is from the first's ripple carry. Then the 74S182's take over, providing input carries for the most significant six 2901's. Last, the carry out of the most significant stage (AUC000) is derived by combining the G and P outputs of the most significant 74S182 chip using a couple of inverters and a 74S51.

The zero detector of the 2901 is an open-collector output requiring pullup. Eight resistors are required since the leading-zero indicators must examine each individual nibble even though the zero arithmetic indicator is partitioned only to the byte.

The signal ALUTOZ is derived as shown (predecoded) in order to enable/disable the 2901's to the zbus in a timely manner (see 5.0).

3.2 AUXILIARY RANDOM ACCESS MEMORY - ARAM (block diagram indentifier II)

The auxiliary random access memory is comprised of eight 1421-45's. Each chip contains 4096 locations of four bit wide static ram. The aram receives a twelve-bit address from an eight-way selector called rmad. The aram may be read/written to/from the sbus.

3.2.1 Aram read or write (see lbd pages 17d and 19d-26d)

When written, the aram is byte partitioned. When read, 32 bits of data are placed upon the sbus for normal reads. The signals RAMWCE, RAMXCE, RAMYCE, and RAMZCE are the four chip enables. They are structured simply to provide the split cycle sbus capability having a gate for first half cycle,a gate for second half cycle and a gate to "bridge" the early and late gates when the ARAM is the sbus source for the full cycle. The four signals ARM0WR, ARM1WR, ARM2WR, and ARM3WR provide the write enable for the w byte, the x byte, the y byte, and the z byte, respectively.

3.2.2 Aram addressing (see pages 18d and 29d)

The three-bit firmware field RM provides control for an eight-to-one selection of aram addressing sources. Twelve 74AS151's perform the selection resulting in RMAD(00-11). The table below shows what address results from each selection:

select code	RMAD(00-03)	RMAD(04-07)	RMAD08-15)
0	zeros	GRBR(00-03)	OPRGA(0-3)
1	zeros	GRBR(00-03)	OPRGB(0-3)
2	zeros	GRBR(00-03)	OPRGC(0-3)
3	zeros	GRBR(00-03)	OPRGD(0-3)
4	CRFT(01-04)	CRDK(00-03)	CRDK(04-07)
5	s p	a r	e
6	BSBR(00-03)	BSAR(00-03)	BSAR(04-07)
7	1,ZBUS(19-21)	ZBUS(22-25)	ZBUS(26-29)

grbr is a four bit "general bank" register which permits the sixteen aram locations accessible by opa, b, c, and d to be any block of sixteen in the first 256 aram locations (see 3.11). bsar and bsbr are a four and an eight-bit register which together provide an independent aram addressing mechanism (see 3.11). crft together with crdk provide the "literal" address where crdk is part of the dbus control (see 3.3) and crft normally controls the temporary flags (see 3.7). The last selection has a rather specialized use. It allows 2048 locations of the aram to become a cache of control structures where a field of the zbus is used to access one of the 2048 locations and data in the accessed location can be used to determines hit or miss at the discretion of the microcoder (see 3.9.2.3).

3.3 THE D BUS (block diagram identifier III)

The dbus is a major node in the 32-bit custom processor. It is one of three 32-bit buses but it 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 transmitted to the zbus. The dbus is structured as four eight-bit buses, allowing up to four dbus sources to be combined (see section 4.3 for a complete list of the combinations and permutations).

3.3.1 dbus byte w - bits 00-07 (see lbd pages 17d, 19d and 20d)

Byte w may receive from six mutually exclusive sources which are listed below along with the signal, emanating from a pal, that enables each source:

source	enable signal
1. the sbus to dbus latch (LTCH00-07)	LTCHWE
2. the h register (HREG00-07)	HREGWE
3. the zbus (BUSZ00-07)	BZTOBD
4. a fill byte (SGNEXT)	FILLWE
5. the 8 bit "j" literal (CRDJ00-07)	CRDJWE
6. the 8 bit "k" literal (CRDK00-07)	CRDKWE

There is more discussion of source #1 in 3.4.

At the firmware level, the normal fill bit is CRDF00 which becomes an input to an eight-to-one multiplexer along with three sbus bits in order to provide three types of sbus sign extension as a function of certain dbus control bits (CRDJ02, CRDJ04, and CRDTRE). The output of the multiplexer is replicated 8, 16, 24 or 32 times providing the required fill or sign extension.

3.3.2 dbus byte x - bits 08-15 (see lbd pages 17d, 21d and 22d)

Byte x may receive from six mutually exclusive sources which are listed below along with the signal, emanating from a pal, that enables each source:

source	enable signal
1. the sbus to dbus latch (LTCH08-15)	LTCHXE
2. the h register (HREG08-15)	HREGXE
3. the zbus (BUSZ08-15)	BZTOBD
4. a fill byte (SGNEXT)	FULLXE
5. pctr/p history (PHST08-15) register	PCTREN
6. the 8 bit "k" literal	CRDKXE

By examination of the last two characters of the enable signal, one may determine if the source is partitionable or if it is a "broadside" source. In the six sources above, the latch, the h register, the fill, and the k literal are partioned sources whereas the zbus and the pctr/p history are not.

3.3.3 dbus byte y (bits 16-23) see lbd pages 17d, 23d and 24d

Byte y may receive from eleven mutually exclusive sources which are listed below along with the signal, emanating from a pal, that enables each source:

source	enable signal
1. the sbus to dbus latch (LTCH16-23)	LTCHYE
2. the h register (HREG16-23)	HREGYE
3. the zbus (BUSZ16-23)	BZTOBD
4. a fill nibble (SGNEXT) a fill nibble (SGNEXT)	FILLYE.03 FILLYE.47
5. the "j" 8 bit literal (CRDJ00-07)	CRDJYE
6. the "k" 4 bit literal (CRDK00-03) the "k" 4 bit literal (CRDK04-07)	CRDKYE.03 CRDKYE.47
7. the pctr/p history register (PHST16-23)	PCTREN
8. the hex decoder (HEXD00-07)	HEXDEN
9. the opa&b registers (OPRGA0-3,OPRGB0-3)	OPRGYE
10. the "procedure" mux (PTOD16-23)	PTOYEN
11. the aram bank registers (GRBR00-03) (BSHR00-03)	ASBREN

Sources #4 and #6 are partitioned at the nibble level in order to support the micro D:RAMAD-X8 (see 4.3).

Source #8 is one byte of the hex decoder, a mechanism which decodes, with a couple of 74S138's, the least significant four bits of the aram address (RMAD08-11) and emits a one in a field of fifteen zeros.

Source #9 allows the op register to be placed upon the dbus.

Source #10 for this byte deposits the output of a 2:1 mux onto the dbus. The two sources on the input of the mux are:

a. the most significant byte of the pbus (BUSP00-07)

b. seven fill bits (CRDF00) and the most significant bit of opc(OPRGC0) which, with the help of the correspon- input on byte z, yields opc/d shifted left one place for the micros D:OPCD-X2 and D:OPCD-X2'1.

Source #11 provides access to the independent aram addressing mechanism.

3.3.4 dbus byte z - bits 24-31 (see lbd pages 17d, 25d and 26d)

Byte z may receive from eleven mutually exclusive sources which are listed below along with the signal, emanating from a pal, that enables each source:

source	enable signal
1. the sbus to dbus latch (LTCH24-31)	LTCHZE
2. the h register (HREG24-31)	HREGZE
3. the zbus (BZUS24-31)	BZTOBD
4. a fill byte (SGNEXT)	FILLZE
5. the rmad times 8 mechanism	PCBREN
6. the 8 bit "k" literal (CRDK00-07)	CRDKZE
7. the pctr/p history register (PHST24-31)	PCTREN
8. the hex decoder (HEXD08-15)	HEXDEN
9. the opc&d registers (OPRGC0-3,OPRGD0-3)	OPRGZE
10. the "procedure" mux (PTOD24-31)	PTOYEN
11. an aram address register (BSAR00-07)	ASBREN

Source #5 implements the least significant eight bits of the micro D:RAMAD-X8 placing onto this byte the high order k literal bit CRDK00, RMAD(08-11), and the next three k literal bits (CRDK01-03).

Source #10 for this byte deposits the output of a 4:1 mux onto the dbus. The four sources on the input of the mux are:

a. the eight least-significant bits of the pbus (BUSP08-15)

b. the three least significant bits of opc (OPRGC1-3), opd (OPRGD0-3), and the fill bit (CRDF00). This input along with the corresponding input on byte y allows the dbus to receive opc/d doubled (CRDF00=0) or dibbled (CRDF00=1) for micros D:OPCD-X2 or D:OPCD-X2'1.

c. the most significant eight bits of the pbus (BUSP00-07)

d. registers opc/d shifted right three bits for the micro D:OP-BIT-AD

3.4 THE S BUS (block diagram identifier IV)

The sbus is a very special kind of 32-bit bus. Although it has a limited number of sources and few destinations, it is unique because, in the same step, two different 32-bit sources may be placed upon it. The justification for this construction is that data from the aram must be read, modified and written in the same step (the 1421's share input/output data pins). Thus the sbus is a split-cycle mechanism capable of having data from one source in the early phase of a step (e.g., the aram) and another source during the late phase (e.g., the zbus-shifter). Another unique property of the sbus is its ability to send either the early data or the late data (but not both) to the dbus. This is accomplished via transparent latches which, on a byte-partioned basis, either go blind at mid-cycle (thus capturing the early sbus data) or remain transparent throughout the cycle (thus delivering to the dbus the late sbus data). "Byte partioned" means that micros are provided which permit each dbus byte independent choice of the early sbus information, the late sbus information, or other sources as described in 3.5.

3.4.1 sbus bytes w, x, y, and z (see lbd pages 14d, 15d, 21d, 22d, 17m, 21m, and 33m)

Bytes x, y, and z may receive from eight mutually exclusive sources which are listed below along with the signal which enables each source. These thirty-two bits of the sbus appear on both the mother and daughter boards. Three of the eight sources are on the daughter board and five are on the mother board.

source	enable signal
1. the aram byte w (ARAM00-08)	RAMWCE
byte x (ARAM08-15)	RAMXCE
byte y (ARAM16-23)	RAMYCE
byte z (ARAM24-31)	RAMZCE
2. the shifter a output (SHFT08-31)	SFT1EN
3. the shifter b output (SHFT08-31)	SFT2EN
4. the stack top (KTOP06-15) and the accounting timer (ACTM00-15)	ACTN2S
5. the syndrome register (SYND08-31)	SYND2S
6. address register a (ADRA08-31)	ADRA2S
7. address register b (ADRB08-31)	ADRB2S
8. address register p (ADRP08-31)	ADRP2S

Regarding source #1, the enable circuitry has been discussed in 3.2 but one should note that when writing the aram, both the chip enable (RAMWCE) and the write enable (ARM0WR) must be on. This condition causes the 1421 rams' output circuitry to go to the high impedance state. This arrangement allows the sbus to receive data from any other source while the aram is writing.

Sources #2 and #3 are listed separately because of the configuration of shifter chips (25S10's) used. Sixteen chips, each with four output pins, are connected in a parallel structure such that, as a function of the high order bit of the shift distance (CRSD00), either one or the other group of eight chips is enabled to the sbus (see 3.8).

Source #4 is, at the full sbus level, a collection of two sources which total less than 33 bits. The upper 16 bits "read" the stack top (the firmware address which the next RETURN micro will utilize) and the lower 16 bits "read" the accounting timer (see 3.11.8).

Source #5, the syndrome register, is a mechanism which captures subsystem configuration information, the most recent error event or reason code. The path to the sbus merely allows the coder to view this reason code (for more about errors, see 3.15).

Sources #6, #7, and #8 allow reading adra, adrb, and adrp respectively. For more about these registers, see 3.11.

3.4.2 Sbus enables (see lbd pages 17d, 10m and 17m)

The sbus control field (CRSS00-02) allows one of six types of sbus cycles in any step. The table shown below summarizes the six cases:

CRSS00-02	early data from	late data from
0	adra/b/p	adra/b/p
1	adra/b/p	zbus
2	synd/actm.stack	synd/actm.stack
3	zbus	zbus
4	spare	
5	spare	
6	aram	aram
7	aram	zbus

The aram split cycle controls are discussed in 3.2. The general technique for the shifter and the address registers is the same; namely, there is an input on each enable circuit for the early half of the cycle and another for the late half.

The shifter enable circuits receive an input from a flop (CRSSE3) which predecodes the control store word looking for the only code which requires the zbus to the sbus during the early phase (i.e., code 3). The late input is decoded in a pal.

The early gate on the address register enable circuits (ADRA2S, ADRB2S, and ADRP2S) is timed with MCLOCK whereas the late inputs are essentially untimed. Notice that the choice among adra/b/p is the responsibility of the BS field and not of the SS field. The 74S139 which decodes two bits of BS for this purpose is enabled for codes 0, 1, 4, and 5, but 4 and 5 are not used and 0 and 1 are the codes which deposit adra/b/p to the sbus.

3.5 THE Z BUS (block diagram identifier V)

The zbus is a place 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, 29d, 32d, 33d, and 34d)

The zbus is thirty-two bits wide and receives from six thirty-two-bit-wide sources. Along with the ralu and the input data registers, the zbus may receive from the v register, the dbus, and from an external connection (i.e., the firmware development facility). The six sources are listed below along with their respective enable signals:

source	enable signal
1. inra (INRA00-31)	CRIA2Z
2. inrb (INRB00-31)	CRIB2Z
3. v register (VREG00-31)	VREGEN
4. ralu (ALUY00-31)	ALUTOZ
5. the dbus (BUSD00-31)	DBTPZB
6. the fdf (BUSZ00-31.EX)	CRTB2Z

3.5.2 zbus enables (see lbd pages 12d, and 32d)

The ZB field controls the zbus sources as shown below:

crzb(00-02)	source	enable signal
0	ralu	ALUTOZ
1	fdf	CRTB2Z
2	inra	CRIA2Z
3	inrb	CRIB2Z
4	dbus	DBTOZB
5	vreg	VTRHEN
6	inra	CRIA2Z
7	inrb	CRIB2Z

All six enables emanate from a pal which decodes crzb. Codes 0 through 5 produce results previously discussed. When inra or inrb is placed upon the zbus as a result of code two or three a parity check is performed on all four bytes. When inra or inrb is placed upon the zbus as a result of code six or seven, a parity check is performed on only the two most significant bytes.

3.6 THE LOCAL BUS (see block diagram identifier VI)

The local-bus hardware can be divided into four sections:

An addressing mechanism

A data storage mechanism

A procedure storage mechanism and

Local-bus control circuitry

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

3.6.1 Local bus addressing (see lbd pages 18m, 19m, and 20m)

Addresses from the processor to the memory subsystem(s) may emanate from one of three sources: adra, adrb, or adrp. A three-to-one mux is formed using 74S241's selected by ADRAEN, ADRBEN, and ADRPEN. Once the selection has been made, the address is latched via another group of 74S241's connected as a batlatch. (A batlatch is used to eliminate propagation delay. Batlatches require delicate timing.) Thirty nanoseconds into a firmware step which has made a local bus request, the flop ABATEN comes on, enabling the batlatch which connects its input to its output. About thirty nanoseconds later, ADRS0F comes on, disabling the multiplexer. The batlatch thereby swallows its tail.

3.6.2 Local bus data storage (see lbd pages 32d, 33d, 34d and 34m)

Data from the local bus is captured off the local bus with a transparent latch (i.e., four 74S373's). The behavior of the signal LBLOOK is that it puts the latch in transparent mode sometime after the request to memory has been made; as soon as the memory signals that valid data is present on the interface (DCNNUS), the latch is closed. The data may now be safely transferred to either inra, or inrb dependent upon which was requested.

The signals DALOOK and DBLOOK perform the transfer of data from the local-bus latch to inra or inrb. During the step in which the request was initiated, both these 'looks' clear. At the beginning of the step immediately following the request, the appropriate look activates causing the latched data to flood into inra or inrb. The look signal remains on until the next local-bus request occurs.

3.6.3 Local-bus procedure storage (see lbd pages 15m, 16m, 28m, 30m, 34m

The mother board captures local-bus data in the same manner as the daughter board, using a duplicate set of 74S373 transparent latches. From the local bus latch, the information moves into the procedure "prefetch" buffer. This buffer receives procedure 32 bits at a time and dispenses procedure either eight or sixteen bits at a time. In order to allow for enough look-ahead, the procedure buffer can remember eight bytes (64 bits). Thus the destination for each 32-bit delivery from the local-bus latch alternates; the first 32 bits are placed in the procedure buffer at bytes a, b, c, and d; the second 32 bits are placed in the procedure buffer at bytes e, f, g, and h; the third 32 bits are placed in a, b, c, and d again etc..The alternation results from the look signals (PALOOK and PELOOK) taking turns, because pareqt and pereqt take turns because FRELOD complements each time a procedure request is made.

In order to dispense one byte at a time, eight 74S373 chips form this 64-bit storage register with 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 latch 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.

In order to dispense two bytes of procedure at a time, eight more 74S373's form a completely duplicate 64 bit storage register also with their outputs connected together to form an eight-bit bus, but this bus is called BUSP08-15 and the network differs from the one previously described only in enabling. Above, when PTAKE4/2/1=1, BUSP00-07 received from INRPB0-7; in that instance, the secondary buffer enables INRPC0-7 to BUSP08-15. Thus whatever byte the take counter sends to BUSP00-07 from the primary procedure buffer, the next byte in line is sent to BUSP08-15 from the secondary buffer.

Decoding the take counter is accomplished with a 74S138. The take counter itself is comprised of three 74S112 j/k flops which, with the help of a pal, increment by zero, one, or two each firmware step, recording the removal of procedure bytes.

3.6.4 Local-bus control circuitry (see lbd pages 9m, 11m, 28m 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 BS field emanates from the control-store array at bits 120-127 (CSBSC0-7) and is captured in eight control register flops (CRBUS0-7). Processor timing is such that CSBSC0-7 are valid 20ns 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 a memory subsystem be intiated?
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 a memory subsystem is in process, the signal REQNOW 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, CSBSC0-7 alerts the request logic (REQTEN, REQUST, and the flop USREQT) 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, REQNOW and CSBSC0-5 have a significant impact upon the clock logic (see 3.12). USREQT gets a chance at the beginning of every step and is, of course, structured to assume that whatever stall was required, has occurred and has been released.

A summary of the control sequence is as follows:

1. USREQT comes on at the beginning of any step which initiates a memory-subsystem transaction.
2. An array of PALs determines whether the request is to be sent to Memory Subsystem #1 (or Memory Subsystem #0)
   a. M1REQT is clocked by USREQT and activates if the tri-state request collector M1RQTD.00 is true
   b. M1RQTD.01-03 are active in 32-bit address systems (ADDR32=1) if M1PRZT (M1 present) is true
   c. M1RQTD.-04,05 are active in 24-bit address systems (ADDR32=0) if M1PRZT (M1 present) is true
   d. M1RQTD.04 is active to disengage the network if M1 is not present
   e. M0REQT is also clocked by USREQT and activates for all cases where M1REQT does not:

      Memory Subsystem #1 not present

      A lock/unlock memory reference

      A memory reference not in M1 space

      A non-memory reference

2. immediately, REQNOW comes on.

3. the memory subsystem "grants" the processor. (If the memory subsystem was not busy, the grant arrives in about 50ns; but if it was busy, who knows.)

4. about 50ns after the grant is received, USREQT goes off.

5. now REQNOW is in charge and will remain on until the transaction completes:

   a. for memory reads, when data arrives from memory (DCNNUS signals the arrival and OUTREN is off)

   b. for i/o reads, when the data arrives from the MEGABUS (second-half bus cycle) via the memory subsystem (DCNNUS signals the arrival and OUTREN is off)

   c. for i/o reads where no data arrives, when grant (RQGTUS) goes off.

   d. for all writes, when grant goes off.

Once the request is initiated, the firmware sequence is permitted to go on its merry way, ignoring the local-bus interface until such time as a resynchronization point is encountered (e.g., a data stall, a new local bus transaction initiation). During this interval, the transaction is remembered in three storage elements:

1. the thirty-two-bit address is in the batlatch of 3.6.1

2. the data to be written (if any) is in outr

3. the control information is in the "fred" register

Item #3 above is comprised of two registered pals. One pal monitors eight inputs and creates six local-bus control signals. Notice that the pal is allowed to change its output values only on the leading edge of each new memory subsystem request, snapshotting its other seven inputs to decide whether this transaction is:

1. a write? (FWRITE)

2. a doubleword read/write? (FRDBLW)

3. a lock/unlock? (FRLOCK)

4. a memory reference? (FRMREF, two copies)

5. to use adra, adrb or abrp? (FRBUS2/3)

6. to capture the ACK/NAK indicator (ACKREN)

A second pal (16R6B) monitors four inputs and generates three outputs deciding whether this transaction is:

1. to send a response notification? (LBSHBC)

2. to write only eight bits of the first sixteen (LBWCT1)
   Note: other system elements refer to this signal as xxBYTE

3. to write all sixteen bits of the second sixteen (LBWCT2)
   Note: other system elements refer to this signal as xxDBPL

A third pal (16R6b) monitors ten inputs and creates four local-bus related outputs which determine whether this step is:

1. one which initiates a local bus request (CRREQT)
2. one which performs a local bus "wrap" test (CRWRAP)
3. one which will consume (one or two bytes of) procedure
4. one which will change the procedure steerage mechanism (FRELOD). If FRELOD=0, arriving prefetched procedure is captured by procedure buffers A, B, C, and D. If FRELOD=1, arriving prefetched procedure is captured by procedure buffers E, F, G, and H.

3.7 TEMPORARY, PERMANENT AND CONTROL FLAGS (block diagram identifier VII)

There are twenty-four flags organized as three groups of eight. Sixteen of the twenty-four are controlled by one firmware field while the other eight have their own.

3.7.1 temporary flags (see lbd pages 9m, 24m and 29m)

The temporary flags are controlled by the FT field (CRFT00-04). Seventeen of the thirty-two possible codes are devoted to the temporary flags; one to clear them all, 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 pal output (CLRTFL) determines when a broadside clear occurs, and another (FLGTEN) determines when a one-bit change occurs. The polarity of the change is determined by the data input CRFT00. Which flop is to change is determined by the select inputs CRFT02-04. The eight outputs are synchronized by a 74S374 and sent to eight test-condition inputs (see 3.14). FLGT6E, temporary flag 6 before synchronization, is the data bit written into the stop code ram (see 3.9.2.1).

3.7.2 permanent flags (see lbd pages 13m, 28m and 29m)

The permanent flags are controlled by the BS field (CRBUS0-7). Sixteen of the possible 256 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. The flops themselves are a 74S259. A broadside clear occurs at master clear. A pal output (FLGPEN) determines when a one bit change occurs. The polarity of the change is determined by CRBUS3. Which flop is to change is determined by the select inputs CRBUS5-7. The eight outputs are synchronized by a 74S374 and sent to eight test conditions (see 3.14). Permanent flag #7 (FLAGP7) is one of the "break" stimuli.

3.7.3 control flags (see lbd pages 24m, 27m, 29m and 33m)

The control flags are controlled by the BS field (CRBUS0-7). Sixteen of the possible 256 codes are devote to the control flags; 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. A broadside clear occurs at master clear. A pal output (FLGCEN) determines when a one-bit change occurs. The polarity of the change is determined by CRBUS3. Which flop is to change is determined by the select inputs CRBUS5-7. The eight outputs are sychronized by a 74S374 and sent to eight test conditions (see 3.14). Some of the control flops have side-effects:

FLAGC0	off line
FLAGC1	qlt failed (if flagc0 is off)
FLAGC2	generate even parity to outr (if flagc0 is off)
FLAGC3	enable the accounting timer
FLAGC4	undedicated
FLAGC5	inhibit disaster
FLAGC6	undedicated
FLAGC7	stack error detection enable

3.8 NIBBLE SHIFTER (see block diagram identifier VIII)

The nibble shifter is the only connection between the zbus and the sbus and is therefore in a critical path. The shifter is really an eight nibble rotator; there is no distinction for instance between, shifting three nibbles left and shifting five nibbles right.

3.8.1 shifter data flow (see lbd pages 11d, 14d and 15d)

The shifter is implemented with sixteen 25S10's connected as two parallel groups of eight. The first group is enabled when the shift distance prescribed by the SD field (CRSD00-02) is either zero, one, two or three, and the second group is enabled when the shift distance is either four, five, six, or seven.

The 25S10 whose output names are SHFT00.SA, SHFT04.SA, SHFT08.SA and SHFT12.SA reveals first that this chip is enabled for shift distances of 0, 1, 2, or 3 and that for a distance of zero, the 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 eight shifter chips operates like the first, but are enabled for shift distances of four or more and thus their data inputs are wired bias by sixteen bit positions.

3.8.2 Shifter control (see lbd pages 10m, 11d, 14d, 15d and 17d)

Since the shifter deposits its result onto a split-cycle bus, the enable requires early/late timing. For SS (CRSS00-2) code three only, the shifter is enabled to the sbus during the early phase; a pal generating signals S1LATE and S2LATE determines what circumstances allow the shifter to the sbus during the late phase.

3.9 ARITHMETIC AND MISCELLANEOUS INDICATORS (block diagram identifier IX)

There are six arithmetic indicators and eight 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 5d, 12d, 13d, 27d and 28d)

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). In addition to storage for the indicators, four 4:1 muxes are employed. Three of the muxes are halves of 74S153's and one is a 74S64.

The overflow mux (OVFMUX) selects one of four 2901 overflow signals as a function of the ID code (CRID00,01) as follows:

CRID00,01	overflow source
00	carry into alu bit 24 not equal to carry out of alu bit 24
01	carry into alu bit 16 not equal to carry out of alu bit 16
10	carry into alu bit 08 not equal to carry out of alu bit 08
11	carry into alu bit 00 not equal to carry out of alu bit 00

The carry mux (CRYMUX) samples one of four alu carries as a function of the ID field as follows:

CRID00,01	carry source
00	carry out of alu bit 24
01	carry out of alu bit 16
10	carry out of alu bit 08
11	carry out of alu bit 00

The sign mux (SGNMUX) samples one of four alu "signs" as a function of the ID field as follows:

CRID00,01	sign source
00	alu output bit 24
01	alu output bit 16
10	alu output bit 08
11	alu output bit 00

The zero mux (SZRMUX) samples four groupings of bits as a function of the ID field as follows:

CRID00,01	zero source
00	if alu output bits 24-31 are all zeros
01	if alu output bits 16-31 are all zeros
10	if alu output bits 08-31 are all zeros
11	if alu output bits 00-31 are all zeros

These four mux outputs are stored into the overflow indicator (OVFIND), the carry indicator (CRYIND), the sign indicator (SGNIND), and the zero indicator (SZRIND) at the behest of a pal output (AINDEN). Also captured are the double zero indicator (DZRIND) and the odd indicator (ODDIND). A code emanating from the FT field causes the arithmetic indicators to "clear" such that four indicators are off and the zero and double-zero indicators are on.

3.9.2 miscellaneous indicators (see lbd page 27d)

The eight miscellaneous indicators can be divided into one group of four and four groups of one. A pal output (BINDEN) determines when the miscellaneous indicators sample their inputs.

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 FLGT6E under command of a pal output (WRSCRM) modulated with appropriate timing. Its data output is captured along with the other seven miscellaneous indicators when the pal output BINDEN occurs.

3.9.2.2 DIFBUF indicator

This mechanism compares eight bits of the dbus with eight bits of the zbus utilizing a 74F521.

3.9.2.3 HASHIT indicator

This mechanism compares thirteen bits of the zbus with thirteen bits of the sbus utilizing two 25LS2521's.

3.9.2.4 ILLADD indicator

This mechanism uses a 16L8A pal to decide whether the rightmost byte of an operand is in the same 512-byte buffer as a specified base address. The pal assumes that the base address and the displacement locating the leftmost byte of an operand are being added in the alu, that one of these values is on the dbus, and that the type register contains the operand-length information (1, 2, 4 or 6 byte operand). The pal would prefer a carry signal out of alu bit 23 but one is not available so the pal derives it from the input carry to alu bit 23 (AUCO24), the dbus bit 23 (BUSD23) and the sum bit 23 (BUSZ23).

3.9.2.5 leading zero detector

This mechanism, utilizing a 16L8A pal, determines the number of leading zeros present in the value from the alu. The pal receives the zero detector from each of the eight 2901's. The outputs from the pal (AUPZ08, AUPZ04, AUPZ02, and AUPZ01) are captured in the miscellaneous indicators. The four indicators (PZ8IND, PZ4IND, PZ2IND, and PZ1IND) are coded such that: 0000 means no leading-zero nibbles were detected, 0101 means that five leading-zero nibbles were detected and 1000 means the entire alu output was zero.

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 four 4-bit wide registers opa, opb, opc and opd provide said storage. These four registers can also be loaded from the sbus, can be incremented or decremented, and have the ability to address the aram. The seven bit field OP controls the entire op area.

3.10.1 the sbus multiplexer (see lbd pages 10d and 31d)

When the op register(s) are to be loaded from the sbus, the sbus multiplexer (SMUX00-15) is required to narrow the 32-bit sbus to a width of 16 bits, matching the width of the op muxtiplexer. CROP02 decides which half of the sbus is chosen.

3.10.2 the op multiplexer (see lbd pages 10d and 31d)

The op multiplexer (OPMX00-15) is comprised of four 16L8A pals. Each pal has four data outputs, nine data and four control inputs. At the first level, the choice is between the pbus data inputs and the sbus data inputs. At the next level, a choice of which nibble of the selected bus is to be directed to the output (sort of a rotate). And at the last level, a choice as to whether the four bit literal inputs (CRFT01-04) should be presented to the outputs. All this is controlled by the inputs CROP00-03.

3.10.3 The op registers (see lbd page 31d)

There is a 74AS169 for each of opa (OPRGA0-3), opb (OPRGB0-3), opc (OPRGC0-3), and opd (OPRGD0-3). Loading, incrementing or decrementing is controlled by a pal which emits, for each register, one load signal and one count signal. For opa, the load signal is OPALOD and the count signal is OPAPAT . When the count signal is on, the appropriate op register increments if crop02 is on and decrements otherwise. OPATOP, OPBTOP, OPCTOP, and OPDTOP are provided to detect when the counter in question is "wrapping" (in the firmware vernacular) i.e., incrementing from F to zero or decrementing from zero to F.

3.10.4 the type register (see lbd page 31d)

The type register stores two bits in a 74S169. The input to the type register is the same as the two most significant inputs of opa. The unused two stages of the 74S169 are wired so that when the type is 11 (three), the "carry" output of the chip (TYPIS3) will be on. A signal from the FT field (CLRTFL) clears the type register and a load signal (TYPELD), also from an FT pal, performs the load honors.

3.11 LOADING VARIOUS REGISTERS (block diagram identifier XI)

Many firmware fields have no other purpose but to control the loading of various registers. These fields are H, LA, LO, and LV. Other fields have a secondary justification for existence by providing the load controls to certain registers in need. These fields are BS and FT.

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

The H register is loaded at the behest of a dedicated control store bit (CSH056) with its attendant control-register flop (CRH000) pulse-formed in HREGLD.

3.11.2 loading the output register (see lbd pages 11d and 35d)

The output register (OUTR00-31) is loaded at the behest of a dedicated control store bit (CSLOA0) with its attendant control-register flop (CRLO00) pulse-formed in outrck. Note that 36 bits are captured with this load stimulus; namely, the 32 bits of the dbus and four byte-parity bits in flops OUTR0P, OUTR1P, OUTR2P, and OUTR3P. These four parity bits are generated by four 82s62 parity generator/checker—chips called MYDP00, MYDP08, MYDP16, and MYDP24. See section 3.15 for more about checking.

3.11.3 loading the v register (see lbd pages 11d and 34d)

The v register (VREG00-31) is loaded at the behest of a dedicated control-store bit (CSLV96) with its attendant control register flop (CRLV00) pulse-formed in vregld.

3.11.4 changing adra, adrb, and adrp/pctr (see lbd pages 10m, 26m 17d, 21d, 23d and 26d)

Four control-store bits (CSLAB6-9) and four cr flops (CRLA00-03) are provided to control the changing of the three address registers. Each address register is thirty-two bits long and is comprised of four 74AS869 chips. Address registers adra (ADRA00-31) and adrb (ADRB00-31) may be loaded from the sbus, or may be incremented by four, or may be decremented by four. The address register adrp (ADRP00-31) may be fully loaded from the sbus or may have only its nine least significant bits copied from the sbus or may have its twenty-three-most-significant bits copied from the sbus. Note that adrp is implicitly incremented by four whenever four procedure bytes are requested from the memory subsystem.

The pal which controls all this generates two mode bits (ADRAM0,1) and a carry-in (ADRACI) for adra, and two mode bits (ADRBM0,1) and a carry-in (ADRBCI) for adrb. When ADRA/BM0=0 and ADRA/BM1=1, the register decrements; when m0=1 and m1=0, the register increments; and when m0=1 and m1=1, the register loads from the sbus.

For adrp, the pal generates two load signals (PULOAD, PLLOAD) for copying twenty-three or nine sbus bits into adrp. Because of the partioned load and the necessity to increment by four, a thirty-third flop is required to store ADRP30.

The register PCTR(08-30) mimics adrp for loading actions. Its bit justification in the three 74AS869 chips permit incrementing by one or two when procedure bytes are removed from the prefetch buffer. When pctr is delivered to the dbus, the least-significant bit of the take counter (PTAKE1) becomes the units position of pctr.

3.11.5 changing grbr (see lbd page 30d)

The four-bit register GRBR(00-03) is used in combination with opa,b,c, and d to address the aram. The OP field controls grbr. A 16R8A pal receives the seven control-store bits of the OP field and generates control signals for the 74AS169 chip which houses grbr. The pal output GRBRLD causes GRBR(00-03) to load from the four most significant bits of the opmux (OPMX00-03). The pal output GRBRPT causes GRBR(00-03) to increment by one or decrement by one as a function of CROP02. If CROP02 is on, grbr increments; if CROP02 is off, grbr decrements. The firmware name for grbr is rbr.

3.11.6 changing bsbr (see lbd page 30d)

The four-bit register BSBR(00-03) is used in combination with BSAR(00-07) to address the aram. The OP field controls bsbr. A 16R8A pal receives the seven control-store bits of the OP field and generates control signals for the 74AS169 chip which houses bsbr. The pal output BSBRLD causes BSBR(00-03) to load from opmux bits 4-7 (OPMX04-07). The pal output BSBRPT causes BSBR(00-03) to increment by one or decrement by one as a function of CROP02. If CROP02 is on, bsbr increments; if CROP02 is off, bsbr decrements. The firmware name for bsbr is rarh.

3.11.7 changing bsar (see lbd page 30d)

The eight-bit register BSAR(00-07) is used in combination with bsbr to address the aram. The OP field controls bsar. A 16R8A pal receives the seven control-store bits of the OP field and generates control signals for the two 74AS169 chips which house bsar. The pal output BSARLD causes BSAR(00-07) to load from the eight-least significant bits of the opmux (OPMX0815). The pal output BSARPT causes BSAR(00-07) to increment by one or decrement by one as a function of CROP02. If CROP02 is on, bsar increments; if CROP02 is off, bsar decrements. The firmware name for bsar is rarl.

3.11.8 loading the accounting timer (see lbd pages 24m and 33m)

The accounting timer is a mechanism which provides real time information for job accounting and other purposes. The accounting timer is constructed from two 74AS869 counter chips wired to either load or increment. The timer (ACTM00-15) has a 100 microsecond period (i.e., it is incremented at a 10khz rate when enabled by FLAGC3). It is loaded by an FT-field micro decoded by a 16L8A pal (ACTMLD) which invokes an implicit clock stall (see3.13) whose purpose is to insure that an accounting timer clock (MHZ001) occurs while the data to be loaded from the sbus is valid at the inputs of the 74AS869 chips. Note that when the accounting timer increments from 65,535, an accounting timer interrupt (ACTINT) is stored. This interrupt (synchronized through ACTITF) asserts the break stimulus and thereby causes a "derail" at the next sample of break by the RNI macro.

3.12 THE FOUR SPEED CLOCK (block diagram identifier XII)

The custom processor resides in an asynchronous world. The custom processor clock is an asynchonous 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 3m and 4m)

The basic clock has three delay lines, three delay-line drivers, one 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 about the same time, MCX010 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 70ns. Having established that a "negative" seventy-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). In

the simplest case (i.e., the fastest clock speed), MCX000 causes the delay line driver MCY000 to go high. The delay line MCY010-030 is now propagating a "positive" seventy-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 4)

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 below are the four speeds and their definition:

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

Control-store bits 64 and 65 (csck64, 65) are sampled by a 16R6A registered pal which generates 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 outputs of the pal. For a very fast speed, no enables from the pal are active and the operation is as described above (i.e., the vf tap is always enabled). For a half-fast step, CKHFEN from the pal 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 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 of the pal (CKHFEN, CKHLEN and CKVLEN) are active, and the high time of THEEND stretches even further creating the longest firmware step available.

The 16R6A pal from which the clock speed enable signals emanate provides another service. It monitors six control store bits involved in return stack functions (see 3.13) and detects certain firmware sequences which require a non-minimum clock speed setting.

3.12.3 clock stalls (see lbd page 3m)

Four 74S64's provide the inputs for stalling the clock from "external" stimuli. The basic clock operation uses two of the sixteen inputs; that is, the MCKPWA 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.

stall gate	explanation
CSBSC0.CSBSC2.REQNOW	stall WHEN the next firmware step is to unconditionally initiate a local-bus request IF previously initiated local-bus activity is still in progress.
STLADU.REQNOW	stall WHEN an explicitly coded stall micro requires that the completion of this step be postponed until inra receives local-bus data from a previous request IF the data has not yet arrived.
CSBSC0.BLOTHR.REQNOW	stall WHEN the next firmware step is to conditionally initiate a local-bus request and the condition is satisfied IF previously initiated local-bus activity is still in progress.
DSTFF1.WAITUP	stall WHEN a hardware error has been detected until such time as the reason code (syndrome) has been captured in HWIA(00-13) and the next-address generator has generated and accessed location zero.
BSYSTL.REQNOW	stall WHEN an explicitly coded stall micro requires that the completion of this step be postponed until all local-bus activity has subsided IF local-bus activity is still in progress.
TBSTOP	stall if the firmware development facility says so.
CSBSC1.CSBSC4.EMPTER	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.
PLLOAD.REQNOW	stall WHEN an explicitly coded load of adrp-lower occurs IF previously initiated local-bus activity is still in progress.
LOADED	stall WHEN an explicit load of the accounting timer is required UNTIL the accounting timer clock (mhz001) has had an opportunity to sample the sbus inputs.
CSBSC0.CSBSC5.REQNOW	stall WHEN the next firmware step will initiate a prefetch IF previously initiated local-bus acivity is still in progress.
STLDBU.REQNOW	stall WHEN an explicitly coded micro re-

quires that the completion of this step be postponed until the data previously requested for inrb arrives IF the data has not yet arrived.

PAUSED.ACT2S — stall WHEN an explicit read of the accounting timer is in process IF the accounting timer is about to increment.

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 seventeen addresses. A firmware address is written into the stack when any of the three push micros is executed. An address is read from the stack when any of the return micros 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 is comprised of:

1. a 16-location return stack
2. a 4-bit return-stack address register
3. a return-stack local register
4. a "relative push" local register
5. an "absolute push" local register
6. a hardware-interrupt address register
7. a return address bus
8. an overflow/underflow error detector

3.13.1 the 16 location return stack (see lbd page 25m)

The sixteen-location return stack is comprised of four 74S189 chips providing a 16-by-16 array. Since the firmware address space is 16k, only fourteen data bits are used. The data out (RTRN00-13) is connected to the return-memory local register (KRAM00-13). The memory is addressed by a four bit address register (FWSTK0-3).

3.13.2 the 4-bit return-stack address register (see lbd page 24m)

The return-stack address register consists of one 74S169 counter chip. The counter may be initialized to a known state via FSTKLD, which is used in testing and other situations. In normal operation, a push causes the counter to increment before a new entry is written into the stack, and a pop (RETURN) causes the address register to be decremented after the entry has been read from the stack. First, notice that the counter chip "clocks" at midcycle(MCX060); then, notice that the counter chip increments if CDPUSH afrom a pal); and last, that it counts if PSNPOP (also from a pal). The 16L8A from which these signals emanate has inputs which permit it to:

1. CDPUSH: in any step in which a push micro is coded unless the previous step performed a successful return.
2. POPING: in any step which performs a successful return.
3. PSNPOP: 1 or 2 above.

3.13.3 the return-stack local register (see lbd pages 24m, 25m)

In a successful return, the return address bus is sent to the next-address generation circuits. The next return candidate is then read from the return stack into the return-stack local register so as to be available for the a subsequent return. A successful return generates POPING which is pulse shaped into POPCLK for capturing, in KRAM(00-13), the data output of the ram chips. The output of the return-stack local register, like the other three registers which capture return addresses, is connected to the return address bus (FSTK00-13).

3.13.4 the relative-push local register (see lbd page 25m)

One of the push micros allows the address saved to be relative to that specified in the next-address field (CRNA01-13). A relative push creates a firmware address to push by combining the eight most significant bits of crna with the five bits of the ft field and always pushing an if bank address. This combination of fourteen bits is captured in the relative push local register (NAFT00-13). Capturing takes place whenever any push is performed since each push source has its own local register. The selection as to which is used is a function of the local register enables. The output of the relative push local register, like the other three registers which capture return addresses, is connected to the return address bus (FSTK00-13).

3.13.5 The absolute push local register (see lbd page 29d)

Another push micro allows fourteen bits of the dbus to be saved in the return stack. At the end of any push step, BUSD(02-15) are captured in the absolute push local register. This register, like the other three registers which capture return addresses, is connected to the return address bus (FSTK00-13).

3.13.6 The hardware-interrupt address register (see lbd page 25)

The hardware-interrupt address register (HWIA00-13) captures the address to which the firmware sequence would have proceeded had an error (DSASTR) not been detected. This register, like the other three registers which capture return addresses, is connected to the return address bus (FSTK00-13).

3.13.7 the return-address bus (see lbd pages 24m and 25m)

The return-address bus connects together all four sources of return address. Constructed as a tri-state mechanism, the sources are mutually exclusive. The bus operates under the following rules to supply a candidate return address to the next-address generator (see 3.14):

1. If the most recent stack operation was a successful return, then the candidate for the next return is the return-stack local register (KRAM00-13). POPPED is generated in the step immediately following the successful return. Pulse-shaped via POOPED, it clears a 74S175 chip. This 74S175 creates an enable for each of the four return address bus (FSTK00-13) sources. Clearing this chip enables the kram source and disables the three other sources.

2. If the most recent stack operation was not a successful return, then one of the "push" sources is enabled to the return address bus. The register enabled is that which received the most recent push. The 74S175 is clocked (FSTKWE) by any push. When the clock occurs, one d input will be on: that corresponding to the type of push performed.

When any push occurs, a new candidate for the next return is "stacked". The previous candidate (former stack top) is written into the return stack memory via FSTKWE formed from CDPUSH with a bit of pulse shaping.

3.13.8 the stack overflow/underflow detector (see lbd page 24m)

The stack overflow/underflow detector is a pal which monitors the address register (FWSTK0-3), the nature of the stack transaction (CDPUSH, POPING) and an error detector enable. It generates two signals:

1. a stack error was detected (STKOUT) which is connected to the syndrome register and to the DSASTR circuitry for causing an unprogrammed branch to firmware location zero (if FLAGC5=0)

2. a stack underflow indicator which is connected to the syndrome register for distinguishing overflow from underflow.

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 (29 control-store bits) to the task of deciding what control-store location shall next be executed. The control-store array is physically implemented as two 8192 location memories; the lower addresses access the else bank and the higher addresses access the if bank. Figure 3-1 diagrams the if and else next address generators.

Of the five fields involved in this activity, the BR field code 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 72 "conditions" shall 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, 29m and 28d)

The TC field (CRTC00-06) can specify one of seventy-two conditions to be tested in order to decide which bank shall be selected. Nine 74S151's and three 74AS151's accept the seventy-two test conditions, the seven bits of the TC field and one bit of the next address field (CRNA00) reducing all this into one signal called TCTRUE. The high order next address bit specifies the test polarity which is why the firmware dictionary has twice as many (144) tests. The signal TCTRUE is fed to four 74S140 drivers for distribution purposes. The drivers are disabled if the firmware development facility is providing the control store output instead of the proms (TBRMEN).

3.14.2 else bank next-address generation (see lbd pages 5m, 6m, 8m, 12m, 30m and 32m)

The else bank may receive an address onto its tri-state address bus from one of three sources which are listed below along with the BR code(s) that enable(s) each source:

else bank source	BR code
1. the next address field (CRNA01-13)	0,1,2,3,4,5,6,7,8,C,D
2. the return stack (FSTK01-13)	9
3. the firmware development facility (NAEB01-13.EX)	all

Note that the determination of which bank shall be selected is the responsibility of the TC field.

Source #1 is the register which stores the thirteen control store bits CSNA67-79 connected through tri-state drivers to the else bank address bus (NAEB01-13). The drivers are enabled by a 16R6A pal output (NA2ELS). The most significant bit of na (CRNA00) determines the polarity of the test condition. This bit is referred to as the BI field.

Source #2 is the return-stack address bus (FSTK01-13) connected through tri-state drivers to the else bank address bus. The drivers are enabled by a 16R6A pal output (RTNELS).

Source #3 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 sources to the else bank address bus by disabling the pal and connecting to the pal output a 74S241 driver enabled by TBNASB whose inputs are biased to inactivate other sources of else bank addresses.

3.14.3 if-bank next-address generation (see lbd pages 12m and 32m)

The if bank may receive an address upon its address bus (NAIB01-13) from any one of six sources which are listed below along with the BR code(s) that enable(s) each source:

if bank source address	BR code(s)
1. the next-address field (CRNA01-13)	0,8
2. the return stack (FSTK01-13)	1,9
3. the firmware development facility (NAIB01-13.EX)	all
4. the 16-way masked splatter (SPLATA-D)	2,3,4,5,6,7
5. the 256-way unmasked splatter (PROC00-07)	C,D
6. the 16-way unmasked "break" splatter (BREK00-07)	C

The if-bank address generator is divided into three partitions: that circuitry which develops bits 1 through 5, that circuitry which develops bits 6 through 9, and that circuitry which develops bits 10 through 13.

3.14.3.1 if-bank address bits 01-05 (see lbd pages 12m and 32m)

The five most significant bits of the address bus (NAIB01-05) receive their inputs from one of three sources as shown below with the BR code(s) that enable(s) each source:

source for if-bank address bits 01-05	BR code(s)
1. the next-address field (CRNA01-05) via one 74S64 chip (NAIB01) and one 74S240 section (NAIB02-05.11) enabled by NA2IFH emanating from a pal.	0,2,3,4,5,6,7,8,C,D
2. the return stack (FSTK01-05) via the 74S64 chip (NAIB01) and portions of 74S240 chips (NAIB02-05.RI) enabled by RTN2IF emanating from a pal.	1,9

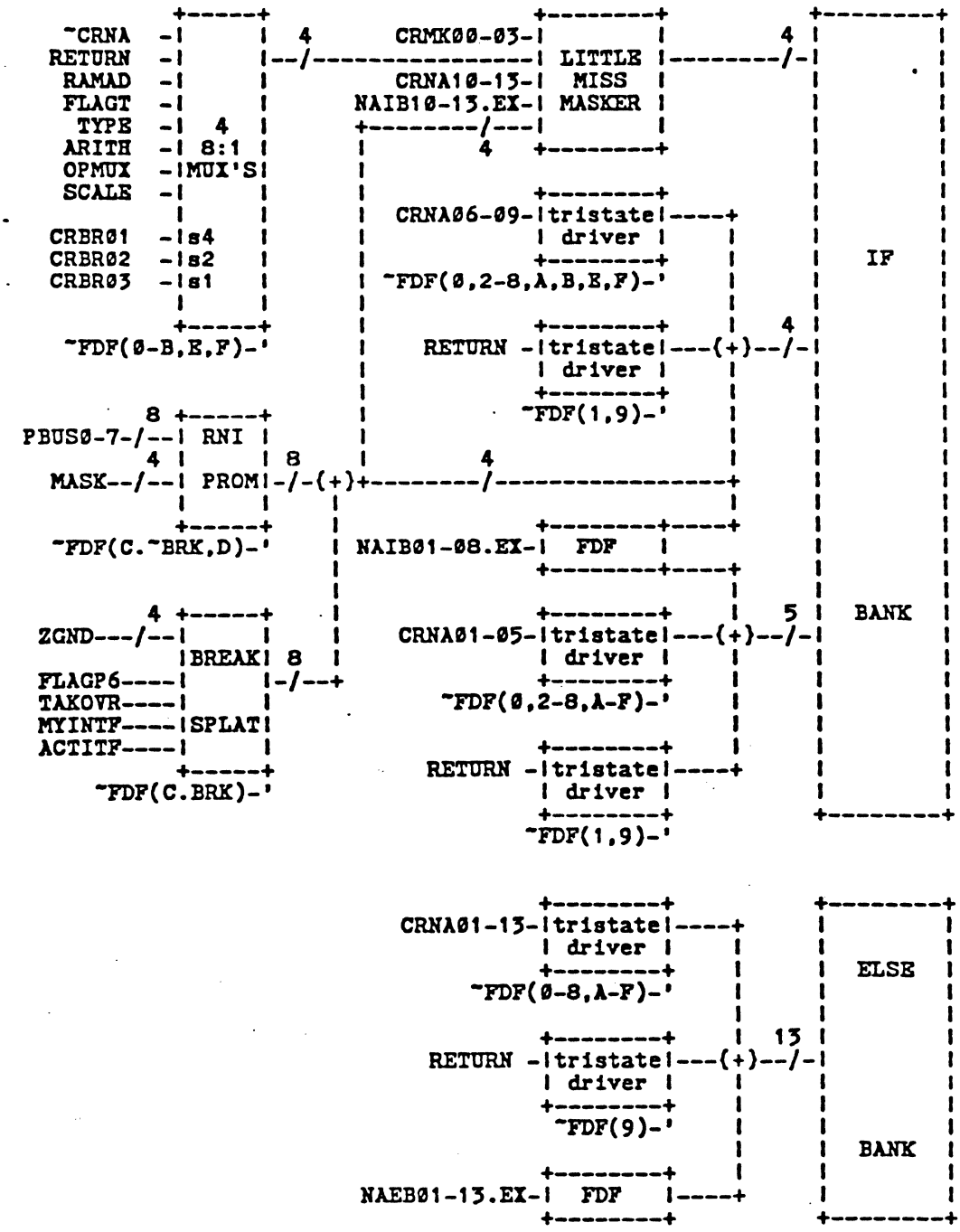


FIGURE 3-1

IF AND ELSE BANK NEXT ADDRESS GENERATORS

3. the firmware development facility (NAIB01-05.ex) via TBNASB. Note that the drivers for this source are in the fdf.	all

3.14.3.2 if-bank address bits 06-09 (see lbd pages 12m and 32m)

The "middle" four bits of the if bank address bus receive their inputs from one of five sources as shown below with the BR code(s) that enable(s) each source:

source for if-bank address bits 06-09	BR code(s)
1. the next-address field (CRNA06-09) via a 74S240 section enabled by NA2IFM emanating from a pal.	0,2,3,4,5,6,7,8
2. the return stack (FSTK06-09) via portions of 74S240 chips (NAIB06-09.RI) enabled by RTN2IF emanating from a pal.	1,9
3. the firmware development facility (NAIB06-09.EX) via TBNASB. Note that the drivers for this source are in the fdf.	all
4. the 256-way unmasked splatter (PROC00-03) via a 3632 prom enabled by BRPROC and no break splatter (BRKSPL).	C,D
5. the 16-way unmasked break splatter via a 74S241 enabled by BRKSPL. Note that when a break splatter occurs, these four address bits are forced off.	C

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

The four least-significant bits of the if-bank address bus receive six types of inputs as shown below with the BR code(s) that enable(s) each input:

source for if-bank address bits 10-13	BR code
1. 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 74AS151 chip per bit (i.e., eight total chips). The three low-order bits of the BR code select among the eight inputs of the 74AS151. With a BR code of zero, ~crna is selected and sent to the S64 anded with the mask (CRMK00-03). Another gate of the S64 ands ~mask with crna. The effect of these two 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

2. the return stack (FSTK10-13) 1,9
The BR code selects input #1 on the 74AS151 (FSTK10-13) whose output is anded with the mask bit on the 74S64. Another gate on the S64 allows the crna field through if the mask is off. This mechanism provides alternate returns.

3. The firmware development facility (NAIB10-13.EX) via TBNASB which, after disabling all other inputs, routes four bits through the 74S64's. all

4. the 256-way unmasked splatter (PROC04-07) via a 3632 prom, enabled by BRPROC and no break-splatter, through the 74S64's. C,D

5. the 16-way unmasked break splatter via a 74S241 enabled by BRKSPL and wire-or'd to the output of the 3632 prom, and connected to the 74S64's. C

6. the 16-way masked splatters 2,3,4,5,6,7
The 74AS151's are selected by the three low-order 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:

splatter	BR code
a. four bits of aram address	2
b. four t flags	3
c. operand-type information	4
d. four arithmetic indicators	5
e. four outputs of the opmux	6
f. four leading-zero indicators	7

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 dectection 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. stack overflow or underflow
8. control store and pal parity

3.15.1.1 procedure parity (see lbd pages 15m, 16m, and 34m)

When procedure is delivered from the memory subsystem, 32 information bits are accompanied by 4 parity bits. These parity bits are captured (by PALOOK or PELOOK) in a register parallel to that which captures the procedure itself. In the meantime, the latched procedure bus (BUSP00-15) is monitored by two 74S280 parity checkers. The trick is to select the parity bits which correspond to the procedure bytes now on the pbus. This selection is performed by a pal which receives as input the 4 parity bits stored for "pa", the 4 parity bits stored for "pe", and the take counter (resychronized to match the pal's needs) emitting a selected parity bit (PARP00 and PARP08) for each 74S280. The output of the 74S280 which monitors BUSP(0-7) is gate by PEKLFT when "left" procedure is sampled while the output of the 74S280 which monitors BUSP(8-15) is gate by PEKRGT when "right" procedure is sampled ("peeked" or "taken").

In order to allow testing of procedure related hardware. both PEKLFT and PEKRGT are inhibitted in a firmware step containing a NOFALT micro. Otherwise PERPRC is sent to the general error collector DSASTR and to the syndrome register.

3.15.1.2 data parity (see lbd pages 32d, 33d and 27m)

When inra or inrb is placed upon the zbus, stimuli coming from a pal (NOFALT.DT) which decodes the field TC, gates the output of the 74S280 parity checkers which monitor the four zbus bytes. Two pair of outputs are collected in PERDTL and PERDTR. The result (PERDAT) is sent to DSASTR and to the syndrome register.

3.15.1.3 procedure red (see lbd pages 16m and 27m)

When procedure is delivered from the memory subsystem, 32 information bits are accompanied by 2 "red" indicators. These indicators, accusing the left 16 bits or the right 16 bits of containing an uncorrectable edac error are captured (by PALOOK or PELOOK) in a register parallel to that which captured the procedure itself. A simple selection is now required as a function of which procedure bytes are being sourced to the pbus. This selection is performed by a 16L8A pal which receives the two red bits stored for "pa", the two red bits stored for "pe", and the take counter (resynchronized to match the pal's needs). The error signal (REDPRC) is sent to the general error collector (DSASTR) and to the syndrome register.

3.15.1.4 data red (see lbd pages 27m and 34m)

When inra or inrb is placed upon the zbus, stimuli coming from a pal which decodes the ZB field, enables two sections of a 74S157 selector (REDDTL and REDDTR) which discover whether the left and/or right data words had good edac. The two data-red error signals are or'd (REDDAT) and sent to the general error collector (DSASTR) and to the syndrome register.

3.15.1.5 procedure uar (see lbd pages 16m and 27m)

When procedure is delivered from the memory subsystem, 32 information bits are accompanied by 2 unavailable resource indicators. The "left" indicator means that the system resource addressed does not exist whereas if the left indicator is off, the "right" indicator means that the address delivered to the memory subsystem from adrp was the last one installed in this system. A pal samples the four error signals (PAUARL, PAUARR, PEUARL, and PEUARR) and determines whether an error (UARPRC) is to be sent to the general error collector (DSASTR) and to the syndrome register.

3.15.1.6 data uar (see lbd pages 34m and 27m)

When inra or inrb is placed upon the zbus, stimuli coming from a pal which decodes the ZB field, enables two sections of a 74S157 selector (uardtl and uardtr) which discover whether the system resource addressed exists (i.e., a left uar) and whether the memory location provided to the memory subsystem from adra or adrb, while requesting four bytes, was the last address of installed memory (i.e., a right uar and no left uar). The outputs of these two sections (UARDTL and UARDTR) are or'd in UARDAT and sent to the general error collector (DSASTR) and to the syndrome register.

3.15.1.7 stack overflow or underflow (see lbd page 24m)

The stack overflow/underflow detector is a pal which monitors the address register (FWSTK0-3), the nature of the stack transaction (CDPUSH, POPING) and an error detector enable (FLAGC7). It generates two signals:

1. STKOUT: a stack error was detected (i.e., a push was attempted and the stack was full, or a return was attempted and the stack was empty). This signal is connected to the general error collector (DSASTR) and to the syndrome register.

2. STKUNW: a stack underflow error was detected (i.e., a return was attempted and the stack was empty). This signal is sent to the syndrome register to distinguish overflow from underflow.

3.15.1.8 control-store parity (see lbd pages 5m, 6m, 8m, 9m, 11m, 3d, 4d, 5d, 6d, 7d, 8d, 9d, 10d, and 11d)

The 128-bit control-store array contains one parity bit in each thirty-two. The parity bits are CSXX16, CSXX48, CSXX80 and CSXXB2. In each thirty-two-bit group, the control-register (cr) outputs are tested for correct parity. In some firmware fields, raw control-register flops do not exist because registered pals are used to perform "in-flight" decoding of the control-store output. In these instances, the pal(s) check(s) parity on the portion of the field in its purview and sends the result along with raw cr bits to a gaggle (14) of 74S280 parity checker chips.

Parity chip EP0007 checks parity on the first nine control-store bits, EP0815 checks parity on the next nine (including the parity bit for this thirty-two-bit group), EP1623 generates parity for the next eight and accepts PORTLY as an input which represents the next three, and finally FWPAR0 gathers them all up along with PARTLY, which represents the last three. The error signal FWPAR0 is sent to the general error collector (DSASTR) and to the syndrome register.

In the second group of 32 control-store bits, EP3239 checks parity on the first nine bits, EP4047 checks parity on the second nine bits including the parity bit for this group, EP4855 checks parity for the next nine bits and FWPAR1 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 (CKPRTY gathers the first two), EP7381 gets the next nine including the parity bit for this group, and FWPAR2 gathers the remaining fourteen (four via BRPRTY, three via TCPRTY.13, and three via TCPRTY.46) in addition to accepting the output of the previous checkers. The error signal is sent to the general error collector (DSASTR) and to the syndrome register.

In the fourth group of 32 control-store bits, EP96A3 checks parity on the only eight of this group (a pal gathers three bits into CRZBPB) which are on the daughter board, and sends this knowledge to the mother board where EPA8B6 checks parity on the next nine bits including the parity bit for this group, and FWPAR3 gets the last fifteen bits (three via FTPRTY, four via BSPRTY.03, and four via BSPRTY.47) in addition to gathering the contributions of the previous checkers. The error signal FWPAR3 is sent to the general error collector (DSASTR) and to the syndrome register.

3.15.2 parity generation circuits (see lbd page 35d)

When outr is loaded, four parity bits are created by four 74S280 chips which each examine one byte of the dbus and place the parity bit in a register loaded by the same signal as that which loads outr. These four parity bits accompany the data wherever its final destination may be (e.g., the memory, an i/o controller). See also 3.11.2.

3.15.3 verifying the integrity circuits (see lbd pages 35d)

When parity is generated on the dbus 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 (FLAGC0 is off) and if FLAGC2 is on. This mechanism, in conjunction with the WRAP micros, allows a firmware routine (presumbly 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.

The micro 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 27m)

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 FLAGC5 controls whether the flop DSTFF1 will set leading toward the hardware interrupt. If FLAGC5 if off, DSTFF1 starts a chain of events which will cause location zero to be the next executed. The delay-line driver DSA000 starts an edge down the delay line DSA200 while, at the same time, the otherwise intended next address is being allowed to settle in the next-address generator. After 200 nanoseconds, DSTFF2 sets, clocking the intended next address into the hardware-interrupt-address register (HWIA00-13). A 100-nanosecond pulse is formed by HOLDIT which clears the next-address control-register flops (CRNA00-13) and the test-condition muxes are disabled, forcing an else bank enable. The next-address generator emits zero and the control-store array is accessed. A clock stall signal (WAITUP), which has been active through this entire ceremony, is allowed to relax and the content of location zero is executed next.

FIRMWARE DESCRIPTION

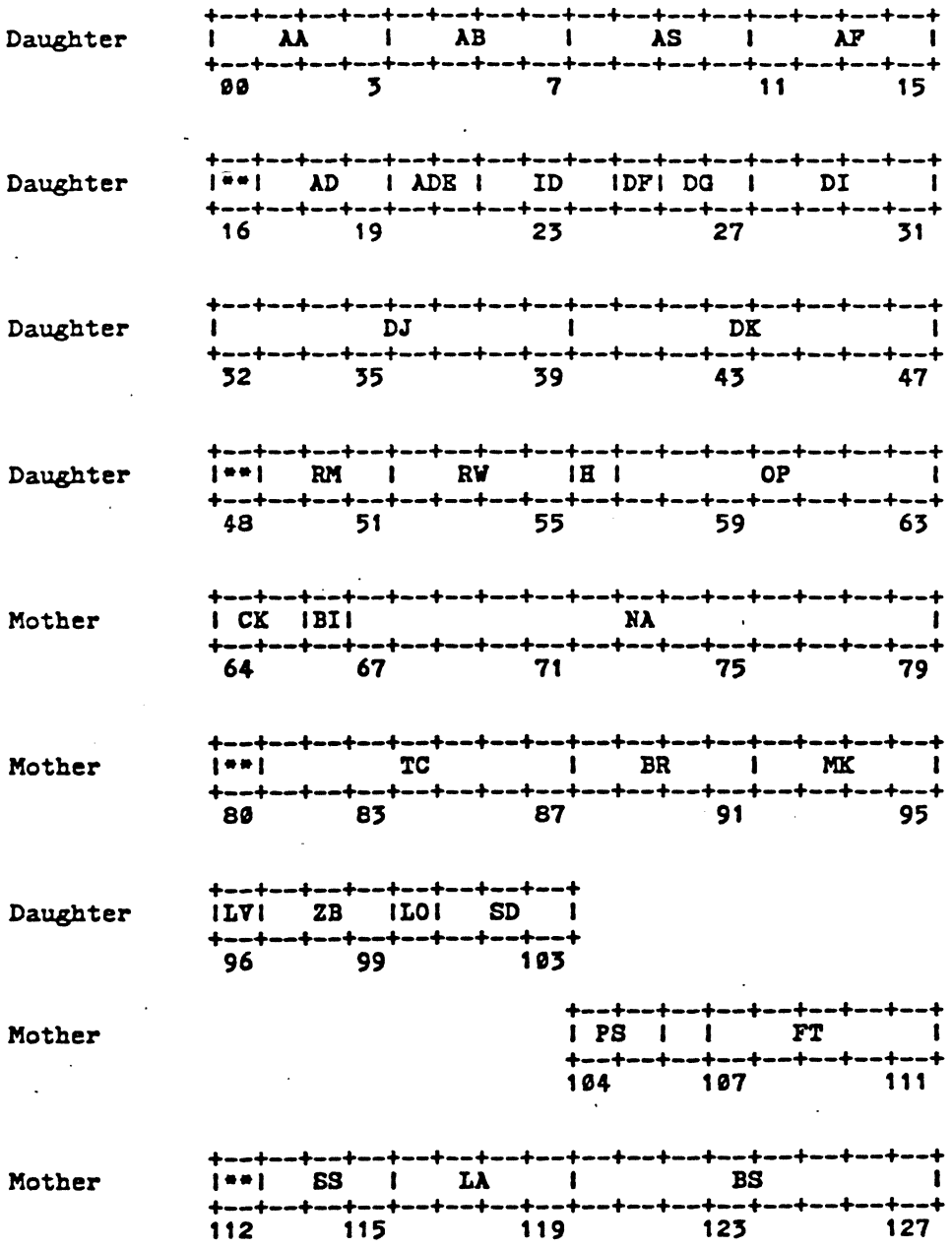
4.0 FIRMWARE DESCRIPTION

This section provides a detailed description of the 32 bit Custom Processor firmware structure.

The custom processor divides its firmware word into thirty-one fields (see figure 4-1). The discussion that follows 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 one of the groups is allowed in each step. The fourteen zones are listed below with a brief description of the related hardware each zone controls:

I. 2901 control	The six fields which control the 2901 are AA, AB, AS, AF, AD, and ADE.
II. ARAM control	The two fields which control the ARAM are RM, and RW.
III. D BUS control	The five fields which control the D bus are DF, DG, DI, DJ, and DK.
IV. S BUS control	The field which controls the S bus is SS.
V. Z BUS control	The field which controls the Z bus is ZB.
VI. LOCAL BUS control	The field which controls the Local bus is BS.
VII. FLAG control	The field which controls the TEMP flags is FT. The field which controls the PERM and CONTROL flags is BS.
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 ID.
X. OP register control	The field which controls the loading and modification of the OP register is OP.
XI. LOAD control	The fields which control the loading of other major registers are LA, LO, LV and H and OP.
XII. CLOCK control	The field which controls the Clock speed is CK.
XIII. STACK control	The field which controls which firmware address (if any) is pushed onto the stack is PS.
XIV. NEXT ADDRESS control	The five fields which control the value of the next firmware address are BI, NA, TC, BR and MK.

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 ** = F/W PARITY CHECKS

FIGURE 4-1

THE CUSTOM PROCESSOR FIRMWARE WORD FORMAT

4.1 2901 CONTROL

Six fields which control the 2901 register file and ALU:

AA provides the four bits which select the register file location available at the A port. The coder specifies the A-port address in 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 in an argument of an AS field and/or AD field micro (see below).

AS selects two operands (r/s) for manipulation by the ALU from among five sources: the A port, the B port, the Q register, the D bus and zero. The available micros are:

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.

AF determines what manipulation shall be performed on the two operands provided by the AS field. The available micros are:

micro	the alu output receives:
F:ADD1	the s input plus the r input plus one
F:ADDC	the s input plus the r input [plus one if the carry indicator (ind1) is true].
F:ADDC'	the s input plus the r input [plus one if the carry indicator (ind1) is false].
F:ADD	the s input plus the r input.
F:S-R	the s input plus not the r input plus one
F:S-R-C'	the s input plus not the r input [plus one if the carry indicator (ind1) in on].
F:S-R-C	the s input plus not the r input [plus one if the carry indicator (ind1) is off].
F:S-R-1	the s input plus not the r input.
F:R-S	the r input plus not the s input plus one.
F:R-S-C'	the r input plus not the s input [plus one if the carry indicator (ind1) is on].
F:R-S-C	the r input plus not the s input [plus one if the carry indicator (ind1) is off].
F:R-S-1	the r input plus not the s input.
F:OR	the s input inclusive ored with the r input.
F:SR	the s input anded with the r input.
F:SR'	the r input anded with not the s input.
F:XOR	the s input exclusive ored with the r input.
F:XNOR	the s input exclusive ored with not the r input.

AD 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 32-bit output of the ALU may be shifted one bit left or right and stored in the register file at location AB. The 64-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:

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,ADE)	the 64-bit concatenation of the alu output and the q register are shifted right one bit position. The shift-end-effect control (see ADE below) determines the value of the most-significant bit. the 32 most-significant bits of the shifted result are written at location AB in the register file. the 32 least-significant bits are written into the q register. the zbus control field may also select the alu output.
Y:F'BQ:FQSL (AB,ADE)	the 64-bit concatenation of the alu output and the q register are shifted left one bit position. The shift-end-effect control (see ADE below) determines the value of the least-significant bit. the 32 most-significant bits of the shifted result are written at location AB in the register file. the 32 least-significant bits are written into the q register. the zbus control field may also select the alu output.

micro	description
Y:F'B:FSR (AB,ADE)	the alu output is shifted right one bit position. The shift-end-effect control (see ADE below) determines the value of the most-significant bit. the 32 most-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the alu output.
Y:F'B:FSL (AB,ADE)	the alu output is shifted left one bit position. The shift-end-effect control (see ADE below) determines the value of the least-significant bit. the 32 least-significant bits of the shifted result are written at location AB in the register file. the zbus control field may also select the alu output.

ADE controls the "shift end effects"; i. e.,the bit which shifts into the ALU (for 32- and 64-bit shifts) and the bit which shifts into Q (for 64-bit shifts). There are six choices for insertion: the most- or the least-significant bit of the ALU, the most- or the least-significant bit of Q, the one-bit value of Flagt1 (early), or a literal zero. Combinations are restricted to four for any given direction and length of shift. As an example, for a 64-bit circular left shift:

```
Y:F'BQ:FQSL(AB,CIRC)
```

4.2 ARAM CONTROL

Two fields control the ARAM:

RM determines which of the seven sources of ARAM addressing is to be used. Four of the sources are restricted to the first 256 ARAM locations where RBR further isolates a group of 16 and OPA, OPB, OPC, or OPD select one of those 16 to be accessed. A twelve-bit literal may be used as an ARAM addressing source or the twelve-bit content of register RAR. The last ARAM addressing source allows the high 2048 locations of the ARAM to be used as a cache mechanism where eleven Z bus bits address the 2048-location ARAM block and information within the addressed location may determine hit versus miss as well as provide appropriate retained values which may be used for algorithm acceleration.

RW determines which (if any) of the ARAM data bytes selected are to be overwritten. If RW=0, the content of the ARAM location is left unchanged; otherwise, one or more bytes will be made equal to the value of the sbus-late.

4.3 D-BUS CONTROL

The D bus has many sources. The five fields discussed in this paragraph determine which source or combination of sources is allowed to the D bus. The DG field specifies how the other four fields are to be interpreted. In general, DF is a fill bit which is replicated where necessary to create 00 or FF bytes; DI has a bit per byte specifying which D bus Byte(s) shall receive the eight bit literal DK; DJ is sometimes a literal and sometimes a code choosing among many and varied D bus sources. The dbus is subdivided into 4 bytes named w, x, y, and z.

4.3.1 DG=literal

When DG=literal, DF, DI, DJ and DK provide five types of literals:

micro	byte w	byte x	byte y	byte z
D:00JK	zeros	zeros	DJ	DK
D:FFJK	ones	ones	DJ	DK
D:JK00	DJ	DK	zeros	zeros
D:JKFF	DJ	DK	ones	ones
D:JKJK	DJ	DK	DJ	DK

4.3.2 DG=broadside

D:HEX	DK	DK	hex decoder
D:HHPP	H	H	procedure bus (00-15)
D:OP	DK	DK	OPA/OPB OPC/OPD
D:OP-BIT-AD	zeros	zeros	DK 0,0,0,OPC(0-3),0
D:OPCD	zeros	zeros	OPC OPD
D:OPCD-X2	DK	DK	0000000,OPC(0-3),OPD(0-3),0
D:OPCD-X2'1	DK	DK	0000000,OPC(0-3),OPD(0-3),1
D:OPHSTRY	DK	DK	OPA/B/C/D of previous step
D:PB1	zeros	zeros	DK PBUS(0-7)
D:PB2	DK	zeros	procedure bus (00-15)
D:-PB2	ones	ones	procedure bus (00-15)
D:PCTR	DK	p r o g r a m c o u n t e r	
D:PHSTRY	DK	program counter at last rni	
D:RAMAD-X8	zeros	000000000000,DK(4-7,0),RAMAD(8-11),DK(1-3)	
D:RBR'RAR	DK	DK	RBR(0-3),RAR(0-11)
D:Z	z bus	z bus	z bus z bus

4.3.3 DG=mix

Certain sources may, on byte boundaries, be mixed onto the dbus. These sources are the ARAM, ADRA, ADRB, or ADRP via the sbus-early; the Z bus via the shifter and the sbus-late; the H register; a literal byte (DK); a fill byte (DF). The micro D:: requires up to six arguments, the first four specify the source for bytes w, x, y and z, respectively; the fifth specifies the eight-bit literal if any, and the sixth specifies the value of the fill byte, if any.

4.3.4 DG=sign-extend

Three sign extend micros are provided to place onto the D bus some right portion of the S bus with the most-significant bit of that right portion replicated in all bits to the left.

micro	byte w	byte x	byte y	byte z
D:SEX8	8*sbus(24)	8*sbus(24)	8*sbus(24)	sbus(24-31)
D:SEX16	8*sbus(16)	8*sbus(16)	sbus(16-23)	sbus(24-31)
D:SEX24	8*sbus(08)	sbus(08-15)	sbus(16-23)	sbus(24-31)

4.4 SBUS CONTROL

One field, SS, controls which source is placed upon the S bus. This field allows two different sources onto the sbus in the same step. A "split cycle" is a firmware step in which the sbus receives from a different source during the early phase than it does during the late phase. Two types of split cycles are provided: the first sends ADRS (ADRA, ADRB, or ADRP) to the sbus during the early phase and the Z bus during the late phase, whereas the second sends the ARAM to the sbus during the early phase and the Z bus during the late phase.

micro		Byte w	byte x	byte y	byte z
S:ADRA	early	adra(00-07)	adra(08-15)	adra(16-23)	adra(24-31)
	late	adra(00-07)	adra(08-15)	adra(16-23)	adra(24-31)
S:ADRA'Z	early	adra(00-07)	adra(08-15)	adra(16-23)	adra(24-31)
	late	Z bus(00-31) nibble shifted by SD			
S:ADRB	early	adrb(00-07)	adrb(08-15)	adrb(16-23)	adrb(24-31)
	late	adrb(00-07)	adrb(08-15)	adrb(16-23)	adrb(24-31)
S:ADRB'Z	early	adrb(00-07)	adrb(08-15)	adrb(16-23)	adrb(24-31)
	late	Z bus(00-31) nibble shifted by SD			
S:ADRP	early	adrp(00-07)	adrp(08-15)	adrp(16-23)	adrp(24-31)
	late	adrp(00-07)	adrp(08-15)	adrp(16-23)	adrp(24-31)
S:ADRP'Z	early	adrp(00-07)	adrp(08-15)	adrp(16-23)	adrp(24-31)
	late	Z bus(00-31) nibble shifted by SD			
S:ARAM	early	aram(0-7)	aram(8-15)	aram(16-23)	aram(24-31)
	late	aram(0-7)	aram(8-15)	aram(16-23)	aram(24-31)
S:ARAM'Z	early	aram(0-7)	aram(8-15)	aram(16-23)	aram(24-31)
	late	Z bus(00-31) nibble shifted by SD			
S:STK'ACCT	early	0,0,top of return stack, accounting timer			
	late	0,0,top of return stack, accounting timer			
S:SYND	early	?	syndrome at latest dsastr or "synd"		
	late	?	syndrome at latest dsastr or "synd"		
S:Z	early	Z bus(00-31) nibble shifted by SD			
	late	Z bus(00-31) nibble shifted by SD			

4.5 Z BUS CONTROL

The zbus may receive from any one of six sources. Two of the six sources are INRA and INRB which each may receive 32 bits of data from the outside world. Two flavors of INRA and INRB sourcing are provided. One flavor validates the parity on all four bytes, whereas the other validates the parity only on the leftmost two bytes. Another zbus source comes from an external element. This interface is intended for testing purposes only. The other three sources to the zbus are the ALU, the V register, and the dbus.

micro	byte w	byte x	byte y	byte z
Z:D	DBUS(0-7)	DBUS(8-15)	DBUS(16-23)	DBUS(24-31)
Z:EXT	FDF(0-7)	FDF(8-15)	FDF916-23)	FDF(24-31)
Z:V	V(0-7)	V(8-15)	V(16-23)	V(24-31)
Z:Y	ALUY(0-7)	ALUY(8-15)	ALUY(16-23)	ALUY(24-31)
Z:INRA	INRA(0-7)	INRA(8-15)	INRA(16-23)	INRA(24-31)
Z:INRB	INRB(0-7)	INRB(8-15)	INRB(16-23)	INRB(24-31)

4.6 LOCAL-BUS CONTROL

The BS field controls the local bus. Thus, the BS 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 BS field, which cause procedure bytes to be consumed by an incrementation of PCTR. If all this sounds complicated, it is. The initiation, monitoring, and consummation of local-bus 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 two bytes are available in the prefetch buffer before entering any step which will examine or consume one or two such bytes; e.g., before entering a step which contains a PTAKE1, an OP:P4, a D:PB2, or an RNI micro.

2. Stall unless and until the local-bus interface is quiescent before entering a step which will initiate a new local-bus 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 local-bus interface is quiescent before entering a step which will consume procedure bytes, if and only if the procedure buffer has at least four "empty" byte positions; i.e. a step which contains a PTAKE or an RNI'REFILL micro.

Other stalls must be stated explicitly as follows:

1. Stall unless and until the local-bus interface is quiescent before leaving a step which contains a STALL micro.
2. Stall unless and until any outstanding non-procedural request has concluded before leaving a step invoking a "data stall"; i.e., before leaving a step which contains PEEK'STL or a PTAKE1'STL, or a PTAKE2'STL micro.

3. Stall unless and until the specified or implied input register has received the requested data before leaving a step invoking such a stall; i.e., before leaving a step which contains STALL-A (waits for new data to enter inra) or STALL-B (waits for new data to enter inrb) or micros of the form RD-..'STL(A/B).

The BS field supplies the following local bus micros:

micro	description
PEEK	view one or two bytes of procedure whose memory addresses are pctr and pctr+1; verify parity. The two procedure bytes are available to be loaded into OP or to be placed onto the d bus or both.
PEEK'STL	same as peek, but stall before leaving this step until any non-procedural local-bus activity terminates.
PREFETCH	stall before entering this step until all local bus activity has concluded; then initiate a procedure read with the address in adrp. When the memory responds, place the four bytes into procedure buffers a, b, c, and d.

PTAKE0	same as peek, but stall before entering this step if the procedure buffer has space for four bytes and if so, initiate a four byte procedure request.
PTAKE1	same as PTAKE0, but add one to pctr.
PTAKE1'STL	same as PTAKE1, but stall before leaving this step if and until any non-procedural local-bus activity terminates.
PTAKE2	same as PTAKE0, but add two to pctr.
PTAKE2'STL	same as PTAKE1'STL, but add two to pctr.
RD-2B(SA)	stall before entering this firmware step until all local-bus activity has concluded; then initiate a read request on the local bus, reading two bytes at memory addresses adra/b-adra/b+1. Inra/b(0-7) and (16-23) will receive the first byte and inra/b(8-15) and (24-31) will receive the second byte.
RD-2B'STL(SA)	same as RD-2B'(SA), but stall before leaving this step until the two bytes requested have arrived into inra/b.
RD-4B(SA)	stall before entering this step until all local-bus activity has subsided; then initiate read request reading four bytes on the local bus at adra/b, adra/b+1, adra/b+2, and adra/b+3.
RD-4B'STL(SA)	same as RD-4B(SA), but stall before leaving this step until the four bytes requested have arrived into inra/b.
RDLK-2B(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a read-and-lock-memory request on the local bus. Note: ack must be tested before attempting to remove data from inra/b. If ack is not received, memory module was previouly locked, indicating the resource was seized by another requester. Note: any successful rdlk must be followed by either a rdul or a wrul micro; otherwise, the memory module will remain seized indefinitely.
RDUL-2B(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a read-and-unlock memory request on the local bus reading two bytes at memory addresses adra/b and adra/b+1. Inra/b(0-7) and inra/b (16-23) will receive the first byte; inra/b (8-15) and (24-31) will receive the 2nd byte. The memory module is released.

RD-IO(SA)	stall before entering this step until all local-bus activity has subsided; then initiate an i/o read on the megabus, 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.
RD-LCL(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a read local using adra(8-9) as the channel number and adra(10-15) as the function code. (For reading memory subsystem control information).
REPLY(SA)	stall before entering this step until all local-bus 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.
WR-1B(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a memory byte write request on the local bus. If adra/b is even, then outr(0-7) is written; if adra/b is odd, then outr(8-15) is written.
WR-1'1B(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a memory write request on the local bus. If adra/b is odd, then outr(8-23) is written; if adra/b is even, the results are unspecified.
WR-1'2B(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a memory write request on the local bus. If adra/b is odd, then outr(8-31) is written; if adra/b is even, the results are unspecified.
WR-2B(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a memory write request on the local bus. Outr(0-15) is written and adra/b(31) is assumed be zero!
WR-2'1B(SA)	same as WR-2B(SA), but outr(0-23) is written.
WR-4B(SA)	same as WR-2B(SA), but outr(0-31) is written.
WR-IO(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a megabus non-memory request to the channel in adra/b(16-25) and the function code in adra/b (26-31). Send outr(0-15) to the specified channel.
WR-LCL(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a local bus non-memory request to the channel adra(8-9) and the function code adra(10-15). Send outr(0-31) to the specified channel.

WRAP(BSA)	copy outr(0-31) into inra(0-31), inrb(0-31), procedure buffers abcd, or procedure buffers efgh. Note: the coder is responsible for guaranteeing that the local bus is quiescent!
WRLK(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a memory-write-lock request on the local bus. If the memory module was not locked, then outr(0-15) is written at adra/b, (adra/b31 is assumed to be zero). If the memory module was previously locked, no data is written. The coder must test ack to determine which result occurred. Note: any successful WRLK must be followed by either a RDUL or a WRUL micro; otherwise, the memory module will remain seized indefinitely.
WRUL(SA)	stall before entering this step until all local-bus activity has subsided; then initiate a write-and-unlock memory request on the local bus. Outr(0-15) is written at adra/b (adra/b31 is assumed to be zero). The memory module is released.

4.7 FLAG CONTROL

There are three groups of flags. Each group has eight elements for a total of twenty-four flags. They are: the eight permanent flags, the eight control flags, and the eight temporary flags.

4.7.1 Permanent flags

Sixteen micros are dedicated to the eight permanent flags. There are no side effects which result from permanent flag actions. However, since the BS field controls the permanent flags, no local bus activity can be initiated in steps which set or clear permanent flags. Also, since the BS field allows consuming the procedure stream and local-bus transaction initiation, ptakes and setting or clearing of control flags are all mutually exclusive with modifications to the permanent flags.

4.7.2 Control flags

Sixteen micros are dedicated to the eight control flags. Each control flag is dedicated to a hardware entity. FlagC0 is dedicated to testing. FlagC1 and C2 are dedicated to testing only when flagC0 is off. FlagC3 is dedicated to the accounting timer. FlagC5 allows the microcoder to ignore hardware detected faults (e.g., memory parity). FlagC7 is dedicated to the return stack. Note that altering the state of a control flag is mutually exclusive with initiating local bus activity, viewing the procedure stream or changing the state of a permanent flag.

4.7.3 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. Micros which control the temporary flags are mutually exclusive with micros of the form RAR:N'S.. or RARH:N (loading rarh with a literal) or PUSH-R (relative push) or RAMAD:LIT (addressing the aram with a literal).

4.8 Nibble-shifter control

The field which controls the nibble shifter is SD. The nibble shifter rotates the 32 bits of the z bus from zero to seven places (any multiple of four bit positions) and puts the result onto the s bus. 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 the sbus to receive the zbus.

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 purpose of performing operations on data whose width is less than 32 bits, arithmetic indicator sensing can be partitioned to the rightmost 8 bits or the rightmost 16 bits or the rightmost 24 bits as well as all 32 bits. The argument (id23) provides control of the partioning using convenient labels:

dblw (doubleword)	ind(0-4) are sensitized to the full width of the alu i.e., 32 bits.
addr (address)	ind(-4) are sensitized to alu(8-31); e.g., ind2 captures alu(8).
word	ind(0-4) are sensitized to alu(16-31); e.g., ind3 becomes true if alu(16-31) are all zero.
byte	ind(0-4) are sensitized to alu(24-31); e.g., ind4 becomes true if alu(24-31) are all zero and ind3 is on.

Arithmetic-indicator partitioning has no effect on ind5 since it monitors only the least significant bit of the z bus [z(31)]. Overflow (ind0) is intended for add or subtract operations and furthermore lives up to its name only in two's complement arithmetic. Ind0 becomes true if the carry into the leftmost bit (as partitioned) does not match the carry out of the same bit.

4.9.2 Miscellaneous indicators

The miscellaneous indicators are all stored in one "register" but may be thought of as four separate groups. The one indicator (ind6) which stores the output of the stop-code ram is the simplest of the lot. Note that the stop-code ram is addressed by byte y of the dbus. The second group, ind(7,8) are dedicated to detecting frame-bound crossings where a frame is a 512-byte contiguous block of memory, starting at any multiple of 512 bytes.

ind6	the stop-code ram indicator stores the output of the stop-code ram as addressed by dbus (16-23).
ind7	frame-bound indicator #7 becomes true if dbus bits 15 through 22 do not match zbus bits 15 through 22.

ind8	frame-bound indicator #8 assumes the addition of three values: a base address, a displacement, and an operand length where: the base address is in a ralu register, the displacement is on the d bus, and the operand length is in type(0,1) where 00 is length one, 01 is length two, 10 is length four, and 11 is length six. Ind8 becomes true if the three way add would produce a carry out of alu bit 23. Note that ind#8 operates equally well if the ralu register contains the displacement and the dbus has the base address.

The next indicator group contains one indicator which becomes true if zbus bits 8 through 20 match sbus bits 16 through 28, respectively. With careful planning, the hash indicator permits detection of a "hit" by addressing the aram with zbus bits 19 through 29, and placing the aram data onto the sbus for viewing by this indicator.

ind9	hash-hit indicator becomes true if zbus bits 8 through 20 are equal to sbus bit 16 through 28.

The last indicator group provides a mechanism for detecting the postion of an operand's most-significant nibble.

ind(10-13)	the leading-zero indicators store the number of leading zeros at the output of the alu.

Micros which appear to allow less than the complete set of miscellaneous indicators to be stored are provided merely to inform the automated clock speed calculation program of the coder's intent since setting the clock speed to allow for the slowest of the eight would be an unnecessary performance sacrifice.

4.10 OP-REGISTER CONTROL

The OP 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 OP may be incremented or decremented and since each nibble may be tested for containing extreme values, each is well suited to provide loop closure functions. For this reason (and others), OP may be loaded from various nibbles of the sbus. The multiplexer which selects what data is presented to the opa register inputs is also made available for branching upon. Thus, sixteen-way branches are available on any pbus nibble or on any sbus nibble (see paragraph 4.14).

4.10.1 Pbus to OP

Those OP-register loads which capture nibbles of the instruction stream are:

micro	description
OP:P0	opa receives pbus(0-3), opb receives pbus(4-7), opc receives pbus(8-11), opd receives pbus(12-15)
OPA:P0	opa receives pbus(0-3)
OPA:P4	opa receives pbus(4-7)
OPA:P8	opa receives pbus(8-11)
OPA:P12	opa receives pbus(12-15)
OPB:P0	opb receives pbus(0-3)
OPB:P4	opb receives pbus(4-7)
OPB:P8	opb receives pbus(8-11)
OPB:P12	opb receives pbus(12-15)
OPC:P0	opc receives pbus(0-3)
OPC:P4	opc receives pbus(4-7)
OPC:P8	opc receives pbus(8-11)
OPC:P12	opc receives pbus(12-15)
OPD:P0	opd receives pbus(0-3)
OPD:P4	opd receives pbus(4-7)
OPD:P8	opd receives pbus(8-11)
OPD:P12	opd receives pbus(12-15)
OPA'B:P0	opa receives pbus(0-3), opb receives pbus(4-7)
OPA'B:P8	opa receives pbus(8-11), opb receives pbus(12-15)
OPC'D:P0	opc receives pbus(0-3), opd receives pbus(4-7)
OPC'D:P8	opc receives pbus(8-11), opd receives pbus(12-15)

4.10.2 Sbus to OP

The OP-register micros which capture nibbles of the sbus are:

micro	description
OP:S0	opa receives sbus(0-3), opb receives sbus(4-7), opc receives sbus(8-11), opd receives sbus(12-15)
OP:S16	opa receives sbus(16-19) opb receives sbus(20-23) opc receives sbus(24-27) opd receives sbus(28-31)
OPA:S0	opa receives sbus(0-3)
OPA:S16	opa receives sbus(16-19)
OPB:S4	opb receives sbus(4-7)
OPB:S20	opb receives sbus(20-23)
OPC:S8	opc receives sbus(8-11)
OPC:S24	opc receives sbus(24-27)
OPD:S12	opd receives sbus(12-15)
OPD:S28	opc receives sbus(28-31)
OPA'B:S0	opa receives sbus(0-3), opb receives sbus(4-7)
OPA'B:S16	opa receives sbus(16-19),opb receives sbus(20-23)
OPC'D:S8	opc receives sbus(8-11),opd receives sbus(12-15)
OPC'D:S24	opc receives sbus(24-27),opd receives sbus(28-31)

4.10.3 OP increment/decrement

The OP-register micros which increment or decrement an op register are:

micro	description
OPA-DEC	opa receives opa minus one (if opa was zero, test condition opa-wr becomes true).
OPB-DEC	opb receives opb minus one (if opb was zero, test condition opb-wr becomes true).
OPC-DEC	opc receives opc minus one (if opc was zero, test condition opc-wr becomes true).
OPD-DEC	opd receives opd minus one (if opd was zero, test condition opd-wr becomes true).
OPA-INC	opa receives opa plus one (if opa was F, test condition opa-wr becomes true).
OPB-INC	opb receives opb plus one (if opb was F, test condition opb-wr becomes true).
OPC-INC	opc receives opc plus one (if opc was F, test condition opc-wr becomes true).
OPD-INC	opd receives opd plus one (if opd was F, test condition opd-wr becomes true).

4.11 LOAD CONTROLS

Many registers are found lurking on the ends of buses just waiting for an opportunity to snarf up the data therefrom. These registers are adra, adrb, adrp/pctr, outr, v, h, rar, and rbr.

4.11.1 Loading adra, adrb or adrp/pctr

Adra may be loaded from the sbus or may be incremented by four or may be decremented by four. The same applies to adrb. Adrp may be loaded from the sbus three different ways. Micros which load adrp also load pctr. The applicable micros are:

micro	description
ADRA:S	adra(00-31) receive sbus(00-31).
ADRB:S	adrb(00-31) receive sbus(00-31).
ADRP:S	adrp(00-31) receives sbus(00-31). pctr(00-31) receives sbus(00-31)
ADRPH:S	adrp(00-22) receives sbus(00-22) pctr(00-22) receives sbus(00-22)
ADRPL:S	adrp(23-31) receives sbus(23-31) pctr(23-31) receives sbus(23-31)
ADRA-DEC	adra receives adra minus four
ADRB-DEC	adrb receives adrb minus four
ADRA:INC	adra receives adra plus four
ADRB-INC	adrb receives adrb plus four

4.11.2 Loading of outr, v, or h

The micros which load these registers are:

micro	description
OUTR:D	outr(00-31) receives dbus(00-31)
V:Z	v(00-31) receives zbus(00-31)
H:S	h(00-31) receives sbus(00-31)

4.11.3 Changing rbr

Rbr is a four-bit register which can participate in addressing the aram. When opa, opb, opc, or opd provide the low nibble of ramad, rbr provides the middle nibble. The micros which load or change rbr are:

micro	description
RBR:S0	rbr receives sbus(0-3)
RBR:S8	rbr receives sbus(8-11)
RBR:S16	rbr receives sbus(16-19)
RBR:S24	rbr receives sbus(24-27)
RBR'RAR:S0	rbr receives sbus(0-3), rar receives sbus(4-15)
RBR'RAR:S16	rbr receives sbus(16-19), rar receives sbus(20-31)
RBR-DEC	rbr receives rbr minus one
RBR-INC	rbr receives rbr plus one

4.11.4 Changing rar

Rar is a twelve-bit register which may be used to address the aram. It is subdivided into a four-bit register (rarh) and an eight-bit register (rarl). Either registers may be incremented or decremented.

4.11.4.1 Loading all of rar

The micros which load rar, other than those of paragraph 4.11.3 are:

micro	description
RAR:N'S0(FT)	rarh receives a four-bit literal (FT), rarl receives sbus(0-7)
RAR:N'S8(FT)	rarh receives a four-bit literal (FT), rarl receives sbus(8-15)
RAR:N'S16	rarh receives a four-bit literal (FT), rarl receives sbus(16-23)
RAR:N'S24	rarh receives a four-bit literal (FT), rarl receives sbus(24-31)
RAR:S4	rar receives sbus(4-15)
RAR:S20	rar receives sbus(20-31)

4.11.4.2 Changing rarh

The micros which or changes of rarh, other than those previously mentioned are:

micro	description
RARH:N(FT)	rarh receives a four-bit literal (FT)
RARH:S4	rarh receives sbus(4-7)
RARH:S12	rarh receives sbus(12-15)
RARH:S20	rarh receives sbus(20-23)
RARH:S28	rarh receives sbus(28-31)
RARH-DEC	rarh receives rarh minus one
RARH-INC	rarh receives rarh plus one

4.11.4.3 Changing rarl

The micros which change rarl, other than those previously mentioned are:

micro	description
RARL:S0	rarl receives sbus(0-7)
RARL:S8	rarl receives sbus(8-15)
RARL:S16	rarl receives sbus(16-23)
RARL:S24	rarl receives sbus(24-31)
RARL-DEC	rarl receives rarl minus one
RARL-INC	rarl receives rarl plus one

4.12 CLOCK CONTROL

A preprocessor to the firmware assembler 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 below:

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 up to seventeen firmware return addresses. The return stack operates as a last-in-first-out (LIFO) mechanism, allowing sufficient levels of nesting for most applications. The coder may push firmware addresses onto the return stack in one of three ways:

micro	description
PUSH-D	return stack receives dbus(02-15); i.e., an absolute address which occupies the dbus in this step.
PUSH-R	return stack receives an if bank address relative to the value in the next-address field but may differ from the next address field in the five least significant bits.
PUSH-HWIA	when a hardware detected error occurs, an unprogrammed branch to firmware location zero occurs. The hardware remembers what location was otherwise destined for execution in a register called hwia. This micro allows the captured address to be placed onto the return stack so that the routine which was interrupted may be resumed.

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:

field	explanation
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($THEDEVIL).
TC	test condition - a seven-bit field which allows the coder to specify that the next step shall be a location in the IF bank or a location in the ELSE bank as determined by the result of testing one of seventy-two unary indicators (e.g., IF-I-ZRO($FAIR,$RAIN) In this example the addresses of $FAIR and $RAIN are required to have the following relationship: 1. $FAIR must be in the IF bank 2. $RAIN must be in the ELSE bank 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. 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.
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 four-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 follows:

mechanism	description
GO-TO($SNOW)	specifies an unconditional successor
IF-Z24(2345#,0452#)	specifies a two-way choice as a function of the value of zbus bit 24. If Z24 is on, then 2345 is executed next; otherwise, 452 is executed next.
RETURN	specifies that the succeeding step is that CSA at the top of the seventeen-level return stack.
RETURN'(9)	specifies that the succeeding step is that CSA at the top of the seventeen-level return stack but with the two-weight bit and the four-weight bit set equal to zero (masked by FFF9). Here the MK field contains the four least-significant bits of the mask and the hardware sets the other mask bits to ones.
IF-ACK RETURN-OR($DONT)	specifies that the succeeding step is that CSA at the top of the seventeen-level return stack only if the ACK indicator is on. If the ACK indicator is off, the succeeding step is $DONT. Note that $DONT must be in the ELSE bank, thus it is advised that CSA's pushed onto the return stack be if-bank addresses only. Note also that a conditional masked return of the form: IF-ACK RETURN'-OR(A,$DONT) is also provided (a conditional masked return.

BR-ARITH(9,3210#)	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-ODD. If they are both off, location 3210 will be next executed; if I-SGN is off and I-ODD is on, location 3211 will be next; if I-SGN is on and I-ODD is off, location 3218 will be next; if they are both on, location 3219 will be next. Note that this mechanism permits the examination of two other arithmetic indicators. By specifying a mask with a four-weight bit, I-ZRO may be examined, and by specifying a mask with a two-weight bit, the dynamic alu carry may be examined. Note also that a conditional version, of the form: IF-Z'Z BR-ARITH(4,$CLOUDY) is also provided.
BR-FLAGS(F,3450#)	specifies that the succeeding step is to be one of sixteen in the block of if-bank locations starting at 3450 as a function of four temporary flags (to, t1, t2, and t3). Note that a conditional version of the form: IF-POOF BR-FLAGS(6,$BANG) is also provided.
BR-P0(D,2AB0#)	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,$WIT) is also provided. Note also that the BR mechanism may examine the other three pbus nibbles with equal alacrity.

BR-S8(5,$SLEET)	specifies that the succeeding step is to be one of four in the IF-bank block in which $SLEET resides, as a function of sbus09 and sbus11. Note that a conditional version of the form: IF-NOT-OPA-WR BR-S8(A,$SLUSH) is also provided. Note also that the BR mechanism is available to examine any of the other seven sbus nibbles.
BR-SCALE(F,$SUNNY)	specifies that the succeeding step is to be one of nine in the IF-bank block in which $SUNNY resides as a function of the leading-zero indicators (ind 10 through 13). In this case the number of destinations is limited, not by the mask, but by the fact that the leading zero indicators may store only nine unique values. Note that a conditional version of the form: IF-RUPT BR-SCALE(F,$SMILE) is also provided.
BR-TYPE(3,222C#)	specifies that the succeeding step is to be one of 4 (222C, 222D, 222E, or 222F as a function of the two type-register bits. Note that other mask values allow examination of I-ODD and sbus00. Note also that a conditional version of the form: IF-NOT-BREAK BR-TYPE(F,$SPLASH) is also provided.

BR-PROC(2,$BEGIN)	specifies that the succeeding step is to be one of 256 IF-bank locations. The $BEGIN CSA is used as a base address. A substitution of the eight least-significant bits emanates from an op code decode unit. This unit is, in essence, a table look-up mechanism which contains sixteen tables selected by the mask field (in this case, table #2 is selected). The value from the table becomes the output of the opcode decode unit. This splatter mechanism is intended to permit 16 different interpretations of procedure stream bytes (e.g., based upon position). When this micro becomes part of the combinatorial micro "RNI" the existance of a BREAK overrides the splatter on pbus, and instead performs a 16-way branch on the four sources of BREAK. In this application, the first 16 locations of the splatter must be reserved.

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.,PTAKE1), 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:PB1), any uncorrectable memory error associated with the information is detected.

3. stack checks - whenever return addresses are added to (e.g., PUSH:$SHOVE) or removed from (e.g.,RETURN) the stack, a full or empty condition, respectively, may optionally be treated as an error.

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

5. 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 loaded from the dbus. 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.

6. 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 the memory subsytem. This routine utilizes the integrity features mentioned above and take advantage of the syndrome register and the hardware interrupt register. The syndrome register captures the reason code for any hardware detected error and the hardware-interrupt-address register captures the next address intended had the interrupt not occurred.

THE NANOSECONDS

5.0 THE NANOSECONDS

The assembler rejects micro-op combinations which would result in a clock speed ouside 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 contians a model which depicts the information of figure 5-1 along with a definition of each clock speed (i.e., 105ns, 125ns, 145ns and 175ns).

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 two of the seventy-two test conditions are listed. This is because the other seventy will each succeed at performing its appointed function (bank selection) in no more than 105ns.

Values 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 require the data path and can be ignored otherwise.

2. non-squares (mostly circles) contain the number of nanoseconds the data must pay in order to propagate through.

5.2 the choices

Choices are essentially of two types:

1. depending upon the micro combination, the nanosecond value from the beginning of the firmware step may or may not be determined. A simple example of this is both the input and the output of the box called "PCB". Its input is valid at nanosecond 16 unless the aram address mux has the zbus selected in which case the output of the aram address mux is valid 6ns later than the zbus. That means that the output of the PCB box is valid at nanosecond 30 if the aram address mux isn't selecting the zbus but if it is, then the output of the PCB box is valid 15ns after the zbus.

2. a non-square contains conditions. A simple example is the path through the alu oval. If the alu is required by the F:... micro to perform 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:DBLW
```

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 16 (from the beginning of the step). From there, the alu is encountered where, because an add is to be performed, the 16 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 25.5ns. The subtotal (16+67.5+25.5) is 109ns which, to arrive at the total, must be increased by 10% to account for non-silicon delays (i.e., media). This yields 119.9 which requires a clock setting of 125ns namely, an HF "box".

In a more complicated step, where there are multiple end points, it is necessary to analyze each path in order to discover the longest one. The length of the step is determined by the longest path. An example is shown below:

```
MICRO                                                    CUMUL

RD-4B(ADRA)                                               0.0
RAMAD:RAR                                                16.0
    S:ARAM'Z                                             56.0
    D::(K,A,A,A,,34)                                     62.0
 OUTR:D                                                  90.0*
    R:D'S:A(3)  F:ADD  Y:F'BQ:FQSL(2,OPEN)              146.5*
    Z:Y                                                 131.5
    V:Z                                                 136.5
IND-D:Z                                                 146.0
OPC'D:P0                                                 48.0*
    S:ARAM'Z (again)                                    143.5
    H:S                                                 148.5*
 ADRB:S                                                 159.0*
IF-NOT-BREAK BR-PROC(7,$NOWAY)                          133.5*
```

*=end point

In this step, RAMAD is valid at 16, the ARAM readout is valid on the sbus at 56 , and the dbus receives twenty-four bits of the sbus at 62 and eight bits at 30 from the literal source. The first end point, OUTR, is encountered at 90 (62+28). F3's content is added to the dbus data in the alu shifted, along with Q, and written into the ram yielding 146.5 (62+67.5+17) for the second end point. The alu output is sent to the zbus at 131.5 (62+67.5+2) and to the third end point, the V register at 136.5. Seven bits of the dbus are compared to seven bits of the zbus and sent to the fourth end point, IND7, at 146 (from z at 131.5 + 14.5). The zbus is sent to the sbus unshifted at 143.5 (131.5+12). From there the fifth end point (H) receives the sbus at 148.5. The sixth end point (ADRB) receives the sbus (148.5) at 159.0. The last end point is the next firmware address at 133.5 (68.5+65). Thus, loading ADRB is the longest path and with the addition of 10% (175.0) requires, the longest gear or a VL "box".

THE FIRMWARE DEVELOPMENT FACILITY

This appendix contains a description of the firmware development facility available for use with the 32-bit Custom Processor. The appendix is divided into three parts:

A general overview

The menu

A description of each menu item

The Firmware Development Facility (FDF) is an equipment which makes firmware, coded and assembled under RTL, easy to checkout. The FDF consists of:

A separate five card cage with an independent power supply

A processor board with a Z80 processor (BF4RMP)

A SILO board with a 4096 location SILO (BF4TFU)

A 16,384 by 128 location control store PROM substitute (BF4CMX) including four memory array daughter boards (BS4CS4)

A terminal/keyboard unit (e.g., 7300)

Appropriate interconnecting cables

Once the firmware has been tested and "burnt" into PROMs, the FDF equipment listed above is no longer required and is never shipped to the end-user. The resulting CUP product connects to the MEGABUS System Bus through the Custom Memory Subsystem and the Custom MEGABUS Adapter unit.

Figure A-1 is a diagram of the interconnections among the elements which comprise a firmware checkout "test bed".

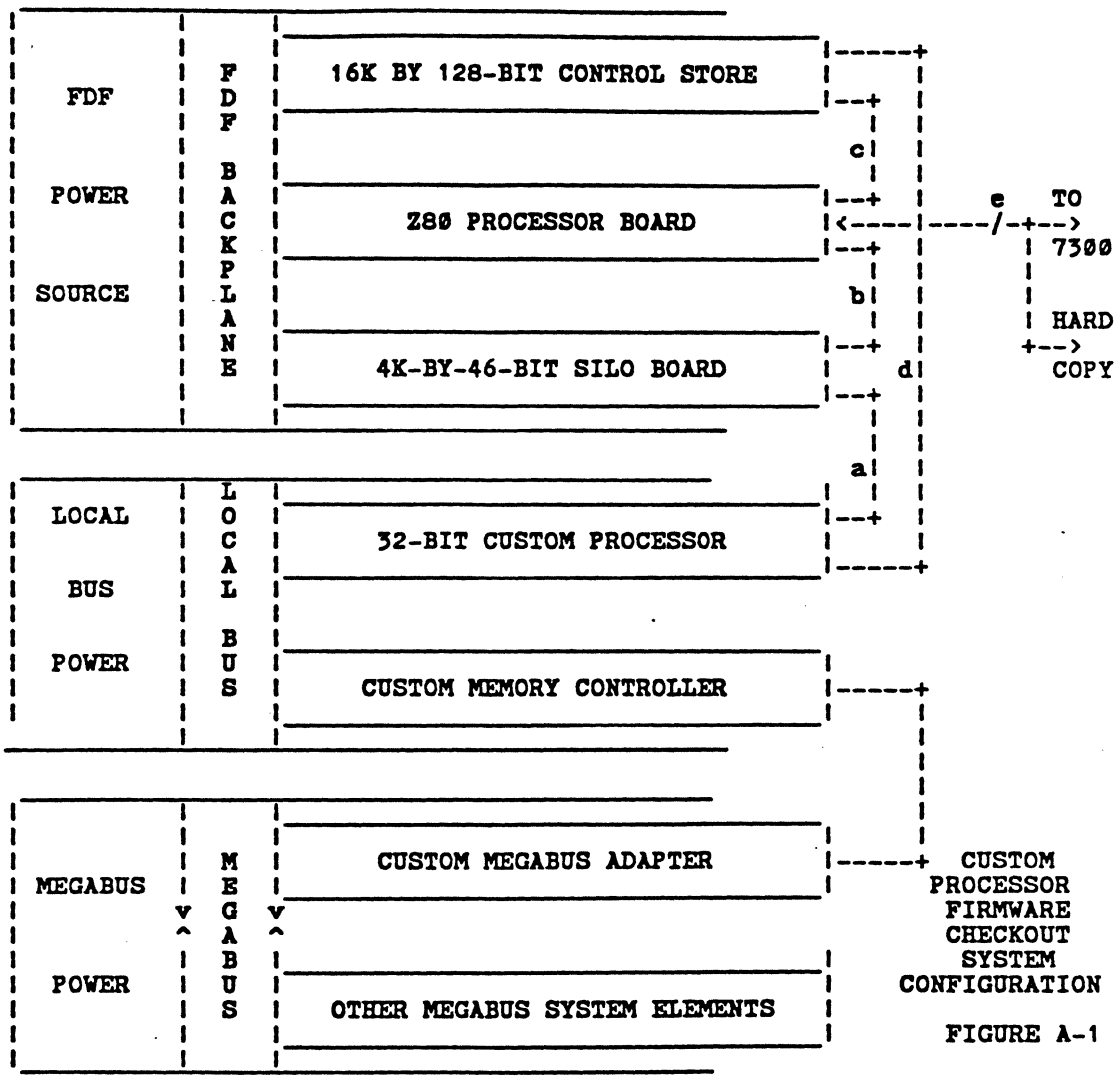


FIGURE A-1

a: Cables (2) CUP to SILO (04910202-001, 04910203-001)
b: Cable (1) SILO to Z80 (04910230-001)
c: Cable (1) Z80 to MEMORY (60128806-001)
d: Cables (4) CUP to MEMORY (04910204-001)
e: Cable (1) Z80 to TERMINAL (60156745-001) for RS232
(60156675-001) for RS422

COMMAND	DISPLAY
?	Summary of commands
-n	Silo location: -n yyyy = zzzzzzzz where n = silo offset from stop point, in decimal (0 - 4095+, default n = current value) yyyy = firmware address zzzzzzzz = content of zbus at that address
NBLK	Next 23 locations in SILO If offset of 0 is reached, "THE END" is displayed.
nNEXT	Next n locations in SILO (n=1 - 4000; default = 1) If offset of 0 is reached, "THE END" is displayed.
PBLK	Previous 23 locations in SILO If offset of 4000 is reached, "THE END" is displayed.
nPREV	Previous n locations in SILO (n=1 - 4000; default = 1) If offset of 4000 is reached, "THE END" is displayed.
nSCAxxxx	Scan SILO for a firmware address = xxxx. Start scanning at an offset of -n (default = 4000). Display each match until either: an offset of zero is reached ("THE END" is displayed) or 23 matches have occurred (hitting the space bar will display the next set of matches; hitting any other key will execute that command.)
nSCDxxxxxxxx	Scan SILO for a data = xxxxxxxx. Start scanning at an offset of -n (default = 4000). Display each match until either: an offset of zero is reached ("THE END" is displayed) or 23 matches have occurred (hitting the space bar will display the next set of matches; hitting any other key will execute that command.)
LEFT ARROW	same as PREV
RIGHT ARROW	same as NEXT
DOWN ARROW	same as NBLK
UP ARROW	same as PBLK

```
A       ADRA=xxxxxxxx  ADRB=xxxxxxxx  ADRP=xxxxxxxx  PHST=xxxxxxxx

AA      ADRA=xxxxxxxx

AB      ADRB=xxxxxxxx

AP      ADRP=xxxxxxxx

F       F0/8      F1/9      F2/A                 F7/F
        xxxxxxxx  xxxxxxxx  xxxxxxxx     ...     xxxxxxxx
        xxxxxxxx  xxxxxxxx  xxxxxxxx     ...     xxxxxxxx

Fn        F register #n where n = 0 through F (alterable by =
          followed by input data)
        Fn=xxxxxxxx

H       H register (alterable by = followed by input data)
        H=xxxxxxxx

I       Input registers and syndrome
        INRA=xxxxxxxx  INRB=xxxxxxxx  SYND=xxxxxxxx

Mn        Main memory location #n: (n = 0 through FFFFFFFF)
        Mn = xxxxxxxx
          Note: If location #n is nonexiestent or otherwise
                inaccessible, the value "xxxxxxxx" displayed
                is the current content of register OUTR.
                No error indication is given.

M.        Main memory location most recently displayed:
        Mn=xxxxxxxx

M+        Next doubleword in main memory:
        Mn=xxxxxxxx

M-        Previous doubleword in main memory:
        Mn=xxxxxxxx

OP      OP=xxxx RBR'RAR=xxxx I=bbbbbbbbbbx
          OP = OPA, OPB, OPC, OPD (1 hex digit each)
          RBR'RAR = RBR (1 digit), RARH (1 digit), RARL (2 digits)
          I = Indicators 0 through 9 (10 bits),
          Indicators 10 through 13 (1 digit)

PB        Procedure bytes (right justified)
        PB=xxxx

PH        PCTR history:
        PHST=xxxxxxxx

Q       Q register (alterable by = followed by input data)
        Q=xxxxxxxx
```

R	R0/8 R1/9 R2/A ... R7/F xxxxxxxx xxxxxxxx xxxxxxxx ... xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx ... xxxxxxxx
Rn	ARAM specified by n where n may be 000 through FFF (Alterable by = followed by input data) Rn=xxxxxxxx
R.	ARAM location most recently displayed: Rn=xxxxxxxx
R+	Next location in ARAM: Rn=xxxxxxxx
S	Stack register: S=xxxxxxxx
V	V register V=xxxxxxxx
=xxxxxxxx	Alter most-recently-displayed "alterable" register to equal xxxxxxxx. (Alterable registers are: AA, AB, AP, Fn, H, Mn, M+, M-, Q, Rn, R+, R-)
:	Define "EPILOG", a list of preselected commands which will be executed when any STOP is encountered. Format: :COMMAND,COMMAND,COMMAND,etc. No blanks allowed between a comma and the next command. 80 characters maximum. see also \,/,^, and ~ keys.
*	Display EPILOG and allow corrections. see also \, /, ^, and ~ keys.
n*	Retrieve EPILOG #n (n = 2 through 6)
n*S	Save current EPILOG as EPILOG #n (n = 2 through 6)
/	Skip next character of EPILOG
\	Skip previous character of EPILOG being corrected
^	Insert blank into EPILOG being corrected
~	Delete character of EPILOG being corrected
nGOm	Repeat previous m EPILOG commands n times (m,n=1 through 9)
B	Display all active Breakpoints
Byyyy:z	Specify Breakpoint at firmware address = yyyy z = D: Disable capturing of history in SILO z = E: Enable capturing of history in SILO z = H: Address Halt not exclusive of D or E
Byyyy	Clear Breakpoint at firmware address = yyyy
	NOTES: 1 - Address is specified by last 14 bits of yyyy. 2 - Breakpoints are armed only if command RUNB or RUNL is used.

CLER	Clear display screen (not breakpoints, addresses, etc)
E	Execute (current) EPILOG
FWRAM	CUP uses firmware in external RAM State appears on line 25
FWPROM	CUP uses firmware in PROMs (mounted on CUP boards) State appears on line 25
INIT [ctl-clear]	Generate a Master Clear in the CUP. Breakpoints are left in the state they were in.
Jxxxx	Transfer firmware control to address xxxx
nRUNB [F6]	Place CUP in RUN mode, prepared to stop after the nnn-th occurrence of a breakpoint halt. Default n = 1. The contents of the EPILOG-preselected registers will then be displayed.
RUNL [F8]	Place CUP in RUN mode, propared to stop aat the first occurrence of a breakpoint halt, or the writing of the n-th SILO entry. (default = 4000) The contents of the EPILOG preselected registers will then be displayed.
RUNN (Transmit)	Place the Cup in RUN mode, and continue in that mode until the "STOP" or "INIT" is actuated.
nSTEP (F4)	Cause CUP to execute nnn firmware steps (default n = 1).
STOP [F2]	Put the CUP in STOP mode. The contents of the EPILOG-preselected registers will then be displayed. Note: The only functions allowed when not in STOP mode are "STOP" and "INIT".
XS+	Enable the CUP to stop when the external signal fed into the FDF goes from the low state to the high state.
XS-	Enable the CUP to stop when the external signal fed into the FDF goes from the high state to the low state.
XSD	Disable external stop.
"	Transmit present FDF screen display to hard-copy printer, if attached. Line 25 status is included, but without underscores, intensity variations, blinks, etc.

COMMAND	MEANING (TO DISPLAY AND CHANGE WRITABLE FIRMWARE ARRAY)
Cxxxx.	Display location xxxx, packed format, ready to modify
Cxxxx\	Display location xxxx, by fields, with headings
Cxxxx	Display location xxxx, by fields
.	Redisplay in field format, with headings
LEFT ARROW	Move cursor to previous field, ready to modify
RIGHT ARROW	Move cursor to next field, ready to modify
.	Revert to packed format, ready to modify
UP ARROW	Move to previous location
DOWN ARROW	Move to next location
LOAD [F10]	Load firmware into RAM. After this command, actuate INITIALIZE to clear the CUP. Line 25 will display "LOAD" until "INIT" is pressed.