V-SERIES SPECIFICATIONS
DISTRIBUTION LIST
GSG - MISSION VIEJO

SPECIFICATION NO: 1993 5212 SPECIFICATION TITLE: V500 FETCH MODULE EDS

REVISION LETTER: B DATE OF ISSUE: 09/10/90

BLUE BELL, PA.

A.	GUIDOTTI	MA E3-113

MISSION VIEJO, CA.

J.	BARRETT	MS 310
W.	BATES	MS 105
F.	BEUGER	MS 380
J.	BLACKBURN	MS 275
D.	BRITTLE	MS 170
J.	DIEKMAN	MS 205
C.	DOOLING	MS 265
J.	GEORGE	MS 135
B.	GOULD	MS 275
F.	HAAS	MS 285
R.	HARRIS	MS 135
A.	HATCH	MS 150
J.	HORKY	MS 170
M.	JACOBS	MS 165
M.	KALLAL	MS 105
C.	KING	MS 225
C.	LEUTHEN	MS 170
D.	MACMILLAN	MS 285
R.	MATHEWS	MS 500
V.	MCCLELLAND	MS 165
R.	MENDOZA	MS 135
J.	MOLASH	MS 135
B.	MORRIS	MS 135
W.	MURRAY	MS 265
M.	NAKAMOTO	MS 195
M.	OGRADY	MS 380
S.	PATRICK	MS 275
D.	PATTEN	MS 135

CONTINUED - MISSION VIEJO, CA.

L.	POTTER	MS 310
J.	RUCKER	MS 285
J.	SINGLETON	MS 205
H.	SOHL	MS 165
D.	STONE	MS 285
W.	TERRY	MS 215
T.	VECCHIARELLI	MS275 (2 COPIES)
R.	WENDT	MS 225
P.	WORCESTER	MS 135
J.	YETTER	MS 165

PASADENA, CA.

C.	NUSE	
T.	STERNAL	C212
W.	TREINEN	CC 6130
J.	TRYETE	
J.	VANZWALUWENBURG	

RANCHO SANTA MARGARITA,CA.

S.	ESCOBAR	RSM115
D.	HANSON	MS 115
W.	LESH	RSM115

For changes in this distribution list, contact D. Patten, x5838, M.V. Engineering Documentation.

BURROUGHS CORPORATION
Pasadena California

INTER-OFFICE CORRESPONDENCE
Hardware Publications

Date: 6/30/86

TO: R. Norman, Mgr. - ENGINEERING RECORDS

FROM: L. Simpson - ENGINEERING OPERATIONS

SUBJECT: PRINTING AND DISTRIBUTION INSTRUCTIONS

SPEC TITLE : V500 Fetch Module

SPEC NUMBER: 1993 5212 REVISION: A

Do not print this page

Printing Instructions:

Print pages one side only	-----
Print pages back to back	X
Specification form paper	-----
Three hole punch	-----

Back to back printing format:

Right hand page	4	6	8	etc	---	---	
Reverse page (Back up)	5	7	9	etc	---	---	

Distribution:

Product spec distribution	-----
Standard SPS distribution	-----
Special distribution (See list attached)	X

File: relform

1993 5212

UNISYS

V500
Fetch Module

Engineering
Design Specification

Company Confidential

1993 5212

Burroughs Corporation

COMPUTER SYSTEMS GROUP
PASADENA PLANT

V500 FETCH MODULE

ENGINEERING DESIGN SPECIFICATION

COMPANY CONFIDENTIAL

REVISIONS

REV LTR	CONTROL DOC & NO	PAGES REVISED, ADDED, DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A	ECN 50520	Initial Issue 7-2-86	M. Ho	H. Li 6/30/86 R. Young 6/27/86 D. Welbourn 6/30/86 J. Chen 6-27-86 D. Lau
			L. Simpson	
B	8/24/90	Incorporates Ree #100591. Adds Section 4.1 V510 System. ECN #97611		M. Jacobs 8/24/90 J. Talbott 9/5/90 N. Clenshaw

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

COMPANY
CONFIDENTIAL

1993 5212

V500 FETCH MODULE

ENGINEERING DESIGN SPECIFICATION

Rev. B

TABLE OF CONTENTS

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

COMPANY
CONFIDENTIAL

1993 5212

V500 FETCH MODULE

ENGINEERING DESIGN SPECIFICATION Rev. B

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

1 PURPOSE

The purpose of this document is to describe in detail the V500 Fetch Module to provide an accurate design goal.

2 APPLICABLE DOCUMENTS

1993 5279	V500 Architecture Specification
1993 5204	V500 Execute Module Specification
1993 5220	V500 Memory Ctl and Cache Module Specification
1993 5238	V500 Memory Data Card Specification
1993 5246	V500 I/O Translator Specification
1993 5303	V500 Maintenance Subsystem Specification
1993 5295	V500 System Maint Controller Specification
xxxx xxxx	V500 Design Guidelines
xxxx xxxx	F100K Logic Rules
xxxx xxxx	MCAII Design Rules
1993 5337	Fault Detection Design Recommendations
1997 5390	V Series Instruction Set Specification

3 FETCH FUNCTIONS

This module is a part of the V500 Processor, along with the Execute and Memory Interface Modules. The Fetch Module has the following basic functions:

1. Read instruction from cache/memory at the base relative address specified by the Instruction address.
2. Parse and format the instruction into syllables.
3. Resolve operand addresses: indirection, indexing and extension.
4. Prefetch operands from cache/memory for the Execute Module.
5. Generate the OPLIT value for instructions.
6. Resolve operand indirect field length.
7. Maintains lock status for overlapped instruction fetch/execution.
8. Predict direction to be taken for branch instructions.
9. Send parsed instruction and resolved operand addresses and lengths to the Execute module.
10. Maintain the Execute Module's Fetch Page address and entry count.

4 GENERAL DESCRIPTION

The Fetch Module consists of two major submodules. These are the Instruction Fetch Module (hereafter referred to as the IF Module), and the Operand Fetch Module (hereafter referred to as the OF Module).

4.1 V510 SYSTEM

For the V510, modifications were made to the two-card OF module. These modifications affected the micro-code as well as the hardware. Micro-code restrictions were made such that index-caching and operand pre-read are no longer performed. The hardware changes made to the Operand Fetch Control card allow Data-Hit to be set constantly on every instruction transferred to the Execute Module. The combination of these changes reduces the system's performance in comparison with the V530 system.

Note that all the changes mentioned above are packaged and they may not be implemented or de-implemented seperately.

The physical implementation of these submodules is done with F100K ECL chips and MCAII gate arrays.

5 GLOSSARY OF TERMS

EQ: Error Queue, the queue associated with the IQ used to contain the flags that will be sent from IF to OF.

FIFO: First In, First Out. Describes the use of a queue structure as a buffer.

IF: Instruction Fetch.

IQ: Instruction Queue, in IF.

LUT: Look-Up-Table, made up of Opclass, Syltst, Clstst, rams. Controls the formatter portion of IF.

NEWPC: NEW PC command, 1 out of 8 ADDRBUS commands which causes the address on the ADDRBUS to be loaded into the IF's Reader and PC. Active with this command are additional lines from OF and XM to indicate the type of pipe clear that is to be performed. A NEWPC is the ADDRBUS command (4 of the 8 NEWPC commands are treated as NO-OP to the Memory Controller while the other 4 will cause the Memory Controller to flush its outstanding read queue) with a destination of '1' (Instruction Fetch) in the TAG field associated with the ADDRBUS.

NIA: Next Instruction Address. This is an address in the PCQ in IF associated with the NTI entry of the IQ which contains the address of the instruction following the NT instruction.

NTI: Not-Taken Instruction entry in the IF IQ. On a predicted branch, this is the instruction that would have been executed had the branch not been taken. It is parsed anyway by the IF since it is already in the RQ when the branch is predicted, to minimize the time penalty on a mis-prediction. The XM's NEWPC as a result of a mis-prediction will cause the NTI entry to be passed to the XM over the FBUSes while IF does an internal NEWPC to the NIA address.

OF: Operand Fetch.

PC: Program Counter, in IF.

PCQ: Program Counter Queue, in IF, used to store the PC's associated with entries in the IQ.

RIFO: Random In, First Out. Describes the method of writing data into the IF's RQ (data that is returning out of order in some seemingly random fashion) and reading it back out sequentially as a means of sorting the data for the IF Formatter.

RQ: Read Queue in the IF.

XM: eXecute Module.

6 DETAILED DESCRIPTION

6.1 INSTRUCTION FETCH MODULE

The IF module will consist of three major sub-sections, a Reader unit which will handle memory requests for instruction data, a Formatter unit which has the task of parsing the instruction data as it returns from memory, and the FBUS interface unit which co-ordinate the parsed instruction to the OF and XM.

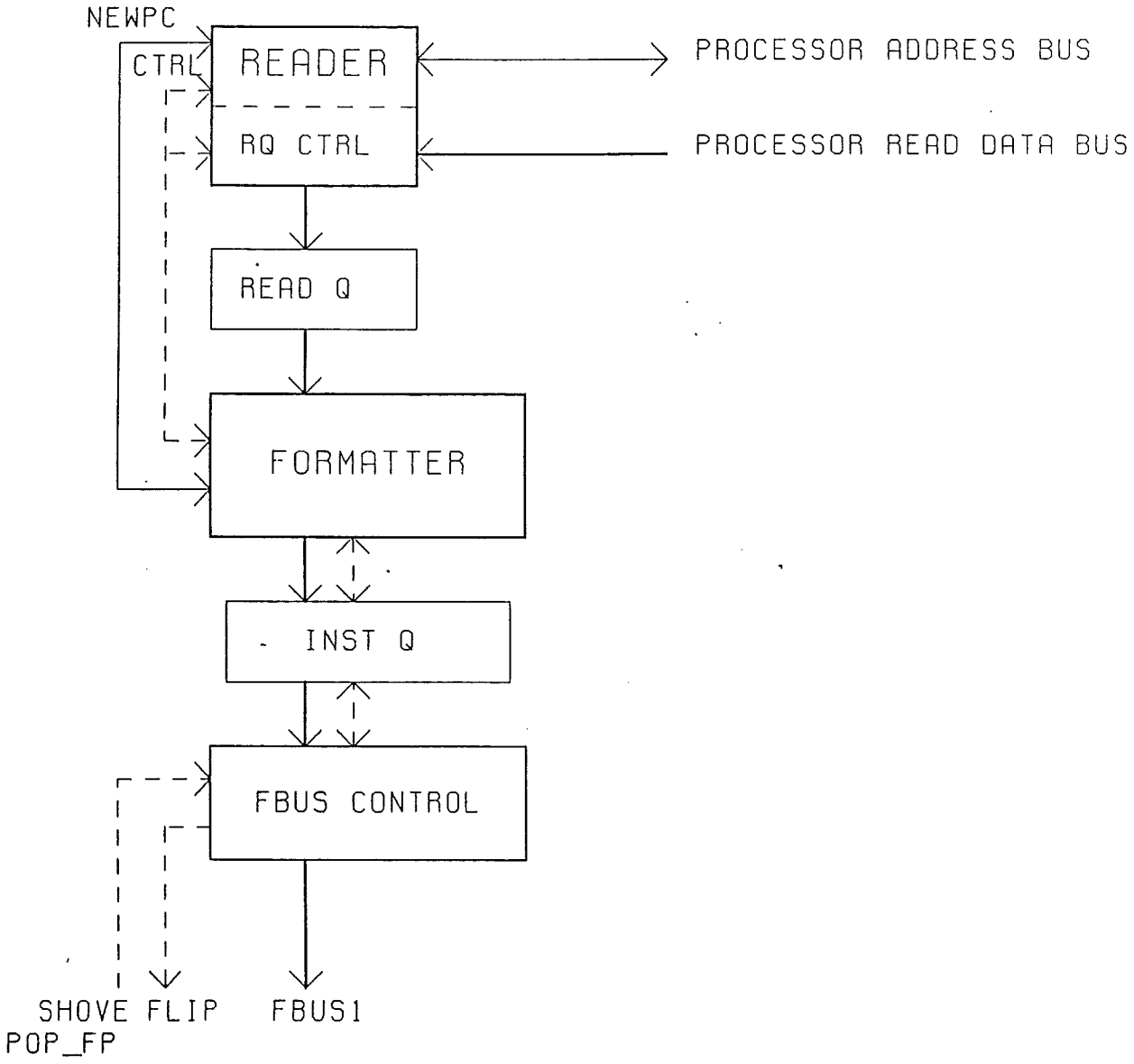


FIGURE 6-1 INSTRUCTION FETCH MODULE

6.1.1 INTERFACES

The IF module interfaces to the following modules:

1. Operand Fetch Module (OF)

   The IF module sends instruction syllables to the OF via the FBUS1 (FBUS1$$$P) and FLIP (FLIPXMFPAGE$P). IF receives NEWPC addresses via the ADDRBUS (XADRBUS$P), and the type of NEWPC command (IF flush, IX/IA done or NTI valid) via the command bus (XWBUSCMD$P) from OF. IF will also monitor the following signals from OF: SHOVE (FSHOVE$$$$$P), OF_REQ (FO$FBUS$REQP).

2. Execute Module (XM)

   The IF module sends instruction syllables to the XM via the FBUS1 (FBUS1$$$P), and receives NEWPC addresses via the ADDRBUS (XADRBUS$P), and the following signals from XM: POP_FP (EPOPFQUE$P), BRANCH_OK (EBRANCHOK$$P), and FLUSHes (system flush, bad branch taken flush, and bad branch not taken flush) via the ADDRBUS command bus (XWBUSCMD$P).

3. Memory Control and Cache Module (MCACM)

   The IF sends read requests for instruction stream data to MCACM via the ADDRBUS (XADRBUS$P) and receives the data via the RBUS (MRBUS$$$P).

4. System Maintenance Controller (SMC)

   SMC provides the human interface into the IF module.

IF interfaces with SMC, OF, XM, MCACM are as follows:

- AFMODENABLEP - fetch module clock enable from SMC.
- A$OCKECL$30P - active high free running clock from SMC.

6.1.1 INTERFACES (Continued)

- A$OCKECL$30N - active low free running clock from SMC.
- APOWERUPOK1N - active low power up clear from SMC.
- ADANGER$F01P - signal to gate with hidden state RAM write from SMC.
- ACLEAR$$F01P - synchronous clear from SMC.
- AFOENFETCHIP - IF clock enable signal from SMC.
- ASHIFTENF01P - shift enable from SMC.
- A1CARDEN$P(2:0) - card enable from SMC.
- ASHIFTINF01P - serial data in signal from SMC.
- XFMODBROKE$P - Fetch module broken to SMC.
- XFMODNBROKEP - Fetch module not broken to SMC.
- X1SHIFTOUT$P - serial data out signal to SMC.
- XSTOP$AND$$N - stop "and" signal to SMC.
- XSTOP$OR$$$P - stop "or" signal to SMC.
- FBUS1ADR$P(2:0) - Fetch page 1 entry address to OF and XM.
- FBUS1PAG$P(1:0) - Fetch page 1 page address to OF and XM.
- FBUS1LOAD$$P - FBUS1 write enable signal to OF and XM.
- FBUS1CMDPARP - FBUS1 control parity to OF and XM: generates even parity across FBUS1ADR$P, FBUS1PAG$P, FBUS1LOAD$$P, and FBUS1CMDPARP.

6.1.1 INTERFACES (Continued)

- FBUS1$$$P(39:0) - FBUS1 data bus to OF and XM.
- FBUS1PRTY40P - FBUS1 data bus parity to OF and XM: generates even parity across FBUS1$$$P(39:0) and FBUS1PRTY40P.
- FBUS1$$$P(52:40) - upper FBUS1 data bus to OF.
- FIPARXDIG$$P - FBUS1(52:48) data bus parity to OF: generates even parity across FBUS1$$$P(52:48) and FIPARXDIG$$P.
- FIPARFEDIG$P - FBUS1(47:40) data bus to OF: generates even parity across FBUS1$$$P(47:40) and FIPARFEDIG$P.
- FIPAROPAFBFP - FBUS1(23:0) data bus parity to OF: generates even parity across FBUS1$$$P(23:0) and FIPAROPAFBFP.
- FBUS10PPRTYP - FBUS1(23:16) data bus parity to OF: generates even parity across FBUS1$$$P(23:16) and FBUS10PPRTYP.
- FLIPXMFPAGEP - fetch page valid signal to OF.
- FWBUSREQOF$P - XADRBUS$P request from OF.
- FIFFLUSH$P - IF redirection signal from OF, NEWPC available on XADRBUS$P.
- FSHOVE$$$$$P - OF Fetch page available from OF.
- FO$FBUS$REQP - FBUS1 request from OF.
- FBUS2$$$$P(9:0) - FBUS2 data bus from OF for bus stop logic.

6.1.1 INTERFACES (Continued)

- FBUS2PAG$P(1:0) - FBUS2 entry address from OF for bus stop logic.
- FBUS2ADR$P(2:0) - FBUS2 page address from OF for bus stop logic.
- EFETFLUSHP - bad branch flush signal from XM, NEWPC available on XADRBUS$P.
- ESYSFLUSHP - environment change signal from XM, NEWPC available on XADRBUS$P.
- EBRANCHOK$$P - XM acknowlege corrected prediction of last predicted taken branch instruction.
- EPOPFQUE$P - XM fetch page available signal from XM.
- EWBUSREQXMDP - XADRBUS$P request from XM.
- EWBUSREQXMXP - XADRBUS$P request from co-processor.
- FWBUSREQIF$P - XADRBUS$P request to MCACM.
- XWBSRQ1D$P(2:0) - XADRBUS$P requestor ID to MCACM. XWBSRQ1D$P=5 if IF is driving XADRBUS$P.
- XWBUSTAG$P(4:0) - XADRBUS$P return tag address ID to MCACM. XWBUSTAG$P in range of "10" to "1F" if IF is driving XADRBUS$P.
- XWBUSTAGPARP - XWBUSTAG$P parity field to MCACM: generates even parity across XWBUSTAG$P and XWBUSTAGPARP.

6.1.1 INTERFACES (Continued)

- XWBUSCMD$P(4:0) - XADRBUS$P command field to MCACM. XWBUSCMD$P=1 (read data) if IF is driving XADRBUS$P. IF monitors XWBUSCMD$P field for the following values:
  8 - IX/IA done from OF. Loads XADRBUS$P as NEWPC.
  9 - IF flush from OF. Loads XADRBUS$P as NEWPC.
  A - NTI valid from OF.
  C - bad branch taken from XM. Loads XADRBUS$P as NEWPC.
  D - bad branch not taken from XM. Loads XADRBUS$P as NEWPC.
  F - environment change from XM. Loads XADRBUS$P as NEWPC.

- XWBUSLEN$P(3:0) - XADRBUS$P length field to MCACM. XWBUSLEN$P=0 (read length to 10 digits) if IF is driving XADRBUS$P.

- XWBUSCTLPARP - XWBUSCMD$P and XWBUSLEN$P parity field to MCACM: generates even parity across XWBUSCMD$P, XWBUSLEN$P, and XWBUSCTLPARP.

- XADRBUS$P(39:0) - address field to MCACM or NEWPC field from OF or XM.

- XADRBUSPRTYP - XADRBUS$P parity field to MCACM or from OF or XM; generates even parity field across XADRBUS$P and XADRBUSPRTYP.

- XWRTBUS$P(39:0) - write data bus from XM to bus stop logic.

- MQUEUEFULL$P - MCACM input queue full flag from MCACM.

- MRBUS$$$P(39:0) - read data from MCACM.

- MRBUSPARITYP - read data bus parity from MCACM: generates even parity across MRBUS$$$P and MRBUSPARITYP.

6.1.1 INTERFACES (Continued)

- MRBUSERR$P(4:0) - error vector from MCACM. IF will recognize MRBUSERR$P = 11 as limit error and MRBUSERR$$P = 12 as undigit error.

- MRBUSVALID$P - read data bus valid flag from MCACM.

- MRBUSTATPARP - partiy field across MRBUSERR$P and MRBUSVALID$P from MCACM; generates even parity across MRBUSERR$P, MRBUSVALID$P, and MRBUSTATPARP.

- MRBUSTAG$P(4:0) - read bus tag for data on following clock from MCACM. IF will only recognize MRBUSTAG$P in range of "10" to "1F".

- MRBUSTAGPARP - read bus tag parity from MCACM; generates even parity across MRBUSTAG$P and MRBUSTAGPARP.

6.1.1.1 FBUS1

The FBUS1 (FBUS1$$$P) is the data path by which the IF module transmits the parsed instruction to the XM's Fetch Queue pages (EFPAGE0 & 1). The FBus1 is used to send the PC, OPAFBF, and operand addresses to the OF and XM.

Along with the data bus are address lines (FBUS1PAG$P) which identify the Fetch Page being written to as well as the syllable (FBUS1ADR$P) on the bus, and load lines to write the syllable into the XM's Fetch Queue. Also included will be request lines (FOBUSREQP) to determine bus priority since the OF module uses the FBUS1 also.

6.1.1.2 ADDRBUS

The ADDRBUS (XADRBUS$P) is the bus used by the Reader to send the address used in its memory requests, and to receive a NEWPC address. (Please refer to the Branch Prediction/Pipe Clear section for more information on the exact mechanism of NEWPC commands).

Associated with the ADDRBUS (XADRBUS$P) are command lines to indicate the purpose of the address (memory read, NEWPC, etc.) and request lines for bus priority resolution, a length field to specify the number of digits to be read, a tag field which will direct returning read data to an address within the Read Queue, and the type of NEWPC for the Reader (example: IXIADONE NEWPC to un-freeze the Reader from waiting for the resolved indexed address).

COMMAND NAME	AWBUS COMMAND (XWBUSCMD$P)
Set IX IA DONE	8
Set IFFLUSH	9
Set NTIVALID	A
Bad Branch Taken	C
Bad Branch Not Taken	D
System FLush	F

6.1.1.3 RBUS

The RBUS (MRBUS$$$P) is the bus by which read data from the Memory Control and Cache Module (MCACM) is received into the Read Queue. This is the data requested by the Reader and which will be parsed by the Formatter into syllables. The tag field associated with the request will return one clock before the data itself, to indicate that data is coming and where in the Read Queue it should be placed. Since the tag contains the write address for the Read Queue (as specified by the Reader when the request was made), and since there is a valid bit for each entry, the IF is thus able to handle data returning from memory out of order.

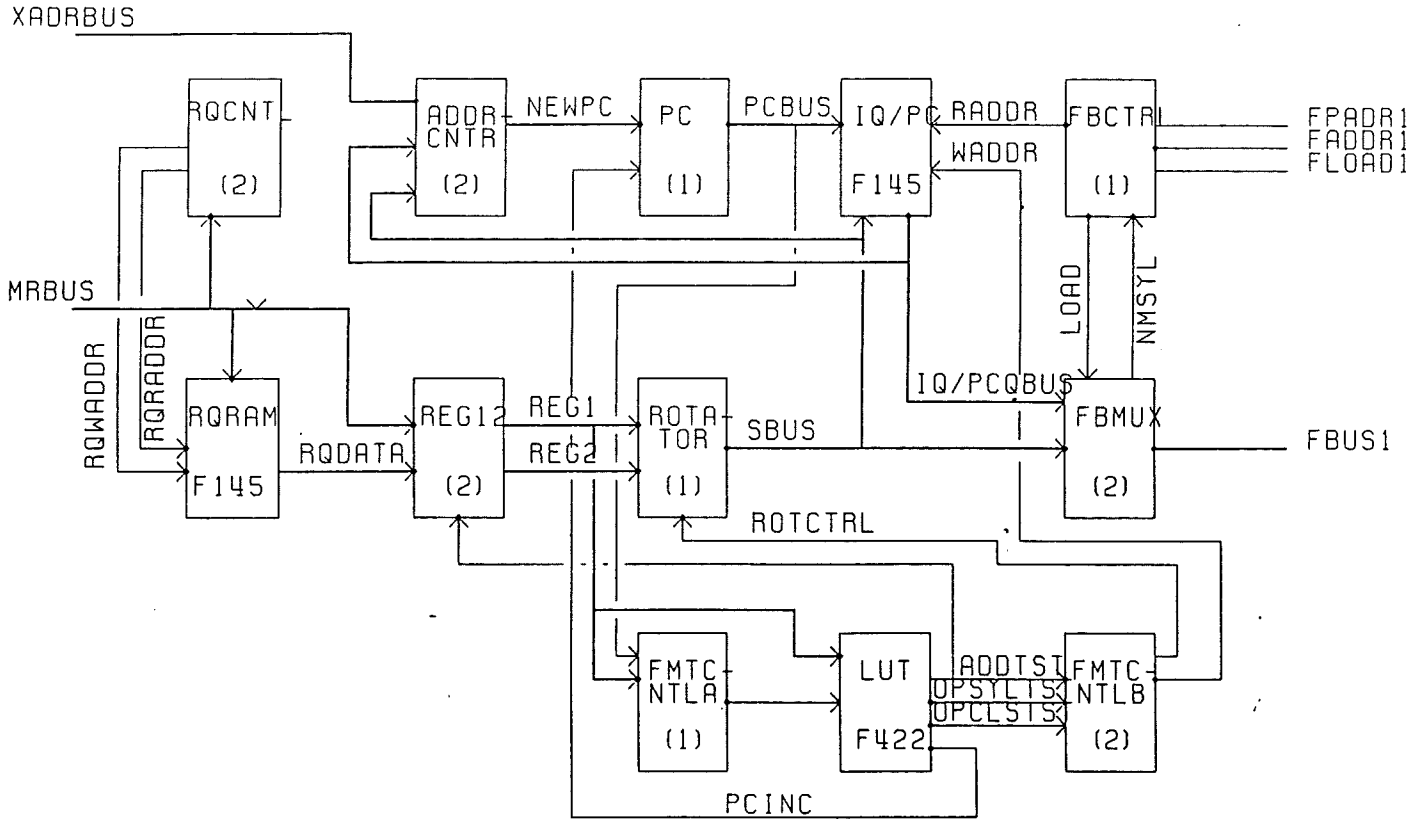


FIGURE 6-2 IF ARRAY FUNCTIONAL DIAGRAM

6.1.2 READER

The Reader's primary purpose is to supply the Formatter with instruction stream data as fast as the Formatter desires it. The Reader does this by asking the MCACM for the data, which is received by the Read Queue and passed on to the Formatter.

The Reader will be supplied with a PC from either the Formatter, the PCQ (in the case of NIA PC), the OF module or the Execute Module. The Reader will then start requesting the instruction data from memory at that address, mod 10. The Reader has to take care not to request more data than the Read Queue can handle, so it must know at all times how full the RQ is, and how many requests are outstanding. In the event that the Formatter wishes to take a branch, the Formatter will provide the Reader with a new PC, (using the internal bus 'S-BUS') causing the Reader to re-load its memory address pointers and start requesting memory from this new address. An added complication to this is the need for the Reader to avoid trashing data returning from requests prior to the Formatter's NEWPC to allow the Formatter to parse the NT instruction (if the NEWPC was due to a predicted branch taken). When the Formatter has finished with the NT instruction, it shall send a 'DONE' to tell the Reader it can start trashing.

The Reader will be closely coupled with the RQ control since the Reader needs to keep track of what is happening in the RQ. As part of this duty, the Reader provides the write address for the RQ within the tag field sent out with the request for data.

6.1.2.1 ADDRESS CONTROL

The address control portion of the Reader contains those address counters needed to maintain control of the Read Queue and to provide the appropriate address for the Reader's requests for memory data.

REQUEST ADDRESS COUNTER

This is an 8 digit decimal up counter/shift register used as an address counter. It is connected to the address bus and can be loaded from either the ADDRBUS (XADRBUS$P), the SBUS (from the Formatter), or the NIA PCQ, and can drive the ADDRBUS (XADRBUS$P). It increments by ten each time a memory request has been granted. Its least significant digit is always zero when it is driving the bus to provide a mod-ten address; however it does store the real LSD for use by the Formatter.

The MSD of the counter is the base indicant. Even if the base indicant is always one, a full 4 bit counter is implemented for flexibility. The seventh and sixth digits of the data bus are not connected to this Address Counter. The NEWPC can be loaded from the Formatter via internal data lines, or from OF or XM via the address bus (XADRBUS$P). The counter checks for undigits and address parity errors, and if an error is detected it will set an error flag and report it to maintenance. There are two types of parity checker/generator: One is for mod 10 address parity, the other is for parity across full eight digits.

MEMORY REQUEST COUNTER

This is a binary up/down counter/shift register. Its function is to keep track of the number of outstanding memory requests plus the number of unused valid entries in the RQ. If the counter counts up to four, no more memory requests will be issued until it is being decremented by popping data from the RQ to REG1 or when Trash condition occurs which will reset this counter back to zero.

6.1.2.1 ADDRESS CONTROL (Continued)

TAG COUNTER

This is a binary up counter/shift register. Its function is to point to the next available cell for memory request since the Read Pointer points to the next cell to be read by the Formatter. When the Reader is granted its memory request, it increments the counter by one if the next cell is available. If the next cell is the end of the bank, then it loads the next available bank address (upper 2 bits of this 4 bit counter) from the Next Bank Selection Logic.

READ QUEUE ENTRY COUNTER

This counter is a set of four 2-bit counters. Each counter shows its bank status. If any of bank is full, then control seeks for next available bank by looking at these 4 counters. It increments when memory request for any one of its bank cell is granted, and decrements when requested data returns. (See section on RQ for more details.)

TRASH COUNTER

Upon the NEWPC command (Not Flush), trash flag will be set (4 flags) for those bank(s) that is(are) waiting for the data to be returned from the MCACM module. When the data returns, the READER will trash it if it is to be trashed and return the busy bank for further usage when trash count gets down to zero.

READ QUEUE WRITE ADDRESS COUNTER

The Write Address Counter is a binary counter which is included in the TAG along with the Reader's request. As the requested data returns from memory, this address in the TAG field is used as the write address for the data.

6.1.2.2 ADDRBUS INTERFACE

The ADDRBUS (XADRBUS$P) interface contains the logic for making requests for memory data (placing request, resolving priority, placing command, length, TAG and address on the bus) as well as logic for receiving commands from the bus--the NEWPC commands from OF and XM.

The parity across mod ten address is ODD PARITY (If the parity bit across 40 bits of data is even, the parity generation logic will be set to make overall parity across 41 bits EVEN).

XWBUSTAG$P(6:0)	SOURCE/DESTINATION
001XXXX	INSTRUCTION FETCH

XWBUSCMD$P(4:0)	FUNCTION
01 (HEX)	READ DATA

6.1.3 FORMATTER STRUCTURE

The Formatter's main goal is to take instruction data from the Read Queue and parse it into syllables for use by the XM and OF, as fast as it can: one syllable per clock, once the pipe has been filled.

When data requested by the Reader is returned from MCACM it is loaded into the RREG1 register of the RQ, and is then passed on to the TESTER, the RREG2 register and the Rotator. The TESTER, part of the Formatter Controller, is used to determine opcode type, i.e, one, two, three or four syllables and/or branch instruction, literal, and to test the addresses for the extended flag to know where the next syllable starts. The Rotator, controlled by the Formatter Controller, rotates and aligns the syllables and places this data onto the S-bus (Syllable bus). Then the syllables will be loaded into the Instruction Queue (IQ) and the FBUS1 register, and will be sent over to the fetch pages.

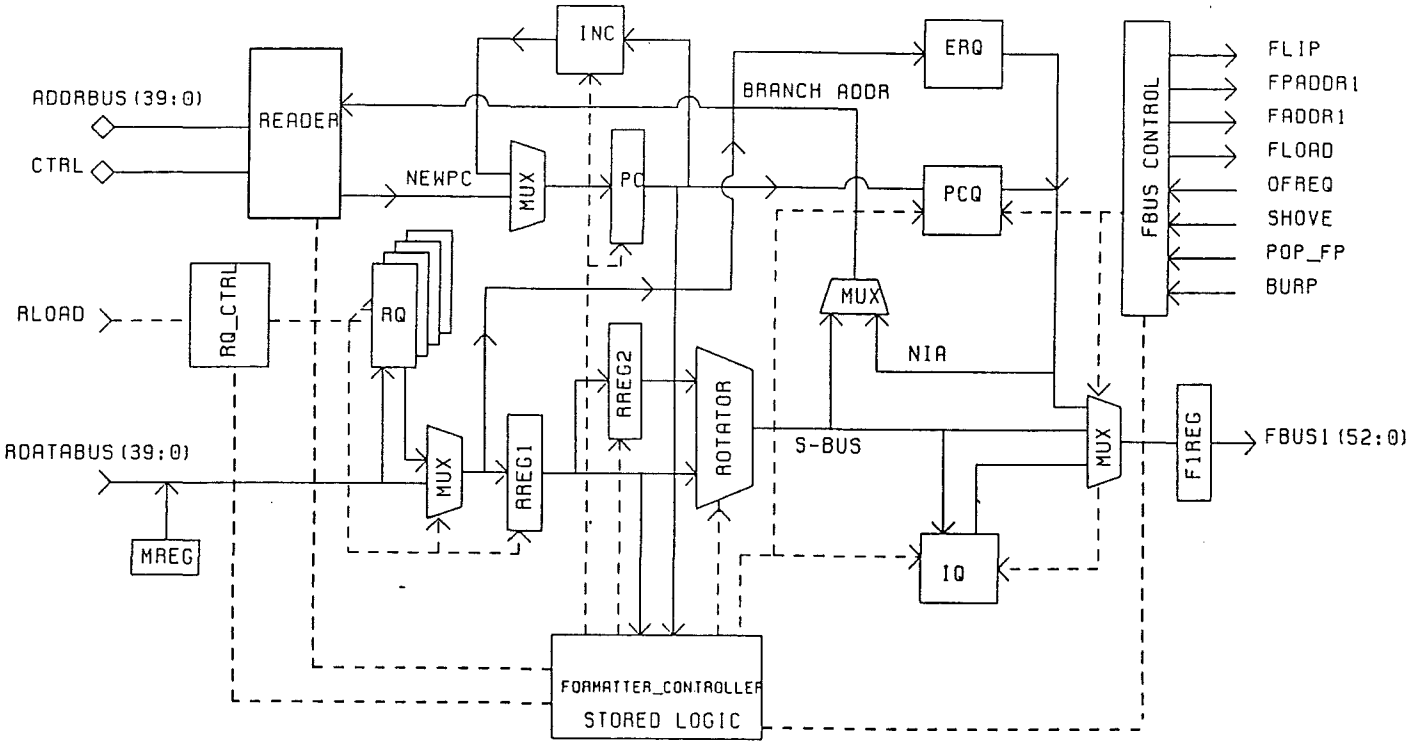


FIGURE 6-3 INSTRUCTION FETCH MODULE: FORMATTER DETAIL

6.1.3.1 READ QUEUE STRUCTURE

RBUS INTERFACE

The RBUS Interface is the IF module's source of instruction stream data. Data that has been requested by the Reader is received off the RBUS (MRBUS$$$P) into either the RQ and/or RREG1 before being passed on to the Formatter. The RQ Control provides the handshaking between the RBUS Interface and the RBUS (MRBUS$$$P), the Reader, and the Formatter. This RBUS (MRBUS$$$P) is 40 bits of data and a parity across 40 bits. The command decode logic will always check the data owner field, and load the data when ID is equal to IF.

RQ

The Reader Queue is organized as 16 deep 46 bits wide register files. This 16 deep queue is divided into 4 banks, each bank is 4 cells deep. Its purpose is to capture returning read data. Each entry in the RQ has a valid bit in a flip-flop, which is set when data is written into the RQ and reset when data is read out into RREG1 or upon SYSFLUSH, FETFLUSH, or IFFLUSH. This queue is used in a RANDOM WRITE SEQUENTIAL READ fashion, (because memory data requests can be returned out of order) to sort the data being given to the Formatter in a sequential manner. The Formatter itself sees the RQ as if it were a FIFO queue with data returning from memory in order. The five bits of error condition will be stored in the RQ until each data is parsed into syllables due to the random write sequential read fashion. The most reasonable PC associated with the error condition will be reported to the OF. If the Formatter is waiting for data from RQ, the first data coming from the MCACM will be loaded to RREG1 and the RQ simultaneously. The R1VALID bit is set when data is loaded into the REG1 and is reset after the data is popped out from the REG1.

6.1.3.1 READ QUEUE STRUCTURE (Continued)

RREG1

The RREG1 register is a ten digit wide register used as a staging register between the RBUS (MRBUS$$$P) and the Formatter. The RREG1 register is loaded from the RQ. The output of the RREG1 is also used by the Formatter LUT RAM address to determine parameters to control the Rotator in the next clock.

RQ CONTROL

The RQ Control has the task of managing the data coming back from memory. Data will be loaded into the RQ. The RQ Control has to keep track of the RQ's read and write pointers, and read out the appropriate RQ data (if valid) to load into the RREG1 register when the Formatter signals to pop RREG1. The RQ Control will inform the Reader when data returns from memory, and will inform the Formatter when RREG1 data is valid.

READ QUEUE POINTER

The Read Queue Pointer is a binary counter which provides the read address for the Read Queue. The value of the counter is changed when the Formatter requests that RREG1 be popped, after a check of the validity bit associated with the current address.

6.1.3.2 FORMATTER

The Formatter is that portion of the Instruction Fetch module that takes data from the RBUS Interface, parses it into syllables, and places it on the S-bus. It consists of the RREG2 register, a Rotator, a Program Counter (PC), and the Formatter Controller. This is the main data section of the IF.

RREG2

The RREG2 register is a ten digit register loaded with data from the RREG1 register upon a command from the Formatter Controller. Five digits of this register are contained in each of the two REG12 arrays.

ROTATOR

Based on the ROTN value (from the formatter control), the Rotator concatenates the data from RREG2 and RREG1 and outputs five bytes starting at the position pointed by the ROTN. The result of the right justified concatenation and the selected syllable definitions (see Operand Fetch section and FROTAT array specification for definitions) are sent to the Instruction Queue and the FBUS control via the Syllable bus. The selected syllable digits such as F,E,X,Y,Z are defined in the OF section.

PC

The PC, or Program Counter, contains the current BASE RELATIVE instruction address. It is loaded from data passed to it from the Reader during a NEWPC command (which the Formatter may initiate in the event of a predicted branch taken) and is incremented by the Formatter as the instructions are parsed. Each instruction's PC is loaded into the IQ along with the instruction syllables. The PC may be incremented by 2, 4, 6, or 8, and will detect odd addresses. If an odd PC is detected, the Formatter will continue parsing the instruction, then inform the OF through the SS digit in the PC syllable. Unlike previous machines, there is no need to decrement the PC by 2.

FORMATTER CONTROLLER

The Formatter Controller controls the parsing of instruction syllables by means of the Look-up Table (LUT) rams which looks at selected bytes of RREG1 data, then enables the loading of RREG2, specifies the amount of rotation required of the Rotator for alignment, and enables the loading of the IQ, based on the results of the tests. As syllables are parsed, the controller pops the RREG1 data, in effect requesting additional data. The controller also has a number of control registers which aid in the pipelining of the parsing of syllables, and counters to keep track of which syllable is currently being parsed. The Formatter Controller is comprised of the FMTCNTLA array, two FMTCNTLB arrays, and the look-up table (LUT) rams.

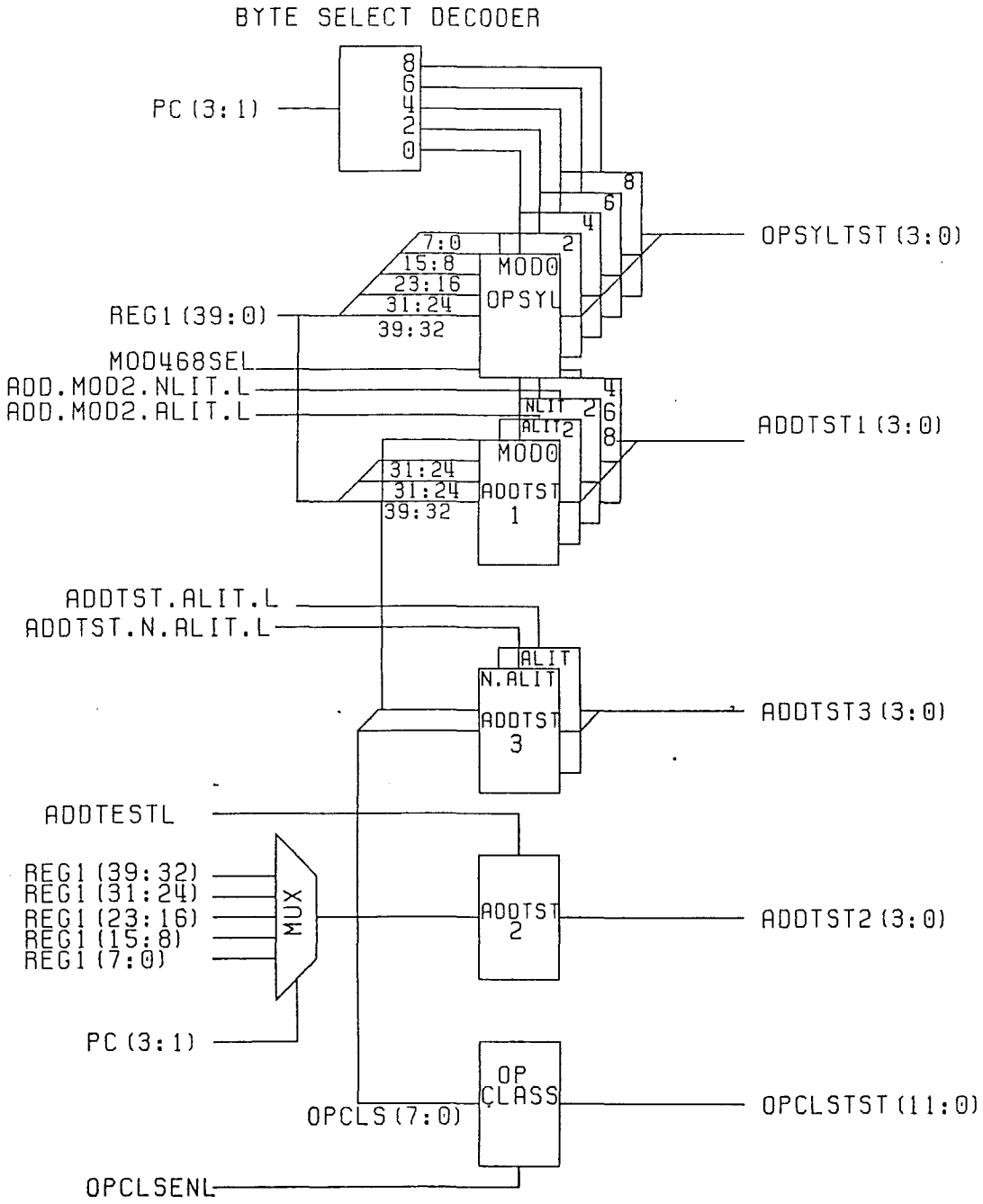


FIGURE 6-4 IF LOOK UP TABLE ORGANIZATION

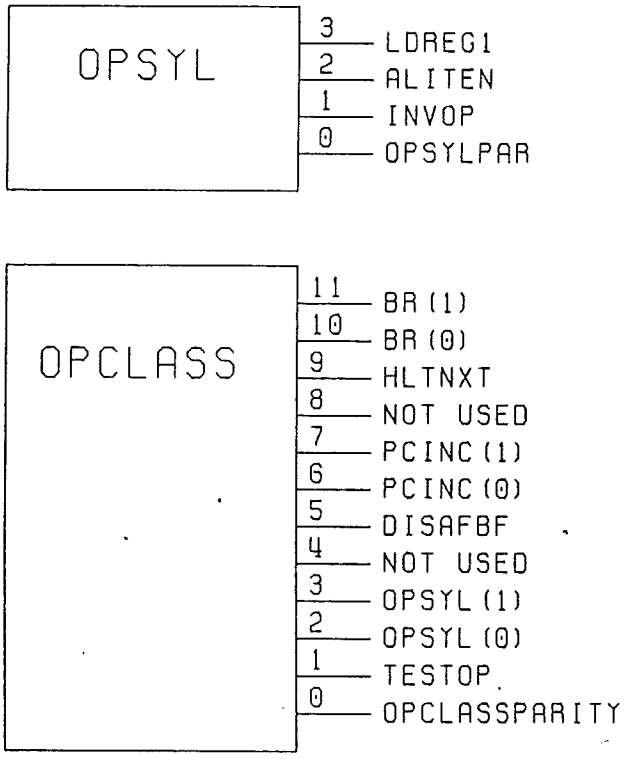


FIGURE 6-5 IF OP SYLLABLE TABLE MAP

6.1.3.2 FORMATTER (Continued)

Assignment of IF OP syllable look up RAM:

- bit 3 - 1 to load RREG1, 0 to hold
- bit 2 - enable ALIT testing for current instruction
- bit 1 - set for invalid opcode
- bit 0 - parity bit to generate even parity across OPSYL RAM output

Assignment of IF OP class look up RAM:

- bit 11:10 - branch classification of current instruction
  0 - no branch
  1 - predicted not taken branch
  2 - predicted taken branch
  3 - always taken branch
- bit 9 - halt IF after current instruction & wait for OF or XM to redirect IF
- bit 8 - not used
- bit 7:6 - PC control (increments PC by 2*bit 7:6)
- bit 5 - set for masking AFBF from opsyl to OF
- bit 4 - not used
- bit 3:2 - no. of address syllable in instruction (tag count)
- bit 1 - set to cycle IF through self check mode
- bit 0 - parity to generate even parity across OPCLASS RAM output

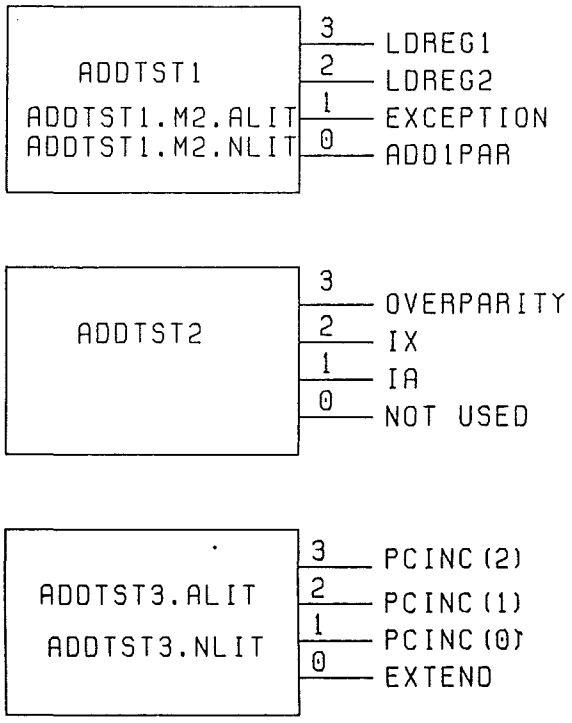


FIGURE 6-6 IF ADDRESS TEST TABLE MAP

6.1.3.2 FORMATTER (Continued)

Assignment of IF ADDTST1 RAM:

- bit 3 - 1 to load RREG1, 0 to hold RREG1
- bit 2 - 1 to load RREG2, 0 to hold RREG2
- bit 1 - set if any exception in current address syllable
- bit 0 - bit to generate even parity across ADDTST1 RAM output

Assignment of IF ADDTST2 RAM:

- bit 3 - bit to generate even parity across ADDTST2 & ADDTST3 RAM output
- bit 2 - 1 if any indexing in current address syllable
- bit 1 - 1 if any indirection in current address syllable
- bit 0 - not used

Assignment of IF ADDTST3 RAM:goto

- bit 3:1 - PC control (increments PC by 2*bit 3:1)
- bit 0 - 1 if current address syllable is extended

6.1.3.2 FORMATTER (Continued)

Tester

The Tester portion of the Formatter controller is in reality three look-up tables (LUTs). Each LUT looks at a byte of RREG1 data as selected by the mod of the PC: e.g., if the PC is mod 2, the byte selected will be RREG1(31:24). The byte selected will in turn address a RAM lookup table which will then drive the various control lines. Upon power-up of the system, this lookup table, the Instruction Fetch's "micro-code" will have to be loaded by the Maintenance processor.

The first tester, the OPSYL Tester has the important task of determining what kind of instruction the Formatter is dealing with, how many addresses it has, whether it will have to predict a branch, etc. In the interest of conserving board space and chips, it is divided into two sections, one which drives control lines based on the instruction and the mod of the PC, and the other which simply looks at the OPSYL to catagorize the instruction for the purpose of loading the syllable counter, branch flags, etc.

The second tester, the ALIT Tester is controlled by the OPSYL Tester and is used to test the digit following the OPSYL for the purpose of setting the ALIT flag flip/flop.

The third and final tester is the ADR Test, which tests the A-, B- and C-SYLs for the extension flag, and IX and IA and exception condition. The extension flag determines where the formatter has to look for the following syllable. The extension flag of the ASYL is ignored if the ALIT FF is set. The exception bit is defined as follow: Indirect addressing, indexing or extension digit of A,B,E,F (address error conditions).

6.1.3.2 FORMATTER (Continued)

A syllable counter loaded by the OPSYL Tester enables the ADR Test and when it has reached its maximum value for that OP class, the formatter will wait for one clock and then enable the OPSYL Tester is enabled. For zero address OP, the formatter will wait for 2 clock before the OPSYL Tester is re-enabled.

Control Registers

The Control Registers of the Formatter Controller are used to pipeline the control signals sourced by the Tester, and to latch the OP class information to remember what kind of OP and how many syllables it has throughout the parsing process. In addition, there is a syllable counter which contains the maximum number of syllables for the OP and counts down as each syllable is loaded onto the S-bus. When the syllable counter reaches zero, IF knows it is time to start the next OP. The Write Address counter provides the write address for the syllables going into the IQ: OP, A, B, and C syllables go into addresses 0, 1, 2, and 3 respectively. The Write Address counter is set to 0 whenever an OP is started and counts up as each syllable is loaded onto the S-Bus. The PC of an instruction is stored separate from the IQ in the PCQ to simpify the syllable counter mechanism.

6.1.3.3 INSTRUCTION QUEUE

The Instruction Queue is used to store the NTI (Not Taken Instruction). It also stores each instruction as it is being transferred to the FREG (FBUS1DATL) or maintenance purposes. The Instruction Queue contains the IQ itself, and the IQ Controller which handles the IQ writes by the Formatter and the IQ reads by the FBUS Control.

The IQ is sixteen deep, divided into four sections dual port ram. Each section contains ten digit wide instruction OP, A, B, and C syllables. One section captures the instruction syllables on their way to the FBUS1 (FBUS1$$$P) (maintenance purposes), two are in reserve for future need, and the "NOT TAKEN" entry. This "NOT-TAKEN" entry is used to hold the parsed instruction that followed a branch instruction that was predicted TAKEN. In the event the prediction was incorrect, a NEWPC command from the XM will cause this instruction (the one that actually should have been used) to be sent to the XM, while the Formatter refetches data from the address pointed by the Next-Instruction-Address (NIA) corresponding to the "NOT TAKEN" entry in the PCQ.

The Instruction Queue is partioned as follows:

```
0000 - Normal Opsyl
0001 -   "      Asyl
0010 -   "      Bsyl
0011 -   "      Csyl
0100 - NTI      Opsyl
0101 -   "      Asyl
0110 -   "      Bsyl
0111 -   "      Csyl
1000 - Not used
1001 -   "
1010 -   "
1011 -   "
1100 - Not used
1101 -   "
1110 -   "
1111 -   "
```

6.1.3.3 INSTRUCTION QUEUE (Continued)

IQ CONTROLLER

The IQ Controller is contained in the Fbus Control, it controls the reading of the IQ and the loading of FBUS1REG. If the FBUS1REG is valid then the OFREQVALID (ZOFREQVLD) signal will be active enabling the Formatter to freeze if OFREQ is active. This enables the formatter to halt the flow in the pipe when OF is writing on FBUS1 (FBUS1$$$P). The IFFREEZE signal from the Fbus Control is active (freezing the pipe) when a Fetch page is unavailable to be written or the NTI is being popped from the IQ.

In the event of a branch mis-prediction ('BBT' IF should have fallen through instead of branching) the IQ controller will pass the NT entry on to the FBUS1 (FBUS1$$$P) while sending the NIA to the Formatter to be used in a NEWPC instruction. This gives the XM a head start on the NT instruction while the Reader is busy requesting the next instruction. For the case of DUALPT (where the NTI is also a predicted taken branch instruction), the NT entry is treated as invalid and no NTI is passed on to the FBUS1 (FBUS1$$$P). The RQ control will request data starting at the NTI PC. For the case of second PT (where the predicted taken branch instruction is also a predicted taken instruction), the formatter will freeze itself until the branch OK or bad branch taken comes (is received from XM).

6.1.3.3 INSTRUCTION QUEUE (Continued)

EXCEPTION CONDITION GENERATION

Three bits of exception conditions are generated on the data being sent to the OF. These three bits are placed in the least significant bit locations of the 'X' digit when the OPSYL is being sent to the OF and XM (refer to the FBUS1 (FBUS1$$$P) format of the OF spec). The following is a detailed explanation of these bits:

ALIT	- FBUS1 (FBUS1$$$P) (50:50) set if AF is literal
AF exception	- FBUS1 (FBUS1$$$P) (49:49) set if AF is indirect ALIT error, invalid infl, invalid field length, or AF contains an undigit
BF exception	- FBUS1 (FBUS1$$$P) (48:48) set if BF is indirect, invalid infl, literal specification, invalid field length, or BF contains an undigit

FBUSREG1

The FBUSREG1 register is a staging register between the Formatter/IQ and the FBUS1 (FBUS1$$$P) drivers. It is 53 bits wide, an its loading is controlled by the Fbus controller directly.

6.1.3.4 PCQ

The PCQ is a read/write ram acting as a companion to the IQ for the purpose of storing the instruction PC. There is a PC entry corresponding to each IQ entry, plus the NIA and NTI.

The PC Queue is partioned as follows:

```
0000 - Normal PC
0001 - \
0010 -  |- not used
0011 - /
0100 - NTI PC
0101 - \
0110 -  |- not used
0111 - /
1000 - NIAPC
1001 - \
1010 -  \
1011 -   \
1100 -    |- not used
1101 -   /
1110 -  /
1111 - /
```

6.1.3.5 ERROR QUEUE

The Error Queue has two entries, one for the normal instruction and one for the Not Taken Instruction. The entry is written during the last syllable, or just prior to the PC being written to the PCQ. The error queue is 6 bits wide with 2 bits as spares.

```
bit 7 - spare
bit 6 - spare
bit 5 - Fetch Event flag
bit 4 - Fetch Event flag
bit 3 - Even parity over bits 2,1,0
bit 2 - Odd PC error
bit 1 - IF undigit error
bit 0 - IF limit error
```

The Fetch Event flag is set by the SI flag located in the FMTCTA array. The IF limit error is base/limit error.

6.1.3.6 FBUS INTERFACE

The FBUS Interface is the logic that controls the passing of data from the IF to the OF and XM, syllable by syllable.

FBUS1 (FBUS1$$$P)

The FBUS1 (FBUS1$$$P) is the Fetch bus used to transmit instruction syllable data to the OF and XM. It is 53 bits wide, and has three bit address bus FBUS1ADR$P (2:0), a two bit page address bus FBUS2PAG$P (1:0), the (Fetch page LOAD FBUS1LOAD$$P) control line, a 5 bit parity field on data, a 1 bit parity field (FBUS1CMDPARP) on page and address control, and the (Operand Fetch FOBUSREQP) line associated with it. The detailed assignment of the 4 bit parity field are as follows:

FIPARXDIG (FIPARXDIG$$P)	-	even parity generated across FBUS1$$$P (52:48)
FIPARFEDIG (FIPARFEDIG$P)	-	even parity generated across FBUS1$$$P (47:40)
FIPAROPAFBF (FIPAROPAFBFP)	-	even parity generated across FBUS1$$$P (23:0)
FIPARFORTY (FBUS1PRTY40P)	-	even parity generated across FBUS1$$$P (39:0)
FIPAROP (FBUS1OPPRTYP)	-	even parity generated across FBUS1$$$P (23:16)
FCPARITY1 (FBUS1CMDPARP)	-	even parity generated across FPADDR1(1:0) (FBUS1PAG$P), FADDR1(2:0) (FBUS1ADR$P), and FLOAD1 (FBUS1LOAD$$P)

6.1.3.6 FBUS INTERFACE (Continued)

FBUS CONTROL

The FBUS Control co-ordinates the sending of instruction data from the formatter, the PC from the PCQ and the NOT TAKEN instruction from the IQ. The control will then place the data onto the FBUS1 (FBUS1$$P), sending with it the appropriate address on the syllable address and page address (FBUS1PAG$P) lines and the FLOAD1 (FBUS1LOAD$$P) signal to write it into the OF's and XM's Fetch Queues. A tag digit included in the OPSYL (the first syllable to be read) indicates the number of syllables to be sent on the FBUS for that instruction. As the last syllable (the PC) is sent, the FBUS Control activates the FLIP signal (FLIPXMFPAGEP), which causes the Fetch Page Address (FBUS1PAG$P) to be incremented if the next page is available and informs OF that IF has finished with that instruction. As the PCQ is read it is concatenated with the ERRORQ. The FBUS Control then can start to send the next instruction's data, if the following conditions are met:

1. XM Fetch Page is available
2. OF Fetch Page is available
3. OF is not requesting the bus (FO$FBUS$REQP)

The Fbus Control will count the number of SHOVEs (FSHOVE$$$$$P) from the OF and POP_FPs (E$P0P$FQUEP) from the XM to keep track if a page is available. The IF can not get more than one instruction ahead of the OF in the XM Fetch page, this is due to the availability of the OF Fetch Page. Note: The first POP received after a SYSFLUSH or MRESET will not be counted, this is due to XM initialization of page pointers. (Refer to the FBCTRL array specification for more details on page availability.)

6.1.3.6 FBUS INTERFACE (Continued)

The Fetch Page Address lines (FBUS1PAG$P) provide the address to select which of four XM Fetch Queue pages to write to. This page address is controlled by logic containing three counters; the IF pointer incremented by a combination of the XM, OF, IF, SHOVE (FSHOVE$$$$$P), and POP (EPOPFQUEP) the OF pointer incremented by OF's SHOVE (FSHOVE$$$$$P), and the XM pointer incremented by XM's POPFP (EPOPFQUEP) which together inform the FBUS Control of the number of active pages in the XM Fetch Queue and the OF Fetch Queue. The IF pointer will remain on the page until the increment conditions are met allowing IFFLUSH to re-write the same page. The FBUS Control freezes itself and the formatter via the IFFREEZE signal if there are no FETCH pages available or during the NTI pop cycle. If FBUS1 (FBUS1$$$P) is in use by OF (FO$FBUS$REQP) the FBUS Control will freeze itself, but the formatter will only freeze itself if OFREQVLD from FBCTRL is valid.

Since the OF has to send its revised addresses and operand lengths lengths to the XM over the FBUS1 (FBUS1$$$P), it does so by activating the OFREQ (FOBUSREQP) line which gives it priority over the IF module (causing the FBUS CONTROL to freeze if it was in the process of sending data too).

The Fbus Control recognizes two types of flushes, FLUSHGEN - general flush such as system flush, and FLUSHSEL - selective flush such as IFFLUSH. The FLUSHGEN resets all the page pointers and entry pointers. Flushsel resets only the entry pointers. The NTI FULL flip flop will be reset on either flush. If a flush occurs on the backplane one clock prior to FLIP (FLIPXMFPAGEP), the FLIP (FLIPXMFPAGEP) will be sent but ignored by the OF and XM.

RESTRICTION: Since IF FLUSH resets the INC_IF flip flop and SHOVE sets this flop if the PAGE_AVAL flop is set, the Operand Fetch unit is restricted not to send IF FLUSH followed by a SHOVE (FSHOVE$$$$$P). There must be at least one clock delay after the FLUSH before sending the SHOVE (FSHOVE$$$$$P).

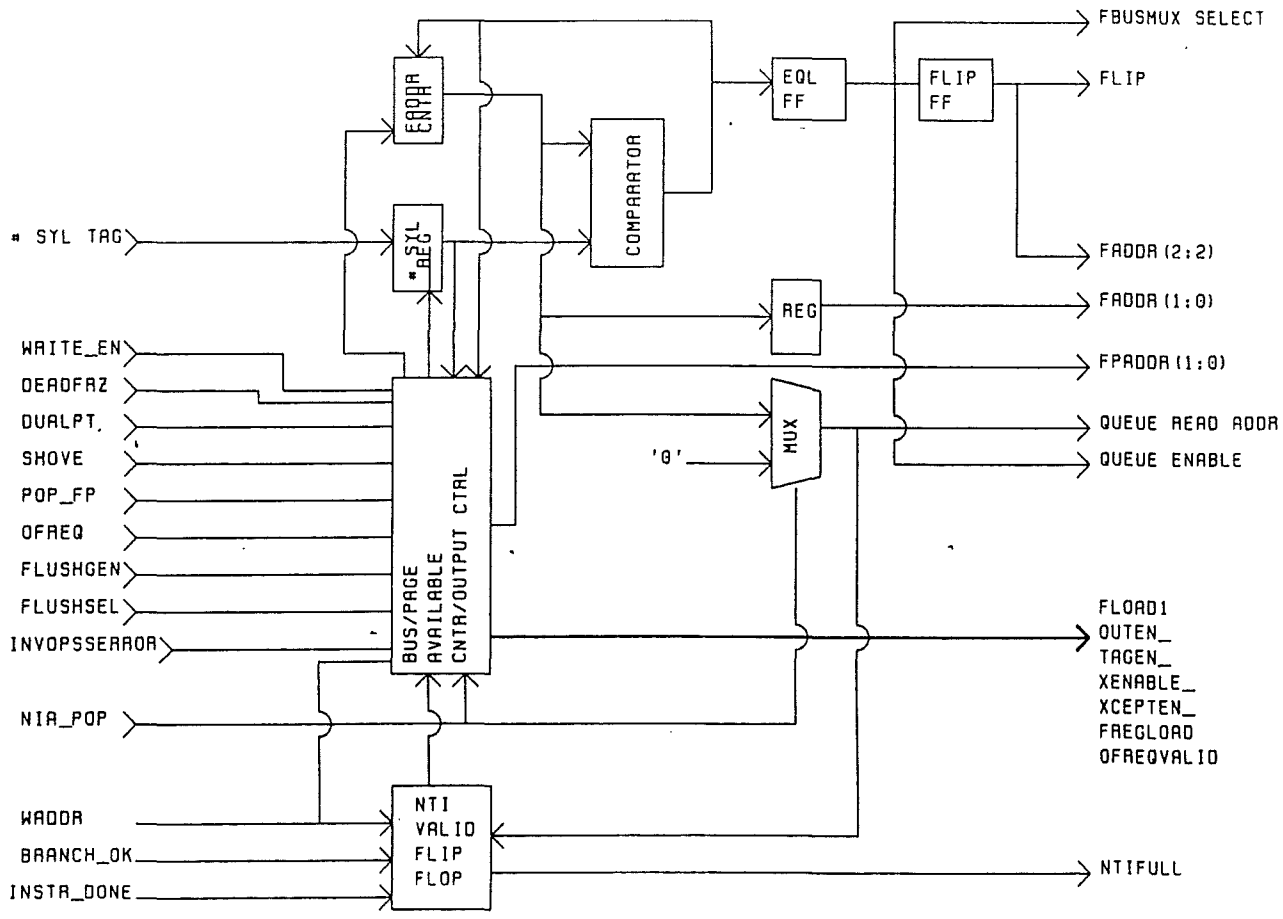


FIGURE 6-7 FBUS CONTROL BLOCK DIAGRAM

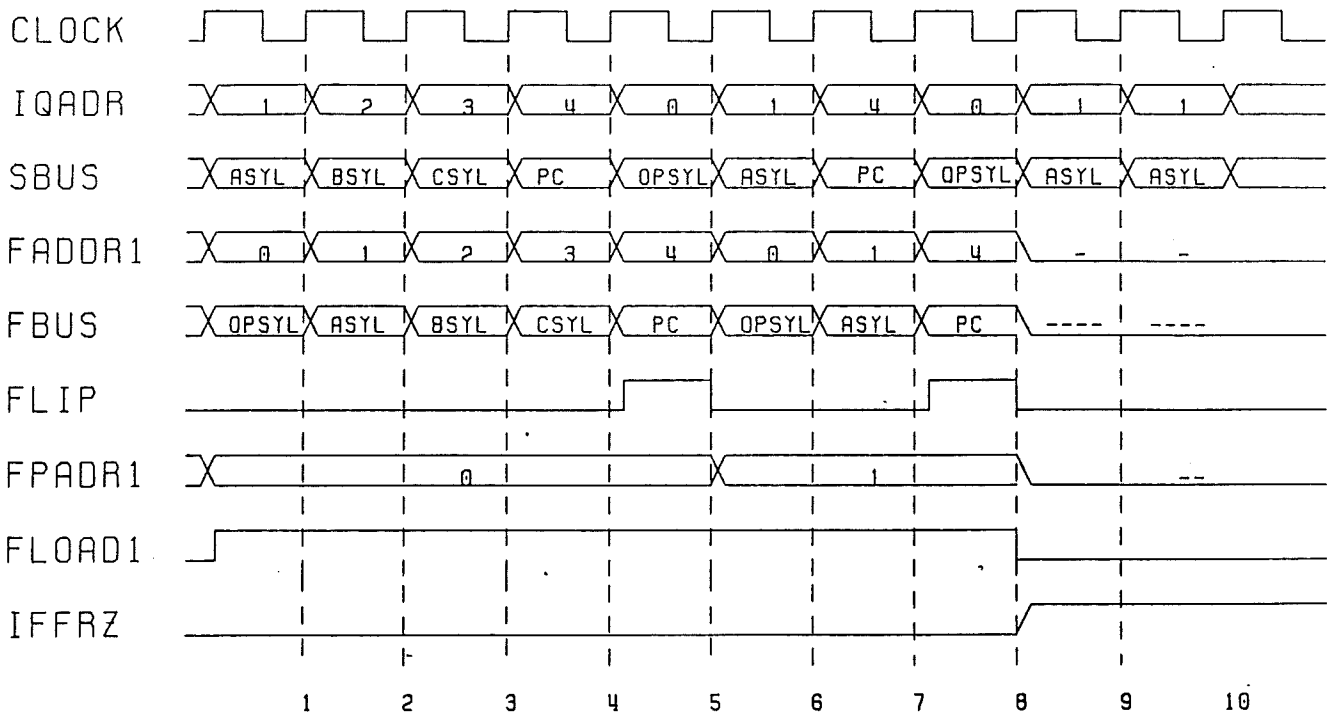


FIGURE 6-8 FBUS TIMING DIAGRAM

The Fbus Control has four different types of instruction lengths. These are controlled by the NUMSYL tag from the FBMUX.

ZERO - OPSYL, NULL, and PC - default for INVOPSS (invalid OP)
ONE - OPSYL, ASYL, and PC
TWO - OPSYL, ASYL, BSYL, and PC
THREE- OPSYL, ASYL, BSYL, CSYL, and PC

Instruction type three and one are displayed in figure 6-5. At time 8 in the Fbus Control is in the frozen state due to the OF pages being full.

6.1.3.6 FBUS INTERFACE (Continued)

Function

The IF has five functional blocks; Reader, PC, Rotator, Formatter and FBUS Control. The Reader supplies the Formatter with instruction stream data as fast as the Formatter desires it. The stream of data (40 bits) is loaded to RREG1 register when the Formatter requests it. Upon loading of the data, the valid bit for RREG1 will be set. This 40 bits of RREG1 data will be the address of the Formatter LUT Ram to determine parameters to control the Rotator. These parameters are indexing, indirection, extension, maximum no. of syllable, PC increment, Literal, type of branch, etc. If RREG1 is empty and the Formatter is ready to parse the next instruction, the first requested data from the memory will be loaded to RREG1 and the Read Queue. Then the Formatter Control sends signals to load RREG2 from RREG1 at appropriate time to parse the data stream. The RREG2 will load the data from the RREG1 every clock except when the last address syllable and next OP is in the RREG2 at the same time. In other words, raw data will be parsed into syllables every clock until the OF Fetch Queue is full or another freeze condition is met. The information from the Formatter Control selects the 5 to 1 multiplexor (Rotator) to align the data. The formatted instruction (OPAFBF, A syllable, B syllable and C syllable) is sent to the IQ and the Fbusmux each clock.

After each syllable of the instruction is formatted it is staged in the FBUS1REG (FBMUX), along with being stored in the instruction queue. The FBUS Control handles sending of the data to the OF page and the FETCH pages of the OF and XM respectively. The FBUS control only sends the formatted instruction when the OF is not requesting the FBUS1 (FBUS1$$$P) and there are available pages in both the OF and the XM. If the FBUS1REG is full and there are no FETCH pages available, the FBUS Control signals the Formatter Control to live-freeze (IFFREEZE).

6.1.3.6 FBUS INTERFACE (Continued)

Branch Prediction: Simple branch prediction is done in the Formatter. If predicted branch is taken, IF will continue to process until NIA and Not-taken information is successfully stored in queues. The Reader will load NEWPC address from the SBUS upon parsing predicted branch taken, but it will not trash any requested data from memory until IF finishes formatting the Not-taken instruction. When it detects Indexed or/and indirect branch, the formatted instruction will be passed along unresolved. Meanwhile the IF will be in a live-freeze state until NEWPC command is issued by the OF. The IF will monitor the command bus where upon receiving any of the NEWPC commands, the FBUS Control and the Formatter control will be notified. The Next Instruction Address stored in the PC queue is now sent out on the FBUS (FBUS1$$$P). The XM can invalidate the Not-taken instruction entry in the queue thru the use of the BRANCH_OK (EBRANCHOK$$P) signal.

Stop Logic: The Instruction Fetch card also contains stop logic for stopping the system upon comparing data on the following busses: ADDRESS BUS (XADRBUS$P), WRITE BUS (XWRTBUS$P), READ BUS (MRBUS$$$P), FBUS1 (FBUS1$$$P), FBUS2 (FBUS2$$$$P), and PC bus internal to IF. Write bus requests and Write bus command bits are also available to be stopped on. The Read bus stop is pipelined with the Read tag which will occur on the backplane one clock prior to the Read data. The system will stop either on the 'or' or the 'and' of the compares from each data field.

6.2 OPERAND FETCH MODULE

6.2.1 FUNCTIONAL OVERVIEW

The main functions performed by the OF module are operand prefetch, address resolution and write lock manipulation. The OF will preread length modulo 10 digits of each A and/or B read operand for the XM. Each preread operand will have its destination address field tagged for return to the operand queue of an appropriate fetch page. A data-hit bit will also be sent to warn the XM that the prefetched operands are not valid. Since the IF module only handles extended address, OF has the responsibility to resolve indirect field length, indexing and indirection. All pertinent information affected by such activities e.g. ALEN, OPSYL etc., will be updated by the OF. The OF will also post write locks for all write operands and determine whether a code modification against the current instruction has occurred. In the event of a branch op with indexed/indirect address syllable, the OF will redirect the IF module.

6.2.1.1 THEORY OF OPERATION

The OF is the second stage of the FETCH pipeline (2 stages). The IF module prefetches instructions through the Reader and parses them before sending to OF and XM. Information regarding each instruction is sent by IF on FBUS1 (FBUS1$$$P) and captured by both OF (in OFPAGE) and XM (in FPAGE). When IF finishes parsing an instruction, it sets FLIP (ONFLIPXMFPAGEP). Before IF FLIPs (ONFLIPXMFPAGEP), OF has already started to examine all the information that is received through the use of a OP-LOOKUP structure. Using the information, the OP-LOOKUP structure will provide a micro-address corresponding to the OPCODE received. If there is no instruction ready in OFPAGE, OF will wait in a loop until one is available (IF flips (ONFLIPXMFPAGEP).

The major functions of the OF is to ensure code integrity, resolve indexing and indirection and to prefetch operands for XM. OF will request a code check against the area of current instruction's PC plus 90 to ensure that there is no code modification. (More details on the section of the code stream checked by the code check command are available in a later section) Also, OF will resolve any indirect field length and/or indexed, indirect address syllables. As the instruction is being parsed, OF will send out OPLIT, FLAGS, ALEN, BLEN, CLEN (if required) to the XM through FBUS2 (FBUS2$$$$P). If the instruction performs any writes, write lock requests will be sent to the lock unit together with the begin and end addresses. Finally, OF will preread any read operands required by the XM (up to 10 digits). In order to inform XM of the validity of the preread operands, a data-hit bit will also be sent. OF finishes the instruction by setting SHOVE (FSHOVE$$$$$P) and sending it to the XM.

Due to the optimization of heavy indexing, OF may resolve BSYL before ASYL if both syllables happen to have an exception. This means that BSYL error will be reported first even if ASYL is also in error.

6.2.1.2 ASSUMPTIONS

The following is a list of assumptions adopted by the OF module :

1. All addresses within the OF are base relative.

2. IF will perform branch prediction by loading the branch address as the new program counter unless the branch address is indexed, indirect or both, in which case, OF will send out the NEWPC after resolving the address.

3. IF can only be one instruction ahead since OF has two OFPAGEs. This also helps to keep code modification checking manageable. There are four pages in the fetch queue in the XM for IF and OF to put data into.

4. OF will preread length modulo 10 digits of each read operand for the XM. The preread data will be sent to the operand queue associated with the fetch page OF is currently using.

5. A data-hit condition is sent along with the SHOVE signal to allow the XM to determine if any of the preread data has been modified. Optimal condition is also sent along on a wire to XM at the time of SHOVE (FSHOVE$$$$$P). The optimal bit will be set according to rules specified by the XM group. In general, optimal will be set if both A and B operands are less than or equal to 10 digits in length. Data-hit (FDATAHIT$$$P), SHOVE (FSHOVE$$$$P) and Optimal (FOPTIMAL$$$P) will be directed to the XM Fetch page being pointed to by the FBUS2PAG$P2 (1:0) address.

6. XM will be responsible for detecting all single bit errors and report them accordingly. Any multi-bit memory errors originated from FETCH will cause XM to assert FETCH-ERROR thereby halting FETCH immediately.

7. OF will mark the fetch page during SI and stop on OP or PC

8. During self check opcode (FE), XM and OF should be synchronized to avoid any flushes while the OF is performing self check. The synchronization is accomplished via the use of XM to OF and OF to XM signals. The protocol calls for XM to notify OF using XM to OF event wire and wait. OF will start the self check opcode when it receives the XM to OF event. Upon completion, OF will then notify XM via the OF to XM event wire so that XM can start.

6.2.1.3 INTERFACES

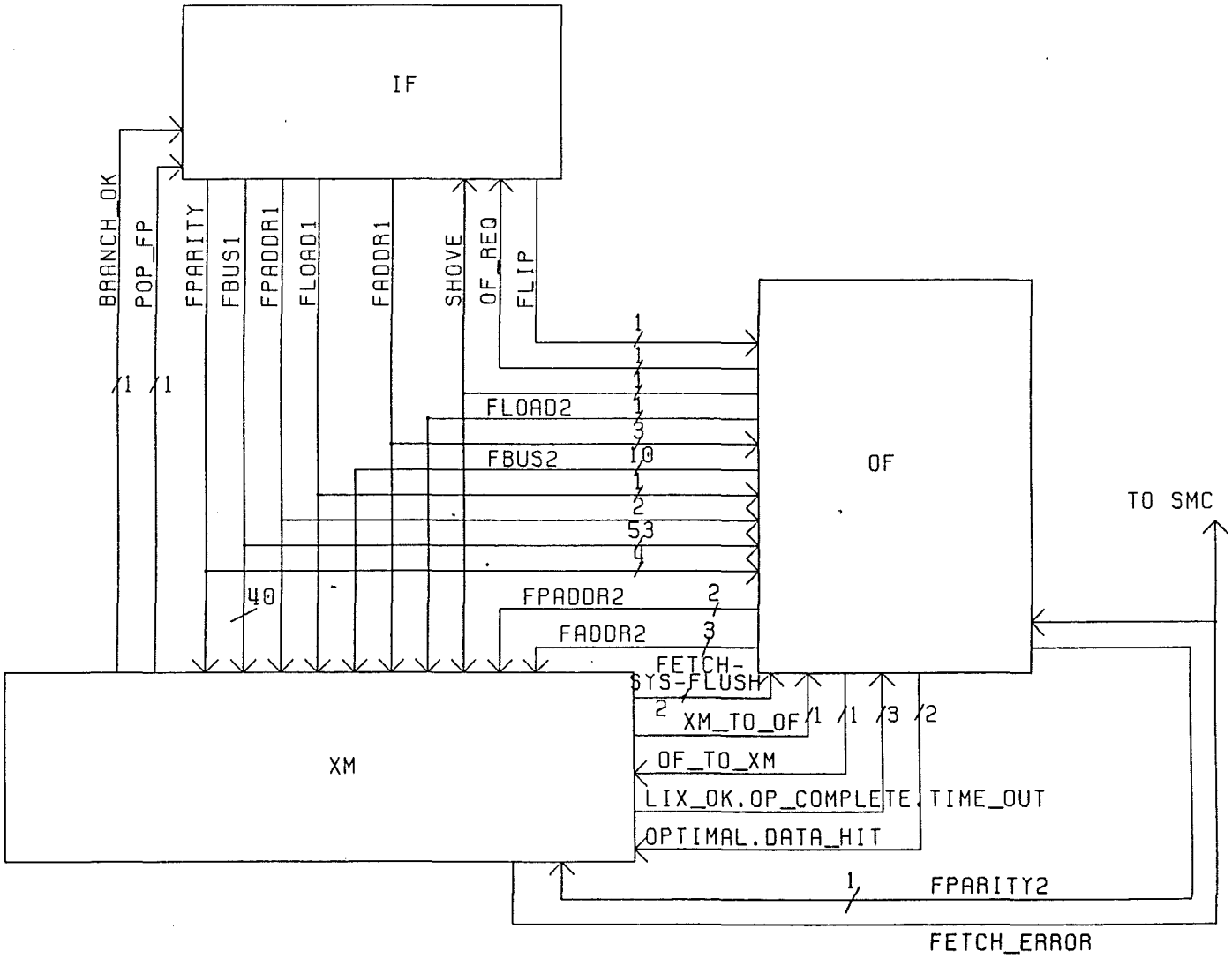


FIGURE 6-9 IF/OF/XM INTERCONNECTIONS

6.2.1.3 INTERFACES (Continued)

The OF interfaces to the following modules :

1. IF Module

FBUS1 (FBUS1$$$P) serves as the primary interface between the IF and the OF. Fetch page data sent to the XM via FBUS1 (FBUS1$$$P) will be saved by the OF in an appropriate OFPAGE for later use. Fetch page data from IF will appear in the following order (no. of address syllable sent depends on the particular instruction):

From	FBUS1 (FBUS1$$$P) (a 53 bit bus)	FADDR1 (FBUS1ADR$P)
clock 1	OP_SYL	0
clock 2	A_SYL	1
clock 3	B_SYL	2
clock 4	C_SYL	3
clock 5	PC	4

Contents of each syllable are as follows:

```
OP_SYL      = XX00  tddd OP AF BF  (initial AF, BF)
              X (bit 4-3)      = 0
              X (bit 2)        = 1 if ALIT else 0
              X  (bit 1)       = 1 if AF-exception else 0
              X  (bit 0)       = 1 if BF-exception else 0
              d                = 0 Reserved
              t                = number of syllables in
                                 the currect instruction
PC          = 00SS 1000 PPPPPP  (for OMEGA)
              'PPPPPP'       base relative program counter
            = 00SS  000 PPPPPPP  (for MCP IX)
              'PP..PP'       program counter
              SS              = 0 if no errors detected by
```

6.2.1.3 INTERFACES (Continued)

```
                                        IF and no fetch event else
                        MSD bit 3       = 0
                            bit 2       = Fetch Event flag or SI flag
                            bit 1       = Fetch Event flag or SI falg
                            bit 0       = 0
                        LSD bit 3       = Even parity over SS digit
                            bit 2       = Odd program counter error
                            bit 1       = IF undigit error
                            t 0       = IF limit error

A_SYL           = 0000  xxxx LLLLLL  (if A-operand is literal)

A,B,C_SYL       (for non-branch ops)
              i) not extended
                 = XX F E 00000 AAAAA
                   bit 3,2 of 'F' = index bits
                   bit 1,0 of 'F' = address controller bits
                   E              = 2nd digit in address
                                    syllable
                   X (bit 4)      = exception flag
                                    set if address is extended
                                    by 'A', 'B', 'F' or ('E'
                                    with no IX1) or any IX or
                                    IA
                   X (bit 3)      = indirect flag
                                    set if address controller
                                    bits is equal to '11'
                   X (bit 2-0)    = index flags
                                    set according to the IX
```

6.2.1.3 INTERFACES (Continued)

```
                                  and extension flag
                                  (IX1-IX7 & no IX)
          'AAAAA'          = base relative begin
                             address of operand
    ii) extended
        = XX F E Z 000 AAAAAA
          Z                = 1 if extension digit='D'
                             and IX indicants = 0
                             and OMEGA else 0
          E                = address extension flag
          'AAAAAA'         = base relative extended
                             begin address of operand

        (for branch ops)
     i) for OMEGA
        = XX F E W 000 Y AAAAA
          W                = 1 if not IX nor IA else 0
    ii) if extended address
          Y                = 1st digit of address
                             syllable
   iii) if not extended
          Y                = 0 if IA else
          bit 3,2 of 'Y'   = 0
          bit 1,0 of 'Y'   = address controller bits
    iv) for MCP IX
        = XX F E 0000 Y AAAAA
```

6.2.1.3 INTERFACES (Continued)

OF will redirect IF in the following exception cases:

A) in case of index or indirection in a branch address syllable, IF will stop to wait for OF to process the particular exception. It is up to the OF to issue a NEWPC command, IX/IA-DONE, together with the new program counter on the ADDRBUS (XADRBUS$P). IF will decode the command (XWBUSCMD$P) and capture the new PROGRAM COUNTER on the ADDRBUS (XADRBUS$P).

B) in case of detected code modification, OF will issue a NEWPC command, IF-FLUSH, to refetch the current instruction. It is up to IF to notice the flush command and capture the new PROGRAM COUNTER being sent on the ADDRBUS (XADRBUS$P).

C) in case of lock table full, OF will issue a NEWPC command, IF-FLUSH, to notify IF to refetch the next instruction. OF will put the new PROGRAM counter on the ADDRBUS (XADRBUS$P) for the IF.

At the end of processing the current fetch page, OF will activate the SHOVE (FSHOVE$$$$$P) signal to notify IF that OF is done with that particular fetch page. (Note that SHOVE (FSHOVE$$$$$P) signal is also passed along to XM to notify XM that it can start working on that paritcular fetch page). IF will notify OF that a particular fetch page is valid by activating the FLIP (FLIPXMFPAGEP) signal.

To send the SHOVE (FSHOVE$$$$$P) signal out, FBUS2 enable must be activated. FBUS2 enable is a bit within the CREG which must be set in conjunction with any SHOVE commands.

6.2.1.3 INTERFACES (Continued)

During a predicted taken branch opcode handling, OF will send the NTI-VALID command to the IF after checking for code modification. This is to validate the not taken instruction in the NTI queue. If an IF_FLUSH occurs before the not taken instruction has received its NTI-VALID, it will be discarded. After receiving a NTI-VALID command, the NTI queue in IF can only be cleared on a SYS-FLUSH (ESYSFLUSHP) or FET-FLUSH (EFETFLUSHP) or BRH-OK (EBRANCHOK$$P) signal from XM. The NTI-VALID command is treated by MCACM as a NO-OP.

IF will not send out a modified OPCODE if it detects an undigit or limit error whose originator is IF. This may result in invalid data from IF and mis-direct the OF OPLOOK-UP structure. It is up to the OF error handling routine to sort out what has occurred and initiate proper action to handle the error. The 'SS' digits in the OPSYL entry will indicate the nature of the error condition. When there is a SI event or PC/OP stop condition met event, the FETCH EVENT bit within the SS digit will be set. The OF will then mark the OPLIT vector of that instruction for the XM. XM will stop after finishing the instruction with the OPLIT vector marked at the next OPLIT branch.

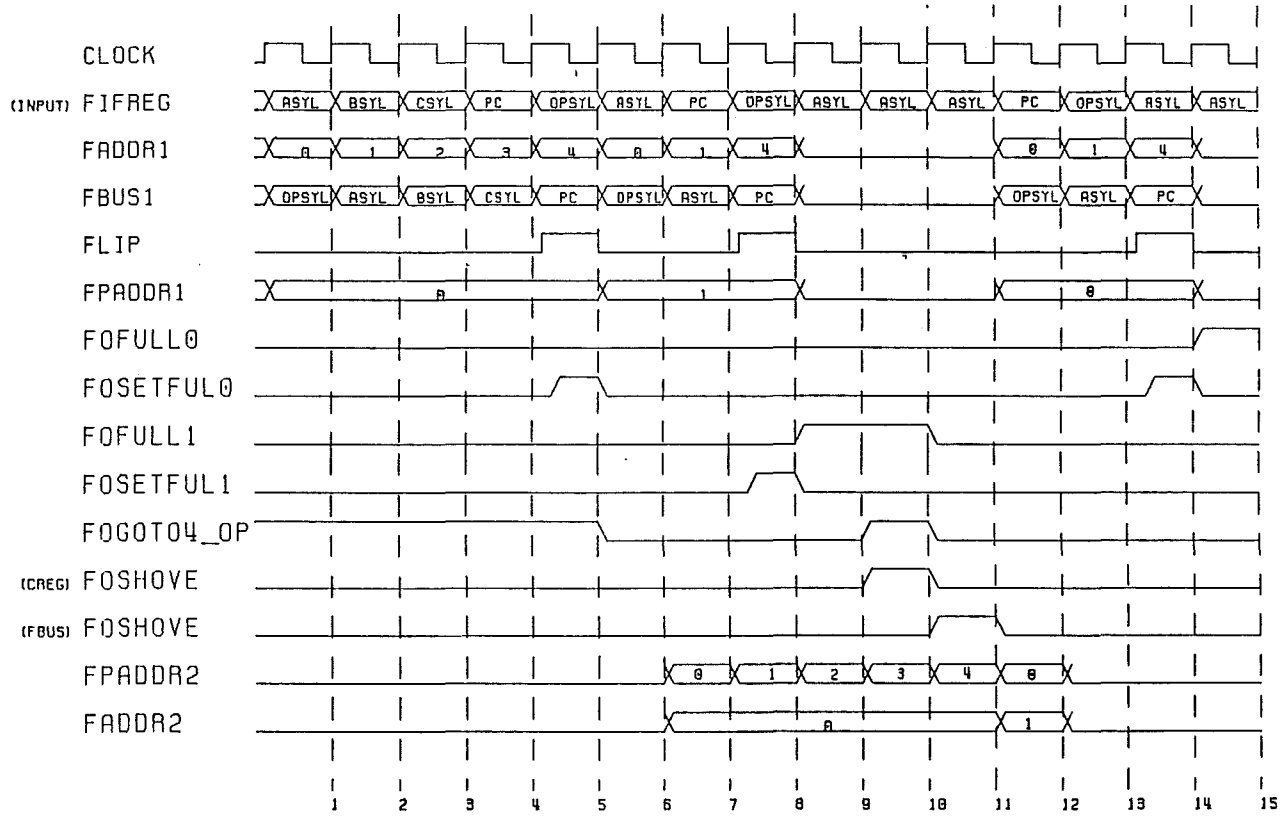


FIGURE 6-10 FBUS CONTROL TIMING

6.2.1.3 INTERFACES (Continued)

2. Execute Module

OF will handle all index, indirection or exception that are not covered by the IF unit. In the process of handling exceptions, OF will use FBUS1 (FBUS1$$$P), where OF has higher priority over the IF, to re-write any necessary entries in XM's fetch pages. Since OF has exclusive use of FBUS2 (FBUS2$$$$P), there are two sets of FBUS addresses and signals, one set for FBUS1 (FBUS1$$$P) and the other set for FBUS2 (FBUS2$$$$P). Note that XM is only going to be capturing 40 bits from FBUS1 (FBUS1$$$P) and the full 10 bits from FBUS2 (FBUS2$$$$P). OF will notify XM that a particular fetch queue page is ready for use by setting the fetch queue valid flip-flop with SHOVE (FSHOVE$$$$$P) for that page. SHOVE (FSHOVE$$$$$P) occurs while the FPADDR2 (FBUS2PAG$P) is set to to point to the page to be set valid. Along with the SHOVE (FSHOVE$$$$$P) signal, OF will pass the DATA-HIT (FDATAHIT$$$P) result and OPTIMAL (FOPTIMAL$$$P) conditions to XM on separate wires on the back plane. By the time fetch page valid bit is set, depending on the opcode, the following information will have been received by XM.

ADDR (FBUS1ADR$P)	FUBS1 (FBUS1$$$P) (used by IF & OF)	FBUS2 (FBUS2$$$$P) (used by OF only)
0	OP_SYL	OPLIT
1	A_SYL	ALEN
2	B_SYL	BLEN
3	C_SYL	CLEN
4	PC	FLAGS
5	LENGTH_SYL	
6	SPECIAL_SYL	

From FBUS1 (FBUS1$$$P), a 53 bit bus (XM only sees lower 40 bits), the following syllables (when required and not shown in order)

6.2.1.3 INTERFACES (Continued)

OP_SYL
= t000 OP AF BF (resolved AF,BF)
= 0000 1A nn xx for address error (for definition of 'nn', refer to the section on instruction related errors in the V Series Instruction Set specification)
= N000 1C OP AF for code-modified inst. (see section on code modification handling for the definition of N)

A_SYL
= LLLLLL 0000 (if A-operand is literal) trailing garbage digits in 'LLLLLL' will also be cleared out by OF

A,B,C_SYL
= BI 00 AAAAAAA
'AAAAAAA' = resolved begin address of each operand
BI = base indicant digit (for MCP IX BI=0)
= 1 000 AAAAAA (OMEGA: branch ops address syllable)
= 000 AAAAAAA (MCP IX: branch ops address syllable)
= xx V xxxxxxx (error detected during IA/IX resolution in branch address - V is a non-zero digit, see the section on instruction related errors in V Series Instruction Set spec for further information)
V = 0 if no errors
= 1 if undigits detected while resolving IA
= 2 if invalid arithmetic, overflow or underflow
= 3 if B/L error while resolving IA
= 5 if undigits in index registers

6.2.1.3 INTERFACES (Continued)

= 6 if undigits in offset during IX add
= E if instruction time out while resolving IA
= F if unspecified address error

PC	= 1 000 PPPPPP	(for OMEGA)
	'PPPPPP'	= base relative program counter)
	= 000 PPPPPPP	(for MCP IX)
	'PPPPPPP'	program counter
LENGTH_SYL	= 000 LLLLLLL	(for MVW,MVC,NTR,TRN,MVD where length of operand is greater than 200 digits)
SPECIAL_SYL	= xxxxxxxxxx	(used for passing information between OF & XM)

From FBUS2 (FBUS2$$$P), a 10 bit bus, the following syllables (when required and not shown in order)

OPLIT	= 10 bit encoded field
	bit 9 = FETCH event - SI
	bit 8 = T.B.S.
	bit 7-0 = 8 bits op-lit vector (ALIT bit is the LSB)
ALEN_SYL	= LLL (length of A-operand in digits)
BLEN_SYL	= LLL (length of B-operand in digits)
CLEN_SYL	= LLL (length of C-operand in digits) (a length of 0 means 400 digits)
FLAGS	= 10 bit encoded field
	bit 9-8 = no. of extended address
	bit 7 = AF > BF
	bit 6 = AF = BF
	bit 5-4 = Ac A address controller
	bit 3-2 = Bc B address controller
	bit 1-0 = Cc C address controller

6.2.1.3 INTERFACES (Continued)

OF will pre-read length modulo-10 digits of A and/or B field when an instruction specifies A and/or B as read operands. For example, OF will send out a request for 6 digits for a 16 UN MVN instruction. For a complete description of operand pre-read algorithm, please refer to the section on instruction dependent handling. Also OF will not re-read literal operand for XM).

OF will require the following signals from XM:

- 1) SYS_FLUSH (ESYSFLUSHP) - sent by XM at the end of an environment change processing.

- 2) FETCH_FLUSH (EFETFLUSHP) - sent by XM at the end of a mis-predicted conditional branch instruction. An ADDRBUS (XADRBUS$P) command will also be sent to notify IF whether the bad branch is predicted Taken or Not Taken.

- 3) LIX_OK (ELIXOK$$$P) - send by XM at the start of the execution of a LIX, BRV or RET (HYPER) instructions.

- 4) OP_COMPLETE (EOPCOMPLETEP) - sent by XM to notify the lock unit that a particular lock entry can now be marked as deleted.

- 5) TIME-OUT (E$TIMEOUT1P) - send by instruction timer (residing in XM) indicating that a potential instruction time-out condition exists.

3. Memory Control and Cache Module (MCACM)

OF forwards memory requests to the MCACM via the ADDRBUS (XADRBUS$P). Data from memory will be sent back through the RBUS (MRBUS$$$P) to OF or to XM if the request is an operand pre-read.

6.2.1.3 INTERFACES (Continued)

4. Maintenance

To implement single instruct (SI), IF requires the maintenance subsystem to set up the appropriate qualification. IF will then set the FETCH EVENT bit when the 'SS' digit is sent. OF will then notify the XM through a bit in the OPLIT vector. Opcodes that are already in XM fetch page will not be affected. During SI, it is up to the MP to invalidate the index valid bits for IX1-3. Also a bad branch not taken FETCH flush should be issued after XM finishes with the current instruction. This will prevent IF from pulling the next opcode from the NTI queue. It also allows the OF to restore any of the mobile index registers which it has cached while the XM is executing the instruction. The flush must be issued at an appropriate time to allow the LIX signal to reach OF. When exiting from SI mode, the MP must unmark all opcodes that are already processed by OF.

To stop on PC/OP, the maintenance subsystem has to set up any appropriate stop requirements and qualifications. Upon detecting a match, the IF will set the FETCH EVENT bit of the 'SS' digit for the OF which will in turn mark the OPLIT vector. XM will stop after executing the instruction at the next OPLIT branch.

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

COMPANY
CONFIDENTIAL

1993 5212

V500 FETCH MODULE

ENGINEERING DESIGN SPECIFICATION Rev. B

6.2.1.3 INTERFACES (Continued)

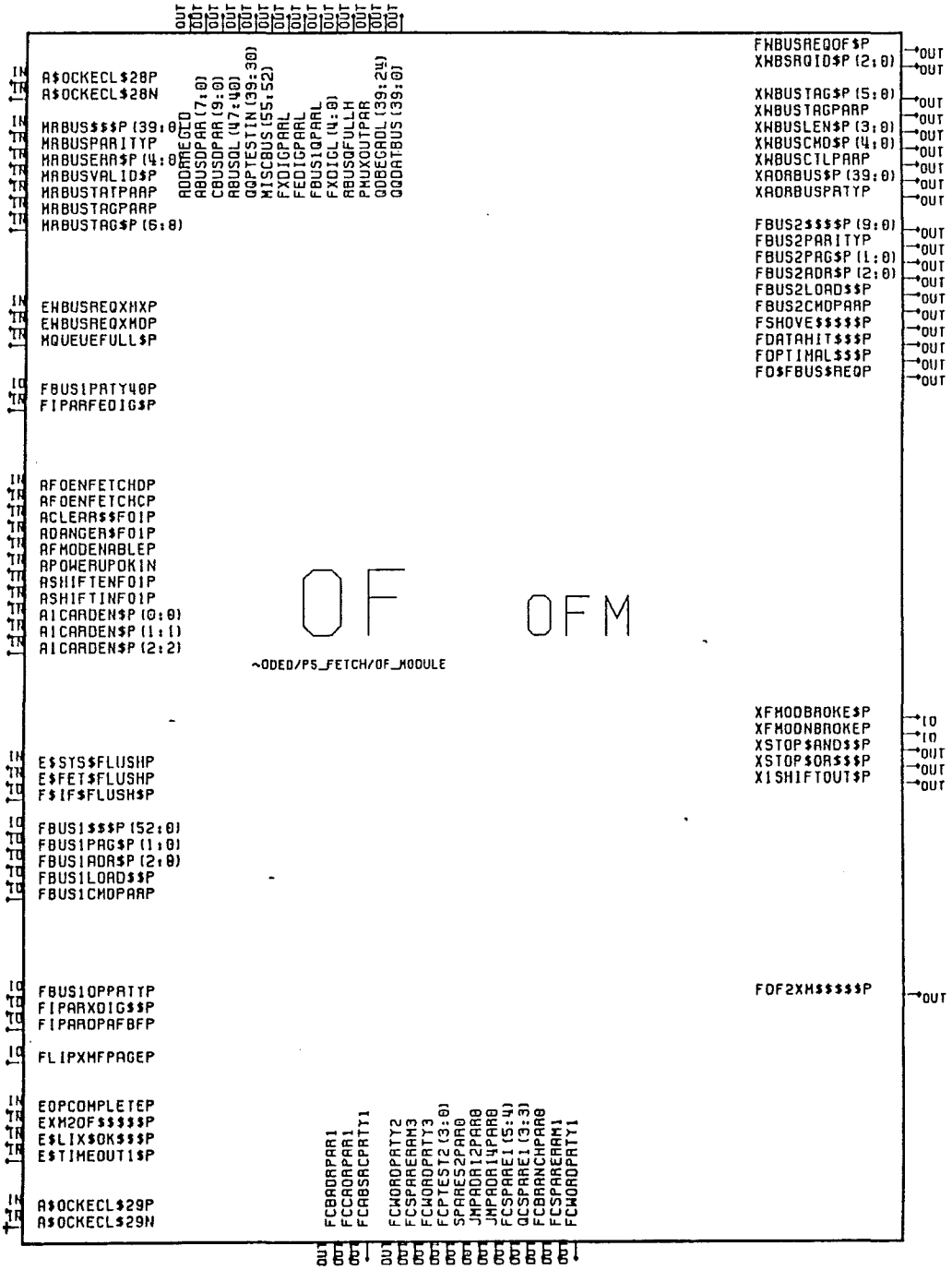


FIGURE 6-11 OF INTERFACE

6.2.1.3 INTERFACES (Continued)

Detailed OF interface to IF, XM, MCACM and SMC are as follows:

- AFM0DENABLEP - fetch module clock enable from SMC.
- A$0CKECL$28P - active high free running clock to OFD from SMC.
- A$0CKECL$28N - active low free running clock to OFD from SMC.
- A$0CKECL$29P - active high free running clock to OFC from SMC.
- A$0CKECL$29N - active low free running clock to OFC from SMC.
- AP0WERUP0K1N - active low power up clear from SMC.
- ADANGER$F01P - signal to gate with hidden state RAM write from SMC.
- ACLEAR$$F01P - synchronous clear from SMC.
- AF0ENFETCHCP - OFC clock enable signal from SMC.
- AF0ENFETCHDP - OFC clock enable signal from SMC.
- ASHIFTENF01P - shift enable from SMC.
- A1CARDEN$P(2:0) - card enable from SMC.
- ASHIFTINF01P - serial data in signal from SMC.
- XFM0DBR0KE$P - Fetch module broken to SMC.
- XFM0DNBR0KEP - Fetch module not broken to SMC.
- X1SHIFT0UT$P - serial data out signal to SMC.

6.2.1.3 INTERFACES (Continued)

- o XST0P$AND$$N - stop "and" signal to SMC.
- o XST0P$0R$$$P - stop "or" signal to SMC.
- o FBUS1ADR$P(2:0) - Fetch page 1 entry address from IF to OF & XM or OF to XM.
- o FBUS1PAG$P(1:0) - Fetch page 1 page address from IF to OF & XM or OF to XM.
- o FBUS1L0AD$$P - FBUS1 write enable signal from IF to OF & XM or OF to XM.
- o FBUS1CMDPARP - FBUS1 control parity from IF to OF & XM or from OF to XM: generates even parity across FBUS1ADR$P, FBUS1PAG$P, FBUS1LOAD$$P, and FBUS1CMDPARP.
- o FBUS1$$$P(39:0) - FBUS1 data bus from IF to OF & XM or from OF to XM.
- o FBUS1PRTY40P - FBUS1 data bus parity from IF to OF & XM or from OF to XM: generates even parity across FBUS1$$$P(39:0) and FBUSPRTY40P.
- o FBUS1$$$P(52:40) - upper FBUS1 data bus from IF.
- o FIPARXDIG$$P - FBUS1$$$P(52:48) data bus from IF: generates even parity across FBUS1$$$P(52:48) and FIPARXDIG$$P.
- o FIPARFEDIG$P - FBUS1$$$P(47:40) data bus parity from IF: generates even parity across FBUS1$$$P(47:40) and FIPARFEDIG$P.
- o FIPAR0PAFBFP - FBUS1$$$P(23:0) data bus parity from IF: generates even parity across FBUS1$$$P(23:0) and FIPAR0PAFBFP.

6.2.1.3 INTERFACES (Continued)

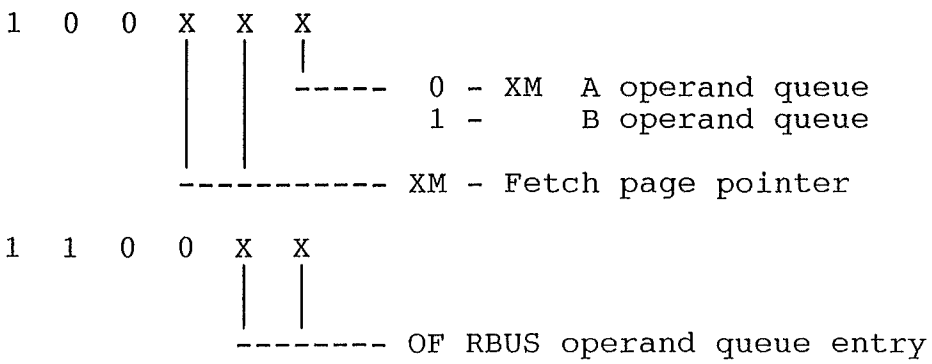
- FBUS10PPRTY - FBUS1$$$P(23:16) data bus parity from IF: generates even parity across FBUS1$$$P(23:16) and FBUS10PPRTYP.
- FLIPXMFPAGEP - fetch page valid signal from IF.
- FWBUSREQ0F$P - XADRBUS$P request from OF.
- FIFFLUSH$P - IF redirection signal to IF, NEWPC available on XADRBUS$P.
- FSH0VE$$$$$P - OF fetch page available to IF & XM.
- F0$FBUS$REQP - FBUS1 request to IF.
- FBUS2$$$$P(9:0) - FBUS2 data bus to XM & IF.
- FBUS2PARITYP - FBUS2$$$P(9:0) data bus parity to XM: generates even parity across FBUS2$$$P(9:0) and FBUS2PARITYP.
- FBUS2PAG$P(1:0) - FBUS2 entry address to IF & XM.
- FBUS2ADR$P(2:0) - FBUS2 page address to IF & XM.
- FBUS2L0AD$$P - FBUS2 write enable signal to XM.
- FBUS2CMDPARP - FBUS2 control signals parity to XM: generates even parity across FBUSPAG$P, FBUS2ADR$P, FBUS2L0AD$$P and FBUS2CMDPARP.
- FDATAHIT$$$P - operand pre-reads validity indicator to XM.
- F0PTIMAL$$$P - optimal case indicator to XM.
- F0F2XM$$$$$P - event wire to XM.
- EFETFLUSHP - bad branch flush signal from XM.

6.2.1.3 INTERFACES (Continued)

- o ESYSFLUSHP - environment change signal from XM.
- o E0PC0MPLETEP - write complete signal from XM.
- o EXM20F$$$$$P - event wire from XM.
- o ELIX0K$$$P - completion of LIX instruction flag from XM.
- o E$TIME0UT1$P - instruction timeout signal from XM.
- o EWBUSREQXMDP - XADRBUS$P request from XM.
- o EWBUSREQXMXP - XADRBUS$P request from co-processor.
- o XWBSRQID$P(2:0) - XADRBUS$P requestor ID to MCACM. XWBSRQID$P=4 if OF is driving XADRBUS$P.
- o XWBUSTAG$P(4:0) - XADRBUS$P return tag address ID to MCACM. XWBUSTAG$P in range of "28" to "2F" (XM operand pre-reads) or "30" to "33" (OF operand reads) if OF is driving XADRBUS$P.

Definition of XWBUSTAG$P from OF.

XWBUSTAG$P



6.2.1.3 INTERFACES (Continued)

- o XWBUSTAGPARP - XWBUSTAG$P parity field to MCACM: generates even parity across XWBUSTAG$P and XWBUSTAGPARP.

- o XWBUSCMD$P(4:0) - XADRBUS$P command field to MCACM. Following commands can be driven by OF:

  1 - operand read
  5 - read base
  7 - read limit
  8 - IX/IA done to IF
  9 - IF flush to IF
  A - NTI valid to IF
  19 - write base
  1B - write limit

- o XWBUSLEN$P(3:0) - XADRBUS$P length field to MCACM. XWBUSLEN$P in range of "0" to "9" if OF is driving XADRBUS$P.

- o XWBUSCTLPARP - XWBUSCMD$P and XWBUSLEN$P parity field to MCACM: generates even parity across XWBUSCMD$P, XWBUSLEN$P and XWBUSCTLPARP.

- o XADRBUS$P(39:0) - address field to MCACM or IF.

- o XADRBUSPRTYP - XADRBUS$P parity field to MCACM or to IF: generates even parity across XADRBUS$P and XADRBUSPRTYP.

- o MQUEUEFULL$P - MCACM input queue full signal from MCACM.

- o MRBUS$$$P(39:0) - read data bus from MCACM.

- o MRBUSPARITYP - read data bus parity from MCACM: generates even parity across MRBUS$$$P and MRBUSPARITYP.

- o MRBUSERR$P(4:0) - error vector from MCACM. OF will recognize MRBUSERR$P = 11 as limit error and MRBUSERR$$P = 12 as undigit error.

6.2.1.3 INTERFACES (Continued)

- MRBUSVALID$P - read data bus valid flag from MCACM.
- MRBUSTATPARP - parity field across MRBUSERR$P and MRBUSVALID$P from MCACM: generates even parity across MRBUSERR$P, MRBUSVALID$P and MRBUSTATPARP.
- MRBUSTAG$P(4:0) - read bus tag for data on following clock from MCACM. OF will only recognize MRBUSTAG$P in range of "30" to "33".
- MRBUSTAGPARP - read bus tag parity from MCACM: generates even parity across MRBUSTAG$P and MRBUSTAGPARP.

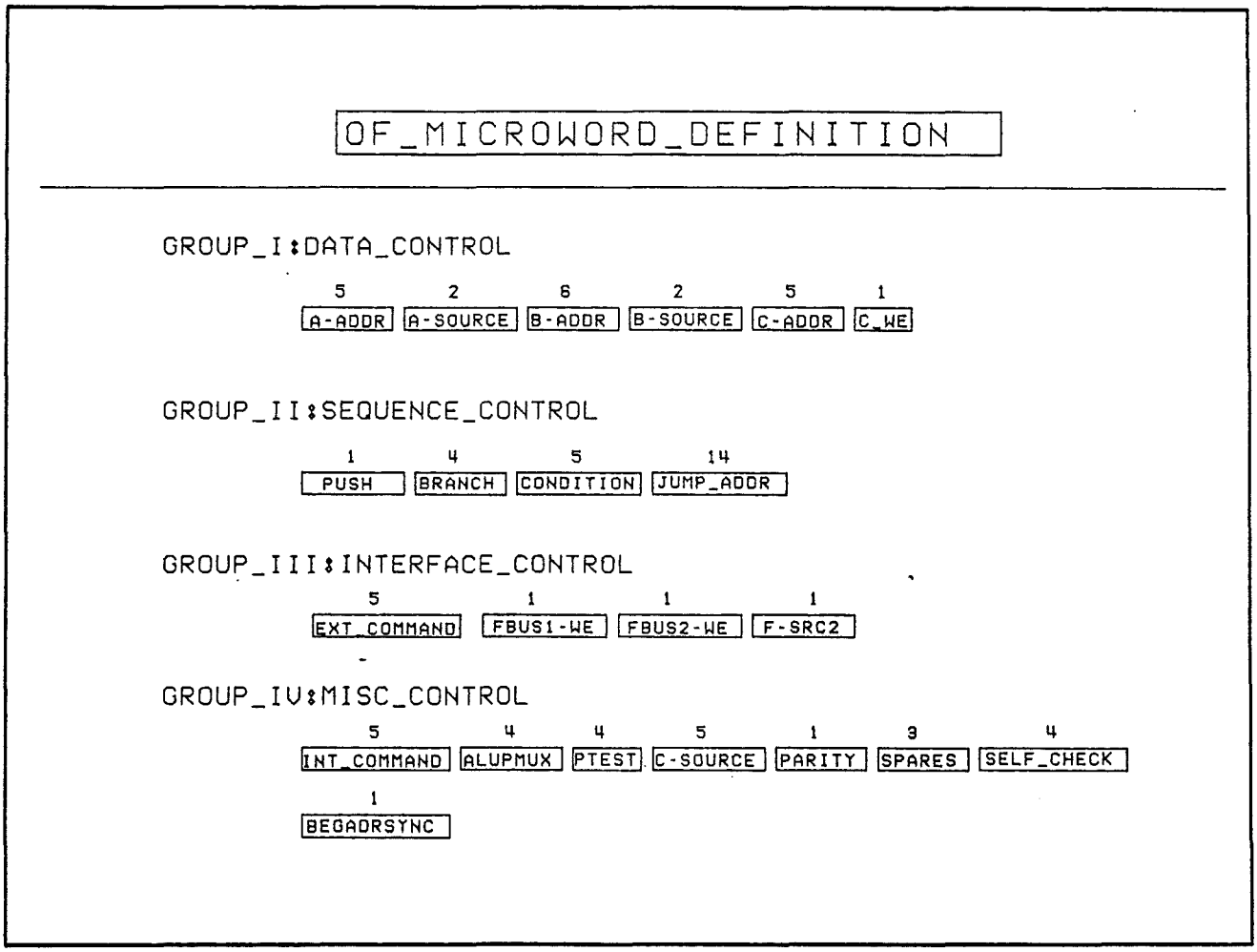


FIGURE 6-11-1 OF MICROWORD DEFINITION

6.2.1.4 OPERAND FETCH CREG DEFINITION

Definition of P5 OF CREG:

Field name	CREG field (bit)	Explanation
SPARES	21:3	
FBUS1-WE	29:1	= 1 Write enable signal to XM fetch queue (write content of CBUS to an entry of the fetch queue page selected by C-ADDR) = 0 No action
FBUS2-WE	28:1	= 1 Write enable signal to XM fetch queue (write content of OPLEN output to fetch queue. If FLAGS are chosen then the FADDR2 will have '100', otherwise, FADDR2 will be the same as B-ADDR) = 0 No action
F-SRC2	27:1	= 1 Select FLAGS from OPLEN output. Send the OPLEN output to XM fetch queue and set FADDR2 to '100'. = 0 Allow B-ADDR to select OPLIT, ALEN, BLEN and CLEN as OPELN output.

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

INT-COMMAND 48:5

= 0 No action
= 1 Load OPLEN OP
= 2 Load OPLEN AF
= 3 Load OPLEN BF
= 4 Load OPLEN Ac
= 5 Load OPLEN Bc
= 6 Load OPLEN Cc
= 7 Set IX1 cache invalidate enable
= 8 Set IX2 cache invalidate enable
= 9 Set IX3 cache invalidate enable
= 10 Set IX4 cache status bit
= 11 Set IX5 cache status bit
= 12 Set IX6 cache status bit
= 13 Set IX7 cache status bit
= 14 Code check and set condition 1
= 15 Initiate self flush
System flush if C-WE = 1
Fetch flush if C-WE = 0
= 16 Clear ucode testable
= condition 1
= 17 Set ucode testable condition 1
= 18 Clear ucode testable
condition 2
= 19 Set ucode testable condition 2
= 20 Pop Lock
= 21 Code check to lock unit
= 22 Data check to lock unit
= 23 IX/IA check to lock unit

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

= 24 Write lock to lock unit
= 25 Write lock and set SHOVE
= FBUS2-WE bit 28 has to be set
= to send SHOVE
= 26 Set SHOVe
= FBUS2-WE bit 28 has to be set
= to send SHOVE
= 27 Clear IX4-IX7 cache status reg
= 28 Set ucode testable condition 3
= 29 Clear ucode testable condition 3

= 30 Set OFTOXM (EOF2XM$$$$$P) event wire

PTEST 78:4 = 0-1 Reserved
= 2 Address Decode
= 3 Auxillary Address Decode
= 4 Address Decode Ignoring Extension Digit
= 5 Clear PTREG
= 6 LIX Decode 1 - dim. override
= 7 LIX Decode 2 - register load
= 8 Load PTREG
= 9 Calculate Literal Length
= 10 Undigit tests & IX sign undigit tests
= 11 T.B.S,
= 12 Infl Decode
= 13 T.B.S.
= 14 Length Calculation Assist
= 15 PTREG hold

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

ALUPMUX 43:4

(ALU portion)
= 0 7 digit BCD subtract
AUREG=000 SSSSSSS
where SSSSSSS is adder output
= 1 7 digit code base BCD IX add
AUREG=1 00 SSSSSSS
where SSSSSSS is adder output
= 2 7 digit dimension override BCD IX add
AUREG=B(9) 00 SSSSSSS
where B(9) is MSD of BBUS
= 3 7 digit BCD IX add
AUREG= 000 SSSSSSS
=4-F 7 digit BCD add
AUREG=000 SSSSSSS
where SSSSSSS is adder output

(PMUX portion)
= 0 Left rotate by 1
D8 D7 D6 D5 D4 D3 D2 D1 D0 D9
= 1 Pass
D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
= 2 Pass
D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
= 3 Left rotate by 3
D6 D5 D4 D3 D2 D1 D0 D9 D8 D7
= 4 Left rotate by 4
D5 D4 D3 D2 D1 D0 D9 D8 D7 D6

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

= 5 Left rotate by 5
D4 D3 D2 D1 D0 D9 D8 D7 D6 D5
= 6 Left rotate by 6
D3 D2 D1 D0 D9 D8 D7 D6 D5 D4
= 7 Left rotate by 7
D2 D1 D0 D9 D8 D7 D6 D5 D4 D3
= 8 Left rotate by 8
D1 D0 D9 D8 D7 D6 D5 D4 D3 D2
= 9 Left rotate by 9
D0 D9 D8 D7 D6 D5 D4 D3 D2 D1
= 10 Non-branch IA syllable justify (not extended)
0 0 0 0 0 D8 D7 D6 D5 D4
= 11 Extended IA syllable justify
0 0 0 0 D7 D6 D5 D4 D3 D2
= 12 Branch IA syllable justify (not extended)
0 0 0 0 D9* D8 D7 D6 D5 D4
D9* = D9 with zeroed out bit 3 & bit 2
= 13 Strip Zone
D8 D6 D4 D2 D0 0 0 0 0 0
= 14 IX rotate
D8 0 D9 0 D7 D6 D5 D4 D3 D2
= 15 Left rotate by 2
D7 D6 D5 D4 D3 D2 D1 D0 D9 D8

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

C-WE 54:1
= 1 Write content of C-BUS to scratchpad selected by C-ADDR
= 0 No action

C-SOURCE 53:5
= 0 PPPPPPPPPP
= 1 PPPPPPPPPB
= 2 PPPPPPPPBB
= 3 PPPPPPPBBB
= 4 PPPPPPBBBB
= 5 PPPPPBBBBB
= 6 PPPPBBBBBB
= 7 PPPBBBBBBB
= 8 PPBBBBBBBB
= 9 PBBBBBBBBB
= 10 PBBBPPPPPP (Extended IA branch rotate)

= 11 BBBBBBPPBB (Infl AF)
= 12 BBBBBBBBPP (Infl BF)

= 13 BBBBBBBBBB
= 14 BBBBBBBBBP
= 15 BBBBBBBBPP
= 16 BBBBBBBPPP
= 17 BBBBBBPPPP
= 18 BBBBBPPPPP
= 19 BBBBPPPPPP
= 20 BBBPPPPPPP
= 21 BBPPPPPPPP
= 22 BPPPPPPPPP

= 23 PPBPPPPPPP (Branch error)
= 24 BPPPPPPPPP (Dimension override)

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

C-ADDR	63:5	=0-31 Scratch pad
SELF-CHECK	17:4	= Initiate self check test f/f's = 0 No action = 1 Perform FBUS1 write back = 2 Perform RBUS write back = 3 Force CREG field parity error = 4 Initiate force error 0 OFD = 5 Initiate force error 1 OFD = 6 Initiate force error 2 OFD = 7 Initiate force error 3 OFD = 8 Initiate force error 4 OFD = 9 Initiate force error 0 OFC = A Initiate force error 1 OFC = B Initiate force error 2 OFC = C Initiate force error 3 OFC = D Initiate force error 4 OFC = E Reserved = F Reserved
PARITY	79:1	= Even parity across microword
B-SOURCE	56:2	= 00 Scratch pad = 01 OPLEN registers = 10 Litfile

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

B-ADDR	74:6	if B-SOURCE=00 = 0-31 Scratch pad if B-SOURCE=01 = 0-31 Litfile if B-SOURCE=10 = 0 Reserved & invalid = 1 OPLEN ALEN register = 2 OPLEN BLEN register = 3 OPLEN CLEN register else Reserved (if any LEN register is selected as B-source then 30 leading 0 bits will be padded to output from OPLEN unit.)
PUSH_STACK	35:1	= 1 Push JUMP-ADR into stack & update stack pointer if BRANCH field=1 (Fallthrough) or Push UPC+1 into stack & update stack pointer if BRANCH field is not equal to 1 (Fallthrough) or 2 (Return) = 0 No action
A-SOURCE	58:2	= 00 Scratch pad = 01 Fetch page = 10 RBUSQ = 11 AUREG

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

A-ADDR	68:5	if A-SOURCE=00 = 0-31 Scratch pad if A-SOURCE=01 = 0-7 Fetch page = 8-31 Reserved if A-source=10 = RBUSQ if A-SOURCE=11 = AUREG
BRANCH	39:4	= 0 GOTO4OP (16k op map branch) = 1 FALLTHRU4 (UPC+1) = 2 RETURN4 = 3 GOTO32P (PTREG branch) = 4 GOTO2 (2-way branch) = 5 GOTO4 (4-way branch) = 6 GOTO16S (IX cache status) = 7 GOTON-EXP (16-way branch) = 8 GOTO16AX (X digit of Asyl) = 9 GOTO16BX (X digit of Bsyl) = 10 GOTO16CX (X digit of Csyl) = 11 GOTO8IX3 (IX 1-3 FIT bit) = 12-15 Reserved
CONDITION	34:5	(4-way branch conditions) = 0 Low 2 bits of OPLIT map address = 1 Low 2 bits of UPC+1

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

```
= 2   Low 2 bits of Top of stack
= 3   Low 2 bits of Branch address
= 4   low 2 bits of Asyl 'X'
= 5   low 2 bits of Bsyl 'X'
= 6   low 2 bits of Csyl 'X'
= 7   Ac
= 8   Bc
= 9   Cc
= 10  low 2 bits of IX4-IX7
= 11  IX2 & IX1 FIT bits
= 12  IF undigit & limit error
= 13  Undigit-AB & AU-EXCEPT
      Undigit-A is PTREG-0
      Undigit-B is Undigit F/F of
      BBUS
= 14  OF undigit & limit error
= 15  AEXCEP & '0'

= 16  T.B.S.
= 17  PTREG-10
= 18  PTREG-32
= 19  T.B.S.
= 20  AF>=BF (From OPLEN)
= 21  Interrupt
      Bit 1: code-hit or addr-error
      or time-out or Odd PC
      Bit 0: Lock-full
= 22  Comparator A>=B
= 23  AU-EXCEPT & AU = 0
= 24  Unidigt-AB or addr-error
```

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

& AU-EXCEP
= 25 Condition-1 & IX1VALID
= 26 Condition-1 & IX2VALID
= 27 Condition-1 & IX3VALID
= 28 CB-EXCEPTION
= 29 Addr-error & condition-1
= 30 Number of extended syllable

(2-way branch)
= 0 IX/IA-hit
= 1 Code-hit
= 2 PTREG-4
= 3 ODD-PC
= 4 ODD-PC or codehit
= 5 XMEVENT
= 6 Undigit-B
= 7 At_least_2_lock_entries_avail
= 8 NEWOPSYL
= 9 Time-out
= 10 Addr-error
(OF undigit err or OF limit err or IF undigit err or OF limit err)
= 11 Data-hit
= 12 LIX_ED (ucode clear/set testable condition-2)
= 13 Optimal flag
= 14 Condition-3
= 15 T.B.S.

= 16 T.B.S.
= 17 PTREG-0
= 18 PTREG-2
= 19 T.B.S.
= 20 AF=BF (From OPLEN)
= 21 Interrupt bit 0
Bit 0: Lock-full

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

= 22 Comparator A=B
= 23 AU = 0
= 24 AU-EXCEP
= 25 IX1VALID
= 26 IX2VALID
= 27 IX3VALID
= 28 B-EXCEPTION
= 29 Condition-1
= 30 LSB - number of extended syl.

EXT-COMMAND	26:5	= 0 No action = 1 A operand pre-read = 2 B operand pre-read = 3 Read memory = 4 Read base = 5 Write base = 6 Read limit = 7 Write limit = 8 Reserved = 9 Reserved = 10 Reserved = 11 Reserved = 12 Reserved = 13 Reserved = 14 Reserved = 15 Set IFFLUSH (BURP) = 16 Set IXIADONE = 17 Set NIAVALID = 18-1F Reserved
BEGADR-SYNC	18:1	Begin address reg sync bit This bit is set whenever there is an external command or lock unit command (OF Live Freeze will result if previous external request has not been sent).

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

JUMP-ADDR 13:14 Jump Address

Physical layout of the CREG :

Due to the implementation of the CREG gate arrays, the CREG is partitioned in the following format:

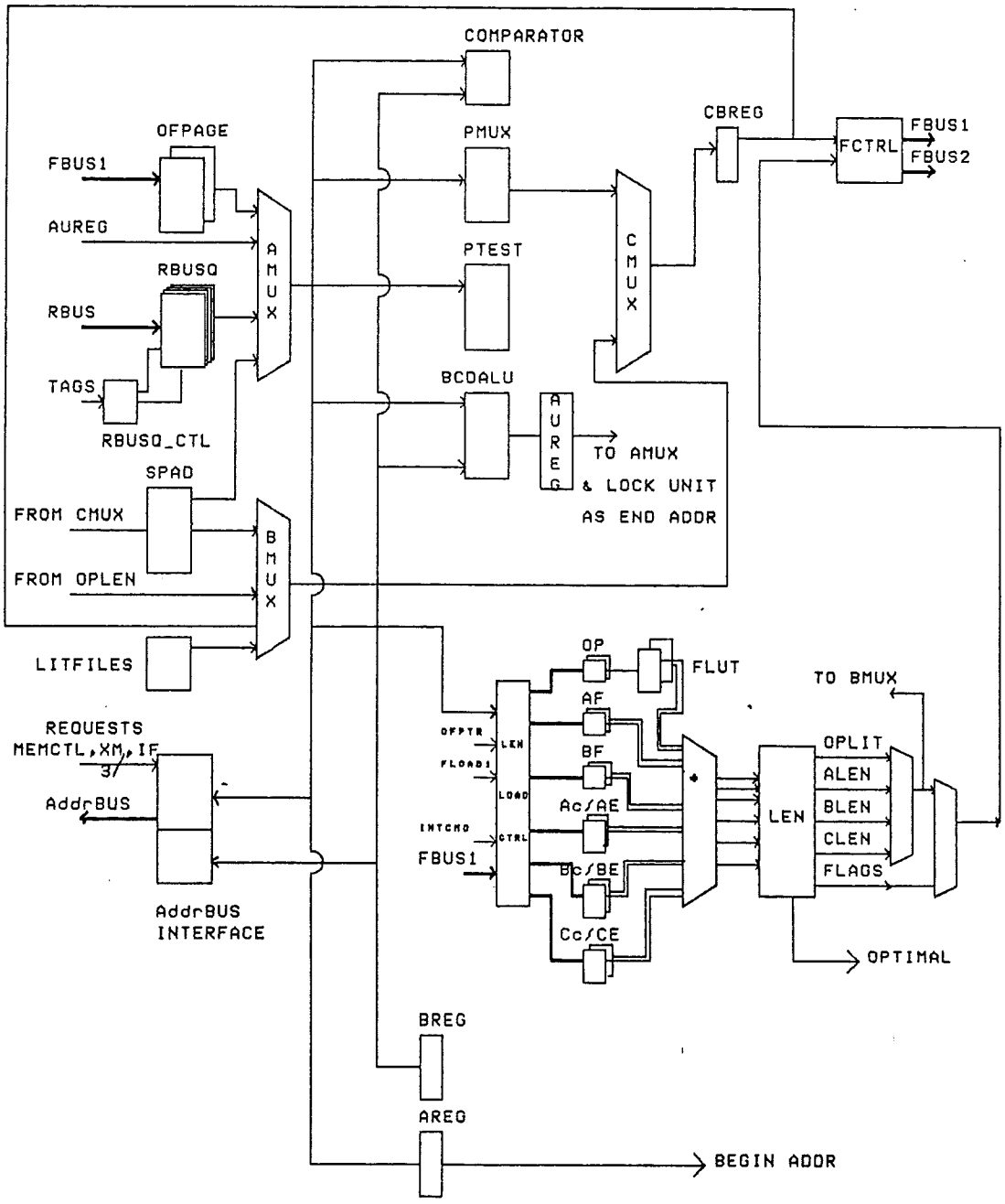
Type 1	Ucode	Definition	Field Parity	
26:1		Not used	nil	
25:1	79:1	Word Parity	nil	
24:4	78:4	PTEST	nil	
20:6	74:6	B-ADDR(5:0)	Covers 20:6	- B address
14:5	68:5	A-ADDR(4:0)	Covers 14:5	- A address
9 :5	63:5	C-ADDR(4:0)	Covers 9:5	- C address
4 :2	58:2	A-SOURCE(1:0)	Covers 4:5	- A,B-source & C-WE
2 :2	56:2	B-SOURCE(1:0)	See above	
0 :1	54:1	C-WE	See above	
Type 2				
26:5	53:5	C-SOURCE(1:0)	Covers 26:5	- C source
21:5	48:5	INTCMD(4:0)	Covers 21:5	- Internal command
16:4	43:4	ALUPMX(3:0)	Covers 16:4	- ALU/PMUX command
12:4	39:4	BRANCH(3:0)	Covers 12:5	- Branch & Push Stack
8 :1	35:1	PUSH STACK	See above	
7 :5	34:5	TESTCOND(4:0)	nil	
2 :1	29:1	FBUS1-WE	Covers 2:3	- FBUS1, 2-WE, FSRC2
1 :1	28:1	FBUS2-WE	See above	
0 :1	27:1	FSRC2	See above	

6.2.1.4 OPERAND FETCH CREG DEFINITION (Continued)

Type 3				
26:5	26:5	EXTCMD(4:0)	Covers 26:5	- External command
21:3	21:3	SPARE(2:0)	nil	
18:1	18:1	BEGADRSYNC	nil	
17:4	17:4	SELF CHECK(3:0)	Covers 17:4	- Self test command
13:14	13:14	JMPADR(13:0)	Covers 13:12	- for sequencer 1
			Covers 1:2	- for sequencer 2

6.2.1.5 OF FORMATTER STRUCTURE

The OF formatter has the responsibility to resolve and preread operands for the XM. It retrieves preprocessed instructions from the OFpage after IF 'flips'. The formatter has 5 major functional submodules to perform data transformation : PMUX, PTEST, ADDER, COMP and PASS/CONCAT. The ABUS and the BBUS serve as operand buses and the CBUS, result bus. Controlled by the CREG, data on the two operand buses are processed by the data transformation submodules. The PMUX and PASS/CONCAT are used for positional transformation and concatenation of either one or both of the operands from the ABUS and BBUS. The PTEST submodule provides fast combinatorial results on ABUS data. The ADDER is basically a 7-digit wide BCD adder. The COMP module is a 10 digit comparator on A- and B-Bus data.



OF DATA PATH

FIGURE 6-12 OF BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

OF FETCH PAGE

The OF Fetch page, OFPAGE is a storage structure in OF for capturing data sent from the IF to the execute module through FBUS1. There are two ofpages with each ofpage subdivided into 8 entries each. Each entry is 48 bits wide and used for capturing data on FBUS1 (FBUS1$$$P). With 2 ofpages, the IF can be working on one instruction while the OF is finishing the previous one. The LSB from the FPADDR (FBUS1PAG$P) will be concatenated with the FADDR (FBUS1ADR$P) to select 1-of-8 entries within 1-of-2 pages. When OF reads from the OFPAGE, it uses a 1-bit internal ofpage pointer to concatenate with 3 bits of AADDR. IF and OF will never use the same ofpage as they run in non-overlap mode.

The OFPAGE is implemented as a single read/single write register file, capable of reading and writing within the same clock cycle from different locations. Data is driven onto the ABUS. The OFPAGE capture data on the FBUS1 (FBUS1$$$P) whenever IF is writing. The OF can also write the the OFPAGE via internal buses during FBUS1 (FBUS1$$$P) write back mode.

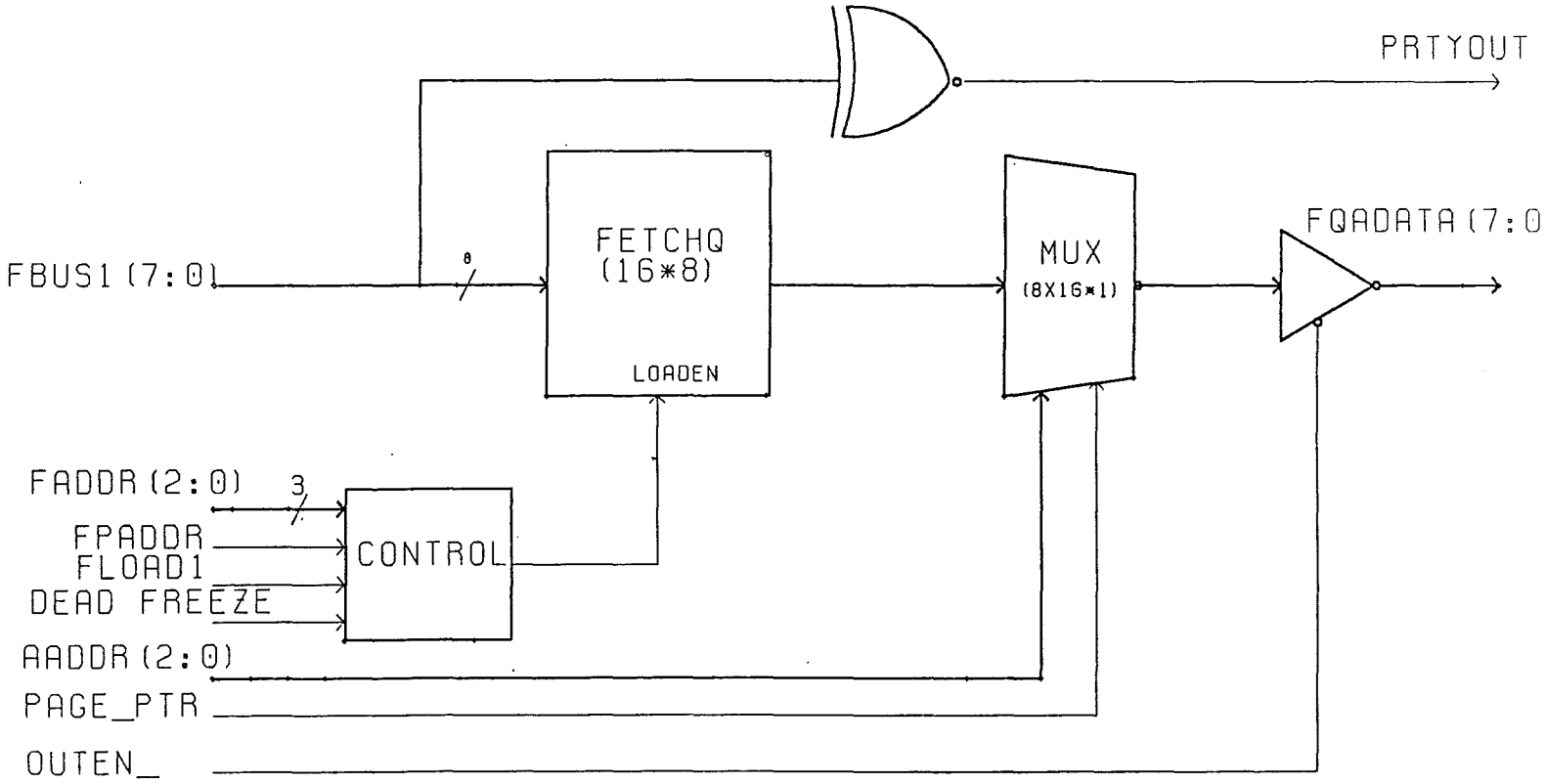


FIGURE 6-13 OFPAGE BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

SCRATCH PAD

The Scratch Pad is used to:

1. Store op syllables while they are being resolved,

2. Cache IX registers,

3. Act as temporary storage locations for microcode.

The Scratch Pad is a dual port read/single port write latch structure. It has 32 entries with each entry being 40 bits wide. Data from the scratch pad can be driven onto both the ABUS and BBUS. Information to be stored back into the scratch pad is put on the CBUS. The scratch pad is capable of reading and writing within the same clock cycle. When a write to the locations containing the primary IX1, IX2, IX3 occurs, the corresponding index valid bits will be set in the following clock. This indicates that the corresponding index register has been cached and is valid.

The hardware of the scratch pad is digit sliced and therefore 11 identical gate arrays are required. Each array consists of write logic to decode the write address, one digit of the CMUX, 32 by 4 bit latch structure, multiplexors for the two read ports, second level latches, AMUX and BMUX to drive the ABUS and BBUS, a one digit comparator and finally one digit of the CBREG as shown in the diagram below. The incoming address lines will be covered by parity as well as the data path which is protected by word parity generated and stored in an eleventh gate array used mainly for fault detection purposes.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

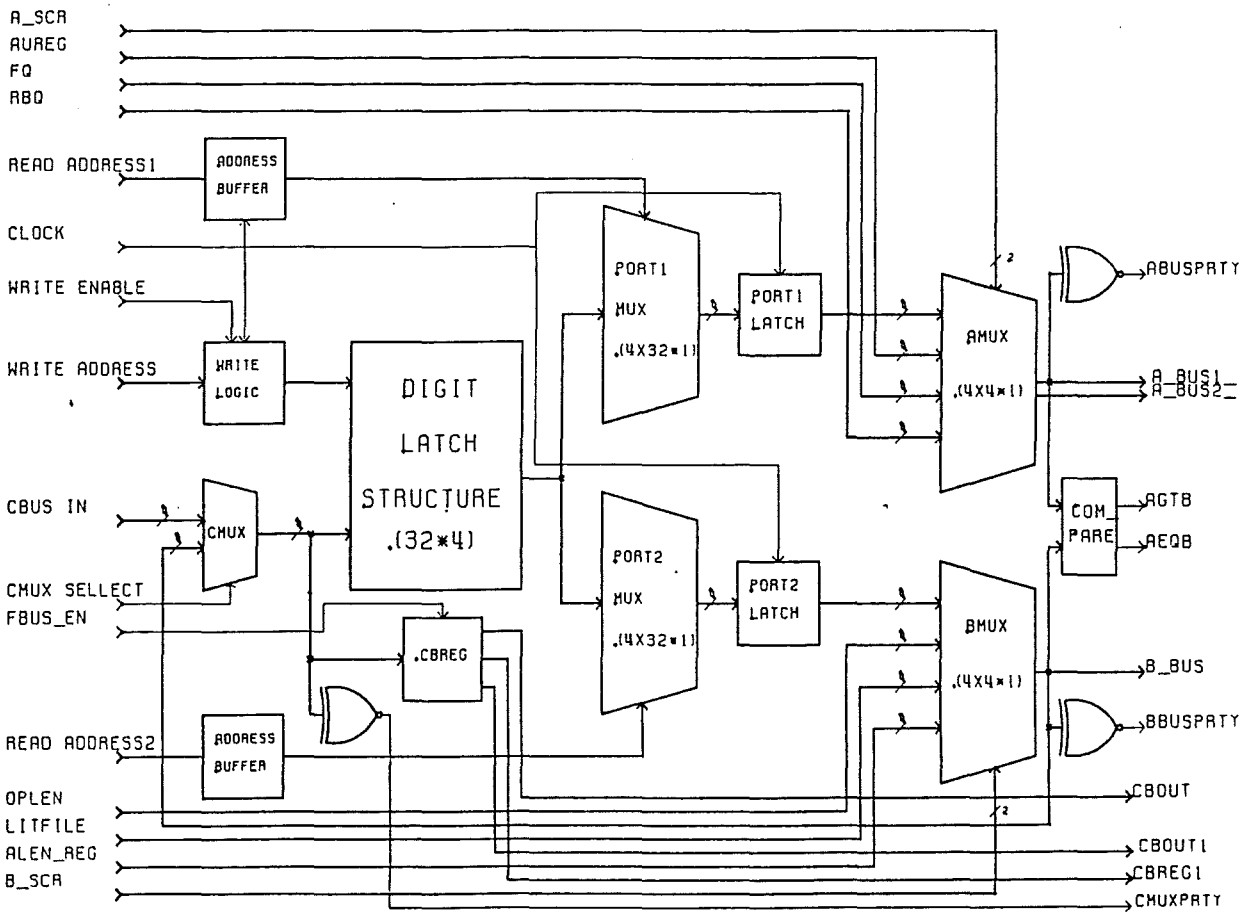


FIGURE 6-14 SCRATCH PAD BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

RBUSQ

The RBUSQ is a 4 deep by 40 bit queue used to latch data coming from memory/cache. Since data can be returning out of order, each one is preceded by a tag field indicating which entry to write into. Reading out of the RBUSQ is in a sequential fashion, meaning that data must be popped in the order of issuance of read commands. If a piece of data required by the formatter is not in, the formatter will be frozen. Data from the RBUSQ is read onto the ABUS through the AMUX.

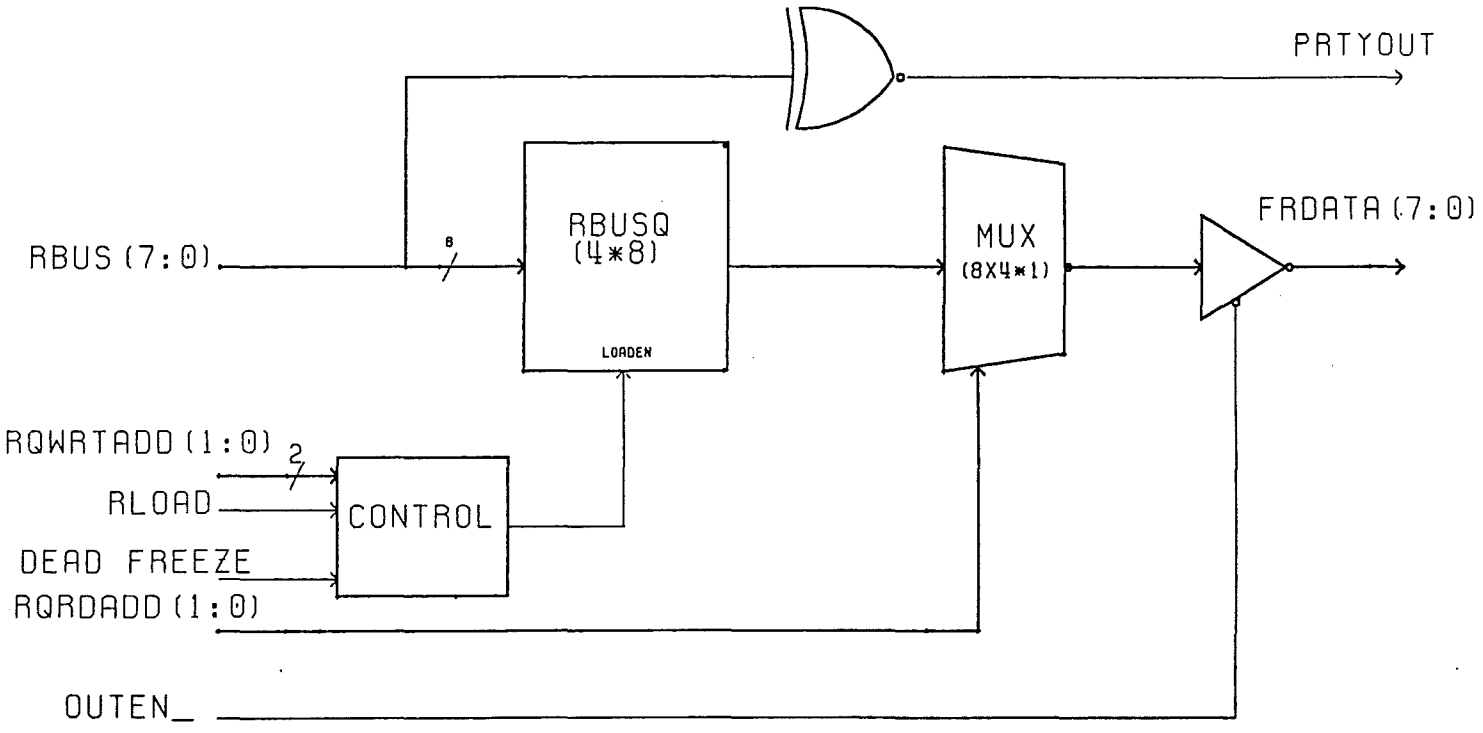


FIGURE 6-15 RBUSQ BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

COMPARATOR

The comparator in the OF performs a 10 digit comparison of A-bus and B-bus data at every clock. Each SCHPAD array slice performs a one digit comparison and the results are combined in a second level comparator located in the OFFCTL array to generate the overall greater than or equal result.

ADDER

The Adder in the OF performs only BCD operations. It subtracts BCD numbers by adding the 10's complement of the subtrahend. When performing index add, the sign digit (8th) is used to determine if a subtraction is needed instead of an addition. During subtraction, the 9's complement of the subtrahend is sent to the BCD adder together with the carry in. Also, the carry out from the adder is conditioned with the add/subtract bit to generate the overflow/underflow exception. The result of the adder is latched onto AUREG at the end of the clock. Based on the ALU/PMUX command, the AUREG will concatenate the 7-digit adder output with various patterns of the top 3 digits.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

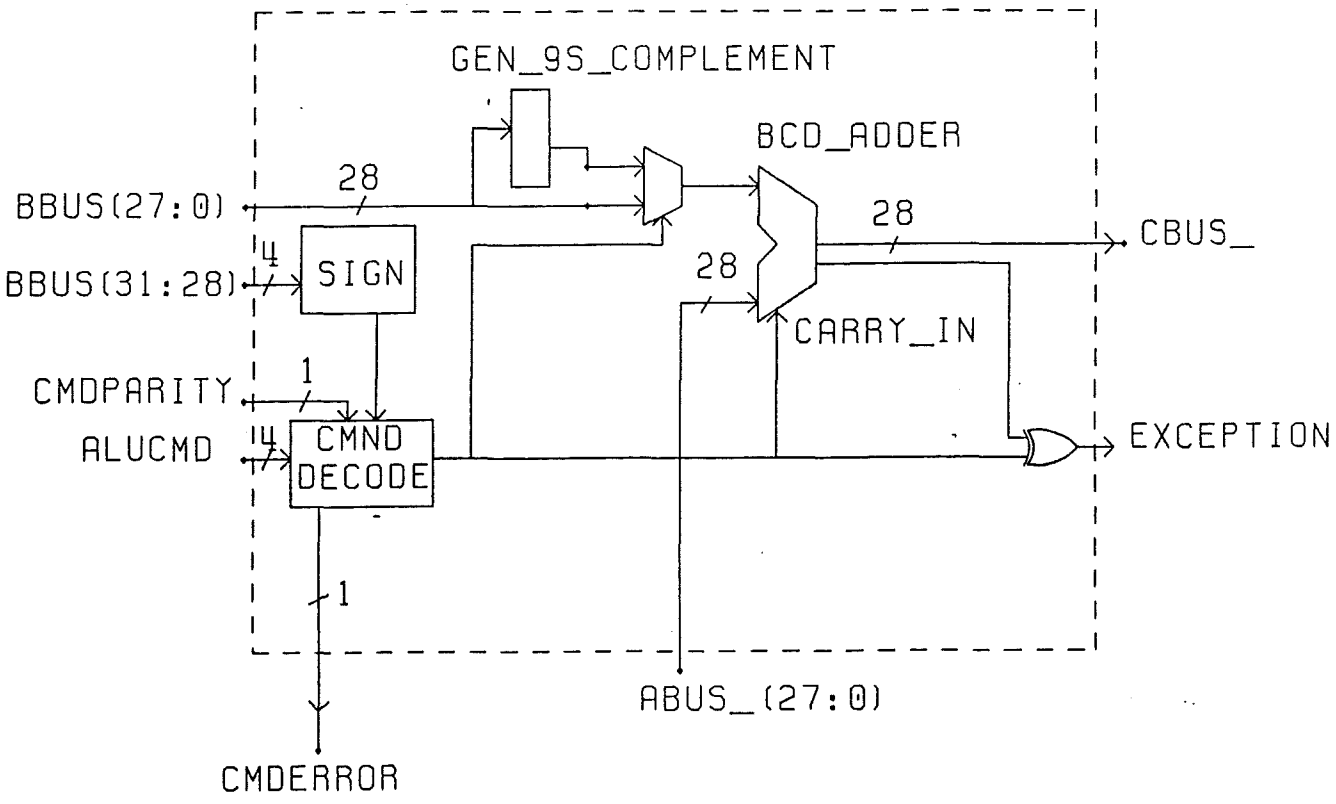


FIGURE 6-16 BCDADDER BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

PMUX

The PMUX is a programmable multiplexor/rotator which when combined with the PASS/CONCAT unit allows various required data patterns ton be set up onto the result bus, CBUS. The PMUX is controlled by the CREG (Control store Register) and rotates on a digit basis. It has 40 bits of input and output with a total of 16 functions.

One full gate array is dedicated to this function which consists of a function select decoder and encoder to drive the 10 to 1 mux dedicated to every digit. There is also a word parity prediction logic which is compared against the word parity generator on the output of the PMUX to detect any possible errors in the data path as shown below.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

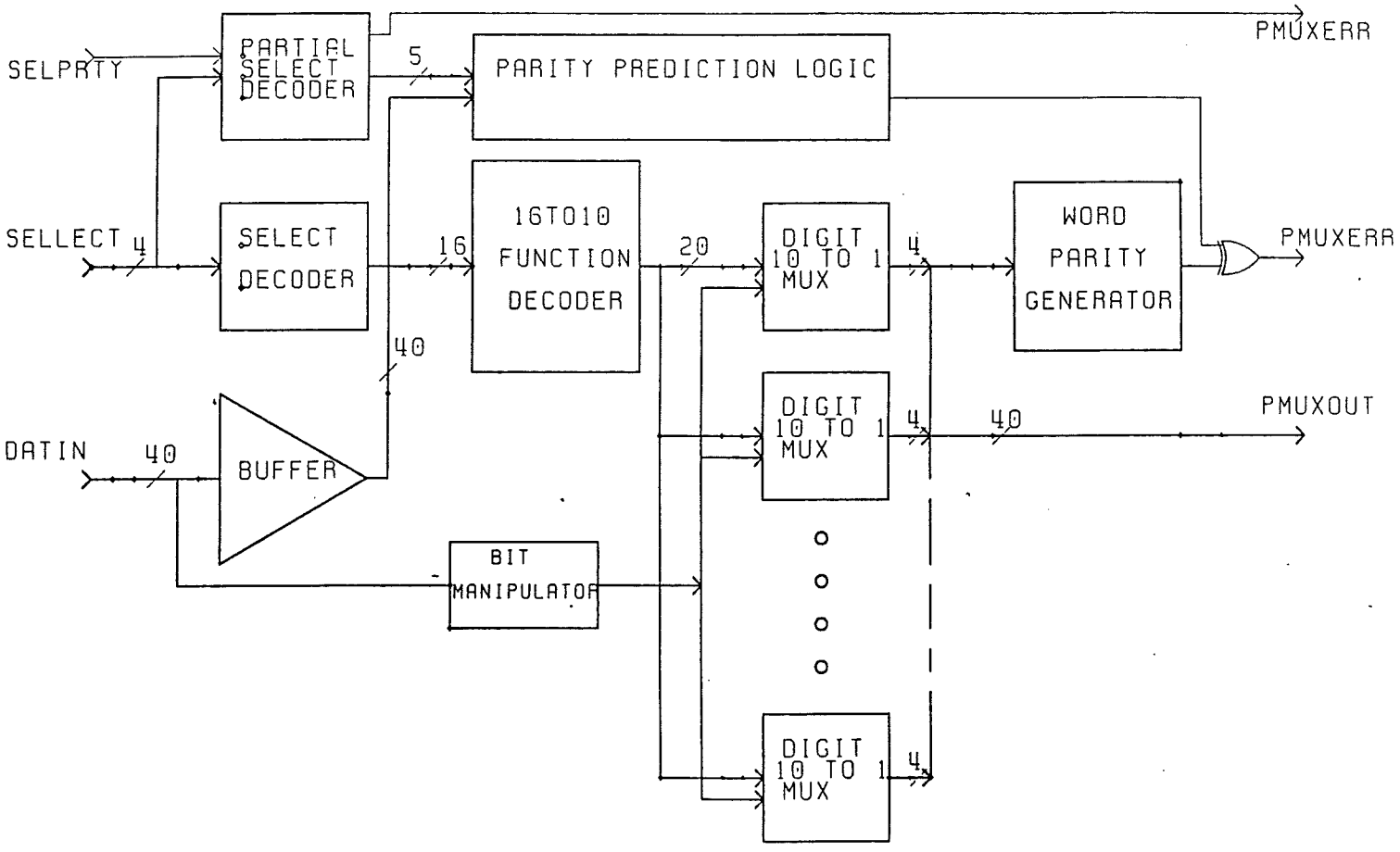


FIGURE 6-17 PMUX BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

PTEST

The PTEST logic in OF is comprised of 2 sections: one section performs combinatorial processing for data on the ABUS while the other section operates on error tags received from both the IF and MCACM. It also receives inputs from other portions of the system indicating whether the BBUS contains undigts (digit 9, digit 6-0). Based on the information received, it will set a BBUS undigit error flip flop. The section that operates on ABUS is selected by the CREG, data on the ABUS is examined and the result latched into PTREG. The entire PTREG or part of it can be used as branch conditions if required. The other section monitors error tags from IF and RBUS for setting appropriate error conditions. The ABUS analyzing section is actually implemented in 2 parts: the first 2 PTEST functions will be implemented by RAMs while the other 14 PTEST functions will be implemented in MCA2 gate array. The detailed descriptions of each PTEST function are as follows:

PTEST FUNCTION

```
0  :
     function: reserved for soft ptest

1  :
     function: reserved for soft ptest

2  :
     function: to perform address decoding, IA, IX, EXTENSION
     input   : 40 bits of ABUS [(top 2 digits) 9 8 )]
     output  : IF NOT EXTENDED
                       PTREG[4]  =0
                       PTREG[3]  =1  IF IA
                                 =0  IF NOT IA
                       PTREG[2-0]=INDEX REGISTER
```

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

```
                    ELSE  (EXTENDED)
                           PTREG[4]  =1
                           PTREG[3]  =0
                           PTREG[2]  =000  EXTENDED BY C, NO IX NOR
                                           IA
                                     =001  (EXTENDED BY E, IX1)  OR
                                           (EXTENDED BY C, IX OR
                                           IA)
                                     =010  EXTENDED BY D, NO IX NOR
                                           IA
                                     =011  EXTENDED BY D, IX OR IA
                                     =100  EXTENDED BY A
                                     =101  EXTENDED BY B
                                     =110  EXTENDED BY E, NO IX1
                                           SPECIFIED
                                     =111  EXTENDED BY F

    3  : AUX ADDRESS DECODE1
         function: to perform auxillary  addresss decode using FE
         digits
         input   : FE digits
         outut   : PTREG[4]  =0 IF NOT EXTENDED
                   PTREG[3]  =0 IF NOT IA
                             =1 IF IA
                   PTREG[2-0]=IX
                   PTREG[4]  =1 IF EXTENDED
                   PTREG[3]  =1 IF A EXTENDED
                   PTREG[2]  =1 IF B EXTENDED
                   PTREG[1]  =1 IF F EXTENDED
                   PTREG[0]  =1 IF E EXTENDED WITH INVALID IX

    4  : ADDRESS DECODE IGNORE EXTENTION
         function : address decode ignoring extension
         input    : 40 bits of ABUS [(top 2 digits) 9 8]
         output   : PTREG[4]  =0
                    PTREG[3]  =0 IF NOT IA
                              =1 IF IA
                    PTREG[2-0]=IX1-IX7
```

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

5 :CLEAR PTREG
function : to clear PTREG to zero.

6 :LIX DECODE1
function : to indicate what register will LIX affect and what dimensional override digit need to be inserted.
input : 40 bits of ABUS [XXXXOP AF BF]
output : PTREG[4] =0
PTREG[3] =1
PTREG[2-0]=DIMENSION OVERRIDE VALUE TO BE INSERTED

7 :LIX DECODE2
function : to indicate what register will LIX, SIX affected
input : 40 bits of ABUS [XXXXOF AF BF]
output : PTREG[4] =0
PTREG[3] =0
PTREG[2-0]=BFM (MOST SIGNIFICANT DIGIT OF BF)

8 :LOAD PTREG
function : load the last 5 bits of Abus data
input : 40 bits of ABUS
output : PTREG[4-0]=ABUS[4-0]

9 :ALEN DECODE
function : decode operand A length
input : ABUS[15:8]
OUTPUT : PTREG[4] =0
PTREG[3] =0
PTREG[2-0]=1 IF ABUS[15:8]=A1
=2 IF ABUS[15:8]=A2,A9,B1
=3 IF ABUS[15:8]=A3,AA
=4 IF ABUS[15:8]=A4,AB,B2
=5 IF ABUS[15:8]=A5,AC
=6 IF ABUS[15:8]=A6,AD,B3

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

10 : .1 UNDIGIT LOWER , .2 INDEX TEST , .3 UNDIGIT TEST
function : .1 detect undigit on 7 digit only
.2 to detect sign and undigit on index register read from memory
.3 to detect undigit across all 10 digits
input : 40 bit of Adata .1 [XBDDDDDDXX]
.2 [BXSDDDDDDD]
.3 [DDDDDDDDDD]
output : .1 PTREG[4] = IF ANY D'S HAS UNDIGIT or B >= 8
.2 PTREG[2] = 0 NO UNDIGIT ON D'S & B
= 1 UNDIGIT ON D'S OR B
PTREG[3] = 0 SIGN DIGIT IS POSITIVE
= 1 SIGN DIGIT IS NEGATIVE
PTREG[1] = 0
.3 PTREG[0] = 0 NO UNDIGIT ON D'S
= 1 UNDIGIT ON D'S

11 : T.B.S.

12 : INDIRECT FIELD LENGTH DECODE
function : to decode indirect field length and detect invalid INFL or invalid field length (literal is considered invalid when there is INFL)
input : Abus[39:8]

The following PTREG definitions are used for testing returned field length because of indirection and original BF.

output : PTREG[4]=1 IF LITERAL IS SPECIFIED
=0 IF NOT LITERAL
PTREG[1]=1 IF INFL
=0 IF NO INFL
PTREG[0]=1 IF INVALID FIELD LENGTH OR INVALID INFL OR LITERAL
=0 IF VALID FIELD LENGTH, INCLUDING VALID INFL

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

That is, if

PTREG[1-0] = 00 VALID FL & NO INFL
= 01 INVALID FL, NO INFL OR LITERAL
= 10 VALID INFL & INFL
= 11 INVALID INFL & INFL OR LITERAL

The remaining PTREG definitions are used for testing original AF.

PTREG[3-2]=00 IF VALID INFL
01 IF INVALID INFL
10 IF LITERAL
11 IF INVALID FIELD LENGTH

13 : T.B.S.

14 : LENGTH CALCULATE ASSIST

function : .1 determine the length of some of the ops e.g. MVW
.2 determine whether the environment number is zero
.3 stack frame decode

input : .1 40 bits of ABUS [XXXXXXAF BF]
.2 40 bits of ABUS [EEEEEE XXXX]
.3 40 bits of ABUS [SS XXXXXXXX]

output : .1 PTREG[0] =0 IF AFBF='0000'
=1 otherwise
PTREG[1] =0
.2 PTREG[4] =1 IF EEEEEE = '000000'
=0 otherwise
.3 PTREG[3-2]=00 if SS = 00-FC
=01 = FD
=10 = FE
=11 = FF

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

15 : PTREG HOLD
function : hold the content of PTREG

The following are functions that the PTEST performs every clock.

IF error detection :

function	:	to detect any error that occurs in IF phase		
input	:	SS digits from FETCH PAGE location 0 (activate when sourcing the PC syllable)		
output	:	IF-LIMIT-ERR-F/F	=1	if limit error occurs
		IF-UNDIGIT-ERR-F/F	=1	if undigit error occurs
		IF-ODD-PC-ERR	=1	if odd pc error occurs
		ADDRER	=1	if undigit or limit error

RBUSQ error detection :

function	:	to detect any error during reads while parsing		
input	:	ERROR TAG from RBUSQ control		
output	:	OF-LIMIT-ERR-F/F	=1	if limit error occurs
		OF-UNDIGIT-ERR-F/F	=1	if undigit error occurs

Other error detection :

function	:	to detect error on both operand buses (A & B)		
input	:	Undigit BBUS signal from OFTEST array PTREG-0		
output	:	UNDGAB-F/F	=1	if PTREG-0=1 or undigit on BBUS
		ADRERR-UNDGAB	=1	if UNDGAB or ADRERR

All of the error flip flops will be reset by SHOVE, system flush or fetch flush.

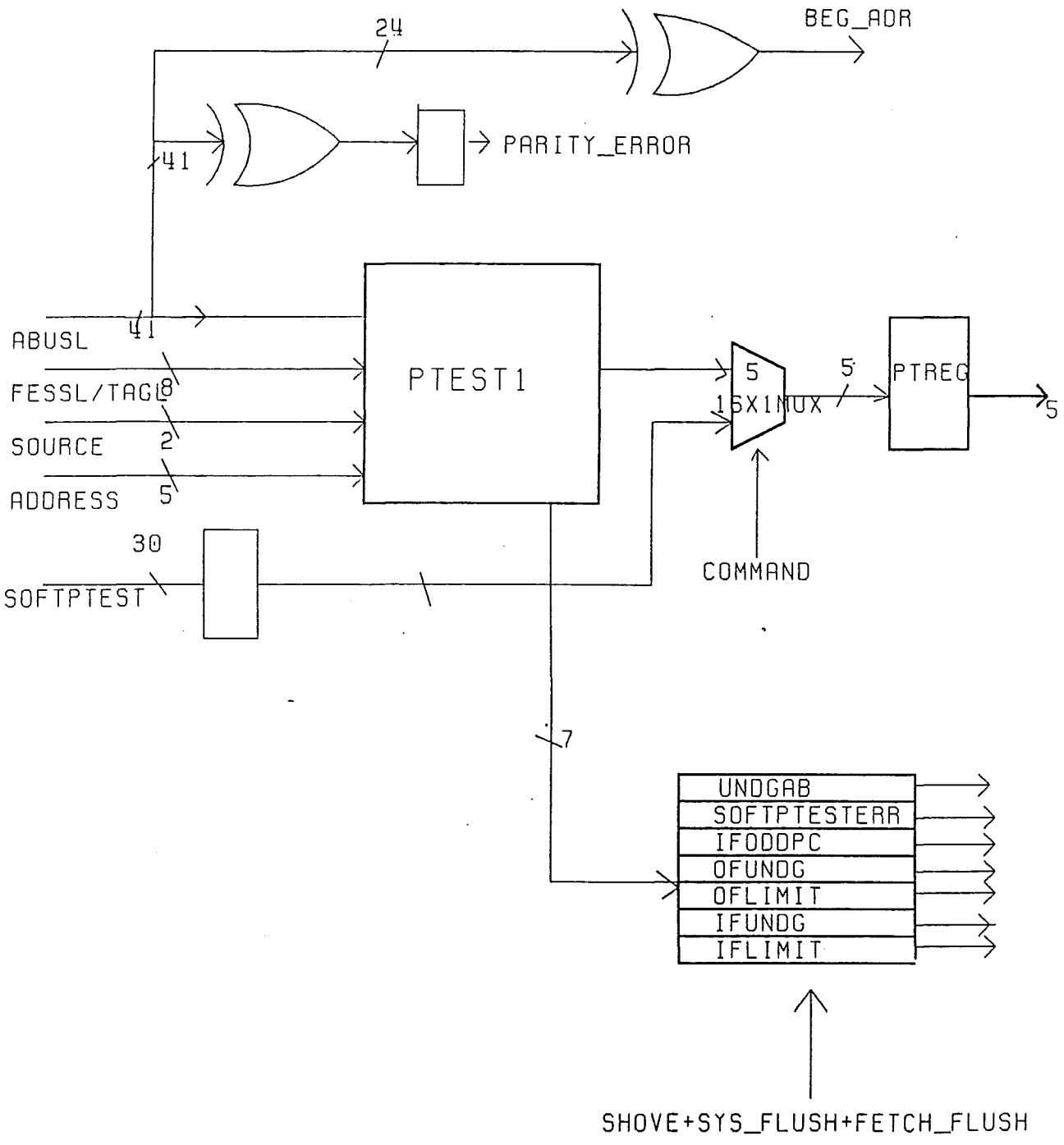


FIGURE 6-18 HARD PTEST MODULE

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

FBUS INTERFACE

The OF sends information to the XM via FBUS1 (FBUS1$$$P) and FBUS2 (FBUS$$$$P). OF has highest priority on FBUS1 (FBUS1$$$P) and has exclusive use of FBUS2 (FBUS$$$$P). FBUS1 (FBUS1$$$P) and FBUS2 (FBUS$$$$P) are independent of each other, each with its own set of controls and addresses. There are two signals, FLOAD1 (FBUS1LOAD$$P) and FLOAD2 (FBUS2LOAD$$P), to indicate whether data on that bus is valid and should be latched. OF reads from one page during address resolution and operand preread while all transactions on the FBUS1 (FBUS1$$$P) from IF to XM are written into another page.

When OF is writing on the FBUS1 (FBUS1$$$P), only portions of the full FBUS1 (FBUS1$$$P) interface will be used. The portions actively used by OF are as follows:

FPARITY1(0)	- combined even parity across FBUS1 (FBUS1$$$P) (39:0)
FBUS1 (FBUS1$$$P) (39:0)	- fetch page data
FPADDR1 (FBUS1PAG$P) (1:0)	- XM fetch queue address
FADDR1 (FBUS1ADR$P) (2:0)	- fetch page entry address
FLOAD1 (FBUS1LOAD$$P)	- load signal
FCPARITY (FBUS1CMDPARP	- combined even parity across FPADDR1 (FBUS1PAG$P) (1:0), FADDR1 (FBUS1ADR$P) (2:0), FLOAD1 (FBUS1LOAD$$P) and OFREQ (FO$FBUS$REQP).
FPARITY1 (FIPARXDIG$$P) (3:1)	- don't care
FBUS1 (FBUS1$$$P) (52:40)	- don't care
OFREQ (FO$FBUS$REQP)	- signal to notify IF that OF is using FBus1

The format of interface on FBUS2 (FBUS$$$$P) are as follows:

FBUS2 (FBUS$$$$P) (9:0)	- fetch page data
FPADDR2 (FBUS2PAGP$P) (1:0)	- XM fetch queue address
FADDR2 (FBUS2ADR$P) (2:0)	- fetch page entry address

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

FLOAD2 (FBUS2LOAD$$P)	- load signal
FPARITY2 (FBUS2PARITYP)	- combined even parity across FBUS2 (FBUS2$$$$P) (9:0)
FCPARITY2 (FBUS2CMDPARP)	- combined even parity across FPADDR2 (FBUS2PAGP$P) (1:0), FADDR2 (FBUS2ADR$P) (2:0), FLOAD2 (FBUS2LOAD$$P), SHOVE (FSHOVE$$$$$P), DATAHIT (FDATAHIT$$$P), and OPTIMAL (FOPTIMAL$$$P).

Also related to the FBUS2 (FBUS$$$$P) interface format are the three signals: SHOVE (FSHOVE$$$$$P), Data-hit (FDATAHIT$$$P) and Optimal (FOPTIMAL$$$P). When SHOVE (FSHOVE$$$$P) is enabled, the FPADDR2 (FBUS2PAG$P) will contain the XM fetch queue page address for the three signals; SHOVE (FSHOVE$$$$P), Data-hit (FDATAHIT$$$$P) and Optimal (FOPTIMAL$$$P) to be captured.

AddrBUS INTERFACE

The formatter sends out read/write addresses through the AddrBUS (XADRBUS$P). There are 9 digits of address and 1 digit of base indicant. In addition there are 7 bits of tag, 5 bits of command and 4 bits of length associated with this bus. For fault detection purposes, this bus also contains three bits of parity; one for the TAG field (XWBUSTAG$P), one for the LENgth (XWBUSLEN$P) and COMMAND (XADRBUS$P) fields and one for the ADDRess (XADRBUS$P) itself (note; 59 bits altogether). When a command is issued, the parity bit accross the write bus must be forced to low. Since OF is not driving the WBUS (XWRTBUS$P), it will float to 0. Since the XM or the coprocessor or MCACM have higher priority over the OF on the AddrBUS (XADRBUS$P), OF will be frozen if it tries to send out a second request while the first is held up due to contention between OF and those modules.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

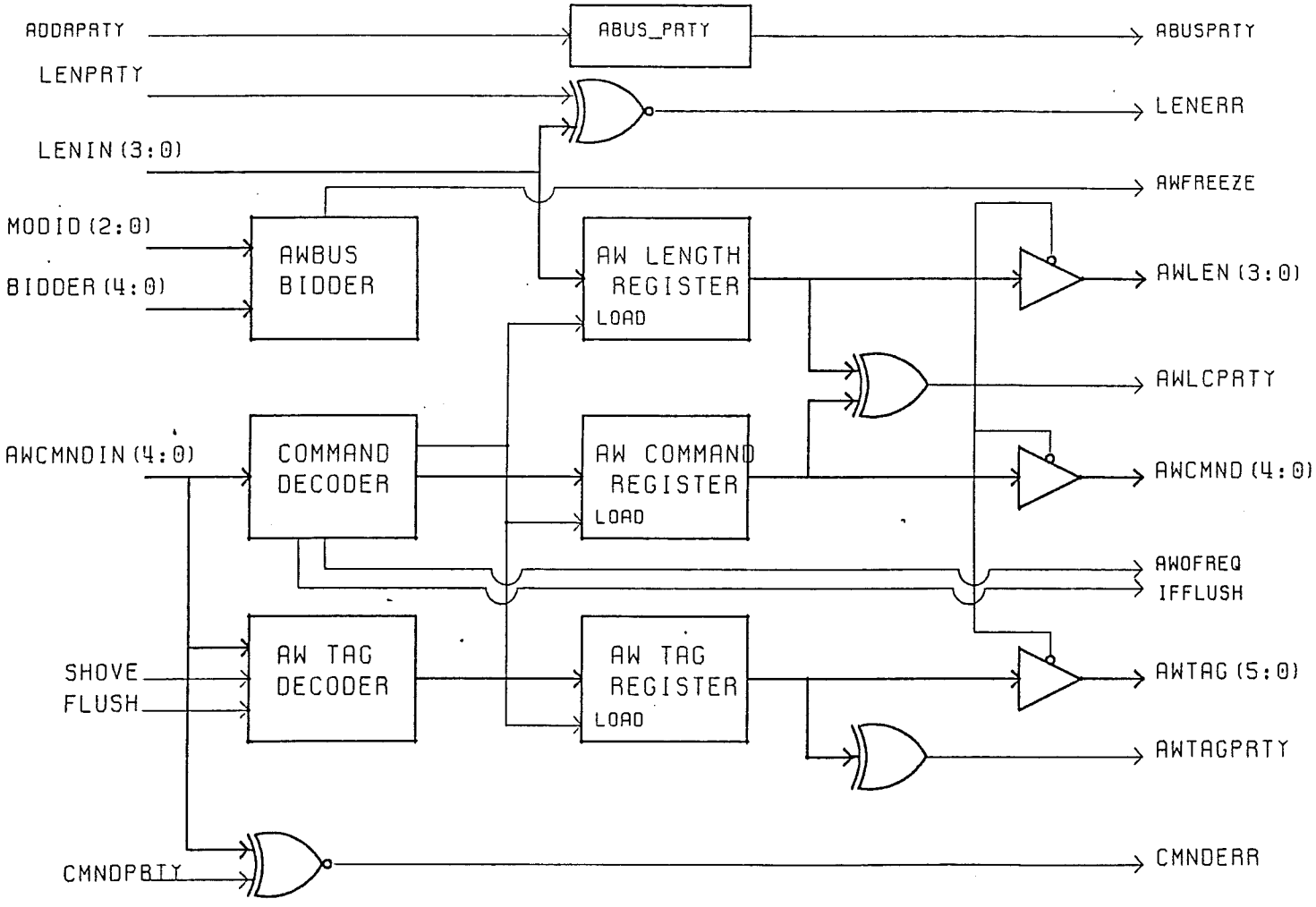


FIGURE 6-19 ADDRBUS INTERFACE BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

RBUS INTERFACE

OF receives data coming back from memory via the RBUS (MRBUS$$P) interface. Associated with the data, there will be a DAT_VLD (MRBUSVALID$P) bit which indicates the validity of the data (note: data will be discarded if this bit is off) and a combined odd parity bit. A tag field informs the RBUS (MRBUS$$P) control to latch data on the RBUS (MRBUS$$P) the following clock. These are the same bits that were generated by the AddrBUS interface (XADRBUS$P). There are 2 bits in the tag field to specify which one of the 4 locations the data is to be latched into. The RBUS (MRBUS$$P) control keeps track of the read pointer and the validity of each entry. A read can only occur if the location indicated by the read pointer is valid, otherwise, OF will live freeze.

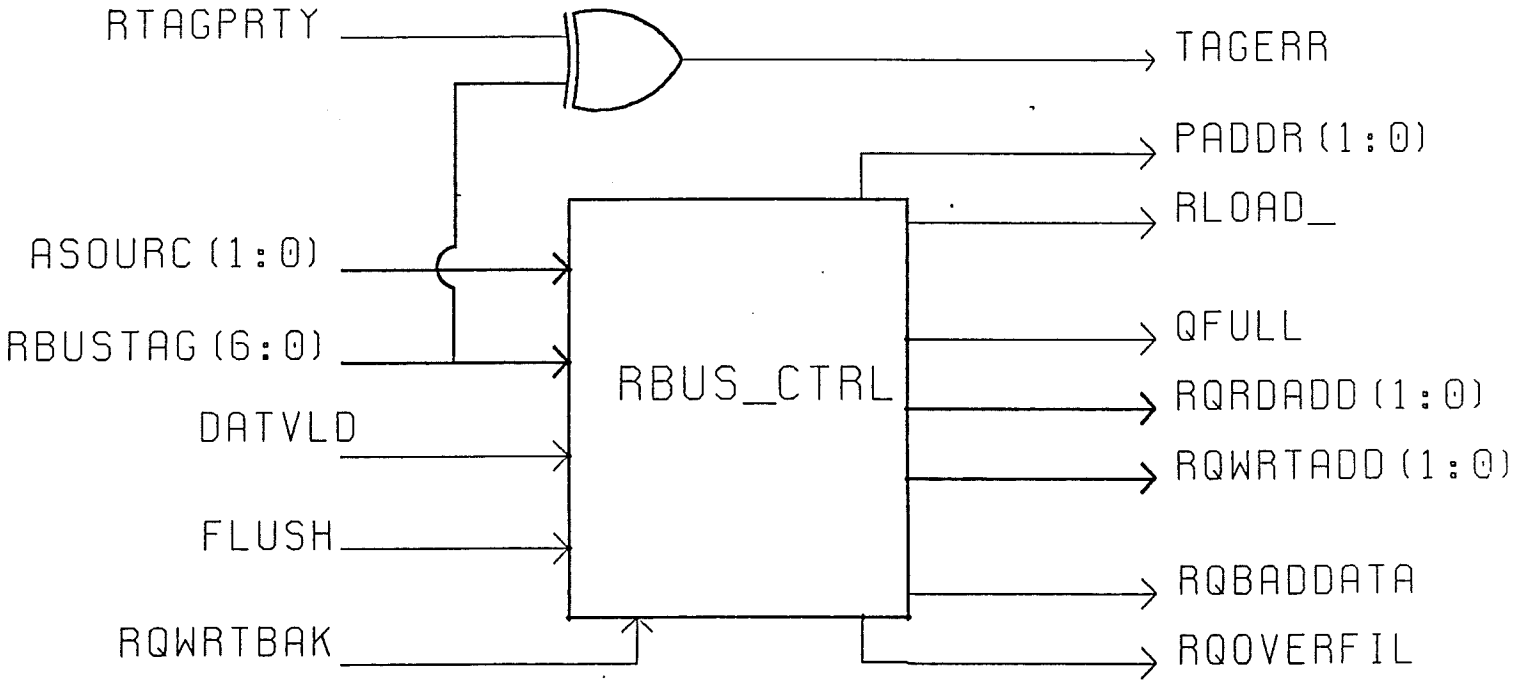


FIGURE 6-20 RBUS INTERFACE BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

OPLENGTH UNIT

The Op Decode and Length Unit has two main functions:

1. It runs the OP-Code through a Look-up table to produce a 7 bit code which, together with an 8th bit signifying whether or not the AF indicates a Literal, forms the OPLIT which the XM uses in the first clock of execution to branch directly into the correct code.

2. The OP-Code is encoded in the FCODE Look-up table, this code is combined with the AF, BF, and the appropriate Controller bits (designating UN, SN or UA data types) to produce the actual data length in digits and various flag condtions.

Interfaces

The OP Length unit is designed to interface to the Operand Fetch module. To do this, it is comprised of two array options, the Oplen input array and the Oplen output array. The input array will contain two pages of OPAFBF, Ac/Ae, Bc/Be, Cc/Ce the will serve as multiplex the A length, B length, C length, OPLIT, and flags to the B-bus and the FBUS2.

The Operand Fetch requires certain conditions to branch on from the Op Length unit. These signals are AF>BF, AF=BF, Ac, Bc, Cc and the number of extended syllables. The Optimal condition (Alen<=10 & Blen<=10) is driven out to the XM and to the OP micro sequencer.

Hardware Implementation

The Op Length Unit will be comporised of three MCAII gate array, two Oplen input arrays and one Oplen output array, and the following number of rams:

3 - 256 x 4

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

2 - 4K x 1
19 - 1K x 4

Three 256 x 4 rams and one 1K x 4 ram will make up the Fetch Look-up Table. The rest of the 1K x 4 rams will calculate the A_length, B-length, C-length while generating literal and optimal conditions.

The output selection from the OP Length unit will be as follows:

The B address will select OPLIT, Alen, Blen, or Clen to the Bbus which will be enabled on the bus by FLOAD2. FBUS2SELECT will select a specified length, OPLIT, or FLAG output to the Fbus2 (FBUS2$$$$P).

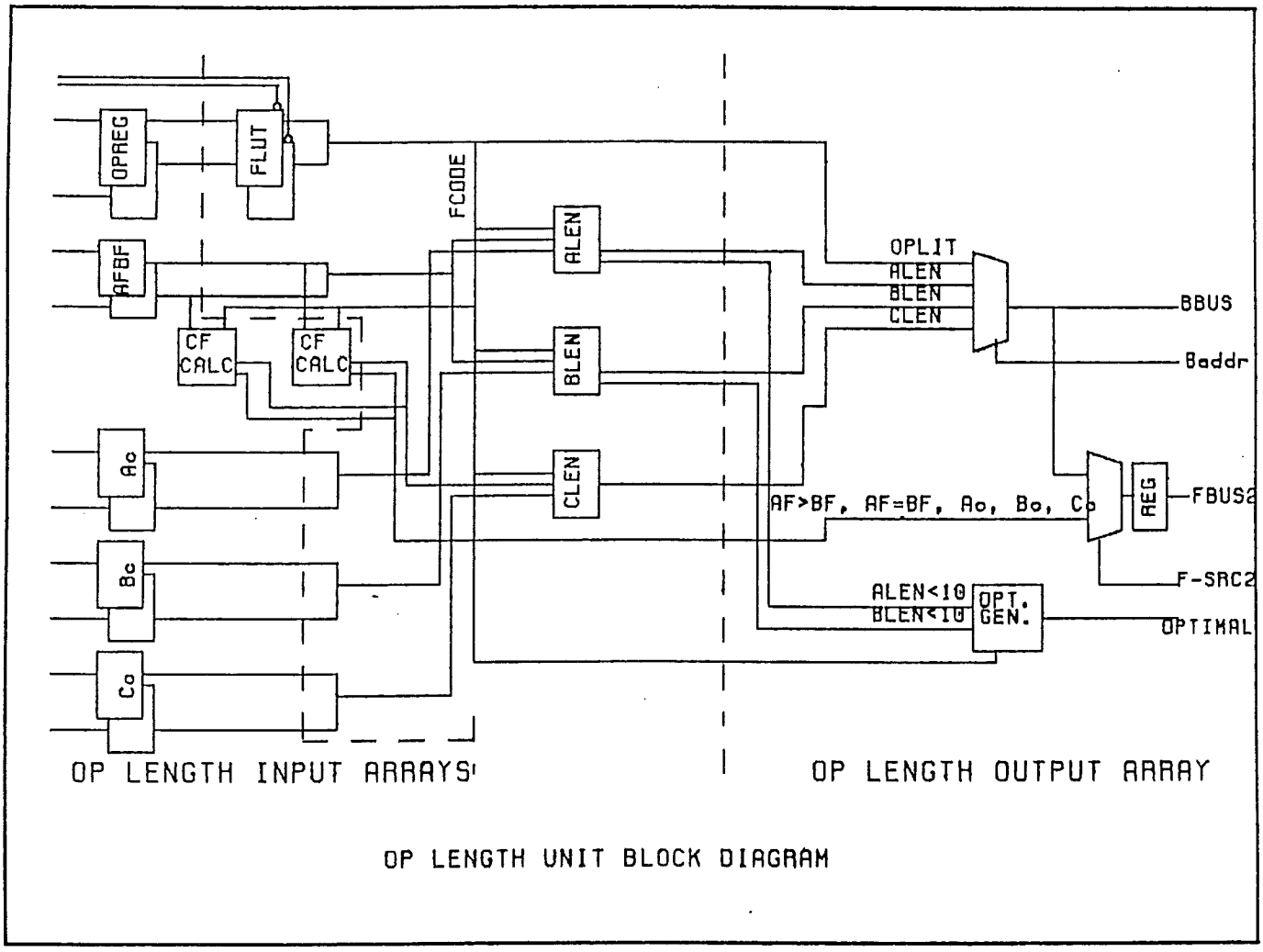
Functional Description

The FLUT being addressed by the OP will generate the OPLIT for the XM while also producing an FCODE. The FCODE is an encoded field for the Alen, Blen and optimal generators. The FCODE for the length calculators selects one of two banks of rams for each syllable. It also determines how literal and optimal is to be calculated. The FCODE will serve as a selector for the CF calculator by selecting AF, larger of AF/BF, AF-BF, or AF+BF to be used for Clen calculation.

The AFBF will be used as an input to the appropriate length generator. It will also feed the CF calculator for CF, AF>BF, and AF=BF generation. As an input to the Alen and Blen generators, it is used to provide the read address should there be an indirect field length. This is accomplished by masking out the 2 most significant bits of the field lengths, AF(for Alen) or BF(for Blen).

The Ac, Bc, and Cc are used as inputs to the length generators to signify the type of data. The Ae, Be, and Ce are combined with the FCODE to produce the number of extended syllables flag.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)



OP LENGTH UNIT BLOCK DIAGRAM

FIGURE 6-21 OPLEN UNIT BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

Assignment of FLUT (Fetch Look Up Table):

- o bit 15 - LITMASK
  When set, allows literal bit to be set/reset as specified by the spare bit.
- o bit 14 - spare
  Used to set (=1), reset (=0) literal bit when lit mask is active.
- o bit 13:12 - FLUT
  Used to control setting of optimal bit:
  = 0 reset optimal
  = 1 set on ALEN < or = to 10
  = 2 set on ALEN, BLEN < or = to 10
  = 3 set optimal
- o bit 11 - FLUTPARITY
  Even parity across FLUT RAM outputs
- o bit 10:4 - OPLIT
  OPLIT vector for XM (does not include literal bit)
- o bit 3 - ALENCODE
  Used to control ALEN, BLEN calculation:
  = 0 if AF & BF are used separately
  = 1 if AFBF is used to generate length
  e.g. MVW, MVC
- o bit 2 - OPITPRTY
  Even parity for OPLIT
- o bit 1:0 - CLEN calculation code:
  0 - pass AF as CF.
  1 - larger of AF or BF as CF.
  2 - sum of AF, BF as CF.
  3 - difference of AF from BF as CF.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

The ALEN(BLEN) RAMs contents are as follows:

- bit 11 - bit to generate even parity across ALEN(BLEN) RAM outputs
- bit 10 - 1 if A-operand(B-operand) <= 10 else 0
- bit 9:0 - length of A-operand(B-operand) in digits

The CLEN RAMs contents are as follows:

- bit 11 - bit to generate even parity across CLEN RAM output
- bit 10:0 - length of C-operand in digits

Timing & RAM Loading

The following timing constraints must be observed in using the Oplength lookup unit:

Changing OP:	Oplit available on next clock. ALEN, BLEN available one clock later. CLEN available two clocks later. FLAGs available two clocks later.
Changing AF,BF:	ALEN, BLEN available one clock later. CLEN available two clocks later. FLAGs available one clock later.
After OPLIT branch:	OPLIT available immediately. ALEN, BLEN available one clock later. CLEN available two clocks later. FLAGs available one clock later.

Loading the FLUT, Alen, Blen, and Clen rams will be done using BREG in the FANDRQ array as the DATA input with the LENIN array providing the address. The WRITE ENABLE for the specific ram group is handled in the OPLOUT array.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

LOCK UNIT

The purpose of the lock unit is to prevent the accessing of data from an area of memory until all pending writes to the area are completed.

The lock mechanism maintains a table of pending writes in the form of begin address and end address pairs. A test is made for access to a data Area in the form of begin address and length by comparing this data area to the pending writes contained in the table. The detection of an overlap by these tests will cause hit signals to be sent to the operand fetch unit ,and these signals will remain as long as the offending pending write entries remains in the table and will be reset as soon as these entries are popped. An entry in the table will be deleted only after the associated write to memory has been completed and enough code checks have been performed against the lock. The lock unit may also perform tests against writing of index registers, IX1, IX2 and IX3.

Interfaces

The lock unit interfaces with the OF module in fetch and the XM module. The AREG provides the begin address to the lock unit and the AUREG provides the end address. The COMMAND bits, ETIX bits, IXCACH bits and SHOVE all come from the oftest array. HICCUP (generated by FETCH_FLUSH (EFETFLVSHP) signal) is a signal that tells the lock unit if a bad branch has taken place and the signal FETCH_FLUSH (EFETFLVSHP) comes from the XM along with OPCOMP (EOPCOMPLETEP) and SYS_FLUSH (ESYSFLUSHP). ALL of the output signals from the lock unit are directed to the SEQUENCER in OF except for the DATHIT (FDATAHIT$$$P) bit which goes to the XM module via FBUS2 (FBUS2$$$$P).

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

Overall Structure

The lock table contains a table of eight (Begin/End Address) pairs of pending writes to memory. This table is organized as a queue which means that the first element written into this table will be the first one to be deleted from the table. The lock unit also maintains a hardwired table of the fixed relative addresses of IX1, IX2 and IX3 in the form of (Partial Begin/Partial End Address) pairs. The begin addresses contain 6 digits and the end addresses are six digits plus an additional bit for every entry refered to as the overflow. The hardwired addresses of the index table are each six bits wide, since the indexes are located in memory locations 000000008 thru 000000031, only 6 bits are required to address any one of the indexes by taking advantage of the fact that the rest of the bits will be zero. The control section of the lock unit is organized in one stage and can accept a new command on every clock. The instructions of the lock mechanism fall into two classes. A write lock command which creates a new entry in the pending write table and simultaneously compares this new element to each of the elements in the index table. A code check, index check or data check command which simultaneously compares an incoming element against all of the valid elements in the pending write table.

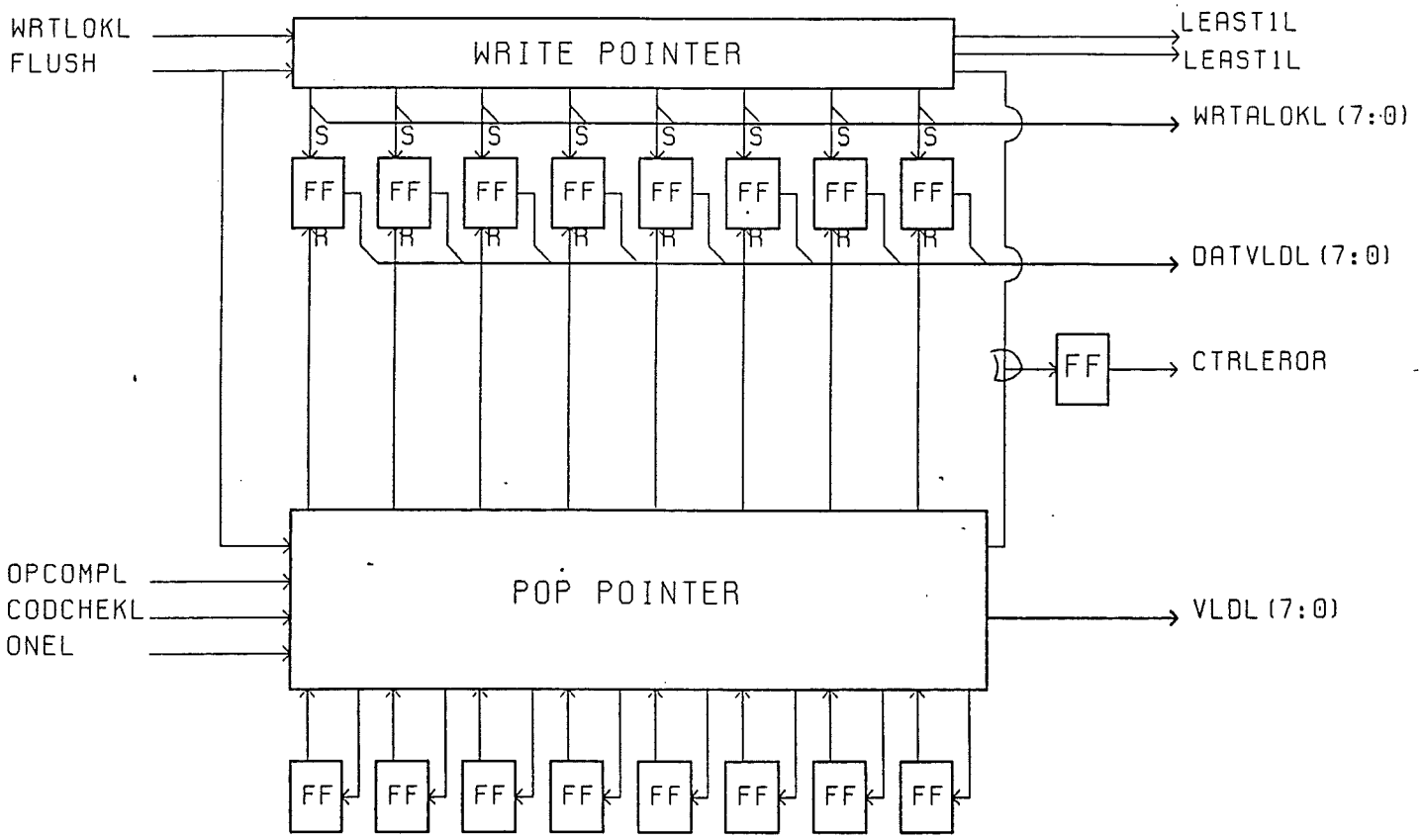


FIGURE 6-23 LOCK CONTROL LOGIC BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

The hardware logic consists of two stages. Stage I is basically a 6 digit BCD adder with a carry logic. The result of this addition is loaded in to the AUREG(note: refer to OF block diagram).

The second stage consists of four separate logic paths. The first and second paths are identical and they do comparisons against the begin and end addresses of the incoming element and the corresponding elements in the lock table. Hit signals are then sent to the control unit for further evaluations. Finally the last two paths perform a test against the begin and end addresses in the index table. The hardware of these paths will be further discussed in later sections.

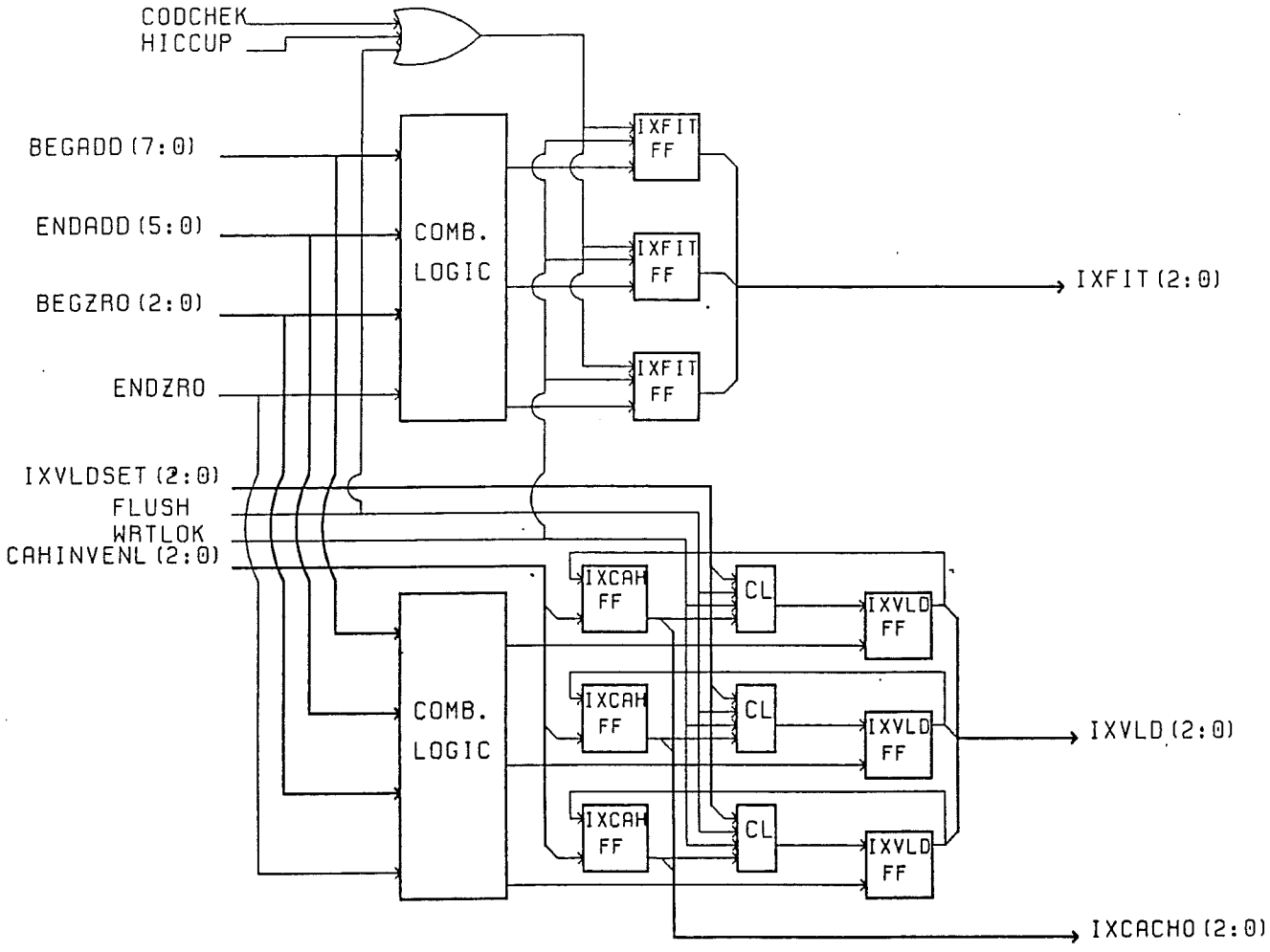


FIGURE 6-24 INDEX_CHECKING LOGIC BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

HARDWARE IMPLEMENTATION

The diagram on the following page explains how the hardware is implemented and the connectivity between the various modules within the lock unit. Due to a long data path all of the inputs to this module are registered, which in fact allows the AUREG to be used to store the result of the addition of the begin address and length.

Seven digit BCD adder

This adder is shared with the OF formatter. The output of this adder is loaded in to AUREG and then fed in to the lock unit. The inputs to the adder come from the A_BUS and B_BUS. For fault detection purposes, the adder is duplicated and the outputs are compared and the AUREG is covered by parity.

Lock comparators

This portion of the lock unit contains the necessary logic for a six digit begin address comparator and a 25 bit end address comparator. This comparison is used to indicate a possible hit within any base_limit address.

For fault detection purposes, part of the comparator logic which consists of Exclusive-or gates are covered by parity. In addition the two A=B and A>B outputs are compared against each other since both of them can not be active at the same time. The diagram below shows the lock comparators and how they are connected to the rest of the module.

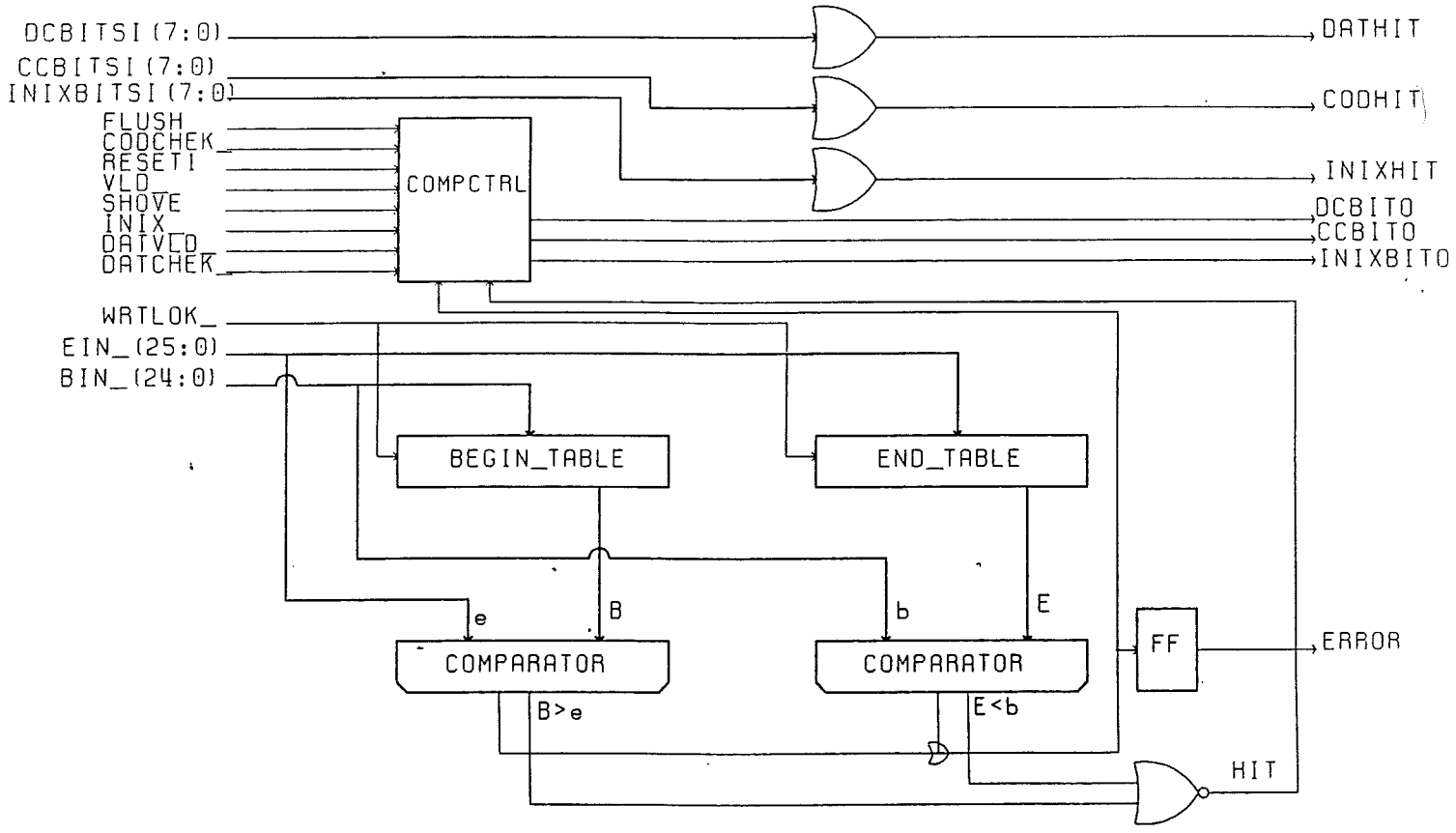


FIGURE 6-25 LOCK COMPARATORS BLOCK DIAGRAM

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

Lock table

As mentioned in previous sections, the lock table consists of eight pair of begin and end entries. These entries are implemented on registers and a parity bit is associated with each entry for fault detection. The above diagram shows how the lock tables are connnected to the reset of the module.

Gate array partitioning

As discussed before, MCA II Gate Arrays will be used to implement this design. Due to pin and cell limitations, ten arrays are required to do the job. The arrays are of three distinct designs as explained below . The first design performs the six digit decimal addition and it is shared with the OF formatter. The second design has been utilized mainly for control logic and contains the logic neccessary to prevent against writing into index registers and finally the third design consists of two six digit comparators and a portion of the lock table corresponding to this comparison. This design is copied into eight of the arrays, in which each array compares the begin and end addresses of the incoming element against one of the elements in the lock table. In addition to the above gate arrays, other storage elements such as the AUREG are needed to interface to the lock unit. The placement of these elements will be discussed later.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

WRITE LOCK OPERATION

This operation is performed for each write address sent to an XM's fetch page. Before posting a write lock, the operand fetch should first make sure that there is room in the lock table by checking the ATLEAST1L and ATLEAST2L signals. Also if the DATHIT bit is on (refer to DATA CHECK SECTION), this command will be ignored.

The operand fetch sends a 7 digit address and a 6 digit length to the bcd adder. An add operation will then take place and the result will be stored in the AUREG. Meanwhile, the begin address will be stored in the AREG and the all the commands corresponding to the lock unit will be latched.

The output of AUREG is then stored into a pending write end address location along with the possible carry bit. The output of AREG is also written into the corresponding begin address location. Also at this time the lock table control is updated to reflect this new entry in the lock table.

In the second stage of the operation, the output of the addition is compared against all three begin addresses in the index table, as the unchanged address is compared against all three end addresses in the index table. The results of these six partial comparisons are then used to determine if this pending write has invalidated any of the indexes stored in the index register cache. If the pending write interval intersects any of the index register address intervals, then the coresponding index register OK flip flop will be reset along with the corresponding IX_CACHE bits. The setting of these flip flops is under the control of the operand fetch. Also, if the pending write fits exactly into one of the index register locations, the corresponding IXFIT bit will be turned on.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

CODE CHECK OPERATION

This check is made for each instruction that is sent to an XM's fetch queue page and it will be performed against all valid entries in the lock unit. A code check request also clears all IXFIT bits.

The operand fetch sends a 7 digit address and a 6 digit length to the bcd adder. An add operation will then take place and the result will be stored in the AUREG. Meanwhile, the begin address will be stored in the AREG and the all the commands corresponding to the lock unit will be latched.

The following comparisons are then performed simultaneously:

1) The result of the addition is compared against the six least significant digits of all begin addresses in the lock table.

2) The six least significant digits of the incoming address are compared against all end addresses in the lock table.

The results of all comparisons are then combined and will be used to set one or more of the internal code bad flip flops.These flip flops correspond to the eight elements in the pending write table. The corresponding flip flop will be turned on if an element in the lock table has a non empty intersection with the address interval under test. Once turned on, these flip flops will remain on until the corresponding element in the write table has been deleted. A code hit signal is sent to the operand fetch if any of the internal code bad flip flops are on or are about to be turned on. This signal is cleared by an explicit microcode branch, after op complete has been received for the last lock detecting a hit. It is then up to microcode to issue three code checks before sending out a NEWPC command to the

current PC. In addition , the lock unit may pop a particular lock entry, only when op complete has already been posted and this is the third code check request after op complete has been posted for this entry.

INDEX/INDIRECT CHECK OPERATION

This check is made when the operand fetch requires an indirect address or index register content from memory and it is performed against all entries that have no corresponding op complete.

The operand fetch sends a 7 digit address and a 6 digit length to the bcd adder. An add operation will then take place and the result will be stored in the AUREG. Meanwhile, the begin address will be stored in the AREG and the all the commands corresponding to the lock unit will be latched.

The following comparisons are then performed simultaneously:

1) The result of the addition is compared against the begin addresses in the lock table.

2) The six least significant digits of the incoming address are compared against all partial end addresses in the lock table.

The results of the comparisons are then combined and will be used to set one or more inix_bad flip flops. These flip flops correspond to the eight elements in the pending write table. The corresponding flip flop will be turned on if an element in the lock table has a non empty intersection with the address interval under test. An inix hit signal will be sent to the operand fetch if any of the internal hit signals are turned on or are about to be turned on. Once turned on, these flip flops will remain on until an op complete is received for the particular lock entry which detected a hit. The INIXHIT flip flop will be

reset if all of the internal flip flops are off, otherwise this flip flop will be set.

DATA CHECK OPERATION

This check is made for each instruction that is sent to an XM's fetch page, since the operand fetch will be performing operand pre_read and it is performed against all entries that have no corresponding op complete.

The operand fetch sends a 7 digit address and a 6 digit length to the bcd adder. An add operation will then take place and the result will be stored in the AUREG. Meanwhile, the begin address will be stored in the AREG and the all the commands corresponding to the lock unit will be latched.

The following comparisons are then performed simultaneously:

1) The result of the addition is compared against the six least significant digits of all begin addresses in the lock table.

2) The six least significant digits of the incoming address are compared against all end addresses in the lock table.

The results of the four comparisons are then combined and will be used to set one or more internal data bad flip flops. These flip flops correspond to the eight entries in the lock table. The corresponding flip flop will turn on if an element in the lock table has a non empty intersection with the address interval under test. A data hit signal is sent to the operand fetch in case of an overlap. Once turned on, these flip flops will remain on until a SHOVE command appears.

6.2.1.5 OF FORMATTER STRUCTURE (Continued)

EFFECTS OF PIPE CLEAR ON LOCK UNIT

The following actions will take place when a flush signal appears:

1) All locks are cleared,
2) All data hit, code hit and in/ix hit flip flops are reset,
3) IXHIT and IXFIT flip flops are reset,
4) IX_CACHE bits are cleared,
5) And the write pointer is updated.

The following actions will take place when the HICCUP signal appears:

1) All locks are cleared,
2) All data hit, code hit and in/ix hit flip flops are reset,
3) IX_OK flip flops are masked with the corresponding IX_CACHE bits,
4) IXFIT flip flops are reset,
5) IX_CACHE bits are cleared,
6) And the write pointer is updated.

6.2.1.6 CONTROL STRUCTURE

The Formatter (Operand Fetch Data Path) is controlled by a micro sequencer, driven by 12-16 K of microcode (80 bits wide). Besides the normal GOTO branching, the micro sequencer has 2-, 4-, 8-, 16-, 32-way branches. It also has the ability to call and push an arbitrary return address onto the stack. Using an additional stage of mapping ram, the micro sequencer can provide the microword corresponding to the next op immediately following the last microword of the previous op.

6.2.1.6 CONTROL STRUCTURE (Continued)

OPCODE LOOK-UP STRUCTURE

Since IF and OF are executing in a pipeline fashion, IF will always be ahead of the OF by 1 instruction, except at startup or after a pipe flush. Taking advantage of this, OF has a look-up structure to allow some preprocessing before IF flips. Information gathered by the look-up unit is used as input to a mapping ram to generate an entry address for the micro sequencer. By the time IF flips, the microword corresponding to that entry address will be ready to be loaded onto the CREG, thus, eliminates the need to wait. Furthermore, the microword produced 'knows' what need to be done. For example, the ram will prevent a BF-exception to be generated for a BOT instruction.

Not all of the information pertaining to the generation of an entry address is fed into the mapping ram because of 2 reasons. First, information from the IF arrive at different times and some of them do not have enough time to go into the ram. Secondly, the size of the mapping ram need to be kept reasonably small. Therefore, only the opcode, A-literal bit, AF-exception and BF-exception are used as inputs (11 bits) to the ram. The output of the mapping ram is 14 bits wide allowing for 16 K of microcode. A-, B-, C- exception flags provided by the IF are muxed with 3 bits from the ram. This scheme allows the entry address to be located anywhere if an instruction does not have address syllables. For 1,2,3 address ops, the entry address must be modulo 16. To allow the lookup unit enough time to provide the entry address, IF will burn 1 extra clock for 0 address ops, e.g. SMF, BCT etc.

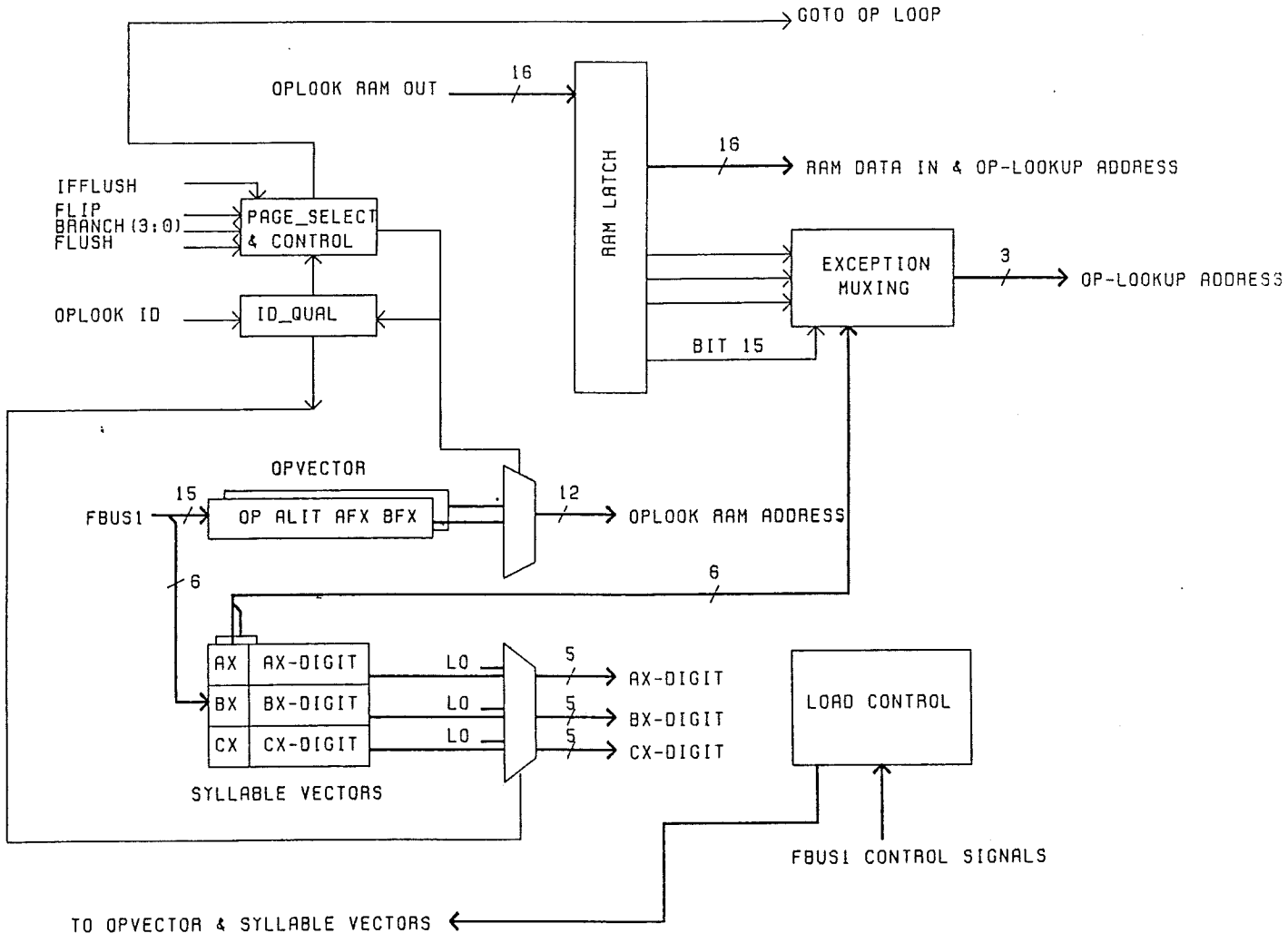


FIGURE 6-26 OPLOOK ARRAY BLOCK DIAGRAM

6.2.1.6 CONTROL STRUCTURE (Continued)

There are two sets of registers to capture information provided by the IF, one for each fetch page. At any given time, 1 set of registers is used to capture information from the IF while the other set provides additional information for OF on the opcode it is currently working on. The OPLOOK structure follows the IF by monitoring the FLIP and GOTO16K OPLIT branch signals. Depending on the current page selector, each accepted GOTO16K OPLIT will cause a particular set of registers to be released to the OF. A GOTO16K OPLIT branch will only be accepted if an OFpage is ready or will be ready (IF is performing a FLIP). An unaccepted GOTO16K OPLIT branch will cause a pseudo-loop to occur by synchronously clearing the CREG. The opcode lookup structure will ignore the first GOTO16K OPLIT branch after a flush as the CREG is being cleared to zero. On an IF flush, the lookup structure will reset any page ready conditions to allow the IF to refill the fetch page with new information.

The following is a list of information inspected by the opcode lookup structure for generation of the microcode address.

Opcode	-	8 bits of the opcode.
Alit	-	A literal indication bit.
AF exception	-	Indicates an AF exception. (ALIT error, AF infl, invalid field length and invalid infl)
BF exception	-	Indicates an BF exception. (BF infl, invalid fl, literal and invalid infl)
A excep	-	Indicates ASYL exception. (Mux with bit 1 of ram output)
B excep	-	Indicates BSYL exception. (Mux with bit 2 of ram output)
C excep	-	Indicates CSYL exception. (Mux with bit 3 of ram output)

The most significant bit from the oplook up ram specifies whether AEXCEP, BEXCEP and CEXCEP should be passed in lieu of bit 1,2,3 of the ram output. This scheme allows the ram to directly control the oplook up address provided to the

sequencer based on the information available. Because of the multiplexing of syllable exception flags and ram data, all syllable exception bits must be cleared when that set of registers is used again. This is acheived by synchronizing the clearing with the loading of opcode for that set of registers.

The following is a list of the information provided by the opcode lookup structure to the OF on the current opcode.

A excep	-	Indicates ASYL exception.
B excep	-	Indicates BSYL exception.
C excep	-	Indicates CSYL exception.
AX digit	-	Indicates the exception condition of ASYL.
BX digit	-	Indicates the exception condition of BSYL.
CX digit	-	Indicates the exception condition of CSYL.

Due to fault detection requirement, the microcode entry address must be in the range of 4 - 3FFF. In the normal operating mode, the opcode and the associated X-digit (Alit, AF excep, BF excep) are captured and used as an 11-bit address to the oplook up ram. Data coming back from the ram is then registered. Bit 1-3 of this data are muxed with the syllable exception bits to create the final microcode entry address for the microsequencer.

Assignment of OPLOOKUP RAM address:

INPUT:	OP - OPCODE	ADDR (10:3)
	AL - ALIT INDICATOR	ADDR (2:2)
	AF - AF EXCEPTION INDICATOR	ADDR (1:1)
	BF - BF EXCEPTION INDICATOR	ADDR (0:0)

Assignment of OPLOOKUP RAM output:

bit 15 - OPMUX : CONTROL TO OPLOOK MUX
1 - OPLOOKUP RAM OUTPUT [3:1] IS PASSED
0 - CEXCEP, BEXCEP, AEXCEP ARE PASSED AS OPLIT[3:1]
bit 14 - bit to generate odd parity across OPLOOKUP RAM output
bit 13:0 - OPLIT vector

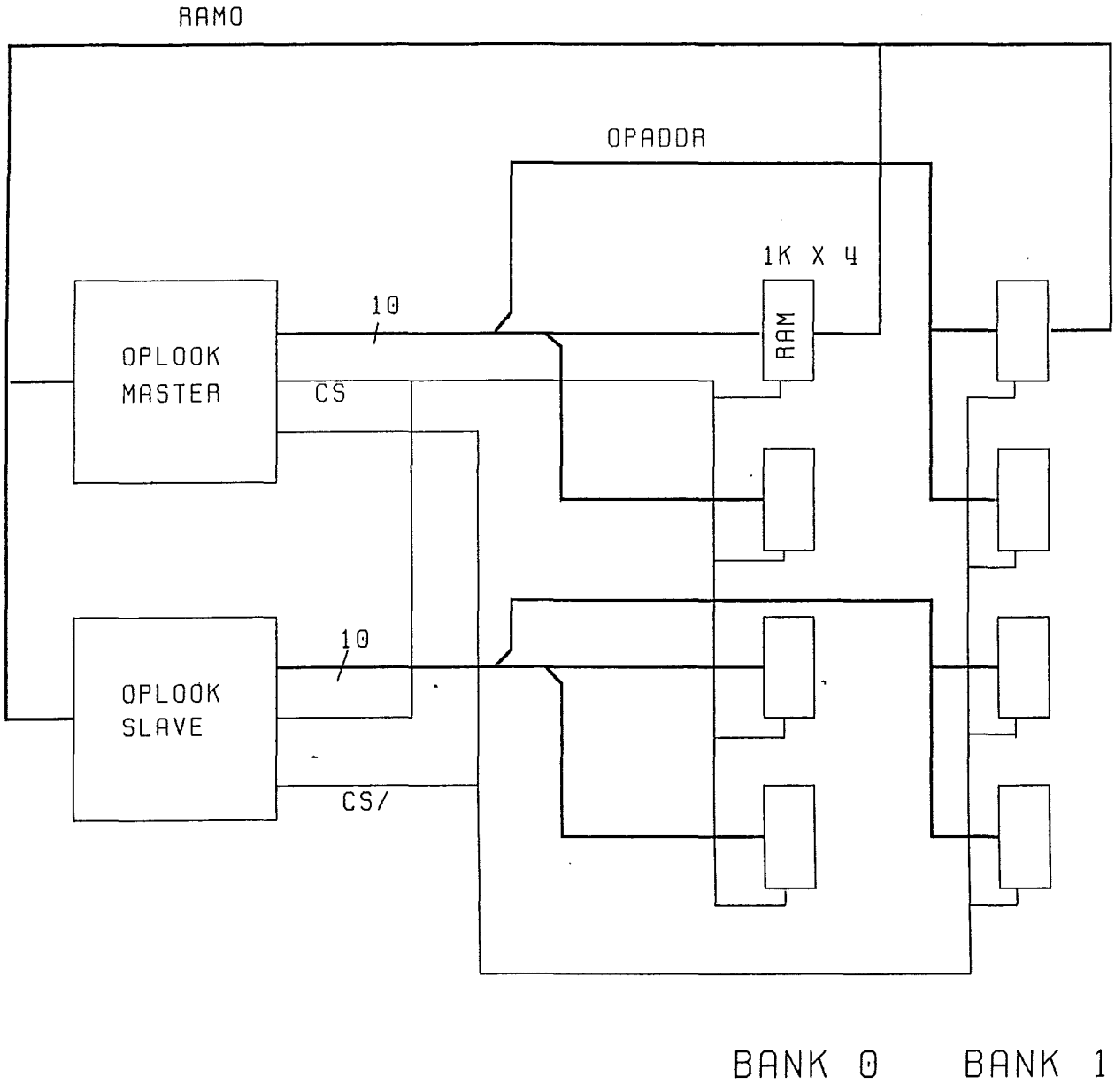


FIGURE 6-27 OPCODE LOOKUP STRUCTURE

6.2.1.6 CONTROL STRUCTURE (Continued)

NEXT ADDRESS SELECT

The next address select are under control of four CREG fields: PUSH_STACK, BRANCH, CONDITION and JUMPADR. The function of the four fields are as follows: PUSH_STACK control the loading of the UPC stack. If PUSH_STACK is set, depending on the BRANCH field, either the JUMPADR or the UPC+1 will be pushed onto the UPC stack. The JUMPADR will be pushed if BRANCH field is equal to 1. The BRANCH field selects the most significant twelve or thirteen bits of the next micro-instruction address. The CONDITION field selects the one or two least significant bits of the next micro-instrution address. There can be 32 unique 2 bits test conditions and 16 unique 1 bit test condition. The least significant bit of the top 16 2 bits test conditions can be tested through a 2 way branch.

BRANCH	NEXT ADDRESS (13:9)	(8:5)	(4:4)	(3:3)	(2:2)	(1:1)	(0:0)	(1:1) (0:0)
0	...	OPLIT(13:2)		...		4way(1:0)		
1	...	UPC+1(13:2)		...		4way(1:0)		
2	...	uSTACK(13:2)		...		4way(1:0)		
3	JMPADR(13:5)	PTREG(4:2)				4way(1:0)		
4	...	JMPADR(13:2)		...		4way(1:0)		
5	...	JMPADR(13:1)		...		2way(0:0)		
6	JMPADR(13:4)	IXcache(3:2)				4way(1:0)		
7	JMPADR(13:4)	T.B.S.(3:2)				4way(1:0)		
8	JMPADR(13:4)	AX(3:2)				4way(1:0)		
9	JMPADR(13:4)	BX(3:2)				4way(1:0)		
10	JMPADR(13:4)	CX(3:2)				4way(1:0)		
11	JMPADR(13:3)	... IXFIT(2:2)				4way(1:0)		
12-15	Reserved							

The exact description of each of the test conditions are as follows:

6.2.1.6 CONTROL STRUCTURE (Continued)

CONDITION

4way (4-way branch conditions)

0 Low 2 bits of OPLIT map address
From output of OPLOOKUP structure
1 Low 2 bits of UPC+1
2 Low 2 bits of Top of stack
3 Low 2 bits of Branch address
From JMPADR field in CREG
4 low 2 bits of Asyl 'X'
5 low 2 bits of Bsyl 'X'
6 low 2 bits of Csyl 'X'
7 Ac (from OPLEN)
8 Bc (from OPLEN)
9 Cc (from OPLEN)
10 IX5 & IX4 cache status (from Lock Unit)
11 IX2 & IX1 FIT bits (from Lock Unit)
12 IF undigit, limit error (From IF error monitor in PTEST array)
13 Undigit-AB & AU-EXCEPT
Undigit-A is the output of PTREG-0
Undigit-B is the output of BBUS undigit f/f in PTEST
Undigit-AB is logical sum of PTREG-0 & undigit check on BBUS
AU-EXCEPT is from Adder unit (set if overflow or underflow)
14 OF undigit, limit error (From OF error monitor in PTEST array)
15 AEXCEP & 0
Asyl-exception bit for the op currently under OF & '0'
16 T.B.S.
17 PTREG-10
18 PTREG-32
19 T.B.S.
20 AF>=BF (From OPLEN)
21 Interrupt
Bit 1: code-hit or addr-error or
time-out or Odd PC
Bit 0: Lock-full
(lockfull is AT_LEAST_1 condition from lock unit)

6.2.1.6 CONTROL STRUCTURE (Continued)

22 Comparator A>=B
23 AU-EXCEPT & AU = 0
24 Undigit-AB or address error & AU-EXCEP
25 Condition 1 & IX1VALID
(condition 1 is microcode clear/set f/f 1 & IX1VALID comes from lock unit's IXOK bit)
26 Condition 1 & IX2VALID
(IX2VALID comes from lock unit's IXOK bit)
27 Condition 1 & IX3VALID
(IX3VALID comes from lock unit's IXOK bit)
28 CB-EXCEPTION
(C-EXCEPTION is the CSYL exception bit from OPLOOK array &
B-EXCEPTION is the BSYL exception bit from OPLOOK array)
29 Address error & Condition 1
30 Number of extended syllables

2way (2-way branch conditions)

0 IX/IA hit (from Lock Unit)
1 Code hit (from Lock Unit)
2 PTREG-4
3 Odd-PC
4 Odd-PC or code-hit
5 XMEVENT
XM to OF event for XM
6 Undigit-B
Undigit-B is output of BBUS undigit f/f in PTEST
7 At_least_2_lock_entries_avail (from Lock Unit)
8 NEWOPSYL
Set when OPSYL location in scratch pad is written into
9 Time-out
10 Addr-error
(OF-undigit-err OR OF-limit-err OR IF-limit-err or IF-undigit-err)
11 Data hit
12 LIX_ED (ucode clear/set testable condition 2)
13 Optimal flag (from OPLEN)

6.2.1.6 CONTROL STRUCTURE (Continued)

14 Condition 3
15 Reserved
16 T.B.S.
17 PTREG-0
18 PTREG-2
19 T.B.S.
20 AF=BF (From OPLEN)
21 Interrupt bit 0
Bit 0: Lock-full
22 Comparator A=B
23 AU = 0
24 AU-EXCEPT
25 IX1VALID
26 IX2VALID
27 IX3VALID
28 B-EXCEPTION
29 Condition 1
30 LSB of number of extended syllables

Note: On Oplit branch (BRANCH type=0), the micro-instruction stack pointer will be reset.

6.2.1.7 IX CACHE HANDLING

For performance reasons, it is important for P5 fetch to try to maintain the current copy of each IX register content in OF internal scratchpad to reduce overhead in resolving indexed addresses. Furthermore, the P5 fetch will attempt to pre-cache IX register content by detecting any instruction that is writing to the exact location of an IX register. Index pre-caching, because of differences in how IX1-IX3 & IX4-IX7 are handled in memory, will be handled in two ways.

The first algorithm described below will deal exclusively with IX1-IX3 index pre-caching. In the general fetch flow, OF will send write lock request to the lock unit & set fetch page vaild on the last clock. In instructions that have high probabilities of being used to set up IX1, IX2 or IX3, OF will check on 3 bits set by the lock unit [IXFIT(2:0)] to indicate an exact match of write lock to any 1 of the IX registers. (The 3 bits mentioned are cleared by code check request). Instructions that OF is planning on servicing are:- MVN, MVA, MVW, MPY, INC, DEC, ADD, SUB, NTR, EXT and 'local' VEN. After determining that an IX has been invalidated by the current instruction, the OF will determine if it can easily handle this particular instruction. (For example, a 100UN to 7SN MVN to IX1 will not be handled by OF). If OF has determined that the instruction is best handled by XM then no further work will be performed.Otherwise, OF will perform the actions as specified by the instruction. OF will not perform any writes to memory nor will the conditional toggles be changed. OF will store the result of the operation in the respective IX cache location in its scratchpad and set the corresponding IX valid and IX-CACHE flag. The IX valid flag will allow subsequent fetch of IX register from OF scratchpad. The IX-CACHE flag will allow lock unit to reset the corresponding IX valid flag after a pipe flush, to prevent inadvertent use of pre-cached data specified by an instruction fetched after a mis-predicted branch. (Note: For ease of implementation, OF will always invalidate any pre-cached IX locations after a pipe flush).

6.2.1.7 IX CACHE HANDLING (Continued)

The second algorithm used to handle IX pre-caching will handle LIX instruction only. LIX instruction can change any one of the 7 IX or all of IX4-IX7 at once. In the handling of the LIX instruction, OF will examine the BF to determine which IX register will be affected. IX1-IX3 pre-caching will be handled by the previously described algorithm. IX4-IX7 pre-caching will be performed by the following algorithm. In the OF scratchpad, there are 2 sets of IX4-IX7 locations. The first set is designated as primary IX cache, the second set as secondary IX cache. On detecting a LIX to one of IX4-IX7, OF will send out the read request for the new IX content. Under the read request, OF will move the contents of the respective primary cache to the secondary cache. OF will also set the corresponding bit(/s) in a 4 bit IX4-IX7-STATUS register and set the LIX_ED flag. OF, by examining the LIX_ED flag, will only pre-cache one LIX instruction that changes IX4-IX7 ahead of XM because of the limitations in scratchpad. It is up to the XM to signal to OF that XM has just executed a LIX instruction through the LIX_OK (ELIXOK$$$P) wire. The LIX_OK (ELIXOK$$$P) is used to clear the 4 bit IX4-IX7-STATUS register and LIX_ED flag in OF. If a bad branch type pipe flush occurs before XM has a chance to execute the LIX instruction then OF will use the IX4-IX7-STATUS register to determine which primary IX4-IX7 cache to restore. After restoring any IX4-IX7 cache, OF will clear the IX4-IX7-STATUS register and LIX_ED flag. And if no bad branch pipe flush occurs between the pre-caching by OF & actual execution by XM then the LIX_OK (ELIXOK$$$P) wire will be used to clear IX4-IX7-STATUS register and LIX_ED flag. A flush (context switch type pipe flush) occurs before execution of LIX instruction in XM will always reset IX4-IX7-STATUS register, LIX_ED flag and IX_OK's flags. (refer to pipe clear section for more details)

6.2.1.7 IX CACHE HANDLING (Continued)

In the case of BRV and Hyper RET instructions, the OF module is required to reload the mobile index registers. In order to increase the amount of parallelism between FETCH and XM, OF will pass the parsed instruction to XM before the mobile index registers are reloaded. This requires the XM to 1) validate the newly loaded mobile index registers and 2) make sure that OF has reloaded the mobile index register before flushing. The synchronization between XM and OF is accomplished through the use of the OF->XM (FOF2XM$$$$$P) event wire. OF will set the OF->XM (FOF2XM$$$$$P) event wire after it has reloaded the mobile index registers. XM, on the other hand, will monitor the OF->XM (FOF2XM$$$$$P) wire and upon receiving it send out the LIX_OK (ELIXOK$$$P) signal. After that, the XM can issue a flush.

6.2.1.8 PIPE CLEAR

Like the IF, the OF will receive the following signals from XM: FETCH_FLUSH (EFETFLUSHP) and SYS_FLUSH (ESYSFLUSHP). After a SYSTEM flush, OF will clear IX4-IX7 STATUS register, LIX_ED flag, upc stack and fetch page counter. The lock unit will pop all lock entries, reset all hit bits and clear IX_OK's signals. After a BAD_BRANCH pipe clear, OF will clear upc stack and fetch page counter. OF will then examine the LIX_ED flag to determine if any of the IX4-IX7 registers needs to be restored from the secondary IX cache. The lock unit will pop all lock entries, reset all hit bits, clear IX-CACHE register after using the register as a mask to clear respective IX_OK's flags.

6.2.1.9 PROTOCOL TO INTERFACE WITH THE LOCK UNIT

OF will notify the lock unit of desired actions through the INT-COMMAND portion of the microword. The protocol of interaction requires OF to source the 7 digit begin address on the ABUS and the 7 digit end address from AUREG, both operands being right justified, when requests are to be made to the lock unit. During a live freeze condition, the hardware will drop the command to the lock unit until the live freeze goes away.

The OPCOMP (EOPCOMPLETEP) coming from XM will be registered first. Therefore, the lock unit will receive the pop lock command 1 clock later. This requires the XM to make sure that the OPCOMP (EOPCOMPLETEP) signal and the write data are in sync. That is, the OPCOMP (EOPCOMPLETEP) signal and the write data must be on the backplane at the same time or OPCOMP (EOPCOMPLETEP) signal send later than the data.

6.2.1.10 PROTOCOL TO INTERFACE WITH THE OPLEN UNIT

OF will observe the following timing restrictions when dealing with the OPLEN unit. The following information to be provided by the OPLEN unit: ALEN, BLEN, Ac, Bc, Cc, 1 clock after OF started on a OFpage; AF>=BF, Optimal, CLEN will be valid after 1 more clock. When an INFL occurs, stripped AF, BF will be available from ALEN and BLEN respectively. However, if there is any error in the field length rather than a valid INFL, ALEN or BLEN should put out invalid values like 3FF instead. This allow the OF module to assume INFL if it sees a field length exception and fires off the read without checking for valid INFL first. MCACM will detect an error and return an error tag. The lock unit, however, might have a hit depending on the current lock table content. The microcode, therefore, must check for validity of the INFL before looking for IX/IA hit.

6.2.1.11 PROTOCOL TO REDIRECT IF

OF, after encountering code-modification or running out of lock entries, will need to redirect IF to refetch the current or next instruction in the current code stream. The protocol needed is for OF to send out the program counter on the ADDRBUS with the corresponding NEWPC command on the ADDRBUS command field. (There is a restriction that OF has to observe. After sending IF FLUSH, OF has to perform at least one lock unit request or one AWBUS request before sending SHOVE). The described action will notify IF to re-write the current fetch page IF was working on before the 'BURP' with the refetched information.

6.2.1.12 CODE MODIFICATION HANDLING

OF has the responsibility of checking for code integrity of instructions about to be executed. Code integrity check is accomplished by performing a code-modification check at the beginning of an instruction handling in OF. The algorithm used is as follows: OF will perform a code check from current program counter to three different lengths depending on the type of instruction being parsed.

a) If the instruction is a predicted not taken conditional branch or non-branch type instruction then OF will use the amount that READER in IF is pre-fetching (90 digits) as length for code check.

b) If the instruction is a predicted taken conditional branch then OF will use the amount equal to sum of maximum length of a predicted taken conditional instruction (10 digits) and maximum length of a medium system instruction (30 digits) as length for code check. It should be noted here that IF may be saving off 1 instruction in the fall-through path of a predicted taken conditional branch, necessitating OF to always check code integrity of the fall through path.

6.2.1.12 CODE MODIFICATION HANDLING (Continued)

c) If the instruction is an unconditional branch type instruction then OF will use the amount equal to the maximum length of that particular type of instruction as length for code check.

OF will change a detected code modified instruction's OPSYL to a 'Nxxx1COPAF'. The related OPLIT will notify XM that neither the instruction in the current fetch queue nor any pre-fetch operand in XM's OPERANDQ is valid. It is up to the XM to treat that instruction as a NOP and disable interrupt checking for this instruction. As operand prereads may have occurred already before code hit is detected, XM needs to pop any such preread before proceeding. To simplify this, the OF module will inform the XM on how many prereads have been sent. Also, when OF detected an error while parsing a code modified opcode, XM should issue a system flush. This alleviates the OF from keeping track of the exact number of prereads fired when an error is detected. These are accomplished by using the 'N' field as follows :

0 - No preread to pop
1 - Preread on A
2 - preread on B
3 - Preread on A & B
4 - XM to issue a system flush as error has occurred

6.2.1.12 CODE MODIFICATION HANDLING (Continued)

The OF microcode will be responsible for providing this information to the XM. Note that OF does not take into consideration whether ALIT is specified. This means that XM must interpret OPAF to determine if PREREAD-A is actually fired. For example, for an opsyl of '30001C46A4', only PREREAD-B has been sent. XM must not pop operand queue A even though a N has a value of '3'. In providing this information to XM, OF microcode will need to invoke a special error classification routine to determine if a code hit has occurred when an error is detected. Within the error classification routine, the OF microcode must execute 2 16-way branch on the 2 digits of the opcode to help in determining the number of fired prereads. The current implementation requires two error classification routines. One is used at the end of an instruction cycle while the other is used when an error is detected while parsing the instruction.

Write locks in the lock unit will be popped from the lock unit after three code check requests have been made against an entry that has received its corresponding op-complete command from XM. This method of unlocking is chosen because the P5 READER, IF and OF can be working on 3 disjoint code streams. To guarantee code integrity, without passing the begin addresses of each code stream for code check to the lock unit, requires a lock entry to be released after all three code stream's begin addresses have been checked against each lock entry.

6.2.1.13 INSTRUCTION DEPENDENT HANDLING

As previously mentioned in the interface section, OF will send out operand pre-read requests for read operands to XM. This section will describe in detail what OF will pre-read for XM and number of write locks OF will post to the lock unit.

PRE-READs = Description of pre-read requests made by OF
e.g. A:MOD 10 => read length mod 10 from A address
A:8 => read 8 digit from A address
A*:MOD 10 => if not literal then read length mod 10 from A address else no read
48:1 => read 1 digit from base 0 relative location 48
A(77):1 => read 1 digit from absolute memory location 77
[IX3]:10 => read address in IX3 for 10 digits

#LOCKs = No. of write locks posted by OF

LOCK-CODE = Description of lock requests made by OF
e.g. C:CL, B:BL => OF will request a lock from C address for CL and a lock from B address for BL
(XM should unlock requests in the order OF had sent to the lock unit since the lock unit works like a FIFO)

6.2.1.13 INSTRUCTION DEPENDENT HANDLING (Continued)

	INSTRUCTION		PRE-READ	#LOCKs	LOCK	NOTE
Arithemtic						
1	INC	A*:MOD 10, B:MOD 10		1	B:BL	
2	ADD	A*:MOD 10, B:MOD 10		1	C:CL	21
3	DEC	A*:MOD 10, B:MOD 10		1	B:BL	
4	SUB	A*:MOD 10, B:MOD 10		1	C:CL	
5	MPY	A*:MOD 10, B:MOD 10		1	C:CL	
6	DIV	A*:MOD 10, B:MOD 10		2	B:CL, C:BL	
50	IAD	A:8				18
51	IAS	A:8		1	A:8	18
52	ISU	A:8				18
53	ISS	A:8		1	A:8	18
54	IMU	A:8				18
55	IMS	A:8		1	A:8	18
57	IMI	A:8		1	A:8	18
58	ILD	A:8				18
59	IST			1	A:8	18

	INSTRUCTION		PRE-READ	#LOCKs	LOCK	NOTE
70	RAA	A:4, A+4:8 or A:10, A+10:10				18
71	RAS	A:4, A+4:8 or A:10, A+10:10		1	A:12 or A:20	18
72	RSU	A:4, A+4:8 or A:10, A+10:10				18
73	RSS	A:4, A+4:8 or A:10, A+10:10		1	A:12 or A:20	18
74	RMU	A:4, A+4:8 or A:10, A+10:10				18
75	RMS	A:4, A+4:8 or A:10, A+10:10		1	A:12 or A:20	18
76	RDV	A:4, A+4:8 or A:10, A+10:10				18
77	RDS	A:4, A+4:8 or A:10, A+10:10		1	A:12 or A:20	18
78	RLD	A:4, A+4:8 or A:10, A+10:10				18

6.2.1.13 INSTRUCTION DEPENDENT HANDLING (Continued)

79	RST		1	A:12 or A:20	18
84	ACM				

Branches

20	NOP	
21	LSS	(B1,F1,E1)
22	EQL	(B2,F2,E2)
23	LEQ	(B3,F3,E3)
24	GTR	(B4,F4,E4)
25	NEQ	(B5,F5,E5)
26	GEQ	(B6,F6,E6)
27	BUN	
28	OFL	
2A	NUL	(BA,FA,EA)
2B	GTN	(BB,FB,EB)
29	HBR	A(48):3 (OMEGA) A(77):1 (MCP IX)
48	HBK	46:3 (OMEGA) 46:1 (MCP IX)

INSTRUCTION		PRE-READ	#LOCKs	LOCK	NOTE
ENVIRONMENT CHANGE					
30	BCT				1
31	NTR	IX3:8	2	24:22, TOS:16+2*AFBF TOS=Top of stack (in memory 40-45)	2
32	EXT	IX3:8	1	24:22	3
35	VEN	B:8	1	24:22, TOS:24+2*AFBF	4
62	HCL	B:4			5
63	RET	[IX3]-6:20			6
61	ASP	A*:MOD 10,40:6	1	40:6	
90	BRE	A(60):10,A(70):10			7
90	INT				7
93	BRV				8

6.2.1.13 INSTRUCTION DEPENDENT HANDLING (Continued)

DATA MOVEMENT					
8	MVD	A:10	1	B:(C-B) if forward C:(B-C) if backward	9
9	MVL	A:MOD 10,B:MOD 10	3	A:AL,B:AL, C:AL	10
10	MVA	A*:MOD 10	1	B:BL	
11	MVN	A*:MOD 10	1	B:BL	
12	MVW	A:MOD 10	1	B:4*AFBF	11
13	MVC	A:MOD 10	2	B:4*AFBF, A:4*AFBF	11
14	MVR	A:MOD 10	1	B:AL*100	
15	TRN	A*:MOD 10	1	C:CC*(AFBF)	12
49	EDT	B:2	1	C:BF*20	
LOGICAL					
16	SDE	A*:MOD 10,B:MOD 10	1	38:2	
17	SDU	A*:MOD 10,B:MOD 10	1	38:2	
18	SZE	A*:MOD 10,B:MOD 10	1	38:2	
19	SZU	A*:MOD 10,B:MOD 10	1	38:2	
39	SEA	A*:MOD 10,B:ALEN MOD 10	1	IX1:8	
37	SLL	A*:MOD 10, B:6	1	IX1:8	
38	SLD	A*:MOD 10, B:6	1	IX1:16	
64	SLT	C:18	1	IX1:16	13
66	STB	C:20	1	IX1:8	14
INSTRUCTION		PRE-READ	#LOCKs	LOCK	NOTE
40	BOT	A*:MOD 10			
41	BZT	A*:MOD 10			
45	CPA	A*:MOD 10,B:MOD 10			
46	CPN	A*:MOD 10,B:MOD 10			
33	BRT	A:MOD 10	1	A:AL	
34	BST	A:MDO 10	1	A:AL	
42	AND	A*:MOD 10,B:MOD 10	1	C:CL	
43	ORR	A*:MOD 10,B:MOD 10	1	C:CL	
44	NOT	A*:MOD 10,B:MOD 10	1	C:CL	

6.2.1.13 INSTRUCTION DEPENDENT HANDLING (Continued)

INPUT/OUTPUT					
85	CIO	A:20	1	B:40	
91	SRD	[AFBF]:8	1	IX1:8	
92	RAD	A:10	1	A:10	
94	IIO	A:20	1	A:8	
98	IOC	A:6	1	B:8	
BINARY/DECIMAL CONVERSION					
88	D2B	A*:MOD 10	1	B:BL	
89	B2D	A*:MOD 10	1	B:BL	
MEASUREMENT					
87	MOP	A:AL, B:BL			
MISCELLANEOUS					
86	ATE	A:8, B:8			16
67	LIX	A:8	See note 19		19
68	SIX	See note 20	1	A:32	20
60	LOK	A:20			
65	WHR	A:9			
47	SMF				
AB	BAD		0		
99	SST		1	A:200	
95	RDT		1	A:20	
97	STT	A:20			
STRING					
A0	MVS	A:10, B:10	1	ALL MEM	17
A1	CPS	A:10, B:10			
A2	HSH	A:10, B:6	1	B:BL	

Notes:

1) BCT - OF may assist in changing the base/limit if signals can be added between OF and XM notifying OF when it is OK to change the base/limit table.

6.2.1.13 INSTRUCTION DEPENDENT HANDLING (Continued)

2) NTR - OF will pass 2*AFBF (if AFBF=0 then 0) to XM on LENSYL. OF will pass content of TOS right justified on CSYL to XM. OF will pre-read IX3 as A-operand for XM. 3) EXT - = OF will pre-read IX3 as A-operand for XM.

4) VEN - OF will pre-read environment field (starting at B-address for 20 digits) for XM. OF will pass content of TOS right justified on CSYL to XM. OF will pass 2*AFBF (if AFBF=0 then 0) on LENSYL to XM. OF will pass IX3 (BI 0 S 0AAAAAA) in SPECSYL to XM. OF will send out new pc if environment no=0 ('local' VEN) else OF will wait for a pipe flush from XM.

5) HCL - OF will wait for an environment change type flush from XM.

6) RET - OF will check stack frame indicator. If RET is of the type local 'VEX' then OF will send out new pc else OF will wait for a pipe flush from XM. OF will read up the mobile index registers if it is a Hyper or Hardware return. Special protocol must be followed when it is a Hyper or Hardware return by OF and XM. OF will pass stack frame indicator left justified on ASYL. OF will pass IX3 on BSYL to XM.

7) BRE - OF may assist in changing base/limit tables if signals can be added between OF and XM notifying OF when it is OK to change the base/limit tables.

INT - = OF will wait for a pipe flush from XM.

8) BRV - OF will wait for a pipe flush from XM. OF will read up the mobile index registers. Special protocol must be followed by both OF and XM. OF will pass IX1 on ASYL to XM.

6.2.1.13 INSTRUCTION DEPENDENT HANDLING (Continued)

9) MVD - OF will pass length of MVD on FBUS1 to XM. OF will check for limit error before prereading Asyl for 10 digits.

10) MVL - OF will pre-read ALEN mod 10 digits from A-address and B-address for XM. Only ALEN is sent. 11) MVW,MVC - = OF will pass 4*AFBF (if AFBF=0 then 40000) on FBUS1 to XM.

12) TRN - OF will pass CC*AFBF on FBUS1 to XM. If AFBF=0 and UN then length = 10000. If AFBF=0 and UA then length = 20000.

13) SLT - OF will pre-read list descriptor for XM.

14) SLB - OF will pre-read table descriptor for XM.

15) SRD - OF will pass AFBF as ASYL for XM.

16) ATE - OF will wait for XM to flush.

17) MVS - OF must lock all memory since MVS can be writing anywhere.

18) ALEN of FBUS2 will not have the appropriate length for these types of ops.

19) LIX - If IX1 or IX2 or IX3 is loaded, a corresponding write lock will be posted.

20) SIX - If IX1 or IX2 or IX3 is stored, a preread and a write lock will be issued. E.g. SIX of IX1 will have 8:8 as preread and write lock.

21) MPY - When the CLEN is 400 digits, a value of zero will be sent.

6.2.2 GENERAL FETCH ALGORITHM

When the OF module is ready for the next opcode, it performs a GOTO16K OPLIT branch. If the microcode entry address provided by the opcode lookup structure is ready, the corresponding microword will be loaded onto the CREG. At this point, the OF knows the type of opcode it is handling and the number of syllable or field length exceptions. The OF module always resolve field length exceptions before sullable exceptions. When all the excpetions are resolved, appropriate actions required by the opcode will be performed.

OF will always perform a code check for the currect opcode to ensure its code integrity. The length of the code check depends on the type of opcode it is handling. Also, prereads will be sent out if necessary. Finally, the OF will check to see if any error of exception has occurred when the opcode is being resolved. This whole process can take as little as 4 clocks. Any action that may result in an 'infinite' loop requires the checking of TIMEOUT condition. Every attempt is make to report the original error if it is possible. In most cases, OF will resolve exceptions in order of AF, BF, ASYL, BSYL and CSYL. However, in some cases, OF will resolve BSYL in parallel with ASYL.

6.3 BRANCH PREDICTION/PIPE CLEAR

6.3.1 BRANCH PREDICTION

In the P5 Fetch, branch prediction is done primarily by the IF module. As the IF parses a branch instruction, its OPCLASS test determines whether the branch should be taken or not, and if taken, whether it is a hard branch (i.e., BUN) or predicted taken branch based on the biased opcode.

On a hard branch, after the IF Formatter has parsed the branch address, it sends a NEWPC command to the IF Reader--unless the branch address was indexed or indirect, in which case IF halts after the instruction has been parsed and waits. The OF module will resolve the branch address and issue a NEWPC to IF.

On a predicted branch, the IF Formatter will send a NEWPC to the Reader after the branch address has been parsed (if it is not indexed or indirect, in which case once again OF gets that task) but instructs the Reader not to wait on clearing out the Read Queue and trashing old read data until after the IF Formatter has completed parsing the Not Taken op (indicated by the DONE signal to the Reader). This NT op will be placed in the NT entry of the Instruction Queue along with the address of the following instruction (NIA), to be held until the XM determines the correctness of the prediction.

It should be stated that XM will examine the condition or overflow toggles to determine if any branch was incorrectly predicted. If any branch instruction has been incorrectly predicted then XM will redirect the code stream by performing a pipe clear.The XM is required to send a BRANCH_OK signal after determining a predicted branch taken was predicted correctly. The BRANCH_OK signal will notify the IF that it can clear its NT entry in the IQ.

6.3.2 PIPE CLEAR

Pipe clear in P5 is achieved by a combination of 3 wires: SYS_FLUSH, FETCH_FLUSH and IF_FLUSH, and 8 dedicated NEWPC commands (only 5 of the 8 reserved commands are used at this point) on the ADDRBUS.

There are seven kinds of NEWPC operations:

- The IF Formatter parses a branch instruction and determines it will be taken. The Reader is informed of this via a NEWPC on internal data lines. (This particular pipe clear is internal to IF and will not require the service of the dedicated wires).

- The OF resolves a branch address that the IF wanted to take, and wishes to pass this address to the IF. This is accomplished by OF sending out the branch address on the ADDRBUS with NEWPC command (IX/IA Done) on the ADDRBUS command.

- The OF discovers a case of code modification, and wishes to redirect IF to refetch the offending instruction and reparse it. This is accomplished by OF sending the address of the current instruction on the ADDRBUS with NEWPC(BURP) on the ADDRBUS command and activation the IF_FLUSH signal.

- The OF discovers a full Lock table, issues code checks to release enough entries to finish posting the write lock for the current instruction, then wishes to redirect IF to refetch the next instruction. This is accomplished by OF sending the address of the next instruction on the ADDRBUS and NEWPC(BURP) on the ADDRBUS command and activation of the IF_FLUSH signal.

- The XM determines that a branch was predicted NOT TAKEN incorrectly, and sends the address of the correct code on the ADDRBUS and NEWPC(BAD_BRANCH_NOT_TAKEN) on ADDRBUS command to IF, OF and MCACM , along with the activation of the FETCH_FLUSH signal. As part of effects of the NEWPC(BAD_BRANCH_NOT_TAKEN), the MCACM will stop returning

any outstanding reads 1 clocks after the arrival of the NEWPC command. This is equivalent to the B4900's pipe clear, with the exception that OF's index cache will not be trashed.

- The XM determines that a branch was predicted TAKEN incorrectly, and sends the address of the correct code on the ADDRBUS and NEWPC(BAD_BRANCH_TAKEN) on ADDRBUS command to IF, OF and MCACM , along with the activation of the FETCH_FLUSH signal. As part of effects of the NEWPC(BAD_BRANCH_TAKEN), the MCACM will stop returning any outstanding reads 1 clocks after the arrival of the NEWPC command. This is also equivalent to a pipe clear, except in this case the OF does not trash its index cache and the IF FBUS Control immediately starts sending the NT instruction in the IQ to OF and XM while the Formatter issues an internal NEWPC to the NIA address in the Instruction Queue's NT entry. This gets the machine executing sooner after a misprediction.

- The XM executes an environment change instruction and sends the address of the new code stream on ADDRBUS together with the NEWPC(ENVIRONMENT_FLUSH) on the ADDRBUS command field to IF, OF and MCACM along with the SYS_FLUSH line. The activation of SYS_FLUSH will cause IF, OF and XM to start over, with the OF trashing any cached index registers it may have had.As part of effects of the NEWPC(ENVIRONMENT_FLUSH), the MCACM will stop returning any outstanding reads 1 clocks after the arrival of the NEWPC command.

Note: A NEWPC on the ADDRBUS is defined as a PC on the data lines and 1 of the 8 NEWPC command on the associated command bus. The length portion of the ADDRBUS is ignored. The MCACM treats 4 of the 8 NEWPC commands as No-op while the other 4 NEWPC commands will stop the MCACM from returning any read requests 1 clock after the arrival of the NEWPC command.

6.4 FREEZE AND STOP LOGIC

There are several forms of clock-stop in the Fetch modules. The clock can be stopped for any of the following reasons:

- Maintenance requests the clocks to stop.
- A "Dead Freeze" condition has occurred.

6.4.1 LIVE FREEZE

A "Live Freeze" occurs when execution must be temporarily stopped due to the non-availability of a required resource. When that resource becomes available, execution resumes.

The following occurances cause a Live Freeze in that section of IF:

- The Reader freezes if:
  1. it tries to access the ADDRBUS (XADRBUS$P) and cannot due to priority restrictions or memory request queue full conditions.
  2. the Read Queue contains no more room for further requested data.
- The Formatter freezes if:
  1. RREG1 does not contain valid data.
  2. it tries to pop RREG1 and no data is available.
  3. IFFRZ is on.
  4. there is an IX or IA branch address to be resolved by OF.
  5. OFREQVLD and OFREQ are active.

6. waiting for a BRH_OK (EBRANCHOK$$P), so it can store the NTI.

o The FBUS Controller freezes if:

1. no valid data on SBUS (write enable high).

2. OF is using the FBUS1 (FBUS1$$$P) (OFREQ FOBUSREQP is active) and FBUS1REG is valid.

3. the OF FETCH PAGE QUEUE is full.

4. the XM Fetch Queue pages are full.

The following occurances cause a Live Freeze in OF:

o lock unit or ADDRBUS (XADRBUS$P) request when previous ADDRBUS (XADRBUS$P) request has not been serviced because of input queue full condition in MCACM or ADDRBUS (XADRBUS$P) was busy.

o needed read data is not in the RBUS Queue.

6.4.2 DEAD FREEZE

A "Dead Freeze" occurs when some fatal error has occurred from which recovery is impossible, and it is desireable to stop as quickly as possible to aid in debugging the problem.

The following occurances cause a Dead Freeze in READER module:

1. Parity error on the Address bus detected by the Reader.

2. Parity error on the Internal S bus.

3. Fault (counter does not count) on the Address counter.

4. Error on the memory bus priority logic.

5. Parity on the Address Counter control.

6. Error on the INCrementer logic.

7. Parity error on the PC.

8. Reader queue parity error.

9. Read Data Bus parity error.

10. Read or Write pointer parity error.

11. Parity error between RQ to RREG1.

12. Error on the rotater.

13. Parity on the PCQ.

14. Parity on IQ.

15. Parity difference from S bus to FREG1.

16. Parity on FBUS CONTROL.

17. Parity on the Formatter Controller Stored Logic.

The following occurances cause a Dead Freeze in IF:

- Parity error in Tester RAM.
- Hardware fault detected.

The following occurances cause a Dead Freeze in OF:

- Control Store parity error.
- Hardware fault detected.

Once set, a Dead Freeze condition will remain until cleared by Maintenance. The rest of the processor will be stopped within 3 clocks of a Fetch module stopping.

6.4.3 SELF STOP

Fetch Module can only be stopped via System Stop (3 clocks after trigger event).

7 TIMING

7.1 INTERFACES

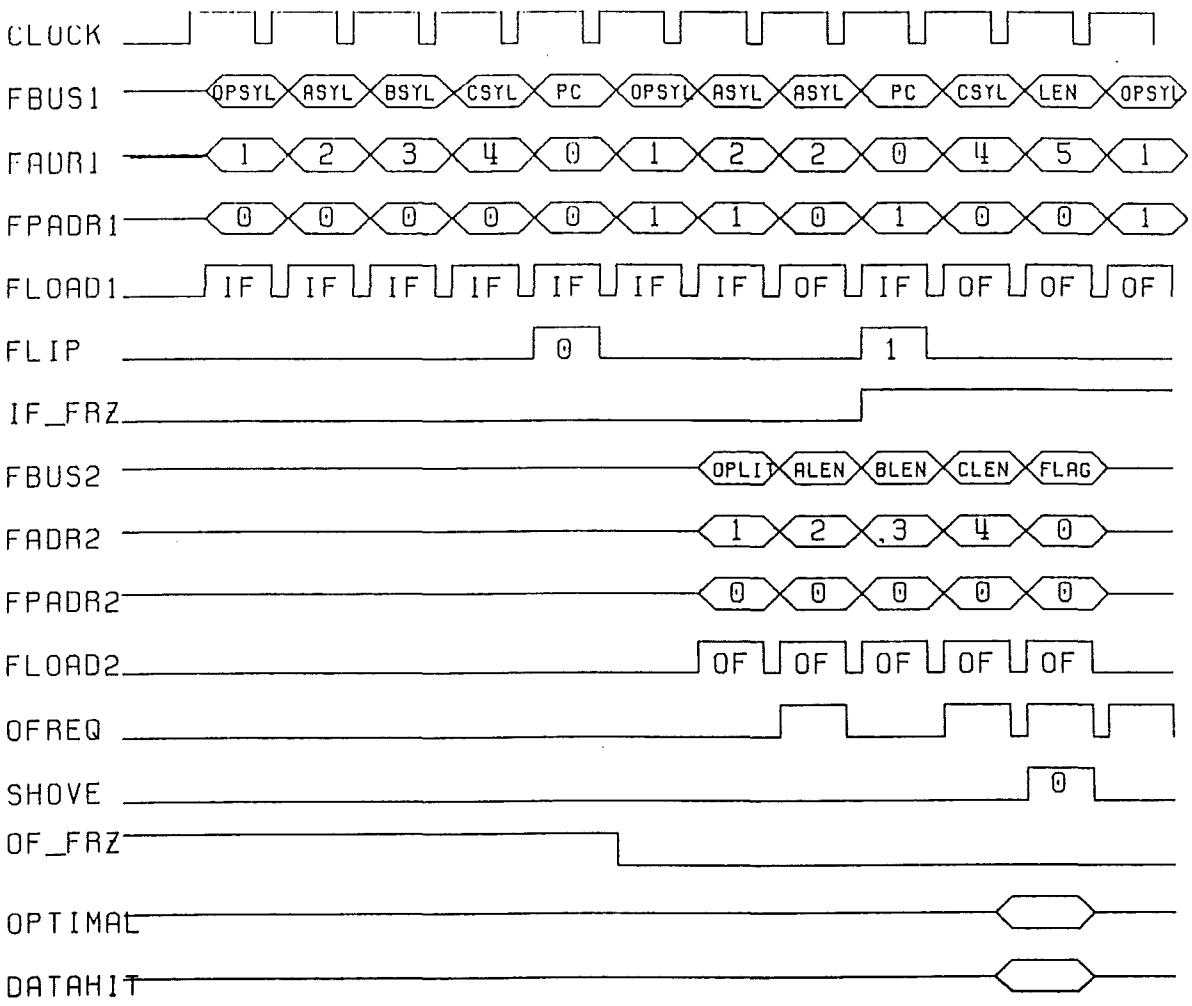


FIGURE 7-1 FBUS TIMING EXAMPLE

7.2 IF INTERNAL

7.2.1 NORMAL INSTRUCTION CYCLE

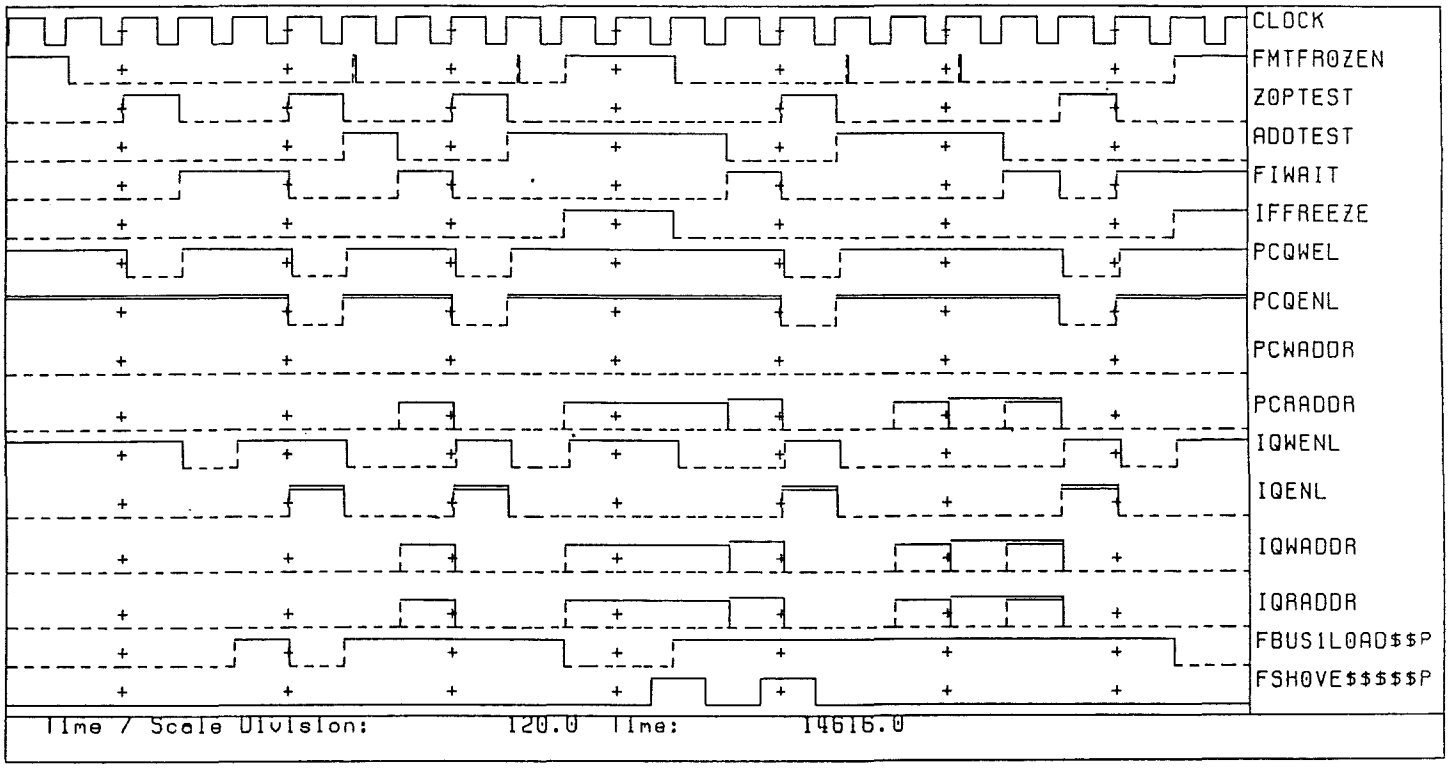
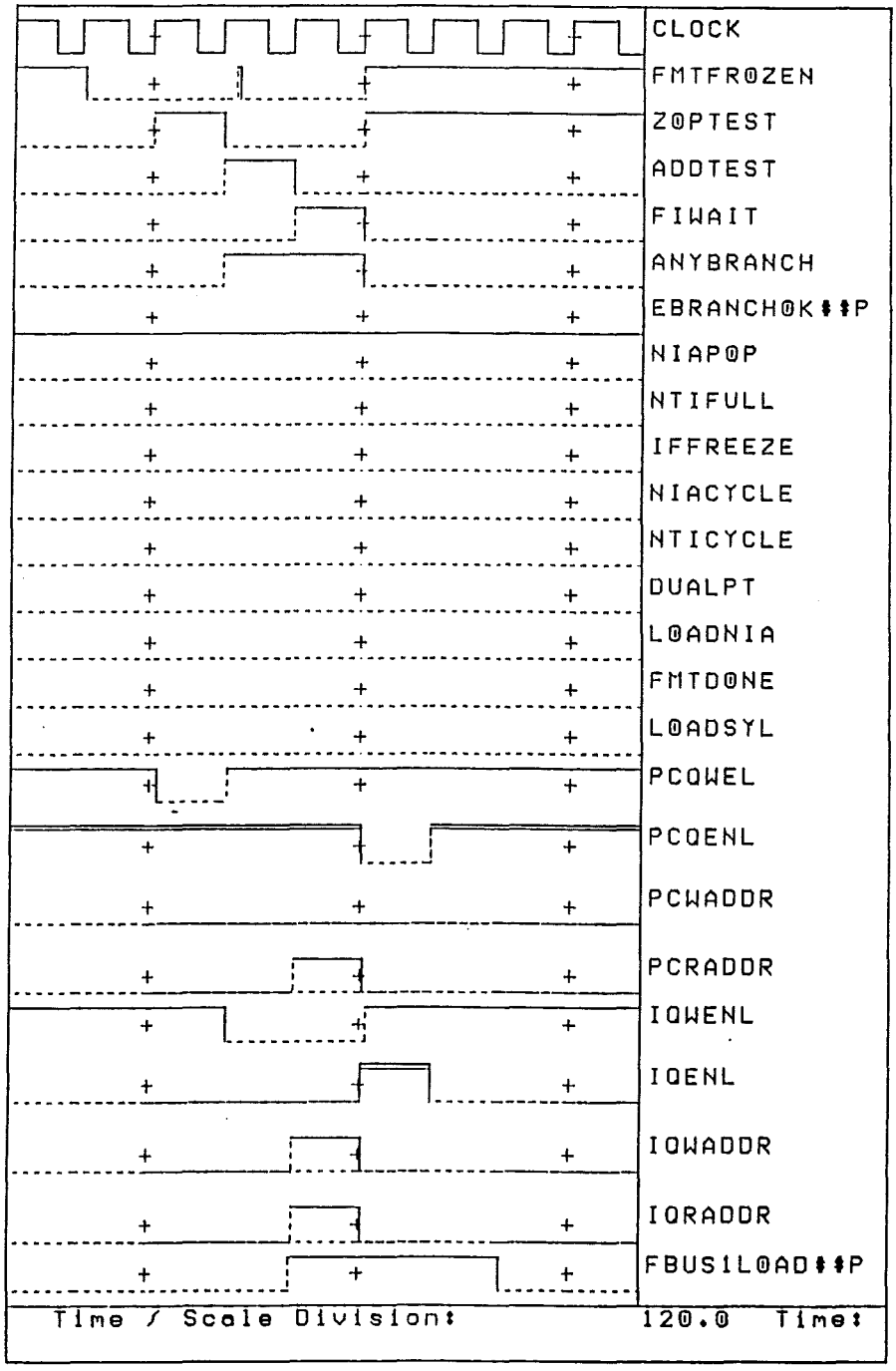


FIGURE 7-2 NORMAL INSTRUCTION TIMING

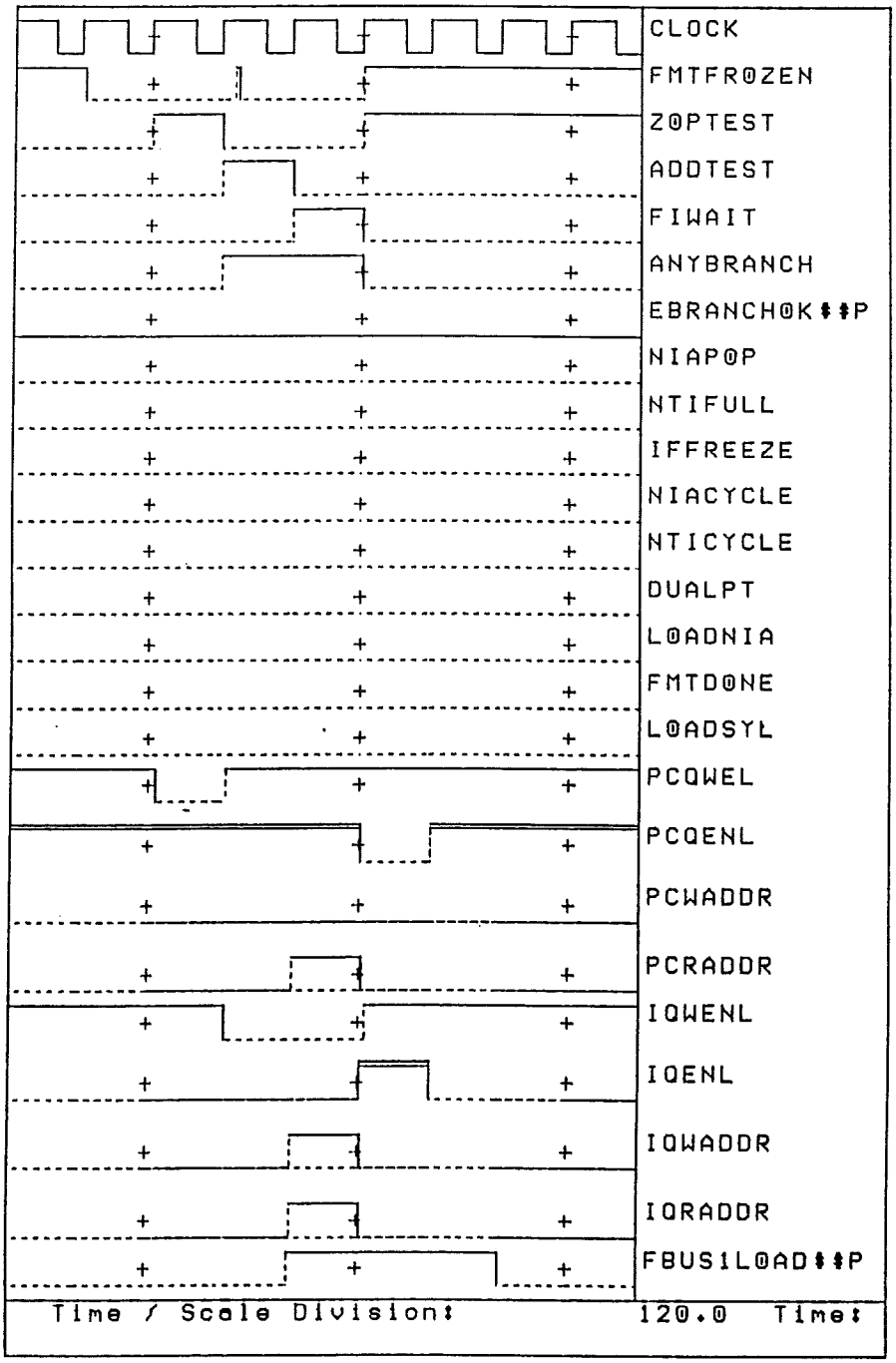
7.2.2 ALIT INSTRUCTION CYCLE



14319.6

FIGURE 7-3 LITERAL INSTRUCTION TIMING

7.2.3 IX IA INSTRUCTION CYCLE



14319.6

FIGURE 7-4 INDEX OR INDIRECT ADDRESSING INSTRUCTION TIMING

7.2.4 HALT NEXT INSTRUCTION CYCLE

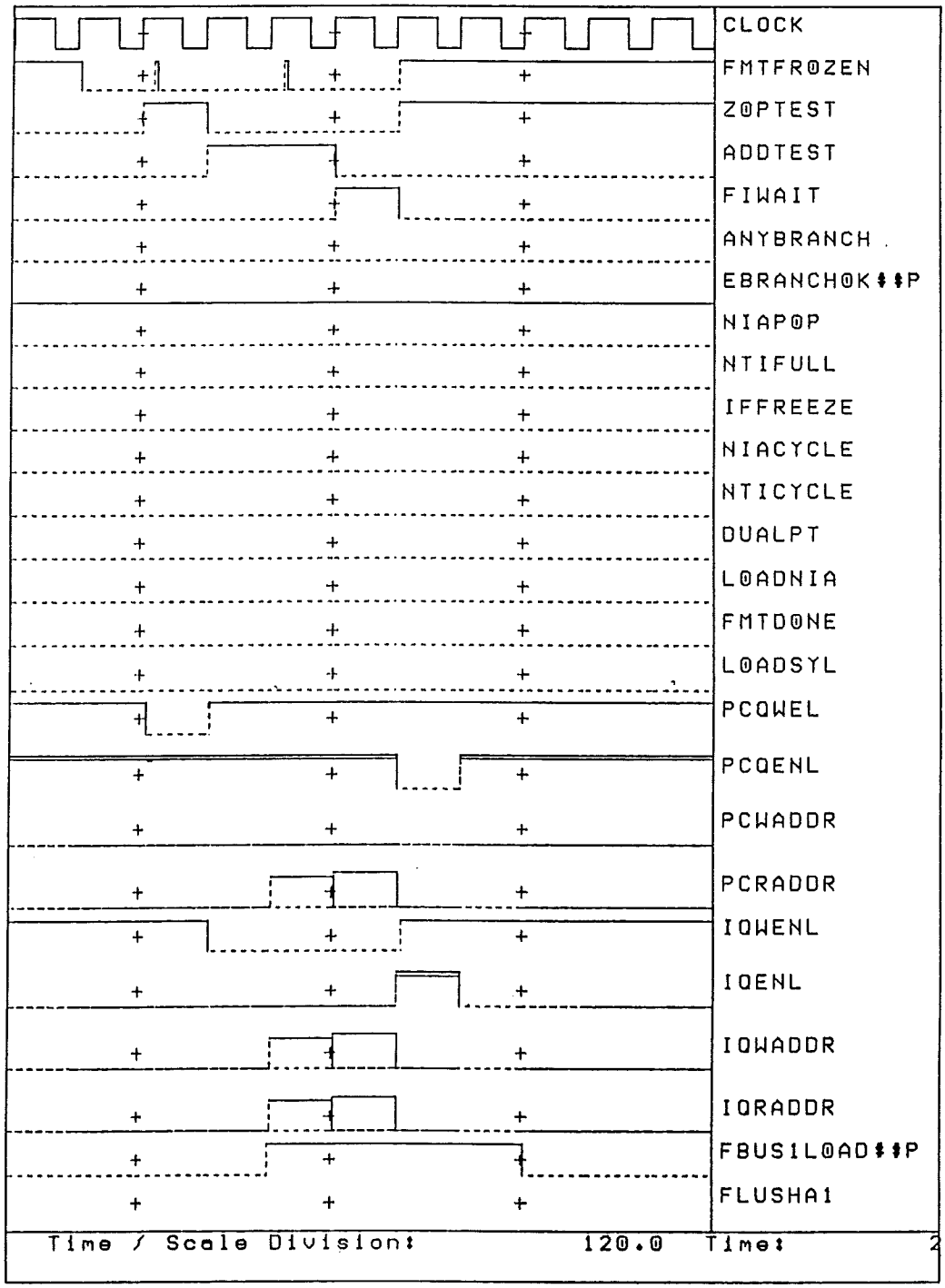


FIGURE 7-5 HALT NEXT INSTRUCTION TIMING

7.2.5 UNCONDITIONAL BRANCH INSTRUCTION CYCLE

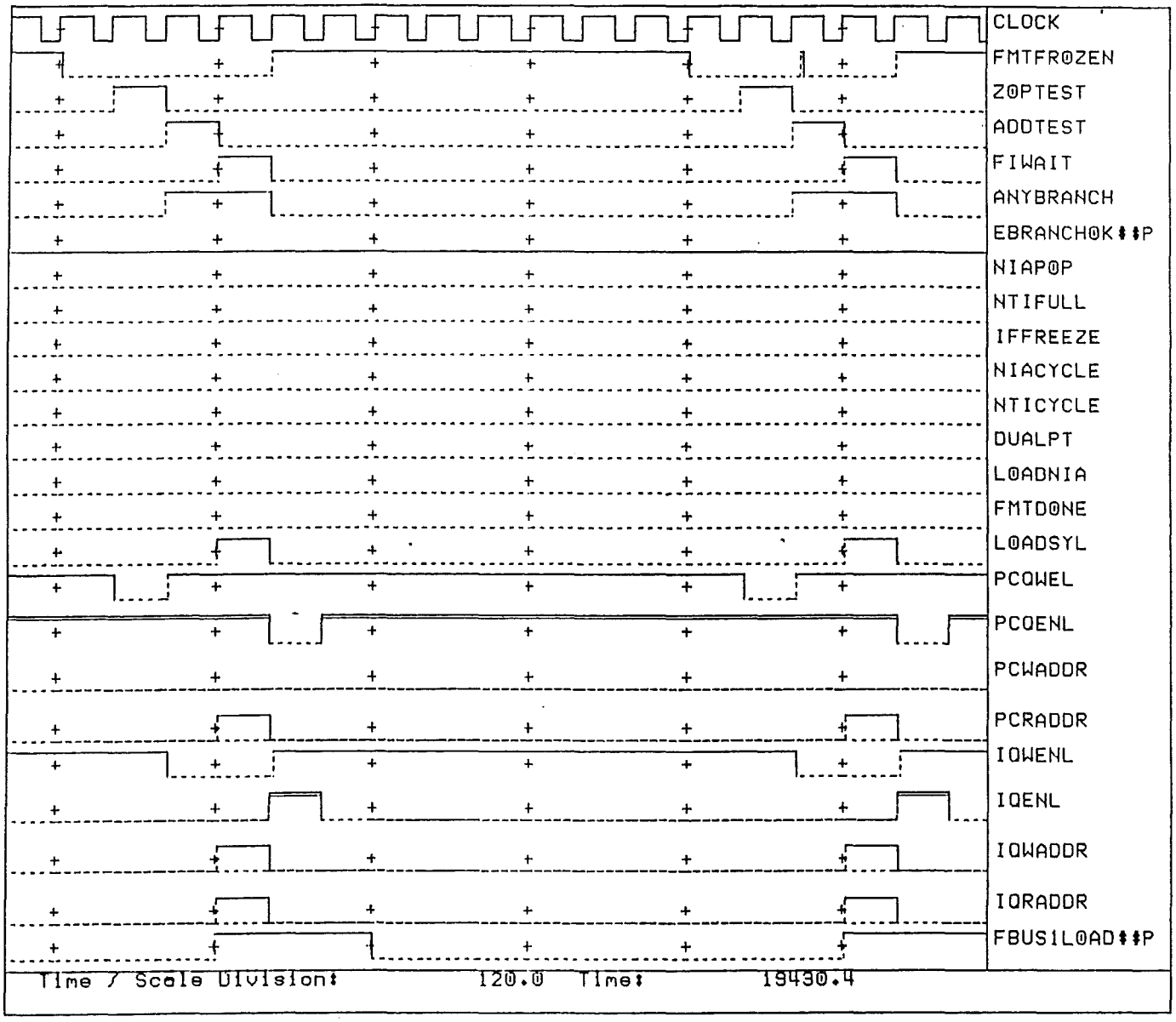


FIGURE 7-6 UNCONDITIONAL BRANCH INSTRUCTION TIMING

7.2.6 PREDICTED NOT TAKEN INSTRUCTION CYCLE

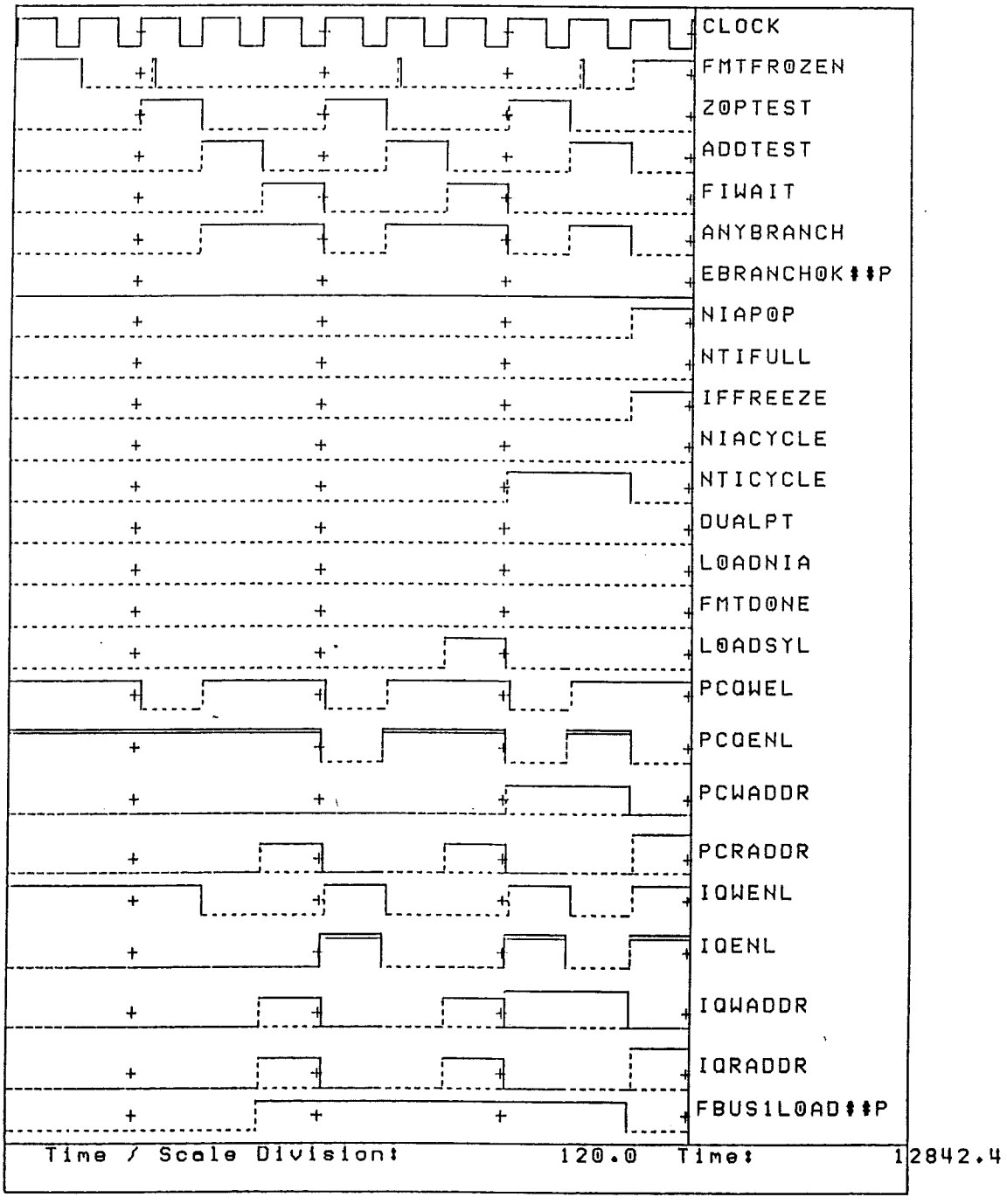


FIGURE 7-7 PREDICTED NOT TAKEN BRANCH INSTRUCTION TIMING

7.2.7 PREDICTED TAKEN INSTRUCTION CYCLE

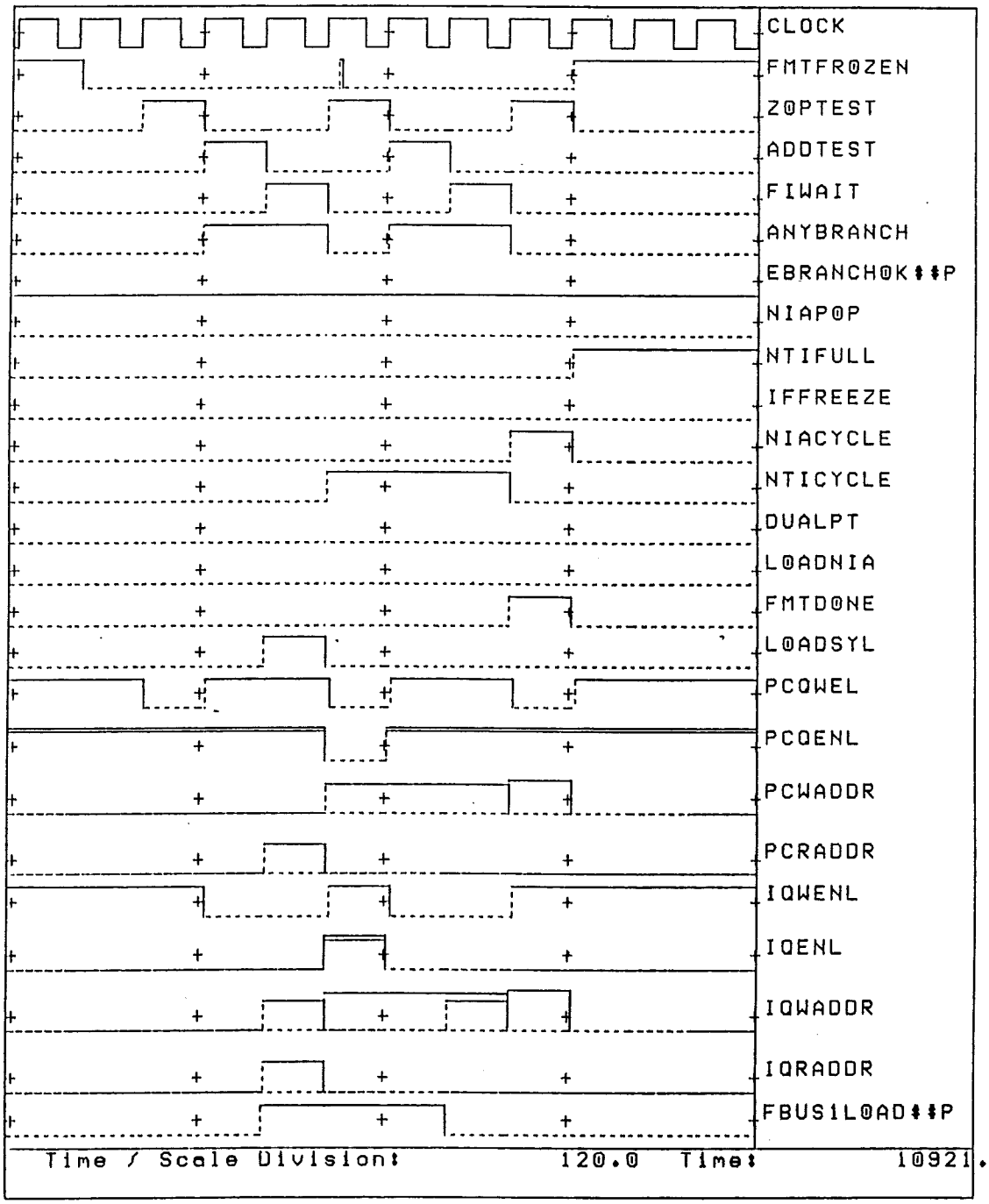


FIGURE 7-8 PREDICTED TAKEN BRANCH INSTRUCTION TIMING

7.2.8 SECOND PREDICTED TAKEN INSTRUCTION CYCLE

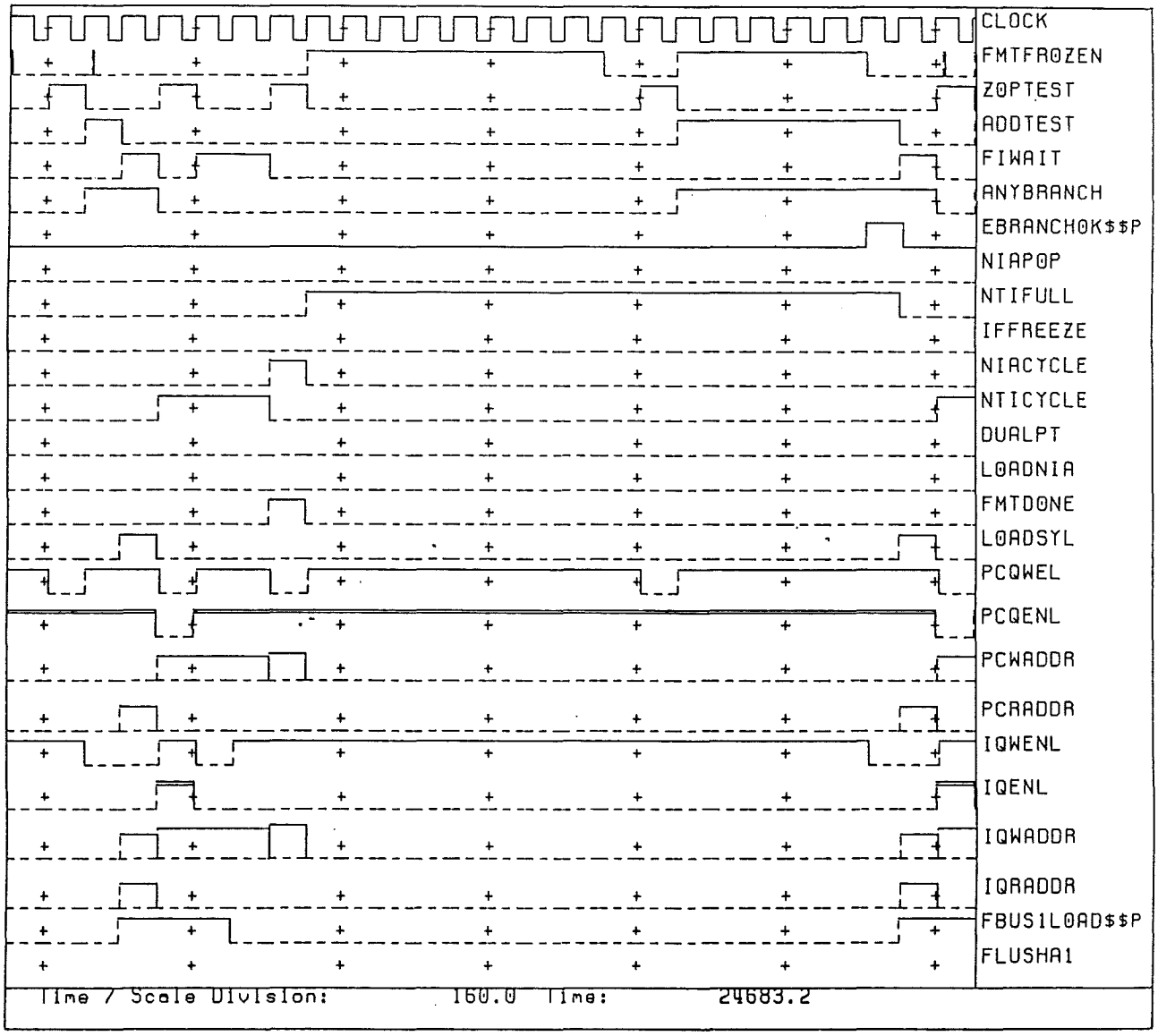


FIGURE 7-9 SECOND PREDICTED TAKEN INSTRUCTION TIMING

7.2.9 DUAL PREDICTED TAKEN INSTRUCTION CYCLE

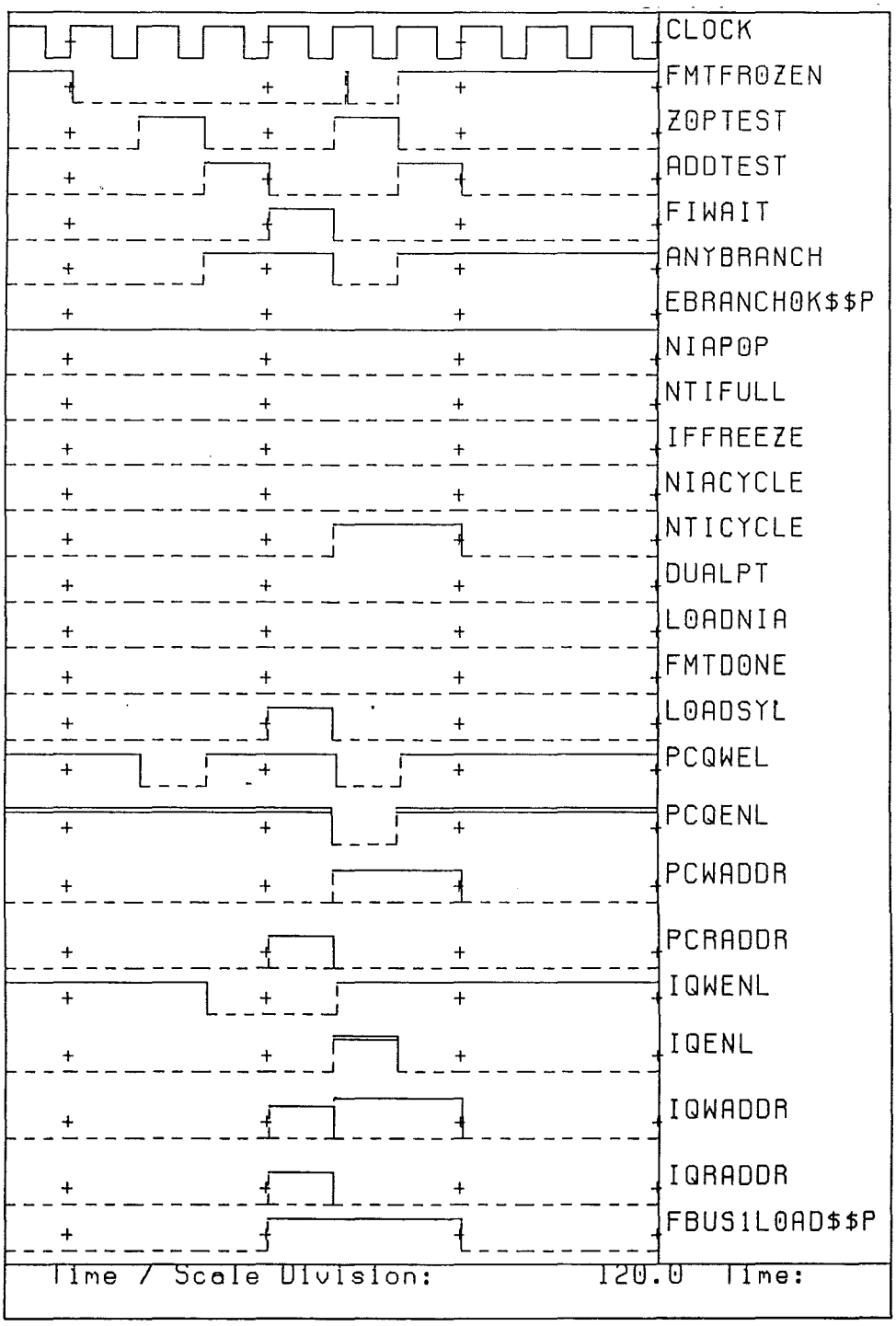


FIGURE 7-10 DUAL PREDICTED TAKEN BRANCH INSTRUCTION TIMING

7.2.10 BAD BRANCH TAKEN INSTRUCTION CYCLE

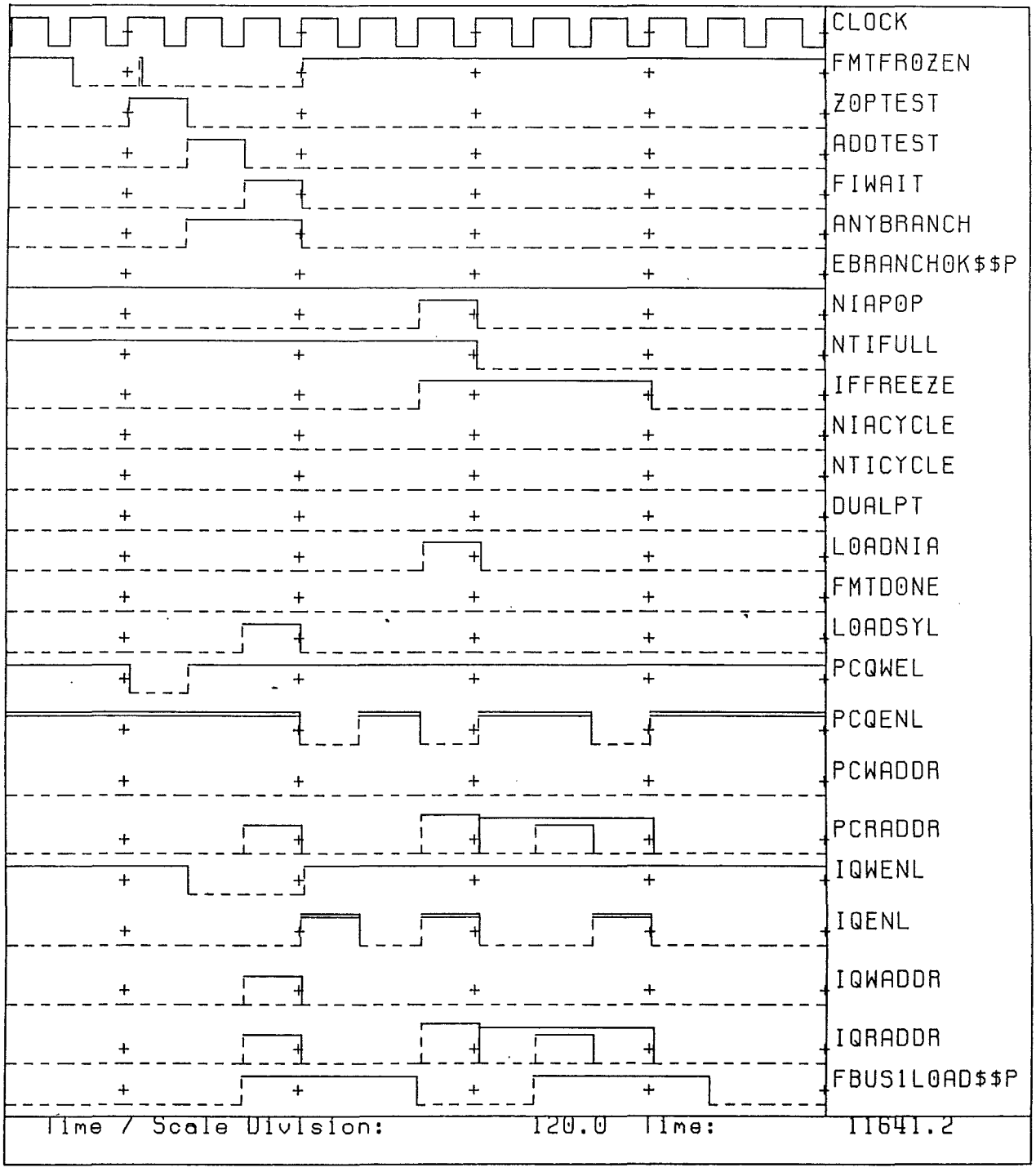


FIGURE 7-11 BAD BRANCH TAKEN INSTRUCTION TIMING

7.2.11 BAD BRANCH NOT TAKEN INSTRUCTION CYCLE

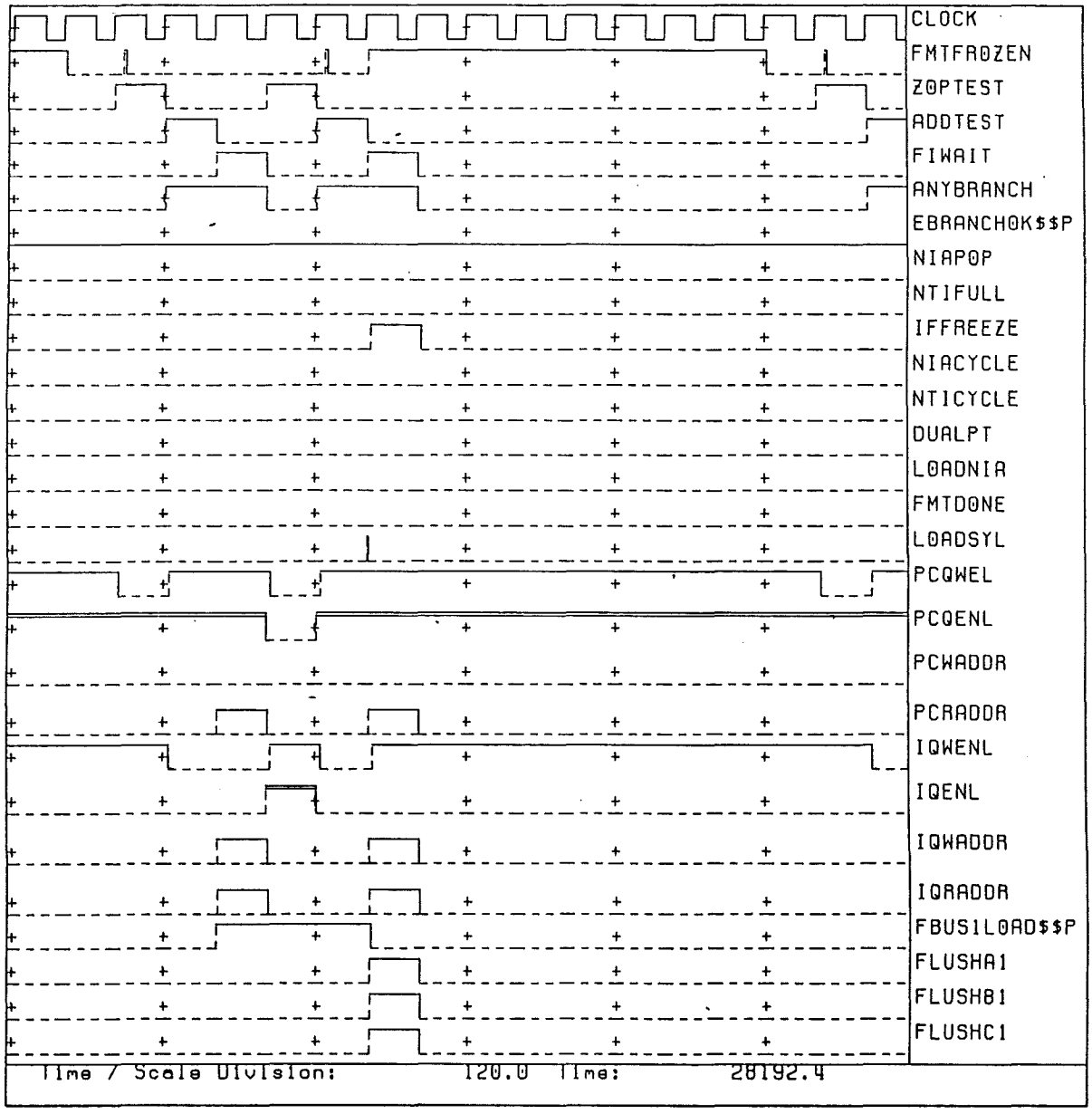


FIGURE 7-12 BAD BRANCH NOT TAKEN INSTRUCTION TIMING

7.2.12 SYSTEM FLUSH INSTRUCTION CYCLE

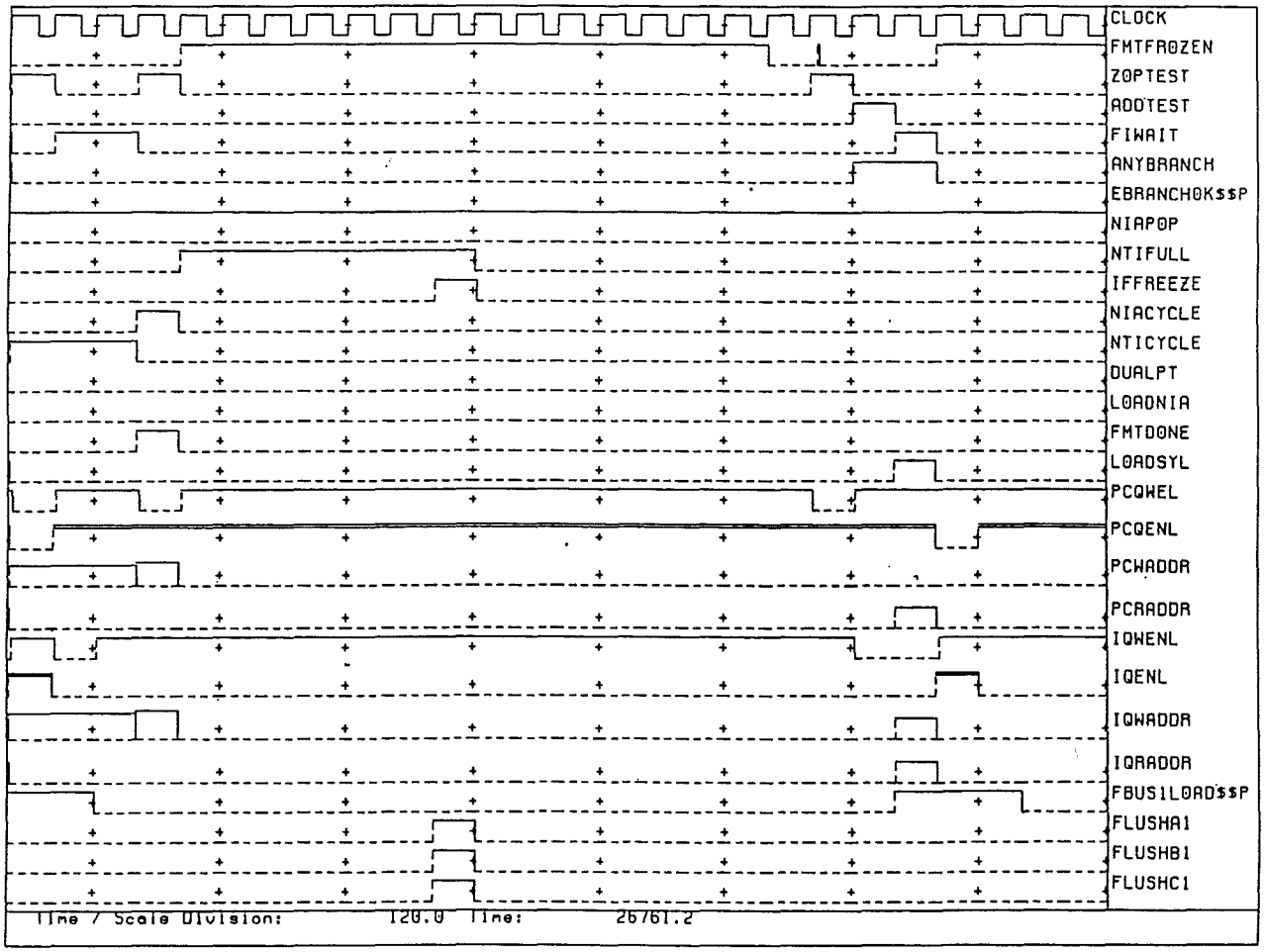


FIGURE 7-13 SYSTEM FLUSH TIMING

8 ERROR DETECTION AND HANDLING

8.1 DETAILED DESCRIPTION OF ERROR DETECTION METHODS

The following PRELIMENARY error detection schemes used in P5 Fetch are based on the assumption that only single stuck-at fault model will be considered. (NOTE: logic that are not covered by any fault detection can be explained by the fact that it will cause too much hardware to achieve the coverage or detailed design implemention has not been started. NO FINAL decision has been made on any of the not covered logic).

8.1.1 IF MODULE

8.1.1.1 IF READER SECTION

ADDRESS COUNTER

This counter is organized as 7 two-to-one multiplexed counters. One data input is loaded from the Address Bus and the other is from the Syllable bus. Since the lowest digit becomes zero when it requests memory, the parity will be different from its original parity. There are two types of parity that will be multiplexed into this Address Counter, then checked one clock later. Parity prediction and undigit check will be used.

COMMAND/TAG DECODER

This logic has only parity check and generation while requesting memory.

READ/WRITE POINTERS FOR THE READER QUEUE

This 4 bit binary up/down counter is actually a 6 by 1 multiplexor feeding the data to flip flops. Parity prediction method will be used.

8.1.1.1 IF READER SECTION (Continued)

QUEUE WRITE PENDING CHECK LOGIC

This logic is organized as 4 sets of 2 bit up/down counters. Parity prediction method will be used. If any cell in the Reader Queue shows an parity error, the associated bank will be disabled for further use. When the operator clears the system, the Maintenance should preset the error bit until the problem is solved.

QUEUE DATA VALIDITY CHECK LOGIC

This logic is organized as 16 bit flip flops. Parity prediction method will be used.

ADDRESS DECODER

This is a 4 to 16 line decoder. Since only one out of 16 outputs is selected to low, the overall parity should always be odd parity. This parity will be inverted to even parity. This method does not need any comparison circuit.

NEXT BANK SELECTION LOGIC

This logic is organized as one 4 bit flip flops_and two 2 bit flip flops. Parity prediction method will be used.

READER QUEUE RAM

Parity checking logic when loading and reading will be used along with a parity bit in the RAM. Since 40 data bits, 5 bits of error are stored in the RAM, storing a parity bit does not cost extra RAM.

RREG1/RREG2

Both registers are orgainized as 42 bit flip flops, forty bits of data, 1 data valid bit and a parity bit. Parity prediction method will be used.

8.1.1.2 IF FORMATTER SECTION

PROGRAM COUNTER

Parity prediction method will be used for this 7 digit counter.

PC INCREMENTER

Duplication method will be used for this 4 bit decimal adder.

ROTATOR

Parity prediction and checking method will be used.

FORMATTER CONTROL STORE RAM

When bit patterns are loaded parity bits are loaded together. When they are read, parity will be checked.

FORMATTER CONTROL LOGIC

Parity prediction method will be used.

8.1.1.3 FBUS MUX

The F-bus multiplexor array will check incoming parity from the S-bus, IQ, and the PC Q, Error Q. The output of the array will be protected by four parity bits across the X digit, FE digits, OPAFBF field and the 40 bit data field.

FBUS CONTROL

The F-bus control array fault detection is to be determined.

8.1.2 OF MODULE

8.1.2.1 OF DATA SECTION

Fault detection will be discussed in the order of data flow in the OF section. Starting with the A-SOURCE in OF, scrathpad which is implemented in MCA array containing the AMUX, BMUX and CMUX will be protected by word parity. RBUSQ and FETCHQ which is in one MCA array and the AUREG implemented in another MCA array will also be protected by word parity. The comparator will be implemented with a self checking algorithm. For fan-out reasons, there will be 2 copies of AMUX output, one copy driving one BCD adder, FBUS interface, comparator, ADDR bus interface and PMUX, the second copy will drive the other BCD adder, the soft and hard PTEST logic in the control card. Word parity check will be performed in the PMUX and the hard PTEST array.

Continuing with the B-SOURCE in OF section, word parity is chosen as the protection mechanism for the following sources: LITFILE, ALEN REG (ALEN register obtained by capturing ALEN-SYL from FBUS2) and OPLEN unit outputs. BMUX output is used to drive the 2 BCD adder. Word parity check is performed in the hard PTEST array on the AMUX output. To aid in covering the CMUX, the BMUX output in the scrathpad array will be generating digit parity on each digit of the BMUX output.

There are only 2 processing elements in P5 OF: BCD adder and the PMUX. Duplicate and comparison is chosen for the BCD adder, output from the 2 BCD adder arrays will be fed into the AUREG array where bit by bit exclusive or on the 2 copies will be performed. The AUREG array also will generate word parity on the BCD adder output for later uses. Parity prediction on output on PMUX array is chosen as fault coverage scheme for the PMUX unit. Word parity will be computed on the output of PMUX to be checked with the predicted parity generated by some other logic in the PMUX array. To aid in covering the CMUX, the PMUX array will be generating digit parity on each digit

of the PMUX output. (A comparator may be added later but it will probably not be covered by any fault detection logic because it will only be used in function test).

The only local storage element in the OF is the scratchpad, and the only data path leading into the scratchpad is through the CMUX (which selects either PMUX or BMUX output into the scratchpad). The CMUX implemented as digit slice will be protected by digit parity prediction scheme, the parity generated from the CMUX will be checked against the predicted parity output generated in parallel. The select lines to CMUX is generated by decoder logic base on the C-SOURCE field in the microword. The C-SOURCE decoder outputs will be checked against the predicted parity output based on the C-SOURCE field. Finally, word parity on predicted CMUX output will be generated from digit parity from PMUX and BMUX in parallel to be stored inside the scratchpad parity bit.

OF will perform word parity checks on data coming from FBUS1, FBUS2 and RBUS. Parity check will also be performed against the FPADDR, FADDR, TAGS and various command fields. Logic not yet covered by any fault detection scheme are: EXT-COMMAND decode logic, RBUSQ read/write logic, input to OPLEN unit, FETCHQ read/write logic and various wires between IF, OF and XM. It is expected that the miscellaneous wires between modules will be covered by some kind of parity.

8.1.2.2 OF CONTROL SECTION

The microword in the control store RAM of OF will have even parity, the parity will be checked at the output of the CREG (control register). In this manner, the control store RAM, wires connecting the RAM to the CREG and the CREG itself will be checked. It should be noted here that the CREG will be the same design as in the P5 XM section, which will be a 27 bit slice of the CREG and with 2 outputs of each microword bit. Parity on sub-fields of the CREG will have their own parity generated by the CREG to be checked at the receiving end. Parity check will be performed on 1 copy of the output of the CREG, Logic not covered by any fault detection at this point includes the second copy of the CREG output and the wires connecting the processing elements to the CREG.

The OF's micro-sequencer will be implemented in 2 MCA options (also shared with P5 XM section), first option providing 4 copies of first 12 bits, the second option providing 8 copies of the low 2 bits of the microword address. Duplicate and comparison method is chosen to protect the sequencer because for fan-out reasons, the OF sequencer will require more than 1 copy of each of the 2 sequencer options, offseting the extra logic required by duplicate and comparison method. The exact alogrithm is as follows: one copy of each microword address will be connected in a round robin manner to the next array where a bit by bit exclusive or function will be performed. Logic not covered by fault detection in this path includes the 3 copies of the first 12 bit microword address from the first sequencer option and the 7 copies of the low 2 bit of the second sequencer option.

Also included in the control section is the PTEST circuitry. PTEST in the OF will be divided into 2 types: soft and hard PTEST. Soft PTEST will be implemented in RAM and parity is chosen as the protection scheme. Hard PTEST will be implemented with MCA array which will contain a 41-bit register to capture data from the AMUX and perform parity check on the A-DATA. Duplication and comparison is being used to cover the hard PTEST. Logic not covered in this path includes PTREG (PTEST register), hard PTEST logic, OPCODE lookup structure and the INT-COMMAND decode logic.

8.1.2.3 OF LOCK UNIT

As discussed before, there are three distinct array designs in the lock unit. The fault detection coverage on these arrays vary and different methods are used to achieve this coverage as discribed below.

The BCD adder will be covered by duplication and comparison (duplication and comparision will be achieved by the AUREG option in the OF data section). Since a single bit fault can cause multi bit errors in the adder logic, parity generation can not be used to detect errors, rather every output needs to be compared against the duplicated logic and any discrepancies will be reported as an error.The cost of coverage in this case is about one extra gate array location on the board.

The LOKCMP which consists of the comparator and the begin and end tables is used eight times as discussed before. Two methods are used to detect faults on this array. The begin and end tables which consist of about 1/3 of the logic are mainly registers and are covered by parity. The rest of this logic is mainly devoted to the comparators and about 1/2 of the gates on this logic are covered by parity. In addition, the comparators are checked as a whole for errors which activate both outputs. This checking increases the fault detection in some cases.

Finally, the LOKCTL array which is more important in terms of fault detection, is mainly covered by parity, since most of this array consists of registers. The rest of this array are not covered and they are basically buffers and combinatorial logic used along with registers for counters.

The command lines will be checked by parity in the control array. The two begin and end address inputs to the lock unit will also be checked for single bit errors by parity.

8.1.2.4 OPLEN SECTION

FLUT

The Fetch Look-up Table will be comprised of 256x4 rams, fault detection will be parity across the output of the table. This parity will be checked in the Oplen input array and again in the Oplen output array.

CF CALCULATOR

The CF calculator will generate the CF field by either adding, subtracting, or passing the AF and BF fields. The duplication and comparison method of fault detection will be implemented.

LITERAL GENERATOR

The Literal Generator will be protected by duplication.

A LENGTH CALCULATOR

The A length calculator consists of 4kx1 rams containing a look-up table, this output of the rams will be protected by parity. The address input to the rams will not be protected do to need for the use of all 12 address lines. No larger rams are available for input parity protection.

B LENGTH CALCULATOR

The B length calculator is identical to the A length calculator.

C LENGTH CALCULATOR

The C length calculator is identical to the A length calculator.

NUMBER EXTENDED SYLLABLES

The number of extended syllables flag will be generated by gating the Ae, Be, and Ce to produce an encoded two bit field. This will be protected by duplication.

8.2 INSTRUCTION RELATED ERRORS

Certain instruction related errors, e.g. invalid AF, undigits in addresses etc., will be detected by fetch. When such errors occur, fetch will attempt to notify XM through various means. Since fetch only parses instructions and resolves addresses, it will only be responsible for generic errors. For example, fetch will not flag an invalid instruction error even if the three address controllers in MVL instruction are not the same; Cc will always be used to calculate the length. Another example will be a value of A1 in the AF of a VEN instruction; while only B1, B2 and B3 are valid AF values, fetch will not flag an invalid instruction error. This implies that XM will need to detect non-generic errors related to the instructions.

The following errors will be detected by fetch :

Type of Error	AEX value	XM notified through
Invalid Arithmetic	11	modified opcode (1A)
due to overflow or underflow during index register add		
Undigits in index register	12	modified opcode (1A)
due to undigits detected in index register		
Invalid base indicant	13	modified opcode (1A)
due to invalid base indicant in index register (8-F)		
Instr fetch B/L error	21	modified opcode (1A)
due to encoded result (SS digits) from IF		
Addr resolutiion B/L error	22	modified opcode (1A)
due to address error tag returned from MCACM during INFL or IA		

8.2 INSTRUCTION RELATED ERRORS (Continued)

Operand Read B/L error	24	modified opcode (1A)

due to base/limit error returned for operand read (e.g. TOS, branch address, environment number, etc.)

Instr fetch address undigit	31	modified opcode (1A)

due to encoded result (SS digits) from IF

Addr resolution address undigit	32	modified opcode (1A)

due to undigit in offset during index resolution

Operand Read undigit error	34	modified opcode (1A)

due to undigit error returned for operand read (e.g. TOS, branch, address, environment number, etc.)

Branch addr resolution addr undigit	42	V digit in syllable

due to various errors detected during branch address resolution

Invalid address controller	03	modified opcode (1E)

not always detected by OF, XM is mainly responsible for this error

Invalid AF or BF	20	modified opcode (1E)

due to invalid field length on AF or BF, ALIT after infl or literal on BF

Literal not allowed	21	modified opcode (1E)

due to literal specification in certain opcodes

8.2 INSTRUCTION RELATED ERRORS (Continued)

Invalid literal	22	modified opcode (1E)
due to invalid literal		
Invalid INFL	23	modified opcode (1E)
due to invalid infl		

In addition to the above, some invalid AF, BF variant errors will be detected by fetch. This is because fetch in the process of performing certain functions required by the instruction must determine the validity of the variants. As the OF microcode progresses, more information will be provided.

8.3 HARDWARE RELATED ERRORS

To be specified.

9 SELF CHECKING

9.1 OF MODULE SELF CHECKING OPERATIONS

The OF module has the ability to initiate various self checking operations to determine the status of the fault detection circuitry and to provide better testability on the module. During self check mode, errors are forced in various parts of the system to activate all the error indicators. If an error indicator does not get set when error is forced, a dead freeze condition will be reported. This protects all of the error indicators against stuck at faults. Current implementation, required the OF and XM to be in sync during self check mode. When a self check opcode is detected, OF will wait for XMEVENT to begin self check. When it is done, OF will signal XM via the OFEVENT. XM must not flush while the OF is in self test mode as errors may occur.

To allow external interface signals to be simulated for more thorough testing, the OF allows write back on Fetch and RBUS queues. Due to the implementation restriction, FBUS2 write back is not provided. OF module can also initiate system or fetch flushes via an internal command. This allows testing of flush handling logic and routines. To obtain a better test coverage on the lock unit, a command is implemented to simulate an opcomp from the XM. Locks can be posted and popped without external intervention.

10 SYSTEM MAINTENANCE INTERFACE

10.1 MAINTENANCE FEATURES

10.1.1 SHIFT CHAINS

10.1.1.1 INSTRUCTION FETCH DATA

```
CHAIN FI;                                  % 567 BIT CHAIN LENGTH

%  BIT LENGTH IS 567
%
%   OPTION      WIDTH  LOCATION    SDI SDO      SDI-NET
%   REG12         51     P2K6      B09 D14     ASHIFTINF01P
%   ADRCTR        19     M8K6      P12 B14     SDATM8K6
%   FBMUX         49     M8N0      P02 H02     SDATM8N0
%   FBMUX         49     F6N0      P02 H02     SDATF6N0
%   FBCTRL        38     D2N0      R08 N15     SDATD2N0
%   REG12         51     F6K6      B09 D14     SDATF6K6
%   RQCNTL2       64     F6I2      C15 G02     SD0R1VLD
%   FMTCTA2       41     F6F8      D15 F15     ROTSELH-2  SDATF6F8
%   FMTCTB2       46     F6D4      P10 P14     SDATF6D4
%   RQCNTL2       64     I0I2      C15 G02     MCPH
%   ADRCTR        19     M8I2      P12 B14     SDATM8I2
%   FMTCTB2       46     M8F8      P10 P14     SDATM8F8
%   PC            30     M8D4      A12 G14     SDATM8D4
%   CLKMNT               R6F8      R08 L13     SDATR6F8
%                                              X1SHIFT0UT$P

%  PC ARRAY

 ERRORPC          1;     % PC error
 FAULTPC          1;     % PC predicted parity
 PC27.4          24;     % PC register
 PC3.0            4;
% 30 BITS

%  FORMATTER CNTLB (SLAVE)

 MCP2             1;     % Mcp
 OPTEST2          1;     % Optest
```

```
      PCCTRL2          1;    % Pc ctrl
      CHECKER2         4;    % Checker check
      RQNEWPCH2        1;    % RQ newpc NTI cycle
      FMTFAULT2        1;    % Formatter fault
      NTINOTAVL2       1;    % NTI not available
      FMTFRZ2          1;    % Formatter B frozen
      PCSBUSEQ2        1;    % PC Sbus equal
      IQWADR.A         2;    % Write address
      SYLCOUNT2        2;    % Syllable count
      PTON2            1;    % Predicted taken on
      DUALPT2          1;    % dual pred branch f/f
      NIACYCLE2        1;    % NIA cycle (load niapc)
      NTICYCLE2        1;    % NTI cycle
      DUCQDLY2         1;    % delayed
      FMTSEQ2          3;    % formatter sequencer
      BRANCHOP2        2;    % branch op type(loadsyl pc)
                             % 3 for unconditional
                             % 2 for Pred. Taken
                             % 1 for Pred Not Taken
                             % 0 for no branch
      HLTNXT2          1;    % halt next
      INVOP2           1;    % invalid OP
      PCINCR2/2        1;    % PC increment for OP
      PCINCR2/1        1;    % PC increment for OP
      PCINCR2/0        1;    % PC increment for OP
      DISAFBF2         1;    % Blank AFBF
      EXTEND           1;    % extend address test
      IX               1;    % index test
      IA               1;    % indirect address test
      EXCEPTION        1;    % exception test
      ADTSTS1/3        1;    % addtst1 bit 3 (loadreg1)
      ADDTST3/0        1;    % addtst3 bit 0
      CLSTST3.0        4;    % OPCLASS test bits 3:0
      ERROR2           2;    % error RBUS
      ODDPC2           1;    % ODD pc
      NIAVLD2          1;    % NIA valid
    % 46 BITS

    %  ADDR COUNTER  (location M8I2)

      ADRCNTA         16;    % Fetch address count
      ADRERRA          1;    % error adrctr
      ADRLODENA        1;    % adrctr load enable
```

```
      ADDRPRTYA          1;     % adrctr parity

     % 19 BITS

     %  RQCNTL (MASTER)

      RQRAD2             4;     % Read queue read addr
      RQWAD2             4;     % Read queue write addr
      IDEQIF2            1;     % Rbus tag equal IF
      FLUSHB2            1;     % Flush encode bit
      FLUSHC2            1;     % Flush encode bit
      NIAPOP2            1;     % Bad branch taken command
      FLUSHA2            1;     % Flush encode bit
      FLUSHSEL2          1;     % flush select for Fbctrl
      FLUSHGEN2          1;     % flush general "    "
      AWQFUL2            1;     % AWQUEUE is full
      IFTESTEN2          1;     % Enable diagnostic test
      MREG2             20;     % Maintenance register
                                %   lower 20 bits
      RQFULL2            1;     % RQ is full
      BANKMSTR           2;     % mem req bank pointer
      SCOUNT2            2;     % mem req pointer
      LSTBNKMSTR         2;     % mem req. last bank pntr.
      MREQFULL2          1;     % memory requset counter
      MREQCOUNT2         2;     % Outstanding mem request
      BNKAVLMSTR         8;     % Bank availability reg.
      VALID2             8;     % Ram cell data valid
      RQFAULT2           1;     % Read Queue fault
     % 64 BITS

     %  FORMATTER CNTLB  (MASTER)

      MCP1               1;     % Mcp
      OPTEST1            1;     % Optest
      PCCTRL1            1;     % Pc ctrl
      CHECKER1           4;     % checker check
      RQNEWPCH1          1;     % RQ newpc NTI cycle
      FMTFAULT1          1;     % Fomatter fault
      NTINOTAVL1         1;     % NTI not available
      FMTFRZ1            1;     % Formatter B frozen
      PCSBUSEQ1          1;     % PC Sbus equal
      IQWADR.B           2;     % Write address
      SYLCOUNT1          2;     % Syllable count
```

```
      PTON1              1;     % Predicted taken on
      DUALPT1            1;     % dual pred branch f/f
      NIACYCLE1          1;     % NIA cycle
      NTICYCLE1          1;     % NTI cycle
      DUCQDLY1           1;     % delayed
      FMTSEQ1            3;     % formatter sequencer
      BRANCHOP1          2;     % branch op type
                                % 3 for unconditional
                                % 2 for Pred. Taken
                                % 1 for Pred Not Taken
                                % 0 for no branch

      HLTNXT1            1;     % halt next
      INVOP1             1;     % invalid OP
      PCINCR1/2          1;     % PC increment for OP
      PCINCR1/1          1;     % PC increment for OP
      PCINCR1/0          1;     % PC increment for OP
      DISAFBF1           1;     % Blank AFBF
      EXTENDM            1;     % extend address test
      IXM                1;     % index test
      IAM                1;     % indirect address test
      EXCEPTIONM         1;     % exception test
      ADTSTM1/3          1;     % addtst1 bit 3 (loadreg1)
      ADDTST1/0          1;     % addtst1 bit 0
      SYLTST/3           1;     % syltst bit 3
      SYLTST/2           1;     % syltst bit 2
      SYLTST/0           1;     % syltst bit 0
      ADDTST1/2          1;     % addtst1 bit 2
      ERROR1             2;     % error RBUS
      ODDPC1             1;     % ODD pc
      NIAVLD1            1;     % NIA valid
    % 46 BITS

    %  FORMATTER CNTLA

      ROTSEL             3;     % Rotator selector
      FIOP               8;     % OP register
      OPPARBIT           1;     % OP parity bit
      FMTAFAULT          1;     % FMTCTA fault
      ALITF              1;     % Literal case on adddress
      ALITX              1;     % A literal flip flop
      MOD8               1;     % MOD 8 selection case
      LITENF             1;     % AF literal allowed
```

```
      IFTEST            1;    % IF test, enables POP/SHOVE
      FLIPTST           1;    % Flip from FBCTRL
      SI                1;    % Single instruct flag
      FIWAIT            1;    % FIWAIT flag, zero addr
      NODFRZ            1;    % No deadfreeze, disables
      IFBROKEN1         1;    % IF board error
      FCTLERR1          1;    % Fbus control error
      FTCHBRKEN1        1;    % Fetch broken
      SBUSERR1          1;    % SBUS error
      ROTERR1           1;    % Rotator error
      FMTCSPERR1        1;    % Fmt ctrl store prtyerr
      TAGERR1           1;    % RBUS tag error
      IFBROKEN2         1;    % duplicate
      FCTLERR2          1;    % duplicate
      FTCHBRKEN2        1;    % duplicate
      SBUSERR2          1;    % duplicate
      ROTERR2           1;    % duplicate
      FMTCSPERR2        1;    % duplicate
      TAGERR2           1;    % duplicate
      REG2FAULT         1;    % REG12 #2 error
      IQREAD            1;    % IQ read normal
      ADDR1ERR          1;    % ADRCTR #1 error
      ADDR2ERR          1;    % ADRCTR #2 error
      PCFAULT           1;    % PC error
    % 41 BITS

    %  RQCNTL LS


      RQRAD1            4;    % Read queue read addr
      RQWAD1            4;    % Read queue write addr
      IDEQIF1           1;    % Rbus tag equal IF
      FLUSHB1           1;    % Flush encode bit
      FLUSHC1           1;    % Flush encode bit
      NIAPOP1           1;    % Bad branch taken command
      FLUSHA1           1;    % duplicate
      FLUSHSEL1         1;    % duplicate
      FLUSHGEN1         1;    % duplicate
      AWQFUL1           1;    % AWQUEUE is full
      IFTESTEN1         1;    % Enable diagnostic test
      MREG1            20;    % Maintenance register
                              %   lower 20 bits
      RQFULL1           1;    % RQ is full
```

```
      BANKSLV          2;     % mem req bank pointer
      SCOUNT1          2;     % mem req pointer
      LSTBNKSLV        2;     % mem req. last bank pntr.
      MREQFULL1        1;     % memory requset counter
      MREQCOUNT1       2;     % Outstanding mem request
      BNKAVLSLV        8;     % Bank availability reg.
      VALID1           8;     % Ram cell data valid
      RQFAULT1         1;     % Read Queue fault
     % 64 BITS

     %  REG12 ARRAY MS

      REG1VALA         1;     % duplicate
      REGFA            1;     % Rbus parity error
      REG2FA           1;     % Reg2 parity fault
      REGRQFA          1;     % RQ parity fault
      REGRBUSFA        1;     % Rbus parity fault
      REG1FA           1;     % Reg1 parity fault
      ERRORF           5;     % ERROR FAULT     ????????? check this
      REG2.39          1;     % bit
      REG2.38          1;     % bit
      REG2.37          1;     % bit
      REG2.36          1;     % bit
      REG2.35          1;     % bit
      REG2.34          1;     % bit
      REG2.33          1;     % bit
      REG2.32          1;     % bit
      REG2.31          1;     % bit
      REG2.30          1;     % bit
      REG2.29          1;     % bit
      REG2.28          1;     % bit
      REG2.27          1;     % bits 27:25 of reg2
      REG2.26          1;
      REG2.25          1;
      REG2.24          1;     % bit
      REG2.23          1;     % bit
      REG2.22          1;     % bit
      REG2.21          1;     % bit
      REG2.20          1;     % bit
      REG1/39.36       4;     % bits 39:36 of register 1
      REG1/35.32       4;     % bits 35:32 of register 1
      REG1/31.28       4;     % bits 31:28 of register 1
      REG1/27.24       4;     % bits 27:24 of register 1
```

```
      REG1/23.20        4;     % bits 23:20 of register 1
     % 51 BITS

     %  FBCTRL ARRAY

%     PCOVERFLOW        1;     % Program counter overflow error
%     above is for FBCTRL2
      NUMSYLLD2         1;     % duplicate
      NTIVLD2           1;     % duplicate
      NTICTRL2          1;     % duplicate
      FREGVLD2          1;     % duplicate
      FLIP2             1;     % duplicate
      EQLFF2            1;     % duplicate
      ADDRCNTR2         2;     % duplicate
      XMPTR2            2;     % duplicate
      OFPTR2            2;     % duplicate
      IFPTR2            2;     % duplicate
      PAGEAVAIL2        1;     % duplicate
      INCIFPTR2         1;     % duplicate
      NUMSYLLD          1;     % number of syl stag f/f
      NTIVLD            1;     % NTI QUEUE VALID
      NTICTRL           1;     % Not Taken Inst CYCLE F/F
      FREGVLD           1;     % FREG HAS VALID DATA
      FADDR             2;     % Fetch Addr Reg (FADDR)
      FLIP              1;     % flip staging reg
      EQLFF             1;     % addr cntr = number syl
      ADDRCNTR          2;     % syllable counter
      SYLTAG            1;     % syl tag, burn clock
      NUMSYL            2;     % number of syl reg
      XMPTR             2;     % XM PAGE POINTER
      OFPTR             2;     % OF PAGE POINTER
      IFPTR             2;     % IF PAGE POINTER (FPADDR)
      PAGEAVAIL         1;     % Fetch Page Avail
      INCIFPTR          1;     % Sets on FLIP, inc IFptr
                               % upon SHOVE
      FLIPMASK          1;     % Sets on first POP
     % 38 BITS  will be 39

     %  FBUSMUX ARRAY MS

      SPAREC            2;     % spare f/f
      FORTY             1;     % prty 'PARIN' & (39:28)
      FEDIG             1;     % prty (47:40) F/F
```

```
      XDIG              1;     % most sig 8 bit prty(55:48)
      FREGL52           1;     % BIT 52
      FREGL53           1;     % BIT 53
      FREGL54           1;     % BIT 54
      FREGL55           1;     % BIT 55
      FREGL51.28       24;     % BIT 51:28
      SPAREB            8;     % spare f/f
      SPAREF            4;     % spare f/f
      PRTYERRL          1;     % PARITY ERROR F/F
      PARIN             1;     % PARITY IN STAGING REG.
      PARALL            1;     % OVERALL PARITY F/F
      CHECKR            1;     % CHECKER CHECK INPUT F/F
    % 49 BITS

    %  FBUSMUX ARRAY LS

      OPAFBF            1;     % BITS 23:0 PARITY OF FREG
      OPPAR             1;     % 23:16 PRTY FREG
      SPAREE            3;     % spare f/f
      FREGL24           1;     % BIT 24
      FREGL25           1;     % BIT 25
      FREGL26           1;     % BIT 26
      FREGL27           1;     % BIT 27
      FREGL23.0        24;     % BIT 23:0
      INVLDOP           8;     % INVALID OP REGISTER
      BEXCEPTL          1;     % 'B' EXCEPTION F/F
      AEXCEPTL          1;     % 'A' EXCEPTION F/F
      ALITL             1;     % ALITERAL F/F
      EXCEPTPARL        1;     % EXCEPTION PARITY F/F
      SPARED            3;     % spare f/f
      SPAREA            1;     % spare f/f
    % 49 BITS

    %  ADDR COUNTER  (location M8K6)MS

      ADRCNTMSDB        4;     % Fetch address MSD
      ADRDIG96          4;     % Fetch address digit 96
      ADRCNTB           8;     % Fetch address count
      ADRERRB           1;     % error adrctr
      ADRLODENB         1;     % adrctr load enable
      ADDRPRTYB         1;     % adrctr parity

    % 19 BITS
```

```
%  REG12 ARRAY  LS

 REG1VAL            1;     % REG1 VALID
 REGF               1;     % Rbuserr vec fault
 REG2F              1;     % Reg2 parity fault
 REGRQF             1;     % RQ parity fault
 REGRBUSF           1;     % Rbus parity fault
 REG1F              1;     % Reg1 parity fault
 SPAREZ             5;     % spare f/f
 REG2/19.16         4;
 REG2/15.12         4;
 REG2/11.8          4;
 REG2/7.4           4;     % least sig 16 bits REG2
 REG2/3.0           4;     % least sig 4 bits REG2
 REG1/19.16         4;     % bits 19:16 of register 1
 REG1/15.12         4;     % bits 15:12 of register 1
 REG1/11.8          4;     % bits 11:8 of register 1
 REG1/7.4           4;     % bits 7:4 of register 1
 REG1/3.0           4;     % bits 3:0 of register 1

% 51 BITS

% Total length 567 bits
CHNEND;                    % CHAIN END


REGLIST  FI;

RQCELLVLD           2;     % Read q cell vaild
  VALID2;
  VALID1#
RQRAD               2;     % Read Q read address
  RQRAD2;
  RQRAD1#
RQWAD               2;     % Read Q write address
  RQWAD2;
  RQWAD1#
IFERRVEC           20;     % IF ERROR VECTOR
  ADDR2ERR;
  ADDR1ERR;
  FCTLERR2;
  FCTLERR1;
```

```
      ERROR2;
      ERROR1;
      FMTCSPERR2;
      FMTCSPERR1;
      FMTFAULT2;
      FMTFAULT1;
      ROTERR2;
      ROTERR1;
      RQFAULT2;
      RQFAULT1;
      SBUSERR2;
      SBUSERR1;
      TAGERR2;
      TAGERR1;
      ODDPC2;
      ODDPC1#
    RQFULL            2;    % Read Queue full
      RQFULL2;
      RQFULL1#
    MEMREQCNT         2;    % Memory request pointer
      SCOUNT2;
      SCOUNT1#
    MREQFULL          2;    % Memory request full
      MREQFULL2;
      MREQFULL1#
    OUTMREQCNT        2;    % Outstanding memory request
      MREQCOUNT2;
      MREQCOUNT1#
    IDEQIF            2;    % ID equal to IF (comm bus)
      IDEQIF2;
      IDEQIF1#
    AWQFULL           2;    % AW que full
      AWQFUL2;
      AWQFUL1#
    INVOP             2;    % Invalid Op
      INVOP2;
      INVOP1#
    REG1VALID         2;    % REG1 valid
      REG1VALA;
      REG1VAL#
    XADRREG           4;    % XADRBUS(31:0) register
      ADRCNTMSDB;
      ADRDIG96;
```

```
      ADRCNTB;
      ADRCNTA#
    ADRLOAD          2;    % ADRCTR loadsyl,
      ADRLODENB;           % loadnia, loadmem
      ADRLODENA#
    PCINCR           3;    % PC increment for OP
      PCINCR2/2;
      PCINCR2/1;
      PCINCR2/0#
    REG1            10;    % REG1
      REG1/39.36;
      REG1/35.32;
      REG1/31.28;
      REG1/27.24;
      REG1/23.20;
      REG1/19.16;
      REG1/15.12;
      REG1/11.8;
      REG1/7.4;
      REG1/3.0#
  REG2              25;    % REG2
      REG2.39;
      REG2.38;
      REG2.37;
      REG2.36;
      REG2.35;
      REG2.34;
      REG2.33;
      REG2.32;
      REG2.31;
      REG2.30;
      REG2.29;
      REG2.28;
      REG2.27;
      REG2.26;
      REG2.25;
      REG2.24;
      REG2.23;
      REG2.22;
      REG2.21;
      REG2.20;
      REG2/19.16;
      REG2/15.12;
```

```
      REG2/11.8;
      REG2/7.4;
      REG2/3.0#
    FMTFRZ              2;     % Formatter freeze
      FMTFRZ2;
      FMTFRZ1#
    FMTSEQ              2;     % Formatter Sequencer
      FMTSEQ2;
      FMTSEQ1#
    BRANCHOP            2;     % Branch Op
                               % F for unconditional
      BRANCHOP2;               % A for Pred. Taken
                               % 5 for Pred Not Taken
      BRANCHOP1#               % 0 for no branch
    OPTEST              2;     % OP syllable test
      OPTEST2;
      OPTEST1#
    PCCTRL              2;     % PC control
      PCCTRL2;
      PCCTRL1#
    PC                  2;     % Program counter
      PC27.4;
      PC3.0#
    IQWADR              2;     % IQ write address
      IQWADR.B;
      IQWADR.A#
    FBUS1DATL           7;     % FBUS1(52:0) register
      FREGL52;
      FREGL51.28;
      FREGL27;
      FREGL26;
      FREGL25;
      FREGL24;
      FREGL23.0#
    FREGVALID           2;     % FREG valid (OFREQVLD)
      FREGVLD2;
      FREGVLD#
    SYLEQL              2;     % syllable tag equal
      EQLFF2;
      EQLFF#
    FADRCNTR            2;     % Fetch address counter
      ADDRCNTR2;
      ADDRCNTR#
```

```
    FADDR              1;     % Fetch page entry
      FADDR#
    FPADDR             2;     % IF page, write address
      IFPTR2;
      IFPTR#
    FLIPPAGE           2;     % FLIP
      FLIP2;
      FLIP#
    XMPAGE             2;     % XM page pointer
      XMPTR2;
      XMPTR#
    OFPAGE             2;     % OF page pointer
      OFPTR2;
      OFPTR#
    PAGEAVAL           2;     % Fetch page available
      PAGEAVAIL2;
      PAGEAVAIL#
    INCIFPAGE          2;     % Increment IF page pointer
      INCIFPTR2;
      INCIFPTR#
    NTINAVAL           2;     % NTI not available
      NTINOTAVL2;
      NTINOTAVL1#
    NIAPOP             2;     % NIAPOP command
      NIAPOP2;
      NIAPOP1#
    NIAVALID           2;     % NIA Valid
      NIAVLD2;
      NIAVLD1#
    NTIVALID           2;     % NTI valid
      NTIVLD2;
      NTIVLD#
    NTIPOP             2;     % NTI POP cycle (B B TAKEN)
      NTICTRL2;
      NTICTRL#
    NIACYCLE           2;     % NIA cycle
      NIACYCLE2;
      NIACYCLE1#
    NTICYCLE           2;     % NTI cycle
      NTICYCLE2;
      NTICYCLE1#
    SYLCOUNT           2;     % Syllable count
      SYLCOUNT2;
```

```
      SYLCOUNT1#
    PTAKEN              2;     % Pred. taken on
      PTON2;
      PTON1#
    DUALPT              2;     % Dual Pred. Taken
      DUALPT2;
      DUALPT1#
    DUPCQDLY            2;     % Delay
      DUCQDLY2;
      DUCQDLY1#
    HLTNXT              2;     % Halt Next
      HLTNXT2;
      HLTNXT1#
    ADDTST3             4;     % addtst3 register
      PCINCR1/2;
      PCINCR1/1;
      PCINCR1/0;
      EXTENDM#                 % HAS BEEN SWAP WITH ADDTST2(3) ON 5/2/86
    ADDTST2             4;     % addtst2 register
      ADDTST3/0;               % HAS BEEN SWAP WITH ADDTST3(0) ON 5/2/86
      IXM;
      IAM;
      MCP1#                    %DUMMY NOT USED
    ADDTST1             4;     % Addtst1 LUT
      ADTSTM1/3;
      ADDTST1/2;
      EXCEPTIONM;
      ADDTST1/0#
    OPSYLTST            4;     % OPsyl LUT
      SYLTST/3;
      SYLTST/2;
      INVOP1;
      SYLTST/0#
    CLSTST              8;     % CLSTST register LUT
      BRANCHOP2;               % 11,10
      HLTNXT2;                 % 9
      MCP1;                    % 8  MCP1 S/B LOW !!!!!!!!!!
      PCINCR2/1;               % 7
      PCINCR2/0;               % 6
      DISAFBF2;                % 5
      MCP1;                    % 4  MCP1 S/B LOW !!!!!!!!!!
      CLSTST3.0#               % 3:0
    MCPH                2;     % Omega set lo
```

```
        MCP2;
        MCP1#
      ERQWDATA          2;    % error queue write data A
        ERROR1;
        ODDPC1#
      ERQWDATB          2;    % error queue write data B
        ERROR2;
        ODDPC2#
      FLUSH2            3;    % Flush encode (upper slice)
        FLUSHC2;
        FLUSHB2;
        FLUSHA2#
      FLUSH1            3;    % Flush encode (lower slice)
        FLUSHC1;
        FLUSHB1;
        FLUSHA1#
      FLUSHSEL          2;    % Flush selective (IF flush)
        FLUSHSEL2;
        FLUSHSEL1#
      FLUSHGEN          2;    % Flush general (SYS flush)
        FLUSHGEN2;
        FLUSHGEN1#
% FAULTS
      IFBROKEN          2;    % IF board broken
        IFBROKEN2;
        IFBROKEN1#
      FETCHBRKEN        2;    % Fetch module broken
        FTCHBRKEN2;
        FTCHBRKEN1#
      CHECKER           2;    % Checker check
        CHECKER2;
        CHECKER1#
      ADRERR            2;    % ADRCTR error
        ADRERRB;
        ADRERRA#
      FBMUXERRL         1;    % Fbusmux error
        PRTYERRL#
      ADRCTRERR         2;    % ADRCTR error
        ADDR2ERR;
        ADDR1ERR#
      FBCTRLERR         2;    % FBCTRL error
        FCTLERR2;
        FCTLERR1#
```

```
      FMTBERR            2;    % Error FMTCTB (RBUS error)
        ERROR2;
        ERROR1#
      FMTCSPERR          2;    % Format ctrl store prty err
        FMTCSPERR2;
        FMTCSPERR1#
      FMTBFAULT          2;    % FMTCTB fault
        FMTFAULT2;
        FMTFAULT1#
      ROTERR             2;    % Rotator error
        ROTERR2;
        ROTERR1#
      RQFAULT            2;    % RQCTRL fault
        RQFAULT2;
        RQFAULT1#
      SBUSERR            2;    % SBUS error
        SBUSERR2;
        SBUSERR1#
      TAGERR             2;    % Tag error
        TAGERR2;
        TAGERR1#
      ODDPC              2;    % Odd pc error
        ODDPC2;
        ODDPC1#
      FMTAERROR          1;    % FMT A FAULT
        FMTAFAULT#
      DISAFBF            2;    % Disable AFBF
        DISAFBF2;
        DISAFBF1#
      MREG               2;    % Maintenance register
        MREG2;
        MREG1#
      IFTESTEN           2;    % Diagnostic test enable
        IFTESTEN2;
        IFTESTEN1#
      RQNEWPCH           2;    % RQ Newpc
        RQNEWPCH2;
        RQNEWPCH1#
      PCSBUSEQ           2;    % PC Sbus equal
        PCSBUSEQ2;
        PCSBUSEQ1#
      NUMSYLLOD          2;     % Number syllable tag load
        NUMSYLLD2;
```

```
      NUMSYLLD#
    FBSPARE          6;    % Spare bits
      SPAREC;
      SPAREB;
      SPAREF;
      SPAREA;
      SPAREE;
      SPARED#
    REGF12           2;    % REG f
      REGFA;
      REGF#
    REG2VLD          2;    % REG 2 F
      REG2FA;
      REG2F#
    REGRQF12         2;    % REG RQ
      REGRQFA;
      REGRQF#
    REGRBUS          2;    % RBUS register
      REGRBUSFA;
      REGRBUSF#
    REG1VLD          2;    % REG 1 valid
      REG1FA;
      REG1F#
    REG2/27.25       3;    % bits 27:25 of reg2
      REG2.27;
      REG2.26;
      REG2.25#
    REG2/23.20       4;    % reg2 23:20
      REG2.23;
      REG2.22;
      REG2.21;
      REG2.20#
    REG2/27.24       4;
      REG2.27;
      REG2.26;
      REG2.25;
      REG2.24#
    REG2/31.28       4;
      REG2.31;
      REG2.30;
      REG2.29;
      REG2.28#
    REG2/23.21        3;    % reg2 23:21
```

```
      REG2.23;
      REG2.22;
      REG2.21#
    REG1/39.32        2;    % 8 bits REG1
      REG1/39.36;
      REG1/35.32#
    REG1/31.24        2;    % 8 bits REG1
      REG1/31.28;
      REG1/27.24#
    REG1/23.16        2;
      REG1/23.20;
      REG1/19.16#
    REG1/15.8         2;    % 8 bit REG1
      REG1/15.12;
      REG1/11.8#
    REG1/7.0          2;    % 8 BIT REG1
      REG1/7.4;
      REG1/3.0#
    REG2/19.4         4;
      REG2/19.16;
      REG2/15.12;
      REG2/11.8;
      REG2/7.4#             % least sig 16 bits REG2
    REG2/31.29        3;    % REG2 31:29
      REG2.31;
      REG2.30;
      REG2.29#
    REG2/39.28       12;    % correct order bits
      REG2.39;
      REG2.38;
      REG2.37;
      REG2.36;
      REG2.35;
      REG2.34;
      REG2.33;
      REG2.32;
      REG2.31;
      REG2.30;
      REG2.29;
      REG2.28#
    REG2/28.39       12;    % reverse order bits clstst
      REG2.28;
      REG2.29;
```

```
      REG2.30;
      REG2.31;
      REG2.32;
      REG2.33;
      REG2.34;
      REG2.35;
      REG2.36;
      REG2.37;
      REG2.38;
      REG2.39#
    XDIGPRTY          1;    % Rename X digit parity
      XDIG#
    FEDIGPRTY         1;    % Rename FE digit parity
      FEDIG#
    FBUS1PRTY         1;    % Rename FBUS1(39:0) parity
      FORTY#
    FBUS1REGL         10;    % FBUS1 register
      FREGL55;
      FREGL54;
      FREGL53;
      FREGL52;
      FREGL51.28;
      FREGL27;
      FREGL26;
      FREGL25;
      FREGL24;
      FREGL23.0#
    ADDRPRTY          2;    % ADRCTR parity
      ADDRPRTYB;
      ADDRPRTYA#
    ADRCNT            3;    % Fetch address count
      ADRDIG96;
      ADRCNTB;
      ADRCNTA#
    FREGL52.28        2;    % FREGL 52:28
      FREGL52;
      FREGL51.28#
    WADTST23L         5;    % Write Addtst 2 & 3 literal
      REG2.35;
      REG2.34;
      REG2.33;
      REG2.32;
      REG2/3.0#
```

```
     RADTST23          8;     % Read Addtst 2 & 3
       PCINCR1/2;
       PCINCR1/1;
       PCINCR1/0;
       EXTENDM#               % HAS BEEN SWAP WITH ADDTST2(3) ON 5/2/86
       ADDTST3/0;             % HAS BEEN SWAP WITH ADDTST3(0) ON 5/2/86
       IXM;
       IAM;
       MCP1#                  %DUMMY NOT USED
     WADTST23NL        5;     % Write Addtst 2 & 3 NON literal
       REG2.39;
       REG2.38;
       REG2.37;
       REG2.36;
       REG2/3.0#



     REGEND;
```

10.1.1.2 INSTRUCTION FETCH MAINTENANCE

```
CHAIN YFI;                          % 496 BIT CHAIN LENGTH

%   OPTION    WIDTH    LOCATION  SDI SDO       SDI-NET
%                                              X1SHIFTOUT$P
%   CLKMNT      16       R6F8    P08 L13       SERDAT05
%   STOP        96       P2D4    C15 H14       SERDAT04
%   STOP        96       K4K6    C15 H14       SERDAT03
%   STOP        96       K4N0    C15 H14       SERDAT02
%   STOP        96       I0K6    C15 H14       SERDAT01
%   STOP        96       I0N0    C15 H14       ASHIFTINF01P
% Instruction Fetch Maintenance chain


% Clock Maintenance array
 ERRORIG            1;      % error ignore
 OVERRUN            1;      % over run
 COUNTERFF          1;      % counter f/f
 CLKBAD             1;      % clock bad
 SKEWLOW            1;      % skew low f/f
 SKEWHI             1;      % skew hi f/f
 MODIFYSKEW         1;      % modify skew f/f
 SKEWREG            9;      % skew register
% 16 BITS


% Stop logic array (IF Stop)
 ANDFF              1;      % AND f/f
 ORR                1;      % OR f/f
 PRETRIG            1;      % pre-trigger f/f
 LOADFF             2;      % load f/f
 SETCLRFF           2;      % set clr f/f
 PC                28;      % program counter 28 bits
 PCQWEL             1;      % PC queue write enable L
 SPAREIF           19;      % spare (low)
 OP                 8;      % OP code on SBUS
 SPAREA             8;      % tied low
 PCENB6             1;      % PC digit 6 enable
 PCENB5             1;      % PC digit 5 enable
```

```
      PCENB4            1;      % PC digit 4 enable
      PCENB3            1;      % PC digit 3 enable
      PCENB2            1;      % PC digit 2 enable
      PCENB1            1;      % PC digit 1 enable
      PCENB0            1;      % PC digit 0 enable
      PCWELNABL         1;      % PC WE enable
      SPAREIFA          2;      % Spare bits
      OPENABLE          1;      % C16 word disable
      C8DIGENB          1;      % C8 digit disable
      C8BITENB          4;      % C8 bit disable
      BPSTOPEN          1;      % backplane stop enable
      LOCALEN           1;      % local A enable
      COREN             1;      % C 'OR' enable
      BOREN             1;      % B 'OR' enable
      AOREN             1;      % A 'OR' enable
      C8NOTSLCT         1;      % C8 not selected
      CANDEN            1;      % C 'AND' enable
      BANDEN            1;      % B 'AND' enable
      AANDEN            1;      % A 'AND' enable
    % 96 BITS


    % Stop logic array (Write bus stop)
      WRTANDFF          1;      % AND f/f
      WRTORR            1;      % OR f/f
      WRTPTRIG          1;      % pre-trigger f/f
      WRTLODFF          1;      % load f/f
      DATAHIT           1;      % data hit (OF)
      SPAREWB           1;      % spare
      OPTIMAL           1;      % optimal
      XWRTBUS          40;      % write bus compare data
      FBUS2LOAD         1;      % FBUS2 load
      FBUS2PAGE         2;      % FBUS2 page address
      FBUS2ADR          3;      % FBUS2 address
      FBUS2DATA        10;      % FBUS2 data
      WRTSPARE          1;      % spare
      WRTBUSTAG         7;      % Write bus tag
      WRTBUSENB9        1;      % Write bus digit 9 enable
      WRTBUSENB8        1;      % Write bus digit 8 enable
      WRTBUSENB7        1;      % Write bus digit 7 enable
      WRTBUSENB6        1;      % Write bus digit 6 enable
      WRTBUSENB5        1;      % Write bus digit 5 enable
      WRTBUSENB4        1;      % Write bus digit 4 enable
```

```
 WRTBUSENB3        1;     % Write bus digit 3 enable
 WRTBUSENB2        1;     % Write bus digit 2 enable
 WRTBUSENB1        1;     % Write bus digit 1 enable
 WRTBUSENB0        1;     % Write bus digit 0 enable
 FBUS2ENB          1;     % C16 word disable
 WRTBSTAGDE        1;     % WRTBUSTAG digit enable
 WRTBSTAGBE        4;     % WRTBUSTAG bit enable
 WRTBPSTPEN        1;     % backplane stop enable
 WRTLOCALEN        1;     % local A enable
 WRTCOREN          1;     % C 'OR' enable
 WRTBOREN          1;     % B 'OR' enable
 WRTAOREN          1;     % A 'OR' enable
 WRT8NOTSLT        1;     % C8 not selected
 WRTCANDEN         1;     % C 'AND' enable
 WRTBANDEN         1;     % B 'AND' enable
 WRTAANDEN         1;     % A 'AND' enable
% 96 BITS


% Stop logic array (Fbus1 stop)
 FBANDFF           1;     % AND f/f
 FBORR             1;     % OR f/f
 FBPRETRIG         1;     % pre-trigger f/f
 FBLOADFF          2;     % load f/f
 FBSETCLRFF        2;     % set clr f/f
 FBUS1ADR          3;     % FBUS1 address
 FBSPARE           1;     % FBUS1 top (not used)
 FBUS1DATA        52;     % FBUS1 data
 XWBUSCMD          5;     % write bus command
 FBUS1PAGE         2;     % FBUS1 page addr
 FBUS1LOAD         1;     % FBUS1 load
 FB1ADRENB         1;     % FBUS1 adr enable
 FB1DATEN12        1;     % FBUS1 data 12 enable
 FB1DATEN11        1;     % FBUS1 data 11 enable
 FB1DATEN10        1;     % FBUS1 data 10 enable
 FB1DATEN9         1;     % FBUS1 data 9 enable
 FB1DATEN8         1;     % FBUS1 data 8 enable
 FB1DATEN7         1;     % FBUS1 data 7 enable
 FB1DATEN6         1;     % FBUS1 data 6 enable
 FB1DATEN5         1;     % FBUS1 data 5 enable
 FB1DATEN4         1;     % FBUS1 data 4 enable
 FB1DATEN03        1;     % FBUS1 data 0-3 enable
 XWBUSCMDEN        2;     % WRTBUSCMD enable
```

```
      FB1PAGEEN          2;      % FBUS1 page enable
      FB1LOADEN          1;      % FBUS1 LOAD enable
      FBBPSTOPEN         1;      % backplane stop enable
      FBLOCALEN          1;      % local A enable
      FBCOREN            1;      % C 'OR' enable
      FBBOREN            1;      % B 'OR' enable
      FBAOREN            1;      % A 'OR' enable
      FB8NOTSLCT         1;      % C8 not selected
      FBCANDEN           1;      % C 'AND' enable
      FBBANDEN           1;      % B 'AND' enable
      FBAANDEN           1;      % A 'AND' enable
     % 96 BITS


     % Stop logic array (Address bus stop)
      ADANDFF            1;      % AND f/f
      ADORR              1;      % OR f/f
      ADPRETRIG          1;      % pre-trigger f/f
      ADLOADFF           2;      % load f/f
      ADSETCLRFF         2;      % set clr f/f
      ANDMSKRXMX         1;      % AND mask of REQXMX
      ORMSKRXMX          1;      % OR  mask of REQXMX
      ANDMSKRXMD         1;      % AND mask of REQXMD
      ORMSKRXMD          1;      % OR  mask of REQXMD
      ANDMSKROF          1;      % AND mask of REQOF
      ORMSKROF           1;      % OR  mask of REQOF
      ANDMSKRIF          1;      % AND mask of REQIF
      ORMSKRIF           1;      % OR  mask of REQIF
      ADDRBUS           40;      % address bus data
      ADSPAREZ           8;      % spare
      WRTBUSLEN          4;      % write bus length
      ADSPAREY           1;      % spare
      WBUSRQID           3;      % write bus I.D.
      WRTREQENB          1;      % WRT bus reqst enb
      FLUSHENB           1;      % FLUSH enable
      ADDRBUSE11         1;      % ADR bus digit 11 enable
      ADDRBUSE10         1;      % ADR bus digit 10 enable
      ADDRBUSE9          1;      % ADR bus digit 9 enable
      ADDRBUSE8          1;      % ADR bus digit 8 enable
      ADDRBUSE7          1;      % ADR bus digit 7 enable
      ADDRBUSE6          1;      % ADR bus digit 6 enable
      ADDRBUSE5          1;      % ADR bus digit 5 enable
      ADDRBUSE4          1;      % ADR bus digit 4 enable
```

```
      ADDRBUSE03          1;      % ADR bus digit 3 enable
      WRTLNGTHEN          2;      % WRT bus length enb
      WBSRQIDENB          3;      % WRT bus req ID enb
      ADBPSTOPEN          1;      % backplane stop enable
      ADLOCALEN           1;      % local A enable
      ADCOREN             1;      % C 'OR' enable
      ADBOREN             1;      % B 'OR' enable
      ADAOREN             1;      % A 'OR' enable
      AD8NOTSLCT          1;      % C8 not selected
      ADCANDEN            1;      % C 'AND' enable
      ADBANDEN            1;      % B 'AND' enable
      ADAANDEN            1;      % A 'AND' enable
     % 96 BITS


     % Stop logic array (Rbus stop)
      RBANDFF             1;      % AND f/f
      RBORR               1;      % OR f/f
      RBPRETRIG           1;      % pre-trigger f/f
      RBLOADFF            2;      % load f/f
      RBSETCLRFF          2;      % set clr f/f
      RBUS               40;      % read bus compare data
      RBUSVALID           1;      % RBUS valid
      RBSPARE            16;      % spare
      RBUSTAG             7;      % read bus tag
      RBUSENB9            1;      % RBUS DATA dig 9 enable
      RBUSENB8            1;      % RBUS DATA dig 8 enable
      RBUSENB7            1;      % RBUS DATA dig 7 enable
      RBUSENB6            1;      % RBUS DATA dig 6 enable
      RBUSENB5            1;      % RBUS DATA dig 5 enable
      RBUSENB4            1;      % RBUS DATA dig 4 enable
      RBUSENB3            1;      % RBUS DATA dig 3 enable
      RBUSENB2            1;      % RBUS DATA dig 2 enable
      RBUSENB1            1;      % RBUS DATA dig 1 enable
      RBUSENB0            1;      % RBUS DATA dig 0 enable
      RBSVLDENB           1;      % RBUSVALID enable
      RBUSTAGDEN          1;      % RBUSTAG enable digit
      RBUSTAGBEN          4;      % RBUSTAG enable bit
      RBBPSTOPEN          1;      % backplane stop enable
      RBLOCALEN           1;      % local A enable
      RBCOREN             1;      % C 'OR' enable
      RBBOREN             1;      % B 'OR' enable
      RBAOREN             1;      % A 'OR' enable
```

```
 RBTAGNOT          1;      % C8 not selected
 RBCANDEN          1;      % C 'AND' enable
 RBBANDEN          1;      % B 'AND' enable
 RBAANDEN          1;      % A 'AND' enable
% 96 BITS

 CHNEND;                   % CHAIN END



REGLIST  YFI;

% Stop logic array (IF Stop)

% Stop logic array (Write bus stop)
 FBUS2STOP         3;
  FBUS2PAGE;
  FBUS2ADR;
  FBUS2DATA#

% Stop logic array (Fbus1 stop)
 FBUS1STOP         2;
  FBUS1PAGE;
  FBUS1ADR#

% Stop logic array (Address bus stop)

% Stop logic array (Rbus stop)

REGEND;


% Clock Maintenance array
```

10.1.1.3 OPERAND FETCH DATA

OPERAND FETCH CONTROL CARD

```
% Operand Fetch Control Card Data Chain List - Chain FC
% There are 977 (9/9/85) bits of registers in  OF Control Card Data
%  Chain.
%
% Register Name    # Bits          % Comments
%
CHAIN             FC;              % This is a 977 bit operand fetch
%                                    control chain
%
%                                  FOFFRZ - 48 bits
  ERRFORCED        1;              % selfcheck in progress register
  ONCARDERR        1;              % OFC card error indicator
  FETCHERR         1;              % Fetch error indicator
  HIDDENMOD        1;              % Hidden state mode register
  IGNORERR         1;              % ignore all errors
  DDRVUSE          1;              % board DDRIVE indicator and control
  FRZSPARE5        2;              % spare register; used for data card
  FRZSPARE4        5;              % spare register; used for data card
  LOKOVRFLW        1;              % lock unit overflow error indicator
  LOKCTLER         1;              % LOKCTL array error indicator
  FRZSPARE3        1;              % spare register; used for data card
  SLFCHKMODE       1;              % Self check mode indicator
  FCREGERR         1;              % Force creg parity error indicator
  RQWRTBACK        1;              % RBUS queue write back indicator
  FQWRTBACK        1;              % FBUS1 queue write back indicator
  OFFRZERR         1;              % FFRZCT array error indicator
  FRZSPARE0        1;              % Spare register; used for data card
  OPLK0ERR0        1;              % OPLOOK copy 0 error indicator
  OPLK1ERR0        1;              % OPLOOK copy 1 error indicator
  CREGERR3         1;              % CREG copy 3 array error indicator
  CREGERR2         1;              % CREG copy 2 array error indicator
  CREGERR1         1;              % CREG copy 1 array error indicator
  CREGERR0         1;              % CREG copy 0 array error indicator
  OFTESTERR        1;              % OFTEST array error indicator
  FRZSPARE6        1;              % spare register; used for data card
  OFCMISC          2;              % misc registers; to be defined later
```

```
   FRZSPARE1        2;              % Spare registers; used for data card
   FLIPXMFPAG       1;              % Flip XM fetch page register
   LOCKFULL         1;              % Lock unit full condition
   FRZSPARE2        8;              % Spare registers; used for data card
   RQEMPTFRZ        1;              % live freeze caused by RQ not valid
   AWFULLFRZ        1;              % live freeze caused by AW int full
   STOPBIT0         1;              % Stop bit register copy 0
   STOPBIT1         1;              % Stop bit register copy 1
%
%                                   XMSEQ2 Copy 1 - 12 bits
   STK1LSB          2;              % Stack Top Register (1:0)
   UPC1LSB          2;              % Micro Program Counter (1:0)
   STKOFLW1         1;              % Stack overflow error register
   STKUFLW1         1;              % Stack underflow error register
   STKFULL1         1;              % Stack full register
   STKEMTY1         1;              % Stack empty register
   LSBDIFF1         1;              % CS (1:0) Mismatch error register
   HISTPTR4         1;              % History File pointer
   STKPTR4          2;              % Stack pointer (1:0)
%
%                                   XMSEQ2 Copy 0 - 12 bits
   STK0LSB          2;              % Stack Top Register (1:0)
   UPC0LSB          2;              % Micro Program Counter (1:0)
   STKOFLW0         1;              % Stack overflow error register
   STKUFLW0         1;              % Stack underflow error register
   STKFULL0         1;              % Stack full register
   STKEMTY0         1;              % Stack empty register
   LSBDIFF0         1;              % CS (1:0) Mismatch error register
   HISTPTR3         1;              % History File pointer
   STKPTR3          2;              % Stack pointer (1:0)
%
%                                   XMSEQ1 Copy 2 - 28 bits
   STK2MSB         12;              % Stack Top Register (13:2)
   UPC2MSB         12;              % Micro Program Counter (13:2)
   MSBDIFF2         1;              % CS (13:2) Mismatch error register
   HISTPTR2         1;              % History File pointer
   STKPTR2          2;              % Stack pointer (1:0)
%
%                                   XMSEQ1 Copy 1 - 28 bits
   STK1MSB         12;              % Stack Top Register (13:2)
   UPC1MSB         12;              % Micro Program Counter (13:2)
   MSBDIFF1         1;              % CS (13:2) Mismatch error register
   HISTPTR1         1;              % History File pointer
```

```
  STKPTR1          2;              % Stack pointer (1:0)
%
%                                  XMSEQ1 Copy 0 - 28 bits
  STK0MSB         12;              % Stack Top Register (13:2)
  UPC0MSB         12;              % Micro Program Counter (13:2)
  MSBDIFF0         1;              % CS (13:2) Mismatch error register
  HISTPTR0         1;              % History File pointer
  STKPTR0          2;              % Stack pointer (1:0)
%
%                                  LOKCTL - 53 bits
  IXCAHEN          3;              % IX(2:0) cache invalidate enable set
  IXVALID          3;              % IX(2:0) valid
  IXFIT            3;              % IX(2:0) fit(invalidated)
  LOKPRTIN         1;              % Lock control parity in
  LOKCTLERR        1;              % Lock control error
  DATAVALID        8;              % Data in the lock entry is valid
  WRITEPTR         8;              % Lock entry write pointer
  FIRSTWRT         1;              % First write after start or flush
  FIRSTPOP         1;              % First pop after start or flush
  POPCTR7          2;              % Pop counter for entry 7
  POPCTR6          2;              % Pop counter for entry 6
  POPCTR5          2;              % Pop counter for entry 5
  POPCTR4          2;              % Pop counter for entry 4
  POPCTR3          2;              % Pop counter for entry 3
  POPCTR2          2;              % Pop counter for entry 2
  POPCTR1          2;              % Pop counter for entry 1
  POPCTR0          2;              % Pop counter for entry 0
  POPPTR           8;              % Lock entry pop pointer
%
%                                  LOKCMP slice 7 - 55 bits
  LOCKERR7         1;              % Lock comparator error for entry 7
  ENDPAR7L         1;              % End address parity for entry 7
  ENDADD7L        25;              % End address for lock entry 7
  BEGPAR7L         1;              % Begin address parity for entry 7
  BEGADD7L        24;              % Begin address for lock entry 7
  INIXHIT7         1;              % Index indirect hit bit for entry 7
  DATAHIT7         1;              % Data hit bit for entry 7
  CODEHIT7         1;              % Code hit bit for entry 7
%
%                                  LOKCMP slice 6 - 55 bits
  LOCKERR6         1;              % Lock comparator error for entry 6
  ENDPAR6L         1;              % End address parity for entry 6
  ENDADD6L        25;              % End address for lock entry 6
```

```
   BEGPAR6L        1;              % Begin address parity for entry 6
   BEGADD6L       24;              % Begin address for lock entry 6
   INIXHIT6        1;              % Index indirect hit bit for entry 6
   DATAHIT6        1;              % Data hit bit for entry 6
   CODEHIT6        1;              % Code hit bit for entry 6
%
%                                  LOKCMP slice 5 - 55 bits
   LOCKERR5        1;              % Lock comparator error for entry 5
   ENDPAR5L        1;              % End address parity for entry 5
   ENDADD5L       25;              % End address for lock entry 5
   BEGPAR5L        1;              % Begin address parity for entry 5
   BEGADD5L       24;              % Begin address for lock entry 5
   INIXHIT5        1;              % Index indirect hit bit for entry 5
   DATAHIT5        1;              % Data hit bit for entry 5
   CODEHIT5        1;              % Code hit bit for entry 5
%
%                                  LOKCMP slice 4 - 55 bits
   LOCKERR4        1;              % Lock comparator error for entry 4
   ENDPAR4L        1;              % End address parity for entry 4
   ENDADD4L       25;              % End address for lock entry 4
   BEGPAR4L        1;              % Begin address parity for entry 4
   BEGADD4L       24;              % Begin address for lock entry 4
   INIXHIT4        1;              % Index indirect hit bit for entry 4
   DATAHIT4        1;              % Data hit bit for entry 4
   CODEHIT4        1;              % Code hit bit for entry 4
%
%                                  LOKCMP slice 3 - 55 bits
   LOCKERR3        1;              % Lock comparator error for entry 3
   ENDPAR3L        1;              % End address parity for entry 3
   ENDADD3L       25;              % End address for lock entry 3
   BEGPAR3L        1;              % Begin address parity for entry 3
   BEGADD3L       24;              % Begin address for lock entry 3
   INIXHIT3        1;              % Index indirect hit bit for entry 3
   DATAHIT3        1;              % Data hit bit for entry 3
   CODEHIT3        1;              % Code hit bit for entry 3
%
%                                  LOKCMP slice 2 - 55 bits
   LOCKERR2        1;              % Lock comparator error for entry 2
   ENDPAR2L        1;              % End address parity for entry 2
   ENDADD2L       25;              % End address for lock entry 2
   BEGPAR2L        1;              % Begin address parity for entry 2
   BEGADD2L       24;              % Begin address for lock entry 2
   INIXHIT2        1;              % Index indirect hit bit for entry 2
```

```
   DATAHIT2          1;              % Data hit bit for entry 2
   CODEHIT2          1;              % Code hit bit for entry 2
%
%                                    LOKCMP slice 1 - 55 bits
   LOCKERR1          1;              % Lock comparator error for entry 1
   ENDPAR1L          1;              % End address parity for entry 1
   ENDADD1L         25;              % End address for lock entry 1
   BEGPAR1L          1;              % Begin address parity for entry 1
   BEGADD1L         24;              % Begin address for lock entry 1
   INIXHIT1          1;              % Index indirect hit bit for entry 1
   DATAHIT1          1;              % Data hit bit for entry 1
   CODEHIT1          1;              % Code hit bit for entry 1

%
%                                    LOKCMP slice 0 - 55 bits
   LOCKERR0          1;              % Lock comparator error for entry 0
   ENDPAR0L          1;              % End address parity for entry 0
   ENDADD0L         25;              % End address for lock entry 0
   BEGPAR0L          1;              % Begin address parity for entry 0
   BEGADD0L         24;              % Begin address for lock entry 0
   INIXHIT0          1;              % Index indirect hit bit for entry 0
   DATAHIT0          1;              % Data hit bit for entry 0
   CODEHIT0          1;              % Code hit bit for entry 0
%
%                                    OPLOOK slice 1 - 68 bits
   OPLUK1ERR1        1;              % OPLOOK 1 error register 1
   OPLUK1ERR0        1;              % OPLOOK 1 error register 0
   PGSEL1            1;              % OPLOOK 1 page select pointer
   PG1FUL1           1;              % OPLOOK 1 page 1 full indicator
   PG0FUL1           1;              % OPLOOK 1 page 0 full indicator
   DLYFLUSH1         1;              % Register to delay flush by 1 clock
   OPRAMWEN1         1;              % OPLOOK up ram write enable 1
   OPLKDAT1         16;              % OPLOOK up ram output 1
   AXPARITY1         1;              % AX digit parity bit from OPLOOK 1
   AXDIGIT1          4;              % AX digit from OPLOOK 1
   PG1AEXCEP1        1;              % Pg 1 A-exception bit from OPLOOK 1
   PG0AEXCEP1        1;              % Pg 0 A-exception bit from OPLOOK 1
   BXPARITY1         1;              % BX digit parity bit from OPLOOK 1
   BXDIGIT1          4;              % BX digit from OPLOOK 1
   PG1BEXCEP1        1;              % Pg 1 B-exception bit from OPLOOK 1
   PG0BEXCEP1        1;              % Pg 0 B-exception bit from OPLOOK 1
   CXPARITY1         1;              % CX digit parity bit from OPLOOK 1
   CXDIGIT1          4;              % CX digit from OPLOOK 1
```

```
   PG1CEXCEP1      1;              % Pg 1 C-exception bit from OPLOOK 1
   PG0CEXCEP1      1;              % Pg 0 C-exception bit from OPLOOK 1
   PG1OPPRTY1      1;              % Pg 1 opvector parity from OPLOOK 1
   PG1OPVECT1     11;              % Pg 1 opvector from OPLOOK 1
   PG0OPPRTY1      1;              % Pg 0 opvector parity from OPLOOK 1
   PG0OPVECT1     11;              % Pg 0 opvector from OPLOOK 1
%
%                                  OPLOOK slice 0 - 68 bits
   OPLUK0ERR1      1;              % OPLOOK 0 error register 1
   OPLUK0ERR0      1;              % OPLOOK 0 error register 0
   PGSEL0          1;              % OPLOOK 0 page select pointer
   PG1FUL0         1;              % OPLOOK 0 page 1 full indicator
   PG0FUL0         1;              % OPLOOK 0 page 0 full indicator
   DLYFLUSH0       1;              % Register to delay flush by 1 clock
   OPRAMWEN0       1;              % OPLOOK up ram write enable 0
   OPLKDAT0       16;              % OPLOOK up ram output 0
   AXPARITY0       1;              % AX digit parity bit from OPLOOK 0
   AXDIGIT0        4;              % AX digit from OPLOOK 0
   PG1AEXCEP0      1;              % Pg 1 A-exception bit from OPLOOK 0
   PG0AEXCEP0      1;              % Pg 0 A-exception bit from OPLOOK 0
   BXPARITY0       1;              % BX digit parity bit from OPLOOK 0
   BXDIGIT0        4;              % BX digit from OPLOOK 0
   PG1BEXCEP0      1;              % Pg 1 B-exception bit from OPLOOK 0
   PG0BEXCEP0      1;              % Pg 0 B-exception bit from OPLOOK 0
   CXPARITY0       1;              % CX digit parity bit from OPLOOK 0
   CXDIGIT0        4;              % CX digit from OPLOOK 0
   PG1CEXCEP0      1;              % Pg 1 C-exception bit from OPLOOK 0
   PG0CEXCEP0      1;              % Pg 0 C-exception bit from OPLOOK 0
   PG1OPPRTY0      1;              % Pg 1 opvector parity from OPLOOK 0
   PG1OPVECT0     11;              % Pg 1 opvector from OPLOOK 0
   PG0OPPRTY0      1;              % Pg 0 opvector parity from OPLOOK 0
   PG0OPVECT0     11;              % Pg 0 opvector from OPLOOK 0
%
%                                  CREG copy 1 of slice 2 CREG - 6 bits
   CREG3ERR        1;              % CREG 3 error register
   CREG3MREG       5;              % CREG 3 spare ram column register
%
%                                  CREG copy 0 of slice 2 CREG - 6 bits
   CREG2ERR        1;              % CREG 2 error register
   CREG2MREG       5;              % CREG 2 spare ram column register
%
%                                  CREG slice 1 CREG - 6 bits
   CREG1ERR        1;              % CREG 1 error register
```

```
  CREG1MREG        5;              % CREG 1 spare ram column register
%
%                                  CREG slice 0 CREG - 6 bits
  CREG0ERR         1;              % CREG 0 error register
  CREG0MREG        5;              % CREG 0 spare ram column register
%
%                                  OFTEST - OF test condition 46 bits
  ADDRSWAP         1;              % Address swap signal for sequencer
  UNDIGITBBS       1;              % B BUS data undigit flip flop
  EXTSYLH          2;              % Number of extended syllables
  NEWOPSYLH        1;              % New OPSYL written indicator
  INTERUPT1        1;              % Interrupt register bit 1
  ODDPC            1;              % ODD PC error register
  ADDRERR          1;              % Address error register
  TIMEOUT          1;              % Time out error register
  OPTIMALH         1;              % Optimal condition register
  AFGTBF           1;              % AF > BF condition register
  AFEQBF           1;              % AF = BF condition register
  AGTB             1;              % Abus > Bbus condition register
  AEQB             1;              % Abus = Bbus condition register
  CC               2;              % C address controller
  BC               2;              % B address controller
  AC               2;              % A address controller
  CMDPARERR        1;              % Internal command error register
  AUEXPERR         1;              % AU excep. bit comparison error reg.
  AUEQ0            1;              % AU = 0 condition register
  OFTSTSPAR1       1;              % Spare bit; not used
  OFTSTSPAR0       1;              % Spare bit; not used
  OFEVENT          1;              % OF to XM event wire
  DLYFLUSHL        1;              % Register to delay flush by 1 clock
  IX4CACHE         1;              % IX4 cache status bit
  IX5CACHE         1;              % IX5 cache status bit
  IX6CACHE         1;              % IX6 cache status bit
  IX7CACHE         1;              % IX7 cache status bit
  CONDFF2          1;              % Conditional f/f 2 - LIX'ED flag
  CONDFF3          1;              % Conditional f/f 3
  CONDFF1          1;              % Conditional f/f 1
  LIXOK            1;              % LIXOK register
  SYSFLUSH         1;              % System flush register
  FETFLUSH         1;              % Fetch flush register
  INIXHIT          1;              % Index/indirect hit register
  DATAHIT          1;              % Data hit register
  CODEHIT          1;              % Code hit register
```

```
   XMEVENT          1;              % XM to OF event register
   CODECHKL         1;              % Code check command register
   DATACHKL         1;              % Data check command register
   INIXCHKL         1;              % Index/indirect check command reg.
   WRTLOCKL         1;              % Write lock command register
   POPLOCKL         1;              % OPCOMP/POPLOCK command register
%
%                                   CREG copy 1 of slice 2 CREG - 27 bits
   SPARERAM1        1;              % spare ram control register; copy 1
   WORDPRTY1        1;              % spare; not used
   PTEST1           4;              % copy 1 of PTEST command bits
   BADR1            6;              % copy 1 of B_address register
   AADR1            5;              % copy 1 of A_address register
   CADR1            5;              % copy 1 of C_address register
   ASRC1            2;              % copy 1 of A_source register
   BSRC1            2;              % copy 1 of B_source register
   CWE1             1;              % copy 1 of C_source register
%
%                                   CREG copy 0 of slice 2 CREG - 27 bits
   SPARERAM0        1;              % spare ram control register; copy 0
   WORDPRTY0        1;              % spare; not used
   PTEST0           4;              % copy 0 of PTEST command bits
   BADR0            6;              % copy 0 of B_address register
   AADR0            5;              % copy 0 of A_address register
   CADR0            5;              % copy 0 of C_address register
   ASRC0            2;              % copy 0 of A_source register
   BSRC0            2;              % copy 0 of B_source register
   CWE0             1;              % copy 0 of C_source register
%
%                                   CREG slice 1 CREG - 27 bits
   CSRC             5;              % C_source control register
   INTCMD           5;              % INTERNAL command registr
   ALUCMD           4;              % ALU command register
   BRANCH           4;              % BRANCH control register
   PUSHSTK          1;              % PUSH STACK control register
   TSTCND           5;              % TEST CONDITION control register
   FBUS1WE          1;              % FBUS1 write enable register
   FBUS2WE          1;              % FBUS2 write enable register
   FB2SRC           1;              % FBUS2 source control register
%
%                                   CREG slice 0 CREG - 27 bits
   EXTCMD           5;              % EXTERNAL command registers
   SPARE1           3;              % spare registers; not used
```

```
  BEGADSYNC       1;              % begin address sync register
  SLFCHK          4;              % self check command registers
  JMPADR         14;              % jump address registers
%
%                                 CS write addr reg in CREG - 14 bits
  CSWADR         14;              % CS write address register
%
CHNEND;
```

10.1.1.3 OPERAND FETCH DATA (Continued)

OPERAND FETCH DATA CARD

```
% Operand Fetch Data Card Data Chain List - Chain FD
%
%There  are 531 (522-7/24/85)  bits of  registers in  OF Data  Card
%Data Chain.
%
% Register Name    # Bits          % Comments
%
CHAIN             FD;             % This is a 531 bit operand fetch
%                                   data chain
%
%                                 FOFFRZ - 48 bits
  ERRFORCED        1;             % selfcheck in progress register
  ONCARDERR        1;             % OFC card error indicator
  FETCHERR         1;             % Fetch error indicator
  HIDDENMOD        1;             % Hidden state mode register
  IGNORERR         1;             % ignore all errors
  DDRVUSE          1;             % board DDRIVE indicator and control
  LITFILEN         1;             % literal file ram write enable
  SFTPTSTEN        1;             % soft PTEST ram write enable
  FLUTPRTY         1;             % FLUT parity bit; from oplout rams
  LITMASK          1;             % LITERAL mask; from oplout rams
  FLUTSPARE        1;             % FLUT spare bit; from oplout rams
  CFCODE0          1;             % CFCODE(0:0); output of oplength ram
  CFCODE1          1;             % CFCODE(1:1); output of oplength ram
  FRZSPARE5        1;             % spare register;used on control card
  BOPTIMAL         1;             % BOPTIMAL; output of oplength ram
  AUREGERR1        1;             % AUREG copy 1 error indicator
  SLFCHKMODE       1;             % Self check mode indicator
  FCREGERR         1;             % Force creg parity error indicator
  RQWRTBACK        1;             % RBUS queue write back indicator
  FQWRTBACK        1;             % FBUS1 queue write back indicator
  OFFRZERR         1;             % FFRZCT array error indicator
  BBUSERR          1;             % BBUS data error indicator
  CLRAMPRTY        1;             % C_length ram parity
  BBUSPRTY         1;             % BBUS parity register
```

```
  BLRAMPRTY       1;              % B_length ram parity
  AOPTIMAL        1;              % AOPTIMAL; from oplout rams
  ALRAMPRTY       1;              % A_length ram parity
  OPTCODE0        1;              % OPTIMAL code; from FLUT(0:0)
  OPTCODE1        1;              % OPTIMAL code; from FLUT(1:1)
  OFFCTRLERR      1;              % OFFCTRL error indicator
  OFDMISC         2;              % misc registers; to be defined later
  FQRQERR2        1;              % FANDRQ error indicator
  FQRQERR1        1;              % FANDRQ error indicator
  FQRQERR0        1;              % FANDRQ error indicator
  MQFULLH         1;              % memory queue full indicator
  PMUXERR         1;              % PMUX error indicator
  AUREGERR0       1;              % AUREG copy 0 error indicator
  BLENRAMERR      1;              % Parity error on BLEN rams
  ALENRAMERR      1;              % Parity error on ALEN rams
  OPLENERR1       1;              % OPLENIN copy 1 error on FE digit
  OPLENERR0       1;              % OPLENIN copy 0 error on FE digit
  AWCTLERR        1;              % AWRCTL array error indicator
  OPLOUTERR       1;              % OPLOUT error indicator
  RQEMPTFRZ       1;              % live freeze caused by RQ not valid
  AWFULLFRZ       1;              % live freeze caused by AW int full
  STOPBIT0        1;              % Stop bit register copy 0
  STOPBIT1        1;              % Stop bit register copy 1
%
%                                 FPTEST 1 -17 bits
  AUEXCPT1        1;              % copy 1 of ALU exception
  IFLMSDO1        1;              % IF limit error and SDO
  IFUNDG1         1;              % IF undigit error
  OFLMT1          1;              % OF limit error
  OFUNDG1         1;              % OF undigit error
  UNDGAB1         1;              % ABUS OR BBUS undigit error
  SOFTERR1        1;              % SOFT PTEST parity error
  APARERR1        1;              % ABUS parity error
  READ1           1;              % hidden read soft ptest state
  SSERR1          1;              % fetch page pc sylable SS par err
  BEGPAR0         1;              % lock begin address parity; copy 0
  SOFTPRD1        1;              % spare for read soft ptest only
  PTREG1          5;              % PTEST REGISTER
%
%                                 FPTEST 0 - 17 bits
  AUEXCPT0        1;              % copy 0 of ALU exception
  IFLMSDO0        1;              % IF limit error and SDO
  IFUNDG0         1;              % IF undigit error
```

```
  OFLMT0          1;              % OF limit error
  OFUNDG0         1;              % OF undigit error
  UNDGAB0         1;              % ABUS or BBUS undigit error
  SOFTERR0        1;              % SOFT PTEST parity error
  APARERR0        1;              % ABUS parity error
  READ0           1;              % hidden read soft ptest state
  SSERR0          1;              % fetch page pc sylable SS par err
  BEGPAR1         1;              % lock begin address parity; copy 1
  SOFTPRD0        1;              % spare for read soft ptest only
  PTREG0          5;              % PTEST REGISTER
%
%                                 OFFCTL - 20 bits
  RBUSMSBERR      1;              % RBUS error tag parity error
  RBUSDATERR      1;              % RBUS data parity error
  FBUSDATERR      1;              % FBUS1 data parity error
  FADDR1          3;              % FBUS1 entry address
  FADDR2          3;              % FBUS2 entry address
  FPADDR1         2;              % FBUS1 page address
  FLOAD1          1;              % FBUS1 load
  FCPARITY1       1;              % FBUS1 control parity
  FCPARITY2       1;              % FBUS2 control parity
  FLOAD2          1;              % FBUS2 load
  FPARITY1        1;              % FBUS1 data parity
  OPTIMAL         1;              % Optimal
  DATAHIT         1;              % Data hit
  SHOVE           1;              % Shove
  FBUS2PRTY       1;              % backplane FBUS2 parity bit
%
%                                 AWRCTL - 36 bits
  RBUSTAGERR      1;              % RBUS tag parity error
  ADDBUSPRTY      1;              % Address bus parity
  AWPRTYIN0       1;              % Control parity for internal use
  AWPRTYIN1       1;              % Control parity for internal use
  AWIFFLUSH       1;              % IF flush
  AWCMNDB0        1;              % Bit 0 of AW_command bits
  AWCMNDB1        1;              % Bit 1 of AW_command bits
  AWCMNDB2        1;              % Bit 2 of AW_command bits
  AWCMNDB3        1;              % Bit 3 of AW_command bits
  AWCMNDB4        1;              % Bit 4 of AW_command bits
  AWOFREQ         1;              % AW bus request for OF
  AWTAGBIT3       1;              % AWTAG(4:4) and AWTAG_(3:3)
  AWTAGBIT2       1;              % AWTAG(2:2)
  AWTAGBIT1       1;              % AWTAG(1:1)
```

```
  AWTAGBIT0        1;              % AWTAG(0:0)
  FQPAGPTR         2;              % Fetch queue page pointer for XM
  RQNTRYPTR        2;              % RBUS queue entry pointer
  RQFULLH          1;              % RBUS queue full(first time only)
  RQNTRYVLD        4;              % RBUS queue entry valid
  RQRDPTR          2;              % RBUS queue read address pointer
  RQWRTPTR         2;              % RBUS queue write address pointer
  DATFOROF         1;              % RBUS data back is for OF
  ADDOUTEN         1;              % Address bus output enable
  DATOUTEN         1;              % Data bus output enable
  AWLENB0          1;              % AW length field bit 0
  AWLENB1          1;              % AW length field bit 1
  AWLENB2          1;              % AW length field bit 2
  AWLENB3          1;              % AW length field bit 3
  MEMQFULLH        1;              % MCACM queue full
%
%                                  LENOUT - 61 bits
  OPRAMWE1         1;              % Oplength RAM write enable; bit 1
  OPRAMWE0         1;              % Oplength RAM write enable; bit 0
  FBUS2PAR         1;              % FBUS2 parity bit
  FBUS2REGB9       1;              % FBUS2 Data register bit 9
  FBUS2REGB8       1;              % FBUS2 Data register bit 8
  FBUS2REGB7       1;              % FBUS2 Data register bit 7
  FBUS2REGB6       1;              % FBUS2 Data register bit 6
  FBUS2REGB5       1;              % FBUS2 Data register bit 5
  FBUS2REGB4       1;              % FBUS2 Data register bit 4
  FBUS2REGB3       1;              % FBUS2 Data register bit 3
  FBUS2REGB2       1;              % FBUS2 Data register bit 2
  FBUS2REGB1       1;              % FBUS2 Data register bit 1
  FBUS2REGB0       1;              % FBUS2 Data register bit 0
  OPLTPRTY         1;              % OPLIT pairity from FLUT
  BLENCODE         1;              % B length function select register
  ALENCODE         1;              % A length function select register
  CLENPRTY         1;              % C length parity
  CLEN            11;              % C length data
  BLENPRTY         1;              % B length parity
  BLEN            10;              % B length data & parity register
  ALENPRTY         1;              % A length parity
  ALEN            10;              % A length data & parity register
  OPGENPRTY        1;              % Internal OPLIT generated parity
  FETEVENT         1;              % Fetch event register
  SPARE            1;              % spare register; not used
  OPLITMSB         3;              % From OPLIT(6:4) of FLUT
```

```
  OPLITLSD        4;          % From OPLIT(3:0) of FLUT
  ALIT            1;          % ALIT register from FLUT
%
%                             LENIN - 47 bits (page 1)
  AFGTRBF1        1;          % copy 1 of current AF > BF
  AFEQLBF1        1;          % copy 1 of current AF = BF
  CF1             9;          % copy 1 of current CF
  IFEVENT1        1;          % page 1 IF event flag register
  PAGE1PAR        1;          % parity across page 1's ac,bc,cc,
                              %   op,af,bf
  PAGE1AC         2;          % page 1 Ac
  PAGE1BC         2;          % page 1 Bc
  PAGE1CC1        1;          % page 1 Cc, bit 1
  PAGE1CC0        1;          % page 1 Cc, bit 0
  EXT1PAR         1;          % page 1 Aext, Bext, Cext parity
                              %   register
  PAGE1AEXT       1;          % page 1 A extension flag register
  PAGE1BEXT       1;          % page 1 B extension flag register
  PAGE1CEXT       1;          % page 1 C extension flag register
  PAGE1OP         8;          % page 1 opcode register
  PAGE1AF         8;          % page 1 AF register
  PAGE1BF         8;          % page 1 BF register
%
%                             LENIN - 47 bits (page 0)
  AFGTRBF0        1;          % copy 0 of current AF > BF
  AFEQLBF0        1;          % copy 0 of current AF = BF
  CF0             9;          % copy 0 of current CF
  IFEVENT0        1;          % page 0 IF event flag register
  PAGE0PAR        1;          % parity across page 0's ac,bc,cc,
                              %   op,af,bf
  PAGE0AC         2;          % page 0 Ac
  PAGE0BC         2;          % page 0 Bc
  PAGE0CC1        1;          % page 0 Cc; bit 1
  PAGE0CC0        1;          % page 0 Cc; bit 0
  EXT0PAR         1;          % page 0 Aext, Bext, Cext parity
                              %   register
  PAGE0AEXT       1;          % page 0 A extension flag register
  PAGE0BEXT       1;          % page 0 B extension flag register
  PAGE0CEXT       1;          % page 0 C extension flag register
  PAGE0OP         8;          % page 0 opcode register
  PAGE0AF         8;          % page 0 AF register
  PAGE0BF         8;          % page 0 BF register
%
```

```
%                                   FAUREG - 41 bits (copy 1)
  CSUMSEL1H        1;               % select bit for carry out bit
  AUREG1L         28;               % copy 1 of AUREGL(27:0)
  CO23C3           1;               % copy 3 of carry out bit of
                                    %   AUREGL(23)
  CO23C2           1;               % copy 2 of carry out bit of
                                    %   AUREGL(23)
  BBUSD9PAR1       1;               % copy 1 of BBUS(39:36) parity
                                    %   register
  BBUS1D9          4;               % copy 1 of BBUS(39:36) register
  CMD1PAR          1;               % copy 1 of ALUCMD parity register
  ALUCMD1          4;               % copy 1 of ALUCMD(3:0) register
%
%                                   FAUREG - 41 bits (copy 0)
  CSUMSEL0H        1;               % select bit for carry out bit
  AUREG0L         28;               % copy 0 of AUREGL(27:0)

  CO23C1           1;               % copy 1 of carry out bit of
                                    %   AUREGL(23)
  CO23C0           1;               % copy 0 of carry out bit of
                                    %   AUREGL(23)
  BBUSD9PAR0       1;               % copy 0 of BBUS(39:36) parity
                                    %   register
  BBUS0D9          4;               % copy 0 of BBUS(39:36) register
  CMD0PAR          1;               % copy 0 of ALUCMD parity register
  ALUCMD0          4;               % copy 0 of ALUCMD(3:0) register
%
%                                   FANDRQ slice 6 - 16 bits
  WFOPPRTY         1;               % write back FBUS1 OP_parity
  WFPRTYAFBF       1;               % write back FBUS1 OP,AF,BF parity
  WFXDIGPRTY       1;               % write back FBUS1 X_digit parity
  WFXDIGMSB        1;               % write back FBUS1 X_digit MSB
  WFXDIGLSD        4;               % write back FBUS1 X_digit LSD
  RFQORRQS4        1;               % read back FBUS1 or RBUS
  RFQORRQS3        1;               % read back FBUS1 or RBUS
  RFQORRQS2        1;               % read back FBUS1 or RBUS
  RFQORRQS1        5;               % read back FBUS1 or RBUS

%
%                                     FANDRQ slice 5 - 16 bits
  WFEDIGMSD        4;               % write back FBUS1 FE_digit MSD
  WFEDIGLSD        4;               % write back FBUS1 FE_digit LSD
  RFQORRQS0        8;               % read back FBUS1 or RBUS
```

```
%
%                                   FANDRQ slice 4 - 16 bits
  BBREGB39          1;              % BBUS register (39:39)
  BBREGB38          1;              % BBUS register (38:38)
  BBREGB37          1;              % BBUS register (37:37)
  BBREGB36          1;              % BBUS register (36:36)
  BBREGB35          1;              % BBUS register (35:35)
  BBREGB34          1;              % BBUS register (34:34)
  BBREGB33          1;              % BBUS register (33:33)
  BBREGB32          1;              % BBUS register (32:32)
  ABREGB4           8;              % ABUS register (39:32)
%
%                                   FANDRQ slice 3 - 16 bits
  BBREGB31          1;              % BBUS register (31:31)
  BBREGB30          1;              % BBUS register (30:30)
  BBREGB29          1;              % BBUS register (29:29)
  BBREGB28          1;              % BBUS register (28:28)
  BBREGB27          1;              % BBUS register (27:27)
  BBREGB26          1;              % BBUS register (26:26)
  BBREGB25          1;              % BBUS register (25:25)
  BBREGB24          1;              % BBUS register (24:24)
  ABREGB3           8;              % ABUS register (31:24)
%
%                                   FANDRQ slice 2 - 16 bits
  BBREGB23          1;              % BBUS register (23:23)
  BBREGB22          1;              % BBUS register (22:22)
  BBREGB21          1;              % BBUS register (21:21)
  BBREGB20          1;              % BBUS register (20:20)
  BBREGB19          1;              % BBUS register (19:19)
  BBREGB18          1;              % BBUS register (18:18)
  BBREGB17          1;              % BBUS register (17:17)
  BBREGB16          1;              % BBUS register (16:16)
  ABREGB2           8;              % ABUS register (23:16)
%
%                                   FANDRQ slice 1 - 16 bits
  BBREGB15          1;              % BBUS register (15:15)
  BBREGB14          1;              % BBUS register (14;14)
  BBREGB13          1;              % BBUS register (13:13)
  BBREGB12          1;              % BBUS register (12:12)
  BBREGB11          1;              % BBUS register (11:11)
  BBREGB10          1;              % BBUS register (10:10)
  BBREGB9           1;              % BBUS register (9:9)
  BBREGB8           1;              % BBUS register (8:8)
```

```
   ABREGB1          8;              % ABUS register (15:8)
%
%                                   FANDRQ slice 0 - 16 bits
   BBREGB7          1;              % BBUS register (7:7)
   BBREGB6          1;              % BBUS register (6:6)
   BBREGB5          1;              % BBUS register (5:5)
   BBREGB4          1;              % BBUS register (4:4)
   BBREGB3          1;              % BBUS register (3:3)
   BBREGB2          1;              % BBUS register (2:2)
   BBREGB1          1;              % BBUS register (1:1)
   BBREGB0          1;              % BBUS register (0:0)
   ABREGB0          8;              % ABUS register (7:0)
%
%                                   SCHPAD slice 10 - 4 bits
   SCHSPARE         1;              % Spares
   ALLRAMWE         1;              % OP_length ram write enable
   PBSELH           1;              %PB_MUX select for hidden state write
   CBREGB40         1;              % CBUS Parity Register
%
%                                   SCHPAD slice 9 - 4 bits
   CBREGD9          4;              % CBUS Register (39:36)
%
%                                   SCHPAD slice 8 - 4 bits
   CBREGD8          4;              % CBUS Register (35:32)
%
%                                   SCHPAD slice 7 - 4 bits
   CBREGD7          4;              % CBUS Register (31:28)
%
%                                   SCHPAD slice 6 - 4 bits
   CBREGD6          4;              % CBUS Register (27:24)
%
%                                   SCHPAD slice 5 - 4 bits
   CBREGD5          4;              % CBUS Register (23:20)
%
%                                   SCHPAD slice 4 - 4 bits
   CBREGD4          4;              % CBUS Register (19:16)
%
%                                   SCHPAD slice 3 - 4 bits
   CBREGD3          4;              % CBUS Register (15:12)
%
%                                   SCHPAD slice 2 - 4 bits
   CBREGD2          4;              % CBUS Register (11:8)
%
```

```
%                                   SCHPAD slice 1 - 4 bits
  CBREGD1          4;               % CBUS Register (7:4)
%
%                                   SCHPAD slice 0 - 4 bits
  CBREGD0          4;               % CBUS Register (3:0)
%
CHNEND;
%
```

10.1.1.4 OPERAND FETCH MAINTENANCE

OPERAND FETCH CONTROL CARD

```
%There are 112 bits of registers in OF Control Card Maintenance Chain.
%
% Register Name    # Bits          % Comments
%
DATE          ]050886;
CHAIN             YFC;             % OF control card maint chain, 112 bit
%
%                                  CLKMNT - 16 bits
  ERRORIG           1;             % error ignore
  OVERRUN           1;             % over run
  COUNTERFF         1;             % counter f/f
  CLKBAD            1;             % clock bad
  SKEWLOW           1;             % skew low f/f
  SKEWHI            1;             % skew hi f/f
  MODIFYSKEW        1;             % modify skew f/f
  SKEWREG           9;             % skew register
%
%                                  STOP - 96 bits
  ANDFF             1;             % AND f/f
  ORFF              1;             % OR f/f
  PRETRIG           1;             % pre-trigger f/f
  LOADFF            2;             % load f/f
  SETCLRFF          2;             % set clr f/f
                                   % 10 1 bit-compare using C40DATA
  ZADDRERR          1;             % address error condition
  ZZERO38           3;             % set to zero for addrerr comparison
  ZIXCACHE0         1;             % IX 1 cache invalidate condition
  ZZERO34           3;             % set to zero for IX1 cache inv. comp
  ZXMEVENT          1;             % XM to OF event condition
  ZZERO30           3;             % set to zero for XMEVENT comparison
  ZDATAHIT          1;             % data hit stop condition
  ZZERO26           3;             % set to zero for data hit comparison
  ZLOCKFULL         1;             % Lock full condition
  ZZERO22           3;             % set to zero for lock full comparison
  ZIXCACHE1         1;             % IX 2 cache invalidate condition
  ZZERO18           3;             % set to zero for IX1 cache inv. comp
```

```
  ZIXCACHE2        1;              % IX 3 cache invalidate condition
  ZZERO14          3;              % set to zero for IX3 cache inv. comp
  ZIXVALID0        1;              % IX 1 cache valid condition
  ZZERO10          3;              % set to zero for IX1 valid comparison
  ZIXVALID1        1;              % IX 2 cache valid condition
  ZZERO6           3;              % set to zero for IX2 valid comparison
  ZIXVALID2        1;              % IX 3 cache valid condition
  ZZERO2           3;              % set to zero for IX3 valid comparison
                                   % 1 16 bit-compare using C16DATA
  ZFUPC           16;              % microprogram counter condition
                                   % 1 4 bit-compare using C8DATA
  ZBRANCH          4;              % branch control field of sequencer
                                   % 4 1 bit-compare using C8DATA
  ZOPCOMPL         1;              % opcomp/pop lock - active low
  ZLIXEDFF         1;              % LIX'ed flip flop
  ZINIXHIT         1;              % indirect/index hit condition
  ZCODEHIT         1;              % code hit condition
  ZADRERREN        1;              % enable addr error comparison
  ZIXCAH0EN        1;              % enable IX1 cache inv. comparison
  ZXMEVNTEN        1;              % enable XM to OF event comparison
  ZDATHITEN        1;              % enable data hit comparison
  ZLOKFULEN        1;              % enable lock full comparison
  ZIXCAH1EN        1;              % enable IX2 cache inv. comparison
  ZIXCAH2EN        1;              % enable IX3 cache inv. comparison
  ZIXVLD0EN        1;              % enable IX1 cache valid comparison
  ZIXVLD1EN        1;              % enable IX2 cache valid comparison
  ZIXVLD2EN        1;              % enable IX3 cache valid comparison
  ZFUPCEN          1;              % enable FUPC comparison
  ZBRANCHEN        1;              % enable branch field comparison
  ZOPCOMPEN        1;              % enable opcomplete comparison
  ZLIXEDEN         1;              % enable LIX'ed f/f comparison
  ZINXHITEN        1;              % enable indirect/ix hit comparison
  ZCODHITEN        1;              % enable code hit comparison
  BPSTOPEN         1;              % backplane stop enable
  LOCALEN          1;              % local A enable
  CMP8NOT          1;              % Compare-8 not selected
  CFUPCOREN        1;              % FUPC 'OR' enable
  CMP8OREN         1;              % Compare-8 'OR' enable
  CMP40OREN        1;              % Compare-40 'OR' enable
  FUPCANDEN        1;              % FUPC 'AND' enable
  CMP8ANDEN        1;              % Compare-8 'AND' enable
  CP40ANDEN        1;              % Compare-40 'AND' enable
%
```

```
CHNEND;
%
REGLIST         YFC;
%
REGEND;
%
%
```

OPERAND FETCH DATA CARD

```
% There are 112 bits of registers in OF Data Card Maintenance Chain.
%
% Register Name    # Bits          % Comments
%
CHAIN            YFD;              % OF data card maint chain, 112 bits
%
%                                  CLKMNT - 16 bits
  ERRORIG          1;              % error ignore
  OVERRUN          1;              % over run
  COUNTERFF        1;              % counter f/f
  CLKBAD           1;              % clock bad
  SKEWLOW          1;              % skew low f/f
  SKEWHI           1;              % skew hi f/f
  MODIFYSKEW       1;              % modify skew f/f
  SKEWREG          9;              % skew register
%
%                                  STOP - 96 bits
  ANDFF            1;              % AND f/f
  ORFF             1;              % OR f/f
  PRETRIG          1;              % pre-trigger f/f
  LOADFF           2;              % load f/f
  SETCLRFF         2;              % set clr f/f
  CBUS            40;              % read bus compare data
  CMP16           16;              % spare - not used
  CADDR            8;              % C address
  CBUSDIGEN       10;              % CBUS digit compare enable
  CMP16EN          1;              % C16 word enable - not used
  CADRDIGEN        1;              % C address upper digit enable
  CADRBITEN        4;              % C address lower 4 bit enable
  BPSTOPEN         1;              % backplane stop enable
```

```
  LOCALEN          1;              % local A enable
  CADRCMPL         1;              % C address compare output invert
  CMP16OREN        1;              % Compare-16 OR enable
  CADROREN         1;              % C address compare OR enable
  CBUSOREN         1;              % CBUS compare OR enable
  CMP16ANEN        1;              % Compare-16 AND enable
  CADRANDEN        1;              % C address compare AND enable
  CBUSANDEN        1;              % CBUS compare AND enable
%
CHNEND;
%
REGLIST         YFD;
%
REGEND;
```

10.1.2 OPERAND FETCH REGISTER LIST

10.1.2.1 OPERAND FETCH CONTROL CARD

```
% Operand Fetch Control Card register list
%
%The folowing is a list of registers defined for the OFC card.
%
REGLIST          FC;
%
  OFCERRORS      27;      % error conditions generated by the
%                            OFC card
%
    MSBDIFF2;             % CS (13:2) Mismatch register; XMSEQ1 copy 2
    MSBDIFF1;             % CS (13:2) Mismatch register; XMSEQ1 copy 1
    MSBDIFF0;             % CS (13:2) Mismatch register; XMSEQ1 copy 0
    LSBDIFF1;             % CS (1:0) Mismatch register; XMSEQ2 copy 1
    LSBDIFF0;             % CS (1:0) Mismatch register; XMSEQ2 copy 0
    LOKOVRFLW;            % Lock table overflow error
    LOKCTLERR;            % Internal LOKCTL error ff
    LOCKERR7;             % Internal LOKCMP error ff from slice 7
    LOCKERR6;             % Internal LOKCMP error ff from slice 6
    LOCKERR5;             % Internal LOKCMP error ff from slice 5
    LOCKERR4;             % Internal LOKCMP error ff from slice 4
    LOCKERR3;             % Internal LOKCMP error ff from slice 3
    LOCKERR2;             % Internal LOKCMP error ff from slice 2
    LOCKERR1;             % Internal LOKCMP error ff from slice 1
    LOCKERR0;             % Internal LOKCMP error ff from slice 0
    OPLK1ERR0;            % OPLOOK copy 1 error ff in FOFFRZ
    OPLK0ERR0;            % OPLOOK copy 0 error ff in FOFFRZ
    OFFRZERR;             % Internal FOFFRZ error ff
    OPLUK1ERR0;           % Internal OPLOOK error ff from slice 1
    OPLUK1ERR1;           % Internal OPLOOK error ff from slice 1
    OPLUK0ERR0;           % Internal OPLOOK error ff from slice 0
    OPLUK0ERR1;           % Internal OPLOOK error ff from slice 0
    CREGERR3;             % Internal CREG parity error from slice 3
    CREGERR2;             % Internal CREG parity error from slice 2
    CREGERR1;             % Internal CREG parity error from slice 1
    CREGERR0;             % Internal CREG parity error from slice 0
    CMDPARERR #           % INTCMD parity error from OFTEST
```

```
%
%
  LIVEFRZ          4;      % Live freeze conditions in OF
%
    RQEMPTFRZ;             % live freeze caused by RQ not valid
    AWFULLFRZ;             % live freeze caused by AW int full
    STOPBIT1;              % Live freeze caused by local stop
    STOPBIT0 #             % Live freeze caused by local stop
%
%
  STOPBITS         2;              % stop bits; both bits must be set
%                                    and reset simultaneously
    STOPBIT1;              % Live freeze caused by local stop
    STOPBIT0 #             % Live freeze caused by local stop
%
%
  CODEHTSTAT       8;              % code hit conditions generated by
%                                    lock entries
    CODEHIT7;
    CODEHIT6;
    CODEHIT5;
    CODEHIT4;
    CODEHIT3;
    CODEHIT2;
    CODEHIT1;
    CODEHIT0 #
%
%
  DATAHTSTAT       8;              % data hit condition generated by
%                                    lock unit entries
    DATAHIT7;
    DATAHIT6;
    DATAHIT5;
    DATAHIT4;
    DATAHIT3;
    DATAHIT2;
    DATAHIT1;
    DATAHIT0 #
%
%
  INIXHTSTAT       8;              % index/indirect hit condition
%                                    generated by lock unit entries
```

```
    INIXHIT7;
    INIXHIT6;
    INIXHIT5;
    INIXHIT4;
    INIXHIT3;
    INIXHIT2;
    INIXHIT1;
    INIXHIT0 #
%
%
  CSADR0          2;            % copy 0 of control store address
%
    UPC0MSB;
    UPC0LSB #
%
%
  CSADR1          2;            % copy 1 of control store address
%
    UPC1MSB;
    UPC1LSB #
%
%
  CSADR2          2;            % copy 2 of control store address
%
    UPC2MSB;
    UPC1LSB #
%
%
  STACK0          2;             % copy 0 of stack registers
%
    STK0MSB;
    STK0LSB #
%
%
  STACK1          2;             % copy 1 of stack registers
%
    STK1MSB;
    STK1LSB #
%
%
  STACK2          2;             % copy 2 of stack registers
%
    STK2MSB;
```

```
    STK1LSB #
%
%
  OPLKDAT          2;              % combined copies of OPLKDAT
%                                    registers
    OPLKDAT1;
    OPLKDAT0 #
%
%
  IXCACHSTAT       4;              % IX4-7 cache status bits
%
    IX7CACHE;
    IX6CACHE;
    IX5CACHE;
    IX4CACHE #
%
%
  CONDFFS          3;              % condition FFs stored in OFTEST
%
    CONDFF3;
    CONDFF2;
    CONDFF1 #
%
%
  CREG            23;              % 80 bits of CREG field to access
%                                    control store rams
    SPARERAM0;
    WORDPRTY0;
    PTEST0;
    BADR0;
    AADR0;
    CADR0;
    ASRC0;
    BSRC0;
    CWE0;
    CSRC;
    INTCMD;
    ALUCMD;
    BRANCH;
    PUSHSTK;
    TSTCND;
    FBUS1WE;
    FBUS2WE;
```

```
    FB2SRC;
    EXTCMD;
    SPARE1;
    BEGADSYNC;
    SLFCHK;
    JMPADR #
%
%
  SPARERAM          2;              % spare ram control registers; both
%                                     copies combined
    SPARERAM1;
    SPARERAM0 #
%
%
  PTEST             2;              % PTEST control registers; both copies
%                                     combined
    PTEST1;
    PTEST0 #
%
%
  BADR              2;              % B_address registers; both copies
%                                     combined
    BADR1;
    BADR0 #
%
%
  AADR              2;              % A_address registers; both copies
%                                     combined
    AADR1;
    AADR0 #
%
%
  FQRDADD0          2;              %FETCH QUEUE read address for hidden
%                                     state read, copy 0; Note that the
    SPARERAM0;                      %top bit is only a dummy register.
    AADR0 #
%
%
  FQRDADD1          2;              %FETCH QUEUE read address for hidden
%                                     state read, copy 1. Note that the
    SPARERAM0;                      %top bit is only a dummy register.
    AADR1 #
%
```

```
%
  CADR            2;              % C_address registers; both copies
%                                   combined
    CADR1;
    CADR0 #
%
%
  ASRC            2;              % A_source control registers; both
%                                   copies combined
    ASRC1;
    ASRC0 #
%
%
  BSRC            2;              % B_source control registers; both
%                                   copies combined
    BSRC1;
    BSRC0 #
%
%
  CWE             2;              % C_write enable registers; both
%                                   copies combined
    CWE1;
    CWE0 #
%
%
  CSDAT0         23;              % Control store data in for
%                                   hidden state access
    SPARERAM0;
    WORDPRTY0;
    PTEST0;
    BADR0;
    AADR0;
    CADR0;
    ASRC0;
    BSRC0;
    CWE0;
    CSRC;
    INTCMD;
    ALUCMD;
    BRANCH;
    PUSHSTK;
    TSTCND;
    FBUS1WE;
```

```
    FBUS2WE;
    FB2SRC;
    EXTCMD;
    SPARE1;
    BEGADSYNC;
    SLFCHK;
    JMPADR #
%
%
  CSDAT1          23;              % Control store data in for
%                                    hidden state access
    SPARERAM1;
    WORDPRTY1;
    PTEST1;
    BADR1;
    AADR1;
    CADR1;
    ASRC1;
    BSRC1;
    CWE1;
    CSRC;
    INTCMD;
    ALUCMD;
    BRANCH;
    PUSHSTK;
    TSTCND;
    FBUS1WE;
    FBUS2WE;
    FB2SRC;
    EXTCMD;
    SPARE1;
    BEGADSYNC;
    SLFCHK;
    JMPADR #
%
%
  LOCKERRORS       8;              % error conditions generated by all
%                                    the LOKCMP arrays
    LOCKERR7;
    LOCKERR6;
    LOCKERR5;
    LOCKERR4;
    LOCKERR3;
```

```
    LOCKERR2;
    LOCKERR1;
    LOCKERR0 #
%
%

REGEND;
```

10.1.2.2 OPERAND FETCH DATA CARD

```
%
%The following is a list of registers defined for the OFD card.
%
REGLIST          FD;
%
  OFDERRORS      21;     % error conditions generated by the
%                        OFD card
%
    AUREGERR1;           % AUREG internal error ff for copy1
    AUREGERR0;           % AUREG internal error ff for copy0
    OFFRZERR;            % Internal FOFFRZ error ff
    BBUSERR;             % BBUS parity error
    OFFCTRLERR;          % OFFCTL internal error
    FQRQERR2;            % AADR parity error from FANDRQ
    FQRQERR1;            % AADR parity error from FANDRQ
    FQRQERR0;            % AADR parity error from FANDRQ
    PMUXERR;             % Pmux internal parity error
    ALENRAMERR;          % Parity error on ALEN rams
    BLENRAMERR;          % Parity error on BLEN rams
    OPLENERR1;           % OPLENIN copy 1 error on FE digit
    OPLENERR0;           % OPLENIN copy 0 error on FE digit
    AWCTLERR;            % AWRCTL internal error ff
    OPLOUTERR;           % OPLOUT internal error ff
    RBUSMSBERR;          % RBUS error field parity error
    RBUSDATERR;          % RBUS data parity error
    RBUSTAGERR;          % RBUS tag parity error
    FBUSDATERR;          % FBUS1 data parity error
    SOFTERR1;            % Soft ptest parity error from PTEST1
    SOFTERR0 #           % Soft ptest parity error from PTEST0
%
%
  LIVEFRZ        4;      % Live freeze conditions in OF
%
    RQEMPTFRZ;           % live freeze caused by RQ not valid
    AWFULLFRZ;           % live freeze caused by AW int full
    STOPBIT1;            % Live freeze caused by local stop
    STOPBIT0 #           % Live freeze caused by local stop
%
%
```

```
  STOPBITS          2;      % stop condition for OFD; both bits
%                             must be set and reset simultaneously
    STOPBIT1;
    STOPBIT0 #
%
%
  AWCOMMAND         5;              % AW_command bits for AW_BUS
%
    AWCMNDB4;
    AWCMNDB3;
    AWCMNDB2;
    AWCMNDB1;
    AWCMNDB0 #
%
%
  AWLENGTH          4;              % AW_length field for AW bus
%
  AWLENB3;
  AWLENB2;
  AWLENB1;
  AWLENB0 #
%
%
  BEGADR            3;              % Begin address for lock unit
%
    ABREGB2;
    ABREGB1;
    ABREGB0 #
%
%
  AWTAGBITS         4;               % AWBUS tag bits; refer to AWRCTL
%                                      array for explanations
    AWTAGBIT3;
    AWTAGBIT2;
    AWTAGBIT1;
    AWTAGBIT0 #
%
%
  OPRAMWE           2;               % oplength ram write enable registers
%
    OPRAMWE1;
    OPRAMWE0 #
%
```

```
%
  AREG            6;              % ABUS register; most significant bit
%                                   is parity
    ADDBUSPRTY;
    ABREGB4;
    ABREGB3;
    ABREGB2;
    ABREGB1;
    ABREGB0 #
%
%
  BREGMSP         7;              % Most significant portion
%                                   of BBUS
    WFOPPRTY;
    WFPRTYAFBF;
    WFXDIGPRTY;
    WFXDIGMSB;
    WFXDIGLSD;
    WFEDIGMSD;
    WFEDIGLSD #

  BREG           40;              % BBUS register;
%
    BBREGB39;
    BBREGB38;
    BBREGB37;
    BBREGB36;
    BBREGB35;
    BBREGB34;
    BBREGB33;
    BBREGB32;
    BBREGB31;
    BBREGB30;
    BBREGB29;
    BBREGB28;
    BBREGB27;
    BBREGB26;
    BBREGB25;
    BBREGB24;
    BBREGB23;
    BBREGB22;
    BBREGB21;
    BBREGB20;
```

```
    BBREGB19;
    BBREGB18;
    BBREGB17;
    BBREGB16;
    BBREGB15;
    BBREGB14;
    BBREGB13;
    BBREGB12;
    BBREGB11;
    BBREGB10;
    BBREGB9;
    BBREGB8;
    BBREGB7;
    BBREGB6;
    BBREGB5;
    BBREGB4;
    BBREGB3;
    BBREGB2;
    BBREGB1;
    BBREGB0 #
%
%
  LITFILIN       41;              % LIT FILE input data for
%                                   pathtest
    WFEDIGLSD;
    BBREGB39;
    BBREGB38;
    BBREGB37;
    BBREGB36;
    BBREGB35;
    BBREGB34;
    BBREGB33;
    BBREGB32;
    BBREGB31;
    BBREGB30;
    BBREGB29;
    BBREGB28;
    BBREGB27;
    BBREGB26;
    BBREGB25;
    BBREGB24;
    BBREGB23;
    BBREGB22;
```

```
    BBREGB21;
    BBREGB20;
    BBREGB19;
    BBREGB18;
    BBREGB17;
    BBREGB16;
    BBREGB15;
    BBREGB14;
    BBREGB13;
    BBREGB12;
    BBREGB11;
    BBREGB10;
    BBREGB9;
    BBREGB8;
    BBREGB7;
    BBREGB6;
    BBREGB5;
    BBREGB4;
    BBREGB3;
    BBREGB2;
    BBREGB1;
    BBREGB0 #
%
%
  ALENIN        12;            % ALENGTH data input for hidden state
%                                  write
    BBREGB35;
    BBREGB34;
    BBREGB33;
    BBREGB32;
    BBREGB31;
    BBREGB30;
    BBREGB29;
    BBREGB28;
    BBREGB27;
    BBREGB26;
    BBREGB25;
    BBREGB24 #
%
%
  ALENINP       13;            % A-LENGTH data input for hidden state
%                                  write with parity bit
    BBREGB36;
```

```
    BBREGB35;
    BBREGB34;
    BBREGB33;
    BBREGB32;
    BBREGB31;
    BBREGB30;
    BBREGB29;
    BBREGB28;
    BBREGB27;
    BBREGB26;
    BBREGB25;
    BBREGB24 #
%
%
  BLENINP          13;            % B-LENGTH data input for hidden
%                                     state write
    BBREGB24;
    BBREGB23;
    BBREGB22;
    BBREGB21;
    BBREGB20;
    BBREGB19;
    BBREGB18;
    BBREGB17;
    BBREGB16;
    BBREGB15;
    BBREGB14;
    BBREGB13;
    BBREGB12 #
%
%
  BCLENIN          12;            % B-, C-LENGTH data input for hidden
%                                     state write ( bank 1 for CLENGTH )
    BBREGB23;
    BBREGB22;
    BBREGB21;
    BBREGB20;
    BBREGB19;
    BBREGB18;
    BBREGB17;
    BBREGB16;
    BBREGB15;
    BBREGB14;
```

```
    BBREGB13;
    BBREGB12 #
%
%
  CCLENIN        12;            % CLENGTH data input for hidden state
%                                 write bank 0
    BBREGB11;
    BBREGB10;
    BBREGB9;
    BBREGB8;
    BBREGB7;
    BBREGB6;
    BBREGB5;
    BBREGB4;
    BBREGB3;
    BBREGB2;
    BBREGB1;
    BBREGB0 #
%
%
  ALENOUT         3;             % ALENGTH data output for hidden state
%                                 read
    ALRAMPRTY;
    AOPTIMAL;
    ALEN #
%
%
  ALENOUTP        4;             % ALENGTH data output for hidden state
%                                 read with parity bit
    ALENRAMERR;
    ALRAMPRTY;
    AOPTIMAL;
    ALEN #
%
%
  BLENOUT         3;             % BLENGTH data output for hidden state
%                                 read
    BLRAMPRTY;
    BOPTIMAL;
    BLEN #
%
%
  BLENOUTP        4;             % BLENGTH data output for hidden state
```

```
%                                   read with parity bit
    BLENRAMERR;
    BLRAMPRTY;
    BOPTIMAL;
    BLEN #
%
%
  CLENOUT          2;              % CLENGTH data output for hidden state
%                                   read
    CLRAMPRTY;
    CLEN #
%
%
  FLUTOUT          9;              % FLUT data output for hidden state
%                                   read
    LITMASK;
    FLUTSPARE;
    OPTCODE1;
    OPTCODE0;
    OPLITLSD;
    ALENCODE;
    OPLTPRTY;
    CFCODE1;
    CFCODE0 #
%
%
  FLUTOUTP         11;              % FLUT data output for hidden state
%                                   read - entire FLUT..422+474
    LITMASK;
    FLUTSPARE;
    OPTCODE1;
    OPTCODE0;
    FLUTPRTY;
    OPLITMSB;
    OPLITLSD;
    ALENCODE;
    OPLTPRTY;
    CFCODE1;
    CFCODE0 #
%
%
  FLUTINP          7;              % FLUT data input for hidden state
%                                   write - entire FLUT..422+474
```

```
    WFXDIGLSD;
    WFEDIGMSD;
    WFEDIGLSD;
    BBREGB39;
    BBREGB38;
    BBREGB37;
    BBREGB36 #
%
%
  FLUTIN          6;              % FLUT data input for hidden state
%                                   write
    WFXDIGLSD;
    WFEDIGLSD;
    BBREGB39;
    BBREGB38;
    BBREGB37;
    BBREGB36 #
%
%
  FLUTPAROUT      2;              % FLUT parity ram output data
%                                   for hidden state read
    FLUTPRTY;
    OPLITMSB #
%
%
  FLUTPARIN       1;              % FLUT parity ram data for
%                                   hidden state write
     WFEDIGMSD #
%
%
  FLUTPARAD0      2;
%
    IGNORERR;
    PAGE0OP #
%
%
  FLUTPARAD1      2;
%
    IGNORERR;
    PAGE1OP #
%
%
  ALENADR0        2;              % address for the ALENGTH rams,
```

```
%                                   copy 0
    PAGE0AC;
    PAGE0AF #
%
%
  ALENADR1        2;              % address for the ALENGTH rams,
%                                   copy 1
    PAGE1AC;
    PAGE1AF #
%
%
  ALDATADR        3;              % address for the ALENGTH rams,
%
    ALENCODE;
    PAGE0AC;
    PAGE0AF #
%
%
  ALPARADR        3;              % address for the ALENGTH
%                                   PARITY rams;
    IGNORERR;
    PAGE0AC;
    PAGE0AF #

  ALPARADR0        3;              % address for the ALENGTH
%                                   PARITY rams; copy 0
    IGNORERR;
    PAGE0AC;
    PAGE0AF #
%
%
  ALPARADR1        3;              % address for the ALENGTH
%                                   PARITY rams; copy 1
    IGNORERR;
    PAGE1AC;
    PAGE1AF #
%
%
  BLENADR0        2;              % address for the BLENGTH rams,
%                                   copy 0
    PAGE0BC;
    PAGE0BF #
%
```

```
%
  BLENADR1          2;              % address for the BLENGTH rams,
%                                       copy 1
    PAGE1BC;
    PAGE1BF #
%
%
  BLDATADR          3;              % address for the BLENGTH rams,
%
    ALENCODE;
    PAGE0BC;
    PAGE0BF #
%
%
  BLPARADR          3;              % address for the BLENGTH
%                                     parity ram;
    IGNORERR;
    PAGE0BC;
    PAGE0BF #
%
%
  BLPARADR0         3;              % address for the BLENGTH
%                                     parity ram; copy 0
    IGNORERR;
    PAGE0BC;
    PAGE0BF #
%
%
  BLPARADR1         3;              % address for the BLENGTH
%                                     parity ram; copy 1
    IGNORERR;
    PAGE1BC;
    PAGE1BF #
%
%
  CLENADR0          2;             % address for the CLENGTH rams,
%                                      copy 0
    PAGE0CC0;
    CF0 #
%
%
  CLENADR1          2;             % address for the CLENGTH rams,
%                                      copy 1
```

```
    PAGE1CC0;
    CF1 #
%
%
  OPSYL0          3;          % Page 0 OPSYL from lenin slice 0
%
    PAGE0OP;
    PAGE0AF;
    PAGE0BF #
%
%
  OPSYL1          3;          % Page 1 OPSYL from lenin slice 0
%
    PAGE1OP;
    PAGE1AF;
    PAGE1BF #
%
%
  FBUS2REG       10;          % FBUS2 register in OPLOUT array
%
  FBUS2REGB9;
  FBUS2REGB8;
  FBUS2REGB7;
  FBUS2REGB6;
  FBUS2REGB5;
  FBUS2REGB4;
  FBUS2REGB3;
  FBUS2REGB2;
  FBUS2REGB1;
  FBUS2REGB0 #
%
%
  PTESTOUT       22;          % SOFT PTEST output data for
%                             hidden state read
    IFLMSDO1;
    IFUNDG1;
    OFLMT1;
    OFUNDG1;
    UNDGAB1;
    SOFTERR1;
    APARERR1;
    SSERR1;
    BEGPAR0;
```

```
    SOFTPRD1;
    PTREG1;
    IFLMSDO0;
    IFUNDG0;
    OFLMT0;
    OFUNDG0;
    UNDGAB0;
    SOFTERR0;
    APARERR0;
    SSERR0;
    BEGPAR1;
    SOFTPRD0;
    PTREG0 #
%
%
  PTESTIN        30;            % SOFT PTEST input data for
%                                 hidden state write
    BBREGB29;
    BBREGB28;
    BBREGB27;
    BBREGB26;
    BBREGB25;
    BBREGB24;
    BBREGB23;
    BBREGB22;
    BBREGB21;
    BBREGB20;
    BBREGB19;
    BBREGB18;
    BBREGB17;
    BBREGB16;
    BBREGB15;
    BBREGB14;
    BBREGB13;
    BBREGB12;
    BBREGB11;
    BBREGB10;
    BBREGB9;
    BBREGB8;
    BBREGB7;
    BBREGB6;
    BBREGB5;
    BBREGB4;
```

```
    BBREGB3;
    BBREGB2;
    BBREGB1;
    BBREGB0 #
%
%
  PTESTADD       10;            % SOFT PTEST address registers for
%                                 hidden state access
    CBREGD9;
    CBREGD8;
    CBREGD7;
    CBREGD6;
    CBREGD5;
    CBREGD4;
    CBREGD3;
    CBREGD2;
    CBREGD1;
    CBREGD0 #
%
%
  PTADRBYTE0      2;             % SOFT PTEST address registers for
%                                % hidden state access
    CBREGD1;                     % byte 0 address
    CBREGD0 #
%
%
  PTADRBYTE1      2;             % SOFT PTEST address registers for
%                                % hidden state access
    CBREGD3;                     % byte 1 address
    CBREGD2 #
%
%
  PTADRBYTE2      2;             % SOFT PTEST address registers for
%                                % hidden state access
    CBREGD5;                     % byte 2 address
    CBREGD4 #
%
%
  PTADRBYTE3      2;             % SOFT PTEST address registers for
%                                % hidden state access
    CBREGD7;                     % byte 3 address
    CBREGD6 #
%
```

```
%
  PTADRBYTE4      2;              % SOFT PTEST address registers for
%                                 % hidden state access
    CBREGD9;                      % byte 4 address
    CBREGD8 #
%
%
  CBREG           11;             % CBUS register; most significant bit
%                                   is parity
    CBREGB40;
    CBREGD9;
    CBREGD8;
    CBREGD7;
    CBREGD6;
    CBREGD5;
    CBREGD4;
    CBREGD3;
    CBREGD2;
    CBREGD1;
    CBREGD0 #
%
%
  SCHPADIN        11;             % scratch pad input data set up; used
%                                   for hidden state write
    CBREGB40;
    CBREGD9;
    CBREGD8;
    CBREGD7;
    CBREGD6;
    CBREGD5;
    CBREGD4;
    CBREGD3;
    CBREGD2;
    CBREGD1;
    CBREGD0 #
%
%
  FQDATIN         17;              % FBUS1 queue write data setup
%

    WFXDIGPRTY;                   % hidden write to the FE_parity is
    WFXDIGPRTY;                   % not supported so a FF is put in
    FPARITY1;
```

```
    WFXDIGMSB;
    WFXDIGLSD;
    WFEDIGMSD;
    WFEDIGLSD;
    CBREGD9;
    CBREGD8;
    CBREGD7;
    CBREGD6;
    CBREGD5;
    CBREGD4;
    CBREGD3;
    CBREGD2;
    CBREGD1;
    CBREGD0 #
%
%
  FQTESTIN       11;              % FBUS1 queue write data setup
%                                   for pathtest

    FPARITY1;
    CBREGD9;
    CBREGD8;
    CBREGD7;
    CBREGD6;
    CBREGD5;
    CBREGD4;
    CBREGD3;
    CBREGD2;
    CBREGD1;
    CBREGD0 #
%
%

  FQDATOUT       10;              % FBUS1 queue read data for hidden
%                                   state read
    RFQORRQS4;
    RFQORRQS3;
    RFQORRQS2;
    RFQORRQS1;
    RFQORRQS0;
    ABREGB4;
    ABREGB3;
    ABREGB2;
```

```
    ABREGB1;
    ABREGB0 #
%
%
  FQTESTOUT       8;              % FBUS1 queue read data for hidden
%                                   state read during pathtest
    RFQORRQS2;
    RFQORRQS1;
    RFQORRQS0;
    ABREGB4;
    ABREGB3;
    ABREGB2;
    ABREGB1;
    ABREGB0 #
%
%
  FQ53OUT         7;              % FBUS1 queue read data for hidden
%                                   state read for IFOF pathtest
    RFQORRQS1;
    RFQORRQS0;
    ABREGB4;
    ABREGB3;
    ABREGB2;
    ABREGB1;
    ABREGB0 #
%
%
  RBUSQOUT       10;              % RBUS queue read data for hidden
%                                   state read
    RFQORRQS4;
    RFQORRQS3;
    RFQORRQS2;
    RFQORRQS1;
    RFQORRQS0;
    ABREGB4;
    ABREGB3;
    ABREGB2;
    ABREGB1;
    ABREGB0 #
%
%
  RBUSQIN        26;              % RBUS queue write data for hidden
%                                   state write
```

```
    BBREGB39;
    BBREGB38;
    BBREGB37;
    BBREGB36;
    BBREGB35;
    BBREGB34;
    BBREGB33;
    BBREGB32;
    BBREGB31;
    BBREGB30;
    BBREGB29;
    BBREGB28;
    BBREGB27;
    BBREGB26;
    BBREGB25;
    BBREGB24;
    CBREGD9;
    CBREGD8;
    CBREGD7;
    CBREGD6;
    CBREGD5;
    CBREGD4;
    CBREGD3;
    CBREGD2;
    CBREGD1;
   CBREGD0 #

REGEND;
```

10.1.3 ERROR DETECTION LOGIC

To be specified.

10.1.4 STOP LOGIC

The Instruction Fetch card contains stop logic to halt the system upon data comparision of the following busses and signals:

- o FBUS1$$$P - 53 bits on backplane
- o FBUS1ADR$P - 3 bits on backplane
- o FBUS1PAG$P - 2 bits on backplane
- o FBUS1LOAD$$P - 1 bit on backplane
- o FBUS2$$$$P - 10 bits on backplane
- o FBUS2ADR$P - 3 bits on backplane
- o FBUS2PAG$P - 2 bits on backplane
- o FBUS2LOAD$$P - 1 bit on backplane
- o FDATAHIT$$$P - 1 bit from OF
- o FOPTIMAL$$$P - 1 bit from OF
- o XWBUSTAG - 7 bits on backplane
- o XADRBUS$P - 40 bits on backplane
- o XWRTBUS$P - 40 bits on backplane
- o XWBUSLEN$P - 4 bits on backplane
- o MRBUS$$$P - 40 bits on backplane
- o MRBUSVALID$P - 1 bit on backplane
- o MRBUSTAG$P - 7 bits pipelined with MRBUS
- o XWBUSCMD$P - 5 bits on backplane

10.1.4 STOP LOGIC (Continued)

- XADRBUS$P REQUEST (FWBUSREQOF$P, FWBUSREQ1F$P, EWBUSREQXMDP, EWBUSREQXMXP) - 4 bits on backplane
- PCBUS - 27 bits internal to IF

These conditions can be 'ANDed' or 'ORed' together to form a stop condition. These conditions must be set up via the maintenance chain.

The Operand Fetch module has stop logic to halt the system upon data comparision of the following busses and signals:

- BRANCH FIELD - Branch field specified within CREG
- FOUPC - OF microprogram counter
- CBUS - 40 bits of CBUS data internal to OF DATA CARD
- IXVALID - 3 bits specifying validity of IX 1-3 cache
- CODE HIT - Code hit condition from OFTEST
- DATA HIT - Data hit condition from OFTEST
- IXIA HIT - Index/Indirect hit condition from OFTEST
- LIX'ED - Outstanding mobile index registers flag
- POPLOCKL - OPCOMP signal from XM during normal operation
- LOCKFUL - Lock full condition, from AT-LEAST-1L

10.1.4 STOP LOGIC (Continued)

- o XMEVENT - XM event wire to OF
- o ADDRERR - Fetch address error register (IF,OF)
- o IXCACHE - 3 bits specifying if any one of the 3 index registers is being pre-cached with data yet to be written by XM.

These stop conditions are separated into four groups with one group on OFD and three groups on OFC.

Group I (OFC)
FOUPC - 16 bit compare

Group II (OFC)
Branch field
Code list
LIX_ED
POPLOCKL

Group III (OFC)
Address
IXCACHE (2:0)
IXVALID (2:0)
Data Hit
XMEVENT
LOCKFUL

Group IV (ODC)
CBUS

If more than one element is specified with a group, they will be ANDed internally before used as a STOP condition. Each group can be ANDed/ORed with other groups or conditions within the processor. OF does not perform self stop and it relies on the SMC to turn its clock off once a stop condition is met (3 locks later).

10.2 TIMING

To be specified.

11 BOARD LAYOUT

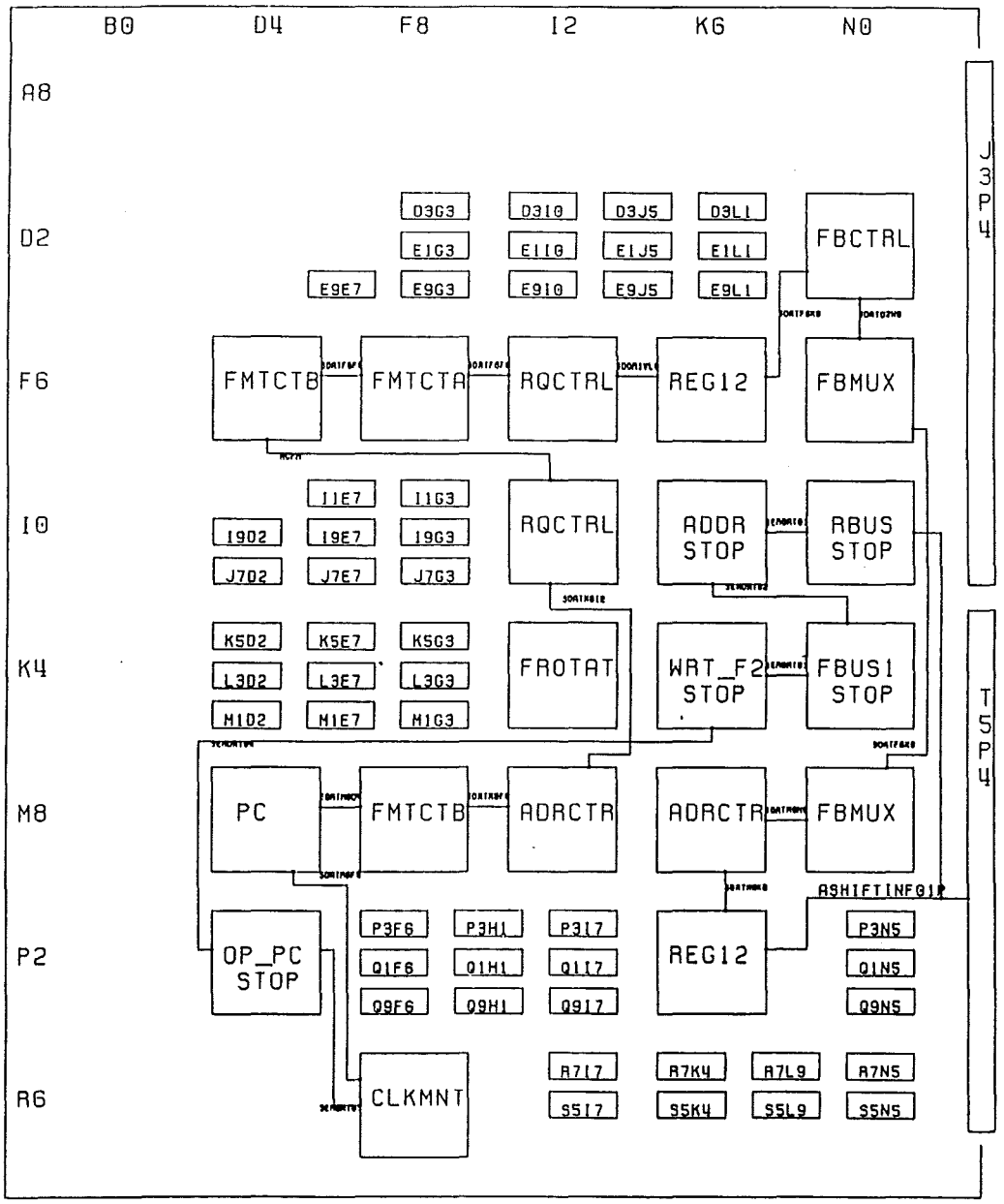


FIGURE 11-1 INSTRUCTION FETCH BOARD

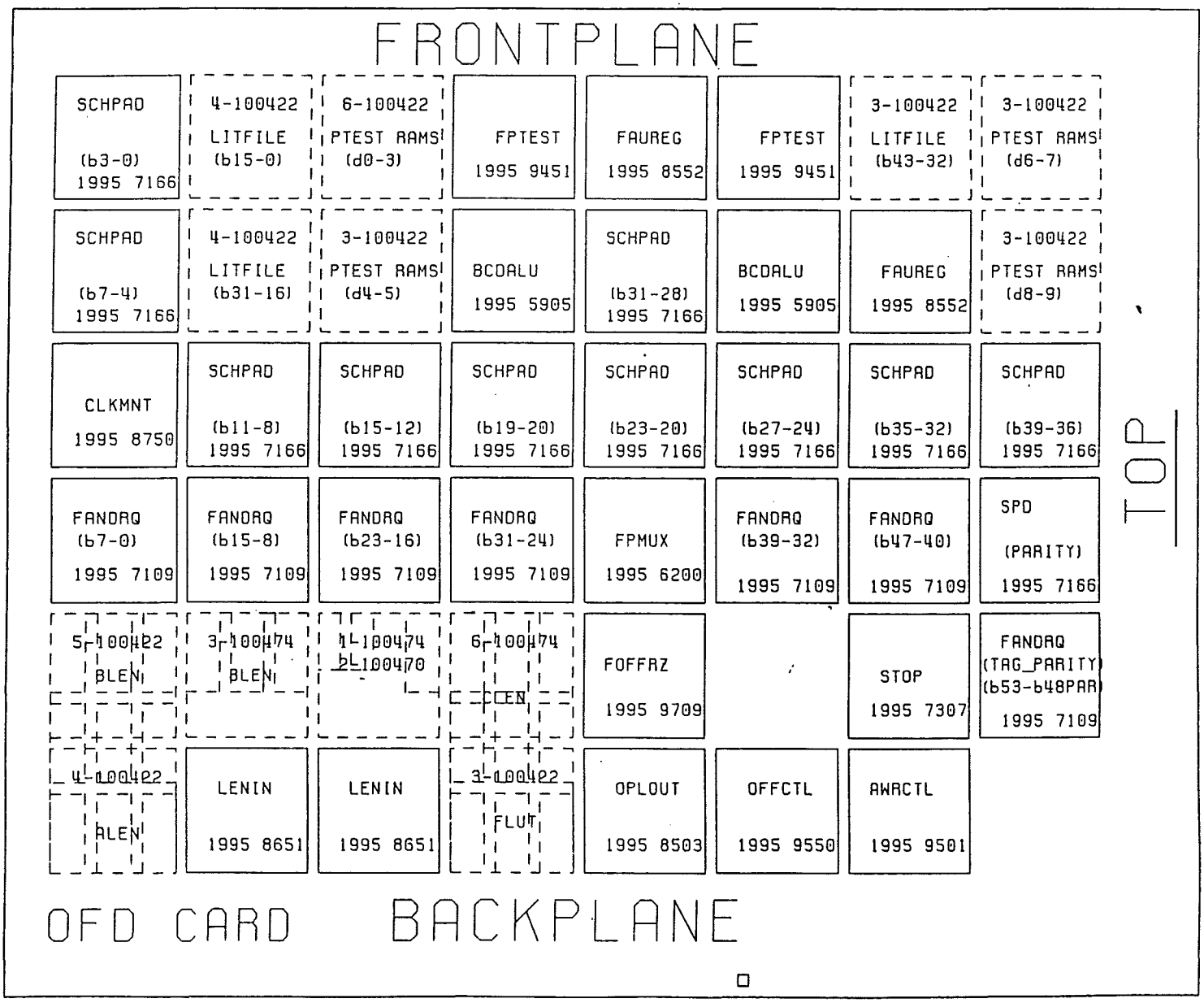


FIGURE 11-2 V500 OF DATA CARD LAYOUT

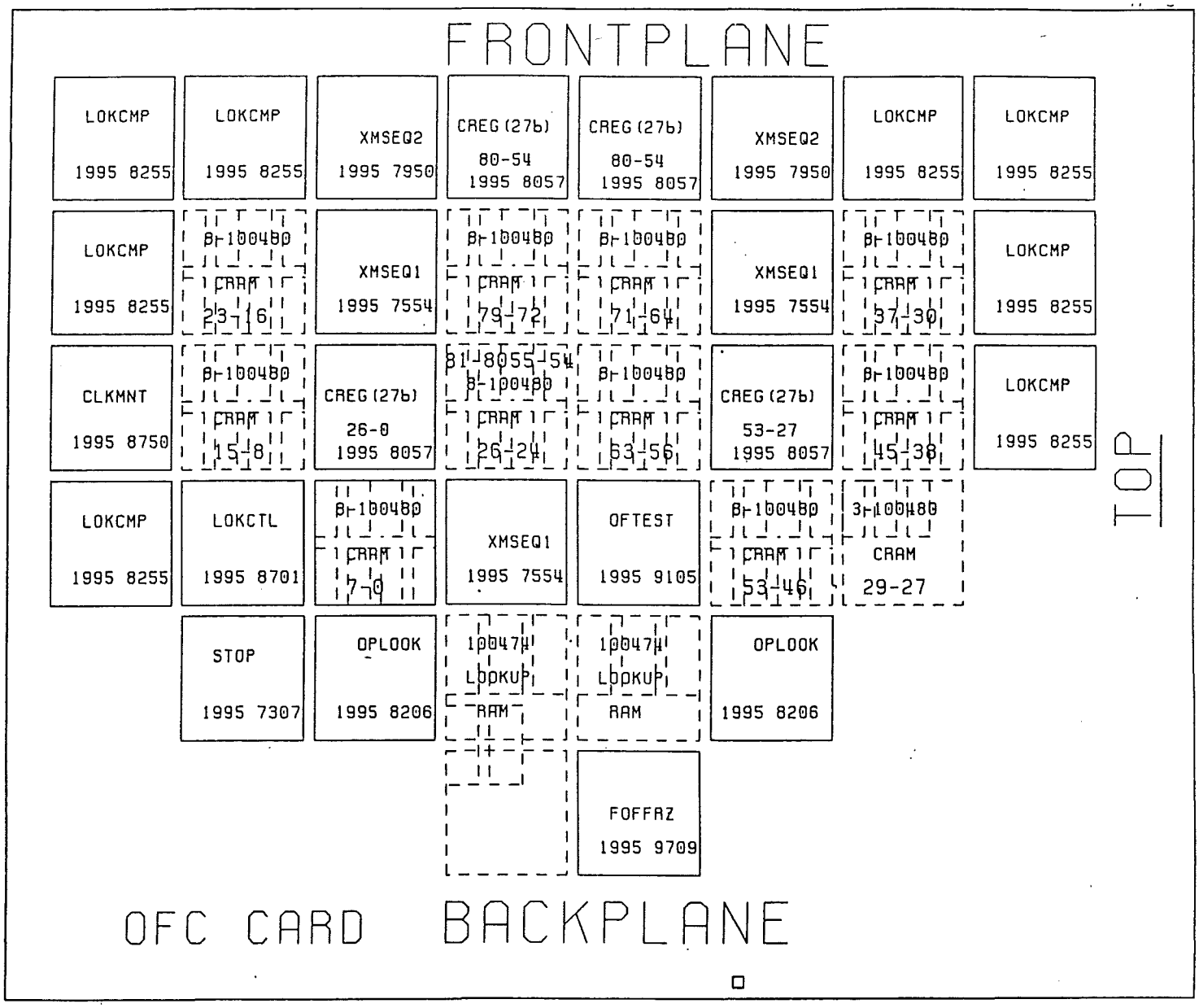


FIGURE 11-3 V500 OF CONTROL CARD LAYOUT

12 BACKPLANE/FRONTPLANE DEFINITION

BACKPLANE CONNECTOR - J3P4

	A	B	C	D
01:	BUS-5V01A3P4	BUS-5V01A3P5	BUS-5V01A3P6	BUS-5V01A3P7
02:	BUS-5V01A4P4	BUS-5V01A4P5	BUS-5V01A4P6	BUS-5V01A4P7
03:	BUS-5V01A5P4	BUS-5V01A5P5	BUS-5V01A5P6	BUS-5V01A5P7
04:	.....	GND...01A6P5	.....	.....
05:	GND...(A7P3)	.....	GND...01A7P7	GND...01A7P7
06:	.....	BUS-2V01A8P5	.....	.....
07:	GND...(A9P3)	.....	GND...01A9P7	GND...01A9P7
08:	.....	BUS-2V01B0P5	.....	.....
09:	GND...(B1P3)	.....	GND...01B1P7	GND...01B1P7
10:	.....	BUS-2V01B2P5	.....	.....
11:	GND...(B3P3)	.....	GND...01B3P7	GND...01B3P7
12:	.....	BUS-2V01B4P5	.....	.....
13:	GND...(B5P3)	E$TIME0UT1$P	GND...01B5P7	GND...01B5P7
14:	.....	BUS-2V01B6P5	.....	.....
15:	GND...(B7P3)	.....	GND...01B7P7	GND...01B7P7
16:	.....	BUS-2V01B8P5	.....	.....
17:	GND...(B9P3)	.....	GND...01B9P7	GND...01B9P7
18:	.....	BUS-2V01C0P5	.....	.....
19:	GND...(C1P3)	.....	GND...01C1P7	GND...01C1P7
20:	.....	BUS-2V01C2P5	.....	.....
21:	GND...(C3P3)	.....	GND...01C3P7	GND...01C3P7
22:	.....	BUS-2V01C4P5	.....	.....
23:	GND...(C5P3)	.....	GND...01C5P7	GND...01C5P7
24:	FCEXTCMDPARP	BUS-2V01C6P5	FDAFGTBF$$$P	FDAFEQBF$$$P
25:	GND...(C7P3)	FCCS0URCE-4P	GND...01C7P7	GND...01C7P7
26:	.....	BUS-2V01C8P5	.....	.....
27:	GND...(C9P3)	.....	GND...01C9P7	GND...01C9P7
28:	.....	BUS-2V01D0P5	.....	.....
29:	GND...(D1P3)	.....	GND...01D1P7	GND...01D1P7
30:	.....	BUS-2V01D2P5	.....	.....
31:	GND...(D3P3)	.....	GND...01D3P7	GND...01D3P7
32:	FCCS0URCE-3P	BUS-2V01D4P5	FCCS0URCE-2P	FCCS0URCE-1P
33:	GND...(D5P3)	FCCS0URCE-0P	GND...01D5P7	GND...01D5P7
34:	.....	BUS-2V01D6P5	.....	.....
35:	GND...(D7P3)	.....	GND...01D7P7	GND...01D7P7

36:	.....	BUS-2V01D8P5	.....	.....
37:	GND...(D9P3)	.....	GND...01D9P7	GND...01D9P7
38:	.....	BUS-2V01E0P5	.....	.....
39:	GND...(E1P3)	.....	GND...01E1P7	GND...01E1P7
40:	.....	BUS-2V01E2P5	.....	.....
41:	GND...(E3P3)	.....	GND...01E3P7	GND...01E3P7
42:	.....	BUS-2V01E4P5	.....	.....
43:	GND...(E5P3)	.....	GND...01E5P7	GND...01E5P7
44:	FBUS10PPRTYP	BUS-2V01E6P5	.....	.....
45:	.....	BUS-2V01F0P5	.....	.....
46:	GND...(F1P3)	.....	GND...01F1P7	GND...01F1P7
47:	.....	BUS-2V01F2P5	.....	.....
48:	GND...(F3P3)	.....	GND...01F3P7	GND...01F3P7
49:	.....	BUS-2V01F4P5	.....	.....
50:	GND...(F5P3)	.....	GND...01F5P7	GND...01F5P7
51:	FBUS1$$$-23P	BUS-2V01F6P5	FBUS1$$$-22P	FBUS1$$$-21P
52:	GND...(F7P3)	FBUS1$$$-20P	GND...01F7P7	GND...01F7P7
53:	.....	BUS-2V01F8P5	.....	.....
54:	GND...(F9P3)	.....	GND...01F9P7	GND...01F9P7
55:	.....	BUS-2V01G0P5	.....	.....
56:	GND...(G1P3)	.....	GND...01G1P7	GND...01G1P7
57:	FBUS1$$$-52P	BUS-2V01G2P5	FBUS1$$$-51P	FBUS1$$$-50P
58:	GND...(G3P3)	FBUS1$$$-49P	GND...01G3P7	GND...01G3P7
59:	FBUS1$$$-48P	BUS-2V01G4P5	FIPARXDIG$$P	.....
60:	GND...(G5P3)	.....	GND...01G5P7	GND...01G5P7
61:	EXM20F$$$$$P	BUS-2V01G6P5	F0F2XM$$$$$P	FQCFETFLSH1P
62:	GND...(G7P3)	FYC0FSH0VE$P	GND...01G7P7	GND...01G7P7
63:	FSH0VE$$$$$P	BUS-2V01G8P5	.....	.....
64:	GND...(G9P3)	FQCINCMD1-0P	GND...01G9P7	GND...01G9P7
65:	F0$FBUS$REQP	BUS-2V01H0P5	ESYSFLUSHP	EFETFLUSHP
66:	GND...(H1P3)	FIFFLUSH$P	GND...01H1P7	GND...01H1P7
67:	ELIX0K$$$P	BUS-2V01H2P5	E0PC0MPLETEP	FD0PTIMAL$$P
68:	GND...(H3P3)	FDEXTENSYL1P	GND...01H3P7	GND...01H3P7
69:	FDEXTENSYL0P	BUS-2V01H4P5	FLIPXMFPAGEP	.....
70:	GND...(H5P3)	.....	GND...01H5P7	GND...01H5P7
71:	.....	BUS-2V01H6P5	.....	.....
72:	GND...(H7P3)	.....	GND...01H7P7	GND...01H7P7
73:	.....	BUS-2V01H8P5	.....	.....
74:	GND...(H9P3)	.....	GND...01H9P7	GND...01H9P7
75:	.....	BUS-2V01I0P5	.....	.....
76:	GND...(I1P3)	.....	GND...01I1P7	GND...01I1P7
77:	FBUS1PAG$-1P	BUS-2V01I2P5	FBUS1PAG$-0P	FQDAWINTFULP
78:	GND...(I3P3)	FBUS1CMDPARP	GND...01I3P7	GND...01I3P7

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

79:	FBUS1ADR$-2P	BUS-2V01I4P5	FBUS1ADR$-1P	FBUS1ADR$-0P
80:	GND...(I5P3)	FBUS1L0AD$$P	GND...01I5P7	GND...01I5P7
81:	.....	BUS-2V01I6P5	.....	FQDRQINVALDP
82:	GND...(I7P3)	.....	GND...01I7P7	GND...01I7P7
83:	.....	BUS-2V01I8P5	.....	.....
84:	GND...(I9P3)	.....	GND...01I9P7	GND...01I9P7
85:	.....	GND...01J0P5	FDINTERFRZEP	FCINTERFRZEP
86:	BUS-5V01J1P4	BUS-5V01J1P5	BUS-5V01J1P6	BUS-5V01J1P7
87:	BUS-5V01J2P4	BUS-5V01J2P5	BUS-5V01J2P6	BUS-5V01J2P7
88:	BUS-5V01J3P4	BUS-5V01J3P5	BUS-5V01J3P6	BUS-5V01J3P7

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

BACKPLANE CONNECTOR - T5P4

	A	B	C	D
01:	BUS-5V01K5P4	BUS-5V01K5P5	BUS-5V01K5P6	BUS-5V01K5P7
02:	BUS-5V01K6P4	BUS-5V01K6P5	BUS-5V01K6P6	BUS-5V01K6P7
03:	BUS-5V01K7P4	BUS-5V01K7P5	BUS-5V01K7P6	BUS-5V01K7P7
04:	FDABSGTBBS$P	GND...01K8P5	FCSLFCKPAR0P	FCEXTCMD1-0P
05:	GND...(K9P3)	FDABSEQBBS$P	GND...01K9P7	GND...01K9P7
06:	FBUS1$$$-19P	BUS-2V01L0P5	FBUS1$$$-18P	FBUS1$$$-17P
07:	GND...(L1P3)	FBUS1$$$-16P	GND...01L1P7	GND...01L1P7
08:	FQCSLFCK1-3P	BUS-2V01L2P5	FQCSLFCK1-2P	FQCSLFCK1-1P
09:	GND...(L3P3)	FQCSLFCK1-0P	GND...01L3P7	GND...01L3P7
10:	.....	BUS-2V01L4P5	.....	.....
11:	GND...(L5P3)	.....	GND...01L5P7	GND...01L5P7
12:	.....	BUS-2V01L6P5	.....	.....
13:	GND...(L7P3)	.....	GND...01L7P7	GND...01L7P7
14:	.....	BUS-2V01L8P5	.....	FCEXTCMD1-3P
15:	GND...(L9P3)	FCEXTCMD1-4P	GND...01L9P7	GND...01L9P7
16:	.....	BUS-2V01M0P5	.....	.....
17:	GND...(M1P3)	.....	GND...01M1P7	GND...01M1P7
18:	.....	BUS-2V01M2P5	.....	.....
19:	GND...(M3P3)	.....	GND...01M3P7	GND...01M3P7
20:	.....	BUS-2V01M4P5	.....	.....
21:	GND...(M5P3)	.....	GND...01M5P7	GND...01M5P7
22:	.....	BUS-2V01M6P5	.....	.....
23:	GND...(M7P3)	.....	GND...01M7P7	GND...01M7P7
24:	.....	BUS-2V01M8P5	.....	.....
25:	GND...(M9P3)	.....	GND...01M9P7	GND...01M9P7
26:	.....	BUS-2V01N0P5	.....	.....
27:	GND...(N1P3)	.....	GND...01N1P7	GND...01N1P7
28:	.....	BUS-2V01N2P5	.....	.....
29:	GND...(N3P3)	.....	GND...01N3P7	GND...01N3P7
30:	.....	BUS-2V01N4P5	.....	.....
31:	GND...(N5P3)	.....	GND...01N5P7	GND...01N5P7
32:	.....	BUS-2V01N6P5	.....	.....
33:	GND...(N7P3)	.....	GND...01N7P7	GND...01N7P7
34:	.....	BUS-2V01N8P5	.....	.....
35:	GND...(N9P3)	.....	GND...01N9P7	GND...01N9P7

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

COMPANY
CONFIDENTIAL

1993 5212

V500 FETCH MODULE

ENGINEERING DESIGN SPECIFICATION Rev. B

36:	FCINTCMD1-4P	BUS-2V0100P5	FCEXTCMD1-1P	FCEXTCMD1-2P
37:	GND...(O1P3)	FQCINCMD1-1P	GND...0101P7	GND...0101P7
38:	.....	BUS-2V0102P5	.....	.....
39:	GND...(O3P3)	.....	GND...0103P7	GND...0103P7
40:	.....	BUS-2V0104P5	.....	.....
41:	GND...(O5P3)	.....	GND...0105P7	GND...0105P7
42:	.....	BUS-2V0106P5	.....	.....
43:	GND...(O7P3)	.....	GND...0107P7	GND...0107P7
44:	.....	BUS-2V0108P5	.....	.....
45:	.....	BUS-2V01P2P5	.....	.....
46:	GND...(P3P3)	.....	GND...01P3P7	GND...01P3P7
47:	FQD$$$$AC-1P	BUS-2V01P4P5	FQD$$$$AC-0P	FQD$$$$BC-1P
48:	GND...(P5P3)	FQD$$$$BC-0P	GND...01P5P7	GND...01P5P7
49:	.....	BUS-2V01P6P5	.....	.....
50:	GND...(P7P3)	.....	GND...01P7P7	GND...01P7P7
51:	.....	BUS-2V01P8P5	.....	.....
52:	GND...(P9P3)	.....	GND...01P9P7	GND...01P9P7
53:	.....	BUS-2V01Q0P5	.....	.....
54:	GND...(Q1P3)	.....	GND...01Q1P7	GND...01Q1P7
55:	.....	BUS-2V01Q2P5	.....	.....
56:	GND...(Q3P3)	.....	GND...01Q3P7	GND...01Q3P7
57:	.....	BUS-2V01Q4P5	.....	.....
58:	GND...(Q5P3)	.....	GND...01Q5P7	GND...01Q5P7
59:	.....	BUS-2V01Q6P5	.....	.....
60:	GND...(Q7P3)	.....	GND...01Q7P7	GND...01Q7P7
61:	FQCINCMD1-2P	BUS-2V01Q8P5	FQD$$$$CC-1P	FQD$$$$CC-0P
62:	GND...(Q9P3)	FCINTCMD1-3P	GND...01Q9P7	GND...01Q9P7
63:	.....	BUS-2V01R0P5	.....	.....
64:	GND...(R1P3)	.....	GND...01R1P7	GND...01R1P7
65:	.....	BUS-2V01R2P5	.....	.....
66:	GND...(R3P3)	.....	GND...01R3P7	GND...01R3P7
67:	.....	BUS-2V01R4P5	.....	.....
68:	GND...(R5P3)	.....	GND...01R5P7	GND...01R5P7
69:	.....	BUS-2V01R6P5	.....	.....
70:	GND...(R7P3)	.....	GND...01R7P7	GND...01R7P7
71:	FCINTCMDPARP	BUS-2V01R8P5	.....	.....
72:	GND...(R9P3)	.....	GND...01R9P7	GND...01R9P7
73:	.....	BUS-2V01S0P5	.....	.....
74:	GND...(S1P3)	.....	GND...01S1P7	GND...01S1P7
75:	.....	BUS-2V01S2P5	.....	.....
76:	GND...(S3P3)	.....	GND...01S3P7	GND...01S3P7
77:	ASHIFTINF01P	BUS-2V01S4P5	XST0P$AND$$N	XST0P$0R$$$P
78:	GND...(S5P3)	XFM0DNBR0KEP	GND...01S5P7	GND...01S5P7

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

COMPANY
CONFIDENTIAL

1993 5212

V500 FETCH MODULE

ENGINEERING DESIGN SPECIFICATION Rev. B

79:	BUS-2V01S6P5	X1SHIFT0UT$P	XFM0DBR0KE$P
80: GND...(S7P3)	.....	GND...01S7P7	GND...01S7P7
81: A1CARDEN$-0P	BUS-2V01S8P5	A1CARDEN$-1P	A1CARDEN$-2P
82: GND...(S9P3)	AFM0DENABLEP	GND...01S9P7	GND...01S9P7
83: ASHIFTENF01P	BUS-2V01T0P5	ADANGER$F01P	AP0WERUP0K1N
84: GND...(T1P3)	ACLEAR$$F01P	GND...01T1P7	GND...01T1P7
85: AF0ENFETCHCP	A$0CKECL$29P	A$0CKECL$29N	GND...01T2P7
86: BUS-5V01T3P4	BUS-5V01T3P5	BUS-5V01T3P6	BUS-5V01T3P7
87: BUS-5V01T4P4	BUS-5V01T4P5	BUS-5V01T4P6	BUS-5V01T4P7
88: BUS-5V01T5P4	BUS-5V01T5P5	BUS-5V01T5P6	BUS-5V01T5P7

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

FRONTPLANE CONNECTOR - J0A0

	A	B	C	D
01:	.....	.....	GND.....A6A2	QDAUREG0-13L
02:	GND.....A7A0	GND.....A7A0	QDAUREG0-03L	GND...(A7A4)
03:	.....	FCCADR2-0...	GND.....A8A2	QDAUREG0-14L
04:	GND.....A9A0	GND.....A9A0	QDAUREG0-04L	GND...(A9A4)
05:	.....	FCCADR2-1...	GND.....B0A2	QDAUREG0-15L
06:	GND.....B1A0	GND.....B1A0	QDAUREG0-05L	GND...(B1A4)
07:	.....	FCCADR2-2...	GND.....B2A2	QDAUREG0-16L
08:	GND.....B3A0	GND.....B3A0	QDAUREG0-06L	GND...(B3A4)
09:	.....	FCCADR2-3...	GND.....B4A2	QDAUREG0-17L
10:	GND.....B5A0	GND.....B5A0	QDAUREG0-07L	GND...(B5A4)
11:	FCCWE2.....	FCCADR2-4...	GND.....B6A2	QDAUREG0-18L
12:	GND.....B7A0	GND.....B7A0	QDAUREG0-08L	GND...(B7A4)
13:	FCASRC2-0...	QDENDPAR0L..	GND.....B8A2	QDAUREG0-19L
14:	GND.....B9A0	GND.....B9A0	QDAUREG0-09L	GND...(B9A4)
15:	.....	QDAUREG0-00L	GND.....C0A2	QDAUREG0-20L
16:	GND.....C1A0	GND.....C1A0	QDAUREG0-10L	GND...(C1A4)
17:	FCASRC2-1...	QDAUREG0-01L	GND.....C2A2	QDAUREG0-21L
18:	GND.....C3A0	GND.....C3A0	QDAUREG0-11L	GND...(C3A4)
19:	QDBEGPAR0...	QDAUREG0-02L	GND.....C4A2	QDAUREG0-22L
20:	GND.....C5A0	GND.....C5A0	QDAUREG0-12L	GND...(C5A4)
21:	.....	YDBEGAD-06L.	GND.....C6A2	QDAUREG0-23L
22:	GND.....C7A0	GND.....C7A0	YDBEGAD-01L.	GND...(C7A4)
23:	YDBEGAD-17L.	YDBEGAD-07L.	GND.....C8A2	QDEND0ADR24L
24:	GND.....C9A0	GND.....C9A0	YDBEGAD-02L.	GND...(C9A4)
25:	YDBEGAD-18L.	YDBEGAD-08L.	GND.....D0A2	YDBEGAD-00L.
26:	GND.....D1A0	GND.....D1A0	YDBEGAD-03L.	GND...(D1A4)
27:	.....	YDBEGAD-09L.	GND.....D2A2	FCBADR6PAR1.
28:	GND.....D3A0	GND.....D3A0	YDBEGAD-04L.	GND...(D3A4)
29:	YDBEGAD-19L.	YDBEGAD-10L.	GND.....D4A2	FCBADR6PAR0.
30:	GND.....D5A0	GND.....D5A0	YDBEGAD-05L.	GND...(D5A4)
31:	YDBEGAD-20L.	YDBEGAD-11L.	GND.....D6A2	FCBADR3-5...
32:	GND.....D7A0	GND.....D7A0	FCAADR2-0...	GND...(D7A4)
33:	.....	YDBEGAD-12L.	GND.....D8A2	FCBADR2-0...
34:	GND.....D9A0	GND.....D9A0	FCAADR2-1...	GND...(D9A4)
35:	YDBEGAD-21L.	YDBEGAD-13L.	GND.....E0A2	FCBADR2-1...

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

36:	GND.....E1A0	GND.....E1A0	FCAADR2-2...	GND...(E1A4)
37:	YDBEGAD-22L.	YDBEGAD-14L.	GND.....E2A2	FCBADR2-2...
38:	GND.....E3A0	GND.....E3A0	FCAADR2-3...	GND...(E3A4)
39:	.....	YDBEGAD-15L.	GND.....E4A2	FCBADR2-3...
40:	GND.....E5A0	GND.....E5A0	FCAADR2-4...	GND...(E5A4)
41:	YDBEGAD-23L.	YDBEGAD-16L.	GND.....E6A2	FCBADR2-4...
42:	FCABSRCPRTY0	QCCADR3-2...	GND.....F0A2	.....
43:	GND.....F1A0	GND.....F1A0	QCASRC3-0...	GND...(F1A4)
44:	.....	QCCADR3-3...	GND.....F2A2	FDPTREG0-0..
45:	GND.....F3A0	GND.....F3A0	QCASRC3-1...	GND...(F3A4)
46:	QCFBUS1WE1..	QCCADR3-4...	GND.....F4A2	FDPTREG0-1..
47:	GND.....F5A0	GND.....F5A0	FD0DDPC.....	GND...(F5A4)
48:	QCFBUS2WE1..	QCBSRC3-0...	GND.....F6A2	FDPTREG0-2..
49:	GND.....F7A0	GND.....F7A0	FDIFUNDIG0..	GND...(F7A4)
50:	.....	QCBSRC3-1...	GND.....F8A2	FDPTREG0-3..
51:	GND.....F9A0	GND.....F9A0	FDIFLIMERR0.	GND...(F9A4)
52:	QCFB2SRC1...	FCBSRC2-0...	GND.....G0A2	FDPTREG0-4..
53:	GND.....G1A0	GND.....G1A0	FD0FUNDIG0..	GND...(G1A4)
54:	FCCMBINEPAR0	FCBSRC2-1...	GND.....G2A2	QCAADR3-0...
55:	GND.....G3A0	GND.....G3A0	FD0FLIMERR0.	GND...(G3A4)
56:	.....	QCCWE3.....	GND.....G4A2	QCAADR3-1...
57:	GND.....G5A0	GND.....G5A0	FDUNDABADER0	GND...(G5A4)
58:	QCDATHIT-0..	QCALUCMD1-0.	GND.....G6A2	QCAADR3-2...
59:	GND.....G7A0	GND.....G7A0	FDUNDIGAB0..	GND...(G7A4)
60:	QCDATHIT-1..	QCALUCMD1-1.	GND.....G8A2	QCAADR3-3...
61:	GND.....G9A0	GND.....G9A0	FD0FADRERR0.	GND...(G9A4)
62:	.....	QCALUCMD1-2.	GND.....H0A2	QCAADR3-4...
63:	GND.....H1A0	GND.....H1A0	FCCADRPAR0..	GND...(H1A4)
64:	FCCSRCPRTY0.	QCALUCMD1-3.	GND.....H2A2	FCBADRPAR0..
65:	GND.....H3A0	GND.....H3A0	QCCADR3-0...	GND...(H3A4)
66:	.....	FCALUCMDPAR0	GND.....H4A2	QCCADR3-1...
67:	GND.....H5A0	GND.....H5A0	FDALUEXCPT0.	GND...(H5A4)
68:	.....	QCCWE3.....	GND.....H6A2	QCCADR3-2...
69:	GND.....H7A0	GND.....H7A0	QCASRC3-0...	GND...(H7A4)
70:	.....	QCBSRC3-0...	GND.....H8A2	QCCADR3-3...
71:	GND.....H9A0	GND.....H9A0	QCASRC3-1...	GND...(H9A4)
72:	QCBEGADRSYNC	QCBSRC3-1...	GND.....I0A2	QCCADR3-4...
73:	GND.....I1A0	GND.....I1A0	QCBADR3-0...	GND...(I1A4)
74:	.....	QCAADR3-0...	GND.....I2A2	QCBADR3-0...
75:	GND.....I3A0	GND.....I3A0	QCBADR3-1...	GND...(I3A4)
76:	QDBBUSUNDIG.	QCAADR3-1...	GND.....I4A2	QCBADR3-1...
77:	GND.....I5A0	GND.....I5A0	QCBADR3-2...	GND...(I5A4)
78:	FDAUREGEQ0..	QCAADR3-2...	GND.....I6A2	QCBADR3-2...

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

79:	GND.....I7A0	GND.....I7A0	QCBADR3-3...	GND...(I7A4)
80:	.....	QCAADR3-3...	GND.....I8A2	QCBADR3-3...
81:	GND.....I9A0	GND.....I9A0	QCBADR3-4...	GND...(I9A4)
82:	FDALUEXCPT1.	QCAADR3-4...	GND.....J0A2	QCBADR3-4...
83:	.....	.....	.....	.....
84:	.....	.....	.....	.....
85:	.....	.....	.....	.....
86:	.....	.....	.....	.....
87:	.....	.....	.....	.....
88:	.....	.....	.....	.....

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

FRONTPLANE CONNECTOR - T2A0

	A	B	C	D
01:	QCASRC1-0...	QCCADR1-0...	GND.....K8A2	FCPTEST1-0..
02:	GND.....K9A0	GND.....K9A0	FCPTEST3-0..	GND...(K9A4)
03:	.....	QCCADR1-1...	GND.....L0A2	FCPTEST1-1..
04:	GND.....L1A0	GND.....L1A0	FCPTEST3-1..	GND...(L1A4)
05:	QCASRC1-1...	QCCADR1-2...	GND.....L2A2	FCPTEST1-2..
06:	GND.....L3A0	GND.....L3A0	FCPTEST3-2..	GND...(L3A4)
07:	QCCWE1.....	QCCADR1-3...	GND.....L4A2	FCPTEST1-3..
08:	GND.....L5A0	GND.....L5A0	FCPTEST3-3..	GND...(L5A4)
09:	.....	QCCADR1-4...	GND.....L6A2	QDAUREG1-14L
10:	GND.....L7A0	GND.....L7A0	QDAUREG1-04L	GND...(L7A4)
11:	QCALUCMD1-3.	QCBSRC1-0...	GND.....L8A2	QDAUREG1-15L
12:	GND.....L9A0	GND.....L9A0	QDAUREG1-05L	GND...(L9A4)
13:	QCALUCMD1-2.	QCBSRC1-1...	GND.....M0A2	QDAUREG1-16L
14:	GND.....M1A0	GND.....M1A0	QDAUREG1-06L	GND...(M1A4)
15:	.....	QCALUCMD1-1.	GND.....M2A2	QDAUREG1-17L
16:	GND.....M3A0	GND.....M3A0	QDAUREG1-07L	GND...(M3A4)
17:	QCALUCMD1-0.	QCCADR3-1...	GND.....M4A2	FDPTREG1-0..
18:	GND.....M5A0	GND.....M5A0	FDIFLIMERR1.	GND...(M5A4)
19:	QCCADR3-0...	QCAADR1-0...	GND.....M6A2	FDPTREG1-1..
20:	GND.....M7A0	GND.....M7A0	FDIFUNDIG1..	GND...(M7A4)
21:	.....	QCAADR1-1...	GND.....M8A2	FDPTREG1-2..
22:	GND.....M9A0	GND.....M9A0	FD0FLIMERR1.	GND...(M9A4)
23:	QCAADR1-3...	QCAADR1-2...	GND.....N0A2	FDPTREG1-3..
24:	GND.....N1A0	GND.....N1A0	FD0FUNDIG1..	GND...(N1A4)
25:	QCAADR1-4...	FDBBYT0-0L..	GND.....N2A2	FDPTREG1-4..
26:	GND.....N3A0	GND.....N3A0	FDUNDABADER1	GND...(N3A4)
27:	.....	FDBBYT0-1L..	GND.....N4A2	QDAUREG1-18L
28:	GND.....N5A0	GND.....N5A0	FDUNDIGAB1..	GND...(N5A4)
29:	.....	FDBBYT0-2L..	GND.....N6A2	QDAUREG1-19L
30:	GND.....N7A0	GND.....N7A0	QDAUREG1-08L	GND...(N7A4)
31:	.....	FDENDEQ0L...	GND.....N8A2	QDAUREG1-20L
32:	GND.....N9A0	GND.....N9A0	QDAUREG1-09L	GND...(N9A4)
33:	.....	QDENDPAR1L..	GND.....O0A2	FCAADRPAR1..
34:	GND.....O1A0	GND.....O1A0	QDAUREG1-10L	GND...(O1A4)
35:	.....	QDAUREG1-00L	GND.....O2A2	FCAADRPAR0..

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

COMPANY
CONFIDENTIAL

1993 5212

V500 FETCH MODULE

ENGINEERING DESIGN SPECIFICATION Rev. B

36:	GND.....O3A0	GND.....O3A0	QDAUREG1-11L	GND...(O3A4)
37:	QDAUREG1-22L	QDAUREG1-01L	GND.....O4A2	QDAUREG1-21L
38:	GND.....O5A0	GND.....O5A0	QDAUREG1-12L	GND...(O5A4)
39:	.....	QDAUREG1-02L	GND.....O6A2	QDAUREG1-23L
40:	GND.....O7A0	GND.....O7A0	QDAUREG1-13L	GND...(O7A4)
41:	QDEND1ADR24L	QDAUREG1-03L	GND.....O8A2	QDBEGPAR1...
42:	.....	QDBEGAD-14L.	GND.....P2A2	FC0FSH0VE...
43:	GND.....P3A0	GND.....P3A0	QDBEGAD-09L.	GND...(P3A4)
44:	.....	QDBEGAD-15L.	GND.....P4A2	QDBEGAD-08L.
45:	GND.....P5A0	GND.....P5A0	QDBEGAD-10L.	GND...(P5A4)
46:	.....	QDBEGAD-16L.	GND.....P6A2	FCBADR1-5...
47:	GND.....P7A0	GND.....P7A0	QDBEGAD-11L.	GND...(P7A4)
48:	.....	QDBEGAD-17L.	GND.....P8A2	FCBADR2-5...
49:	GND.....P9A0	GND.....P9A0	QDBEGAD-12L.	GND...(P9A4)
50:	.....	QDBEGAD-18L.	GND.....Q0A2	QCBADR1-0...
51:	GND.....Q1A0	GND.....Q1A0	QDBEGAD-13L.	GND...(Q1A4)
52:	.....	QDBEGAD-19L.	GND.....Q2A2	QCBADR1-1...
53:	GND.....Q3A0	GND.....Q3A0	.....	GND...(Q3A4)
54:	.....	QDBEGAD-20L.	GND.....Q4A2	QCBADR1-2...
55:	GND.....Q5A0	GND.....Q5A0	.....	GND...(Q5A4)
56:	.....	QDBEGAD-21L.	GND.....Q6A2	QCBADR1-3...
57:	GND.....Q7A0	GND.....Q7A0	.....	GND...(Q7A4)
58:	.....	QDBEGAD-22L.	GND.....Q8A2	QCBADR1-4...
59:	GND.....Q9A0	GND.....Q9A0	.....	GND...(Q9A4)
60:	.....	QDBEGAD-23L.	GND.....R0A2	.....
61:	GND.....R1A0	GND.....R1A0	.....	GND...(R1A4)
62:	.....	.....	GND.....R2A2	QCCWE1.....
63:	GND.....R3A0	GND.....R3A0	QCBSRC1-0...	GND...(R3A4)
64:	.....	.....	GND.....R4A2	QCAADR1-0...
65:	GND.....R5A0	GND.....R5A0	QCBSRC1-1...	GND...(R5A4)
66:	.....	.....	GND.....R6A2	QCAADR1-1...
67:	GND.....R7A0	GND.....R7A0	QDBEGAD-00L.	GND...(R7A4)
68:	.....	.....	GND.....R8A2	QCAADR1-2...
69:	GND.....R9A0	GND.....R9A0	QDBEGAD-01L.	GND...(R9A4)
70:	.....	QCASRC1-0...	GND.....S0A2	QCAADR1-3...
71:	GND.....S1A0	GND.....S1A0	QDBEGAD-02L.	GND...(S1A4)
72:	.....	QCASRC1-1...	GND.....S2A2	QCAADR1-4...
73:	GND.....S3A0	GND.....S3A0	QDBEGAD-03L.	GND...(S3A4)
74:	.....	QCCADR1-0...	GND.....S4A2	QCBADR1-0...
75:	GND.....S5A0	GND.....S5A0	QDBEGAD-04L.	GND...(S5A4)
76:	.....	QCCADR1-1...	GND.....S6A2	QCBADR1-1...
77:	GND.....S7A0	GND.....S7A0	QDBEGAD-05L.	GND...(S7A4)
78:	.....	QCCADR1-2...	GND.....S8A2	QCBADR1-2...

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

79:	GND.....S9A0	GND.....S9A0	QDBEGAD-06L.	GND...(S9A4)
80:	.....	QCCADR1-3...	GND.....T0A2	QCBADR1-3...
81:	GND.....T1A0	GND.....T1A0	QDBEGAD-07L.	GND...(T1A4)
82:	.....	QCCADR1-4...	GND.....T2A2	QCBADR1-4...
83:	.....	.....	.....	.....
84:	.....	.....	.....	.....
85:	.....	.....	.....	.....
86:	.....	.....	.....	.....
87:	.....	.....	.....	.....
88:	.....	.....	.....	.....

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

BACKPLANE CONNECTOR - J3P4

	A	B	C	D
01:	BUS-5VF1A3P4	BUS-5VF1A3P5	BUS-5VF1A3P6	BUS-5VF1A3P7
02:	BUS-5VF1A4P4	BUS-5VF1A4P5	BUS-5VF1A4P6	BUS-5VF1A4P7
03:	BUS-5VF1A5P4	BUS-5VF1A5P5	BUS-5VF1A5P6	BUS-5VF1A5P7
04:	.....	GND...F1A6P5	.....	.....
05:	GND...(A7P3)	.....	GND...F1A7P7	GND...F1A7P7
06:	MRBUS$$$-39P	BUS-2VF1A8P5	MRBUS$$$-38P	MRBUS$$$-37P
07:	GND...(A9P3)	MRBUS$$$-36P	GND...F1A9P7	GND...F1A9P7
08:	XADRBUS$-39P	BUS-2VF1B0P5	XADRBUS$-38P	XADRBUS$-37P
09:	GND...(B1P3)	XADRBUS$-36P	GND...F1B1P7	GND...F1B1P7
10:	FBUS1$$$-39P	BUS-2VF1B2P5	FBUS1$$$-38P	FBUS1$$$-37P
11:	GND...(B3P3)	FBUS1$$$-36P	GND...F1B3P7	GND...F1B3P7
12:	.....	BUS-2VF1B4P5	.....	.....
13:	GND...(B5P3)	.....	GND...F1B5P7	GND...F1B5P7
14:	.....	BUS-2VF1B6P5	.....	.....
15:	GND...(B7P3)	.....	GND...F1B7P7	GND...F1B7P7
16:	MRBUSPARITYP	BUS-2VF1B8P5	XADRBUSPRTYP	FBUS1PRTY40P
17:	GND...(B9P3)	XWBUSPARITYP	GND...F1B9P7	GND...F1B9P7
18:	XADRBUS$-35P	BUS-2VF1C0P5	XADRBUS$-34P	XADRBUS$-33P
19:	GND...(C1P3)	XADRBUS$-32P	GND...F1C1P7	GND...F1C1P7
20:	FBUS1$$$-35P	BUS-2VF1C2P5	FBUS1$$$-34P	FBUS1$$$-33P
21:	GND...(C3P3)	FBUS1$$$-32P	GND...F1C3P7	GND...F1C3P7
22:	MRBUS$$$-35P	BUS-2VF1C4P5	MRBUS$$$-34P	MRBUS$$$-33P
23:	GND...(C5P3)	MRBUS$$$-32P	GND...F1C5P7	GND...F1C5P7
24:	FCEXTCMDPARP	BUS-2VF1C6P5	FDAFGTBF$$$P	FDAFEQBF$$$P
25:	GND...(C7P3)	FCCS0URCE-4P	GND...F1C7P7	GND...F1C7P7
26:	.....	BUS-2VF1C8P5	.....	.....
27:	GND...(C9P3)	.....	GND...F1C9P7	GND...F1C9P7
28:	XADRBUS$-31P	BUS-2VF1D0P5	XADRBUS$-30P	XADRBUS$-29P
29:	GND...(D1P3)	XADRBUS$-28P	GND...F1D1P7	GND...F1D1P7
30:	FBUS1$$$-31P	BUS-2VF1D2P5	FBUS1$$$-30P	FBUS1$$$-29P
31:	GND...(D3P3)	FBUS1$$$-28P	GND...F1D3P7	GND...F1D3P7
32:	FCCS0URCE-3P	BUS-2VF1D4P5	FCCS0URCE-2P	FCCS0URCE-1P
33:	GND...(D5P3)	FCCS0URCE-0P	GND...F1D5P7	GND...F1D5P7
34:	MRBUS$$$-31P	BUS-2VF1D6P5	MRBUS$$$-30P	MRBUS$$$-29P
35:	GND...(D7P3)	MRBUS$$$-28P	GND...F1D7P7	GND...F1D7P7

36:	.....	BUS-2VF1D8P5	.....	.....
37:	GND...(D9P3)	.....	GND...F1D9P7	GND...F1D9P7
38:	XADRBUS$-27P	BUS-2VF1E0P5	XADRBUS$-26P	XADRBUS$-25P
39:	GND...(E1P3)	XADRBUS$-24P	GND...F1E1P7	GND...F1E1P7
40:	FBUS1$$$-27P	BUS-2VF1E2P5	FBUS1$$$-26P	FBUS1$$$-25P
41:	GND...(E3P3)	FBUS1$$$-24P	GND...F1E3P7	GND...F1E3P7
42:	FBUS1$$$-47P	BUS-2VF1E4P5	FBUS1$$$-46P	FBUS1$$$-45P
43:	GND...(E5P3)	FBUS1$$$-44P	GND...F1E5P7	GND...F1E5P7
44:	.....	BUS-2VF1E6P5	.....	.....
45:	MRBUS$$$-27P	BUS-2VF1F0P5	MRBUS$$$-26P	MRBUS$$$-25P
46:	GND...(F1P3)	MRBUS$$$-24P	GND...F1F1P7	GND...F1F1P7
47:	.....	BUS-2VF1F2P5	.....	.....
48:	GND...(F3P3)	.....	GND...F1F3P7	GND...F1F3P7
49:	XADRBUS$-23P	BUS-2VF1F4P5	XADRBUS$-22P	XADRBUS$-21P
50:	GND...(F5P3)	XADRBUS$-20P	GND...F1F5P7	GND...F1F5P7
51:	FBUS1$$$-23P	BUS-2VF1F6P5	FBUS1$$$-22P	FBUS1$$$-21P
52:	GND...(F7P3)	FBUS1$$$-20P	GND...F1F7P7	GND...F1F7P7
53:	FBUS1$$$-43P	BUS-2VF1F8P5	FBUS1$$$-42P	FBUS1$$$-41P
54:	GND...(F9P3)	FBUS1$$$-40P	GND...F1F9P7	GND...F1F9P7
55:	MRBUS$$$-23P	BUS-2VF1G0P5	MRBUS$$$-22P	MRBUS$$$-21P
56:	GND...(G1P3)	MRBUS$$$-20P	GND...F1G1P7	GND...F1G1P7
57:	FBUS1$$$-52P	BUS-2VF1G2P5	FBUS1$$$-51P	FBUS1$$$-50P
58:	GND...(G3P3)	FBUS1$$$-49P	GND...F1G3P7	GND...F1G3P7
59:	FBUS1$$$-48P	BUS-2VF1G4P5	FIPARXDIG$$P	FIPARFEDIG$P
60:	GND...(G5P3)	FIPAR0PAFBFP	GND...F1G5P7	GND...F1G5P7
61:	.....	BUS-2VF1G6P5	.....	FQCFETFLSH1P
62:	GND...(G7P3)	FYC0FSH0VE$P	GND...F1G7P7	GND...F1G7P7
63:	FSH0VE$$$$$P	BUS-2VF1G8P5	FDATAHIT$$$P	F0PTIMAL$$$P
64:	GND...(G9P3)	FQCINCMD1-0P	GND...F1G9P7	GND...F1G9P7
65:	F0$FBUS$REQP	BUS-2VF1H0P5	ESYSFLUSHP	EFETFLUSHP
66:	GND...(H1P3)	FIFFLUSH$P	GND...F1H1P7	GND...F1H1P7
67:	.....	BUS-2VF1H2P5	.....	FD0PTIMAL$$P
68:	GND...(H3P3)	FDEXTENSYL1P	GND...F1H3P7	GND...F1H3P7
69:	FDEXTENSYL0P	BUS-2VF1H4P5	.....	MRBUSVALID$P
70:	GND...(H5P3)	MRBUSERR$-4P	GND...F1H5P7	GND...F1H5P7
71:	MRBUSERR$-3P	BUS-2VF1H6P5	MRBUSERR$-2P	MRBUSERR$-1P
72:	GND...(H7P3)	MRBUSERR$-0P	GND...F1H7P7	GND...F1H7P7
73:	MRBUSTAG$-6P	BUS-2VF1H8P5	MRBUSTAG$-5P	MRBUSTAG$-4P
74:	GND...(H9P3)	MRBUSTAG$-3P	GND...F1H9P7	GND...F1H9P7
75:	MRBUSTAG$-2P	BUS-2VF1I0P5	MRBUSTAG$-1P	MRBUSTAG$-0P
76:	GND...(I1P3)	MRBUSTAGPARP	GND...F1I1P7	GND...F1I1P7
77:	FBUS1PAG$-1P	BUS-2VF1I2P5	FBUS1PAG$-0P	FQDAWINTFULP
78:	GND...(I3P3)	FBUS1CMDPARP	GND...F1I3P7	GND...F1I3P7

79: FBUS1ADR$-2P	BUS-2VF1I4P5	FBUS1ADR$-1P	FBUS1ADR$-0P
80: GND...(I5P3)	FBUS1L0AD$$P	GND...F1I5P7	GND...F1I5P7
81: FBUS2PAG$-1P	BUS-2VF1I6P5	FBUS2PAG$-0P	FQDRQINVALDP
82: GND...(I7P3)	FBUS2CMDPARP	GND...F1I7P7	GND...F1I7P7
83: FBUS2ADR$-2P	BUS-2VF1I8P5	FBUS2ADR$-1P	FBUS2ADR$-0P
84: GND...(I9P3)	FBUS2L0AD$$P	GND...F1I9P7	GND...F1I9P7
85: MRBUSTATPARP	GND...F1J0P5	FDINTERFRZEP	FCINTERFRZEP
86: BUS-5VF1J1P4	BUS-5VF1J1P5	BUS-5VF1J1P6	BUS-5VF1J1P7
87: BUS-5VF1J2P4	BUS-5VF1J2P5	BUS-5VF1J2P6	BUS-5VF1J2P7
88: BUS-5VF1J3P4	BUS-5VF1J3P5	BUS-5VF1J3P6	BUS-5VF1J3P7

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

BACKPLANE CONNECTOR - T5P4

	A	B	C	D
01:	BUS-5VF1K5P4	BUS-5VF1K5P5	BUS-5VF1K5P6	BUS-5VF1K5P7
02:	BUS-5VF1K6P4	BUS-5VF1K6P5	BUS-5VF1K6P6	BUS-5VF1K6P7
03:	BUS-5VF1K7P4	BUS-5VF1K7P5	BUS-5VF1K7P6	BUS-5VF1K7P7
04:	FDABSGTBBS$P	GND...F1K8P5	FCSLFCKPAR0P	FCEXTCMD1-0P
05:	GND...(K9P3)	FDABSEQBBS$P	GND...F1K9P7	GND...F1K9P7
06:	FBUS1$$$-19P	BUS-2VF1L0P5	FBUS1$$$-18P	FBUS1$$$-17P
07:	GND...(L1P3)	FBUS1$$$-16P	GND...F1L1P7	GND...F1L1P7
08:	FQCSLFCK1-3P	BUS-2VF1L2P5	FQCSLFCK1-2P	FQCSLFCK1-1P
09:	GND...(L3P3)	FQCSLFCK1-0P	GND...F1L3P7	GND...F1L3P7
10:	XWBUSLEN$-3P	BUS-2VF1L4P5	XWBUSLEN$-2P	XWBUSLEN$-1P
11:	GND...(L5P3)	XWBUSLEN$-0P	GND...F1L5P7	GND...F1L5P7
12:	XWBUSCMD$-4P	BUS-2VF1L6P5	XWBUSCMD$-3P	XWBUSCMD$-2P
13:	GND...(L7P3)	XWBUSCMD$-1P	GND...F1L7P7	GND...F1L7P7
14:	XWBUSCMD$-0P	BUS-2VF1L8P5	XWBUSCTLPARP	FCEXTCMD1-3P
15:	GND...(L9P3)	FCEXTCMD1-4P	GND...F1L9P7	GND...F1L9P7
16:	MQUEUEFULL$P	BUS-2VF1M0P5	XWBSRQID$-2P	XWBSRQID$-1P
17:	GND...(M1P3)	XWBSRQID$-0P	GND...F1M1P7	GND...F1M1P7
18:	EWBUSREQXMDP	BUS-2VF1M2P5	EWBUSREQXMXP	.....
19:	GND...(M3P3)	FWBUSREQ0F$P	GND...F1M3P7	GND...F1M3P7
20:	XWBUSTAG$-6P	BUS-2VF1M4P5	XWBUSTAG$-5P	XWBUSTAG$-4P
21:	GND...(M5P3)	XWBUSTAG$-3P	GND...F1M5P7	GND...F1M5P7
22:	XWBUSTAG$-2P	BUS-2VF1M6P5	XWBUSTAG$-1P	XWBUSTAG$-0P
23:	GND...(M7P3)	XWBUSTAGPARP	GND...F1M7P7	GND...F1M7P7
24:	.....	BUS-2VF1M8P5	.....	.....
25:	GND...(M9P3)	.....	GND...F1M9P7	GND...F1M9P7
26:	XADRBUS$-19P	BUS-2VF1N0P5	XADRBUS$-18P	XADRBUS$-17P
27:	GND...(N1P3)	XADRBUS$-16P	GND...F1N1P7	GND...F1N1P7
28:	MRBUS$$$-19P	BUS-2VF1N2P5	MRBUS$$$-18P	MRBUS$$$-17P
29:	GND...(N3P3)	MRBUS$$$-16P	GND...F1N3P7	GND...F1N3P7
30:	.....	BUS-2VF1N4P5	.....	.....
31:	GND...(N5P3)	.....	GND...F1N5P7	GND...F1N5P7
32:	XADRBUS$-15P	BUS-2VF1N6P5	XADRBUS$-14P	XADRBUS$-13P
33:	GND...(N7P3)	XADRBUS$-12P	GND...F1N7P7	GND...F1N7P7
34:	FBUS1$$$-15P	BUS-2VF1N8P5	FBUS1$$$-14P	FBUS1$$$-13P
35:	GND...(N9P3)	FBUS1$$$-12P	GND...F1N9P7	GND...F1N9P7

36:	FCINTCMD1-4P	BUS-2VF100P5	FCEXTCMD1-1P	FCEXTCMD1-2P
37:	GND...(O1P3)	FQCINCMD1-1P	GND...F101P7	GND...F101P7
38:	MRBUS$$$-15P	BUS-2VF102P5	MRBUS$$$-14P	MRBUS$$$-13P
39:	GND...(O3P3)	MRBUS$$$-12P	GND...F103P7	GND...F103P7
40:	.....	BUS-2VF104P5	.....	.....
41:	GND...(O5P3)	.....	GND...F105P7	GND...F105P7
42:	XADRBUS$-11P	BUS-2VF106P5	XADRBUS$-10P	XADRBUS$-09P
43:	GND...(O7P3)	XADRBUS$-08P	GND...F107P7	GND...F107P7
44:	FBUS2PARITYP	BUS-2VF108P5	FBUS2$$$$-9P	FBUS2$$$$-8P
45:	FBUS1$$$-11P	BUS-2VF1P2P5	FBUS1$$$-10P	FBUS1$$$-09P
46:	GND...(P3P3)	FBUS1$$$-08P	GND...F1P3P7	GND...F1P3P7
47:	FQD$$$$AC-1P	BUS-2VF1P4P5	FQD$$$$AC-0P	FQD$$$$BC-1P
48:	GND...(P5P3)	FQD$$$$BC-0P	GND...F1P5P7	GND...F1P5P7
49:	MRBUS$$$-11P	BUS-2VF1P6P5	MRBUS$$$-10P	MRBUS$$$-09P
50:	GND...(P7P3)	MRBUS$$$-08P	GND...F1P7P7	GND...F1P7P7
51:	.....	BUS-2VF1P8P5	.....	.....
52:	GND...(P9P3)	.....	GND...F1P9P7	GND...F1P9P7
53:	XADRBUS$-07P	BUS-2VF1Q0P5	XADRBUS$-06P	XADRBUS$-05P
54:	GND...(Q1P3)	XADRBUS$-04P	GND...F1Q1P7	GND...F1Q1P7
55:	FBUS1$$$-07P	BUS-2VF1Q2P5	FBUS1$$$-06P	FBUS1$$$-05P
56:	GND...(Q3P3)	FBUS1$$$-04P	GND...F1Q3P7	GND...F1Q3P7
57:	FBUS2$$$$-7P	BUS-2VF1Q4P5	FBUS2$$$$-6P	FBUS2$$$$-5P
58:	GND...(Q5P3)	FBUS2$$$$-4P	GND...F1Q5P7	GND...F1Q5P7
59:	MRBUS$$$-07P	BUS-2VF1Q6P5	MRBUS$$$-06P	MRBUS$$$-05P
60:	GND...(Q7P3)	MRBUS$$$-04P	GND...F1Q7P7	GND...F1Q7P7
61:	FQCINCMD1-2P	BUS-2VF1Q8P5	FQD$$$$CC-1P	FQD$$$$CC-0P
62:	GND...(Q9P3)	FCINTCMD1-3P	GND...F1Q9P7	GND...F1Q9P7
63:	.....	BUS-2VF1R0P5	.....	.....
64:	GND...(R1P3)	.....	GND...F1R1P7	GND...F1R1P7
65:	XADRBUS$-03P	BUS-2VF1R2P5	XADRBUS$-02P	XADRBUS$-01P
66:	GND...(R3P3)	XADRBUS$-00P	GND...F1R3P7	GND...F1R3P7
67:	FBUS1$$$-03P	BUS-2VF1R4P5	FBUS1$$$-02P	FBUS1$$$-01P
68:	GND...(R5P3)	FBUS1$$$-00P	GND...F1R5P7	GND...F1R5P7
69:	FBUS2$$$$-3P	BUS-2VF1R6P5	FBUS2$$$$-2P	FBUS2$$$$-1P
70:	GND...(R7P3)	FBUS2$$$$-0P	GND...F1R7P7	GND...F1R7P7
71:	FCINTCMDPARP	BUS-2VF1R8P5	.....	.....
72:	GND...(R9P3)	.....	GND...F1R9P7	GND...F1R9P7
73:	MRBUS$$$-03P	BUS-2VF1S0P5	MRBUS$$$-02P	MRBUS$$$-01P
74:	GND...(S1P3)	MRBUS$$$-00P	GND...F1S1P7	GND...F1S1P7
75:	.....	BUS-2VF1S2P5	.....	.....
76:	GND...(S3P3)	.....	GND...F1S3P7	GND...F1S3P7
77:	ASHIFTINF01P	BUS-2VF1S4P5	XST0P$AND$$N	XST0P$0R$$$P
78:	GND...(S5P3)	XFM0DNBR0KEP	GND...F1S5P7	GND...F1S5P7

79:	BUS-2VF1S6P5	X1SHIFT0UT$P	XFM0DBR0KE$P
80: GND...(S7P3)	.....	GND...F1S7P7	GND...F1S7P7
81: A1CARDEN$-0P	BUS-2VF1S8P5	A1CARDEN$-1P	A1CARDEN$-2P
82: GND...(S9P3)	AFM0DENABLEP	GND...F1S9P7	GND...F1S9P7
83: ASHIFTENF01P	BUS-2VF1T0P5	ADANGER$F01P	AP0WERUP0K1N
84: GND...(T1P3)	ACLEAR$$F01P	GND...F1T1P7	GND...F1T1P7
85: AF0ENFETCHDP	A$0CKECL$28P	A$0CKECL$28N	GND...F1T2P7
86: BUS-5VF1T3P4	BUS-5VF1T3P5	BUS-5VF1T3P6	BUS-5VF1T3P7
87: BUS-5VF1T4P4	BUS-5VF1T4P5	BUS-5VF1T4P6	BUS-5VF1T4P7
88: BUS-5VF1T5P4	BUS-5VF1T5P5	BUS-5VF1T5P6	BUS-5VF1T5P7

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

FRONTPLANE CONNECTOR - J0A0

	A	B	C	D
01:	.....	.....	GND.....A6A2	QDAUREG0-13L
02:	GND...(A7A1)	GND...(A7A1)	QDAUREG0-03L	GND...(A7A4)
03:	.....	FCCADR2-0...	GND.....A8A2	QDAUREG0-14L
04:	GND...(A9A1)	GND...(A9A1)	QDAUREG0-04L	GND...(A9A4)
05:	.....	FCCADR2-1...	GND.....B0A2	QDAUREG0-15L
06:	GND...(B1A1)	GND...(B1A1)	QDAUREG0-05L	GND...(B1A4)
07:	.....	FCCADR2-2...	GND.....B2A2	QDAUREG0-16L
08:	GND...(B3A1)	GND...(B3A1)	QDAUREG0-06L	GND...(B3A4)
09:	.....	FCCADR2-3...	GND.....B4A2	QDAUREG0-17L
10:	GND...(B5A1)	GND...(B5A1)	QDAUREG0-07L	GND...(B5A4)
11:	FCCWE2.....	FCCADR2-4...	GND.....B6A2	QDAUREG0-18L
12:	GND...(B7A1)	GND...(B7A1)	QDAUREG0-08L	GND...(B7A4)
13:	FCASRC2-0...	QDENDPAR0L..	GND.....B8A2	QDAUREG0-19L
14:	GND...(B9A1)	GND...(B9A1)	QDAUREG0-09L	GND...(B9A4)
15:	.....	QDAUREG0-00L	GND.....C0A2	QDAUREG0-20L
16:	GND...(C1A1)	GND...(C1A1)	QDAUREG0-10L	GND...(C1A4)
17:	FCASRC2-1...	QDAUREG0-01L	GND.....C2A2	QDAUREG0-21L
18:	GND...(C3A1)	GND...(C3A1)	QDAUREG0-11L	GND...(C3A4)
19:	QDBEGPAR0...	QDAUREG0-02L	GND.....C4A2	QDAUREG0-22L
20:	GND...(C5A1)	GND...(C5A1)	QDAUREG0-12L	GND...(C5A4)
21:	.....	QDBEGAD-06L.	GND.....C6A2	QDAUREG0-23L
22:	GND...(C7A1)	GND...(C7A1)	QDBEGAD-01L.	GND...(C7A4)
23:	QDBEGAD-17L.	QDBEGAD-07L.	GND.....C8A2	QDEND0ADR24L
24:	GND...(C9A1)	GND...(C9A1)	QDBEGAD-02L.	GND...(C9A4)
25:	QDBEGAD-18L.	QDBEGAD-08L.	GND.....D0A2	QDBEGAD-00L.
26:	GND...(D1A1)	GND...(D1A1)	QDBEGAD-03L.	GND...(D1A4)
27:	.....	QDBEGAD-09L.	GND.....D2A2	FCBADR6PAR1.
28:	GND...(D3A1)	GND...(D3A1)	QDBEGAD-04L.	GND...(D3A4)
29:	QDBEGAD-19L.	QDBEGAD-10L.	GND.....D4A2	FCBADR6PAR0.
30:	GND...(D5A1)	GND...(D5A1)	QDBEGAD-05L.	GND...(D5A4)
31:	QDBEGAD-20L.	QDBEGAD-11L.	GND.....D6A2	FCBADR3-5...
32:	GND...(D7A1)	GND...(D7A1)	FCAADR2-0...	GND...(D7A4)
33:	.....	QDBEGAD-12L.	GND.....D8A2	FCBADR2-0...
34:	GND...(D9A1)	GND...(D9A1)	FCAADR2-1...	GND...(D9A4)
35:	QDBEGAD-21L.	QDBEGAD-13L.	GND.....E0A2	FCBADR2-1...

36: GND...(E1A1)	GND...(E1A1)	FCAADR2-2...	GND...(E1A4)
37: QDBEGAD-22L.	QDBEGAD-14L.	GND.....E2A2	FCBADR2-2...
38: GND...(E3A1)	GND...(E3A1)	FCAADR2-3...	GND...(E3A4)
39:	QDBEGAD-15L.	GND.....E4A2	FCBADR2-3...
40: GND...(E5A1)	GND...(E5A1)	FCAADR2-4...	GND...(E5A4)
41: QDBEGAD-23L.	QDBEGAD-16L.	GND.....E6A2	FCBADR2-4...
42: FCABSRCPRTY0	YCCADR3-2...	GND.....F0A2	.....
43: GND...(F1A1)	GND...(F1A1)	YCASRC3-0...	GND...(F1A4)
44:	YCCADR3-3...	GND.....F2A2	FDPTREG0-0..
45: GND...(F3A1)	GND...(F3A1)	YCASRC3-1...	GND...(F3A4)
46: QCFBUS1WE1..	YCCADR3-4...	GND.....F4A2	FDPTREG0-1..
47: GND...(F5A1)	GND...(F5A1)	FD0DDPC.....	GND...(F5A4)
48: QCFBUS2WE1..	YCBSRC3-0...	GND.....F6A2	FDPTREG0-2..
49: GND...(F7A1)	GND...(F7A1)	FDIFUNDIG0..	GND...(F7A4)
50:	YCBSRC3-1...	GND.....F8A2	FDPTREG0-3..
51: GND...(F9A1)	GND...(F9A1)	FDIFLIMERR0.	GND...(F9A4)
52: QCFB2SRC1...	FCBSRC2-0...	GND.....G0A2	FDPTREG0-4..
53: GND...(G1A1)	GND...(G1A1)	FD0FUNDIG0..	GND...(G1A4)
54: FCCMBINEPAR0	FCBSRC2-1...	GND.....G2A2	YCAADR3-0...
55: GND...(G3A1)	GND...(G3A1)	FD0FLIMERR0.	GND...(G3A4)
56:	YCCWE3.....	GND.....G4A2	YCAADR3-1...
57: GND...(G5A1)	GND...(G5A1)	FDUNDABADER0	GND...(G5A4)
58: QCDATHIT-0..	QCALUCMD1-0.	GND.....G6A2	YCAADR3-2...
59: GND...(G7A1)	GND...(G7A1)	FDUNDIGAB0..	GND...(G7A4)
60: QCDATHIT-1..	QCALUCMD1-1.	GND.....G8A2	YCAADR3-3...
61: GND...(G9A1)	GND...(G9A1)	FD0FADRERR0.	GND...(G9A4)
62:	QCALUCMD1-2.	GND.....H0A2	YCAADR3-4...
63: GND...(H1A1)	GND...(H1A1)	FCCADRPAR0..	GND...(H1A4)
64: FCCSRCPRTY0.	QCALUCMD1-3.	GND.....H2A4	FCBADRPAR0..
65: GND...(H3A1)	GND...(H3A1)	QCCADR3-0...	GND...(H3A4)
66:	FCALUCMDPAR0	GND.....H4A2	QCCADR3-1...
67: GND...(H5A1)	GND...(H5A1)	FDALUEXCPT0.	GND...(H5A4)
68:	QCCWE3.....	GND.....H6A2	QCCADR3-2...
69: GND...(H7A1)	GND...(H7A1)	QCASRC3-0...	GND...(H7A4)
70:	QCBSRC3-0...	GND.....H8A2	QCCADR3-3...
71: GND...(H9A1)	GND...(H9A1)	QCASRC3-1...	GND...(H9A4)
72: QCBEGADRSYNC	QCBSRC3-1...	GND.....I0A2	QCCADR3-4...
73: GND...(I1A1)	GND...(I1A1)	YCBADR3-0...	GND...(I1A4)
74:	QCAADR3-0...	GND.....I2A2	QCBADR3-0...
75: GND...(I3A1)	GND...(I3A1)	YCBADR3-1...	GND...(I3A4)
76: QDBBUSUNDIG.	QCAADR3-1...	GND.....I4A2	QCBADR3-1...
77: GND...(I5A1)	GND...(I5A1)	YCBADR3-2...	GND...(I5A4)
78: FDAUREGEQ0..	QCAADR3-2...	GND.....I6A2	QCBADR3-2...

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

```
79: GND...(I7A1)    GND...(I7A1)    YCBADR3-3...    GND...(I7A4)
80: ............    QCAADR3-3...    GND.....I8A2    QCBADR3-3...
81: GND...(I9A1)    GND...(I9A1)    YCBADR3-4...    GND...(I9A4)
82: FDALUEXCPT1.    QCAADR3-4...    GND.....J0A2    QCBADR3-4...
83: ............    ............    ............    ............
84: ............    ............    ............    ............
85: ............    ............    ............    ............
86: ............    ............    ............    ............
87: ............    ............    ............    ............
88: ............    ............    ............    ............
```

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

FRONTPLANE CONNECTOR - T2A0

	A	B	C	D
01:	YCASRC1-0...	YCCADR1-0...	GND.....K8A2	FCPTEST1-0..
02:	GND...(K9A1)	GND...(K9A1)	FCPTEST3-0..	GND...(K9A4)
03:	.....	YCCADR1-1...	GND.....L0A2	FCPTEST1-1..
04:	GND...(L1A1)	GND...(L1A1)	FCPTEST3-1..	GND...(L1A4)
05:	YCASRC1-1...	YCCADR1-2...	GND.....L2A2	FCPTEST1-2..
06:	GND...(L3A1)	GND...(L3A1)	FCPTEST3-2..	GND...(L3A4)
07:	YCCWE1.....	YCCADR1-3...	GND.....L4A2	FCPTEST1-3..
08:	GND...(L5A1)	GND...(L5A1)	FCPTEST3-3..	GND...(L5A4)
09:	.....	YCCADR1-4...	GND.....L6A2	QDAUREG1-14L
10:	GND...(L7A1)	GND...(L7A1)	QDAUREG1-04L	GND...(L7A4)
11:	YCALUCMD1-3.	YCBSRC1-0...	GND.....L8A2	QDAUREG1-15L
12:	GND...(L9A1)	GND...(L9A1)	QDAUREG1-05L	GND...(L9A4)
13:	YCALUCMD1-2.	YCBSRC1-1...	GND.....M0A2	QDAUREG1-16L
14:	GND...(M1A1)	GND...(M1A1)	QDAUREG1-06L	GND...(M1A4)
15:	.....	YCALUCMD1-1.	GND.....M2A2	QDAUREG1-17L
16:	GND...(M3A1)	GND...(M3A1)	QDAUREG1-07L	GND...(M3A4)
17:	YCALUCMD1-0.	YCCADR3-1...	GND.....M4A2	FDPTREG1-0..
18:	GND...(M5A1)	GND...(M5A1)	FDIFLIMERR1.	GND...(M5A4)
19:	YCCADR3-0...	YCAADR1-0...	GND.....M6A2	FDPTREG1-1..
20:	GND...(M7A1)	GND...(M7A1)	FDIFUNDIG1..	GND...(M7A4)
21:	.....	YCAADR1-1...	GND.....M8A2	FDPTREG1-2..
22:	GND...(M9A1)	GND...(M9A1)	FD0FLIMERR1.	GND...(M9A4)
23:	YCAADR1-3...	YCAADR1-2...	GND.....N0A2	FDPTREG1-3..
24:	GND...(N1A1)	GND...(N1A1)	FD0FUNDIG1..	GND...(N1A4)
25:	YCAADR1-4...	FDBBYT0-0L..	GND.....N2A2	FDPTREG1-4..
26:	GND...(N3A1)	GND...(N3A1)	FDUNDABADER1	GND...(N3A4)
27:	.....	FDBBYT0-1L..	GND.....N4A2	QDAUREG1-18L
28:	GND...(N5A1)	GND...(N5A1)	FDUNDIGAB1..	GND...(N5A4)
29:	.....	FDBBYT0-2L..	GND.....N6A2	QDAUREG1-19L
30:	GND...(N7A1)	GND...(N7A1)	QDAUREG1-08L	GND...(N7A4)
31:	.....	FDENDEQ0L...	GND.....N8A2	QDAUREG1-20L
32:	GND...(N9A1)	GND...(N9A1)	QDAUREG1-09L	GND...(N9A4)
33:	.....	QDENDPAR1L..	GND.....O0A2	FCAADRPAR1..
34:	GND...(O1A1)	GND...(O1A1)	QDAUREG1-10L	GND...(O1A4)
35:	.....	QDAUREG1-00L	GND.....O2A2	FCAADRPAR0..

36:	GND...(O3A1)	GND...(O3A1)	QDAUREG1-11L	GND...(O3A4)
37:	QDAUREG1-22L	QDAUREG1-01L	GND.....O4A2	QDAUREG1-21L
38:	GND...(O5A1)	GND...(O5A1)	QDAUREG1-12L	GND...(O5A4)
39:	.....	QDAUREG1-02L	GND.....O6A2	QDAUREG1-23L
40:	GND...(O7A1)	GND...(O7A1)	QDAUREG1-13L	GND...(O7A4)
41:	QDEND1ADR24L	QDAUREG1-03L	GND.....O8A2	QDBEGPAR1...
42:	.....	QDBEGAD-14L.	GND.....P2A2	FC0FSH0VE...
43:	GND...(P3A1)	GND...(P3A1)	QDBEGAD-09L.	GND...(P3A4)
44:	.....	QDBEGAD-15L.	GND.....P4A2	QDBEGAD-08L.
45:	GND...(P5A1)	GND...(P5A1)	QDBEGAD-10L.	GND...(P5A4)
46:	.....	QDBEGAD-16L.	GND.....P6A2	FCBADR1-5...
47:	GND...(P7A1)	GND...(P7A1)	QDBEGAD-11L.	GND...(P7A4)
48:	.....	QDBEGAD-17L.	GND.....P8A2	FCBADR2-5...
49:	GND...(P9A1)	GND...(P9A1)	QDBEGAD-12L.	GND...(P9A4)
50:	.....	QDBEGAD-18L.	GND.....Q0A2	YCBADR1-0...
51:	GND...(Q1A1)	GND...(Q1A1)	QDBEGAD-13L.	GND...(Q1A4)
52:	.....	QDBEGAD-19L.	GND.....Q2A2	YCBADR1-1...
53:	GND...(Q3A1)	GND...(Q3A1)	.....	GND...(Q3A4)
54:	.....	QDBEGAD-20L.	GND.....Q4A2	YCBADR1-2...
55:	GND...(Q5A1)	GND...(Q5A1)	.....	GND...(Q5A4)
56:	.....	QDBEGAD-21L.	GND.....Q6A2	YCBADR1-3...
57:	GND...(Q7A1)	GND...(Q7A1)	.....	GND...(Q7A4)
58:	.....	QDBEGAD-22L.	GND.....Q8A2	YCBADR1-4...
59:	GND...(Q9A1)	GND...(Q9A1)	.....	GND...(Q9A4)
60:	.....	QDBEGAD-23L.	GND.....R0A2	.....
61:	GND...(R1A1)	GND...(R1A1)	.....	GND...(R1A4)
62:	.....	.....	GND.....R2A2	QCCWE1.....
63:	GND...(R3A1)	GND...(R3A1)	QCBSRC1-0...	GND...(R3A4)
64:	.....	.....	GND.....R4A2	QCAADR1-0...
65:	GND...(R5A1)	GND...(R5A1)	QCBSRC1-1...	GND...(R5A4)
66:	.....	.....	GND.....R6A2	QCAADR1-1...
67:	GND...(R7A1)	GND...(R7A1)	QDBEGAD-00L.	GND...(R7A4)
68:	.....	.....	GND.....R8A2	QCAADR1-2...
69:	GND...(R9A1)	GND...(R9A1)	QDBEGAD-01L.	GND...(R9A4)
70:	.....	QCASRC1-0...	GND.....S0A2	QCAADR1-3...
71:	GND...(S1A1)	GND...(S1A1)	QDBEGAD-02L.	GND...(S1A4)
72:	.....	QCASRC1-1...	GND.....S2A2	QCAADR1-4...
73:	GND...(S3A1)	GND...(S3A1)	QDBEGAD-03L.	GND...(S3A4)
74:	.....	QCCADR1-0...	GND.....S4A2	QCBADR1-0...
75:	GND...(S5A1)	GND...(S5A1)	QDBEGAD-04L.	GND...(S5A4)
76:	.....	QCCADR1-1...	GND.....S6A2	QCBADR1-1...
77:	GND...(S7A1)	GND...(S7A1)	QDBEGAD-05L.	GND...(S7A4)
78:	.....	QCCADR1-2...	GND.....S8A2	QCBADR1-2...

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

79:	GND...(S9A1)	GND...(S9A1)	QDBEGAD-06L.	GND...(S9A4)
80:	.....	QCCADR1-3...	GND.....T0A2	QCBADR1-3...
81:	GND...(T1A1)	GND...(T1A1)	QDBEGAD-07L.	GND...(T1A4)
82:	.....	QCCADR1-4...	GND.....T2A2	QCBADR1-4...
83:	.....	.....	.....	.....
84:	.....	.....	.....	.....
85:	.....	.....	.....	.....
86:	.....	.....	.....	.....
87:	.....	.....	.....	.....
88:	.....	.....	.....	.....

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

BACKPLANE CONNECTOR - J3P4

	A	B	C	D
01:	BUS-5VW1A3P4	BUS-5VW1A3P5	BUS-5VW1A3P6	BUS-5VW1A3P7
02:	BUS-5VW1A4P4	BUS-5VW1A4P5	BUS-5VW1A4P6	BUS-5VW1A4P7
03:	BUS-5VW1A5P4	BUS-5VW1A5P5	BUS-5VW1A5P6	BUS-5VW1A5P7
04:	XWRTBUS$-39P	GND...W1A6P5	XWRTBUS$-38P	XWRTBUS$-37P
05:	GND...(A7P3)	XWRTBUS$-36P	GND...W1A7P7	GND...W1A7P7
06:	MRBUS$$$-39P	BUS-2VW1A8P5	MRBUS$$$-38P	MRBUS$$$-37P
07:	GND...(A9P3)	MRBUS$$$-36P	GND...W1A9P7	GND...W1A9P7
08:	XADRBUS$-39P	BUS-2VW1B0P5	XADRBUS$-38P	XADRBUS$-37P
09:	GND...(B1P3)	XADRBUS$-36P	GND...W1B1P7	GND...W1B1P7
10:	FBUS1$$$-39P	BUS-2VW1B2P5	FBUS1$$$-38P	FBUS1$$$-37P
11:	GND...(B3P3)	FBUS1$$$-36P	GND...W1B3P7	GND...W1B3P7
12:	.....	BUS-2VW1B4P5	.....	.....
13:	GND...(B5P3)	.....	GND...W1B5P7	GND...W1B5P7
14:	XWRTBUS$-35P	BUS-2VW1B6P5	XWRTBUS$-34P	XWRTBUS$-33P
15:	GND...(B7P3)	XWRTBUS$-32P	GND...W1B7P7	GND...W1B7P7
16:	MRBUSPARITYP	BUS-2VW1B8P5	XADRBUSPRTYP	FBUS1PRTY40P
17:	GND...(B9P3)	XWBUSPARITYP	GND...W1B9P7	GND...W1B9P7
18:	XADRBUS$-35P	BUS-2VW1C0P5	XADRBUS$-34P	XADRBUS$-33P
19:	GND...(C1P3)	XADRBUS$-32P	GND...W1C1P7	GND...W1C1P7
20:	FBUS1$$$-35P	BUS-2VW1C2P5	FBUS1$$$-34P	FBUS1$$$-33P
21:	GND...(C3P3)	FBUS1$$$-32P	GND...W1C3P7	GND...W1C3P7
22:	MRBUS$$$-35P	BUS-2VW1C4P5	MRBUS$$$-34P	MRBUS$$$-33P
23:	GND...(C5P3)	MRBUS$$$-32P	GND...W1C5P7	GND...W1C5P7
24:	.....	BUS-2VW1C6P5	.....	.....
25:	GND...(C7P3)	.....	GND...W1C7P7	GND...W1C7P7
26:	XWRTBUS$-31P	BUS-2VW1C8P5	XWRTBUS$-30P	XWRTBUS$-29P
27:	GND...(C9P3)	XWRTBUS$-28P	GND...W1C9P7	GND...W1C9P7
28:	XADRBUS$-31P	BUS-2VW1D0P5	XADRBUS$-30P	XADRBUS$-29P
29:	GND...(D1P3)	XADRBUS$-28P	GND...W1D1P7	GND...W1D1P7
30:	FBUS1$$$-31P	BUS-2VW1D2P5	FBUS1$$$-30P	FBUS1$$$-29P
31:	GND...(D3P3)	FBUS1$$$-28P	GND...W1D3P7	GND...W1D3P7
32:	.....	BUS-2VW1D4P5	.....	.....
33:	GND...(D5P3)	.....	GND...W1D5P7	GND...W1D5P7
34:	MRBUS$$$-31P	BUS-2VW1D6P5	MRBUS$$$-30P	MRBUS$$$-29P
35:	GND...(D7P3)	MRBUS$$$-28P	GND...W1D7P7	GND...W1D7P7

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

36:	XWRTBUS$-27P	BUS-2VW1D8P5	XWRTBUS$-26P	XWRTBUS$-25P
37:	GND...(D9P3)	XWRTBUS$-24P	GND...W1D9P7	GND...W1D9P7
38:	XADRBUS$-27P	BUS-2VW1E0P5	XADRBUS$-26P	XADRBUS$-25P
39:	GND...(E1P3)	XADRBUS$-24P	GND...W1E1P7	GND...W1E1P7
40:	FBUS1$$$-27P	BUS-2VW1E2P5	FBUS1$$$-26P	FBUS1$$$-25P
41:	GND...(E3P3)	FBUS1$$$-24P	GND...W1E3P7	GND...W1E3P7
42:	FBUS1$$$-47P	BUS-2VW1E4P5	FBUS1$$$-46P	FBUS1$$$-45P
43:	GND...(E5P3)	FBUS1$$$-44P	GND...W1E5P7	GND...W1E5P7
44:	FBUS10PPRTYP	BUS-2VW1E6P5	.....	.....
45:	MRBUS$$$-27P	BUS-2VW1F0P5	MRBUS$$$-26P	MRBUS$$$-25P
46:	GND...(F1P3)	MRBUS$$$-24P	GND...W1F1P7	GND...W1F1P7
47:	XWRTBUS$-23P	BUS-2VW1F2P5	XWRTBUS$-22P	XWRTBUS$-21P
48:	GND...(F3P3)	XWRTBUS$-20P	GND...W1F3P7	GND...W1F3P7
49:	XADRBUS$-23P	BUS-2VW1F4P5	XADRBUS$-22P	XADRBUS$-21P
50:	GND...(F5P3)	XADRBUS$-20P	GND...W1F5P7	GND...W1F5P7
51:	FBUS1$$$-23P	BUS-2VW1F6P5	FBUS1$$$-22P	FBUS1$$$-21P
52:	GND...(F7P3)	FBUS1$$$-20P	GND...W1F7P7	GND...W1F7P7
53:	FBUS1$$$-43P	BUS-2VW1F8P5	FBUS1$$$-42P	FBUS1$$$-41P
54:	GND...(F9P3)	FBUS1$$$-40P	GND...W1F9P7	GND...W1F9P7
55:	MRBUS$$$-23P	BUS-2VW1G0P5	MRBUS$$$-22P	MRBUS$$$-21P
56:	GND...(G1P3)	MRBUS$$$-20P	GND...W1G1P7	GND...W1G1P7
57:	FBUS1$$$-52P	BUS-2VW1G2P5	FBUS1$$$-51P	FBUS1$$$-50P
58:	GND...(G3P3)	FBUS1$$$-49P	GND...W1G3P7	GND...W1G3P7
59:	FBUS1$$$-48P	BUS-2VW1G4P5	FIPARXDIG$$P	FIPARFEDIG$P
60:	GND...(G5P3)	FIPAR0PAFBFP	GND...W1G5P7	GND...W1G5P7
61:	.....	BUS-2VW1G6P5	.....	.....
62:	GND...(G7P3)	.....	GND...W1G7P7	GND...W1G7P7
63:	FSH0VE$$$$$P	BUS-2VW1G8P5	FDATAHIT$$$P	F0PTIMAL$$$P
64:	GND...(G9P3)	E$P0P$FQUE$P	GND...W1G9P7	GND...W1G9P7
65:	F0$FBUS$REQP	BUS-2VW1H0P5	ESYSFLUSHP	EFETFLUSHP
66:	GND...(H1P3)	FIFFLUSH$P	GND...W1H1P7	GND...W1H1P7
67:	.....	BUS-2VW1H2P5	.....	.....
68:	GND...(H3P3)	.....	GND...W1H3P7	GND...W1H3P7
69:	EBRANCH0K$$P	BUS-2VW1H4P5	FLIPXMFPAGEP	MRBUSVALID$P
70:	GND...(H5P3)	MRBUSERR$-4P	GND...W1H5P7	GND...W1H5P7
71:	MRBUSERR$-3P	BUS-2VW1H6P5	MRBUSERR$-2P	MRBUSERR$-1P
72:	GND...(H7P3)	MRBUSERR$-0P	GND...W1H7P7	GND...W1H7P7
73:	MRBUSTAG$-6P	BUS-2VW1H8P5	MRBUSTAG$-5P	MRBUSTAG$-4P
74:	GND...(H9P3)	MRBUSTAG$-3P	GND...W1H9P7	GND...W1H9P7
75:	MRBUSTAG$-2P	BUS-2VW1I0P5	MRBUSTAG$-1P	MRBUSTAG$-0P
76:	GND...(I1P3)	MRBUSTAGPARP	GND...W1I1P7	GND...W1I1P7
77:	FBUS1PAG$-1P	BUS-2VW1I2P5	FBUS1PAG$-0P	.....
78:	GND...(I3P3)	FBUS1CMDPARP	GND...W1I3P7	GND...W1I3P7

UNISYS CORPORATION
GENERAL SYSTEMS GROUP
MISSION VIEJO PLANT

1993 5212

V500 FETCH MODULE

COMPANY
CONFIDENTIAL ENGINEERING DESIGN SPECIFICATION Rev. B

79:	FBUS1ADR$-2P	BUS-2VW1I4P5	FBUS1ADR$-1P	FBUS1ADR$-0P
80:	GND...(I5P3)	FBUS1L0AD$$P	GND...W1I5P7	GND...W1I5P7
81:	FBUS2PAG$-1P	BUS-2VW1I6P5	FBUS2PAG$-0P	.....
82:	GND...(I7P3)	.....	GND...W1I7P7	GND...W1I7P7
83:	FBUS2ADR$-2P	BUS-2VW1I8P5	FBUS2ADR$-1P	FBUS2ADR$-0P
84:	GND...(I9P3)	FBUS2L0AD$$P	GND...W1I9P7	GND...W1I9P7
85:	MRBUSTATPARP	GND...W1J0P5	.....	.....
86:	BUS-5VW1J1P4	BUS-5VW1J1P5	BUS-5VW1J1P6	BUS-5VW1J1P7
87:	BUS-5VW1J2P4	BUS-5VW1J2P5	BUS-5VW1J2P6	BUS-5VW1J2P7
88:	BUS-5VW1J3P4	BUS-5VW1J3P5	BUS-5VW1J3P6	BUS-5VW1J3P7

12 BACKPLANE/FRONTPLANE DEFINITION (Continued)

BACKPLANE CONNECTOR - T5P4

	A	B	C	D
01:	BUS-5VW1K5P4	BUS-5VW1K5P5	BUS-5VW1K5P6	BUS-5VW1K5P7
02:	BUS-5VW1K6P4	BUS-5VW1K6P5	BUS-5VW1K6P6	BUS-5VW1K6P7
03:	BUS-5VW1K7P4	BUS-5VW1K7P5	BUS-5VW1K7P6	BUS-5VW1K7P7
04:	.....	GND...W1K8P5	.....	.....
05:	GND...(K9P3)	.....	GND...W1K9P7	GND...W1K9P7
06:	FBUS1$$$-19P	BUS-2VW1L0P5	FBUS1$$$-18P	FBUS1$$$-17P
07:	GND...(L1P3)	FBUS1$$$-16P	GND...W1L1P7	GND...W1L1P7
08:	.....	BUS-2VW1L2P5	.....	.....
09:	GND...(L3P3)	.....	GND...W1L3P7	GND...W1L3P7
10:	XWBUSLEN$-3P	BUS-2VW1L4P5	XWBUSLEN$-2P	XWBUSLEN$-1P
11:	GND...(L5P3)	XWBUSLEN$-0P	GND...W1L5P7	GND...W1L5P7
12:	XWBUSCMD$-4P	BUS-2VW1L6P5	XWBUSCMD$-3P	XWBUSCMD$-2P
13:	GND...(L7P3)	XWBUSCMD$-1P	GND...W1L7P7	GND...W1L7P7
14:	XWBUSCMD$-0P	BUS-2VW1L8P5	XWBUSCTLPARP	.....
15:	GND...(L9P3)	.....	GND...W1L9P7	GND...W1L9P7
16:	MQUEUEFULL$P	BUS-2VW1M0P5	XWBSRQID$-2P	XWBSRQID$-1P
17:	GND...(M1P3)	XWBSRQID$-0P	GND...W1M1P7	GND...W1M1P7
18:	EWBUSREQXMDP	BUS-2VW1M2P5	EWBUSREQXMXP	FWBUSREQIF$P
19:	GND...(M3P3)	FWBUSREQ0F$P	GND...W1M3P7	GND...W1M3P7
20:	XWBUSTAG$-6P	BUS-2VW1M4P5	XWBUSTAG$-5P	XWBUSTAG$-4P
21:	GND...(M5P3)	XWBUSTAG$-3P	GND...W1M5P7	GND...W1M5P7
22:	XWBUSTAG$-2P	BUS-2VW1M6P5	XWBUSTAG$-1P	XWBUSTAG$-0P
23:	GND...(M7P3)	XWBUSTAGPARP	GND...W1M7P7	GND...W1M7P7
24:	XWRTBUS$-19P	BUS-2VW1M8P5	XWRTBUS$-18P	XWRTBUS$-17P
25:	GND...(M9P3)	XWRTBUS$-16P	GND...W1M9P7	GND...W1M9P7
26:	XADRBUS$-19P	BUS-2VW1N0P5	XADRBUS$-18P	XADRBUS$-17P
27:	GND...(N1P3)	XADRBUS$-16P	GND...W1N1P7	GND...W1N1P7
28:	MRBUS$$$-19P	BUS-2VW1N2P5	MRBUS$$$-18P	MRBUS$$$-17P
29:	GND...(N3P3)	MRBUS$$$-16P	GND...W1N3P7	GND...W1N3P7
30:	XWRTBUS$-15P	BUS-2VW1N4P5	XWRTBUS$-14P	XWRTBUS$-13P
31:	GND...(N5P3)	XWRTBUS$-12P	GND...W1N5P7	GND...W1N5P7
32:	XADRBUS$-15P	BUS-2VW1N6P5	XADRBUS$-14P	XADRBUS$-13P
33:	GND...(N7P3)	XADRBUS$-12P	GND...W1N7P7	GND...W1N7P7
34:	FBUS1$$$-15P	BUS-2VW1N8P5	FBUS1$$$-14P	FBUS1$$$-13P
35:	GND...(N9P3)	FBUS1$$$-12P	GND...W1N9P7	GND...W1N9P7

36:	.....	BUS-2VW100P5	.....	.....
37:	GND...(O1P3)	.....	GND...W101P7	GND...W101P7
38:	MRBUS$$$-15P	BUS-2VW102P5	MRBUS$$$-14P	MRBUS$$$-13P
39:	GND...(O3P3)	MRBUS$$$-12P	GND...W103P7	GND...W103P7
40:	XWRTBUS$-11P	BUS-2VW104P5	XWRTBUS$-10P	XWRTBUS$-09P
41:	GND...(O5P3)	XWRTBUS$-08P	GND...W105P7	GND...W105P7
42:	XADRBUS$-11P	BUS-2VW106P5	XADRBUS$-10P	XADRBUS$-09P
43:	GND...(O7P3)	XADRBUS$-08P	GND...W107P7	GND...W107P7
44:	FBUS2PARITYP	BUS-2VW108P5	FBUS2$$$$-9P	FBUS2$$$$-8P
45:	FBUS1$$$-11P	BUS-2VW1P2P5	FBUS1$$$-10P	FBUS1$$$-09P
46:	GND...(P3P3)	FBUS1$$$-08P	GND...W1P3P7	GND...W1P3P7
47:	.....	BUS-2VW1P4P5	.....	.....
48:	GND...(P5P3)	.....	GND...W1P5P7	GND...W1P5P7
49:	MRBUS$$$-11P	BUS-2VW1P6P5	MRBUS$$$-10P	MRBUS$$$-09P
50:	GND...(P7P3)	MRBUS$$$-08P	GND...W1P7P7	GND...W1P7P7
51:	XWRTBUS$-07P	BUS-2VW1P8P5	XWRTBUS$-06P	XWRTBUS$-05P
52:	GND...(P9P3)	XWRTBUS$-04P	GND...W1P9P7	GND...W1P9P7
53:	XADRBUS$-07P	BUS-2VW1Q0P5	XADRBUS$-06P	XADRBUS$-05P
54:	GND...(Q1P3)	XADRBUS$-04P	GND...W1Q1P7	GND...W1Q1P7
55:	FBUS1$$$-07P	BUS-2VW1Q2P5	FBUS1$$$-06P	FBUS1$$$-05P
56:	GND...(Q3P3)	FBUS1$$$-04P	GND...W1Q3P7	GND...W1Q3P7
57:	FBUS2$$$$-7P	BUS-2VW1Q4P5	FBUS2$$$$-6P	FBUS2$$$$-5P
58:	GND...(Q5P3)	FBUS2$$$$-4P	GND...W1Q5P7	GND...W1Q5P7
59:	MRBUS$$$-07P	BUS-2VW1Q6P5	MRBUS$$$-06P	MRBUS$$$-05P
60:	GND...(Q7P3)	MRBUS$$$-04P	GND...W1Q7P7	GND...W1Q7P7
61:	.....	BUS-2VW1Q8P5	.....	.....
62:	GND...(Q9P3)	.....	GND...W1Q9P7	GND...W1Q9P7
63:	XWRTBUS$-03P	BUS-2VW1R0P5	XWRTBUS$-02P	XWRTBUS$-01P
64:	GND...(R1P3)	XWRTBUS$-00P	GND...W1R1P7	GND...W1R1P7
65:	XADRBUS$-03P	BUS-2VW1R2P5	XADRBUS$-02P	XADRBUS$-01P
66:	GND...(R3P3)	XADRBUS$-00P	GND...W1R3P7	GND...W1R3P7
67:	FBUS1$$$-03P	BUS-2VW1R4P5	FBUS1$$$-02P	FBUS1$$$-01P
68:	GND...(R5P3)	FBUS1$$$-00P	GND...W1R5P7	GND...W1R5P7
69:	FBUS2$$$$-3P	BUS-2VW1R6P5	FBUS2$$$$-2P	FBUS2$$$$-1P
70:	GND...(R7P3)	FBUS2$$$$-0P	GND...W1R7P7	GND...W1R7P7
71:	.....	BUS-2VW1R8P5	.....	.....
72:	GND...(R9P3)	.....	GND...W1R9P7	GND...W1R9P7
73:	MRBUS$$$-03P	BUS-2VW1S0P5	MRBUS$$$-02P	MRBUS$$$-01P
74:	GND...(S1P3)	MRBUS$$$-00P	GND...W1S1P7	GND...W1S1P7
75:	.....	BUS-2VW1S2P5	.....	.....
76:	GND...(S3P3)	.....	GND...W1S3P7	GND...W1S3P7
77:	ASHIFTINF01P	BUS-2VW1S4P5	XST0P$AND$$N	XST0P$0R$$$P
78:	GND...(S5P3)	XFM0DNBR0KEP	GND...W1S5P7	GND...W1S5P7

79:	BUS-2VW1S6P5	X1SHIFT0UT$P	XFM0DBR0KE$P
80: GND...(S7P3)	.....	GND...W1S7P7	GND...W1S7P7
81: A1CARDEN$-0P	BUS-2VW1S8P5	A1CARDEN$-1P	A1CARDEN$-2P
82: GND...(S9P3)	AFM0DENABLEP	GND...W1S9P7	GND...W1S9P7
83: ASHIFTENF01P	BUS-2VW1T0P5	ADANGER$F01P	AP0WERUP0K1N
84: GND...(T1P3)	ACLEAR$$F01P	GND...W1T1P7	GND...W1T1P7
85: AF0ENFETCHIP	A$0CKECL$30P	A$0CKECL$30N	GND...W1T2P7
86: BUS-5VW1T3P4	BUS-5VW1T3P5	BUS-5VW1T3P6	BUS-5VW1T3P7
87: BUS-5VW1T4P4	BUS-5VW1T4P5	BUS-5VW1T4P6	BUS-5VW1T4P7
88: BUS-5VW1T5P4	BUS-5VW1T5P5	BUS-5VW1T5P6	BUS-5VW1T5P7