Burroughs Corporation

COMPUTER SYSTEMS GROUP
PASADENA PLANT

B4900 ARCHITECTURE

COMPANY CONFIDENTIAL

SYSTEMS DESIGN SPECIFICATION

REVISIONS

REV. LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A	8/30/82	Initial Issue ECN 46775	L. Simpson M. Rooke	P. Lamson N. Ruddick S. Davis J. Swensson B. Gaither R. Cole J. Wise
B	6/16/83	General revision to incorporate all hardware changes made since the A revision (REE 87162).	L. Simpson M. Rooke	P. Lamson N. Ruddick J. Swensson R. Cole

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

1 SCOPE

This document is intended to cover topics that relate to the system in general and/or subjects that encompass more than one module. Its purpose is to represent an accurate and consistent picture of the system's hardware architecture.

2 RELATED DOCUMENTS

1984 0255	Fetch Module EDS
1984 0230	XM Module EDS
1982 8037	Memory Interface Module EDS
1982 8060	Memory Module SDS
1983 6915	I/O Subsystem EDS
1983 6899	Maintenance Interface Specification
1987 1227	SMC Specification

3 GENERAL DESCRIPTION

The B4900 is a high performance medium-scale computer that is instruction set compatible with past Burroughs Medium Systems computers. Its primary goal is to achieve a performance level of 1.8 to 2.0 times the B4800 processor while making use of inexpensive off-the-shelf components.

3.1 OVERVIEW OF B4900 ARCHITECTURE

The B4900 consists of seven basic modules:

1. FETCH
2. XM
3. MEMORY INTERFACE MODULE
4. MEMORY
5. IOP
6. MAINTENANCE PROCESSOR
7. SMC

Refer to the System Block Diagram in Section 3.2.

The FETCH module retrieves raw instructions from memory and resolves their field lengths and addresses in preparation for sending to the XM. In addition, it handles code and data interlocks.

The EXECUTE module (XM) performs the necessary processing of the execute cycle of an instruction. The B4900 Processor may be configured in either a single or dual XM configuration.

The Memory Interface Module (MIM) interfaces the XM and FETCH modules to memory in addition to keeping track of certain state.

The MEMORY module contains the system main memory in addition to intelligent addressing and formatting logic.

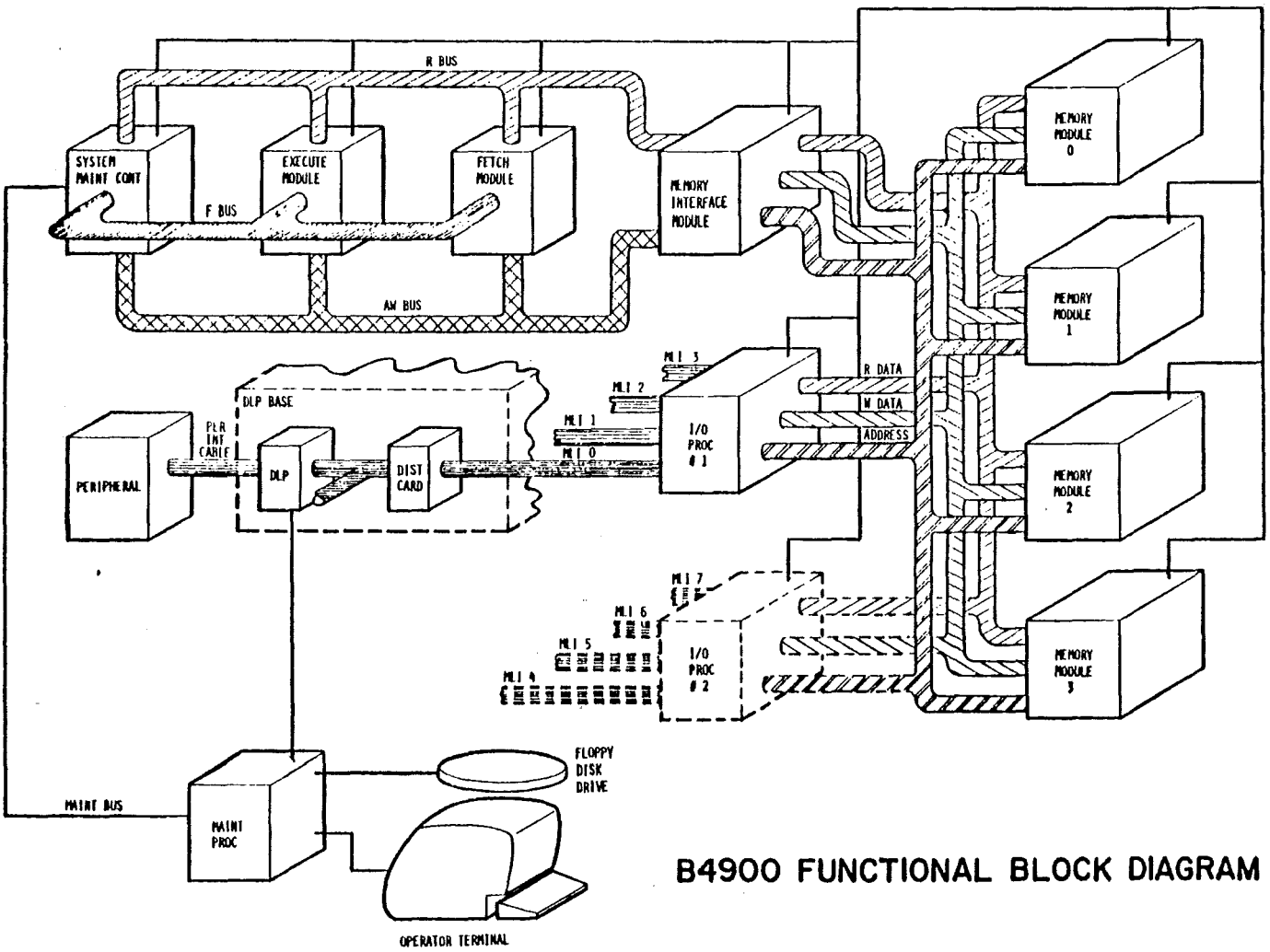
The I/O Processor (IOP) is an I/O processing element that interfaces the DLP's and peripherals to main memory and the processor. The B4900 Processor supports up to two IOP's, with each IOP handling up to four DLP bases.

The MAINTENANCE Processor (MP) handles the operator interface to all of the hardware elements of the B4900 Processor including system initialization. It is the vehicle for troubleshooting problems in the processor.

3.2 SYSTEM BLOCK DIAGRAM

Refer to the following System Block Diagram.

B4900 FUNCTIONAL BLOCK DIAGRAM

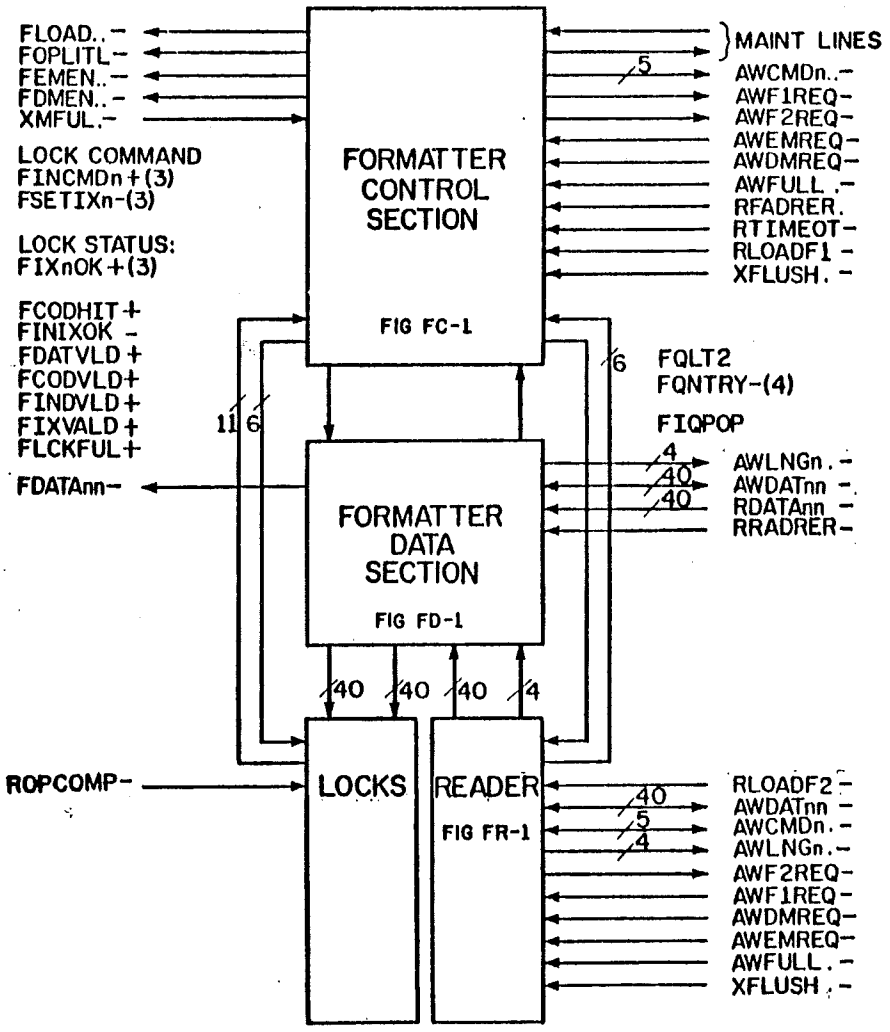


B4900 FUNCTIONAL BLOCK DIAGRAM

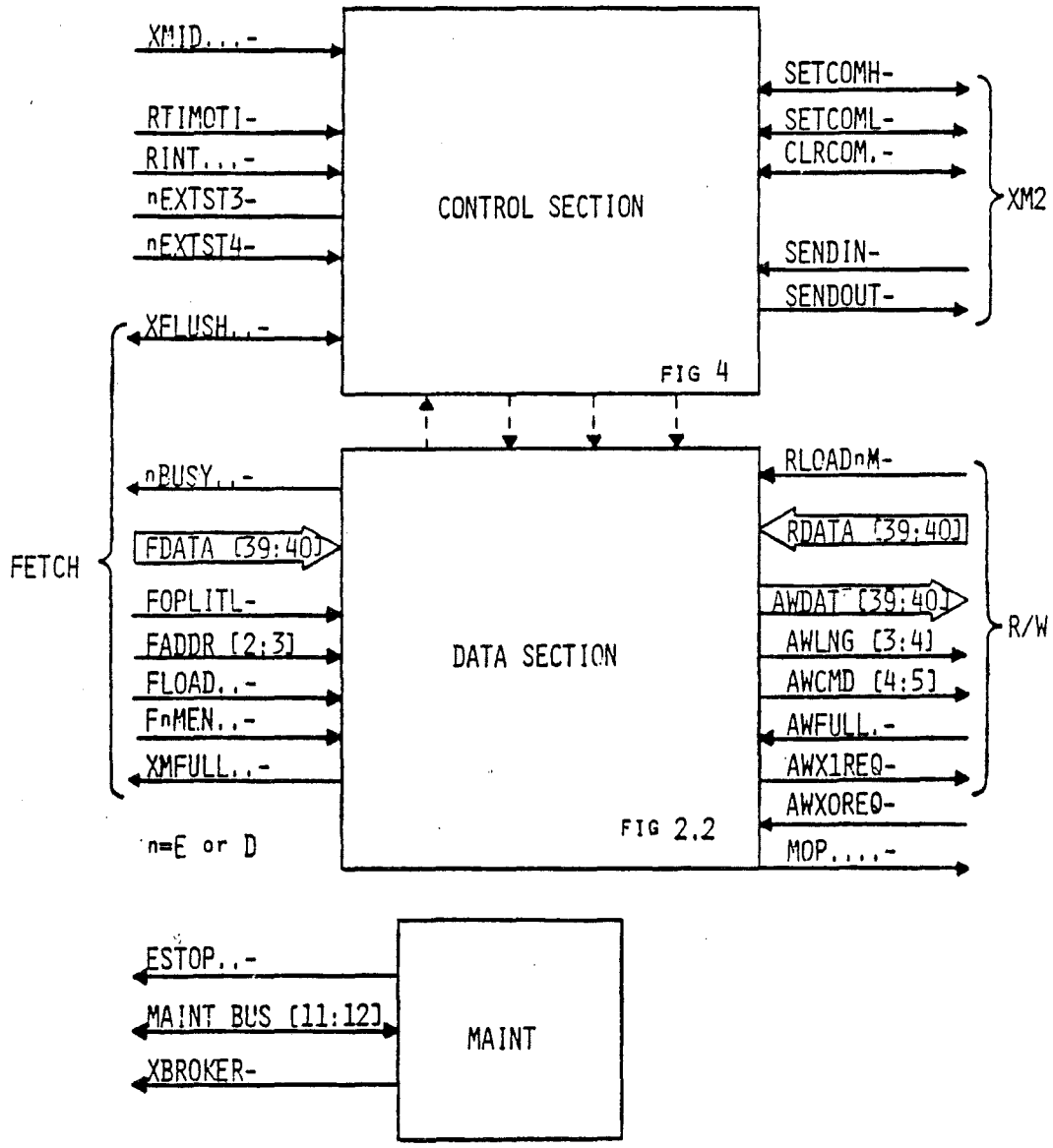
3.3 MODULE FUNCTIONS/INTERFACES

Refer to the following block diagrams of the individual modules.

FETCH BLOCK DIAGRAM

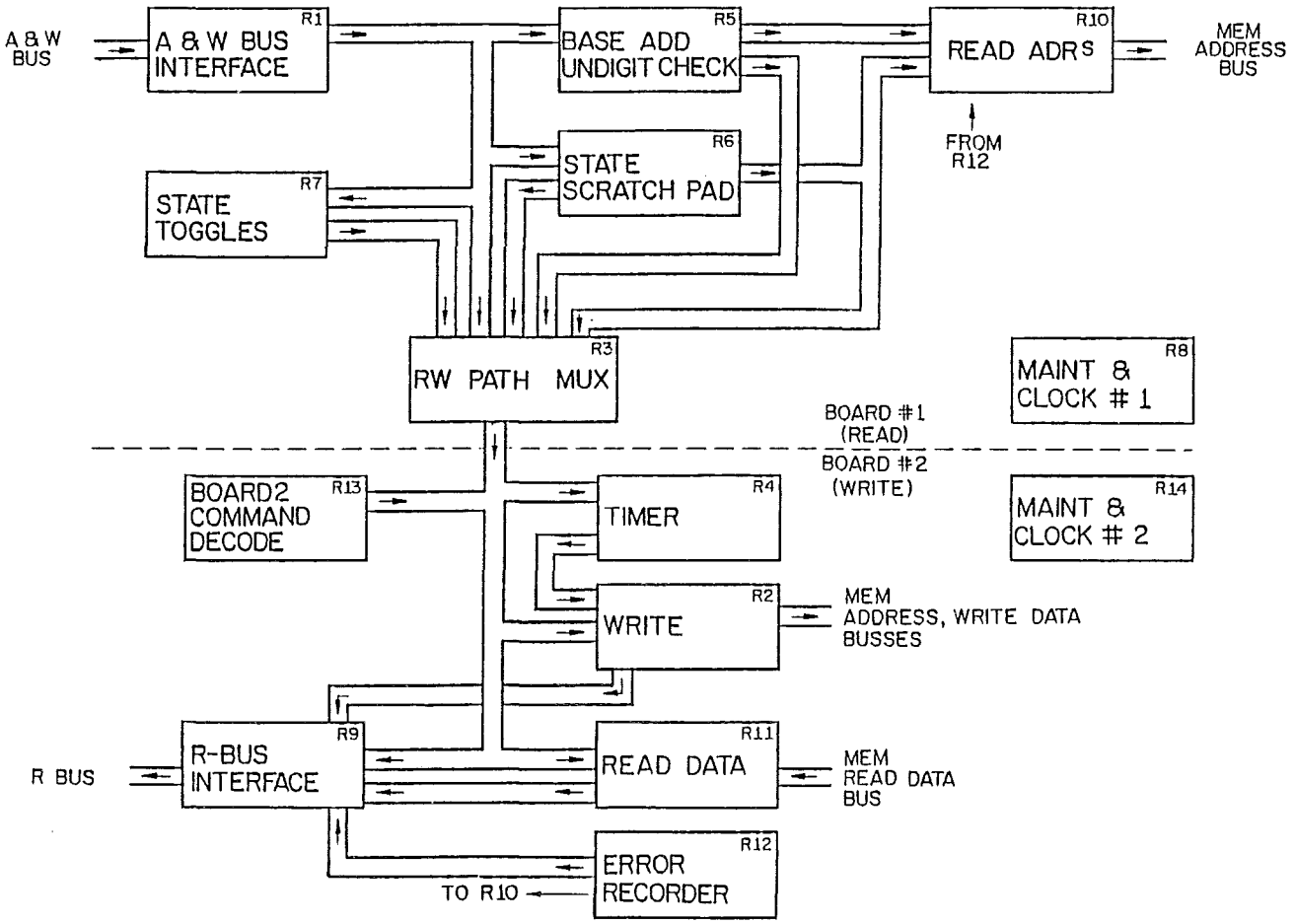


XM OVERALL BLOCK DIAGRAM



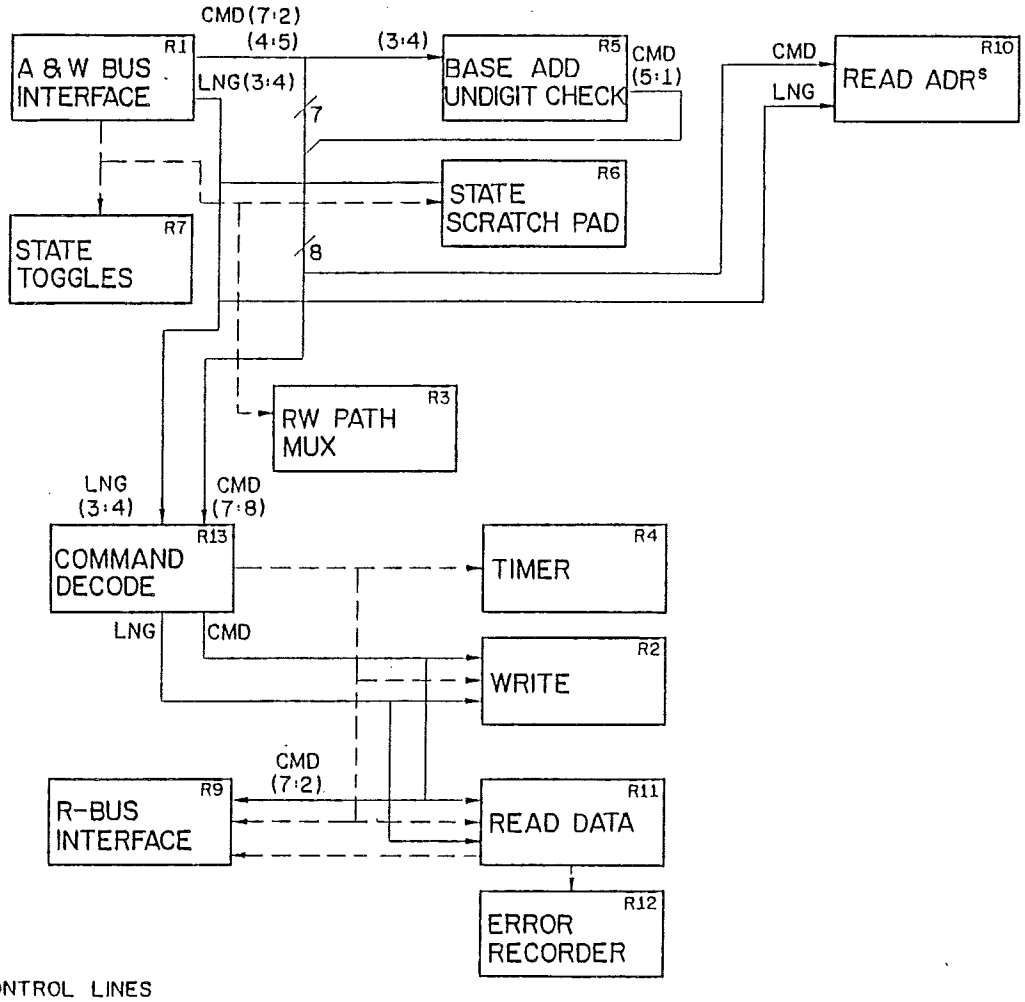
MEMORY INTERFACE MODULE

Top Block Diagram with Data Paths

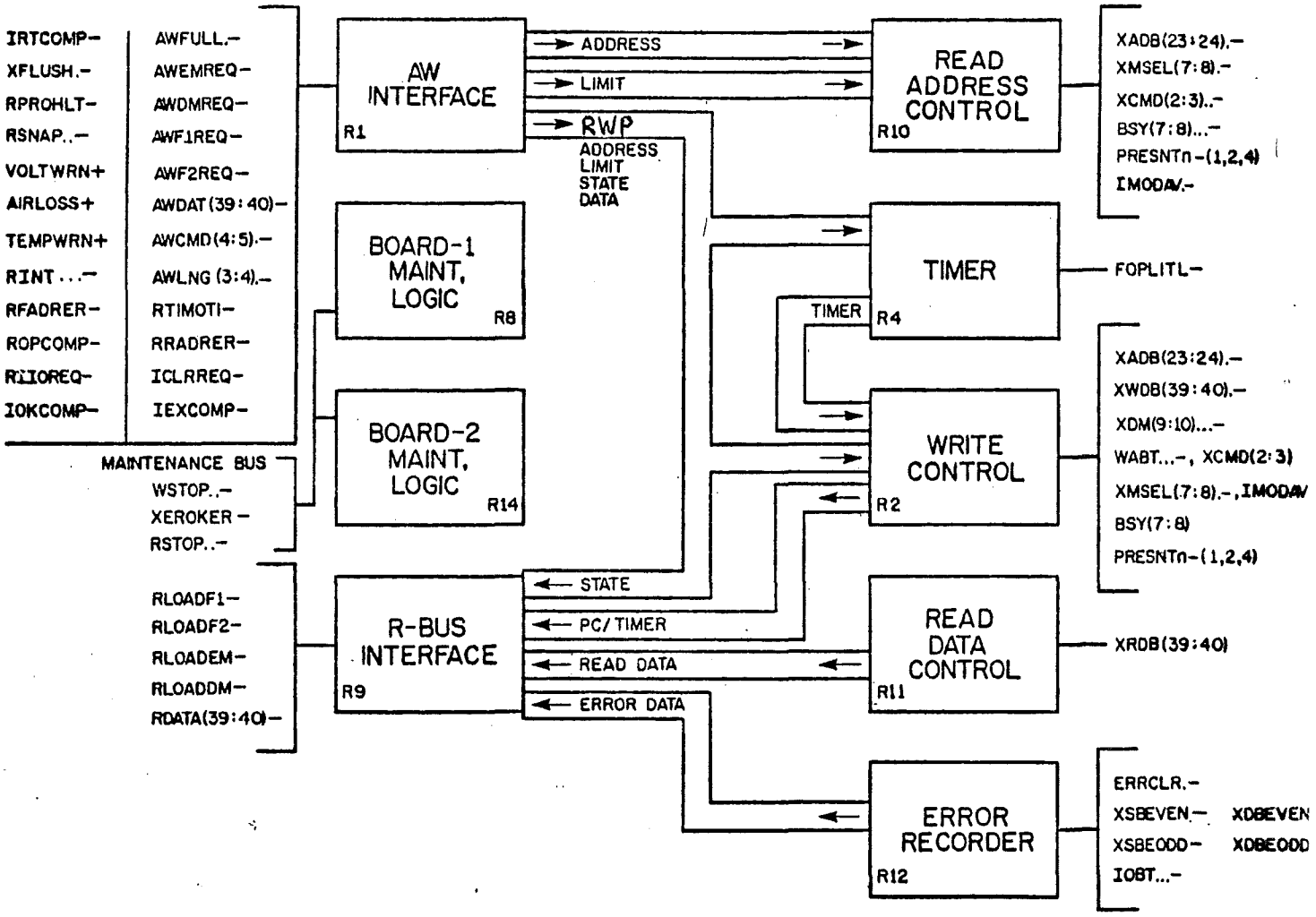


MEMORY INTERFACE MODULE

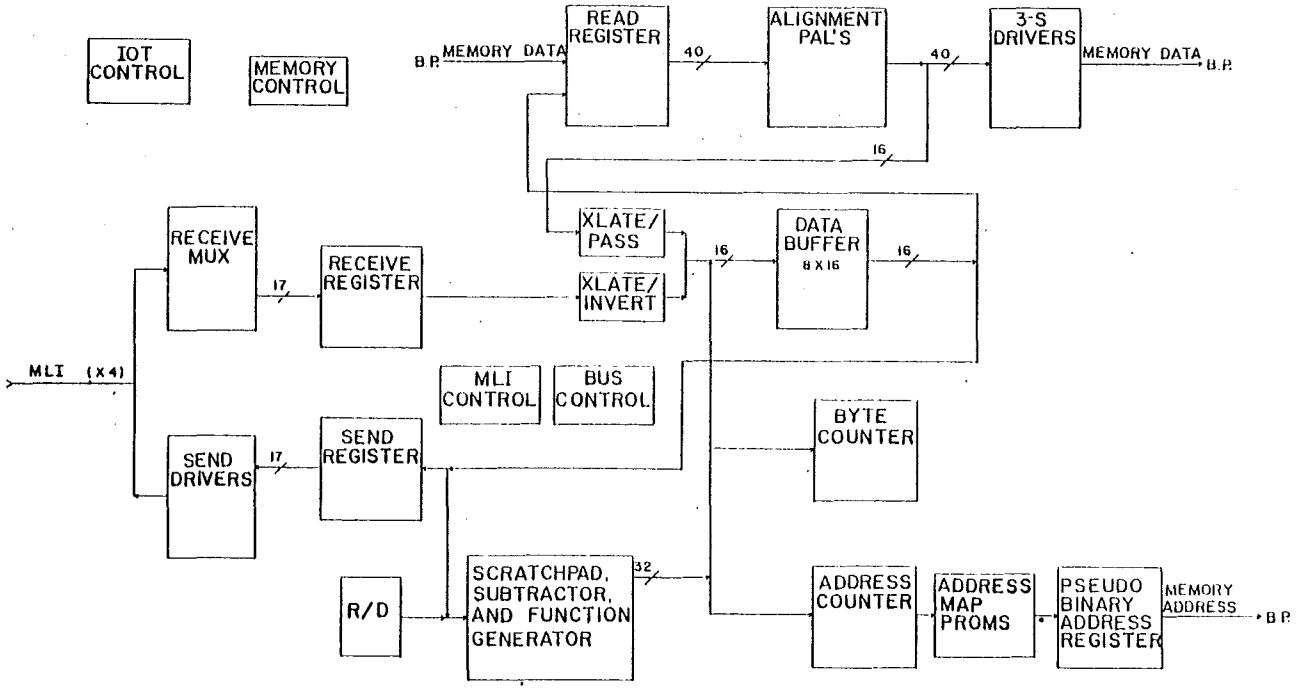
Top Block Diagram with Control Paths



MEMORY BLOCK DIAGRAM



IOP OVERVIEW



3.3.1 Fetch Module

The FETCH Module contains three major functional groups: the READER, the FORMATTER, and the LOCK CHECKER.

The function of the READER is to read the instruction stream out of main memory into the FETCH READ QUEUE of 50 digits. The FORMATTER parses the instruction out of the READ QUEUE into the XM FETCH PAGES. The FORMATTER also resolves any indexing by index registers, indirect addressing, and indirect field lengths. Each FETCH PAGE has a defined location and format for each type of instruction syllable. Several FETCH PAGE locations contain additional information read from memory or calculated by the FORMATTER for use by the XM's. The LOCK CHECKER works in close cooperation with the FORMATTER to detect instruction code modification and data interlocks in the pipeline.

The format of the XM FETCH PAGES is as follows:

ADDRESS	CONTENTS	
0	0000PPPPPP:	Program Address of instruction
1	OPAFBFgggg:	OPSYL with final AF and BF (INFL's resolved) with trailing digits.
2	cE0AAAAAAA:	Non-literal data ASYL (Address right-justfied)
	LLLLLLgggg:	Literal ASYL with trailing unknown data
	cEfAAAAAAA:	Branch ASYL
3	cE0BBBBBBB:	BSYL (Address right-justified) (Or other information depending on opcode)
4	cE0CCCCCCC:	CSYL (Address right-justified) (Or other information depending on opcode)
5	000mmmmmmm:	ALEN (usually the length of A operand in digits; UNDEFINED when AF is a literal) see following opcode table for contents;
6	00000nnnnn:	BLEN (usually the length of B operand in digits; see following opcode table for contents)
7	gggggggggg:	Undefined

3.3.1 Fetch Module (Continued)

c = Address controller (with indexing bits still present)
P = Decimal digits for program counter
L = Unmodified literal data (may contain garbage digits after the significant digits within the literal field itself)
E = Extended address flag of original address syllable (0 =non-extended address; C,D,E,F(HEX)= extended address)
m,n= Decimal digits

f = This Branch Error Digit is used when an error is encountered by FETCH in resolving any indexing and/or indirection in ASYL.

The values for this 4-bit field are as follows:

0 = No error
1 = Improper undigits in Formatter memory read
2 = Resolved indexed address negative or greater than 10 MD
3 = Limit error in Formatter memory read
5 = Improper undigit in index register
6 = Improper undigits in indexed address syllable

All other values unused.

The ASYL format for instructions which branch (BUN, HBR, NO-OP, NTR, EXT, OFL Conditional branches) is different than other instructions. This is due to the requirement that invalid A addresses for conditional branches are only reported if the branch is taken.

This branch interface allows FETCH to perform all of the normal instruction fetch functions even when the branch is not predicted taken by FETCH. If the XM determines that the branch must now be taken, then the XM does not need to perform any of the FETCH functions in resolving indexing and indirection. It simply checks the ASYL branch error flag to know whether FETCH had found any errors. Before sending the new PC address to FETCH however, the XM must check the address for undigits, non-MOD 2, and BASE/LIMIT error, which means reading an entry from the BASE/LIMIT Table and checking the address against the current base/limit.

3.3.1 Fetch Module (Continued)

The following table defines some of the more important details of the FETCH interface to the XM. See the paragraphs following the table for explanation of the columns.

OP	ALEN	BLEN	INFL	OPLIT LSB USED	WRITE LOCK	COMMENTS
01 INC	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BF*Bc	
02 ADD	AF*Ac	BF*Bc	AF,BF	Y	CSYL:(GTR (AF,BF))*Cc	
03 DEC	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BF*Bc	
04 SUB	AF*Ac	BF*Bc	AF,BF	Y	CSYL:(GTR (AF,BF))*Cc	
05 MPY	AF*Ac	BF*Bc	AF,BF	Y	CSYL: (AF+BF)*Cc	
06 DIV	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BF*Bc CSYL:(BF=AF)*Cc	
08 MVD	=	=	AF	Y	BSYL:(CADR=BADR)FORWARD CSYL:(BADR=CADR)BACKWARD	
09 MVL	AF*Ac	=	AF,BF	Y	ASYL:AF*Ac BSYL:AF*Ac CSYL:AF*Ac	
10 MVA	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BF*Bc	
11 MVN	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BF*Bc	
12 MVW	AFBF*4	=	AF,BF	Y	BSYL:AFBF*4	
13 MVC	AFBF*4	=	AF,BF	Y	ASYL:AFBF*4 BSYL:AFBF*4	
14 MVR	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BLEN*100	
15 TRN	AFBF*Ac	=	AF,BF	Y	CSYL:AFBF*Cc	
16 SDE	AF*Ac	BF*Bc	AF,BF	Y	38:2	
17 SDU	AF*Ac	BF*Bc	AF,BF	Y	38:2	
18 SZE	AF*Ac	BF*Bc	AF,BF	Y	38:2	
19 SZU	AF*Ac	BF*Bc	AF,BF	Y	38:2	
20 NOP	=	=	=	=	=	
21 LSS	=	=	=	=	=	LSS: Predict Not Taken, Last time NT.

3.3.1 Fetch Module (Continued)

22	EQL	"	"	"	"	"	EQL/NT/NT
23	LEQ	"	"	"	"	"	LEQ/NT/NT
24	GTR	"	"	"	"	"	GTR/NT/NT
25	NEQ	"	"	"	"	"	NEQ/NT/NT
26	GEQ	"	"	"	"	"	GEQ/NT/NT
27	BUN	"	"	"	"	"	
28	OVF	"	"	"	"	"	Always Predicted Not Taken
29	HBR	"	"	"	"	"	
30	BCT	"	"	AF,BF	Y	"	Stops fetching after this op.
31	NTR	AFBF*2	"	AF,BF	Y	24:22 (40:6):2*AFBF+16	If AFBF=0000, then BLEN = 0. See Note 2.
32	EXT	"	"	"	"	24:22	See Note 3.
33	BST	AF*Ac	"	AF	Y	ASYL:AF*Ac	
34	BRT	AF*Ac	"	AF	Y	ASYL:AF*Ac	
37	SLL	AF*Ac	BF*Bc	AF,BF	Y	8:8	IF BF=00,BLEN=0.
38	SLD	AF*Ac	BF*Bc	AF,BF	Y	8:16	IF BF=00,BLEN=0.
39	SEA	AF*Ac	BF*Bc	AF,BF	Y	8:8	
40	BZT	AF*Ac	"	AF	Y	"	
41	BOT	AF*Ac	"	AF	Y	"	
42	AND	AF*Ac	BF*Bc	AF,BF	Y	CSYL:GTR (AF,BF)*Cc	Note Ac=Bc=Cc
43	ORR	AF*Ac	BF*Bc	AF,BF	Y	CSYL:GTR (AF,BF)*Cc	Note Ac=Bc=Cc
44	NOT	AF*Ac	BF*Bc	AF,BF	Y	CSYL:GTR (AF,BF)*Cc	Note Ac=Bc=Cc
45	CPA	AF*Ac	BF*Bc	AF,BF	Y	"	Note SN ALEN or BLEN will be one digit too large.
46	CPN	AF*Ac	BF*Bc	AF,BF	Y	"	
47	SMF	"	"	"	"	"	
48	HBK	"	"	AF	"		Note only 1 level of INFL on AF

3.3.1 Fetch Module (Continued)

49	EDT		BF*2	AF,BF	Y	CSYL:BF*20
50	IAD					note 1
51	IAS					ASYL:8 note 1
52	ISU					note 1
53	ISS					ASYL:8 note 1
54	IMU					note 1
55	IMS					ASYL:8 note 1
57	IMI					ASYL:8 note 1
58	ILD					
59	IST					ASYL:8
70	RAA					
71	RAS					ASYL:12 (SP) or 20 (DP)
72	RSU					
73	RSS					ASYL:12 (SP) or 20 (DP)
74	RMU					
75	RMS					ASYL:12 (SP) or 20 (DP)
76	RDV					
77	RDS					ASYL:12 (SP) or 20 (DP)
78	RLD					
79	RST					ASYL:12 (SP) or 20 (DP)
80	FAD					Invalid opcode (becomes error opcode:1E)
81	FSU					Invalid opcode (becomes error opcode:1E)
82	FMP					Invalid opcode (becomes error opcode:1E)
83	FDV					Invalid opcode (becomes error opcode:1E)
*84	ACM					
88	D2B	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BF*Bc
89	B2D	AF*Ac	BF*Bc	AF,BF	Y	BSYL:BF*Bc
90	BRE			AF,BF	Y	Resolved AVBV must be decimal; literals not allowed.
91	SRD			AF,BF	Y	8:8
92	RAD			AF,BF	Y	ASYL=2:8
94	IIO			AF,BF	Y	
95	RDT				Y	ASYL:6
96	RCT				Y	ASYL:6
97	STT			AF,BF	Y	

*ACM is not invalid.

3.3.1 Fetch Module (Continued)

B1	NT1					LSS/NT/T: Predict Not Taken, Last time Taken.
B2	NT2					EQL/NT/T
B3	NT3					LEQ/NT/T
B4	NT4					GTR/NT/T
B5	NT5					NEQ/NT/T
B6	NT6					GEQ/NT/T
E1	TN1					LSS/Taken/Not Tak.
E2	TN2					EQL/T/NT
E3	TN3					LEQ/T/NT
E4	TN4					GTR/T/NT
E5	TN5					NEQ/T/NT
E6	TN6					GEQ/T/NT
F1	TT1					LSS/Taken/Taken
F2	TT2					EQL/T/T
F3	TT3					LEQ/T/T
F4	TT4					GTR/T/T
F5	TT5					NEQ/T/T
F6	TT6					GEQ/T/T
1A	INT					Interpreter Enter
1B	IXT					Interpreter Exit
1D	DMY					Dummy Opcode
1E	ERR					Internal Error OP (See note 4)
DB	DBG					Locks all memory debug opcode
DC	ECC					Locks all memory Read/Write ECC opcode (See note 5)

Note 1: FETCH fetches the 8 digit memory operand and passes it left-justified in the BSYL location in the FETCH page.

Note 2: NTR Fetch Page Format

Location 0: PC
1: OPSYL
2: ASYL
3: 0000mmmmmm (contents of memory 40:6)
4: nnnnnnnnnn (2xAFBF)

dule (Continued)

Fetch Page Format

tion 0: PC
1: OPSYL
2: ASYL
3: SgSmmmmmmm (Contents of Ix3, S=sign)

Error Opcode is an internally generated opcode used informing the XM of errors found by FETCH.

AF field specifies the error as follows:

valid Instruction = Non-specific
valid Instruction = Improper INFL
valid Instruction = Improper AF or BF, or improper literal

dress Error = Non-specific detected by FETCH
dress Error = Undigits in memory read address
dress Error = Resolved indexed address negative or greater than 10 MD
dress Error = Limit Error in memory read address
dress Error = Improper undigits in index register
dress Error = Improper undigits in indexed address syllable
dress Error = READER limit error reading code stream
dress Error = READER undigit error reading code stream

struction Timeout detected by FETCH

Bit Memory Parity Error detected by FETCH

3.3.1 Fetch Module (Continued)

Note 5: ECC Fetch Page Format

Location 0: PC
1: OPSYL
2: ASYL
3: BSYL
4: mmmmmmmmmm (contents of memory specified by BSYL plus 10)

The ALEN and BLEN columns define whether that FETCH PAGE entry is used and, if so, how that value is calculated.

The INFL column specifies on which fields indirect field lengths are resolved by FETCH.

The OPLIT LSB USED column defines whether FETCH detects literals in the AF field and then stores the result in the least significant bit of the OPLIT register. This field does not define whether literals are actually legal for each op code. An illegal literal is detected by the XM by the fact that the OPLIT register points to the literal entry point for an instruction op code for which literals are illegal. (See the section on the OPLIT Register Branch Mechanism.)

The WRITE LOCK field shows the calculations for the memory area(s) that FETCH locks for each instruction (which must include all memory areas to which that instruction writes).

The FETCH module does not normally check the numeric address portion of the ASYL, BSYL, and CSYL for undigits. FETCH only checks for undigits in the index register data and indirect addresses it encounters while resolving the final address in non-direct addresses.

3.3.2 XM Module

The XM Module basically performs the instruction execution, which in most cases includes operand Fetch.

The single XM model consists of only XM(E). The dual XM model consists of XM(E) and XM(D).

The XM module has the capability of branching on an external signal to determine whether it is XM(E) or XM(D). Since the READ/WRITE CONTROL reports errors by XM address, the XM's check for themselves in error cases to see if their specific address bit is set.

3.3.3 Memory Interface Module

The Memory Interface Module performs the various memory read/write functions, Base/Limit Table read/write functions, STATE Toggle read/write functions, and the instruction Timeout detection function.

The Memory Interface Module receives the following commands over the A&W BUS from the XM and FETCH modules. Note that these command field values may differ from the XM's internal representation of these commands up until it is output onto the A&W BUS.

	COMMAND DESCRIPTION	COMMAND	DATA	LENGTH
1.	READ ADDRESS (Legal bases=0-9)	05	xxxIAAAAAA	L
2.	READ ADDRESS (Legal bases=A,B)	07	xxxIAAAAAA	L
3.	READ SYNDROME	04	xxx0000000	0
4.	READ TIMER(Writes timer in MEM)	0C	xxxxxxxxxx	6
5.	READ and CLEAR TIMER(Writes timer in MEM and clears)	0D	000000xxxx	6
6.	SET TIMER (Loads timer)	19	DDDDDDxxxx	x
7.	WRITE ADDRESS (Legal bases=0-9)	02	xxxIAAAAAA	"AH"
8.	WRITE ADDRESS (Legal bases=A,B)	03	xxxIAAAAAA	"AH"
9.	WRITE DATA	0E	DDDDDDDDDD	L
10.	WRITE PC (OP COMPLETE)	0B	xxxAAAAAAA	"AH"
11.	WRITE PC (NOT OP COMPLETE)	0A	xxxAAAAAAA	"AH"
12.	READ PC	17	xxxxxxxxxx	x
13.	SET STATE TOGGLES	18	DDDDxxMMDD	x
14.	READ STATE TOGGLES	10	xxxxxxxxxx	x
15.	READ BASE/LIMIT TABLE	12	xxxxxxxxxx	"SA"
16.	WRITE BASE/LIMIT TABLE	1A	DDDDDDDDDD	"SA"
17.	MEASUREMENT OP	1E	DDDDDDDDDD	x
18.	NEW PC FOR FETCH(PIPELINE CLEAR)	1F	xxxAAAAAAA	0
19.	READ ECC	06	xxxIAAAAAA	0
20.	WRITE ECC DATA	0F	xxxxxxxxDD	"AH"

Data is left-justified; addresses are right-justified.
L = Length of 0-9, where 0 implies 10.
A = Decimal digits for addresses.
I = Base indicant.
D = Data.
M = Mask.
x = Unused.
"AH"= Hex digit "A" is required on the bus.
"SA"= An address in the range 0-F (HEX).

3.3.4 Memory Module

The B4900 Processor can be configured with one, two, four, or eight memory modules. Software must be enhanced to reference more than 10 million digits due to the current limitations of the size of an index register.

3.3.5 IOP

The B4900 Processor can be configured with one or two IOPs.

3.3.6 Maintenance Subsystem

Refer to the Maintenance Interface Specification or SMC Specification for information.

4 INSTRUCTION FLOW

4.1 GENERAL DESCRIPTION

Instructions are fetched out of memory into the FETCH INSTRUCTION QUEUE within the FETCH module by the FETCH READER. The FETCH FORMATTER parses the instructions out of the QUEUE and into the XM FETCH PAGES, alternating between the two XM's after every instruction. Each XM contains two FETCH PAGES so that while an XM is processing an instruction in one FETCH PAGE, the FORMATTER may fill up the other FETCH PAGE with one of the succeeding instructions.

The FORMATTER resolves any indexing, indirection, or indirect field lengths found. In addition, it fetches memory operands for some of the instructions. The FORMATTER and LOCK CHECKER work together to detect any data interlock situations in the pipeline, so that an instruction in the pipeline needing an area of memory being written by another instruction ahead of it in the pipeline will know it must wait until the latest data is present in memory (See Section 4.4 on Pipeline Data Interlocks).

4.1 GENERAL DESCRIPTION (Continued)

When an XM has completed one instruction, its clock is frozen until a new instruction is present in its next FETCH PAGE. In the single XM model the XM may perform any function necessary to execute the instruction at anytime. However, in the dual XM model, the XM may issue memory read requests at anytime but is generally not allowed to read or write any other machine state until is has received the SEND signal from the other XM.

For those instructions for which the XM must write to memory, it must send a PC syllable to WRITE CONTROL as the last memory write transaction for each memory write lock to signal that all memory writes for that memory write lock are complete. Upon receipt of the PC syllable WRITE CONTROL signals FETCH LOCK CHECKER, so that any instruction which is data interlocked on this completed write lock may then be allowed to execute in an XM.

After the XM has finished processing its instruction, it must check for any instruction-related errors which were detected by other modules and for any external interrupt conditions. Instruction-related errors include "undigit address" errors, limit errors, or memory parity errors detected by the Memory Interface Module or memory modules. External interrupts include I/O completes, timer, air loss/over temperature, and TRACE interrupts.

The XM performs all memory writes required by interrupt processing. However, if no memory write is done (i.e., I/O complete in CONTROL state), then the XM can simply branch on the OPLIT register to get the next FETCH page. If a processor R/D memory write is required (i.e., timer interrupt in CONTROL STATE/MCP BASE), then a pipeline clear is performed to refetch the next instructions because the memory write to the Processor R/D had not gone through the pipeline data interlock mechanisms.

If no interrupts are present, the single XM model passes on the SEND signal and branches on the OPLIT register in the same clock. However, in the dual XM model, the XM must pass on the SEND signal one clock before branching on the OPLIT register to prevent deadlocks from occurring between FETCH, XME, and XMD. This is due to the SEND signal being inhibited while the XM is live frozen waiting on the OPLIT register.

4.2 OPLIT REGISTER BRANCH MECHANISM

In order to eliminate the clocks the XM would have spent on every instruction just to decode the opcode and check for a literal, the OPLIT register branch interface between the FETCH module and the XM was developed. The OPLIT register is an 8 bit register in the XM loadable by FETCH and is associated with the current fetch page.

As FETCH parses the instruction opcode, it determines the unique 7-bit encoded OPLIT value for that opcode. FETCH also remembers if it detected the AF literal flag. After FETCH has completed filling the FETCH PAGE, it fills the OPLIT register with the 7-bit encoded opcode value and the 1-bit literal flag.

On the last clock of instruction processing by the XM, the XM switches its FETCH PAGE and performs a 256-way branch based upon the contents of the OPLIT register. Thus on the very first clock of the next instruction execution, the XM already knows the opcode and whether the A address syllable contains a literal. If not at a literal data entry point, the XM immediately issues its memory request for the data; thus eliminating possible dead clocks of memory wait time later.

An OPLIT register branch also clears the XM subroutine stack in preparation for the next instruction.

4.3 MACHINE STATE

The machine state consists basically of main memory, the processor memory scratchpad, the IOP memory scratchpad, the BASE/LIMIT Table and STATE Toggles in Memory Interface Module, and the COMPARISON toggles, SEND signal, and ACCUMULATOR contained in the XM's.

4.3.1 Send Signal

If the XM in a single XM model starts processing an instruction, then there can be no instructions ahead of it in the pipeline. Therefore it is not on a bad branch path and will fully execute. However, in the dual XM, an XM must receive the SEND signal before it knows that its current instruction is really going to be executed. As a result, an XM in the dual XM model must not change machine state before receiving the SEND signal. In addition the SEND signal is used for interlocking some other lesser used resources.

The following actions must not be performed by an XM before it receives the SEND signal.

a. WRITE to Main Memory
b. WRITE to STATE Toggles
c. WRITE to BASE/LIMIT Table
d. WRITE to ACCUMULATOR (in XME)
e. Force GLOBAL COMS Update
f. Issue I/O DESCRIPTOR READY to IOP
g. READ certain STATE Toggles (some may not be safely read as they may be changed by the previous instruction)
h. READ/WRITE Processor Memory Scratchpad in Sub-MCP Base Memory
i. Report any errors to the MCP (error may be on bad branch path)
j. Perform Pipeline Clear

Thus the SEND signal enforces the sequentiality of instructions (along with the data interlocking mechanism in FETCH CONTROL) and validates the oldest instruction in the pipeline with the right to change machine state. The machine state must always be managed such that it reflects only the results of previously executed instructions and never instructions in the pipeline that may never execute.

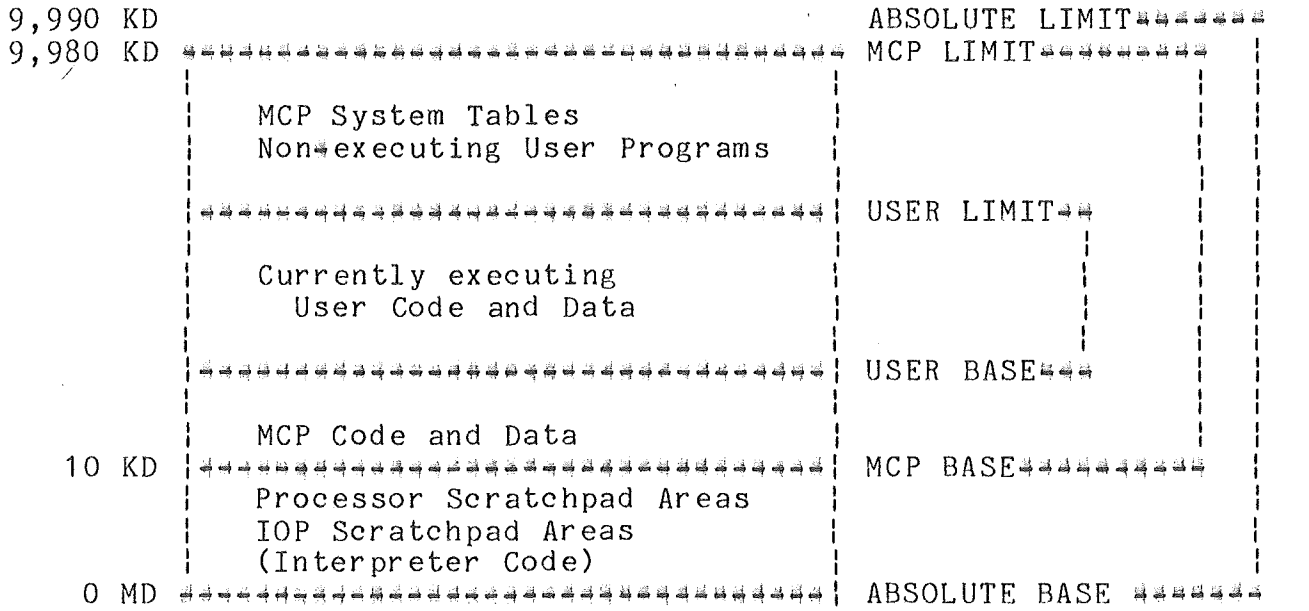
Note: Some STATE toggles are always valid and may be read by an XM at anytime, such as the USER/MCP toggle and CONTROL/NORMAL state toggle.

4.3.2 Main Memory Layout (Absolute, MCP, and User Bases)

The B4900 Processor has three base/limits addressable by an XM or FETCH at any time. The Privileged Base/Limits (Absolute and MCP) are always available for accessing the first million digits of those memory areas. The third base/limit (current code) reflects the currently executing code stream, which is either the user base/limit, MCP base/limit, or Absolute base/limit. The Current Code base/limit implementation uses ten base/limit table entries to allow access to a full ten million digit memory.

The following diagram illustrates the hierarchical nature of the base/limits which allows a possible interpreter to simulate Medium Systems instructions used by either the user programs or the MCP.

MAIN MEMORY LAYOUT



The Sub-MCP Base Area of memory has the following areas of its memory reserved for special uses:

a. Addresses 380-389: SNAP Picture Report Address (Contains an ABSOLUTE BASE relative address of the form:000AAAAAAA)

4.3.2 Main Memory Layout (Absolute, MCP, and User Bases) (Continued)

b. Addresses 390-399: Memory Error Report Address (Contains an MCP BASE relative address of the form:000AAAAAAA)
c. Addresses 400-3999: IOP Channel Scratchpad Memory

4.3.3 Base/Limit Table

The BASE/LIMIT Table is a 16-deep RAM contained in READ/WRITE CONTROL. The first twelve entries are reserved for base/limit handling under MCPIX and are allocated as follows:

a. Entries 0-9 contain the base and limit for one million digit sections of the machine. The base entries are indexed into by the millions digit (digit 6) of an address. The first 5 digits of the selected entry provide the absolute base (in thousands of digits) and the last 5 digits provide the absolute limit (in thousands of digits).
b. Entry A (HEX) contains the total MCP base/limit.
c. Entry B (HEX) contains the Absolute base/limit.
d. Entry C (HEX) contains the TIMER H word.
e. Entries D-F (HEX) are unused.

4.3.4 Comparison Toggles

Due to the performance impact of conditional branches and timing considerations in the XM, each XM contains a duplicate copy of the COMPARISON toggles. Both copies receive the same inputs and are therefore changed at the same times.

When the XM issues an external command to set the COMPARISON toggles (COMS), the new COMS value is saved off in an internal store in the XM. If at least one SETCOMS external command has been issued, the COMS value in the internal store is copied to the XMs' global COMPARISON toggles on the same clock the SEND signal is transferred. An XM can not issue an external command to set the COMS on the same clock that it passes on the SEND signal, since that new value would not yet have been transferred to the internal store. On the same clock that the SEND signal is passed on, the internal toggle that signifies that an external SETCOMS command was issued by this XM is also reset as part of the reinitialization for the next instruction.

Since the interrupt routines in the XM must get the value of the COMS without passing on the SEND signal, the XM has another special command to force the global COMS to be updated immediately to the value of the local COMS.

4.3.5 State Toggles

The following STATE toggles are maintained in READ CONTROL:

	RFFmn	SET BY	CLEARED BY
I/O DESC READY	39	XM	IOP
PROCESSOR HALT	38	XM	NEXT CLOCK
ADDRESS ERROR → SNAP	37	XM	MAINT
INSTR ERROR → SNAP	36	XM	MAINT
*TIME OUT 1	35	TIMER	XM
*TIMER INTERRUPT	34	TIMER	XM
*I/O COMPLETE	33	IOP	XM
*XM WRITE 2→BIT MEM ERROR	32	WRITE	XM
*F/XM 1→BIT,IOP MEM ERR	31	WRITE	XM
*SNAP GATE PIC REPORTED	30	MAINT	XM $
*AIR LOSS/OVER TEMP	29	S	XM
TEMPERATURE WARNING	28	S	XM
VOLTAGE WARNING	27	S	XM
TIME OUT 2	26	TIMER	MAINT
$*REAL→TIME I/O COMPLETE	25	IOP	XM
$*EXCEPTION I/O COMPLETE	24	IOP	XM
FORMATTER LIMIT ERROR	23	READ	XM;NEWPC,PIPECLEAR
FORMATTER UNDIGIT ERROR	22	READ	XM;NEWPC,PIPECLEAR
READER LIMIT ERROR	21	READ	XM;NEWPC,PIPECLEAR
READER UNDIGIT ERROR	20	READ	XM;NEWPC,PIPECLEAR
*XME READ LIMIT ERROR	19	READ	XM
*XME READ UNDIGIT ERROR	18	READ	XM
*XMD READ LIMIT ERROR	17	READ	XM
*XMD READ UNDIGIT ERROR	16	READ	XM
*XME WRITE LIMIT ERROR	15	WRITE	XM
*XME WRITE UNDIGIT ERROR	14	WRITE	XM
*XMD WRITE LIMIT ERROR	13	WRITE	XM
*XMD WRITE UNDIGIT ERROR	12	WRITE	XM

4.3.5 State Toggles (Continued)

	FMTTR READ 2-BIT MEM ERR	11	WRITE	XM;NEWPC,PIPECLEAR
	READR READ 2-BIT MEM ERR	10	WRITE	XM;NEWPC,PIPECLEAR
*XME	READ 2-BIT MEM ERR	09	WRITE	XM
*XMD	READ 2-BIT MEM ERR	08	WRITE	XM
	OVF	07	XM	XM
	NORMAL	06	XM	XM
	USER	05	XM	XM
	MEM ERROR RPRT ENBL	04	XM	XM
	SNAP ENABLE	03	XM	XM
	CPINT	02	XM	XM
	TIMER INT. R/D STORED	01	XM	XM
*TRACE		00	XM	XM

The STATE Toggles denoted with dollar signs ($) are under development.

The STATE Toggles with asterisks above are "ORR-ed" together to form the INTERRUPT signal (RINT) that feeds into the XM.

The FETCH UNDIGIT ERROR, FETCH LIMIT ERROR, and FETCH READ 2-BIT MEM ERR STATE Toggles are "ORR-ed" together to form the FETCH FORMATTER ERROR signal (RFADRER), which is returned back along with the memory data.

The READER UNDIGIT ERROR, READER LIMIT ERROR, and READER READ 2-BIT MEM ERR STATE Toggles are "ORR-ed" together to form the FETCH READER ERROR signal (RRADRER), which is returned back along with the memory data.

Thus, since all of the above toggles are "ORR-ed" together in the FETCH module, the FORMATTER must read up the STATE Toggles to see the sources for the error signal.

4.3.6 Accumulator

The ACCUMULATOR is stored and maintained only in XM(E). In the dual XM model FETCH CONTROL therefore sends all ACCUMULATOR instructions to XM(E), which means sending dummy opcodes to XM(D) instead of ACCUMULATOR instructions. This dummy opcode does nothing except wait on the SEND signal and then pass it back to XM(E).

4.3.7 Snap Picture Report Address, Memory Error Report Address

The SNAP Picture Report Address is located in Sub-MCP Base memory at absolute memory locations 380-389. It is a ABSOLUTE BASE-relative address of the form: "000AAAAAAA".

The Memory Error Report Address is located in Sub-MCP Base memory at absolute locations 390-399. It is a MCP BASE-relative address of the form: "000AAAAAAA".

4.4 PIPELINE DATA INTERLOCKS

The FETCH FORMATTER and LOCK CHECKER detect and handle the memory data interlock cases which occur in the instruction pipeline. This happens when an instruction in the pipeline needs to read a memory location which is being written by another instruction ahead of it in the pipeline. The FORMATTER issues to the LOCK CHECKER write locks for those memory areas which an instruction will write. When a memory write is complete, WRITE CONTROL notifies the LOCK CHECKER which then removes that write lock from its table. At various points in its flow the FORMATTER checks for data interlocks by sending the starting and ending addresses of a memory area to the LOCK CHECKER to see if it intersects with any pending write locks. If a hit is found, then the FORMATTER waits until that write lock is removed before proceeding.

When an instruction with a write lock completes, WRITE CONTROL notifies the LOCK CHECKER, which increments a local counter. This counter contains the number of completed write locks not yet removed from the write lock table. The counter is necessary because write locks can only be removed from the table at distinct, synchronized points. These points are the code modification check, the LOCK CHECKER FULL condition, and a DATA/INDEX/CODE INTERLOCK FOUND condition.

After the LOCK CHECKER performs the code modification check (PC + 90), it uses the counter to remove those write locks whose writes had completed by the clock the FORMATTER made its code modification request. If the FORMATTER tries to give the LOCK CHECKER another request and the write lock table is full, then the FORMATTER freezes until the LOCK CHECKER removes some of the write locks due to write completions. In the data interlock case, the LOCK CHECKER will continue to remove write locks until the one causing the interlock is removed. To allow these removals at a point other than the FORMATTER code modification check, the FORMATTER must not clear the READER to a new PC until all write locks for an instruction have been posted and any potential freezes have been passed.

4.4.1 Address and Operand Data Interlocks

The FETCH FORMATTER and LOCK CHECKER have the responsibility for detecting and handling address and operand data interlocks due to the instruction pipeline. When the FETCH FORMATTER processes an instruction, it must issue write locks to the LOCK CHECKER for those memory areas into which this instruction will write. In some instances the FORMATTER locks all of memory rather than issuing multiple write locks for some instructions.

When the FETCH FORMATTER encounters an instruction requiring a memory read to resolve a field (i.e., index register, indirect address, or indirect field length), then the FORMATTER issues the memory request and asks the LOCK CHECKER to check for any outstanding locks posted for the desired memory. If the data is found to be interlocked, then FETCH throws the data away and waits for the appropriate instruction to complete before continuing. Otherwise, FETCH uses the data to resolve the appropriate field and continues parsing.

The FORMATTER also fetches the memory operand data for some of the instructions and stores it in the FETCH PAGE for the XM. Likewise the FORMATTER must perform a data interlock check for these memory reads.

In the dual XM model the FETCH module must also check for data operand interlocks between consecutive instructions to prevent the latter instruction in the second XM from accessing interlocked data before it is actually written to memory. When the FETCH module finds such a data interlock, it waits until the preceding instruction has completed its writes before marking the current instruction's FETCH PAGE full and thus ready for execution.

4.4.2 Code Modification Checking

The FETCH FORMATTER and LOCK CHECKER also have the responsiblity to detect and handle code modification. When the FORMATTER starts processing a new instruction, it asks the LOCK CHECKER to see if there are any outstanding write locks for the current PC plus 90 digits (40 digits=the number of 10-digit words that cover the largest instruction; 50 digits= maximum size of the FETCH INSTRUCTION READ QUEUE). If no write locks are pending for this memory area, then the FORMATTER continues parsing down this path. However, if code modification is found, then the FORMATTER waits for the LOCK CHECKER to inform it that the offending write lock has now completed its memory write. Now the FORMATTER may refetch the updated instruction from memory and proceed as normal.

4.5 DEFINITION OF PIPELINE CLEAR

The pipeline clear function is used to remove the instructions already in the instruction pipeline and begin processing a new instruction stream. This occurs when it has been determined that the instructions currently in the pipeline are not the ones to be executed next, such as when an XM encounters an incorrectly predicted conditional branch (see section 4.6) or an interrupt event requiring a context switch (INTERRUPT BCT 94). This mechanism may also be used to signal FETCH CONTROL to start using the new base and PC values during the execution of BCT and BRE instructions.

A pipeline clear is performed as follows:

1. The XM waits until WRITE CONTROL has received all of its bus transactions from the A&W BUS input queue; thus guaranteeing that all memory writes will complete.
2. XM issues NEWPC bus transaction containing the new PC into its A&W BUS output queue.

4.5 DEFINITION OF PIPELINE CLEAR (Continued)

3. During the sme clock that the NEWPC transaction is transferred on the bus, the XM sends a signal to a pipeline clear flipflop on the FETCH, XM, and READ/WRITE boards. In addition, the FETCH READER captures the NEWPC PC value off the A&W BUS.
4. During the succeeding clock the following actions are performed:
   a. FETCH INSTRUCTION READ queue and outstanding code requests are cleared.
   b. FETCH READER is cleared to start fetching using the new base and PC value.
   c. FETCH LOCK CHECKER is cleared.
   d. FETCH FORMATTER is cleared with its XM pointer reset to XM(E) and micro-PC reset to zero.
   e. The FETCH index register cache valid bits are reset.
   f. READ CONTROL is cleared.
   g. READ/WRITE A&W BUS input pointers are cleared.
   h. XM micro-PC is cleared to zero.
   i. XM RBUS input queue pointers are cleared.
   j. XM fetch page pointer is reset to internal FETCH PAGE #0.
   k. XM SEND signal is set for XM(E), reset for XM(D).
   l. The FETCH LIMIT ERROR, FETCH UNDIGIT ERROR, FETCH MULTI-BIT MEMORY ERROR, READER LIMIT ERROR, READER UNDIGIT ERROR, and READER MULTI-BIT MEMORY ERROR STATE Toggles are reset.
5. During the next few initialization clocks XM(E) executes microcode to send an A&W BUS transaction to clear the XME READ LIMIT ERROR, XME READ UNDIGIT ERROR, XMD READ LIMIT ERROR, and XMD READ UNDIGIT ERROR STATE Toggles. These toggles may have been set by instructions in the pipeline which did not ever execute.
6. XM(E) executes microcode to branch on the OPLIT register without transferring the SEND signal so that it can start processing its first instruction.
7. The pipeline clear function is now complete.

4.6 FETCH READER CLEAR FUNCTION

The FETCH READER Clear function is initiated by the FORMATTER when it has been determined that the instructions in the INSTRUCTION READ QUEUE are not the ones it wants to parse next. Therefore, the FORMATTER issues a NEWPC command to the READER which causes the following to occur.

1. FETCH READER INSTRUCTION READ QUEUE is cleared and the number of outstanding READER memory requests is added into a garbage read counter, so that the READER knows how many outstanding memory reads to throw away.
2. FETCH READER issues new memory read requests for the code stream starting at the new PC address plus 10.
3. When the NEWPC command is received by the MEMORY CONTROL, it is treated as READER memory read request. In addition, the READER LIMIT ERROR, READER UNDIGIT ERROR, READER MULTI-BIT MEMORY ERROR, FETCH LIMIT ERROR, FETCH UNDIGIT ERROR, and FETCH MULTI-BIT MEMORY ERROR STATE Toggles are reset.
4. A flag is pipelined through the MEMORY CONTROL along with the NEWPC memory read to reset the READER READ 2-Bit Memory Error and FORMATTER 2-Bit Memory Error State Toggle. This prevents any garbage outstanding reads from erroneously setting this STATE TOGGLE.

4.7 BRANCH PREDICTION

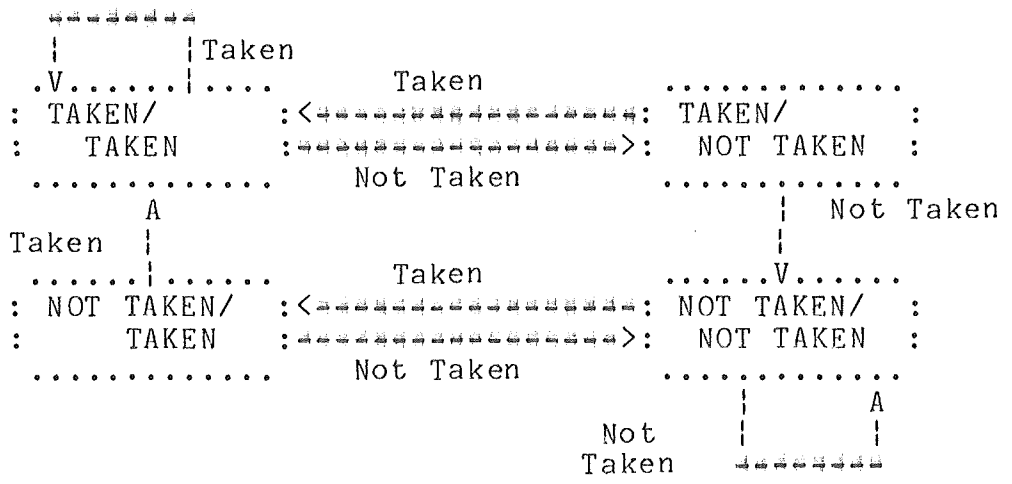
When the FETCH module encounters a conditional branch, it selects one of the two paths as most likely, based solely upon the opcode. It then continues fetching instructions down that path. Since FETCH CONTROL is allowed to keep processing by not having to wait for the correct COMPARISON toggles, a higher machine performance is achieved. It is the responsibility of the XM to then verify the correctness of the FETCH module's branch prediction against the correct COMPARISON toggles and, in the event of a bad prediction, to force the pipeline to execute instructions down the other path by performing a pipeline clear function.

4.7 BRANCH PREDICTION (Continued)

Modeling has demonstrated that if one maintains information about the most recent direction trend occurring at a conditional branch, one may achieve a higher overall correct branch prediction rate. The current design keeps track of the latest direction trend by re-writing the opcode with one of the four variants of each conditional branch opcode. Taking the same direction twice in a row is sufficient for establishing a different direction trend and therefore branch prediction direction.

a. TAKEN/TAKEN - Predict branch taken. The predominant direction trend is taken.
b. TAKEN/NOT TAKEN - Predict branch taken. The last time it was NOT taken (before then the branch was taken).
c. NOT TAKEN/TAKEN - Predict branch not taken. The last time it was taken (before then the predominant path was NOT taken).
d. NOT TAKEN/NOT TAKEN - Predict branch not taken. The predominant direction trend is NOT taken.

The opcode transitions are defined by the following graphic.



4.8 CONTEXT SWITCH EVENTS (BCT, BRE, INTERRUPTS)

The two programmatic context switch instructions (BCT and BRE), instruction-related errors (i.e., address errors), and external interrupts (i.e., I/O complete) can all cause a change from one set of base/limits to another.

For a programmatic BCT, the XM must first make sure all previous memory WRITES complete and check whether any had errors. If so, then the address of the instruction in error is reported in the RCW (MCP Base 60-76) instead of the next instruction address. Next the XM reads the current BASE/LIMIT'S, reads the Interrupt or BCT Address, and calculates the RCW values. Once the XM receives the SEND signal, it then writes the RCW, sets up the new BASE/LIMIT entries and STATE Toggles for the MCP, and pipeline clears the processor to the address specified by the Interrupt or BCT Address. For an Interrupt BCT, a similar approach is used except that the STATE Toggles are examined for the reason for the interrupt and a processor result descriptor written, if necessary. The reason for an interrupt may also be a program error detected by the microcode which passes its error to the same microcode interrupt handler routine via a scratchpad location instead of the STATE Toggles.

In the BRE instruction, the XM must examine the CPINT (Interrupt Toggle) and TIMER INTERRUPT R/D STORED STATE Toggles and the particular STATE (CONTROL or NORMAL) and BASE (USER/MCP) of the program being re-instated. If the new STATE is NORMAL and CPINT is set, then an Interrupt BCT is performed instead of the BRE without disturbing the RCW contents. If the new STATE is CONTROL, the BASE is USER, and the CPINT and TIMER INTERRUPT R/D STORED Toggles are set, then an Interrupt BCT is also immediately performed due to TIMER INTERRUPTS causing interrupts in the new environment.

Any other combination of these 3 toggles will cause the BRE to continue its normal re-instatement procedure. In this case the new BASE/LIMITS of the new program are converted into absolute BASE/LIMITS (10 KD added to offset past the sub-MCP base area) and written to the BASE/LIMIT Table. The STATE Toggles are set up to the new environment. The new program address is made base-relative rather than MCP BASE-relative, and a pipeline clear to this new PC is performed.

4.8 CONTEXT SWITCH EVENTS (BCT, BRE, INTERRUPTS) (Continued)

If an INVALID I/O DESCRIPTOR, TIMER INTERRUPT, or AIR LOSS/OVER TEMP interrupt condition is detected while in CONTROL STATE, then the Processor R/D is written and CPINT (and for TIMER INTERRUPTS, TIMER INTERRUPT R/D STORED STATE Toggle) is set. However, since an unanticipated memory write has occurred which did not go through the pipeline data interlock checking, then a pipeline clear to the next instruction is performed. If an I/O COMPLETE is detected while in CONTROL STATE, then the CPINT STATE toggle is simply set without an Interrupt BCT. This CPINT toggle is only checked by the BRE instruction and only reset by SRD.

4.9 TRACE HANDLING

A BRE instruction can specify that the TRACE Toggle be turned on, which forces an Interrupt BCT to occur at the end of the execution of the next instruction. To force this Interrupt BCT the XM fakes an interrupt by issuing an A&W BUS command to set the TRACE STATE Toggle during the BRE instruction. Since the TRACE Toggle is "ORR-ed" into the XM interrupt line, the XM will notice the pending interrupt at the end of the execution of the next instruction and perform the Interrupt BCT.

5 ERROR HANDLING INTERFACES

5.1 FETCH PHASE

All errors are reported to the XM as some variant of an internally assigned error opcode. Then the XM waits for the SEND signal before reporting the error to make sure that the instruction in error is really supposed to execute.

Any errors detected for FETCH CONTROL by READ/WRITE CONTROL are signalled via a line "ORR-ing" the FETCH UNDIGIT ERROR, FETCH LIMIT ERROR, READER UNDIGIT, AND READER LIMIT ERROR STATE Toggles.

The following errors are detected during the FETCH Phase:

1. Improper literals (undigit in length, length too long, length equal zero, etc.)
2. Invalid Indirect Field Length
3. Illegal undigits in index registers or indirect addresses
4. Undigits in resolved AF or BF (non-mask and non-literal)
5. Invalid opcode
6. Odd address specified for INFL's.
7. Improper Fall-thru Code Stream Address (READER address had improper undigits or limit error on fall-thru code)
8. Indirect address limit errors
9. Instruction timeout due to programmatic infinite indirection in indirect addressing or indirect field lengths.

5.2 EXECUTE (XM) PHASE

For all errors the XM waits for the SEND signal before reporting the the error to make sure that the instruction in error is really supposed to execute.

If at any point during the execution of an instruction by a specific XM, the instruction gets a READ ERROR or a WRITE ERROR; WRITE CONTROL inhibits any more writes to memory from that XM. This prevents any data read beyond the limit from being stored in memory within the current base/limit and still allows writes from the second XM to finish per normal.

The XM must detect all of the rest of the types of errors that FETCH and READ/WRITE are not responsible for, which includes many opcode-specific errors. FETCH passes its detected errors to the XM as error opcodes or as the Branch Error Flag in the ASYL (as in cond. branch). READ/WRITE CONTROL interfaces by setting certain STATE Toggles when errors are detected. After completion of the micro-code for an instruction, the XM checks the interrupt line for instruction-related errors (and interrupts). If a READ ADDRESS ERROR or READ LIMIT ERROR is signalled over the external interrupt line, the XM must check the correct toggles to see if it was actually the XM for which the error was detected before reporting any such error.

The errors which the XM must detect include the following:

1. Undigits in addresses (READ/WRITE CONTROL will detect any undigits in those addresses sent to it and set the appropriate STATE Toggle. However, the XM detects undigits in addresses in all other cases)
2. Limit errors on memory read/writes (READ/WRITE CONTROL will detect limit errors and set the appropriate STATE Toggle)
2. Invalid data (undigits in arithmetic data)
3. Invalid I/O Descriptor
4. Literal not allowed in the instruction (XM simply calls error routine at instruction literal entry point)

5.2 EXECUTE (XM) PHASE (Continued)

5. Instruction timeout in programmatic infinite loops (i.e., SLL) or extremely long instruction executions (i.e., MVD, SEA)
6. Privileged instructions in non-privileged base (i.e., BRE, IIO)
7. Non-MOD 2 address, improper undigits in address, or limit error in taken branches (i.e., LSS, OFL, NTR, EXT, BCT, BRE, BUN)
8. Other opcode-specific errors (i.e., invalid data type, invalid BCT addresses)

Memory errors detected by the MEMORY Modules are retained by the ERROR RECORDER in READ/WRITE CONTROL until the XM requests the information for the Memory Error Report. There are five STATE Toggles "ORR-ed" into the external interrupt line to notify the XM that a multi-bit memory parity error has ocurred. Four of the multi-bit memory parity error STATE Toggles are set on memory read errors depending upon the requestor. The fifth multi-bit memory parity error STATE Toggle is set on partial memory word write errors. Another STATE Toggle, MEMORY ERROR REPORT REQUIRED Toggle, signifies that either FETCH or an XM received a single-bit error which was corrected, or that the IOP received either a single-bit or multi-bit error. In this latter case, a memory error report will be written to memory at the next Interrupt BCT.

6 I/O INTERFACE

6.1 IOP MEMORY SCRATCHPAD

The IOP Memory Scratchpad is located in a portion of main memory relative to the Interpreter Base (absolute base) but below the MCP Base. Since this memory is below MCP base, the MCP code is unaware of its existence.

The IOP Memory Scratchpad contains 64 32-digit entries for maintaining channel and I/O information. The address of an entry (Interpreter Base-relative) for a particular channel equals the channel number times 32 plus 400.

Each entry contains the following information.

a. Current buffer address (8 digits)
b. Buffer end address (8 digits)
c. Extended R/D (8 digits)
d. IOP Use only (accumulated R/D) (4 digits)
e. Channel Busy Flag (1 digit)
f. Unused (3 digits)

In addition the 32 digit entry for channel 8 (the IOP channel) is used for the mailbox interface between the IOP and the XM's. This mailbox entry is defined as follows:

a. Channel number for IIO (2 digits)
b. I/O descriptor command (6 digits)
c. I/O descriptor 'A' address (8 digits)
d. I/O descriptor 'B' address (8 digits)
e. I/O descriptor 'C' address (8 digits)

If the I/O descriptor command is not in the 60-69 range, then the "C" address is left-justified into the "C" address mailbox field with two trailing zeros.

6.2 IIO, RAD, AND I/O COMPLETE HANDLING

The XM-IOP interface consists of the IOP Memory Scratchpad, the I/O REQUEST STATE Toggle, and the I/O COMPLETE STATE Toggle. The XM has the responsibility for detecting all errors which cause a Processor R/D and several address errors which are reported to the I/O channel R/D. All other errors are detected by the IOP or DLP's and are reported by the IOP to the I/O channel R/D.

For the RAD instruction the XM must verify that the specified channel is valid and not busy. If so, it moves the appropriate data either to or from the channel entry in the IOP memory scratchpad depending upon the RAD variants.

The IIO instruction is performed as follows:

a. The XM verifies that the address of the I/O descriptor contains no undigits and is MOD 2. If an error is found, then an ADDRESS ERROR Processor R/D is written. However, if the I/O descriptor address has undigits or is beyond the limit, an INVALID I/O Processor R/D may also be written, depending upon the unpredictable data read from memory.
b. Next the XM checks for an invalid channel number, a busy channel, and an invalid I/O descriptor opcode. If found, it writes an INVALID I/O Processor R/D.
c. The XM examines the I/O descriptor for undigits in the 'A' or 'B' addresses of the I/O descriptor, the 'A' or 'B' addresses of the I/O descriptor not being MOD 2, and the 'A' address of the I/O descriptor not being less than the 'B' address of the I/O descriptor. For any of these conditions the XM writes an ADDRESS ERROR I/O channel R/D (C002).
d. The XM must wait until the I/O REQUEST STATE Toggle is reset.
e. The XM then marks the appropriate channel busy.
f. For standard I/O, the XM then sets up all of the I/O descriptor information in the IIO mailbox entry. For datacomm I/O, it moves only the channel number (09) and places the IOCB address in the 'A' address field of the mailbox entry.

6.2 IIO, RAD, AND I/O COMPLETE HANDLING (Continued)

g. The XM then waits until the memory writes are complete. If any multi-bit memory parity errors occur, then the IIO is terminated. Otherwise the XM sends a bus transaction to set the I/O REQUEST STATE Toggle.
h. The IOP then notices the I/O REQUEST signal, moves the data from the mailbox entry, and resets the I/O REQUEST signal.
i. The IOP then performs the I/O operation.

The I/O Complete interface is as follows:

a. When an I/O operation is completed, the IOP stores the R/D in the appropriate channel R/D area and resets the channel busy flag for that channel in the Scratchpad entries.
b. Lastly, the IOP sets the I/O COMPLETE STATE Toggle to signal the presence of this interrupt condition to the XM's.

7 MAINTENANCE PROCESSOR/OPERATOR INTERFACES

7.1 SYSTEM INITIALIZATION

First the processor and I/O cabinets must be powered on. Second the PANEL or PANDLP floppy must be inserted into the floppy drive and the SMC cleared. Next the FM, XM, and IOP control store must be loaded via a "LOAD CS" command. To load the uniline firmware, one should clear the DLP base, execute the UNILINE FIRMWARE load program and then check the green LED on the DLP to see if it loaded properly. One may then halt/load by typing "CLEAR MEM ABS;CLEAR;LOAD MCP".

A "LOAD MCP" or "INIT" command clears the machine state, sets up the values in BASE/LIMIT Table then sets up the rest of the processor as if a pipeline clear had just occurred, and starts issuing clocks. A "PA" command is similar except that the STATE Toggles and BASE/LIMIT Table are unchanged.

7.2 HALT INSTRUCTIONS

The HBK (HALT BREAKPOINT) and HBR (HALT BRANCH) instructions both cause the processor to halt depending upon the circumstances. Once an XM has detected one of these instructions which is to halt the machine, it waits for the SEND signal, sends a READ STATE Toggle request and waits for the data (to make sure any writes are out of the A&W BUS input queue). Then the XM issues another READ STATE Toggle request to see if any previous memory writes had errors. If so, an Interrupt BCT is performed reporting that previous instruction address. If no previous WRITE error then, it may safely issue a SET STATE TOGGLE command to set the PROHALT STATE Toggle. Once the MP detects the PROHALT signal, it performs a PROCESSOR CLOCK STOP (i.e., stops the clocks only to FETCH, XM's, and READ/WRITE). After issuing the SET PROHALT tansaction, the XM loops waiting for the PROHALT signal to be reset by the MP. The MP resets PROHALT before starting the clocks again so that the XM knows that the halt is over. Now the XM determines the next program address according to the particular halt instruction and starts the pipeline clear function.

In the cases where PROHALT is set (i.e. HBK, HBR), the MP finds out the data about the instruction from the FETCH PAGE being executed by the XM with the SEND signal and not the FETCH PAGE being pointed to by the FETCH module.

7.3 INSTRUCTION TIMEOUT

The instruction timeout timer is used to determine when an instruction is caught in an infinite loop or performing some unreasonably long variant of an instruction. This timer is cleared everytime FETCH CONTROL signals that another FETCH PAGE has been filled with a new instruction. This timer function has two phases: soft instruction timeout (TIMEOUT=1) and hard instruction timeout (TIMEOUT=2). After the first 4 seconds the TIMEOUT=1 STATE Toggle is turned on, thereby allowing the FETCH CONTROL and XM micro-code to test this toggle in certain possible infinite or long loops to detect instruction timeout themselves.

If the timer runs for an additional 4 seconds, then the TIMEOUT=2 STATE Toggle is turned on which notifies the MP. The MP assumes a hardware failure and stops the processor.

7.4 SNAP PICTURE REPORTING

There are effectively three modes for SNAP Picture Reporting: Operational, Diagnostic Functional, and Diagnostic Maintenance.

The Operational SNAP occurs whenever the processor detects a fatal hardware error (XBROKER, IOERROR signals).

The Diagnostic Functional SNAP is enabled by the the SNAP Enable Toggle which is set/reset by the MCP, and occurs then whenever an instruction error, address error, or instruction timeout occurs. The MAINTENANCE Processor saves off a SNAPSHOT of the machine state in a reserved space of main memory. The INSTRUCTION ERROR SNAP TOGGLE and ADDRESS ERROR SNAP Toggle are "ORR-ed" together to form this "SNAP-TIME" signal to the MP. (Instruction Timeout=1 is signified by setting both the Address Error-Snap and Instruction Error-Snap State Toggles.) Once this signal has been detected by the MP, it must shift out strings to examine which toggle was on. If the INSTRUCTION ERROR SNAP Toggle or ADDRESS ERROR SNAP Toggle was set, then the MP simply takes the SNAP Picture, sets the SNAP PICTURE REPORT TAKEN STATE Toggle, and then starts the processor clock again. The XM writes the Processor R/D and performs an Interrupt BCT.

7.4 SNAP PICTURE REPORTING (Continued)

The Diagnostic Maintenance SNAP is under control of the Maintenance Processor and can be set up to take a SNAP Picture whenever some condition is detected by the MP stop logic. It may be a non-recoverable halt within two clocks of the condition or a "graceful halt" (DRYUP) within about eight clocks of the condition.

7.5 OPERATOR STOP/SINGLE INSTRUCT

These functions are controlled by the MAINTENANCE Processor working in cooperation with the FETCH module. When the MP receives an OPERATOR STOP or SI command, it issues an SI signal to FETCH CONTROL which the FETCH FORMATTER detects just before passing its current instruction to an XM. Once detected, the FORMATTER loops until the XM's are empty and then issues a READ STATE Toggle command and waits for the data. This guarantees FETCH that all writes have been received by WRITE CONTROL and the data has been written. Now FETCH sends an SI-time signal back to the MP, which then stops the processor. FETCH microcode has several delay clocks waiting for the halt before passing its current instruction to the XM for execution. Then FETCH waits for the XM to finish the instruction before re-fetch the next instruction from memory.

7.6 OPERATOR CLEAR, TERM

(To be specified)

7.7 CHAINS

(To be specified)

7.8 PROCESSOR RUN MODES

The processor has the following three run modes:

a. Overlapped Mode
b. Non-overlapped Non-stop Mode
c. Non-overlapped SINGLE INSTRUCT Mode

Overlapped mode is the normal, fully overlapped, full speed mode.

Non-overlapped NON-STOP mode is essentially the sequential FETCH and EXECUTION of an instruction. Non-overlapped SINGLE INSTRUCT mode is the same as non-overlapped NON-STOP mode execpt that there is operator intervention via the MP after the FETCH phase.

In either non-overlapped mode the MP signals the FORMATTER via the MSIMODE line that non-overlapped SINGLE INSTRUCT is needed. The FORMATTER notices this signal just before passing its next instruction to an XM. Next the FORMATTER waits for the pipeline to empty and all memory writes to be received by WRITE CONTROL. It then signals the MP via the FSIFLAG line that the SINGLE-INSTRUCT stop point has been reached.

In the non-overlapped NON-STOP variant the MP simply ignores the FSIFLAG, never stopping the processor clock. In the SINGLE INSTRUCT variant the MP stops the processor clock, signals the operator, waits for the operator's response, and then starts the processor clocks again. In either case the FORMATTER then passes the current FETCH PAGE to the XM and waits again for the XM to finish and all memory writes to be received by WRITE CONTROL. Then it clears the READER to read the next instruction from memory, parses the instruction into a FETCH PAGE, and continues as above. The advantage of this mode is that no instruction on a bad branch path is ever processed by the FORMATTER or XM, so that most STOP conditions found are perfectly valid. The only possible false STOP condition hits are due to hits on memory reads due to the READER fetching instructions ahead.

7.9 MAINTENANCE STOP CONDITIONS/PANEL

The MP accepts from one to nine STOP condition signals from the processor. If the "AND-ing" of these signals is true, then the MP knows that a STOP is to occur. The operator may specify a SYSTEM CLOCKSTOP, PROCESSOR CLOCKSTOP, PROCESSOR INSTRUCTION STOP, or PROCESSOR GRACEFUL STOP. A SYSTEM CLOCKSTOP will stop the system clock within two clocks of the condition but may not be recoverable due to I/O's being lost. A PROCESSOR CLOCKSTOP will stop the processor within two clocks of the condition but may not be recoverable due to processor memory reads or writes being lost. A PROCESSOR GRACEFUL STOP will stop the processor within about eight clocks of the condition and is completely recoverable by restarting the processor clock. An PROCESSOR INSTRUCTION STOP will stop the processor after the FETCH phase of the next instruction in FETCH and the rest of the pipeline has emptied.

Unfortunately, in normal overlapped mode some CLOCKSTOP's and GRACEFUL STOP's can stop on bad branch paths due to branch prediction or interrupts. Also some INSTRUCTION STOP's may stop several instructions after the STOP condition due to the pipelining. If the desired STOP condition point still occurs in non-overlapped mode, then using this mode may solve most of the problems. In non-overlapped mode all STOP's halt only on good branch paths (except those few R/W STOP's which might halt on a memory read request by the FETCH READER while fetching ahead) and INSTRUCTION STOP's halt either on the instruction containing the stop condition or on the first instruction after it.

Each module performs any additional logic functions necessary to specify any STOP condition within its own module. See the Maintenance Panel Stop Logic Specification for these details.

7.9 MAINTENANCE STOP CONDITIONS/PANEL (Continued)

Following is a list of FETCH Module (FM) micro-PC STOP locations for stopping when the error is detected by the microcode.

	ERROR	ADDRESS
1.	Invalid Instruction	002
2.	Address Error	003
3.	Multi-bit Memory Parity Error	004
4.	Instruction Timeout	005
5.	Any of the above errors	006

Following is a list of Execute Module (XM) micro-PC STOP locations for stopping when the error is detected by the microcode.

	ERROR	ADDRESS
1.	Invalid Instruction	10C
2.	Invalid I/O Descriptor	110
3.	Instruction Timeout	115
4.	Invalid Data	118
5.	Address Error	119
6.	Multi-bit Memory Parity Error	11B
7.	Any of the above errors	11A

7.9 MAINTENANCE STOP CONDITIONS/PANEL (Continued)

The following table summarizes some of the useful INSTRUCTION STOP conditions and the state of the machine at the time of the stop.

STOP CONDITION	HOW DETECTED	NON-OVERLAPPED INSTR STOP	USEFULXM CLOCK STOP
1. Instr Addr (PC)	FETCH hdw	On instr	
2. Opcode Equal	FETCH hdw	On instr	
3. Priv. Opcode	XM u-addr	1st after	
4. Memory Addr	SMC hdw	1st after	
5. PRIVILEGED BASE	R/W hdw	On instr	
USER BASE	R/W hdw	On instr	
6. CONTROL State	R/W hdw	On instr	
NORMAL State	R/W hdw	On instr	
7. Instr Error	F. u-addr	On instr	
	XM u-addr (XM u-addr IS THE ONLY WAY TO STOP ON ALL INSTR. ERRORS)	1st after BCT	On instr
8. Addr Error	F. u-addr	On instr	
	XM u-addr (XM u-ADDR IS THE ONLY WAY TO STOP ON ALL ADDRESS ERRORS)	1st after BCT	On instr
	READ LIMIT FF	1st after BCT	
	READ UNDIG FF	1st after BCT	
	WRITE LIM. FF	1st after BCT	
	WRI. UNDIG FF	1st after BCT	
9. Memory Error	PROC M-BIT FF	1st after BCT	
	XM u-addr (Proc. multi-bit)	1st after BCT	On instr
	MEM ERR REP. REQUIRED FF	1 in. after	
10. Proc. Interrupt	XM u-addr	1st after BCT	On instr

7.9 MAINTENANCE STOP CONDITIONS/PANEL (Continued)

This table summarizes some of the useful GRACEFUL STOP conditions and the state of the machine at the time of the stop.

STOP CONDITION	HOW DETECTED	OVERLAPPED GRACEFUL STOP	NON-OVERLAPPED GRACEFUL STOP
1. Instr Addr (PC)	FETCH hdw	False stops	On instr.
2. Opcode Equal	FETCH hdw	False stops	On instr.
3. Opcode/Literal combination	XM u-addr	On instr. (1 XM only)	On instr.
4. Priv. Opcode	XM u-addr	On instr. (1 XM only)	On instr.
5. Memory Addr	SMC hdw	False stops	On instr.
6. PRIVILEGED BASE	R/W hdw	On instr.	On instr.
USER BASE	R/W hdw	On instr.	On instr.
7. CONTROL State	R/W hdw	On instr.	On instr.
NORMAL State	R/W hdw	On instr.	On instr.
8. Instr Error	F. u-addr	On instr	On instr
	XM u-addr (XM u-ADDR IS THE ONLY WAY TO STOP ON ALL INSTR. ERRORS)	On instr.	On instr.
9. Addr Error	F. u-addr	On instr	On instr
	XM u-addr (XM u-ADDR IS THE ONLY WAY TO STOP ON ALL ADDRESS ERRORS)	On instr.	On instr.
	READ LIMIT FF	False stops	On instr.
	READ UNDIG FF	False stops	On instr.
	WRITE LIM. FF	On or after	On instr.
	WRI. UNDIG FF	On or after	On instr.
10. Memory Error	PROC M-BIT FF	False stops	On instr.
	XM u-addr (Proc. multi-bit)	On instr.	on instr.
	MEM ERR REP. REQUIRED FF	On occurrence	On occurrence
11. Proc. Interrupt	XM u-addr	On instr.	On instr.

7.10 SINGLE CLOCK

The SYSTEM CLOCK is the only clock that may be single-clocked and still guarantee proper system operation.

8 OMEGA ARCHITECTURE CONSIDERATIONS

T.B.D.