UNISYS CORPORATION
Entry/Medium Systems Group

COMPANY CONFIDENTIAL
Pasadena Development Center

DISTRIBUTION

SDS No. 1993 5279 Title: V500 ARCHITECTURE

Revision: A Release Date: March 24, 1987

6090 J. Chen
6085 M. Emm
6143 N. Fleming
6100 R. Okimoto
6095 M. Rooke

59 Pgs.

1993 5279

Burroughs Corporation
ENTRY/MEDIUM SYSTEMS GROUP
PASADENA PLANT

V500 ARCHITECTURE

SYSTEMS DESIGN SPECIFICATION

COMPANY CONFIDENTIAL

REVISIONS

REV. LTR	REVISION ISSUE DATE	PAGES REVISED, ADDED, DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A		Initial Issue	L. Simpson	M. Rooke
	03-26-87	ECN 51274-A	M. Rooke	J. Chen 3-11-87
				M. Emm 3/23/87
				R. Okimoto 3/23/87

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

1 SCOPE

This engineering specification describes the hardware architecture of the V500 multiprocessor system.

2 RELATED DOCUMENTS

1993 5212	V500 Fetch Module
1993 5204	V500 Execute Module
1993 5253	I/O Memory Concentrator
1993 5220	V500 Memory Control And Cache Module
1993 5303	V500 Maintenance Subsystem
1993 5238	V500 Memory Data Card

3 GENERAL DESCRIPTION

The V500 is a high performance V-Series computer that is instruction-set-compatible with past Burroughs B2/3/4000 and V-Series computers. The performance goals of the V500 are:

- An RPM range of 250-1290.
- Multi-processing capabilities. This proposed hardware architecture permits up to 4 processors per system, sharing the same main memory and same peripherals.

3.1 OVERVIEW OF V500 ARCHITECTURE

The V500 System Architecture supports up to four processors in a tightly coupled multiprocessor configuration.

The V500 System Architecture is built around the following basic modules (see Figure 3-1).

Instruction Fetch (IF)

Operand Fetch (OF)

Execute Module (XM)

Memory Control And Cache Module (MCACM)

Memory Data Card (MDC); also called the memory module.

I/O Memory Concentrator (IOMC)

Data Transfer Module (DTM)

Maintenance Processor (MP)

System Maintenance Controller (SMC)

The Instruction Fetch (IF) module performs the following functions:

- Reads the raw instruction from memory (or cache).
- Parses and formats the instruction into syllables.
- Predicts the direction to be taken at conditional branches.

The Operand Fetch (OF) module performs the following functions:

- Resolves operand addresses for extension, indexing, and indirection as the instruction is parsed
- Resolves operand indirect field lengths.
- Maintains lock status for overlapped pipelined fetch/execution.

3.1 OVERVIEW OF V500 ARCHITECTURE (Continued)

- Directs IF on predicted taken/unconditional taken branches which have indexing/indirection on branch syllable.
- Directs IF on local environment procedure change.
- Sends parsed instruction and resolved operand addresses and lengths to the Execute Module.

The Execute Module (XM) does the following:

- performs the instruction function on the operand data
- performs the hardware call and interrupt functions when needed

The Memory Control And Cache Module (MCACM) does the following:

- read/write functions
- memory error detection and correction (ECC)
- base addition
- limit checking
- data rotation and concatenation

The memory modules are on memory data cards (MDC). Each MDC contains one memory module. The MDC is accessed by a single address/data bus. A memory word is 160 bits wide (4 processor words), plus the error correction bits.

The Data Transfer Module (DTM) provides a single I/O interface for the V500 I/O subsystem. It is responsible for transferring data between memory and the I/O interface.

There is no direct relationship between the number of processors, Data Transfer Modules, DLPs, and the number of peripherals. These numbers are determined by the I/O performance and system reliability (redundancy) required by the customer.

The I/O subsystem and the V500 memory require different word lengths. The I/O Memory Concentrator (IOMC) converts the words to the proper length, and aligns the data for transfers between the I/O subsystem and memory.

3.1 OVERVIEW OF V500 ARCHITECTURE (Continued)

The Maintenance Processor (MP) handles the operator interface to all of the hardware elements of the V500 processor; it is the vehicle for troubleshooting problems in the processor. The MP is also used for system initialization. The operator interface to the MCP is via a separate operator display terminal (ODT).

The System Maintenance Controller (SMC) connects the Maintenance Processor to the V500 processor, memory, the I/O subsystem, the Environmental Control Module (ECM), and the DLP Test Bus.

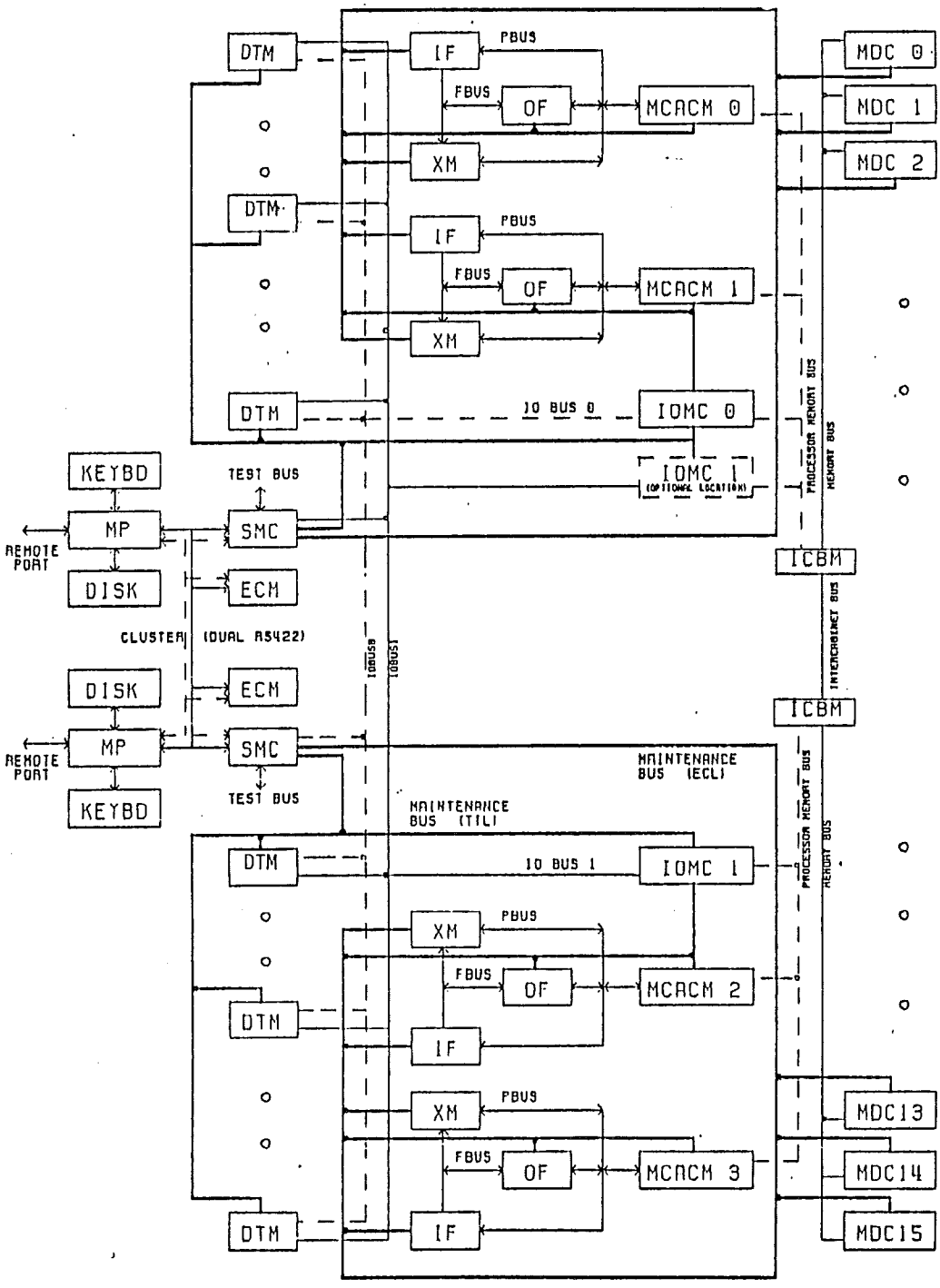


Figure 3-1 V500 System Architecture

3.2 MODULE FUNCTIONS/INTERFACES

The following sections describe the individual modules and their interfaces.

3.2.1 INSTRUCTION FETCH MODULE

The Instruction Fetch Module (IF) is the first stage of the Fetch pipeline (Figure 3-2). It contains the Reader and the Formatter.

The Reader reads instructions from memory and places them into the Fetch Read Queue (RQ). The Formatter then takes the data out of the RQ and parses it into syllables, which are then loaded into the Instruction Queue (IQ). The FBUS Controller in IF takes these parsed syllables from the IQ and passes them (via FBUS1) to the Operand Fetch module (OF) and the XM Fetch Pages.

The Formatter is also responsible for redirecting the Reader down the "taken branch instruction path" for those local environment branch instructions considered "taken". In this "taken branch" case, the Formatter still saves the first instruction on the not-taken path just in case the branch was mis-predicted. This allows the Formatter to send the the first instruction on the correct path to the XM sooner on mis-predicted "taken branches".

The FBUS Controller counts SHOVE signals from the OF, and POP_FP signals from the XM to determine if a fetch page is available. The SHOVE and POP_FP signals indicate that the OF and XM have finished processing their current fetch page (the fetch page is a storage structure that captures data from the IF module). The IF cannot get more than one fetch page ahead of the OF due to an OF fetch page limitation.

3.2.1 INSTRUCTION FETCH MODULE (Continued)

The IF provides the initial values to the OF and XM for the following syllables (if appropriate for the instruction):

Information provided by IF to XM and saved by OF:

CLOCK	FBUS1	FADDR
clock 1	OP_SYL	0
clock 2	A_$\overline{\text{SYL}}$	1
clock 3	B_$\overline{\text{SYL}}$	2
clock 4	C_$\overline{\text{SYL}}$	3
clock 5	$\overline{\text{PC}}$	4

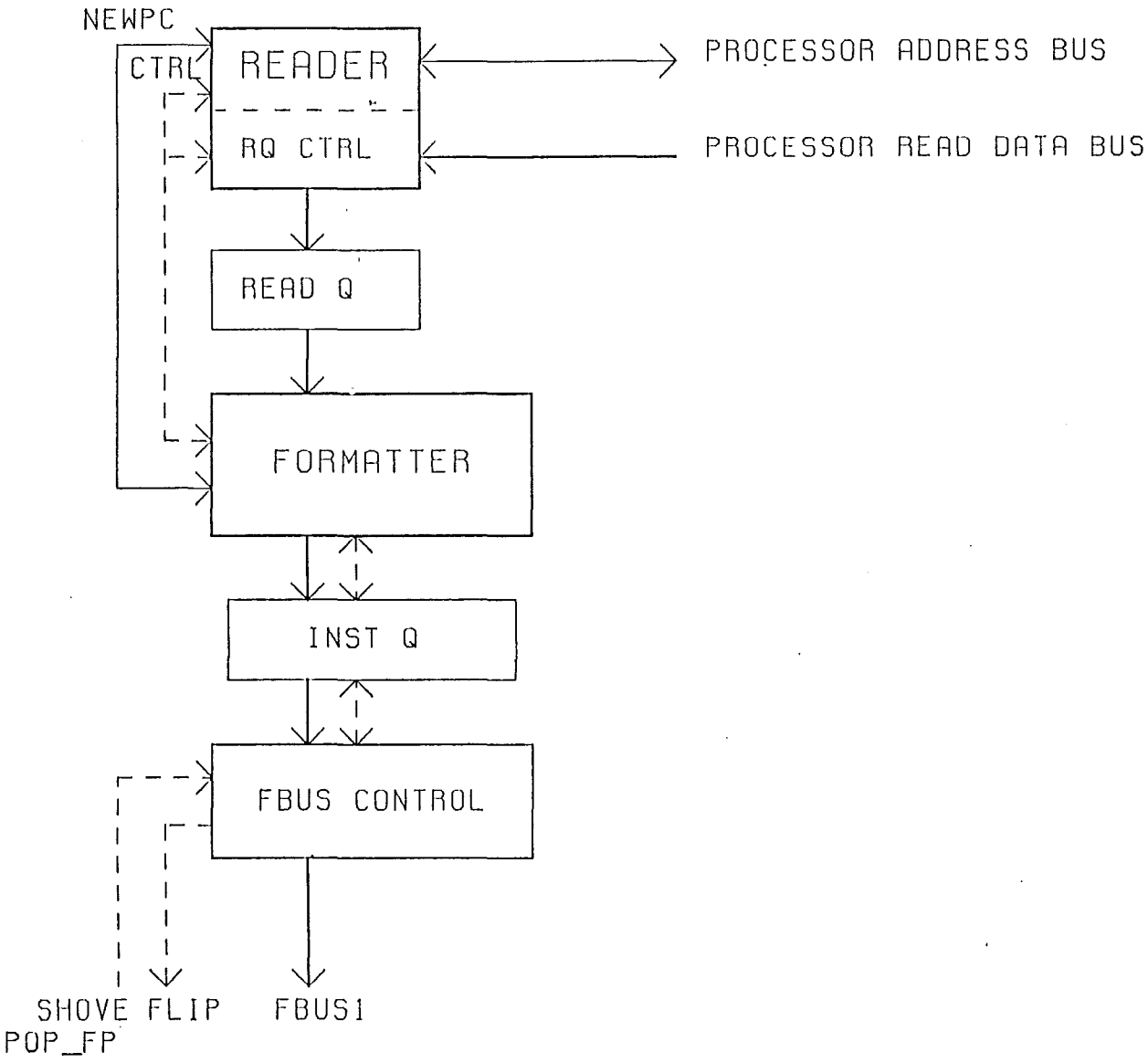


Figure 3-2 Instruction Fetch Module

3.2.2 OPERAND FETCH (OF) MODULE

The Operand Fetch module is the second stage of the 2-stage Fetch pipeline. The OF module contains a Formatter, a Lock Unit, and an OPLEN Unit.

The major functions of the OF are to ensure code integrity, resolve indexing and indirection, and to prefetch data operands for the XM. OF requests a code check (in the area specified by the current instruction's PC plus 90) to ensure that there is no code modification.

OF also resolves indirect field lengths, and indexed or indirect addresses. As the instruction is being parsed, OF sends the OPLIT, FLAGS, ALEN, BLEN, and CLEN values (if required) to the XM (via FBUS2).

If the instruction performs any memory writes, write lock requests for that area are sent to the Lock Unit containing the begin address and end address.

3.2.2 OPERAND FETCH (OF) MODULE (Continued)

OF pre-reads some of the read data operands required by XM (usually up to 10 digits per operand). To inform the XM of the validity of the pre-read operands, OF sends a data-hit bit to the XM after performing a Lock Unit data check against the read operand's addresses. OF finishes the instruction by sending the SHOVE signal to XM, thus indicating that another fetch page is available.

The following information will have been received by XM before the the fetch page valid information is set:

ADDRESS	FBUS1	FBUS2
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	

The following information is provided on FBUS1:

```
OP_SYL      = xxxx op af bf    resolved AF, BF
            = xxxx 1E nn op    for invalid instruction
            = xxxx 1A nn op    for address error
            = xxxx 1C xx op    for code modified instruction

A_SYL       = 111111 0000      A-operand literal

A,B,C_SYL   = b 00 aaaaaaa
              b               = base indicant
              aaaaaaa         = resolved operand begin address
              1 000 aaaaaa    = OMEGA branch ops address_SYL
              000 aaaaaaa     = MCP IX branch ops address_SYL
              xx v xxxxxxx    = error during IA/IX in branch
                                address resolution
                                (V is non-zero if error has
                                 occurred)

PC          = 1 000 pppppp     OMEGA
            = 000 ppppppp      MCP IX
                               (ppppppp = program counter
                                 value)
```

3.2.2 OPERAND FETCH (OF) MODULE (Continued)

LENGTH_SYL = 000 1111111 for operand length >200 digits

SPECIAL_SYL = xxxxxxxxxx for passing information between OF and XM

XM receives the following syllables on FBUS2:

OPLIT = 10 bit encoded field in which:
bit 9 = Fetch event - SI
bit 8 = reserved
bits 7-0 = 8 bit OPLIT vector (ALIT = LSB)

ALEN_SYL,
BLEN_SYL,
CLEN_SYL = 111; length of operand in digits (except 0 => 400 digits)

FLAGS = 10 bit encoded field in which:
bits 9-8 = number of extended address
bit 7 = AF > BF
bit 6 = AF = BF
bits 5-4 = A address controller
bits 3-2 = B address controller
bits 1-0 = C address controller

The Formatter handles all indexing, indirection, or exceptions that are not covered by the IF. After resolving an address, it rewrites that entry in the XM fetch page, and may flush the IF if it resolved a "taken branch" address. It notifies the XM that a fetch page is available by sending the SHOVE signal.

The OPLEN Unit analyzes the OPcode, AF, A address controller, BF, B address controller, and C address controller to provide the OPLIT vector, ALEN, BLEN, CLEN, AF>BF flag, AF=BF flag, and OPTIMAL case flag.

The Fetch Page is a storage structure that captures the data sent from the IF module via FBUS1. There are 2 Fetch Pages, with each page divided into eight 48 bit entries. The use of 2 pages allows IF to work on one instruction in one page, while OF is working on the previous one.

3.2.2 OPERAND FETCH (OF) MODULE (Continued)

The Fetch Pages are implemented as a single read/write register file, capable of reading and writing different locations in the same clock cycle. Data is driven onto the ABUS.

The Lock Unit contains a table of 8 begin/end address pairs of pending writes to memory. An additional hardwired table of fixed relative addresses for the index registers is also provided. The Lock Unit detects data interlocks in the instruction pipeline by comparing code and data begin/end address pairs against the memory write locks in its table. Because the IF module can get up to 3 instructions ahead of the XM, a Lock Unit entry is removed only after it has received the OP Complete signal from the XM, and three subsequent code check requests.

For performance reasons, the V500 always maintains the current copy of IX4 - IX7 in its internal scratchpad, and attempts to maintain a copy of IX1 - IX3 when feasible. In addition, for certain instructions that are modifying an index register, it reads the new value of the index register from memory and pre-caches the new value before the OF knows whether this instruction will be executed. This helps eliminate wasted clocks due to the frequent index register data interlocks occurring on index register loads.

For IX4 - IX7, the pre-cached values become the current valid values when the XM sends the LIX_OK signal to notify the OF that the LIX instruction execution is starting. Because IX1 - IX3 are changed in memory by a variety of instructions, they cannot use this interface, and are not always valid in the OF scatchpad. In any case, a pipe flush occuring first will always invalidate any pre-cached value of an index register.

The OF prefetches the operand by issuing memory reads of the operands for the XM (usually up to 10 digits). The OF, however, directs the MCACM to return the operand data back to the proper XM Operand Queue for the instruction, rather than back to the OF. This increases the OF instruction throughput by allowing it to start on a second instruction, without waiting for operand data for the first instruction to be returned.

3.2.3 XM MODULE

XM executes the instruction according to the fetch page and other registered information passed from the IF modules. As shown in Figure 3-3, it uses the operand prefetch memory data from the Operand Queues (memory reads previously initiated by the the Fetch modules).

Further memory reads are performed by the XM; the memory data is returned to the RBUS Queue. Memory read data is requested by address and length (up to 10 digits), and returned left-justified with zero-fill.

The results of the instruction execution are sent to the MCACM over the AWBUS. The accumulator, Comparison toggles, and Overflow toggle are in the XM.

When the XM finishes writing in an area of memory that is protected by a memory write lock, it sends the OP_COMPLETE signal to the OF. When it completes an instruction, it sends the POP_FP signal to the IF to indicate that another fetch page is available.

The XM contains a "smart requestor" interface to the AWBUS to align consecutive memory write requests into 10 digit word-oriented requests for to the MCACM.

If the instruction needs an operand wider than 10 digits, the XM issues multiple read commands to the MCACM to obtain 10 digit chunks of the operand. The read address of each chunk need not align with the memory module boundary because the MCACM takes care of the cross module read and returns the data in one piece.

To write to the cache memory, the XM Write Requestor is given an address, followed by multiple chunks of data. The XM Write Requestor then issues multiple write commands to the MCACM. Each write command can write up to 10 digits to the memory. The write address must match the MOD 10 boundary.

When the XM completes an instruction and has written the entire result to memory, it sends the OP COMPLETE signal to the OF, thus causing it to pop the write lock from the LOCK UNIT. The data is written directly to the cache only since the cache is a write-back cache. All the write operations are handled as read-modify-write.

3.2.4 MEMORY CONTROL AND CACHE MODULE

The Memory Control And Cache Module (MCACM) provides the read and write functions for the IF, OF, and XM for up to 500 MB of main memory. It has a write-back cache memory of 81920 digits, divided into 1024 sets of 2 blocks each. There are 40 digits per block. MCACM monitors the memory address bus so that it can coordinate its cached data with the memory modules and other MCACM caches.

In a write-back cache design, the main memory does not necessarily contain a valid copy of a cache element, as opposed to a write-through design, in which main memory always maintains a valid copy.

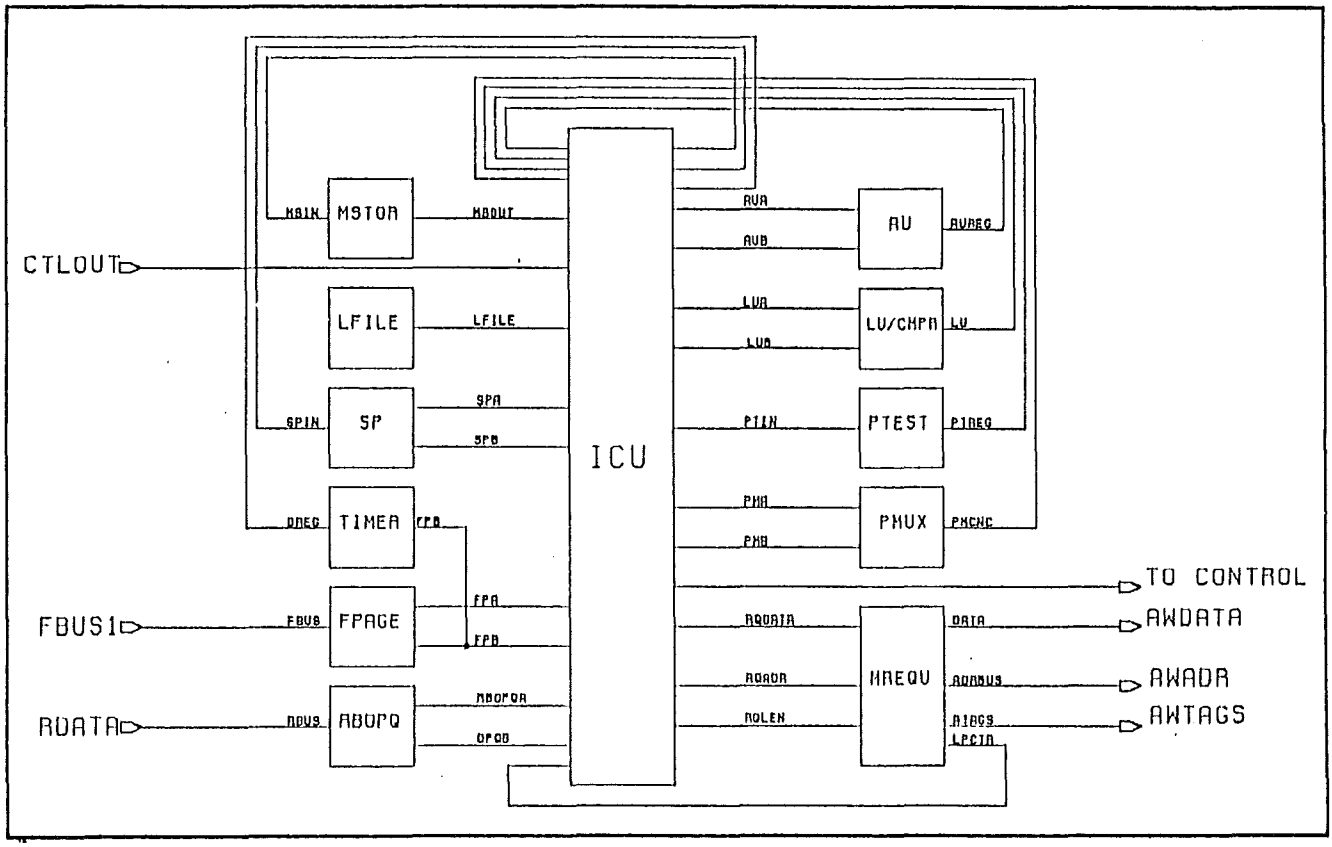


Figure 3-3 XM Data Section

3.2.4 MEMORY CONTROL AND CACHE MODULE (Continued)

The data paths to the memory modules are 40 digits (160 bits) wide; the data paths to the IF and XM are 10 digits (40 bits) wide. Due to the digit addressability of the V-Series architecture, the MCACM also provides alignment functions on read data requested by the IF and XM.

The MCACM consists of the address section and the data section.

The address section (Figure 3-4) receives memory operation requests from the processor modules, performs base addition and conversion from BCD to binary, performs limit checking, and makes and controls system memory requests.

Memory requests from the processor modules are placed into the Input Interface, (a FIFO state machine), from which the incoming memory requests are processed in the order received. The type of memory request is indicated by a command code.

The memory command codes are described in the V500 Memory Control And Cache Module specification (1993 5220).

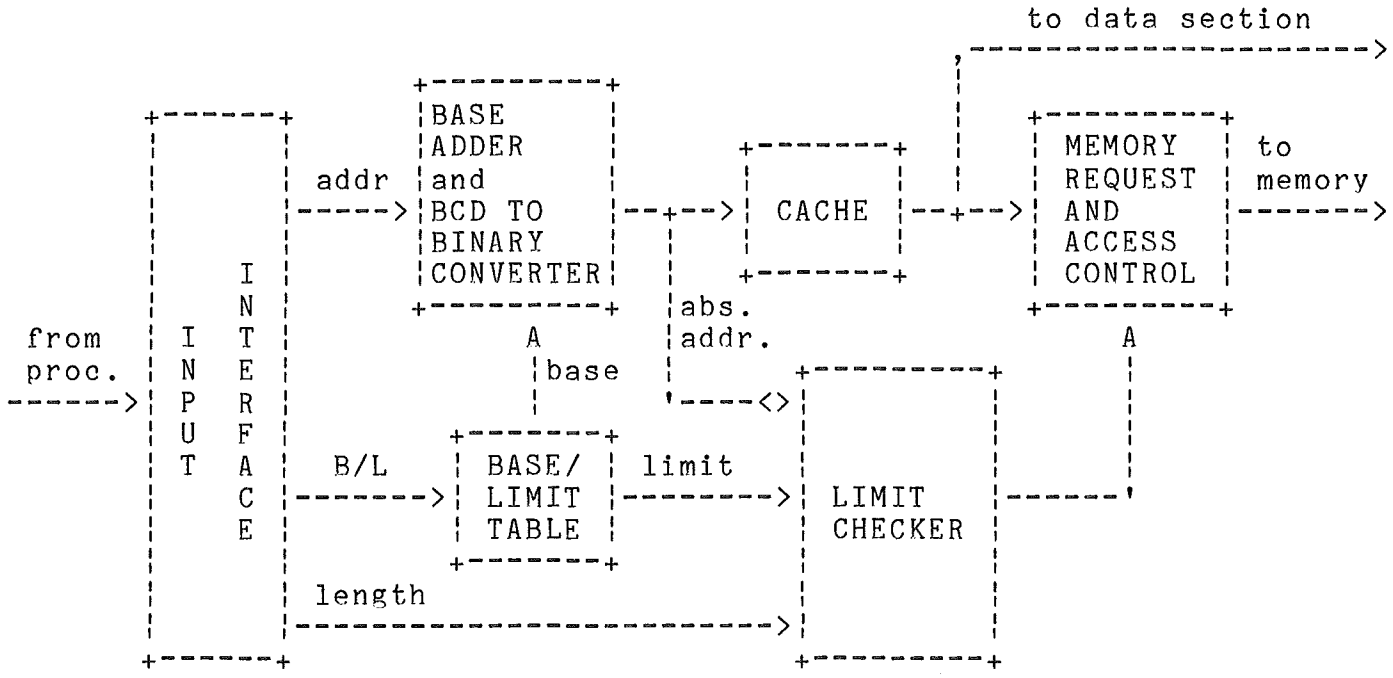


Figure 3-4 Memory Control Address Section

3.2.4 MEMORY CONTROL AND CACHE MODULE (Continued)

The command codes sent across the AW Bus are:

CODE	COMMAND
00	Reserved
01	Read Data
02	Write I/O
03	Read I/O
04	Test Error
05	Read Base
06	Read ECC
07	Read Limit
08	NOP 0
09	NOP 1
0A	NOP 2
0B	NOP 3
0C	Flush 0
0D	Flush 1
0E	Flush 2
0F	Flush 3
10	Write PC
11	Read PC
12	Read Error Address
13	Read Error Report
14	Disable Cache
15	Enable Cache
16	Read With Lock
17	Reserved
18	Write Back
19	Write Base
1A	Write ECC
1B	Write Limit
1C	Reserved
1D	Write Data
1E	Read Uncorrected
1F	Reserved

3.2.4 MEMORY CONTROL AND CACHE MODULE (Continued)

The memory request command code accompanies the memory request. The request includes:

- o memory data field (read or write)
- o memory address field (begin address)
- o length field (number of digits in the operation)
- o tag field (links MCACM to requesting processor)

The address portion of the request is converted to binary and added to the base (indicated by the base indicant) to form the absolute address. The absolute address is sent to the cache, where it is checked against the limit for the indicated base, and then to memory request and access control, which selects the memory data cards.

The data section of MCACM handles full or partial read or write data that has been requested by the address section. Specific data section functions include:

- o receive data from memory
- o generate ECC for data written to memory
- o correct single bit errors
- o detect and reports multiple bit errors
- o rotate and concatenate data read from memory
- o interface the MCACM and the processor modules

The Cache module (Figure 3-5) contains a Cache Memory, a Cache Control, a Base Add/Limit Check Unit (which also performs undigit detection for addresses), and a Spy unit.

The Cache Memory is an 81920 digit set-associative, write-back cache, organized as 1024 sets, with each set containing two 40-digit blocks.

A set-associative cache contains n sets, with each set containing m blocks of i digits each. A key (k) is stored with each block.

3.2.4 MEMORY CONTROL AND CACHE MODULE (Continued)

The memory address (in binary form) is separated into the following fields:

m set address

k key

i digit offset within the block

A look-up into the cache is made by directly addressing the set address (m), and by comparing all of the keys stored in the set to the incoming key field of the binary address. If the incoming key field matches one of the keys stored in the set, then the data is already present in the cache. If the data is not in the cache, the module requests it from memory.

If the data is in the cache, it will be read or written from/to the cache directly. Otherwise, if the original request is a read operation, then the data returned from memory is put into the "least recently modified" block in the set, and is sent back to the requestor. The "least recently modified" block in the set is the block which has remained unmodified the longer of the two blocks. If the operation is a write, the block to be modified is loaded from memory before the cache can be updated. If the cache location to be used is already in use by another memory block, which has been modified, then that modified block of data must be written back to memory before its cache location can be overwritten by the new data.

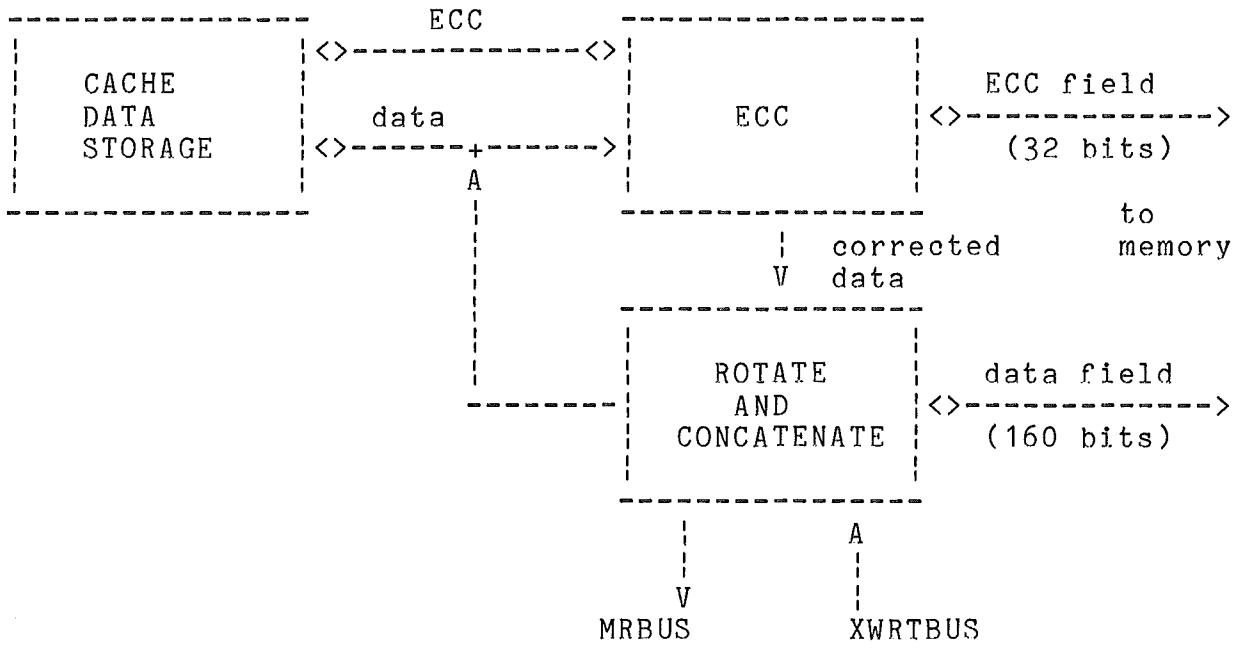


Figure 3-5 Cache Data Interface

3.2.4 MEMORY CONTROL AND CACHE MODULE (Continued)

Cache requests are received as absolute addresses from the MCACM (base addition and undigit checking was previously performed by the Address Section), and are processed in the order received, whenever the cache is free.

If the request being served generates a cache miss (requested data not present), the cache issues a memory request to MCACM, then starts serving the next (second) CM request. If the second CM request also generates a cache miss, the CM holds the request until the first miss completes. The CM then continues processing cache requests, but at a lower rate (one request every other clock).

If 3 successive misses occur, the CM stops processing requests until the first miss is completed.

If, however, the next CM request generates a cache hit, the CM will serve that request immediately, without waiting for the data of the last request coming back from memory. Thus, the data sent back from CM is not necessarily in the same order as the requests received. However, Read and Write to the same address will always be in order.

The 40 digit data block returned from memory is stored in the cache without rotation. The rotation is done by a Rotator in Memory Control before the data is sent to the IF, OF, or XM.

When a requestor makes a request, the CM tags that request, indicating the destination of the return data. The CM holds this tag.

One clock prior to completion of the cache/memory operation, the CM broadcasts the tag back to all modules. The module that is to receive the data recognizes the tag, and prepares to receive the data at the next clock. Any error information (such as limit error or parity error) is added to the tag. The destination module detects the error information in the tag and decides what to do with the data.

3.2.4 MEMORY CONTROL AND CACHE MODULE (Continued)

The format of the tag (without error information) is as follows:

Destination Module ID	FP/RBUS	Queue Address	Destination
10	0	nn0	XM,FP#nn,entry A
10	0	nn1	XM,FP#nn,entry B
11	0	nn0	XM,FP#nn,entry A
11	0	nn1	XM,FP#nn,entry B
11	1	mmm	XM,RBUSQ,word mmm
01	x	mmm	Reader,word mmm
10	x	mmm	OF,word mmm

In a multiprocessor configuration, only one cache can contain modified data that has not yet been written into memory. If a processor wants to update a piece of data, it must first get the data from another cache (or from memory), and invalidate all other copies in the other caches.

Either the memory, or any one of the caches, can be the source of data. If a processor wants to read a piece of data, the read request is examined by the Spy units of all caches, and by the MCACM. Either the MCACM, or the Spy of the cache containing modified data finds that it is holding the data being requested. Whichever unit (MCACM or Spy) is holding the data then captures the memory data bus, and sends the data to the requesting processor.

Each Cache Module contains a Spy which monitors activities of all memory requestors to insure that only a single copy of modified data is allowed to exist. For each type of memory request (whether from MCACM or IOMC) detected by the Spy for a particular cache, the cache does one of the following:

- return the data
- invalidate the block
- change the status of the cache block
- invalidate the cache block
- take no action

3.2.4 MEMORY CONTROL AND CACHE MODULE (Continued)

The action taken by the cache enforces the write-back algorithm, and insures that only one copy of modified data exists in a multiprocessor system. This is done independently by each cache in a multiprocessor system.

3.2.5 MEMORY MODULE

The V500 processor can handle up to 500 megabytes of memory. The MCACM supports 1-, 2-, or 4-way interleaving, and 1, 2, or 4 groups of boards interleaved 4-ways. For maximum performance in a multiprocessor configuration, at least 8 boards must be interleaved (in groups of 4). Each board contains a single module, which can be 1/4, 1/2, or fully populated (20 MB per fully populated board).

3.2.6 INPUT/OUTPUT SUBSYSTEM

The I/O subsystem consists of 1 or more Data Transfer Modules (DTMs). There is a Memory Concentrator (IOMC) on each processor bus (either 1 or 2). The IOMC provides a gateway from the DTMs to the memory subsystem through the Processor Bus(s) and the MCACM(s).

All the DTMs talk to the IOMC(s) via identical buses. Thus, any processor can use any DTM on the bus. A separate MCACM interfaces the I/O subsystem to the memory.

There can be several types of Data Transfer Modules (DTMs). Each communicates over a specific type of local bus or network. The initial DTM will support the MLI protocol.

The DTM is responsible for communication with the processor for initiation of I/O commands, for reporting the results of such commands, and handling the memory to I/O interface data transfer.

DTMs have different outboard interfaces, based on I/O device requirements. For those I/O devices connected to Data Link Processors, a Message Level Interface is supplied. Other interfaces (such as Ethernet) may be supplied by other DTMs. A single DTM does not provide multiple interface types.

The MLI DTM is similar to previous systems such as the B4900. DTMs with non-MLI interfaces are micro-controlled state machines with interface control provided by the appropriate industry-standard VLSI chip set.

A master DTM examines the I/O descriptor in the I/O mailbox and notifies the DTM connected to that channel.

3.2.7 MAINTENANCE SUBSYSTEM

The Maintenance Subsystem handles system initialization, debugging, and field maintenance. The subsystem includes the video monitor used for maintenance console functions. The architecture is modular, providing for single or multiprocessor configurations, and for fault tolerance.

During initialization, the Maintenance Subsystem verifies correct processor operation, then loads and initializes the MCP boostrap program from its dedicated Winchester disk file.

For debugging and field maintenance, the Maintenance Subsystem can access the internal state of the processor, and control clocking at the system and module level under operator or program direction. The Maintenance Processor software includes automatic system fault isolation procedures.

The Maintenance Subsystem (see Figure 3-6) consists of the following hardware modules:

1) Maintenance Processor (MP). A B27 workstation with a monitor and 1MB of RAM.

2) System Maintenance Controller (SMC). The SMC is a 80186-based single board microcomputer that provides the physical connectivity between the MP and the other modules in the system.

3) Maintenance Disk. A 40 MByte Winchester disk drive interfaced to the MP; used for storing all MP and SMC software and data files.

4) Environmental Control Module (ECM). This module monitors and controls system power and cooling at the cabinet level.

3.2.7 MAINTENANCE SUBSYSTEM (Continued)

A single Maintenance Processor provides the high level control functions for system configurations of one to four processors, via one or two SMCs connected to a local network. The SMC has enough intelligence to perform many tasks with a minimum of MP involvement, thus permitting distribution of the Maintenance Subsystem workload, and resulting in increased efficiency.

An optional second Maintenance Processor provides increased operational flexibility and fault tolerance.

The SMC shares a parallel interface with the DTMs to system memory via the IOMC. During normal operation, the SMC emulates a DTM on this interface, thus allowing the host system to communicate with the Maintenance Subsystem components as I/O devices (eg. for the ODT function). This interface is also used for memory testing.

Processor state is accessed by the Maintenance Subsystem via shift chains that link the state elements, and connect to the SMC. RAMs are indirectly accessed by transferring their contents in or out of registers on the shift chains.

The error detection logic in the processor produces module level error signals that can stop the module clock and interrupt the SMC. When an error signal is received by the SMC, an error handling procedure is initiated. This may include a recovery attempt via instruction retry. All errors are logged by the MP, and reported to the operating system.

Event Stop Logic is included in the processor modules to permit selective halts for debug (hardware and software), and for field maintenance. The occurrence of a selected stop event causes a module-level interrupt to the SMC, and may optionally halt the module in which the event was detected. The SMC brings the system to an orderly halt on receipt of the interrupt, and reports the interrupt to the MP.

The MP has an RS-232 port for the interface to the Remote Support Center. This allows all locally-available maintenance procedures, including mainframe power on and off, to be executed from the RSC.

3.2.7 MAINTENANCE SUBSYSTEM (Continued)

A Testbus interface on the SMC permits DLPs to be tested online or offline.

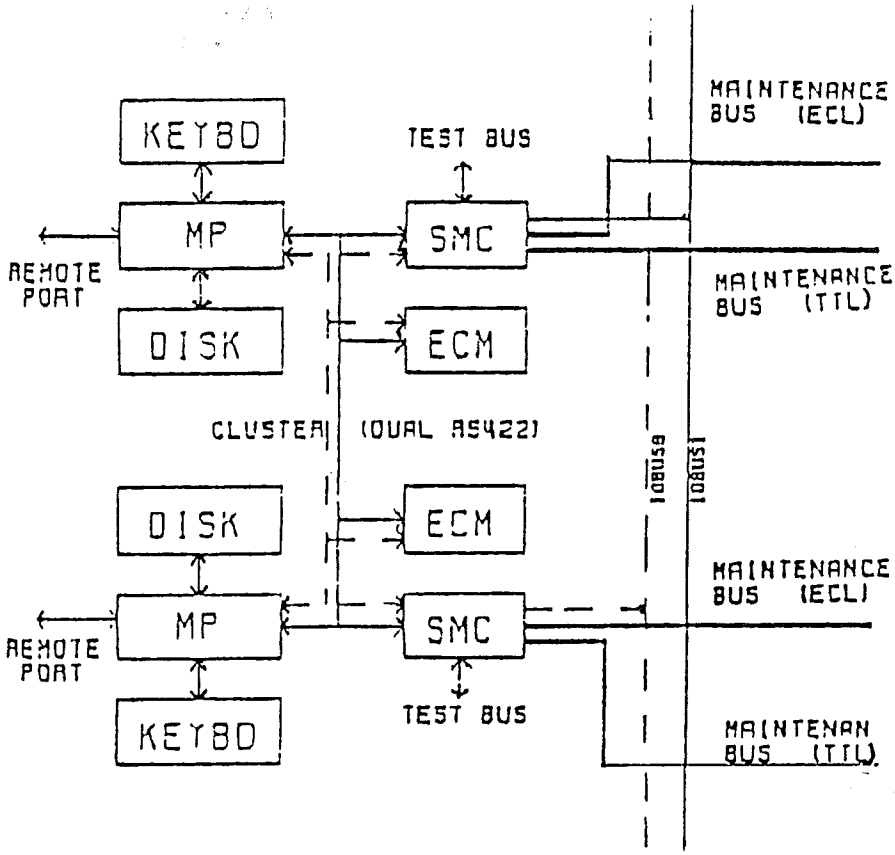


Figure 3-6 Maintenance Subsystem

3.2.8 INTERPROCESSOR COMMUNICATION

Multiprocessor activity is controlled by using a multiprocessor shared memory area, and the Interprocessor Communicate (IPC) instruction to be implemented in the future.

When a processor detects the IPC signal on the V500 IPC bus, it checks its Interprocessor Communicate flags in the multiprocessor shared memory area. If an IPC flag is set (by another processor), it executes an interrupt to perform the function. Otherwise it ignores the IPC signal and continues as before.

The multiprocessor shared memory area contains a common area, and a local region for each individual processor. Each local region contains the following flags:

Start Yourself
IPC Executed
Interrupt Idle

The Start Yourself flag causes a stopped processor to execute an interrupt to the MCP Kernal so that the processor may start itself. This flag is ignored if the processor is not stopped.

The IPC Executed flag causes a processor to execute an interrupt to the MCP Kernal so that it may perform an indicated function (stop itself). This flag is ignored if the processor is stopped or idle.

The IPC Idle flag causes a processor to execute an interrupt to the MCP Kernal if the processor is idle. The flag is ignored if the processor is not idle.

3.2.8 MULTIPROCESSING (Continued)

The multiprocessor flags indicate the presence of a specific IPC condition; they do not contain data (such as IPC variants or processor IDs).

There are no locks on the individual processor's local regions. Although all processors verify the IPC signal, there is no guarantee that multiple coincidental IPC signal assertions will not occur. Therefore, software using the IPC signal must not depend on any particular timing window.

When a processor executes an IPC instruction, it asserts the proper digit flag for the variant being executed. The flag is asserted for the multiprocessor local area for each processor indicated by the processor mask field of the Interprocessor Communicate instruction (see V-Series Instruction Set specification 1997 5390).

The processor executing the IPC asserts the IPC line, thus causing each processor to check the IPC Executed field in its local processor region. The selected processor then executes an IPC interrupt to perform the indicated action.

4 INSTRUCTION FLOW

4.1 GENERAL DESCRIPTION

The IF Reader issues memory read requests to fetch instructions from main memory. The IF Formatter parses the instructions from the IF Read Queue into the Instruction Queue (IQ), setting validity bits on the IQ entries as each is written. The IF FBUS Controller then sends this IQ and LQ information to the OF and XM fetch page.

The XM contains four fetch pages; while an XM is processing an instruction in one fetch page, the IF and OF may each fill up the other pages with succeeding instructions.

Example:

fetch page 1: XM executing instruction #1
fetch page 2: Instruction #2 waiting XM execution
fetch page 3: OF working on instruction #3
fetch page 4: IF parsing instruction #4

Refer to Sections 3.2.1, 3.2.2, and 3.2.3 for the fetch page management protocols for IF, OF, and XM.

The OF resolves any indexing, indirection, or indirect field lengths. In addition, it fetches memory operands for some of the instructions. When the OF fetches the operand for the instruction, it tags the operand fetch request with an Operand Fetch Tag, which consists of the current OF_PTR value, operand queue ID, and the destination module ID.

The CM receives the operand fetch tag and holds it until the requested operand is read from the cache or memory. Then, the CM puts the operand and the tag onto the RBUS. The tag directs the operands to the proper destination.

The OF and the Lock Unit work together to detect any data interlock situations in the pipeline. If an instruction in the pipeline needs to access an area in memory, and that area is being written into by a preceding instruction, the second instruction will know that it must wait until the latest data is present in memory.

4.1 GENERAL DESCRIPTION (Continued)

When an XM has completed one instruction, its clock is frozen until a new instruction is in its next fetch page. For those instructions in which XM must write to memory, OF sends the write address to the Lock Unit and posts a write lock. After XM notifies the Lock Unit that the write data has been transferred to the CM, the write lock is marked 'written' in the Lock Unit. Thereafter, any instruction which was data interlocked on this completed write lock may then be allowed to proceed in the OF. However, a 'written' write lock must remain in the Lock Unit for code modification checks until three further code checks have occurred.

When the XM completes an instruction, it checks for instruction-related errors such as undigits in addresses, limit errors, or memory parity errors. Undigit address errors may be detected by hardware in XM, which sets the appropriate interrupt flag. Parity errors are detected by the memory subsystem. They cause an external interrupt, which may be detected by the XM after execution of a subsequent instruction. Other external interrupts include instruction timeout, I/O completes, timer, air loss/over temperature, and Trace interrupts.

Non-optimal cases such as pipe clear conditions and branching will be described in the following sections.

4.2 OPLIT REGISTER BRANCH MECHANISM

To eliminate the clocks XM would have spent on each instruction to decode the OP and check for a literal, the OPLIT Register Branch Interface between the Fetch modules and the XM was developed. The OPLIT Register is in XM; it is loadable by Fetch, and is associated with the current fetch page.

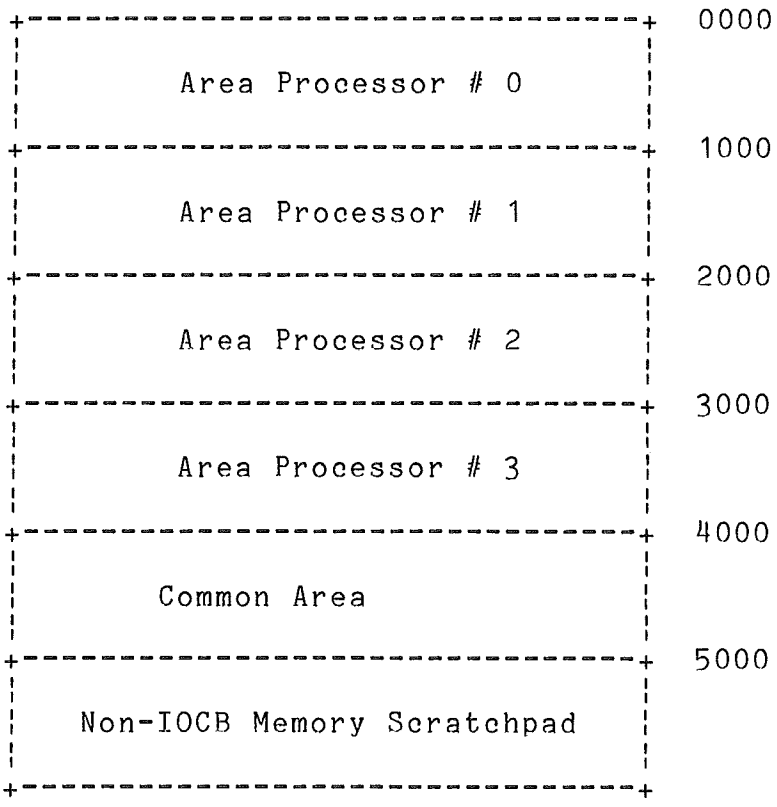
On the last clock of instruction processing, XM switches its fetch page and performs an n-way branch based on the contents of the OPLIT register and optimal flags. As a result, on the first clock of the next instruction execution, the XM knows the OPcode, and if the A address syllable contains a literal.

4.3 MACHINE STATE

The machine state consists of main memory, the multiprocessor shared memory area, the processor main memory scratchpad, the Base/Limit tables in memory control, the State toggles, the Comparison toggles, and the accumulator.

4.3.1 MAIN MEMORY LAYOUT

Each V500 processor has access to all of main memory. However, each processor has its own region of main memory which is not known to the MCP. This area is called the processor main memory scratchpad. It is intended that these processor main memory scratchpads be allocated at the high end of main memory, pointed to by Base/Limit pair "D". The layout of the main memory scratchpad is as follows:



4.3.1 MAIN MEMORY LAYOUT (Continued)

The scratchpad area dedicated to each individual processor (Area Processor #n) is defined as follows:

Information	Digits
Mobile Index Registers	
IX4	000-007
IX5	010-017
IX6	020-027
IX7	030-037
Mode Descriptor	040
Interprocessor Communicate:	
Start Yourself	041
IPC Executed	042
Interrupt Idle	043
Coprocessor State	050-999

4.3.1 MAIN MEMORY LAYOUT (Continued)

The Multiprocessor Shared Memory Area ("Common Area") contains the following information:

Information	Digits
Kernel Lock	000
Kernel Reserved	001
SNAP Lock	002
SNAP Picture Enable	003
M.E.R. Lock	004-005
M.E.R Enable	006-007
Over Temperature	008
Hardware Registers:	
Reinstate List Address	010-018
Snap Picture Address	020-028
Memory Area Status Table Address	030-038
Memory Error Report Address	040-048
Cell Table Address	050-058
Volume Table Address	060-068
Channel Table Address	070-078
Direct I/O Field Address	080-088
Cell Header Table Address	090-099
Time of Day:	
Lock	100
Day	101-102
Month	103-104
Year	105-108
Reserved	109
System Status	110-111
System I/D	112-311

4.3.1 MAIN MEMORY LAYOUT (Continued)

The Non-IOCB Memory Scratchpad is 3080 digits, starting at offset 5000, and contains the following:

Information	Digits
Each I/O channel scratchpad entry is as follows:	
Current Buffer Address	000-009
Buffer End Address	010-019
Extended R/D	020-031
Channel Busy Flag	032-033
IOP use	034-039

The 50 digit mailbox is located at an offset of 320 into the I/O channel scratchpad memory as follows:

Channel Number for Initiate I/O	000-003
Resolved I/O Descriptor Command	004-009
Resolved I/O Descriptor A Address	010-019
Resolved I/O Descriptor B Address	020-029
I/O Descriptor C Address	030-037
Resolved I/O Descriptor D Address (SMD Length)	038-045
Mailbox Busy Byte (I/O Request)	046-047

4.3.2 BASE/LIMIT TABLE

The Base/Limit Table is in the MCACM; it is composed of two 16-word, dual port register files. One file contains the base table entries, and the other contains the limit table entries. Both tables are addressed by the base indicant field from the processor modules. Base/Limit entries have been allocated as follows:

Base Indicant	Base/Limit Pair
0	Pair 0
1	Pair 1
2	Pair 2
3	Pair 3
4	Pair 4
5	Pair 5
6	Pair 6
7	Pair 7
8	Reserved Multipurpose
9	Reserved Multipurpose
A	Reserved Multipurpose
B	Reserved Multipurpose
C	Reserved Multipurpose
D	Reserved Multiprocessor Shared Area B/L
E	Reserved MCP Data Area B/L
F	Reserved Absolute Address B/L

4.3.3 COMPARISON TOGGLES

The Comparison Toggles are maintained in the XM.

4.3.4 ACCUMULATOR

The Accumulator is maintained in the XM.

4.4 PIPELINE DATA INTERLOCKS

The Operand Fetch module and the Lock Unit detect and handle the memory data interlocks that may occur in the instruction pipeline. An interlock happens when an instruction in the pipeline needs to read a memory location that is being written into by a preceding instruction.

4.4.1 ADDRESS AND OPERAND DATA INTERLOCKS

The Operand Fetch module and the Lock Unit have the responsibility for detecting and handling address and operand data interlocks. When the OF module processes an instruction, it issues write locks to the Lock Unit for those memory areas into which the instruction will write. In some instances, the OF locks all of memory rather than issuing multiple write locks. These locks are the starting and ending base-relative addresses for the data.

If OF encounters an instruction that requires a memory read to resolve a field (indexing, indirection, or indirect field length), OF issues a memory request, then asks the Lock Unit to check for outstanding locks posted for the desired memory area. If the data is interlocked, it is discarded, and OF waits for the instruction that posted the lock to complete. Otherwise, OF uses the data to resolve the appropriate field.

The OF also requests the operand prefetch data for some of the instructions. The data from this read automatically goes into one of the Operand Queues in the XM instead of being sent to the OF. The OF must perform a data interlock check for these memory reads.

4.4.2 CODE MODIFICATION CHECKING

The OF module is responsible for detecting and handling code modification. When the OF starts processing a new instruction, it asks the Lock Unit to check for outstanding write locks for the memory area bounded by by the PC, and the PC plus the instruction length.

If there is a write lock, then the instruction is considered to be code-modified. OF discards this code-modified instruction by changing it to an internal NO-OP, then waits for the OP Complete from XM for the instruction that caused the write lock to be posted. After the OP Complete is received, OF issues a NEWPC command to IF so that the IF can refetch the instruction. The old instruction is replaced by this refetched instruction.

4.5 DEFINITION OF PIPELINE FLUSH

There are three kinds of pipeline flush:

- Context Switch
- Taken_Branch
- Not_Taken_Branch

In case of a conditional branch, the branch predicton algorithm (see section 4.7) predicts whether the branch path should be taken. If the algorithm predicts that the branch path should be taken, the Formatter clears the Reader to start issuing memory requests down the taken path.

Meanwhile, because in most cases the first instruction in the fall-through path has already been read by the Reader, the Formatter parses it and stores the parsed syllables in the NIA register. Later, if the XM finds that the prediction is wrong and the fall-through path should have been taken, a Taken_Branch_ Flush is issued, and OF and XM take the information out of the NIA register and restart the instruction immediately.

If, however, the branch prediction algorithm predicts that the branch should not be taken, XM processes the normal path only, and no branch path instructions are accessed by the pipeline. If the XM finds that the prediction was wrong, the Not_Taken_Branch_Flush is issued. The entire pipeline is flushed, and the branch path is restarted from the beginning.

The Context Switch Flush is issued in situations such as environment changes or context switching. It is the most drastic of the flushes.

All pipeline flushes cause the following actions to occur:

- OF and XM are cleared
- all bus transactions are ignored
- memory control requests terminated (except writes)

4.6 FETCH CLEAR FUNCTION

There are two types of flushes that affect only the Fetch Modules. The Code Modification Flush discards the fetched instruction in the Read Queue of the Reader when code modification is detected. It also refetches instructions in Single Instruction mode. Also, if the Lock Table in the Lock Unit is full, the OF issues a Lock Full Flush, which discards the instructions in the Read Queue.

4.7 BRANCH PREDICTION

When the IF module encounters a conditional branch, it selects one of the two paths as the one most likely to be taken by the XM when the instruction is executed. The selection is based solely on the OP code. IF continues fetching instructions down that path.

Because IF and OF can continue processing without waiting for the Comparison Toggles to be set, a higher machine performance is achieved. It is the responsibility of the XM to verify the correctness of the branch prediction against the state of the Comparison Toggles and, in the event of a bad prediction, to force the pipeline to execute instructions down the other path by performing the appropriate pipeline flush function.

Modeling has demonstrated that if you maintain information about the most recent direction trend occurring at a conditional branch, you 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, a branch prediction direction.

a. TAKEN/TAKEN - Predict branch taken. The predominant direction trend is taken (FX opcodes).

b. NOT TAKEN/TAKEN - Predict branch taken. The last time it was NOT taken; before then the branch was taken (EX opcodes).

c. TAKEN/NOT TAKEN - Predict branch not taken. The last time it was taken, before then the predominant path was NOT taken (BX opcodes).

d. NOT TAKEN/NOT TAKEN - Predict branch not taken. The predominant direction trend is NOT taken (2X opcodes).

The OPcode transitions are illustrated in Figure 4-1.

4.7 BRANCH PREDICTION (Continued)

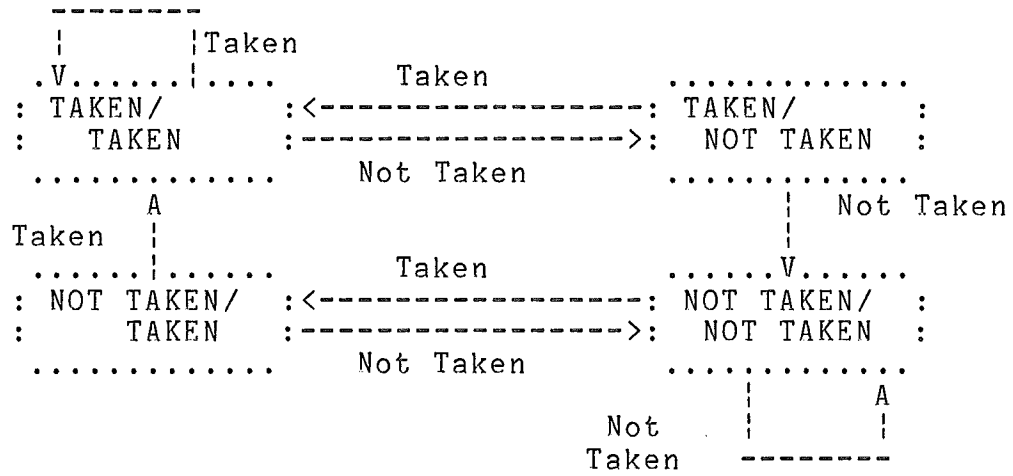


Figure 4-1 Opcode Transitions

5 I/O SUBSYSTEM INTERFACES

Figure 5-1 shows the basic blocks of the I/O Subsystem and their interconnectivity.

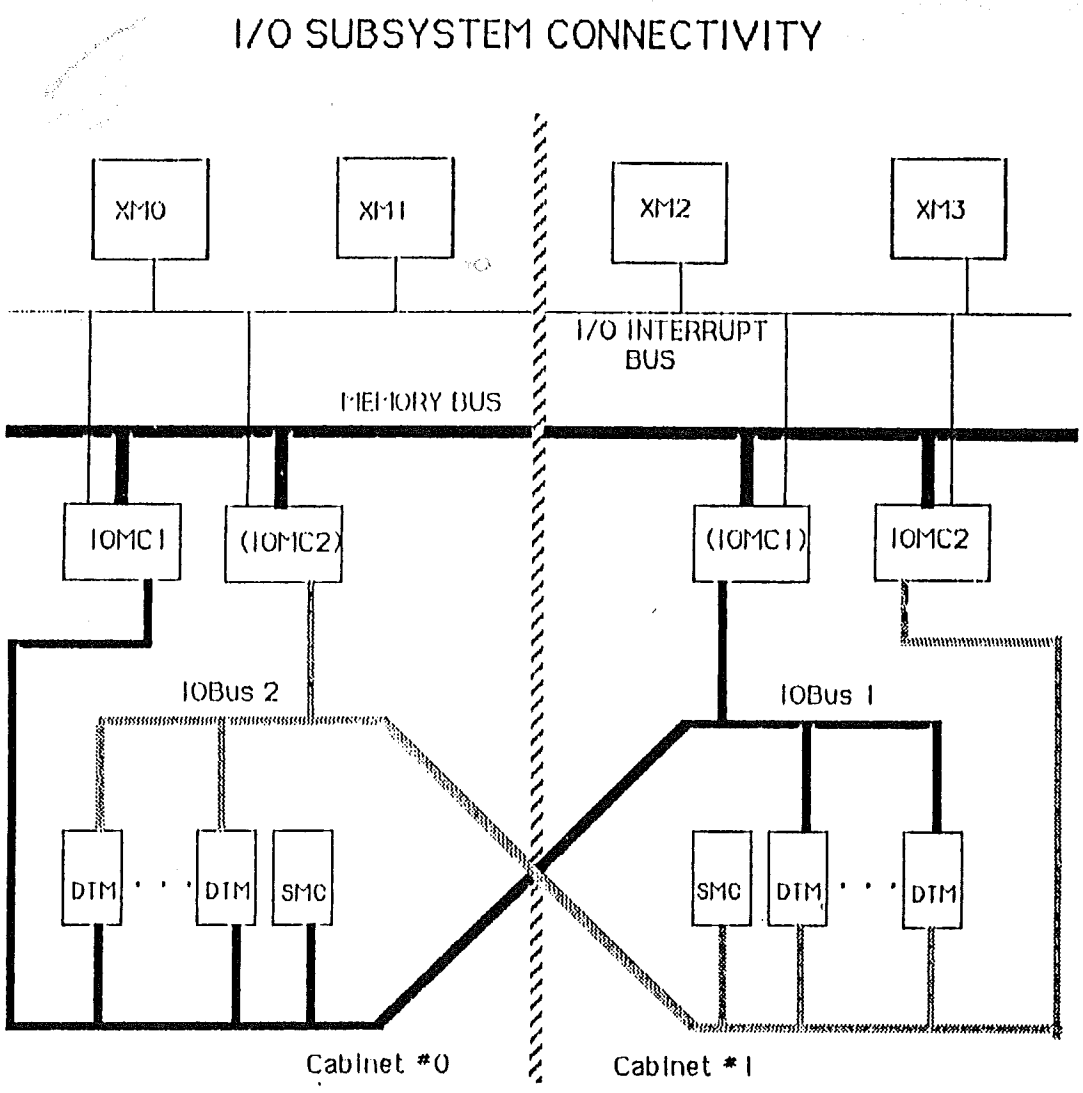


Figure 5-1 I/O Subsystem Connectivity

5.1 I/O SUBSYSTEM MEMORY INTERFACE

Data Transfer Modules interface to memory via the I/O Memory Concentrator (IOMC). The primary functions of the IOMC are:

- To prevent excessive loading on the memory subsystem bus.
- To convert the TTL levels in the I/O subsystem to the ECL levels in the rest of the system.
- To Provide a scratchpad to interface the processor to DTMs.
- To select the DTM for I/O initiation.
- To select the XM to interrupt when reporting I/O completes.

As indicated in Figure 5-1, the I/O subsystem contains two 32-bit data buses. When there are two IOMCs, data can flow between either IOMC and any DTM. This structure doubles the memory bandpass to the I/O subsystem.

Data is passed between memory and a DTM via a memory concentrator in maximum transfers of 4096 bytes (1024 bus transactions). Each transaction takes 2 system clocks, resulting in a data transfer rate of 41.6 Mbytes/sec, based on a 48ns system clock. To ensure synchronization with memory, a half-frequency clock is provided within the subsystem.

To initiate a transaction, the memory begin address (in BCD) is sent to the IOMC. The data field length is sent with the begin address. The field length (in bytes) is a hexadecimal countdown value between 0-3FF.

The IOMC performs the address translation and the data conversion required to align the I/O data with the memory word size.

5.1 I/O SUBSYSTEM MEMORY INTERFACE (Continued)

All memory addresses generated by the I/O subsystem are absolute addresses.

There are 3 types of memory operations; they are:

- o read
- o write
- o read with lock (an indivisible read modify write cycle)

A 2 bit field associated with the memory address identifies the type of memory operation.

5.2 I/O SUBSYSTEM/CENTRAL PROCESSOR INTERFACE

The central processor and I/O subsystem communicate via the IOMC initiate interface, fields in main memory, and the I/O Interrupt Bus.

After the CPU has written an I/O descriptor into the I/O mailbox in main memory, it sends a transaction to an IOMC. The IOMC then signals the master DTM, which reads the mailbox and signals the DTM connected to the I/O channel in the I/O descriptor.

The DTM performs the I/O operation and reports the results to the appropriate channel result descriptor in main memory. Then the DTM reads a processor status field in memory, selects the best processor to interrupt (i.e., an idle processor), and then sends the processor number and type of I/O complete (i.e., normal, real-time, or error) to the IOMC.

The IOMC first polls the selected XM, via the I/O Interrupt Bus, to see if it will accept any of the pending (not yet accepted by an XM) types of I/O complete. An XM will only accept those types of I/O complete which are not currently masked by its I/O Interrupt Mask (part of its processor state). As I/O complete types are accepted by an XM, the IOMC stops reporting them until the next I/O complete of that type occurs. If not all of the I/O complete types are immediately accepted, the IOMC will continue to accumulate them and poll all of the XMs in a round-robin fashion.

6 MAINTENANCE PROCESSOR/OPERATOR INTERFACES

6.1 SYSTEM INITIALIZATION

First the processor, I/O cabinets, and MP must be powered on. The following functions are selectable:

1. CLEAR MEMORY. Writes zeroes or a specific pattern to all or part of memory.

2. CONTROL STORE LOAD AND VERIFY (one or more modules)

3. INIT

4. PA. Processor restart at specific address. Control Store, memory, and other system attributes unchanged.

5. LOAD CONTROL STATE PROGRAM

6. LOAD MCP

The SYSTEM STATUS display exhibits the status of the processor (running, stopped, error halted, etc.)

6.2 HALT INSTRUCTIONS

The HBK (Halt Breakpoint) and HBR (Halt Branch) instructions cause the processor to halt, depending upon the circumstances. Once an XM has detected one of these instructions, it issues a Write PC followed by a Read PC (to make sure that all writes are out of the A&W Bus input queue, and that error vectors from previous memory writes are recorded in an interrupt array).

The Fault bit in the interrupt array is examined; if it is set (by a previous memory write error), an HCP (Hardware Call Procedure) is performed reporting that previous instruction address.

If there are no write errors, XM sets the PROHALT Toggle and drives the A&W Queue Full net to inhibit any additional memory requests from Fetch. After seeing XM-STOP-TIME (a result of PROHALT Toggle being set) SMC waits for the MCACM IDLE signal, then performs a processor clock stop (i.e., stops the clocks only to Fetch and XM).

After issuing the PROHALT transaction, XM loops, waiting for the PROHALT signal to be reset by the MP. The MP resets PROHALT before starting the clocks again so that XM knows that the halt is over. Now, XM determines the next program address according to the particular halt instruction and starts the pipeline clear function.

The MP gets the data about the instruction from the fetch page being executed by the XM.

6.3 INSTRUCTION TIMEOUT

The instruction timeout timer determines if an instruction is caught in an infinite loop, or if it is performing an unreasonably long variant of an instruction. This timer is cleared each time XM flips to a new fetch page. This timer function has a soft instruction timeout (TIMEOUT-1), and hard instruction timeout (TIMEOUT-2).

After approximately one second, the TIMEOUT-1 toggle is turned on, thereby allowing the Fetch and XM micro-code to test this toggle in infinite or long loops so as to detect instruction timeouts.

If the timer runs for an additional second, then the TIMEOUT-2 toggle is turned on, thus notifying the MP. The MP assumes that a hardware failure occurred, and stops the procesor.

6.4 SNAP PICTURE REPORTING

There is one SNAP Picture allocated for an entire multiprocessor system. The SNAP Picture Address and SNAP Picture Enable are stored in the Multiprocessor Shared Memory Area and are updated whenever the SNAP Picture Address variant of the Write Hardware Register instruction is executed.

There are three modes for Snap Picture Reporting: Hardware Error, Software Error, and Maintenance.

The Hardware Error SNAP occurs whenever the processor detects a fatal hardware error (MOD BROKEN signals).

The Software Error SNAP is enabled by the Global Snap Enable toggle (set and reset by the MCP), and occurs whenever an instruction error, address error, or instruction timeout occurs for a task which has been "task SNAP enabled". The Maintenance Processor saves a snapshot of the machine state on its own disk when the Snap-time signal is received from the XM. The XM does not have a dedicated line to the SMC; it uses the XM_STOP_TIME line.

When the MP detects the signal, it examines the XM to determine the reason for the halt. If an Instruction Timeout, Instruction Error, or Address Error occurred, and Snap is enabled, then the MP takes the picture, sets the Snap Picture Report Taken toggle, and then starts the processor clock. The XM writes the Processor R/D and performs a Hardware Call Procedure.

The Maintenance SNAP is under control of the Maintenance Processor; it can be set up to take a picture whenever some condition is detected by the MP stop logic, or a user command.

For all SNAP modes, MP stores the header record of the SNAP picture in memory at the location specified by the SNAP picture address. This record includes the name of the file on MP disk containing the complete SNAP picture.

For all SNAP modes, after the MP restarts processor clocks, the XM writes the Processor R/D and performs a Hardware Call Procedure.

6.5 MEMORY ERROR REPORTING

There is one Memory Error Report allocated for an entire multiprocessor system. The Memory Error Report Address and Memory Error Report Enable are updated whenever the Memory Error Report Address variant of the Write Hardware Register instruction are executed.

Pending corrected Single-bit parity, error reports are written into memory and reported by the XM whenever it executes a System Status instruction.

Double-bit parity errors are handled by the XM according to these constraints:

> If the error was for the current instruction or for a future instruction and the instruction is still retryable, then the XM will deadfreeze (signaling the SMC), write the error report (after SMC restarts XM), deadfreeze again and then retry the current instruction (after SMC restarts XM again).
>
> If the error is for a future instruction and the current instruction is no longer retryable, then the error is handled at the end of the current instruction.

6.6 HALT/SINGLE INSTRUCT

These functions are controlled by the Maintenance Processor, working in conjunction with the XM module. When the MP receives a HALT or SI command, it shifts a bit indicating SI time into the XM maintenance chain. At pop fetch page time, XM detects the SI bit in the maintenance chain and does a live-freeze.

When the XM's output buffer is empty, XM_STOP_TIME and A&W Queue Full signals are generated. The SMC then waits for MCACM-IDLE before interrupting its micro-processor. The SMC software must handle the interrupt and turn off clocks to Fetch and XM.

6.7 PROCESSOR RUN MODES

a. BURST/SINGLE CLOCK

The hardware and MP software supports single and burst mode clocking. All processors, memory, and I/O must be clocked together for proper system operation. Up to 64K consecutive clocks may be issued in a burst. If only one clock is issued per burst, then the system is singled clocked.

b. HALT/RUN MODE

The running and halting of the system is controlled by the operator in conjunction with the system error and stop logic. If operator issues a run command, the clocks are enabled to the system until the operator issues a stop command, or until maintenance panel stop condition event occurs, or an error (MOD BROKEN) occurs.

6.8 ERROR RETRY

If an error is detected, either the macro- or micro-instruction may be retried. The micro-instruction retry is performed if a control store parity error is detected. The macro-instruction is retried in hardware error detected cases. The MP/SMC software controls when and if instructions should be retried based upon whether the XM has attempted to change any processor state (i.e., accumulator, main memory, etc.).

6.9 MAINTENANCE STOP CONDITIONS/PANEL

There are three types of stops. They are:

- o System stop. Stop all modules immediately (or as soon as possible).
- o Instruction stop. Stop XM and Fetch at a point between instructions (after the stop condition occurs).
- o Dryup stop. Stop XM and Fetch after XM live freeze, and Fetch and MCACM go idle.

A stop condition is set by shifting a bit pattern into the module's maintenance chain. When the stop condition is met, SMC stops either the entire system, or XM and Fetch only (depending on the type of stop).

6.9.1 SYSTEM STOP

This stop causes the SMC to disable the clocks to all modules and memory data chains, which causes the whole system to stop. The stop occurs on the third clock after the stop event is detected. This should be a recoverable stop if no I/O's are in process, otherwise any I/O's in process may be lost.

When the module stop condition(s) are met, the STOP-AND signal on the card is driven LOW (since STOP-AND is a low-active signal) or the STOP-OR signal HIGH, depending on whether the AND or OR have been requested. If the stop condition occurs in the XM module, the XM also does a LIVE-FREEZE (keeping the interfaces alive). The SMC disables clocks to all modules 3 clocks later.

System stop is available for various conditions in modules XM, Fetch, MCACM, IOMC, and DTM. The Fetch Module also has logic to stop on conditions on processor buses.

6.9.2 INSTRUCTION STOP

This stop idles XM and the Fetch modules at the next single instruct breakpoint; MCACM keeps running. In this case, the stop occurs gracefully, and no data is lost.

Instruction Stop is available only on conditions detected by Fetch, and is limited to Stop on Op and Stop on Instruction address.

When an instruction stop condition is met during fetch, the fetch module sets the SI flag bit in PC_SYL. The fetch page that has an SI bit set is considered to be "marked". When the XM executes from the marked fetch page, it performs an instruction stop.

There are two bits in the XM maintenance chain that enable the single instruct stop. When the bits are set, and XM reads SI flag from the marked fetch page, XM completes the current instruction, then does a live-freeze. When the XM's output buffer is empty, the XM-STOP-TIME signal is generated, and XM drives the A&W Queue Full signal true. This allows, at most, one additional Fetch A&W Bus transaction before the halt. All Fetch processing stops gracefully when A&W Bus access is required. The clocks to Fetch may be stopped any time after all data has been returned from MCACM.

The SMC waits for MCACM-IDLE before interrupting its microprocessor. More precisely, the SMC delays XM-STOP-TIME by 1 clock, ANDing the result with MCACM-IDLE in SHIFT ARRAY to produce an interrupt.

The SMC software must then handle the interrupt (which is the same interrupt line as Mod-Broken). Having determined that this is an SI interrupt, the SMC software stops the XM and Fetch clocks. Since XM is live-frozen, and Fetch and MCACM are idle, the number of clocks required to stop is not an issue.

6.9.3 DRYUP STOP

Dryup is used to stop the processor only for conditions that are not supported by Instruction Stop. For Dryup Stop, the XM monitors the backplane to see whether a stop event has occurred. When the event occurs, XM live-freezes. Because the XM is live-frozen, Fetch and then MCACM will eventually idle since XM is no longer feeding queues. When MCACM is idle, the SMC will get an SI form of interrupt. At this time, the MP software will turn off PROCESSOR CLOCKS only. I/O and memory will keep running.

For dryup, the number of clocks between occurrance of the stop event and the idling of Fetch is indeterminate. The stop should be recoverable, even though it has not occurred at an SI point.

Dryup Stop is available only for stops monitored in XM, Fetch, or MCACM. This form of stop can be requested for any stop(s) in these three modules.

6.9.4 COMBINING STOP CONDITIONS

The selected stop conditions can be AND'ed or OR'ed to drive the STOP-AND or STOP-OR signals to the SMC.

If an OR stop is selected, then the stop (either System or Single Instruct) occurs whenever any of the conditions in that module are met.

If an AND stop is selected, then the stop occurs only if all conditions in that module have been met. Note that some conditions are combined (functionally AND'ed) when they are enabled. The result of AND'ing can be put on either the STOP-AND or the STOP-OR line.

There are some restrictions in combining STOP conditions. The different cases are as follows:

1. If conditions in each module are AND'd, then the result of AND in each module is AND'd with the result of AND from another module.

2. If conditions in each module are OR'd, then the result of OR in each module is OR'd with the result of OR from another module.

3. If conditions in one module are AND'd and conditions in another module are OR'd, then the result of ANDing in one module is AND'd with the result of ORing in another module.

4. Note that the result of ANDing in one module cannot be OR'd with result of ORing in another module.