CRAY T3E Processing Element Operations

HMM-162-0
Last Modified: October 1996

Record of Revision

October 1996

Original printing.

Introduction

When a processing element fails, you may need to replace a field replaceable unit (FRU) or map out a processing element. An FRU for a processing element failure may be a processing element module or a daughter card.

This document describes the components of the processing element and the types of memory access. This document also lists the diagnostic tests you can use to troubleshoot a processing element. Specifically, this document answers the following questions:

1. What memory sizes are offered for the CRAY T3E system?
2. How does a microprocessor access data from another PE's local memory?
3. How does the CRAY T3E system hide the latency of retrieving information from another PE's local memory?
4. How does the support circuitry translate information from the microprocessor into a physical memory address?
5. How do PEs within a partition work together to process a user application?

Terms

You will need to know how the following terms are used in this document to fully understand the material discussed in this document:

Backmap table- The backmap table stores partial address tags of the microprocessor's secondary cache and is used to maintain cache coherency.

Communication links - Communication links transfer data and control between two network routers in the interconnect network.

Dirty data - Dirty data is data that was modified by the microprocessor but was not transferred to the support circuity to be updated in local memory.

E registers - E registers are latency-hiding registers that are the source and destination for all global data transfers.

History table - The history table stores memory addresses of recently accessed data (by direct local memory reads) and is used to detect streams.

Interconnect network - The interconnect network provides communication paths among the nodes in the CRAY T3E system.

Network router - The network routers transfer data and control information though communication links.

Partition - A partition is a group of PEs that are assigned to one application.

Stream - A stream is a set of load commands that references contiguous addresses in local memory.

Stream buffers -The stream buffers store read-ahead data from consecutive memory locations. These buffers are used to reduce the memory latency of direct local memory reads.

Victim - A victim is a valid, dirty secondary cache entry that the microprocessor removes to make room for new data.

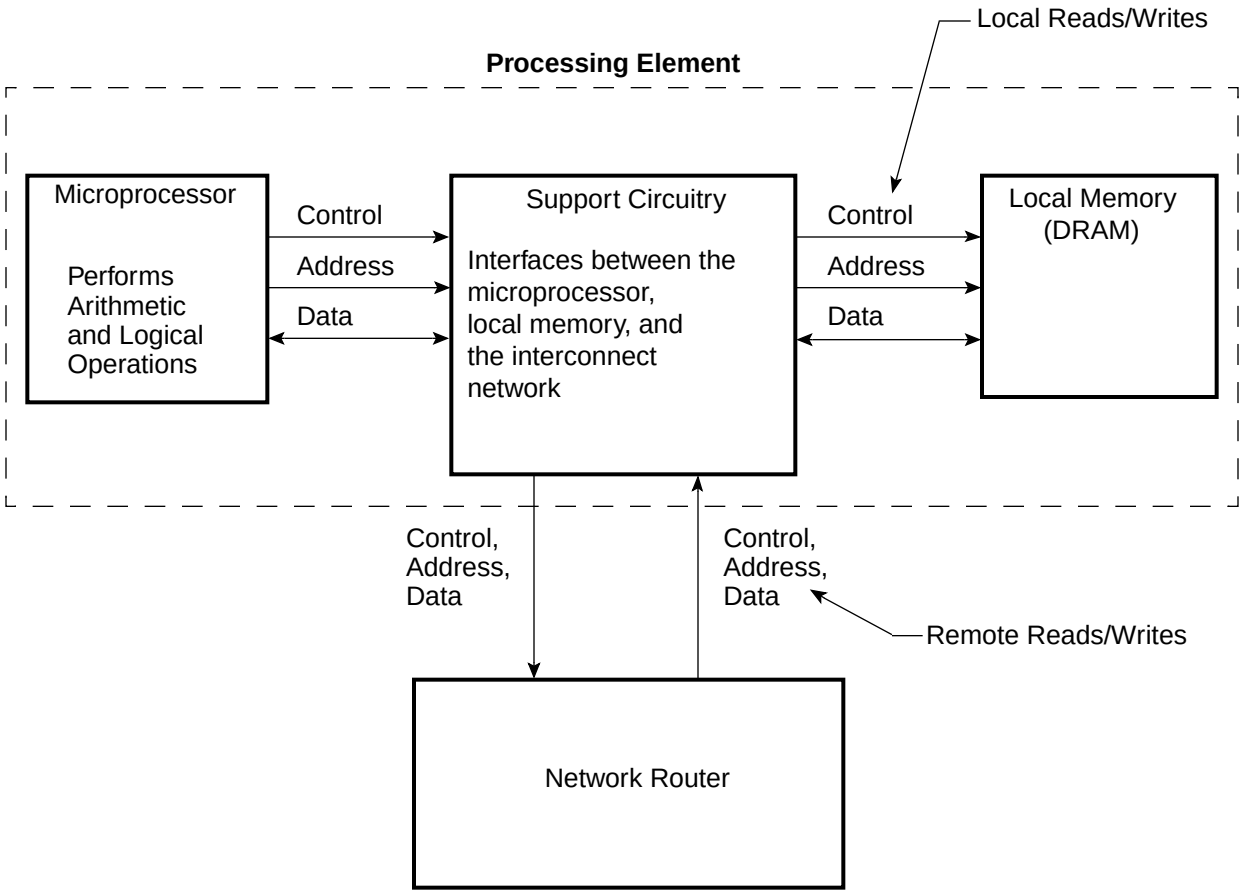
Virtual PE number - The virtual PE number is a number that an application uses to reference each PE in a partition.

Components of a Processing Element

A processing element consists of a microprocessor, support circuitry, and local memory (refer to Figure 1). The microprocessor is a reduced instruction set computer (RISC) microprocessor that performs arithmetic and logical operations on 64-bit integer registers and 64-bit floating-point registers. The support circuitry is an interface between the microprocessor, local memory, and the interconnect network. The local memory consists of dynamic random access memory (DRAM) that stores system data. A processing element with a network router is referred to as a node.

There are two types of PEs: support PEs and user PEs. Both types of PEs have identical hardware; however, they perform different functions. User PEs run user applications. Support PEs function only as operating system servers.

Figure 1. Processing Element with Network Router

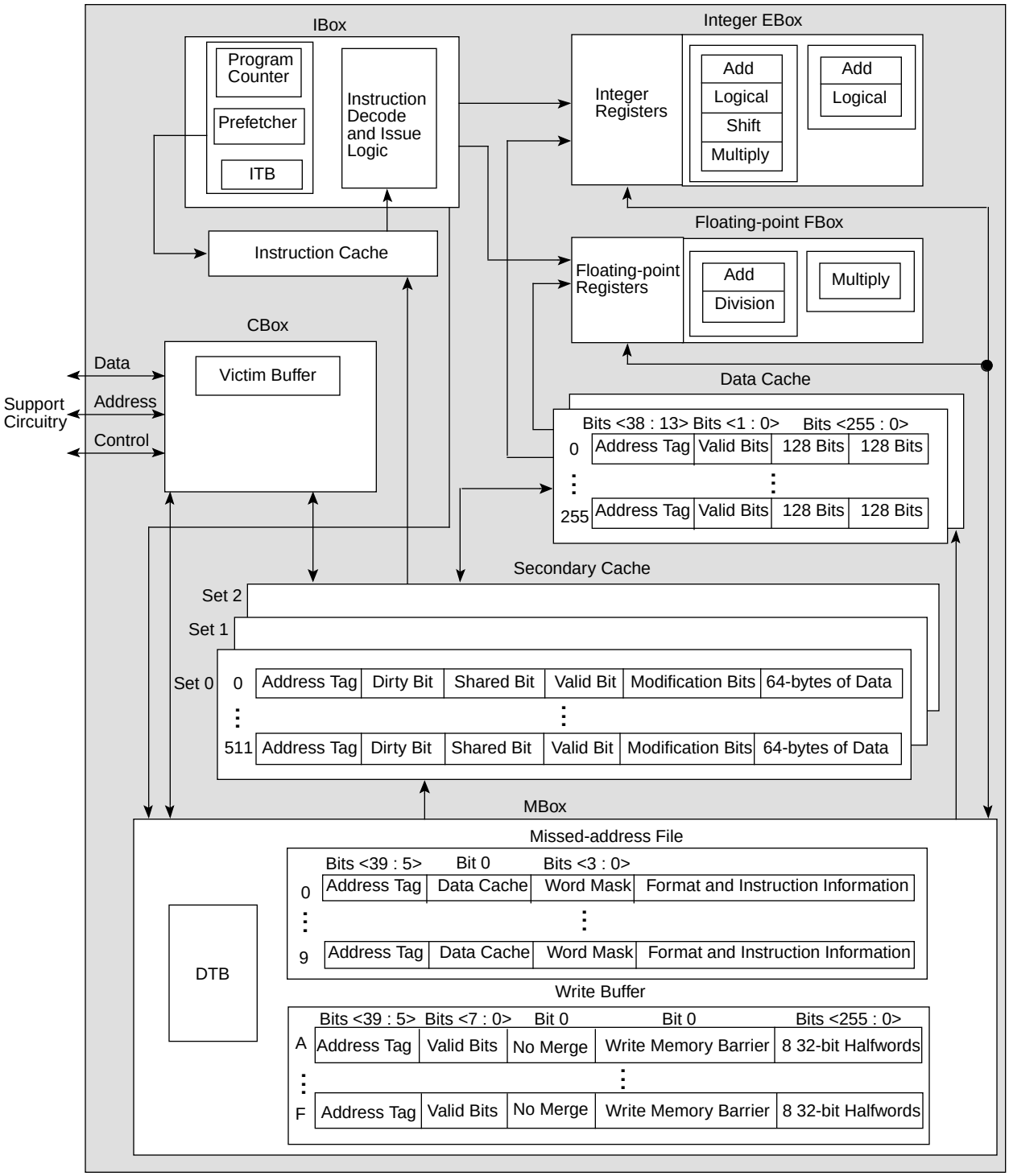


Microprocessor

The microprocessor contains the following components (refer to Figure 2):

- The instruction fetch/decode unit and branch unit (IBox) retrieves, decodes, and issues the instructions to the execution units. The instruction circuitry contains the following components:
  - The prefetcher retrieves instructions from the secondary cache.
  - The program counter determines the next instruction to decode.
  - The instruction translation buffer (ITB) is a 48-entry buffer that stores recently used instruction address translations.
  - The instruction decode and issue logic issues instructions to the execution units.
- The instruction cache is an 8-Kbyte cache that temporarily stores frequently accessed or recently accessed instructions.
- The execution units performs the arithmetic and logical operations. The integer execution unit (EBox) performs integer operations and the floating-point execution unit (FBox) performs floating point operations.
- The memory address translation unit (Mbox) consists of the following components:
  - The write buffer is a 6-entry buffer that temporarily stores write data in an attempt to merge 32- or 64-bit write operations before data is transferred to the secondary cache or system memory.
  - The missed-address file is a 10-entry buffer that temporarily stores Dcache and Icache misses in an attempt to merge 32- or 64-bit loads from the same 32-byte line into one operation.
  - The data translation buffer (DTB) is a 64-entry buffer that stores recently used data (load) address translations.
- The data cache is an 8-Kbyte cache that temporarily stores frequently or recently accessed data.
- The secondary cache is a 96-Kbyte cache that stores instructions and data.
- The cache control and bus interface unit (CBox) sends/receives data, address, and control signals to/from the support circuitry. The CBox also contains the victim buffer that temporarily stores victims.

Figure 2. Microprocessor



Support Circuitry

The support circuitry provides an interface between the microprocessor, local memory, and the interconnect network. For the microprocessor, the support circuitry performs direct local memory operations, global memory operations, global address translation, barrier and eureka synchronization operations, and messaging.

Direct local memory operations allow the microprocessor to read and write local memory. For example, when software updates information in the secondary cache and the microprocessor must remove valid data from the secondary cache, the microprocessor creates a victim in order to make room for the new data. The CBox of the microprocessor stores the victim in its victim buffer. When the victim buffer contains two victims and the microprocessor creates a third victim, the microprocessor sends the three victims to the support circuitry. The support circuitry performs a direct local memory access write (store) to update data in local memory. When software requests data that is not in the secondary cache, the support circuitry performs a direct local memory access read (load) to write new data into the secondary cache.

Global memory operations allow the microprocessor to read or write data from any PE's local memory. The support circuitry controls all data transfers between the microprocessor and global memory by using a set of latency-hiding registers called E registers. For example, when the microprocessor needs data from global memory, the support circuitry reads the data from the global memory location and stores the read data in a designated E register(s). When the microprocessor needs the data, the microprocessor signals the support circuitry to transfer the data from the E register to the microprocessor.

When the microprocessor initiates a global memory operation, the support circuitry translates a microprocessor-supplied global index into a PE number, a global segment, and a segment offset. The PE number identifies the destination PE. The global segment and segment offset are used by the destination PE to create a physical address.

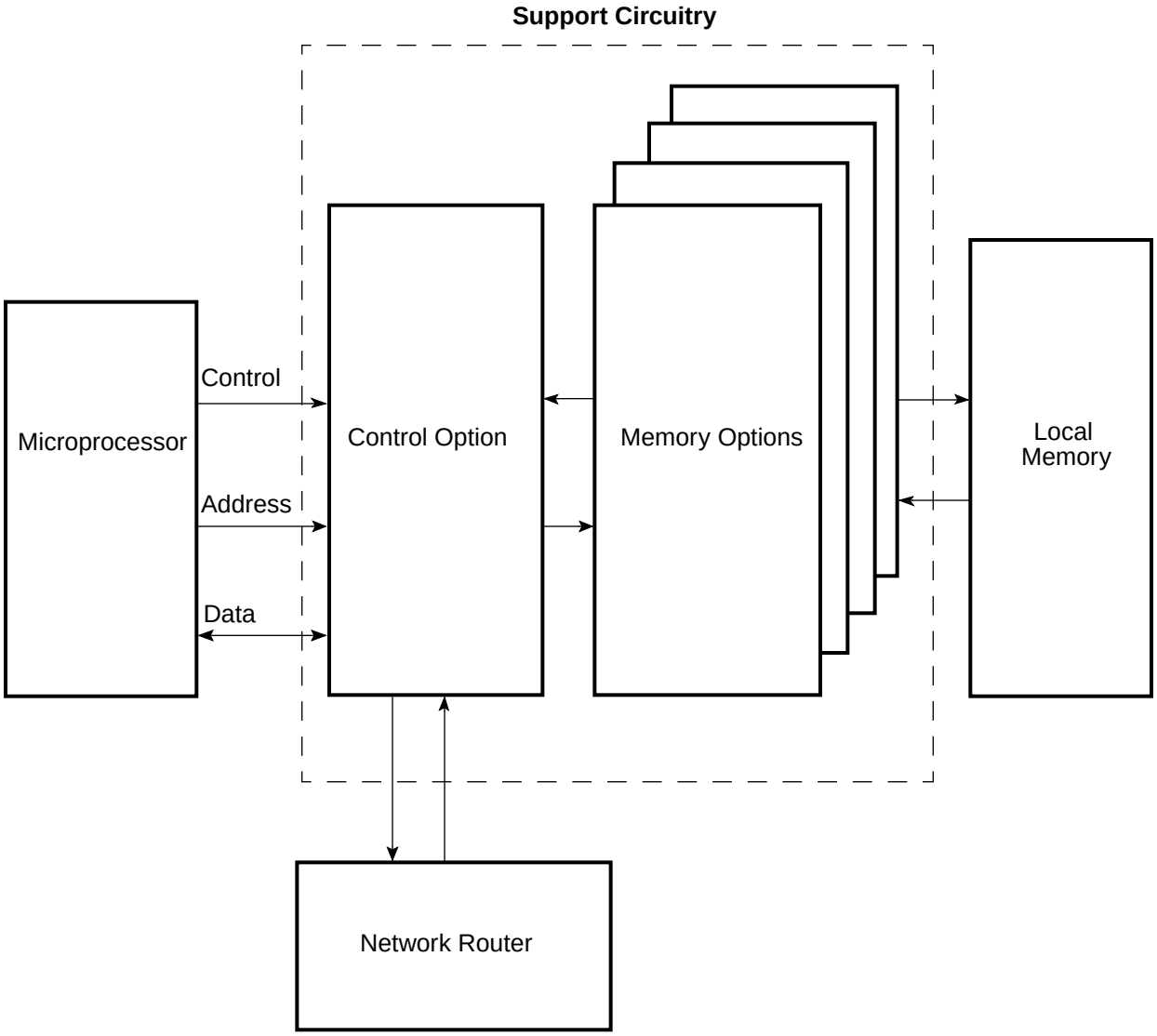
Barrier and eureka synchronization provide a low-latency method of synchronizing all or part of the PEs in a partition. When the microprocessor arrives at a barrier in a program, the microprocessor signals the support circuitry. The support circuitry transfers the barrier information from the microprocessor to the network router.

Messaging allows one PE to communicate with another PE. When a PE receives a message packet, the support circuitry stores an incoming message in a designated location in local memory called the message queue. When the message queue contains more than the specified amount of messages, the support

circuitry interrupts the microprocessor. The support circuitry then retrieves the message from the message queue and transfers the message to the microprocessor.

The support circuitry is made up of two option types: the control option and the memory option (refer to Figure 3).

Figure 3. Support Circuitry



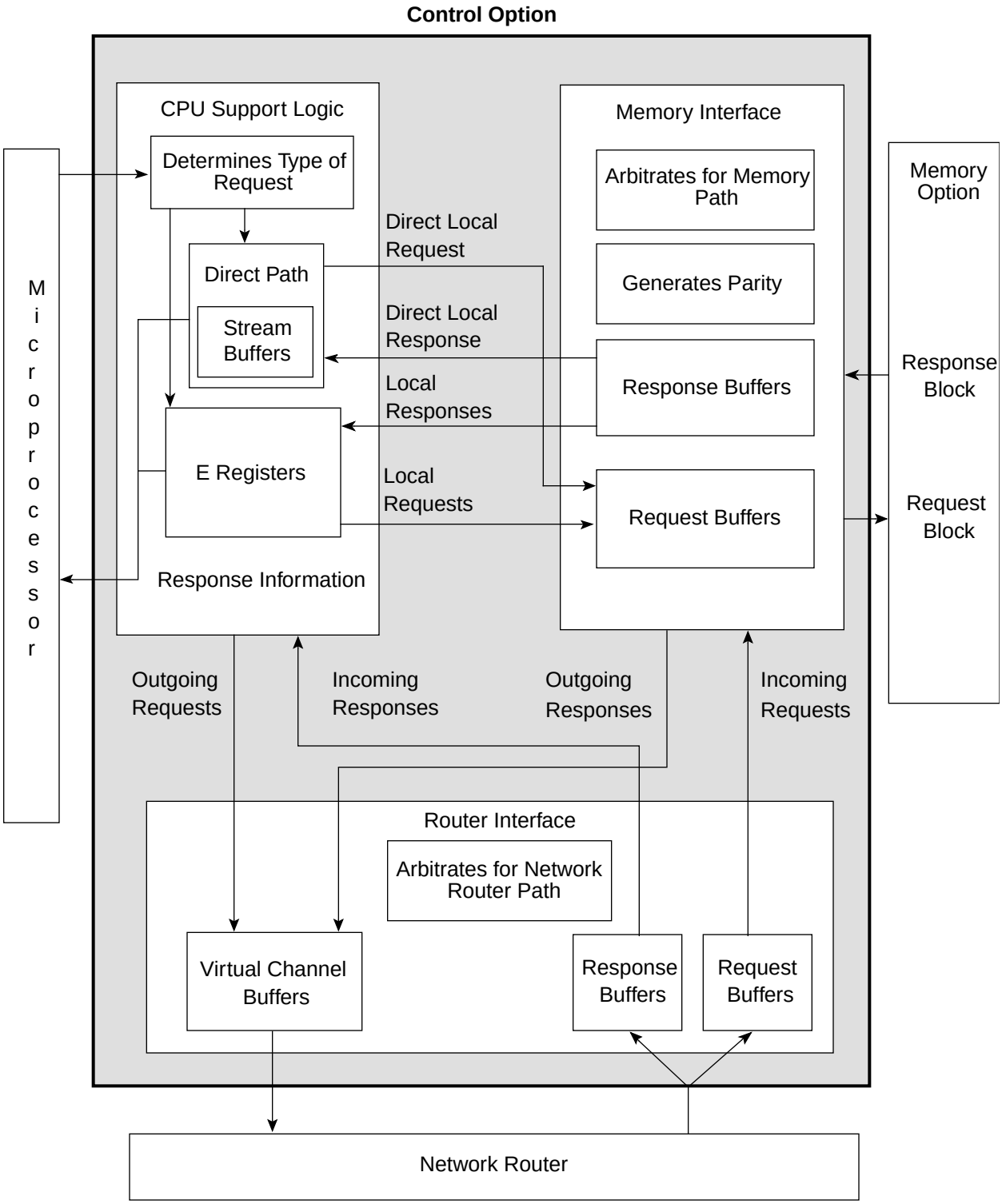
Control Option

Each processing element contains one control option (refer to Figure 4). The control option consists of the following components.

- CPU support logic - The CPU support logic performs the following functions:
  - Receives requests from the microprocessor
  - Determines the types of requests
  - Steers request information to the next appropriate logic block
  - Controls stream buffers
  - Controls E-register reads and writes
  - Sends response information to the microprocessor
- Memory interface - The memory interface performs the following functions:
  - Stores write request information in a first-in, first-out (FIFO) buffer
  - Arbitrates for a memory path to the daughter cards
  - Generates parity for the write request information
  - Controls the backmap
  - Converts the global virtual address to a physical address
  - Sends write request information to the memory option
  - Receives read data from the memory option
  - Stores read response information in a FIFO buffer
  - Sends read response information to the CPU support logic or the router interface
- Router interface - The router interface performs the following functions:
  - Stores outgoing requests and outgoing responses in virtual channel buffers
  - Arbitrates for the path to the network router
  - Sends outgoing requests and responses to the network router
  - Receives incoming requests and responses from the network router
  - Stores incoming requests in request buffers
  - Stores incoming responses in response buffers

    The CPU support logic controls the retrieval of data from the response buffers and the memory interface controls retrieval of data from the request buffers.

Figure 4. Control Option



The control option uses the packet format shown in Table 1 and Table 2 to send requests and responses to the M option and the R option.

Table 1. C Option to M Option Packet Formats

Command	Flit 0	Flit 1	Flit 2	Flit 3	Flit 4
IDLE	Idle	N/A	N/A	N/A	N/A
DPUT	DPUT Control	DPUT Data	DPUT Data	N/A	N/A
MSG_WRITE	Read/Write (R/W) Control	Data8	Data8	Data8	Data8
DGET	R/W Control	N/A	N/A	N/A	N/A
Private READ	R/W Control	N/A	N/A	N/A	N/A
PUT8	R/W Control	Data8	Data8	N/A	N/A
PUTV8	R/W Control	Data8	Data8	Data8	Data8
PUT4	R/W Control	Data4	N/A	N/A	N/A
PUTV4	R/W Control	Data4	Data4	N/A	N/A
GET8	R/W Control	N/A	N/A	N/A	N/A
GETV8	R/W Control	N/A	N/A	N/A	N/A
GET4	R/W Control	N/A	N/A	N/A	N/A
GETV4	R/W Control	N/A	N/A	N/A	N/A
GET_INC	R/W/Atomic Op Control	N/A	N/A	N/A	N/A
GET_ADD	R/W/Atomic Op Control	Data4	N/A	N/A	N/A
SWAP8	Compare/ Swap (C/S) Control	Data8	Data8	N/A	N/A
CWSAP8	C/S Control	Data8	Data8	Data8	Data8
SWAP4	C/S Control	Data4	N/A	N/A	N/A
CSWAP4	C/S Control	Data4	Data4	N/A	N/A
GET_SET8	R/W Control	Data8	Data8	N/A	N/A
GET_SET4	R/W Control	Data4	N/A	N/A	N/A
GET_CLEAR8	R/W Control	Data8	Data8	N/A	N/A
GET_CLEAR4	R/W Control	Data4	N/A	N/A	N/A
MMR_READ	MMR Control	N/A	N/A	N/A	N/A
MMR_WRITE	MMR Control	Data4	N/A	N/A	N/A
MEM_MOVER	Copy R16 Control	Copy R16 Control	N/A	N/A	N/A

Table 2. C Option to R Option Packet Formats

Command	Flit 0	Flit 1	Flit 2	Flit 3	Flit 4	Flit 5	Flit 6	Flit 7	Flit 8
Request									
DGET	Head	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DPUT	Head	Body	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SEND	Head	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
PUT4,PUT8	Head	Body	N/A	N/A	N/A	N/A	N/A	N/A	N/A
PUTV4	Head	Words 0, 1	Words 2, 3	Words 4, 5	Words 6, 7	N/A	N/A	N/A	N/A
PUTV8	Head	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
GET	Head	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
GET_INC	Head	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
GET_ADD	Head	Body	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SWAP	Head	Body	N/A	N/A	N/A	N/A	N/A	N/A	N/A
CSWAP	Head	Comp-erand	Swap-erand	N/A	N/A	N/A	N/A	N/A	N/A
SPUT	Head	Body	N/A	N/A	N/A	N/A	N/A	N/A	N/A
SGET	Head	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Response									
GET, DGET, Atomic Op, and SEND	Flit 0 contains hardware error information, software error information and response data.								
SEND	Flit 0 contains hardware error information, software error (SW) information and no acknowledge.								
PUT, DPUT, SEND Acknowledge	No response packet for the PUT, DPUT, and SEND acknowledges; the TID is embedded in other flits.								
PUT, DPUT	SW Error	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
GETV4	Words 0, 1	Words 2, 3	Words 4, 5	Words 6, 7	N/A	N/A	N/A	N/A	N/A
GETV8	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	N/A

Memory Option

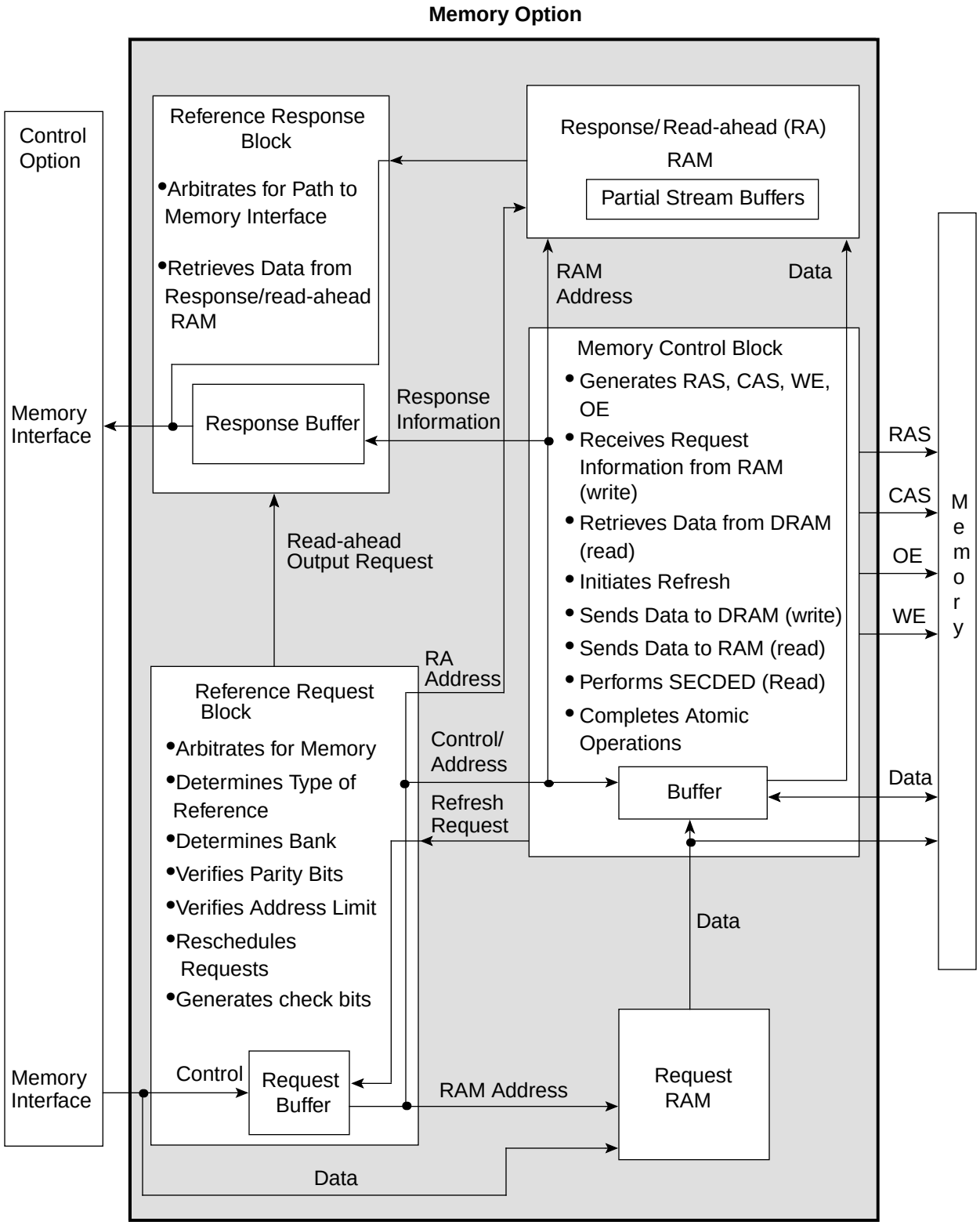
Each processing element contains four memory options. These memory options are physically located on the daughter cards; each daughter card printed circuit board houses one memory option. Each memory option controls the reads and writes for two banks of memory. (Each memory bank stores 64 bits of data.) Each memory option consists of the following components (refer to Figure 5).

- Reference request block - The reference request block performs the following functions:
  - Checks for parity errors, data errors, and address limit errors (When an error occurs, the reference request block writes the error information into the M_ERR register.)
  - Determines the type of reference request (read, write, or read-ahead)
  - Checks read-ahead buffers for requested data

    When the request is a write operation and data from the requested memory location is in the read-ahead buffer, the reference request block invalidates the read-ahead buffer. When the request is a read operation and data from the requested memory location is in the read-ahead buffer, the reference request block signals the response block to retrieve the data from the read-ahead buffer.
  - Determines which bank is being requested
  - Arbitrates for memory

    When a conflict occurs, the reference request block buffers the memory request information and reschedules the priority of the reference requests using to the reference priority scheme listed in Table 3.
  - Generates seven check bits for each half-word (32 bits) of data

Figure 5. Memory Option



NOTE: The memory options and the DRAM are physically located on the daughter cards.

Table 3. Reference Request Priority

Priority	Reference Description
1 (Highest)	Refresh requests
2	Request that resides in location 0 of the request buffer. This is only true after 8 requests have issued from the buffer.
3	In-page private read requests
4	In-page remote read requests
5	In-page read-ahead buffer requests
6	In-page atomic operations
7	In-page write requests
8	In-page copy read
9	In-page copy write
10	Out-of-page private read requests
11	Out-of-page remote read requests
12	Out-of-page read-ahead buffer requests
13	Out-of-page atomic operations
14	Out-of-page write requests
15	Out-of-page copy read
16	Out-of-page copy write

- Request random access memory (RAM) - The memory option stores data in the request RAM until the memory request has priority to reference memory.

- Memory control block - The memory control block generates the row address strobe (RAS), the column address strobe (CAS), the write enable (WE) for a write reference, and the output enable (OE) for a read reference. The memory control block sends the address and control signals to memory. The memory control block also sends data to memory during a write reference, receives data during a read reference, and initiates the refresh.

- Response/read-ahead RAM - The response/read-ahead RAM (also referred to as partial stream buffers) stores data from read operations and read-ahead operations. For a read operation, the response/read-ahead RAM stores the data until the data has priority to be sent to the C option. For a read-ahead operation, the response/read-ahead RAM stores the data until the microprocessor requests the data.

- Reference response block - The reference response block buffers response information and arbitrates for the path to the memory interface on the C option.

The memory option uses the packet formats shown in Table 4 to send requests and responses to the C option.

Table 4. M Option to C Option Packet Formats

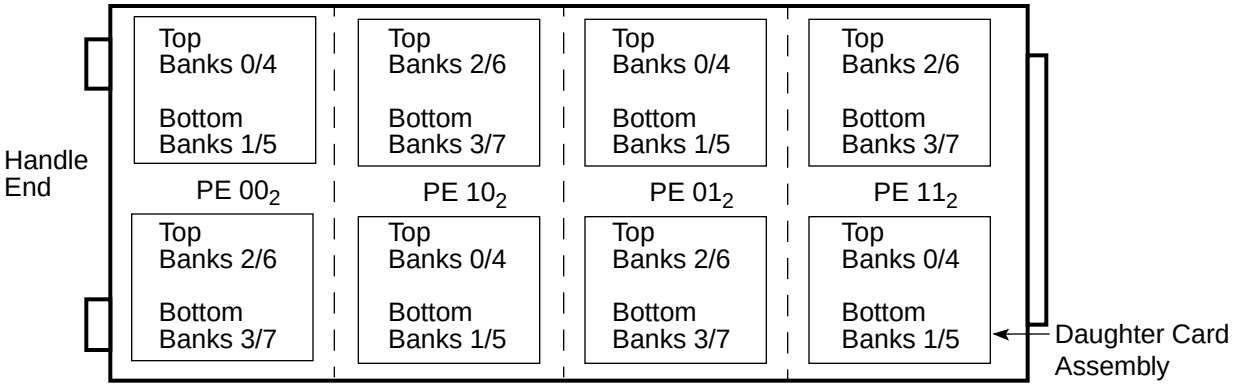
Command	Flit 0	Flit 1	Flit 2	Flit 3
IDLE	idle	N/A	N/A	N/A
DPUT	RESP0	N/A	N/A	N/A
MSG_WRITE	RESP0	RESP0	N/A	N/A
DGET	DGET_RESPA	DGET_RESPB	N/A	N/A
Private READ	RESP8A	RESP8B	RESP8A	RESP8B
PUT8	RESP0	N/A	N/A	N/A
PUTV8	RESP0	N/A	N/A	N/A
PUT4	RESP0	N/A	N/A	N/A
PUTV4	RESP0	N/A	N/A	N/A
GET8	RESP8A	RESP8B	N/A	N/A
GETV8	RESP8A	RESP8B	RESP8A	RESP8B
GET4	RESP4	N/A	N/A	N/A
GETV4	RESP4	RESP4	N/A	N/A
GET_INC	RESP4	N/A	N/A	N/A
GET_ADD	RESP4	N/A	N/A	N/A
SWAP8	RESP8A	RESP8B	N/A	N/A
CWSAP8	RESP8A	RESP8B	N/A	N/A
SWAP4	RESP4	N/A	N/A	N/A
CSWAP4	RESP4	N/A	N/A	N/A
GET_SET8	RESP8A	RESP8B	N/A	N/A
GET_SET4	RESP4	N/A	N/A	N/A
GET_CLEAR8	RESP8A	RESP8B	N/A	N/A
GET_CLEAR4	RESP4	N/A	N/A	N/A
MMR_READ	RESP4	N/A	N/A	N/A
MMR_WRITE	RESP0	N/A	N/A	N/A
MEM_MOVER	RESP0	RESP0	N/A	N/A

Local Memory

Local memory consists of DRAM integrated circuits (ICs) that are mounted on two daughter card assemblies. Each daughter card assembly consists of 2 printed circuit boards (PCBs) that are separated by a coldplate.

Local memory is divided into 8 banks. The top daughter card PCB contains the even banks, and the bottom daughter card PCB contains the odd banks (refer to Figure 6).

Figure 6. Daughter Cards on a Processing Element Module



The size of local memory varies depending on the type of DRAM ICs that are used in the system (refer to Table 5).

Table 5. Sizes of Local Memory

Size in Bytes	Size in 64-bit Words	Chip Select	Number of Memory Chip Stacks	ICs
64 Mbytes	8 Mwords	0	1	1M x 16
128 Mbytes	16 Mwords	0 and 1	1	1M x 16
256 Mbytes	32 Mwords	0	1	4M x 16
512 Mbytes	64 Mwords	0 and 1	1	4M x 16
1024 Mbytes	128 Mwords	0 and 1	2	8M x 8
2048 Mbytes	256 Mwords	0 and 1	4	8M x 8

Types of Memory Access

The microprocessor can access memory via two methods: the direct local memory access method and the E-register access method.

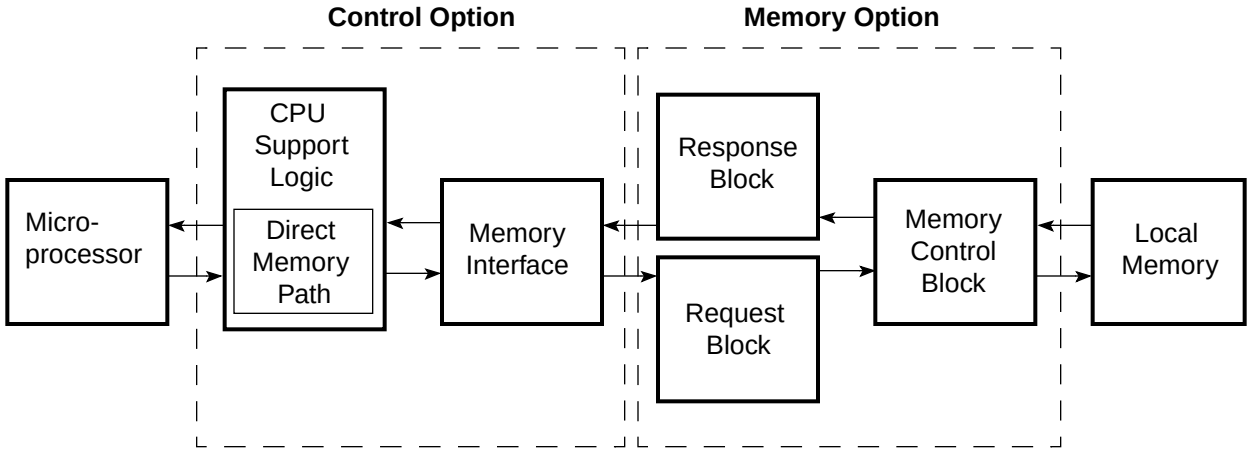
Direct Local Memory Access Method

The microprocessor uses the direct local memory access method as an automatic response to a cache update. For example, when software updates information in the secondary cache and the microprocessor must remove valid data from the secondary cache, the microprocessor creates a victim in order to make room for the new data. When the microprocessor has three victims, the microprocessor signals the CRAY T3E hardware to perform a direct local memory access store to update data in local memory. Likewise, when software requests data that is not in the secondary cache, the CRAY T3E hardware performs a direct local memory access load to write new data into the secondary cache.

With the direct local memory access method, the CPU support logic transfers the data, address, and control signals between the microprocessor and the memory interface by using the direct memory path (refer to Figure 7).

NOTE: The diagnostic tests may refer to the direct local memory access as a private data read or write.

Figure 7. Direct Local Memory Access Method



Example of a Direct Local Memory Write

The following text describes the steps in which a microprocessor performs a direct local memory write. The step numbers correspond to the numbers in Figure 8.

NOTE: All buffers and random access memory (RAMs) mentioned in this example perform internal parity checks of the data. To do this, the buffer generates parity for the incoming data. When the data leaves the buffer, the buffer checks the parity for errors. When a parity error occurs, the hardware discards the data and logs the error in the appropriate error register. For example, when an error is detected on the C option, the C option logs the error in the C_ERR0 register. When an error is detected on the M option, the M option logs the error in the M_ERR0 register. When an error is detected on the R option, the R option logs the error in the R_ERR0 register.

1. The microprocessor initiates a direct local memory write by sending a WRITE_BLOCK command to the support circuitry. The WRITE_BLOCK command indicates that the microprocessor needs to write data from the secondary cache to local memory.

2. The microprocessor also sends address, data, and control information to the CPU support logic in the control option. This information is protected by parity that is generated by the microprocessor (one parity bit per byte of data). The CPU support logic uses the parity to determine whether the channel between the microprocessor and the C option is functioning properly. When a parity error occurs on the channel, the CPU support logic discards the data and logs the error in bits 26, 27, or 28 of the C_ERR0 register. When the channel is error free, the CPU support logic of the control option uses the address and control to determine the type of command.

3. For a direct local memory write, the CPU support logic passes the address and data to the memory interface of the control option via the direct memory path. The CPU support logic also compares the address to the stream buffer addresses. When a match occurs, the CPU support logic invalidates the appropriate stream buffer location.

   The memory interface stores the write data and address in a first-in, first-out (FIFO) request buffer. After the data passes through the buffer, the memory interface arbitrates for the memory path to the daughter cards.

4. When the request has priority to use the memory path, the memory interface retrieves the data from the request buffer, generates parity for the data, and sends this data to the reference request block.

The reference request block checks for data and address parity errors, data errors, and address limit errors. When an error occurs, the reference request block writes the error information into bits 0, 1, or 2 of the M_ERR0 register.

In addition, the reference request block determines the type of request, determines which bank the microprocessor is requesting, and arbitrates for this bank in memory. When a conflict occurs, the reference request block buffers the memory address and control. The reference request block sends the data to the request RAM.

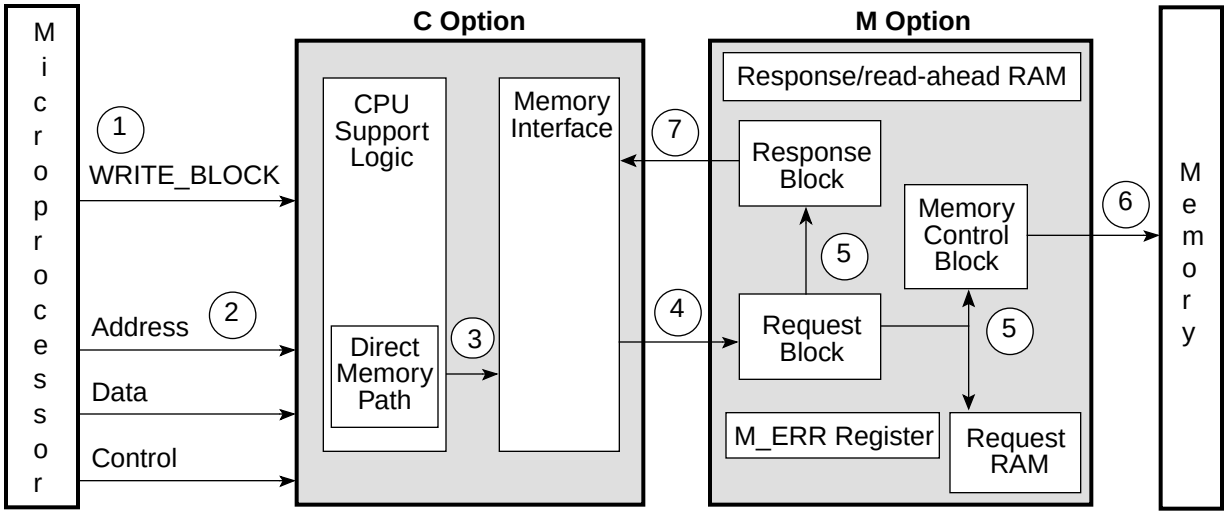
NOTE: While the reference information is buffered, the reference request block may use the request priority scheme listed in Table 3 to reschedule the priority of the request.

5. Once the request has priority to reference memory, the reference request block sends signals to the memory control block to indicate the bank and request type, generates a Write Acknowledge signal and sends it to the response block, and addresses the request RAM.

6. The memory control block receives the data from the request RAM, generates the RAS, CAS, and WE signals, and sends the address and control signals to local memory. After the memory control block addresses memory, the memory control block writes the data and check bits into memory.

7. When the Write Acknowledge signal has priority to access the path to the C option, the response block sends it to the memory interface.

Figure 8. Example of a Direct Local Memory Write



Example of a Direct Local Memory Read

The following text describes the steps in which a microprocessor performs a direct local memory read. The step numbers correspond to the numbers in Figure 9.

NOTE: All buffers and random access memory (RAMs) mentioned in this example perform internal parity checks of the data. To do this, the buffer generates parity for the incoming data. When the data leaves the buffer, the buffer checks the parity for errors. When a parity error occurs, the hardware discards the data and logs the error in the appropriate error register. For example, when an error is detected on the C option, the C option logs the error in the C_ERR0 register. When an error is detected on the M option, the M option logs the error in the M_ERR0 register. When an error is detected on the R option, the R option logs the error in the R_ERR0 register.

1. The microprocessor initiates the direct local memory read when the microprocessor does not find the requested data in the data cache or the secondary cache.To do this, the microprocessor sends a READ_MISS command to the support circuitry. The READ_MISS command instructs the support circuitry to read data from local memory.

2. The microprocessor also sends the address, control, and parity bits to the CPU support logic of the control option. The CPU support logic of the control option uses the parity bits to determine whether the address or the control contains a parity error. When a parity error occurs, the CPU support logic discards the data and logs the error in bit 26, 27, or 28 of the C_ERR0 register. When there are no parity errors, the CPU support logic uses the address and control to determine the type of command.

3. For a direct local memory read, the CPU support logic compares the address to the stream buffer addresses. When a match occurs, the CPU support logic reads the data from the stream buffer. When the address does not match an address in the stream buffer, the CPU support logic compares the address to the history table addresses. When a match occurs, the CPU support logic detects a stream. When a stream is detected and the stream buffers are enabled, the CPU support logic assigns the address to a stream buffer and passes the address to the memory interface by using the direct memory path. When the address does not match an address of the history buffer, the CPU support logic passes the address to the memory interface by using the direct memory path.

   The memory interface stores the address in a request buffer (FIFO). After the address passes through the buffer, the memory interface arbitrates for the memory path to the daughter cards.

4. When the request has priority to use the memory path, the memory interface retrieves the address from the request buffer, generates parity for the address, and sends the address and parity bits to the reference request block in the memory option.

   The reference request block checks the address for parity errors and address limit errors. When a parity error occurs, the reference request block logs the error in bit 0 of the M_ERR0 register. When an address range error occurs, the reference request block logs the error in bit 1 of the M_ERR0 register.

   In addition, the reference request block determines the type of request, determines which bank the microprocessor is requesting, and arbitrates for this bank in memory. When a conflict occurs, the reference request block buffers the memory address and control.

   NOTE: While the reference information is buffered, the reference request block may use the request priority scheme listed in Table 3 to reschedule the priority of the request.

5. Once the request has priority to reference memory, it sends signals to the memory control block to indicate the bank and request type.

6. The memory control block generates the RAS, CAS, and OE signals and sends the address and control signals to local memory.

7. After the memory control block addresses the memory, the memory control block reads the data from the memory and performs single-error correction, double-error detection (SECDED). When a single-bit error occurs, the memory control block corrects the bit and logs the error in bit 17, 18, 19, or 20 of the M_ERR0 register.

   NOTE: Bit 5 of the M_CFG register enables single-bit error correction.

   When a double- or multiple-bit error occurs, the memory control block discards the data and logs the error in bit 4, 5, 6, or 7 of the M_ERR0 register.

8. Next, the memory control block sends the read data to the response/read-ahead RAM. The memory control block also sends signals to the response block that indicate the response type. The reference response block buffers the signals while it arbitrates for the path to the memory interface in the control option.

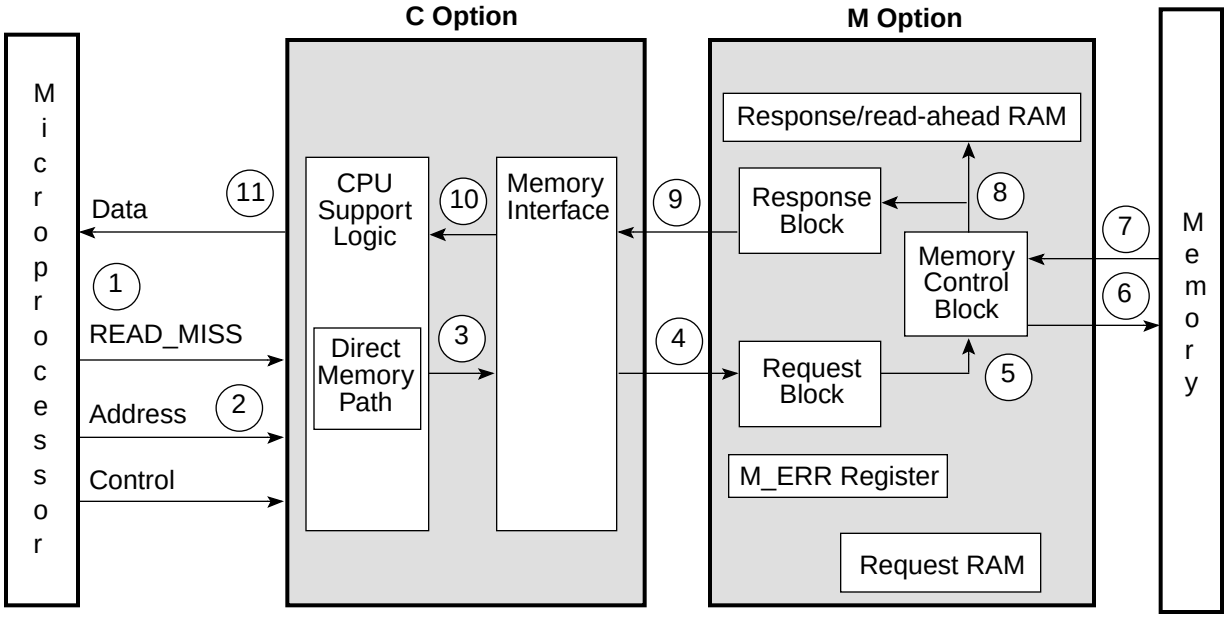
9. When the read data has priority to access the path, the reference response block retrieves the data from the response/read-ahead RAM, generates parity for the read data and sends the read data to the memory interface.

   The memory interface checks the data for parity errors. When a parity error occurs, the memory interface discards the data and logs the error in bit 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, or 11 of the C_ERR0 register. When the data is correct, the memory interface stores the read data in a response buffer (FIFO).

10. After the data passes through the response buffer, the memory interface sends the data to the CPU support logic.

11. The CPU support logic generates parity for the read data and sends the data to the microprocessor. The microprocessor checks the data for parity errors. When the data is correct, the software that is running in the microprocessor writes the data to a destination register and/or cache. When a parity error occurs, the software jumps to a machine check routine.

Figure 9. Example of a Direct Local Memory Read

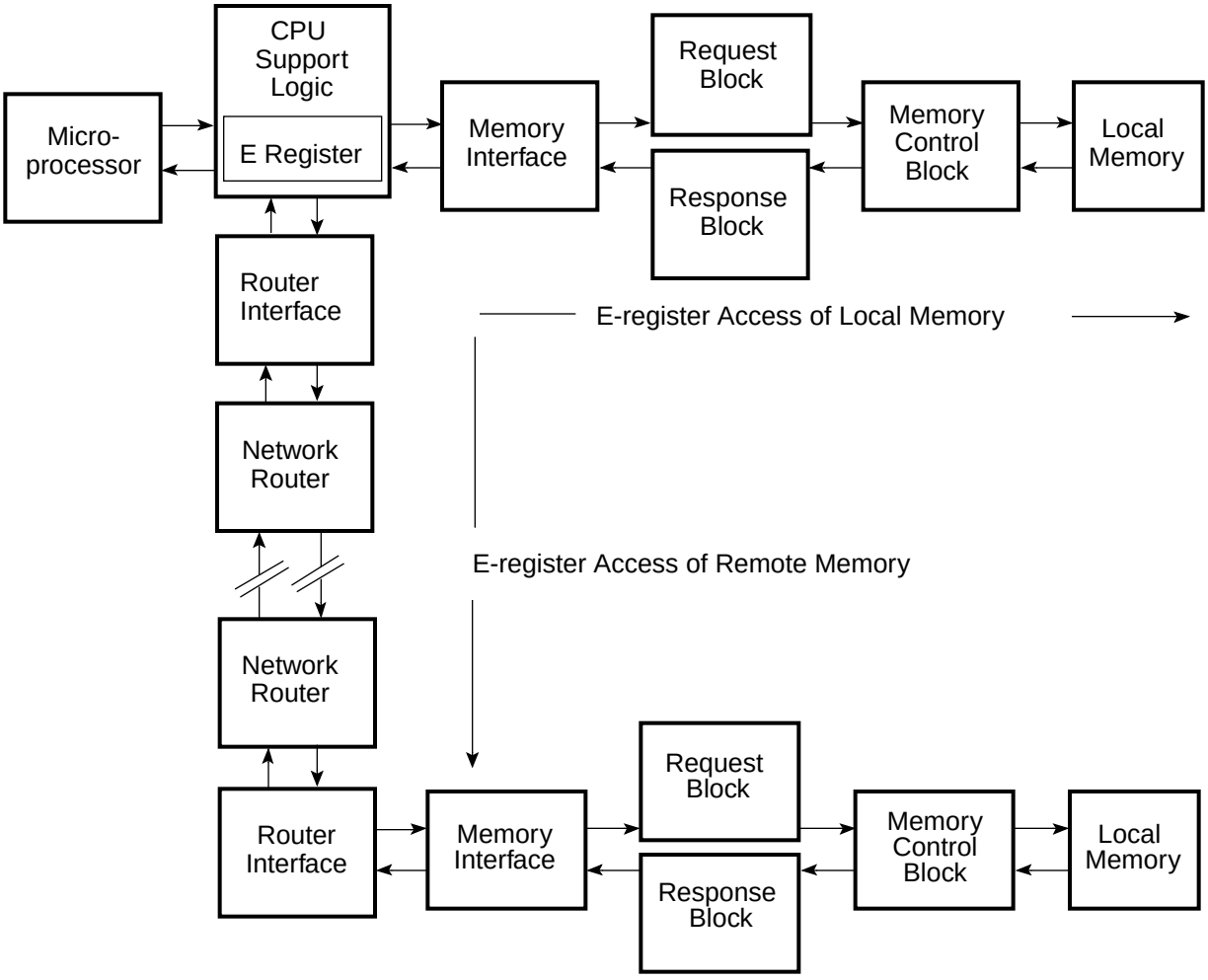


E-register Access Method

Software uses the E-register access method to transfer data to and from any location in global memory (refer to Figure 10). For example, to write data to a location in global memory, software issues a STORE command followed by a PUT command. The STORE command signals the support circuitry to transfer data from the microprocessor to an E register. The PUT command signals the support circuitry to transfer data from the E register to a location in global memory.

To read data from global memory, software issues a GET command followed by a LOAD command. The GET command signals the support circuitry to transfer data from a location in global memory to an E register in the issuing PE. The LOAD command signals the support circuitry to transfer data from the E register to the microprocessor.

Figure 10. E-register Access Method



Example of a Global Write Operation

The following text describes the steps in which a microprocessor performs a global write operation. The step numbers correspond to the numbers in Figure 11.

NOTE: All buffers and random access memory (RAMs) mentioned in this example perform internal parity checks of the data. To do this, the buffer generates parity for the incoming data. When the data leaves the buffer, the buffer checks the parity for errors. When a parity error occurs, the hardware discards the data and logs the error in the appropriate error register. For example, when an error is detected on the C option, the C option logs the error in the C_ERR0 register. When an error is detected on the M option, the M option logs the error in the M_ERR0 register. When an error is detected on the R option, the R option logs the error in the R_ERR0 register.

1. For a remote write, the microprocessor issues a STORE command followed by a PUT command. For more information on the STORE command and the PUT command, refer to the *Commands* document, publication number HMM-142-0.

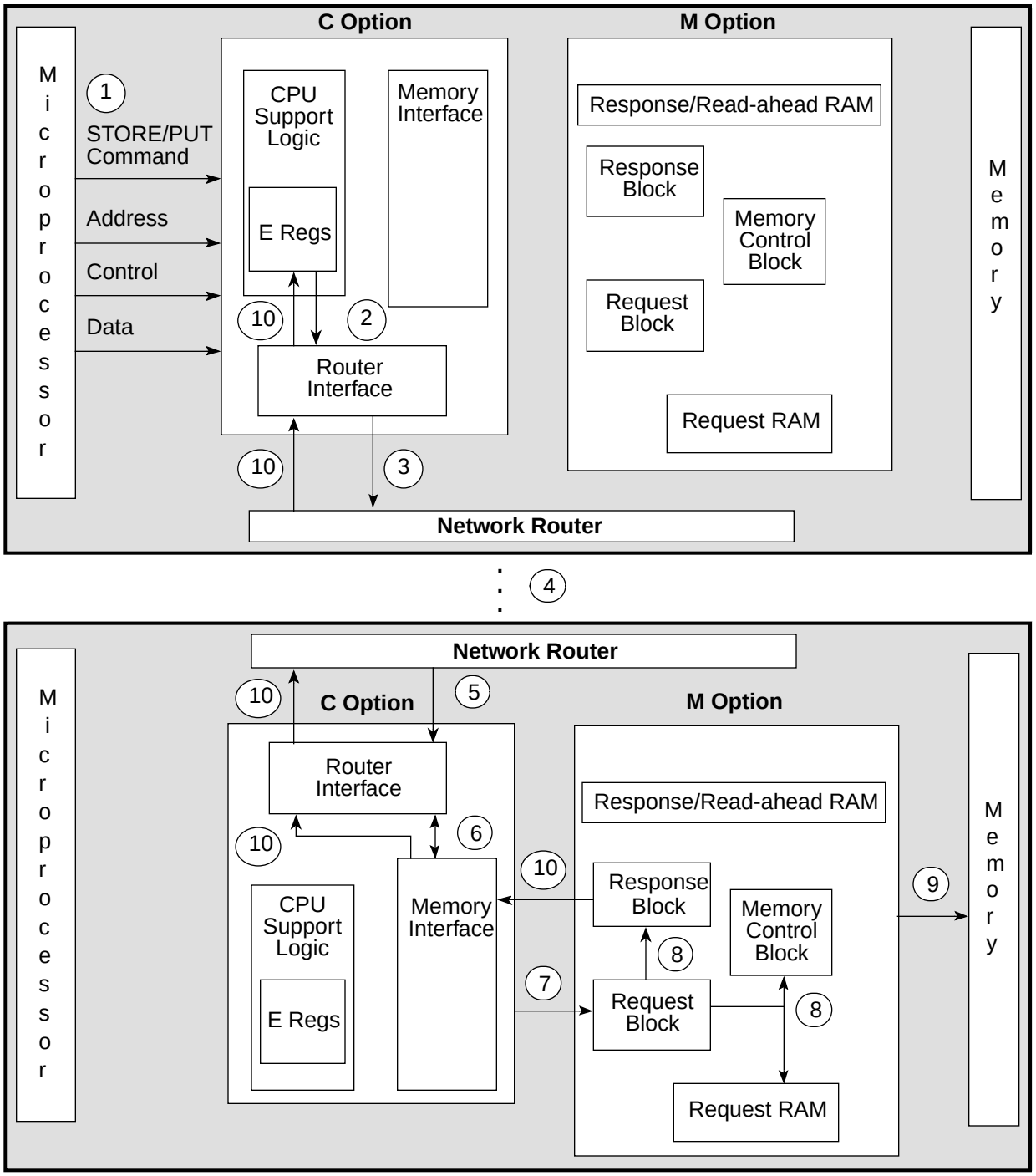
   The microprocessor initiates the STORE command by sending an address, data, and control information to the control option. This information is protected by parity that is generated by the microprocessor (one parity bit per byte of data). The CPU support logic uses the parity to determine whether the channel between the microprocessor and the C option is functioning properly. When a parity error occurs on the channel, the CPU support logic discards the data and logs the error in bits 26, 27, or 28 of the C_ERR0 register. When the channel is error free, the CPU support logic of the control option uses the address and control to determine the type of command. For an E-register STORE command, the CPU support logic writes the data to an E register that is specified by the address.

   The E register data is also protected by parity. When an E-register parity error occurs, the CPU support logic discards the data and logs the error in bit 20, 21, 22, or 23 of the C_ERR0 register.

   The microprocessor initiates the PUT command by sending an address, data, and control information to the control option. This information is protected by parity that is generated by the microprocessor (one parity bit per byte of data). When a parity error occurs on the channel, the CPU support logic discards the data and logs the error in bits 26, 27, or 28 of the C_ERR0 register. When the address, data, and control are correct, the CPU support logic of the control option uses the address and control to

determine the type of command.

Figure 11. Example of a Global Write Operation



2. For an E-register PUT command, the CPU support logic retrieves data from an E register that the address specifies, generates parity for the data, and sends this data along with the address and control to the router interface or the memory interface. The CPU support logic sends the data to the router interface when the destination of the data is a remote memory location. The CPU support logic sends the data to the memory interface when the destination of the data is a local memory location.

   The router interface buffers the outgoing request in a virtual channel buffer (FIFO) and arbitrates for the path to the network router.

3. Once the information has priority to use the path, the router interface retrieves the information from the virtual channel buffer, generates parity for the address, data, and control, and sends this information to the network router.

4. The network router checks the address, data, and control for parity errors. When the network router detects a parity error, the network router discards the data and logs the error in bit 30 of the R_ERR0 register. When the address, data, and control are correct, the network router generates check bits for the packet and passes the packet to the next network router in the interconnect network.

   Each network router that receives the packet checks the packet for single-, double-, and multiple-bit errors. When a single-bit error occurs, the network router corrects the bit and passes the packet to the next network router. When a double- or multiple-bit error occurs, the network router logs the error as a parity error in bit 24, 25, 26, 27, 28, or 29 of the R_ERR0 register and passes the packet to the next network router. When the data is correct, the network router passes the packet to the next network router.

5. When the packet reaches the destination node and the packet contains a double- or multiple-bit error, the network router discards the packet and sets bit 58 of the R_ERR0 register. When the packet reaches the destination node without any errors, the network router of the destination node generates parity for the packet and passes the packet to the router interface.

6. The router interface checks the packet for parity errors. When a parity error occurs, the router interface discards the packet and logs the error in bit 16 of the R_ERR0 register. When the packet is correct, the router interface stores the packet in a first-in, first-out (FIFO) request buffer of the control option and signals the memory interface that an incoming request arrived.

   The memory interface sends control signals to the router interface of the control option to retrieve the request information from the FIFO request buffer. When the memory interface receives the request information, it stores the information in a FIFO request buffer.

   After the data passes through the buffer, the memory interface converts the global virtual address to a physical address. The memory interface compares the physical address to the addresses of the backmap. When the address does not match an address of the backmap, the memory interface arbitrates for the memory path to the daughter cards. When a match occurs, the memory interface sends a flush request to the microprocessor.

   When the microprocessor receives the flush request and the requested data is dirty, the microprocessor invalidates the cacheline of data and flushes (writes) it to local memory. When the microprocessor receives a flush request and the requested data is not dirty, the microprocessor invalidates the cacheline of data but does not write it to local memory.

   After the flush request completes (this includes the write to local memory), the memory interface arbitrates for the memory path to the daughter cards.

7. When the request has priority to use the memory path, the memory interface retrieves the request information from the request buffer, generates parity for this information, and sends the request information and parity bits to the reference request block in the memory option.

   The reference request block checks the request information for parity errors, data errors, and address limit errors. When an error occurs, the reference request block discards the data and logs the error in bit 0, 1, or 2 of the M_ERR register.

   When there are no errors, the reference request block determines the type of request, determines which bank the source PE is requesting, and arbitrates for this bank in memory. When a conflict occurs, the reference request block buffers the memory address and control. The reference request block sends the data to the request RAM.

   NOTE: While the reference information is buffered, the reference request block may use the request priority scheme listed in Table 3 to reschedule the priority of the request.

8. Once the request has priority to reference memory, the reference request block sends signals to the memory control block to indicate the bank and request type, generates a Write Acknowledge signal and sends it to the response block, and addresses the request RAM.

9. The memory control block receives the data from the request RAM, generates the RAS, CAS, and WE signals, and sends the address and control signals to memory. After the memory control block addresses memory, the memory control block writes the data and check bits to memory.

10. When the Write Acknowledge signal has priority to access the path to the C option, the response block sends it to the memory interface. The signal passes through the memory interface, the router interface, and the network router to the source PE. The source PE receives this signal as a response.

Example of a Global Read Operation

The following text describes the steps in which a microprocessor performs a global read operation. The step numbers correspond to the numbers in Figure 12 and Figure 13.

NOTE: All buffers and random access memory (RAMs) mentioned in this example perform internal parity checks of the data. To do this, the buffer generates parity for the incoming data. When the data leaves the buffer, the buffer checks the parity for errors. When a parity error occurs, the hardware discards the data and logs the error in the appropriate error register. For example, when an error is detected on the C option, the C option logs the error in the C_ERR0 register. When an error is detected on the M option, the M option logs the error in the M_ERR0 register. When an error is detected on the R option, the R option logs the error in the R_ERR0 register.

1. For a global read, the microprocessor issues a GET command followed by a LOAD command. For more information on the GET command and the LOAD command, refer to the *Commands* document, publication number HMM-142-0.

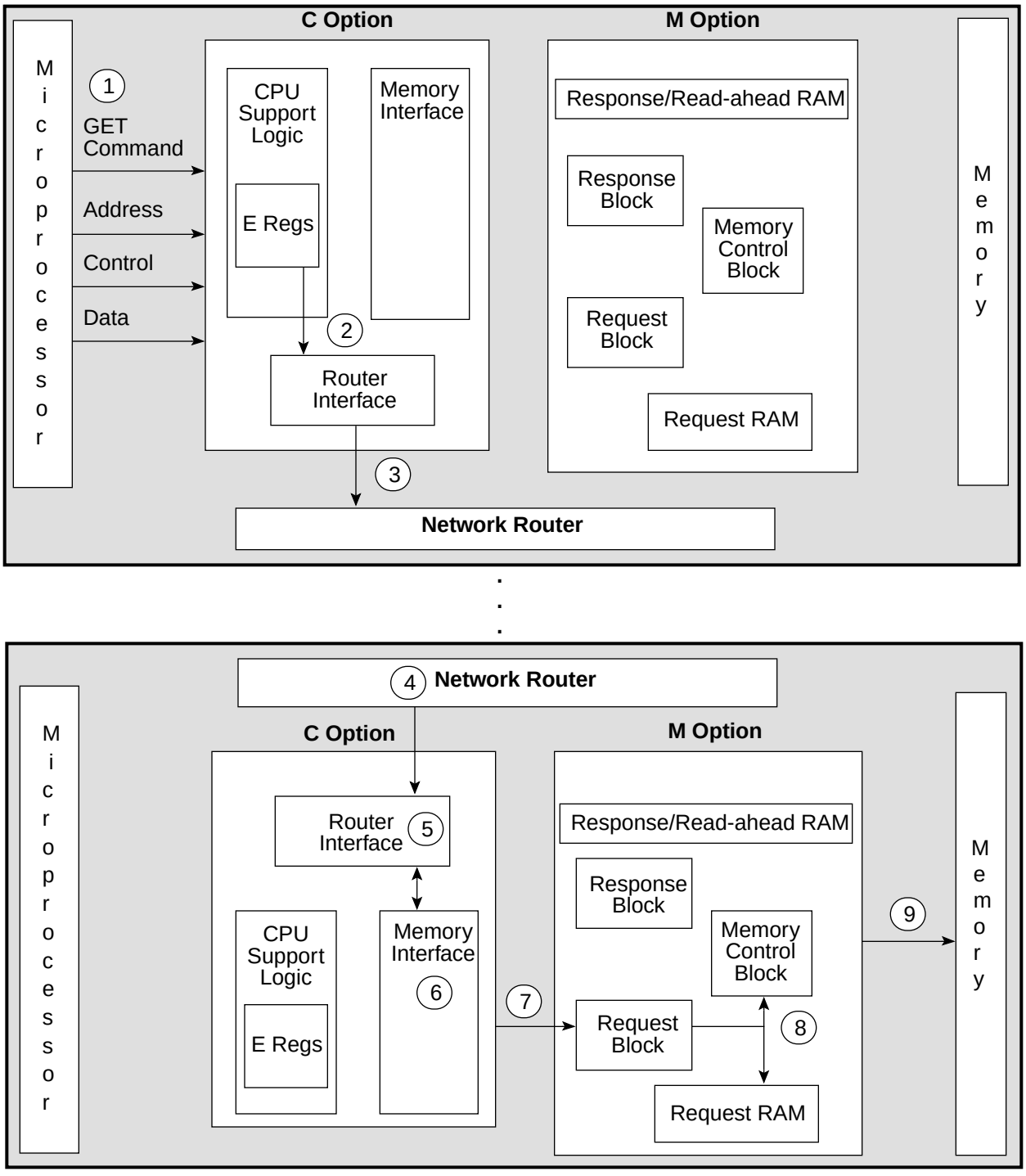
   The microprocessor initiates the GET command by sending an address, data, control, and parity bits to the control option. The CPU support logic uses the parity to determine whether the channel between the microprocessor and the C option is functioning properly. When a parity error occurs on the channel, the CPU support logic discards the data and logs the error in bit 26, 27, or 28 of the C_ERR0 register. When the channel is error free, the CPU support logic of the control option uses the address and control to determine the type of command.

2. For a remote GET command, the CPU support logic sends the address and control signals to the router interface.

   The router interface buffers the outgoing request in a virtual channel buffer (FIFO). After the information passes through the buffer, the router interface arbitrates for the path to the network router.

   Once the information has priority to use the path, the router interface retrieves the information from the virtual channel buffer, generates parity for the information, and sends it to the network router.

Figure 12. Example of a Global Read Operation (Part 1)



3. The network router checks the data for parity errors. When the network router detects a parity error, the network router discards the data and logs the error in bit 30 of the R_ERR0 register. When the data is correct, the network router generates check bits for the packet and passes the packet to the next network router in the interconnect network.

   Each network router that receives the packet checks the packet for single-, double-, and multiple-bit errors. When a single-bit error occurs, the network router corrects the bit and passes the packet to the next network router. When a double- or multiple-bit error occurs, the network router logs the error in bit 24, 25, 26, 27, 28, or 29 of the R_ERR0 register and passes the packet to the next network router. When the packet is correct, the network router passes the packet to the next network router.

4. When the packet reaches the destination node and the packet contains a double- or multiple-bit error, the network router discards the packet. When the packet reaches the destination node without any errors, the network router of the destination node generates parity for the packet and passes the packet to the router interface in the control option.

5. The router interface checks the packet for parity errors. When a parity error occurs, the router interface discards the data and logs the error in bit 16 of the R_ERR0 register. When the packet is correct, the router interface stores the packet in a FIFO request buffer and signals the memory interface that an incoming request arrived. The memory interface sends control signals to the router interface to retrieve the request information from the request buffer.

6. The memory interface stores the request information in a FIFO request buffer. After the data passes through the buffer, the memory interface converts the global virtual address to a physical address. The memory interface compares the physical address to the addresses of the backmap. When the address does not match an address of the backmap, the memory interface arbitrates for the memory path to the daughter cards. When a match occurs, the memory interface sends a flush request to the microprocessor.

   When the microprocessor receives the flush request and the requested data is dirty, the microprocessor invalidates the cacheline of data and flushes it to local memory. When the microprocessor receives a flush request and the requested data is not dirty, the microprocessor invalidates the cacheline of data but does not write it to local memory.

   After the flush request completes (this includes the write to local memory), the memory interface arbitrates for the memory path to the daughter cards.

7. When the request has priority to use the memory path, the memory interface retrieves the request information from the request buffer, generates parity for this information, and sends the request information and parity bits to the reference request block in the memory option.

   The reference request block checks the request information for parity errors and address limit errors. When an error occurs, the reference request block logs the error in bit 0 or bit 1 of the M_ERR0 register.

   The reference request block also determines the type of request, determines which bank the source PE is requesting, and arbitrates for this bank. When a conflict occurs, the reference request block buffers the memory address and control.

   NOTE: While the reference information is buffered, the reference request block may use the request priority scheme listed in Table 3 to reschedule the priority of the request.

8. Once the request has priority to reference memory, the reference request block sends signals to the memory control block to indicate the bank, request type, and memory address.

9. The memory control block generates the RAS, CAS, and OE signals and sends the address and control signals to memory.

10. After the memory is addressed, the memory control block reads the data from memory and checks for single-, double-, and multiple-bit errors (refer to Figure 13). When a single-bit error occurs, the memory control block corrects the bit and logs the error in bit 17, 18, 19, or 20 of the M_ERR0 register. When a double- or multiple-bit error occurs, the memory control block discards the data and logs the error in bit 4, 5, 6, or 7 of the M_ERR0 register.

11. When the data is free of errors, the memory control block sends the read data to the response/read-ahead RAMs. The memory control block also sends a signal to the response block, indicating the response type. The reference response block buffers the signals while it arbitrates for the path to the memory interface in the control option.

12. When the read data has priority to access the path, the reference response block generates parity for the data and sends the read data to the memory interface.

13. The memory interface checks the incoming data for parity errors. When a parity error occurs, the memory interface discards the data and logs the error in bit 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, or 11 of the C_ERR0 register. When the data is correct, the memory interface stores the read data in a response buffer (FIFO). After the information passes through the buffer, the memory interface sends the data to the router interface.

14. The router interface buffers the outgoing response information in a virtual channel buffer (FIFO). After the information passes through the buffer, the router interface arbitrates for the path to the network router. Once the information has priority to use the path, the router interface retrieves the information from the virtual channel buffer, generates parity for the information, and sends it to the network router.

15. The network router checks the data for parity errors. When the network router detects a parity error, the network router discards the data and logs the error in bit 30 of the R_ERR0 register. When the data is correct, the network router generates check bits for the packet and passes the packet to the next network router in the interconnect network.

16. Each network router that receives the packet checks the packet for single-, double-, and multiple-bit errors. When a single-bit error occurs, the network router corrects the bit and passes the packet to the next network router. When a double- or multiple-bit error occurs, the network router logs the error in bit 24, 25, 26, 27, 28, or 29 of the R_ERR0 register and passes the packet to the next network router. When the data is correct, the network router passes the packet to the next network router. When the packet reaches the destination node and the packet contains a double- or multiple-bit error, the network router discards the packet. When the packet reaches the destination node without any errors, the network router of the destination node generates parity for the packet and passes the packet to the router interface.

17. The router interface checks the incoming packet for parity errors. When a parity error occurs, the router interface discards the data and logs the error in bit 16 of the C_ERR0 register. When the packet is correct, the router interface stores the incoming packet in a response buffer and signals the CPU support logic that the read response packet arrived.

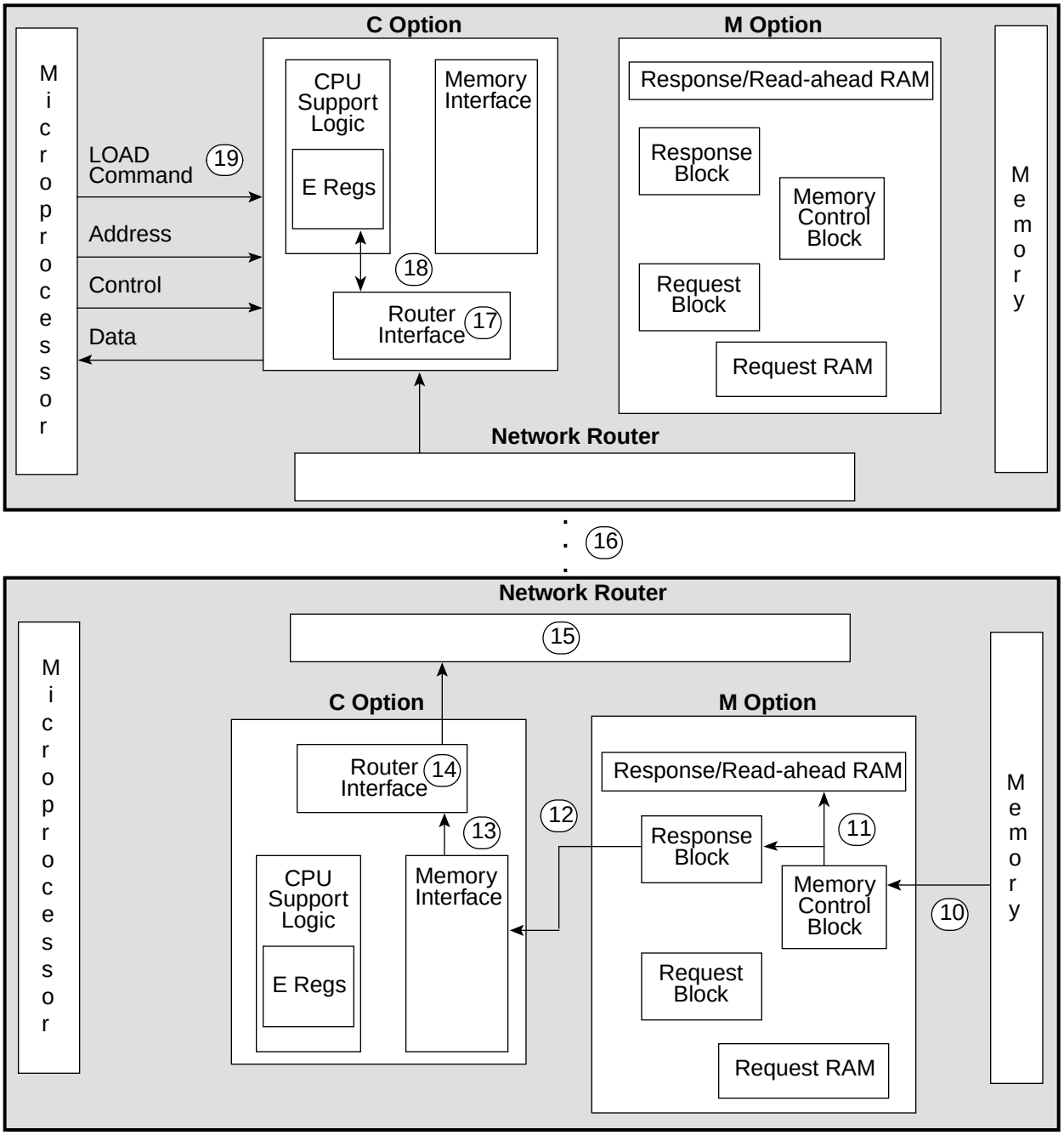
18. The CPU support logic sends control signals to the router interface to retrieve the response packet from the response buffer. The CPU support logic stores the read data in the specified E register.

19. The microprocessor initiates the LOAD command by sending an address and control to the control option. The CPU support logic of the control option checks this information for parity errors. When a parity error occurs, the CPU support logic discards the address and control and logs the error in bit 26, 27, or 28 of the C_ERR0 register. When the address and control are correct, the CPU support logic uses the address and control to determine the type of command.

    For a LOAD command, the CPU support logic retrieves data from an E register specified by the address. The E register data is protected by parity. When an E-register parity error occurs, the CPU support logic discards the data and logs the error in bit 20, 21, 22, or 23 of the C_ERR0 register. When the data is correct, the CPU support logic generates parity for the data and sends this data to the microprocessor.

    The microprocessor checks the data for parity errors. When the data is correct, the software that is running in the microprocessor writes the data to a destination register and/or cache. When a parity error occurs, the software jumps to a machine check routine.

Figure 13. Example of a Global Read Operation (Part 2)



Example of Four PEs Accessing Local and Global Memory

Table 6 contains an explanation of how four PEs in a partition work together to solve a user application. This user application is a program that calculates the year-end financial result of a company by performing the following steps. The step numbers correspond to the numbers in Figure 14.

1. The application distributes the financial data within the partition so that virtual PE 0 computes the first-quarter results, virtual PE 1 computes the second-quarter results, virtual PE 2 computes the third-quarter results, and virtual PE 3 computes the fourth-quarter results.

   Using a direct local memory read, each microprocessor retrieves its portion of the application from the user segment of memory and performs the computation.

2. Once the PEs complete their computations, virtual PEs 1 through 3 perform global write operations to send their quarterly results to the shared segment of the local memory for virtual PE 0.

3. Using a direct local memory read, virtual PE 0 retrieves the quarterly results from memory and adds the quarterly results together.

4. Virtual PE 0 distributes the year-end result to the other PEs in the partition by performing a global write to their shared segment of memory.

NOTE: In order to synchronize the PEs in this partition, software uses the barrier synchronization polled method. In addition, software assigns this partition to B/E context 1.

Figure 14. Virtual PEs 0 through 3 Computing Quarterly Results

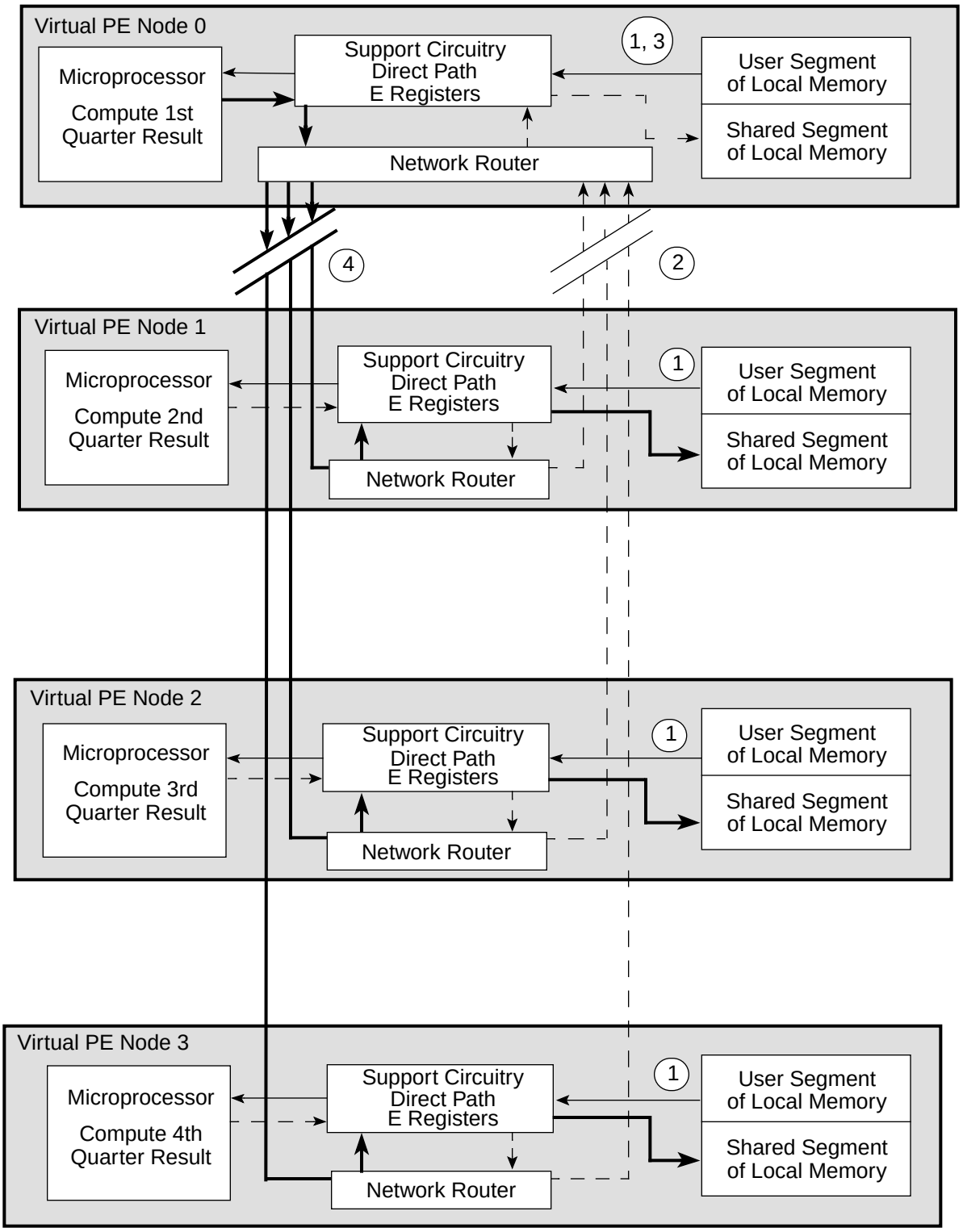


Table 6. A Partition Calculates Company's Year-end Financial Result

Virtual PE 0	Virtual PE 1	Virtual PE 2	Virtual PE 3
Retrieve 1st-quarter earnings from the user segment of local memory	Retrieve 2nd-quarter earnings from the user segment of local memory	Retrieve 3rd-quarter earnings from the user segment of local memory	Retrieve 4th-quarter earnings from the user segment of local memory
Compute 1st-quarter result	Compute 2nd-quarter result	Compute 3rd-quarter result	Compute 4th-quarter result
Signal B/E circuitry to wait for a barrier	Issue STORE command to write the 2nd-quarter result to an E register	Issue STORE command to write the 3rd-quarter result to an E register	Issue STORE command to write the 4th-quarter result to an E register
B/E Circuitry sends control information to the root node	Issue PUT command to transfer 2nd-quarter result from the E register to the local memory of virtual PE 0 (shared segment)	Issue PUT command to transfer 3rd-quarter result from the E register to the local memory of virtual PE 0 (shared segment)	Issue PUT command to transfer 4th-quarter results from the E register to the local memory of virtual PE 0 (shared segment)
Enter program loop that periodically reads the state identification number of the B/E circuit	Signal B/E circuitry to wait for a barrier	Signal B/E circuitry to wait for a barrier	Signal B/E circuitry to wait for a barrier
	B/E circuitry sends control information to the root node	B/E circuitry sends control information to the root node	B/E circuitry sends control information to the root node
	Enter program loop that periodically reads the state identification number of the B/E circuit	Enter program loop that periodically reads the state identification number of the B/E circuit	Enter program loop that periodically reads the state identification number of the B/E circuit

NOTE: When the root node receives the barrier control information from all of the nodes within the partition, the root node notifies all the PEs in the partition that a barrier event occurred. After this notification, the B/E circuitry changes the state identification number from 100 (wait for barrier) to 110 (barrier complete state). When the state identification number changes to the barrier complete state, the microprocessors continue with other program instructions.

Table 6. A Partition Calculates Company's Year-end Financial Result (continued)

Virtual PE 0	Virtual PE 1	Virtual PE 2	Virtual PE 3
Issue three load instructions to retrieve the quarterly results from local memory	Signal B/E circuitry to wait for a barrier	Signal B/E circuitry to wait for a barrier	Signal B/E circuitry to wait for a barrier
Add quarterly results	B/E circuitry sends control information to the root node	B/E circuitry sends control information to the root node	B/E circuitry sends control information to the root node
Issue STORE command to write the year-end result to an E register	Enter program loop that periodically reads the state identification number of the B/E circuit	Enter program loop that periodically reads the state identification number of the B/E circuit	Enter program loop that periodically reads the state identification number of the B/E circuit
Issue four PUT commands to write the year-end result to its own local memory and to the local memory of the other PEs (shared segment)			
Signal B/E circuitry to wait for a barrier			
B/E Circuitry sends control information to the root node			
Enter program loop that periodically reads the state identification number of the B/E circuit			

NOTE: When the root node receives the barrier control information from all of the nodes within the partition, the root node notifies all the PEs in the partition that a barrier event occurred. After this notification, the B/E circuitry changes the state identification number from 100 (wait for barrier) to 110 (barrier complete state). When the state identification number changes to the barrier complete state, the microprocessors continue with other program instructions.

Address Translation

An application generates a virtual address that must be converted into a physical address before the support circuitry can use it to address memory. You will need to know the following terms to fully understand this address translation.

Base virtual address - The base virtual address is added to an index offset to create the final user virtual address or global virtual address. The base virtual address is stored in the MO block of E registers.

Centrifuge - The centrifuge is a component in the support circuitry that separates the global index into a PE number (virtual PE number or logical PE number) and an index offset.

Centrifuge mask - The centrifuge mask is a software-defined value that indicates to the centrifuge which bits of the global index are the PE number and which bits of the global index are the index offset.

Global memory - Global memory is made up of the local memory of all PEs in the CRAY T3E system. You can calculate the global memory size of a CRAY T3E system by adding the local memory of all PEs.

Global segment - Global segments can be one of four sizes: 256 Mbytes, 512 Mbytes, 1 Gbyte, and 2 Gbytes. The size of the global segment, which is usually equal to or greater than the size of local memory, is determined by the GTB_CTL register. A global segment is further divided into a variable number of 512 Kbyte global pages. The number of global pages is determined by the global segment size. Each global page or contiguous global pages map to physical pages.

Global translation array (GTA) - The global translation array is a software-defined look-up table that is stored in the local memory of each PE and used to translate global virtual addresses into physical addresses. The GTB_CTL register bits <5 : 4> determine the size of the global segments (refer to Table 7). The GTB_EA_BASE register specifies the table's starting location in memory.

Table 7. Sizes of the Global translation Array Table

GTB_CTL Bits <5 : 4>	Global Segment Size	Global Page Per Segment	Total Number of Global Pages	Global Translation Array Size
00	256Mbytes	512	32,769	128 Kbytes
01	512 Mbytes	1024	65,536	256 Kbytes
10	1 Gbyte	2048	131,072	512 Kbytes
11	2 Gbytes	4096	262,144	1 Mbyte

GTA entry - Each GTA entry contains a page frame number, page size mask, small page bit, and a valid bit.

Global translation buffer (GTB) - The GTB is a buffer in the support circuitry that stores frequently or recently accessed global translation array entries.

Global virtual address -The global virtual address is an intermediate address that the operating system uses to reference different types of data in the CRAY T3E system memory space.

Index offset - The index offset is a value that is added to a base virtual address to generate the final user virtual address or global virtual address.

More operands (MO) pointer - The MO pointer is a field on the microprocessor data bus that indicates the location of the MO block of E registers when the microprocessor is requesting a global data transfer.

MO block of E registers - The MO block of E registers is a set of 4 contiguous E registers that contain additional information for an E register command. For example, the MO block of E registers may contain the centrifuge mask, the base virtual address, a comperand value, and a swaperand value.

Page frame number - The page frame number combined with the segment offset forms the 31-bit physical address.

Page size mask - The page size mask indicates which bits of the page frame number and which bits of the segment offset make up bits <26 : 16> of the physical address (refer to Table 8).

Table 8. Page Size Mask

Page Size Mask Bits	Physical Page Size
00000000000	128 Mbytes
10000000000	64 Mbytes
11000000000	32 Mbytes
11100000000	16 Mbytes
11110000000	8 Mbytes
11111000000	4 Mbytes
11111100000	2 Mbytes
11111110000	1 Mbytes
11111111000	512 Kbytes
11111111100	256 Kbytes
11111111110	128 Kbytes
11111111111	64 Kbytes

Physical page - A physical page is a portion of local memory and can vary in size from 64 Kbytes to 128 Mbytes. The page size is specified by the page size mask in the global translation array table entry. Small pages (64 Kbytes to 256 Kbytes) map to the upper or lower portion of the 512-Kbyte physical page and there can only be one small page per 512-Kbyte page.

Segment Offset - The segment offset is a component of the user virtual address or global virtual address that references a byte of data in a user segment or a global segment.

Segment Translation Table (STT) - The STT is a component in the support circuitry that translates a user segment into a global segment.

Small page bit - The small page bit indicates whether the small page is located in the upper or lower portion of the 512-Kbyte global page.

User segment - The user virtual address is divided into 8 equal address spaces. A user segment is one of these 8 address spaces.

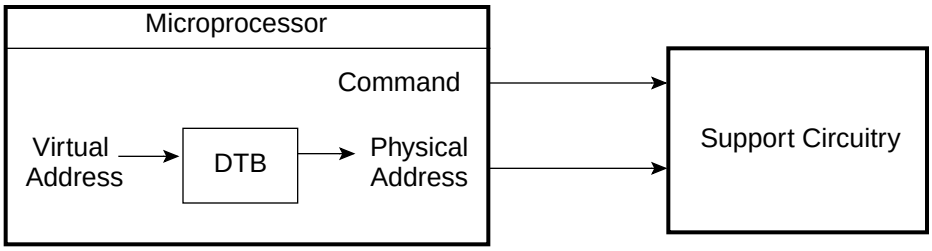
User virtual address - The user virtual address is a byte-oriented address that the program compiler generates and that references different types of data in an application's address space.

Valid bit - A valid bit indicates that the GTA entry contains valid data.

Virtual Address Space - Software maps the local memory of each PE into virtual address space. The virtual address space of a PE is divided into 64 global segments.

For the direct local memory access method, the microprocessor translates the virtual address into a physical address by using a data translation buffer (DTB) (refer to Figure 15). Software loads the DTB with the appropriate translation parameters.

Figure 15. Data Translation Buffer



The E-register access method can be used for local memory or global memory references. For a local memory reference, the control option receives a local index from the microprocessor and for a global memory reference, the control option receives a global index from the microprocessor. The index-type bit of the opcode indicates to the control option whether the index is local or global (1 = local, 0 = global).

Global Memory Reference

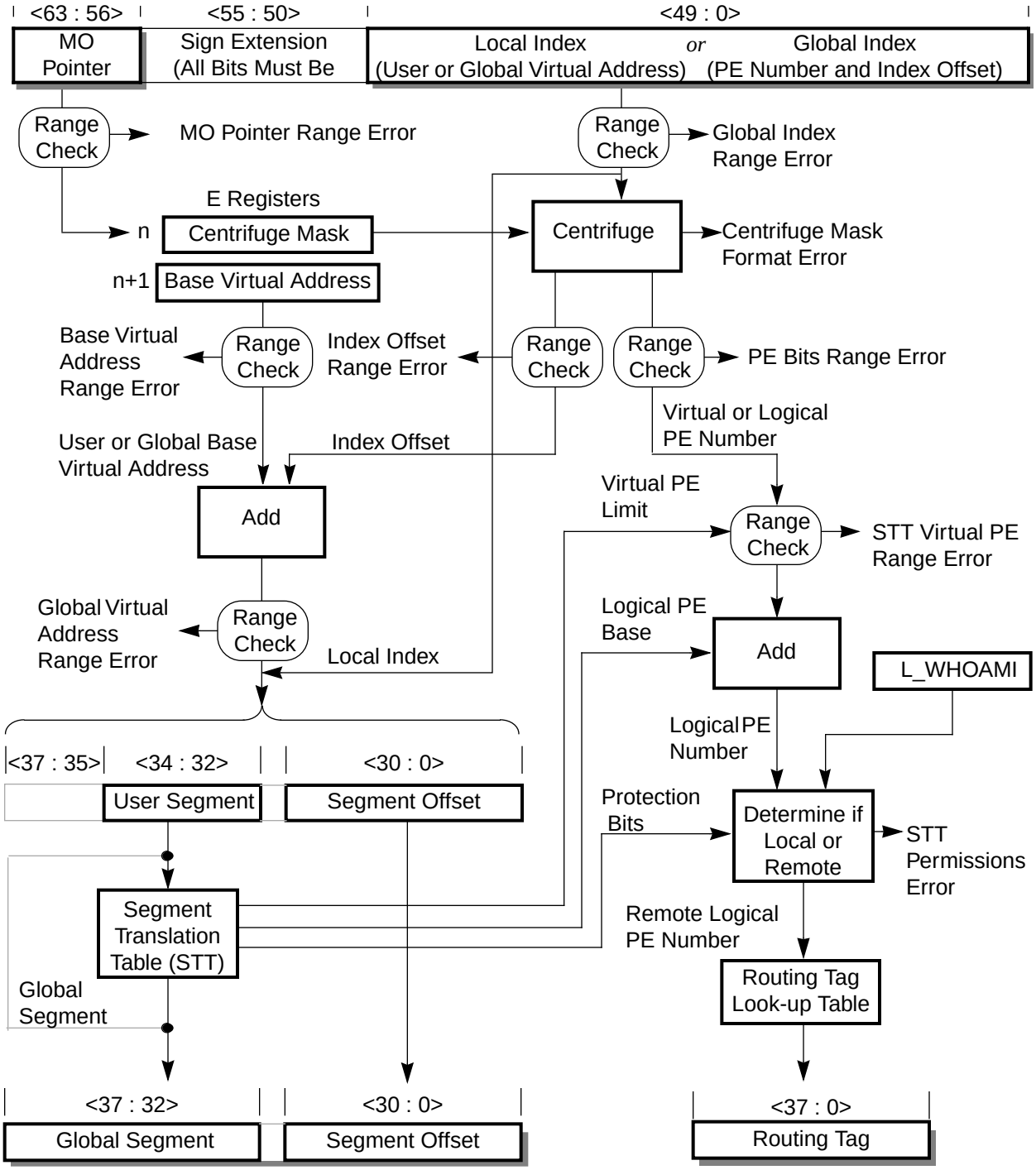
For a global memory reference, the control option translates the global index (software formatted address) into a logical PE number, global segment, and segment offset. To do this, the control option uses the global index and MO pointer it receives from the microprocessor (refer to Figure 16). The control option reads a centrifuge mask and base virtual address (global index format parameters) from the E registers identified by the MO pointer. The control option uses the centrifuge mask to separate the global index into a PE number and an index offset.

The PE number can be a logical PE number or a virtual PE number; the STT bypass bit indicates whether the PE number is a logical PE number or a virtual PE number (1 = logical PE number, 0 = virtual PE number). When the PE number is a virtual PE number, the control option adds the virtual PE number to the base logical PE number to create a logical PE number.

The control option uses the logical PE number to determine whether the reference is local or remote. When the reference is remote, the network router option uses the PE number to address a routing tag look-up table. From this look-up table, the network router option retrieves a routing tag, which the interconnect network uses to steer the packet to the destination PE.

The control option adds the index offset to the base virtual address to create a user or global virtual address (STT bypass bit = 1 - global virtual address, STT bypass bit = 0 - user virtual address). When the control option creates a user virtual address, the control option converts the user virtual address into a global virtual address using the segment translation table. The control option sends the global virtual address to the destination PE (local or remote).

Figure 16. Global Address Translation



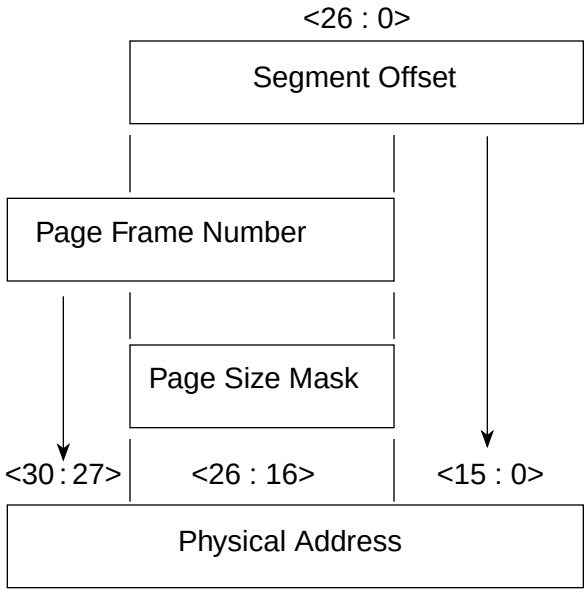
At the destination PE, the support circuitry uses the GTA and/or the GTB to convert the global virtual address (global segment and segment offset) into a physical address. A GTA entry contains a page frame number, page size mask, small page bit, and a valid bit. The GTB stores the most recently accessed GTA entries along with the appropriate GTA-entry address tags (refer to Figure 17).

Figure 17. GTB Entry

	<31 : 28>	27	26	<25 : 15>	<14 : 0>
Address Tag	Should be Zero (SBZ)	Valid Bit	Small Page	Page Size Mask	Page Frame Number

The page size mask contains a mask that the control option uses to merge the segment offset and the page frame number; this merger creates the physical address (refer to Figure 18).

Figure 18. Physical Address Generation



Local Memory Reference

For a local memory reference, the control option receives a local index from the microprocessor via the data bus. This address is either a user virtual address or a global virtual address. The STT bypass bit indicates whether the local index is a user virtual address or a global virtual address (0 = user, 1 = global)

When the local index is a user virtual address, the control option uses bits <34 : 32> to reference the STT. The control option uses the information from the STT to create a global virtual address.

The control option then converts the global virtual address into a physical address using the GTA and/or the GTB.

For a more detailed description about how the support circuitry translates the virtual address into a physical address, refer to the flowcharts in Figure 19 and Figure 20.

Figure 19. Address Translation

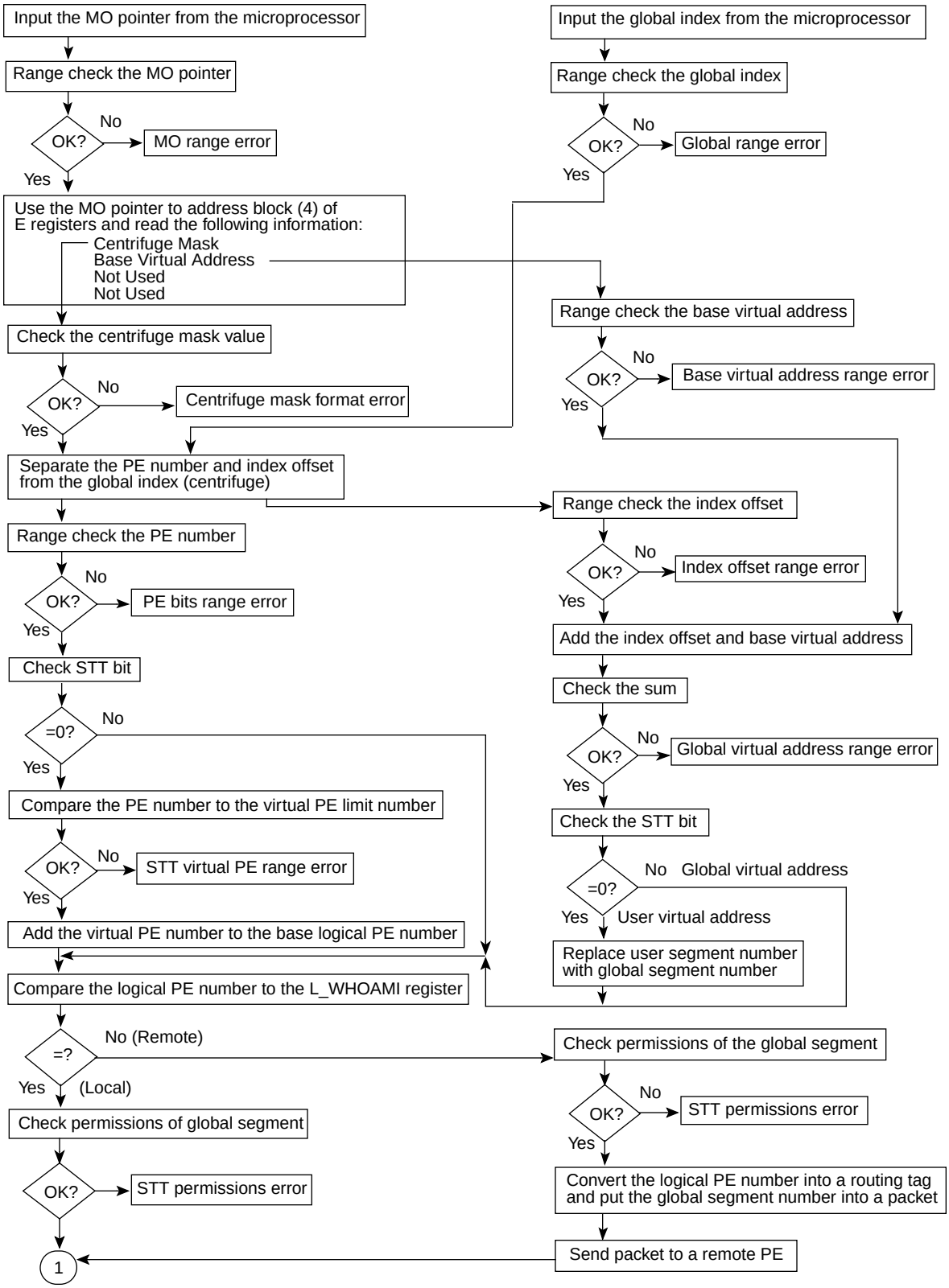
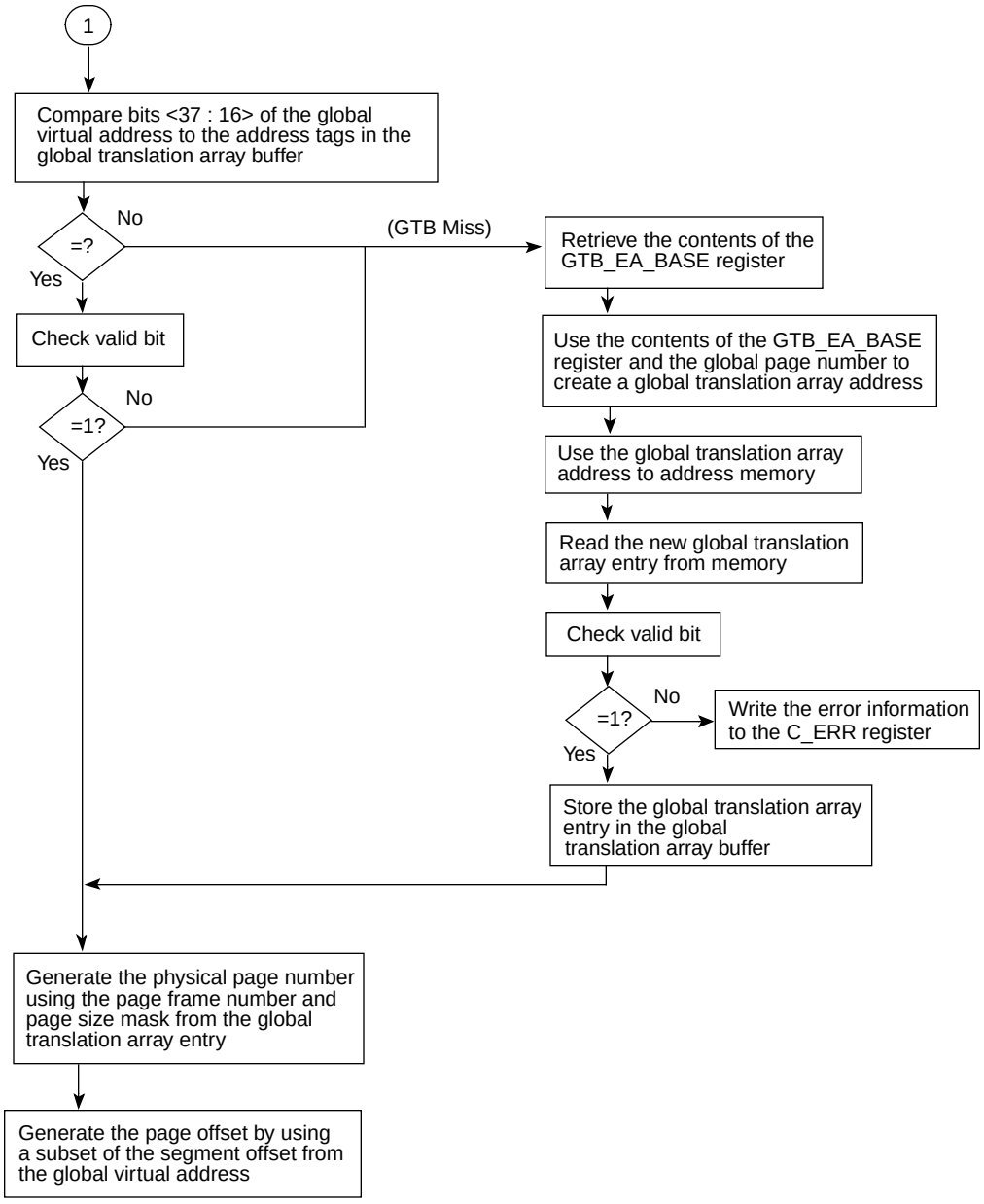


Figure 20. Address Translation (continued)



Atomic Memory Move Operation

Software uses the atomic memory move operation to move data from one area in memory to another area in memory. This area in memory is referred to as a page. There are two types of memory pages: small and large. Small pages are less than 512 Kbytes; large pages are greater than 512 Kbytes. A single memory move operation can move from 64 KBytes to 128 MBytes of data.

The software advantage of small pages is that a small page does not reserve a large amount of memory. For example, when an application needs more memory space, the application increases the size of the user memory. Because it uses small pages, the application increases the size of user memory by small increments, which prevents the allocation of excess memory.

The software advantage of large pages is that a large page does not require as many page bits as a group of small pages. (Page bits identify physical pages in memory.) For example, when software runs out of page bits because there are many small pages in memory, software can move some of the small pages into a large page.

NOTE: Combining multiple small pages into a large page requires more than one data-move command.

The atomic memory move operation is a software and hardware operation. Software remaps a global page to a new physical page by storing parameters into the memory move registers. The hardware, which is located in the control option, moves the data from the old physical page to a new physical page. This transfer is transparent to the user applications. The hardware also keeps the data in local memory and the data in the secondary cache of the microprocessor consistent during the data move.

There are three memory move registers: the GTB_MM_OLD_PFN register, the GTB_MM_NEW_PFN register, and the GTB_MM_GVA register. The GTB_MM_OLD_PFM register contains the old page translation parameters (size mask and frame number). The GTB_MM_NEW_PFN register contains the new page translation parameters (size mask and frame number). And the GTB_MM_GVA register contains the global virtual address [global segment (GSEG) and page (VPAGE)] of the entry to be invalidated in the GTB.

In addition, the memory move operation uses bits 0 through 3 of the GTB_CTL register. Bit 0 is the memory move valid bit (V); when software sets bit 1 to a 1, the memory move hardware starts the data move; when the hardware sets this bit to a 0, the memory move operation is complete. Bit 1 is the hold bit (H); when software sets this bit to a 1, the operation of the GTB halts. Bit 2 is the invalidate all-entries bit (I_A); when software sets this bit to a 1, the hardware invalidates all of the GTB entries. Bit 3 is the invalidate one-entry bit (I_O); when software sets this bit to a 1, the hardware invalidates the GTB entry that is identified in the GTB_MM_GVA register.

During the memory move operation, the hardware compares all of the incoming remote references to the old translation parameter and the new translation parameter. The memory move hardware handles any remote reference that matches one of the translation parameters.

System software may poll for the completion of the memory move or enable an interrupt to indicate the completion of the transfer.

NOTE: The memory move transfers have low priority; however, forward progress is ensured.

The following text describes the steps in which the CRAY T3E system performs a memory move operation using the polled method. The step numbers correspond to the numbers in Figure 21.

1. Ensure that the memory move hardware is idle by reading bit 0 of the GTB_CTL register.

2. Store the old page frame numbers in the GTB_MM_OLD_PFN register and the new page frame numbers in the GTB_MM_NEW_PFN register.

3. Ensure that no SEND commands are in progress.

4. Halt the operation of the global translation buffer by writing a 1 into bit 1 of the GTB_CTL register.

5. Invalidate the GTB entries for the global page that is being remapped.

    1. Write the global virtual address for the corresponding entry into the GTB_MM_GVA register.

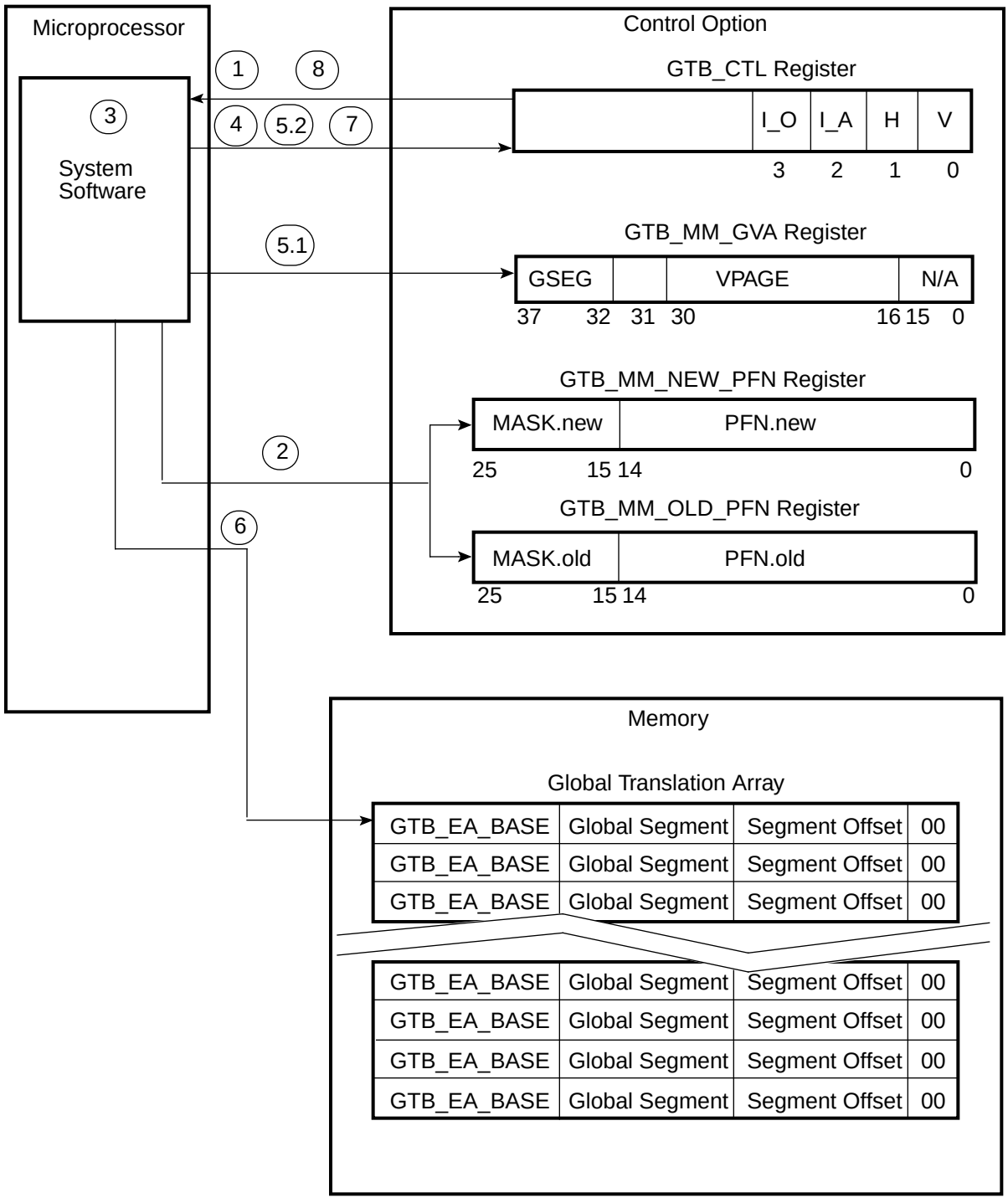
    2. Write a 1 to bit 3 of the GTB_CTL register.

6. Write values for the new page assignment in the appropriate entries of the global translation array.

7. Enable the memory move transfer by writing a 1 to bit 0 of the GTB_CTL register and a 0 to bit 1 of the GTB_CTL register.

8. Periodically read bit 0 of the GTB_CTL register. When bit 0 of the GTB_CTL register is a 0, the data move is complete.

Figure 21. Memory Move Operation



Messaging

A message is a packet of information that one PE sends to another PE. After receiving a message, the support circuitry in the PE stores the message in a designated location in local memory called the message queue. When the message queue contains more than a specified amount of messages (this number is specified by software), the support circuitry interrupts the microprocessor (if software enables the interrupt). The microprocessor then reads the messages from the message queue.

Messaging uses several hardware and software components (refer to Figure 22). These components include: the message queue, the message queue control word (MQCW), the software-maintained head pointer, the SEND E-register command, the interrupt status (IR_STATUS) register, and the message address (MSG_ADDR) register.

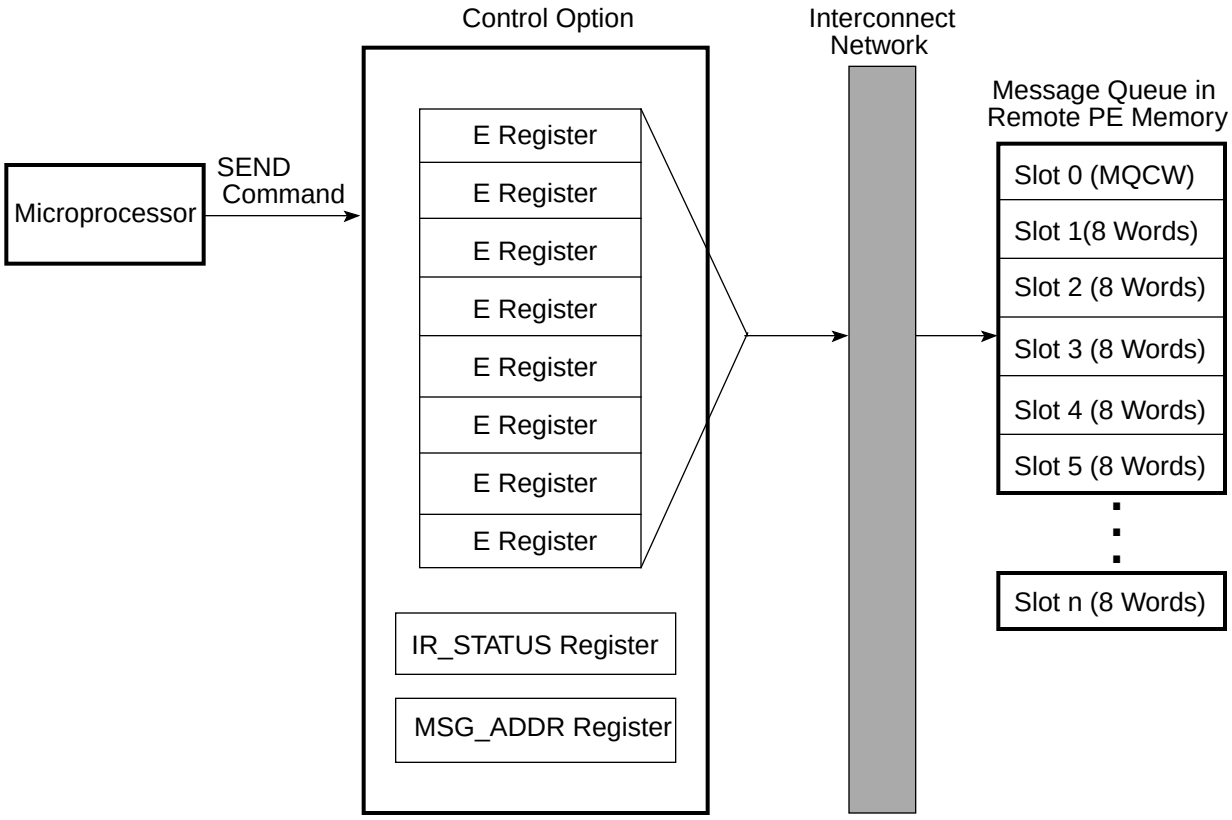
The following subsections provide brief descriptions of these components.

Message Queue

A message queue is a portion of local memory that software designates as the destination for messages. Software may define one, several, or many individual message queues for each PE.

Software also defines the size of each message queue. A message queue may store from 1 to 2,097,151 messages. Each message contains 64 bytes of information. Each 64-byte address location in the message queue is called a slot.

Figure 22. Messaging Components

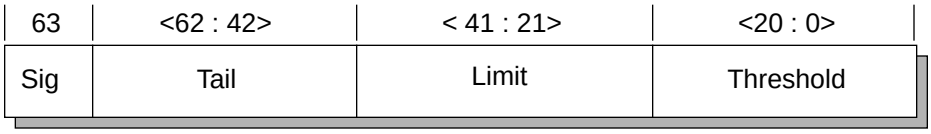


Message Queue Control Word

The message queue control word (MQCW) is a 64-bit word of information that software stores in bits <63 : 0> of the first slot in a message queue. The support circuitry and software use the information stored in the MQCW to perform messaging functions and to define a message queue.

Figure 23 shows the format of the MQCW. The MQCW contains 4 fields: a limit field, a tail field, a threshold field, and a signal (Sig) bit. All of the fields except the signal bit are offset values with respect to the address of the MQCW.

Figure 23. MQCW Bit Format



The limit field indicates the number of slots in the message queue. For example, when the limit field is set to 50, the message queue contains 50 slots for 50 messages. Software may set the limit field to any value between 1 and 2,097,151.

The tail field indicates the slot in the message queue that will receive the next message. When the support circuitry receives a message, it stores the message in the slot that is indicated by the tail field of the MQCW. After storing the message in the slot, the support circuitry increments the tail field by 1.

The threshold field indicates which slot will cause the support circuitry to set an interrupt for the microprocessor. For example, when the threshold field is set to 30, the support circuitry sets an interrupt for the microprocessor when the tail field is equal to 30.

The signal bit indicates that the tail field is equal to the threshold value. Although the support circuitry sets the signal bit to 1, only software can reset the signal bit to 0 by writing a 0 to the MQCW.

NOTE: Hardware modifies the MQCW in an atomic read-modify-write operation. Because of this characteristic, software manipulation of the MQCW does not conflict with hardware updates of the MQCW.

Head Pointer

The head pointer is a software-maintained value that indicates the slot of the message queue that software will read next. System hardware does not use, modify, or provide storage for the head pointer.

After reading a message from the message queue, software increments the head pointer to point to the next slot in the message queue. When the head pointer points to the same slot as the tail pointer, software has read all of the messages out of the message queue.

SEND E-register Command

The message send (SEND) E-register command signals the support circuitry to transfer a 64-byte (8-word) message from 8 contiguous source E registers to a 64-byte slot in a destination PE message queue. The *Commands* document, publication number HMM-142-0, provides detailed information on the SEND E-register command.

IR_STATUS Register

The interrupt status (IR_STATUS) register contains two bits that are used for messaging: bit 32 (MSG_ADDR_VALID bit) and bit 33 (MSG_MULTIPLE bit).

The message address valid (MSG_ADDR_VALID) bit of the IR_STATUS register indicates that the tail value for a message queue is the same as the threshold value for that message queue. (The message queue is identified by the message address (MSG_ADDR) register.) The MSG_ADDR_VALID bit remains set to 1 until software resets it. Software resets this bit by writing a 1 to the MSG_ADDR_VALID bit of the IR_STATUS register.

The multiple messages (MSG_MULTIPLE) bit indicates that the tail value is equal to the threshold value in more than one message queue. The MSG_MULTIPLE bit remains set to 1 until software resets the bit. Software resets this bit by writing a 0 to the MSG_MULTIPLE bit of the IR_STATUS register.

MSG_ADDR Register

The message address (MSG_ADDR) register contains the global virtual address (GVA) of the MQCW for the message queue that set the MSG_ADDR_VALID bit of the IR_STATUS register to 1 (refer to Figure 24). The GVA in the MSG_ADDR register remains locked in the register until software resets the MSG_ADDR_VALID bit of the IR_STATUS register. Then, the support circuitry stores a new GVA into the register the next time the support circuitry sets the MSG_ADDR_VALID bit to 1.

Figure 24. MSG_ADDR Register Format

<63 : 38>	< 37 : 32>		<30 : 0>
Read as Zero (RAZ)	MQCW GVA Segment Field	0	MQCW GVA Segment Offset

Sending a Message

To create a message, software issues 8 STORE commands, which transfer data from the microprocessor to 8 contiguous E registers (refer to Figure 25). This data contains the message. After storing the message data in the E registers, the software issues a SEND command (refer to Figure 26). The SEND command transfers the message from the E registers to the message queue of a destination PE.

Figure 25. STORE Command

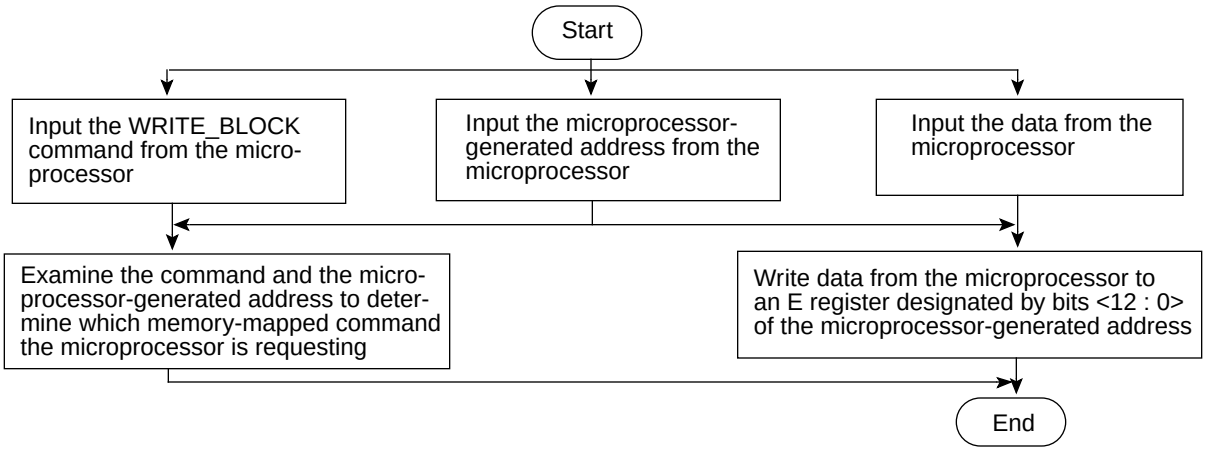
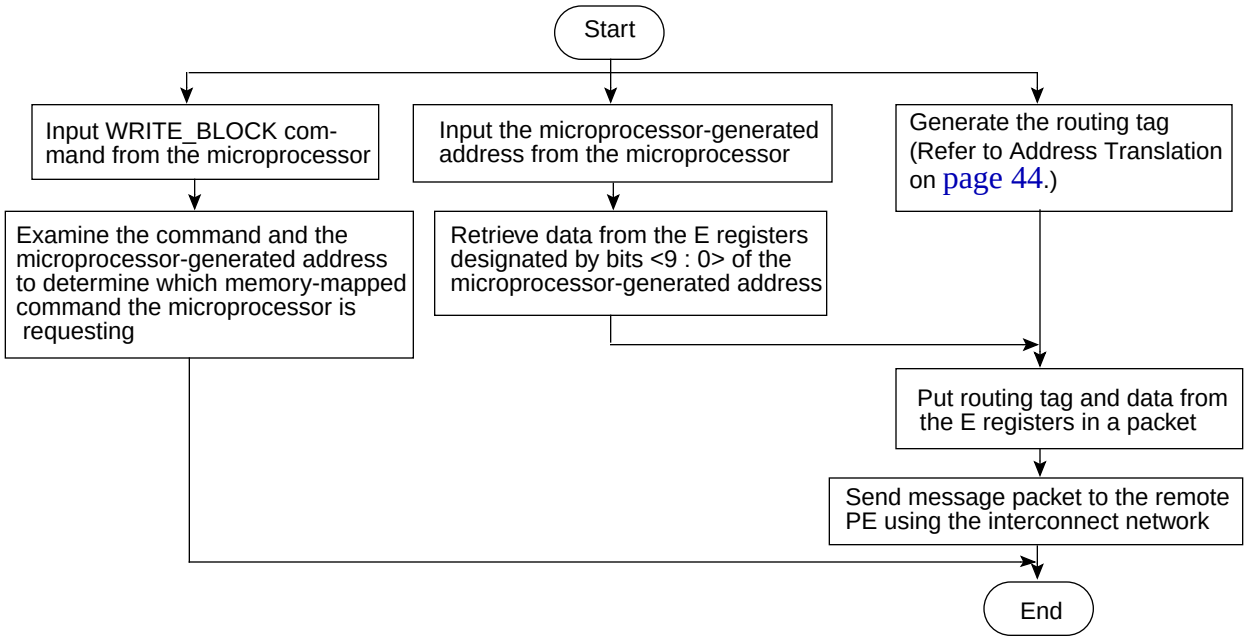


Figure 26. SEND Command



Receiving and Storing a Message

When the support circuitry receives a message, the support circuitry retrieves the MQCW. The support circuitry compares the tail value of the MQCW to the limit value. When the tail value is equal to the limit value, the support circuitry and the network interface create a SEND-rejected packet and return it to the source PE (refer to Figure 27). The SEND-rejected packet indicates that the message queue is full and cannot accept another message.

When the tail value is less than the limit value, the support circuitry returns a SEND-accepted packet to the source PE. The support circuitry then stores the message in the message queue at the slot pointed to by the tail pointer of the MQCW. Next, the support circuitry increments the tail pointer and updates the MQCW in memory.

Figure 27. Receiving and Storing a Message

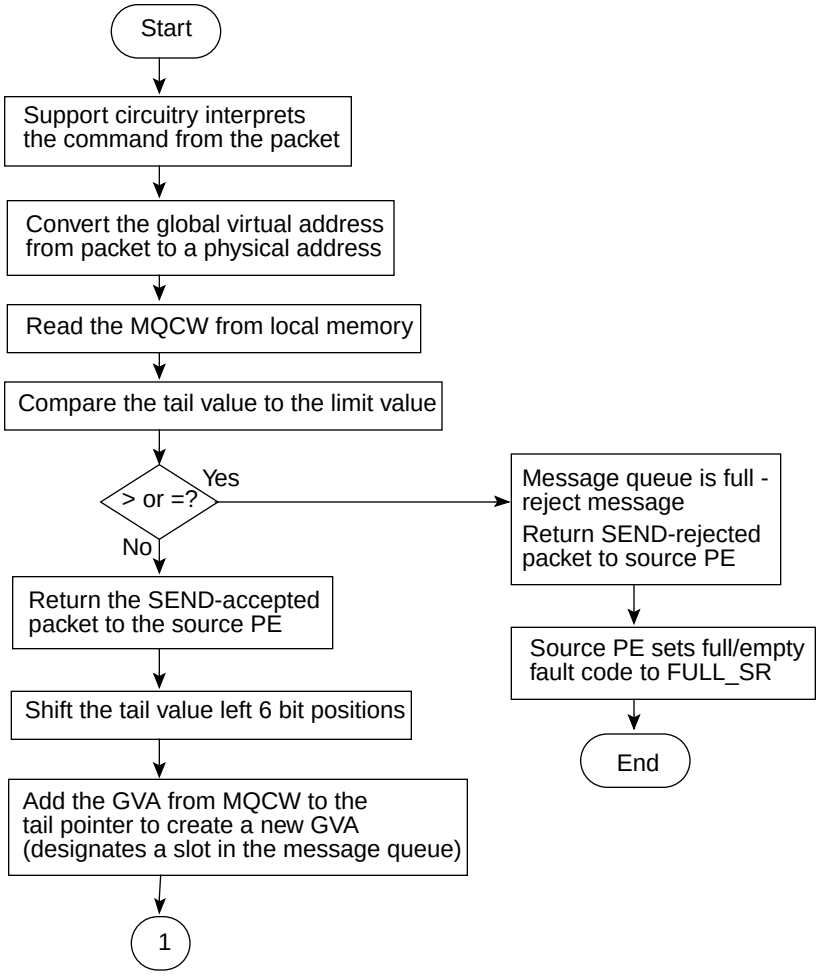
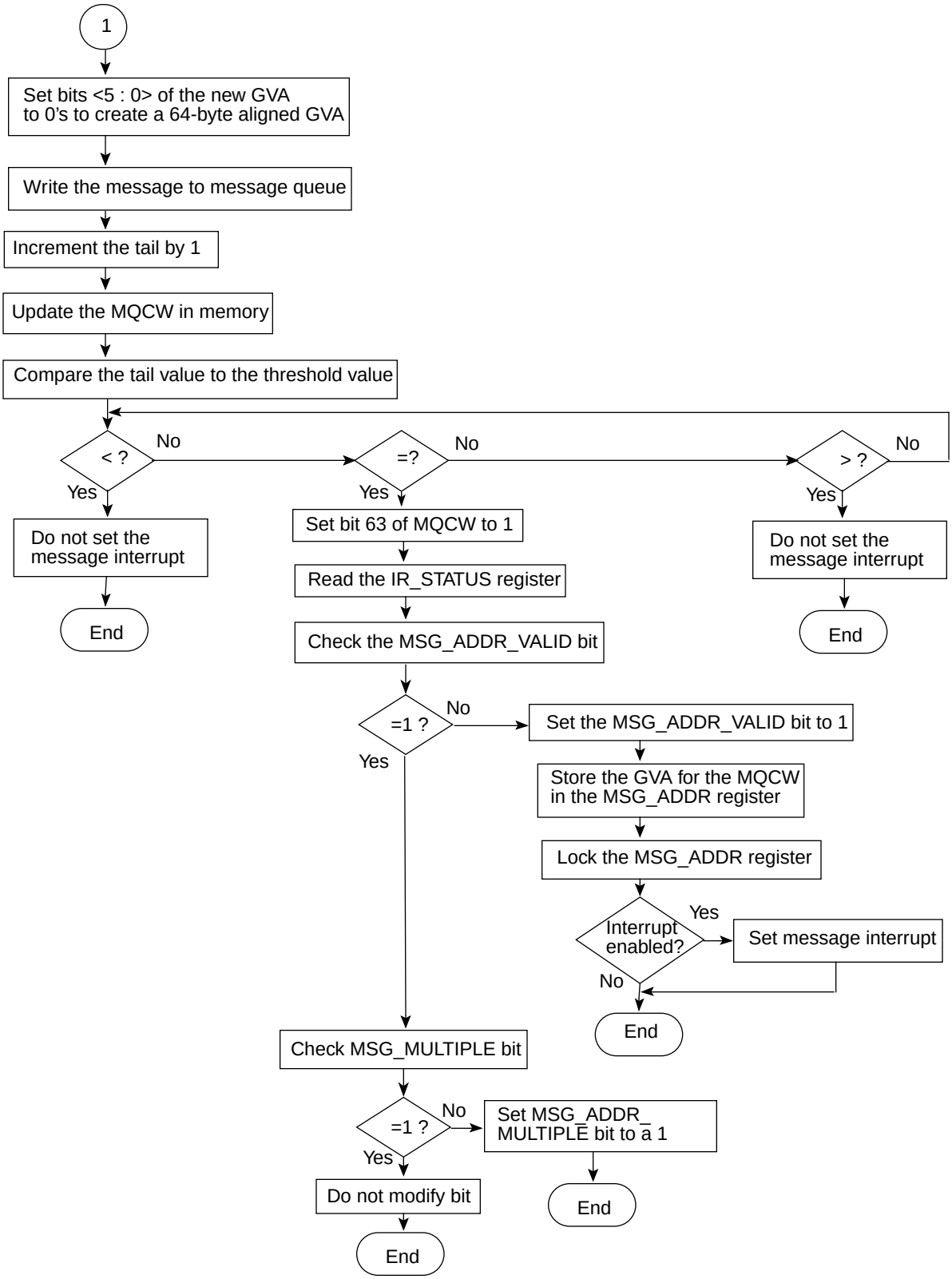


Figure 28. Receiving and Storing a Message (Continued)



Reading a Message

Figure 29 shows an example of how software may read a message from the message queue.

Figure 29. Reading a Message

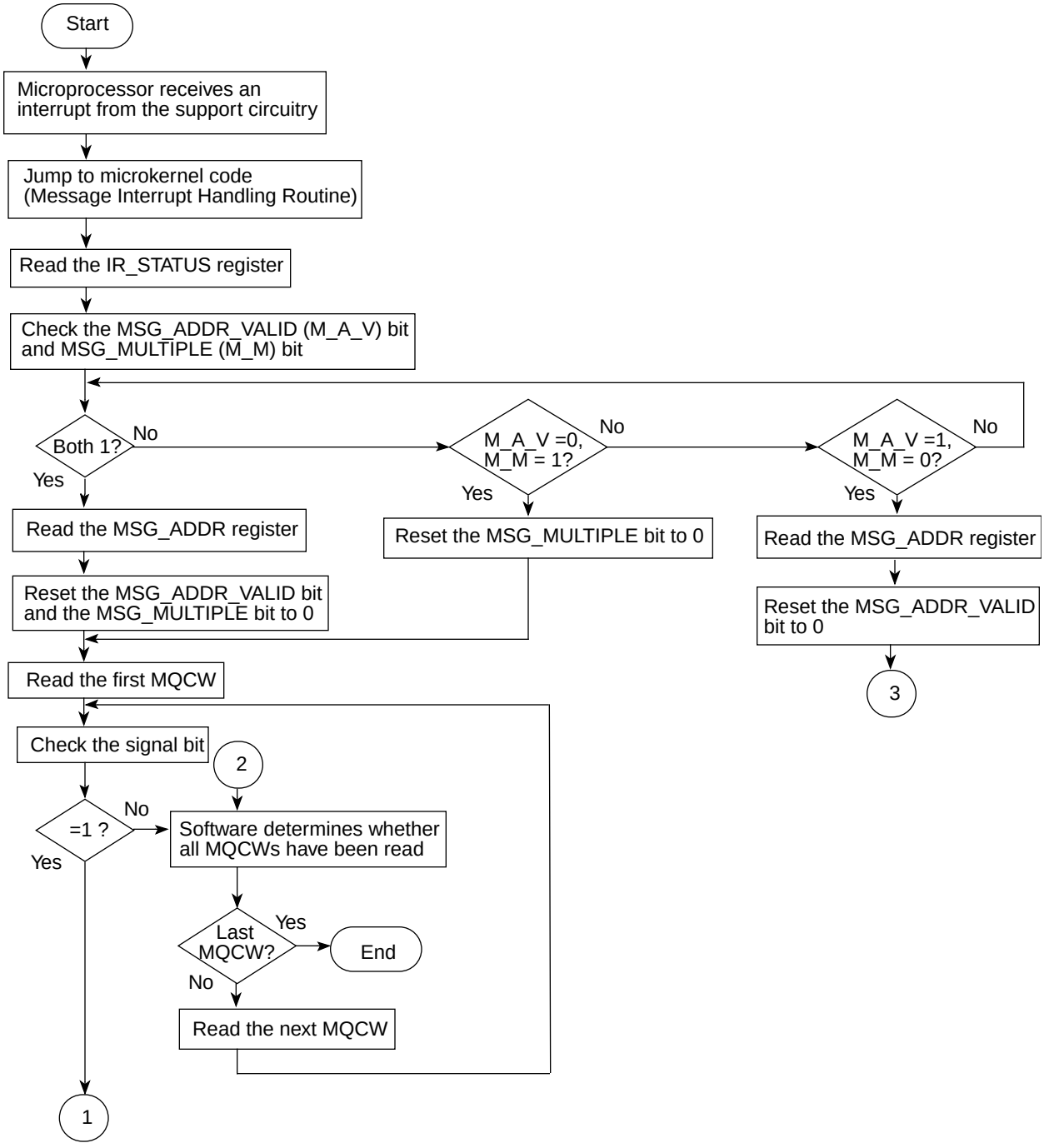
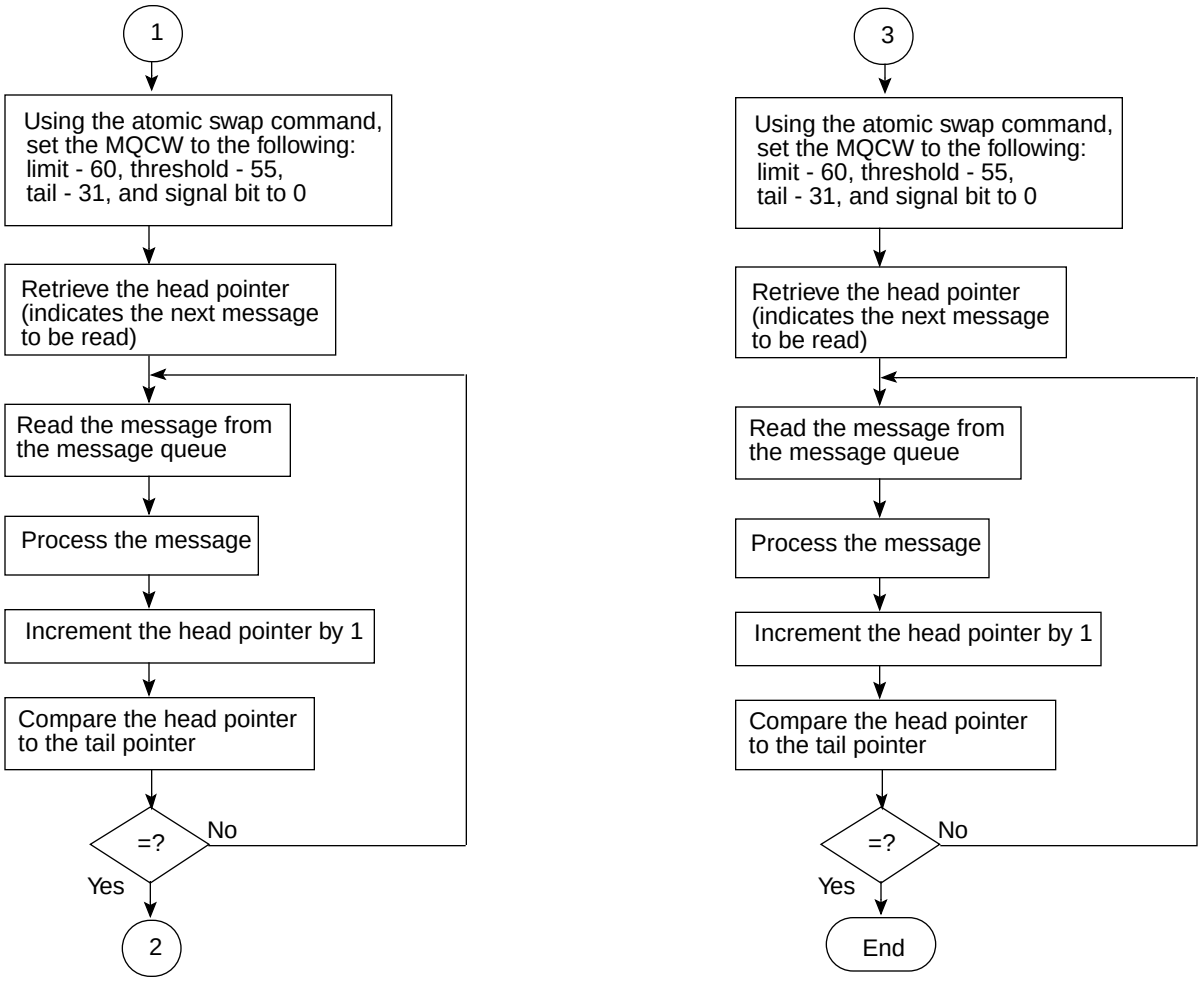


Figure 30. Reading a Message (Continued)



PE Errors

The C option logs errors in the C_ERR [3 : 0] registers. The C_ERR0 register contains status bits that indicate the type of errors that occurred. The C_ERR1 register enables individual status bits; the C_ERR2 register contains the first error code; and the C_ERR3 register contains the first error payload.

C_ERR0

When a bit in the C_ERR0 register is set to 1, the bit indicates that the corresponding error occurred (refer to Table 9 and Figure 31).

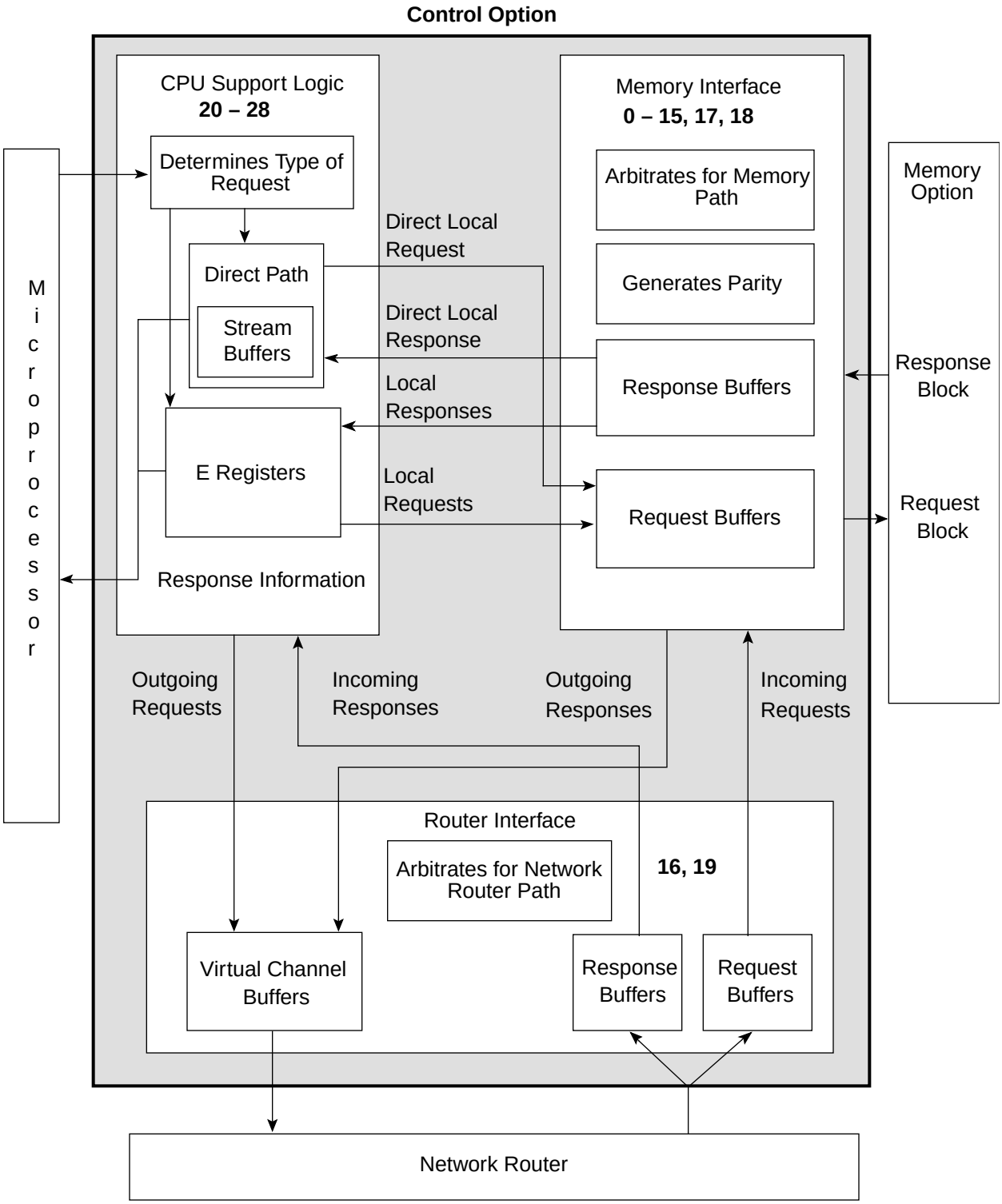
Table 9. C_ERR0 Register Bit Format

Bits	Error Description
0	M0-to-C option channel data error (daughter card 0 to C option)
1	M1-to-C option channel data error (daughter card 1 to C option)
2	M2-to-C option channel data error (daughter card 2 to C option)
3	M3-to-C option channel data error (daughter card 3 to C option)
4	M0-to-C option channel control error (daughter card 0 to C option)
5	M1-to-C option channel control error (daughter card 1 to C option)
6	M2-to-C option channel control error (daughter card 2 to C option)
7	M3-to-C option channel control error (daughter card 3 to C option)
8	M0-to-C option return buffer parity error (daughter card 0 to C option)
9	M1-to-C option return buffer parity error (daughter card 1 to C option)
10	M2-to-C option return buffer parity error (daughter card 2 to C option)
11	M3-to-C option return buffer parity error (daughter card 3 to C option)
12	Local shared data parity error
13	Remote shared data parity error
14	Data control FIFO parity error
15	Return data FIFO parity error
16	R-to-C option channel parity error
17	Secondary-cache back map parity error
18	Shared request FIFO parity error
19	R-to-C option response FIFO parity error
20	E-register request queue parity error 0
21	E-register request queue parity error 1
22	E-register RAM parity error
23	E-register ERS code parity error
24	Private (direct local memory) read data buffer parity error 0

Table 9. C_ERR0 Register Bit Format (continued)

Bits	Error Description
25	Private read data buffer parity error 1
26	Microprocessor data parity error 0
27	Microprocessor data parity error 1
28	Microprocessor address command error
<59 : 29>	These bits are not used.
60	E-register context 0 software error
61	E-register context 1 software error
62	Invalid barrier error
63	Invalid hardware register error

Figure 31. C Option Error Reporting



NOTE: The bold numbers correlate to bit numbers 0 through 28 of the C_ERR0 register.

C_ERR1

The C_ERR1 register enables the error interrupts. When a bit of the C_ERR1 register is set to 1, the corresponding interrupt of the C_ERR0 register is enabled. When a bit of the C_ERR1 register is set to 0, the corresponding interrupt of the C_ERR0 register is disabled. Although the interrupt is disabled, the corresponding bit of the C_ERR0 register for that interrupt still indicates the state of the interrupt.

Bit 59 of the C_ERR1 register does not function as described in the previous paragraph. When set to 0, this bit signals the support circuitry to set the fill_nocheck_h microprocessor pin during any transfer of data from the support circuitry to the microprocessor. When set to 1, this bit signals the support circuitry to set the fill_nocheck_h microprocessor pin only during transfers of data from an auxiliary register in the support circuitry to the microprocessor.

C_ERR2

Bits 0 through 5 of the C_ERR2 register identify the first enabled interrupt in the C_ERR0 register that sets to 1. For example, when multiple bits are set to 1 in the C_ERR0 register and the FIRST_ERROR bits of the C_ERR2 register are set to 19_{10} (10011_2), the R-to-C option response FIFO parity error occurred first.

C_ERR3

When the C_ERR2 register is set to 60, 61, 62, or 63, bits 0 through 57 of the C_ERR3 register contain error payload information, bit 58 specifies the E-register context, and bits 59 through 63 indicate the software error type (refer to Table 10). When the C_ERR2 register is set to any value other than 60, 61, 62, or 63, the contents of the C_ERR3 register are undefined.

Table 10. Error Types

Error Type	Description
1	E-register reference field out of range
2	MO pointer out of range
3	E-register load from an E register with its ERS code set to FULL_F
4	ERS code set to FULL_F in one of the MO block E registers
5	Attempted a PUT command to an E register with its ERS code set to FULL_F
6	Illegal E-register operation (bypassing the STT without the SP bit set)
7	Remote register bit (bit 63) was set, but SP bit was not set; or attempted an E-register command other than a PUT command, GET command, or SEND command for a remote register
8	Content of one of the MO block E registers or the microprocessor data path argument was not aligned.
9	Global or local index out of range
10	The information on the microprocessor data bus was out of range for an SGET or SPUT command
11	User or global virtual address from the local index is out of range or bit 31 of the user or global virtual address is set to 1
12	Centrifuge mask has more than 50 significant bits or more than 20 bits set to 1
13	Stride value has more than 50 significant bits
14	Base virtual address (user or global) is out of range
15	Post centrifuge PE range error (more than 12 significant bits)
16	Post centrifuge index offset range error (more than 38 significant bits)
17	User or global virtual address generated from adding the base virtual address to the index offset is out of range or bit 31 of the user or global virtual address is set to 1
18	Virtual PE from the STT is out of range
19	STT permissions violation
20	GTB range check violation or the valid bit was set to 0 when a global translation array entry was read from memory
21	MQCW update completed, but a range error occurred when storing the message (For this error type, bits <5 : 0> of the C_ERR3 register are undefined; in addition, the system virtual address in the C_ERR3 register is the address for the message payload, not the MQCW.)
22	Not used

Table 10. Error Types (continued)

Error Type	Description
23	Tried to access an undefined barrier space, attempted a privileged barrier operation, or a must be zero (MBZ) violation occurred
24	Tried to access an undefined auxiliary register, attempted to store to an auxiliary register, or an MBZ violation occurred

The error payload information can be displayed in one of three formats: A, B, or C. The format is specified by the value of the ERROR_TYPE bits (bits <63 : 59> of the C_ERR3 register).

Format A

Format A provides the microprocessor-generated address error information (refer to Table 11).

Table 11. Format A Bit Layout

Bits	Description
<2 : 0>	These bits are set to 0's.
<30 : 3>	These bits contain bits <30 : 3> of the microprocessor-generated address
<38 : 31>	These bits have different values for different operations: For auxiliary register WRITE_BLOCK requests, MASK_BITS <7 : 0> indicate which 32-bit halfwords were written. For noncacheable READ_BLOCK requests, MASK_BITS <3 : 0> indicate which 64-bit words were requested. MASK_BITS <7 : 4> are undefined. For LOAD E-register commands, MASK_BITS <7 : 4> indicate which registers were requested. MASK_BITS <7 : 4> indicate which E registers had their ERS codes set to FULL_F.
<49 : 39>	These bits are set to 0's.
<57 : 50>	These bits contain the more operand (MO) pointer field. The MO pointer references additional E-register command arguments that are stored in a group of four E registers.

Format B

Format B provides the MO pointer and the global index error information (refer to Table 12).

Table 12. Format B Bit Layout

Bits	Description
0	When set to 1, this bit indicates that the sign of the global index is negative. When set to 0, this bit indicates that the sign of the global index is positive.
1	When set to 1, this bit indicates that the STT is bypassed. When set to 0, this bit indicates that the STT is used. This bit is the same as bit 21 of the microprocessor-generated address.
<49 : 2>	These bits contain bits <49 : 2> of the global index or bits <37 : 2> of the local index.
<57 : 50>	These bits contain the MO pointer field. The MO pointer references more E-register command arguments that are stored in a group of four E registers.

Format C

Format C provides the user virtual address error information or the global virtual address error information (refer to Table 13).

Table 13. Format C Bit Layout

Bits	Description
<1 : 0>	These bits are set to 0's.
<30 : 2>	These bits contain the segment offset bits of the user virtual address or the global virtual address.
31	This bit is set to 0.
<37 : 32>	These bits contain a user segment (in bits <34 : 32>) or a global segment (in bits <37 : 32>).
<49 : 38>	These bits contain the logical PE number that is referenced.
50	When set to 1, this bit indicates that the virtual address is actually a hardware register address that references a remote register.
<55 : 51>	These bits contain the 5 least-significant bits of the E-register command opcode (bits <17 : 13> of the microprocessor-generated address).
<57 : 56>	These bits are set to 0's.

The M options log memory errors in the M_ERR[3 : 0] registers; each memory option contains its own M_ERR registers. The M_ERR0 registers contain status bits that indicate the type of errors that occurred. The M_ERR1 registers enable individual status bits. The M_ERR2 registers contain the first error code. And the M_ERR3 registers contain the first error payload.

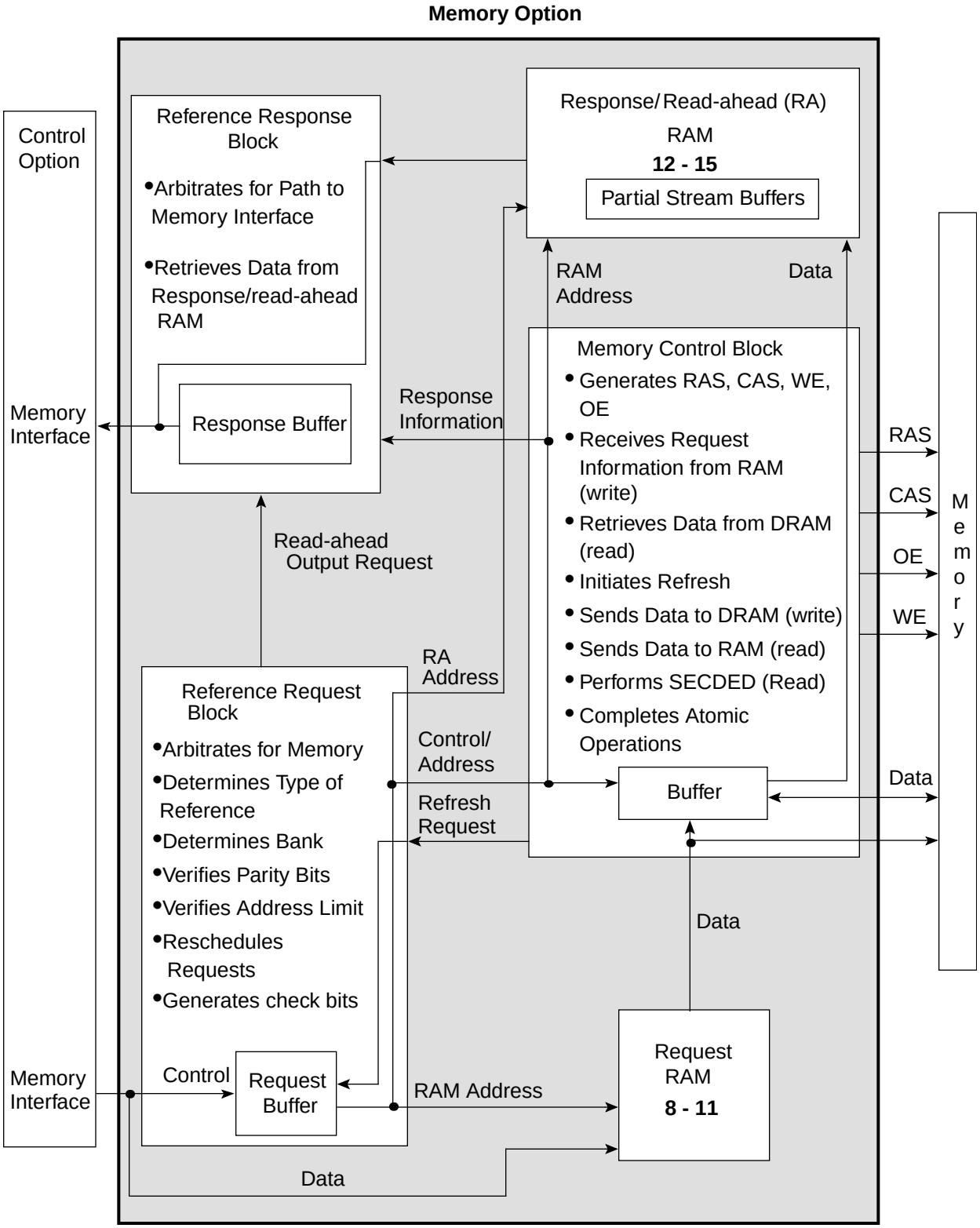
M[3 : 0]_ERR0

The M_ERR0 registers indicate which errors occurred (refer to Table 14 and Figure 32).

Table 14. M_ERR0 Register Bit Format

Bits	Description
0	When set to 1, this bit indicates that a channel parity error occurred.
1	When set to 1, this bit indicates that a channel range error occurred.
2	When set to 1, this bit indicates that a channel data error occurred.
3	When set to 1, this bit indicates that a hardware register access error occurred.
<7 : 4>	When set to 1, each bit indicates that a multiple-bit error occurred. 4 = Bits <31 : 0> of bank n+0 6 = Bits <31 : 0> of bank n+4 5 = Bits <63 : 32> of bank n+0 7 = Bits <63 : 32> of bank n+4
<11 : 8>	When set to 1, each bit indicates that a reference-queue RAM error occurred. 8 = Bits <31 : 0> of bank n+0 10 = Bits <31 : 0> of bank n+4 9 = Bits <63 : 32> of bank n+0 11 = Bits <63 : 32> of bank n+4
<15 : 12>	When set to 1, each bit indicates that a response-queue RAM error occurred. 12 = Bits <31 : 0> of bank n+0 14 = Bits <31 : 0> of bank n+4 13 = Bits <63 : 32> of bank n+0 15 = Bits <63 : 32> of bank n+4
16	When set to 1, this bit indicates that an address compare interrupt occurred from an M_ADDR_CMP register.
<20 : 17>	When set to 1, each bit indicates that a single-bit error occurred. 17 = Bits <31 : 0> of bank n+0 19 = Bits <31 : 0> of bank n+4 18 = Bits <63 : 32> of bank n+0 20 = Bits <63 : 32> of bank n+4
<63 : 21>	These bits are read as 0's.

Figure 32. M Option Error Reporting



NOTE: The bold numbers correlate to bits 0 through 20 of the M_ERR0 register.

M[3 : 0]_ERR1

The M_ERR1 register enables the error interrupts. When a bit of the M_ERR1 register is set to 1, the corresponding interrupt of the M_ERR0 register is enabled. When a bit of the M_ERR1 register is set to 0, the corresponding interrupt of the M_ERR0 register is disabled. Although an interrupt is disabled, the corresponding bit of the M_ERR0 register for that interrupt still indicates the state of the interrupt.

M[3 : 0]_ERR2

Bits 0 through 4 of the M_ERR2 register identify the first enabled interrupt in the M_ERR0 register that sets to 1. For example, if multiple bits are set to 1 in the M_ERR0 register and the FIRST_ERROR bits of the corresponding M_ERR2 register are set to 2, the channel data error occurred first.

M[3 : 0]_ERR3

The M_ERR3 register contains error payload information (refer to Table 15). The contents of the M_ERR3 register depend on the value of the FIRST_ERROR bits in the M_ERR2 register.

Table 15. M_ERR3 Contents

FIRST_ERROR Bits	Contents
0	When bit 9 (EN_ALL_P) bit of the M_CFG register is set to 1, this bit enables the capture of all channel packets into the M_ERR2 register and the M_ERR3 register. Bits <41 : 0> = Channel packet Bits <63 : 42> = Not used When the EN_ALL_P bit of the M_CFG register is set to 0, this bit indicates the capture of all channel packets into the M_ERR2 register and the M_ERR3 register. In this case, only valid channel requests are captured. Data and the idle packets are not captured.
3 - 1	Bits <41 : 0> = Channel packet Bits <63 : 42> = Not used
7 – 4	Bits <6 : 0> = Check bits Bit 7 = 0 Bits <14 : 8> = Syndrome bits Bits <63 : 15> = Not used
11 – 8	Bits <2 : 0> - Reference queue RAM address Bits <63 : 3> = Not used
15 – 12	Bits <2 : 0> = Response RAM address Bits <63 : 3> = Not used
16	Bits <41 : 0> = Channel packet Bits <63 : 42> = Not used
20 – 17	Bits <6 : 0> = Check bits Bit 7 = 0 Bits <14 : 8> = Syndrome bits Bits <63 : 15> = Not used

Offline Diagnostics

Table 16 lists the offline diagnostic tests for the support circuitry; Table 17 lists the offline diagnostic tests for local memory; and Table 18 lists special offline diagnostic tests for the PE. For more information about these diagnostics, refer to the *Offline Diagnostics* document, HDM-136.

Table 16. Support Circuitry Diagnostic Tests

Test Name	Description
ereg	Tests basic E register operations
gtb	Tests the global address translation process
peri	Tests the microprocessor's ability to execute random instruction sequences
segt	Tests the segment translation table

Table 17. Local Memory Diagnostic Tests

Test Name	Description
lmem	Tests local memory operations, SECDED, stream buffering, and cache coherence
rlo	Tests random local operations
rmem	Tests remote memory operations

Table 18. Special Diagnostic Tests

Test Name	Description
etrap	Tests the software-generated errors that are trapped by the C option
mqcw	Tests the message queue control word
pst	Tests the microprocessor stall timeout function and the PROD function
todt	Tests the time-of-day clock

Store Instruction

The store instruction writes data from an integer register of the microprocessor to system memory or it initiates a memory-mapped register command. The type of operation that the support circuitry performs depends upon the state of virtual address bit 39. When virtual address bit 39 is a 0, the support circuitry writes data from an integer register to system memory and the microprocessor writes the data to the secondary cache and data cache when necessary. When virtual address bit 39 is a 1, the support circuitry performs a memory-mapped register command.

Store Instruction for a Memory-mapped Register Command

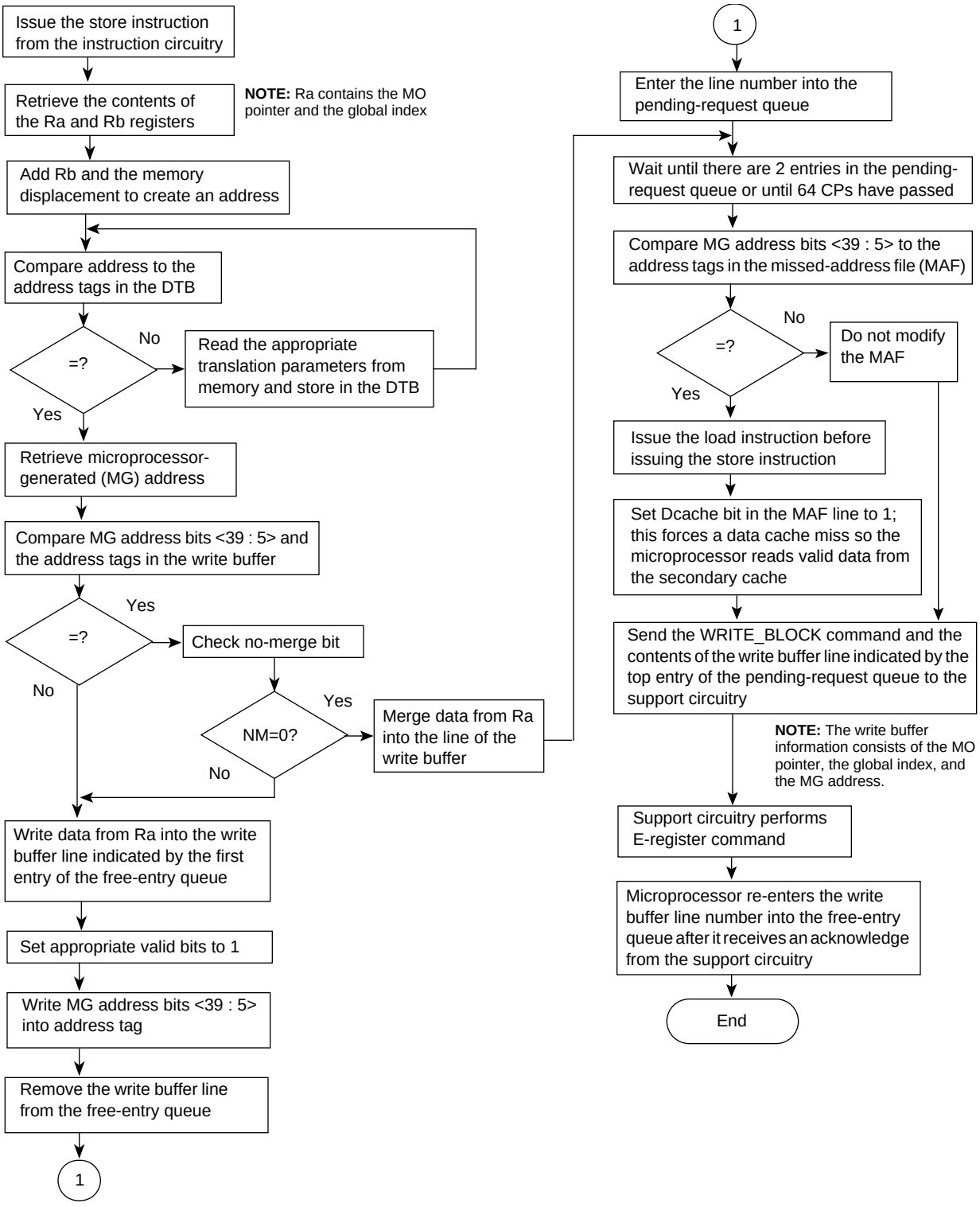
When the instruction circuitry of the microprocessor issues a store instruction, the microprocessor retrieves the contents of two integer registers, Ra and Rb, which are specified by the instruction. The Ra integer register contains the MO pointer and the global index. The Rb integer register contains a value that the microprocessor adds to a memory displacement value (also from the instruction) to create a microprocessor-generated address.

The microprocessor compares the microprocessor-generated address bits <39 : 5> to the address tags in the write buffer. When address bits <39 : 5> and an address tag are equal and the no-merge bit within that entry of the write buffer is 0, the microprocessor writes the contents of the Ra integer register into that line of the write buffer. When address bits <39 : 5> and an address tag are not equal or the no-merge bit within that entry of the write buffer is a 1, the microprocessor writes the contents of the Ra integer register into a new line of the write buffer.

The microprocessor also compares virtual address bits <39 : 5> to the address tags in the missed-address file to determine whether the missed-address file is holding a load instruction that is destined for the same address. When the missed-address file contains a load instruction that is destined for the same address, the microprocessor issues the load instruction before proceeding with the store instruction. When the missed-address file does not contain a load instruction that is destined for the same location, the microprocessor does not modify the missed-address file.

Next, the microprocessor sends the contents of the write buffer and a WRITE_BLOCK command to the support circuitry. The support circuitry uses the command and the address to determine what command to perform, performs the command, and notifies the microprocessor when the command is complete.

Figure 33. Store Instruction for a Memory-mapped Register Command



Store Instruction for a Direct Local Memory Command

When the instruction circuitry of the microprocessor issues a store instruction, the microprocessor retrieves the contents of two integer registers, Ra and Rb, which are specified by the instruction. The Ra integer register contains the data that will be written to memory. The Rb integer register contains a value that the microprocessor adds to a memory displacement value (also from the instruction) to create a virtual address.

The microprocessor compares virtual address bits <39 : 5> to the address tags in the write buffer. When address bits <39 : 5> and an address tag are equal and the no-merge bit within that entry of the write buffer is 0, the microprocessor writes the contents of the Ra integer register into that line of the write buffer. When address bits <39 : 5> and an address tag are not equal or the no-merge bit within that entry of the write buffer is a 1, the microprocessor writes the contents of the Ra integer register into a new line of the write buffer.

The microprocessor also compares virtual address bits <39 : 5> to the address tags in the missed-address file to determine whether the missed-address file is holding a load instruction that is destined for the same address. When the missed-address file contains a load instruction that is destined for the same address, the microprocessor issues the load instruction before proceeding with the store instruction. When the missed-address file does not contain a load instruction that is destined for the same location, the microprocessor does not modify the missed-address file.

Next, the microprocessor sends the contents of the write buffer and a WRITE_BLOCK command to the support circuitry and determines whether it should update the data cache and the secondary cache. The support circuitry uses the command and the address to determine what command to perform, writes the data to local memory, and notifies the microprocessor when the operation is complete.

Figure 34. Store Instruction for a Direct Local Memory Command

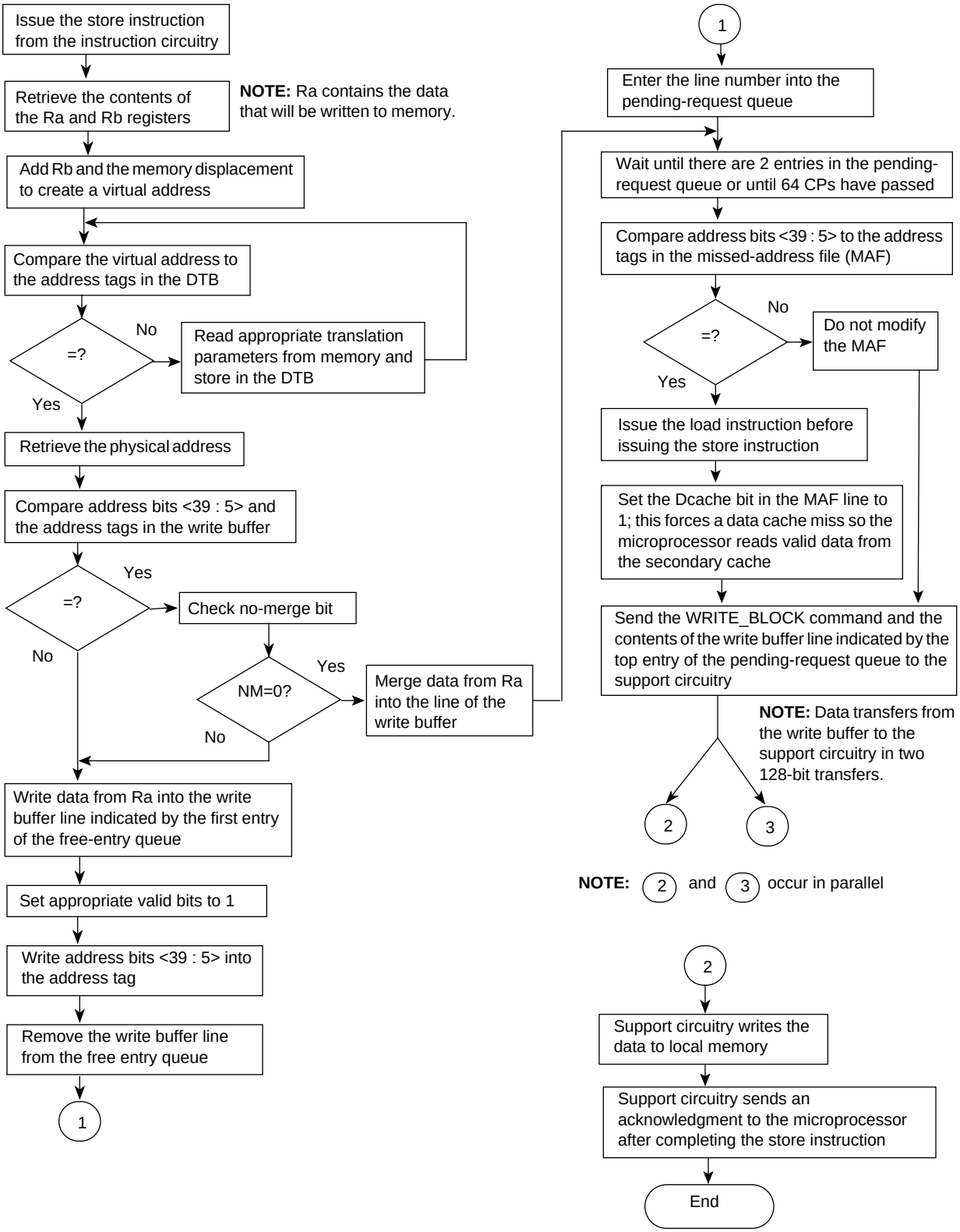
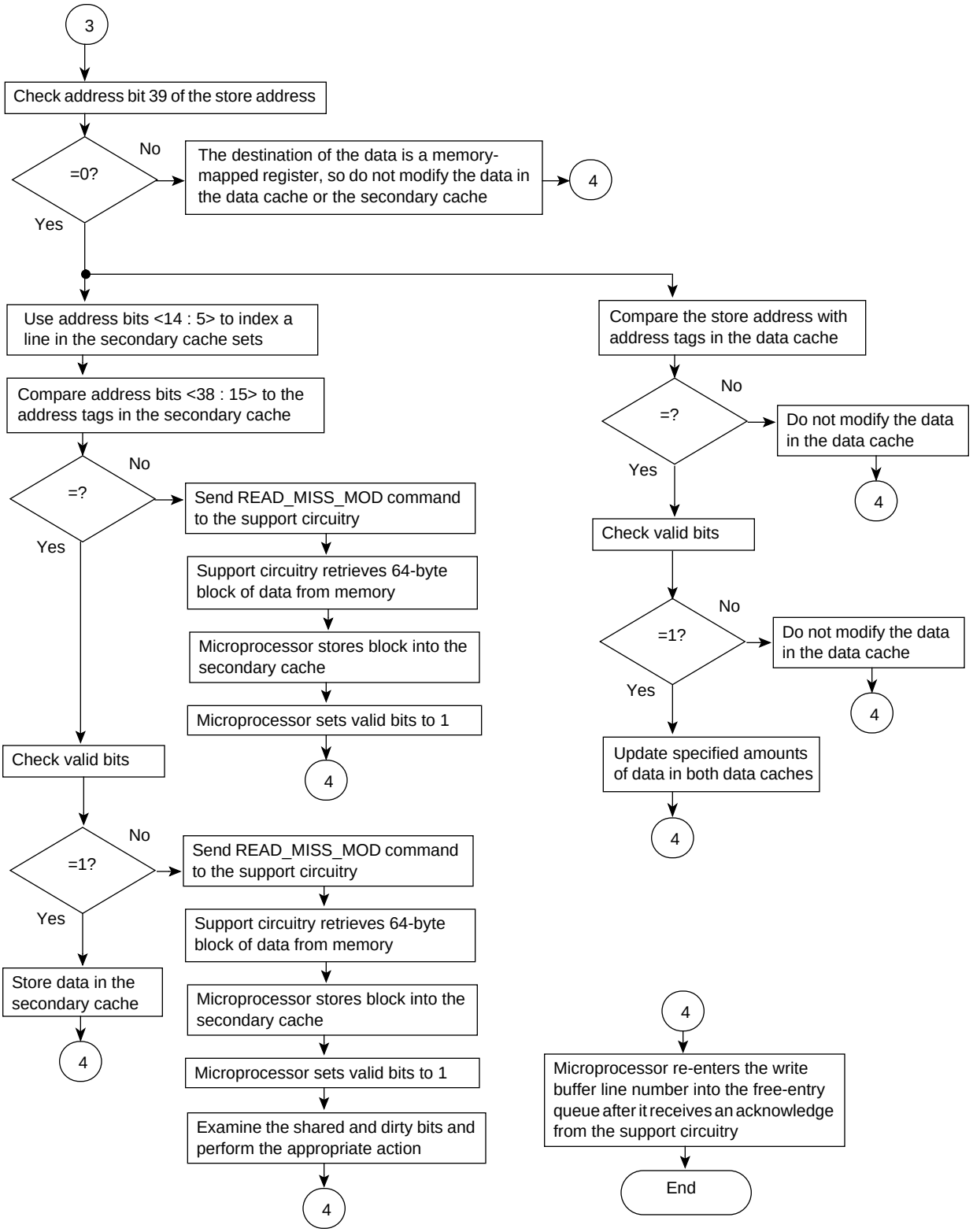


Figure 35. Store Instruction for a for a Direct Local Memory Command (Continued)



Load Instruction

The load instruction reads data from the data cache, secondary cache, or system memory or initiates an E-register command. The type of operation that the support circuitry performs depends upon the state of virtual address bit 39. When virtual address bit 39 is a 0, the support circuitry reads data from system memory, the data cache, or the secondary cache. When virtual address bit 39 is a 1, the support circuitry performs a memory-mapped register command.

When the instruction circuitry of the microprocessor issues a load instruction, the microprocessor retrieves the contents of the Rb register, which is specified by the instruction. The Rb integer register contains a value that the microprocessor adds to a memory displacement value (also from the instruction) to create a virtual address.

The microprocessor verifies whether the data is in the data cache. When the data is not in the data cache, a data cache miss occurs. When the data is in the data cache, the microprocessor retrieves the data and writes it into the Ra register.

When a data cache miss occurs, the microprocessor enters the address in the missed-address file and checks the write buffer to see if there is a pending store instruction that is destined for the same location. When the write buffer contains a store instruction that is destined for the same location, the microprocessor sets the no-merge bit in that line of the write buffer and flushes the write buffer.

Next, the microprocessor checks the secondary cache for the requested data. When the data is in the secondary cache, the microprocessor reads the data from the secondary cache and transfers it to the instruction or data cache and the Ra register. When the data is not in the secondary cache, a secondary cache miss occurs.

A secondary cache miss causes the microprocessor to send the address and a READ_MISS command to the support circuitry. The support circuitry retrieves the data from system memory and sends it to the microprocessor. The microprocessor writes the data into the Ra register. The microprocessor also attempts to write this data into the secondary cache.

Figure 36. Load Instruction

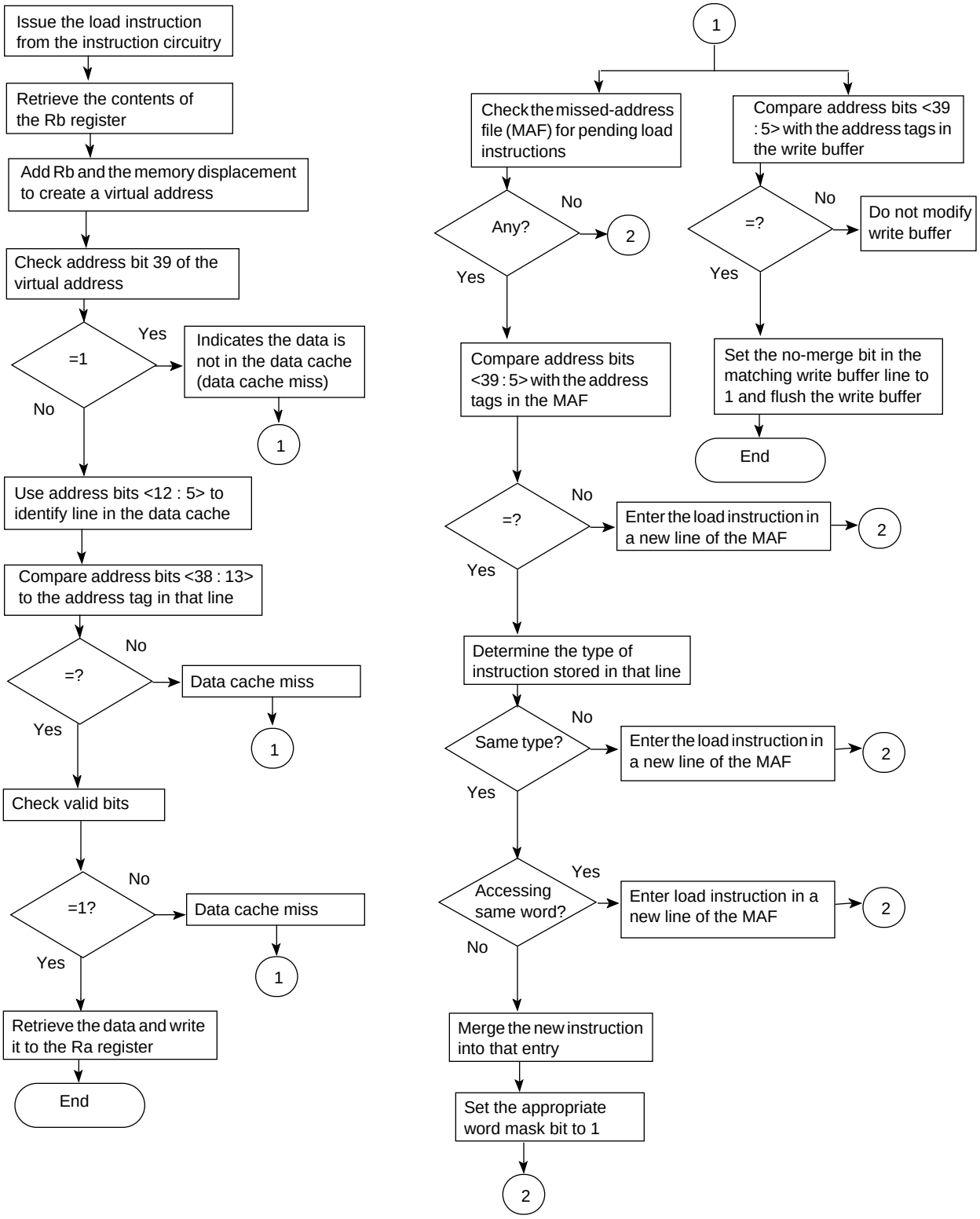
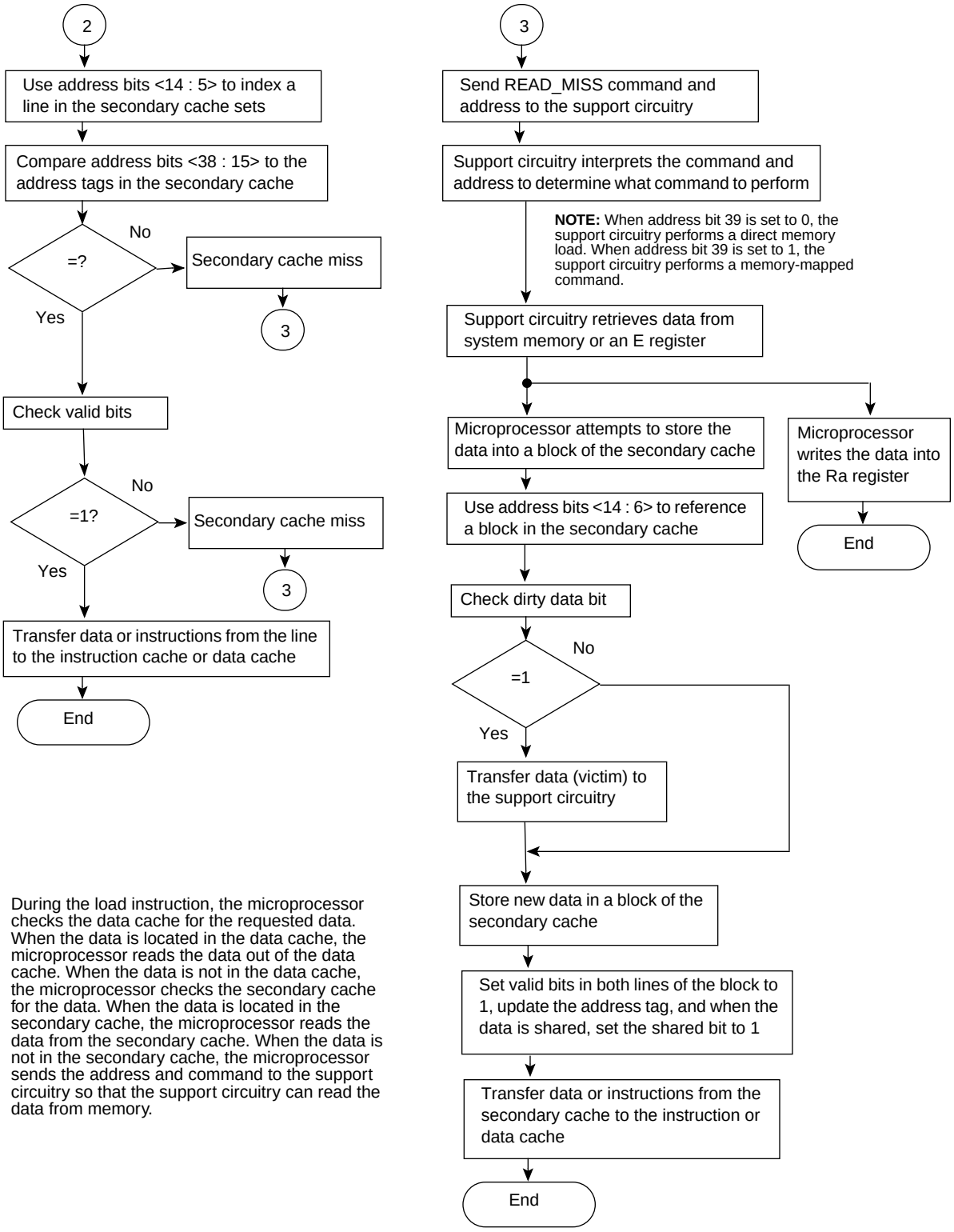


Figure 37. Load Instruction (Continued)



During the load instruction, the microprocessor checks the data cache for the requested data. When the data is located in the data cache, the microprocessor reads the data out of the data cache. When the data is not in the data cache, the microprocessor checks the secondary cache for the data. When the data is located in the secondary cache, the microprocessor reads the data from the secondary cache. When the data is not in the secondary cache, the microprocessor sends the address and command to the support circuitry so that the support circuitry can read the data from memory.