System Components

HMM-150-0

CRAY T3E System

The CRAY T3E system contains tens, hundreds, or thousands of microprocessors, each accompanied by local memory and support circuitry. Each microprocessor, local memory, and support circuitry set is called a processing element (PE). The PEs in the system are designed to support different styles of massively parallel processor (MPP) programming such as data parallel, work sharing, and message passing.

PEs communicate with each other and share the work load of user applications. The PEs communicate using a system of channels called the interconnect network.

The PEs also communicate with other computer systems or peripherals. To do this, the PEs use I/O controllers. The I/O controllers receive and transmit data with the PEs using the interconnect network. The I/O controllers receive and transmit data with other computer systems or peripherals using an external channel.

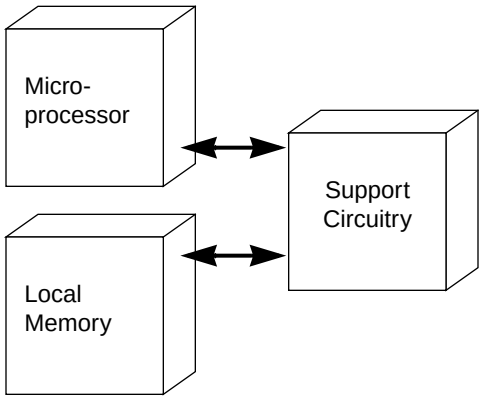
The following subsections describe, in detail, the major components of the CRAY T3E system. These components are:

- Processing elements
- The interconnect network
- I/O controllers

Processing Elements

Figure 1 shows the components of a PE. Most of the PEs in a system run user applications; however, some PEs are reserved for running the CRAY T3E operating system software. For example, in a system that has 48 PEs, 45 PEs may run user applications and 3 PEs may run the operating system software.

Figure 1. PE Components



Although PEs may run either user applications or operating system software, all PEs in the system contain identical hardware. Software determines the number of PEs used for user applications and the number of PEs used for the operating system software.

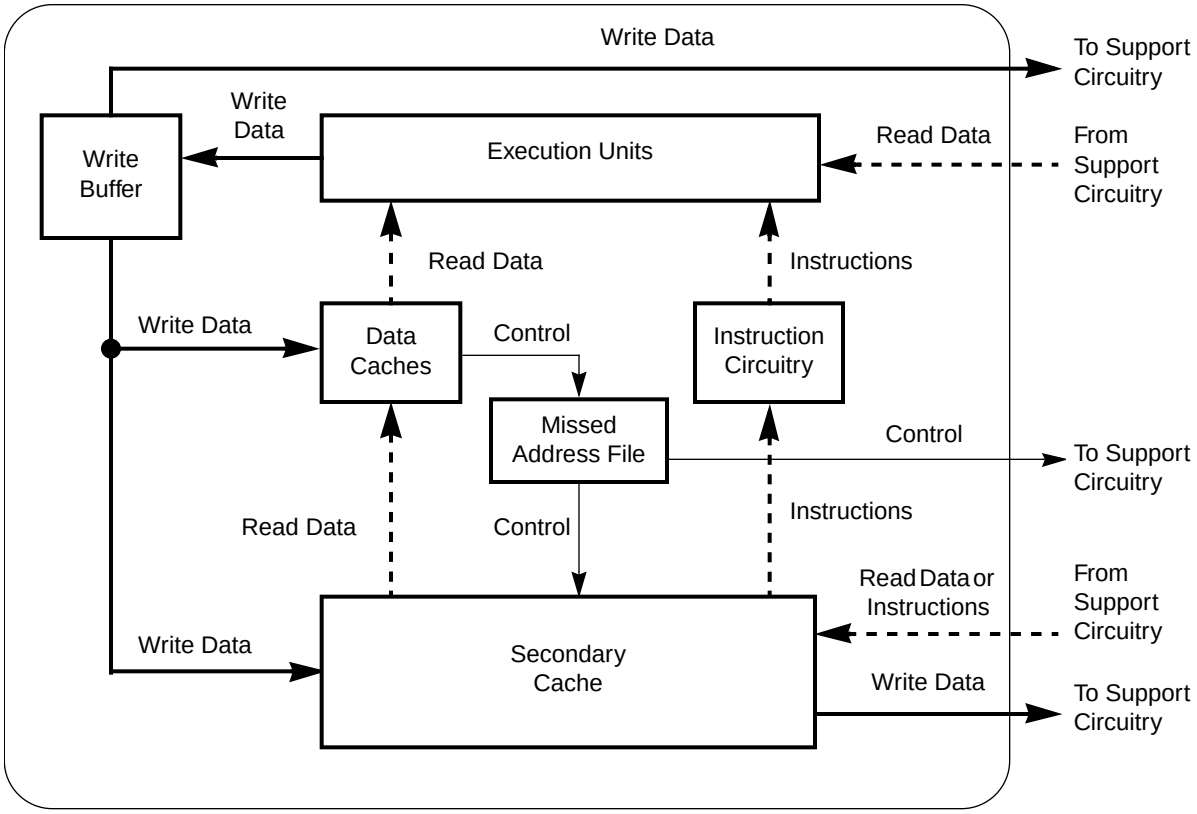
The following subsections describe the microprocessor, support circuitry, and local memory components of a PE.

Microprocessor

The microprocessor is a DECchip™ 21164 reduced instruction set computer (RISC) 64-bit microprocessor developed by Digital Equipment Corporation. The microprocessor performs arithmetic operations on 64-bit integer and 64-bit floating-point registers. These operations include the Institute of Electrical and Electronic Engineers (IEEE) floating point arithmetic.

The microprocessor contains six main components: instruction circuitry, execution units, the write buffer, data caches, the missed-address file, and the secondary cache (refer to Figure 2).

Figure 2. Simplified Microprocessor Components



NOTE: The following subsections provide only enough information for the reader to obtain a basic understanding of the major components of the microprocessor. The *Alpha Architecture Handbook*, publication number CZM-0871-000, provides detailed information on the microprocessor.

Instruction Circuitry

The instruction circuitry retrieves and decodes instructions and issues the instructions to the execution units. The instruction circuitry processes four 32-bit instructions at a time and does not advance to a new set of instructions until all instructions in the current set have issued.

The instruction circuitry may issue four instructions simultaneously to the execution units; however, the instructions must contain one integer multiply or shift instruction, one integer add or logical instruction, one floating-point multiply instruction, and one floating-point instruction other than multiply. To achieve maximum issue rates, software can insert one or more no-operation instructions in any four-instruction set.

The instruction circuitry contains a prefetcher, program counter logic, and an instruction translation buffer (ITB). The prefetcher retrieves instructions from the secondary cache. The program counter logic determines the next instruction to decode. The ITB is a 64-entry table that temporarily stores recently used instruction address translations.

The instruction circuitry also contains an instruction cache that temporarily stores frequently or recently accessed instructions. The instruction cache stores 256 lines; each line contains 32-bytes of instructions (eight 32-bit instructions).

Execution Units

After receiving instructions from the instruction circuitry, the execution units perform the instructions. There are two types of execution units: integer execution units and floating-point execution units.

Integer

The microprocessor contains two integer execution units. Both integer execution units perform addition and logical operations. In addition to these operations, one of the integer execution units performs shift and multiply integer operations.

The integer execution units perform operations on 64-bit integer registers. There are 32 integer registers.

Floating-point

The microprocessor also contains two floating-point execution units. One of the floating-point execution units performs floating-point multiply operations. The other unit performs all other floating-point operations, including addition and division.

The floating-point execution units perform operations on 64-bit floating-point registers. There are a total of 32 floating-point registers. Floating-point operations include the IEEE floating-point arithmetic.

Write Buffer

Each microprocessor contains a write buffer. The write buffer temporarily stores write data before it transfers to the secondary cache or system memory. The write buffer receives write data at a high bandwidth and maximizes the rate at which data transfers to the secondary cache or system memory. The write buffer does this by attempting to merge 32- or 64-bit stores for the same 32-byte line into a single 32-byte store operation.

Format of Write Buffer

The write buffer stores up to six 32-byte lines (eight 32-bit halfwords in a line) of data. Each line in the write buffer contains an address tag, valid bits, a no-merge (NM) bit, a write memory barrier (WMB) bit, and data. Figure 3 shows the format of the write buffer.

Figure 3. Write Buffer Format

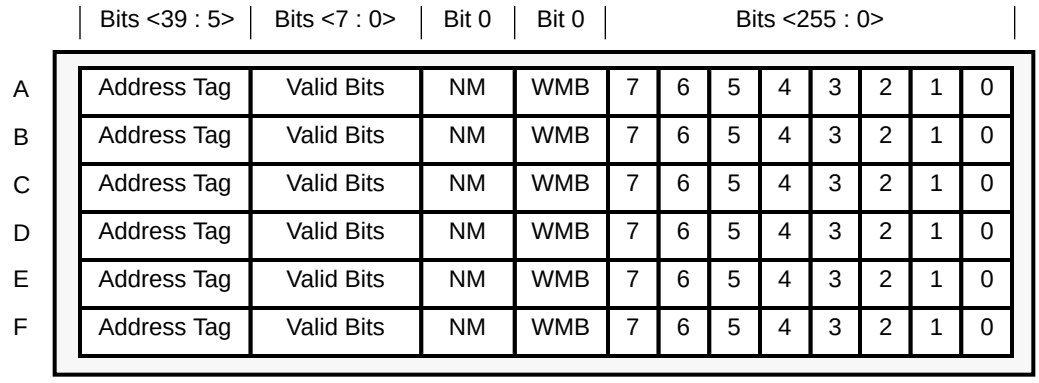


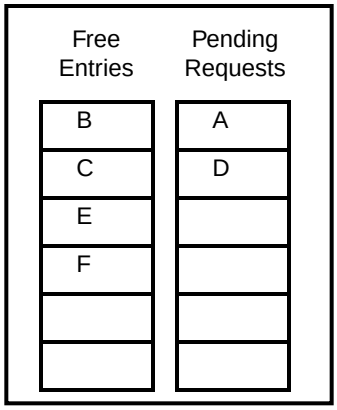
Table 1 describes the bit format of each line in the write buffer.

Table 1. Write Buffer Line Bit Format

Bits	Description
Address tag	The address tag is a 35-bit value that contains bits <39 : 5> of the address used for a store-related instruction. The address tag identifies a line in the write buffer.
Valid bits	The valid bits indicate which 32-bit halfwords are valid in a line of the write buffer. For example, when bits 0 and 7 of the valid bits are set to 1, the valid bits indicate that halfwords 0 and 7 contain valid data in the write buffer line.
No-merge (NM) bit	The NM bit indicates whether the microprocessor can merge new data into a line of the write buffer. When this bit is set to 1, the microprocessor does not merge data into the write buffer line.
Write memory barrier (WMB) bit	The write memory barrier bit indicates that a WMB instruction issued. The WMB instruction controls the ordering of write-related instructions.
Data bits	A line in the write buffer contains a maximum of eight halfwords of data. Some of the data in a line may be invalid.

The write buffer circuitry in the microprocessor also contains two queues: the free-entry queue and the pending-request queue (refer to Figure 4). The free-entry queue contains a list of all the write buffer lines that are empty (do not contain valid data) and indicates which write buffer line will be filled with valid data next. The pending-request queue contains a list of all the write buffer lines that contain valid data.

Figure 4. Free-entry Queue and Pending-Request Queues



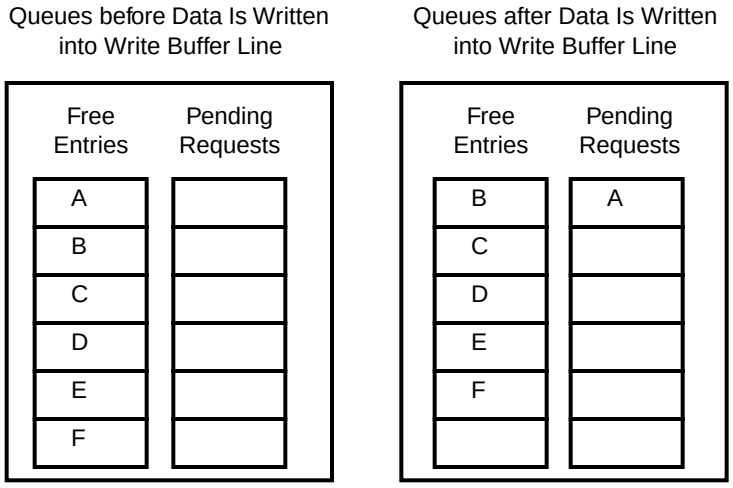
Merging Data or Writing to a New Line in the Write Buffer

When the microprocessor issues a store instruction, it compares bits <39 : 5> of the address for the store instruction with the address tag in each line of the write buffer. When it finds a match and the no-merge bit is reset to 0, the microprocessor merges the new data into the corresponding line of the write buffer. The microprocessor then sets the appropriate valid bits to indicate which new and old 32-bit halfwords in the write buffer line contain valid data.

When bits <39 : 5> of the address for a store instruction do not match an address tag or when the no-merge bit in a matching line is set to 1, the microprocessor writes the data into the write buffer line indicated by the first entry of the free-entry queue. The microprocessor also sets the appropriate valid bits in that line to 1 and writes bits <39 : 5> of the address into the address tag of that line.

After the microprocessor writes data into an empty write buffer line, the microprocessor removes the write buffer line from the free-entry queue and enters the line in the pending-request queue. For example, Figure 5 shows the free-entry and pending-request queues before and after data is written into an empty write buffer line. In this example, all of the lines within the write buffer were initially empty.

Figure 5. Free-entry and Pending-request Queues Example



The microprocessor only merges store instruction data (for example, the store quadword instruction) in an attempt to fill a write buffer line before sending the data to the secondary cache or system memory. The write buffer stores, but does not merge memory barrier (`mb`), store conditional (`stx_c`), `fetch`, and `fetchm` instructions.

Transferring Data Out of the Write Buffer

The microprocessor sends the contents of the write buffer line indicated by the top entry of the pending-request queue to the secondary cache or system memory when the pending-request queue contains two or more entries. Unless the pending-request queue is empty, the microprocessor also sends the contents of a write buffer line indicated by the top entry of the pending-request queue to the secondary cache or system memory once every 64 clock periods. This action ensures that a pending request will not reside in the pending-request queue for more than 64 clock periods.

When the microprocessor sends the contents of a write buffer line to the secondary cache or system memory, the microprocessor removes the write buffer line number from the pending-request queue; however, the microprocessor does not re-enter the write buffer line number in the free-entry queue until the write operation completes. Because of this characteristic, write buffer line numbers may re-enter the free-entry queue in a different order than they were read out of the pending-request queue.

Certain conditions cause the microprocessor to flush the write buffer. To flush the write buffer means that the microprocessor continues to send the contents of each line indicated in the pending-request queue to the secondary cache or system memory until the pending-request queue is empty. The following conditions cause the microprocessor to flush the write buffer:

- One of the write buffer lines in the pending-request queue contains an `mb`, a `stx_c`, a `fetch`, or a `fetchm` instruction. The microprocessor flushes the write buffer to ensure that these instructions issue and complete as soon as possible.

- One of the write buffer lines in the pending-request queue has the write memory barrier bit set to 1.

- The microprocessor tries to issue a load instruction and bits <39 : 5> of the address for the load instruction match an address tag in the write buffer. The microprocessor flushes the write buffer to ensure that the data in the write buffer is written into the secondary cache or system memory before the load instruction attempts to read data from the same location in the secondary cache or system memory.

When the microprocessor receives a `WMB` instruction, the microprocessor sets the NM bit to 1 in each line of the write buffer that is identified in the pending-request queue. The microprocessor also sets the WMB bit to 1 in the line indicated by the top entry of the free-entry queue.

After storing data in the line that has the WMB bit set to 1, the microprocessor removes that line number from the free-entry queue and enters the line number in the pending-request queue. The microprocessor then flushes the write buffer with the exception of the write buffer line with the WMB bit set to1. For this line, the microprocessor waits until all previous store operations complete before sending the line to the secondary cache or system memory.

When bits <39 : 5> of the address for a store instruction and the address tag for a line match and the NM bit is set to 1 for that line, the microprocessor places the store data in the line indicated by the top entry of the free-entry queue and sets the WMB bit for that line to 1.

When data and address information for the store instruction (possibly merged store instructions) leaves the write buffer, the microprocessor first checks the value of bit 39 of the address. When set to 1, the store instruction does not affect data in the data cache and secondary cache. In this case, the microprocessor transfers the data and address information for the store instruction to the support circuitry.

When the microprocessor presents the data and address information to the support circuitry, the microprocessor also generates a WRITE_BLOCK command. Data transfers from the write buffer in the microprocessor to the support circuitry in two 128-bit transfers.

Using the valid bits from the write buffer line, the microprocessor indicates to the support circuitry which 32-bit halfwords in each 128-bit transfer contain valid data. The support circuitry uses this information to determine which halfwords to update in system memory.

When bit 39 of the address is set to 0, the store instruction affects data in the data cache and secondary cache. In this case, the microprocessor updates data in the data cache and secondary cache instead of transferring the data and address information to the support circuitry.

Load Instructions and the Write Buffer

When the microprocessor encounters a data cache miss (described in the next subsection), it compares bits <39 : 5> of the address for the load instruction with each address tag in the write buffer. When it finds a match, the microprocessor sets the no-merge bit to 1 in the matching write buffer line and flushes the write buffer. This process ensures that the data in the write buffer is written into the secondary cache or system memory before the microprocessor attempts to read data from that same location. When bits <39 : 5> of the address do not match an address tag, the load instruction does not affect the write buffer.

Data Cache

A data cache is a small, high-speed random access memory that temporarily stores frequently or recently accessed data. The microprocessor contains two data caches that store identical data. Each data cache is associated with one integer execution unit and floating-point execution unit.

Each data cache stores 256 lines of data with 32-bytes per line (total of 8 Kbytes of data). Parity bits protect data in the data cache, and the microprocessor ensures that the data cache is always a subset of the secondary cache.

Format of Data Cache

Each line in the data cache contains an address tag, 2 valid bits, and data. Figure 6 and Table 2 describe the format of the data cache.

Figure 6. Data Cache

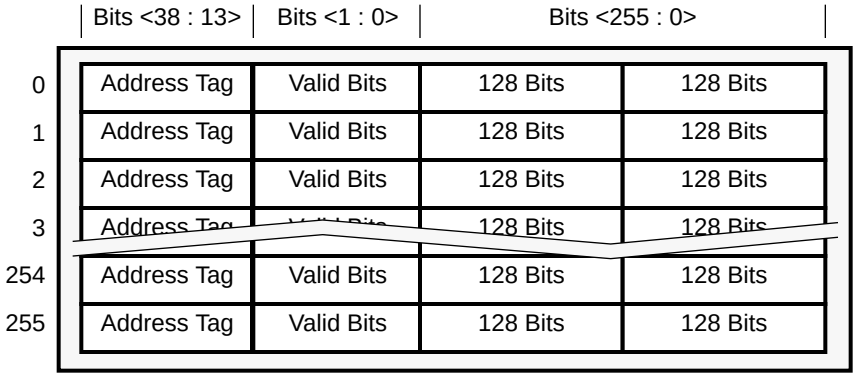


Table 2. Data Cache Line Bit Format

Bits	Description
Address Tag	The address tag contains address bits <38 : 13> for the data in the cache line.
Valid Bits	The valid bits indicate whether the data in the data cache line is valid. For example, when bit 0 of the valid bits is set to 1, this bit indicates that data bits <127 : 0> contain valid data. Generally, the data in a data cache line is considered valid if it matches valid data stored in a corresponding secondary cache line.
Data	Each line in the data cache contains 32 bytes of data. Some of the data in a line may be invalid.

Loads and the Data Cache

When the microprocessor issues a load instruction, it checks the value of bit 39 of the address for the load instruction. When bit 39 is set to 1, this bit indicates that the data is not stored in the data cache and the microprocessor enters the address in the missed-address file. When this occurs, it is called a data cache miss. A data cache miss causes the microprocessor to read the correct data from the secondary cache or system memory instead of reading invalid data in the data cache.

When bit 39 of the address is set to 0, this bit indicates that the data may be stored in the data cache. The microprocessor uses bits <12 : 5> of the address for the load instruction as an index that points to one of the 256 lines in the data cache. For example, when bits <12 : 5> of the address are set to 12, the microprocessor references data cache line 12.

After referencing a data cache line, the microprocessor compares address bits <38 : 13> to the address tag of the data cache line. When they match, the line may contain the data requested by the load instruction. The microprocessor then examines the valid bits of the cache line. When the appropriate valid bit is set to 1, the microprocessor retrieves the valid data from the data cache line and transfers the data to the execution unit register specified in the load instruction. When the valid bit is set to 0, the data is invalid. In this case, the microprocessor encounters a data cache miss.

When bits < 38 : 13> of the address and the address tag of the selected data cache line do not match, the cache line does not contain the data requested by the load instruction. When this occurs the microprocessor encounters a data cache miss. When a data cache miss occurs, the microprocessor enters the address into the missed-address file.

Stores and the Data Cache

After issuing a store instruction with address bit 39 set to 0, the microprocessor compares the address for the store instruction with the address tags in the data cache. When the microprocessor finds a match, and the appropriate valid bit is set to 1, the microprocessor updates the specified amount of data in the data cache line of both data caches before completing the store instruction. When the address and an address tag do not match or the valid bit is not set to 1 in a matching line, the microprocessor does not modify data in the data cache.

Missed-address File

When the microprocessor issues a load instruction and the data is not in the data cache or the data is noncacheable, it enters the address and destination integer or floating-point register number for the load instruction in the missed-address file (MAF). If the microprocessor load circuitry is busy, the MAF temporarily stores load instructions in an attempt to merge 32- or 64-bit load instructions from the same 32-byte line into one 32-byte load operation.

Format of the MAF

The MAF contains 10 entries: six entries store data load misses and four entries store instruction fetch or prefetch misses. Figure 7 and Table 3 show a simplified format of the data load entries in the MAF. Each line in the MAF contains an address tag, a data cache (Dcache) bit, and a word mask. In addition to this information, each line in the MAF also contains format and instruction identification information (not shown in Figure 7).

Figure 7. Missed-address File

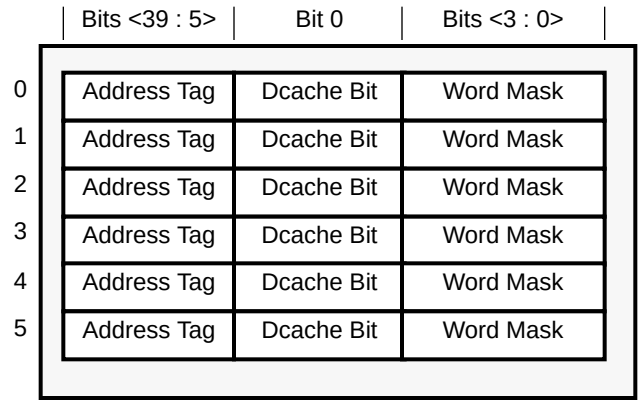


Table 3. Missed-address File Line Bit Format

Bits	Description
Address tag	The address tag contains address bits <39 : 5> for the load instruction.
Data cache (Dcache) bit	The Dcache bit indicates whether the microprocessor writes returning load data into the data cache. When set to 1, this bit indicates the microprocessor will not write data into the data cache.
Word mask	The word mask indicates which 64-bit words in a 4-word line are requested by the load instruction. For example, if bit 1 of the word mask is set to 1, word 1 (bits <127 : 64>) of a line is requested.

Loads and the MAF

When the microprocessor issues a load and encounters a data cache miss, the microprocessor first determines whether the MAF contains any pending load instructions. When the MAF does not contain pending instructions, the microprocessor loads data from the secondary cache or system memory, and does not enter the load instruction in the MAF.

When the MAF does contain pending load instructions, the microprocessor compares address bits <39 : 5> for the new load instruction with the address tag in each line of the MAF. When the microprocessor does not find a match, the microprocessor enters the load instruction in a new line of the MAF.

When the microprocessor does find a match, the microprocessor determines whether the instruction information of that line in the MAF contains the same type of instruction as the new load instruction. The microprocessor does not merge floating-point and integer load instructions in the same MAF entry. In addition, the microprocessor does not merge loads of different sizes. For example, the microprocessor does not merge 32-bit and 64-bit load operations.

When the new load instruction and the instruction stored in the MAF entry are the same type of instruction, the microprocessor attempts to merge the instructions. The microprocessor only merges the instructions when the current instruction accesses a different 64-bit word than the instruction stored in the MAF entry. After merging the instructions, the microprocessor sets the appropriate word mask bit to 1 for that MAF entry.

When the microprocessor load circuitry is free, the microprocessor performs a load of data from the secondary cache or system memory for each entry in the MAF. When a MAF entry is full (all of the word mask bits are set to 1), the microprocessor loads 32-bytes (4 words) from the secondary cache or system memory.

Stores and the MAF

When the microprocessor issues a store instruction, the microprocessor compares address bits <39 : 5> for the store instruction to the address tag in each line of the MAF. When it does not find a match, the store instruction does not affect the MAF.

When the microprocessor does find a match between the store instruction address and an address tag, the MAF contains a load instruction that references the same data as the store instruction. When this occurs, the microprocessor issues the load instruction in the MAF before issuing the store instruction.

In addition, when this occurs, the microprocessor sets the Dcache bit to 1 in the MAF line that contains the load instruction with the matching address. When set to 1, the Dcache bit ensures that the microprocessor does not write the returned data from the load instruction into the data cache. This forces a data cache miss the next time the microprocessor tries to load data from that address location. The data cache miss causes the microprocessor to read the correct data from the secondary cache or system memory instead of reading invalid data in the data cache.

Secondary cache

The secondary cache is a combined instruction and data cache. The secondary cache contains three 32-Kbyte sets for a total of 96 Kbytes of storage. Each set stores 512 blocks with 64 bytes of data per block.

Parity bits protect data in the secondary cache. The support circuitry ensures that data in the secondary cache is always a subset of local memory.

Format of the Secondary Cache

Each 32-Kbyte set in the secondary cache contains 512 blocks. Each block contains two lines that store 32-bytes of data.

Figure 8 shows the format of the secondary cache. In addition to data, each line in a set contains an address tag, a dirty bit, a shared bit, a valid bit, and 2 modification (MOD) bits.

Figure 8. Secondary Cache

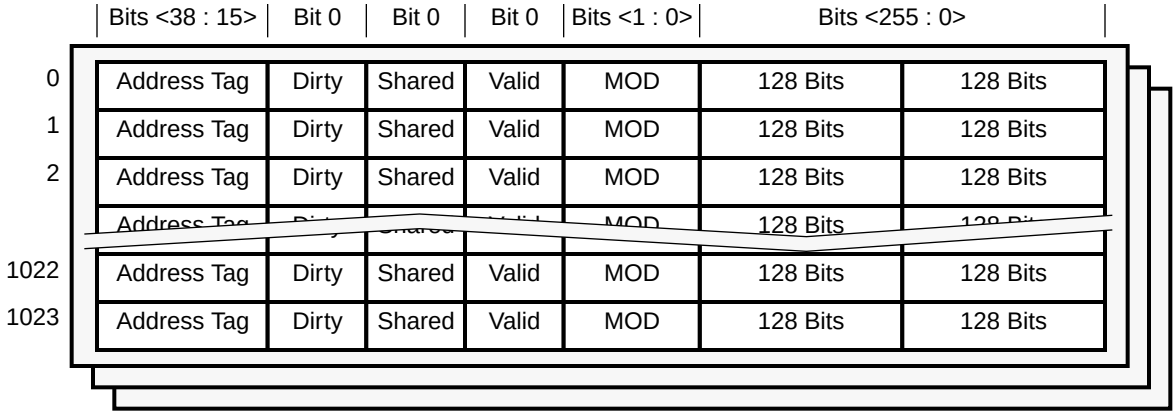


Table 4 shows the bit format of each secondary cache line.

Table 4. Secondary Cache Line Bit Format

Bits	Description
Address tag	The address tag contains address bits <38 : 15> for the data in the secondary cache line.
Dirty bit	The dirty bit indicates when a block contains dirty data. Dirty data is data that has been modified by the microprocessor and has not been transferred to the support circuitry to be updated in local memory. Because each block contains two lines, the dirty bits for the two lines in a block are always identical. In the CRAY T3E system, the support circuitry does not use the dirty bit.
Shared bit	The shared bit indicates when a block contains shared data. Shared data is data in system memory that can be accessed and modified by more than one microprocessor. When data is shared and the microprocessor modifies the data in the secondary cache, the data is also transferred to the support circuitry to be updated in system memory. Because each block contains two lines, the shared bits for the two lines in a block are always identical. In the CRAY T3E system, the support circuitry does not use the shared bit.
Valid bit	The valid bit indicates the validity of the data in a block. Valid data is data that was modified by the microprocessor or was updated by the support circuitry and the data still matches the original data in system memory. Because each block contains two lines, the valid bits for the two lines in a block are always identical.
Modification (MOD) bits	The 2 modification bits help prevent unnecessary writes from the secondary cache to system memory. Each bit corresponds to one 128-bit block of the secondary cache line. For example, when MOD bit 1 is set to 1, this bit indicates that the microprocessor modified data is located within bits <255 : 128> in the line.
Data bits	Each line stores two 128-bit blocks of data. This corresponds to one write buffer line, one data cache line, or one instruction cache line.

Loads and the Secondary Cache

The microprocessor attempts to load data or instructions from the secondary cache when the microprocessor receives a load request from the MAF and bit 39 of the address is set to 0, or when an instruction fetch or prefetch misses in the instruction cache. The microprocessor uses address bits <14 : 5> as an index that points to one of the 1,024 lines in each set of the secondary cache. For example, when bits <14 : 5> of the address are set to 12, the microprocessor references line 12 in all three secondary cache sets.

The microprocessor then compares bits <38 : 15> of the address to the address tag in each of the three lines referenced in the secondary cache. When the microprocessor does not find a match, it is called a secondary cache miss. The secondary cache miss causes the microprocessor to signal the support circuitry to retrieve the data or instructions from system memory.

When the microprocessor does find a match in one of the three lines, the microprocessor checks the valid bit in that line. When the valid bit is not set, the line does not contain valid data or instructions and the microprocessor encounters a secondary cache miss. When the valid bit is set, the microprocessor transfers the requested data or instructions from the secondary cache line to the data or instruction cache.

When the microprocessor encounters a secondary cache miss, the microprocessor sends a READ MISS command and an address to the support circuitry. The support circuitry then retrieves a 64-byte block of data from system memory and transfers the data to the microprocessor. The microprocessor then attempts to write the data into a block (two lines) of one of the sets in the secondary cache. The block is referenced by bits <14 : 6> of the address.

When the corresponding block in one of the three sets does not contain valid, dirty data, the microprocessor updates that block with the new data. After updating the secondary cache block, the microprocessor sets the valid bits for the two lines in the block to 1 and updates the address tags. When the data is shared data, the microprocessor also sets the shared bits to 1.

When the corresponding block in all three sets contains valid, dirty data, the microprocessor transfers the valid, dirty data from one of the blocks to the support circuitry before updating that block with the new data. When this occurs, the data that is removed from the secondary cache to make room for the new data is called a victim.

Stores and the Secondary Cache

The microprocessor stores data in the secondary cache when the microprocessor transfers a store request out of the write buffer and bit 39 of the address is set to 0. The microprocessor uses address bits <14 : 5> as an index that points to one of the 1,024 lines in each set of the secondary cache. For example, when bits <14 : 5> of the address are set to 12, the microprocessor references line 12 in all three secondary cache sets.

The microprocessor then compares bits <38 : 15> of the address to the address tag in each of the three lines referenced in the secondary cache. When the microprocessor does not find a match, it presents a read miss and modify (READ MISS MOD) command to the support circuitry. This signals the support circuitry to retrieve a 64-byte block of data from system memory. The microprocessor stores this block of data into the secondary cache and sets the valid bits to 1. This process ensures that the microprocessor finds a match between the address for the store instruction and an address tag, and the data in that line is valid.

When the microprocessor finds a match between the address and an address tag, the microprocessor examines the valid bit for that line. When the valid bit is set to 1, the microprocessor continues the store. When the valid bit is set to 0, the microprocessor sends a READ MISS MOD command to the support circuitry. After receiving data from the support circuitry, the microprocessor stores the data in the secondary cache, sets the valid bit to 1, and continues the store.

Through store operations, the microprocessor in the local PE may modify data in its secondary cache. Through remote write operations, a remote PE may modify the data in local memory of the PE. Because of these characteristics, the support circuitry must perform functions that keep the data in the secondary cache consistent with the data in local memory.

The support circuitry contains a backmap of the secondary cache. A backmap is a table that contains a copy of bits <21 : 15> of the address tag for each block in the secondary cache. The support circuitry uses the backmap and a microprocessor function called FLUSH, to keep data in the secondary cache consistent with data in local memory.

FLUSH and the Secondary Cache

After receiving a request from a remote PE to store data in local memory, the support circuitry compares the address for the remote request to the address tags in the backmap. When the support circuitry does not find a match between the address and an address tag in the backmap, the support circuitry stores the remote data into local memory and sends an acknowledgment to the remote PE. In this case, the support circuitry does not notify the local microprocessor that the data transfer occurred.

When the support circuitry does find a match between the address and address tag in the backmap, the support circuitry sends a FLUSH command and an address to the microprocessor. The microprocessor uses bits <14 : 5> of the address to point to one of the 1,024 lines in each set of the secondary cache. The microprocessor then compares bits <38 : 15> of the address to the address tag in each of the three lines referenced in the secondary cache.

When the microprocessor finds a match between the address and an address tag in the secondary cache line, the microprocessor examines the valid and dirty bits for the block. Table 5 lists the values of the valid and dirty bits and describes the actions performed for each value.

Table 5. Valid and Dirty Bit Values for a FLUSH

Valid	Dirty	Result
0	X	The microprocessor acknowledges the FLUSH command. The support circuitry then stores the remote data into local memory.
1	0	The microprocessor sets the valid bit to 0 for the secondary cache block and acknowledges the FLUSH command. The support circuitry then stores the remote data into local memory.
1	1	The microprocessor transfers the contents of the block from the secondary cache to the support circuitry so it can be stored in local memory. The microprocessor then resets the valid bit and dirty bits to 0 and acknowledges the FLUSH command. The support circuitry then stores the remote data into local memory.

NOTE: X means the value of this bit does not matter.

When the microprocessor does not find a match between the address and an address tag in the secondary cache line, the microprocessor acknowledges the FLUSH command. The support circuitry then stores the remote data into local memory.

Microprocessor Specifications

Table 6 lists specifications of the microprocessor.

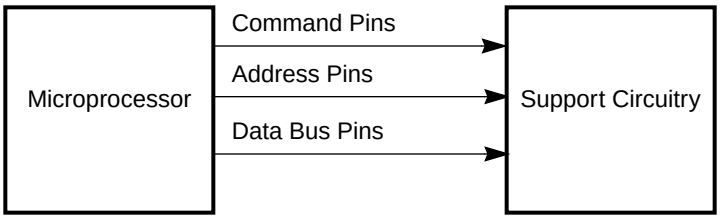
Table 6. Microprocessor Specifications

Characteristic	Specification
Microprocessor type	Reduced instruction set computer (RISC) 64-bit microprocessor
Bi-directional data bus	128 data bits plus 16 check bits
Address bus	40 bits of address information
Issue rate	Four instructions per clock period maximum (two integer and two floating-point instructions)
Internal instruction cache	8 Kbytes of storage
Two integer execution units	One issues addition and logical operations and one issues addition, logical, shift, and multiply operations
Total number of integer registers	Thirty-two 64-bit registers
Two floating-point execution units	One issues floating-point multiply operations and one issues all floating-point operations except multiply
Total number of floating-point registers	Thirty-two 64-bit registers
Write buffer	Stores up to six 32-byte lines (8 halfwords per line) of store data
Two Internal data cache memories	Each stores up to 256 lines (8 halfwords per line) of data
Missed-address file	Stores up to six 32-byte data load misses and four 32-byte instruction fetch misses
Secondary cache	Combined instruction and data cache that stores up to 96 Kbytes of data and instructions in three 32-Kbyte sets (512 64-byte blocks per set)
Data error protection	Hardware corrects single-bit errors and detects double-bit errors
Integrated circuit type	Complimentary metal-oxide semiconductor (CMOS)
Peak floating-point rate	>500 MFLOPS
Peak data bandwidth	>1,333 MBytes/s
Total number of pins	503
Number of signal pins	289

Support Circuitry

The support circuitry is an interface between the microprocessor and local memory in a PE, and between the microprocessor and the interconnect network. The support circuitry receives information from the microprocessor over three types of microprocessor pins (refer to Figure 9).

Figure 9. Microprocessor Pins



The command pins indicate to the support circuitry when the microprocessor is requesting a write-related command (WRITE_BLOCK) or a read-related command (READ_MISS or READ_MISS_MOD). During write-related commands, the microprocessor transfers data or command information to the support circuitry. During read-related commands, the microprocessor receives data from the support circuitry.

The address pins transfer two types of information to the support circuitry: a physical address or command information. When software places a physical address on the address pins, the support circuitry uses the address to directly reference data in local memory. When software places command information on the address pins, the support circuitry uses this information to perform a global-memory command. (Global memory is a sum of all the local memories in the CRAY T3E system.)

NOTE: Software may place other types of address information on the address pins that reference external registers.

The data pins transfer two types of information to the support circuitry: data, or command and address information. When software places data on the data pins, the support circuitry stores the data directly into local memory. When software places command and address information on the data pins, the support circuitry uses the command and address information to reference data in global memory.

The support circuitry performs two main types of operations: local operations and global operations.

Local Operations

The support circuitry performs several types of local operations. These operations include direct local memory loads, direct local memory stores, secondary cache control, and local memory control.

Direct Local Memory Load

The microprocessor issues a direct local memory load command when the microprocessor encounters a secondary cache miss (refer to “Secondary Cache” in this document for more information on a secondary cache miss). When this occurs, the microprocessor presents a read-related command and a physical address to the support circuitry. The support circuitry then loads data from local memory and transfers the data to the microprocessor.

In addition to transferring data during a direct local memory load, the support circuitry contains circuitry that detects streams. A stream is a set of load commands from the microprocessor that reference contiguous addresses in local memory.

When the support circuitry detects that a stream is occurring, it automatically prefetches data from the next location in local memory and temporarily stores the data in a stream buffer. When the support circuitry receives the next command in the stream, the support circuitry transfers data from the stream buffer to the microprocessor instead of accessing local memory. This process results in a reduction of the data load latency from the support circuitry to the microprocessor.

Direct Local Memory Store

The microprocessor issues a direct local memory store command when the microprocessor creates a secondary cache victim (refer to “Secondary Cache” in this document for more information on a secondary cache victim). When this occurs, the microprocessor presents a write-related command, a physical address, and data to the support circuitry. The support circuitry then writes the data in the appropriate local memory location.

Secondary Cache Control

When the support circuitry receives requests from remote PEs to modify data in local memory, the support circuitry ensures that data stored in the microprocessor secondary cache is always consistent with data in local memory. The support circuitry uses a secondary cache backmap to maintain a record of the contents of the secondary cache. Using this backmap, the support circuitry determines when to send a FLUSH or invalidate command to the microprocessor.

Refer to “Secondary Cache” in this document for more information on the secondary cache backmap and secondary cache control.

Local Memory Control

The support circuitry provides control for the local memory. This control includes memory refresh and memory reference scheduling.

Global Operations

The support circuitry performs several types of global operations. These operations include global address translation, global data transfers, atomic E-register commands, barrier and eureka synchronization, and messaging.

Global Address Translation

The support circuitry receives address information from the microprocessor. For direct local memory transfers (as described in the previous subsection), software places the complete local memory physical address on the microprocessor address pins. For global memory transfers, software places address and command information on the address pins and the data pins.

When software requests a global data transfer, the data pins contain a software-defined global index. The support circuitry translates the global index into a PE number and a physical address into the local memory of that PE.

The support circuitry uses several components to translate the global index into a PE number and physical address. These components include the centrifuge, an offset adder, the segment translation table (STT), and the global translation buffer (GTB).

The centrifuge uses software-defined parameters to separate the global index into a PE number and an index offset. The index offset is an offset in the memory of a PE with respect to a software-defined base virtual address.

The offset adder uses the index offset and the base virtual address to generate a user virtual address. The user virtual address is a byte-oriented address that the program compiler uses.

The STT is a look-up table that stores virtual address translation parameters. The support circuitry uses these parameters to convert a user virtual address into a global virtual address. The global virtual address is an intermediate address used by the operating system.

The GTB stores frequently or recently accessed global-to-physical translation parameters. The support circuitry uses these parameters to convert a global virtual address into a physical address.

Global Data Transfers

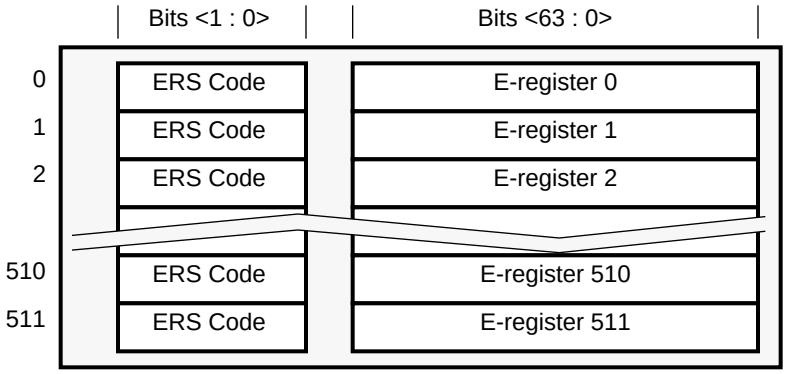
The support circuitry controls all data transfers between the microprocessor and global memory. The support circuitry does this using a set of latency-hiding registers called E registers.

E Registers

E registers are the source or destination for all global data transfers. The support circuitry has two sets of E registers: context 0 and context 1. Context 0 contains 512 E registers. Context 1 contains 128 E registers.

Each E register is 64-bits wide and has an associated 2-bit E-register state (ERS) code (refer to Figure 10).

Figure 10. E-register Bit Format of Context 0



The ERS code indicates the present state of the E register. Table 7 lists the possible ERS code values and describes each state.

Table 7. ERS Codes

Bit 1	Bit 0	Name	Description
0	0	EMPTY	The E register is empty.
0	1	FULL_F	The E register is full; however, a deferred software error was detected during the transfer.
1	0	FULL	The E register is full and no errors occurred during the transfer.
1	1	FULL_SR	The E register is full; however, a previous message send (SEND) command that used the E register was rejected.

Sample Global Data Transfer

As a simple example, the following paragraphs describe a transfer of data from global memory to the microprocessor.

To start the transfer, the microprocessor presents an E-register command (on the command and address pins) and a global address (on the data pins) to the support circuitry. This information signals the support circuitry to get data from a global memory location (referenced by the global index on the data pins) and return it to a destination E register.

The support circuitry interprets the command and address information and sets the ERS code for the destination E register to indicate that the E register is empty. Then, the support circuitry transfers system data from a location in global memory to the destination E register in the support circuitry. After storing data in the destination E register, the support circuitry sets the ERS code for the E register to indicate that the E register is full.

When the microprocessor needs the data, the microprocessor presents a read-related command and an E register address to the support circuitry. After receiving this information, the support circuitry transfers the system data from the E register to the microprocessor.

Atomic E-register Commands

The support circuitry contains circuitry that performs atomic E-register commands. Atomic E-register commands read, modify, and/or write system data in an indivisible operation. The atomic E-register commands include:

- Get-and-add (GET_ADD)
- Get-and-increment (GET_INC)
- Get-and-set (GET_SET)
- Get-and-clear (GET_CLEAR)
- Store centrifuge result (STORE_CENT)
- Store population and leading zero count (STORE_PLZ)

Barrier and Eureka Synchronization

The support circuitry contains barrier and eureka (B/E) synchronization circuitry. B/E synchronization provides a low-latency method of synchronizing all or part of the PEs in a partition. A partition is a group of PEs assigned to one application.

Using barrier synchronization, software can identify when all of the PEs in a partition have reached the same point in a process. Software may use barrier synchronization for operations such as continuing a matrix multiply operation only after all the PEs in a partition have completed their portion of the calculation.

Using eureka synchronization, software can identify when one or more of the PEs in a partition have reached a point in a process. Software may use eureka synchronization for operations such as terminating a distributed data base search as soon as one PE finds the information.

Messaging

The support circuitry contains messaging circuitry. A message is a packet of information that is sent from one PE 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. The microprocessor then reads the messages off of the message queue.

Local Memory

Local memory consists of eight banks of DRAM that store system data. A low-latency, high-bandwidth data path through the support circuitry connects the microprocessor to local memory in each PE. The size of local memory varies depending on the type of DRAM ICs used in the system (refer to Table 8).

Table 8. Local Memory

Size in Bytes	Size in 64-bit Words	ICs
128 Mbytes	16 Mwords	16-Mbit stacked DRAMs
256 Mbytes	32 Mwords	64-Mbit DRAM
512 Mbytes	64 Mwords	64-Mbit stacked DRAMs

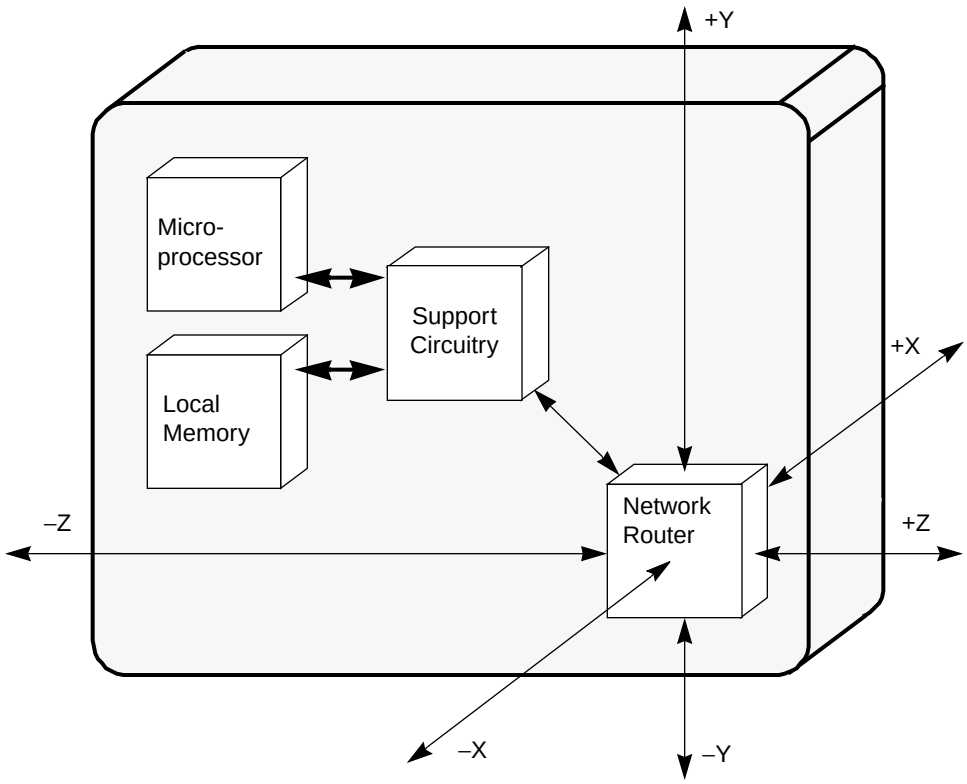
Each local memory is part of a physically distributed, logically shared global memory. Global memory is physically distributed because each PE contains local memory. Global memory is logically shared because any microprocessor can access data in the memory of any PE without interrupting and involving the microprocessor in that PE.

Interconnect Network

In the CRAY T3E system, the interconnect network provides communication among the PEs. The interconnect network contains three types of components: network ports, network routers, and communication links.

Figure 11 shows how a network port, a network router, and six communication links connect to one PE. The PE and interconnect network components shown in Figure 11 are also collectively referred to as a node.

Figure 11. PE and Interconnect Network Components



The following subsections provide a brief description of the interconnect network components.

Network Port

The network port is a pair of unidirectional channels that connect the support circuitry to a network router. Each PE uses a network port to send and receive packets of information over the interconnect network.

Network Routers

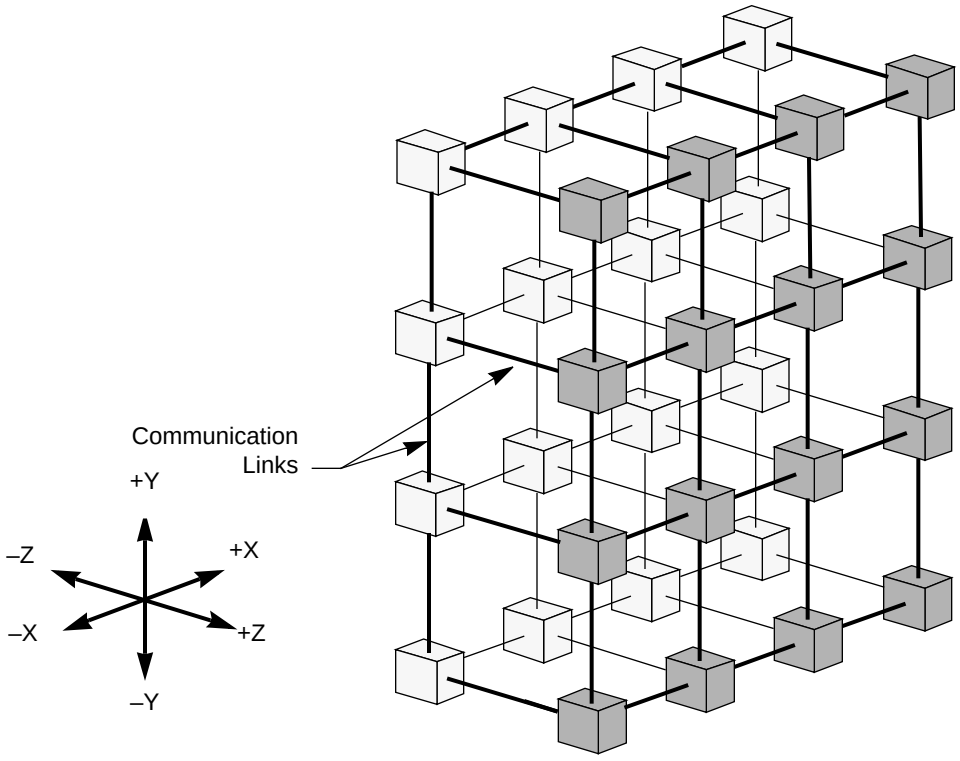
Network routers steer data and control information from a communication link or the network port to another communication link or the network port.

Communication Links

Communication links transfer data and control information between network routers in the interconnect network. Each communication link connects two network routers.

The communication links form a three-dimensional matrix of paths that connect nodes in the X, Y, and Z dimensions (refer to Figure 12). For clarity, Figure 12 shows a simplified interconnect network and may not reflect an actual configuration of the CRAY T3E system.

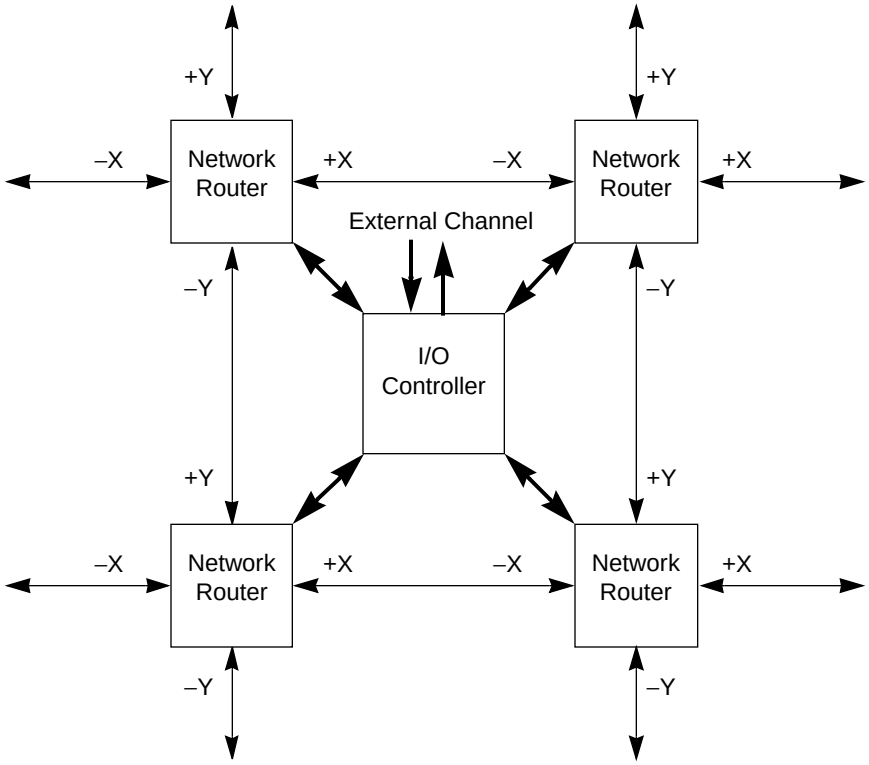
Figure 12. Communication Links in the Interconnect Network



I/O Controllers

I/O controllers transfer data between any PE in the CRAY T3E system and another computer system or a peripheral device. The CRAY T3E system contains one I/O controller for every four network routers (refer to Figure 13). For clarity, Figure 13 does not show the Z-dimension communication links.

Figure 13. I/O Controller Connections



Although each I/O controller is physically associated with a group of four network routers, I/O data is received from or sent to any PE in the CRAY T3E system.

Software may enable or disable each of the I/O controller network router ports. This feature enables software to logically remove the connection between a failing network router port or network router and the I/O controller.

Index

G

I

L

M

N

P

R

S

V

W