CRAY SSD-T90: Barrier Synchronization Operations

Record of Revision

February 1997

Original printing.

Introduction

When a barrier and eureka (B/E) synchronization failure occurs, you may need to replace a field replaceable unit (FRU) or map out a processing element node. An FRU for a B/E synchronization failure may be a processing element module or a ribbon cable.

This document describes the components of the B/E synchronization circuitry. This document also lists the diagnostic tests that are available for troubleshooting a B/E synchronization failure. Specifically, this document answers the following questions:

1. What components make up the barrier and eureka synchronization circuitry?
2. How are barrier and eureka partitions created?
3. How are barrier signals transferred between the nodes in a partition?
4. How does a node determine when a barrier or eureka synchronization operation is complete?

Terms

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

Processing element (PE) - A PE contains a microprocessor, support circuitry, and local memory.

Support circuitry - The support circuitry contains a portion of the barrier circuitry and passes barrier signals between the microprocessor and the network router option.

Network router option- A network router option contains a portion of the barrier circuitry and passes barrier signals through the interconnect network.

Node - A node contains a PE and a network router option.

Partition - A partition is a group of processing element nodes that software assigns to one application.

Barrier and Eureka Synchronization

Barrier synchronization provides a low-latency method of synchronizing all or part of the PEs in the CRAY SSD-T90 device. The barrier synchronization circuitry may perform two types of synchronization: barrier and eureka. A barrier is an event that a process initiates to indicate that all of the PEs in a partition are at a predetermined point in a program. A eureka is an event that a process initiates to indicate that at least one of the PEs in a partition is at a specific point in a program.

Each processing element node contains circuitry to support B/E synchronization. When software initializes a partition, one node is assigned as the root node. The root node acts as the master administrator for B/E functions within the partition.

Barrier Synchronization

Programmers use barrier synchronization to ensure that all PEs within a partition reach a specified point in the program before the PEs continue with the next task. The following text describes the barrier synchronization for a 4-PE partition. The step numbers correspond to the numbers in Figure 1.

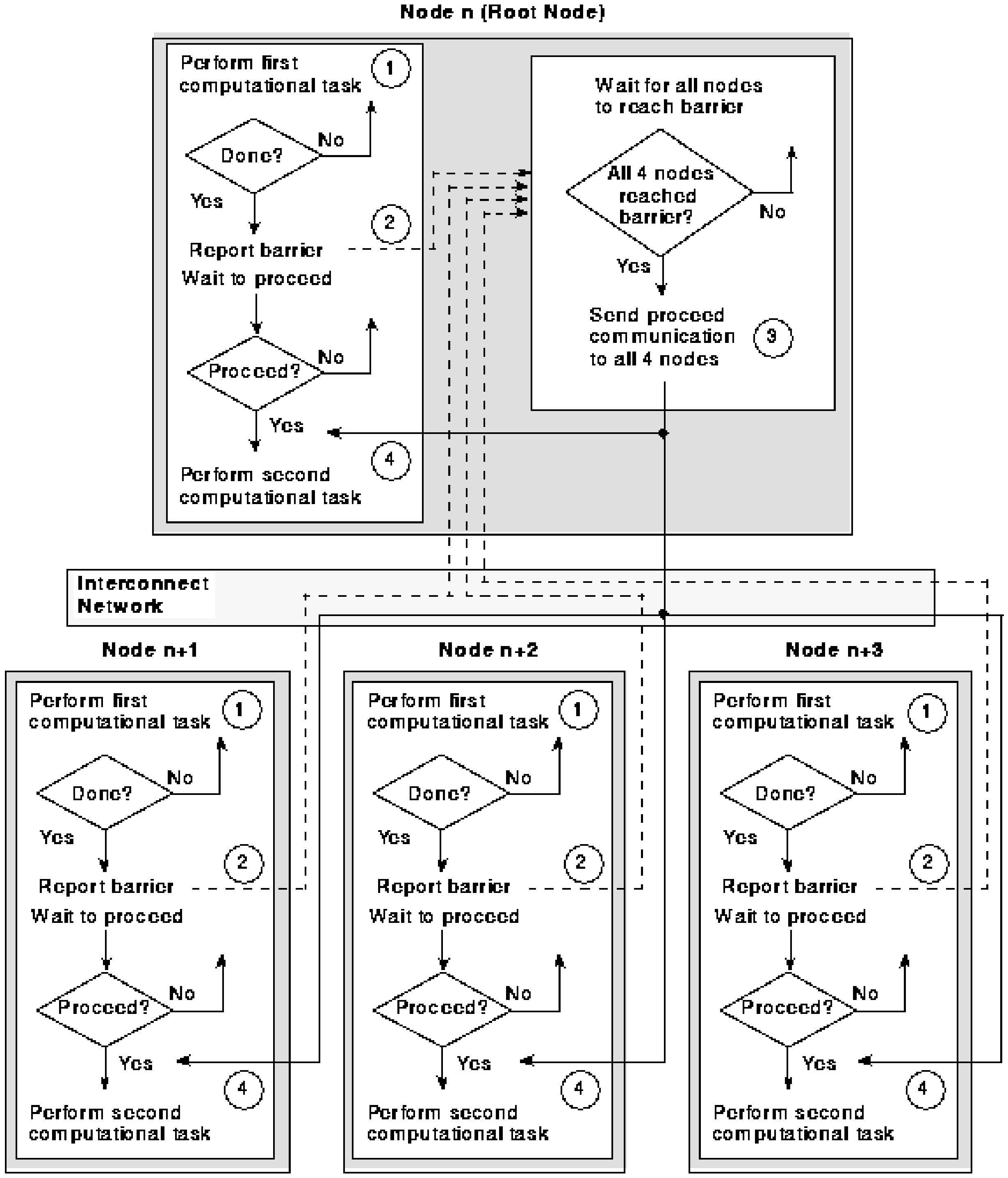
1. A 4-PE application includes several computational tasks. All four PEs must complete the first computational task before they can continue with the second task. The barrier resources are used to provide this synchronization. Software assigns node n as the root node for this 4-PE partition.

2. Each node completes the first computational task asynchronously and sends a barrier communication to inform the root node. Nodes n+1, n+2, and n+3 use the interconnect network to transmit this communication to the root node.

3. The root node receives barrier signals from all nodes in the partition (including itself) indicating that the barrier event completed. The root node initiates a communication back to all the nodes within the partition.

4. Each PE receives the communication from the root node and proceeds to the next computational task.

Figure 1. Example of Barrier Synchronization


Node n (Root Node)
Perform first computational task
1
Done?
No
Yes
2
Report barrier
Wait to proceed
Proceed?
No
Yes
4
Perform second computational task
Wait for all nodes to reach barrier
All 4 nodes reached barrier?
No
Yes
Send proceed communication to all 4 nodes
3
Interconnect Network
Node n+1
Node n+2
Node n+3

Eureka Synchronization

Typically, programmers use eureka synchronization for database searches. To do this, the programmers divide the search among the PEs in the partition. When a PE finds the requested data, that PE uses eureka synchronization to notify the other PEs in the partition to stop the search. For example, all of the PEs in a partition may be performing a distributed database search on different segments of data in a file. As soon as one PE finds the requested data, it informs the root node. The root node informs all other PEs that the search can stop.

The B/E circuit uses the interconnect network to signal the root node that a eureka event occurred. The root node initiates a communication to inform the other PEs in the partition.

If software wants to reuse this B/E circuit for eureka synchronization, software issues a barrier to ensure that the PEs are synchronized before they begin the next computational task.

The following text describes a eureka event followed by barrier synchronization for a 4-PE partition. The step numbers correspond to the numbers in Figure 2.

1. All four PEs in a partition search different segments of a database looking for a specific piece of data. Software assigns node n as the root node for this 4-PE partition.

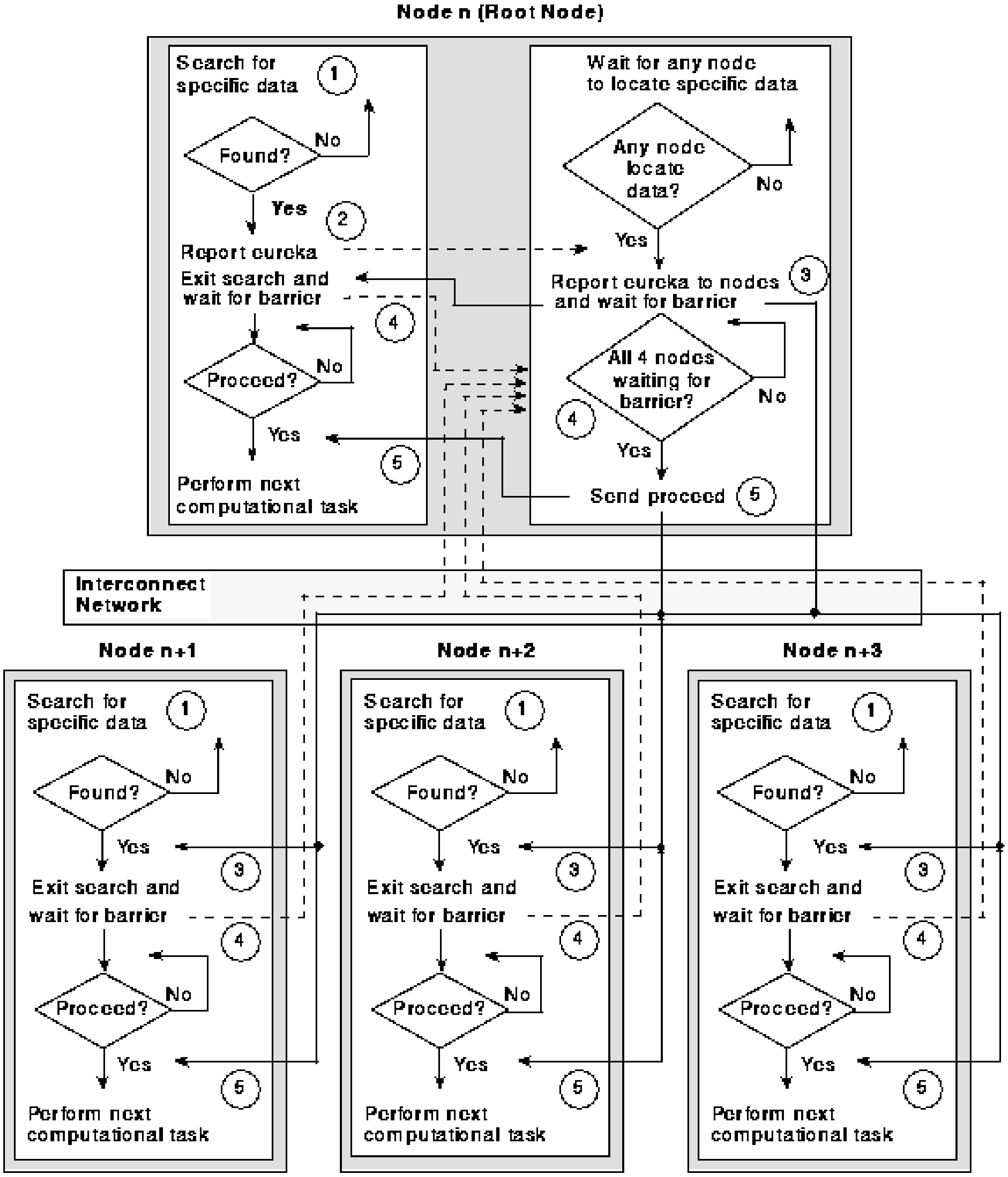
2. In this example, node n (the root node) locates the specified data while the other nodes continue to search. The root node notifies its B/E circuit that the data was found (eureka occurred).

3. The root node initiates a communication to inform all the PEs within the partition that the eureka occurred. This communication causes each PE to exit the search.

4. Each PE exits its search asynchronously and reports to the root node that they are waiting for a barrier. After the root node receives this barrier information from all nodes in the partition (including itself), it initiates a communication instructing all nodes to proceed with the next task.

5. All PEs in the partition are informed that the barrier completed (the partition is synchronized) and proceed to the next computational task.

Figure 2. Example of Eureka Synchronization


Node n (Root Node)
Search for specific data
1
Found?
No
Yes
2
Report eureka
Exit search and wait for barrier
4
Proceed?
No
Yes
5
Perform next computational task
Wait for any node to locate specific data
Any node locate data?
No
Yes
Report eureka to nodes and wait for barrier
3
All 4 nodes waiting for barrier?
No
4
Yes
Send proceed
5
Interconnect Network
Node n+1
Node n+2
Node n+3
Search for specific data
Found?
Yes
Exit search and wait for barrier
Proceed?
Perform next computational task

B/E Partition

To initialize a B/E partition, the software identifies a root node and establishes parent-to-child relationships between the nodes in the partition. When illustrated, a B/E partition resembles a tree-type structure (refer to Figure 3 and Figure 4). The parent-to-child relationships between the nodes identify the communication links used to transfer B/E control information between the root node and all other nodes in the partition. This communication occurs in either direction based on the type of control information that is transferred; some B/E control information transfers up the tree to the root node (fanin) while other control information transfers down the tree to each PE (fanout).

For example, node 000300 reaches a barrier and passes this control information to its parent node 000301. When node 000301 receives the B/E control information from node 000300 and has reached the barrier itself, node 000301 transfers the control information to the root node 000201. The root node also receives similar B/E control information from its other two children (nodes 000101 and 000200) to complete the fan-in process.

To complete a barrier sequence, the root node fans out B/E control information to its child nodes. These nodes fan out the B/E control information to their child nodes. This fan-out process continues until all of the nodes within the partition receive the B/E control information.

The nodes in a partition pass B/E control information to one another using the interconnect network. Packets that contain B/E control information receive high priority as they contend for resources on the interconnect network.

Figure 3. Parent-to-child Node Relationships

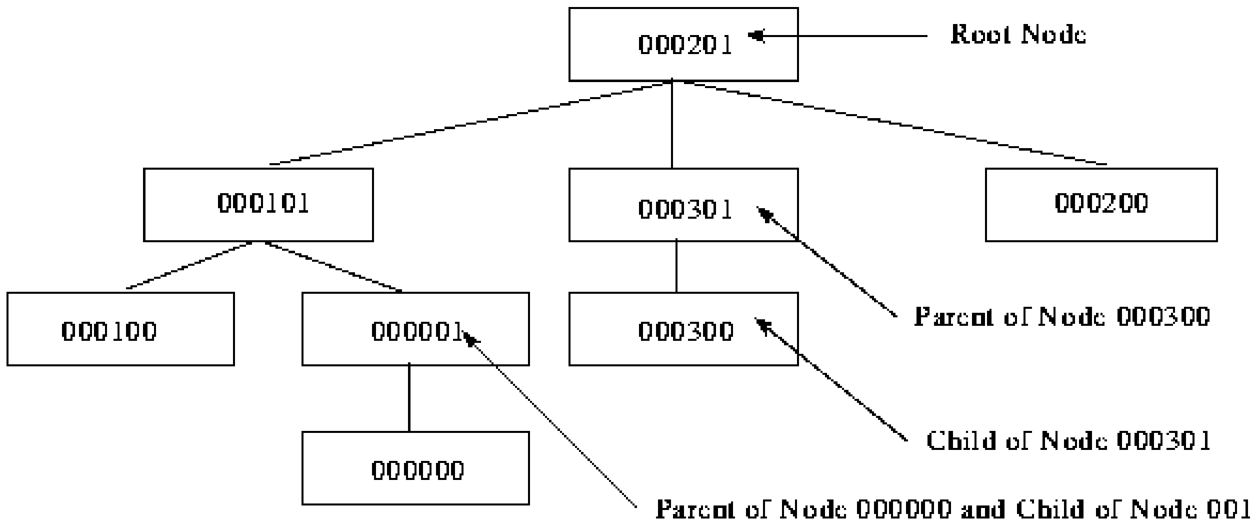
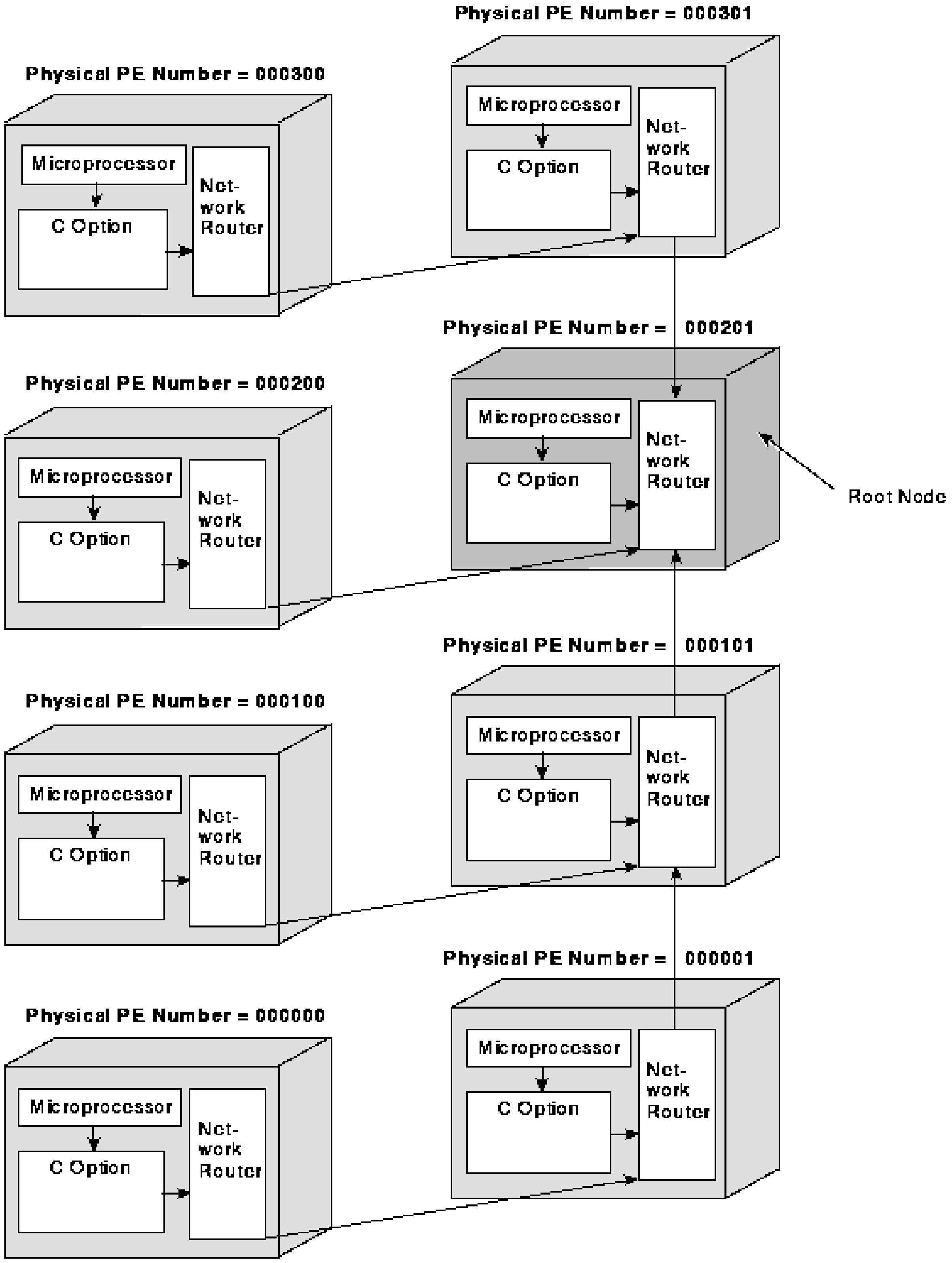


Figure 4. Example of a B/E Partition


Physical PE Number = 000301
Physical PE Number = 000300
Physical PE Number = 000201
Physical PE Number = 000200
Physical PE Number = 000101
Physical PE Number = 000100
Physical PE Number = 000001
Physical PE Number = 000000
Microprocessor
C Option
Net-work Router
Root Node

Barrier and Eureka Synchronization Logic

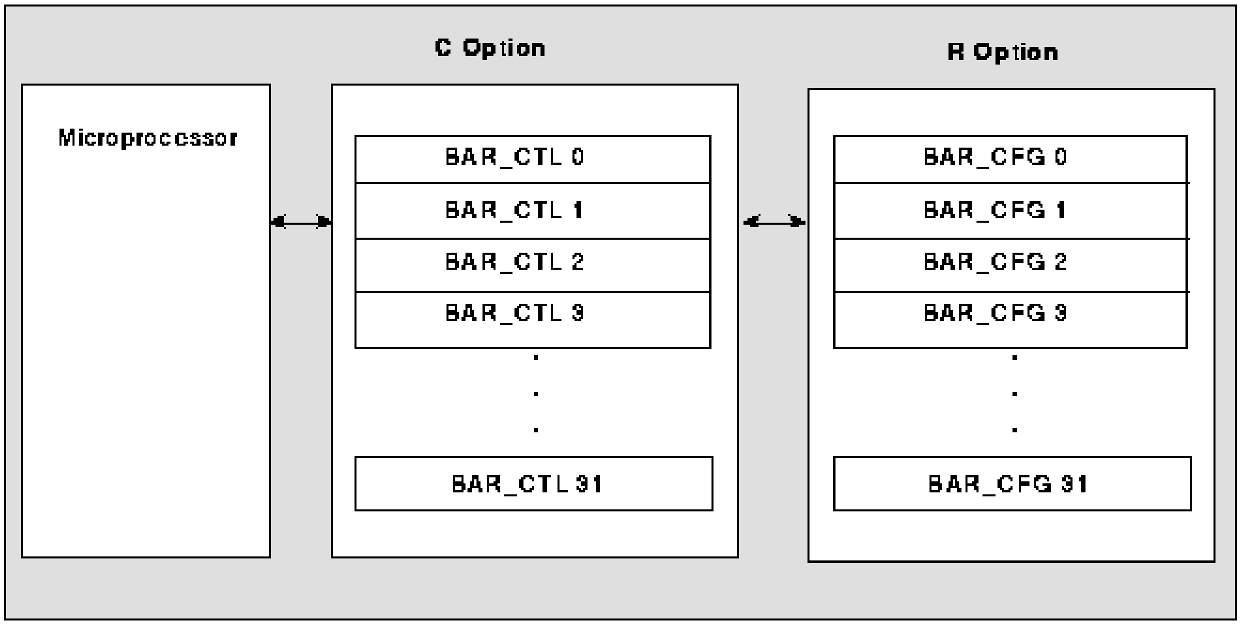
Each processing element node in the CRAY SSD-T90 device includes barrier and eureka synchronization logic. The barrier and eureka logic supports 32 separate barrier/eureka synchronization units (BESUs). Each BESU functions as an independent synchronization resource.

To define B/E partitions and track the current state of each partition, each of the 32 BESUs is supported by:

- A BAR_CFG register that defines the node's position in a B/E partition
- A BAR_CTL circuit that holds the current state of a BESU; the BAR_CTL circuit is an active circuit that is accessed through a memory-mapped function address

Figure 5 illustrates the B/E logic that comprises the 32 BESUs on each processing element node.

Figure 5. B/E Synchronization Logic



The BAR_CFG Register

To define a partition, the software assigns values to a specific BAR_CFG register on each node. Each of the 32 BESUs has an associated BAR_CFG register. The software uses the same BESU in each node of the partition to establish the parent-to-child relationships between the nodes in that partition.

Figure 6 shows the BAR_CFG register. This register contains the following bits:

- PE bit - When the PE bit is set to 1, it indicates that the PE is an active member of the partition. When the PE bit is set to 0, it indicates that its PE is not an active member of the partition.
- Child bits - When a child bit is set to 1, the child bit indicates that the neighboring node on the specified link is a child of this node. A node may have more than one child.
- Parent bits - The parent bits identify the parent node or that this node is the root node (refer to Table 1).

  NOTE: When the parent bits are set to 0100, the BAR_CFG register is not connected to a partition; therefore, the child bits and the PE bits are not used. Software sets the parent bits to 0100 during a reset.

Figure 7 shows the contents of the BAR_CFG register for each node in a barrier tree example.

When the network router of a parent node receives a packet that contains fan-in barrier information (BESU number and opcode), the network router references its BAR_CFG registers to determine whether the packet arrived on a communication link that is assigned to a child. The network router also references its BAR_CFG register to determine whether its local PE is part of the partition. When the PE bit is a 1, the local PE is considered an active child node. When the parent node receives the appropriate B/E opcode from all active child nodes, the parent node sends a packet that contains the B/E information to its parent node. The barrier control packet continues passing from child to parent until it arrives at the root node.

When the root node receives barrier information from its child nodes and itself (for a eureka, the root node receives the eureka information from one child node or itself), the root node fans out B/E information to the other nodes in the partition. To fan out the B/E information to the other nodes in the partition, the network router option of the root node places the B/E information in packets and sends the packet out onto the interconnect network to its child nodes.When a child node receives the packet and its PE bit is set to a 1, the child node uses the B/E opcode in the packet to change the state of the appropriate BESU. Each child node then sends the packet out onto the interconnect network to its child nodes. This fan-out process continues until all nodes within the partition receive the barrier information.

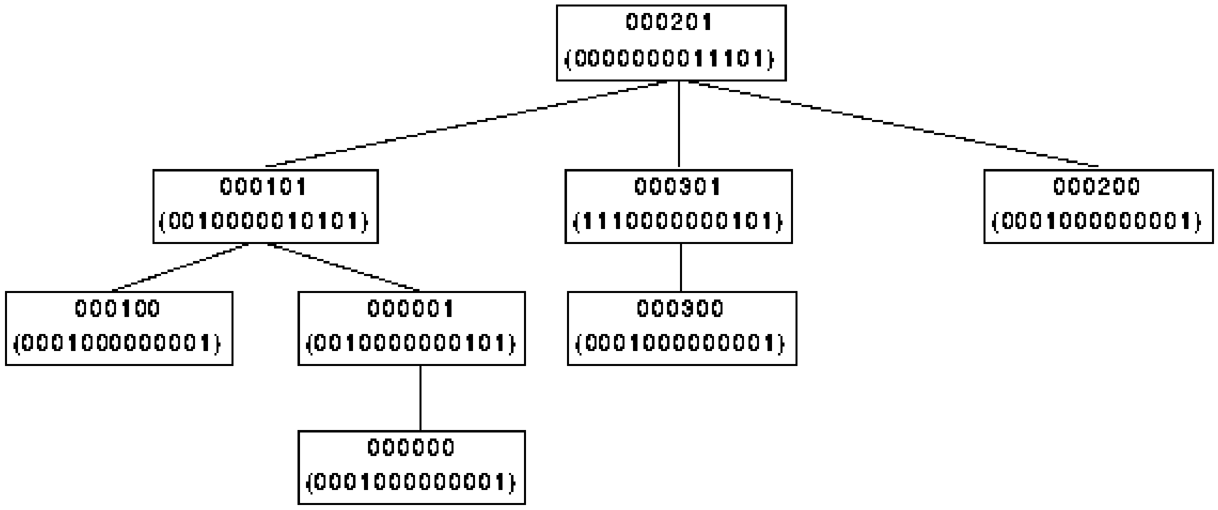
Figure 6. BAR_CFG Register

Parent Bits 0 through 3	00	-Z Child Bit	+Z Child Bit	-Y Child Bit	+Y Child Bit	-X Child Bit	+X Child Bit	PE Bit

Table 1. Description of Parent Bits

Parent Bits 3 through 0	Description
1101 (-3)	The node connected to -Z is the parent of this node.
1110 (-2)	The node connected to -Y is the parent of this node.
1111 (-1)	The node connected to -X is the parent of this node.
0000 (0)	This node is the root node.
0001 (1)	The node connected to +X is the parent of this node.
0010 (2)	The node connected to +Y is the parent of this node.
0011 (3)	The node connected to +Z is the parent of this node.
0100 (4)	The BESU is not connected to a partition.

Figure 7. Example of BAR_CFG Register Values



The BAR_CTL Circuit

The BAR_CTL circuit contains a 3-bit state code that defines the current state of the BESU. Table 2 describes these state codes.

Table 2. B/E Synchronization Unit State Codes

State Code	Name	State Description
000	S_IDLE	Idle
001	S_IDLE_I	The microprocessor will be interrupted when a eureka event occurs
010	S_EUR	A eureka event occurred (no interrupt generated)
011	S_EUR_I	A eureka event occurred and an interrupt was generated
100	S_ARM	The barrier circuit is armed
101	S_ARM_I	The barrier circuit is armed and the microprocessor will be interrupted when the barrier completes
110	S_BAR	A barrier event completed (no interrupt generated)
111	S_BAR_I	A barrier event completed and an interrupt was generated

The state of a BESU changes when a BESU receives B/E information from one of the following two sources:

- Local microprocessor (change initiated by local program code)
- Interconnect network (change initiated by a remote PE node)

The local microprocessor writes a a 3-bit control code to one of the 32 BAR_CTL circuits to change the state of a BESU. Table 3 describes the B/E control codes.

Table 3. B/E Control Codes

Control Code	Name	Description
000	OP_CLEAR	Clear
001	Reserved	Reserved
010	OP_EUR	Send eureka
011	OP_INT	Arm eureka interrupt
100	OP_BAR	Wait for barrier event
101	OP_BAR_I	Arm barrier interrupt
110	OP_EUR_B	Send eureka and wait for barrier event
111	OP_RESET	Reset

When the local microprocessor changes the state of a BESU, the C option may translate the 3-bit B/E control code into a 2-bit B/E opcode and send this opcode to the network router. When the network router receives this opcode from all active child nodes, the network router places this opcode and the BESU number into a packet for transmission to its parent node. Table 4 shows how the B/E control codes translate to the B/E opcodes.

Table 4. Relationship Between B/E Control Code and B/E Opcode

B/E Control Code	B/E Opcode
000 - Clear	00 - Normal idle
001 - Reserved	00 - Normal idle
010 - Send eureka	01 - Issue eureka
011 - Arm eureka interrupt	00 - Normal idle
100 - Wait for barrier event	10 - Join barrier
101 - Arm barrier interrupt	10 - Join barrier
110 - Send eureka and wait for barrier event	11 - Eureka/barrier
111 - Reset	00 - Normal idle

All B/E communication between the C option and the R option uses the 2-bit B/E opcodes. When the local microprocessor changes the state of a BESU, the resulting B/E opcode travels from the C option to the R option; when B/E information arrives from the interconnect network, the B/E opcode travels from the R option to the C option. The definition of these 2-bit opcodes is somewhat dependent on the source of this communication (refer to Table 5).

Table 5. B/E Opcode

Opcode	C option to R option	R option to C option
00	Normal idle	Normal idle

01	Issue eureka	Eureka received
10	Join barrier	Barrier completed
11	Eureka/barrier	Barrier/eureka

Figure 8 shows a B/E control code issued from the local microprocessor. In this example, the BAR_CTL[1] circuit changes from an idle state (000) to a barrier armed state (100). This occurs when the microprocessor writes a 100 to the BAR_CTL[1] circuit. As a result, the C option sends an opcode of 10 (join barrier) to the R option for transmission to the node's parent.

Figure 8. Local Microprocessor Changing the State of a BESU

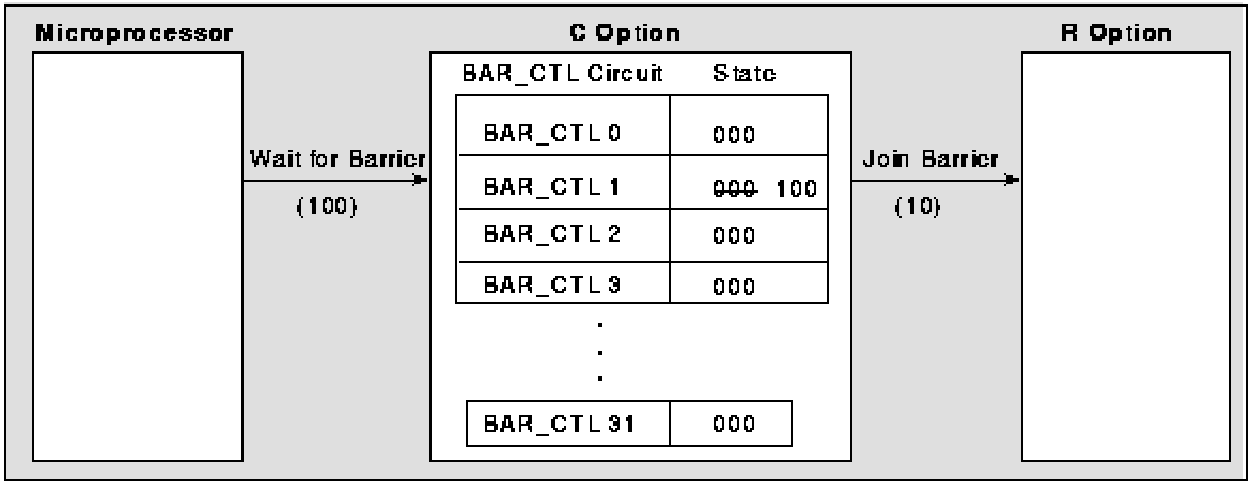
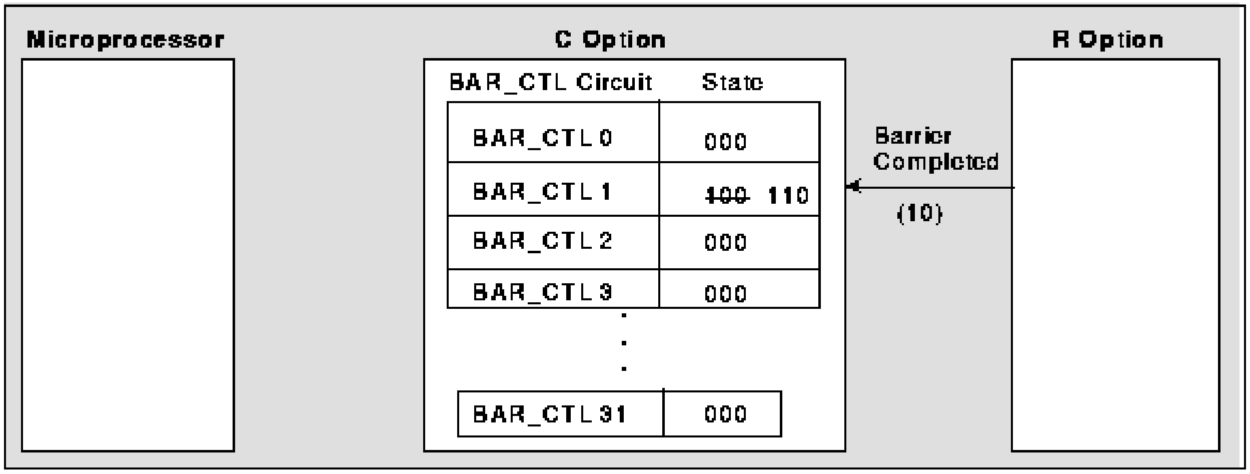


Figure 9 shows B/E control information arriving from the interconnect network through the R option. In this example, the R option sends an opcode of 10 (barrier completed) to the C option. The C option uses this code to change the state of the BAR_CTL[1] circuit from a barrier armed state (100) to a barrier completed state (110).

Figure 9. Network Information Changing the State of a BESU



Examples of B/E Synchronization Operations

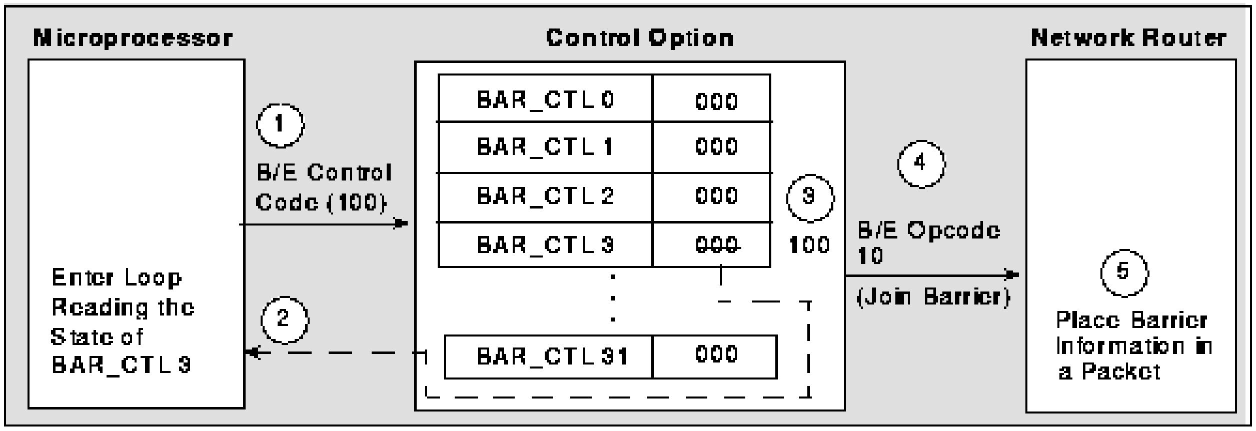
During a barrier or eureka synchronization operation, an application determines when a B/E operation completes by using a polled method or an interrupt method.

Polled Method

When software issues a barrier synchronization operation using the polled method, the BESU performs the barrier operation but does not interrupt the microprocessor when the operation is complete. Instead, the microprocessor must periodically read the state code. When the state code indicates that the barrier is complete, the microprocessor continues with the next task. The following text describes this method of barrier synchronization using BESU 3. The step numbers correspond to the numbers in Figure 10 through Figure 12.

1. When a process that is running in the microprocessor reaches a barrier, the process writes a B/E control code of 100 (arm barrier) to BAR_CTL 3.
2. The software enters a loop that periodically reads the state of BAR_CTL 3.
3. After receiving the B/E control code, BAR_CTL 3 changes to a 100 (arm barrier).
4. The control option sends the B/E opcode 10 (join barrier) to the network router option.
5. After receiving the barrier information from its active child nodes and itself, the network router option places the barrier information into a packet.

Figure 10. Initiating a Barrier Operation Using the Polled Method



6. The network router option then sends the packet out onto the interconnect network to its parent node. When the parent node receives packets that contain barrier information from all of its child nodes, it passes the barrier information to its parent node. This fan-in process continues until the barrier information reaches the root node.
7. After the root node receives the barrier information from all of its child nodes, the network router option in the root node sends the packet out onto the interconnect network in the dimensions that connect to its child nodes. The child nodes fan out the packet to all of their child nodes. This fan-out process continues until all of nodes within the partition receive the packet.

8. When the network router option within each node of the partition receives the packet, the network router option signals the control option with an opcode of 10 (barrier completed). The control option changes the state from 100 to 110 (barrier complete state).

9. When the microprocessor reads this new state, the microprocessor exits the loop and continues with the next task.

Figure 11. Fanin to the Root Node

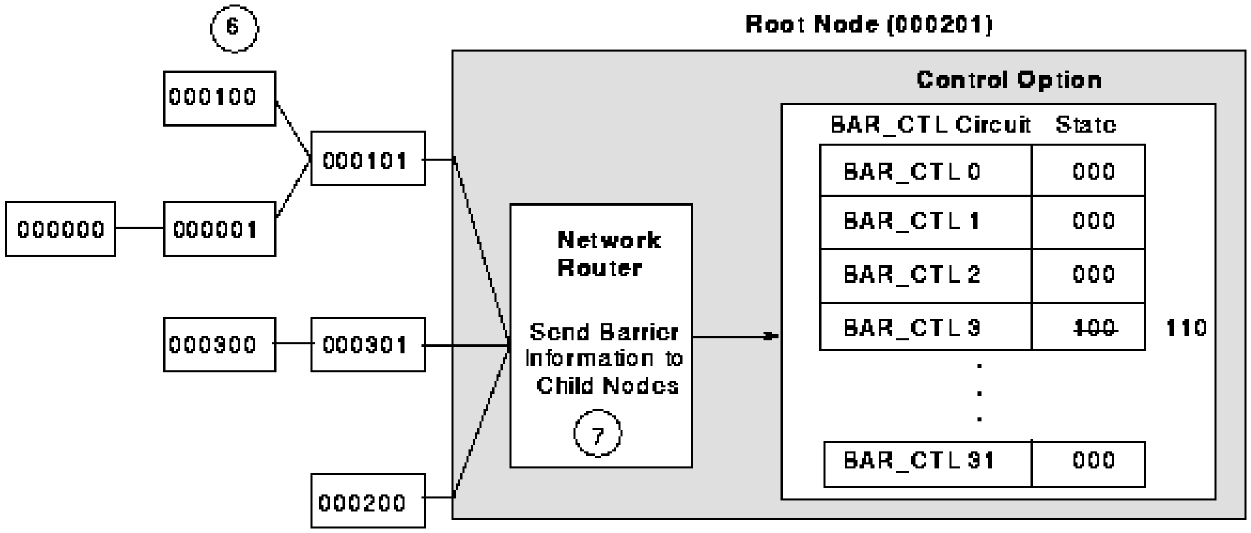
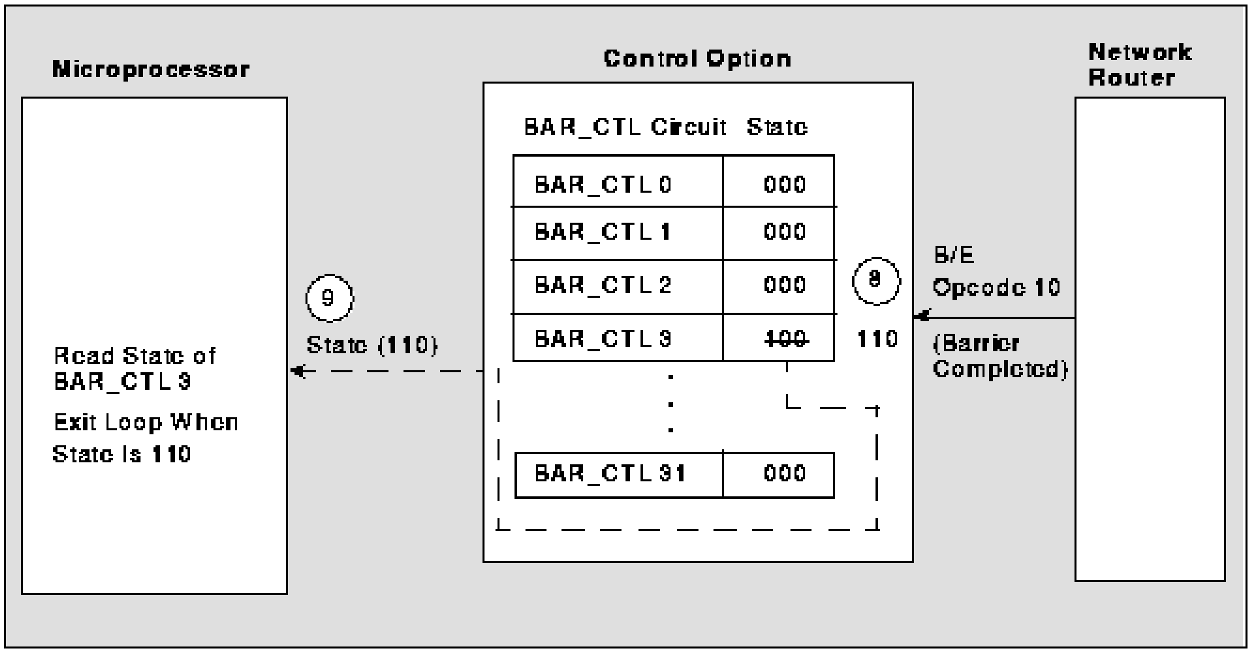


Figure 12. Using the Polled Method to Determine that the Barrier Operation is Complete



Interrupt Method

Software can use the interrupt method for both barrier synchronization and eureka synchronization.

Barrier Synchronization Using the Interrupt Method

For a barrier synchronization operation using the interrupt method, the BESU performs the barrier operation and interrupts the microprocessor when the operation is complete. The following text describes this method of barrier synchronization for BESU 3. The step numbers correspond to the numbers in Figure 13 and Figure 14.

1. When a process that is running in the microprocessor reaches a barrier, the process enables the B/E interrupt for BESU 3.

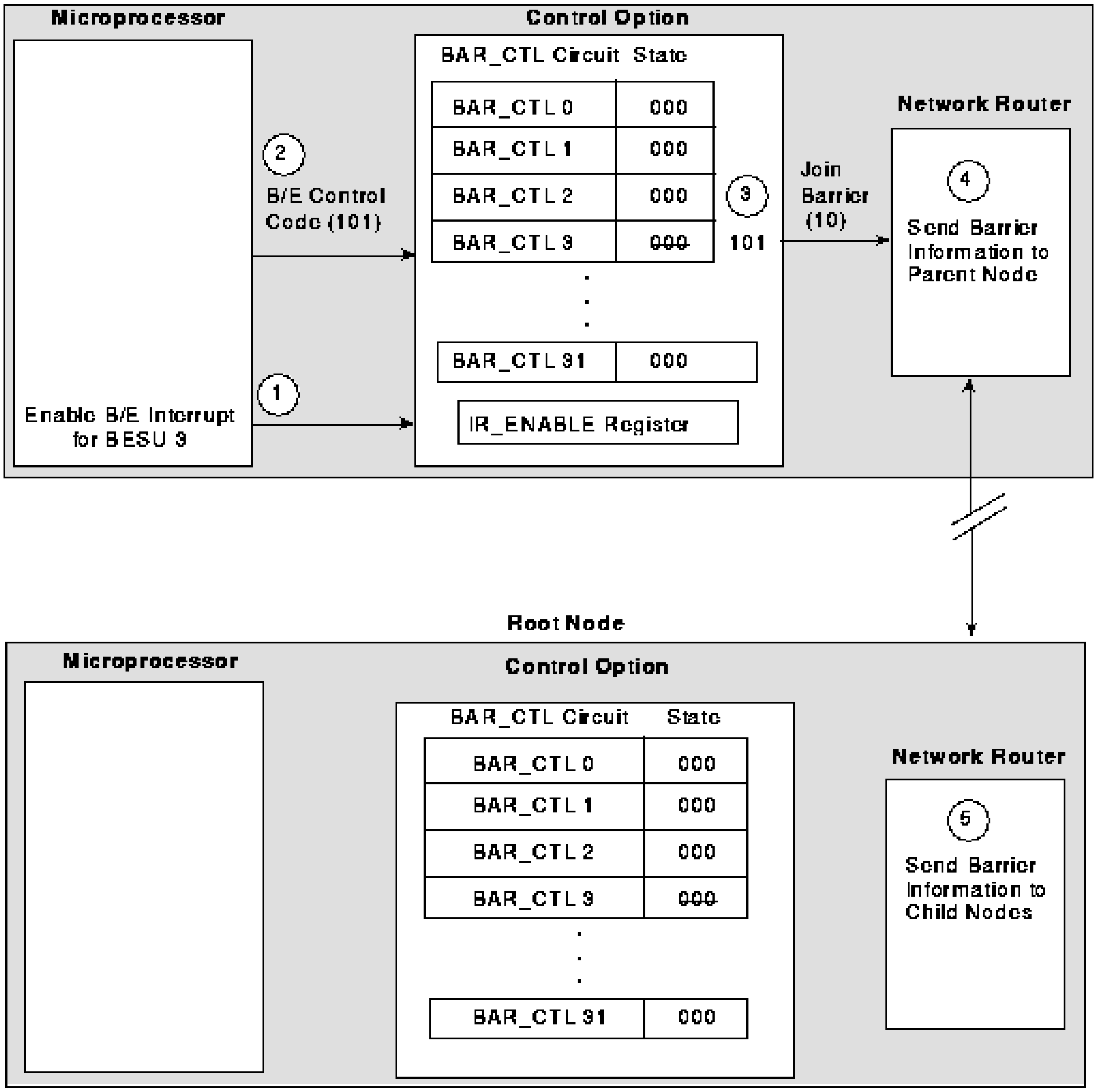
2. The microprocessor writes a B/E control code of 101 (arm barrier interrupt) to BAR_CTL 3. Once this is done, the software continues running other program instructions.

3. After receiving the B/E control code, BAR_CTL 3 changes to a state of 101 and sends a B/E opcode of 10 (join barrier) to the network router option.

4. The network router option sends the barrier information out onto the interconnect network as described in the polled method.

5. After the root node receives the barrier information from all of its child nodes, the network router option in the root node sends the packet out onto the interconnect network in the dimensions that connect to its child nodes. The child nodes fan out the packet to all of their child nodes. This fan-out process continues until all of the nodes within the partition receive the packet.

Figure 13. Initiating a Barrier Synchronization Operation Using the Interrupt Method



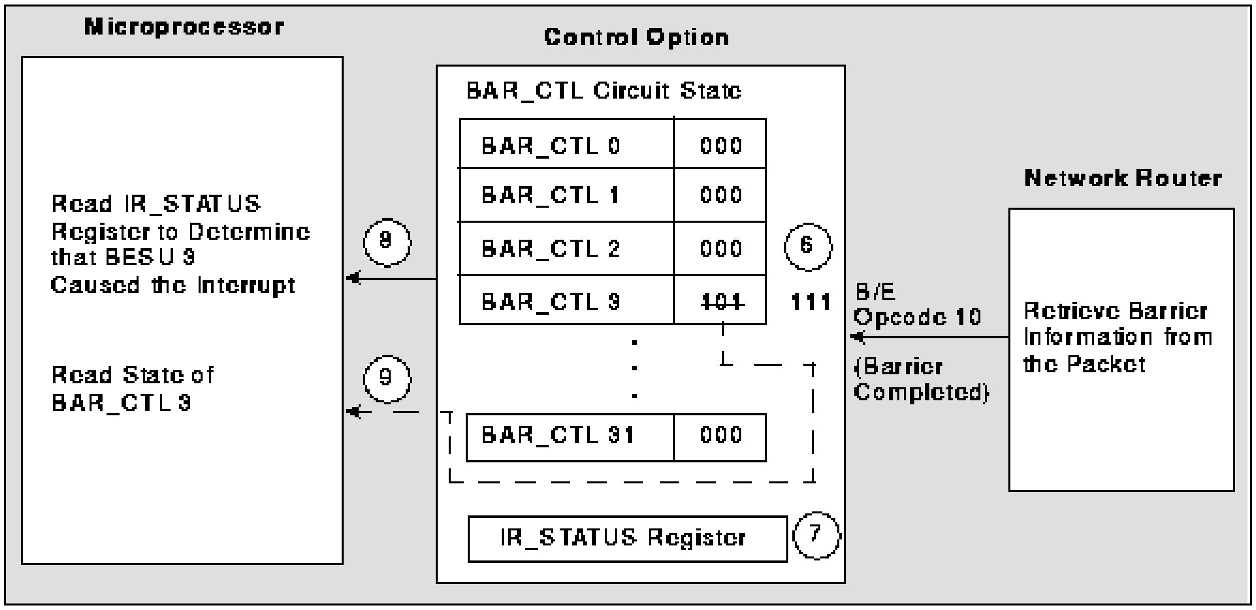
6. When the network router option within each node of the partition receives the packet, the network router option signals the control option with an opcode of 10 (barrier completed). The state changes from 101 to 111 (barrier completed with interrupt state).

7. BAR_CTL 3 also sets the appropriate barrier interrupt bit in the IR_STATUS register.

8. The support circuitry interrupts the microprocessor, which causes the process that is running in the microprocessor to read the IR_STATUS register. The contents of the IR_STATUS register indicate that BESU 3 caused the interrupt.

9. Next, the process reads the state for BAR_CTL 3. At this point, the state for BAR_CTL 3 is 111, which indicates that the barrier is complete for BESU 3.

Figure 14. Interrupting the Microprocessor to Indicate that the Barrier Operation is Complete



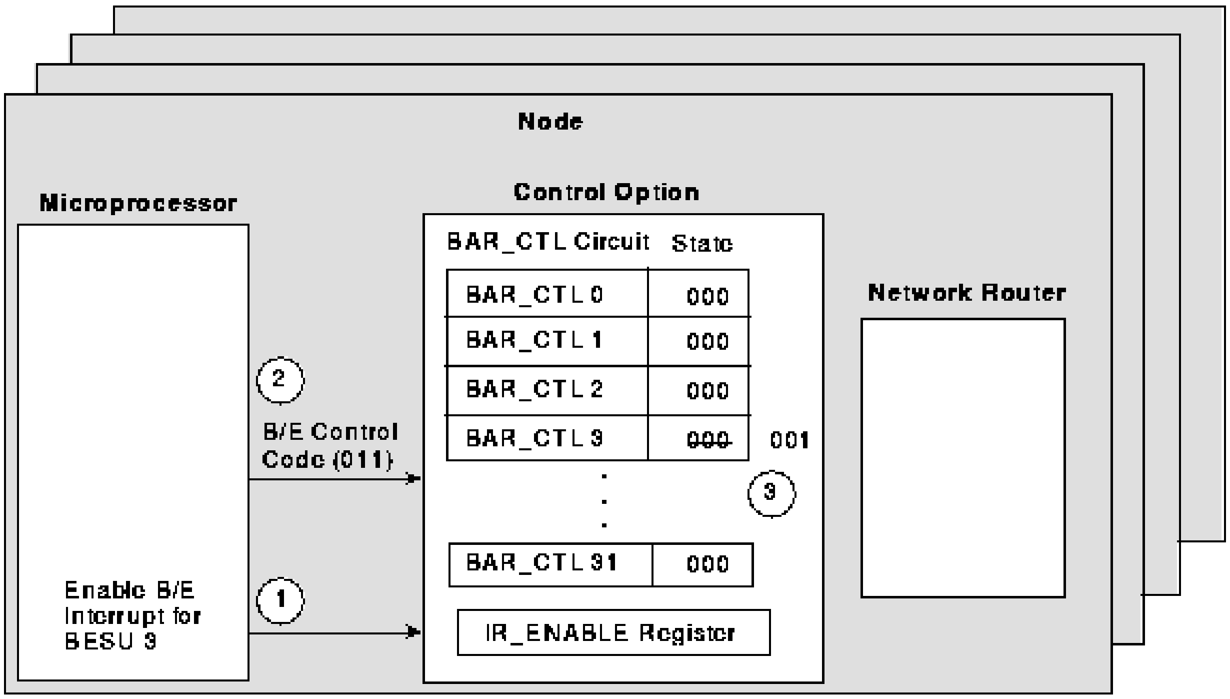
Eureka Synchronization Using the Interrupt Method

When software issues a eureka synchronization operation using the interrupt method, the BESU performs a eureka operation, which is normally followed by a barrier operation. For example, during the eureka operation all of the PEs within the partition work on the same task. When one of the PEs completes the task, all of the PEs exit the program. The barrier operation ensures that all of the PEs are informed of the eureka event before software reuses the BESU for another eureka operation.

For eureka synchronization, the BESU interrupts the microprocessor after the eureka operation is complete and after the barrier operation is complete. The following text describes this method of eureka synchronization for BESU 3. The step numbers correspond to the numbers in Figure 15 through Figure 17.

1. Before a process that is running in the microprocessor initiates a task, the process enables the B/E interrupt for BESU 3.
2. The microprocessor also writes a B/E control code of 011(arm eureka interrupt) to BAR_CTL 3. Once this is done, the software continues running program instructions that relate to the task.
3. After receiving the B/E control code, the state changes to 001.

Figure 15. Initiating the Eureka Synchronization Operation by Using the Interrupt Method



4. When one of the microprocessors completes the task, that microprocessor writes a B/E control code of 010 (send eureka) to BAR_CTL 3.

5. After receiving the B/E control code, BAR_CTL 3 changes state to 011.

6. BAR_CTL 3 also sends a B/E opcode of 01 (issue eureka) to the network router option; this network router option passes the information to the root node.

7. After the root node receives a packet that indicates a eureka was issued, the network router sends the packet out onto the interconnect network in the dimensions that connect to its child nodes. The child nodes fan out the packet to all of their child nodes. This fan-out process continues until all of the nodes within the partition receive the packet.

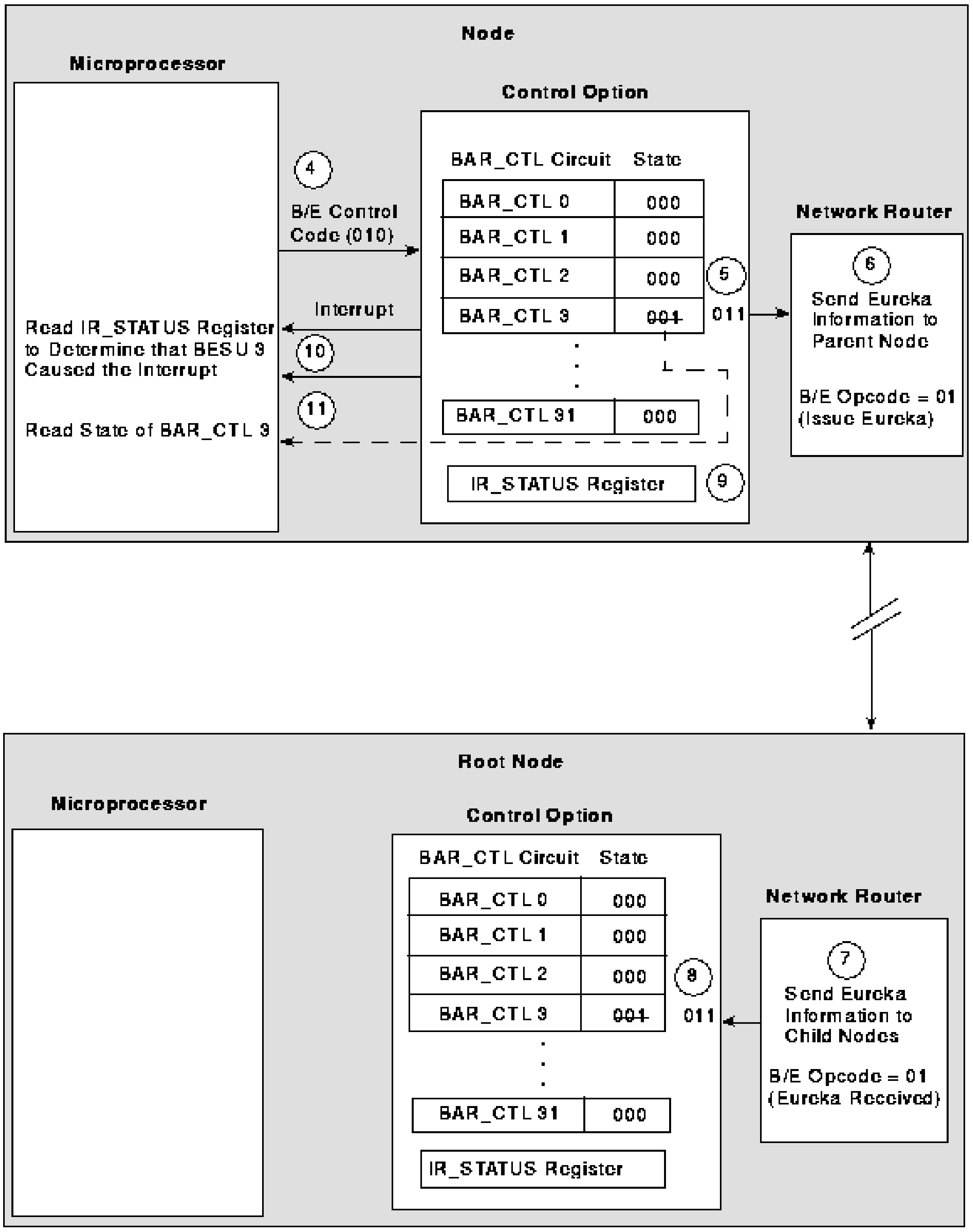
8. After receiving the B/E opcode of 01 (eureka received), BAR_CTL 3 of all nodes within the partition changes from 001 to 011 (eureka occurred with interrupt state).

9. BAR_CTL 3 also sets the appropriate barrier interrupt bit in the IR_STATUS register and interrupts the microprocessor.

10. When the microprocessor is interrupted, the software that is running in the microprocessor reads the IR_STATUS register and determines that BESU 3 caused the interrupt.

11. Next, the software reads the state for BAR_CTL 3. At this point, the state for BAR_CTL 3 is 011, which indicates that the eureka operation is complete.

Figure 16. Completing the Eureka Operation and Notifying the Root Node



12. Next, software enables the B/E interrupt for BESU 3.

13. The microprocessor writes a B/E control code of 101 (arm barrier interrupt) to BAR_CTL 3. Once this is done, the software continues to run other program instructions.

14. After receiving the B/E control code, BAR_CTL 3 changes to a state of 101 and sends a B/E opcode of 10 (join barrier) to the network router option.

15. The network router option sends the barrier information out onto the interconnect network as described in the polled method.

16. After the root node receives the barrier information from all of its child nodes, the network router option in the root node sends the packet out onto the interconnect network in the dimensions that connect to its child nodes. The child nodes fan out the packet to all of their child nodes. This fan-out process continues until all of the nodes within the partition receive the packet.

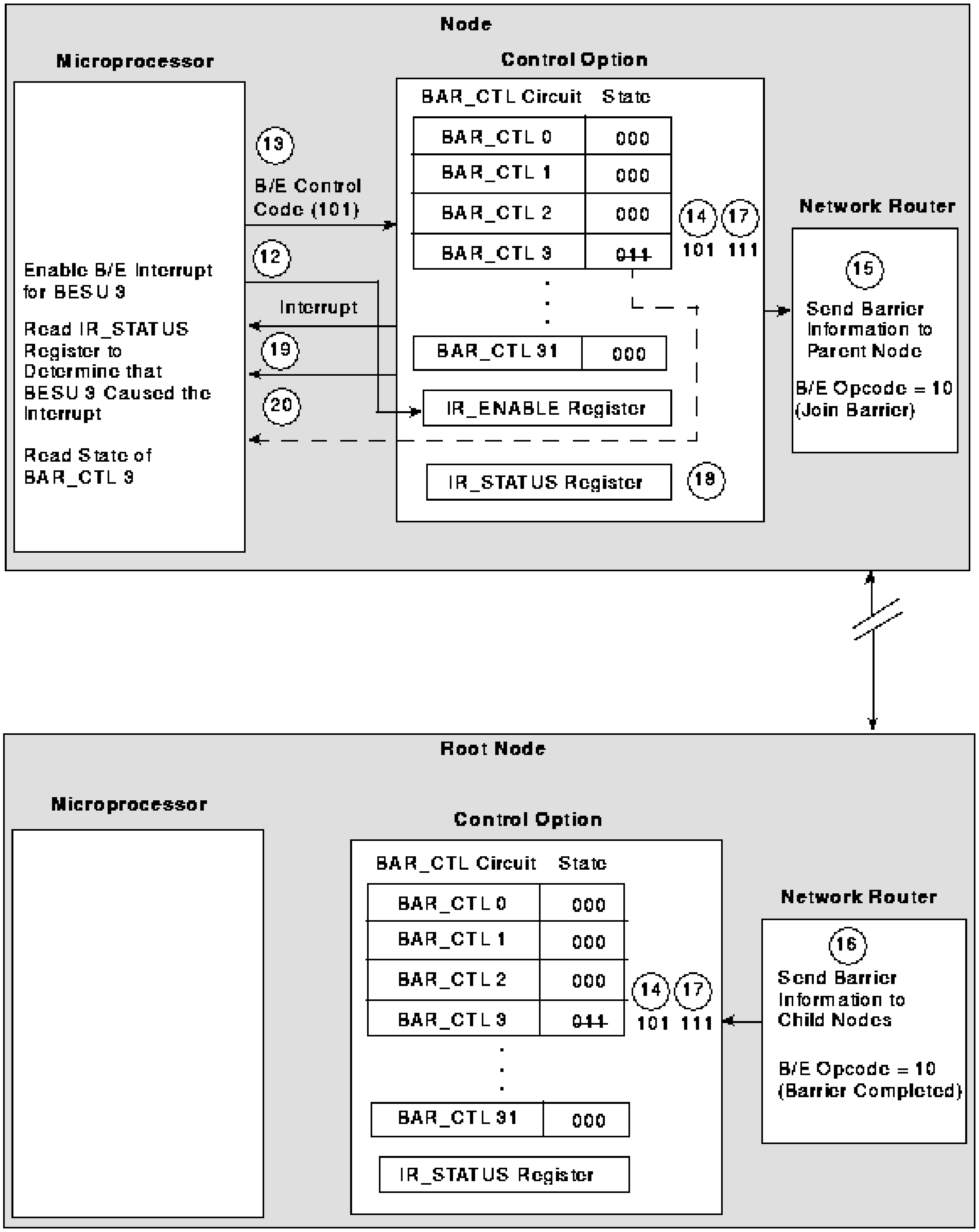
17. When the network router option within each node of the partition receives the packet, the network router option signals the control option with an opcode of 10 (barrier completed). The state changes from 101 to 111 (barrier completed with interrupt state).

18. BAR_CTL 3 also sets the appropriate barrier interrupt bit in the IR_STATUS register.

19. The support circuitry interrupts the microprocessor, which causes the process that is running in the microprocessor to read the IR_STATUS register. The contents of the IR_STATUS register indicate that BESU 3 caused the interrupt.

20. Next, the process reads the state for BAR_CTL 3. At this point, the state for BAR_CTL 3 is 111, which indicates that the barrier is complete for BESU 3.

Figure 17. Using the Barrier Operation to Synchronize the PEs after a Eureka Operation



B/E Synchronization Errors

When an invalid barrier error occurs, the C option logs the B/E synchronization error in bit 62 of the C_ERR0 register. The C option also logs an error type of 23 in the C_ERR3 register when a program tries to access an undefined barrier space or when it attempts to perform a privileged barrier operation.

Offline Diagnostics

The offline diagnostic test, `barr`, verifies that the microprocessor can write to and read from the BESUs and the BAR_CFG register. This test also verifies that when a barrier partition is created, the barrier and eureka signals propagate correctly and the BESUs can change state.