Microprogram Level Flow: ADDW3

Now that all of the hardware components and control signals of the data path module have been described in detail, this section takes an ADDW3 (add word 3 operand) macroinstruction and describes the decoding and execution of this instruction at the microprogram level.

An ADDW3 macroinstruction adds the word (16 bits) at the address specified by the first operand specifier to the word at the address specified by the second operand specifier, and stores the sum in the location specified by the third operand specifier. A sample ADDW3 instruction is:

```
ADDW3  B↑5(R0)[R1], @(R5)+, R2
```

This instruction uses several addressing modes. The first operand specifier uses byte displacement indexed addressing; the second operand specifier uses autoincrement deferred, and the third operand specifier uses register mode.

At some virtual address in memory, this instruction looks like this:

52	95	05	A0	41	A1	:VA

where A1 is the opcode, 41 specifies index mode using R1, 05A0 is byte displacement mode using R0 (05 is the displacement), 95 is autoincrement deferred mode using R5, and 52 is register mode using R2. The 05A0 specifies the base address of a table of words, and the content of R1 is an index into this table. Before the microprogram level description begins, the next few paragraphs summarize the steps needed to execute this ADDW3 instruction.

Step 1. Evaluate the opcode to select the proper microroutine for this macroinstruction.

Step 2. Evaluate the first operand specifier and obtain the first operand. This is accomplished as follows:

a. Add 5 to the contents of R0; the sum is called the base operand address.

b. Multiply the contents of R1 by 2.

c. Add the result from step b to the base operand address from step a to get the address of the first operand.

d. Use the address computed in step c to fetch the first operand and store it in a register on the data path chip.

Step 3. Evaluate the second operand specifier, obtain the second operand, and increment the contents of R5. This is accomplished as follows:

a. R5 contains a longword address; fetch the data at this address.

b. The fetched data are the address of the second operand; fetch the second operand and store it in a register on the data path chip.

c. Add 4 to the contents of R5.

Step 4. Add the first operand to the second operand, and store the sum in R2.

The remainder of this chapter describes the microprogram steps necessary to decode and execute the ADDW3 macroinstruction. Assume that the six bytes of the instruction are already in the prefetch logic, that all physical addresses are in the translation buffer, and that all requested data are located in the memory controller cache. Figure 4-12 at the end of this chapter diagrams all of the microinstructions.

Evaluating the Opcode: Decode A1

The ADDW3 opcode, A1, is in the IBYTE register because a decode was already done on the previous byte in the IBYTE register and the signal LOAD I BYTE H asserted. The last microinstruction of the microroutine that decoded and executed the previous byte in the IBYTE register is the Decode microinstruction for a macroinstruction opcode. Whenever a Decode microinstruction is executed, the data path chip increments the program counter. So, the PC on the data path chip contains the virtual address of the first byte (A1) of the ADDW3 instruction, the microprogram counter (μPC) contains the microaddress of the Decode microinstruction that decodes macroinstruction opcodes (IRD), and the IBYTE register contains A1.

The contents of μPC are driven onto the NμA bus, selected by the NμA MUX and latched into the CSA register at T2 (125 ns into the microcycle). Control store is accessed with this microaddress and the IRD microinstruction is the output. (IRD means the Decode microinstruction that decodes macroinstruction opcodes.) The microinstruction bits are sent all over the data path: bits <36:16> are sent to the data path chip as the DPC microinstruction; bits <24:23> (the IFUNC field, see Table 3-4) are sent to the decode ROMs; bits <15:08> and <24> are sent to the OR MUX control logic; bits <36:32> (the microopcode) and bit <24> are sent to the ID bus address decode logic; bits <28:24> are sent to the IBYTE control logic.

Bits <24:23> are available at the input to the decode ROMs 20 ns before the next clock edge (T0). The IBYTE control logic detects that an IRD is executing because DAPA CS 24 H is asserted; that is, bit <24> of the current microinstruction is set. The bits in the IBYTE register are accessing the decode ROMs as soon as the new byte is loaded into the IBYTE register, but the decode ROMs are not enabled until the rising edge of DLYD CPU CLOCK (62.5 ns into the current microcycle). The decode ROMs are enabled by the signal DAPC EN ROMS L, which is generated from the OR MUX control logic. Bits <15:08> and <24> as inputs to the OR MUX control logic PAL determine that an IRD microinstruction is executing and assert DAPC EN ROMS L as the output.

When the decode ROMs are enabled, the contents of the location being accessed by the byte in the IBYTE register are driven onto the NμA bus. The contents are 12 bits of microaddress (<11:0>) because this is an opcode decode. The NμA MUX selects these 12 bits and forces a zero as the high-order bit, bit <13> (see Table 4-1). These 13 bits are the microaddress of the first microinstruction in the microroutine for ADDW3. This microaddress is latched into the CSA register at T2 (125 ns).

In addition to 12 bits of microaddress, the output from the decode ROMs includes 2 bits of condition code class, and 2 bits of data type. The condition code class bits are sent to the condition code class register; the data type bits are sent to the size register. The data type bits from the decode ROMs for this IRD are 01 to indicate word. This value is loaded into the size register at the next T0. Thus, the size register contains a value of 1, specifying word.

Meanwhile, the signal DAPR LOAD I BYTE H is asserted by the IBYTE control logic because a Decode microinstruction has just been executed. At the next rising edge of CPU CLOCK, (the next T0), DAPR CLOCK I BYTE H is generated from DAPR LOAD I BYTE H and the next byte in the instruction stream is clocked into the IBYTE register from the memory control bus. The PC on the data path chip is incremented by one because a Decode microinstruction was just executed.

At the next rising edge of DLYD CPU CLK (T1), the signal DAPR IB TAKEN L is generated from LOAD I BYTE H. This signal when asserted informs the memory controller that the next instruction stream byte is needed, so the memory controller drives the third byte of the ADDW3 instruction (A0) from the prefetch logic onto the memory control bus.

At this point, the PC on the data path chip contains the virtual address of the second byte (41) of ADDW3, the IBYTE register contains the second byte, 41, and the microaddress of the first microinstruction in the ADDW3 microroutine is latched in the CSA register.

Step 1 is now complete; the macroinstruction opcode has been evaluated and the proper microroutine selected.

Evaluating the First Operand Specifier

Decode 41

The contents of the CSA register select a microinstruction in control store; the microinstruction selected is the first microinstruction in the microroutine for ADDW3. This microinstruction is an operand specifier Decode. The Decode microinstruction bits are distributed to the proper data path elements. Bits <24:23> of this microinstruction (the IFUNC field) have the value 0, indicating an operand specifier decode type 1.

The IFUNC field and the contents of the IBYTE register (41) are used to access the decode ROMs. Since this is an operand specifier decode, the output from the ROMs to the NμA bus is the low eight bits of the microaddress. The high five bits are driven onto the NμA bus from the jump register. The NμA MUX selects the bits on the NμA bus and latches them into the CSA register.

The size register is loaded at T0 from the CC/DT field of the microinstruction when operand specifier Decodes are executed, unless the CC/DT field contains the encoding 2 to specify the size register. Bits <38:37> of this Decode do contain the value 2, so the size register is unaffected; that is, the size register still contains the value 01, specifying word.

Any time an operand specifier decode is executed, bits <5:0> of the IBYTE register are passed through the IBYTE buffer, driven onto the ID bus, then to the data bus, and into one of the two pointer registers on the data path chip. Bit <26> (the pointer register select bit) of the Decode microinstruction just decoded is zero, so bits <5:0> of the IBYTE register (=000001) are saved in pointer 1 on the data path chip. Thus, pointer 1 is pointing to R1. Assume that R1 contains the value 3; that is, the contents of R1 will select the third entry in the table of words defined by the base address.

Another result of this operand specifier decode is that the current microaddress plus 1 is pushed on the microstack. This happens for every operand specifier decode when the addressing mode is not register mode, and the content of the IBYTE register is valid.

The PC on the data path chip is incremented by one because a Decode was just executed.

Shift by 2

Next, the 13 microaddress bits latched in the CSA register select a Shift microinstruction from control store. Bits <36:16> of this Shift microinstruction are latched in the control store register on the data path chip. The CC/DT field of this Shift is 00, so data path chip pins SIZE1 and SIZE0 are both zero. This encoding means that the chip operation (shift) uses data type long. (The data path chip size pins are determined by the CC/DT field of the current microinstruction when the microinstruction is not a Memory Request or an I-stream Request and the CC/DT field does not contain the value 3.)

This Shift microinstruction causes the contents of R1 to be shifted left by two bits, and stores the result in the RESULT2 register. A left shift by two effectively multiplies the contents of R1 by 4. This Shift is executed in case the table to be indexed is a table of longwords. But the next address control field of this Shift microinstruction uses the CASE format; this Shift microinstruction cases on the contents of the size register. The result is that the next microaddress is the address of another Shift microinstruction, but one that multiplies by 2 instead of by 4.

To further explain how this happens, assume that the first Shift microinstruction is located at control store address 1903, and that the next address control field (bits <15:0>) of this Shift microinstruction is 7C18, or 0111/1100/0001/1000. Bits <15:13> have the value 011, which specifies the CASE format (see Figure 3-3). Bits <9:8> are defined as the jump control field (JC<1:0>); the value of 0 in this field specifies that the output of the OR MUX is to be ORed with the low four bits on the NµA bus to obtain the next microaddress. Bits <12:10> are defined as the OR<2:0> field; the value of 7 in this field selects the OR MUX input line with these four signals: 0, 0, SIZE1, SIZE0. SIZE1 and SIZE0 are signals from the size register (DAPE SIZE 1 H and DAPE SIZE 0 H) and have the value 01.

The microsequencer computes the next microaddress as follows. The control logic determines from bits <15:13> that the next address control field format is a CASE, and enables the output of the OR MUX because of the value in the jump control field. Bits <7:0> of the Shift microinstruction (18 hex) are driven onto the NµA bus from the jump register. The output from the OR MUX: 0001 (binary), is ORed with <7:0> (=18 hex) from the jump register; thus, the value of the low eight bits on the NµA bus is 19 (hex). The NµA MUX selects these eight bits off the NµA bus, and combines them with the bits in the page register to generate the next microaddress. The page register contains the value 19 from the high-order five bits of the current Shift microinstruction; thus, the next microaddress is 1919.

Shift by 1

Control store location 1919 contains the second Shift microinstruction. This Shift microinstruction shifts the contents of the register pointed to by pointer 1, left by 1. Pointer 1 is still pointing at R1, which still contains the value 3. Shifting the value 3 left by one bit effectively multiplies by 2; the result 6 is stored in the RESULT2 register on the data path chip. This is now the correct index value because in the table of words (that is, each table entry is two bytes wide) that will be accessed shortly, the sixth byte from the base address of the table is the address of the third entry.

The CC/DT field of this Shift is also 00, so data path chip pins SIZE1 and SIZE0 are zero, and therefore the chip operation uses data type long.

While these two Shift microinstructions were executing, the IBYTE control logic has caused the third byte (A0) of the ADDW3 instruction to move off the memory control bus into the IBYTE register, and the memory controller has driven the next instruction byte, 05, onto the memory control bus. Thus, the IBYTE register contains A0, the PC contains the virtual address of the third byte of ADDW3 (A0; the PC was incremented by one when the Decode for 41 was executed), and the microcode is ready to compute the base address of the table of words.

Decode A0

The next microaddress generated from the execution of the second Shift microinstruction selects another Decode microinstruction in control store. This Decode is part of a microroutine used to calculate base addresses. As this Decode micro-instruction is evaluated and executed, the same steps that happened when 41 was decoded are repeated:

- The IFUNC field and the contents of the IBYTE register (A0) are used to access the decode ROMs.
- The size register is unaffected because the CC/DT field of this Decode contains the value 2; thus, the contents of the size register is still 01, specifying word.
- Bits <5:0> of the IBYTE register are latched into the IBYTE buffer, driven onto the ID bus, then to the data bus, and into pointer 1 on the data path chip (bit <26> of this Decode microinstruction is also a zero); pointer 1 now contains the value 0, that is, pointer 1 now points to R0.
- The current microaddress plus 1 is pushed on the microstack. This happens for every operand specifier decode when the addressing mode is not register mode, and the content of the IBYTE register is valid.
- The PC on the data path chip is incremented by one because a Decode was just executed.
- The microsequencer calculates the address of the next microinstruction using the low eight bits from the decode ROMs and the high five bits from the jump register. The NμA MUX selects these combined 13 bits off the NμA bus and latches them into the CSA register.

Since the Decode just completed, LOAD I BYTE H is asserted, the next instruction byte, 05, is loaded into the IBYTE register, and the memory controller drives the fifth byte of ADDW3 (95) onto the memory control bus. Thus, the IBYTE register contains 05, the PC contains the virtual address of 05, and the CSA register contains the micro-address of the next microinstruction.

Add

The microaddress in the CSA register selects an Add microinstruction in control store. This Add computes the base operand virtual address. The short operand of this Add microinstruction specifies @pointer 1; that is, use the contents of the register pointed to by pointer 1. The long operand of the Add specifies IB.BYTE; that is use the contents of the IBYTE register.

Pointer 1 points to R0; R0 contains a virtual address, say, 0200. The IBYTE register contains the byte displacement, 05. Any time the long operand of a microinstruction specifies IB.BYTE, the byte currently in the IBYTE register is driven over the ID bus, to the data bus, and into the data path chip, having been sign-extended on the data bus. IB.BYTE as the long operand also causes the data path chip to increment the PC by one.

The CC/DT field of this Add is 00 (binary), so data path chip pins SIZE1 and SIZE0 are zero, and therefore the chip operation uses data type long.

The execution of the Add microinstruction within the data path chip happens as follows. (See Figure 4-4.) The internal data path chip logic decodes the 21 bits of the Add microinstruction stored in the chip's CSR. As a result, the contents of the register pointed to by pointer 1 (00000200) are driven from the register file (where R0 is) over bus A to the

ALU. The sign-extended byte displacement (05) from the IBYTE register is driven from the data bus over the internal chip bus B, and to the ALU. The ALU adds 00000200 and 00000005, and stores the result in the RESULT1 register. The RESULT1 register stores the sum because bit <31> in the Add microinstruction is set.

While the data path chip is executing the Add, the data path microsequencer uses the next address control field of the microinstruction to compute the next microaddress. This field of the Add has the hex value A601; that is, the next address control field format is TRAP. The OR MUX condition that would cause a trap is IB invalid. Since the signal IB INVALID H is not asserted at this time, no trap occurs, and the NμA MUX selects the contents of the μPC (microprogram counter), which is the microaddress of the Add microinstruction plus 1, as the next microaddress.

When IB.BYTE is specified as the long operand, the IBYTE control logic asserts the same series of signals as when a Decode has just been executed, to load the next instruction stream byte into the IBYTE register from the memory control bus. So at this point, the IBYTE register contains the next byte of ADDW3: 95, and the PC contains its virtual address; the last byte of ADDW3 (52) is on the memory control bus, and the CSA register contains the microaddress of the Add microinstruction plus 1.

Move

The microinstruction following the Add is a Move. This Move stores the computed base operand address in a temporary register. The CC/DT field of this Move is 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

A Move microinstruction moves the contents of the location specified by the long operand to the location specified by the short operand. The long operand of this Move is RESULT1, and the short operand specifies a temporary register, TMP(12). When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, 00000205 (the contents of RESULT1), is driven over bus B and stored in TMP(12) in the register file.

Meanwhile, the data path microsequencer computes the address of the next microinstruction from the next address control field of the Move, which is a return. The microaddress at the top of the microstack is the address of the last Decode microinstruction plus 1. So the data path micro-sequencer pops this microaddress off the stack to generate the address of the next microinstruction. The microaddress now at the top of the microstack is the microaddress plus 1 of the Decode micro-instruction that decoded 41 (the second byte of ADDW3).

The IBYTE register still contains 95, 52 is still on the memory control bus, the PC still contains the virtual address of 95, and the CSA register contains the microaddress that was just popped off the top of the microstack.

Add

The microinstruction following the Move is another Add. This Add computes the final effective address of the first operand by adding the base address to the index. The short operand of this Add microinstruction specifies RESULT2, which contains the value 6 from the second Shift operation. The long operand specifies TMP(12), which contains 0205.

The CC/DT field of this Add is 00, so data path chip pins SIZE1 and SIZE0 are zero, and therefore the chip operation uses data type long. The result registers are all 32 bits wide, so this Add operation is manipulating longwords of data.

The value 6 (actually 00000006) is driven over bus A to the ALU, 00000205 is driven over bus B to the ALU, and the sum, 0000020B, is stored in RESULT1 because bit <31> is set in the Add microinstruction.

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Add, which is a branch. The specified branch condition being tested for is register mode. Since this condition is not met, the branch is not taken, and the next microaddress generated is the current microaddress plus 1.

The IBYTE register still contains 95, 52 is still on the memory control bus, the PC still contains the virtual address of 95, and the CSA register latches the contents of μPC, which is the microaddress of the Add plus 1.

Memory Request

The microinstruction following the Add is a Memory Request. This microinstruction sends the computed address of the first operand to the memory controller. The memory controller will then return the data at that address.

The memory function specified in bits <27:23> of the microinstruction is VREAD.RCHECK. The data flow bit is a zero (bit <28>) as the data flow will be from the memory controller to the data path (a read). Thus, a value of 01 (hex) is assembled in the low-order six bits of the memory function latch. The other two latch bits are set by signals from the size register. The last time the size register was loaded was during the Decode for A1; the size register still contains the value 01, which is therefore also the value of the two high-order memory function latch bits. Thus, the value of the output signals BUS MEM CTL <7:0> from the memory function latch is 41 (hex).

Four additional signals are sent to the memory controller over the backplane: DAPT MEM REQ MODE <1:0>, DAPT MODIFY, and DAPT SECOND PART L. For this Memory Request, DAPT MEM REQ MODE <1:0> have the value of the current access mode from the PSL.MODE register, and neither MODIFY or SECOND PART is asserted.

The CC/DT field of this Memory Request is 10 (binary). A value of 2 in the CC/DT field of a Memory Request causes the data path chip size control pins to carry the encoding from the size register. Since the size register contains 01 indicating word, the data path chip pins SIZE1 and SIZE0 are 0 and 1, respectively. Therefore, the memory controller will return a word of data at the specified address.

The long operand specifies the address of the RESULT1 register, so the virtual address 0000020B is driven from RESULT1, over bus B, latched into the MD bus latch, and driven over the memory data bus as BUS MEM DATA <31:00> to the memory controller.

The memory controller asserts the signal MCTT REQ ACK L when it accepts the virtual address 0000020B off the memory data bus and the memory function request information off the memory control bus.

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Memory Request, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Memory Request micro-instruction.

The IBYTE register still contains 95, 52 is still on the memory control bus, the PC still contains the virtual address of 95, and the CSA register latches bits <12:0> of the Memory Request micro-instruction, which were driven onto the NμA bus from the jump register.

Move

The microinstruction following the Memory Request is a Move. This Move sets a register number in pointer 1. The CC/DT field of this Move is 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

A Move microinstruction moves the contents of the location specified by the long operand to the location specified by the short operand. The long operand of this Move is hex 43, which is a location in the constants ROM. The contents of location 43 is the value 14 (hex); hex 14 is the address of a temporary register, TMP(4). The short operand specifies the address of pointer 1. When bits

<36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, 14 (the contents of location 43), is driven over bus B and stored in pointer 1. Thus, pointer 1 points to TMP(4).

The data path microsequencer computes the address of the next microinstruction from the next address control field of the Move, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Move microinstruction.

The IBYTE register still contains 95, 52 is still on the memory control bus, the PC still contains the virtual address of 95, and the CSA register latches bits <12:0> of the Move microinstruction, which were driven onto the NμA bus from the jump register.

Move

Bits <12:0> of the Move microinstruction are the microaddress of another Move. The previous Move was the one intervening cycle between the Memory Request and the availability of the requested data; this Move microinstruction moves the data supplied by the memory controller into the data path chip. The CC/DT field of this Move is 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

The long operand of this Move is MEMORY.DATA which is essentially the address of the memory data bus. The requested data (the first operand) are currently on the memory data bus and latched in the MD bus input latch. The short operand specifies TMP(4). When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, the first operand is driven onto the data bus, into the data path chip over bus B, and stored in TMP(4).

The data path microsequencer computes the address of the next microinstruction from the next address control field of the Move, which is a return. The microaddress currently at the top of the microstack is the one that was stored when the Decode for 41 was executed, which is the microaddress of that Decode microinstruction plus 1. (The microaddress that was stored when the Decode for A0 was executed, was popped for the return from the Move microinstruction that stored the base address in TMP(12).)

The data path microsequencer pops the microaddress off the top of the microstack to generate the address of the next microinstruction. The microstack is now empty.

The IBYTE register still contains 95, 52 is still on the memory control bus, the PC still contains the virtual address of 95, and the CSA register latches the microaddress from the top of the microstack.

Step 2 is now complete; the first operand of the macroinstruction has been evaluated and fetched from memory.

Evaluating the Second Operand Specifier

Decode 95

The popped microaddress selects an operand specifier Decode microinstruction. This Decode is for the current contents of the IBYTE register: 95. As this Decode microinstruction is evaluated and executed, the same steps that happened when A0 was decoded are repeated:

- The IFUNC field and the contents of the IBYTE register (95) are used to access the decode ROMs.
- Bits <38:37> of this Decode have the value 2; that is, use the size register, which still contains the value 01 for word.
- Bits <5:0> of the IBYTE register are latched into the IBYTE buffer, driven onto the ID bus, then to the data bus, and into pointer 2 on the data path chip (bit <26> of this Decode microinstruction is a one). Pointer 1 still points to TMP(4), and pointer 2 points to R5.
- The current microaddress plus 1 is pushed on the microstack. This happens for every operand specifier decode when the addressing mode is not register mode, and the content of the IBYTE register is valid.
- The PC on the data path chip is incremented by one because a Decode was just executed.
- The microsequencer calculates the address of the next microinstruction using the low eight bits from the decode ROMs and the high five bits from the jump register. The NμA MUX selects these combined 13 bits off the NμA bus and latches them into the CSA register.

Since the Decode just completed, LOAD I BYTE H is asserted, the next instruction byte, 52, is loaded into the IBYTE register, and the memory controller drives the next byte in the instruction stream onto the memory control bus. (The next byte is the opcode of the next macroinstruction in the I-stream.) Thus, the IBYTE register contains 52, the PC contains the virtual address of 52, and the CSA register contains the microaddress of the next microinstruction.

Move

The contents of the CSA register select a Move microinstruction from control store. This microinstruction moves the first operand to a temporary register. The CC/DT field of this Move is 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

The long operand of this Move specifies @pointer 1; that is, use the contents of the register pointed to by pointer 1. Pointer 1 currently points to TMP(4); TMP(4) contains the first operand. The short operand also specifies TMP(4).

When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, the first operand is moved from TMP(4) to TMP(4). This register is 32 bits wide, and 32 bits of data are moved because the size control pins specified data type long, but only the low-order word (16 bits) is relevant here.

The data path microsequencer computes the address of the next microinstruction from the next address control field of the Move, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Move microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the CSA register latches bits <12:0> of the Move microinstruction, which were driven onto the NµA bus from the jump register.

Move

Bits <12:0> of the Move microinstruction are the microaddress of another Move. This Move is the first microinstruction in a microroutine that computes the effective address of an operand using the autoincrement deferred addressing mode. The purpose of the Move is to store a new address in pointer 1.

The CC/DT field of this Move is 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

The long operand of this Move is hex 43, which is a location in the constants ROM on the data path chip. The contents of location 43 is the value 14 (hex); hex 14 is the address of a temporary register, TMP(4). The short operand specifies the address of pointer 1. When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, 14 (the contents of location 43), is driven over bus B and stored in pointer 1. Thus, pointer 1 again points to TMP(4).

The data path microsequencer computes the address of the next microinstruction from the next address control field of the Move, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Move microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the CSA register latches bits <12:0> of the Move microinstruction, which were driven onto the NμA bus from the jump register.

Add

Bits <12:0> of the Move microinstruction are the microaddress of an Add. This Add handles the autoincrement for the specified register.

The CC/DT field of this Add is 00, so data path chip pins SIZE1 and SIZE0 are zero, and therefore the chip operation uses data type long.

The short operand of the Add specifies @pointer 2; that is, use the contents of the register pointed to by pointer 2. Pointer 2 currently points to R5; R5 contains a virtual address and is located in the register file (see Figure 4-4). The long operand specifies address 65 (decimal) which is a location in the constants ROM on the data path chip. The contents of location 65 is the value 4; that is, the literal 4.

When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, the virtual address in R5 is driven over bus A to the ALU, the literal 4 is driven over bus B to the ALU, and the sum is stored in RESULT1 because bit <31> is set in the Add microinstruction.

In addition, the register save bit, bit <30>, of the Add microinstruction is set, so the contents of the register specified by the short operand, plus the low 4 bits of the register address, are pushed onto the register save stack on the data path chip. The top entry on the register save stack now looks like this:

35 4	3 0
longword virtual address in R5	0101

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Add, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Add microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the CSA register latches bits <12:0> of the Add microinstruction, which were driven onto the NµA bus from the jump register.

Memory Request

The microinstruction following the Add is a Memory Request. The virtual address of the second operand is located at the address contained in R5. This microinstruction sends the virtual address in R5 to the memory controller.

The memory function specified in bits <27:23> of the microinstruction is VREAD.RCHECK. The data flow bit is a zero (bit <28>) as the data flow will be from the memory controller to the data path (a read). Thus, a value of 01 (hex) is assembled in the low-order six bits of the memory function latch. The other two latch bits are set by signals from the size register. The size register still contains the value 01, which is therefore also the value of the two high-order memory function latch bits. Thus, the value of the output signals BUS MEM CTL <7:0> from the memory function latch is 41 (hex).

Four additional signals are sent to the memory controller over the backplane: DAPT MEM REQ MODE <1:0>, DAPT MODIFY, and DAPT SECOND PART L. For this Memory Request, DAPT MEM REQ MODE <1:0> have the value of the current access mode from the PSL.MODE register, and neither MODIFY or SECOND PART is asserted.

The CC/DT field of this Memory Request is 11 (binary). A value of 3 in the CC/DT field of a Memory Request selects a data type of longword, so data path chip pins SIZE1 and SIZE0 are both ones. Therefore, the chip operation uses data type longword.

The long operand specifies @pointer 2, so the longword virtual address contained in R5 is driven over bus B, latched into the MD bus latch, and driven over the memory data bus as BUS MEM DATA <31:00> to the memory controller.

The memory controller asserts the signal MCTT REQ ACK L when it accepts the virtual address off the memory data bus and the memory function request information off the memory control bus.

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Memory Request, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Memory Request microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the CSA register latches bits <12:0> of the Memory Request, which were driven onto the NµA bus from the jump register.

Move

Bits <12:0> of the Memory Request microinstruction are the microaddress of another

Move. The purpose of the Move is to store the updated address in the register pointed to by pointer 2.

The CC/DT field of this Move is also 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

The long operand of this Move specifies RESULT1, which currently contains the virtual address in R5 plus 4; that is, the incremented address. The short operand specifies @pointer 2. Pointer 2 is still pointing to R5. When bits <36:16> of this micro-instruction are clocked into the CSR on the data path chip, decoded and executed, the contents of RESULT1 (the incremented address) are driven over bus B and stored in the register pointed to by pointer 2; that is, R5. Thus, R5 now contains the incremented address.

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Move, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Move microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the CSA register latches bits <12:0> of the Move, which were driven onto the NμA bus from the jump register.

Move

Bits <12:0> of the Move microinstruction are the microaddress of another Move. The previous Move was the one intervening cycle between the Memory Request and the availability of the requested data;

this Move microinstruction moves the data supplied by the memory controller into the data path chip.

The CC/DT field of this Move is also 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

The long operand of this Move is MEMORY.DATA which represents the "address" of the memory data bus. The requested data (the address of the second operand) are currently on the memory data bus and latched in the MD bus input latch. The short operand specifies TMP(12). When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, the address of the second operand is driven onto the data bus, into the data path chip over bus B, and stored in TMP(12).

The next address control field format of this Move is TRAP. The OR<2:0> field selects the input to the OR MUX that has the signals memory error, page crossing, TB miss, and modify refuse (see Figure 3-3). The JC field of the Move tests for the condition OR MUX not equal to 0. If any of the above OR MUX signals are active, a trap occurs to the microroutine that handles memory errors; this microroutine is located in page zero of control store. A trap would also cause the current microaddress plus 1 to be saved on the microstack.

None of these signals are active at the moment, however, so the NμA MUX selects the contents of μPC, which is the current microaddress plus 1.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the current microaddress plus 1 is latched in the CSA register.

Memory Request

The current microaddress plus 1 selects a Memory Request microinstruction. The last Move microinstruction moved the address of the second operand into TMP(12) on the data path chip; this Memory Request uses this address to fetch the actual operand.

The memory function specified in bits <27:23> of the microinstruction is VREAD.RCHECK. The data flow bit is a zero (bit <28>) as the data flow will be from the memory controller to the data path (a read). Thus, a value of 01 (hex) is assembled in the low-order six bits of the memory function latch. The size register still contains the value 01, which is therefore also the value of the two high-order memory function latch bits. Thus, the value of the output signals BUS MEM CTL <7:0> from the memory function latch is 41 (hex).

For this Memory Request, DAPT MEM REQ MODE <1:0> have the value of the current access mode from the PSL.MODE register, and neither MODIFY or SECOND PART is asserted.

The CC/DT field of this Memory Request is 10 (binary). A value of 2 in the CC/DT field of a Memory Request causes the data path chip size control pins to carry the encoding from the size register. The size register still contains 01 to indicate word, so data path chip pins SIZE1 and SIZE0 are 0 and 1, respectively. Therefore, the memory controller will return a word of data (the second operand) at the specified address.

The long operand specifies TMP(12), so the virtual address contained in TMP(12) is driven over bus B, latched into the MD bus latch, and driven over the memory data bus as BUS MEM DATA <31:00> to the memory controller.

The memory controller asserts the signal MCTT REQ ACK L when it accepts the virtual address off the memory data bus and the memory function request information off the memory control bus.

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Memory Request, which is a jump; that is, the next microaddress is supplied in bits <12:0> of the Memory Request microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the CSA register latches bits <12:0> of the Memory Request, which were driven onto the NµA bus from the jump register.

Move

Bits <12:0> of the Memory Request microinstruction are the microaddress of another Move. The purpose of the Move is to store a new address in pointer 2.

The CC/DT field of this Move is also 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

The long operand of this Move is hex 44, which is a location in the constants ROM. The contents of location 44 is the value 16 (hex); hex 16 is the address of a temporary register, TMP(6). The short operand specifies pointer 2. When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, 16 (the contents of location 44) is driven over bus B and stored in pointer 2. Thus, pointer 2 now points to TMP(6).

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Move, which is a jump; the next microaddress is supplied in bits <12:0> of the Move microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the CSA register latches bits <12:0> of the Move, which were driven onto the NμA bus from the jump register.

Move

Bits <12:0> of the Move microinstruction select another Move. The previous Move was the one intervening cycle between the Memory Request and the availability of the requested data; this Move microinstruction moves the data supplied by the memory controller into the data path chip.

The CC/DT field of this Move is also 10 (binary), so data path chip pins SIZE1 and SIZE0 are 1 and 0, respectively. Therefore, the chip operation uses data type long.

The long operand of this Move is MEMORY.DATA which represents the "address" of the memory data bus. The requested data (the second operand) are currently on the memory data bus and latched in the MD bus input latch. The short operand specifies TMP(6). When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, the second operand is driven onto the data bus, into the data path chip over bus B, and stored in TMP(6).

The next address control field format of this Move is a return. The microaddress currently at the top of the microstack is the one that was stored when the Decode for 95 was executed, which is the microaddress of that Decode microinstruction plus 1.

The data path microsequencer pops the microaddress off the top of the microstack to generate the address of the next microinstruction. The microstack is now empty.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and the current microaddress plus 1 is latched in the CSA register.

Step 3 is now complete; the second operand of the macroinstruction has been evaluated and fetched from memory, and the contents of R5 incremented.

Adding the Operands

Add

The popped microaddress selects an Add microinstruction. This Add handles the actual addition of the operands.

The CC/DT field of this Add is 11 (binary). A value of 3 in the CC/DT field of an Add means use the data type in the size register. Thus, data path chip pins SIZE1 and SIZE0 reflect the contents of the size register, which is still 01 to indicate word.

Therefore, the chip operation uses data type word, which is appropriate since this is the add operation of the Add Word macroinstruction.

The short operand of the Add specifies @pointer 1; that is, use the contents of the register pointed to by pointer 1. Pointer 1 is still pointing to TMP(4), which contains the first operand. The long operand specifies @pointer 2; that is, use the contents of the register pointed to by pointer 2. Pointer 2 is still pointing to TMP(6), which contains the second operand.

When bits <36:16> of this microinstruction are clocked into the CSR on the data path chip, decoded and executed, the first operand is driven over bus A to the ALU, the second operand is driven over bus B to the ALU, and the sum is stored in RESULT0 because bit <31> is clear in the Add microinstruction.

When the opcode for ADDW3 was decoded, the signals DAPF CC CLASS <1:0> were part of the output from the decode ROMs; the value of this field was 1, meaning arithmetic (see Table 4-2). The CC/DT field value of 3 and CC CLASS value of 1 are combined and encoded in the condition code PALs to generate the field DAPE CC <F3:F0>. The result is a value for <F3:F0> that means load ALU and PSL CCs arithmetic. Consequently, when this Add microinstruction is executed in the data path chip, the ALU condition codes are set depending on the result, and the PSL condition codes are set from the ALU condition codes.

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Add, which is a jump; the next microaddress is supplied in bits <12:0>

of the Add microinstruction.

The IBYTE register still contains 52 and the PC still contains its virtual address, the next byte in the I-stream is still on the memory control bus, and bits <12:0> of the Add microinstruction are latched in the CSA register.

Decode 52

Bits <12:0> of the Add microinstruction are the microaddress of an operand specifier Decode. This Decode is for the current contents of the IBYTE register: 52. As this Decode microinstruction is evaluated and executed, the following steps occur:

- The IFUNC field and the contents of the IBYTE register (52) are used to access the decode ROMs.
- Bits <38:37> of this Decode have the value 2; that is, use the size register, which still contains the value 01 for word.
- Bits <5:0> of the IBYTE register are latched into the IBYTE buffer, driven onto the ID bus, then to the data bus, and into pointer 2 on the data path chip (bit <26> of this Decode microinstruction is a one). Pointer 1 still points to TMP(4), and pointer 2 now points to R2.
- The current microaddress plus 1 is **not** pushed on the microstack because the addressing mode is register mode. Thus, the microstack remains empty.
- The PC on the data path chip is incremented by one because a Decode was just executed, and now contains the virtual address of the next byte in the instruction stream.

- The microsequencer calculates the address of the next microinstruction as μPC plus 1 because the operand specifier, 52, specifies register mode. The NμA MUX selects μPC plus 1 off the NμA bus and latches them into the CSA register.

Since the Decode just completed, LOAD I BYTE H is asserted, the next byte in the I-stream is loaded into the IBYTE register (the opcode of the next macroinstruction), and the memory controller drives the next byte in the instruction stream onto the memory control bus.

Move

The microaddress latched in the CSA register selects a Move. The purpose of the Move is to move the sum computed in the Add to the location specified by the instruction byte, 52.

The CC/DT field of this Move is 11 (binary). A value of 3 in the CC/DT field of a Move selects the data type specified in the size register, which is still 01, indicating word. Therefore, the chip operation uses data type word.

The long operand of this Move specifies RESULT0, which is where the sum from the Add microinstruction is stored. The short operand specifies @pointer 2; pointer 2 is pointing to R2 because of the operand specifier decode just executed. When bits <36:16> of this Move microinstruction are clocked into the CSR on the data path chip, decoded and executed, the sum in RESULT0 is driven over bus B and stored in R2.

The data path microsequencer computes the address of the next microinstruction using the next address control field of the Move, which is a jump; the next microaddress is supplied in bits <12:0> of the Move microinstruction.

Step 4 is now complete; the first and second operands of the macroinstruction have been added together and the sum stored in R2.

The microaddress supplied in bits <12:0> of the Move selects an opcode Decode microinstruction (an IRD), and the decoding and execution of the next macroinstruction in the I-stream begins.

Figure 4-13 summarizes the microinstructions used to decode and execute the ADDW3 macro-instruction. It also completes this discussion of the data path microprogram level flow.

Chapter 3 describes the data path microcode and this chapter describes the data path hardware. Similarly, the next two chapters describe the memory controller microcode and hardware.

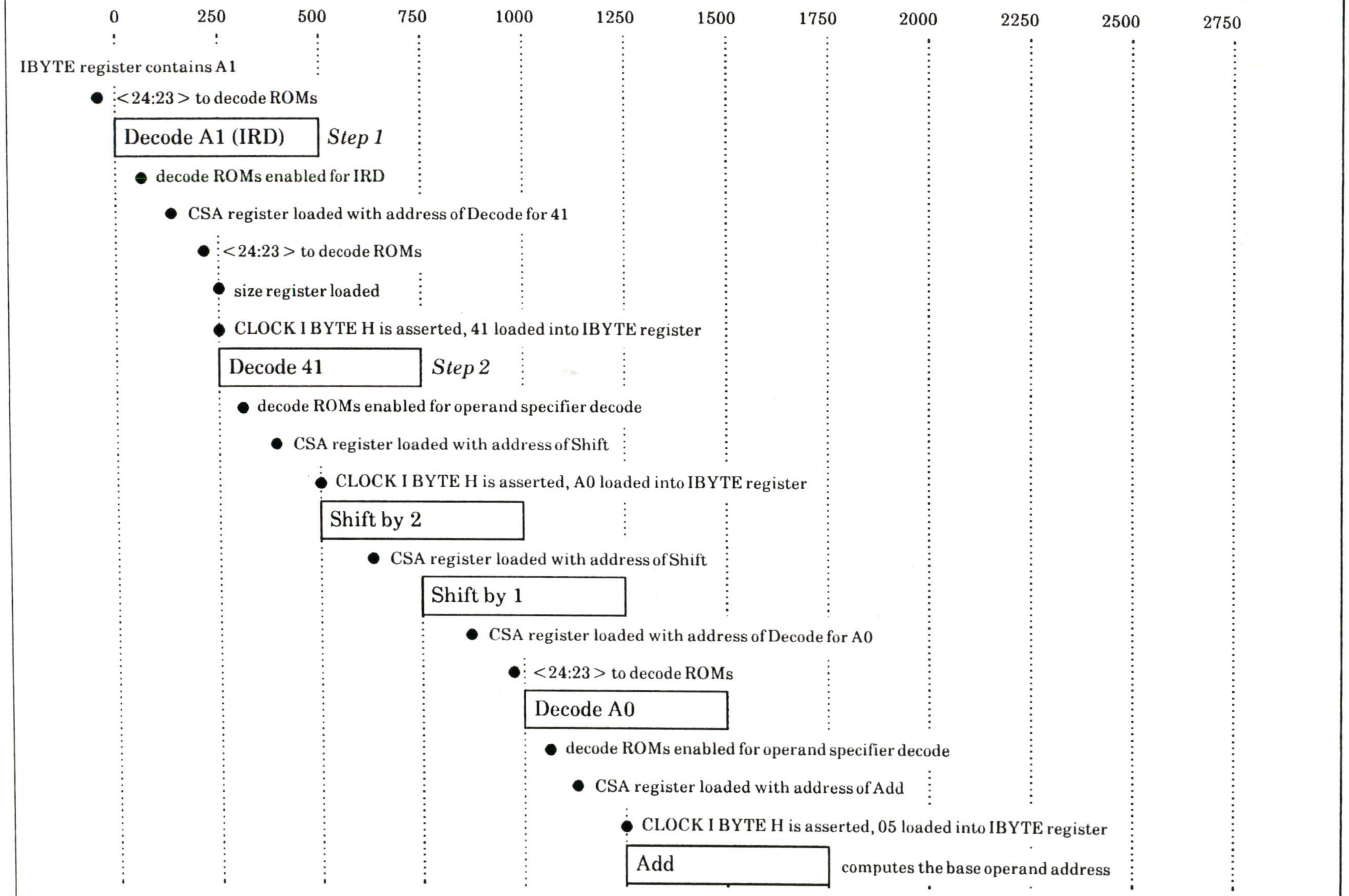
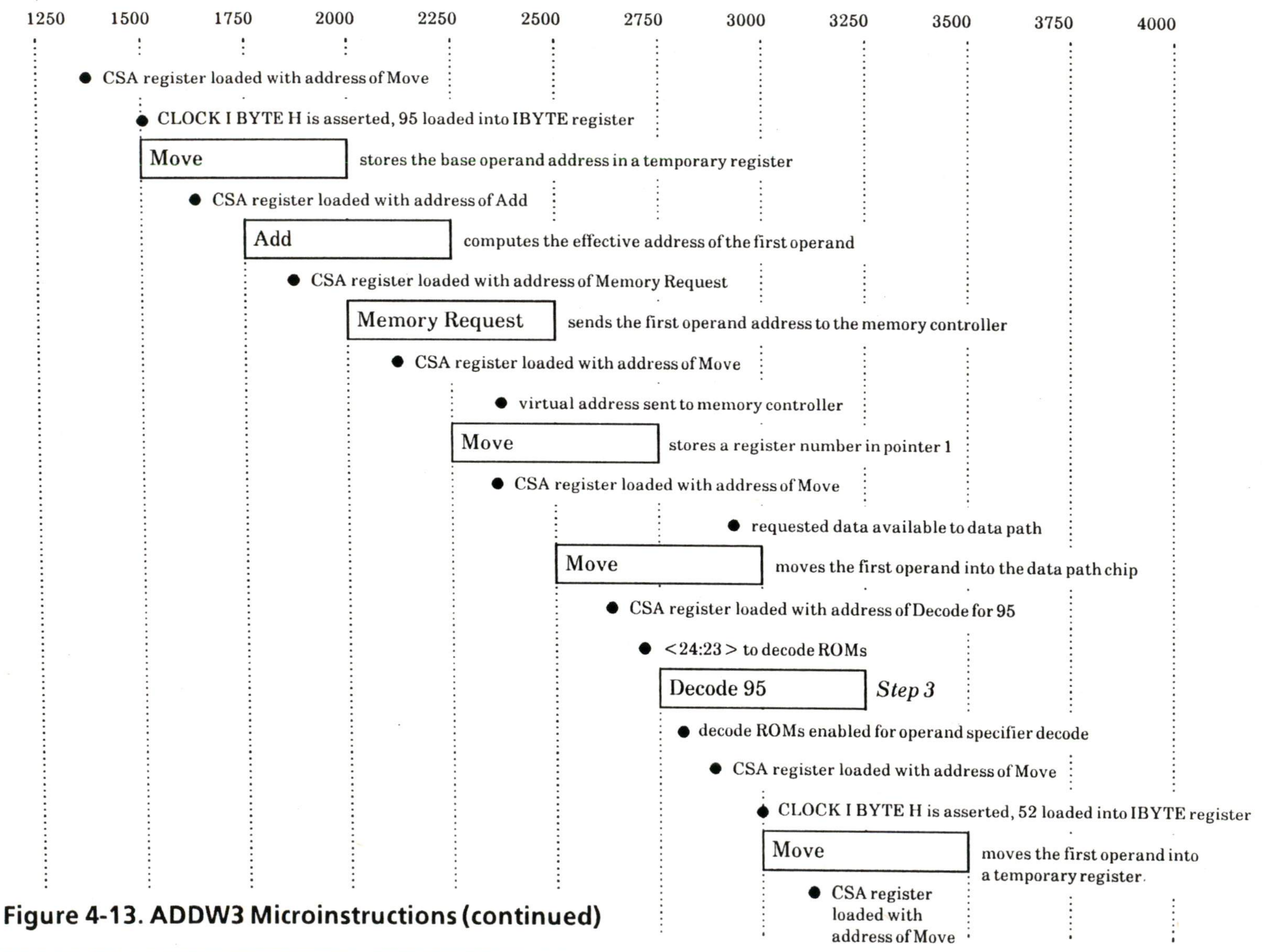

0
250
500
750
1000
1250
1500
1750
2000
2250
2500
2750
IBYTE register contains A1
<24:23> to decode ROMs
Decode A1 (IRD)
Step 1
decode ROMs enabled for IRD
CSA register loaded with address of Decode for 41
<24:23> to decode ROMs
size register loaded
CLOCK I BYTE H is asserted, 41 loaded into IBYTE register
Decode 41
Step 2
decode ROMs enabled for operand specifier decode
CSA register loaded with address of Shift
CLOCK I BYTE H is asserted, A0 loaded into IBYTE register
Shift by 2
CSA register loaded with address of Shift
Shift by 1
CSA register loaded with address of Decode for A0
<24:23> to decode ROMs
Decode A0
decode ROMs enabled for operand specifier decode
CSA register loaded with address of Add
CLOCK I BYTE H is asserted, 05 loaded into IBYTE register
Add
computes the base operand address

Figure 4-13. ADDW3 Microinstructions



1250
1500
1750
2000
2250
2500
2750
3000
3250
3500
3750
4000
● CSA register loaded with address of Move
● CLOCK I BYTE H is asserted, 95 loaded into IBYTE register
Move
stores the base operand address in a temporary register
● CSA register loaded with address of Add
Add
computes the effective address of the first operand
● CSA register loaded with address of Memory Request
Memory Request
sends the first operand address to the memory controller
● CSA register loaded with address of Move
● virtual address sent to memory controller
Move
stores a register number in pointer 1
● CSA register loaded with address of Move
● requested data available to data path
Move
moves the first operand into the data path chip
● CSA register loaded with address of Decode for 95
● <24:23> to decode ROMs
Decode 95
Step 3
● decode ROMs enabled for operand specifier decode
● CSA register loaded with address of Move
● CLOCK I BYTE H is asserted, 52 loaded into IBYTE register
Move
moves the first operand into a temporary register.
● CSA register loaded with address of Move

Figure 4-13. ADDW3 Microinstructions (continued)

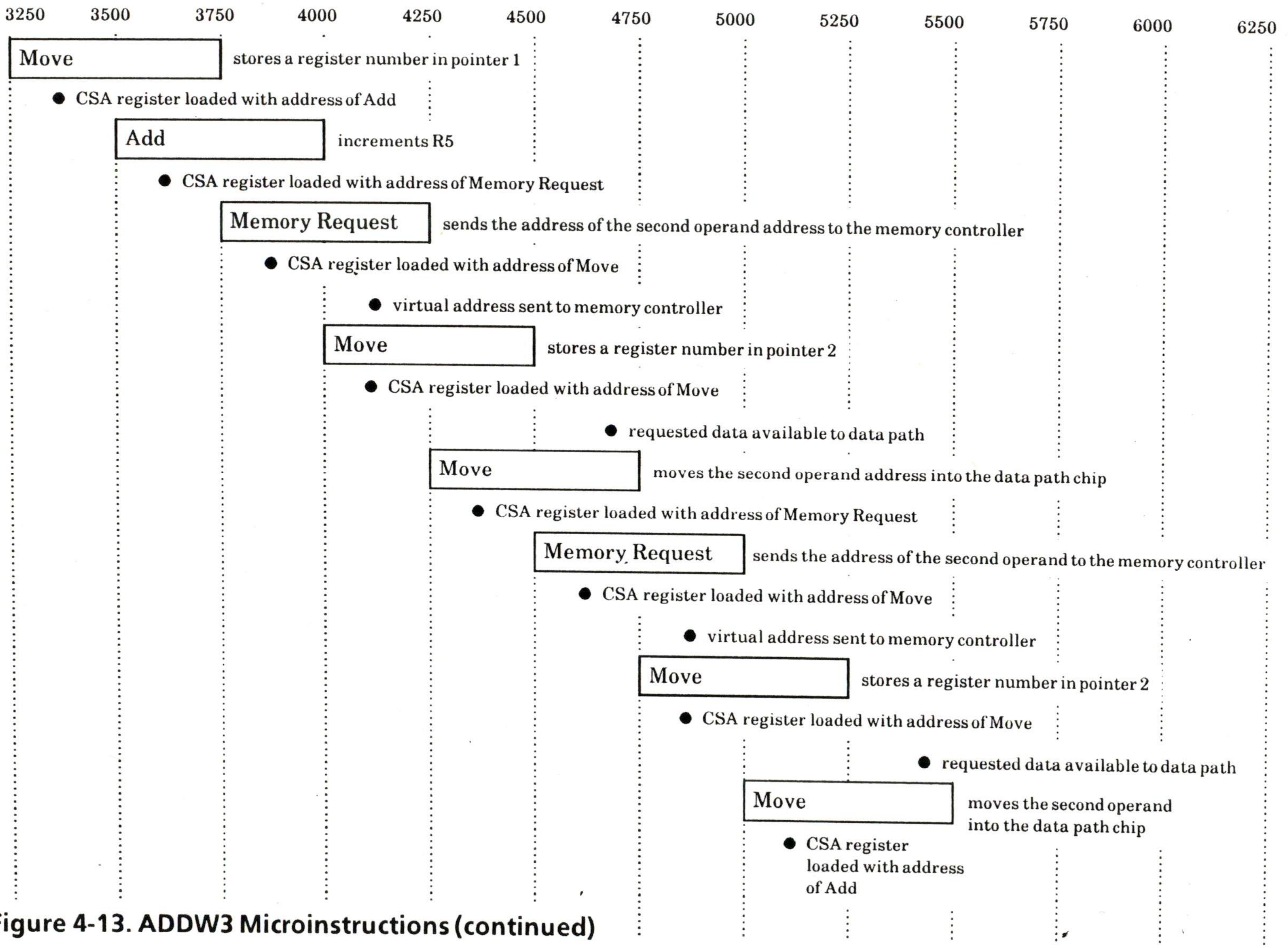

3250
3500
3750
4000
4250
4500
4750
5000
5250
5500
5750
6000
6250
Move
stores a register number in pointer 1
● CSA register loaded with address of Add
Add
increments R5
● CSA register loaded with address of Memory Request
Memory Request
sends the address of the second operand address to the memory controller
● CSA register loaded with address of Move
● virtual address sent to memory controller
Move
stores a register number in pointer 2
● CSA register loaded with address of Move
● requested data available to data path
Move
moves the second operand address into the data path chip
● CSA register loaded with address of Memory Request
Memory Request
sends the address of the second operand to the memory controller
● CSA register loaded with address of Move
● virtual address sent to memory controller
Move
stores a register number in pointer 2
● CSA register loaded with address of Move
● requested data available to data path
Move
moves the second operand into the data path chip
● CSA register loaded with address of Add

Figure 4-13. ADDW3 Microinstructions (continued)

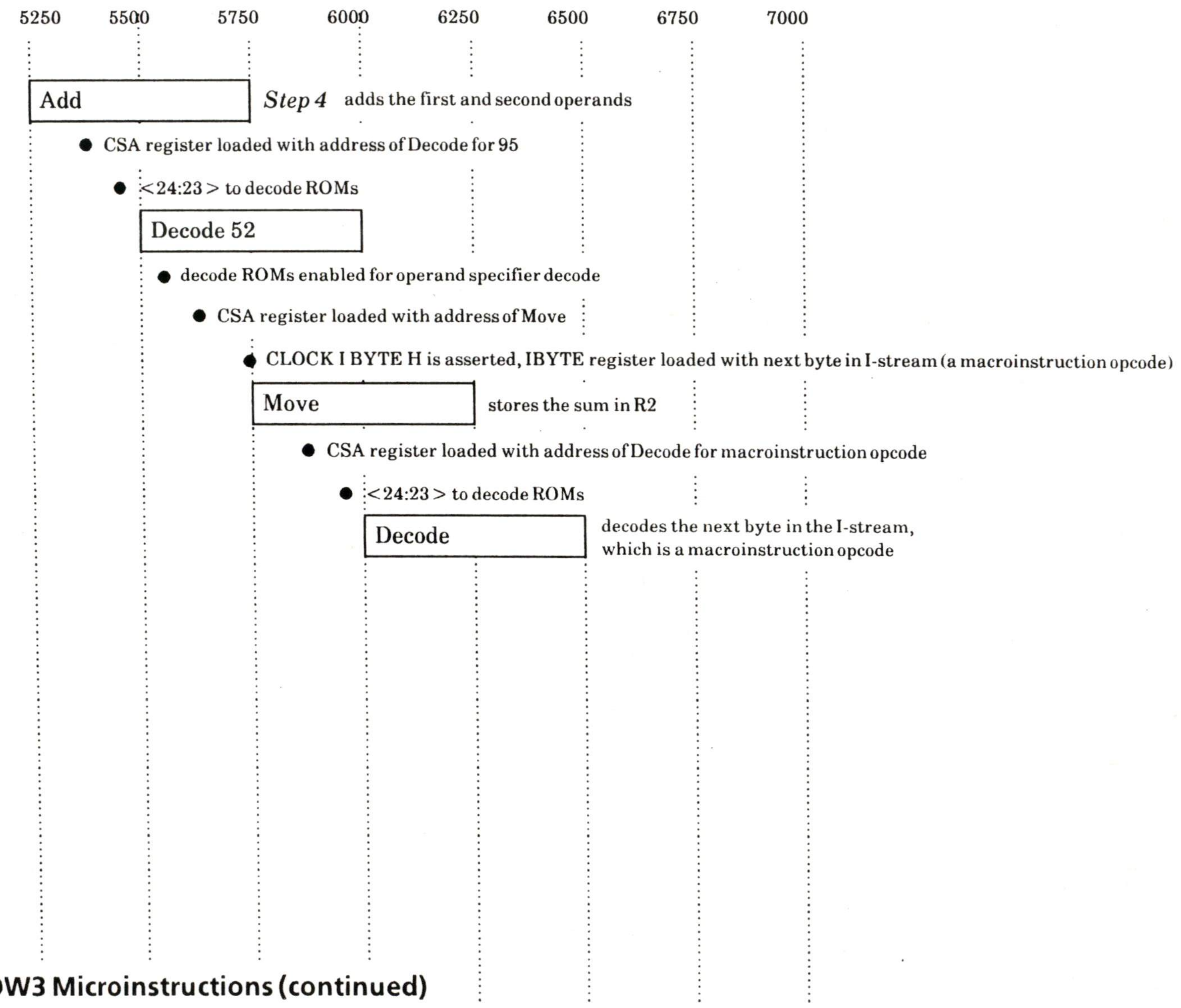

5250
5500
5750
6000
6250
6500
6750
7000
Add
Step 4 adds the first and second operands
● CSA register loaded with address of Decode for 95
● <24:23> to decode ROMs
Decode 52
● decode ROMs enabled for operand specifier decode
● CSA register loaded with address of Move
● CLOCK I BYTE H is asserted, IBYTE register loaded with next byte in I-stream (a macroinstruction opcode)
Move
stores the sum in R2
● CSA register loaded with address of Decode for macroinstruction opcode
● <24:23> to decode ROMs
Decode
decodes the next byte in the I-stream, which is a macroinstruction opcode

Figure 4-13. ADDW3 Microinstructions (continued)

Chapter 5
Memory Controller Microcode

The memory controller module accepts memory request commands issued by the data path micromachine and sequences the necessary functional blocks to carry out the command. The memory controller has its own set of microcode, stored in the MCT control store, to implement the commands from the data path module. Each memory controller microinstruction allows simultaneous activity of several functional blocks. This chapter describes these memory controller microinstructions.

Memory Controller Function Parameters

Every memory controller function involves a set of parameters; that is, the memory controller must know the following information to carry out the memory request from the data path module:

• Address	A virtual or physical memory address, or sometimes the actual data to be written.
• Access Mode	The mode used to check if the operation can be performed. The access mode is specified as either the current mode or kernel mode.
• Data Flow	The direction that data will flow on the memory data bus during the memory operation; that is, whether the operation is a read or a write.

• Data Type	The size of the data to be read or written. The size is specified as byte, word, or longword, or determined by the contents of the size register.
• Map Enable	A memory controller state flag that specifies whether the translation buffer is to be used. It is set via a MTPR instruction at the macromachine level.
• Modify Intent	Indicates whether the data access is to be checked for read or write access intent. Modify intent does not signify whether data are read or written, but rather which access intent is checked.
• Previous Function	The previously latched memory function, data flow, and data type bits. These bits are saved in the second memory function latch when bit <31>, the latch bit, of a Memory Request is set.
• Second Part Flag	A flag that specifies whether the first or second part of a function is being executed.

When the data path module executes a Memory Request or I-stream Request microinstruction, eight bits of information are latched in the memory function latches and delivered to the memory controller over the memory control bus: the 5-bit memory function code, two bits of data type, and one data flow bit. An additional four bits are delivered over the backplane: two access mode bits, a modify intent bit, and the second part bit. These twelve bits are recombined on the memory controller module; the following eight bits are presented to the MCT control store as the low-order bits of the 10-bit microaddress:

9	8	7	6 5	4 0
1	1	second part flag	data type	memory function code

Figure 5-1. MCT Microaddress Format

The MCT microsequencer forces the two high-order bits of the microaddress to ones. The remaining four bits now are the data flow bit, the two access mode bits (MEM REQ MODE <1:0>), and the modify intent bit (MODIFY).

The signals generated by the data flow and modify intent bits are two of the inputs to the MCT branch MUX. The modify intent bit is also one input to the access violation logic, and the two access mode bits are inputs to the access violation logic.

The address (or data) needed by the memory controller is specified by the long operand of the data path microinstruction that is making the memory request. Thus, the address is delivered to the memory controller over the memory data bus.

The memory controller saves the state of the map enable flag in a control and status register (CSR); this bit is cleared and set by the data path.

So for each memory request from the data path, the memory controller receives the proper function parameters: four bits from the backplane, eight bits from the memory control bus, 32 bits of address or data over the memory data bus, and has access to the CSR that contains the map enable bit.

When the memory controller receives these function parameters, the proper bits are presented to control store as the next microaddress, and the first microinstruction is accessed in the microroutine that handles the requested memory function.

Microinstruction Format

The memory controller microinstruction is sixty-four bits wide, but only sixty bits are used. The bits are divided into three major functional fields: Q22 bus interface control, functional block control, and microprogram control.

63 56	55 52	51 18	17 0
MCA Bus Source Field	Spare	Functional Block Control	Microprogram Control

Figure 5-2. MCT Microinstruction Format

The spare bits are unused and cause no action in the memory controller module. The following sections describe the three major functional fields in more detail.

MCA Bus Source Field

Seven of the eight bits in the MCA bus source field select which MCA bus sources drive the bus; the eighth bit allows the memory controller module to communicate with the Q22 bus interface. The MCA bus source field consists of the Q22 bus function request bit, the merge register output enable, the physical address register (PAR) output enable, the adder output enable, the register file output enables, and the transceiver control bits:

63	62	61	60	59 58	57 56
Q22 bus function request	merge output enable	PAR output enable	adder output enable	register file output enables	transceiver control

Figure 5-3. MCA Bus Source Field

Q22 Bus Function Request

Bit <63> of each memory controller microinstruction is the function request bit, active high. This bit allows the writing of a function code (MCT microinstruction bits <47:45>) to the Q22 bus interface. (Referencing Figure 2-1, the Q22 bus interface is the Q22 bus controller and the Q22 bus registers.)

If bit <63> is clear, the function code of the microinstruction must be a zero to indicate no operation. If bit <63> is set, the Q22 bus interface latches the function code, and arbitrates for the Q22 bus. Upon acquiring the bus, the Q22 bus controller asserts the memory function address, and waits for the go bit. The bus is held until the go bit is asserted to allow the function to proceed, or until a no operation function code is received causing the Q22 bus controller to abort the function.

Merge Register Output Enable

Bit <62> of each memory controller microinstruction is the merge register output enable bit, active low. This bit enables the current contents of the merge register onto the memory controller address (MCA) bus (see Figure 2-1).

PAR Output Enable

Bit <61> of each memory controller microinstruction is the physical address register output enable bit, active low. This bit enables the current contents of the physical address register (PAR) onto the memory controller address bus as MCA <29:28> and MCA <21:09>.

MCA <29> when set indicates that the physical address is located in I/O space (the high-addressed half) of physical memory. MCA <28> when set indicates that the physical address is not to be saved in the cache because it is an address in a shared physical memory. MCA <21:09> are the translated physical address bits after a TB/cache access. (Seahorse physical addresses are 22 bits long plus the I/O space flag; the remaining nine bits—the page offset bits—are supplied from either the 9-bit adder or the page offset portion of the register file. See Figure 6-1 for the locations of the adder and the register file.)

Adder Output Enable

Bit <60> of each memory controller microinstruction is the adder output enable bit, active low. This bit enables the current contents of the 9-bit adder (some previously incremented and saved value) onto the memory controller address bus as MCA <8:0>.

Register File Output Enables

Bits <59:58> of each memory controller microinstruction are the register file output enable bits, active low. These bits cause the addressed location of the register file to be driven onto the memory controller address (MCA) bus.

The register file is divided into a 23-bit page portion and a 9-bit offset portion. The page portion, the offset, or both may be driven onto the MCA bus. Microinstruction bit <59> enables the offset portion onto the MCA bus; bit <58> enables the page portion onto the MCA bus.

The register file can be used as the source of physical and virtual addresses. When addresses are supplied from the register file to the MCA bus to access the TB/cache, the register file must be addressed the cycle before the TB/cache access is made and the address maintained into the next cycle.

Microinstruction bit <59> is also the output enable for the control and status registers (CSRs). The four CSRs control memory management functions and reflect error status; they share the register file address space but are read and written over the memory controller data (MCD) bus. When bit <59> is a zero, the CSRs are enabled as well as the offset portion of the register file.

Transceiver Control

Bit <57> of each memory controller microinstruction is the transceiver enable bit, and bit <56> specifies the transceiver direction. These two bits control the operation of the MCT transceiver, which isolates the memory data bus (the 32-bit bus between the two modules) from the MCA bus (the 32-bit bus internal to the MCT; see Figure 2-1). The MCT transceiver is the data communication port to the DAP module for all data transfers except bytes from the I-stream. Table 5-1 shows the encoding for the transceiver control field of the MCT microinstruction.

Table 5-1. Transceiver Control Field

Transceiver Enable <57>	Transceiver Direction <56>	Result
0	0	DAP to MCT
0	1	MCT to DAP
1	x	no operation

Functional Block Control Field

The 34-bit functional block control field provides clocking and output enables for each functional block in the memory controller. Figure 5-4 shows the bit definitions for this field. The following sections describe these subfields in more detail.

51 50	49 43	42	41	40	39 36	35 34	33 32	31	30 29	28	27	26 24	23 22	21 18
prefetch FIFO control	Q22 bus control	PAR latch enable	reverse pass output enable	reverse pass latch enable	merge register selects	byte rotate select	TB/ cache RAM control	TB/ cache valid bit	TB/ cache access select	adder latch enable	adder subtract enable	adder constant select	register file write enables	register file address

Figure 5-4. Functional Block Control Field

Prefetch FIFO Control

Bits <51:50> of each memory controller microinstruction are the instruction prefetch FIFO control bits, active low. The two FIFO chips and the associated control logic provide first-in-first-out storage for up to 16 bytes of prefetched data from the instruction stream. The control logic asserts the prefetch enable flag whenever less than 8 bytes of I-stream data are contained in the prefetch FIFO.

Prefetch FIFO Clear. Microinstruction bit <51> causes the entire contents of the I-stream prefetch FIFO to be cleared. This direct clear function is used to synchronize the FIFO to the instruction stream after program flow changes.

Prefetch FIFO Load Clock. Microinstruction bit <50> controls the clocking of data from the low byte of the MCA bus into the I-stream prefetch FIFO. This clock occurs at the end of the current MCT microcycle so that data is fetched and written into the FIFO in one cycle.

Q22 Bus Control

Bits <49:43> of each memory controller microinstruction are the Q22 bus control field, subdivided as shown in Figure 5-5.

49	48	47 45	44	43
clear function code	Q22 bus go bit	Q22 bus function code	Q22 bus read data output enable	Q22 bus write enable

Figure 5-5. Q22 Bus Control Field

Q22 Bus Clear Function Code Enable. Bit <49> is active high and allows the Q22 bus controller to clear a function code at the end of a bus cycle, releasing the Q22 bus. The Q22 bus controller repeatedly executes the last given function until this bit is asserted to allow it to return to the quiescent state.

Q22 Bus Go Bit. Bit <48> is active high. For all Q22 bus operations, bus arbitration begins with the posting of a function code. When the Q22 bus controller acquires the bus, it asserts the address and waits for the go bit before proceeding with the bus cycle. The go bit may be posted at the same time as the function code (<47:45>) and the function request bit (<63>) to cause a bus cycle to begin immediately, or it may be posted during a later cycle. The go bit is posted immediately for operations such as I/O space accesses, interrupt vector reads, and memory writes; it is posted later for operations such as a Q22 bus read after a cache miss.

Q22 Bus Function Code. Bits <47:45> select the type of Q22 bus operation to be performed. The encoding for this field is shown in Table 5-2. When bit <63> is a one, the function code in bits <47:45> is delivered to the Q22 bus controller, causing the controller to begin arbitration for the bus. When the controller gains control of the bus, it drives an address onto the bus and holds the bus until it receives the go bit or a function code of 000, which specifies no operation.

Table 5-2. Function Code Field

<47:45>	Operation	Mnemonic

000	no operation	
001	write word	DATO
010	write byte	DATOB
011	write block	DATBO
100	read word	DATI
101	read block	DATBI
110	read interrupt vector	
111	read interlocked	DATIO

Q22 Bus Read Data Output Enable. Bit <44> is active high and allows data from the Q22 bus interface to be driven onto the memory controller data bus as MCD <21:00>. These 22 bits may represent a 16-bit datum read from the Q22 bus, a 9-bit Q22 bus interrupt vector, or a 22-bit cache invalidate address.

Q22 Bus Write Enable. Bit <43> is active high and causes the data stable on MCA <29> and MCA <21:00> to be written to the Q22 bus write register (see Figure 6-1). The data written may represent a 22-bit physical address, a 13-bit I/O space address (if MCA <29> is a one), or a 16-bit datum to be written to Q22 bus memory or an I/O device.

PAR Latch Enable

Bit <42> of each memory controller microinstruction is the physical address register (PAR) latch enable bit, active low. This bit enables the PAR to latch the PTE information or test data on its inputs. This signal also latches the 4-bit protection field and the modify bit. The protect field and the modify bit are read from the PTE in the translation buffer (TB) and are used to perform the access violation and modify refused checks.

Reverse Pass Output Enable

Bit <41> of each memory controller microinstruction is the reverse pass output enable bit, active low. This bit causes the current contents of the reverse pass latch to be enabled onto the MCD bus. (See Figure 6-1 for the location of the reverse pass latch.)

Reverse Pass Latch Enable

Bit <40> of each memory controller microinstruction is the reverse pass latch enable bit, active low. This bit causes the data on the MCA bus to be latched into the 32-bit-wide reverse pass latch. The reverse pass latch is transparent, allowing data from the merge register to be passed back to the MCD bus, rotated, and presented to the merge register inputs in one cycle.

Merge Register Selects

Bits <39:36> of each memory controller microinstruction are the merge register select bits, active low. These bits control the clocking of data into the four bytes of the rotator and merge register (see Figure 2-1 or Figure 6-1 for the location of these components). Since each byte is individually enabled, bytes from separate sources can be merged to accomplish nonlongword-aligned reads and writes to memory.

Bits <39:36> correspond to merge register bytes 3 through 0, respectively; that is, bit 39 controls the clocking of data into merge register byte 3, and so on. Byte 3 of the merge register is the most significant byte of the output longword. Table 5-3 shows the encoding for the merge register select field.

Table 5-3. Merge Register Selects

<39:36>	Action
1111=0F	load no bytes
1110=0E	load byte 0
1101=0D	load byte 1
1100=0C	load bytes 1 and 0
1011=0B	load byte 2
1010=0A	load bytes 2 and 0
1001=09	load bytes 2 and 1
1000=08	load bytes 2, 1, and 0
0111=07	load byte 3
0110=06	load bytes 3 and 0
0101=05	load bytes 3 and 1
0100=04	load bytes 3, 1, and 0
0011=03	load bytes 3 and 2
0010=02	load bytes 3, 2, and 0
0001=01	load bytes 3, 2, and 1
0000=00	load all bytes

Byte Rotate Select

Bits <35:34> of each memory controller microinstruction are the byte rotate select bits. This two bit field controls the circular byte shift performed on the data from the MCD bus that are presented to the merge register. The encoding is shown in Table 5-4.

Table 5-4. Byte Rotate Select

<35:34>	Action
00	circulate longword 0 bytes right

01	circulate longword 1 byte right
10	circulate longword 2 bytes right
11	circulate longword 3 bytes right

TB/Cache RAM Control

Bits <33:32> of each memory controller microinstruction are the TB/cache RAM control bits. Bit <33> is the chip select, and bit <32> is the write enable. These two bits control the read and write operations of the TB/cache RAM. (The translation buffer and the instruction and data cache share one RAM.) All data are read to, and written from, the MCD bus. The encoding is shown in Table 5-5.

Table 5-5. TB/Cache RAM Control

Chip Select <33>	Write Enable <32>	Result
1	1	read
1	0	write
0	x	no operation

TB/Cache Valid

Bit <31> of each memory controller microinstruction is the TB/cache valid bit, active high. This bit directly controls a hardware TB/cache valid bit that is written into the TB/cache as part of the tag. If bit <31> is a one, the hardware valid bit is written high to indicate that the associated information being stored in the TB or cache is a valid copy of the same information in memory.

If the hardware valid bit is written high, it enables a comparison between the tag for the cached entry and the presented address; this comparison may then produce a TB/cache hit. If the valid bit is written low, the comparison is disabled and a TB/cache miss is forced.

TB/Cache Access Select

Bits <30:29> of each memory controller microinstruction are the TB/cache access select bits, active high. These bits select the type of TB/cache access to be performed. The encoding is shown in Table 5-6.

Table 5-6. TB/Cache Access Select

<30:29>	Access Type
00	normal TB access read or write
01	normal cache read or write
10	TB invalidate via adder
11	conditional cache invalidate

When <30:29> are 10, the translation buffer (TB) is accessed by MCA <8:2>, which select one of the 256 entries in the system or process TB. All 256 entries are then accessed by using the adder to increment the value in MCA <8:2>. The page crossing flag is set to indicate the end of the TBIA (translation buffer invalidate all) function. MCA <31> selects the TB to be accessed: MCA <31>=1 selects the system TB; MCA <31>=0 selects the process TB. All other MCA bits (<30:09> and <1:0>) are ignored.

When <30:29> are 11, and the TB/cache RAM chip select and write enable bits are asserted (bits <33:32>), the cache is accessed for a conditional cache invalidate. If a cache hit occurred in the previous cycle, the hardware valid bit is written as asserted by bit <31> (the TB/cache valid control bit), and all other cache entry bits are written to an undefined state. If a cache hit did not occur in the previous cycle, this cycle is a no operation.

Adder Latch Enable

Bit <28> of each memory controller microinstruction is the adder latch enable bit, active low. This bit enables the adder register to latch the output of the 9-bit adder. (See Figure 6-1 for the location of the adder and register.)

Adder Subtract Enable

Bit <27> of each memory controller microinstruction is the adder subtract enable bit. This bit selects whether the 9-bit adder adds or subtracts, in effect.

If bit <27> is a zero, a value between 0 and +7 inclusive is supplied to the adder by bits <26:24> of the microinstruction; the adder adds this supplied value to MCA <8:0>. (The supplied 3-bit value is zero-extended to nine bits.)

If bit <27> is a one, a value between −1 and −8 inclusive is supplied to the adder by bits <26:24> of the microinstruction; the adder adds this supplied value to MCA <3:0>. MCA <8:4> are unchanged, and the page crossing flag is negated.

Adder Constant Select

Bits <26:24> of each memory controller microinstruction are the adder constant select bits; that is, they provide the 9-bit adder with a value between −8 and +7 (inclusive) to be added to the bits on the MCA bus. Only a 9-bit value is incremented; a carry into the tenth bit is flagged as the page crossing branch condition. Table 5-7 lists the effective values added to MCA <8:0> or MCA <3:0> for the various states of microinstruction bits <27> and <26:24>.

Table 5-7. Adder Control

<27>	<26:24>	Effective Value	Added To
0	111	+7	MCA <8:0>
0	110	+6	MCA <8:0>
0	101	+5	MCA <8:0>
0	100	+4	MCA <8:0>
0	011	+3	MCA <8:0>
0	010	+2	MCA <8:0>
0	001	+1	MCA <8:0>
0	000	0	MCA <8:0>
1	111	−1	MCA <3:0>
1	110	−2	MCA <3:0>
1	101	−3	MCA <3:0>
1	100	−4	MCA <3:0>
1	011	−5	MCA <3:0>
1	010	−6	MCA <3:0>
1	001	−7	MCA <3:0>
1	000	−8	MCA <3:0>

Register File Write Enables

Bits <23:22> of each memory controller microinstruction are the register file write enables, active low. These bits cause the data that are stable on the MCA bus to be written into the addressed location of the register file.

Bit <23> is the offset register file write enable; bit <22> is the page register file write enable.

When bit <23> is asserted, data on the MCA bus are written into the 9-bit offset portion of the register file. When bit <22> is asserted, data from the MCA bus are written into the page portion of the register file.

Additionally, if the register file address is 8 through F inclusive, and the offset write enable is asserted (<23>), the datum MCD <0> is also written into the selected control and status register (CSR).

Register File Address Field

Bits <21:18> of each memory controller microinstruction are the register file address field, active high. These bits specify a 4-bit register file address which defines an address space shared by the register file and the control and status registers (CSRs). The encoding is listed in Table 5-8.

Table 5-8. Register File Address Space

<21:18> (hex)	Register File Location	Content
00	0	virtual address
01	1	physical address
02	2	I-stream PC
03	3	error code
04	4	zero
05	5	unused
06	6	unused
07	7	unused
08	do not use	
09	do not use	
0A	do not use	
0B	do not use	
0C	map enable control register	

0D	cache enable control register
0E	error flag status register
0F	IB.ERROR status register

When a CSR is read, its register file address is specified in microinstruction bits <21:18> and its content enabled onto the memory controller data bus as MCD <0>. The microcode guarantees that a CSR read is specified for two adjacent microcycles, and that a dead cycle occurs on the MCD (that is, no source enabled) in the microcycle following the read.

Microprogram Control Field

The 18-bit microprogram control field provides the information for the MCT microsequencer to determine the address of the next MCT microinstruction. The generated microaddress is then used to access control store and retrieve the next MCT microinstruction.

The control store address space consists of 1,024 locations; each location contains a 64-bit microinstruction. A 10-bit microaddress is presented to the MCT control store to access the next microinstruction.

Figure 5-6 shows the microprogram control field divided into four subfields: busy control, microsequencer control, branch control, and next address.

The following sections describe these subfields in more detail.

Busy Control

17 16	15 13	12 10	9 0
busy control	microsequencer control field (MCF)	branch control (BCF)	next address field (NAF)

Figure 5-6. Microprogram Control Field

Bits <17:16> of each memory controller microinstruction are the busy control field bits. These bits determine the state of the MEM BUSY signal to the data path module. Together, these bits synchronize data transfers between the memory controller module and the data path module. The encoding is shown in Table 5-9.

Table 5-9. Busy Control Field Encoding

<17:16>	Function
00	unconditionally clear busy
01	conditionally clear busy
10	not used
11	no operation

Microsequencer Control

Bits <15:13> of each memory controller microinstruction are the microsequencer control field bits. These bits control the next microaddress to be executed by the MCT micromachine. The three bits are defined as follows:

<15>	enable trap
<14>	enable dispatch
<13>	return/not jump

where trap is defined as a cache invalidate trap;

dispatch means dispatch on the memory request supplied by the data path; jump is the microaddress created by bits <9:0> of the microinstruction, with bits <3:0> modified by branch conditions. The legal values for the microsequencer control field are shown in Table 5-10.

Branch Control

Bits <12:10> of each memory controller microinstruction are the branch control field bits. This field selects one of eight groups of status conditions that can be ORed with the four least significant bits of the next address field to cause conditional microprogram branches. The branch control field influences the next microaddress in the same way that the OR <2:0> field does in the data path microcode. Figure 5-6 shows the encodings for the branch control field, BCF <2:0>.

Next Address

Bits <9:0> of each memory controller microinstruction are the next address field. These bits specify the next microaddress that the microsequencer will execute if no conditional branching, dispatch, or trap occurs.

The four lowest bits, NAF <3:0>, can be modified by the branch conditions as selected by the branch control field. Figure 5-6 shows the branch control field and corresponding branch conditions. The result is that two, four, eight, or sixteen-way branching can occur.

When a two-way branch is coded in a memory controller microinstruction, one of the four lowest bits in the microinstruction is zero; when a four-way branch is coded, two of the four lowest bits in the microinstruction are zero; when an eight-way branch is coded, three of the four lowest bits in the microinstruction are zero; when a sixteen-way branch is coded, the low four bits are all zeros. The bits that are zeros can be ORed with selected branch conditions.

Branch Conditions

When a branch condition signal is asserted, it causes a branch to the address of the memory controller microroutine that handles that branch condition. There are sixteen branch conditions that can influence the address of the next MCT microinstruction; these branch conditions are listed in Figure 5-7, and described further in the following paragraphs.

Table 5-10. Microsequencer Control Field Encoding

<15:13>	Function
000	disable trap, disable dispatch, jump
001	not used
010	not used
011	not used
100	enable trap, disable dispatch, jump
101	enable trap, disable dispatch, return from trap
110	enable trap, enable dispatch, jump
111	not used

12	11	10	09 08 07 06 05 04	03	02	01	00
0	0	0	<9:4>	NO.MAP	DATAFLOW	MCA <1>	MCA <0>
0	0	1	<9:4>	PAGE.CROSS	MODIFY	MCA <1>	MCA <0>
0	1	0	<9:4>	0	QBUS.BLOCK	QBUS.SYNCH	QBUS.BUSY
0	1	1	<9:4>	0	0	TB.ERROR	MCA <29 or 28>
1	0	0	<9:4>	0	0	0	0
1	0	1	<9:4>	0	0	QBUS.TIMEOUT	QBUS.ERROR
1	1	0	<9:4>	0	0	0	TBC.MISS
1	1	1	<9:4>	0	0	PREFETCH.DIS	IB.ERROR

Figure 5-7. Branch Control Field and Next Address Field Formats

NO.MAP

When this signal is asserted, memory management is turned off; that is, no address translation takes place so all addresses are treated as physical addresses, and no access checking is performed so there is no memory protection.

The map enable bit is one bit in an internal processor register (IPR) on the data path chip; this bit is set and cleared by executing a MTPR macro-instruction. The data path then places a copy of the map enable bit in the memory controller map enable control register (one of the CSRs in the register file) by executing a Memory Request with the function WRITE.MCT; the address of the CSR is specified as part of the function code.

DATAFLOW

This branch condition signal is asserted directly from bit <28> in the data path microinstruction that is making the memory request. If bit <28> is a zero, the requested memory operation is a read; a one indicates a write.

This bit is latched in the memory function latch on the data path module, and transmitted to the memory controller as BUS MEM CTL 5. On the memory controller module, this signal is one of the inputs to the branch MUX.

MCA <1> and MCA <0>

These bits are the low two bits of the virtual or physical address currently on the MCA bus. They are used to indicate what data alignment must be performed.

PAGE.CROSS

If the adder is enabled and the add operation results in a carry into the tenth bit, the page crossing signal is asserted.

MODIFY

This branch condition signal is asserted directly from bit <29> in the data path microinstruction that is making the memory request. If bit <29> is a zero, the access intent is read; a one indicates an access intent of write.

The modify intent signal (DAPT MODIFY H) is transmitted from DAP to MCT over pin DP2 in the backplane. Once on the memory controller module, this signal is one of the inputs to the branch MUX.

QBUS.BLOCK

The Q22 bus controller sends this signal to the memory controller branch condition logic when the physical memory on the Q22 bus supports block mode; that is, data can be read or written in blocks of 1 to 16 words (each word is 16 bits) within one Q22 bus cycle. The words are transferred one at a time but the Q22 bus controller only needs to drive one address onto the bus at the beginning of the operation, and the cycle does not end until the desired number of words are transferred.

If the physical memory does not support block mode, the Q22 bus controller must drive a new address onto the Q22 bus for each word to be read or written.

QBUS.SYNCH

The Q22 bus controller sends this signal to the memory controller branch condition logic to indicate Q22 bus controller status.

On a read operation, SYNC/READY means that the requested data are available and can be driven onto the memory controller MCD bus. On a write, SYNC/READY means that the Q22 bus controller is ready to receive the write-to address or the data to be written.

QBUS.BUSY

The Q22 bus controller sends this signal to the memory controller branch condition logic when a Q22 bus function is in progress. This signal informs the memory controller that the Q22 bus controller is busy and not yet ready to receive the next function request.

TB.ERROR

This signal is asserted when anything goes wrong during a translation operation. It is actually the OR of the signals that indicate a page crossing, a TB miss, an access violation, or modify refused.

MCA <29 or 28>

Bit <29> is the high-order bit of the physical address currently on the MCA bus. Seahorse physical addresses are 23 bits long, with the address specified in <22:00> and the I/O space flag, bit <29>, appended as bit <23>. Bit <29> becomes part of the physical address in the address translation procedure.

When bit <23> of a physical address is a one, that address is located in I/O space, and is therefore not in the cache. (No I/O space address is cached.)

Bit <28> is one of the fifteen bits latched in the physical address register after an address translation operation. When set, it indicates that the address is located in a physical memory that can be shared by Q22 bus processors, and therefore, the address should not be cached. Although bit <28> is driven on the MCA bus as MCA <28>, it is an internal flag and is not sent over the Q22 bus as part of the physical address.

Thus, if either bit <29> or bit <28> is set, the address should not be cached, and the branch condition input signal NON CACHE REF H is generated.

QBUS.TIMEOUT

The Q22 bus controller sends this signal to the memory controller branch condition logic when a Q22 bus device does not reply within the allowed time limit of 10 microseconds. This signal generally means that a read or write to a nonexistent memory location was attempted.

QBUS.ERROR

The Q22 bus controller sends this signal to the memory controller branch condition logic when either a timeout or a parity error occurs on the Q22 bus. This signal causes the current memory request to be aborted.

TBC.MISS

This signal is asserted when an address translation or cache access fails. It is not available until the cycle after the one in which the actual translation or cache access took place.

PREFETCH.DIS

This is the disable prefetch signal. It is asserted when the prefetch FIFO is full. It is deasserted when the FIFO content falls below the target value of eight bytes. This deassertion causes the memory controller to reload the prefetch FIFO from the I-stream.

IB.ERROR

When a page crossing occurs during prefetch, a bit is set in the IB.ERROR status register (one of the CSRs). The IB.ERROR signal is then asserted to the memory controller branch condition logic. The assertion of this signal causes the prefetch operation to halt until the correct page address can be supplied by the data path module.

Q22 Bus Controller Interface

Just as the memory controller module executes commands delivered by the data path, the Q22 bus controller executes commands delivered by the memory controller. The memory controller microcode communicates with the Q22 bus controller via six microcode bits: three control signals, and three bits of function code. The Q22 bus controller sends back six status flags to communicate its state to the memory controller microcode. All of the microcode bits and most of the status flags are described earlier in this chapter, but are also summarized here for convenience.

Interface Microcode

The memory controller sends six microcode bits to the Q22 bus controller. They are:

- the function request bit, which is microinstruction bit <63>. If this bit is a one, the Q22 bus controller latches the function code specified in microinstruction bits <47:45> and arbitrates for the bus.
- the clear function code enable bit, which is microinstruction bit <49>. The Q22 bus controller will execute the function specified by the last function code it received until bit <49> (active low) is asserted.
- the go bit, which is microinstruction bit <48>. After the Q22 bus controller receives bit <63> and the function code in bits <47:45>, it acquires the bus and asserts an address. It then waits for the go bit before proceeding with the bus cycle.
- the function code, microinstruction bits <47:45>. These bits select the Q22 bus operation to be performed: no operation, write word, write byte, write block, read word, read block, read interrupt vector, or read interlocked. These Q22 bus operations are described in Chapter 7. See Table 5-2 for the encoding of the function code field.

The Q22 bus controller has its own control store and microsequencer. The three bit function code and the function request bit from the memory controller microinstruction are latched in the Q22 bus function register (part of the Q22 bus controller). The three function code bits are then used to address the Q22 bus controller control store and the Q22 bus controller microcode takes over to carry out the function requested by the memory controller.

Q22 Bus Controller Status

The Q22 bus controller communicates its state to the memory controller through six status flags. They are:

- QBUS.BLOCK. This signal is asserted during a write block or read block operation to signify that the memory will be able to handle the next data transfer as a block mode transfer.
- QBUS.SYNCH. The signal SYNC/READY is asserted when data are available on a read from a Q22 bus device, and when data or address is needed for a write to a Q22 bus device.
- QBUS.BUSY. This signal is asserted when a Q22 bus function requested by the memory controller is in progress. As long as this signal is asserted, the memory controller cannot start a new request.
- QBUS.TIMEOUT. This signal is asserted when the address of a nonexistent memory location is driven on the bus.
- QBUS.ERROR. This signal is the OR of the two error signals, bus timeout and parity error. It causes the current memory request to be aborted.
- Cache Invalidate. This signal is asserted whenever a bus device writes to physical memory. It alerts the memory controller to

invalidate its copy of the written-to memory location if that address is in the cache.

This chapter describes the memory controller microcode and the memory controller/Q22 bus controller interface. The next chapter describes the hardware that implements the memory controller microcode.

Chapter 6 Memory Controller Module

This chapter is a detailed description of the components on the memory controller module and how they interact. First, the major logic elements and their hardware components are described. Then, the basic transfers of data between the logic elements are described on a microprogram level.

Overview of MCT Functions

The memory controller module contains hardware to perform the following eight functions:

- generate clock signals
- control MCT microinstruction flow
- translate virtual addresses
- access the data cache
- transfer data within the memory controller module
- prefetch instruction stream bytes
- track and report status
- communicate with the Q22 bus controller to read and write data

The next eight sections describe these functions, and the hardware components that implement them, in detail. The hardware components are illustrated in the MCT block diagram, Figure 6-1.

Generating the Clock Signals

All of the clocks for the Seahorse CPU are generated from a single 64 MHz clock on the memory controller module. The clock generator logic produces clocks for the data path module, the memory controller module, and the Q22 bus interface. The major clocks for the memory controller module are described in the next section.

MCT Clocks

The master clock on the memory controller module is a 16 MHz clock MCTM BASE CLK (62.5 ns period). This clock signal goes to the data path module over backplane pin CN2, and is the source for all the clock signals on the data path module. BASE CLK also synchronizes DAPL DCOK L to generate the DAPL INIT signals. One of the DAPL INIT signals is DAPL MCT INIT L. DAPL MCT INIT L initializes the memory controller module to a known state and synchronizes the clock signals between the DAP and MCT modules.

MCTM CLK125 is the 125 ns period clock that controls the memory controller microcycle. Thus, two memory controller microcycles occur for every one data path microcycle. (DAPL CPU CLOCK is the 250 ns period clock that controls the data path microcycle.)

MCTM CLK62 simply divides the CLK125 signal in half, providing clocking control for each half of a memory controller microcycle.

MCTM MEMCLK is asserted for the first 31 ns of a microcycle, and deasserted for the remaining 94 ns. This clock signal controls TB and cache reads.

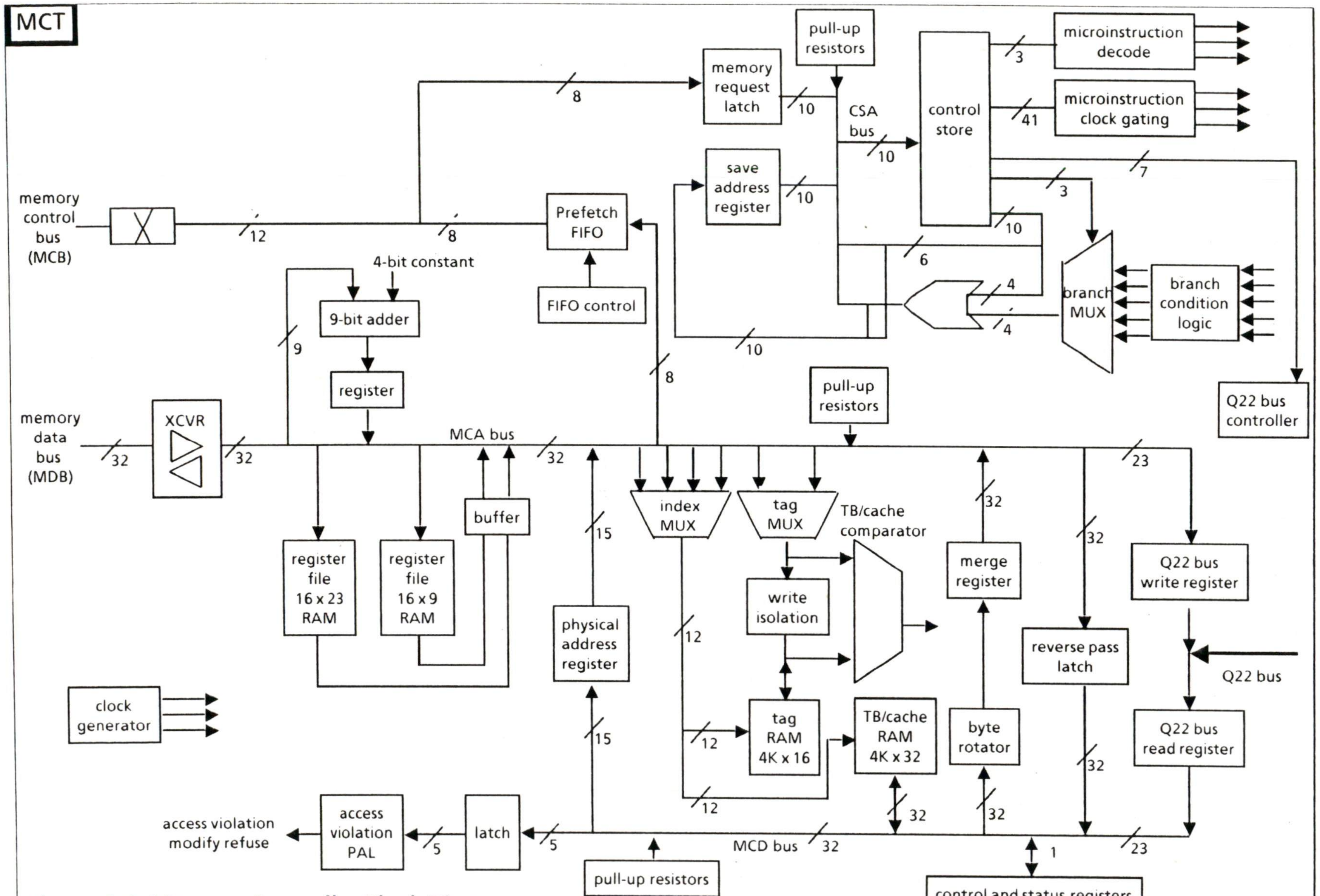


Figure 6-1. Memory Controller Block Diagram

MCTM DPC SRC L is sent to the DAP module over backplane pin CP1. This clock signal is inverted to generate the signal DAPL DPC CLK H. DPC CLK H is the input clock signal to the data path chip.

MCTM DLYD CLK125 delays the CLK125 signal by 31 ns. This delayed clock enables writes to the register file and latches the reverse pass latch.

MCTM ADV CLK125 advances the CLK125 signal by 15 ns. This advanced clock is used to clock or latch data into MCA bus destinations.

Timing

A memory controller microcycle begins at the rising edge of MCTM CLK125 when the MCT microinstruction is available as the output of the control store. At this point, the microinstruction clock gating logic takes over to distribute the microinstruction control bits to their respective functional blocks.

Branch conditions are available to the MCT microsequencer by 50 ns into the current microcycle.

For reads from the translation buffer or the cache, the access address becomes available on the MCA bus between 0 and 31 ns into a microcycle, and the read occurs between 31 and 125 ns.

All MCA bus destinations are written by 110 ns into the current microcycle. If the bus destination is a latch part, the latch is open between 47 ns and 110 ns. If the bus destination is an edge sensitive part, data are clocked into the destination at 110 ns, which is the rising edge of MCTM ADV CLK125 H.

Controlling the MCT Microinstruction Flow

The memory controller module has its own set of hardware components to sequence the flow of memory controller microinstructions. These components are the memory request latch, the CSA bus, pull-up resistors, the control store, microinstruction clock gating, branch condition logic, and the microsequencer. The following paragraphs describe each of these components in turn.

Memory Request Latch

When the data path module executes a Memory Request or I-stream Request microinstruction, eight bits of control information are sent to the memory controller over the memory control bus, and four additional bits are sent over the backplane. These twelve bits are recombined on the memory controller module and the following eight bits are latched in the memory request latch:

- the second part flag, bit <7>
- the data type, bits <6:5>
- the memory function code, bits <4:0>.

The CSA PAL in the MCT microsequencer supplies ones for bits <9:8>.

The memory request latch is a tri-state latch located at the memory controller end of the memory control bus. The signal DAPR MEM REQUEST H is asserted by the data path to inform the memory controller when a new memory function code is on the memory control bus. This signal is synchronized with two clock signals to generate the signal MCTN MRL LE H (memory request latch latch enable). This signal causes the proper eight bits to be latched into the memory request latch. If bit <14> in the current MCT microinstruction is set, enabling dispatches, the contents of the memory request latch, plus the two high-order ones from the microsequencer, are presented to the control store as the next microaddress.

CSA Bus

The control store address bus conveys the next microaddress to be executed to the control store. The CSA is a tri-state bus passively asserted by pull-up resistors.

The CSA bus is driven by one of the following four sources: the next address buffer, the save address register, the memory request latch, or the pull-up resistors.

The CSA bus destination is the address inputs to the control store.

Pull-up Resistors

When no other source is driving the bus, the pull-up resistors cause the default condition on the bus to be a logical high; that is, they pull up the bus. Thus, CSA bus bits <7:0> are all ones when the next address buffer, the save address register, and the memory request latch are not enabled.

MCT Control Store

The control store for the memory controller microinstructions is 1K deep by 64 bits wide. Each of the 1K locations contains one MCT microinstruction.

Only 60 of the 64 bits are used.

The input to the control store is a 10-bit microaddress, which selects one location, and therefore one microinstruction.

The output from the control store is a 64-bit microinstruction that controls the MCT microsequencer, the Q22 bus interface, and all functional blocks of the memory controller. The encoding of the microinstruction bits is detailed in Chapter 5, but basically:

- the eight high-order bits control the MCT components that source the MCA bus, with the exception of bit <63> which is the Q22 bus function request bit. The MCA bus sources are the adder, the register file, the MDB transceiver, the PAR, and the merge register.
- microinstruction bits <51:50> and <42:18> are the clocking and output enables for every functional block in the memory controller.
- bit <63> and bits <49:43> control the Q22 bus interface.
- bits <17:0> control the memory controller microsequencer.

Microinstruction Clock Gating

This block of logic uses 41 of the 64 microinstruction bits from control store as input, and gates the appropriate latch and output enables to the proper functional blocks. Thus, the microinstruction clock gating logic controls when the various bus sources are enabled onto the MCA and MCD buses.

Branch Condition Logic

The branch condition logic monitors a group of conditions that affect the MCT status as reported to the data path, and can affect the MCT microprogram flow. Part of the logic is a flip-flop that acts as a pipeline to save status for one additional cycle before it is discarded. The signals saved are:

- MCTL TBC HIT, which indicates a translation buffer or cache hit,
- MCTE INCR PGXR, a signal from the 9-bit adder which indicates a page crossing has occurred,
- MCTN DONE, which is asserted when the MCT has finished processing the memory request and is ready to return data or status to the data path,
- MCTS EN ACC CHECK, which is asserted when the translation buffer has been accessed; after a TB access, the cache is accessed next and therefore the access check logic is enabled, and
- DAPR MEM REQUEST, which is the signal from the data path indicating a new memory function code is on the memory control bus.

The branch condition logic sends these three signals to the branch MUX in the MCT micro-sequencer:

- MCTT MAP ENB, which indicates when memory mangement is enabled and sets up the branch condition NO.MAP,
- MCTT PAGE CROSSING, which indicates when a carry into the tenth bit has occurred in the 9-bit adder, and sets up the branch

condition PAGE.CROSS, and

- MCTT IB ERROR, which indicates to the data path that the MCT cannot supply the next instruction stream byte because a page crossing has occurred, and sets up the branch condition IB.ERROR.

MCT Microsequencer

The memory controller microsequencer generates the next 10-bit microaddress every 125 ns. It provides conditional branching based on MCT internal and external conditions. Branch conditions are accepted no later than 75 ns before the next clock edge.

The MCT microsequencer consists of these components: the microinstruction decode logic, CSA PAL, the branch MUX, save address register, and next address buffer. These components are described in the following paragraphs. Figure 6-2 is a block diagram of the MCT microsequencer.

Microinstruction Decode Logic

The 10-bit microaddress used to access the control store comes from one of the following sources: the memory request latch, the save address register, or the next address buffer. The next microaddress can also be 3FF which is the address of the first microinstruction in the trap microroutine. The microinstruction decode logic selects which of these sources provides the next microaddress.

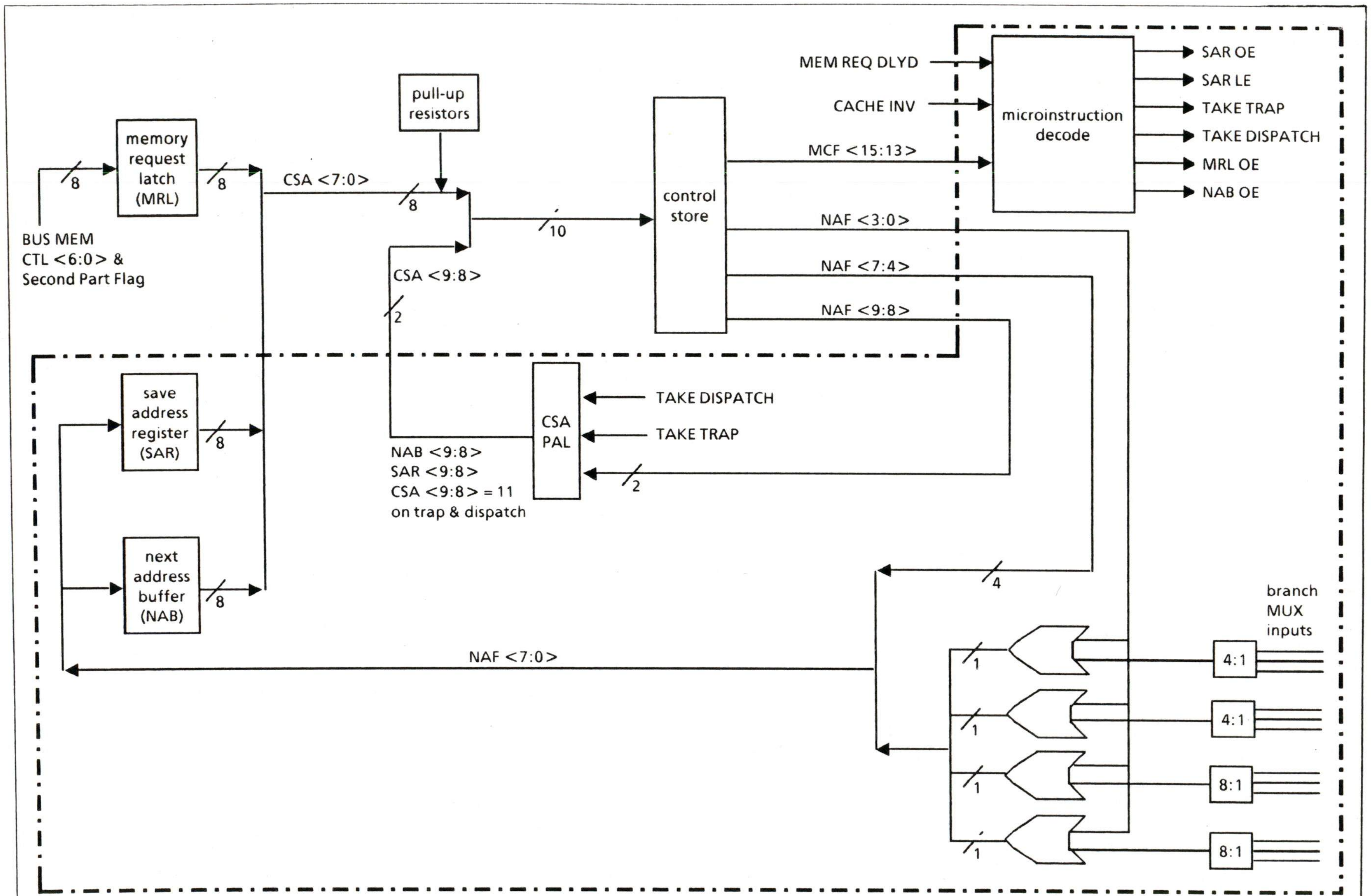


Figure 6-2. MCT Microsequencer Block Diagram

There are basically three inputs to this decode logic: microinstruction bits <15:13> (the microseqencer control field in Figure 5-5), the signal MCTT MEM REQ DLYD which is generated from a pipelined version of DAPR MEM REQUEST H, and the signal MCTB CACHE INV H which is a signal from the Q22 bus.

Microinstruction bit <15> enables traps, bit <14> enables dispatches on memory requests from the data path, and bit <13> enables returns from traps. MCTT MEM REQ DLYD is asserted when the data path drives a new memory function code onto the memory control bus. MCTB CACHE INV H is asserted when a Q22 bus device writes to physical memory.

When bit <14> and MEM REQ DLYD are both asserted, the microinstruction decode logic asserts the signals MCTJ TAKE DISPATCH, and MCTJ MRL OE (memory request latch output enable). When MCTJ MRL OE is asserted, the eight memory function bits in the memory request latch are presented to the control store as the low-order eight bits of the next microaddress. The high-order two bits are both ones, and they are driven onto the CSA bus by the CSA PAL in the microsequencer whenever the signal TAKE DISPATCH is asserted.

When bit <15> and CACHE INV H are both asserted, the microinstruction decode logic asserts the signals MCTJ TAKE TRAP and MCTJ SAR LE, and the pull-up resistors control the bus. The pull-up resistors force next microaddress bits <7:0> to ones. TAKE TRAP also causes the CSA PAL in the microsequencer to force the high-order two bits of the next microaddress to ones.

Therefore, the next microaddress is 3FF. The signal MCTJ SAR LE enables the save address register (SAR). So when a trap is taken, the low eight bits of the microaddress that would have been presented to control store next if the trap had not occurred, are saved in the SAR. The two high-order bits are saved in the CSA PAL.

Bit <13> is asserted when a return from a trap is needed. For example, the last microinstruction in the trap microroutine located at 3FF has bit <13> set. The signal SAR OE is asserted as the output of the microinstruction decode logic when bit <13> is set. SAR OE enables the contents of the save address register onto the CSA bus to be presented to the control store as the low eight bits of the next microaddress. Bit <13> set also causes the CSA PAL to drive the two high-order bits that it saved when the trap was taken, onto the CSA bus. Thus, a return from trap is executed.

When bit <13> is a zero, and traps and dispatches are either not enabled or don't occur, the next microaddress is bits <9:0> of the current microinstruction from control store. If any branch conditions are in effect, they are ORed with the low four bits <3:0>. This microaddress, modified as appropriate, is passed through the next address buffer and presented to control store as the next microaddress.

CSA PAL

The next address buffer, save address register, and memory request latch drive microaddress bits <7:0> onto the CSA bus. The CSA PAL provides the two high-order bits, MCTN CSA <09:08>.

On a cache invalidate trap or memory request dispatch, the CSA PAL forces bits <09:08> to ones.

When a trap is taken, the two high-order bits of what would have been the next microaddress are saved in the CSA PAL. On a return from trap, the CSA PAL drives these saved bits onto the CSA bus. The low eight bits are driven onto the CSA bus from the save address register, which is enabled by the microinstruction decode logic.

For a normal operation where the next microaddress is provided by bits <9:0> of the current microinstruction, bits <9:8> are driven onto the CSA bus from the CSA PAL. Bits <7:0> are provided by the next address buffer; the low four bits are modified by any asserted branch conditions.

Save Address Register

When a trap occurs, the save address register stores microaddress bits <7:0> of the microinstruction that would have executed next. The signal MCTJ SAR LE asserted by the microinstruction decode logic causes the save address register to latch the microaddress bits.

When a return from trap is executed, the microinstruction decode logic asserts the signal MCTJ SAR OE to enable the save address register to drive the saved bits onto the CSA bus.

Next Address Buffer

The next address buffer passes bits <7:4> of the current microinstruction, and bits <3:0> of the current microinstruction after they have passed through the branch MUX and been modified by any asserted branch conditions, to the control store. These are the low eight bits of the next microaddress if a trap or a dispatch does not occur.

When trapping and dispatching are not enabled or do not occur, the microinstruction decode logic generates the signal MCTJ NAB OE which causes next microaddress bits <7:0> to be passed through the next address buffer and presented to control store.

Branch MUX

The branch MUX consists of two 4:1 multiplexers, two 8:1 multiplexers, and four OR gates. The branch conditions described in Chapter 5 and listed in Figure 5-7 are the inputs to the branch MUX.

Each 4:1 and 8:1 MUX selects one branch condition that is ORed with one of the low four bits of the microinstruction from the control store. The ORed signals are then passed through the next address buffer and become the low four bits of the next microaddress if the next address buffer is enabled by the microinstruction decode logic.

Translating Virtual Addresses

One of the main functions of the memory controller is to translate virtual addresses supplied by the data path module into physical addresses. The components used to do this are the index MUX, the tag MUX, the tag RAM, the TB/cache RAM, the write isolation buffer, the TB/cache comparator, the physical address register, the register file, and the 9-bit adder. The following paragraphs describe each of these components in turn, and the different kinds of TB accesses.

Index MUX

The index MUX selects the correct bits from the MCA bus to access the TB location that is to be read, written or invalidated. The twelve bits selected from the MCA bus by the index MUX form an address that accesses a location in the tag RAM; the same address is used to access a location in the translation buffer.

If a single translation buffer entry is being read, written, or invalidated, the index MUX selects virtual address bits <31> and <16:9> off the MCA bus. The index MUX supplies zeros for the high-order three bits, thus selecting the translation buffer portion of the tag RAM and of the TB/cache RAM. The address then presented to the tag RAM and to the TB/cache RAM by the index MUX is:

11	10	9	8	7 0
0	0	0	VA <31>	VA <16:9>

If VA <31> is a zero, the presented address selects a process space tag entry and translation buffer entry. If VA <31> is a one, the presented address selects a system space tag entry and translation buffer entry.

If the entire translation buffer is being invalidated by a Memory Request microinstruction with the INVALID.MULTIPLE function code, the index MUX selects virtual address bits <31> and <8:1> off the MCA bus. The index MUX again supplies zeros for the high-order three bits. The address then presented to the tag RAM and to the translation buffer by the index MUX is:

11	10	9	8	7 0
0	0	0	VA <31>	VA <8:1>

Tag MUX

The tag MUX selects virtual address bits <30:17> from the MCA bus to form the low-order fourteen tag bits for the translation buffer entry being read, written or invalidated.

When the data path issues a Memory Request microinstruction with a function code that requires a translation buffer read or invalidate, the tag MUX passes bits <30:17> of the virtual address on the MCA bus to the TB/cache comparator.

When the data path issues a Memory Request microinstruction with a function code that requires a translation buffer write, the tag MUX passes bits <30:17> of the virtual address on the MCA bus to the write isolation buffer, which in turn passes them to the tag RAM, where they are written into the location accessed by the index MUX.

Tag RAM

The tag RAM is a 4K locations by 16-bit-wide memory array that stores the address tag bits associated with each translation buffer and cache entry. The first 512 locations in the tag RAM contain the translation buffer tags for process and system space, the next 1.5K locations are unused, and the last 2K locations contain the translation buffer tags for the data and instruction cache. Figure 6-3 shows the organization of the tag RAM.

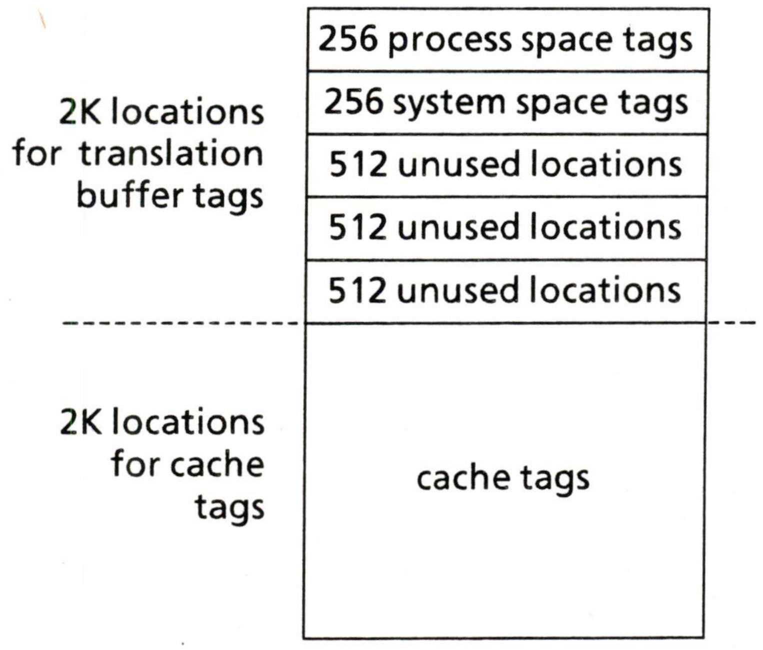


Figure 6-3. Organization of Tag RAM

Each translation buffer tag is 16 bits wide, consisting of one valid bit controlled by bit <31> of every memory controller microinstruction, one spare bit, and 14 tag bits which are virtual address bits <30:17>. Figure 6-4 shows the organization of one tag entry in the translation buffer:

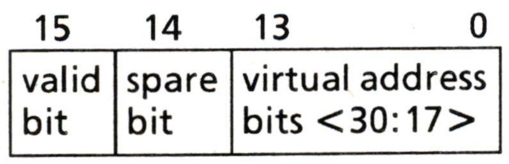


Figure 6-4. Translation Buffer Tag

The tag RAM is written whenever the TB/cache RAM is written. When a new TB tag is written into the tag RAM, bits <30:17> of the virtual address on the MCA bus are stripped off by the tag MUX, passed through the write isolation buffer, and written into the tag RAM location selected by the index MUX. Tag bit <15>, the tag valid bit, is set or cleared by the memory controller microinstruction executing the write to the translation buffer, to mark the entry as valid or invalid.

The tag RAM is read whenever the TB/cache RAM is read. For a read, the tag MUX passes bits <30:17> of the virtual address on the MCA bus to the TB/cache comparator. The tag at the tag RAM location selected by the index MUX is also sent to the comparator. If tag bit <15> is clear, indicating an invalid tag, the comparator generates a TB miss indication. If tag bit <15> is set and tag bits <13:0> match virtual address bits <30:17> stripped off the MCA bus by the tag MUX, the comparator generates the signal MCTL TBC HIT.

When a Memory Request microinstruction from the data path requests a translation buffer invalidate function, the tag MUX selects virtual address bits <30:17> from the MCA bus and passes them through the write isolation buffer to the tag RAM, where they are written into the tag RAM location accessed by the index MUX bits. As the tag is written, the valid bit is cleared marking the tag invalid.

TB/Cache RAM

The TB/cache RAM is a 4K locations by 32-bit wide static memory. It is organized the same way the tag RAM is organized, with the lower 2K locations containing process and system space page table entries (PTEs), and the upper 2K locations containing data and instruction cache entries. Figure 6-5 shows the organization of the TB/cache RAM.

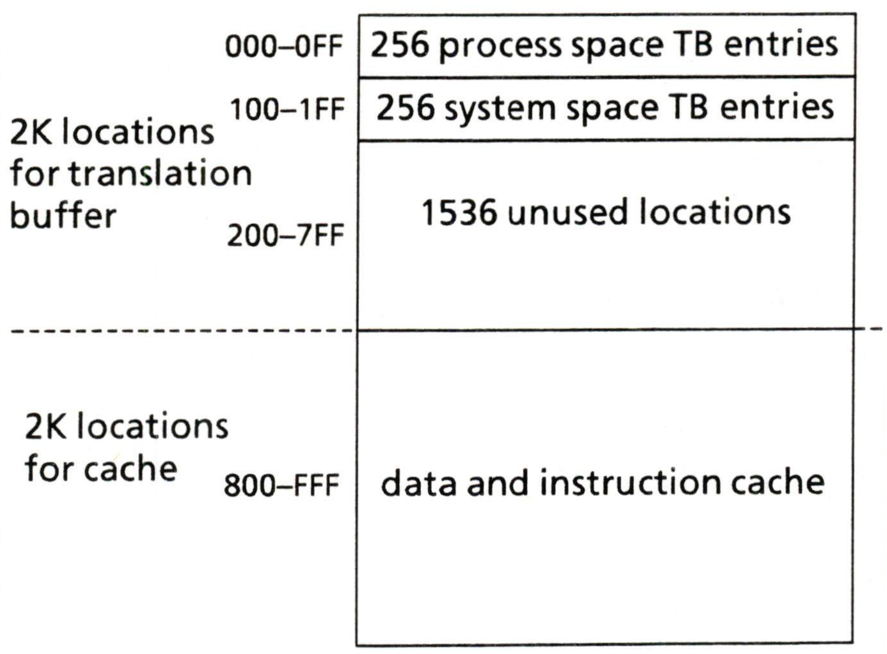

000–0FF
256 process space TB entries
100–1FF
256 system space TB entries
2K locations for translation buffer
200–7FF
1536 unused locations
2K locations for cache
800–FFF
data and instruction cache

Figure 6-5. Organization of TB/cache RAM

Each translation buffer location contains a PTE. Figure 6-6 shows the organization of one page table entry in the translation buffer:

PTEs are read from or written to the translation buffer from the MCD bus when a Memory Request from the data path specifies a function requiring a translation buffer access. Bits <33:32> of the

31	30 27	26	25 21	20 0
valid bit	protection field	modify bit	reserved	page frame number

Figure 6-6. Translation Buffer PTE

MCT microinstruction that is executed to implement the requested function, determine whether the current TB access is a read or a write. (See Table 5-5 for the encoding of these bits). Bits <30:29> of the MCT microinstruction determine whether the current TB access is a normal access (that is, a read or a write), or a translation buffer invalidate. (See Table 5-6 for the encoding of these bits).

The TB/cache RAM address selected by the index MUX bits on a TB access is the same address selected in the tag RAM by these same index MUX bits. For example, if the address presented to the tag RAM by the index MUX is OCD (hex), then location OCD in the TB/cache RAM is accessed at the same time.

Write Isolation Buffer

This 16-bit-wide tri-state isolator allows the tag portion of a TB entry to be written into the tag RAM during a translation buffer write access or a translation buffer invalidate operation. For a read from the translation buffer, the write isolation buffer is disabled to allow the tag from the tag RAM to be read to the comparator.

TB/Cache Comparator

The TB/cache comparator is a 16-bit comparator that asserts a TB/cache hit indication for a translation buffer read when the stored address tag from the tag RAM equals the search address tag supplied by the tag MUX. When the comparison results in a match, the comparator asserts the signal MCTL TBC HIT H. If this signal is not asserted, the compared tags did not match.

Physical Address Register

The physical address register is hard-wired to form the physical page address from the page frame number (PFN) of the PTE read during a translation buffer access.

A total of fifteen bits are latched in the physical address register (PAR) following an address translation. Bits <12:0> of the PTE PFN are latched as the low-order thirteen bits of the PAR; these bits form physical address bits <21:09> when driven onto the MCA bus. Bit <19> from the PTE is the next highest-order bit in the PAR; it is driven onto the MCA bus as MCA <28>. This bit indicates whether the address is located in shared memory. The logical AND of PFN <20> and the TB/cache hit signal is latched as the high-order bit in the PAR; this bit forms physical address bit <29> when driven onto the MCA bus. When <29> is a one, the physical address is located in MicroVAX I/O space.

When the PAR output enable bit (bit <61>) is a zero in the currently executing MCT microinstruction, the PAR contents are driven onto the MCA bus as MCA <29:28> and MCA <21:09>.

Register File

The register file is a 16-location by 32-bit-wide block of general storage within the memory controller. It is organized into a 9-bit-wide offset address portion and a 23-bit-wide (virtual or physical) page address portion; that is, the 23 bits select the memory page, and the 9 bits select the byte offset within the memory page. The register file address space is shared with the control and status registers (CSRs). Table 5-8 shows the register file address space.

Virtual addresses are stored at register file address 0, which is the first of the sixteen register file locations, and physical addresses are stored at address 1, the second location. (These addresses are assigned by the microprogram, not fixed by the hardware.) Virtual and physical addresses can be sourced onto the MCA bus from the register file. The third location stores the instruction stream program counter, which always points to the last instruction stream byte that was stored in the prefetch FIFO. Location four stores error codes, location five is zero, and the next seven locations are not used. The last four locations contain the CSRs.

The register file is written from the MCA bus when the register file write enable bits (<23:22>) in the current MCT microinstruction are asserted; microinstruction bits <21:18> determine the register file location that is written.

The contents of the addressed location in the register file are driven on the MCA bus when the register file output enable bits (<59:58>) in the current MCT microinstruction are asserted; microinstruction bits <21:18> determine the register file location that is addressed. A buffer/isolator passes the output from the register file to the MCA bus. Like the register file, the buffer/isolator is divided into a 23-bit page portion, and a 9-bit offset portion.

Adder and Adder Register

The 9-bit adder contains page-crossing detection, and has a tri-state register associated with it. The adder and register provide 9-bit counts by successive increments of −8 through +7 inclusive, and the means for modifying virtual or physical page offset addresses. The adder and register are controlled by six bits in the MCT microinstruction: the three adder constant select bits (<26:24>), the adder subtract enable bit (<27>), the adder output enable (<60>), and one latch enable bit (<28>) for the register.

The source for the adder is always address data bits <8:0> from the MCA bus. The output from the adder is the modified address data bits <8:0> which are driven onto the MCA bus.

The adder is used for generating the multiple memory addresses required by unaligned data accesses, and for probing the last bytes of word and longword data types to insure access privilege for the entire datum during writes. For translation buffer accesses, the adder provides successive TB entry addresses when the entire translation buffer must be invalidated.

Translation Buffer Operations

Each translation buffer entry can be read, written, or invalidated. The following paragraphs describe how these accesses are accomplished.

Address Sources

Virtual addresses for translation buffer operations can come from any of the following: the data path, the register file, or the register file modified by the 9-bit adder.

TB Reads

For a TB read operation, the selected PTE is driven onto the MCD bus and fourteen of the page frame number (PFN) bits are latched into the physical address register to form physical address bits <21:09>. The contents of the PAR are then driven onto the MCA bus and to the data path via the memory data bus. The TB hit or miss status is available to the data path the same cycle and the data on the memory data bus are used or ignored accordingly.

TB Writes

For a TB write operation, the PTE is written into the addressed location in the translation buffer from the MCD bus. Meanwhile, the corresponding tag bits are selected from the virtual address on the MCA bus by the tag MUX and written into the tag RAM at the same address that the PTE is written into in the TB/cache RAM.

TB Invalidates

The translation buffer can also be accessed to invalidate a single entry, or to invalidate the entire buffer. A TB invalidate function (single or multiple) writes the valid bit, bit <15>, invalid. The remaining tag bits and the associated PTE in the translation buffer are undefined.

For an invalidate multiple operation, the index MUX selects bits <8:1> from the virtual address on the MCA bus because the incremented addresses are supplied by the 9-bit adder. An INVALID.MULTIPLE Memory Request invalidates eight TB entries per request; this allows interrupts to be processed in between the series of INVALID.MULTIPLE Memory Requests required to invalidate the entire translation buffer. The first INVALID.MULTIPLE request specifies a virtual address that causes the index MUX to select the first location in the tag RAM (and in the translation buffer). The first eight tags in the tag RAM are then invalidated by clearing bit <15> in each tag by sequential TB writes. The MCT microcode enables the 9-bit adder and supplies it with a constant, and from that point on, the adder provides the low-order nine bits of the virtual address on the MCA bus.

Thus, by successive Invalidate Multiple requests, all of the TB locations are marked invalid.

Accessing the Cache

Another main function of the memory controller is to supply the data path with the requested data. The memory controller accesses the data and instruction cache first to try to supply the requested data. If the data is not in the cache, the memory controller initiates a Q22 bus cycle.

A cache access uses much of the same hardware as an address translation operation: the index MUX, the tag MUX, the tag RAM, the TB/cache RAM, the write isolation buffer, and the TB/cache comparator. The following paragraphs describe each of these components in turn, and the different kinds of cache accesses.

Index MUX

The index MUX selects the correct bits from the MCA bus to access the location in the cache that is to be read, written or invalidated. The twelve bits selected from the MCA bus by the index MUX form an address that accesses a location in the cache portion of the tag RAM, and the same location in the cache.

For any cache access, the index MUX selects physical address bits <12:2> off the MCA bus. The index MUX supplies a one for the high-order bit. The address then presented to the tag RAM and to the cache by the index MUX is:

11	10 0
1	PA <12:2>

Tag MUX

The tag MUX selects physical address bits <21:13> from the MCA bus and supplies zeros for the five high-order bits to form the low-order fourteen tag bits for the cache entry being read, written or invalidated.

When the cache is accessed for a read or an invalidate operation, the tag MUX passes bits <21:13> of the physical address on the MCA bus, plus the five high-order zeros, to the comparator.

For a cache write, the tag MUX passes bits <21:13> of the physical address on the MCA bus, plus the five high-order zeros, to the write isolation buffer. The write isolation buffer in turn passes them onto the cache portion of the tag RAM, where they are written into the location accessed by the index MUX.

Tag RAM

The tag RAM stores the address tag bits associated with each translation buffer and cache entry. The first 2K locations in the tag RAM contain the tags for process and system space addresses, and the last 2K locations contain the tags for the data and instruction cache. Figure 6-3 shows the organization of the tag RAM.

Each cache tag is sixteen bits wide, consisting of one valid bit controlled by bit <31> of every memory controller microinstruction, one spare bit, five zeros, and nine tag bits which are physical address bits <21:13>. Figure 6-7 shows the organization of one tag entry in the cache portion of the tag RAM:

15	14	13 9	8 0
valid bit	spare bit	00000	physical address bits <21:13>

Figure 6-7. Cache Tag

The tag RAM is written whenever the TB/cache RAM is written. When a new tag is written into the cache portion of the tag RAM, the tag is assembled by the tag MUX, passed through the write isolation buffer, and written into the tag RAM location selected by the index MUX. Tag bit <15> is set or cleared by the memory controller microinstruction executing the write to the cache.

The tag RAM is read whenever the TB/cache RAM is read. For a read, the cache tag assembled by the tag MUX is passed to the comparator. The cache tag at the tag RAM location selected by the index MUX is also sent to the comparator. If tag bit <15> is clear, indicating an invalid tag, the comparator generates a cache miss indication. If tag bit <15> is set and tag bits <8:0> match physical address bits <21:13> stripped off the MCA bus by the tag MUX, the comparator generates the signal MCTL TBC HIT.

When the cache invalidate signal is sent to the memory controller from the Q22 bus, the tag MUX passes physical address bits <21:13> from the MCA bus to the comparator. The cache tag at the tag RAM location accessed by the index MUX is also sent to the comparator. If the two tags match, the cache tag at the location accessed by the index MUX is marked invalid by clearing bit <15>.

TB/Cache RAM

The cache portion of the TB/cache RAM is the upper 2K locations. Figure 6-5 shows the organization of the TB/cache RAM. Each cache location contains data or an instruction.

Data are read from or written to the cache from the MCD bus when a Memory Request from the data path specifies a function requiring a cache access. Bits <33:32> of the MCT microinstruction that is executed to implement the requested function, determine whether the current cache access is a read or a write. (See Table 5-5 for the encoding of these bits). Bits <30:29> of the MCT microinstruction determine whether the current cache access is a normal access (that is, a read or a write), or a conditional cache invalidate. (See Table 5-6 for the encoding of these bits).

Write Isolation Buffer

The write isolation buffer is used the same way for cache operations as for translation buffer accesses: it allows the tag portion of a cache entry to be written into the tag RAM during a cache write, and is disabled to allow the tag from the tag RAM to be read into the comparator during a cache read or a cache invalidate.

TB/Cache Comparator

The TB/cache comparator asserts a TB/cache hit indication for a cache read or invalidate operation when the stored address tag from the tag RAM equals the search address tag supplied by the tag MUX. When the comparison results in a match, the comparator asserts the signal MCTL TBC HIT H. If this signal is not asserted, the compared tags did not match.

Cache Operations

Each cache entry can be read, written, or conditionally invalidated. The following paragraphs describe how these accesses are accomplished.

Address Sources

Physical addresses for cache operations can come from any of the following: the data path, the register file, the register file modified by the adder, the PAR, or the merge register.

For cache accesses, the adder and its register are used to supply modified physical addresses for unaligned data accesses, and for probes to check access privilege (RCHECK and WCHECK Memory Request functions).

Cache Reads

For a cache read operation, the selected data are driven onto the MCD bus, through the byte rotator, and latched into the merge register. From the merge register, the data are driven over the memory data bus to the data path. The TB/cache hit or miss status is available to the data path in the same cycle and the data on the memory data bus are used or ignored accordingly.

Cache Writes

If the physical address presented to the TB/cache RAM during a cache read access results in a miss, the physical address is driven on the Q22 bus, and the data at that address obtained from physical memory. The Q22 bus delivers the data to the MCD bus for a cache write operation, and the data are written into the addressed location in the cache. Meanwhile, the corresponding tag bits are selected from the physical address on the MCA bus by the tag MUX and written into the tag RAM at the same address that the data are written into in the TB/cache RAM.

If either bit <29> or bit <28> is set in the physical address on the MCA bus, a cache write does not occur. Bit <29> set means that the physical address is located in I/O space, and no I/O space addresses are cached.

Physical address bit <28> is part of the PTE stored in the translation buffer. It is latched in the access violation PAL from the MCD bus after a TB cycle (see Figure 6-1). When bit <28> is set, the physical address is located in that part of physical memory that can be shared by another processor on the Q22 bus. The addresses in the shared portion of physical memory are not cached either. Thus, no cache write occurs if either bit <29> or <28> is set.

The cache is a write-through cache with write allocation: any macroinstruction that causes a write updates physical memory and also causes a write to the cache. The write to physical memory is handled by a Q22 bus cycle. Next, a cache access is performed. If the cache access results in a hit, the cached data are driven onto the MCD bus and latched in the merge register. The data that were written to memory are also stored in the register file. From there, the data are driven onto the MCA bus, through the reverse pass latch and the byte rotator, and latched in the merge register, writing over part or all of the data just read from the cache. The updated data are then written into the cache at the specified physical address; the cached data now matches the data in physical memory.

Conditional Cache Invalidates

A cache invalidate operation is conditional: the addressed data cache location is invalidated if there is a match between the stored tag and the search tag. Only one cache entry at a time can be invalidated. For a cache invalidate operation, the valid bit, bit <15>, is cleared in the associated cache tag; the actual data in the cache entry are undefined.

Since the cache invalidate signal arrives from the Q22 bus asynchronously, conditional logic is implemented in hardware to enable the write to cache as soon as possible after the cache invalidate signal arrives. The cache invalidate signal is generated whenever an I/O device on the bus writes to physical memory and causes a cache invalidate trap in the memory controller. When bits <30:29> of the current MCT microinstruction are both ones, the TB/cache access selected is conditional cache invalidate (see Table 5-6) and the signal MCTN COND CINV is asserted. This signal is ANDed with MCTT TBC HIT DLYD to enable the cache write.

Transferring Data

Transferring data within the memory controller module is the fifth of the eight functions that the memory controller module performs. The hardware components are the MCA bus, the MCD bus, the memory data bus transceiver, the memory control bus, the merge register and byte rotator, and the reverse pass latch. The following paragraphs describe each of these components in turn.

MCA Bus

The memory controller address (MCA) bus is completely contained on the MCT module. It is a 32-bit, tri-state bus that normally supplies virtual and physical addresses to the TB/cache and to the Q22 bus write register. It is also the path used to transfer read data to the data path module, and write data from the data path module.

Addresses or data are asserted onto the MCA bus by 22 ns into the MCT microcycle, and are held until the end of the microcycle. The MCA bus destinations are clocked or latched by 110 ns into the microcycle. The MCT microcode guarantees that only one driver is enabled onto the bus during every cycle.

Table 6-1 lists the possible MCA bus sources and destinations. When the physical address register (PAR) is driving the MCA bus, it only drives bits MCA <29> and <21:09>. MCA bits <31:30> and <28:22> are passively asserted by pull-up resistors, and <8:0> are provided by the offset register file or the adder.

Table 6-1. MCA Bus Sources and Destinations

Sources	Data written to bus:
MDB transceiver	MCA<31:00>
page register file	MCA<31:09>
offset register file	MCA<08:00>
adder	MCA<08:00>
PAR	MCA<29:28>, <21:09>
merge register	MCA<31:00>
pull-up resistors	MCA<31:30, 27:22>
Destinations	**Data written from bus:**
MDB transceiver	MCA<31:00>
page register file	MCA<31:09>
offset register file	MCA<08:00>
adder	MCA<08:00>
TB/cache MUXs	MCA<31:00>
reverse pass latch	MCA<31:00>
Q22 bus write register	MCA<29>, <21:00>

prefetch FIFO	MCA<07:00>

MCD Bus

The memory controller data (MCD) bus is also completely contained on the MCT module. It is normally used to transfer data to and from the TB/cache RAMs, transfer data from the Q22 bus read register, and transfer data to the PAR. It is also passively asserted by pull-up resistors. Table 6-2 lists the possible sources and destinations for the MCD bus.

Table 6-2. MCD Bus Sources and Destinations

Sources	Data written to bus:
TB/cache RAM	MCD<31:00>
reverse pass latch	MCD<31:00>
Q22 bus read register	MCD<15:00> or MCD<29>, <21:00>
control and status registers (CSRs)	MCD<00>
pull-up resistors	MCD<31:00>
Destinations	**Data written from bus:**
merge register via rotator	MCD<31:00>
TB/cache RAM	MCD<31:00>
control and status registers (CSRs)	MCD<00>
PAR	MCD<20:19>, <12:00>
access violation latch	MCD<30:26>

Memory Data Bus Transceiver

The MDB transceiver is located between the memory controller and the data path modules. It isolates the interboard memory data bus from the memory controller MCA bus and consists of both receive and transmit buffers.

Bits <57:56> in the MCT microinstruction control the transceiver direction, and enable the output. Table 5-1 shows the transceiver control field encoding.

Memory Control Bus

This 8-bit bidirectional bus is physically part of the CD interconnect on the backplane. The memory control bus conveys memory function requests from the data path module to the memory controller. The memory function requests are then latched in the MCT memory request latch. In addition, instruction stream bytes from the memory controller prefetch FIFO are time-multiplexed over the memory control bus and latched into the IBYTE register on the DAP module.

Merge Register and Rotate Logic

The merge register actually consists of four registers, each one byte wide. Each byte-wide register is enabled by one bit in the MCT microinstruction (the merge register selects, bits <39:36>). The entire 32 bits in the merge register are driven onto the MCA bus when bit <62> in the current MCT microinstruction is asserted.

The byte rotator consists of eight shifters; each shifter handles four bits. The 32 bits from the MCD bus are the inputs to the shifters and are driven onto the MCD bus from the cache, the reverse pass latch, or from the Q22 bus read register. The shifters are controlled by the byte rotate select field (bits <35:34>) in the MCT microinstruction. The byte rotator shifts the longword into the desired alignment (see Table 5-4).

Data to be read or written across byte or word boundaries are rotated by 0, 1, 2, or 3 bytes in the byte rotator, and latched in the merge register. Since the byte-wide registers in the merge register are individually enabled, any number and combination of the rotated bytes can be latched (see Table 5-3). If data from a previous rotate/merge operation are still in the selected register, the data are written over. This is exactly how data are updated on write-through cache operations, for example.

Reverse Pass Latch

The reverse pass latch allows data from the MCA bus to be latched and presented to the MCD bus. From the MCD bus, the data may be written to the translation buffer, the cache, or presented to the rotate/merge logic for alignment.

The input to the reverse pass latch is MCA BUS <31:00> and the output is MCD bus <31:00>. The reverse pass latch is controlled by one latch enable bit (<40>) and one output enable bit (<41>) in the MCT microinstruction.

Prefetching Instruction Stream Bytes

Prefetching bytes from the instruction stream is the sixth of the eight functions that the memory controller module performs. Three hardware components handle the prefetching: the prefetch FIFO, the FIFO control logic, and the prefetch program counter. The following paragraphs describe these components and how the prefetching operates.

Prefetch FIFO

The prefetch FIFO is a 16-locations-deep by 8-bits-wide RAM that holds the next 0 to 16 bytes from the instruction stream. The prefetch FIFO is loaded from the low byte of the MCA bus. The FIFO is loaded at the end of the current MCT microcycle if bit <50>, the prefetch FIFO load clock bit, is asserted in the current microinstruction.

The output from the FIFO is the next byte in the instruction stream; it is sent to the IBYTE register on the DAP module via the memory control bus.

When a change in the program flow occurs (for example, because a Memory Request with IB.REFILL is executed), an MCT microinstruction with bit <51> asserted is executed to clear the entire contents of the FIFO.

Prefetch FIFO Control Logic

The control logic is a counter that keeps track of how many bytes have been loaded into the prefetch FIFO. When the FIFO contains less than eight bytes, the control logic asserts the prefetch enable flag, MCTP PREFETCH EN. This signal is one of the inputs to the branch MUX in the MCT microsequencer. When it is asserted, the MCT microcode branches to a microroutine that refills the prefetch FIFO.

Prefetch Program Counter

The I-prefetch program counter is maintained in location 2 of the register file. This PC always contains the physical address of the last instruction stream byte loaded into the FIFO.

The prefetch PC is incremented whenever a byte from the MCA bus is loaded into the prefetch FIFO. The PC is also incremented when an I-stream Request microinstruction is executed by the data path. When the data path executes a Memory Request with IB.REFILL, the data path sends a 32-bit virtual address over the memory data bus; this address is translated and saved in location 2 of the register file as the new prefetch PC.

Prefetch Operation

The IBYTE control logic on the DAP module asserts the signal DAPR IB TAKEN to inform the memory controller that the instruction stream byte that was on the memory control bus is now latched in the IBYTE register. DAPR IB TAKEN causes the prefetch control logic to decrement the count by one, and causes the prefetch FIFO to drive the next I-stream byte onto the memory control bus.

When the prefetch FIFO drives another I-stream byte onto the memory control bus, it asserts the signal MCTP NXT VALID to inform the DAP module that a valid I-stream byte is on the memory control bus.

When the prefetch enable flag is asserted because the prefetch FIFO contains less than eight bytes, a memory controller microroutine is invoked to refill the prefetch FIFO. The microroutine uses the physical address in the prefetch PC and performs a cache access.

If there is a cache hit, the retrieved data are driven into the byte rotator via the MCD bus, and rotated so that the desired byte (that is, the next byte in the I-stream) is in the low-byte position, and latched in the merge register. All 32 bits are then driven onto the MCA bus, but only the low eight bits are loaded into the prefetch FIFO. The prefetch PC is incremented by one. The 32 bits are then driven off the MCA bus, through the reverse pass latch, to the MCD bus. From there, they are driven into the rotator and shifted again so that the next byte in the I-stream is in the low-byte position, and latched in the merge register. From the merge register, the rearranged 32 bits are driven onto the MCA bus, and the low-order eight bits loaded into the next location in the prefetch FIFO.

This process continues until the prefetch FIFO contains more than eight bytes and the prefetch enable flag is negated. If there is a cache miss instead of a cache hit, the data are retrieved from physical memory, written to the cache, and then the same procedure carried out.

Thus, the prefetching is handled entirely by the memory controller. The result is that the memory controller always has the next byte of instruction stream data ready to be latched into the IBYTE register from the memory control bus.

Tracking and Reporting Status

Since the memory controller operates essentially as a slave responding to commands from the data path module, one of the memory controller's functions is to track status and report it to the data path. The hardware components that are involved with tracking and reporting status are the four CSRs (control and status registers), the access protection latch, and the access violation PAL. The next paragraphs describe these components and their activities.

Control and Status Registers

The four single-bit CSRs control memory management functions performed by the MCT, and reflect error status. The CSRs share the register file address space, but they are read and written over the MCD bus as MCD <0>.

The CSRs are selected by the MCT microinstruction register file address selects, bits <21:18>. (See Table 5-8 for the encoding.) MCD <0> is loaded into the selected CSR when bit <23>, the offset register file write enable bit, is asserted in the current MCT microinstruction. The content of the selected CSR is driven onto the MCD bus as bit <0> when bit <59>, the offset register file output enable bit, is asserted in the current MCT microinstruction.

The four CSRs are the map enable control register, the cache enable control register, the error flag status register, and the IB error status register. Each of these is described further in the following paragraphs.

Map Enable Control Register

This single-bit register is a copy of the Memory Management Enable Register, as defined by VAX architecture. (For more information about VAX architecture, see the *VAX Architecture Handbook*, order number EB-19580-20.)

When the bit in the map enable control register is set, address translation and access checking are enabled. When this bit is clear, all addresses are assumed to be physical addresses.

Cache Enable Control Register

The data and instruction cache is enabled when the bit in this CSR is set.

Error Flag Status Register

The bit in this CSR is set when an error has been detected during the execution of a Memory Request from the DAP module which the hardware is unable to report to the DAP micromachine directly. A code indicating the cause of the error is stored in location 3 of the register file. The code is created by reading a zero from location four of the register file, and adding a constant to it in the adder. Errors and their corresponding codes are shown in Table 6-3.

Table 6-3. MCT Error Codes

Error Code	Error
0	Invalid State
1	Parity Error
2	Q22 Bus Timeout

3	Illegal Operation
4	Access Violation
5	Translation Check Error

When one of the errors listed in Table 6-3 is detected, the MCT microcode writes the corresponding error code in the error code location in the register file, and sets the error flag status bit by writing a 1 to the error flag status register via MCD <0>. The error flag status bit is set directly by the hardware when an access violation is detected. The signal MCTT MEM ERROR is sent to the DAP module over the backplane indicating a memory error has occurred. This signal is one of the inputs to the data path OR MUX.

IB Error Status Register

The bit in this CSR is set by the prefetch microroutine when an error has been detected during an I-stream prefetch operation such that prefetching cannot continue without intervention by the DAP micromachine; for example, the prefetching crosses a page boundary. The signal MCTT IB ERROR is sent to the DAP module over the backplane. MCTT IB ERROR is ANDed with the signal DAPR IB INVALID to generate the signal DAPR IB ERROR; DAPR IB ERROR is one of the inputs to the jump MUX.

Access Protection Latch

The memory controller checks for access violations after every translation buffer access. The 8-bit access protection latch captures memory access information for determining access violations. The latch closes at the same time as the PAR; it stores

PTE bits <30:26> (the protection field and the modify bit) from the MCD bus at the end of a translation buffer access. The latch also saves the TB hit or miss indication from the TB access. The PAR latch enable bit in the MCT microinstruction (bit <42>) also enables the access protection latch. The four protection field bits, the modify bit, and the TB hit or miss bit are passed on as the inputs to the access violation PAL.

Access Violation PAL

There are ten inputs to the access violation PAL:

- the six bits from the latch,
- the two access mode bits DAPT MEM REQ MODE <1:0> which are sent over the backplane on a Memory Request and specify the mode (kernel, executive, supervisor, or user) of the current Memory Request,
- the modify intent bit, DAPT MODIFY H, also sent over the backplane, which specifies the modify intent (read or write) of the current Memory Request,and
- the signal MCTS EN ACC CHECK; this signal is asserted after each translation buffer access to enable the access violation PAL.

The logic in the PAL does two comparisons. It compares the protection field from the PTE with the access mode bits and the modify intent bit from the current Memory Request microinstruction. The PAL asserts the signal MCTS ACCVIOL if the process that issued the Memory Request does not have sufficient privilege to access the page (corresponding to the PTE just retrieved from the translation buffer) for the intended read or write operation. MCTS ACCVIOL generates the signal MCTT MEM ERROR to the DAP module, and causes an error code of 4 to be loaded in register file location 3.

For convenience, Table 6-4 shows the encoding of the PTE protection field, reproduced from the *VAX Architecture Handbook*.

The second comparison is between the modify bit from the PTE and the modify intent bit from the Memory Request. If the PTE modify bit is a 0 (meaning the associated page has not yet been modified), and the modify intent bit is a 1 indicating a write intent, the PAL asserts the signal MCTS DAP MOD REF to indicate modify refused. MCTS DAP MOD REF is sent over the backplane and is also one of the inputs to the data path OR MUX.

If neither an access violation or a modify refused is found, the next microcycle is the cache access. If an access violation occurs, the DAP microcode traps to a memory management fault service routine, and generally, the process is aborted. If modify refused occurs, the DAP microcode takes a modify refused trap and a new Memory Request microinstruction is issued to rewrite that translation buffer entry (the PTE) with the modify bit set. Then a return is executed to the microroutine containing the original Memory Request, and the cache is accessed in the next microcycle.

The translation buffer miss signal that is one of the inputs to the access violation PAL, is passed through the PAL and sent over the backplane to the data path OR MUX as MCTS TB MISS.

Table 6-4. Protection Codes

Protection Field	Mnemonic	Kernel	Executive	Supervisor	User	Comment
0000	NA	no access	no access	no access	no access	No Access
0001		unpredictable	unpredictable	unpredictable	unpredictable	Reserved
0010	KW	read/write	no access	no access	no access	
0011	KR	read only	no access	no access	no access	
0100	UW	read/write	read/write	read/write	read/write	All Access
0101	EW	read/write	read/write	no access	no access	
0110	ERKW	read/write	read only	no access	no access	
0111	ER	read only	read only	no access	no access	
1000	SW	read/write	read/write	read/write	no access	
1001	SREW	read/write	read/write	read only	no access	
1010	SRKW	read/write	read only	read only	no access	
1011	SR	read only	read only	read only	no access	
1100	URSW	read/write	read/write	read/write	read only	
1101	UREW	read/write	red/write	read only	read only	
1110	URKW	read/write	read only	read only	read only	
1111	UR	read only	read only	read only	read only	

Communicating with the Q22 Bus Interface

Communicating with the Q22 bus interface is the eighth of the eight functions that the memory controller performs. The Q22 bus interface consists of the controller, the Q22 bus write register, and the Q22 bus read register. This section describes these components and how they interact with the memory controller.

Q22 Bus Controller

The controller handles the Q22 bus protocol, freeing the memory controller from this task. It is implemented as a programmable state machine and consists of a microsequencer, a function register, several logic PALs and status registers. These Q22 bus controller hardware components are described in more detail in Chapter 7.

The Q22 bus controller handles all bus arbitration, and is capable of executing block mode transfers to and from memory, if block mode is supported by the memory. If the Q22 bus is free, the arbitration logic in the controller sets up the bus address while waiting for the cache hit or miss signal.

The memory controller communicates with the Q22 bus controller via six bits of the MCT microinstruction. The controller reports status to the memory controller via six status flags. The microcode bits and the status flags are described in the section titled "Q22 Bus Controller Interface" in Chapter 5. The controller latches the three bit function code and the function request bit from the memory controller microinstruction in the function register, and the function code bits are then used to address the controller's control store. The Q22 bus controller microcode takes over to carry out the function requested by the memory controller.

Q22 Bus Write Register

The write register latches addresses and data from the MCA bus that need to be driven onto the Q22 bus. The write register is actually one side of the bus transceivers that separate the memory controller module from the Q22 bus.

The Q22 bus write register is controlled by bit <43> in the current MCT microinstruction. When this bit is asserted, the data stable on MCA <29> and <21:00> are latched in the write register. The data written can be a 22-bit physical address, a 13-bit I/O space address, or 16 bits of data to be written to physical memory or an I/O device.

The outputs from the write register are BDAL <21:00>; these are the Q22 bus data/address lines. If MCA <29> is set, it is driven onto the Q22 bus as the signal BBS7 to indicate that BDAL <12:00> represent a physical adddress in I/O space.

Q22 Bus Read Register

The read register latches addresses and data from the Q22 bus that need to be driven onto the MCD bus. The read register is actually the other side of the bus transceivers that separate the memory controller module from the Q22 bus.

The read register is controlled by bit <44> in the current MCT microinstruction. When this bit is asserted, the data on BDAL <21:00> are latched in the read register. The data latched can be a 22-bit cache invalidate address, a 9-bit Q22 bus interrupt vector, or 16 bits of data read from physical memory or an I/O device.

The outputs from the read register are MCD bus bits <21:00>. If the signal BBS7 is asserted, it is driven onto the MCD bus as MCD <29> to indicate that MCD <12:00> represent a physical adddress in I/O space.

Microprogram Level Flow: MOVW

Now that the hardware components of the memory controller module have been described, this section takes a MOVW (move word) macroinstruction and describes how the memory controller accomplishes the TB accesses, cache accesses and data retrieval necessary to return the requested data to the data path.

A MOVW macroinstruction replaces the destination operand with the source operand. A sample MOVW instruction is:

MOVW (R0), R1

The first operand specifier, "(R0)," uses register deferred address mode, and the second operand specifier, "R1," uses register mode. At some virtual address in memory , this instruction looks like this:

51	60	B0	:VA

where B0 is the opcode, 60 specifies register deferred mode using R0, and 51 specifies register mode using R1. For this example, R0 contains the virtual address 00000211 (hex). This instruction obtains the word of data located at 00000211 and moves it into R1.

The next few paragraphs summarize the data path steps needed to decode and execute this MOVW macroinstruction.

Step 1. Evaluate the opcode to select the proper DAP microroutine for this macroinstruction.

Step 2. Evaluate the first operand specifier (the source) and obtain the first operand.

Step 3. Evaluate the second operand specifier (the destination) and write the first operand to the destination.

The remainder of this chapter describes the microprogram steps executed by the memory controller to translate the virtual address in R0, obtain the data located at the corresponding physical address, and return the data to the data path.

Whenever the memory controller is not doing anything, it loops in an "idle state" microroutine. In the idle state, the memory controller monitors the prefetch branch condition. If the branch condition is asserted, a jump is taken to the microroutine that refills the prefetch FIFO. Dispatches are enabled during portions of the idle routine so that the correct location in control store can be accessed if a memory function request is received from the data path. Assume that the memory controller is in the idle state, that the prefetch FIFO is full, and that the entire MOVW macroinstruction is located in the prefetch FIFO.

Evaluating the Opcode

The MOVW opcode, B0, is clocked into the IBYTE register and decoded. The decode causes a dispatch to the microroutine that handles MOVW macroinstructions in the DAP control store.

The data path asserts the signal DAPR IB TAKEN, the prefetch FIFO drives the first operand specifier, 60, onto the memory control bus, and the prefetch counter is decremented by one. The memory controller asserts the signal MCTP NXT VALID to inform the data path that the next instruction stream byte is on the memory control bus. The data path asserts DAPR LOAD I BYTE and 60 is clocked into the IBYTE register.

Evaluating the First Operand Specifier

Next, the first operand specifier is decoded. The decode causes a dispatch to the DAP microroutine that handles the evaluation of operand specifiers with register deferred mode.

Once the decode has completed, the data path asserts IB TAKEN and the prefetch FIFO drives 51 (the second operand specifier) onto the memory control bus. The memory controller asserts NXT VALID, the data path asserts LOAD I BYTE, and 51 is clocked into the IBYTE register. So far, the memory controller has remained idle except for supplying the next instruction stream byte.

Meanwhile, the DAP microinstructions in the register deferred mode routine begin executing. The first microinstruction is a Memory Request with the function VREAD.RCHECK specified; the microinstruction in hex is 1E80B61628, where 1E is the opcode. This function asks the memory controller to perform a virtual read operation, with a check for read access. The data path assembles the twelve bits of information to send the memory controller. It latches these eight bits in both memory function latches because bit <31> in the Memory Request microinstruction is set:

- the two high-order bits are 01 from the size register, indicating a data type of word;
- the next bit is the data flow bit (<28>) from the Memory Request microinstruction which is a 0, indicating the data flow will be from MCT to DAP;
- the low-order five bits are the function code from the Memory Request microinstruction (bits <27:23>), and they are 00001 (binary) indicating the function VREAD.RCHECK.

The data path sends an additional four bits over the backplane: two access mode bits, one modify intent bit, and the second part flag. Since bit <30> (the mode bit) in the microinstruction is a 0, the two access mode bits sent over the backplane reflect the contents of the PSL.MODE register. The MOVW is part of the program running in a User process, so the PSL.MODE contains 11 (binary).

The modify intent bit in the Memory Request microinstruction (bit <29>), is a 0 indicating read intent. The second part flag is also a 0 because this is the first part of this Memory Request to be executed.

The long operand of the Memory Request (bits <22:16>), specifies @pointer 1; that is, use the contents of the register pointed to by pointer 1. When 60 was decoded, pointer 1 was loaded with the value 0, pointing to R0. R0 contains the virtual address 00000211. Therefore, 00000211 is driven over the memory data bus to the memory controller.

Meanwhile, the eight bits in the first memory function latch are driven over the memory control bus, and the additional four bits of memory request information are driven over the backplane. The memory function control logic on the DAP module asserts the signal DAPR MEM REQUEST to inform the memory controller that a new function code is on the memory control bus. This signal generates the signal MCTN MRL LE which is the latch enable for the memory request latch, causing the following eight bits to be latched in the memory request latch on the memory controller:

- the high-order bit is the second part flag from the backplane, so it is a zero;
- the next two high-order bits are the data type bits, and they are 01, indicating data type word;
- the low-order five bits are the function code: 00001 (binary), indicating VREAD.RCHECK.

The data flow bit from the DAP memory function latch is one input to the branch MUX in the MCT microsequencer. The modify intent bit from the backplane is an input to the branch MUX, and an input to the access violation PAL. The two access mode bits from the backplane are also inputs to the access violation PAL.

Up to now, the memory controller has remained in the idle state microroutine. Those microinstructions within the idle microroutine that have dispatch enabled also have 00 in the transceiver control field to enable the transceiver to drive data from the memory data bus onto the MCA bus. Thus, the virtual address 00000211 is now stable on the MCA bus. The memory controller asserts the signal MCTT REQ ACK to inform the data path that the 32 bits on the memory data bus have been accepted, and the data path stops driving 00000211 over the memory data bus.

At this point, dispatches are enabled, the correct eight bits are latched in the MCT memory request latch, and a virtual address is stable on the MCA bus. The microinstruction decode logic in the MCT microsequencer sends 11 (binary) as the two high-order bits over the CSA bus, and control store is accessed with the 10-bit address 321 (hex). This is the address of the microroutine that handles the reading and writing of data words. The dispatch causes the MEM BUSY signal to be asserted.

Obtaining the Operand

At the rising edge of MCTM CLK125, the first microinstruction in the read/write data words microroutine is available at the output of the MCT control store. This first microinstruction causes:

- the data stable on the MCA bus (the virtual address 00000211) to be written into the register file at location 0 (the virtual address location),
- the low nine bits of the virtual address to be written into the adder, the constant 1 added to them, and the sum latched in the adder register,

- the MEM BUSY signal to remain asserted,
- a translation buffer read access, and
- the physical address register latch enable signal to be asserted.

TB Access

To accomplish the translation buffer read access, the tag MUX selects VA <30:17> from the MCA bus, which is 0000 (hex), and passes it to the TB/cache comparator.

At the same time, the index MUX selects VA <31> and VA <16:9> from the MCA bus, and provides zeros for the three high-order bits. Thus, location 001 (hex) is accessed in the tag RAM and in the TB/cache data RAM. The 16-bit tag at location 001 in the tag RAM is passed to the TB/cache comparator. The tag is not 0000 so it does not match the search tag provided by the tag MUX, and the comparator does not assert the signal MCTL TBC HIT H.

Simultaneously, the PTE at location 001 in the TB/cache RAM is driven onto the MCD bus. PTE bits <12:0> from the MCD bus are latched into the PAR as physical address bits <21:09>. PFN <19> is 0, so 0 is latched in the PAR as physical address bit <28>. The AND of the TB/cache hit signal and PFN <20> is 0, so 0 is latched in the PAR as physical address bit <29>. The four-bit protection field and the modify bit from the PTE are also latched in the access protection latch.

The microprogram control field of this first microinstruction disables traps and dispatches, so the MCT microsequencer selects bits <9:0> from the first microinstruction, modified by any asserted conditions in the branch MUX, as the next microaddress. Bits <9:0> are 0F0. The branch MUX input that modifies bit 0 is MCA <0> (see Figure 5-7), which is a 1 because of the VA 00000211 currently on the MCA bus. Thus, 0F1 is passed through the next address buffer and used to access control store as the next microaddress, and the first MCT microcycle ends.

Cache Access

At the rising edge of MCTM CLK125, the microinstruction at 0F1 is available at the output of the control store. This microinstruction causes:

- the 9-bit offset portion of the virtual address location in the register file to be driven onto the MCA bus as MCA <8:0>; that is, 011 hex,
- the fifteen bits in the PAR to be driven onto the MCA bus as MCA <29:28> and <21:09>,
- MCA <31:09> to be written into the page portion of the virtual address location in the register file (MCA <31:30> and <27:22> are asserted by pull-up resistors, MCA <29:28> and <21:09> are from the PAR),
- MEM BUSY to remain asserted,
- a cache read access,
- the data on the MCD bus from the cache access to be rotated 1 byte to the right and all four bytes latched into the merge register,
- the physical address on the MCA bus to be latched into the Q22 bus write register, and
- the Q22 bus function request bit and the

function code for a read block operation (DATBI) to be sent to the Q22 bus controller; the go bit is not sent.

To accomplish the cache read access, the tag MUX selects physical address bits <21:13> from the MCA bus, supplies five zeros for the high-order bits and passes these fourteen bits to the TB/cache comparator.

At the same time, the index MUX selects PA <12:2> from the MCA bus, and provides a one for the high-order bit. The 16-bit tag at the accessed location in the tag RAM is passed to the TB/cache comparator. The tag does not match the search tag provided by the tag MUX, and the comparator does not assert the signal MCTL TBC HIT H.

Simultaneously, the data at the accessed location in the TB/cache RAM are driven onto the MCD bus, rotated one byte to the right and latched into the merge register.

The microprogram control field of this second microinstruction also disables traps and dispatches, so the MCT microsequencer selects bits <9:0> from the first microinstruction, modified by any asserted conditions in the branch MUX, as the next microaddress. Bits <9:0> are 0D0. The branch MUX input that modifies bit 1 is TB.ERROR (see Figure 5-7), which is a 1 because of the miss on the translation buffer access. Thus, 0D2 is passed through the next address buffer and used to access control store as the next microaddress, and the second MCT microcycle ends.

Q22 Bus NOP

At the rising edge of MCTM CLK125, the

microinstruction at 0D2 is available at the output of the control store. This microinstruction causes:

- the offset and page portions of location four in the register file, which contain zeros, to be driven onto the MCA bus as MCA <31:0>,
- the constant 4 to be added to the low nine bits on the MCA bus (currently zeros), and latched in the adder register,
- no cache or TB access,
- the Q22 bus function request bit and the function code for no operation to be sent to the Q22 bus controller.

The microprogram control field of this third microinstruction disables dispatches, but enables traps and jumps. The branch control field is 100 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction as the next microaddress (see Figure 5-7). Bits <9:0> are 379. This is the microaddress of a routine that sets error codes. Thus, 379 is passed through the next address buffer and used to access control store as the next microaddress, and the third MCT microcycle ends.

Set Error Code

At the rising edge of MCTM CLK125, the microinstruction at 379 is available at the output of the control store. This microinstruction causes:

- the value 4 from the adder register to be written into the error code location of the register file, and
- the MEM BUSY signal to be cleared.

The value 4 in the error code location of the register file indicates an access violation (see Table 6-3). This code is always set for TB.ERROR in case an access violation is the cause. Three conditions in addition to access violation cause TB.ERROR: modify refused, page crossing, and TB miss. The data path microcode is notified of these conditions through inputs to the OR MUX. If TB.ERROR occurs in the MCT and none of these inputs is asserted in the DAP OR MUX, then access violation is the cause and the DAP microcode examines the error code location of the register file.

So even though the access violation error code is set by this microinstruction, no access violation has occurred—only a TB miss—and the TB MISS signal is asserted as an input to the DAP OR MUX.

The microprogram control field of this microinstruction causes a jump back to the MCT idle state microroutine to wait for the next function request from DAP.

TB Read

While the memory controller was executing the five MCT microinstructions described above, the data path executed a Move microinstruction to set a register number in a pointer register, then another Move to try to move the requested data into the data path chip from the memory controller. The requested data are not available because of the TB miss. The DAP microcode branches to a microroutine that handles TB misses because of the TB miss input signal to the OR MUX.

The DAP microcode determines that the PTE for the VA 00000211 must be read since 00000211 wasn't in the translation buffer. So the DAP microcode completes an entire P0 virtual to physical translation operation to obtain the PTE (full memory management is enabled). Briefly, the steps performed by the DAP microcode are:

- check the virtual address that was sent to the memory controller (00000211) to determine whether it is a system or process space address, and since it is a process space address, whether it is in P0 or P1;
- rotate the virtual address to obtain the virtual page number (VPN);
- check that the VPN is within the P0LR limits;
- add the VPN to the virtual address in the P0 base register (P0BR)—this sum is the virtual address of the desired PTE;
- ask the memory controller to do a virtual read using this computed virtual address (this reference is made as a kernel read).

The virtual address of the PTE is located in system space (where process page tables reside), so the memory controller accesses the system space portions of the tag and TB/cache RAMS for a translation buffer read. Assuming a TB hit and a subsequent cache hit (that is, the desired PTE was found in the cache), the memory controller returns the PTE to the data path via the memory data bus. The PTE is A4000000 (hex). Now the data path has the PTE for the virtual address 00000211. (If the PTE was not in the cache, the memory controller would have asked the Q22 bus controller to perform a Q22 bus read, and the PTE would have been returned from physical memory.)

TB Write

Next, the PTE needs to be written into the translation buffer. The memory management routine in the DAP microcode sends the memory controller a Memory Request with WRITE.TB specified. These eight bits are latched in the first memory function latch:

- the two high-order bits are 00 (binary) indicating a data type of byte (byte is always specified for TB.WRITE even though a longword is written into the translation buffer);
- the next bit is the data flow bit (<28>) from the Memory Request microinstruction which is a 1, indicating the data flow will be from DAP to MCT;
- the low-order five bits are the function code; they are 01001 (binary) indicating WRITE.TB.

The following four bits are sent over the backplane:

- two access mode bits which indicate the mode of the current process (access mode is ignored on writes to the TB);
- one modify intent bit from the microinstruction; (It is a 0, indicating read intent even though the intended function is a write; this is because the modify intent does not matter on a TB.WRITE.)
- a second part flag of 0 because this is the first part of this Memory Request to be executed.

The long operand of the TB.WRITE Memory

Request specifies the virtual address 00000211. The eight bits in the first memory function latch are driven over the memory control bus, and the additional four bits of Memory Request information over the backplane. The data path asserts the signal DAPR MEM REQUEST and these eight bits are latched in the memory request latch on the memory controller:

- the second part flag, which is a 0,
- the data type bits, 00, and
- the function code, 01001.

The data flow and modify intent bits are sent to the MCT branch MUX, and the modify intent bit and the access mode bits are sent to the access violation PAL.

Since the memory controller is in the idle microroutine and dispatches are enabled during certain portions of the loop, the MCT microsequencer causes a dispatch on the contents of the memory request latch plus two high-order ones from the CSA PAL. Thus, the control store address 309 is accessed. This is the address of the MCT microroutine that handles writes to the translation buffer. The dispatch causes the MEM BUSY signal to be asserted.

The transceiver between the memory data bus and the MCA bus is also enabled in the idle routine, so the virtual address 00000211 is now stable on the MCA bus. The memory controller asserts MCTT REQ ACK to inform the data path that the VA has been accepted.

Save Virtual Address. At the next rising edge of MCTM CLK125, the first microinstruction in the MCT write TB microroutine is available at the output of the control store. This first microinstruction saves the virtual address on the MCA bus in the virtual address location of the register file. MEM BUSY remains asserted, and traps and dispatches are disabled. The microprogram control field specifies a jump to microaddress 138.

Wait. At the next rising edge of MCTM CLK125, the next microinstruction in the MCT write TB microroutine (the microinstruction at address 138) is available at the output of the control store. This microinstruction turns off the transceiver. MEM BUSY remains asserted, and traps and dispatches are disabled. The microprogram control field specifies a jump to microaddress 13A.

Clear Busy. At the next rising edge of MCTM CLK125, the microinstruction at microaddress 13A in the MCT write TB microroutine is available at the output of the control store. This microinstruction unconditionally clears busy. Traps and dispatches are disabled, and the microprogram control field specifies a jump to microaddress 164.

After the DAP microcode issues the TB.WRITE Memory Request, it executes a Moveout microinstruction which puts the PTE—A4000000—for the VA 00000211 on the memory data bus.

Accept PTE. At the next rising edge of MCTM CLK125, the microinstruction at microaddress 164 in the MCT write TB microroutine is available at the output of the control store. This microinstruction enables the transceiver to pass data from the memory data bus to the MCA bus, so the PTE to be written is now stable on the MCA bus. The microinstruction also enables the reverse pass latch, so the PTE is captured there. Traps and dispatches are disabled, and the microprogram control field specifies a jump to microaddress 166.

Write PTE. At the next rising edge of MCTM CLK125, the microinstruction at microaddress 166 is available at the output of the control store. This microinstruction causes:

- the data in location 0 of the register file to be driven onto the MCA bus (location 0 currently contains the VA 00000211),
- the content of the reverse pass latch, which is the PTE, to be driven onto the MCD bus, and
- a translation buffer write access.

To accomplish the translation buffer write access, the index MUX selects VA <31> and VA <16:9> from the MCA bus, and provides zeros for the three high-order bits. Thus, location 001 (hex) is accessed in the tag RAM and in the TB/cache data RAM.

The tag MUX selects VA <30:17> from the MCA bus, which are 0000 (hex), and passes these fourteen bits through the write isolation buffer to the tag RAM location accessed by the index MUX—001. The 14 bits are written as the low-order bits of the tag at location 001. The fifteenth bit in the tag is a spare, and the high-order bit, the valid bit, is written as a 1 because microinstruction bit <31> is asserted, indicating that this is a valid tag.

Simultaneously, the PTE from the MCD bus is written into location 001 in the TB/cache data RAM, and the TB write is complete. Thus, location 001 in the TB/cache now contains A4000000 (hex).

The microprogram control field of this microinstruction disables dispatches, but enables traps and jumps. The branch control field is 100 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction as the next microaddress (see Figure 5-7). Bits <9:0> are 002, so the memory controller returns to the idle state.

The last microinstruction executed by the DAP microcode was the Moveout microinstruction that put the PTE for the VA 00000211 on the memory data bus. Next, the DAP microcode executes another Memory Request; this time, the function code is REPEAT.FIRST. This causes the contents of the second memory function latch to be driven onto the memory control bus. The second memory function latch still contains the function code for the initial VREAD.RCHECK that failed. Thus, the memory controller is asked to retry the virtual read with read check, and the virtual address 00000211 is driven over the memory data bus again.

The data path asserts the signal DAPR MEM REQUEST and these eight bits are latched in the memory request latch on the memory controller:

- the second part flag, which is a 0,
- the data type bits, 01, indicating a word of data is to be read,
- the function code, 00001 (binary), indicating VREAD.RCHECK.

The data flow and modify intent bits are sent to the MCT branch MUX, and the modify intent bit and the access mode bits are sent to the access violation PAL.

Since the memory controller is in the idle micro-routine and dispatches are enabled during certain portions of the loop, the MCT microsequencer causes a dispatch on the contents of the memory request latch plus two high-order ones from the CSA PAL. Thus, control store address 321 is accessed again; this is the beginning address of the MCT read/write data words microroutine. The dispatch causes the MEM BUSY signal to be asserted.

The transceiver between the memory data bus and the MCA bus is also enabled in the idle routine, so the virtual address 00000211 is now stable on the MCA bus. The memory controller asserts MCTT REQ ACK to inform the data path that the VA has been accepted.

TB Access Retried

At the next rising edge of MCTM CLK125, the first microinstruction in the MCT read/write data words microroutine is available at the output of the control store. Once again, this microinstruction causes:

- the VA 00000211 from the MCA bus to be stored in register file location 0,
- the low nine bits of the VA to be incremented by 1 in the adder and the sum latched in the adder register,
- MEM BUSY to remain asserted,
- a translation buffer read access, and
- the physical address register latch enable signal to be asserted.

This time, the translation buffer access is successful; the index MUX accesses location 001 in the tag RAM, the fourteen low-order bits of the tag are sent to the comparator (0000 hex), and they match MCA bits <30:17> supplied by the tag MUX. The comparator asserts the signal MCTL TBC HIT H.

Simultaneously, the PTE at location 001 in the TB/cache RAM (A4000000) is driven onto the MCD bus. PTE bits <12:0> from the MCD bus are latched into the PAR as physical address bits <21:09>. PFN <19> is 0, so 0 is latched in the PAR as physical address bit <28>. The AND of the TB/cache hit signal and PFN <20> is 0, so 0 is latched in the PAR as physical address bit <29>. So the PAR now contains the 14-bit value 0000 (hex). The four-bit protection field and the modify bit from the PTE are also latched in the access protection latch.

The microprogram control field of this microinstruction is decoded by the MCT microsequencer and 0F1 is passed through the next address buffer and used to access control store as the next microaddress.

Cache Access Retried

At the rising edge of MCTM CLK125, the microinstruction at 0F1 is available at the output of the control store. Once again, this microinstruction causes:

- the value 011 (hex) to be driven from the offset portion of the virtual address location in the register file onto the MCA bus as MCA <8:0>;
- the fifteen bits in the PAR to be driven onto the MCA bus as MCA <29:28> and <21:09> (MCA <31:30> and <27:22> are ones via the pull-up resistors); thus, the 32 bits on the MCA bus form the hex value

DFC00011, and the physical address is the low 22 bits plus MCA <29>, or the hex value 000011.

- MCA <31:09> (DCF000) to be written into the page portion of the virtual address location in the register file,
- MEM BUSY to remain asserted,
- a cache read access,
- the data on the MCD bus from the cache access to be rotated 1 byte to the right and all four bytes latched into the merge register (the data are whatever happened to be in the accessed location),
- the physical address on the MCA bus (000011) to be latched into the Q22 bus write register, and
- the Q22 bus function request bit and the function code for a read block operation (DATBI) to be sent to the Q22 bus controller; the go bit is not sent.

To accomplish the cache read access, the tag MUX selects physical address bits <21:13> from the MCA bus, supplies five zeros for the high-order bits and passes these fourteen bits, which have the value 0000, to the TB/cache comparator.

At the same time, the index MUX selects PA <12:2> from the MCA bus, and provides a one for the high-order bit. Thus, location 804 is accessed in the tag RAM and in the TB/cache RAM. The 16-bit tag at location 804 in the tag RAM is passed to the TB/cache comparator. The tag does not match the search tag 0000 provided by the tag MUX, and the comparator does not assert the signal MCTL

TBC HIT H.

However, the data at location 804 in the TB/cache RAM are driven onto the MCD bus anyway, rotated one byte to the right and latched into the merge register.

The microprogram control field of the microinstruction is decoded and the MCT microsequencer selects bits <9:0> from the first microinstruction, modified by any asserted conditions in the branch MUX, as the next microaddress. Bits <9:0> are 0D0. This time, no branch conditions are set, so 0D0 is passed through the next address buffer and used to access control store as the next microaddress.

Incorrect Data Returned

At the rising edge of MCTM CLK125, the microinstruction at 0D0 is available at the output of the control store. This microinstruction causes:

- the contents of the merge register (whatever was stored in location 804 in the TB/cache RAM) to be driven onto the MCA bus as MCA <31:0>, and
- the transceiver to drive the data on the MCA bus over the memory data bus to the data path, even though it is not the desired data.

The microprogram control field of this microinstruction disables dispatches, but enables traps and jumps. The branch control field is 110 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction, modified by the branch condition TBC.MISS which is asserted because of the cache miss in the previous microcycle, as the next microaddress (see Figure 5-

7). Bits <9:0> are 090 and modified by TBC.MISS, the next address is 091. Thus, 091 is passed through the next address buffer and used to access control store as the next microaddress.

The cache miss signal is also sent to the data path, so the incorrect data on the memory data bus are ignored.

Q22 Bus Go

At the rising edge of MCTM CLK125, the microinstruction at 091 is available at the output of the control store. This microinstruction sends the go bit to the Q22 bus controller. This causes the controller to begin the DATBI (read block) cycle using the physical address (000011) saved in the Q22 bus write register two microcycles earlier.

The microprogram control field of this microinstruction disables dispatches, but enables traps and jumps. The branch control field is 100 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction as the next microaddress (see Figure 5-7). Bits <9:0> are 1C5. Thus, 1C5 is passed through the next address buffer and used to access control store as the next microaddress.

First Block Read

At the rising edge of MCTM CLK125, the microinstruction at 1C5 is available at the output of the control store. This microinstruction causes:

- the contents of the virtual address location in the register file to be driven onto the MCA bus; the virtual address location currently contains the hex value DCF00011,

- the adder to add the constant 1 to the low nine bits on the MCA bus and latch the sum in the adder register; so the adder register now contains 012, and
- the contents of the Q22 bus read register to be driven onto the MCD bus, rotated 1 byte to the right in the rotator, and latched in the merge register.

The microprogram control field of this microinstruction causes the microprogram to loop on this instruction until the SYNC/READY signal is received from the Q22 bus, indicating that the data are available and latched in the Q22 bus read register.

Q22 bus block reads use word-aligned addresses, so although the physical address 000011 was latched in the Q22 bus write register, the low byte is ignored, and a word of data is read at address 000010.

Suppose the data at physical address 000010 in memory is:

DD	CC	BB	AA	:000010

where the data requested by the data path is the word CCBB. (For this example, each letter represents one hex digit; AA, for instance, is one byte.) The first block transfer from the Q22 bus read returns the bytes BBAA and latches them in the Q22 bus read register.

This microinstruction moves the contents of the MCD bus into the rotator. The value FFFFBBAA is stable on the MCD bus at this point; the FFFF is provided by the pull-up resistors, and the BBAA is from the Q22 bus read register. When FFFFBBAA is rotated one byte to the right and latched in the merge register, the merge register contains AAFFFFBB, where BB is the low-byte.

SYNC/READY is received as an input to the MCT branch MUX, so when it is asserted, it causes a jump to microaddress 1C7. Thus, 1C7 is passed through the next address buffer and used to access control store as the next microaddress.

Clear Q22 Bus Function

At the rising edge of MCTM CLK125, the microinstruction at 1C7 is available at the output of the control store. This microinstruction causes:

- the Q22 bus controller to clear the function code at the end of the read block cycle,
- the page portion of the virtual address location in the register file to be driven onto the MCA bus; that is, the hex value DCF000,
- the contents of the adder register to be driven onto the MCA bus; that is, the hex value 012,
- the physical address on the MCA bus (000012) to be latched into the Q22 bus write register, and
- the adder to subtract the constant 2 from the low nine bits on the MCA bus and latch the sum in the adder register; so the adder register now contains 010.

The microprogram control field of this microinstruction disables dispatches, but enables traps and jumps. The branch control field is 100 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction as the next microaddress, modified by the branch conditions

QBUS.TIMEOUT and QBUS.ERROR (see Figure 5-7). Bits <9:0> are 0DA and neither branch condition is asserted. Thus, 0DA is passed through the next address buffer and used to access control store as the next microaddress.

At the rising edge of MCTM CLK125, the microinstruction at 0DA is available at the output of the control store. This microinstruction causes the contents of the merge register (AAFFFFBB) to be driven onto the MCA bus. This is an intermediate cycle to allow a bus timeout error to be detected. The microprogram control field causes a jump to address 1CD.

Second Block Read

At the rising edge of MCTM CLK125, the microinstruction at 1CD is available at the output of the control store. This microinstruction causes:

- the contents of the merge register (AAFFFFBB) to be driven onto the MCA bus,
- the contents of the Q22 bus read register to be driven onto the MCD bus, rotated 3 bytes to the right in the rotator, and bytes 2 and 1 latched in the merge register.

The microprogram control field of this microinstruction causes the microprogram to loop on this instruction until the SYNC/READY signal is received from the Q22 bus, indicating that the second word of data is available and latched in the Q22 bus read register. This second block transfer from the Q22 bus read returns the bytes DDCC and latches them in the Q22 bus read register.

This microinstruction moves the contents of the MCD bus into the rotator. The value FFFFDDCC is stable on the MCD bus at this point; the FFFF is provided by the pull-up resistors, and the DDCC is from the Q22 bus read register. When FFFFDDCC is rotated three bytes to the right and bytes 1 and 2 latched in the merge register, the merge register contains AADDCCBB, where BB is the low-byte. (The AA and BB are still in the merge register from before.) Notice that the requested data CCBB are now aligned as the low-order word of the merge register.

SYNC/READY is received as an input to the MCT branch MUX, so when it is asserted, it causes a jump to microaddress 1CF. Thus, 1CF is passed through the next address buffer and used to access control store as the next microaddress.

At the rising edge of MCTM CLK125, the microinstruction at 1CF is available at the output of the control store. This microinstruction causes the contents of the merge register (AADDCCBB) to be driven onto the MCA bus. This is also an intermediate cycle to allow bus timeout errors to be detected. The microprogram control field causes a jump to address 0E2 if no bus errors occur.

Return Correct Data

At the rising edge of MCTM CLK125, the microinstruction at 0E2 is available at the output of the control store. This microinstruction causes:

- the contents of the merge register, AADDCCBB, to be driven onto the MCA bus as MCA <31:0>,
- the transceiver to drive the data on the MCA bus over the memory data bus to the data path,

- the reverse pass latch to be enabled for input and output, so AADDCCBB is latched there from the MCA bus and driven onto the MCD bus,
- the AADDCCBB from the reverse pass latch to be rotated three bytes to the right and all four bytes latched in the merge register; the merge register now contains DDCCBBAA, where AA is the low-order byte, and
- the MEM BUSY signal to be cleared.

The longword AADDCCBB is returned to the data path over the memory data bus. Because the data path is executing a macroinstruction with data type word (MOVW), the data path will only read the low-order word off the memory data bus. The low-order word is the desired data CCBB and it is, in fact, the first operand.

After being rotated again by this microinstruction and latched in the merge register as DDCCBBAA, the longword is now restored to the correct order for a cache write.

The microprogram control field of this microinstruction disables dispatches, but enables traps and jumps. The branch control field is 100 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction as the next microaddress (see Figure 5-7). Bits <9:0> are 382. Thus, 382 is passed through the next address buffer and used to access control store as the next microaddress.

Prepare for Cache Write

At the rising edge of MCTM CLK125, the microinstruction at 382 is available at the output of the control store. This microinstruction causes:

- the contents of the merge register, DDCCBBAA, to be driven onto the MCA bus as MCA <31:0>, and
- the reverse pass latch to be enabled for input, so DDCCBBAA is once again written into the reverse pass latch.

The microprogram control field of this microinstruction only enables jumps. The branch control field is 100 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction as the next microaddress. Bits <9:0> are 35C. Thus, 35C is passed through the next address buffer and used to access control store as the next microaddress.

Cache Write

At the rising edge of MCTM CLK125, the microinstruction at 35C is available at the output of the control store. This microinstruction causes:

- the contents of the adder register, 010, to be driven onto the MCA bus as MCA <8:0>,
- the page portion of the virtual address location in the register file to be driven onto the MCA bus; so MCA <31:9> are the hex value DCF000,
- the reverse pass latch to be enabled for output, so DDCCBBAA is driven onto the MCD bus, and
- a cache write access.

To accomplish the cache write access, the index MUX selects PA <12:2> from the MCA bus, and provides a one for the high-order bit. These twelve bits form the hex value 804. Thus, location 804 is accessed in the tag RAM and in the TB/cache data RAM.

At the same time, the tag MUX selects physical address bits <21:13> from the MCA bus, supplies five zeros for the high-order bits and passes these fourteen bits, which have the hex value 0180, to the write isolation buffer. From the write isolation buffer, 0180 hex is written into the low-order fourteen bits of tag RAM location 804. The high-order bit—the valid bit—is written as a one, indicating that this is a valid tag, because bit <31> in the microinstruction is a one. The next high-order bit is a spare. Tag RAM location 804 now contains the hex value 8180.

Simultaneously, the data on the MCD bus, DCBA, are written into location 804 in the TB/cache RAM, and the cache write operation is complete.

The microprogram control field of this microinstruction disables dispatches, but enables traps and jumps. The branch control field is 010 (binary), so the MCT microsequencer selects bits <9:0> from the microinstruction as the next microaddress, modified by the branch conditions QBUS.BLOCK, QBUS.SYNCH and QBUS.BUSY (see Figure 5-7). Bits <9:0> are 0A6 and none of the branch conditions are asserted. Thus, 0A6 is passed through the next address buffer and used to access control store as the next microaddress.

The microinstruction at 0A6 enables dispatches, starts the prefetch sequence to refill the prefetch FIFO if the FIFO contains less than eight bytes, and then jumps to the idle state microroutine.

Move Data

After the memory controller clears MEM BUSY and returns the data CCBB on the memory data bus, the data path executes a MOVL microinstruction, which moves CCBB off the memory data bus and into temporary storage in the data path chip. The data path has now obtained the first operand.

Evaluating the Second Operand Specifier

The second operand specifier, 51, was loaded into the IBYTE register many cycles ago—soon after the decode of the first operand specifier, 60, was completed. Now the second operand specifier is decoded. The decode causes the DAP micro-sequencer to use the current microaddress plus one as the next microaddress since the operand specifier 51 indicates register mode.

Once the decode has completed, the data path asserts IB TAKEN and the prefetch FIFO drives the next byte in the instruction stream onto the memory control bus. The memory controller asserts NXT VALID, the data path asserts LOAD I BYTE, and the next I-stream byte is clocked into the IBYTE register.

The microinstruction that follows the Decode is a Move to @pointer 2. Pointer 2 is currently pointing to R1 because bit <26> in the Decode microinstruction just executed is a 1, causing bits <5:0> from the IBYTE register to be stored in pointer 2. When this Move microinstruction is executed, the word of data, CCBB, in temporary storage in the data path chip, is moved into R1. The entire MOVW macroinstruction is now complete.

Chapter 5 describes the memory controller microcode and this chapter describes the memory controller hardware. The next chapter describes the Q22 bus controller microcode and hardware. Although the Q22 bus controller hardware is physically located on the memory controller module, it is described separately because it is an independent micromachine.

Chapter 7
Q22 Bus Controller

The Q22 bus controller accepts function requests from the memory controller microcode and carries them out through its own set of microinstructions. This chapter describes the Q22 bus controller microcode as well as the hardware components on the memory controller module that make up the Q22 bus controller, and how they interact.

Q22 Bus Controller Function Parameters

The Q22 bus controller must receive the following information to carry out the function requested by the memory controller module:

• Function Code	A three bit field from the memory controller microinstruction that defines the function requested by the memory controller microcode.
• Function Request	One bit from the memory controller microinstruction that informs the Q22 bus controller that a function request is pending.
• Go Bit	One bit from the memory controller microinstruction that informs the Q22 bus controller to proceed with the current function request.
• Function	One bit from the memory

Clear Enable	controller microinstruction that enables the Q22 bus controller to clear the current function request.

The Q22 bus controller monitors the Q22 bus for direct memory access (DMA) activity as well as function requests from the memory controller. The Q22 bus controller is the default bus master. It remains in an idle state unless it is servicing a request from the Q22 bus or the memory controller. The Q22 bus controller uses the same 125 ns microcycle as the memory controller.

When the Q22 bus controller receives a request for DMA from a Q22 bus device, the device cannot proceed with the direct memory access until it receives the signal SDMGO from the Q22 bus controller. SDMGO informs the device that its DMA request is granted, and the device then becomes bus master.

When a function request is sent from the memory controller, the 3-bit function code forms part of the 10-bit address that the Q22 bus controller uses to access its control store. The selected microinstruction from control store generates the appropriate signals to control the bus operation that accomplishes the requested function.

Control Store Addresses

At 0 ns of each MCT microcycle, the MCT microinstruction is available at the output of the MCT control store. If the microinstruction contains an asserted function request bit and 3-bit function code, these bits are sent to the Q22 bus controller within the current microcycle; call it microcycle 1.

The address for the data to be read from or written to is also latched in the Q22 bus write register during microcycle 1.

The microsequencer within the Q22 bus controller uses the 3-bit function code to present a 10-bit microaddress to the Q22 bus controller control store slightly before the end of microcycle 1. The Q22 bus controller microinstruction is then available from the controller control store at 0 ns of microcycle 2.

The 10-bit microaddress presented to the Q22 bus controller control store has one of two formats, depending on the high-order bit. The high-order bit of the microaddress is QBUS BUSY; QBUS BUSY is asserted at 0 ns of microcycle 2 if the memory controller sends a function request in microcycle 1. QBUS BUSY is the signal sent to the memory controller branch MUX by the Q22 bus controller to inform the memory controller that the Q22 bus controller is busy with a function request, just as the memory controller sends MEM BUSY to the data path module to indicate that the memory controller is busy with a memory request.

QBUS BUSY Asserted

When QBUS BUSY is asserted, the 10-bit microaddress presented to the Q22 bus controller control store has this format:

9	8 7	6 4	3 0
BUSY = 1	bus state	function code	state bits

Figure 7-1. Microaddress Format, BUSY Asserted

Bus State Field

The bus state is a 2-bit field that is encoded to reflect the state of several signals on the Q22 bus. These signals are:

- SSYNC, which is a signal from the Q22 bus controller to the slave device to indicate that an address has been placed on the bus and a transfer is in process, (the "S" in the signal name indicates that the signal is from the Q22 bus controller to a bus device),
- RRPLY, which is a response from a slave device indicating that it is there and is ready to read or write data, (the first "R" in the signal name indicates that the signal has been synchronized with the 125 ns clock), and
- RREF, which is a response from a slave device indicating that it can accept another block mode transfer; RREF is asserted and negated with RRPLY. (RREF is a synchronized version of BREF; the "B" in the signal name indicates that the signal is from a bus device to the Q22 bus controller.)

Table 6-1 shows the encoding for the bus state field.

Table 7-1. Bus State Field Encoding

<8:7>	Meaning
0	memory is there and can support another block mode transfer
1	SSYNC has been asserted on the bus
2	memory is there and will soon read or

	write data
3	SSYNC has not been asserted on the bus

Function Code Field

This field is bits <47:45> from the MCT microinstruction; they determine the type of Q22 bus operation to be performed. The encoding is shown in Table 5-2, and repeated here in Table 6-2 for convenience.

Table 7-2. Function Code Field

<6:4>	Operation	Mnemonic
000	no operation	
001	write word	DATO
010	write byte	DATOB
011	write block	DATBO
100	read word	DATI
101	read block	DATBI
110	read interrupt vector	
111	read interlocked	DATIO

State Bits

These four bits are the low-order four bits from the previous Q22 bus controller microinstruction.

QBUS BUSY Not Asserted

When QBUS BUSY is not asserted, meaning there is currently no function request from the memory controller, the 10-bit microaddress presented to the Q22 bus controller control store has this format:

9	8	7	6	5	4	3 0
BUSY =0	RRPLY	MREQ	RSYNC	RSACK	RDMR	state bits

Figure 7-2. Microaddress Format, BUSY Not Asserted

where microaddress bits <8:4> are determined by the state of the signals shown in Figure 6-2.

RRPLY is a synchronized signal from the memory to the Q22 bus controller; it informs the controller that the memory is there and that the memory is ready to accept data from the bus (write) or place requested data on the bus (read).

MREQ is the OR of the three function code bits, so it is asserted when any one of the function code bits is asserted.

RSYNC is a synchronized signal from a bus device to the controller to inform the controller that an address is on the bus and a transfer is in process.

RSACK is a synchronized signal from a bus device to the controller to inform the controller that the device is accepting bus mastership.

RDMR is a synchronized signal from a bus device to the controller to request bus mastership in order to perform a DMA transfer.

The state bits are the low-order four bits from the previous Q22 bus controller microinstruction.

Microinstruction Format

Once a location in the controller control store is accessed, a 16-bit microinstruction is available as the control store output at 0 ns of the present 125 ns microcycle. Figure 6-3 shows the microinstruction format, and the following sections describe the fields.

Busy

The signal MCTA QBUS BUSY H is the high-order bit of the Q22 bus controller microinstruction. It is also the high-order bit of the Q22 bus controller microaddress, and one input to the MCT branch MUX.

Enable Parity

This bit when asserted (active low) allows the error logic that is part of the Q22 bus controller to detect a memory parity error.

SDMGO

This bit is the signal MCTA SDMGO H, which stands for send DMA grant output. This signal, when asserted, grants bus mastership to a bus device that just requested a direct memory access. Bus devices request DMA by asserting the signal MCTB RDMR H. The signal RDMR is one of the ten microaddress bits.

SSYNC Hold

The signal SYNC means that an address has just been placed on the bus. When the signal name is preceded by an "S," the Q22 bus controller is generating it. The SSYNC hold bit in the Q22 bus microinstruction, when asserted (active high), causes SSYNC to remain asserted on the bus as long as the signal RRPLY is asserted. RRPLY is the response from a bus device to inform the Q22 bus controller that it is there.

Enable SIAKO

This bit is the signal MCTA EN SIAKO H; SIAKO stands for send interrupt acknowledge output. This signal, when asserted, causes the signal MCTA SIAKO H to be generated 62 ns into the current 125 ns microcycle. SIAKO informs the bus device with the highest priority interrupt request that it has been granted bus mastership.

SDOUT

The signal DOUT stands for data output and means that valid data are available on the bus data/address lines. When the SDOUT bit in the Q22 bus microinstruction is asserted (active high), the Q22 bus controller generates the signal MCTA SDOUT H to inform the slave device that valid data is currently on the bus.

Enable SDIN

The signal DIN stands for data input. When the enable SDIN bit in the Q22 bus microinstruction is asserted (active high), the Q22 bus controller generates the signal MCTA SDIN H at 30 ns into the present 125 ns microcycle. The signal SDIN informs the slave bus device that the Q22 bus controller wants the requested data. The device must respond with RRPLY to indicate that it is ready to place the data on the bus. The assertion of RRPLY causes the negation of SDIN; when SDIN negates, the data on the bus data/adddress lines are latched in the Q22 bus read register.

15	14	13	12	11	10	9	8	7	6	5	4	3 0
BUSY	enable parity	SDMGO	SSYNC hold	enable SIAKO	SDOUT	enable SDIN	enable SSYNC	enable data	address OK	request clear	SYNC/READY	state bits

Figure 7-3. Q22 Bus Controller Microinstruction Format

Enable SSYNC

The enable SSYNC bit is the signal MCTA EN SSYNC H. When the enable SSYNC bit in the Q22 bus microinstruction is asserted (active high), the Q22 bus controller generates the signal MCTA SSYNC H if the go bit from the MCT microinstruction has been received.

If, in microcycle 1, the MCT microinstruction sends a function code and the function request bit to the Q22 bus controller, the Q22 bus controller actually begins the requested operation by accessing the appropriate routine in its control store; the address in the Q22 bus write register is also enabled onto the Q22 bus in microcycle 1.

When the memory controller sends the go bit, the Q22 bus controller saves it in the bus control PAL as MCTB SYNC OK H. The go bit could be sent in microcycle 1, or in one of the three successive microcycles.

The MCTA EN SSYNC H signal from the current Q22 bus controller microinstruction is ANDed with the go bit, or with the latched version of the go bit (MCTB SYNC OK) to generate the signal MCTA SSYNC, which informs the slave bus device that the address is on the bus, and causes the Q22 bus cycle to proceed. Thus, MCTA SSYNC is actually the signal generated from the MCT microinstruction go bit if the enable SSYNC bit in the Q22 bus microinstruction is asserted.

Enable Data

The enable data bit is the signal MCTA ENDATA H. MCTA ENDATA H is one input to the bus control PAL and causes the signal MCTB ENDAL

L to be asserted as an output from the PAL. MCTB ENDAL L causes the data in the Q22 bus write register to be driven onto the Q22 bus as DAL <15:00>.

Address OK

The address OK bit is the signal MCTA ADDOK H. This signal is always asserted while the Q22 bus controller is in the idle state. MCTA ADDOK enables the address in the Q22 bus write register on the Q22 bus; ADDOKis another input to the bus control PAL .

When ADDOK is asserted and a function request is received from the memory controller, the control PAL asserts MCTB ENDALADD L and MCTB ENDAL L. ENDALADD L causes bits <22:16> in the Q22 bus write register to be driven onto the Q22 bus as BBS7 and DAL <21:16> (bit <22> in the write register is MCD <29>, which when set indicates the address is located in I/O space, and is asserted on the Q22 bus as the signal BBS7); ENDAL L causes bits <15:0> in the Q22 bus write register to be driven onto the Q22 bus as DAL <15:00>.

Request Clear

The request clear bit is the signal MCTA CLRREQ H. CLRREQ is also one input to the Q22 bus control PAL. When CLRREQ is asserted from the Q22 bus controller microinstruction, and bit <49>, the clear function code bit, is received from the MCT microinstruction, the control PAL generates the signal MCTB CLRFUN H which clears the last function code received from the memory controller.

MCTA CLRREQ is also used to clear bus timeout errors.

SYNC/READY

The SYNC/READY bit is the signal MCTA SYNC/READY H. The SYNC/READY signal communicates the Q22 bus controller's status to the memory controller during reads and writes. Its meaning depends on the operation being performed; the Q22 bus controller asserts SYNC/READY when data are available on a read from a Q22 bus device, and when data or address is needed for a write to a Q22 bus device.

Microstate Bits

These four bits are the low-order bits of the next microaddress that the Q22 bus controller microsequencer presents to the Q22 bus controller control store.

Overview of Q22 Bus Controller Functions

The Q22 bus controller performs the following four functions:

- control microinstruction flow
- service function requests from the memory controller
- monitor direct memory accesses
- communicate with the memory controller

The next sections describe these functions, and the hardware components that implement them, in detail.

Controlling the Microinstruction Flow

The Q22 bus controller has its own microsequencer logic to generate the next microaddress. The microsequencer is a 16L8 PAL that supplies bits <8:4> of the next microaddress. The high-order bit and the four low-order bits are supplied by the previous microinstruction.

The following signals are inputs to the microsequencer PAL from the Q22 bus:

- RDMR—a bus device is requesting bus mastership for a direct memory access,
- RSACK—a bus device is accepting bus masterhip; this signal is issued in response to the signal SDMGO from the Q22 bus controller,
- RSYNC—a bus device has placed an address on the bus and a transfer is in progress,
- BREF—a slave bus device that supports block mode transfers is ready for the next word in the block read or write,
- RREF—a version of BREF synchronized with MCTM CLK125,
- RRPLY—a slave bus device is acknowledging its presence and indicating that it will accept data from the bus or that it will place its data on the bus (depending on the current operation).

The following signals are the remaining inputs to the microsequencer PAL:

- SSYNC—a valid address is on the bus; generated from bit <8> of the previous microinstruction if the go bit has been received,

- MFUN <2:0>—the 3-bit function code field from the current MCT microinstruction,
- QBUS BUSY—the high-order bit from the previous Q22 bus controller microinstruction,
- TIMEOUT—a bus operation has exceeded the 10 microsecond time limit.

If QBUS BUSY is asserted, the microsequencer supplies the MFUN signals and a 2-bit encoded field indicating the state of SSYNC, RRPLY, and RREF, as microaddress bits <8:4> (see Figure 6-1).

If QBUS BUSY is not asserted, the microsequencer presents the signals RRPLY, MREQ, RSYNC, RSACK, and RDMR as microaddress bits <8:4> (see Figure 6-2).

Servicing MCT Function Requests

One of the main functions of the Q22 bus controller is carrying out requests from the memory controller. This is accomplished by the control store microinstructions plus the Q22 bus control PAL and the bus transceivers. The control PAL and the bus transceivers are described in the next paragraphs.

Q22 Bus Control PAL

The control PAL is a 16L8 PAL that has several functions:

- it enables data and addresses onto the Q22 bus,

- it clears the current memory controller function request,
- it saves the go bit from the memory controller microinstruction, and
- it controls block read and write byte bus cycles.

The inputs to the control PAL are: the function code field, the go bit, and the clear enable bit from the MCT microinstruction; the enable data, address OK and request clear signals from the Q22 bus controller microinstruction; and the RDMR signal from the Q22 bus.

If the enable data signal is asserted, the control PAL asserts MCTB ENDAL L which causes the contents of the Q22 bus write register to be driven onto the Q22 bus as DAL <15:00>. If both the enable data and the address OK signals are asserted, the control PAL asserts MCTB ENDAL L and MCTB ENDALADD L which together cause the contents of the Q22 bus write register to be driven onto the Q22 bus as BBS7 and DAL <21:00>.

If the request clear and clear enable signals are asserted, the control PAL asserts the signal MCTB CLRFUN H which clears the last function code received from the memory controller.

When the go bit is asserted by the memory controller microcode, the control PAL saves it as MCTB SYNC OK H.

If the 3-bit function code specifies a read block operation, the control PAL asserts the signal SBS7 with the first data transfer until the last transfer to indicate to the block mode slave device that there will be subsequent transfers.

If the 3-bit function code specifies a write word, write byte or write block operation, the control PAL asserts the signal MCTB SWTBT H during the address portion of the cycle to indicate that an output cycle is to follow rather than an input cycle. During the data portion of a write byte (DATOB) or a read/modify/write byte (DATIOB) bus cycle, the control PAL asserts SWTBT to indicate a byte rather than a word transfer is to take place.

Q22 Bus Transceivers

Six quad registered transceivers and three quad transceivers are the hardware components that make up the Q22 bus read and write registers. The transceivers are bidirectional, allowing data and memory addresses to be transmitted to and received from the memory controller.

The memory controller microcode controls the input side of the write register transceivers, and the output side of the read register transceivers. Thus, the memory controller microcode causes addresses and data to be latched in the write register, and to be driven on the MCD bus from the read register.

The Q22 bus control PAL controls the output side of the write register transceivers with the signal ENDAL, which controls bits <15:0>, and the signal ENDALADD, which controls bits <21:16>.

Data are latched in the Q22 bus read register from the Q22 bus on the falling edge of the signal MCTA SDIN H, which is generated by the Q22 bus controller. An address is latched in the Q22 bus read register on the falling edge of the signal BSYNC when BSACK and BWTBT are asserted.

BSYNC is generated by a bus master device to indicate that it has placed an address on the bus; BSACK is asserted by a bus device when it has assumed bus mastership.

Arbitrating the Q22 Bus

Another major function of the Q22 bus controller is to arbitrate the Q22 bus; that is, the Q22 bus controller monitors the bus as well as the memory controller and decides when to assume bus mastership to execute a function request from the memory controller, and when to grant bus mastership to allow DMA transfers.

The hardware involved in arbitrating the bus is the microsequencer, the cache invalidate logic, and the bus error logic. These components are described in the next paragraphs.

Q22 Bus Controller Microsequencer

DMA requests as well as memory function requests are the inputs to the Q22 bus controller microsequencer. The microsequencer logic generates the next microaddress as the output; the microinstruction routine selected by the micro-address causes the Q22 bus controller to assume bus mastership, or to grant bus mastership. Thus, in controlling the microinstruction flow, the microsequencer is also arbitrating the bus.

When a Q22 bus device wishes to read from or write to physical memory, it asserts the signal RDMR, which is one input to the microsequencer. If the Q22 bus controller is in an idle state when RDMR is asserted, the controller grants bus mastership by asserting the signal SDMGO. The bus device assumes bus mastership by asserting RSACK. As long as the bus device asserts RSACK, it retains bus mastership. The bus device relinquishes bus mastership by negating RSACK.

If the memory controller sends a function request while some other bus device is bus master, the Q22 bus controller is aware of the request because the three function code signals are ORed to form bit <7>, MREQ, in the next microaddress (see Figure 6-2).

When the Q22 bus controller is in the idle state, the signal ADDOK (address OK) is always asserted. If a function request from the memory controller is received, ADDOK plus the ENDATA signal from the Q22 bus controller microinstruction cause ENDALADD and ENDAL to be asserted. This locks out RDMR, so the function request must be completed before the Q22 bus controller relinquishes bus mastership.

Thus, when the Q22 bus controller is in the idle state, a DMA request takes precedence over a function request from the memory controller if the DMA request is received first, or at the same time. Once the Q22 bus controller assumes bus mastership to execute a function request, however, DMA requests are locked out until the request is completed.

Cache Invalidate Logic

The cache invalidate logic is part of the Q22 bus interface on the memory controller module. This logic monitors the RSACK signal from the bus; when it is asserted, the cache invalidate logic enables the input side of the Q22 bus read register. When the bus master device asserts BSYNC, indicating it has just placed an address on the bus, the cache invalidate logic latches the Q22 bus read register, capturing the DMA address on the bus.

The combination of RSACK asserted and BSYNC asserted also sets the cache invalidate flag. This signal is one input to the MCT microinstruction decode logic, and causes the MCT to trap to a microroutine that reads the address out of the Q22 bus read register, checks if the address is in the cache, and marks that cache entry invalid if it is.

Bus Error Logic

The Q22 bus interface contains hardware to detect parity errors on the bus, and bus timeouts. The bus error logic ORs these two error conditions together and saves them in a state element. If either error is set during the execution of a function request, the function request is cleared, and the signal MCTB QBUS ERROR is asserted; this signal is one input to the MCT branch MUX. The error conditions are cleared at the start of any function request from the memory controller.

The bus timeout logic limits the length of time the Q22 bus controller waits for a reply from a slave or a DMA device. The timeout limit is 10 microseconds. The timer is started by the assertion of SDIN, SDOUT, or SDMGO. The timer is reset by the assertion of BRPLY (in response to SDIN or SDOUT) or by the assertion of BSACK (in response to SDMGO).

On the Q22 bus, DAL <17> and <16> are used for parity detection. During the portion of the data transfer bus cycles in which data is being placed on the bus by the slave for the bus master, <DAL> 17 is asserted to enable parity error detection logic, and <DAL> 16 is asserted when a parity error occurs. If a parity error has occurred and the enable parity signal is asserted from the current Q22 bus controller microinstruction (bit <14>), the parity detection logic asserts MCTB QBUS ERROR.

Communicating with the MCT

Communicating its progress to the memory controller during the execution of a function request is the fourth function performed by the Q22 bus controller. There are six status signals generated by various pieces of the Q22 bus controller hardware; each signal is described in the following paragraphs.

Block Mode

The signal MCTA BLOCK MODE OK is asserted during a write block or read block bus operation to signify that the memory is able to handle the next data transfer as a block mode transfer. The signal is generated by the 16L8 logic PAL in the Q22 bus controller microsequencer, and is one input to the memory controller branch MUX.

During a read block transfer, the block mode OK flag is only used by the Q22 bus controller since the memory controller microcode always loads the next address after the data from the previous transfer is latched.

During a write block operation, both the Q22 bus controller and the memory controller use this flag to determine whether data or address should be loaded into the Q22 bus write register.

SYNC/READY

The signal MCTA SYNC/READY is true 30 ns maximum after the rising edge of the 125 ns clock to allow the Q22 bus controller to coordinate read and write events on the bus with the memory controller. SYNC/READY is asserted when data are available on a read from the Q22 bus, and when data or address is needed for a write to a Q22 bus device. The signal is generated by bit <4> in the current Q22 bus controller microinstruction, and is one input to the memory controller branch MUX.

For block read transfers, SYNC/READY is asserted during the cycle that data are loaded in the Q22 bus read register and remains asserted for one 125 ns microcycle. If block mode is not asserted with SYNC/READY, the next address that is always loaded by the memory controller microcode for block read operations is used to start another microcycle.

During a write cycle, the word to be written is placed in the Q22 bus write register during the microcycle following the first assertion of SYNC/READY. If the clear request enable signal is not asserted at this time, the address of the next word is loaded during the cycle following the next assertion of SYNC/READY. The cycle is then repeated for each subsequent word transfer.

During block write operations, the block mode OK signal is used with SYNC/READY to coordinate the loading of data or address into the Q22 bus write register. The memory controller microcode loads the first word in the Q22 bus write register during the microcycle following the assertion of

SYNC/READY. If block mode is set when SYNC/READY is reasserted, the next word is loaded into the write register during the next microcycle. If block mode is not asserted, the next address is loaded into the Q22 bus write register.

Q22 Bus Busy

The signal MCTA QBUS BUSY is asserted while the Q22 bus controller is executing a function request from the memory controller microcode. This flag is set and reset 25 ns maximum after the rising edge of the 125 ns clock. QBUS BUSY is asserted when the Q22 bus controller begins a bus transfer. It is deasserted the cycle after RRPLY is asserted from the slave device, or the cycle after a bus timeout error occurs. During the time that busy is asserted, the memory controller microcode cannot start a new request. MCTA QBUS BUSY is generated by bit <15> in the current Q22 bus controller microinstruction, and is one input to the memory controller branch MUX.

Q22 Bus Timeout

The signal MCTB TIMEOUT is true 10 ns maximum after the rising edge of the 125 ns clock, when a nonexistent memory request bus timeout occurs. This signal can be read within one microcycle after the assertion of SYNC/READY. The signal is true for 250 ns before the Q22 bus controller clears it. MCTA TIMEOUT is generated by the timeout logic and is one input to the memory controller branch MUX.

Q22 Bus Error

The signal MCTB QBUS ERROR is the OR of the bus timeout and parity error signals. Once set, it is cleared at the start of a new function request. This signal causes any single or multiple memory function request to abort. MCTB QBUS ERROR is generated by the bus error logic and is one input to the memory controller branch MUX.

A Q22 bus parity error (MCTB QBUS ERROR and not MCTB TIMEOUT) can be detected 10 ns after the start of the microcycle following SYNC/READY for a bus read transfer. Bus timeout is true when SYNC/READY is asserted after the start of the data portion of the bus cycle.

Cache Invalidate

The signal MCTB CACHE INV is asserted during DMA write cycles. This signal is synchronized to the 125 ns clock and holds the bus reply line asserted until the memory controller microcode acknowledges the cache invalidate signal by reading the address in the Q22 bus read register. MCTB CACHE INV is generated by the cache invalidate logic and is one input to the memory controller microinstruction decode logic.

Q22 Bus Operations

This section contains brief descriptions of the bus operations handled by the Q22 bus controller as the result of function requests from the memory controller.

Write Byte

Write Word