Company Confidential

Seahorse Central Processing Unit Technical Description

First Draft - May 1983

d i g i t a l

May 1983

This manual describes the KD32-AA central processing unit used in Seahorse.

KD32-AA Central Processing Technical Description

Document Order Number: XX-12345-XX

Version: 0.0

digital equipment corporation
maynard, massachusetts

First Draft Printing, May 1983

The material in this manual is for informational purposes and is subject to change without notice. The information in this document should not be construed as a commitment by Digital Equipment Corporation.

Digital Equipment Corporation assumes no responsibility for any errors which may appear in this manual.

The postage-paid READER'S COMMENTS form on the last page of this document requests your critical evaluation to assist us in preparing future documentation.

Contents

List of Figures

List of Tables

Preface

Manual Scope

This manual is a technical description of the KD32-AA central processing unit (CPU) used in the Seahorse system. It is intended as a field reference for DIGITAL Field Service personnel and a resource for training programs conducted by Educational Services and Manufacturing. A knowledge of VAX architecture is assumed.

Chapter 1 is a general description of the Seahorse system. The remaining chapters describe the microcode and the hardware for the two modules that make up the CPU board set:

- Data Path Module M7135
- Memory Controller Module M7136

Related Documentation

The *Seahorse Central Processing Unit Technical Description* is part of the hardware documentation set for the Seahorse system. Related manuals that may be of interest are:

- *Seahorse Owner's Manual.* This book contains site preparation, installation, operation, diagnostics, and system configuration information for the Seahorse system.
- *NPM Handbook.* This book contains ?
- *VAX Architecture Handbook.* The Seahorse

system design is based on the VAX architecture described in this handbook.

- *Microcomputer Interfaces Handbook.* This handbook is a reference guide for the interface and peripheral hardware options that can be installed on the Extended LSI-11 Bus used in Seahorse .
- *Microcomputers and Memories.* This manual contains a detailed description of the Extended LSI-11 Bus.
- *Seaboard Manuals?*
- *MicroVMS Manuals?*

Chapter 1
Introduction

Chapter 1 is a general description of the Seahorse system. It contains information about the system necessary for understanding the Seahorse central processing unit (CPU). The remaining chapters provide a detailed technical description of the KD32-AA CPU.

System Overview

The Seahorse system is a 32-bit, high-performance, microprogrammed computer. The processor executes a subset of the native VAX instruction set and contains an interface to the extended LSI-11 bus (Q22 bus). PDP-11 compatibility mode is not supported.

The major components of the Seahorse system, shown in Figure 1-1, are:

- KD32-AA CPU:
  - memory controller (MCT)
  - data path (DAP)
- Q22 bus
- RQDX1 controller
- RX50 floppy disk drives
- RD51 or RDXX fixed disk
- Q22 memory, with block mode capability
- console terminal
- front control panel

KD32-AA CPU

The KD32-AA CPU implements the MicroVAX architecture on two quad-height modules, and provides an interface to the Q22 bus. As such, it contains:

- an interface to the Q22 bus which supports block mode transfers and up to four megabytes of physical memory
- an 8 KB direct-mapped cache
- a 512 entry (longword) translation buffer
- a 10 ms interval timer
- a console serial line unit
- an 8 KB boot PROM

Q22 Bus

The Seahorse system backplane uses the extended LSI-11 bus (also called the Q22 bus), which has 22-bit addressing. The Q22 bus consists of 42 bidirectional and 2 unidirectional signal lines. These are the lines along which the processor, memory, and I/O devices communicate with each other. Seahorse performs the following Q22 bus data transfer functions:

DATI	read word
DATO	write word
DATOB	write byte
DATIO	read, modify, write word
DATIOB	read, modify, write byte
DATBI	read block

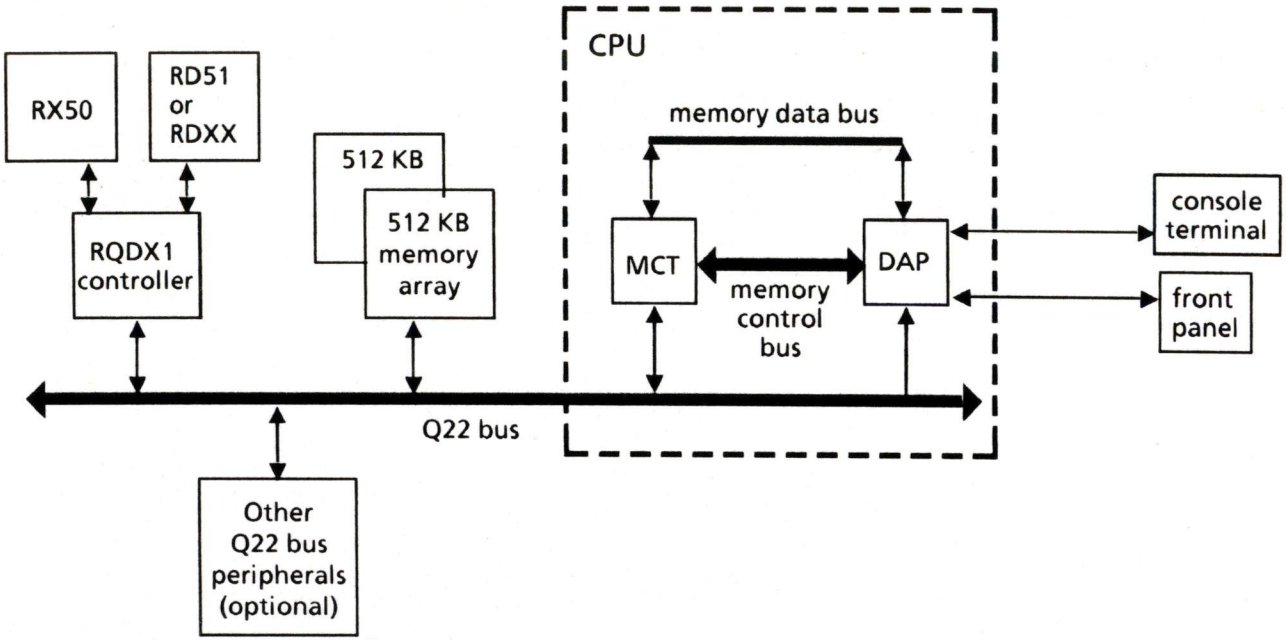


Figure 1-1. Seahorse System

DATBO write block

RQDX1 Controller

The RQDX1 controller (M8639) is a quad-height module that occupies the last-used slot in the backplane. It is the interface between the Q22 bus and the disk drives (floppy and fixed). The controller is a direct memory access (DMA) interface and uses mass storage control protocol (MSCP).

RX50 Floppy Disk Drive

The RX50 is a random access storage device with two floppy disk drives. It uses single-sided 5.25 inch (13.34 cm) diskettes. The total drive capacity is 800K bytes of formatted data. Each drive has an access door and slot for inserting and removing diskettes. A head load LED for each diskette slot informs the user when that unit is busy.

The RX50 is a field replaceable unit (FRU) that mounts in the Seahorse system box. One cable (part number xxxxx) connects the RX50 to the RQDX1 controller. Another cable (part number xxxxx) connects the RX50 to the power supply. See the *Seahorse Owner's Manual* for removal and replacement procedures.

RD51 and RDXX Fixed Disk Drives

The RD51 is a random access storage device which uses two nonremovable 5.25 inch (13.34 cm) disks as storage media. One movable head per disk surface services 153 data tracks. The total formatted capacity of the four heads and surfaces is 10 megabytes.

The RDXX is a random access storage device which uses one nonremovable 5.25 inch (13.34 cm) disk as storage media. The total formatted capacity is 25 MB.

The RD51 and RDXX are field replaceable units (FRUs) that mount in the Seahorse system box. One cable (part number xxxxx) connects the RD51 or RDXX drive to the RQDX1 controller. Another cable (part number xxxxx) connects the RD51 or RDXX drive to the power supply. See the *Seahorse Owner's Manual* for removal and replacement procedures.

Memory

Seahorse relies on block mode Q22 bus data transfer functions to realize its performance goals. Therefore, although the KD32-AA CPU is compatible with other extended LSI-11 bus memories, Seahorse systems support only those Q22 memories that have block mode capability. Both the MSV11-P and the MSV11-J memory families offer this capability.

MSV11-P Memory

The MSV11-P family of memory modules are quad-height modules that implement an 18-bit wide random access memory array (16 data bits and 2 parity bits), parity generation and detection, and on-board refresh circuitry. There are three variations:

MSV11-PL	512 KB of storage using 64K MOS RAMs
MSV11-PK	256 KB of storage using 64K MOS RAMs

MSV11-PF	128 KB of storage using 16K MOS RAMs

MSV11-J Memory

The MSV11-J memory is a quad-height module that implements two 22-bit blocks of random access memory (16 data bits and 6 ECC bits). Both blocks of memory are read simultaneously on reads (so that each access retrieves a longword of data from the array), and the requested word is transferred on the bus, while the other word is latched on the memory module. This provides the potential for a reduction in access time if this word is subsequently requested. On a write, only one block is accessed. The MSV11-J also implements ECC (single-bit correction) and on-board refresh logic. There are two variations:

MSV11-JA	512 KB of storage using 64K RAMs
MSV11-JB	2 MB of storage using 256K RAMs

An MSV11-J memory module must be plugged into a Q22/CD slot in the backplane.

Console Terminal

The console terminal may be any member of the VT100 or VT200 family of terminals. A cable connects the terminal to a serial line unit (SLU) connector on the rear of the Seahorse system box. A terminal interface UART and an RS232 driver and receiver pair are located on the data path module (M7135). The baud rate is set by DIP switches on the data path module; the choices are 300, 1200, 9600, and 19.2K baud.

Front Control Panel

The front panel provides control and status of the various components of the system. The switches and indicators are shown in Figure 1-2. The switches and their functions are listed in Table 1-1. The indicators and their meanings are listed in Table 1-2.

Power Supply

The power supply (H7864) is a modular, 230-watt power supply that supplies from 4.5 amps minimum to 36 amps maximum at +5 volts, and 0 to 6 amps at +12 volts. There are also two outputs designed to accommodate DC brushless fans, not included in the 230-watt power specification. These outputs supply 0.45 amps at +12 volts and +9 volts.

Additional power supply features include thermal shut down, overvoltage and overcurrent protection, AC input transient suppression, and three Q22 bus signals (BPOK H, BDCOK H, BEVNT L).

The power supply includes connectors that provide the necessary power and signal interfaces to the logic backplane, mass storage units, front panel, and fans.

The power supply is a field replaceable unit (FRU). See the *Seahorse Owner's Manual* for removal and replacement procedures.

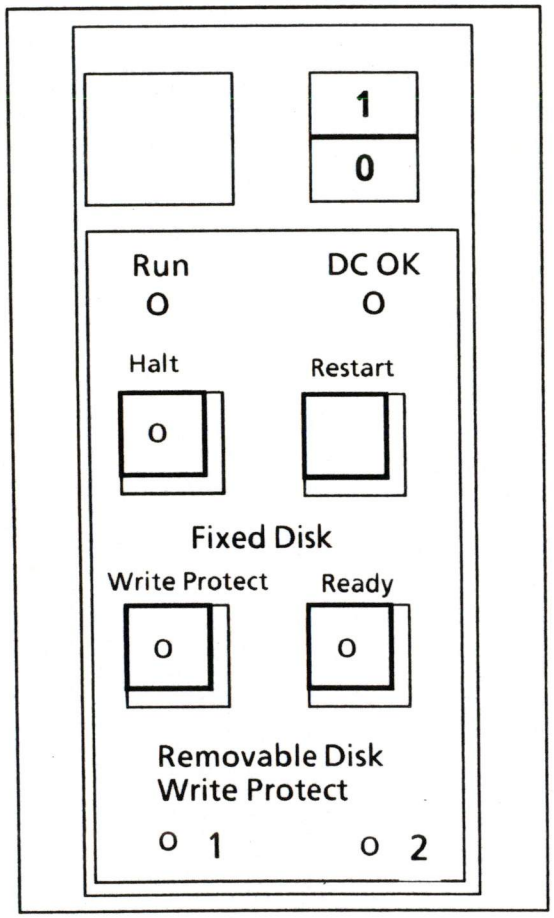


Figure 1-2. Seahorse Front Panel

Table 1-1. Front Panel Switches

Switch	Position	Function
1, 0	1	Turns on the system power.
	0	Turns off the system power.
Halt	In (LED lit)	The processor halts and responds to console commands.
	Out	Enables the processor to run.
Restart	In (momentary switch)	When the halt switch is out (LED off), the processor carries out a power-up sequence. When the halt switch is in (LED lit), this button has no effect.
RD Protect	In (LED lit)	Write protects the fixed disk.
	Out (LED off)	Enables writing to the fixed disk.
RD Ready	In (LED off)	Places the fixed disk off-line.
	Out (LED lit)	Places the fixed disk on-line.

Table 1-2. Front Panel Indicators

LED	Function
Run	The Run LED is on when the processor is operating; the LED goes off when the processor is not executing instructions.
DC OK	This LED is on when the power supply is generating correct DC power output voltages.
Removable Disk Write Protect	
1	When lit indicates the floppy in drive 1 is write protected.
2	When lit indicates the floppy in drive 2 is write protected.

Backplane

The backplane (H9278-A) is a four-row by eight slot backplane capable of accepting either quad- or double-height modules. The backplane uses the Q22 bus structure in the A and B connectors of slots 1 through 8, and in the C and D connectors of slots 4 through 8. A slot-to-slot interconnection scheme (referred to as the CD interconnect) is wired in the C and D connectors of slots 1 through 3. The CD interconnect connects selected side two pins in rows C and D of a given slot to side one pins of the slot immediately following. There are 32 such connections per slot.

The backplane receives and distributes two voltages and ground. Maximum ratings are +5 volts at 36 amps, and +12 volts at 6 amps.

The backplane includes four connectors, J1 through J4, which are mounted on side two of the backplane. J1 (eighteen pins), J3 (four pins), and J4 (four pins), connect power supply outputs to the backplane. J2 (ten pins) connects the backplane to the front panel.

The backplane also includes provision for the insertion of four resistor packs (p/n 1318110-00) into positions XZ1, XZ2, XZ3, and XZ4. In a Seahorse system (single backplane), these resistor packs are inserted to terminate the Q22 bus lines. (Characteristic impedance is 220 ohms).

Figure 1-3 shows the backplane organization. The numbers in the parentheses following the Q22 designations show the path of interrupt and direct memory access grant continuity for options installed in the backplane; increasing value denotes lower priority. The BIAKO L/BIAKI L

and BDMGO L/BDMGI L signals are daisy-chained. Therefore, each slot requires the insertion of a module to pass these grant signals on, as no jumpers are provided on the backplane for this purpose.

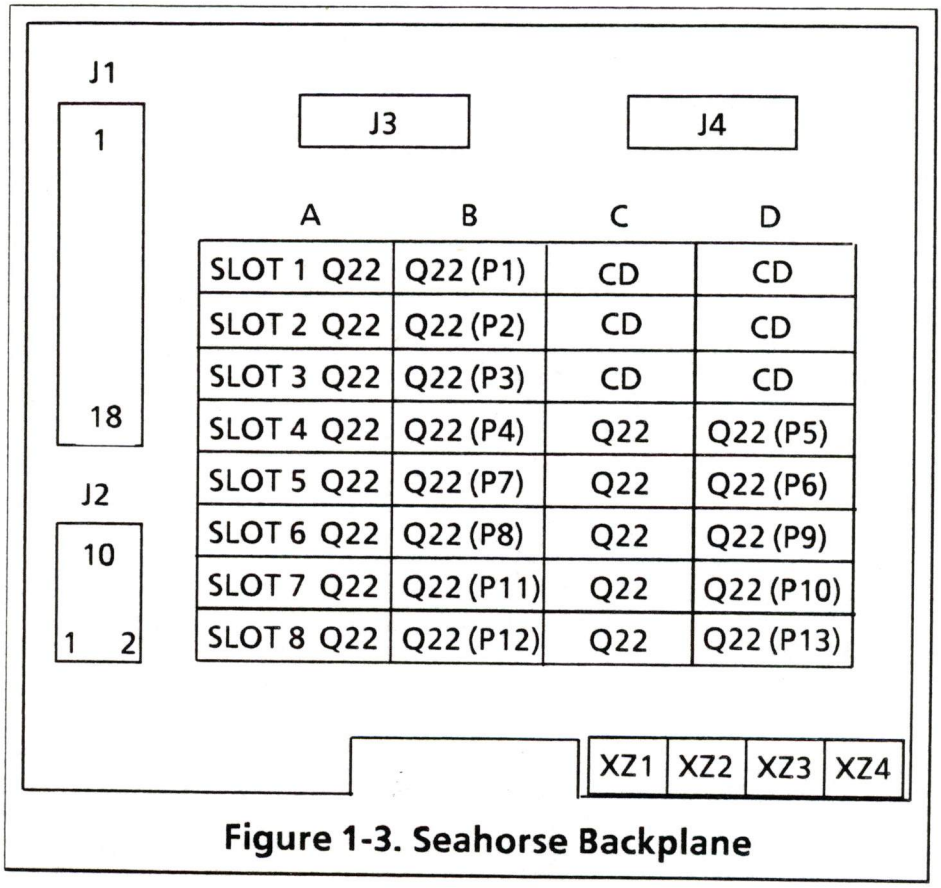


Figure 1-3. Seahorse Backplane

Patch Panel Assembly

External option cables and serial lines connect to Seahorse through the patch panel assembly. The patch panel provides shielding for EMI and accommodates a variety of connectors by providing four inlay patch panel areas. Connectors are

mounted to these patch panels, and the patch panels screw to the back of the patch panel sheet-metal frame.

The four inlay patch panel areas are configured as follows:

Patch Panel 1	Two 25-pin EIA connectors and two rotary switches for baud rate selection (M8189). **Note:** The baud rate select switches are disabled. Baud rate is set by DIP switches on the data path module.
Patch Panel 2	Four 25-pin EIA connectors
Patch Panel 3	The post between two of the patch panel areas may be removed to install three 40- or 50-pin EIA connectors.
Patch Panel 4	Two more cutouts are present to accommodate one 40- or 50-pin EIA connector each.

System Architecture

Seahorse implements the MicroVAX architecture which is a strict subset of the VAX architecture. The MicroVAX architecture features are:

- a 4 gigabyte virtual address space
- 32-bit word size
- sixteen 32-bit general purpose registers
- 32 interrupt levels
- vectored hardware and software interrupts

- 21 addressing modes
- variable instruction size
- full memory management
  - virtual to physical address translation
  - page protection mechanism
- stack processing
- emulation support for the full VAX instruction set (except PDP-11 compatibility mode)

The differences between the VAX and MicroVAX architectures are as follows.

- MicroVAX supports a subset of VAX data types:
  - no D_floating data type
  - no H_floating data type
  - no octaword data type
  - no numeric string data types
  - no packed decimal data type
- MicroVAX supports a subset of the VAX instruction set and provides emulation support for nonimplemented instructions (except compatibility mode):
  - no decimal string instructions
  - no character string instructions (except LOCC, MOVC3, MOVC5, SCANC, SKPC, and SPANC)
  - no EDITPC or CRC instructions
  - no compatibility mode instructions
  - no D_floating or H_floating instructions

- MicroVAX physical addresses can be up to 30 bits long. A physical address on Seahorse is 23 bits long, allowing a physical address space of eight megabytes; four megabytes are in memory space (low-addressed half), and four megabytes are in I/O space (high-addressed half).

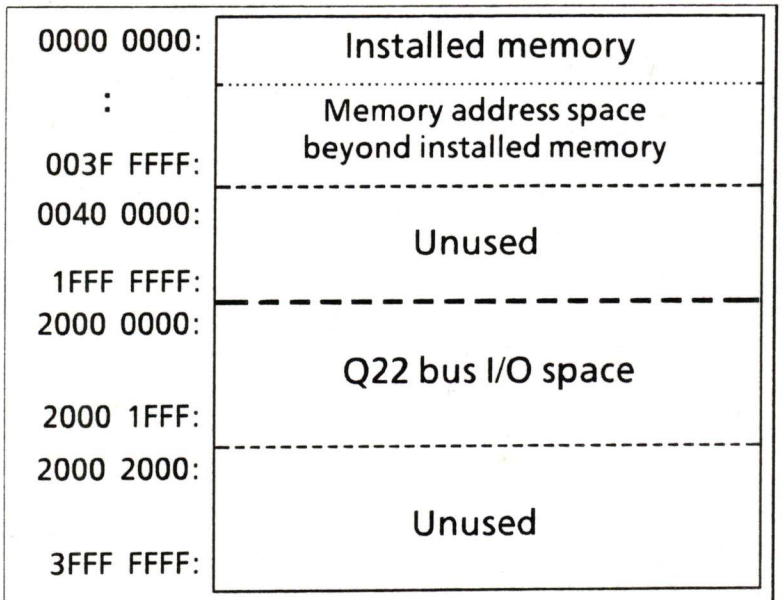


Figure 1-4. Seahorse Physical Memory

- MicroVAX supports a subset of VAX processor registers. The following eleven internal processor registers (IPRs) are not implemented by the MicroVAX architecture:

ICCS	interval clock control/status register
NICR	next interval count register
ICR	interval count register
TODR	time of year register

RXCS	console receive control status
RXDB	console receive data buffer
TXCS	console transmit control status
TXDB	console transmit data buffer
TBIS	translation buffer invalidate single
PMR	performance monitor enable
TBCHK	translation buffer check

Of these eleven IPRs, Seahorse implements the following six as defined by the VAX architecture:

RXCS	console receive control status
RXDB	console receive data buffer
TXCS	console transmit control status
TXDB	console transmit data buffer
TBIS	translation buffer invalidate single
TBCHK	translation buffer check

Seahorse implements the following registers uniquely:

ICCS	interval clock control/status register
CADR	cache disable
MCESR	machine check error summary

IORESET initialize bus

The differences between the VAX and MicroVAX architectures have been pointed out here. For more information about the VAX architecture, see the *VAX Architecture Handbook*, EB-19580-20.

System Timing

The Seahorse system clocks are generated on the CPU memory controller module (MCT). A basic clock with a 64 Mhz frequency is generated by a crystal oscillator. All the other clocks in the data path (DAP) and memory controller (MCT) modules are derived from this basic clock.

The system clock (CPU CLOCK H) has a symmetrical 250 nanosecond period. The start of a microcycle is defined as occurring on the leading edge of this clock and is referred to as T0. The trailing edge of the clock occurs 125 ns later.

Seahorse is a pipelined, microprogrammed machine. The basic microcycle is 250 ns, and the pipeline is one deep. A new microinstruction is accessed every 250 ns, and requires two 250 ns microcycles to complete. The first 250 ns is DECODE, and the second 250 ns is EXECUTE. The EXECUTE microcycle of the first microinstruction is overlapped with the DECODE microcycle of the next microinstruction. Thus, one microinstruction is retired every 250 ns. This timing is illustrated in Figure 1-5.

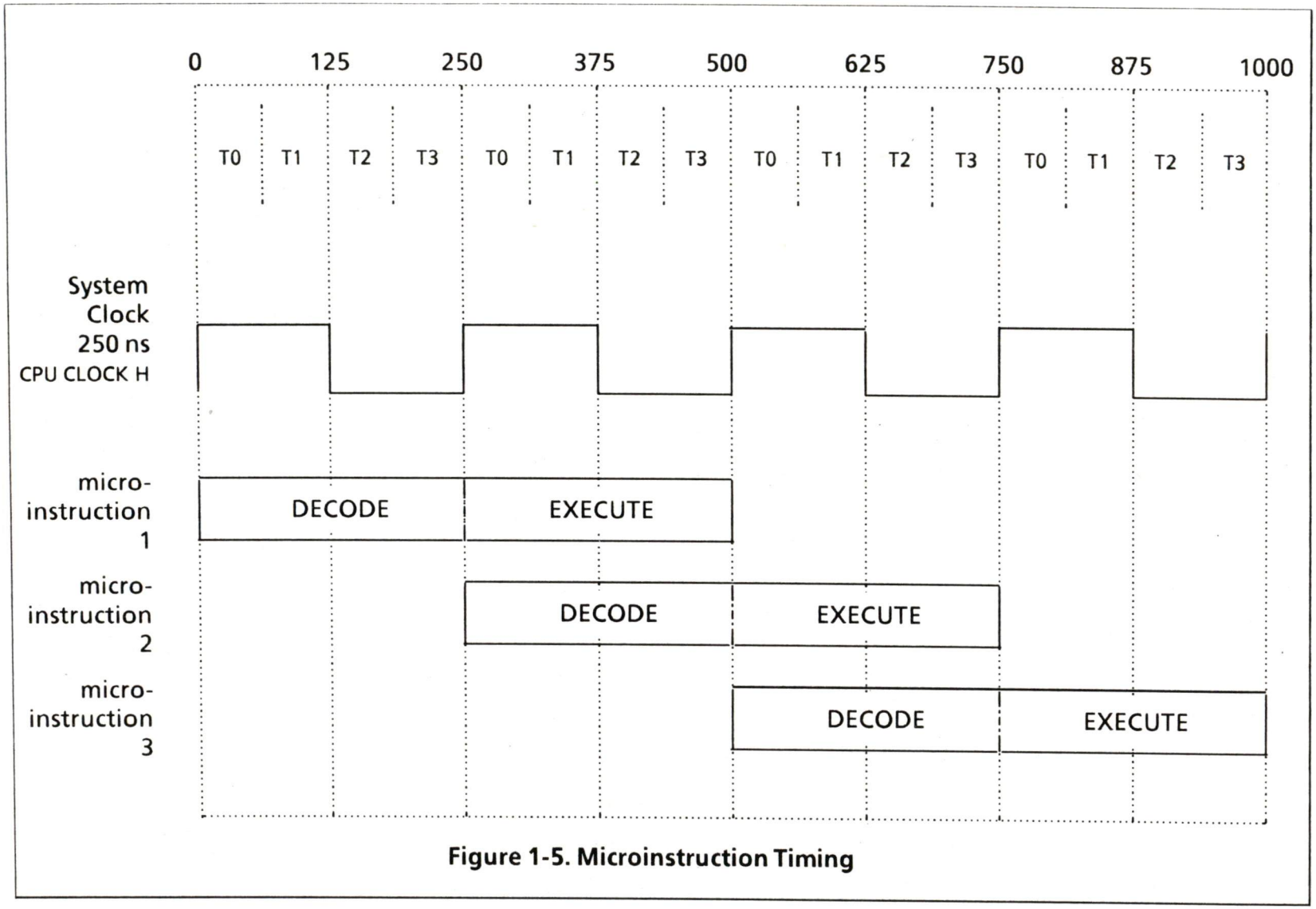


Figure 1-5. Microinstruction Timing

System Bus Summary

The system buses which interconnect the modules in the Seahorse system are the memory data bus (MDB), the memory control bus (MCB), and the extended LSI-11 bus (Q22 bus). (Those buses that are completely contained within a module are not discussed in this section.)

The memory data bus and the memory control bus connect the two CPU modules (DAP and MCT). The memory data bus is implemented using an over-the-top 50-pin cable. It is a 32-bit bidirectional data bus. The 8-bit memory control bus is implemented using the CD interconnect on the backplane. The remaining lines on the CD interconnect are used for clock distribution, status, and miscellaneous control logic. Slots 1 and 2 on the backplane are reserved for the two CPU modules as both must be placed in Q22/CD slots (see Figure 1-3).

The Q22 bus connects the CPU to the system's peripheral I/O devices. Four basic kinds of transactions take place on the bus:

- power up/down signal sequencing
- transfer of bus mastership from the CPU to a direct memory access (DMA) device
- transfer of data between a bus master and a slave
- interrupts to the CPU

Most of the bus interface logic is located on the memory controller module. The data path module contains logic to handle power up and down signal sequencing and interrupts.

The 42 signal lines used in the Q22 bus are :

- Sixteen data/address lines—BDAL<15:00>
- Two address/parity lines—BDAL<17:16>
- Four extended address lines—BDAL<21:18>
- Six data transfer control lines—BBS7, BDIN, BDOUT, BRPLY, BSYNC, BWTBT
- Six system control lines—BHALT, BREF, BEVNT,BINIT, BDCOK, BPOK
- Eight interrupt control and direct memory access control lines—BIAKO/BIAKI, BIRQ4, BIRQ5, BIRQ6, BIRQ7, BDMGO/BDMGI, BDMR, BSACK

All Q22 bus signals are asserted low and negated high, except BPOK and BDCOK, which are asserted high and negated low to indicate an event such as impending loss of power.

With the exception of DMA grant and interrupt acknowledge signals, Q22 bus signals are bidirectional; that is, they can be driven or received at any point along the signal line. When driven, bidirectional signals travel from the driver to the near end terminator, and from the driver to the far end terminator. The exceptions are BIAKO L, BIAKI L, BDMGO L, AND BDMGI L.

BIAKI L (interrupt acknowledge) is received by a bus device on one pin and conditionally transmitted out on a different pin as BIAKO L to the next bus device. (The signal is not transmitted to the next bus device if the receiving bus device has the highest priority interrupt pending.)

Bus wiring connects BIAKO L as output from one device to BIAKI L as input to the next device on

the bus. BDMGI L and BDMGO L form a similar priority daisy chain for Bus Mastership Grant.

Devices connect to all Q22 bus lines through high impedance receivers and gated, high current, open-collector drivers. Receivers and drivers are considered part of the bus.

For more information about extended LSI-11 bus signals and protocols, see the *Microcomputers and Memories* handbook, EB-20912-20.

This chapter presented a brief overview of the Seahorse system components. For more detail about the Seahorse system, see the *Seahorse Field Maintenance Print Set.* The remaining chapters describe the KD32-AA CPU.

Chapter 2
Functional Overview

This chapter is a functional overview of the major CPU components. The macroprogram flow is discussed using several instructions as examples.

Figure 2-1 is a high-level block diagram of the Seahorse CPU.

Data Path

The data path module (M7135) contains the data path, instruction decode, microsequencer and miscellaneous logic needed to implement the MicroVAX instruction set. It is contained on a single quad-height printed circuit board and has connectors to interface to the memory controller and the console terminal.

The major data path components are:

- a 32-bit-wide data path implemented as a custom VLSI chip
- an 8K by 40-bit-wide control store
- a 13-bit-wide microsequencer
- a byte-wide internal data path which provides visibility to various processor states
- an 8K by 8-bit-wide boot PROM
- a console interface

Each of these components is discussed briefly in the following paragraphs.

Data Path Chip

The execution of each microinstruction takes place in the data path chip. This 68-pin custom VLSI chip contains the main 32-bit data path. The chip is controlled by the microprogram. Twenty-one bits of the 40-bit microword control the chip operations. The chip consists of:

- a 21-bit control store register
- a 32-bit bidirectional I/O port
- two 32-bit internal buses
- a 32-bit ALU
- a 64-bit barrel shifter (32-bit output)
- forty-eight 32-bit registers
- thirty-two 32-bit constants
- a 10 ms interval timer
- two register file pointer registers
- hardware to accomplish parallel program counter and register maintenance
- hardware support for multiply

The chip is pipelined; each microinstruction requires 500 ns to execute, but microinstructions are retired every 250 ns (see Figure 1-5).

Control Store

The data path microword is 40 bits wide, implemented in five 8K by 8 PROMs. This provides 8K microwords. The control store is used as follows.

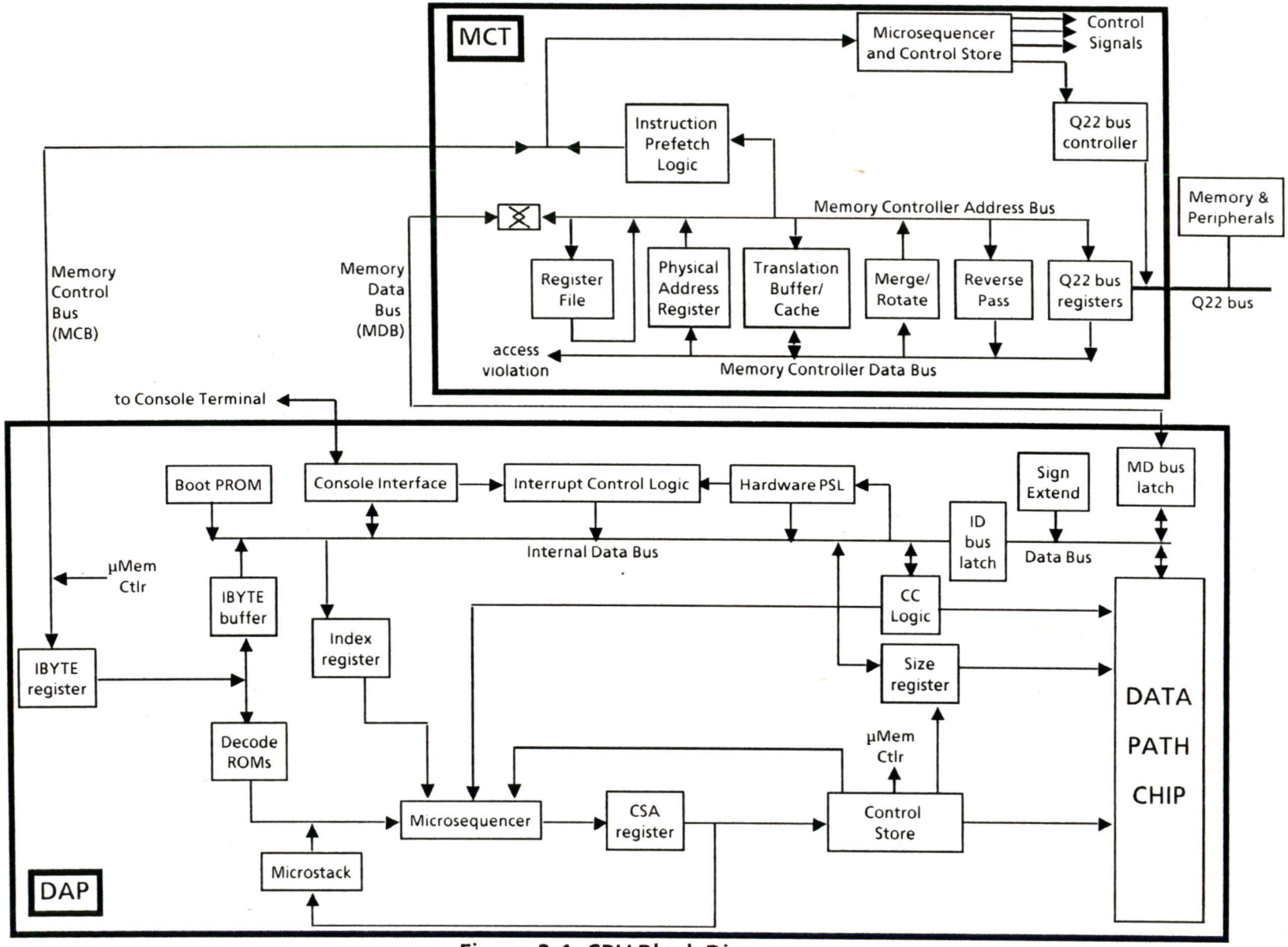


Figure 2-1. CPU Block Diagram

0–4K	MicroVAX base microcode
4–6K	floating point microcode
6–7K	console microcode
7–8K	microverify

Microsequencer

The microsequencer controls the execution flow of the microcode in the CPU. It decodes the microinstructions, performs condition testing and branching, and generates the microaddress of the next microinstruction to be executed. Thus, it generates a 13-bit microaddress every 250 ns. The functions provided by the microsequencer are described further in the next chapter.

Internal Data Bus

The internal data bus is an 8-bit-wide bus completely contained within the data path module. This bus is the interface between the main data path elements in the data path chip, and control and status information which must be available to the remainder of the machine. The internal data bus is also used during instruction decode to pass operand specifier information to the data path chip.

Boot PROM

The boot PROM is 8K by 8-bits-wide, and stores the VAX macrocode necessary to boot the system. The boot PROM is accessible only to the microcode.

Console Interface

The data path module contains the hardware and the microcode to provide the interface to a single

console terminal. The hardware is a standard EIA RS232 line interface. The external connection to this interface is a 10-pin Berg cable mounted on the data path board.

A UART is connected through a buffer to the internal data bus and can be read or written directly by the microcode. The baud rate is selectable from a switch pack and can be set for 300, 1200, 9600, or 19.2K baud. Both transmitter and receiver always operate at the same speed. The microcode reads the switch pack on power up and programs the UART for the selected baud rate.

Memory Controller

The memory controller module (M7136) is the interface between the main data path and micromachine, and the Q22 I/O and memory subsystem. The memory controller is an asynchronous subsystem that provides the following services to the data path micromachine.

- The memory controller disguises the 16-bit Q22 bus data path by implementing commands that allow memory to be accessed as byte, word, or longword without regard to data alignment; I/O devices can also be accessed for byte and word data transfers without regard to data alignment.
- It controls and maintains a translation buffer and a data and instruction cache to reduce the number of memory accesses and increase the effective speed of those accesses.
- It maintains a 16-byte instruction prefetch buffer to allow data path opcode and operand specifier decodes to occur rapidly and at the

same time as memory accesses.

The memory controller is contained on a single quad-height printed circuit board and has connectors to interface to the data path module and the Q22 bus. The major memory controller components are:

- an 8 KB direct-mapped cache
- a 512-entry translation buffer
- a microsequencer and 1K by 64 control store
- Q22 bus interface logic.

Each of these components is discussed briefly in the following paragraphs.

Cache

The data and instruction cache consists of a 2K by 32-bit-wide data store, and a 2K by 16-bit-wide tag store. The cache is the main element of the mechanism that transparently translates 16-bit data from the Q22 bus into 32-bit data that the data path micromachine needs. The cache also provides increased system throughput. The cache is a direct-mapped, write-through cache, and is implemented in the same 4K by 4 RAMs that contain the translation buffer.

Translation Buffer

The translation buffer contains the corresponding physical addresses for recently used virtual addresses. It has a total of 512 entries: 256 entries for mapping system space addresses and 256 entries for mapping process space addresses.

Memory Controller Micromachine

The memory controller micromachine consists of a 1K by 64 control store and a simple microsequencer which, in most instances, generates microaddresses directly from the previous microword. The memory controller microsequencer accepts memory request commands issued by the data path micromachine and sequences the memory controller data path to carry out the command. A wide, parallel microword allows several memory controller functions to take place at the same time.

Q22 Bus Interface Logic

The Q22 bus interface logic allows the Seahorse CPU to communicate with the Q22 bus. Although it is physically located on the memory controller module, it is discussed as a separate controller in the next section.

Q22 Bus Interface

The Q22 bus interface consists of a state sequencer, a write register and a read register. The sequencer handles the bus sequencing and arbitration, freeing the memory controller from this task.

Interrupts from Q22 bus devices are arbitrated according to their interrupt priority levels (IPLs), but reported to the data path module as IPL 17 (hex). Software may subsequently lower the IPL to the level of the interrupting device.

Seahorse allows only byte and aligned-word accesses to Q22 I/O space. All other attempted accesses result in a machine check. Additionally, aligned-longword writes to memory are atomic; that is, no other bus operations are allowed

between the two 16-bit-writes executed on the Q22 bus to accomplish an aligned longword write.

Seahorse does not check parity on the Q22 bus. However, memory parity errors are reported to the CPU via the Q22 bus.

Macroprogram Level Instruction Flow

This section takes three instructions as examples, and describes the data transfers on a macroprogram level. This should illustrate how the major functional components, described above, interact.

No Operation

A no operation (NOP) macroinstruction is one byte long; it consists only of an opcode: 01. The following steps describe the data transfers that take place as this instruction is fetched and executed. Figure 2-2 illustrates the data paths that correspond to the steps.

1. The program counter (PC), located on the data path chip, contains the virtual address of the NOP instruction.
2. The virtual address is transferred to the memory controller along the memory data bus (MDB).
3. The translation buffer on the memory controller translates the virtual address to a physical address.
4. The physical address is sent to the cache. (It is also copied into a Q22 bus register in case the address is not in the cache and Q22 memory must be accessed to obtain the

data.) Assume a cache hit; that is, the cache contains the data for that physical address: the NOP opcode, 01.

5. The cache data (in this case, the instruction byte, 01) is sent through the merge/rotate logic to the prefetch logic (see Figure 2-1), and out onto the memory control bus to the IBYTE register.

6. From the IBYTE register, the instruction byte is sent to the decode logic and the data path microsequencer for decoding. The proper microinstructions are invoked to execute the NOP macroinstruction. The hardware on the data path chip increments the program counter (PC) by one. This cycle ends with the PC containing the virtual address of the next byte in the instruction stream; in this case, the virtual address of the next macroinstruction (because NOP is only one byte long).

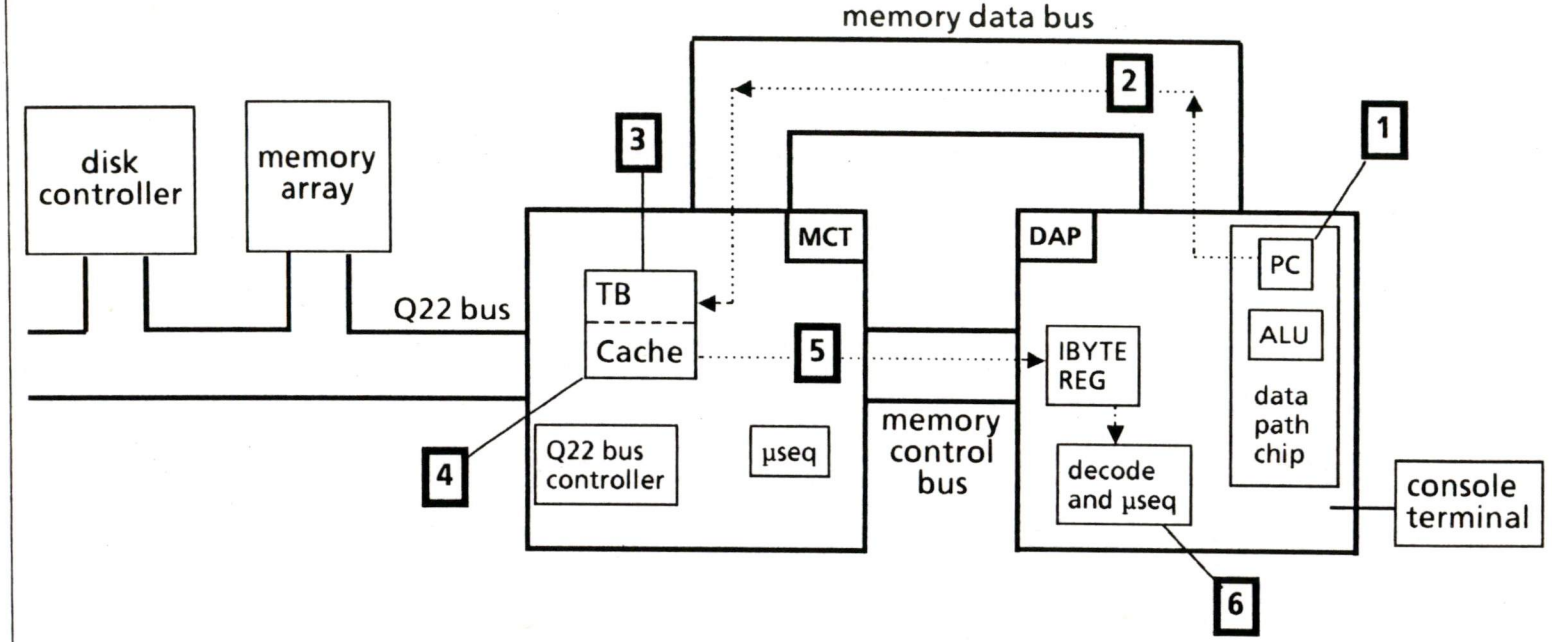


Figure 2-2. NOP Macroinstruction Data Flow

Move Byte

A move byte (MOVB) macroinstruction copies the byte at the address specified by the first operand into the location specified by the second operand. A sample move byte instruction is: MOVB (R0), R1. This instruction means: locate the byte of data at the address contained in R0 (general processor register 0), and move it to R1 (general processor register 1). At an assigned virtual address in memory, say 0200, the instruction looks like this:

51	60	90	:0200

where 90 is the opcode for move byte, 60 is the operand specifier for register deferred mode specifying R0, and 51 is the operand specifier for register mode specifying R1. The following steps describe the data transfers that take place as this instruction is fetched and executed. Figure 2-3 illustrates the data paths that correspond to the steps.

1. The program counter (PC), located on the data path chip, contains the virtual address of the MOVB opcode (0200).
2. The virtual address is transferred to the memory controller along the memory data bus (MDB).
3. The translation buffer on the memory controller translates the virtual address to a physical address.
4. The physical address is sent to the cache. (It is also copied into a Q22 bus register in case the address is not in the cache and Q22 memory must be accessed to obtain the

data.) Assume a cache hit; that is, the cache contains the data for that physical address: the MOVB opcode, 90. The cache actually contains the entire MOVB instruction plus some adjacent bytes of data because each cache entry is 32 bits wide, and the address is longword-aligned.

5. The cache data (in this case, the entire MOVB instruction: 516090) is sent through the merge/rotate logic to the prefetch logic (see Figure 2-1). The prefetch logic actually holds up to eleven bytes at a time to facilitate rapid instruction stream decoding. From the prefetch logic, the first byte of the MOVB instruction (the opcode) is sent out the memory control bus to the IBYTE register. As the opcode is clocked into the IBYTE register, the prefetch logic drives the next instruction byte (60) onto the memory control bus.

6. From the IBYTE register, the opcode (90) is sent to the decode logic and the data path microsequencer for decoding. The proper microinstructions are invoked to execute the MOVB opcode.

7. The hardware on the data path chip increments the program counter (PC) by one. The PC now contains the virtual address of the next byte in the instruction stream; in this case, the virtual address (0201) of the first operand specifier (60).

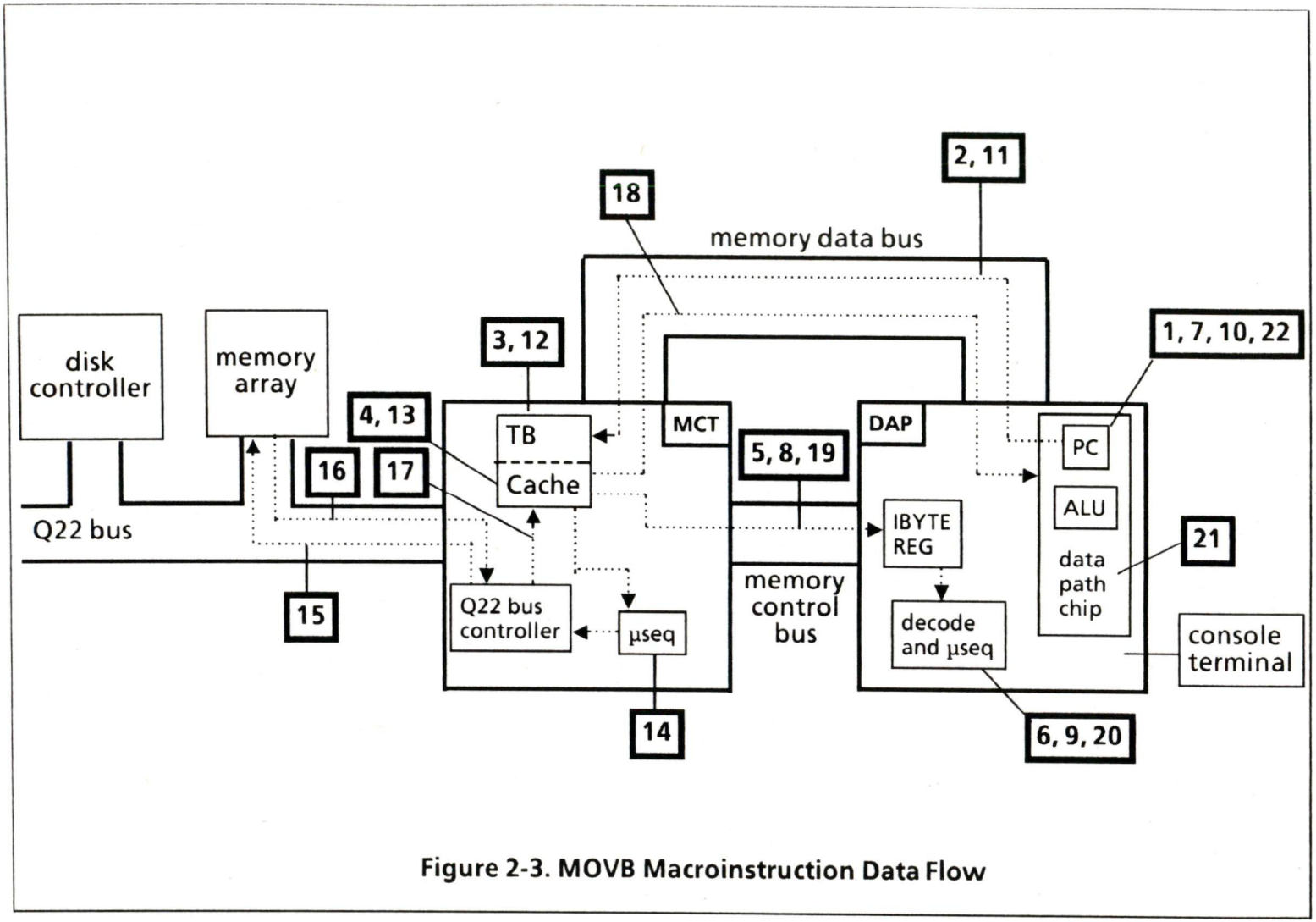


Figure 2-3. MOVB Macroinstruction Data Flow

8. On the next clock edge, the instruction byte sitting on the memory control bus (in this case, 60), is loaded into the IBYTE register and the prefetch logic drives the next instruction byte (51) onto the memory control bus. Since the IBYTE register now contains the next instruction byte (60), there is no need to send its virtual address (0201) from the PC to the memory controller for translation.

9. From the IBYTE register, the operand specifier (60) is sent to the decode logic and the data path microsequencer. The proper microinstructions are invoked to execute it.

10. The hardware on the data path chip increments the PC so that it now contains the virtual address (0202) of the second operand specifier, 51.

11. Decoding and executing the operand specifier 60 causes the contents of R0 to be examined. R0 is located on the data path chip and contains some virtual address, say 0100. The 0100 is sent over the memory data bus to the memory controller for translation.

12. The translation buffer on the memory controller translates the virtual address to a physical address.

13. The physical address is sent to the cache. (It is also copied into a Q22 bus register in case the address is not in the cache and memory must be accessed to obtain the data.) This time, assume a cache miss; that is, the cache does not contain the data at that physical

address.

14. The memory controller microsequencer detects the cache miss condition and informs the Q22 bus controller that a data transfer operation is needed.

15. Since the physical address is already conveniently stored in a Q22 bus register, the Q22 bus controller takes over and initiates two read word data transfers (DATI), sending the physical address over the Q22 bus to the memory array. (Two read word data transfers are necessary to retrieve the 32 bits needed to fill the cache.)

16. Two words of data at the physical address are located in the memory array and sent over the Q22 bus, one word at a time, to the Q22 bus read register on the memory controller module.

17. From the Q22 bus read register, the data are sent to the rotate/merge logic (see Figure 2-1). The Q22 bus read register is 16 bits wide, so as each word is read, it is passed to the rotator, rotated, and held in the merge register (the merge/rotate logic includes a rotator and a 32-bit-wide merge register). The purpose of the rotation is to position the requested byte of data (the first operand) in the low-order byte of the merge register. Once both words are latched in the merge register in the proper order, all 32 bits are written into the cache.

18. In parallel with the cache write operation, the two words are moved from the merge register over the memory data bus to the data path chip on the DAP module. Because

this is a move byte instruction, only the low-order byte on the memory data bus (the first operand) is saved in the data path chip. Steps 11 through 18 have all happened as a result of the microinstructions invoked from decoding and executing the first operand specifier, 60.

19. Now that the byte to be moved into R1 has been obtained from memory and stored in a temporary register on the data path chip, the next instruction byte, 51, which was sitting on the memory control bus, is clocked into the IBYTE register.

20. From the IBYTE register, the second operand specifier (51) is sent to the decode logic and the data path microsequencer. The proper microinstructions are invoked to execute it.

21. Decoding and executing the operand specifier 51 causes the byte of data stored in the temporary register to be moved into R1. The move byte macroinstruction is now complete.

22. The hardware on the data path chip increments the PC to point at the next byte in the instruction stream (in this case, the opcode of the next instruction).

Subtract One and Branch

The subtract one and branch on greater (SOBGTR) macroinstruction maintains a loop count and a branch address, causing the macroprogram to loop on a set of instructions a desired number of times. The loop count is decremented by 1 and a branch taken to the starting address of the loop until the

loop count is less than or equal to 0. As long as the loop count is greater than 0, the sign-extended branch displacement is added to the PC and the PC replaced by the result to cause the branch to the first instruction in the loop.

At an assigned virtual address in memory, say 0203, a SOBGTR instruction might look like this:

E0	52	F5	:0203

where F5 is the opcode for SOBGTR, 52 is the operand specifier for register mode specifying R2, and E0 is the number −32. R2 contains the loop count that is decremented each time the loop is executed. The −32 is sign-extended and added to the PC to cause the branch back to the start of the loop. Assume that the SOBGTR instruction follows right behind the MOVB instruction in the instruction stream, and that the entire SOBGTR instruction is also in the prefetch logic.

The following steps describe the data transfers that now take place as the SOBGTR instruction is fetched and executed. Figure 2-4 illustrates the data paths that correspond to the steps.

1. From the prefetch logic, the first byte of the SOBGTR instruction (the opcode, F5) is sent out the memory control bus to the IBYTE register. As the opcode is clocked into the IBYTE register, the prefetch logic drives the next instruction byte (52) onto the memory control bus.

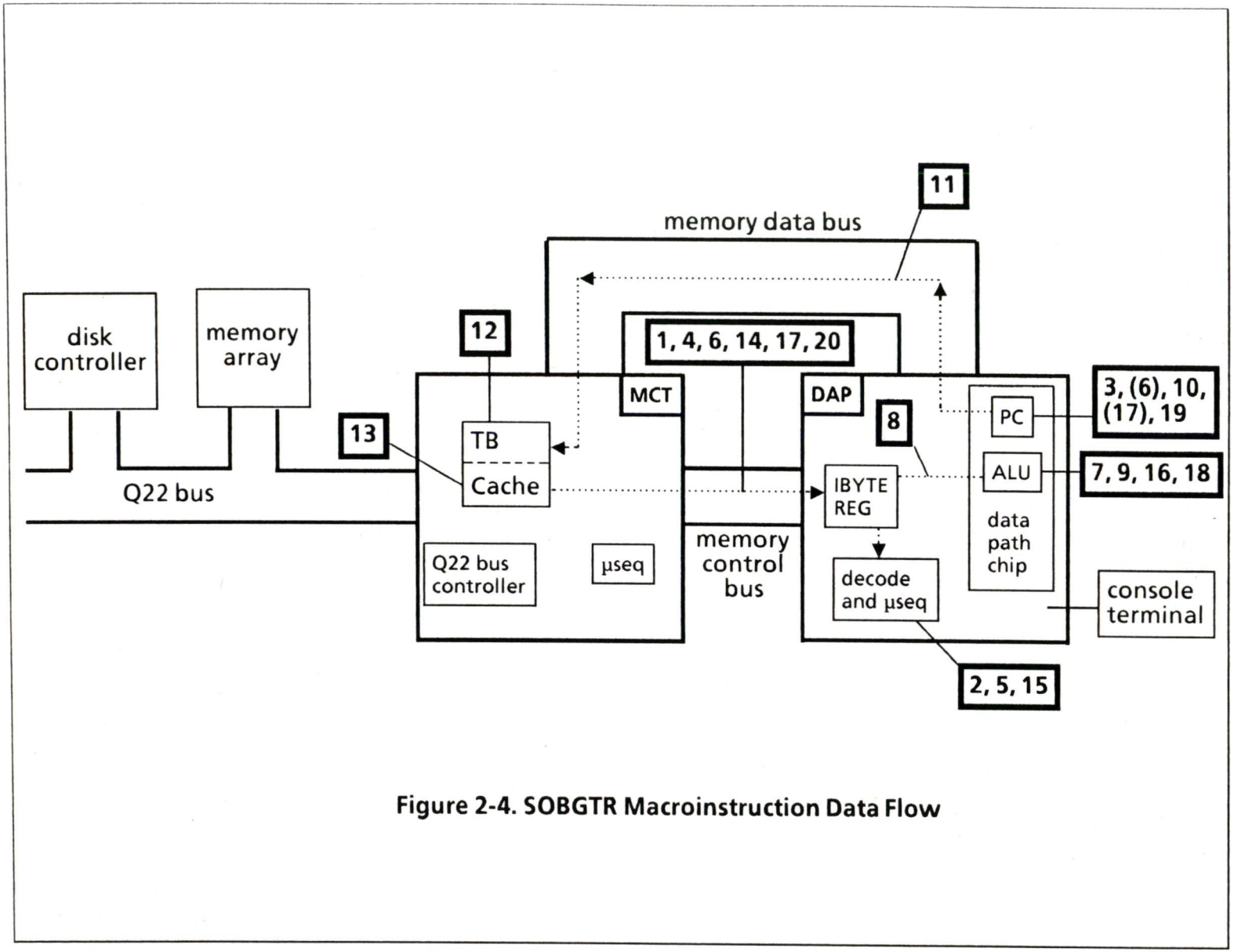


Figure 2-4. SOBGTR Macroinstruction Data Flow

2. From the IBYTE register, the opcode (F5) is sent to the decode logic and the data path microsequencer for decoding. The proper microinstructions are invoked to execute the SOBGTR opcode.

3. The hardware on the data path chip increments the program counter (PC) by one. The PC now contains the virtual address of the next byte in the instruction stream; in this case, the virtual address (0204) of the first operand specifier (52).

4. On the next clock edge, the instruction byte sitting on the memory control bus (in this case, 52), is driven into the IBYTE register and the prefetch logic drives the next instruction byte (E0) onto the memory control bus. Since the IBYTE register now contains the next instruction byte (52), there is no need to send its virtual address (0204) from the PC to the memory controller for translation.

5. From the IBYTE register, the first operand specifier (52) is sent to the decode logic and the data path microsequencer. The proper microinstructions are invoked to execute it.

6. The hardware on the data path chip increments the PC to 0205 (the virtual address of E0, the next instruction byte), and E0 is loaded into the IBYTE register from the memory control bus.

7. Decoding operand specifier 52 and executing its microinstructions causes the contents of R2 to be decremented by 1, and the condition codes to be cleared (set to zeros).

8. From the IBYTE register, E0 is driven over the internal data bus (see Figure 2-1), and sign extended on the data bus. From the data bus, E0 is sent to the data path chip.

9. The microinstructions invoked from the opcode decode (step 5) compute the virtual address for the start of the loop as: PC+1+E0=01E6. This branch destination address is stored in a result register on the data path chip.

10. Next, the condition codes are tested. Since condition codes Z and N are clear, the loop count contained in R2 is still greater than 0. (Z is set when the loop count equals 0; N is set when the loop count is less than 0.) Therefore, the virtual address 01E6 stored in a result register is moved into the PC to cause the program to branch back to the beginning of the loop.

11. The virtual address 01E6 is sent to the memory controller over the memory data bus.

12. The translation buffer on the memory controller translates the virtual adddress to a physical address.

13. The physical address is sent to the cache. (It is also copied into a Q22 bus register in case the address is not in the cache and memory must be accessed to obtain the data.) Assume a cache hit; that is, the cache contains the instruction bytes at the physical address for the start of the loop.

14. The cache data is sent through the merge-rotate logic to the prefetch logic. The first

instruction byte is then sent out onto the memory control bus to the IBYTE register.

15. From the IBYTE register, the instruction byte is sent to the decode logic and the data path microsequencer for decoding. The proper microinstructions are invoked for executing the instruction byte.

16. This flow continues: moving the next byte from the instruction stream into the IBYTE register, decoding and executing it, until the opcode for the SOBGTR instruction, F5, is once again loaded into the IBYTE register. Steps 2 through 15 are repeated until the loop count, when decremented at step 7, is zero. When this occurs, condition code Z is set.

17. The hardware on the data path chip increments the PC to 0205 (the virtual address of E0, the next instruction byte), and E0 is driven off the memory control bus into the IBYTE register.

18. The microinstructions invoked from the most recent decode of operand specifier 52 compute the virtual address for the start of the loop as: PC+1+E0=01E6. This branch destination address is stored in a result register on the data path chip.

19. Next, the condition codes are tested. Since condition code Z is set, the loop count contained in R2 is equal to 0. Therefore, the branch destination address, 01E6, is left in the result register and *not* moved into the PC. Instead, the hardware on the data path chip increments the PC to 0206. This is the virtual address of the next byte in the

instruction stream; in this case, the opcode of the macroinstruction that follows the SOBGTR instruction.

20. The opcode is moved off the memory control bus into the IBYTE register, and the process of decoding and executing continues.

Microcode

The microcode controls all the functions on both modules. Each macroinstruction in the microVAX instruction set is implemented by an associated routine of microinstructions. All of the macroinstruction data transfers described above, for example, happen as a result of their associated microinstructions.

The microinstruction routines are stored in three places: the control store on the DAP module, the control store on the MCT module for the memory controller, and the control store on the MCT module for the Q22 bus controller. The flow from one microinstruction to the next is controlled by three microsequencers, one for each control store: the data path microsequencer, the memory controller microsequencer, and the Q22 bus microsequencer. Understanding these three microsequencers and the microinstructions they execute is the key to understanding the Seahorse CPU.

The data path microsequencer and control store are the master source of control. The microinstructions in the data path control store are invoked to execute the decoded macroinstruction. The data path microsequencer controls the microinstruction flow.

The memory controller microsequencer acts as a slave receiving commands from the data path microsequencer, performing the desired function, and delivering data or status back to the data path micromachine. The control store for the MCT microsequencer contains the microinstructions that enable the MCT microsequencer to perform the desired memory control function.

The Q22 bus microsequencer accepts commands for data from the MCT microsequencer and handles the bus sequencing and arbitration to get the requested data. The Q22 bus microsequencer returns data and status back to the MCT microsequencer. The control store for the Q22 bus microsequencer contains the microinstructions that enable the Q22 bus microsequencer to perform the desired data transfer functions.

The following chapters describe these three micromachines, and the hardware that implements and surrounds them, in detail.

Chapter 3
Data Path Microcode

All of the functions that happen in the KD32-AA CPU happen as the result of microinstructions. This chapter describes the microinstructions that run the data path module.

Microinstruction Format

The data path microinstruction is forty bits wide. The bits are divided into four fields that accomplish different functions. These fields are parity, condition code/size control, data path control, and next address control. Memory controller functions are encoded within the data path control field.

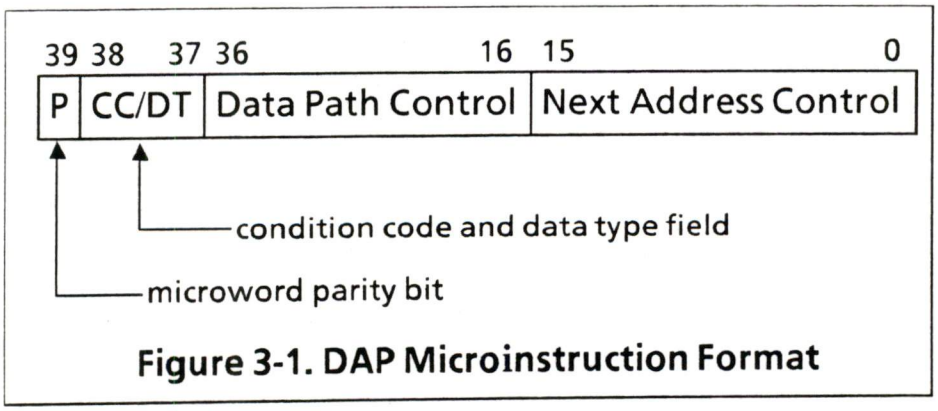


Figure 3-1. DAP Microinstruction Format

The following sections describe each of these fields in more detail.

Parity Field

The highest order bit (bit 39) of the microinstruction contains the parity bit. It is used to detect single bit errors across the entire microinstruction. Odd parity is used; that is, the parity bit is a one when the sum of the one bits in the remainder of the microinstruction is even.

Condition Code/Data Type Field

This field has two functions. It controls the setting of the condition codes, and it determines the data type to be used for the current operation. (Data type is also referred to as size.) Which function this field is used for in any given microinstruction depends on the purpose of the microinstruction. For example, if the microinstruction is a Move or a Moveout, bits <38:37> are interpreted as data type. If the microinstruction is an Add, this field controls the setting of the condition codes. Table 3-1 shows all the microinstruction types and which way the CC/DT field is interpreted for each.

When bits <38:37> are interpreted as the condition code field, the encoding is as follows:

<38:37>	CC FUNCTION
0	condition codes are unaffected; data type is long
1	set ALU condition codes; data type is long
2	set ALU and PSL condition codes; data type is long
3	set ALU and PSL condition codes; data type is size dependent

When bits <38:37> are interpreted as the data

type field, the encoding is as follows:

<38:37>	DATA TYPE
0	byte
1	word
2	use size register
3	longword

Data Path Control Field

The data path control field is the 21 bits that are sent to the data path chip to control its functions. These 21 bits are also referred to as the data path chip microinstruction, or DPC microinstruction. The data path control field for all microinstructions (except Memory Requests and I-stream Requests) is divided into six function fields, as shown in Figure 3-2. (The encoding of the data path control field for Memory Request and I-stream Request microinstructions is shown in Figure 3-4 later in this chapter.)

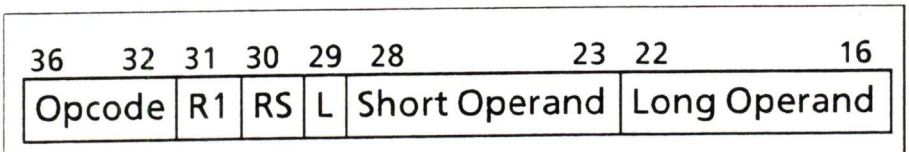


Figure 3-2. Data Path Control Field

The opcode field, bits <36:32>, defines the microinstruction type. Table 3-1 shows the available opcodes and their functions.

The result register bit, bit <31>, selects the destination result register for the current ALU operation. The result of any ALU operation is

stored in one of two result registers on the data path chip: result register 0 or result register 1. If bit <31> is clear, the result of the current ALU operation is stored in result register 0. If bit <31> is set, the result of the current ALU operation is stored in result register 1.

The register save bit, bit <30>, determines whether or not a register save operation occurs. (This is true unless the microinstruction is a NOP, Decode, Restore, Clear Save Stack, Multiply Step, I-stream Request, or Memory Request; bit <30> is ignored in these microinstructions.) The data path chip contains a register save stack, which is a pushdown stack capable of holding seven 36-bit items. When bit <30> is set, 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. When bit <30> is clear, no register save operation occurs.

The literal bit, bit <29>, determines the interpretation of the short operand field. If bit <29> is clear, the short operand field specifies a register. If bit <29> is set, the short operand field is literal data. If the short operand is literal data, the data path chip zero-extends the data to 32 bits for use inside the chip.

The short operand field, bits <28:23>, is the first operand of the DPC microinstruction. The short operand field can specify address locations 0 to 63 and may designate a register directly or indirectly. If the literal bit is set, the short operand field is a 6-bit literal value.

Table 3-1. Opcode Assignments

Opcode	CC/DT	Function	Interpretation
0	CC	NOP	no operation
1	CC	AND	dest ← short operand AND long operand
2	CC	OR	dest ← short operand OR long operand
3	CC	XOR	dest ← short operand XOR long operand
4	CC	Mask	dest ← (NOT short operand) AND long operand
5	CC	Reverse Mask	dest ← short operand AND (NOT long operand)
6	CC	NOT	dest ← NOT short operand
7	CC	Reverse NOT	dest ← NOT long operand
8	CC	Add	dest ← short operand + long operand
9	CC	Add + 1	dest ← short operand + long operand + 1
10	CC	Addwc	dest ← short operand + long operand + carry
11	CC	Sub	dest ← short operand – long operand
12	CC	Sub – 1	dest ← short operand – long operand – 1
13	CC	Reverse Sub	dest ← long operand – short operand
14	CC	Reverse Sub – 1	dest ← short operand – long operand – 1
15	CC	Compare	CCs ← short operand – long operand (The result registers are unaffected.)
16	CC	Shift Left	dest ← long operand shift left logical by shift count register
17	CC	Shift Right	dest ← long operand shift right logical by shift count register
18	CC	Shift Right Arithmetic	dest ← long operand shift right arithmetic by shift count register
19	CC	Double Shift	dest ← 32 bits from 64-bit quantity formed from SOP and LOP, shift right by shift count register
20	CC	Shift Left Literal	dest ← long operand shift left logical by literal
21	CC	Shift Right Literal	dest ← long operand shift right logical by literal
22	CC	Shift Right Arith. Lit.	dest ← long operand shift right arithmetic by literal
23		reserved	undefined
24	DT	Decode	Decode generates a new microaddress for the current macroinstruction opcode or operand specifier.
25	DT	Restore	The top entry in the register save stack is moved to the register whose address is stored in the entry.
26	DT	Clear Save Stack	All entries in the register save stack are marked as being empty.
27	DT	Multiply Step	Multiply Step controls the "shift and add" algorithm for multiplication.
28	DT	I-stream Request	A memory request in which the long operand specifies IB.BYTE, IB.WORD, IB.LONG, or IB.SIZE.
29	DT	Move	Move from long operand to short operand.
30	DT	Memory Request	A memory request in which the long operand is the memory address of the desired data.
31	DT	Move Out	Move from short operand to an external destination specified by the long operand.

The long operand field, bits <22:16>, is the second operand of the DPC microinstruction. It can specify any address location that the short operand can, and in addition, specify addresses 64 to 127. Thus, the long operand can designate any internal or external register, or any constant (the constants are implemented as ROM on the data path chip).

The encodings for long and short operands are described further in the section titled "Operand Field Encoding" in this chapter.

Next Address Control Field

The next address control field determines the next microinstruction address. As each microinstruction is retrieved from control store, the microsequencer decodes this field and generates the next microaddress. The next microaddress is used to access control store to retrieve the next microinstruction.

The control store address space is divided into 32 pages; each page is 256 words. The next address control field of some microinstructions specifies an address within the current page. Other next address control fields specify a full 13-bit address. The next address control field always has one of the nine formats shown in Figure 3-3.

A next address control field must be specified for every microinstruction. In the microcode listing, there are microinstructions with no explicit next address control field given. For these instances, an unconditional jump to the current microaddress plus 1 is supplied by default.

Six of the nine formats shown in Figure 3-3 have

jump control fields, either JC<3:0> or JC<1:0>, corresponding to next address control field bits <11:8> and <9:8>, respectively. The return format has a split jump control field, consisting of JC<2> (bit 12), and JC<1:0> (bits 9:8). The jump control field is used to specify conditions which are being tested by the microcode. If the condition is not met, the next microaddress is the current microaddress plus 1. If the condition is met, the next address is within the current page at the 256 word offset specified by J<7:0>. The jump control field encoding is shown in Table 3-2. A jump control field value of 0 means there are no jump conditions to be tested and the next microaddress is conditioned only by the output of the OR MUX. (Note: A jump control field value of 0 is meaningless when the next address control field format is a branch.)

Five of the nine formats shown in Figure 3-3 use the OR field, either OR<2:0> or OR<1:0>, corresponding to next address control field bits <12:10> and <11:10>, respectively. The OR bits control the OR MUX, one of the hardware components of the data path microsequencer. The OR MUX is discussed in Chapter 4, but some information about it is called for here.

Conceptually, there are eight sets of inputs to the OR MUX, and each set contains four signals which are used to conditionally affect the low-order 4 bits of the microaddress. Table 3-3 shows the eight sets of inputs with four signals in each set. The value of the OR field selects one set of four signals, thereby determining the output of the OR MUX.

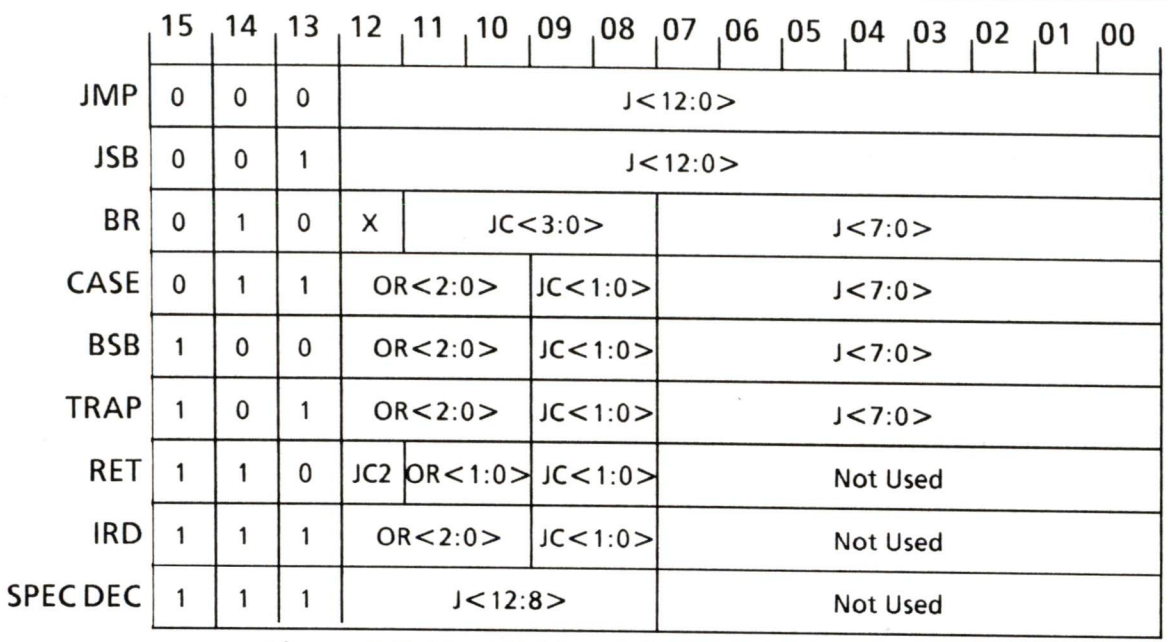


Figure 3-3. Next Address Control Field Formats

Table 3-2. Jump Control Field

JC <3:0>	Condition Tested	JC <3:0>	Condition Tested
0	Use OR MUX	8	Console Halt
1	OR MUX = 0	9	Interrupt
2	OR MUX ≠ 0	A	Stack Register
3	IB OK	B	Register Dest. (not PC)
4	NOT ALU N	C	NOT ALU V
5	NOT ALU Z	D	NOT ALU C
6	ALU N	E	ALU V
7	ALU Z	F	ALU C

Table 3-3. OR<2:0>

2:0	ORMUX3	ORMUX2	ORMUX1	ORMUX0
0	0	0	0	0
1	0	0	0	IB invalid
2	0	0	1	0
3	MEM ERR	Page Crossing	TB Miss	Modify Refuse
4	0	0	BR False	IB invalid
5	Overflow & Chk or Arithmetic trap request	Interrupt Request	T Bit or Console halt	IB invalid
6	INDEX<3>	INDEX<2>	INDEX<1>	INDEX<0>
7	0	0	SIZE<1>	SIZE<0>

For example, one of the inputs to the OR MUX has these four signals on it: MEM ERR, page crossing, TB miss, and modify refuse (see the fourth row in Table 3-3). Suppose, at a given point in time, the MEM ERR signal is set (a one), and the other three are clear (zeros). This input to the OR MUX, then, is 1000 binary. Now, if the value of the OR field in the microinstruction at this same point in time is 3, this input is selected. Therefore, the output from the OR MUX is 1000 binary.

Given this general information about the next address control field, each of the nine formats is described in more detail in the following paragraphs.

Jump and Jump to Subroutine

The jump (JMP) format causes an unconditional jump to any address in control store. The 13-bit control store address is supplied by next address control field bits, labeled J <12:0>.

The jump to subroutine (JSB) format causes an unconditional jump to any address in control store. The 13-bit control store address is supplied by next address control field bits, J <12:0>. A JSB also saves the current microaddress plus 1 on the microstack.

Branch

The branch (BR) format causes a jump to a destination within the current page, if the jump condition specified is met. The jump condition is specified in next address control bits <11:8>. These bits are labeled JC<3:0> in Figure 3-3.

If the jump condition is met, the next microaddress

is constructed from the current page, and next address control bits <7:0>. Bits <12:8> of the address come from the current page, and bits <7:0> specify the address within that page.

If the jump condition is not met, the next microaddress is the current microaddress plus 1.

Case

The case format causes a branch to a destination within the current page if the condition specified in JC <1:0> is met. The value of the field OR<2:0> determines the output of the OR MUX.

If the jump condition is met, the next microaddress is J<7:4> and the logical sum of the OR MUX output and J<3:0>. The current page is specified in bits <12:8>.

If the jump condition is not met, the next microaddress is the current microaddress plus 1.

Branch to Subroutine

The branch to subroutine format (BSB) causes a branch to a microaddress within page zero if the jump condition specified in JC<1:0> is met. The value of OR<2:0> determines the output of the OR MUX. A BSB also saves the current microaddress plus 1 on the microstack.

If the jump condition is met, the next microaddress is <12:8> = 0, J<7:4>, and the logical sum of the OR MUX output and J<3:0>.

If the jump condition is not met, the next microaddress is the current microaddress plus 1.

Trap

The trap format causes a trap to a destination within page zero if the jump condition specified in JC<1:0> is met. The value of OR<2:0> determines the output of the OR MUX. A trap also saves the current microaddress on the microstack.

If the jump condition is met, the next microaddress is <12:8> = 0, J<7:4>, and the logical sum of the OR MUX output and J<3:0>.

If the jump condition is not met, the next microaddress is the current microaddress plus 1.

Return

The return format causes a return to the microaddress at the top of the microstack if the jump condition specified in JC<2:0> is met. The JC field is split for the return format: JC<1:0> correspond to next address control field bits <9:8>, and JC<2> corresponds to bit <12>. The value of OR<1:0> determines the output of the OR MUX (OR<2> is defined as zero for the return format).

If the jump condition is met, the next microaddress is the logical sum of the top entry in the microstack and the OR MUX output.

If the jump condition is not met, the next microaddress is the current microaddress plus 1.

Instruction Register Decode

The Decode microinstruction (opcode 24 in Table 3-1) is used to decode macroinstruction opcodes and macroinstruction operand specifiers. A macroinstruction opcode decode is also called an IRD: an instruction register decode. When the Decode

microinstruction is used for an IRD, its next address control field has the IRD format shown in Figure 3-3. A jump condition is specified in JC<1:0> and the value of OR<2:0> determines the output of the OR MUX.

If the jump condition is met, the next microaddress is <12:4>=0 and the OR MUX output as bits <3:0>. The current microaddress is saved on the microstack.

If the jump condition is not met, the next microaddress is <12>=0 and decode ROM <11:0>.

The decode ROMs (shown in Figure 2-1) are used to select the proper microcode routine to process the current contents of the IBYTE register. If the IBYTE register contains a macroinstruction opcode, the decode ROMs supply twelve bits of microaddress. If the IBYTE register contains a macroinstruction operand specifier, the decode ROMs supply eight bits of microaddress. Chapter 4 contains more information about the decode ROMS.

Operand Specifier Decode

When the Decode microinstruction is used to decode a macroinstruction operand specifier, its next address control field has the SPEC DEC (specifier decode) format shown in Figure 3-3. Operand specifier decode microinstructions have three possible sources for the next microaddress.

If the content of the IBYTE register is valid and the operand is not contained in a general register, the next microaddress is J<12:8> and decode ROM <7:0>. The address of the current microinstruction plus 1 is pushed on the

microstack.

If the content of the IBYTE register is valid and the operand *is* contained in a general register other than the PC, the next microaddress is the address of the current microinstruction plus 1.

If the content of the IBYTE register is not valid, the next microaddress is J<12:4>=0 and OR MUX <3:0>. The OR MUX encoding is defined as 1 for this condition, so OR MUX <3:0> is 0, 0, 0, IB invalid, and IB invalid=1 (see Table 3-3). Thus, a trap to microaddress 00001 occurs. The address of the current microinstruction is saved on the microstack.

Data Path Microinstructions

The microinstructions listed in Table 3-1 are grouped by function and discussed in more detail in the following paragraphs.

ALU Microinstructions

Data path microinstructions involving the ALU are those with opcodes 0 through 15. The ALU is located on the data path chip. The result of an ALU operation is written into one of two registers on the data path chip: RESULT0 or RESULT1. Each destination ("dest") listed in Table 3-1 is one of these two result registers.

When a NOP microinstruction is executed (opcode 0), no operations occur in the data path chip; bits <31:16> of the microinstruction are ignored. See Figures 3-1 and 3-2.)

Shift Microinstructions

Shift microinstructions are those with opcodes 16 through 22. For these microinstructions, the shift count can come from either the shift count register which is located on the data path chip, or from a literal in the short operand field. The range of the shift count must be 0:31. Different opcodes are used to select the type of shift and the source of the shift count. The result of the shift is always placed in the RESULT2 register, also located on the data path chip.

A Double Shift microinstruction (opcode 19) concatenates the short operand and the long operand, and selects 32 bits from this 64-bit quantity. The long operand specifies the lower-order longword. The shift count comes from the shift count register. A rotate operation is obtained by making the long and short operands the same.

Move Microinstructions

The two move microinstructions are Move, opcode 29, and Moveout, opcode 31.

Move transfers the contents of the location specified by the long operand to the location specified by the short operand. The short operand cannot be a literal. The data transfer takes place within the data path chip.

Moveout transfers the contents of the location specified by the short operand to the external data pins of the data path chip. The external destination is specified by the long operand. The range of the destination address must be 96:127.

Other Microinstructions

The following microinstructions don't fit in any of

the categories listed above.

23	Reserved
24	Decode
25	Restore
26	Clear Save Stack
27	Multiply Step
28	I-stream Request
30	Memory Request

Reserved is simply an unassigned opcode. Clear Save Stack causes all of the entries in the register save stack to be marked as empty. The register save bit (bit <14> in the DPC microinstruction) is ignored. The remaining microinstructions are described in the following paragraphs.

Decode

The Decode microinstruction selects the routine of microinstructions to be executed to process the macroinstruction opcode or operand specifier in the IBYTE register. For the Decode microinstruction, the low five bits of the short operand are redefined as shown in Table 3-4.

Table 3-4. Decode Microinstruction Short Operand

Bit	Function	Explanation
27	Enable V bit check	This bit enables checking of the V bit by the OR MUX.
26	Pointer Register	This bit selects which of the two pointer registers is loaded from the data bus.

25	Register Save Stack Initialize	When set, this bit resets the register save stack to empty and pushes the old PC onto the stack.
24	IFUNC 1	This bit is used by the decode ROMs to distinguish an opcode decode from an operand specifier decode.
23	IFUNC 0	This bit is used with bit 24 to define the type of decode selected.

Redefining these bits in this manner enables the following functions to be performed during a Decode:

- Bits <5:0> of the byte in the IBYTE register are extracted from the data bus (see Figure 2-1) and written into one of two 6-bit pointer registers located on the data path chip: pointer 1 or pointer 2. These pointers are used to hold register numbers and literals. Bits <5:0> are written into pointer 1 if bit <26> of the microinstruction is a zero, and into pointer 2 if bit <26> is a one.
- If bit <25> of the Decode microinstruction is a one, the register save stack is cleared, and the unincremented content of the PC is pushed on the register save stack.
- Bits <24:23> of the Decode microinstruction form a two-bit control field which is part of the input to the decode ROMs. (The rest of the input is the macroinstruction byte from the IBYTE register.) Bits <24:23> are encoded

as follows:

24	23	Selected Decode
0	0	operand specifier decode type 1
0	1	operand specifier decode type 2
1	0	IRD for single byte opcodes
1	1	IRD on second byte of two byte opcode

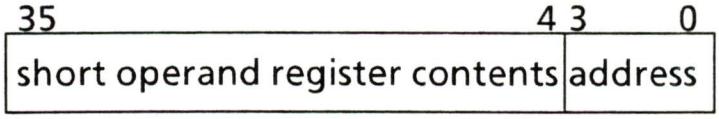
Operand specifier decode type 1 and type 2 refer to different ways the short operands are handled. Basically, type 1 indicates that an integer macroinstruction operand specifier is to be decoded. Type 2 indicates that a floating point macroinstruction operand specifier is to be decoded.

- The macroprogram counter (PC) on the data path chip is incremented by one.
- These microinstruction bits are ignored for IRDs: <31:28> and <22:16>. For operand specifier decodes, bits <22:16> specify IB.BYTE so that the contents of the IBYTE register are driven onto the internal data bus.

The next address control field of the Decode microinstruction then generates the next microaddress, which is the address of the appropriate microinstruction routine for executing the current macroinstruction byte.

Restore

The register save stack, located on the data path chip, is a pushdown stack capable of holding seven 36-bit items. When bit <30> of a microinstruction is set, both the contents of the register specified by the short operand and the low four bits of the register address are pushed on the stack in this format:



The following microinstructions are exceptions to this in that bit <30> (the RS bit) is ignored: NOP, Decode, Restore, Clear Save Stack, Multiply Step, I-stream Request, and Memory Request.

The Restore microinstruction pops the top entry off the register save stack; that is, the top entry in the register save stack is moved to the register whose address is stored in bits <3:0> of the stack.

Multiply Step

The Multiply Step microinstruction performs multiplication using a "shift and add" approach. The two ALU result registers, RESULT0 and RESULT1, are combined to form a 64-bit shift register with RESULT1 the lower order longword. The required setup conditions are:

- The positive multiplier is placed in RESULT1. RESULT0 is cleared.
- The multiplicand is specified by the long operand (range = 0:95).
- The CC/DT field of the microinstruction is longword.

Multiply Step is actually called a total of thirty-two times to complete one multiplication of two longwords. The second and third setup conditions listed above must be in place for all thirty-two executions. The first setup condition only occurs for the first execution. The following functions are performed for each Multiply Step microinstruction.

a. If RESULT1 <0> = 1, add the multiplicand to

RESULT0.

b. Right shift RESULT0 and RESULT1 by one bit, such that RESULT0<0> becomes RESULT1 <31>.

c. If the add in step a) was executed, set RESULT0<31> equal to V XOR N, where V and N are condition code bits from the add executed in step a). If the add was not executed, RESULT0<31> is unchanged.

d. Set the condition code bits according to the result of the addition of the multiplicand and RESULT0 in part a. (If RESULT1<0> is not equal to 1 in part a, the addition does not actually happen and the Multiply Step consists of parts b through d. But as far as the condition code bits are concerned, the addition always happens and the CC bits set accordingly.)

Once these four parts of the Multiply Step microinstruction are executed thirty-two times, the longword multiplication is complete.

Memory Request

The Memory Request microinstruction is an explicit request from the data path to the memory controller to read and write instruction operands and data. The microinstruction supplies a 9-bit function code and 32 bits of data. The 32 bits of data are a virtual address, a physical address, or the actual data to be written.

The 9-bit function code is encoded in bits <31:23> of the Memory Request microinstruction. These nine bits describe the memory function to be performed. The function code is described in more

detail in the section titled "Memory Function Encoding" later in this chapter.

The 32 bits of data are contained in a register on the data path chip. The register address is specified by the long operand of the Memory Request microinstruction. The 32 bits of data are driven from the register onto the data bus, and sent to the memory controller over the memory data bus (see Figure 2-1). The 32 bits of data are also saved in a temporary register on the chip [TMP(0)]. If the 32 bits of data are a physical or virtual address, the data located at this address are the desired instruction operand, or the data to be read or written.

A Memory Request microinstruction is followed by one intervening cycle, and then the proper microinstruction (either a Move or a Moveout) is executed to move the requested data to or from the data path. The long operand of the appropriate move instruction specifies MEMORY.DATA to indicate that the data to be moved are the 32 bits currently on the memory data bus. If the requested data are not available at this time, the microsequencer stalls the execution of the microprogram by continuously repeating the Move or Moveout microinstruction until the requested data are available.

The status of the memory function is available at the same time the requested data are available, and remains valid until the next memory function. The memory function status consists of the four signals on the fourth OR MUX input: MEM ERR, Page Crossing, TB Miss and Modify Refuse (see Table 3-3). If a move in or out of the requested data does not occur in the second cycle after the memory

function, the data path assumes that status is not required and proceeds; the requested function is still carried to completion on the memory controller module.

I-stream Request

The I-stream Request microinstruction acts as a command from the data path to the memory controller to read bytes, sign-extended words, and longwords from the instruction stream. It can be thought of as a special case of the Memory Request microinstruction where the 9-bit function code can only be an instruction stream read (IB.READ) and the 32 bits of data are the unincremented contents of the PC on the data path chip. The data located at this 32-bit address in the instruction stream are the data to be read.

The long operand of the I-stream Request microinstruction specifies IB.BYTE, IB.WORD, IB.LONG, OR IB.SIZE to indicate the amount of data to be read from the instruction stream. The unincremented contents of the PC on the data path chip are driven onto the data bus, and sent to the memory controller over the memory data bus. The PC on the data path chip is then incremented by 1, 2, or 4, depending on the amount of data read from the instruction stream.

The prefetch logic on the memory controller keeps the prefetch buffers filled with instruction stream bytes. An I-stream Request first clears the prefetch buffers, then reads a byte, word, or longword from the cache, sends it to the data path module via the memory data bus, and refills the prefetch buffers beginning with the next byte in the instruction stream following this byte, word, or

longword.

After the I-stream Request microinstruction, one intervening microinstruction is executed. Then a Move microinstruction is executed to return the byte, word, or longword from the cache to the data path over the memory data bus. If a word is read from the cache, it is returned over the memory data bus and sign-extended on the data bus (see Figure 2-1).

Operand Field Encoding

The short and long operands of the microinstructions can specify addresses 0 through 63. The long operand can specify addresses 0 through 127. Figure 3-4 shows the address space organization. Chapter 4 describes the registers in more detail.

Table 3-5. Register Address Organization

ADDR	+0	+1	+2	+3
000	GPR(0)	GPR(1)	GPR(2)	GPR(3)
004	GPR(4)	GPR(5)	GPR(6)	GPR(7)
008	GPR(8)	GPR(9)	GPR(10)	GPR(11)
012	GPR(12)	GPR(13)	GPR(14)	PC
016	TMP(0)	TMP(1)	TMP(2)	TMP(3)
020	TMP(4)	TMP(5)	TMP(6)	TMP(7)
024	TMP(8)	TMP(9)	TMP(10)	TMP(11)
028	TMP(12)	TMP(13)	TMP(14)	TMP(15)
032	TMP(16)	TMP(17)	TMP(18)	TMP(19)
036	TMP(20)	TMP(21)	TMP(22)	TMP(23)
040	TMP(24)	TMP(25)	TMP(26)	TMP(27)
044	TMP(28)	TMP(29)	TMP(30)	TMP(31)
048	RESULT0	RESULT1	RESULT2	Shift Count
052	Pointer 1	Pointer 2	@Pointer 1	@Pointer 2
056	Timer Control/Status (TMRCSR)	Reserved	Reserved	Reserved
060	Reserved	Reserved	Reserved	Reserved
064	Constants ROM	Constants ROM	Constants ROM	Constants ROM
068	Constants ROM	Constants ROM	Constants ROM	Constants ROM
072	Constants ROM	Constants ROM	Constants ROM	Constants ROM
076	Constants ROM	Constants ROM	Constants ROM	Constants ROM
080	Constants ROM	Constants ROM	Constants ROM	Constants ROM
084	Constants ROM	Constants ROM	Constants ROM	Constants ROM
088	Constants ROM	Constants ROM	Constants ROM	Constants ROM
092	Constants ROM	Constants ROM	Constants ROM	Constants ROM
096	CON.DATA (UART data)	CON.STATUS (UART status)	CON.MODE (UART mode)	CON.CMD (UART command)
100	Reserved	Reserved	Reserved	Reserved
104	Size register	Index register	PSL.MODE	Misc. register, bits <3:0>
108	PSL.EN (write), REQ.ST (read)	PSL.IPL (write), INT.SRC (read)	PSL.CC	ALU.CC
112	System ID	Option switches	Misc. register, bits <7:4>	Boot ROM
116	Reserved	Reserved	Reserved	Reserved
120	IB.BYTE	IB.WORD	Reserved	IB.LONG
124	MEMORY.DATA	MEMORY.DATA	MEMORY.DATA	MEMORY.DATA

Memory Controller Interface Microcode

The data path has three ways in which it requests data from the memory controller.

1. The memory controller prefetch logic keeps the prefetch buffers filled with instruction stream bytes so there is a valid byte on the memory control bus as often as possible. The data path implicitly fills the IBYTE register from the memory control bus as a side effect of instruction and operand specifier decodes.
2. The data path executes Memory Request microinstructions to explicitly read and write instruction operands and data.
3. The data path executes I-stream Request microinstructions to explicitly read bytes, sign-extended words, and longwords from the instruction stream.

The first item is actually a function of the data path hardware and is discussed further in Chapter 4. The Memory Request and I-stream Request microinstructions are described earlier in this chapter. The memory functions that can be encoded within these microinstructions are described in more detail in the following paragraphs.

Memory Function Encoding

The Memory Request and I-stream Request microinstructions contain a 9-bit memory function field which is used by the DAP microsequencer to specify a function for the memory controller to perform. The memory function field is bits <31:23> of the microinstruction and specifies the

desired memory function. The nine bits are formatted as follows:

<8> latch function parameters:
When this bit is set, the microsequencer latches the state of the other eight bits of the function code in a register. These eight bits, plus one bit provided by the microsequencer, are referred to as the function parameters.

<7> access mode:
When this bit is set, the access mode is kernel. A zero bit specifies current mode. The access mode is used to check that the protection allows the specified operation to be performed.

<6> modify intent:
When this bit is set, the intended access is write. A zero bit specifies read intent. Modify intent does not signify whether data will be read or written, but rather which access intent is to be checked.

<5> data flow:
When this bit is set, data flows from the data path chip to the memory controller (write). A zero bit specifies that data flows from the memory controller to the data path chip (read).

<4:0> memory function:
This is a 5-bit encoded value that specifies which memory function to execute.

The data size is specified separately in the data type field (bits<38:37>) of the microinstruction.

The format of memory request microinstructions is shown in Figure 3-4.

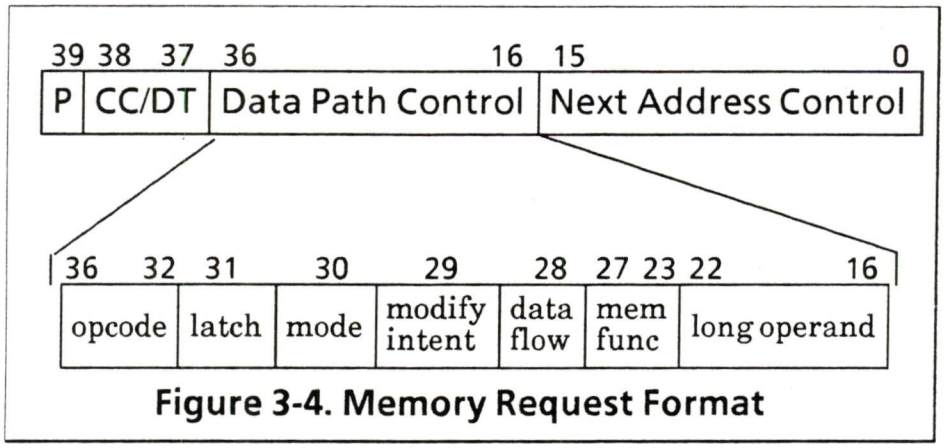


Figure 3-4. Memory Request Format

The data path hardware expands the 9-bit memory function field from the microinstruction into a 10-bit function code. This 10-bit function code, plus two bits specifying the data size, are what is actually delivered to the memory controller. The 10-bits have the format:

<9:8> access mode: the mode for which all accesses are to be checked.

<7> modify intent: the intended access type.

<6> data flow: the direction in which data will flow on the data bus.

<5:1> memory function code: a 5-bit encoded value that specifies the memory function to be performed.

<0> second part: When clear, this bit specifies that the first part of a memory function is to be executed. When set, it specifies that the second

part of a memory function is to be executed. This second part bit is one of the function parameters saved in a register when the latch function parameter bit is set. It is used only during memory management error recovery for unaligned reads and writes across page boundaries.

Memory Functions

The following describes each memory function as specified by microinstruction bits <27:23>. The descriptions give the values for bits <29:23> (see Figure 3-4) as the state of the data flow and modify intent bits is the only difference between some memory functions.

READ.VECTOR

Memory Request microinstruction bits <29:23> have the value 00. This memory function grants bus mastership to the highest level interrupting device and reads its vector from the Q22 bus. The memory controller completes the bus grant cycle and transmits the vector address to the data bus on the data path.

The contents of the register specified by the long operand are ignored as a physical or virtual address is meaningless for this memory function. The microinstruction data type field must specify byte even though a word of data is returned.

VREAD.RCHECK

This is a virtual read with read check Memory Request microinstruction; bits <29:23> have the value 01. This memory function requests that a

virtual read operation, with a check for read access, be performed. The Size register specifies the amount of data to be read. The register specified by the long operand contains a 32-bit virtual address. If mapping is not enabled, then no access check is performed and the virtual address is interpreted as a physical address.

VREAD.WCHECK

This is a virtual read with write check Memory Request microinstruction. Bits <29:23> have the value 41; that is, the 5-bit memory function code is the same as for VREAD.RCHECK (a value of 01) but the modify intent bit is set. This memory function requests that a virtual read operation, with a check for write access, be performed. The Size register specifies the amount of data to be read. The register specified by the long operand contains a 32-bit virtual address. If mapping is not enabled, then no access check is performed and the virtual address is interpreted as a physical address.

If the resultant physical address has bit <29> set, it is a reference to I/O space. Only word and byte references to I/O space are legal, and all word references must be word aligned. If bit <29> is set, the memory controller converts the read to an interlocked read on the Q22 bus (a DATIO—read, modify, write word, or DATIOB—read, modify, write byte).

VWRITE.WCHECK

This is a virtual write with write check Memory Request microinstruction. Bits <29:23> have the value 61; that is, the 5-bit memory function code

has the value 01 but the modify intent bit and the data flow bit are set. This memory function requests that a virtual write operation, with a check for write access, be performed. The Size register specifies the amount of data to be written. The register specified by the long operand contains a 32-bit virtual address. If mapping is not enabled, then no access check is performed and the virtual address is interpreted as a physical address.

If the previous operation was an interlocked read, this function is effectively a write unlock.

VREAD.LOCK

This is a virtual read with write check interlocked Memory Request microinstruction. Bits <29:23> have the value 42; that is, the 5-bit memory function code has the value of 02 but the modify intent bit is set. This memory function requests that a virtual read operation, with a check for read access, be performed. The data type field specifies byte or word. The register specified by the long operand contains a 32-bit virtual address. If mapping is not enabled, then no access check is performed and the virtual address is interpreted as a physical address.

This microinstruction causes a byte or word to be read from memory with a locked Q22 bus cycle; that is, either a DATIOB or a DATIO data transfer takes place. The memory controller is bus master and will not release the bus until a write is performed. The VREAD.LOCK memory function must be followed by a VWRITE.WCHECK function or by a PWRITE function.

IB.REFILL

This is an instruction stream refill Memory Request microinstruction. Bits <29:23> have the value 03. This memory function requests that a new place in the instruction stream be selected. The data type field specifies byte. The register specified by the long operand contains a 32-bit virtual address which is the address of the new place in the instruction stream. No data are delivered on the move in from memory.

If mapping is enabled, a read access check is performed for the specified mode. As subsequent sequential bytes are read from the instruction stream, no access check is necessary until a page boundary crossing. If mapping is not enabled, then no access check is performed.

PREAD

This is a physical read Memory Request microinstruction. Bits <29:23> have the value 04. This memory function requests a physical read operation. The amount of data to be read is specified by the Size register. The register specified by the long operand contains a 32-bit physical address that is the address of the data to be read.

PWRITE

This is a physical write Memory Request microinstruction. Bits <29:23> have the value 64; that is, the 5-bit memory function code has the value 04 but the modify intent bit and the data flow bit are set. This memory function requests a physical write operation. The amount of data to be written is specified by the Size register. The register specified by the long operand contains a

32-bit physical address that is the address of the data to be written.

If the previous function was an interlocked read, this function is effectively a write unlock.

XLATE.RCHECK

This is a translate virtual address with read check Memory Request microinstruction. Bits <29:23> have the value 05. This memory function translates virtual addresses to physical addresses to check if certain operations such as pushing onto the stack can be performed without a fault. This memory function insures that the page is accessible and that the appropriate entry is in the translation buffer. The data type is specified by the Size register. The register specified by the long operand contains the 32-bit virtual address to be translated. The data returned to the data path is the physical address of the first byte corresponding to the translated virtual address. Both the address and the address plus the data size − 1 are checked for read access. If mapping is not enabled, then no access check is performed and the virtual address is treated as a physical address.

XLATE.WCHECK

This is a translate virtual address with write check Memory Request microinstruction. Bits <29:23> have the value 45; that is, the 5-bit memory function code has the value 05, the modify intent bit is set and the data flow bit is clear. (A zero data flow bit specifies that the data flows from the memory controller to the data path chip—a read.) This memory function translates virtual addresses to physical addresses to check if certain operations

such as pushing onto the stack can be performed without a fault. This memory function insures that the page is accessible and that the appropriate entry is in the translation buffer. The data type is specified by the Size register. The register specified by the long operand contains the 32-bit virtual address to be translated. The data returned to the data path is the physical address of the first byte corresponding to the translated virtual address. Both the address and the address plus the data size−1 are checked for write access. If mapping is not enabled, then no access check is performed and the virtual address is treated as a physical address.

IB.READ

This is the memory function for the I-stream Request microinstruction. Bits <29:23> have the value 0D. This memory function reads a byte, sign-extended word, or longword from the instruction stream. The instruction stream PC is implicitly incremented so that the next instruction stream read addresses the correct data.

The amount of data to be read from the instruction stream is specified as IB.BYTE, IB.WORD, IB.SIZE, or IB.LONG in the long operand. The data type field also specifies byte, word, size, or longword and must match the data type specified in the long operand. The data read from the instruction stream is returned to the data path over the memory data bus. (Sign-extended bytes can also be read via the data path by specifying IB.BYTE as the long operand specifier.)

REPEAT.FIRST

This is a Memory Request microinstruction that repeats a previous memory function. REPEAT.FIRST is only meaningful when preceded by an error condition. It is intended for use in memory management error recovery microcode that fills the translation buffer and handles memory modify refuse. Bits <29:23> have the value 06. When bit <31> of a Memory Request microinstruction is set, the data path microsequencer latches the current function parameters in a register. REPEAT.FIRST references this register, enabling the previous Memory Request microinstruction to be repeated. The data type field in the REPEAT.FIRST microinstruction is ignored; the data type field from the previous Memory Request microinstruction is used.

When any Memory Request microinstruction is executed, the 32 bits of data it supplies are saved in the TMP(0) register on the data path chip. The contents of TMP(0) are copied into TMP(15) for a REPEAT.FIRST memory function. The long operand of the REPEAT.FIRST microinstruction specifies the address of TMP(15). Therefore, the 32 bits of data supplied by the REPEAT.FIRST microinstruction are the same 32 bits supplied by the previous Memory Request microinstruction.

If mapping is not enabled, then no access check is performed and the virtual address is treated as a physical address. Note that the REPEAT.FIRST memory function is only meaningful for virtual functions since the failure of a physical function is never retried.

REPEAT.SECOND

This Memory Request microinstruction acts just like REPEAT.FIRST in that it repeats the previous memory function. In addition, it sets the second part flag. REPEAT.SECOND is only meaningful when preceded by an error condition. It is intended for use in memory management error recovery microcode for unaligned reads or writes across page boundaries. Bits <29:23> have the value 07. REPEAT.SECOND references the previous function register, enabling the previous Memory Request microinstruction to be repeated. The data type field in the REPEAT.SECOND microinstruction is ignored. The REPEAT.SECOND memory function is used specifically for memory operations that read across a page boundary.

The virtual address of the first byte in the next page is stored in the RESULT0 register on the data path chip. It is computed by adding 4 to the 32-bit virtual address supplied by the previous Memory Request microinstruction, and clearing the two low order bits. The long operand of the REPEAT.SECOND microinstruction specifies the address of RESULT0. Therefore, the 32-bit address supplied by the REPEAT.SECOND microinstruction is the virtual address of the first byte in the next page.

If mapping is not enabled, then no access check is performed and the virtual address is treated as a physical address. Note that the REPEAT.SECOND memory function is only meaningful for virtual functions since the failure of a physical function is never retried.

READ.CACHE

This is a Memory Request microinstruction that reads a cache block entry. Bits <29:23> have the value 0B. This memory function requests a read operation. The data type field must specify byte, even though a longword of data is returned. The register specified by the long operand contains a 32-bit physical address that is the address of the data to be read from the cache.

WRITE.CACHE

This is a Memory Request microinstruction that writes a cache block entry. Bits <29:23> have the value 2C; that is, the 5-bit memory function code has the value 0C but the data flow bit is set. This memory function requests a write operation. The data type field must specify byte, even though a longword of data is written. The register specified by the long operand contains a 32-bit physical address that is the address of the data to be written to the cache.

READ.MCT

This is a Memory Request microinstruction that reads a memory controller internal register. The internal register number is supplied as part of the function code; bits <29:23> have the values 10 through 17. The data type field must specify byte, even though a longword of data is returned. The contents of the register specified by the long operand are ignored.

WRITE.MCT

This is a Memory Request microinstruction that writes data to a memory controller internal register. The internal register number is supplied

as part of the function code; bits <29:23> have the values 38 through 3F. The data type field must specify byte, even though a longword of data is written. The register specified by the long operand contains the actual data to be written.

READ.TB

This Memory Request microinstruction reads a translation buffer entry. Bits <29:23> have the value 08. The data type field must specify byte, even though a longword of data is returned. The register specified by the long operand contains a 32-bit virtual address. The translation buffer entry specified by the virtual address is read regardless of whether mapping is enabled or not.

WRITE.TB

This Memory Request microinstruction writes a translation buffer entry. Bits <29:23> have the value 29; that is, the 5-bit memory function code has the value 09 but the data flow bit is set. The data type field must specify byte, even though a longword of data is written. The register specified by the long operand contains a 32-bit virtual address. The translation buffer entry specified by the virtual address is written regardless of whether mapping is enabled or not.

INVALID.SINGLE

This is an invalidate single Memory Request microinstruction. Bits <29:23> have the value 0E. This memory function invalidates a single translation buffer entry. The data type field must specify byte.

The register specified by the long operand contains

a 32-bit virtual address. If the specified virtual address is in the translation buffer, then that entry is set invalid. Otherwise, no operation is performed. No useful data are returned by the memory controller on the move in from memory.

INVALID.MULTIPLE

This is an invalidate multiple Memory Request microinstruction. Bits <29:23> have the value 0F. This memory function invalidates multiple translation buffer entries. The data type field must specify byte.

The register specified by the long operand contains a 32-bit virtual address. Translation buffer entries are unconditionally invalidated. Bit <31> of the virtual address selects whether the process or system translation buffer is invalidated. Translation buffer entries are invalidated starting with the specified address and continuing until a page crossing occurs. No useful data are returned by the memory controller on the move in from memory.

RCHECK

This is a read check Memory Request micro-instruction. Bits <29:23> have the value 0A. This memory function performs a read check to determine the accessibility of the first byte of a virtual address. The data type field specifies byte.

The register specified by the long operand contains the 32-bit virtual address of the byte to be checked. If the byte at this virtual address is not accessible, a translation buffer miss is reported in the error summary register as the TB-Check code. If mapping is not enabled, then no access check is

performed. No useful data are returned by the memory controller on the move in from memory.

WCHECK

This is a write check Memory Request microinstruction. Bits <29:23> have the value 4A; that is, the 5-bit memory function code has the value 0A but the modify intent bit is set. This memory function performs a write check to determine the accessibility of the first byte of a virtual address. The data type field specifies byte.

The register specified by the long operand contains the 32-bit virtual address of the byte to be checked. If the byte at this virtual address is not accessible, a translation buffer miss is reported in the error summary register as the TB-Check code. If mapping is not enabled, then no access check is performed. No useful data are returned by the memory controller on the move in from memory.

Memory Controller Status

After a Memory Request microinstruction and an intervening microinstruction have been executed, the next microinstruction executed can test the results of a memory function. The status of a memory function is available at the same time that the data requested by the memory function are available on a read. This status remains available until another memory function is executed.

Memory controller status is returned to the data path via four bits of status, available as microsequencer OR MUX inputs:

- TB Miss. The memory controller cannot complete the current virtual function because the appropriate page table entry is not in the

translation buffer.

- Memory Modify Refuse. The memory controller cannot complete the current virtual write function because the modify bit is not set in the translation buffer copy of the page table entry.
- Page Crossing. The memory controller cannot complete the current virtual read/write function because a page crossing is necessary.
- Error Summary. A bit in the memory controller's error code register has been set, indicating one of the following errors:
  - Access Violation. The memory controller cannot complete the current virtual function because the desired access is not allowed.
  - Parity Error. A memory read error that is not correctable has been detected.
  - Nonexistent Memory. An attempt has been made to access a nonexistent memory location.
  - Illegal Operation. An attempt has been made to access I/O space as a longword or as an unaligned word, or an attempt has been made to execute an interlocked read/write to a longword or an unaligned word.
  - Translation Buffer Check. A read or write check function encountered a translation buffer miss.

The DAP microcode determines which of these errors occurred by reading the error code register. This is accomplished by issuing a READ.MCT Memory Request microinstruction with the number of the error code register specified.

Microverify

Console Microcode

Chapter 4
Data Path Module

This chapter is a detailed description of the components on the data path 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 DAP Functions

The data path module contains hardware to perform the following eight functions:

- control microinstruction flow
- decode macroinstructions
- execute microinstructions
- transfer data within the data path module
- process interrupts
- communicate with the console terminal
- power on
- communicate with the memory controller

The next eight sections describe these functions, and the hardware components that implement them, in detail. The hardware components are illustrated in the DAP block diagram, Figure 4-1.

Controlling the Microinstruction Flow

Controlling the microinstruction flow is the main

function of the data path module, and much of the hardware is dedicated to it. The hardware components are the CPU clocks, the control store, the control store address register, the parity checker, the index register, the microsequencer, and the microstack and microstack pointer. These components, plus some control signals, determine which microinstruction is executed next. The following paragraphs describe each of these components in turn.

Clock Signals

The clocks for the system are generated on the MCT module. A basic clock with a 64 Mhz frequency is generated by a crystal oscillator and is used to derive all the other clocks in the system.

The CPU clock (DAPL CPU CLOCK H) consists of a symmetrical 250 ns period clock. The start of a microcycle is defined as occurring on the leading edge of this clock and is referred to as T0. All the internal data bus registers are written on this edge. The trailing edge of the clock occurs 125 ns later.

The signal DAPL CPU PHASE is a clock with the same timing as CPU CLOCK, but it is not affected by stall conditions.

The delayed CPU clock (DAPL DLYD CPU CLK H) is asserted from T62.5 to T187.5. This clock is used to clock PALs which first decode the microinstruction and generate discrete control signals.

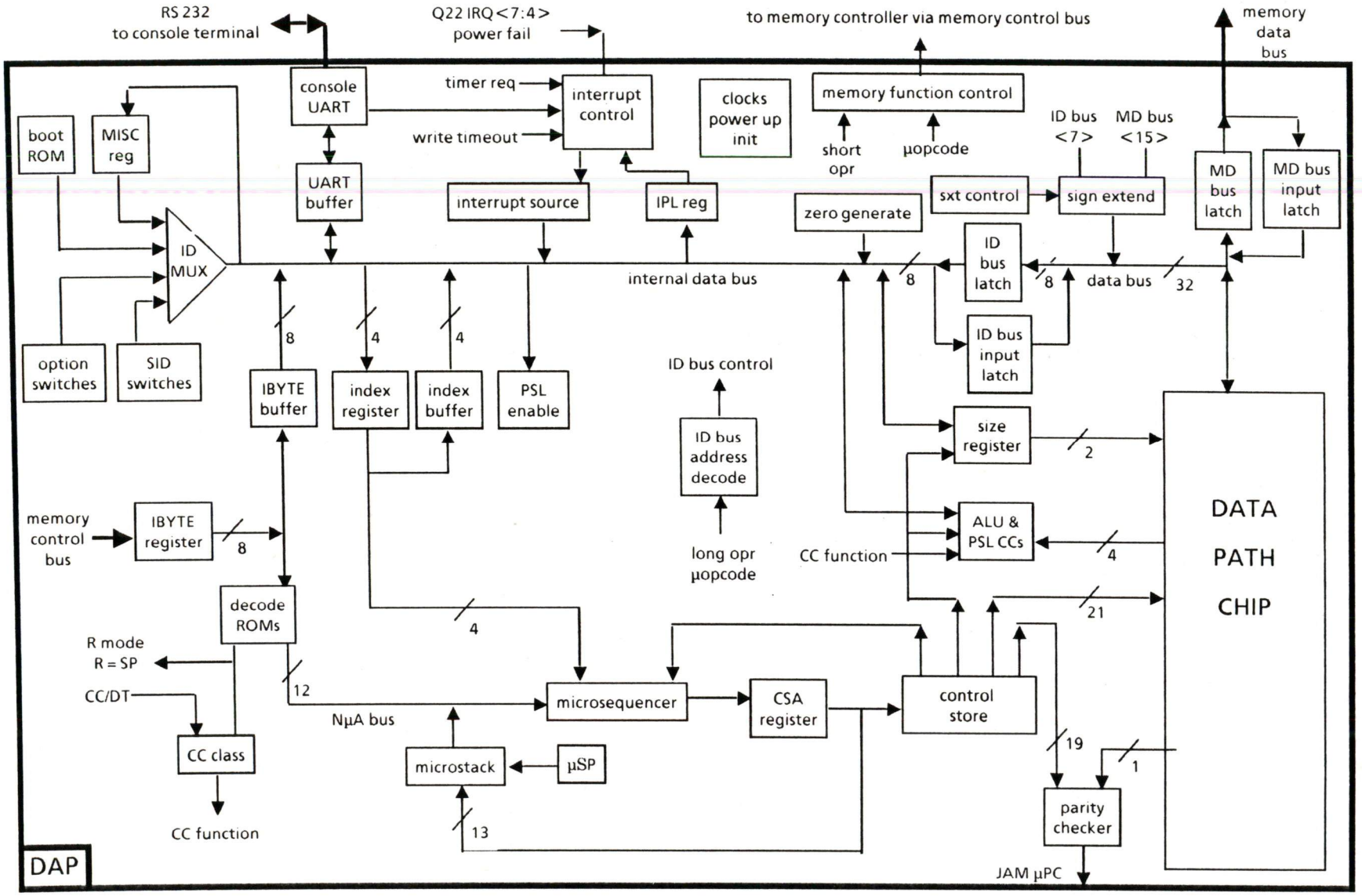


Figure 4-1. Data Path Block Diagram

The complement of the delayed CPU clock is the load control store signal (LD CSR) required by the data path chip. When LD CSR is asserted, the 21-bit DPC microinstruction is stored in the control store register (CSR) located on the data path chip.

The control store address clock (DAPL CSA CLOCK H) is asserted from T125 to T250. This signal clocks the control store address register.

Control Store

The control store consists of five 8K by 8-bit ROMs. The address space within the control store is organized as 32 pages, each page containing 256 words. Each word is one data path microinstruction and is 40 bits wide. All the data path microinstructions are stored in this control store.

The input to the control store is a 13-bit microaddress supplied by the control store address register (DAPB CSA <12:00> H). The high-order five bits specify the page, and the low-order eight bits select the word within the page.

The control store output is a 40-bit microinstruction (DAPA CS <39:00> H). The various microinstruction bits are sent different places.

- Nineteen bits are sent from the control store to the parity checker: CS <39>, CS <38:37>, and CS <15:00>.
- CS <36:16> are sent to the data path chip as the DPC microinstruction.
- The thirteen low-order bits of the next address control field, CS <12:00>, are sent to the jump register in the microsequencer.
- The eight high-order bits of the next address

control field, CS <15:08>, and CS <24>, are sent to the jump MUX control logic and the OR MUX control logic in the microsequencer. The logic decodes CS <15:08> and generates various control signals and selects to govern the microsequencer elements.

- Microinstruction bits CS <38:37>, the CC/DT field, are sent to a flip-flop and then to the PAL that contains the size register; they are also sent to the condition code control logic.
- Microinstruction bits CS <24:23> are sent to the decode ROMs to indicate the type of opcode or operand specifier decode.
- The microinstruction opcode CS <36:32>, CS <24>, and the long operand CS <22:16>, are sent to the block of logic labeled ID bus address decode. This block of logic controls the driving of the appropriate data on the internal data bus when a Move or Moveout microinstruction is executed, and the long operand specifies an address external to the data path chip.
- The microinstruction memory function bits CS <31:23> are sent to the block of logic labeled memory function control in case the microinstruction is a memory request. Information about the microinstruction opcode is sent to the memory function control logic from the ID bus address decode logic.
- Microinstruction bits CS <28:24> are sent to the IBYTE control logic. Information about the microinstruction opcode is sent to the IBYTE control logic from the ID bus address decode logic. Information about the long

operand is sent to the IBYTE control logic from the memory function control logic.

Control Store Address Register

The control store address (CSA) register holds the microaddress used to access the control store. While the inputs to the control store are held stable in this register, the outputs can be used to control the operation of the data path.

The input to the CSA register is the thirteen microaddress bits from the next microaddress MUX. The output from the CSA register is the input to the control store: CSA <12:00>. The control store address register is clocked at T2.

Parity Checker

The parity checker consists of three chips that check the parity of the 40-bit microinstruction. If a parity error is found, the next microaddress is forced to zero, and a flag is set. The microinstruction causing the parity error is executed but produces undefined results. The microinstruction executed at location zero reads the flag by reading bit 5 at the same address as the index register. (The index register itself is four bits wide.)

The input to the parity checker is bits <39:37> and <15:0> from the control store, and one parity bit from the data path chip. When the data path chip receives control store bits <36:16> (the DPC microinstruction), it generates a parity bit for these bits; this DPC parity bit is sent to the parity checker.

If no parity error is found, there is no output from the parity checker. If a parity error is detected, the

output is the signal JAM μPC, which forces the next microaddress to zero.

Index Register

The index register is a four bit register used to store some of the microcode state. The input to the index register is the low four bits from the internal data bus (BUS ID <03:00>). The four bits in the index register are sent as one of the four-signal inputs to the OR MUX (INDEX <3:0>, see Table 3-3), or they can be driven back onto the ID bus through the index buffer (BUS ID <03:00>).

Microsequencer

The microsequencer generates a 13-bit microaddress every 250 ns. It accomplishes this by decoding certain bits in the previous micro-instruction while monitoring certain control and status lines.

The microsequencer consists of these components: the page register, the microprogram counter (μPC), the conditional decrementer, the jump register, the OR MUX, the jump MUX, and the next microaddress MUX (NμA MUX). These components are described briefly in the following paragraphs. Figure 4-2 is a block diagram of the data path microsequencer.

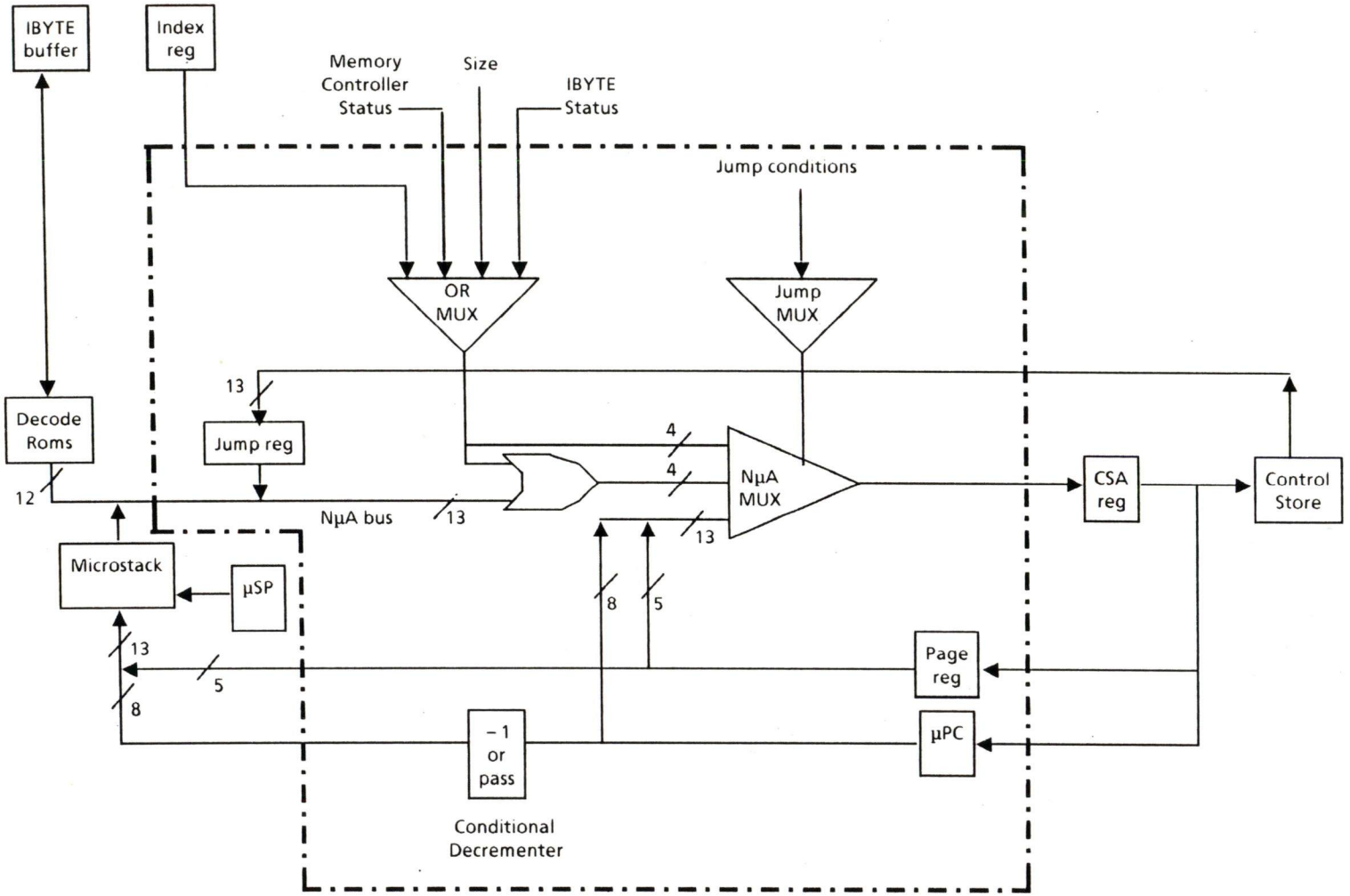


Figure 4-2. Microsequencer Block Diagram

Page Register and Microprogram Counter

These two components together hold the next 13-bit microaddress. The page register contains bits <12:8>; these form the page address. The μPC contains bits <7:0>; these form the address of the word within the page. The μPC is loaded with the address of the current microinstruction plus one, and cannot count beyond the end of the current page. The μPC is clocked at T0 of each microcycle.

Conditional Decrementer

The conditional decrementer is an adder located in the microsequencer logic between the μPC and the microstack. All eight bits from the μPC run through the decrementer. When a microtrap is taken, the decrementer adds a negative one to the μPC bits. Otherwise, the μPC bits pass through unaffected.

Jump Register

This register is used to allow the outputs from control store to be driven onto the next microaddress bus (NμA bus). The jump register is open from T0 to T125.

The input to the jump register is the micro-instruction next address control field, CS <12:0>, from the control store. When enabled, the jump register drives these same thirteen bits onto the next microaddress bus (NμA bus).

OR MUX

The OR multiplexer allows the microcode to "case" on certain signals in the data path, based on the value of the OR field in the current

microinstruction. Conceptually, there are eight inputs to the OR MUX, each with four signals. Some signal values are fixed, others reflect the microcode state. (See Table 3-3).

The OR MUX is enabled when the microinstruction next address control field format is CASE, BSB, TRAP, RET, IRD, or SPEC DEC. (Although there is no OR field in the SPEC DEC next address control field format, the OR MUX is enabled to test for IB invalid.) The OR MUX logic decodes the format and OR fields, and enables the appropriate input. The value of the four signals on that input then becomes the value of the OR MUX output. (For an example, see the section titled "Next Address Control Field" in Chapter 3.) The OR MUX output is then logically ORed with the low four bits of the microaddress on the next microaddress bus (NμA bus). The result is the low four bits of the address of the next microinstruction.

The input to the OR MUX logic is bits <24> and <15:8> of the microinstruction; the input to the OR MUX itself is the various microcode state signals listed in Table 3-3. The OR MUX output is the value of the signals on the selected input line. This value is sent to the NμA bus.

Jump MUX

The jump MUX is part of the jump control logic that controls the next microaddress multiplexer (NμA MUX). The jump MUX selects input signals according to the JC field of the current microinstruction (see Figure 3-3). The JC field specifies conditions to be tested (see Table 3-2).

If the specified condition is met, the jump control

logic enables the NμA MUX to select the address specified in J<7:0> of the current microinstruction as the next microaddress. If the specified condition is not met, the jump control logic enables the NμA MUX to select the current microaddress plus one for the next microaddress.

The input to the jump control logic is next address control field bits <24> and <15:8> of the current microinstruction, and the current values for the conditions that could be tested. The output of the jump control logic is the select lines to the next microaddress MUX.

Next Microaddress MUX

This multiplexer provides the inputs to the CSA register. It is used to select either the contents of the microprogram counter (μPC), or the contents of the next microaddress bus (NμA bus).

When performing conditional jumps, the desired jump-to address is driven onto the NμA bus early in the microcycle. Later in the cycle, the NμA MUX select lines are changed by the jump control logic depending on whether the jump is to be taken.

The NμA MUX actually consists of two 2-to-1 MUXs and three 4-to-1 MUXs. Two of the 4-to-1 MUXs make up the low 4-bit slice of the NμA MUX. One of following three inputs to the NμA MUX is selected by the jump control logic as the NμA MUX output:

- the current microaddress contained in the page register and the microprogram counter, which is the current microaddress plus one, or
- the microaddress currently on the NμA bus,

but with the value of the low four bits determined by the OR MUX output ORed with NμA bus <3:0>, or

- microaddress bits <12:4> forced to zero and the value of the low four bits determined directly by the output of the OR MUX.

These inputs to the NμA MUX are stable at T0+112. The third case described above allows traps which may be taken during decode instructions to use the output of the OR MUX directly. For all other instances, the low four bits of the NμA MUX input are determined by the OR MUX output ORed with the address supplied from the NμA bus (the second case described above), or by μPC+1 (the first case described above).

In short, the inputs to the NμA MUX are: NμA bus <12:0>, OR MUX <3:0>, and μPC <12:0>. The output of the NμA MUX is referred to as NμA <12:0>; these bits have the same value as the bits of the selected input.

When a microinstruction has a BR or CASE next address control field format (see Figure 3-3), the destination microaddress must be within the current page. The NμA MUX has separate selects for bits <12:08> and bits <07:00> so that for BR and CASE, the select for bits <12:08> is not changed even if the branch is taken.

During certain microinstructions, it is necessary to force zeros to be output from the NμA MUX. The following table lists these conditions.

Table 4-1. Forced Zeros on NμA MUX Output

Bits	Conditions
12	IRD

	BSB trap control store parity error power up
11:08	BSB trap decode and trap control store parity error power up
07:04	Decode microinstruction when a trap is being taken control store parity error power up
03:00	control store parity error power up

Microstack

The microstack is a 16 deep, LIFO (last-in-first-out) stack used to save return microaddresses when subroutine calls or microtraps are executed. The address of the current microinstruction plus 1 is also saved on the microstack when a valid operand specifier decode is executed and the operand is not contained in a general register.

If the current microinstruction is a subroutine call or an operand specifier decode (not register mode), the conditional decrementer adds zeros to the address in the microprogram counter, causing the microaddress of the current instruction plus 1to be saved in the stack. If the current microinstruction is a trap, or a Decode and the OR MUX is not equal to zero, the conditional decrementer subtracts one from the address in the microprogram counter, causing the microaddress of the current

instruction to be saved in the stack.

The input to the microstack is supplied by the conditional decrementer. The decrementer always supplies a microaddress to the microstack, but the microaddress does not get written into the microstack unless the current operation is a subroutine call, a trap, an operand specifier decode (not register mode), or an IRD and the OR MUX is not equal to zero. The microstack is written at T250 of the microcycle (T0 of the next cycle).

The signal DAPC STACK PUSH L is asserted by the microsequencer control logic when it decodes microinstruction bits CS<15:08> and <24> and determines that the next address control field format is a subroutine call, a trap, or an operand specifier decode. STACK PUSH L enables writes to the microstack.

The output from the microstack is the top entry in the stack which is driven onto the next microaddress bus when the operation is a return.

Microstack Pointer

The microstack pointer (μSP) always points to the top entry in the microstack; that is, the microstack pointer contains the address of the microstack location that contains the most recently stored microaddress.

When the operation is a "push" (a subroutine call or a microtrap), the address of the next location in the microstack is calculated and used to address the microstack so that the microaddress from the conditional decrementer is written into the microstack at that location. If the branch is taken, the calculated microstack address is stored in the microstack pointer. When the operation is a "pop"

(a return), the current microstack location address in the microstack pointer is used to address the microstack so that the microaddress at that location is written onto the next microaddress bus. Then the microstack pointer is updated at the next T0 clock edge to contain the previous microstack location address.

The inputs to the microstack pointer are signals to indicate when the current operation is a branch, a push, or a return. The outputs from the microstack pointer are four microstack address lines.

Decoding Macroinstructions

Decoding macroinstructions is the second of the eight functions that the data path module performs. The hardware components are the IBYTE register, IBYTE control, the decode ROMS, condition code control, condition code class register, condition code PALs, macrolevel branch control, PSL enable, and the size register. These components decode macroinstruction opcodes and operand specifiers. The following paragraphs describe each of these components in turn, and the ALU and PSL condition codes.

IBYTE Register

The instruction byte register is an eight-bit register that holds the next byte of instruction stream data to be evaluated at the inputs to the decode ROMs; that is, it contains the macrolevel instruction byte currently being processed.

The IBYTE register is read on the internal data (ID) bus when the long operand of the current microinstruction specifies the IBYTE register's

unique address. The contents of the IBYTE register are also driven on the ID bus during operand specifier decodes and stored in one of the two pointer registers in the data path chip. If the operand specifier mode is not short literal, bits <5:4> of the IBYTE register are forced to zero to extract the register number. The high two bits need not be set to zero because the pointer registers are only six bits wide.

The IBYTE register is loaded at T0 from the memory control bus whenever the signal LOAD I BYTE H is asserted. LOAD I BYTE H asserted means that the next byte from the instruction stream is needed at the end of the current microcycle. There are two reasons why the next I-stream byte is needed. The first reason is that the byte currently in the IBYTE register is valid, but the current microinstruction uses that byte, so at the end of this microcycle, the byte in the IBYTE register will no longer be needed. The second reason is that the byte currently in the IBYTE register is not valid. The signal DAPR IB INVALID H is asserted to indicate when this is the case.

The input to the IBYTE register is the byte from the memory control bus, BUS MEM CTL <7:0>. The output from the IBYTE register is eight bits labeled DAPF I BYTE <7:0>. These bits go two places; they are the input to the decode ROMs, and they are latched in the IBYTE buffer (see Figure 4-1).

IBYTE Control

The IBYTE register is controlled by the IBYTE control PAL. The IBYTE control logic informs the

memory controller when the next instruction stream byte is needed by asserting DAPR LOAD I BYTE H. The next instruction stream byte is needed either because the byte currently in the IBYTE register is valid and is used by the current microinstruction, or because the byte in the IBYTE register is not valid.

When DAPR LOAD I BYTE H is asserted, and DAPL CPU CLOCK H is asserted, the signal DAPR CLOCK I BYTE H is generated. DAPR CLOCK I BYTE H clocks the bits BUS MEM CTL <7:0> off the memory control bus into the IBYTE register at T0.

When the IBYTE control logic asserts DAPR LOAD I BYTE H and DAPL DLYD CPU CLK H is asserted, the signal DAPR IB TAKEN L is generated. DAPR IB TAKEN L informs the memory controller that the instruction stream byte that was on the memory control bus has been loaded into the IBYTE register, and another instruction stream byte needs to be sent from the prefetch logic to the memory control bus. Thus, DAPR IB TAKEN causes the memory controller prefetch logic to drive an instruction stream byte onto the memory control bus. DAPR LOAD I BYTE H, DAPR CLOCK I BYTE H, AND IB TAKEN L are the signals asserted when the next instruction stream byte is needed because the byte currently in the IBYTE register is valid but is no longer needed because it was just used by the current microinstruction.

If the byte in the IBYTE register is not valid, the IBYTE control logic assserts the signal DAPR IB INVALID H. The memory controller continues to send instruction stream bytes to the memory

control bus as long as IB INVALID H is asserted; that is, LOAD I BYTE H is always true when DAPR IB INVALID H is asserted. When IB INVALID H is deasserted, this means the byte in the IBYTE register is valid, and the memory controller stops sending instruction stream bytes from the prefetch logic.

The memory controller, meanwhile, generates the signal MCTP NXT IB VALID H, which when asserted means that the byte on the memory control bus is valid data. The memory controller deasserts MCTP NXT IB VALID H when the byte on the memory control bus becomes invalid for any reason; for example, the prefetch buffers become empty, an I-stream Request microinstruction (which flushes the prefetch buffers) is executed, a Memory Request microinstruction with the IB.REFILL function is executed, or the microinstruction long operand specifies IB.BYTE.

IB INVALID H is deasserted by the signal MCTP NXT IB VALID H from the memory controller. As long as the memory controller can supply valid instruction stream bytes from the prefetch logic to the memory control bus, MCTP NXT IB VALID H remains asserted. IB INVALID H is asserted by any of the following microinstructions if the MCTP NXT IB VALID H signal from the memory controller is deasserted: Decode, I-stream Request, Memory Request specifying IB.REFILL, a microinstruction in which the long operand specifies IB.BYTE.

Figure 4-3 illustrates the timing relationship between these signals for both cases:

Case 1: The IBYTE register needs to be refilled because the current byte is valid but a

Decode microinstruction was just executed.

Case 2: The IBYTE register needs to be refilled because the current byte is not valid.

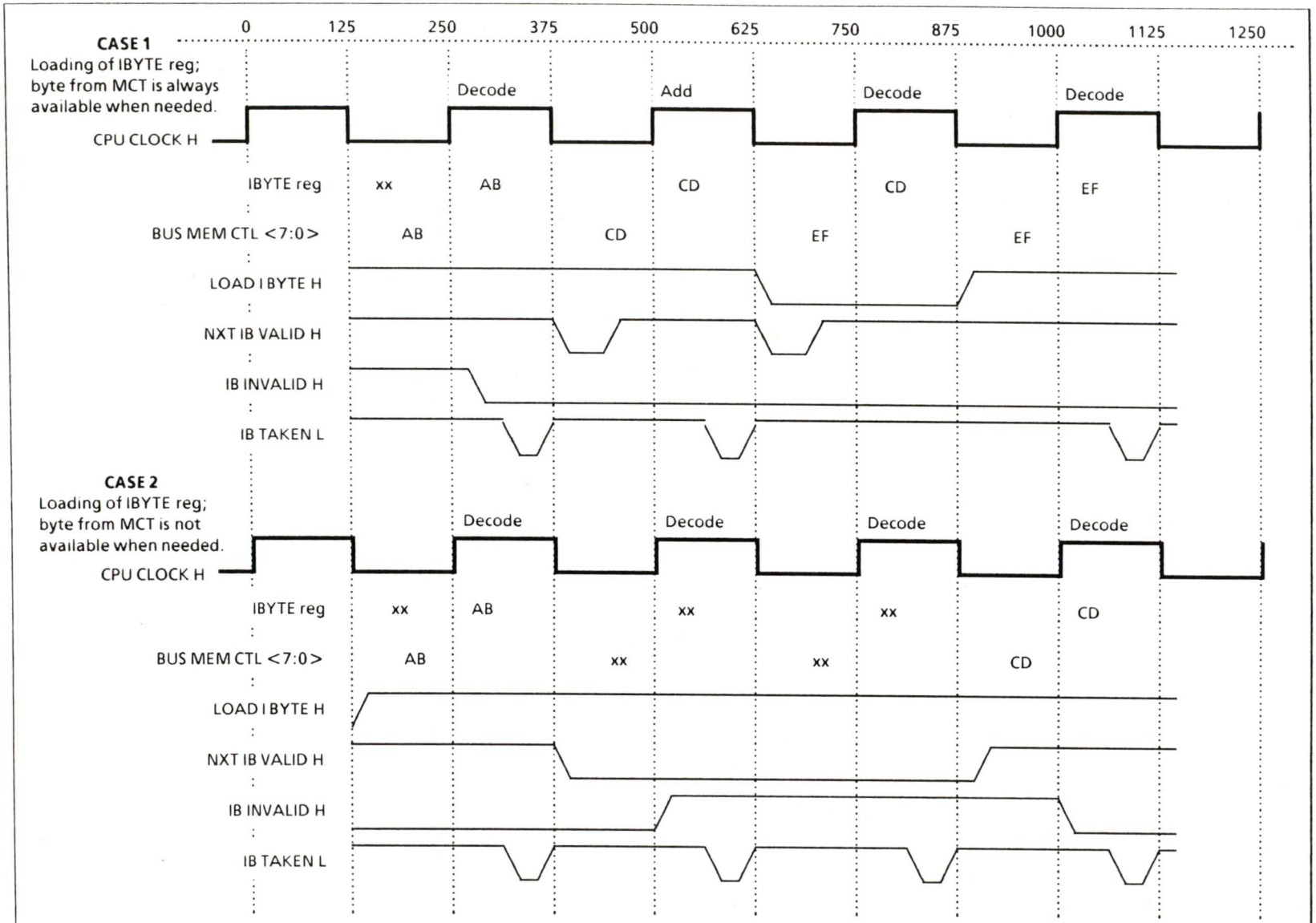


Figure 4-3. IBYTE Register Loading

Decode ROMs

The decode ROMs are logically 1K by 16 bits. They are used to select the microcode routine to be executed depending on the current contents of the IBYTE register.

Conceptually, a Decode microinstruction is the last microinstruction of the currently executing microcode routine. When the current microinstruction is a Decode, the output from the decode ROMs is driven onto the NμA bus.

The inputs to the ROMs are bits <7:0> from the IBYTE register (DAPF I BYTE <7:0> H), and the two bit control field from the current Decode microinstruction, bits <24:23> (DAPA CS <24:23> H). Bits <24:23> are encoded as follows:

24	23	Selected Decode
0	0	operand specifier decode type 1
0	1	operand specifier decode type 2
1	0	IRD for single byte opcodes
1	1	IRD on second byte of two byte opcode

If the decode operation is an IRD, the outputs from the ROMs are:

- two bits of condition code class (DAPF CC CLASS <1:0> H). For all instructions except conditional branches, these two bits define how the PSL condition codes are set. The encoding is shown in Table 4-2.
- two bits of data type (DAPF DT1/RMODE H and DAPF DT0/SP H). For all instructions except conditional branches, these two bits are encoded as follows:

00	byte	10	not used
01	word	11	longword

- twelve bits of microaddress (BUS NUA <11:00> H)
- if the instruction is a macrolevel branch, the low-order bit of the condition code class (CC CLASS <0>) and the data type field are combined to form a code that indicates which condition codes need to be tested for that specific branch. The encoding is listed in the section titled "Macrolevel Branch Control" in this chapter.

If the decode operation is an operand specifier decode, the outputs from the ROMs are:

- eight bits of microaddress (BUS NUA <07:00> H)
- one bit to specify register mode and not PC (DAPF DT1/RMODE H)
- one bit to indicate that the stack pointer (R14) is specified (DAPF DT0/SP H)
- one bit to indicate that the operand specifier being decoded is not a short literal (DAPF CC CLASS 0 H).

ALU and PSL Condition Codes

There are two separate sets of condition codes stored in the data path. The first set is the ALU condition codes which operate at the microprogram level. These condition codes result from the last ALU operation in the data path chip. They are available as jump conditions to the microcode and are also used to load the PSL condition codes.

The other set of condition codes is the PSL

condition codes. These are part of the PSL and are available to the macrolevel code. They are used to determine if a macrobranch should be taken.

Both sets of condition codes (ALU and PSL) can be read or written on the internal data bus as bits <3:0>.

Condition Code Control

The setting of the condition codes is controlled by the CC/DT field of the microinstruction, the microinstruction opcode, and the condition code class register.

For microinstruction opcodes Move, Moveout, Memory Request, I-stream Request, Multiply Step, Restore, Clear Save Stack, and Decode, the condition codes are never set and the CC/DT field is used only for data type.

For all other microinstruction opcodes, the condition codes are set as follows for the given values of the CC/DT field:

0	data type is long, CCs not affected
1	data type is long, set ALU CCs
2	data type is long, set ALU and PSL CCs
3	data type is SIZE, set ALU and PSL CCs

Condition Code Class Register

The condition code class register is part of the logic that sets the condition codes. It is loaded from the decode ROMs at the end of every macroinstruction opcode decode (also referred to as an instruction register decode, or IRD). The bits DAPF CC CLASS 1 H, DAPF CC CLASS 0 H, and DAPF DT0/SP H are loaded into this register from the decode ROMs. The first two, CC CLASS <1:0>,

contain an encoded value; the encoding is shown in Table 4-2. These register encodings are essentially setup conditions; when the value of the two bits is as given, the PSL condition codes will be set as defined in the Function column.

Table 4-2. Condition Code Class Register Encoding

<1:0>	CC Class	Function
0	Logical	ALU N to PSL N ALU Z to PSL Z ALU V to PSL V PSL C to PSL C
1	Arithmetic	ALU N to PSL N ALU Z to PSL Z ALU V to PSL V ALU C to PSL C
2	Compare	ALU N to PSL N ALU Z to PSL Z Clear PSL V ALU C to PSL C
3	Floating Point	ALU N to PSL N ALU Z to PSL Z ALU V to PSL V Clear PSL C

The output of the condition code class register is the same two CC CLASS bits, labeled DAPE CC CLASS <1:0> H.

Condition Code PALs

The ALU and PSL condition codes are stored in two 16R4 PALs. One PAL stores the ALU and PSL N and V bits, and the other stores the ALU and PSL Z and C bits. PSL <3:0> are contained in

these two PALs; that is, these two PALs contain the low four bits of the PSL register. The PALs are controlled by a four-bit condition code function field, DAPE CC <F3:F0>. This CC function field is the output of another PAL, called the CC Function, or CC Pipeline PAL. The CC function field is generated from the following five bits: DAPE CC CLASS <1:0> H, DAPC CS REG <38:37> H, and DAPC NO CC OP (1) L. This last bit indicates whether the current microinstruction is one that affects the condition codes. When NO CC OP is low, the CC function field is 0000. The encoding of the CC function field is as follows:

Table 4-3. CC Function Field Encoding

DAPE CC <F3:F0>	Function
0000	no operation, CCs unaffected
0100	load ALU CCs logical
0101	load ALU CCs arithmetic
0110	load ALU CCs compare
0111	load ALU CCs floating
1100	load ALU and PSL CCs logical
1101	load ALU and PSL CCs arithmetic
1110	load ALU and PSL CCs compare
1111	load ALU and PSL CCs floating

Because the data path chip is pipelined (that is, the microcycles overlap; see Figure 1-5), the condition codes are affected by the previous microinstruction and not the current one. The first microinstruction is decoded and the control information (the CC function field) stored until the following T0. The stored information is then used to directly control the PALs that store the condition codes. Figure 4-4

shows when the ALU condition codes are available and when they are loaded into the PALs from the data path chip for an Add microinstruction.

Condition codes are set as follows. The signals DAPF CC CLASS <1:0> H are the output from the decode ROMS during IRDs. These signals are the input to the condition code class register, and also the output from the condition code class register as DAPE CC CLASS<1:0> H. These two bits, plus DAPC CS REG <38:37> H and DAPC NO CC OP (1) L generate the CC function field, labeled DAPE SET CC <F3:F0> H. The CC function field bits are sent through a flip-flop to delay them one microcycle. As the output from the flip-flop, they are labeled DAPE CC <F3:F0> H. From there, the CC function field bits become part of the input to the two condition code PALs that store the ALU and PSL condition codes. The other inputs to these two PALs are the C, V, Z and N condition codes themselves from the last data path chip operation (DAPH DPC <C,V,Z,N> H). The PSL condition codes that are stored in these PALs (PSL <3:0>) are set according to the encoding of the CC function field and the values of DAPH DPC <C,V,Z,N> from the data path chip.

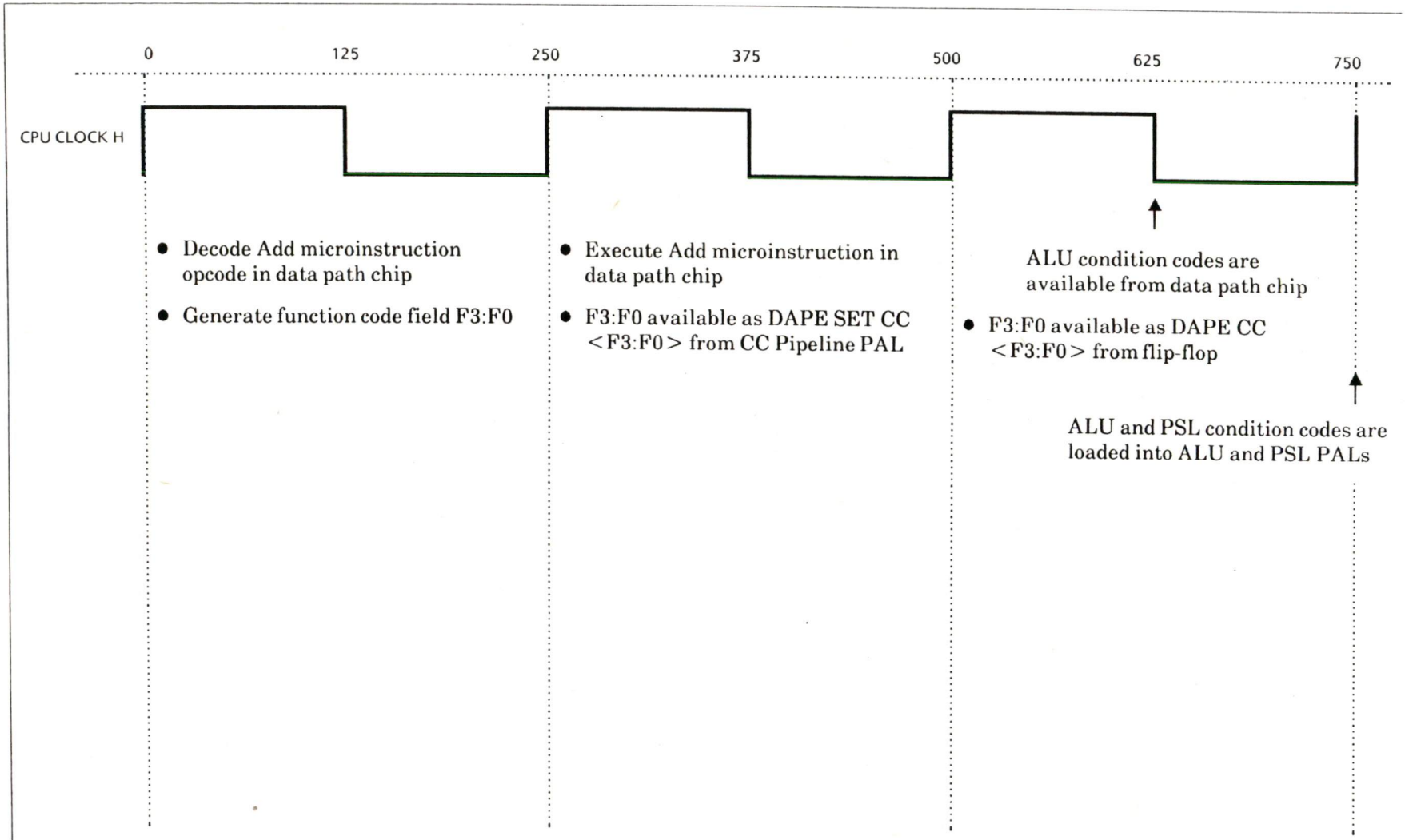


Figure 4-4. Condition Code Setting Timing Diagram

The ALU condition codes available as the output of these PALs (DAPE ALU <C,V,Z,N> H) are the stored ALU condition codes, and are available as jump conditions to the microcode, along with DAPH DPC <C,V,Z,N>. These eight signals are the inputs to a multiplexer (ALU BR MUX) that allows microbranches to be taken on either the result of the current data path chip operation (DAPH DPC <C,V,Z,N,>) or the stored ALU condition codes (DAPE ALU <C,V,Z,N>). Both the true and the inverted output of this MUX (DAPE ALU BR H and DAPE ALU BR L) go to the jump MUX as part of the microbranch control logic.

Macrolevel Branch Control

The condition code test for the macrobranch instructions is performed in the CC Class & Branch PAL. (This is the same PAL that contains the condition code class register.) The inputs to this PAL are:

- the same three bits from the decode ROMs used for the condition class register: DAPF CC CLASS 1 H, DAPF CC CLASS 0 H, and DAPF DT0/SP H,
- bit zero from the IBYTE register, and
- the PSL condition code bits, DAPE PSL <C,V,Z,N> H from the output of the condition code PALs.

The three bits in the first category of inputs listed above form a code to indicate what PSL condition code bits need to be checked:

Code	CCs Checked	Opcodes (hex)

0	N	BGEQ, BLSS (18, 19)
1	Z	BNEQ/BNEQU, BEQL/BEQLU (12, 13)
2	V	BVC, BVS (1C, 1D)
3	C	BGEQU, BCC (1E, 1F)
4	N OR Z	BGTR, BLEQ (14, 15)
5	C OR Z	BGTRU, BLEQU (1A, 1B)

Bit zero from the IBYTE register is the low-order bit of the macroinstruction opcode and indicates whether or not the branch should be taken if the tested condition is met.

The PSL condition code inputs are the current values of the conditions being checked.

The output from this CC Class & Branch PAL is the signal DAPE BR FALSE H. This signal is one of the inputs to the OR MUX, indicating that the branch is not to be taken. At IRD, this signal is always true.

PSL Enable

The PSL enable logic is contained in a 16R4 PAL. This PAL stores PSL bits 5 and 4: the integer overflow enable bit (IV) and the trace trap bit (T).

These two bits are shown on the DAP block diagram, Figure 4-1, as PSL enable. The bits are written at T0 with internal data bus <5:4>.

Size Register

The size register is used to control the data type of operations being performed in the data path chip, or the size of a datum to be transferred during a memory request operation.

Size Register Value	Data Type
0	byte (8 bits)

1	word (16 bits)
2	not used
3	longword (32 bits)

The size register is loaded at T0 of the next cycle from:

- internal data bus <1:0> when the size register is explicitly specified in the long operand of a Moveout microinstruction.
- the decode ROMs during macroinstruction opcode decodes (IRDs).
- microinstruction bits <38:37> (the data type field) during macroinstruction operand specifier decodes if the data type field specifies byte, word, or long. If an encoding of 2 is specified, then the size register is unaffected.

The outputs of the size register are:

- DAPE DPC DT1 H and DAPE DPC DT0 H. These bits are sent to the data path chip.
- DAPE SIZE 1 H and DAPE SIZE 0 H. These two signals are two of the OR MUX inputs (see Table 3-3).
- BUS ID 01 H and BUS ID 00 H. These signals are driven onto the internal data bus.

The size register is controlled by read and write signals from the ID bus address decode logic.

Executing Microinstructions

Executing microinstructions is the third of the eight functions that the data path module performs. The execution phase of almost all microinstructions takes place in the data path chip (DPC). The data path chip consists of a 32-bit data

path, register file, and ALU, and is implemented in 3 micron NMOS technology. Figure 4-5 is a block diagram of the data path chip. The chip components are described in the following paragraphs.

Clock Signal

Internally, the chip runs on a two-phase clock system consisting of Phase 1 (PH1) and Phase 2 (PH2). The clock phases are derived by dividing the clock input signal, DAPL DPC CLK H, by two internally on the chip. The clock circuitry external to the chip synchronizes the internal clock phases with the signal DAPL DPC RESET L. The low going edge of DPC CLK H that occurs immediately after DPC RESET L is deasserted forces the internal clock phases to PH1.

DPC RESET is an active low signal which has the following effects on the data path chip:

- it disables the data bus tri-state drivers,
- it presets the timer, and clears bits 0 and 1 in the timer control/status register (TMRCSR), and
- it clears the control store register, so the chip will execute NOPs.

The DPC RESET L signal is typically used on power up or testing. Parity and condition code signals are undefined during this time. The DPC RESET L signal must be active for at least eight clock periods (four microcycles). It can be asserted asynchronously to DPC CLK H, but is deasserted synchronously to DPC CLK H. Figure 4-6 shows the timing relationship between the chip clock signals, phases, and the signal LD CSR. LD CSR

causes the DPC microinstruction from the data path control store to be loaded into the control store register on the data path chip.

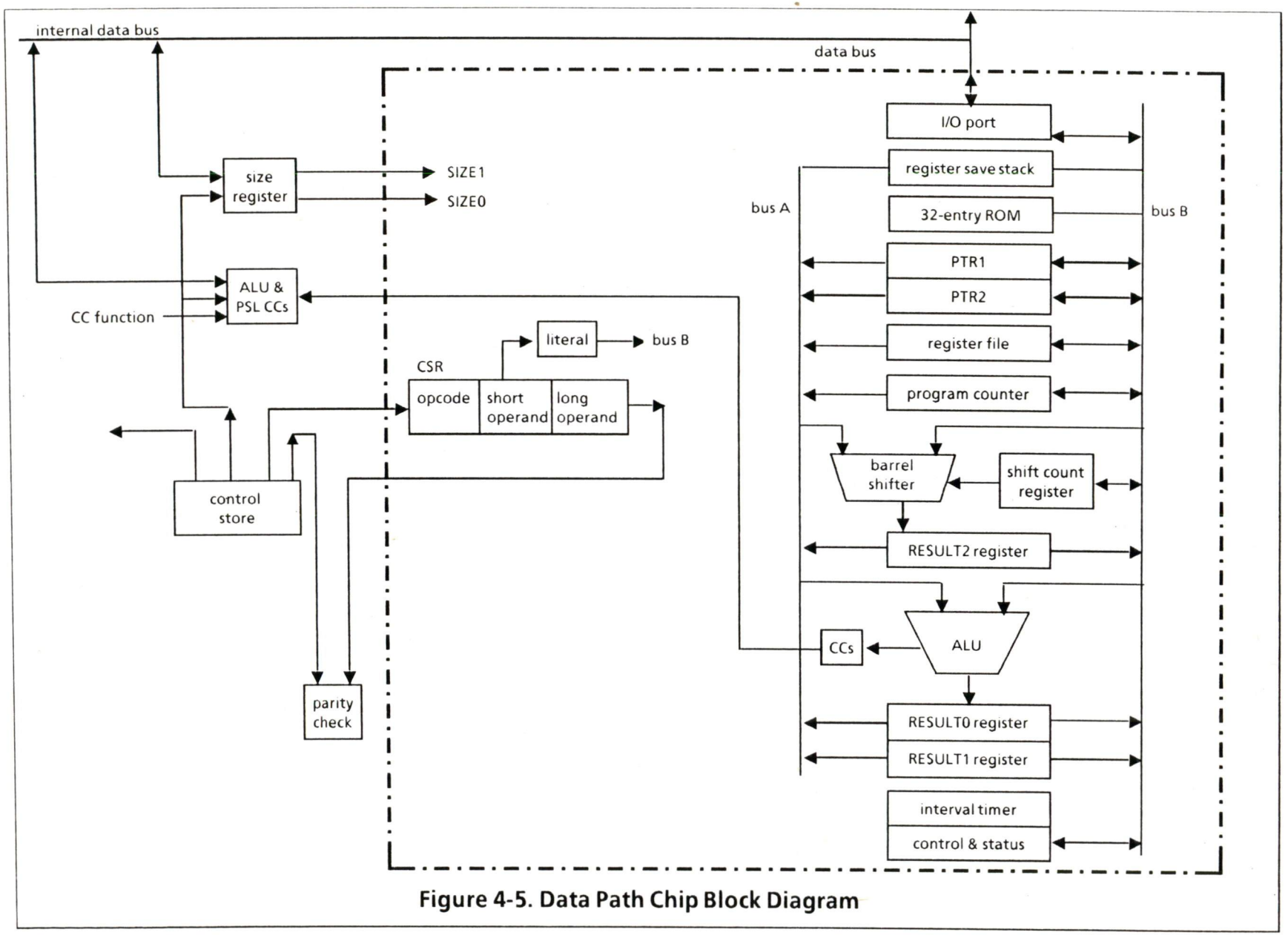


Figure 4-5. Data Path Chip Block Diagram

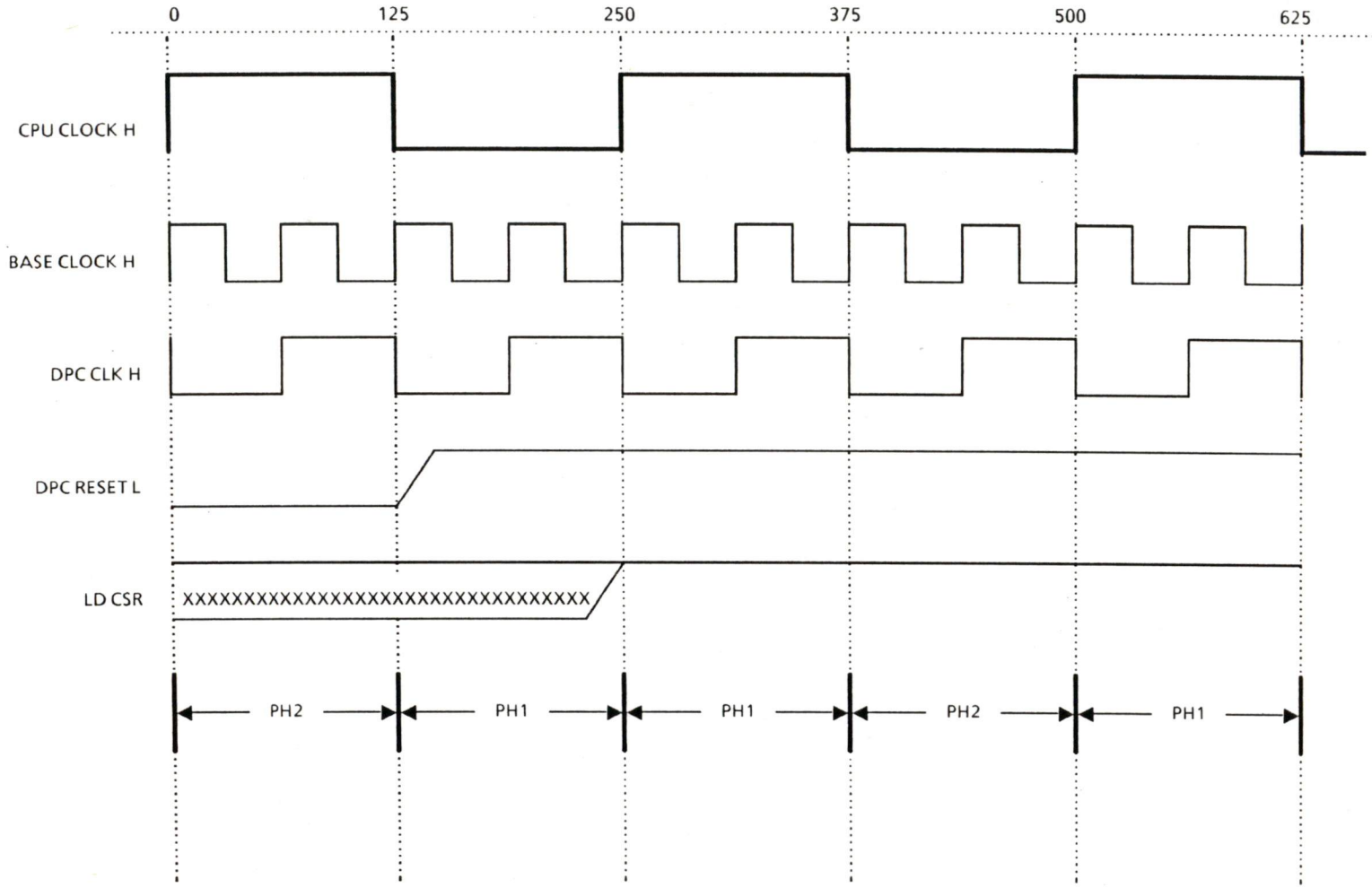


Figure 4-6. Data Path Chip Timing Diagram

Control Store Register

The control store register (CSR) is the 21-bit register that holds the DPC microinstruction. (The DPC microinstruction is bits <36:16> of the microinstruction from the control store.) A new DPC microinstruction (DAPA CS <36:16> H) is loaded into the CSR at the leading edge of every PH1, when the LD CSR signal (DAPL DLYD CPU CLK L) is active. If LD CSR is not active, the CSR is unchanged and the chip executes the same DPC microinstruction again; the size control pins are not sampled during this repeat microcycle.

Parity Generator

The parity generator on the data path chip computes parity on the 21-bit DPC micro-instruction contained in the control store register. The result is driven on the parity output pin to the parity checker external to the chip. Odd parity is generated; the parity output is one if the sum of the one bits in the DPC microinstruction is even.

Size Control

The chip supports three data types: byte, word, and longword. The size of the operation performed in the data path chip is controlled by the CC/DT field of the current microinstruction and the size register.

For all microinstructions except Memory Requests and I-stream Requests, the CC/DT field determines the size information that is sent to the data path chip, and is encoded as follows:

0, 1, or 2	data type is long

3 use size register

For Memory Request and I-stream Request microinstructions, the CC/DT field determines the size information that is sent to the data path chip, but the field encoding is interpreted this way:

0 byte

1 word

2 use size register

3 longword

The size register is loaded during IRDs (macroinstruction opcode decodes) from the decode ROM signals DAPF DT1/RMODE H and DAPF DT0/SP H; the size register may be loaded during operand specifier decodes from the CC/DT field of the Decode microinstruction, signal names DAPC CS REG <38:37> H.

When LD CSR is asserted, the data type for the current DPC microinstruction is sent to the data path chip via the size control pins, SIZE1 and SIZE0. The signals DAPE DPC DT1 H and DAPE DPC DT0 H carry the encoded data type to the pins. The encoded data type on the size control pins is long for all microinstructions (except Memory Requests and I-stream Requests) if the CC/DT field does not contain the value 3. If the current microinstruction CC/DT field does contain the value 3, or the current microinstruction is a Memory Request or I-stream Request, the data type on the size control pins is the same as the data type currently stored in the size register and is encoded as follows:

Size

Register	SIZE1	SIZE0	Data Type
0	0	0	byte
1	0	1	word
2	1	0	not used
3	1	1	longword

The data type specified by the size control pins affects the writing of the general purpose registers and the setting of the ALU condition codes; the shift microinstructions are not affected.

Data Path Chip Buses

The data path is 32 bits wide and contains two 32-bit buses called bus A and bus B. The buses are precharged during PH2 and are selectively discharged during PH1.

Bus A is used for short operand sources, with the following exceptions:

- During the Multiply Step microinstruction, bus A transfers RESULT0 back to the ALU.
- During the Decode microinstruction for a macroinstruction opcode (IRD), bus A transfers the PC to the register save stack if bit <30>, the register save bit, is set (see Figure 3-2).

Bus B is used for long operand sources and short operand destinations, with the following exceptions:

- During the Moveout microinstruction, bus B is used for the short operand source.
- During the Decode microinstruction, bus B transfers the data on the external data bus to the pointer registers.
- During the Restore microinstruction, bus B

transfers the contents of the register save stack to the specified general purpose register (GPR).

- During the I-stream Request microinstruction, bus B transfers the PC to the external data bus.

Arithmetic and Logic Unit

The ALU reads two input longwords, one from bus A and one from bus B, operates on the longwords, and writes the result into one of the result registers: either RESULT0 or RESULT1. The ALU microinstructions are those with opcodes 0 through 15 and are defined in Table 3-1.

Barrel Shifter

The barrel shifter provides four primitive functions: left shift, right shift, arithmetic right shift, and double shift (extract).

The barrel shifter concatenates two longwords, one from bus A, bits A<31:0>, and one from bus B, bits B<31:0>, to form a quadword. The higher-order longword is B<31:0>. The longword result, R<31:0>, is extracted as 32 consecutive bits from the quadword and is written in register RESULT2. The bit-offset of the 32 consecutive bits extracted from the quadword is determined by the shift count, which can come from either the shift count register, or from a literal in the short operand field. The range for the shift count is 0–31. Table 4-4 summarizes the input configurations and extract counts for the four primitive functions of the barrel shifter. "LOP" means long operand, "SOP" means short operand, and N represents the shift count.

Table 4-4. Barrel Shifter Functions

Function	B<31:0>	A<31:0>	Extract Count
left shift	LOP	zeros	32−N
right shift	zeros	LOP	N
arith. right shift	sign ext.	LOP	N
double shift	SOP	LOP	N

Register File

The register file is a RAM array containing 47 registers, each 32 bits wide. The registers can be read from bus A and bus B, and can be written from bus B. The register addresses are 0–14 and 16–47. Registers with addresses 0–14 are general purpose registers (GPRs) and may be written as bytes, words, or longwords. When a GPR is written with a length less than longword, the higher order portion is not affected. Registers with addresses 16–47 are always written as entire longwords.

Table 4-5 briefly describes the registers contained on the data path chip. Registers with addresses 48–95 are described in more detail later in this section.

Table 4-5. DPC Registers

Address	Register Name	B bus	A bus	Description
0–14	GPR(0)–GPR(14)	R/W	R	Macrolevel general purpose registers; writable as B, W, L
15	PC	R/W	R	program counter
16–23	TMP(0)–TMP(7)	R/W	R	General purpose temporary registers; writable as B, W, L
24–47	TMP(8)–TMP(31)	R/W	R	General purpose temporary registers
48	RESULT0	R	R	Result register 0 from ALU
49	RESULT1	R	R	Result register 1 from ALU
50	RESULT2	R	R	Result register from barrel shifter
51	SC	R/W		shift count register
52	PTR1	R/W	R	Pointer register for first operand specifier; pointer registers are zero-extended when read
53	PTR2	R/W	R	Pointer register for second operand specifier; pointer registers are zero-extended when read
54	@PTR1	indirect	indirect	Select working register specified by PTR1 register
55	@PTR2	indirect	indirect	Select working register specified by PTR2 register
56	TMRCSR	R/W		timer control and status register
57–63	RSVD			Reserved internal to chip
64–95	ROM	R		Constants ROM

Program Counter

The macrolevel program counter (PC) is R15; it is readable from both buses and writable from bus B. An entire longword must always be written to the PC.

The PC can be incremented by 1, 2, or 4. It is incremented by hardware on the data path chip for each of the following situations:

- an opcode Decode microinstruction is executed
- an operand specifier Decode is executed
- the long operand of the current microinstruction specifies IB.BYTE
- an I-stream Request microinstruction is executed.

Result Registers

The result of any ALU operation is stored in one of two 32-bit ALU result registers, RESULT0 or RESULT1, as specified by the result bit (bit<31>) in the DPC microinstruction (see Figure 3-2). RESULT0 and RESULT1 can be addressed using the short or long operand, and the register contents driven onto either bus A or bus B.

RESULT0 and RESULT1 combine to form a 64-bit wide shift register which is used for Multiply Step microinstructions. RESULT0 is the high-order longword. During a Multiply Step microinstruction, RESULT0 and RESULT1 are shifted right so that the least significant bit (LSB) of RESULT0 becomes the most significant bit (MSB) of RESULT1. The MSB of RESULT0 is the exclusive-OR (XOR) of the V and N condition code bits from the last add operation. (For more

information about Multiply Step, see the section titled "Multiply Step" in chapter 3.)

The result of any barrel shifter operation is stored in the 32-bit wide shift result register: RESULT2.

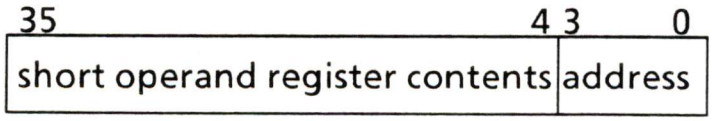
ROM

There are 32 constants stored in ROM. ROM locations are addressed by the long operand and are read onto bus B. (See Table 3-5, addresses 64–95.)

Register Save Stack

The register save stack is a pushdown stack capable of holding seven 36-bit items.

When bit <30> of the DPC microinstruction is set, both the contents of the register specified by the short operand, and the low four bits of the register address are pushed onto the register save stack in the following format:



The following microinstructions are exceptions to this: Decode, NOP, Restore, Clear Save Stack, Multiply Step, I-stream Request, and Memory Request. During these microinstructions, bit <30> is ignored, and nothing is saved on the register save stack.

The register save stack is popped using the Restore microinstruction, and is initialized by the Clear Save Stack microinstruction or by setting the

register save stack initialize bit (<25>) in a Decode microinstruction. (For more information about the register save stack initialize bit, see Table 3-4.)

Pointer Registers

Two 6-bit pointer registers, PTR1 and PTR2, can be used to indirectly address registers 0–31.

The pointer registers can also be used directly as source operands. When this is the case, their contents are zero-extended.

PTR1 and PTR2 can be written from bus B, and read on either bus A or bus B. One of the pointer registers is always written during a Decode microinstruction with the number of the register specified in the operand specifier, or with literal data if the operand specifier is a short literal; bit <26> of the DPC microinstruction selects which one (see Table 3-4). During a Decode microinstruction, data from the DBUS (data bus, Figure 4-1), are written into PTR1 for bit <26> = 0, and into PTR2 for bit <26> = 1.

Shift Count Register

The shift count register is a 5-bit register that controls the shift amount in a barrel shifter operation. The shift count register is readable and writable via bus B, and it is zero-extended when read.

Interval Timer and TMRCSR

The data path chip contains an interval timer that is available for use by any macrolevel software running on the system. The interval timer is controlled by the timer control/status register,

TMRCSR.

The interval timer is a 16-bit counter which is clocked once every microcycle. (One microcycle is 250 ns.) The counter is loaded with the constant 40,000, which causes the counter to overflow once every 10 msec. Every time the counter overflows, TMRCSR<1> is set, and the counter reloads itself with the constant. TMRCSR<1> stays set until it is written with a zero via software.

TMRCSR<0> is the Interrupt Enable bit. The timer interrupt pin of the data path chip is the logical AND of TMRCSR<0> and TMRCSR<1>. When TMRCSR<0> and TMRCSR<1> are both set, the signal DAPH TIMER REQ L is sent from the timer interrupt pin to the interrupt control logic on the external data path. DAPH TIMER REQ L is the signal represented by the label "timer req" in Figure 4-1.

Writing a zero to TMRCSR<1> clears the interrupt. Writing the timer control/status register has no effect on the contents of the counter. The Reset signal loads the constant into the counter, and clears TMRCSR<0> and TMRCSR<1>.

Condition Codes

The condition codes are the N, Z, V, C and S bits. The N, Z, V and C bits are set only when an ALU microinstruction (opcodes 1–15) or a Multiply Step microinstruction (opcode 27) is performed. During a logical microinstruction (opcodes 1–7), both the V bit and the C bit are cleared. The S bit is set only for a Shift microinstruction (opcodes 16–22). The N, Z, V, and C bits are set according to the size of the operand as specified by the size control pins.

There are only four output pins assigned to the condition codes. Since there are five condition codes, the Z bit and the S bit share the ZS pin; opcodes 16–22 drive the S bit onto the pin, and all other opcodes drive the Z bit onto the pin. Table 4-6 describes how the condition codes are set.

Table 4-6. Data Path Chip Condition Codes

CC	Affected by Opcodes	Description
N	1–15 or 27	N is set when the MSB of the result = 1; that is, the result is negative.
Z	1–15 or 27	Z is set when the result = 0.
V	1–15 or 27	V is set when an overflow on arithmetic operations (opcodes 8–15 and 27) occurs. Overflow is implemented by taking the XOR of the carry in and carry out of the MSB of the ALU. V is cleared on logical operations (opcodes 1–7).
C	1–15 or 27	For addition and multiplication (opcodes 8–10 and 27), C is the carry out from the MSB of the ALU. For subtraction and compare (opcodes 11–15), C is the complement of the carry out. C is cleared on logic operations (opcodes 1–7).
S	16–22	S is set when RESULT2<0> = 1.

I/O Port

The external registers (addresses 96–127) are located outside the data path chip in the external data path; they can only be referenced in the DPC microinstruction long operand. The I/O port is the interface between the external data path and the data path chip. The I/O port is connected to bus B. Table 4-7 briefly describes the registers located outside the data path chip.

There are 32 data path chip pins that connect the chip to the external data bus (DBUS). These pins carry the bidirectional tri-state signals labeled BUS DBUS <31:00> H. The outputs from these pins are disabled during Reset. The I/O port drives the DBUS pins only for the Moveout, I-stream Request, and Memory Request microinstructions.

Table 4-7. External Registers

Address	Register Name	Read/Write	Description
96	CON.DATA	R/W	UART data register
97	CON.STATUS	R/W	UART status register
98	CON.MODE	R/W	UART mode register
99	CON.CMD	R/W	UART command register
104	size register	R/W	bits <1:0> only; zero-extended when read
105	index register		bits <3:0> are index register, read/write; bits <7:0> are the low eight address bits of the boot ROM, write only
106	PSL.MODE	R/W	bits <1:0> only; zero-extended on reads
107	MISC register	Write	bits<7:5> diagnostic LEDs; bit 2 UART receive interrupt enable bit 4 break detect enable; bit 1 request arithmetic trap bit 3 UART transmit interrupt enable; bit 0 send Q22 bus init.
108	PSL.EN	Write only	writing bits <5:4> to this register sets the PSL IV and T bits, respectively
108	REQ.ST	Read only	these bits indicate the status of the saved memory request: bit 1 when set, the memory request mode is kernel; when clear, the mode is current bit 0 when set, access type is DAP to MCT (write); when clear, MCT to DAP (read)
109	PSL.IPL	Write only	bits <4:0>; these bits are also the high five address bits for the boot ROM
109	INT.SRC	Read only	interrupt source register; encoding: 0 no interrupt; 8 write timeout; 12 Q22 bus level 5 1–5 reserved; 9 Q22 bus level 7; 13 console receive 6 Q22 bus level 4; 10 timer request; 14 console transmit 7 power failure; 11 Q22 bus level 6; 15 not used
110	PSL.CC	R/W	bits <3:0>; zero-extended when read
111	ALU.CC	R/W	bits <3:0>; zero-extended when read
112	SID	Read	system ID register switch pack, bits <7:0> only
113	option	Read	option switch pack, bits <7:0> only
114	MISC register	Read	see 107
115	boot ROM	Read	a single byte from the boot ROM
116–119	RSVD		reserved
120	IB.BYTE	EXT	read a byte from the I-stream; PC incremented by one
121	IB.WORD	EXT	read a wordfrom the I-stream; PC incremented by two
122	not used		
123	IB.LONG	EXT	read a longword from the I-stream; PC incremented by four
124–127	MEMORY.DATA	R/W	external memory, allocated as a block

Transferring Data

Transferring data within the data path module is the fourth of the eight functions that the data path module performs. The hardware components are the internal data bus (ID bus) and the data bus (Dbus), the sign-extension logic, the ID bus latch, the ID MUX, the IBYTE buffer, the miscellaneous register, ID bus control, and zero-generator. These components transfer data within the DAP module. The following paragraphs describe each of these components in turn, and the ID bus timing.

Internal Data Bus

There is an 8-bit data path on the DAP module used to access the registers that must be visible to external hardware, such as the console UART and the switch packs. This data path is also used during instruction decode to pass operand specifier information into the data path chip. The information transfer portion of this data path is a tri-state bus called the internal data bus (ID bus).

All of the tri-state enables on the ID bus are disabled during T1. The control outputs are changed during this time and the bus re-enabled at T2. Data are always clocked into the ID bus destination at T0.

Data may be driven onto the ID bus from one of several sources, and may be written from the ID bus to one of several destinations. The following components can be sources or destinations: size register, ALU and PSL condition code PALs, index register, console UART, MISC register, and the data path chip (via the ID bus latch when it is a

source and via the ID bus input latch when it is a destination).

These components can only be sources: ID MUX, interrupt source register, IBYTE register, and zero-generator. The hardware PSL, made up of three separate registers: current mode register (PSL.MODE), PSL enable, and the IPL register, can only be destinations.

Data Bus

The DAP module communicates with the MCT module over a 32-bit tri-state bus called the memory data bus, implemented in the 50-pin, over-the-top cable. The extension of this bus on the DAP module is the data bus, or Dbus. The Dbus transfers data between the data path chip and the memory controller, and between the data path chip and the rest of the DAP module. There is buffering between the Dbus and the memory data bus (the MD bus latches in Figure 4-1) to provide the required drive for the signals transmitted over the cable.

Sign-Extension

The Dbus may also be driven by the sign-extenders. The sign-extend logic is used when displacements from the instruction stream are read into the data path chip. Word displacements are read from the memory controller over the memory data bus, while byte displacements are read from the IBYTE register directly. The sign-extension control enables the sign-extenders for a read from the ID bus, or for a word displacement read during an I-stream request microinstruction.

The input to the sign-extenders is bit 7 from the

IBYTE register, bit 15 from the memory data bus, and information about the data type. The output is data bus bits <31:16> for words (BUS DBUS <31:16> H) or data bus <31:08> for bytes (BUS DBUS <31:08> H).

ID Bus Latch

This latch holds data being driven from the low eight bits of the data path chip. The ID bus latch is needed because of the data hold times required by the UART.

ID MUX

The ID bus multiplexer gates one of the following sets of inputs onto the ID bus:

- miscellaneous register <7:0>
- boot PROM <7:0>
- option switches <7:0>
- system ID switches <7:0>

The output of the ID MUX is ID bus bits <7:0>, labeled BUS ID <07:00> H.

IBYTE Buffer

The IBYTE buffer is a 74F373 latch located between the IBYTE register and the ID bus. The contents of the IBYTE register are driven onto the ID bus through the IBTYE buffer.

The contents of the IBYTE register are read on the ID bus when the long operand of the current microinstruction specifies the IBYTE register's unique address.

The contents of the IBYTE register are also driven on the ID bus during operand specifier decodes,

and stored in one of the two pointer registers on the data path chip. When the operand specifier mode is not short literal, bits <5:4> of the IBYTE register are forced to zero to extract the register number. (Except for literal mode, operand specifier bits <3:0> always specify a register number. The register number is always saved for an operand specifier decode.) The high two bits of the IBYTE register contents (the operand specifier) do not need to be set to zero because the pointer registers are only six bits wide.

The input to the IBYTE buffer is DAPF I BYTE <7:0> H. The output is BUS ID <07:00> H.

Miscellaneous Register

This is a read/write register that contains various control bits. When a write to this register is performed, the register number specified is 107; when a read from the MISC register is performed, the register number specified is 114. The register bit definitions are the same regardless of the operation.

The input to this register is the ID bus bits: BUS ID <07:00> H. The output is eight lines to the ID MUX; some of these lines are also used for various control functions. The MISC register bit definitions are as follows.

07:05	LED bits Diagnostic LEDs 1, 2, and 3 are lit by writing zeros to these bits.
04	break detect enable When this bit is set, a break condition on the serial line causes a HALT.
03	UART transmit interrupt enable

02 UART receive interrupt enable

01 arithmetic trap request
When this bit is set, a trap is taken at the next instruction decode.

00 send Q22 bus init
This bit is used to initialize the I/O bus when requested by a MTPR instruction.

ID Bus Address Decode Logic

The operation of the ID bus is controlled by the opcode and long operand field of the microinstruction. The ID bus address decode logic receives bits CS <36:32> from control store (the microinstruction opcode), bits DT1/RMODE and CC CLASS 0 from the decode ROMs, and CS<20:16> from control store (the microinstruction long operand). With these inputs, the ID bus address decode logic generates signals to control read and write operations on the ID bus. The microinstruction opcode specifies the direction of the data transfer, and the long operand is used as an address to determine if an ID bus register is the source or the destination of the data to be transferred.

Because of the pipeline in the data path chip, the timing on the ID bus is different for reads and writes. On a read operation, the data is driven onto the ID bus at T2 of the microinstruction requesting the read. The difference on write operations is that the data is not available from the data path chip until just before T2 of the microinstruction following the one requesting the write. The long operand and some of the control information is stored in a pipeline register which

then provides the necessary write enable signals to the destination registers one cycle later. Figure 4-7 shows the timing for reads from ID bus registers. Figure 4-8 shows the timing for writes to ID bus registers.

Logic to decode the microinstruction opcode is part of the ID bus address decode logic; the microinstruction opcode needs to be decoded to allow the data path elements to behave differently depending on the operation required. For example, the memory controller needs to detect Memory Request and I-stream Request opcodes. The inputs to the microinstruction opcode decode logic are the microinstruction opcode field CS <36:32>, and CS <24> to differentiate between operand specifier and opcode decodes. The outputs are:

- a signal named NO CC OP H to inform the condition code logic that the condition codes are not changed for this instruction
- a signal named DECODE L to indicate that the current microinstruction is a Decode
- a signal named MEMORY OP L to inform the memory controller that the current microinstruction is a memory function
- a signal named MOVEOUT L to control the direction of data flow in the data path.

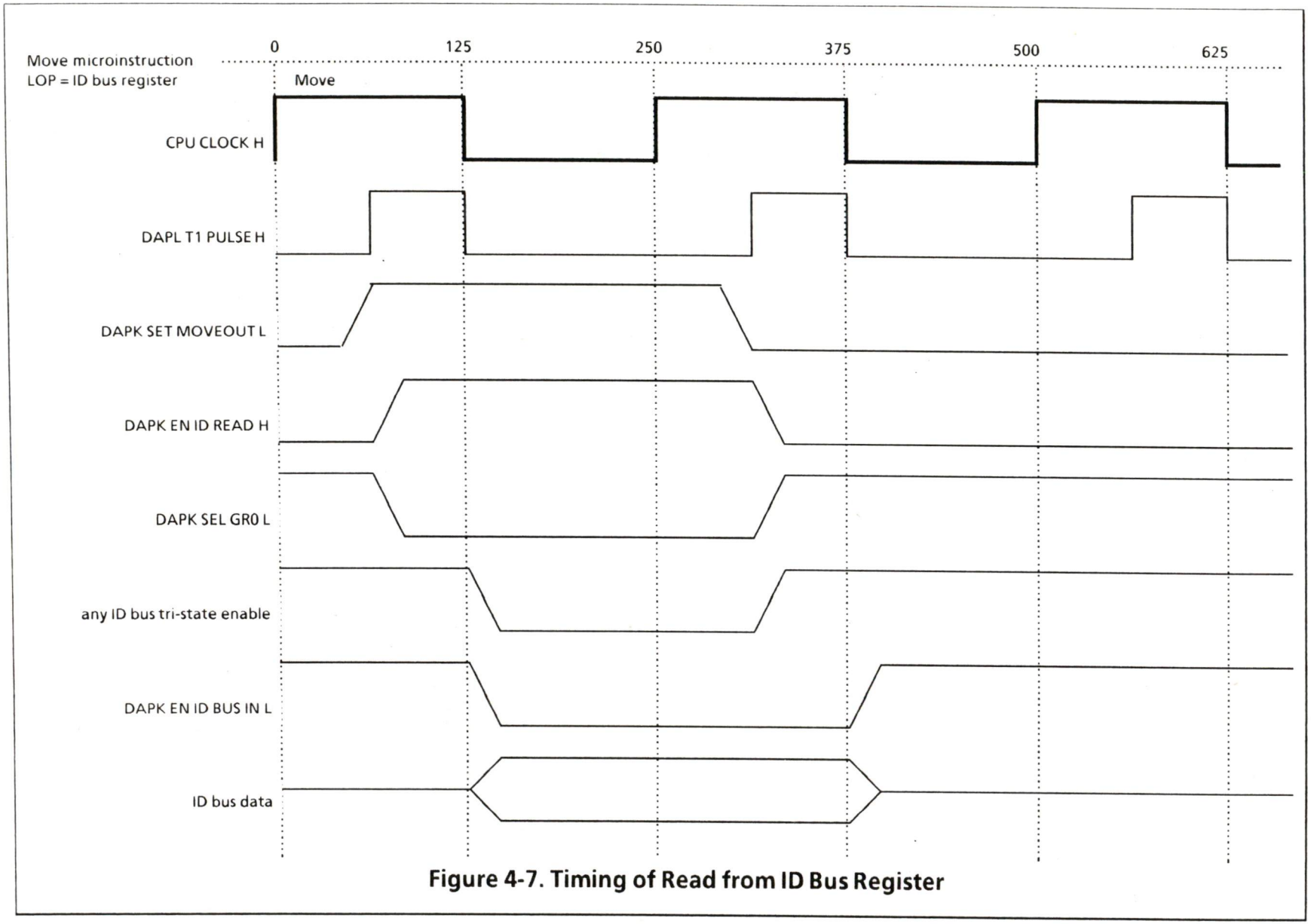


Figure 4-7. Timing of Read from ID Bus Register

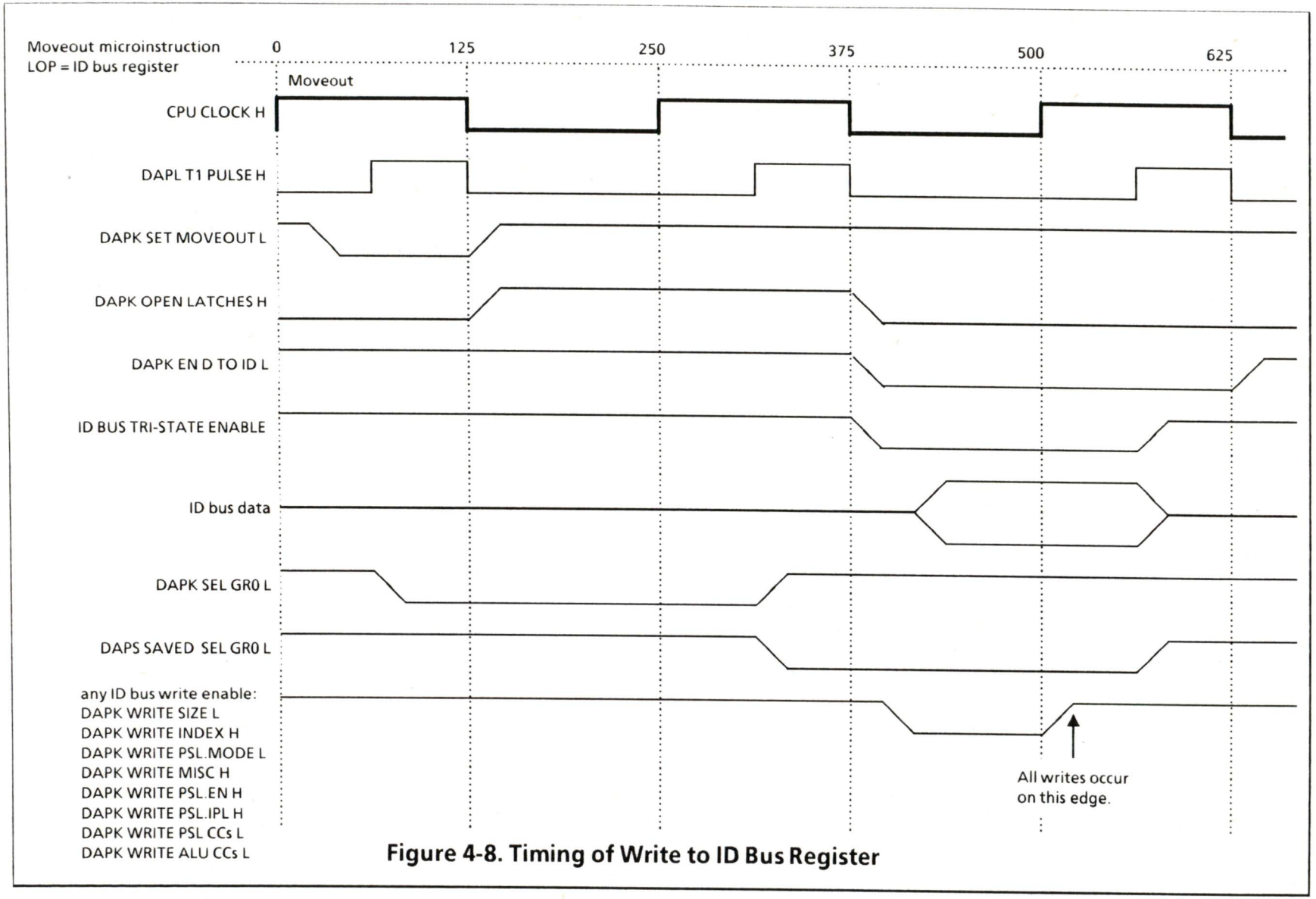


Figure 4-8. Timing of Write to ID Bus Register

Zero-Generator

Several of the readable registers on the ID bus contain less than eight bits. The microcode requires that these registers be zero-extended when read, so a zero generator is connected to the ID bus. The zero-generator is implemented as a 16L8 PAL. It is enabled when any register containing less than eight bits is read; the zero-generator drives zeros on the unimplemented bits of that register.

The input to the zero-generator is the low-order three bits of the microinstruction long operand and some control signals. The output is bits BUS ID <07:02> H. If the ID bus register being read is INT.SRC (interrupt source), PSL.CC, or ALU.CC, BUS ID <07:04> are driven onto the ID bus as zeros. If the ID bus register being read is the size register, PSL.MODE (current mode register), or REQ.ST (memory request status), BUS ID <07:02> are driven onto the ID bus as zeros.

Processing Interrupts

Processing interrupts is the fifth of the eight functions that the data path module performs. The hardware components are the interrupt priority level (IPL) register, the interrupt control logic, the priority encoder, and the interrupt source register. The following paragraphs describe each of these components in turn.

IPL Register

The interrupt priority level register stores the current processor priority. This priority is used by

the interrupt control logic to determine if an interrupt request is to be granted.

The IPL is changed when an interrupt is taken, when a MTPR or REI macroinstruction is executed, or during certain exception conditions. These instructions use a temporary register on the data path chip to store the new IPL until it is written into the IPL register. The IPL register is written at T0 from ID bus bits <4:0>.

Interrupt Control Logic

The interrupt control logic on the data path module informs the microcode of pending interrupt requests. These requests can be generated by local hardware (for example, power fail) or can come from the Q22 bus. The priority encoder and the interrupt source register are actually part of the interrupt control logic; this logic is always enabled.

The interrupt request lines from the Q22 bus are received in an 8640 bus receiver, synchronized to the CPU clock (DAPL CPU CLOCK H) and sent to the priority encoder. Interrupt request signals from internal sources are also sent to the priority encoder.

The hardware compares the IPL of the Q22 bus device requesting the interrupt with the current processor IPL. If the IPL of the Q22 bus device is higher, the interrupt is served at IPL 17 (hex). The microcode that services Q22 bus interrupts then reads the interrupt source register to determine which Q22 bus device actually caused the interrupt.

Priority Encoder

All active interrupt requests are prioritized in the

priority encoder. The encoded output value is compared with the interrupt priority level (IPL) from the hardware PSL. If the priority of the request is greater than the current IPL, the interrupt request flag (DAPN INT REQ H) is sent to the OR MUX and jump control logic in the microsequencer.

If an interrupt request is pending during an IRD (macroinstruction opcode decode), it causes a microtrap. INT REQ H is the third signal in one of the OR MUX inputs (see Table 3-3). Since the OR MUX is enabled for an IRD, the OR MUX output is 0100 if an interrupt request is pending and no other condition is present; the other next microaddress bits are forced to zeros (see Table 4-1). Thus, if an interrupt request is pending and an IRD is executed, a microtrap is taken to control store address 0004. A microinstruction routine to handle interrupt requests starts at this address.

The comparison between the encoded output value from the priority encoder and the current IPL is done in a 16R4 PAL; this PAL also contains the interrupt source register (INT.SRC).

Interrupt Source Register

The encoded output value from the priority encoder is the input to the interrupt source register. This value is compared with the processor IPL; the comparison produces a 4-bit code which is loaded into the interrupt source register if the request priority is higher than the current processor IPL. The microcode identifies the source of an interrupt request by reading this 4-bit code in the interrupt source register. The register encoding is shown in Table 4-8.

Table 4-8. Interrupt Source Register Encoding

Interrupting Event	IPL (hex)	INT.SRC Register
power failure	1E	1111
write timeout	1D	1000
Q22 bus level 7	17	1001
timer request	16	1010
Q22 bus level 6	16	1011
Q22 bus level 5	15	1100
console receive	14	1101
console transmit	14	1110
Q22 bus level 4	14	0111

When the interrupt source register is read by the microcode, the following interrupt requests are cleared by the hardware if they are the highest priority: write timeout, console receive, and console transmit.

The output from the interrupt source register is bits BUS ID <03:00> and the interrupt request signal to the microsequencer.

Communicating with the Console Terminal

Communicating with the console terminal is the sixth of the eight functions that the data path module performs. The console port consists of an EIA standard RS232 line interface and a 2661 UART. The external connection to this interface is through a 10-pin Berg cable header mounted on the DAP board. The hardware components are the console UART and registers, the UART buffer, option switches, the charge pump, and break and halt detection. The following paragraphs describe

each of these components in turn.

Console UART

The console UART and the RS232 line interface provide the connection to the console terminal. The UART is connected through the UART buffer to the ID bus, and can be read or written directly by the microcode. The baud rate is selected in the option switches and can be set for 300, 1200, 9600 or 19.2K baud. The RS232 transmitter and receiver always operate at the same speed. The microcode reads the option switches during power up and programs the UART for the selected baud rate.

The console UART can request interrupts for either "transmit done" (DAPP XMIT DONE L) or "input ready" (DAPP REC RDY L).

The UART clock (DAPP UART CLK H) comes from a 5.0688 Mhz crystal oscillator that is driven directly into the UART.

Console UART Registers

The UART has programmable mode and status registers to select different speed and character length options. There is also a break detect; the signal DAPP BREAK H is asserted when the BREAK key on the console terminal is pressed.

The UART mode and status registers are written by the microcode on power up to allow the UART to correctly interface with the console terminal. Four addresses, 96–99, are assigned to the UART in the long operand address space. On power up, the microcode initializes the UART to operate in the mode required. Once the UART is initialized, it is accessed only to read and write characters to the

console terminal. Table 4-9 gives the UART register addresses and a brief description.

Table 4-9. UART Registers

Address	Register	Description
96	CON.DATA	Contains character received or to be transmitted
97	CON.STATUS	Contains UART status
98	CON.MODE	Consists of two mode registers that set operating conditions
99	CON.CMD	UART command register; sets operating mode

The following paragraphs describe these registers in more detail.

UART Data Register

CON.DATA is an eight bit register that contains the ASCII character to be transmitted to the console terminal, or the ASCII character received from the console terminal.

If an ASCII character is to be transmitted to the console terminal, the character written into CON.DATA is ID bus bits <07:00>; BUS ID <07:00> H are written into the UART buffer from the ID bus, then transmitted to CON.DATA in the console UART.

Similarly, an ASCII character received from the console terminal and stored in CON.DATA is read onto the ID bus as BUS ID <07:00> through the

UART buffer.

UART Status Register

CON.STATUS contains bits that indicate the status of the RS232 receiver and transmitter. The bits are defined as follows.

7:6 data set status
Seahorse does not use the modem control feature of the 2661 UART, so these bits are ignored by the microcode.

5 framing error
This bit is set when a stop bit is not received following the last data bit of a received character. Bit 5 is cleared by writing a one to the reset error bit in the command register (CON.CMD <4>).

4 overrun error
This bit is set when an incoming character is received before the previous received character has been read by the microcode. This bit is cleared by writing a one to the reset error bit in the command register (CON.CMD <4>).

3 parity error
This bit is not used in Seahorse.

2 data set change
This bit is not used in Seahorse.

1 receiver ready
This bit is set when a character is received from the serial line. It is cleared when the UART data register

(CON.DATA) is read.

0 transmit done
This bit is set when the RS232 transmitter has completed transmission of a character. It is cleared when the UART data register (CON.DATA) is written.

Bits <1> and <0> of CON.STATUS are read by reading the same address as the index register. CON.STATUS <1> is the signal DAPP REC RDY L; it is stored in bit 6 of the index register as DAPK REC RDY (1) H. CON.STATUS <0> is the signal DAPP XMIT DONE L; it is stored in bit 7 of the index register as DAPK XMIT DONE (1) H.

The signals DAPK REC RDY (1) H and DAPK XMIT DONE (1) H generate the interrupt requests for "input ready" and "transmit done," respectively.

UART Mode Registers

Mode registers 1 and 2 define the general operational characteristics of the UART and are accessed only during power up. The two mode registers are accessed by performing either the read or the write operation at that address twice. The first operation accesses mode register 1, and the second accesses mode register 2.

Mode Register 1. This register is initialized to 4E (hex) in the Seahorse system to define the following setup conditions:

bits <7:6> stop bit length
This bit is initialized to 01 to define one stop bit at the end of the eight-bit character being sent or received.

bits <5:4> parity control
This bit in initialized to 00 to define no parity checking.

bits <3:2> character length
This bit is initialized to 11 to define 8-bit characters.

bits <1:0> baud rate multiplier
This bit is initialized to 10 to define an asynchronous, 16X clock rate.

Mode Register 2. This register is used to set operating conditions and the baud rate of the UART. Only four baud rates are supported. The bit definitions for mode register 2 are:

bits <7:4> clock source, break detect enable
This bit is initialized to 1111 to define the internal baud rate generator as the clock source, and to enable break detection.

bits <3:0> baud rates:

0101	300 baud
0111	1200 baud
1110	9600 baud
1111	19200 baud

UART Command Register

CON.CMD is used to enable the UART and set the operating mode to either normal or self-test. The bits are defined as follows.

7:6 operating mode

00	normal operation
01	not used
10	local loop back In this mode, a character written to the transmitter

		will be received by the receiver.
	11	not used
5		request to send (RTS) This bit is initialized to a one.
4		reset error Writing a one to this bit clears the receive error flags in the status register (CON.STATUS).
3		force break This bit is initialized to zero.
2		receiver enable This bit is initialized to a one.
1		data terminal ready (DTR) This bit is initialized to a one.
0		transmitter enable This bit is initialized to a one.

Initializing the UART

The correct sequence must be used to set up the UART initial conditions. The sequence is:

1. write mode register 1
2. write mode register 2
3. write command register

If the baud rate is to be changed, the UART must first be disabled by clearing the receiver and transmitter enable bits in the command register (CON.CMD <2> and <0>). The UART must then be reset following the above sequence.

UART Buffer

The UART buffer is a 74LS245 bus transceiver. The input to the UART buffer is bits BUS ID <07:00> H when a write to CON.DATA occurs. The eight bits stored in the UART buffer are then written into CON.DATA.

When a read from CON.DATA occurs, the input to the UART buffer is the eight bits from the CON.DATA register. The eight bits stored in the UART buffer are then driven onto the ID bus as BUS ID <07:00>.

Option Switches

The option switches (an eight-switch DIP) select the UART baud rate, the default boot device, and the halt recovery action. The output from the switches is eight data lines to the ID bus MUX. The switch definitions are as follows; the default switch settings are in bold.

<7> reserved

<6> break detect enable
This switch determines the state of bit <4> in the MISC register:

0 **break detect disabled (MISC <4> = 0)**

1 break detect enabled (MISC <4> = 1)

<5:4> halt recovery action
These switches specify the action to be taken when the machine halts:

0 halt

1 boot/halt

2 **restart/boot/halt**

3 restart/halt

<3:2> boot device selection

These switches specify the boot device:

0	**RX50**
1	RDXX
2	?
3	?

<1:0> baud rate selection

These switches specify the console terminal baud rate:

0	300
1	1200
2	**9600**
3	19200

The boot device selection and halt recovery actions are explained in more detail in the section titled "Powering Up" in this chapter.

– 12 Volt Generator

The RS232 drivers require a −12 volt power source; −12 volts is not available from the system power supply. Therefore, the DAP module contains a charge pump circuit to generate this voltage. The circuit operates by alternately charging two capacitors to +12V and using them to charge a third capacitor; the −12 volt output is taken from this third capacitor.

Break and Halt Detection

There are three situations in which break or halt detection needs to occur:

- the HALT button on the Seahorse system front panel is pressed
- the BREAK key on the console terminal is pressed

- a Halt macroinstruction is executed in kernel mode.

Pressing the HALT button on the front panel asserts the Q22 bus halt line, B HALT L. This is received by an 8640 bus receiver. The output of the receiver is the signal DAPP RCVD HALT H. DAPP RCVD HALT H generates the signal DAPS HALT REQ H.

Pressing the BREAK key on the console terminal asserts the signal DAPP BREAK H. If the break detect enable bit in the MISC register is set (bit <4>), DAPP BREAK H generates the signal DAPS HALT REQ H. (The setting of option switch <6> determines the state of bit <4> in the MISC register.)

Once DAPS HALT REQ H is asserted because either the HALT button or the BREAK key was pressed, it generates two signals: DAPK CONSOLE HALT H which is one input to the jump MUX, and DAPP T BIT OR CON H which is one input to the OR MUX. When DAPP T BIT OR CON H is asserted, the output from the OR MUX is 0010 (binary). At the next IRD, a decode trap is taken to address 0002 in control store (see Table 4-1). From this location, a jump is taken to the address of the console halt microroutine. The console halt microroutine checks if the trace bit (T bit) in the PSL is set. If it is, the microcode branches to the trace pending fault microroutine. If the T bit is not set, the microcode branches to the console stop microroutine. The console stop microroutine displays a halt code on the console terminal, and examines the option switches to determine the halt recovery action. Thus, pressing the BREAK key has the same effect as pressing the

HALT button, if MISC register bit <4> is set. If MISC <4> is not set, pressing the BREAK key has no effect.

When a Halt macroinstruction is decoded, the output from the decode ROMs is the address of a microroutine that checks if the PSL mode is kernel. If the mode is not kernel, a jump to the reserved instruction fault microroutine is taken. If the mode is kernel, a jump is taken to the address of the console stop routine. The console stop microroutine displays a halt code on the console terminal, and examines the option switches to determine the halt recovery action.

Powering Up

Powering up is the seventh of the eight functions that the data path module performs. This section describes the power up signals, power failure, the initialization state of the CPU, initialization signals on power up, the option switches, the boot PROM, and the system identification (SID) register.

Power Up Signals

A power up sequence is usually the result of turning the Seahorse system power switch on; however, a power up sequence is also initiated on the Q22 bus when the RESTART button on the system front panel is pressed, or when power fails then returns.

When DC voltages are first supplied to the Q22 bus from the power supply, the power supply logic negates the signal BDCOK H, and then asserts it 3 ms after DC voltages have reached their specified

levels.

The signal DAPL INIT L is asserted by the DAP module as soon as DC voltages appear, synchronized with the 62.5 ns clock (MCTM BASE CLOCK H), and deasserted two clock cycles (125 ns minimum) after BDCOK H is asserted. DAPL INIT L is generated from BDCOK H.

The signal BINIT L is asserted by the DAP board as soon as any DC voltages appear; it is deasserted as soon as DAPL INIT L is deasserted. BINIT L initializes the Q22 bus.

Another power supply logic signal, BPOK H, is negated when DC voltages first appear, and is asserted 70 ms after BDCOK H is asserted. If power does not remain stable for 70 ms, BDCOK H is negated; therefore, Q22 bus devices must suspend critical actions until BPOK H is asserted. BPOK H must remain asserted for a minimum of 3 ms.

Power Failure

The DAP module monitors the Q22 bus power status signals: BPOK H and BDCOK H. A power failure occurs when the AC voltage to the power supply drops below 75% of the nominal voltage for one full line cycle (15–24 ms). When a power failure is detected, BPOK H is negated. Once BPOK H is negated, the entire power down sequence, as follows, must be completed.

BPOK H is synchronized with the CPU clock (CPU CLOCK H) to generate the signal DAPK PWR DWN L; DAPK PWR DWN L is asserted when BPOK H is negated. DAPK PWR DWN L is the interrupt signal generated to inform the macrolevel software that a power failure has

occurred. A power fail interrupt is initiated if the current IPL is less than 1E (see Table 4-8).

The software executes a MTPR macroinstruction that sets bit <0> of the MISC register; bit <0> is the "send Q22 bus initialization" flag. The MTPR instruction is executed no later than 3 ms after the negation of BPOK H. This causes BINIT L to be asserted for 8–20 μs.

Once BPOK H is negated, the power supply guarantees a minimum of 4 ms before BDCOK H is negated. This 4 ms allows mass storage and similar devices to protect themselves against erasures and erroneous writes during a power failure.

DAPL INIT L is a synchronized version of BDCOK H; it is asserted two clock cycles (125 ns minimum) after BDCOK H is deasserted.

The DAP module asserts BINIT L again, no later than 1 μs after the negation of BDCOK H.

DC power must remain stable for a minimum of 5 μs after BDCOK H is negated.

Figure 4-9 shows the power up and power down sequences, and the signal states for normal power.

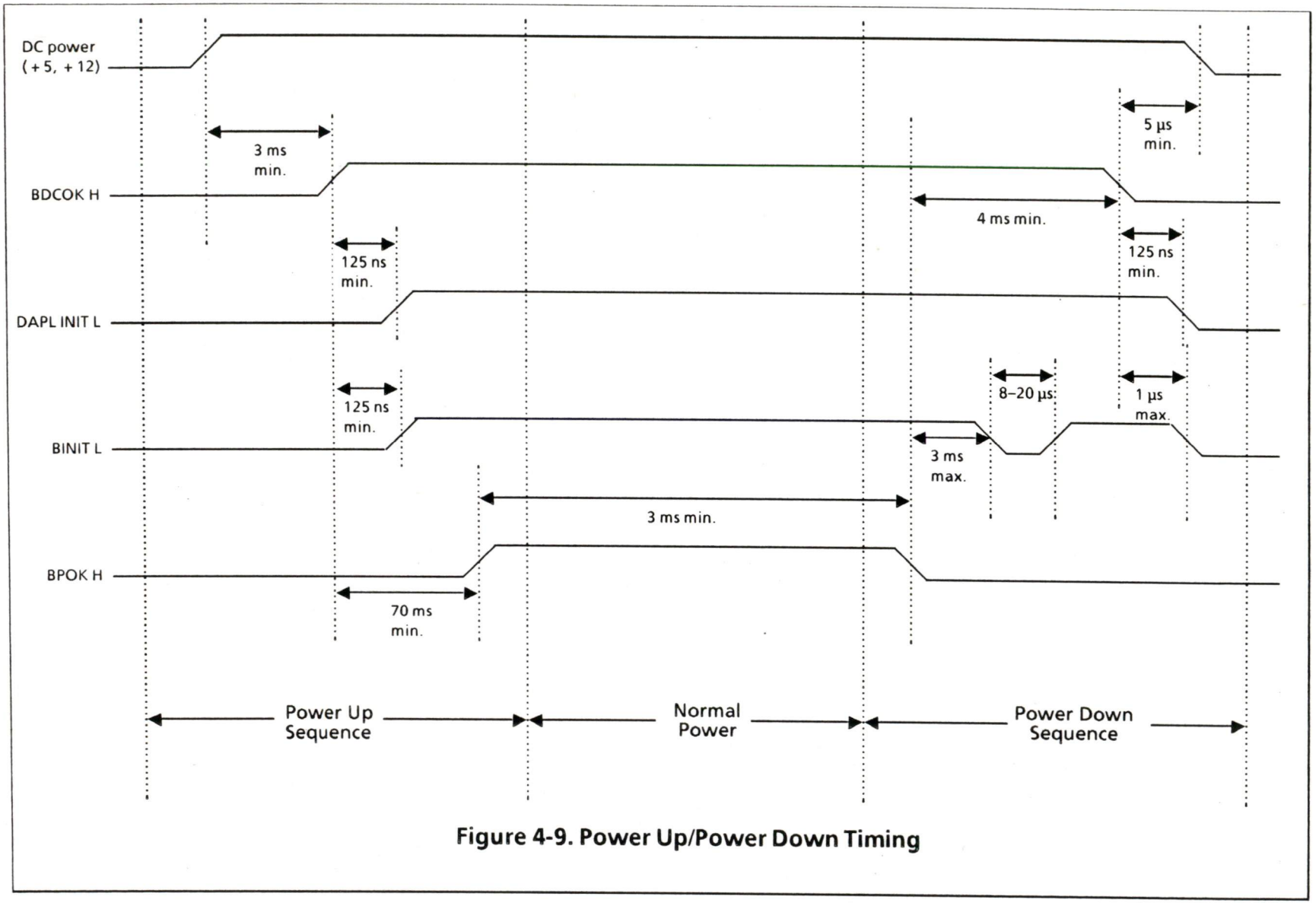


Figure 4-9. Power Up/Power Down Timing

Initialization State

The initial state of the KD32-AA CPU is set by the INIT signals: DAPL INIT L, DAPL INIT A L, and DAPL MCT INIT L, which are the buffered outputs of DAPL INIT H. (DAPL INIT H is generated by the Q22 bus signal BDCOK H.) The initial state of the CPU is defined as follows.

- The current microaddress is ZERO; that is, the first microinstruction executed following the deassertion of DAPL INIT L will be from location 0000 in control store.
- The control store parity error flag is cleared.
- No interrupt requests are pending.
- The index register is cleared.
- The hardware PSL is cleared.
- The MISC register is cleared, causing the three diagnostic LEDs to be lit.
- The memory request signal (DAPR MEM REQUEST L) is in the deasserted state.
- Q22 bus signal BINIT L is asserted during the deassertion of BDCOK H.

In addition, all of the flip-flops that synchronize the DAP clock signals are set to a known state. This guarantees that the clock signals have the correct relationship to each other.

Initialization Signals on Power Up

When the signal DAPL INIT L is asserted during power up, it generates the signal JAM UPC L, which causes the microprogram to jump to the microinstruction located at control store address

0000.

The signal BPOK H is synchronized with the CPU clock (CPU CLOCK H) to generate the signal DAPK PWR DWN L. The signal DAPS PUP H is generated by DAPK PWR DWN L, but is synchronized with the delayed CPU clock (DLYD CPU CLK H). DAPS PUP H, when asserted, clears JAM UPC L.

DAPL INIT L generates the signal DAPL SET DPC INIT L, which in turn generates the signal DAPK DPC INIT L. The assertion of DAPS PUP H causes the signal DAPK DPC INIT L to be deasserted on the next leading edge of the delayed CPU clock. DAPK DPC INIT L generates the signal DAPL DPC INIT L. The deassertion of DAPK DPC INIT L causes the signal DAPL DPC INIT L to be deasserted on the next trailing edge of the 125 ns data path chip clock (DPC CLK H). (The assertion of DPC INIT L resets and synchronizes the data path chip.)

On the next leading edge of the CPU clock following the deassertion of DPC INIT L, the microinstruction located at control store address 0000 is executed. Figure 4-10 shows all of these power up and initialization signals, and their relationship to the various clock signals.

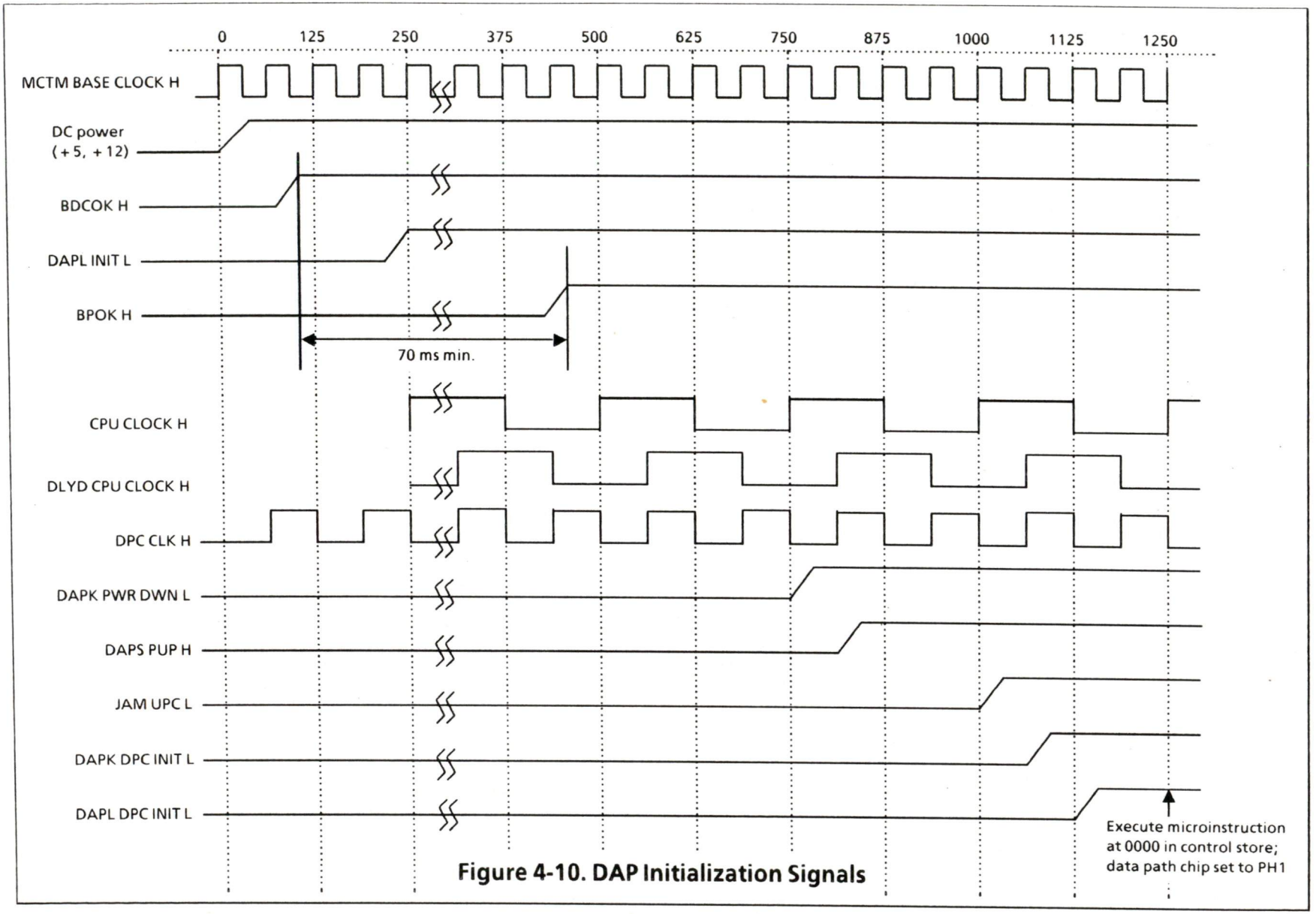


Figure 4-10. DAP Initialization Signals

Option Switches

In addition to specifying the baud rate and defining the default boot device, the option switches select the halt recovery action.

When a halt condition is encountered, the console stop microroutine prints a halt code on the console terminal. The microcode then examines option switches <5:4> to determine the halt recovery action. These switches are also examined during power up after the successful completion of microverify to determine the power up action. The switches select one of four possible strategies:

- If <5:4>=0, the system enters console mode and waits for input from the console terminal.
- If <5:4>=1, the system attempts to boot using the default boot device specified in switches <3:2>. If the boot fails, the system enters console mode.
- If <5:4>=2, the system attempts to restart. If restart fails, the system attempts to boot using the default boot device. If boot fails, the system enters console mode. This is the default configuration for the switches.
- If <5:4>=3, the system attempts to restart. If restart fails, the system enters console mode.

Boot PROM

This is an 8192 by 8-bit-wide EPROM used to store the VAX macrocode necessary to boot the Seahorse system. Four macrocode routines are stored in the EPROM, one for each possible boot device. Option switches <3:2> select which of the four macrocode

routines is addressed. Once the appropriate macrocode routine is selected, the PROM is addressed by loading the low eight bits of the address into the index register and the high five bits into the PSL.IPL register. The proper address bits are loaded into the index and PSL.IPL registers by the initialization microcode routine. Thus, a byte at a time is accessed in the selected bootstrap macrocode routine.

As each byte is accessed, it is driven onto the ID bus and out the memory data bus to the memory controller. From there, the byte is written into the Q22 bus memory. In this manner, the entire bootstrap macrocode from the PROM is copied into main memory. Once it is copied, the main memory address of the first byte is loaded into the program counter, and the bootstrap macrocode executed.

The bootstrap macrocode reads logical block one of the selected boot device into the second page of main memory and sets the stack pointer and the program counter to that address; that is, to the address of the first byte in the second page. The program in the second page of main memory is then executed. It is the responsibility of this program to complete the bootstrap process.

System Identification Register

The system identification register is a read-only constant register, four-bytes wide; the high-order three bytes are built by software and the low-order byte is set in an eight-switch DIP. The high-order byte contains a number that uniquely identifies the processor; Seahorse is identified by the number 7. The next high-order byte (bits <23:16>) is reserved; bits <15:8> specify the microcode

revision level. The low-order byte specifies the hardware revision level; this is the hardware part of the register and is read on the ID bus. A buffer is located between the switch pack and the ID bus to drive the data set in the switches onto the ID bus. (*Note: SID is not read on power up; only when an MFPR is executed. Info in SID is used for error logs. This info does not belong here and will be moved somewhere else for second draft.*)

Communicating with the MCT

Communicating with the memory controller is the eighth of the eight functions that the data path module performs. This section describes the data interface, the control interface, interface control signals, stalls, the MD bus latches, memory function latches, memory function control, the sign extenders, the PSL.MODE register,and memory reference timing.

Data Interface

The data interface between the data path and memory controller modules is the memory data bus (MDB) which carries 32 tri-state signals. The memory data bus is part of the 50-pin, over-the-top cable that connects the two boards. The tri-state signals are named BUS MEM DATA <31:00>. The tri-state enables for these data bus signals are controlled by either the DAP module or the MCT module, depending on the direction of data transfer.

The following situation causes BUS MEM DATA <31:00> to be sent from the memory controller to the data path module over the memory data bus: a memory request microinstruction is executed,

followed two cycles later by the execution of a microinstruction that is not a Moveout and that has MEMORY.DATA specified as the long operand. (MEMORY.DATA represents addresses 124–127; these addresses are allocated as a block. MEMORY.DATA can be thought of as the address of the memory data bus. When the long operand of a microinstruction specifies MEMORY.DATA, the data to be operated on are the 32 bits currently on the memory data bus.)

There are three microinstructions that cause BUS MEM DATA <31:00> to be sent from the data path module to the memory controller over the memory data bus. They are:

- a Memory Request; BUS MEM DATA <31:00> represent a virtual address.
- an I-stream Request; BUS MEM DATA <31:00> are the unincremented contents of the program counter.
- a Moveout; the long operand specifies MEMORY.DATA.

Control Interface

The control interface between the data path and memory controller modules consists of eight bidirectional signals, seven unidirectional lines from DAP to MCT, and eleven unidirectional lines from MCT to DAP, which return the status of the memory controller to the DAP microsequencer. All of these control signals and the clocks are carried on the C/D slots of the backplane (see Figure 1-3). The eight bidirectional signals are the memory control bus (MCB) and are named BUS MEM CTL <7:0>.

The memory control bus is a time-multiplexed tri-state bus which may be driven from either the DAP or the MCT module. Control information from the DAP microinstruction is driven in the first half of the microcycle (during T1). Instruction stream bytes are driven from the memory controller to the IBYTE register during the second half of the microcycle (during T3).

Interface Control Signals

When a microinstruction specifying a memory request function is decoded, the data path module drives the contents of the register specified by the long operand (usually a virtual address, but possibly a physical address or the actual data) out the data bus and onto the memory data bus. The encoded memory function is driven onto the memory control bus. The data path module then asserts the memory request line, DAPR MEM REQUEST H. This signal informs the memory controller that a new function code is on the control bus.

The memory controller responds by accepting the 32 bits on the memory data bus (a virtual address, physical address, or data), starting the appropriate cache or bus cycle, and asserting the request acknowledge signal, MCTT REQ ACK L. When the data path sees the request acknowledge signal, it removes the 32 bits from the data bus. If the memory function is a read, the data path also disables the tri-state drivers to allow the data being read to be driven from the memory controller to the data path.

The microcode does not expect a response from the memory controller until the microcycle following

the next microcycle; the memory controller error status signals are in an undefined state until then. After this intervening microcycle, a Move or Moveout microinstruction to read or write the data may be executed, and a microbranch taken to test the status of the operation. (The data to be read or written are the data currently on the memory data bus; this is specified by MEMORY.DATA in the long operand of the Move or Moveout microinstruction.)

If the microcode does not execute the Move or Moveout microinstruction in the second cycle following the memory request, the data path assumes that final status is not required and terminates the memory function sequence.

Byte displacements are read from the instruction stream by enabling the IBYTE register onto the ID bus; the Memory Request microinstruction is not used. For this case, the microcode must always test whether the byte in the IBYTE register is valid, to insure that valid data has been read. The data in the IBYTE register is valid when the signal DAPR IB INVALID H is not asserted.

Stalls

Stalls are caused by one of three situations. If the microcode executes a Move or Moveout microinstruction following a Memory Request or an I-stream Request and REQ ACK has not been received from the memory controller, the data path hardware stalls the operation for one full cycle (250 ns) by not asserting the LD CSR signal to the data path chip and by delaying the clock edges to the data path control logic. At the end of this cycle, REQ ACK is tested again and the stall repeated if

REQ ACK is still not asserted. Thus, the DAP hardware stalls the execution of the microprogram by continuously repeating the Move or Moveout microinstruction until REQ ACK is asserted. Note that this type of stall only occurs when a microinstruction with MEMORY.DATA specified in the long operand is executed following a memory request microinstruction.

The second situation causing a stall occurs when the memory controller asserts the signal MCTN MEM BUSY H. If the memory controller is unable to deliver status or data in the cycle in which the information is expected, the memory controller asserts MEM BUSY. Upon receiving MEM BUSY, the DAP hardware causes a stall until MEM BUSY is negated.

A stall also occurs when a microinstruction selects one of the console UART registers. A stall condition is generated for a single cycle. This is because of the long write pulse and read time needed by the 2661 UART.

MD Bus Latches

The 32 signals on the data bus are latched into four 74F373 latches, collectively named the MD bus latch. From this latch, the signals are driven onto the memory data bus and then to the memory controller. Similarly, signals coming into the DAP module from the memory controller on the memory data bus are latched into four more 74F373 latches, collectively named the MD bus input latch. From the MD bus input latch, the signals are driven onto the data bus.

The MD bus latch and the MD bus input latch are needed because the memory controller uses a 125

ns cycle and may not be in the correct half of the 250 ns DAP cycle when data are being sent to the memory controller or received by the data path.

The signal DAPK OPEN LATCHES H controls the MD bus latch, opening it to capture the data that are to be driven from the data bus onto the memory data bus. The memory controller module controls the MD bus input latch with the signal MCTN MD BUS IN LE H, opening it to capture the data that are to be driven from the memory data bus onto the data bus.

Memory Function Latches

The memory function latches are part of the memory function control block shown in Figure 4-1. There are two 74F373 latches: the first one holds the current memory function code and the second holds the previous memory function code.

The bits saved in the first memory function latch are microinstruction bits DAPA CS <38:37> and <28:23> when a Memory Request microinstruction is decoded. Bits <38:37> actually come from the size register but still represent the data type. Bit <28> is the data flow bit, and <27:23> are the memory function code (see Figure 3-4).

These eight bits are saved in the first memory function latch until the memory controller is available. The latch is normally open and is closed when a memory request is started. When the memory controller is ready for the function code, the latched bits are driven from the first memory function latch onto the memory control bus as BUS MEM CTL <7:0>.

If the latch bit, bit <31>, of a Memory Request microinstruction is set when the microinstruction

is decoded, bits <38:37> and <28:23> are also saved in the second memory function latch. If a page crossing or memory management fault occurs when this microinstruction is executed, the microcode retries the microinstruction after it fixes the condition that caused the failure. The memory request information needed by the microcode to retry the microinstruction that failed is the information latched in the second memory function latch. Thus, when a Memory Request microinstruction is repeated, the contents of this second memory function latch are driven onto the memory control bus as BUS MEM CTL <7:0> instead of the contents of the first memory function latch. (The first memory function latch contains memory request information from the most recent Memory Request microinstruction that was executed as part of the microroutine invoked to fix the condition that caused the failure.)

Memory Function Control

When a Memory Request or I-stream Request microinstruction is decoded and executed, twelve bits of control information are sent to the memory controller from the data path module. These twelve bits inform the memory controller about the requested memory function.

Eight of the twelve bits are the microinstruction bits latched in the memory function latch and driven over the memory control bus as BUS MEM CTL <7:0>. These eight bits consist of the microinstruction data type field (bits <38:37>), the 5-bit memory function field (bits <27:23>), and the data flow bit (<28>).

The other four bits of control information are sent

to the memory controller over the backplane. They are: DAPT MEM REQ MODE <1:0>, DAPT MODIFY, and DAPT SECOND PART L.

The two MEM REQ MODE bits indicate the access mode, which is used for protection checking. If the access mode bit in the Memory Request microinstruction (bit <30>) is set, the value of MEM REQ MODE <1:0> is 0 to indicate kernel. If the access mode bit in the Memory Request microinstruction is clear, MEM REQ MODE <1:0> have the same value as the current mode bits (DAPR CUR MODE <1:0>) in the current mode register (PSL.MODE).

The signal DAPT MODIFY H is asserted when the modify intent is write; that is, bit <29> in the Memory Request microinstruction is a one.

The signal DAPT SECOND PART L is the second part flag. This signal is always part of the control information sent to the memory controller when there is a memory function request, but it is usually not asserted. When the signal is not asserted, it means that the first part of a memory request is to be executed; the majority of memory requests only have one part.

If a page crossing or memory management fault occurs when a microinstruction is executed, the microcode jumps to a subroutine to fix the condition that caused the failure. The microroutines that fix these conditions contain Memory Request microinstructions with REPEAT.FIRST or REPEAT.SECOND memory functions.

When a REPEAT.FIRST Memory Request is executed, the signal DAPR REPEAT L is asserted. When DAPR REPEAT L is not asserted, it means

that the current memory request is specified in the current microinstruction and no repeat is necessary. When DAPR REPEAT L is asserted, the previous memory function bits latched in the second memory function latch are driven onto the memory control bus as BUS MEM CTL <7:0>.

When a REPEAT.SECOND Memory Request is executed, DAPR REPEAT L is also asserted and with the same effect. But in addition, the second part flag is set; that is, the signal DAPT SECOND PART L is asserted.

When the memory controller receives these twelve signals, it reassembles them into a control word and uses this control word to access its own control store. The selected memory controller microcode routine then carries out the requested memory function.

PSL.MODE Register

PSL.MODE is the current mode register, address 106. The current mode bits of the processor status longword (PSL bits <25:24>) are stored here. The current mode register is used to inform the memory controller of the access mode of the current memory request.

PSL.MODE can be read or written. The register is written when an REI (return from exception or interrupt), a CHM (change mode), or an LDPCTX (load process context) macroinstruction is executed. The PSL.MODE register is also written for interrupts and some exceptions. The new current mode is computed in the data path chip and written to the PSL.MODE register from the low two bits of the internal data bus, BUS ID <01:00>.

The output from PSL.MODE is DAPR CUR MODE <1:0>; these signals are the input to a PAL whose output is DAPT MEM REQ MODE <1:0>. Thus, the value of DAPR CUR MODE <1:0> is the value of DAPT MEM REQ MODE <1:0>. (This is true unless the microinstruction is a repeated memory request; in this case, the value of DAPT MEM REQ MODE <1:0> is the access mode that was saved in the second memory function latch.) The MEM REQ MODE signals tell the memory controller what access mode to use for the protection check on the requested memory function.

Sign-Extenders

If an I-stream Request microinstruction is executed and the long operand specifies IB.WORD, a word of data is read from the instruction stream and returned to the data path module over the memory data bus. After the word of data is latched in the MD bus input latch, it is driven onto the data bus. Here, it is extended to 32 bits by the sign-extension logic, and delivered to the data path chip.

For more information about sign-extension, see the paragraphs titled "Sign-Extension" in the "Data Transfers" section of this chapter.

Memory Request Timing

Figure 4-11 shows the timing of a read from memory, and Figure 4-12 shows the timing of a write to memory. Both diagrams assume a cache hit. The signal DAPK OPEN LATCHES is asserted to open the MD bus latch. This latch is opened once for a read, to capture the contents of

the location specified by the long operand before those contents are driven onto the memory data bus. (The contents are usually a virtual address, but can also be a physical address or the actual data.)

DAPK OPEN LATCHES is asserted twice for a write to memory; first to capture the virtual address (or physical address or data) to be driven onto the memory data bus, and second to capture the data to be written to memory before the data are driven onto the memory data bus. The data to be written appear at the output pins of the data path chip 80 ns into the EXECUTE cycle of the Moveout microinstruction.

Table 4-10 summarizes the DAP/MCT interface signals, lists the time during which the signal is asserted, and briefly describes the function of the signal.

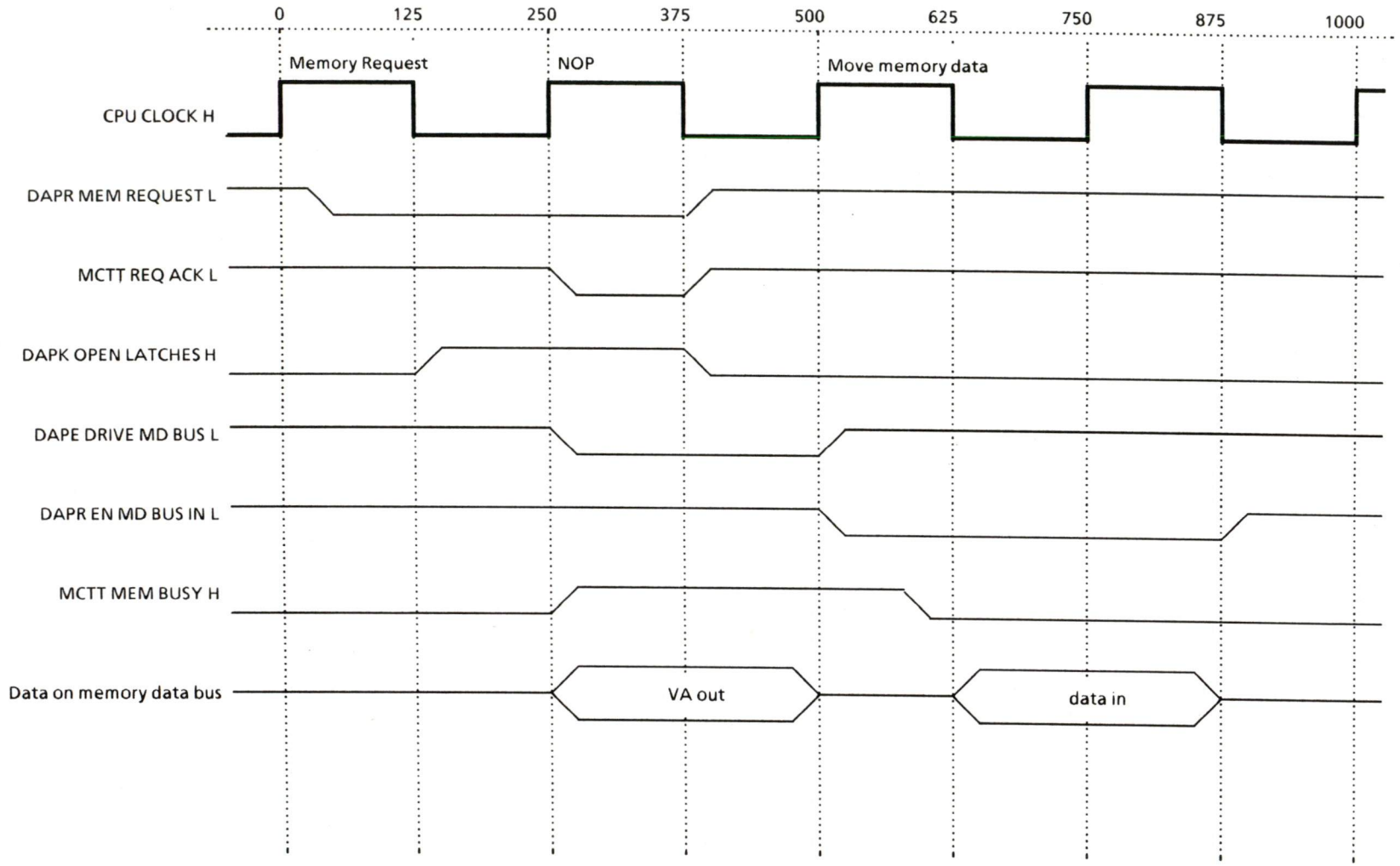


Figure 4-11. Timing of a Read from Memory

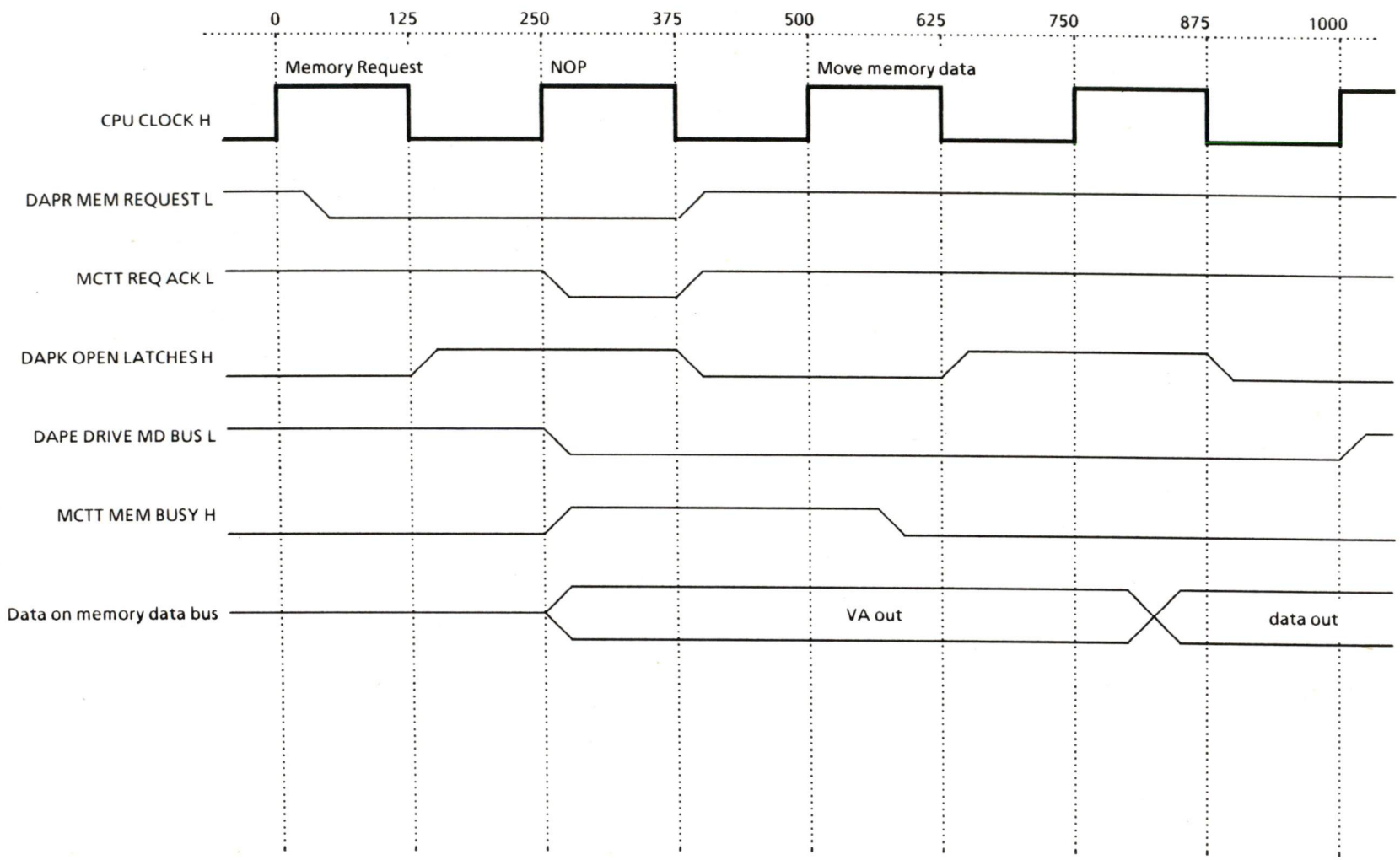


Figure 4-12. Timing of a Write to Memory

Table 4-10. DAP/MCT Interface Signals and Timing

Signal Name	Timing (at DAP pins)	Function
DAPL MCT INIT L		Initialize system to a known state; asserted asynchronously, negated 12 ns following low–high edge of 16 Mhz clock
DAPL MCT 250 L	T0 + (0–8) to T0	Memory controller copy of the CPU clock
MCTM BASE CLK H		Basic clock source used on DAP module
MCTM DPC SRC L		Inverted version of clock needed for data path chip
BUS MEM DATA <31:00>	T0 + 342 ns to 500 T0 + 730 ns to 790	Data or virtual address from data path to memory controller Data from memory controller to data path
BUS MEM CNTL <07:00>	T0 + 83 ns to 130 T0 + 220 ns to 260	Memory function code from DAP to MCT Instruction stream byte from MCT to DAP
DAPR MEM REQUEST L	T0 + 79 ns to 750	Informs memory controller of new function code on control bus
DAPT MEM REQ MODE <1:0>	T0 + 90 ns to 750	Access mode used for protection check; encoding as defined in PSL
DAPT MODIFY H	T0 + 90 ns to 750	A write will be attempted to the current address.
DAPT SECOND PART H	T0 + 90 ns to 750	The first part of this request has already been attempted. This signal is asserted when a Memory Request microinstruction with the REPEAT.SECOND function is executed. The signal is deasserted when a Memory Request microinstruction with the latch bit (<31>) set is executed.
MCTT REQ ACK L	T0 + 345 ns to 395	Request acknowledged; the MCT has accepted the command and the virtual address.
MCTT MEM ERROR H	T0 + 76 ns to 150	A memory error has occurred.
MCTN MEM BUSY H	T0 + 100 ns to 150	The memory controller is busy.
MCTN MD BUS IN LE H		MD bus input latch control for incoming data; needed to keep the data stable across a 250 ns edge for the data path chip. This signal is asserted as long as the MCT memory data bus transceivers are directed out.
MCTT TB MISS H	T0 + 76 ns to 250	Translation buffer miss
MCTT MOD REF H	T0 + 76 ns to 250	Modify request refused
MCTP NXT IB VALID H	T0 + 76 ns to 250	IBYTE valid; the IBYTE register may be loaded when this signal is active.
MCTN PAGE CROSS H	T0 + 76 ns to 250	A memory reference across a page boundary has occurred.
MCTB WRT TMO L	T0 + 200 ns to 275	A Q22 bus timeout occurred on the last write operation; valid for a single cycle
MCTT IB ERROR H	T0 + 76 ns to 250	MCT cannot supply the next byte from the I-stream due to an error or a page crossing
MCTN DATA 15 H	T0 + 200 ns to 250	Data bit 15; used for sign extending
DAPR IB TAKEN L	T0 + 70 ns to 125	The microsequencer has used the current contents of the IBYTE register; asserted during decode microinstructions, I-stream requests, IB refills and microinstructions specifying IB.BYTE as the source.