VAX-11/730
Central Processing Unit
Technical Description

digital

VAX-11/730
Central Processing Unit
Technical Description

This document was set on DIGITAL's DECset-8000 computerized typesetting system.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECSYSTEM-20	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	RSTS
UNIBUS	VAX	RSX
	VMS	IAS

CONTENTS

CHAPTER 6   DATA PATH

CHAPTER 7   INTERRUPT PROCESSING HARDWARE

APPENDIX A   PROGRAMMED ARRAY LOGIC DEVICES (PALS)

APPENDIX B   FLOW DIAGRAM SYMBOLS

FIGURES

TABLES

viii

CHAPTER 1
INTRODUCTION AND SYSTEM OVERVIEW

1.1 MANUAL SCOPE AND RELATED DOCUMENTS

This technical description is intended for use as a field reference for DIGITAL Field Service personnel and as a resource for training programs conducted by Educational Services and Manufacturing.

Chapter 1 is a general description of the VAX-11/730 system. The remaining chapters provide a detailed technical description of the VAX-11/730 Central Processing Unit (CPU). All CPU components with the exception of the memory (and UNIBUS) control logic are described. A description of the memory (and UNIBUS) control logic is included in the technical description for the VAX-11/730 Memory System. This and other related documents detailing VAX-11 system architecture and the various system components are listed in Table 1-1.

Table 1-1 Related VAX-11/730 Documents

Title	Document Number
VAX-11/730 Memory System Technical Description	EK–MS730–TD
FP730 Floating-Point Accelerator Technical Description	EK–FP730–TD
Integrated Disk Controller Technical Description	EK–RB730–TD
H7202B Power System Technical Description	EK–PS730–TD
DMF32 Multi-function Communications Interface Technical Description	EK–DMF32–TD
VAX-11/730 System Installation Guide	EK–SI730–IN
VAX-11/730 Hardware User's Guide	EK–11730–UG
VAX-11/730 Systems Maintenance Guide	EK–11730–MG
VAX-11/730 Diagnostic System Overview Manual	EK–DS730–UG
VAX Hardware Handbook	EB–17281
VAX-11 Architecture Handbook	EB–17580
Micro 2 User's Guide	AA–H531A–TE

1.2 INTRODUCTION TO THE VAX-11/730

The VAX-11/730 is the current low-end member of the VAX-11 family of 32-bit computer systems. The synchronous microprogrammed CPU executes the VAX-11 instruction set (in native mode) and supports the VAX/VMS operating system. Non-privileged PDP-11 instructions may also be executed (in compatibility mode), allowing existing user mode PDP-11 programs to be run without modification. System features include:

- A main memory using standard array modules that allow expansion in 1.0 MB increments to a maximum capacity of 5.0 MB

- Virtual memory management employing a hardware translation buffer that minimizes memory references for virtual to physical address conversion

- An instruction buffer that allows for the fetching of the next instruction while the current instruction is executing

- Sixteen 32-bit general registers

- Thirty-two interrupt priority levels

- Optional floating-point accelerator and implementation of all floating data types including GRAND and HUGE

- Interval timer and time of year clock

- A microprocessor-controlled console subsystem that (optionally) supports remote diagnosis of the system from a DIGITAL diagnostic center

- A UNIBUS integral to the system that allows the connection of the general purpose UNIBUS peripherals supplied by DIGITAL as well as UNIBUS-compatible devices supplied by the customer.

1.3 VAX-11/730 SYSTEM CONFIGURATION

A VAX-11/730 system block diagram is shown in Figure 1-1. The KA730 CPU consists of three standard HEX modules; the data path (DAP) module, the writable control store (WCS) module, and the memory controller (MCT) module. Included in the CPU is an 8085A console processor that contains three full-duplex asynchronous line interfaces for connecting an LA120 console terminal, a TU58 (dual) cassette tape unit, and a modem to the system. The modem (and supporting hardware and software) is an option that allows for the remote diagnosis of the system by a DIGITAL diagnostic center. The modem port is also used for APT (automated product test) during system manufacture.

Other possible VAX-11/730 system components each consisting of a standard HEX module, include an FP730 Floating-Point Accelerator (FPA), an RB730 Integrated Disk Controller (IDC), and a DMF32 synchronous/asynchronous serial line controller. The DMF32 also contains a parallel interface that may be operated as either a line printer control (an enhanced LP11 control) or as a general purpose interface similar to the DR11-C.

The FPA and IDC are connected to the CPU by the FPA/Port bus. This is a VAX-11/730 reserved bus that allows CPU microcode control of both options. The DMF32 connects to the CPU via the UNIBUS. The UNIBUS is the VAX-11/730 system's peripheral I/O bus.
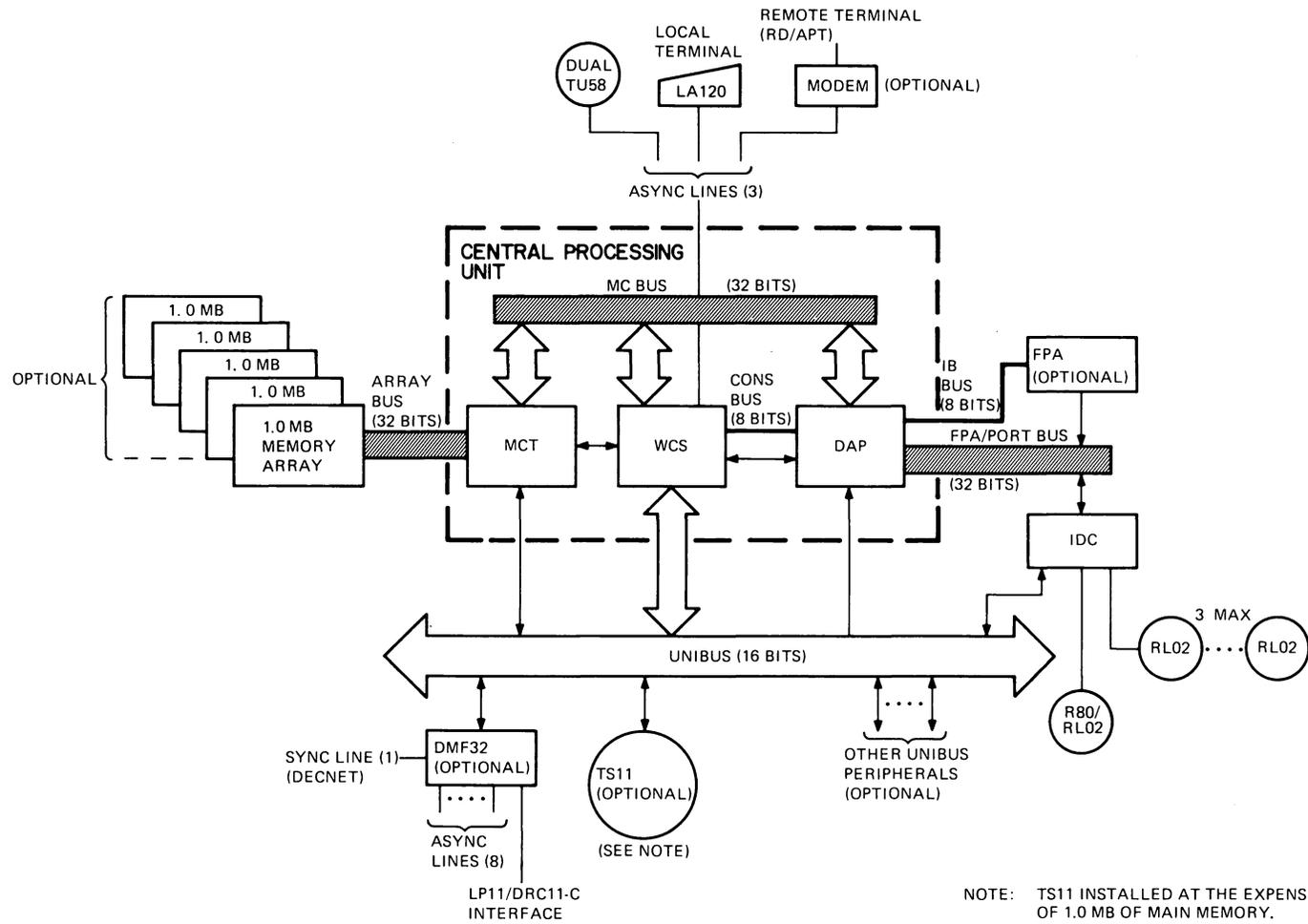
Figure 1-1 VAX-11/730 System

VAX-11/730 peripherals, other than the DMF32 and its associated I/O devices, include the TS11 magnetic tape drive, LP32 line printer and other VAX-11 supported UNIBUS options. An R80 (or RL02) disk drive and up to three RL02 disk drives may be connected to the system through the IDC. Disk data transfers by the IDC are over the FPA/port bus, not the UNIBUS.

The main memory in the VAX-11/730 consists of from one to five MOS memory array modules connected to the CPU by the array bus. Access to the array modules (and UNIBUS device registers) is controlled by the memory (and UNIBUS) control logic in the CPU. Up to five 1.0 MB array modules may be installed to give a maximum memory capacity of 5.0 MB. The minimum memory configuration, which is 1.0 MB, consists of one array module. Whenever a TS11 is connected to the system, memory capacity is reduced by 1.0 MB. This is because the TS11 UNIBUS interface module is installed in a module slot otherwise reserved for an array module.

The module designs for the CPU and the FPA, IDC, and DMF32 options all make extensive use of semi-custom designed programmed array logic (PAL) chips to increase logic density and reduce cost. (PALs are described in Appendix A.) The high logic density allows most system configurations to be contained in a single H9642 cabinet. An H9642-DH expander cabinet is required only when the installation of more than two disk drives or of optional UNIBUS peripherals requires expansion out of the basic cabinet.

1.3.1 KA730 Central Processing Unit (CPU)

The hardware for the CPU is contained on the MCT (M8391) module, the WCS (M8394) module, and the DAP (M8390) module. The major components are the 8085A console processor, the CPU data path and associated microcontroller (CPU control store and CPU microsequencer), and the memory (and UNIBUS) control logic. A basic block diagram is shown in Figure 1-2.

The console processor is contained mostly on the WCS module. Its main logic element is an 8-bit 8085A microprocessor supported by 16KB of RAM and 4KB or 6KB of ROM. (The basic ROM is 4KB but an additional 2KB is installed as part of the remote diagnosis option.) The console processor is the main operator interface to the system, acting in response to switch panel control and a console command language entered via the local terminal (the LA120) or the remote terminal. It also interfaces to the system a mass storage device (the TU58) that is used mainly for bootstrapping and diagnostic purposes. The console processor is controlled by a console program executed by the 8085A. Part of the console program is resident (in ROM) and the rest is loaded (into the RAM) from the TU58 during the system bootstrap operation.

The CPU data path, which is contained on the DAP module, performs the arithmetic and logical operations necessary to execute the instruction set. The principal data path components are 2901A 4-bit processor slices. Eight 2901As are connected in parallel to give an arithmetic and logical processing element 32-bits wide. The data path also contains a 256 location × 32-bit local store (a RAM) that contains among other things the general registers and several of the architecturally defined privileged processor registers.

The data path and other hardware on the DAP module used for instruction processing purposes (e.g., instruction and interrupt processing hardware) are controlled by microcode executing in the CPU's microcontroller. The microcontroller consists of a microsequencer on the DAP module and a soft (writable) control store on the WCS module having a basic storage capacity of 16K 24-bit microwords. An additional 4K of control store can be installed as an option to support the IDC and to provide user expansion space. The CPU microcode is loaded into the control store from the TU58 during the system bootstrap operation.

MC BUS

ASYNC LINES

DATA CTL

SWITCHES/
INDICATORS

DATA CTL

DATA CTL

MCT M8391

MC BUS

VAR

DATA
ROTATOR

VIRT ADRS BUS

ECC

TB

PAB

PHYS ADRS BUS

CTL

MICRO—
SEQUENCER/
CONTROL
STORE

ARB

ARRAY
MODULES

DATA

ARRAY BUS

ADRS

WCS M8394

MC
BUS

8085A
CONSOLE
PROCESSOR

CONS
BUS

CLK
GEN

UNIBUS DATA
XCVRS/
LATCHES

CPU
CONTROL
STORE

UNIBUS

TO
FPA/IDC

CLK

CTL

CLKS

CTL

MICRO ADRS

MICROWORD

CONS

DAP M8390

MC
BUS

INSTRUCTION
PROCESSING
HARDWARE

IB
BUS

IB

FPA

NAD BUS

CONS
BUS

CONS
REGS

DATA
PATH

DATA

CTL

FPA/
IDC

FPA/PORT BUS

CONS INTR

PORT INTR

CPU
MICRO—
SEQUENCER

INTERRUPT
PROCESSING
HARDWARE

CTL

CTL ADRS

DATA

CTL

BR'S

UNIBUS

UNIBUS PERIPHERALS

TK-6646

Figure 1-2   KA730 Block Diagram

The memory (and UNIBUS) control logic controls the data transfers to and from the main memory array modules over the array bus, and the transfers to and from the peripheral devices over the UNIBUS. Transfers are initiated by the CPU data path under the control of the CPU microcode, or by the UNIBUS devices when direct (NPR) data transfers are made between the UNIBUS devices and main memory.

The memory (and UNIBUS) control logic contains a translation buffer for virtual to physical address conversion, a UNIBUS arbitrator, a data rotating and substituting network for aligning memory data, and its own microsequencer and control store. These, and the other major components (except for the UNIBUS data transceivers and latches which are on the WCS module) are contained on the MCT module. Communications between the MCT module and the WCS and DAP modules during memory and UNIBUS device references are over the processor's internal memory control (MC) bus.

1.3.2 Main Memory Array

Main memory consists of one to five memory array modules that use 64K MOS (metal oxide semiconductor) RAM chips for data storage. An array module stores 1.0 MB of data. This allows VAX-11/730 memory capacities ranging from 1.0 MB to 5.0 MB.

The array modules are connected to the CPU by the array bus. Memory data transfers over the array bus are 39 bits; that is, one 32-bit longword (four bytes) of data and seven associated ECC (error correction and checking) bits are transferred at a time. The ECC bits provide for the detection and correction of all single-bit errors when a longword is read from the memory array. Double-bit errors are detected but not corrected. The error detection and correction circuitry is part of the memory (and UNIBUS) control logic on the MCT module.

1.3.3 FP730 Floating-Point Accelerator (FPA)

The optional FPA (M8389) module is an independent processor that works in parallel with the CPU to speed the execution of floating-point instructions. The principal processing elements used by the FPA (as by the CPU) are the 2901A 4-bit processor slices. A basic block diagram of the FPA is shown in Figure 1-3.
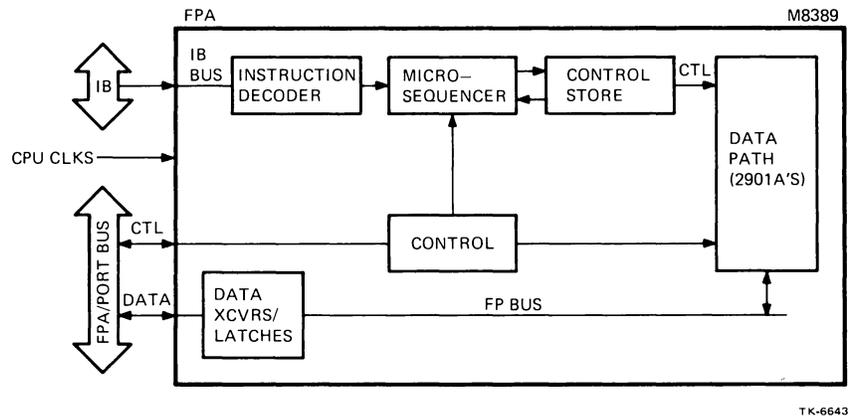
Figure 1-3 Floating-Point Accelerator

When an FPA is installed in a system, the standard floating-point microcode is not executed in the CPU. Instead, for certain instructions, the CPU sends operands to the FPA over the FPA/Port bus, and the FPA then performs the floating-point arithmetic at a high speed using its own dedicated hardware and microcode. When calculations are complete, the results are sent back to the CPU, again over the FPA/port bus. The CPU's IB bus also connects to the FPA to supply opcode information.

1.3.4 RB730 Integrated Disk Controller (IDC)

The optional IDC (M8388) module interfaces an R80/RL02 disk drive and from one to three RL02 disk drives to the system. The R80, which has a non-removable disk with a storage capacity of 124 MB, has an average seek time of 25 ms, an average latency of 8.33 ms, and a peak data transfer rate of 1.2 MB per second. An RL02, which has one removable cartridge with a capacity of 10 MB, has an average seek time of 55 ms, an average latency of 12.5 ms, and a peak data transfer rate of 0.5 MB per second.

Data transfer between the IDC and CPU are over the FPA/port bus and controlled in part by dedicated microcode in the CPU. One 32-bit longword (four bytes) of disk read/write data is transferred at a time, following the generation of a micro level (fast) processor interrupt request by the IDC. Data silos (FIFOs) in the IDC provide up to one KB of data buffering for both read and write data.

A basic block diagram for the IDC is shown in Figure 1-4. As indicated, the IDC connects to the UNIBUS in addition to the FPA/port bus. This is for generating interrupts other than the fast interrupts generated for disk data transfers. (The IDC asserts a UNIBUS BR line.) The IDC's UNIBUS connection also provides for monitoring the system's power fail levels.
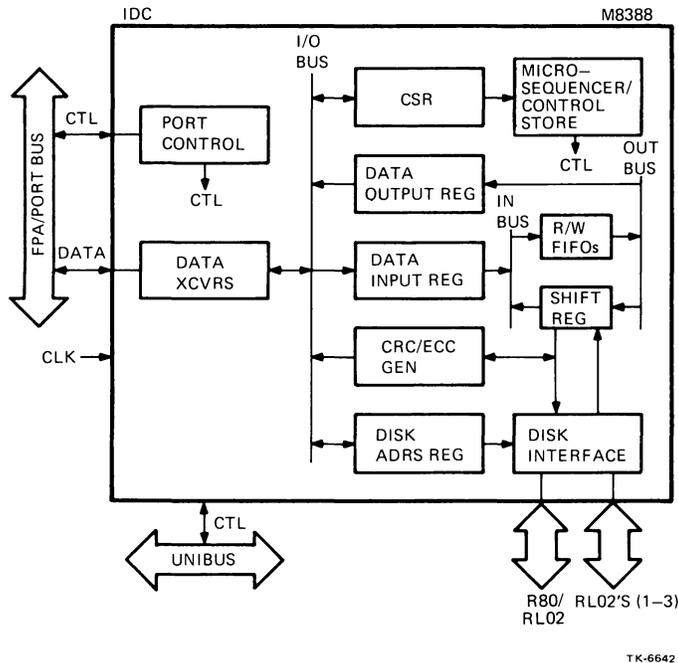
Figure 1-4   Integrated Disk Controller

1.3.5 DMF32

The DMF32 (M8396) module, a UNIBUS option, controls eight serial asynchronous lines, one serial synchronous line, and a parallel port that may be operated as either an enhanced LP11 line printer control or as a DRC11-C general purpose interface. The main data processing element is the 2901A 4-bit processor slice which is also used on the CPU and FPA modules. A basic block diagram is shown in Figure 1-5.
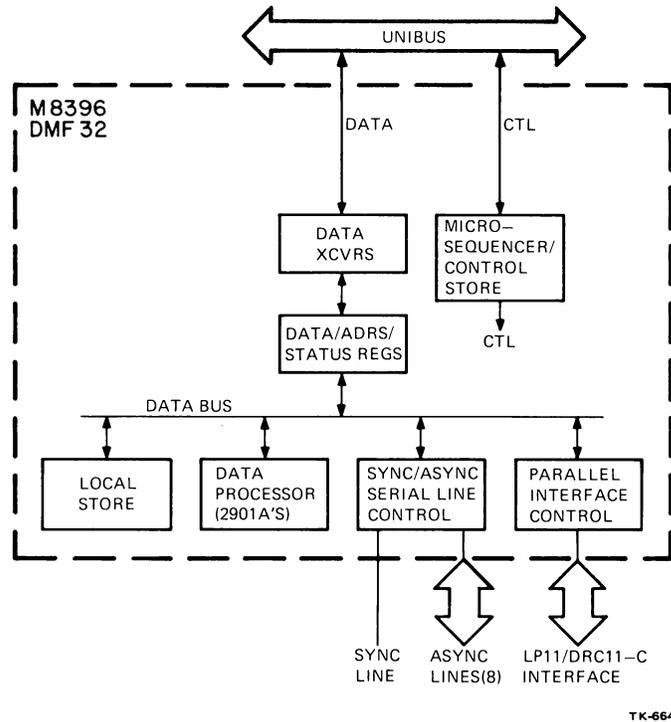
Figure 1-5   DMF32

The asynchronous line control portion of the DMF32 operates as an enhanced version of the DZ11-A. Enhancements include split baud rate and extended modem control for two channels, plus transmit data silos (32 characters) for all channels.

The DMF32's synchronous line control transfers message data to and from main memory by means of NPR data transfers. It also provides low-level support (e.g., framing messages, generating and checking CRC) for DDCMP, SDLC, HDLC, and BISYNC protocols. All high-level network support functions are performed by the CPU.

The DMF32's parallel port, in line printer operating mode, acts as an enhanced LP11 control that allows LP11-compatible line printers (such as LP25 and LP26) to be connected to the system. This enhanced LP11 control does (optionally) several formatting functions previously done by software. In its parallel operating mode, the parallel port is used as a DRC11-C general purpose interface that allows user peripheral devices to be connected to the system.

1-8

1.4 SYSTEM ARCHITECTURE

VAX-11/730 system architecture (including data types, instruction set, addressing, processor registers, interrupts and exceptions) is discussed in detail in the *VAX-11 Architecture Handbook* and the *VAX Hardware Handbook*. The information will not be duplicated in this technical description.

1.5 SWITCHES AND INDICATORS

The VAX-11/730 has two front panel switches and four front panel indicators. One of the switches is a six-position key switch for powering up the system and for setting the mode of operation for the system's local and remote terminals. The other is a three-position toggle switch for bootstrapping the system and controlling automatic restarts.

One of the four indicators (LEDs) indicates normal power on. Two other indicators are controlled by the 8085A console program. One shows the system's run state and the other shows the status of the remote diagnostic link. A fourth indicator, marked BATT, is reserved for future use. Switch and indicator functions are discussed in detail in the *VAX-11/730 Hardware User's Guide*.

1.6 CONSOLE COMMANDS/BASIC OPERATOR CONTROL

The console program running in the 8085A console processor is normally operating in one of two modes; that is, either in console mode or program mode. In console mode, commands from the local terminal (the LA120) and/or the remote terminal are directed to the 8085A console processor. By means of the VAX-11 console command language, an operator may use the console processor to perform a number of console functions such as resetting and bootstrapping the system, depositing and examining memory, and halting and starting program execution. In program mode, terminal input and output data is generally passed directly to or from the CPU, character by character. The data is handled by the VAX-11/730 VMS level software. Refer to the *VAX-11/730 Diagnostic System Overview Manual* for a detailed description of the various console commands and the basic operator control functions.

1.7 DIAGNOSTIC AND MAINTENANCE AIDS

Diagnostic and maintenance aids in the VAX-11/730 are listed in Table 1-2. They are discussed where applicable in this technical description and in the other manuals in the VAX-11/730 document set.

Table 1-2 VAX-11/730 Diagnostic and Maintenance Aids

System Component(s)	Diagnostic and Maintenance Aid(s)
CPU/Memory	Control store has parity and microsync test point.
	Control store register (CSR) and micro-PC may be written and read by microdiagnostics.
	Basic clocks may be single-stepped by microdiagnostics.
	ECC bits provide single-bit error correction and double-bit error detection. Error address register and syndrome bit control are implemented.
	Translation buffer has parity. Virtual address and translated physical address are readable. UNIBUS address and data loopback capability is provided.

Table 1-2 VAX-11/730 Diagnostic and Maintenance Aids (Cont)

System Component(s)	Diagnostic and Maintenance Aid(s)
Console processor	Independent 8085A console processor allows testing of other system components by console-based microdiagnostics.
	Remote diagnostics option provides automated checkout capability.
	ROM-resident self-test is invoked (optionally) at power-up and on command.
	Voltage monitoring circuits are incorporated to check +15 V and +5 V. UNIBUS AC LO and DC LO are also monitored.
FPA	Control store has parity.
	Micro-PC is writable and readable.
IDC	Diskless data loop is provided (RL02 port only).

1.8 PHYSICAL DESCRIPTION
The major physical components in the VAX-11/730 basic (H9642) cabinet are an 874 Power Control, an H7202B Low-End Modular (LEM) Power Supply, a BA11-Z Mounting Box, and the dual-drive TU58. The basic cabinet can also accommodate two of the rack-mounted disk drives supported by the IDC; that is, either two RL02s or one RL02 and an R80. The major components in the (H9642-DH) expander cabinet are an 874 Power Control and a BA11-A Mounting Box. There is also space for one RL02. Another cabinet is required if a fourth disk drive (an RL02) is installed in the system. A detailed physical description of the VAX-11/730 is included in the *VAX-11/730 System Installation Guide*.

1.9 SYSTEM TIMING
The system's basic clocks are generated on the CPU's WCS module. The clocks sequence the basic machine (the CPU's microcontroller and data path) on the WCS and DAP modules, and control the generation of clocks on several other modules (such as MCT and FPA) so the system's operations are synchronized.

The basic clock is a continuous clock train with a 90 ns period. Three other clocks are also generated, each 90 ns out of phase with the one before. The first two clock phases (P0 and P1) are free-running like the basic clock. The third clock phase (P2) is gated to produce and define the CPU microcycle. Timing of this basic machine cycle is the time required to execute a single CPU microinstruction. Timing for the system's basic clocks is shown in Figure 1-6.
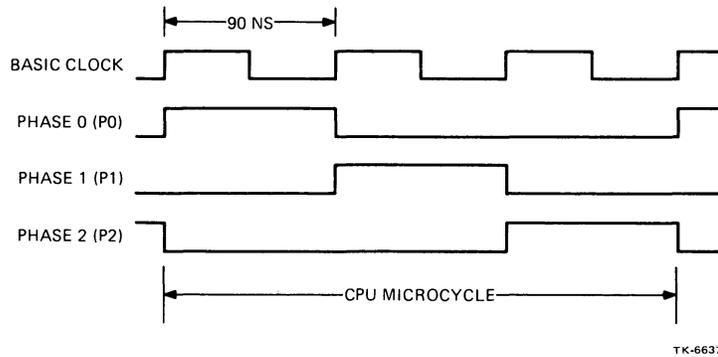
Figure 1-6   Basic System Clocks

1.10   SYSTEM BUS SUMMARY

The system buses which interconnect the modules in the VAX-11/730 system are the UNIBUS, the memory control (MC) bus, the memory array bus, the FPA/port bus, the console bus, and the IB bus. (Those buses that are completely contained within a module are not discussed in this section.) Figure 1-7 shows data flow over the system buses during the major data transfer operations.
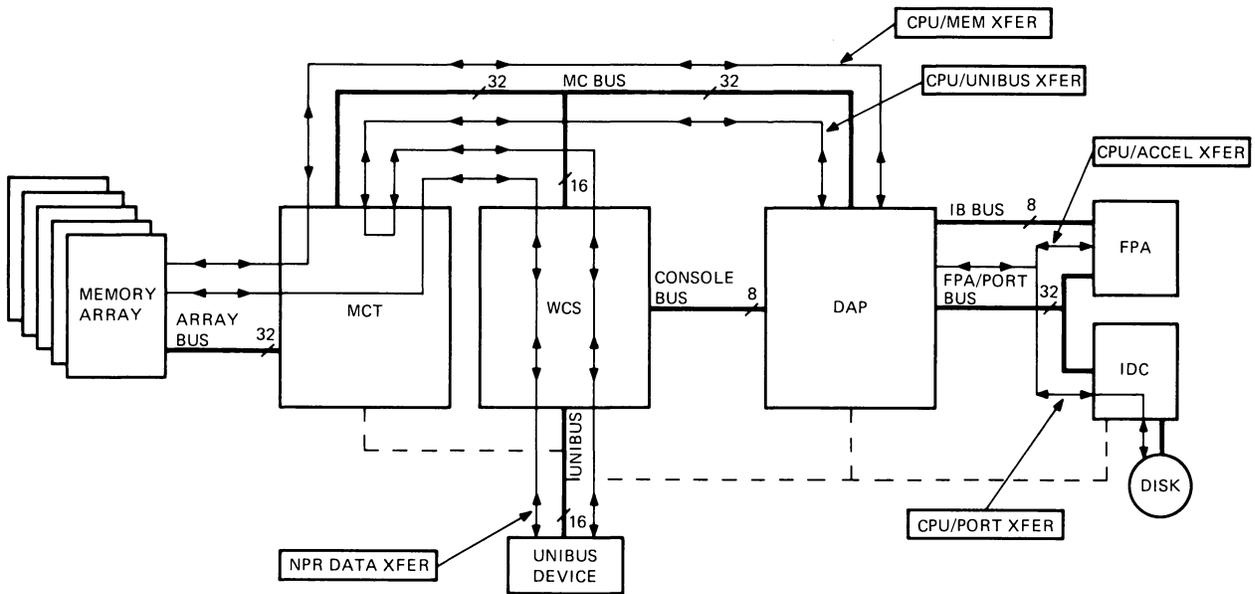
Figure 1-7   Major Bus Data Transfers, Data Flow

1.10.1 UNIBUS

The UNIBUS connects the CPU to the system's peripheral I/O devices. Most of the CPU's UNIBUS control logic, including the bus arbitrator, is on the MCT module. However, the transceivers for the 16 UNIBUS data lines are located on the WCS module, requiring that data be transferred over the MC bus in addition to the UNIBUS during a UNIBUS operation. (The MC bus is discussed in Paragraph 1.10.2.)

Figure 1-8 shows the UNIBUS lines and their connection to each of the CPU modules and the other VAX-11/730 components. Bus signals are described in Table 1-3. A detailed description of UNIBUS operation is given in the *VAX-11/730 Memory System Technical Description*.
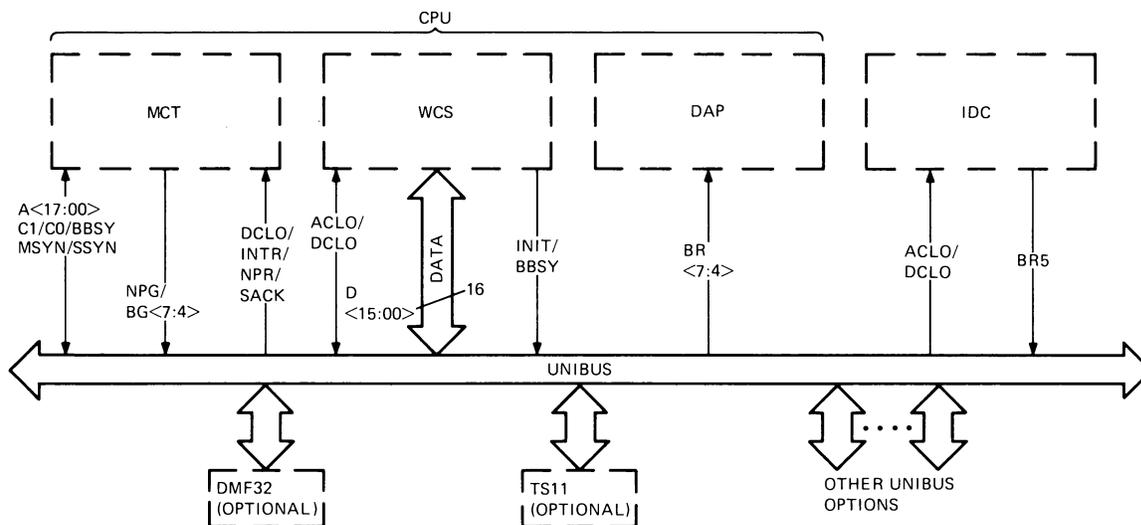
Figure 1-8 VAX-11/730 UNIBUS

Table 1-3 VAX-11/730 UNIBUS Signal Summary

Signal(s)	Assertion Level	Description
A⟨17:00⟩	L	Address lines (18). Select UNIBUS device register when asserted by CPU (MCT module). Select memory address (via UNIBUS map registers in MCT module's translation buffer) when asserted by UNIBUS device during an NPR transfer.
D⟨15:00⟩	L	Data lines (16). Transfer data between UNIBUS device and CPU. UNIBUS transceivers are on WCS module. Data is transferred between WCS and MCT modules over MC bus.

1-12

Table 1-3 VAX-11/730 UNIBUS Signal Summary (Cont)

Signal(s)	Assertion Description Level	
C1/C0	L	Control lines (2). Asserted with address lines to specify type of data transfer.

	C1	C0	Function
	0	0	DATI
	0	1	DATIP
	1	0	DATO
	1	1	DATOB

Signal(s)	Assertion Description	Level
MSYN	L	Master synchronize. Asserted by bus master to initiate a data transfer.
SSYN	L	Slave synchronize. Asserted by slave in response to MSYN.
INTR	L	Interrupt. Asserted by UNIBUS device to initiate an interrupt vector transfer to CPU.
BR⟨7:4⟩	L	Bus request (4). Asserted by UNIBUS device to request use of bus for interrupt vector transfer.
BG⟨7:4⟩	H	Bus grants (4). Asserted by bus arbitrator in CPU (MCT module) to grant use of bus for interrupt vector transfer.
NPR	L	Non-processor request. Asserted by UNIBUS device to request use of bus for data transfer.
NPG	H	Non-processor grant. Asserted by bus arbitrator in CPU (MCT module) to grant use of bus for data transfer.
SACK	L	Selection acknowledge. Asserted by UNIBUS device to acknowledge bus grant (BG or NPG).
BBSY	L	Bus busy. Asserted by bus master when it assumes control of bus for data (or interrupt vector) transfer.
INIT	L	System reset. Asserted by CPU (console processor on WCS module) to initialize UNIBUS devices.
ACLO	L	Indicates loss of ac power.
DCLO	L	Indicates loss of dc power.

1.10.2 Memory Control (MC) Bus

The memory control (MC) bus interconnects the three CPU modules: the MCT, WCS, and DAP modules. It is used to transfer address and data information during memory and UNIBUS references by the CPU. It is also part of the data path during NPR data transfers between a UNIBUS device and the memory array. This is because the UNIBUS data transceivers are on the WCS module and not on the MCT module which connects to the array. As a result, the MC bus is used to transfer the NPR data between the two modules. The MC bus signals connecting to each of the CPU modules are shown in Figure 1-9. Bus signal descriptions are summarized in Table 1-4.
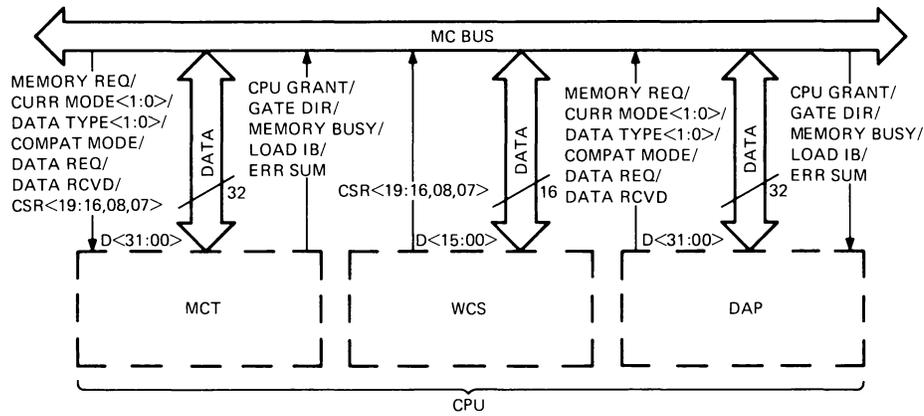
Figure 1-9   VAX-11/730 Memory Control (MC) Bus

Table 1-4   VAX-11/730 Memory Control (MC) Bus Signal Summary

Signal(s)	Assertion Level	Description
D⟨31:00⟩	H	Data lines (32). Transfer address and data information between MCT and DAP modules during CPU memory and UNIBUS references. Low-order 16 lines also transfer UNIBUS data between the MCT module and the UNIBUS transceivers on the WCS module.
MEMORY REQ	H	Memory request. Asserted by DAP module to initiate a CPU memory or UNIBUS reference.
CURR MODE⟨1:0⟩	H	Current mode lines (2). Asserted by DAP module to specify processor's current access mode.

CURR MODE		
1	**0**	**PROCESSOR ACCESS MODE**
0	0	Kernel
0	1	Executive
1	0	Supervisor
1	1	User

**Table 1-4   VAX-11/730 Memory Control (MC) Bus
Signal Summary (Cont)**

Signal(s)	Assertion Level	Description
DATA TYPE⟨1:0⟩	H	Data type lines (2). Asserted by DAP module to specify type of data transfer requested.

DATA TYPE 1	0	DATA MODE
0	0	Byte
0	1	Word
1	0	Not used
1	1	Longword

Signal(s)	Assertion Level	Description
COMPAT MODE	H	Compatibility mode. Asserted by DAP module to indicate processor is executing a PDP-11 program.
CSR⟨19:16, 08, 07⟩	H	The six memory function lines specify type of CPU reference. Signals come from control store register on WCS module.
CPU GRANT	L	Asserted by MCT module to grant CPU memory reference request.
MEMORY BUSY	H	Asserted by MCT module to indicate memory control logic is busy performing a CPU memory or UNIBUS reference.
GATE DIR	H	Gate direction. Negated by MCT module during execution of a CPU memory or UNIBUS reference that sends data to the DAP module (e.g., memory read). Asserted at all other times. Used to condition MC bus transceivers on DAP module.
DATA REQ	H	Data request. Asserted by DAP module to request transfer of read/write data following a CPU memory or UNIBUS reference request.
LOAD IB	H	Asserted by MCT module to load prefetched instruction data from data lines into the instruction buffer (the PFR) on DAP module.
DATA RCVD	H	Data received. Asserted by DAP module to signal end of read/write data transfer.
ERR SUM	L	Error summary. Asserted by MCT module to indicate one or more error conditions detected.

The MC bus has 32 data lines. During CPU memory and UNIBUS references, each of which consists of two seperate MC bus cycles, the data lines transfer both address and data information.

During the first MC bus cycle of a memory or UNIBUS reference by the CPU, the DAP module asserts the MEMORY REQ line and transmits the address information on the data lines. This request (or address-out) cycle is followed by a data cycle. For all references except a prefetch of instruction data, the data cycle is initiated by the DAP module when it asserts the DATA REQ line on the bus. The read/write data is then transferred over the data lines to end the MC bus operation. A detailed description for this type of CPU reference over the MC bus is given in Paragraph 6.11.

For a prefetch of instruction data, the MCT module (not the DAP module) initiates the data cycle by transmitting the instruction data on the data lines and asserting the LOAD IB line. LOAD IB automatically loads the instruction data into the DAP module's instructions buffer. No bus signals need be asserted by the DAP module to transfer the data. Bus operation during the instruction data prefetch operation is detailed in Paragraph 5.2.1.2.

During memory references by the CPU, the only data flow over the MC bus data lines is between the DAP and MCT modules. (The MCT module transfers data to and from the memory array over the array bus, which is discussed in Paragraph 1.10.3.) However, during UNIBUS references by the CPU, additional data transfers take place over the MC bus between the WCS and MCT modules. That is, during a UNIBUS read reference and following the MC bus request cycle, incoming UNIBUS data is transferred from the UNIBUS transceivers on the WCS module to the MCT module's data rotator. This is so the UNIBUS read data may be repositioned (if necessary) on the MC bus data lines before it is transferred to the DAP module during the MC bus data cycle that follows.

During a UNIBUS write reference, data may also need to be repositioned, and (similar to a memory write) it is first transferred from the DAP module to the MCT module during the MC bus data cycle. The data is then sent from the MCT module to the UNIBUS transceivers on the WCS module for transfer to the device.

1.10.3 Memory Array Bus
The memory array bus connects the MCT module to the memory array modules. The bus signals are shown in Figure 1-10. Their functions are given in Table 1-5.

Fifteen array bus lines are used to select the referenced 39-bit (32-bit longword plus 7-bit ECC) memory location in the array. Depending on the physical memory address, the MCT module asserts one of five lines to select one of the five array modules, and two bank select lines to select a 64K-location bank within the selected module. It also asserts a row address followed by a column address on eight multiplexed address lines to address a location in the (256 row × 256 column) MOS chips in the selected bank.

Address strobes are asserted on the bus by the MCT module to load the row and column addresses on the address lines into the MOS chips. A write strobe is also asserted for memory write references. A read enable is generated to specify a memory read reference. Refer to the *VAX-11/730 Memory System Technical Description* for a more detailed description of array bus operation.

TK-6634
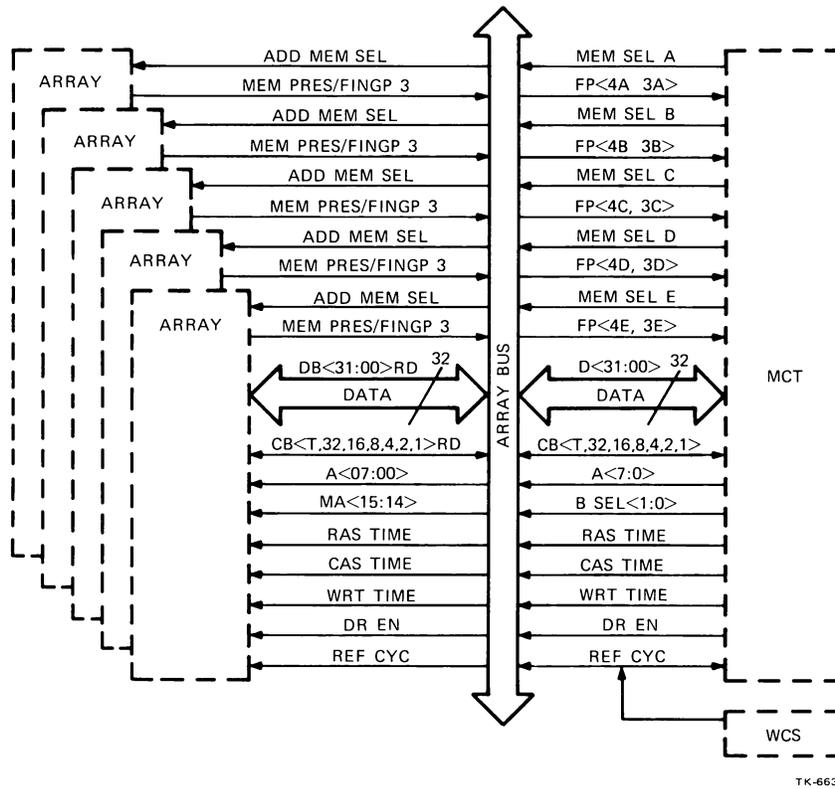
Figure 1-10   VAX-11/730 Memory Array Bus

Table 1-5   VAX-11/730 Memory Array Bus Signal Summary

Signal(s)*	Assertion Level	Description
MEM SEL⟨A:E⟩	L	Memory (module) select lines (5, 1 per array module). One line is asserted by MCT module to select a module in memory array.
B SEL⟨1:0⟩	H	Bank select lines (2). Asserted by MCT module to select one of four banks in selected array module.

B SEL		Bank
1	**0**	**Bank**
0	0	Bank
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

1-17

Table 1-5 VAX-11/730 Memory Array Bus Signal Summary (Cont)

Signal(s)*	Assertion Level	Description
A⟨7:0⟩	H	Multiplexed row/column address lines (8). Asserted by MCT module to select one of 64K longword locations in selected bank.
D⟨31:00⟩	L	Data lines (32). Transfer memory read/write data between MCT module and selected location in memory array.
CB⟨T,32,16 8,4,2,1⟩	L	Check bit lines (7). Transfer ECC read/write data between MCT module and selected location in memory array.
RAS TIME	L	Row address strobe.
CAS TIME	L	Column address strobe.
WRT TIME	L	Write strobe.
DR EN	L	Data read enable. Enable data and check bits from selected location in memory array onto data lines.
REF CYC	L	Refresh cycle. Asserted by WCS module to refresh a row address in all storage elements in memory array. Selects all array modules and allows address strobes for all four banks to be generated. Causes refresh address (a row address) to be asserted on A lines.
FP4A/3A FP4B/3B FP4C/3C FP4D/3D FP4E/3E	L L L L L	Fingerprint lines (10, two per array module).

FP4x	FP3x	Status
0		Module x not present
1	1	Module x present, fully-populated

*Signal names listed are as given in MCT module print set. Refer to Figure 1-10 for signal names as they are given in array module print set.

1.10.4 FPA/Port Bus

The FPA/port bus connects both the FPA and the IDC (the only current port device) to the CPU. (A port device can transfer data to and from the CPU by means of the CPU's fast interrupt facility.)

Bus line connections are shown in Figure 1-11. Table 1-6 describes the bus signals.

The 32 data lines on the FPA/port bus are an extension of the CPU's Y bus within the DAP module. During FPA transfers, the data lines transfer operand data to the FPA and result data from the FPA. The FPA's micro-PC may also be read and written over the data lines. During port transfers, the data lines transfer commands and device write data to the IDC, and status and device read data from the IDC. One 32-bit longword of device read/write data is transferred at a time, following a fast interrupt request by the IDC.

The other FPA/port bus lines (other than data lines) are the data strobes and synchronizing signals necessary to control the different types of transfers over the bus. The function and time relation of these signals during a transfer are discussed in Paragraph 6.12 of this technical description and in both the *VAX-11/730 Floating Point Accelerator Technical Description* and the *VAX-11/730 Integrated Disk Controller Technical Description.*
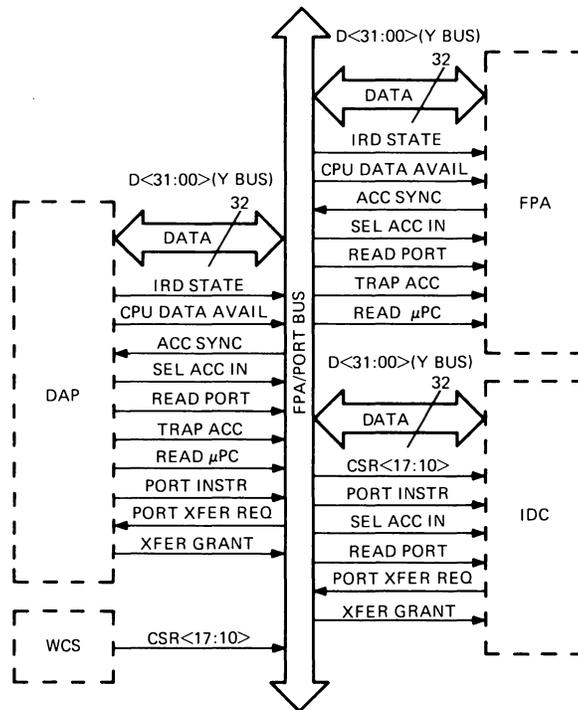
TK-6638

Figure 1-11   VAX-11/730 FPA/Port Bus

Table 1-6 VAX-11/730 FPA/Port Bus Signal Summary

Signal(s)	Assertion Level	Description
D⟨31:00⟩	H	Data lines (32). Extension of Y bus in CPU. Transfer data between CPU and FPA or port device (e.g., IDC).
IRD STATE	L	Instruction decode state. Asserted by DAP module during class decode operation by CPU. Used to indicate to FPA that opcode data is asserted on IB bus.
CPU DATA AVAIL	L	CPU data available. Asserted by DAP module to indicate to FPA that operand data is present on the data lines.
ACC SYNC	H	Accelerator synchronize. Asserted by FPA to synchronize the transfer of result data over the data lines to the CPU. Also synchronizes transfer of operand data from the CPU during execution of POLY instruction.
SEL ACC IN	H	Select accelerator. Asserted by DAP module to select FPA for transfer of result data over the data lines.
READ PORT	L	Asserted by DAP module to indicate that CPU is ready to receive result (or micro-PC) data from FPA or device read data from port device.
TRAP ACC	L	Trap accelerator. Asserted by DAP module to cause FPA to trap to micro-address asserted on data lines.
READ μPC	L	Read accelerator micro-PC. Asserted by DAP module to indicate CPU wants to read FPA's micro-PC over data lines during next CPU microcycle. (READ PORT asserted by DAP module during next microcycle.)
CSR⟨17:10⟩	H	Command byte for port device. Signals are from control store register on WCS module.
PORT INSTR	H	Port instruction. Asserted by DAP module to indicate to port device that command byte is present on CSR lines CSR⟨17:10⟩. Device write data (or other data depending upon the command) may be present on data lines.
PORT XFER REQ	L	Port transfer (interrupt) request. Asserted by port device to indicate to CPU that it is ready to transfer device read/write data. Causes a fast interrupt in the CPU.
XFER GRANT	L	Transfer grant. Asserted by DAP module to clear fast interrupt request (PORT XFER REQ) in port device.

1.10.5 Console Bus
The 8-bit console bus is a buffered extension of the AD bus in the 8085A console processor. It transfers data between the console processor on the WCS module and the data path on the DAP module as discussed in Paragraph 2.6.

1.10.6 IB Bus
The 8-bit IB bus, which is part of the instruction processing hardware on the DAP module, connects to the FPA so that the FPA may sample opcode data during the CPU's class decode operation. The transfer of opcode data is discussed in Paragraph 6.12.

1.11 DEFINITION OF THE CPU FOR DOCUMENTATION PURPOSES
As described previously, the KA730 CPU consists of the MCT, WCS, and DAP modules. However, for documentation purposes, the memory (and UNIBUS) control logic which is located on the MCT module and partly on the WCS module (the UNIBUS data transceivers and memory refresh logic are located on the WCS module) is treated as part of the memory system (described in the *VAX-11/730 Memory System Technical Description*).

NOTE
The memory (and UNIBUS) control logic described in the *VAX-11/730 Memory System Technical Description* is sometimes referred to as the memory controller (MCT) in the following chapters.

The chapters that follow in this technical description describe the rest of the CPU on the WCS and DAP modules. This hardware, hereafter referred to in this document as the CPU, consists of the following major logic groups.

1. 8085A console processor
2. CPU clock generator
3. CPU control store and microsequencer
4. Instruction processing hardware
5. Data path
6. Interrupt processing hardware

These major logic groups are connected as shown in Figure 1-12.
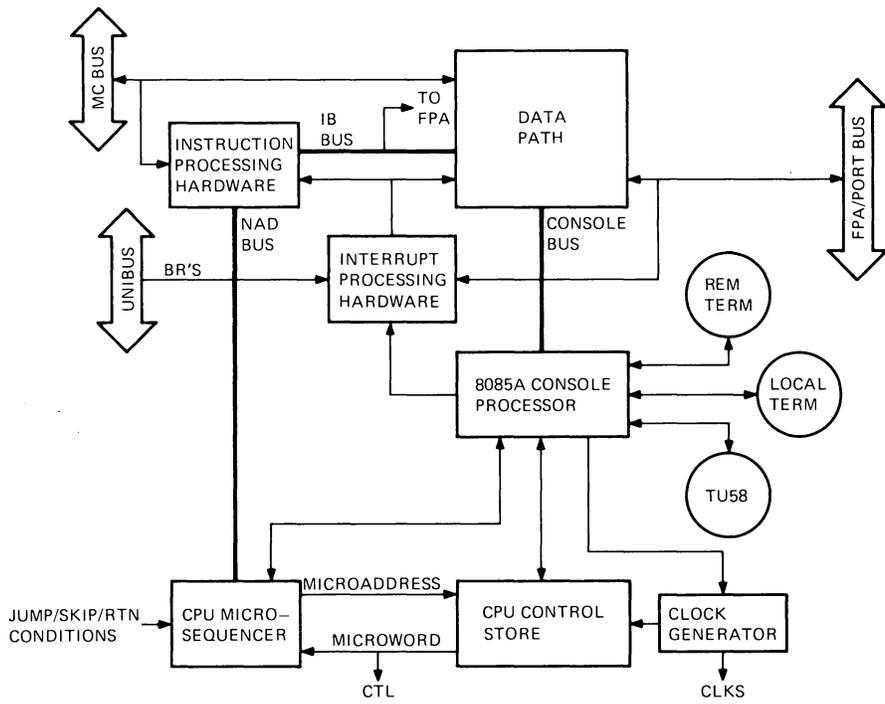
Figure 1-12 Central Processing Unit Functional Block Diagram

CHAPTER 2
CONSOLE PROCESSOR

2.1 INTRODUCTION

The CPU's console processor provides the main operator and maintenance interface to the VAX-11/730. A block diagram is shown in Figure 2-1.

The principal logic element in the console processor is an 8-bit 8085A microprocessor that executes a program controlling all console functions. Associated with the 8085A is a 4K × 8-bit ROM that stores the resident portion of the console program, (the ROM is 6K × 8 bits if the RD option is installed), and a 16K × 8-bit RAM that stores the main body of the console program loaded from the TU58 during the system bootstrap operation. The RAM also stores the console-based microdiagnostics and microdiagnostic monitor executed by the 8085A and loaded from the TU58 during diagnostic operations.

Other logic elements in the console processor include an interval timer and three universal synchronous/asynchronous receiver/transmitters (USARTs) for interfacing the CPU to the console terminal; the TU58; and the remote line (connecting to a modem) that is used in the field for remote diagnostic purposes. The remote line is also used for automated product testing (APT) during the manufacturing process.

Data and address information is multiplexed and transferred within the console processor over the bidirectional AD bus. This 8-bit bus is an extension of the internal address/data bus in the 8085A (AD⟨7:0⟩). During console operations, address information is first transmitted on the bus by the 8085A. Read/write data is then transferred over the bus between the 8085A and the addressed console device.

The address information transmitted on the AD bus is either the low-order eight bits of a memory (ROM or RAM) address, or an 8-bit I/O address. I/O addresses are the addresses of console devices other than memory, such as the interval timer or USARTs. Other I/O devices in the console processor include console command (output) registers and input data multiplexers as shown on sheet 2 of Figure 2-1.

The I/O address transmitted by the 8085A on the AD bus is also transmitted on a set of eight conventional (non-multiplexed) address lines. These address lines (A⟨15:08⟩), not the AD lines, are used to select most of the console's I/O devices. The A lines also address the memory devices (together with the AD bus) and supply the high-order portion of a ROM or RAM address.

Although most of the console processor is located on the WCS module, the 8-bit console write register (CWR) and the 8-bit console read register (CRR) are located on the DAP module. These registers are used to transfer data, one byte at a time, between the console processor and the CPU's data path. The CWR is addressed and loaded by the 8085A and then read onto the data path's D bus by the CPU microcode. The CRR is addressed and read by the 8085A after being loaded by the CPU microcode from the data path's Y bus. Other parts of the console processor located on the DAP module include a command register and an input data multiplexer. These components, plus the CWR and CRR, are shown on sheet 3 of Figure 2-1.
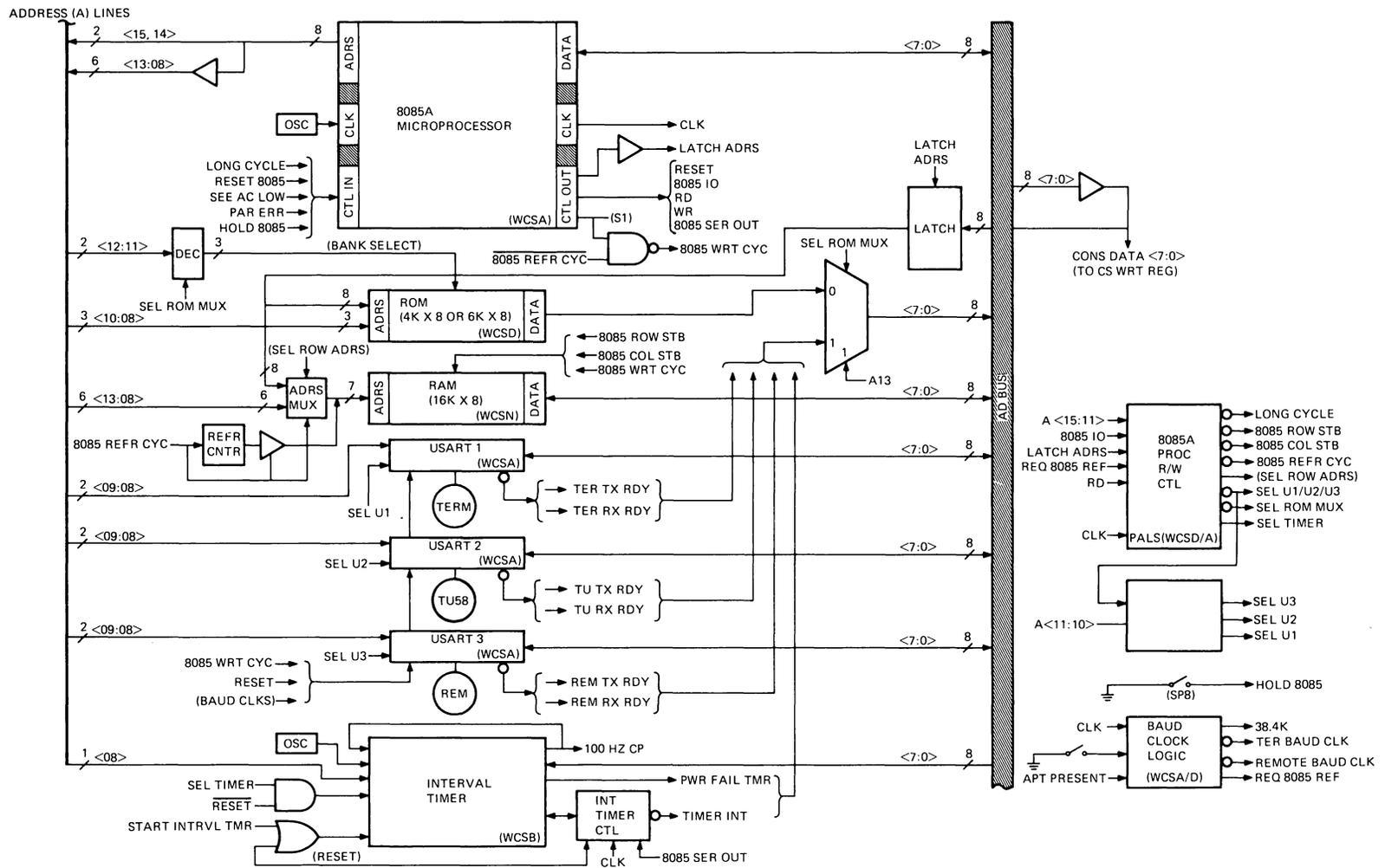
Figure 2-1 8085A Console Processor (Sheet 1 of 3)
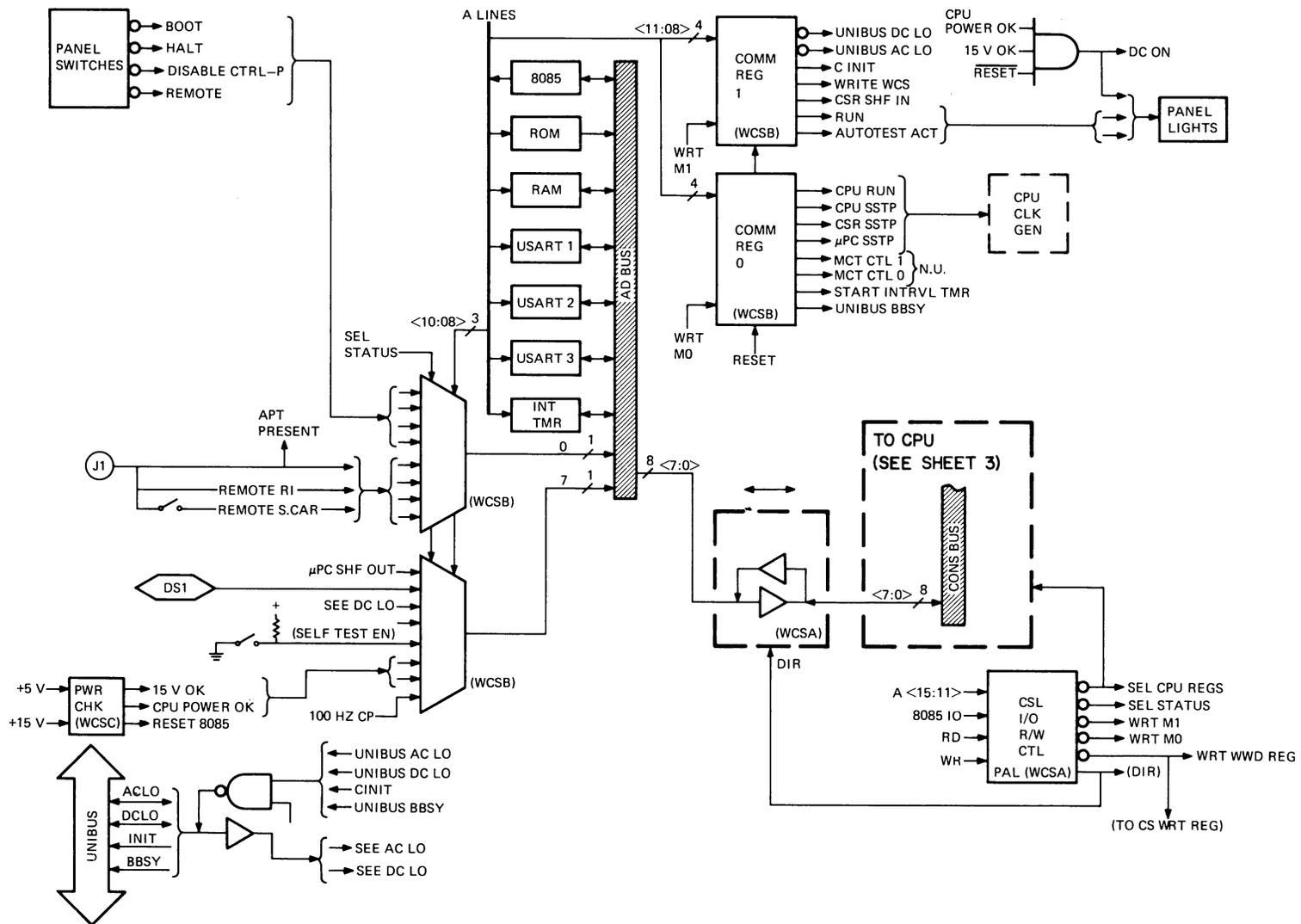
Figure 2-1 8085A Console Processor (Sheet 2 of 3)
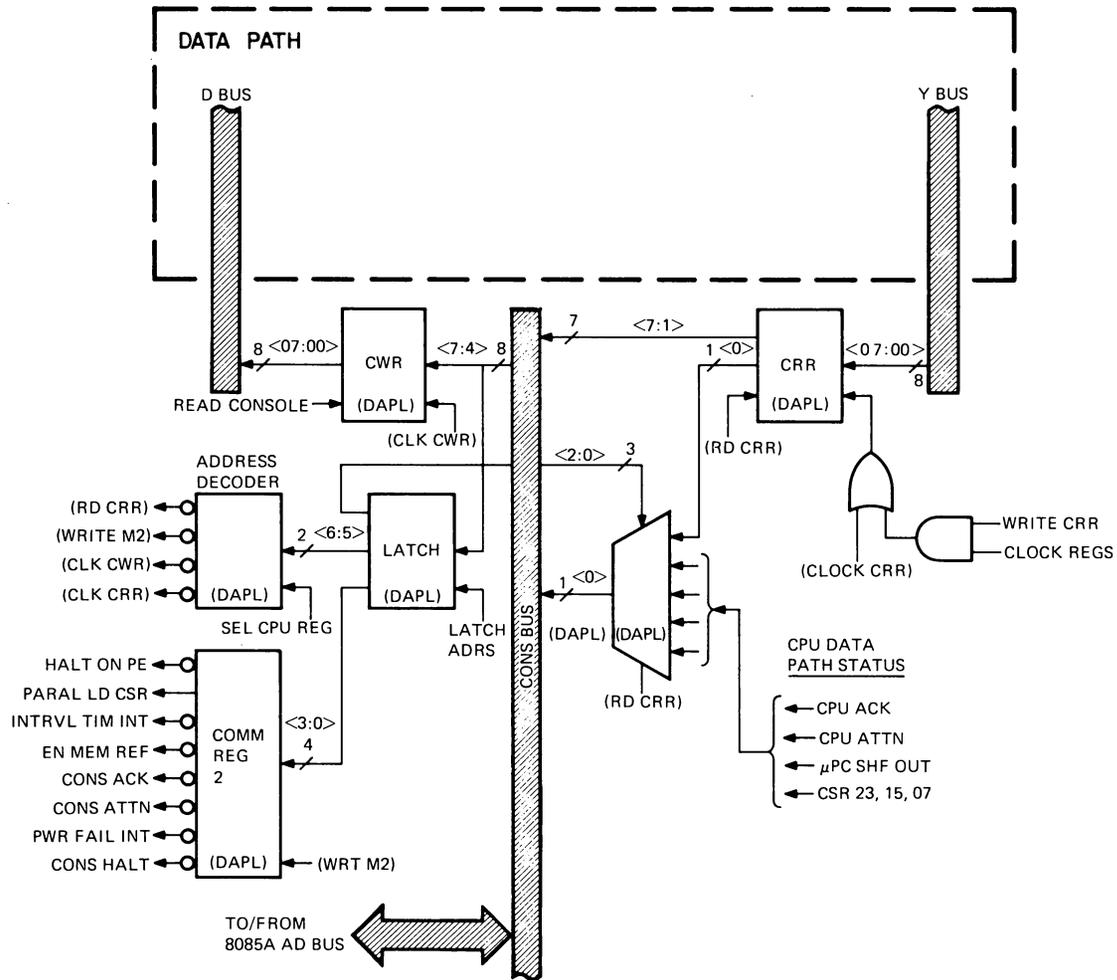
Figure 2-1   8085A Console Processor (Sheet 3 of 3)

The transfer of information between the WCS and DAP modules is over the 8-bit console (CONS) bus. This bus is a buffered extension of the WCS module's AD bus. As a result, device address and data information is multiplexed on the console bus as described previously for the AD bus.

2.2 8085A MICROPROCESSOR

The 8085A is an 8-bit parallel CPU that executes the program stored in the console processor's ROM and/or RAM. A block diagram of the 8085A is shown in Figure 2-2. Input/output pin definitions are given in Table 2-1.
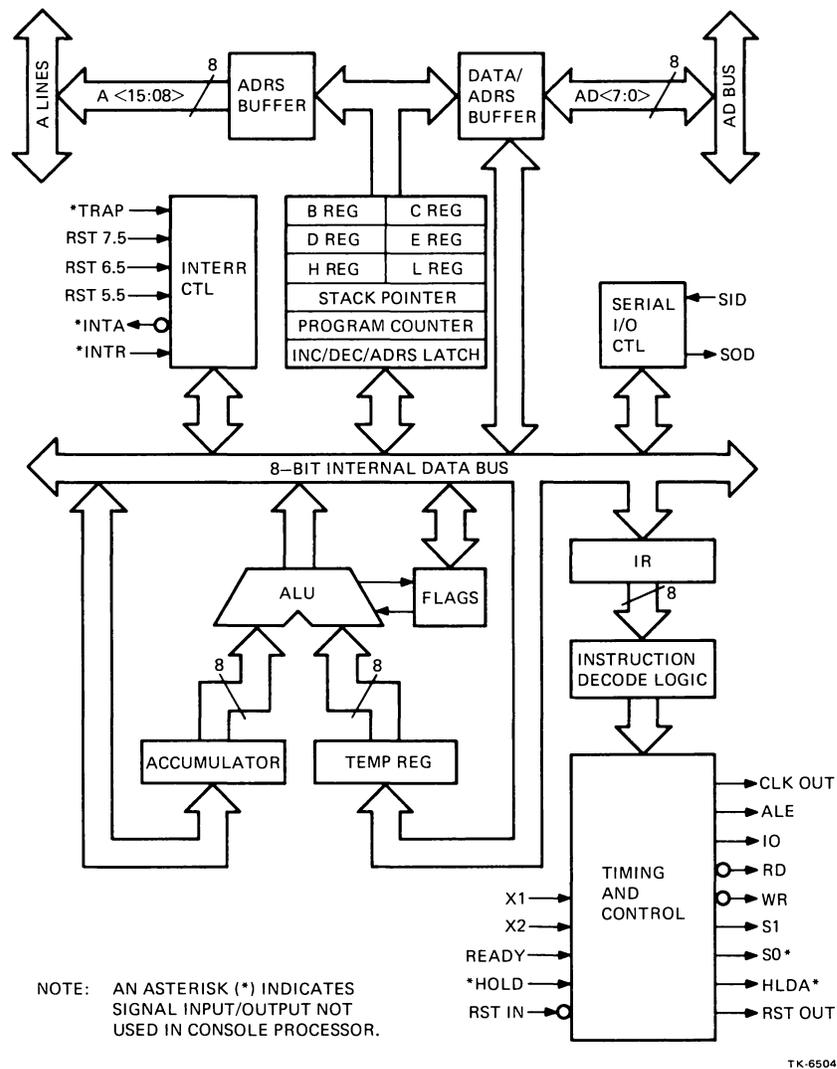
Figure 2-2   8085A Microprocessor

Table 2-1   8085A Input/Output Pin Definitions

Pin(s)	Function
A⟨15:08⟩	Address line outputs (tri-state). The eight most significant bits of memory address or the eight bits of I/O address. A⟨15⟩ is the most significant address bit.
AD⟨7:0⟩	Multiplexed address and data bus inputs/outputs (tri-state). The eight least significant bits of memory address or the eight bits of I/O address during the first machine state (T1) of a machine cycle. Data bus during second and third machine states (T2 and T3).
S1, S0	Status outputs. Encoded status of the machine cycle as follows:

S1	S0	Status
0	0	Halt
0	1	Write
1	0	Read
1	1	Instruction fetch

Pin(s)	Function
RD	Read. An output (tri-state) that indicates current machine cycle is a memory or I/O read operation and that the AD bus is available for data transfer. Asserted during T2. Negated during T3.
WR	Write. An output (tri-state) that indicates current machine cycle is a memory or I/O write operation. Write data may be strobed from AD bus at trailing edge. Asserted during T2. Negated during T3.
IO	I/O operation. An output (tri-state) that indicates the current machine cycle is an I/O read or write operation. (Negated during a memory read or write operation.)
HOLD	Not used in console processor.
HOLDA	Not used in console processor.
INTR	Not used in console processor.
INTA	Not used in console processor.
RST 7.5, 6.5, 5.5	Restart (interrupt) inputs. Cause automatic jump to interrupt service routine.

RST	Restart Address	Comments
7.5	3C	Highest priority
6.5	34	–
5.5	2C	

Table 2-1   8085A Input/Output Pin Definitions (Cont)

Pin(s)	Function
TRAP	Not used in console processor.
READY	Ready input. If asserted at start of T2, indicates read/write data may be transferred during next machine state (T3). If not asserted at start of T2, indicates 8085A is to wait for READY to be asserted before completing read or write operation.
SOD	Serial output data line.
SID	Serial input data line.
X2, X1	Clock inputs (10 mHz in console processor). Divided by two to determine the internal clock frequency.
CLK	Internal clock output (5 mHz in console processor).
RST IN	Reset input. Clears interrupt enable and HOLDA flip-flops. Resets program counter to zeros.
RST OUT	Reset output. Indicates 8085A is being reset.

The execution of instructions in the 8085A consists almost entirely of a series of read or write data operations between the 8085A and the PROM, RAM, and console I/O devices. Of course the devices addressed, the data (as it is processed by the 8085A), and the sequence of read/write operations vary depending on the instructions being executed.

Each read or write operation that occurs during the execution of an 8085A program is called a machine cycle, and each 8085A instruction requires from one to five machine cycles for execution. Each machine cycle, in turn, consists of a minimum of three to six machine states where a state is equal to one 8085A clock period (T). The clock period for the type of 8085A used in the console processor (an 8085A-2) is 200 ns.

The types of 8085A machine cycles are the instruction fetch, memory read, memory write, I/O read, I/O write, interrupt acknowledge, and bus idle. The 8085A input/output signal timing for all cycles but the interrupt acknowledge (which is not invoked by the console program) and the bus idle cycle (which reads or writes no console devices) is shown in Figure 2-3.

One of the first 8085A output signals asserted during a machine cycle is an address latch enable (ALE). This control signal is asserted during the first machine state (T1) of all machine cycles to facilitate loading of the address information on the AD bus into external latch circuits. This is necessary because the multiplexed AD bus lines also carry data later in the machine cycle. For example, during an instruction fetch cycle, which is the first (and sometimes only) machine cycle during execution of an 8085A instruction, the low-order memory address is transmitted on the AD bus by the 8085A during the first machine state only. The AD bus lines are then used to transfer instruction data from the addressed ROM or RAM location to the 8085A.

2-7
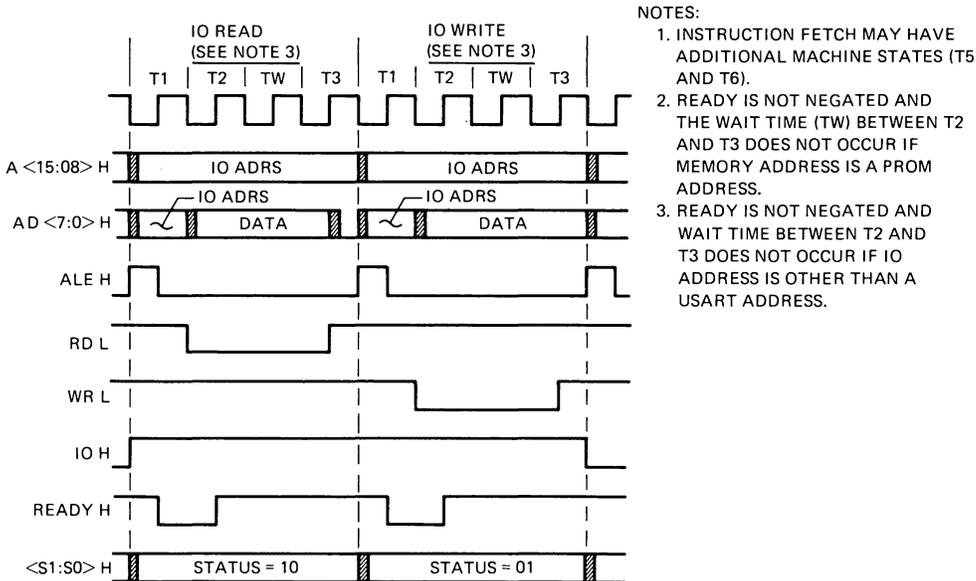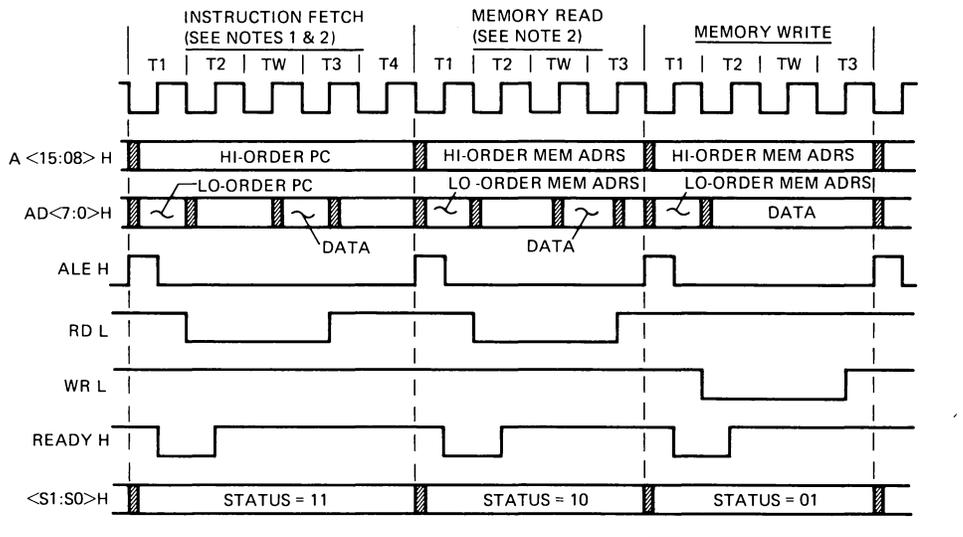
Figure 2-3 8085A Machine Cycles Timing Diagram

Other 8085A outputs asserted at the beginning of a machine cycle are an IO signal and two status lines, S1 and S0. These three outputs define the type of machine cycle. The IO signal is asserted for the entire machine cycle when accessing an I/O (not a memory) device.

The status lines, which are also asserted for the entire cycle, further define the machine cycle as a read or write. That is, S1 and S0 have a binary value of 01 for both an I/O write cycle and a memory write cycle, and they are equal to 10 for both an I/O read and a memory read. Also, the status lines specify an instruction fetch when they are equal to 11, and a bus idle cycle (due to a processor halt) when they are 00.

Two 8085A outputs are provided to act as read/write data enable or strobe signals during a machine cycle. An RD signal, which is asserted after T1 of a read cycle, is used to gate data onto the AD bus from an addressed console device following the transmission of the address data on the bus. A WR signal, the trailing edge of which occurs during T3 of a cycle, is used to strobe or latch write data asserted on the AD bus into an addressed device.

Except for an instruction fetch, the machine cycles transferring data to or from a console device require only three consecutive machine states to execute normally. (The instruction fetch requires a minimum of four or six states.) However, when more time is required to read or write data from a particular device, the 8085A's READY input may be negated to introduce additional time between T2 and T3 in the cycle. The READY signal is negated in the console processor to cause a wait time equal to one 8085A clock period (TW) when reading and writing the RAM or the USARTs.

2.3 2651 USARTS

The console processor contains three 2651 universal synchronous/asynchronous receiver/transmitters (USARTs). One USART connects to the console terminal, the second to the TU58, and the third to the remote line modem. Each USART operates in asynchronous mode to transfer data over a serial line to and from its connecting device. A block diagram of the 2651 USART is shown in Figure 2-4. Table 2-2 defines the input/output pins.
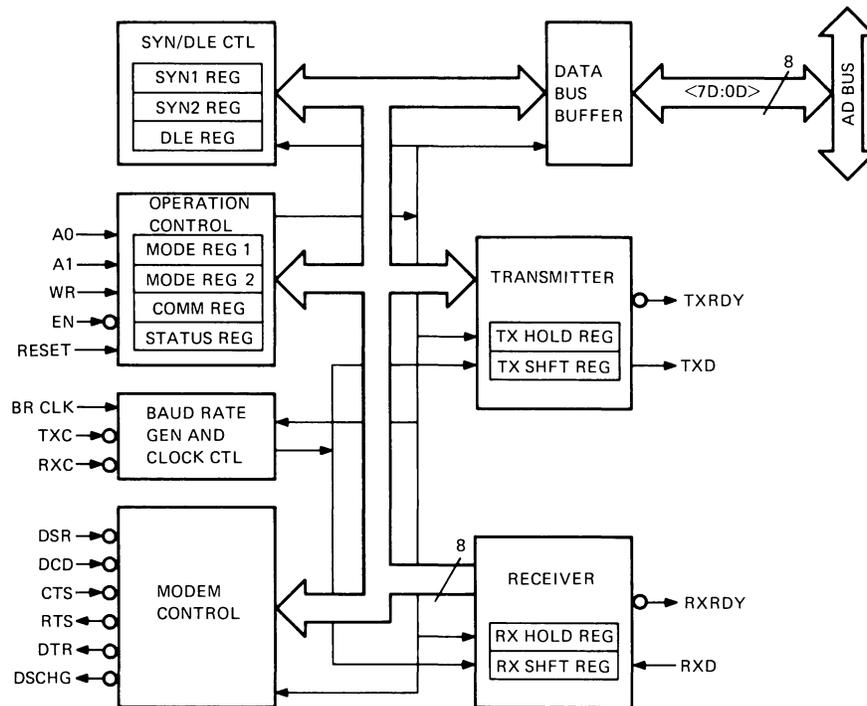
TK-6503

Figure 2-4   2651 USART

Table 2-2   2651 USART Input/Output Pin Definitions

Pin(s)	Function
⟨7D:0D⟩	Data bus inputs/outputs (tri-state). Transfer read/write data plus command and status information. 7D is the most significant bit.
A1, A0	Address inputs. Select internal registers.
WR	Write input. Specifies a read when negated.
EN	Chip enable input. Perform operation specified by A1, A0, and WR.

EN	A1	A0	WR	R/W Operation
1	0	0	0	Read data (receive holding) register
1	0	0	1	Write data (transmit holding) register
1	0	1	0	Read status register
1	0	1	1	Write SYN1/SYN2/DLE registers
1	1	0	0	Read mode registers 1 and 2
1	1	0	1	Write mode registers 1 and 2
1	1	1	0	Read command register
1	1	1	1	Write command register
0	–	–	–	No transfer – data bus off (tri-state condition)

Pin(s)	Function
TXRDY	Transmit ready output. Transmit holding register ready to accept a character. Negated when holding register is loaded.
RXRDY	Receive ready output. Receive holding register contains a character. Negated when holding register is read.
TXD	Serial data output from transmitter.
RXD	Serial data input to receiver.
DSR	Data set ready input. Also ring indicator condition input.

Table 2-2   2651 USART Input/Output Pin Definitions (Cont)

Pin(s)	Function
DCD*	Data carrier detect input. Must be asserted for receiver to operate.
CTS*	Clear to send input. Must be asserted for transmitter to operate.
DTR	Data terminal ready output.
DSCHG	Not used in console processor.
BRCLK	Not used in console processor.
TXC	Transmitter clock input.
RXC	Receiver clock input.
RESET	Reset input. Force idle state. Clear mode, command, and status registers.

*Input from modem. Always asserted (pin tied to ground) at input to console (local) terminal USART and TU58 USART.

2.3.1   Basic Operations

One of the basic functions of a USART is to accept data characters in parallel format and then convert them into a continuous serial data stream for transmission. The other is to receive a serial data stream and convert it into parallel data characters. In the console processor, USART operations are controlled by the 8085A microprocessor. Bit maps and I/O addresses for the 2651 registers that may be accessed by the 8085A for control purposes are shown in Figure 2-5.

Register addressing in the USARTs is controlled by two address inputs, A1 and A0, which connect to two of the 8085A's A lines (A09 and A08). The A lines select either a data register address, a mode register, the command register, or a status register whenever a USART's chip select level is true. (Generation of the chip select levels is described in Paragraph 2.5.4.1.) Whether the addressed register is to be read or written is determined by the USART's WR input. This input is asserted by 8085 WRT CYC during an AD bus write operation. The 8085 WRT CYC signal is derived from one of the 8085A's status line outputs (S1).

The 8085 loads a character into the USART's data register to transmit the character over a serial line. (A write to the data register address loads the character into the USART's TX hold register.) However, the USART must be conditioned to transmit prior to the load. That is, the data set ready (DSR) input pin must be asserted, the data terminal ready (DTR) output is asserted, and the transmit enable (TXEN) bit must be set in the command register. The transmit ready (TXRDY) bit is asserted to indicate that the USART is ready to transmit data. Loading the TX hold register causes TXRDY to be negated. The data in the TX hold register is loaded from TX hold into the TX shift register, and TX RDY is asserted again. Request to send (RTS) is asserted and the USART waits for clear to send (CTS). The contents of the TX shift register are transmitted over the transmit data (TXD) line.

IO
REGISTER | ADDRESS | READ/WRITE | 07 06 05 04 03 02 01 00

DATA | 40(U3) 44(U2) 48(U1) | W | CHARACTER TO BE TRANSMITTED

DATA | 40(U3) 44(U2) 48(U1) | R | RECEIVED CHARACTER

STATUS | 41(U3) 45(U2) 49(U1) | R | DSR | DCD | STB ERR | OVN ERR | PAR ERR | | |

RXRDY

DSCHG TXRDY

MODE 1 (SEE NOTE 3) | 42(U3) 46(U2) 4A(U1) | R/W | STOP BITS | EVN PAR | EN PAR | CHAR LENGTH | MODE AND BR

1 = EVEN PAR
* 0 = ODD PAR

00 = SYNC, CLK
01 = ASYNC, CLK
* 10 = ASYNC, CLK/16
11 = ASYNC, CLK/64

00 = INVALID
* 01 = 1 STOP BIT
10 = 1½ STOP BITS
11 = 2 STOP BITS

00 = 5 BITS
01 = 6 BITS
10 = 7 BITS
* 11 = 8 BITS

TK-6560

REGISTER | IO ADDRESS | READ/WRITE | 07 06 05 04 03 02 01 00

MODE 2 (SEE NOTE 3) | 42(U3) 46(U2) 4A(U1) | R/W | NOT USED | TXC | RXC | INTERNAL BAUD RATE SEL

(NOT USED IN CONSOLE PROCESSOR)

* 0 = EXTERNAL
1 = INTERNAL

COMMAND | 43(U3) 47(U2) 48(U1) | R/W | SUBMODE | RTS | CLR ERR | | RX EN | DTR | TX EN

TXD (SPACE)

* 00 = NORMAL
01 = AUTO ECHO
* 10 = LOCAL LOOP BACK
11 = REMOTE LOOP BACK

(WRITE ONLY, AUTO-CLEAR AFTER SET)

NOTES: 1. SYNCHRONOUS MODE REGISTERS (SYN1/SYN2/DLE) AND SYNCHRONOUS MODE CONTROL BITS IN OTHER REGISTERS NOT USED IN CONSOLE PROCESSOR AND ARE NOT SHOWN.
2. AN ASTERISK (*) INDICATES PARAMETERS USED IN CONSOLE PROCESSOR.
3. ADDRESSING OF MODE 1 AND MODE 2 REGISTERS (WHICH HAVE THE SAME I/O ADDRESS) IS CYCLIC. THAT IS, THE FIRST READ OR WRITE ADDRESSES MODE 1, THE SECOND MODE 2. THE MODE 1 REGISTER IS ALWAYS READ OR WRITTEN WHEN THE MODE REGISTERS ARE ADDRESSED FOLLOWING A RESET OR A COMMAND REGISTER READ.

TK-6488

Figure 2-5 Bit Formats for 2651 USART Registers

The USART is conditioned to receive data over the serial line when the data carrier detect (DCD) input is asserted and the receive ready (RXEN) bit in the command register is set. Whenever a complete character has been assembled in the USART's RX shift register, it is transferred into the RX holding register and the RXRDY output (also a status register bit) is asserted to indicate the 8085A may take the data.

To take a received data character, the 8085A reads the USART's data register address. (Reading the data register address transmits the RX holding register contents onto the USART's data outputs and thus onto the console processor's AD bus.) RXRDY, which is negated by the data register read, is then reasserted whenever the next data character has been assembled in the RX shift register and transferred into the RX holding register.

Errors that may be detected when receiving data are overrun (OVN) errors, parity (PAR) errors, and stop bit (STB) errors. Each type of error sets a bit in the USART's status register which may be read by the 8085A. The overrun error indicates that a data character was not read by the 8085A before a new character was assembled and loaded into the RX holding register. The parity error indicates received bad parity. The stop bit error indicates that the received character was not framed by the correct number of stop bits.

Error bits are cleared when the receiver is disabled (RXEN cleared in the command register) or by a reset error command (CLR ERR set in the command register).

NOTE
The USART parity error flag is not checked by the 8085A console program.

The operating mode for a USART (character length, the number of stop bits, etc.) is specified by writing the appropriate bits in the USART's two mode registers. Asterisks entered on the register bit maps (Figure 2-5) indicate the operating modes used in the console processor. For example, character length is 8 bits which is specified by making the 2-bit CHAR LENGTH field in mode register 1 equal to a binary 11.

2.3.2 USART Clocks
All three USARTs in the console processor use an external clock source. Also, the transmit and receive clock input pins for any USART are tied together. The mode register is set so that the USART divides the clock input by 16.

The USART's external clock generator and related logic is shown in Figure 2-6. The clock outputs are derived from the 200 ns 8085A clock, which is divided by four and used to clock a counter. The counter outputs, when divided by 16 by the USART, then provide the clocks for operation at baud rates of 38,400, 9600, 2400, 1200, and 300. The counter also provides an output (REQ 8085 REF) that is used to initiate the refresh cycle for the 8085A RAM on a periodic basis; that is, every 12.8 μs.

The USART for the TU58 operates at a baud rate of 38,400. The clock for the USART connecting to the remote line (REMOTE BAUD CK) also is 38,400 baud, but only during APT. During remote diagnostic operation, operation is switch-selectable at baud rates of either 300 or 1200. The clock for the console terminal's USART (TER BAUD CK) is also switch-selectable and operation may be at baud rates of 9600, 2400, 1200, or 300.
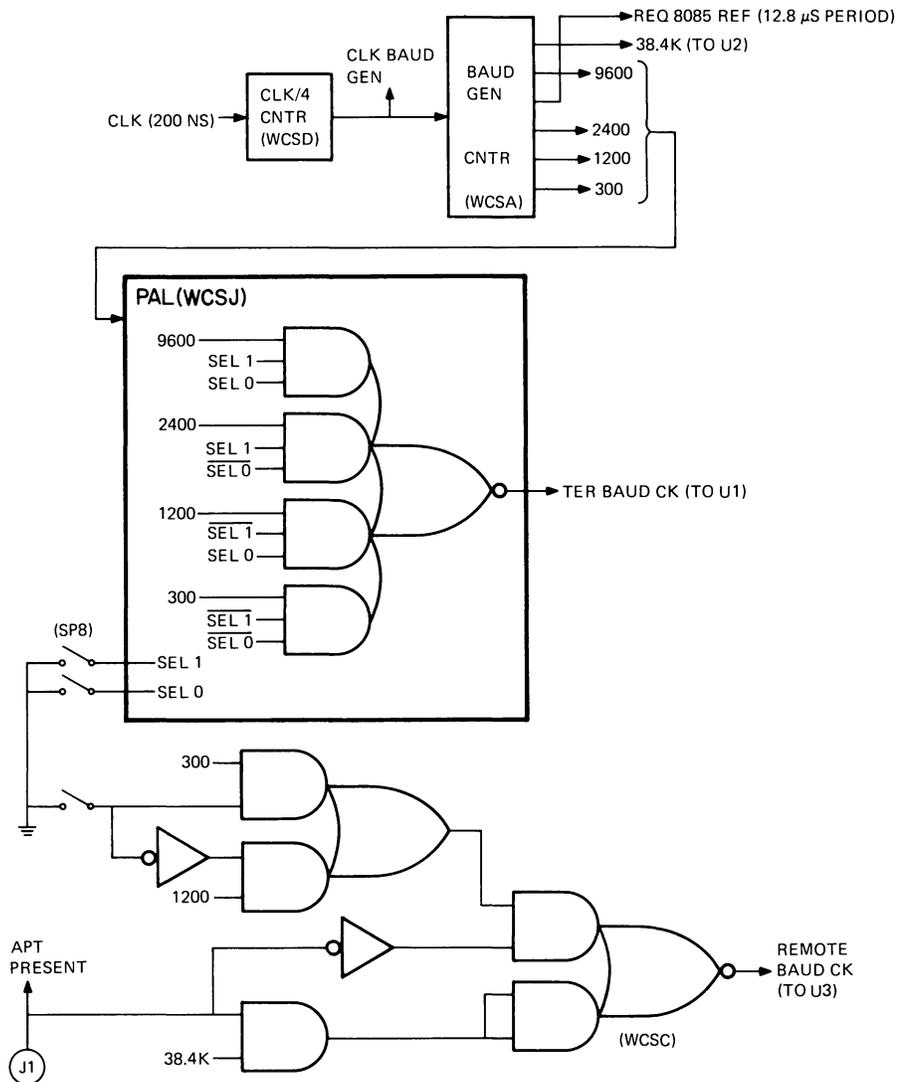
Figure 2-6 Baud Clock Logic

2.3.3 Terminal and Tape Data Transfers

The way that data transfers are made to and from a terminal or a tape via the USART differs, depending on whether the console program is operating in console mode or program mode.

In console mode, the data transfers are controlled completely by the console program itself. That is, the program determines when to transmit characters (prompts and error messages to a terminal, for example) or when to read and process received characters (e.g., indirect command data from a tape). The program examines a USART's TXRDY bit to determine when the next character to be transmitted may be loaded into the USART's data register. Similarly, it examines a USART's RXRDY bit to determine when a received character may be read from the data register. The program examines the TXRDY and RXRDY bits by reading the console processor's ready bit status register. (The I/O READ address is equal to 20.) The ready bits are read through the ROM multiplexer as discussed in Paragraph 2.5.4.2.

In program mode, terminal and tape data transfers are controlled by the console program; but the data is passed to and from the data path following processor interrupts, and only in response to MFPRs and MTPRs executed by the program running in the CPU.

To control the generation of interrupts, the console program (when in the program mode idle loop) polls the USART's status registers to check the state of the TXRDY and RXRDY bits. Then, when one is asserted, the program sets a corresponding bit in an 8-bit interrupt summary register. A TXRDY bit from either the local or remote terminal's USART sets one bit in the register; an RXRDY bit sets another. Two other bits in the register correspond to TXRDY and RXRDY in the TU58's USART. This interrupt summary register is not a hardware register. It is a software register contained in the console program (a RAM address). Bit formats are shown in Figure 2-7.

	07	06	05	04	03	02	01	00
INTERRUPT SUMMARY REGISTER	TTR IS	TTR IE	TTX IS	TTX IE	TUX IE	TUX IS	TUR IE	TUR IS

* TERM RXRDY   * TERM TXRDY   TU58 TXRDY   TU58 RXRDY

┌─ HIGHEST PRIORITY

	07	06	05	04	03	00
INTERRUPT PRIORITY REGISTER	PRI 1	PRI 2	PRI 3	PRI 4	NOT USED	

TU58 RX INTERR   * TERM RX INTERR
TU58 TX INTERR   * TERM TX INTERR

* NOTE:
TERMINAL MAY BE EITHER LOCAL OR REMOTE TERMINAL.

TK-6628
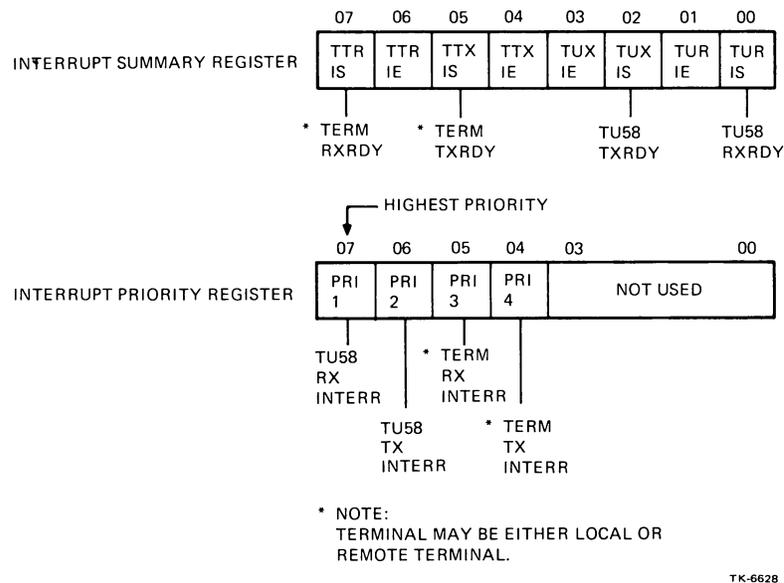
Figure 2-7   Bit Formats for Interrupt Summary and Priority
Registers in Console Program

Associated with each of the four interrupt summary register bits set as result of an asserted USART ready bit is an interrupt enable bit. Each enable bit is set by an MTPR (with bit 06 in the register data equal to one) that addresses the corresponding transmit or receive control/status register for the terminal or tape. Bit formats for the console and tape device registers are given in Figures 2-8 and 2-9.

The transmit and receive control/status registers for the terminal are the TXCS and RXCS. Those for the tape are the CSTS and CSRS. (The other registers in Figures 2-8 and 2-9 are the transmit and receive data buffer registers for the terminal and tape.) The control/status registers (as well as the data buffer registers) are pseudo registers, in that they are not dedicated hardware registers. They are implemented by the console program and CPU microcode. For example, when an MTPR instruction addressing a control/status register is executed, the CPU microcode sends the interrupt bit as part of a data packet from the data path to the console processor over the console bus. The console program then sets the appropriate interrupt enable bit in the console interrupt summary register. The transfer of data packets between the CPU microcode and console processor is discussed in Paragraph 2.6.

2-15

PROC REG	IPR ADRS	READ WRITE			
RXCS	20	R/W			

31   08 07 06 05   00

```
+--------------------------------------------------+--+--+--------+
|                      MBZ                         |  |  |  MBZ   |
+--------------------------------------------------+--+--+--------+
```
INT EN

DONE (READ ONLY)

RXDB	21	R

31   16 15 14   12 11   08 07   00

```
+------------------------------+--+--+--------+--------+
|              0               |  | 0| SELECT |  DATA  |
+------------------------------+--+--+--------+--------+
```
ERROR

0 = LOCAL/REMOTE TERMINAL
F = MISC. COMMUNICATIONS (SEE NOTE)

TXCS	22	R/W

31   08 07 06 05   00

```
+--------------------------------------------------+--+--+--------+
|                      MBZ                         |  |  |  MBZ   |
+--------------------------------------------------+--+--+--------+
```
INT EN

READY (READ ONLY)

TXDB	23	W

31   12 11   08 07   00

```
+------------------------------------------+--------+--------+
|                    MBZ                   | SELECT |  DATA  |
+------------------------------------------+--------+--------+
```

0 = LOCAL/REMOTE TERMINAL
F = MISC. COMMUNICATION (SEE NOTE)
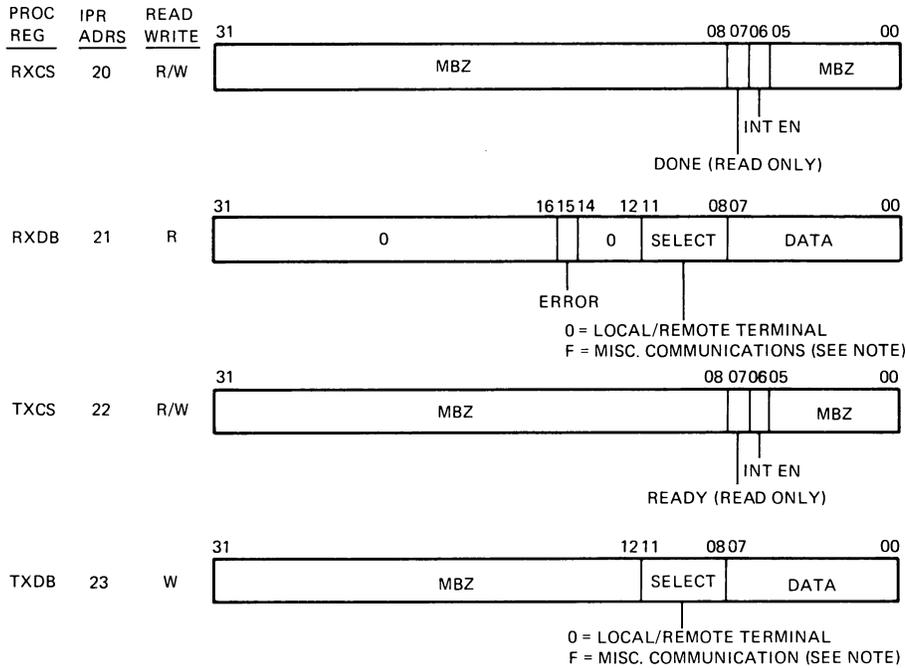
NOTE: SELECT FIELD EQUAL TO F INDICATES
             TXDB HAS SPECIAL FUNCTION SPECIFIED
             BY DATA FIELD

SELECT/DATA	FUNCTION
F/1	SOFTWARE DONE
F/2	BOOTSTRAP CPU
F/3	CLEAR WARM RESTART FLAG
F/4	CLEAR COLD RESTART FLAG

TK-6491

Figure 2-8   Bit Formats for Console Terminal Data and
Control/Status Registers

If an interrupt enable bit in the interrupt summary register is set by an MTPR, it causes processor interrupt requests to be generated [at an IPL of 14 (HEX)] by the console program when or if the associated interrupt summary bit is set. The console program generates an interrupt request by first setting one of four bits in another software register, the interrupt priority register (Figure 2-7).

The bit set depends on which interrupt summary bit (and interrupt enable bit) is set; that is, what type of data transfer request has been made (i.e., terminal read or write, or tape read or write). The console program then asserts CONS ATTN to interrupt the program running in the CPU. After asserting CONS ATTN, the console program returns to the program mode idle loop where it continues to check the USART's TXRDY and RXRDY bits.

Whenever a USART data transfer interrupt request (CONS ATTN) is serviced by the CPU microcode, a data packet (one byte) is sent from the data path to the console processor to acknowledge the request. In response, the console program clears CONS ATTN and returns the contents of the interrupt priority register in another data packet (again, one byte) so that the CPU microcode may determine which USART data transfer request (or requests) has caused the interrupt. (More than one USART ready bit may be asserted at a time.) If there is more than one USART data transfer request, the console program also reasserts the CONS ATTN signal.

2-16
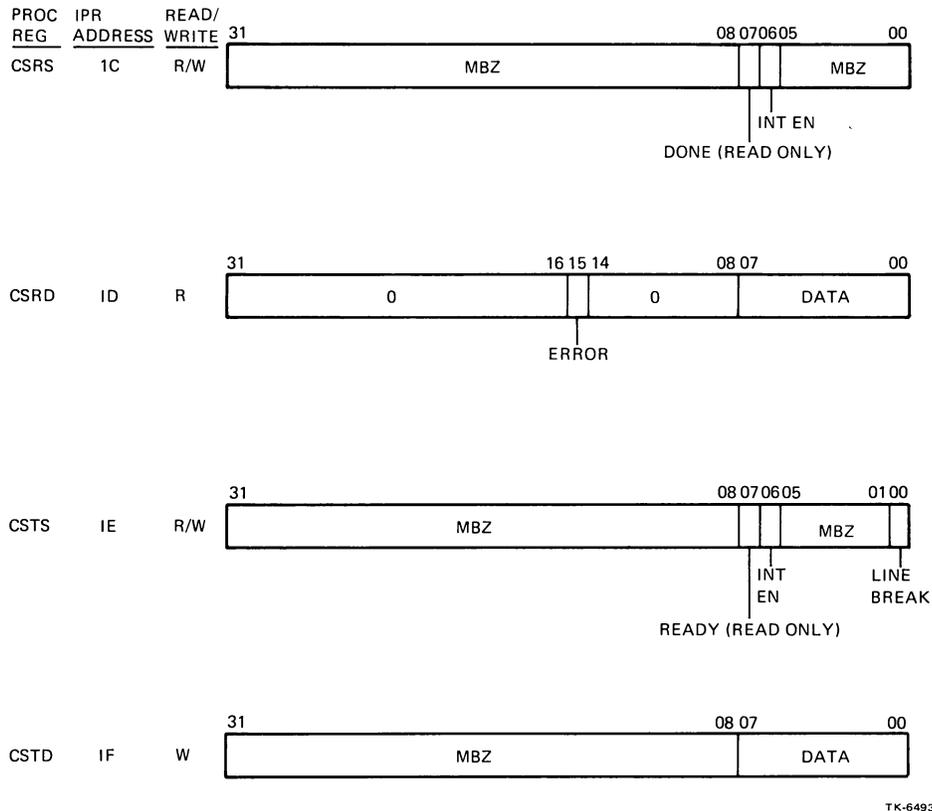
Figure 2-9　Bit Formats for Console Storage (Tape) Data and
Control/Status Registers

When the CPU microcode receives the contents of the interrupt priority register, it initiates a dispatch to an instruction level interrupt service routine for the highest priority USART data transfer request. As shown in Figure 2-7, the tape read transfer has the highest priority and the terminal transmit operation has the lowest.

During the instruction level interrupt service routine, an MTPR is executed to transfer transmit data to a terminal or tape, or an MFPR is executed to transfer received data from a terminal or tape. The MTPR or MFPR addresses a data buffer register. The data buffer registers for the terminal are the TXDB and RXDB. The tape's data buffer registers are the CSRD and CSTD.

With reference to Figures 2-8 and 2-9, the transmit data character (one byte) is contained in the low-order eight bits of register data for the TXDB and CSTD. When an MTPR referencing either one of these data buffer registers is executed by the program, the transmit data character is passed from the data path to the console processor (as part of a data packet) by the CPU microcode.

The console program loads a character sent in response to a TXDB reference into a pseudo transmit data buffer. It then loads the character into the data register of the appropriate USART for transmission to either the local or remote terminal. A character sent in response to a CSTD is loaded directly into the data register of the USART for the TU58. Also, for both a TXDB and CSTD, the console program clears the appropriate interrupt summary bit in the interrupt summary register, and the appropriate "interrupt pending" bit in the interrupt priority register, now that the interrupt request has been serviced. Once a USART's data register is loaded, the transmit data character is sent out serially to the terminal or tape.

2-17

The select field in the TXDB's register data is equal to zero for transmit data transfers. It can also be an F to indicate miscellaneous communications between the CPU microcode and the console program. In this case, the data field specifies an operation (other than a transmit data transfer) to be performed by the console program (bootstrap, etc.).

An MFPR referencing the RXDB or CSRD causes the console program (after receiving a command data packet from the CPU microcode) to read both a software data buffer, which contains the received data character, and the software control/status register for the USART. (The received character was read from the USART's data register and stored in the software data buffer following the assertion of the USART's RXRDY bit.)

The received character and software control/status register are then sent to the CPU microcode as part of a two-byte data packet. The CPU microcode unpacks the information and assembles it into the correct register bit format for the MFPR. The received data character is placed in the low-order eight bits of the RXDB or CSRD data. Also, an error bit in the RXDB or CSRD data (bit 15) is made equal to one when any of the error bits in the control/status information supplied by the console program are set. These are the data overrun and stop bit errors sensed by the USART. Parity errors sensed by the USART will not cause the MFPR's error bit to be set. This is because the USART's parity error bit is masked out by the console program whenever the USART's status register is loaded into the software control/status register.

The software control/status register is also sent to the CPU microcode by the console program when an MFPR is executed addressing the TXCS, RXCS, CSRS, or CSTS. In this case, the CPU microcode uses the TXRDY or RXRDY bit in the control/status information to set DONE (bit 07) in the register data it assembles. (The DONE bit indicates to the program the transmit or receive ready status of the terminal or tape.) The state of the corresponding interrupt enable bit is also made part of the register data (bit 06) assembled for MFPR. This bit is not part of the control/status data sent by the console program. The bit was stored previously by the CPU microcode at the time it was last set or cleared by an MTPR.

2.4 THE 9513 INTERVAL TIMER

The 9513 interval timer chip contains a 16-bit frequency scaler, a 4-bit divider circuit, and five general purpose 16-bit counters. Each counter and its associated control circuitry, called a counter logic group, may be programmed to specify a number of count modes, count sources, and input/output control functions. The 9513, which is controlled by the 8085A, also contains a data bus interface that connects to the console processor's AD bus. The data bus interface consists of both a data port and a control port for addressing the various registers internal to the chip. A block diagram of the 9513 interval timer is shown in Figure 2-10. Input/output pin definitions are given in Table 2-3.

Each counter logic group in the 9513 contains a 16-bit load register for pre-setting the counter, and a 16-bit hold register for storing the current count. Also, counter logic groups 1 and 2 contain an alarm register and a comparator circuit. If enabled to do so, the counter logic group's single output pin is asserted whenever the counter has a value equal to the alarm register contents.
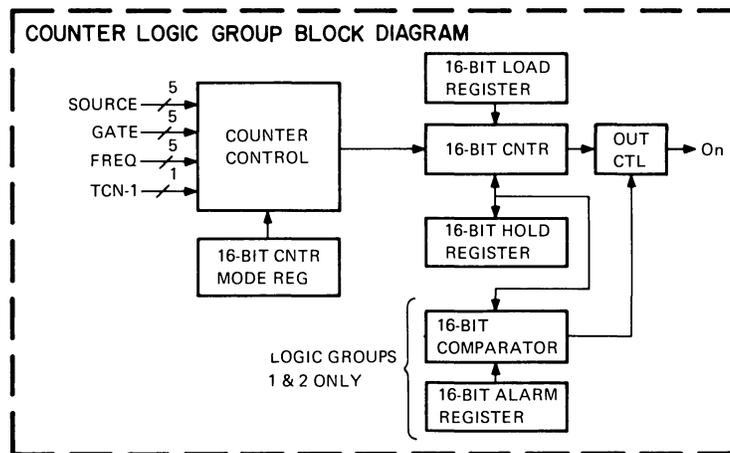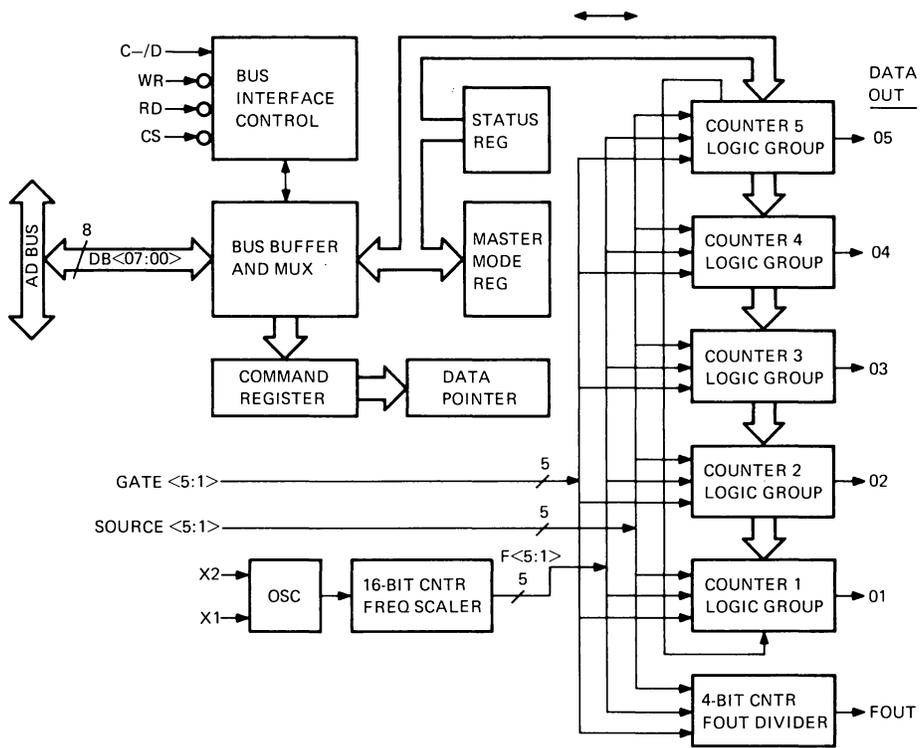
Figure 2-10   9513 Interval Timer Block Diagram

Table 2-3   9513 Interval Timer Input/Output Pin Definitions

Pin(s)	Function
DB ⟨07:00⟩	Data bus inputs/outputs (tri-state). Transfer command/status information and register read/write data. DB ⟨07⟩ is the most significant bit. (DB ⟨15:08⟩ not used; that is, data bus may be configured for 16-bit or 8-bit width and only 8-bit configuration is implemented in console processor.)
CS	Chip select input. Enables data bus read/write transfers.
RD	Read input. Specifies a data bus read.
WR	Write input. Specifies a data bus write.
C-/D	Control not/data input. Select port used for read/write transfers.

CS	C-/D	RD	WR	Operation
1	1	0	1	Write command register
1	1	1	0	Read status register
1	0	0	1	Write data register addressed by data pointer
1	0	1	0	Read data register addressed by data pointer
–	–	1	1	No transfer. Data bus off (tri-stated)
0	–	–	–	No transfer. Data bus off (tri-stated condition)

Pin(s)	Function
GATE ⟨5:1⟩	Gate inputs. A GATE n signal enables the counter in logic group n, n+1, or n−1 to count its source. The logic group counter enabled depends upon the gating control bits in the logic group's counter mode register. The same control bits also determine if the counter is enabled to be active when GATE n is high or low, or following a high or low transition of GATE n.
SOURCE ⟨5:1⟩	Source inputs. The SOURCE n signals provide an external count source for any or all of the counters (i.e., logic group counters and FOUT divider). Selection is by control bits in the counter mode registers (for the logic group counters) or the master mode register (for the FOUT divider). The control bits may also select any of the gate inputs or the outputs of the frequency scaler as count sources, and determine whether a count occurs on the positive or negative transition of the source.

Table 2-3 9513 Interval Timer Input/Output Pin Definitions (Cont)

Pin(s)	Function
⟨5:1⟩	Counter outputs (tri-state). Each output n signal is the output from the counter in the corresponding logic group (logic group n). Output may be a pulse, square wave, or have a complex duty cycle as determined by control bits in the logic group's counter mode register.
FOUT	Frequency out. Output of FOUT frequency divider. The input (source) for the divider may be divided by any value from 1 through 16, as determined by control bits in the master mode register. The source for the divider, which may be any of the SOURCE or GATE inputs, or any of the outputs of the frequency scaler, is also determined by control bits in the master mode register.
X2, X1	Clock inputs (one kHz in console processor). Determines frequency of internal oscillator which provides the input to the frequency scaler.

In addition to its use as an interval timer, the 9513 chip provides a time of year count and a power fail time-out. The utilization of the five counter logic groups to implement these three functions is shown in Figure 2-11.

TK-6500
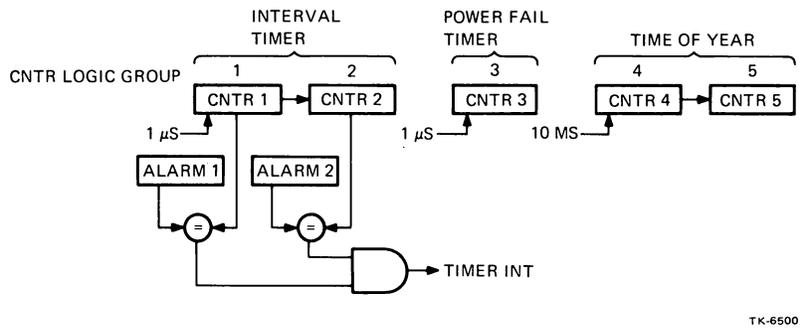
Figure 2-11   Utilization of 9513 Counter Logic Groups

2.4.1 9513 Register Addressing

Bit maps and I/O addresses for the registers in the 9513 accessible by the 8085A are shown in Figure 2-12. When the 9513 is referenced, the chip select (CS) input and either the read or write (RD or WR) input is asserted. Also, either the control or data bus port is selected, depending on the state of the control not/data (C-/D) input. Registers accessed via the 9513's control port are the command and status registers. All other addressable registers (as well as the status register) may be accessed via the 9513's data port.
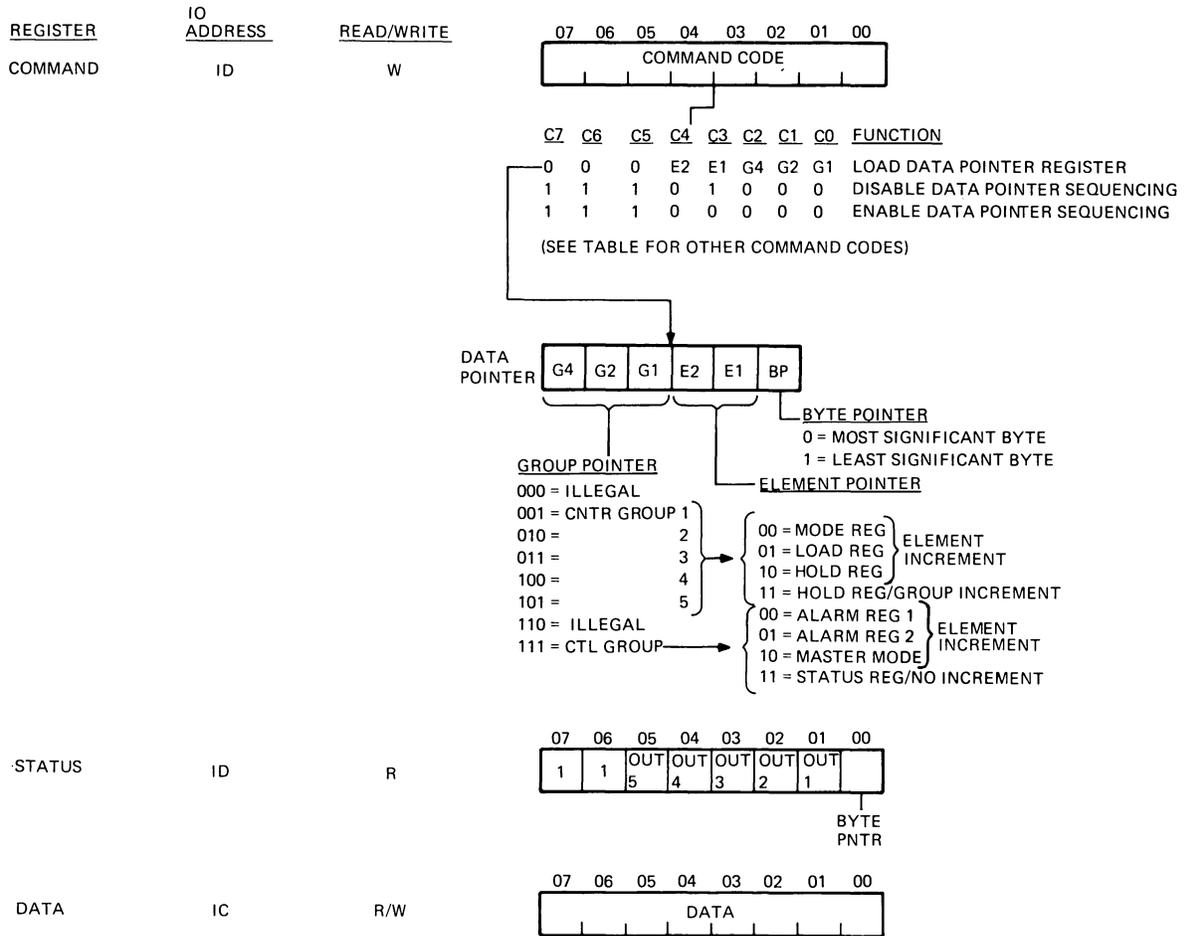
Figure 2-12   Bit Formats for 9513 Interval Timer Registers
(Sheet 1 of 2)

An 8085A I/O write operation to the 9513's control port address loads the command register. An I/O read to the control port address reads the status register. The register accessed by an I/O read or write to the data port address is determined by a previously loaded 6-bit data pointer internal to the 9513.

The data pointer consists of a 3-bit group pointer, a 2-bit element pointer, and a 1-bit byte pointer as indicated on sheet 1 of Figure 2-12. It is loaded by writing the command register with the appropriate command code. (Command codes in the range 00 to 1F load the data pointer.) For example, to allow the load register in counter group 1 to be accessed, a command code is first loaded which sets the group pointer to a binary value of 001 and the element pointer to a binary value of 01 (command code equals 09).

The byte pointer, a single bit, is automatically set when the data pointer is loaded. If not in 16-bit mode (the 9513 in the console processor operates in 8-bit mode), the byte pointer toggles following a data port access so that the high-order 8-bit byte in the selected register may be read or written over the eight AD bus lines, after the read or write of the low-order 8-bit byte. All registers accessed via the data port (except for the command and status registers) are 16-bits wide.

REGISTER   BYTE

MASTER MODE   HI(BP=0)

07	06	05	04	03	02	01	00
	DP CTL			FOUT DIVIDER CTL			

SCLR CTL
* 0 = BINARY
  1 = BCD

DB WDTH
* 0 = 8–BIT BUS
  1 = 16–BIT BUS

* 0 = EN DATA PNTR INC   FOUT
  1 = DIS DATA PNTR INC   GATE
                               * 0 = FOUT ON
                                   1 = FOUT OFF

0000 = DIVIDE BY 16
0001 =   1
0010 =   2
  :   :
*1010 =   10
  :   :
1110 =   14
1111 =   15

MASTER MODE   LO(BP = 1)

07	06	05	04	03	02	01	00
FOUT SOURCE						TOD MODE	

CMP2 EN   CMP1 EN

* 0000 = F1 (OSC)
0001 – 0101 = SRC1–SRC5
0110 – 1010 = GATE 1 – GATE 5
1011 – 1111 = F1 – F5

* 00 = DISABLED
01 = ENABLED, /5
10 = ENABLED, /6
11 = ENABLED, /10

COUNTER MODE   HI (BP = 0)

07	06	05	04	03	02	01	00
GATING CONTROL			SOURCE SELECTION				

000 = NO GATING
001 = HI LEVEL (TCN–1)
010 = HI LEVEL (GATE N+1)
011 = HI LEVEL (GATE N–1)
100 = HI LEVEL (GATE N)
101 = LO LEVEL (GATE N)
110 = HI EDGE (GATE N)
111 = LO EDGE (GATE N)

0XXXX = ACTIVE HIGH SENSE
1XXXX = ACTIVE LOW SENSE
X0000 = TCN–1
X0001–X0101 = SRC1–SRC5
X0110–X1010 = GATE1–GATE5
X1011–X1111 = F1–F5

COUNTER MODE   LO (BP=1)

07	06	05	04	03	02	01	00
COUNT CONTROL					OUTPUT CONTROL		

0XXXX = DISABLE SPECIAL GATE
1XXXX = ENABLE SPECIAL GATE
X0XXX = RELOAD FROM LOAD
X1XXX = RELOAD FROM LOAD OR HOLD
XX0XX = COUNT ONCE
XX1XX = COUNT REPETITIVELY
XXX0X = BINARY COUNT
XXX1X = BCD COUNT
XXXX0 = COUNT DOWN
XXXX1 = COUNT UP

000 = INACTIVE, OUTPUT LO
001 = TERM CNT PLS, HI
010 = TOGGLE (DLYD), HI

011 = TOGGLE, HI
100 = INACTIVE, OUTPUT HI Z
101 = TERM CNT PLS, LO
110 = TOGGLE (DLYD), LO
111 = TOGGLE, LO

NOTE: AN ASTERISK (*) INDICATES MASTER MODE PARAMETERS USED IN CONSOLE
        PROCESSOR. COUNTER MODE PARAMETERS USED VARY FOR EACH
        COUNTER LOGIC GROUP.

TK-6489

Figure 2-12   Bit Formats 9513 Interval Timer Registers
(Sheet 2 of 2)

In addition to the automatic toggling of the byte pointer, the element and group pointers can be made to increment so that the entire data pointer will sequence through all register addresses in the counter logic groups. (Only the element pointer will sequence when accessing the control group via the data port.) When the element pointer is sequencing, it sequences only through the range 00 to 10. If the element pointer is loaded with a value of 11 for a counter logic group address, it causes a special sequencing mode. That is, only the group pointer increments, allowing only the hold registers in all logic groups to be accessed one after the other. When the group pointer indicates a control group address, then an element pointer value of 11 will point to the status register (no automatic increment).

2.4.2 9513 Control Registers

The command and status registers are accessed via the 9513's control port, as described previously. The various command codes that may be loaded into the command register (and their functions) are listed in Table 2-4. The bits that may be examined in the read-only status register include the byte pointer bit in the data pointer and five bits indicating the state of the counter logic group's output pins $\langle 05:01 \rangle$.

Table 2-4   9513 Command Code Summary

C7	C6	C5	C4	C3	C2	C1	C0	Function
0	0	0	E2	E1	G4	G2	G1	Load data pointer register with contents of E and G fields.
0	0	1	S5	S4	S3	S2	S1	Arm counting for the selected counters.
0	1	0	S5	S4	S3	S2	S1	Load contents of specified source into the selected counters.
0	1	1	S5	S4	S3	S2	S1	Load and arm the selected counters.
1	0	0	S5	S4	S3	S2	S1	Disarm and save selected counters.
1	0	1	S5	S4	S3	S2	S1	Save the selected counters in hold register.
1	1	0	S5	S4	S3	S2	S1	Disarm the selected counters.
1	1	1	0	1	N4	N2	N1	Set output bit n (n = 000 to 101).
1	1	1	0	0	N4	N2	N1	Clear output bit n (n = 000 to 101).
1	1	1	1	0	N4	N2	N1	Step counter n (n = 000 to 101).
1	1	1	0	1	0	0	0	Disable data pointer sequencing.*
1	1	1	0	1	1	1	0	Gate off FOUT.*
1	1	1	0	1	1	1	1	Enter 16-bit mode.*
1	1	1	0	0	0	0	0	Enable data pointer sequencing.*
1	1	1	0	0	1	1	0	Gate on FOUT.*
1	1	1	0	0	1	1	1	Enter 8-bit mode.*
1	1	1	1	1	1	1	1	Master reset.*

*Sets or clears appropriate bit in master mode register.

2-24

Other control registers in the 9513 include a mode register for each counter logic group and a single master mode register. These 16-bit registers are accessed via the data port, one byte at a time. Each of the five counter mode registers controls the gating, counting, source, and output select functions within its corresponding logic group. The master mode register controls those functions not controlled by the individual counter mode registers.

The gating control field in a counter mode register determines when the counter, if armed (enabled), is allowed to count. (A counter is armed by loading the appropriate command code in the command register.) Gating is mainly by the chip's five-gate input pins GATE⟨5:1⟩. For example, a gating control field of 101 allows counting to proceed only when the corresponding gate input (GATE n) is at a low level. Gating by a high level, high or low edge, or other gate input (GATE n+1 or GATE n-1) may also be specified. A field of 000 allows counting to proceed unconditionally, as long as the counter is armed and clocked.

The source control field in a counter mode register selects the clock input to the counter and the active edge (high or low) that is counted. The counter inputs that may be selected include any one of the chip's five source inputs SOURCE⟨5:1⟩ or any one of the gate inputs. Also, any one of the chip's internally-generated frequency scaler outputs (F⟨5:1⟩) may be selected as well as the terminal count of the adjacent lower-numbered counter (TCn-1). The TCn-1 option allows one 16-bit counter to be internally concatenated with another, in order to give a longer count capability. For example, counters 4 and 5 are configured together in the console processor to form the 32-bit time of year counter, with the count rippling from counter 4 to counter 5.

The other control bits in a counter mode register are the count control field and the output control field. The count control field specifies the type of count (i.e., up/down, binary/BCD, etc.). The output control field specifies the type of signal asserted by the counter logic group's output pin (such as, high or low terminal count pulse or toggle).

The master mode register controls the 16-bit frequency scaler, the 4-bit divider circuit, and other circuitry including the enables for the comparator circuits in logic groups 1 and 2. The output from the divider circuit is a chip output (FOUT).

The frequency scaler may be programmed to operate in binary or BCD. The outputs from the scaler (which is driven by the internal oscillator, which in turn is controlled by an external 1 kHz oscillator) is the oscillator frequency itself (F1) plus four scaled (divided) outputs (F2 through F5), as shown in Table 2-5. In the console processor, the scaler is programmed for binary operation but only the F1 (OSC) output is used. That is, the master mode register is set to select F1 as the input to the FOUT divider. Also, the divider is programmed to divide the F1 input by 10 to provide a FOUT frequency of 100 Hz. This 100 Hz signal is connected externally to the SOURCE 1 input pin and is used to clock the low-order time of year counter.

Table 2-5   9513 Frequency Scaler Ratios

Output	BCD Scaling	Binary Scaling
F1	OSC	OSC
F2	F1/10	F1/16
F3	F1/100	F1/256
F4	F1/1000	F1/4,096
F5	F1/10,000	F1/65,536

2.4.3 CPU Interval Timer (Counter Logic Groups 1 and 2)

The CPU interval timer is a 1 μs time base that can also generate processor interrupts at specified time intervals. As a time base, it can be used to make high-resolution elapsed time measurements. Its interrupt capability is used mainly for scheduling and accounting by the operating system.

Most of the circuitry for the CPU interval timer is contained in counter logic groups 1 and 2 of the 9513 chip. The two counters are programmed to form a single counter 32 bits wide that increments at a rate of 1 μs per count. Other logic external to the 9513 (shown in the upper half of Figure 2-13) generates a signal called TIMER INT when the timer reaches its specified count. The external logic also generates a timer reset pulse.

Instruction level control of the CPU interval timer is by the interval count register (ICR), the next interval count register (NICR), and the interval counter control and status register (ICCS). These registers (which may be accessed by the MTPR and/or MFPR instructions) are like the instruction level data and control/status registers that control USART data handling, in that they are pseudo registers implemented in the console program and the CPU microcode. Bit formats are shown in Figure 2-14.
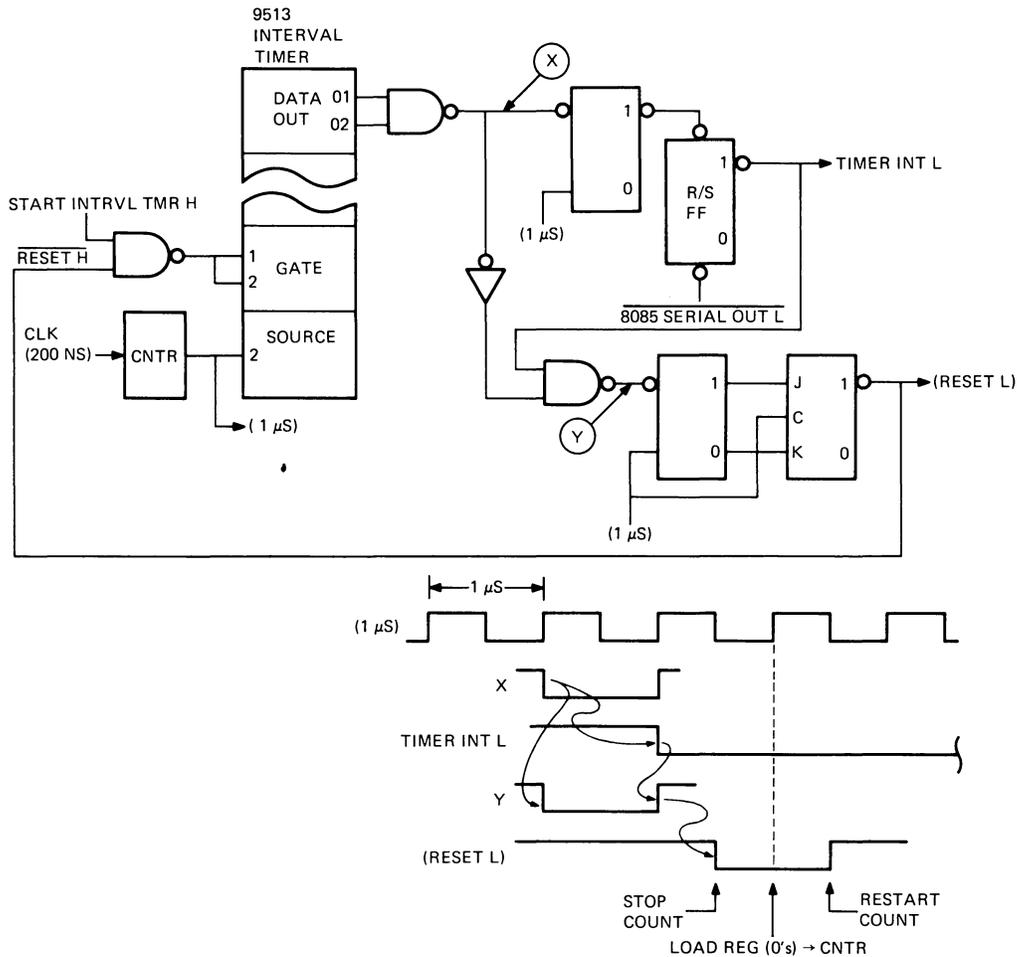
Figure 2-13 Interval Timer Control Logic

2-26
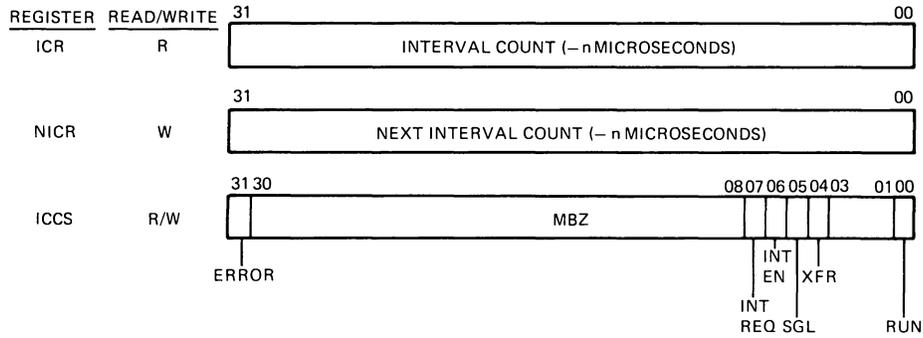
NOTES:
1. ERROR (BIT 31) AND INTERRUPT REQUEST (BIT 07) CLEARED BY WRITING A ONE TO CORRESPONDING BIT POSITION.

2. SINGLE CLOCK (BIT 05) AND TRANSFER (BIT 04) ARE WRITE-ONLY BITS AND ARE READ AS A ZERO.

TK-6510

Figure 2-14   Bit Formats for Interval Timer Control Registers

The program specifies an interval count by loading the negative count n into the NICR, where n is the period in microseconds. Ordinarily, the NICR value determines what the next interval count following the current count will be. However, the NICR value may be made the current count at any time by setting the XFR bit in the ICCS. This is how a current count value is loaded initially when the CPU interval timer is started.

The interval timer is started by setting the RUN bit in the ICCS. It then increments at a one μs rate until the RUN bit is cleared. (The current count may be determined at any time by reading the ICR.) Whenever RUN is cleared, the interval timer may be single-stepped by setting the SGL (single clock) bit in the ICCS.

When the end of the current count n occurs, the INT (interrupt) bit is set in the ICCS. Also, if or when the INT EN (interrupt enable) bit in the ICCS is set, a processor interrupt is generated at an IPL of 18 (HEX). The INT bit and the processor interrupt are cleared by clearing the INT EN bit in the ICCS or by writing a 1 bit into the INT bit position in the ICCS. If INT has not been cleared by the end of the next count, the ICCS's ERROR bit is set. The ERROR bit, like the INT bit, is cleared by writing a one bit into its bit position in the register.

Basic operation of the CPU interval timer with respect to its instruction level control registers is as follows. When the NICR or ICSS is loaded by an MTPR, the CPU microcode sends the register data, as part of a data packet to the console where it is stored in assigned RAM locations. (The microcode assembles the six ICCS control bits that are used into one data byte for the transfer and for storage by the console.) Also, if the NICR is loaded, the register data (four bytes) is complemented by the CPU microcode so that a positive count is stored by the console program.

When the ICCS is loaded and if the XFR bit in the register data is set, the console program loads the 32-bit positive count, previously stored in the RAM locations reserved for NICR data, into the two alarm registers in counter logic groups 1 and 2. (The hold registers and counters in the two logic groups are loaded with zeros during system initialization.) Also, when or if RUN is set in the ICCS, the console program asserts command register output START INTRVL TMR to start the count. The counters in logic groups 1 and 2 are programmed as shown in Table 2-6.

2-27

Table 2-6 Counter/Master Mode Selection for Counter
Logic Groups 1 and 2

| | Mode Selected | |
Register/Function	Counter 1	Counter 2
Counter Mode Register		
Gating control	Low level, gate 1	None
Source selection	Active high, source 2	Active high, TCn-1
Output control*	Active high, TC pulse	Active high, TC pulse
Count control	Enable special gate	Enable special gate
	Reload from load	Reload from load
	Count repetitively	Count repetitively
	Binary	Binary
	Up	Up
Master Mode Register		
Compare enables*	Enabled	Enabled
TOD mode	Disabled	Disabled

*Compare enables in master mode register override output control field in counter mode register to cause a logic group output whenever the counter is equal to the alarm register. However, the polarity of the output is still defined by the output control field.

START INTRVL TMR starts the count by driving the 9513's GATE 1 and 2 inputs low. Counter 1, the low-order 16 bits of the counter, then begins a binary up-count at the 1 μs clock rate. (The 1 mHz clock, which connects to the 9513's SOURCE 2 input, is the 5 mHz 8085A clock divided by five by a counter external to the chip.) Counter 2, the high-order 16 bits of the counter, is incremented by the terminal count from counter 1.

When the counters have reached a value equal to the alarm register contents, the comparator circuits within the 9513 cause both logic group outputs to be asserted, setting TIMER INT in the interval timer's external control logic. A reset pulse is also generated, which causes the chip's GATE 1 input to go high. This stops the count and loads counter 1 with zeros from its load register. Counter 1 then begins incrementing again when the reset signal goes false. Timing is shown in the lower half of Figure 2-13.

When the console program detects the assertion of TIMER INT, it clears counter 2 by loading zeros from its load register (this counter is not cleared by the reset pulse), and it generates a processor interrupt request by asserting command register output INTRVL TIM INT if the INT EN bit is set in the ICCS. The console program also reloads the alarm registers if the NICR data has been changed since the last load, and it sets the INT bit in the ICCS unless it is already set, in which case it sets the ERROR bit.

After the INT (or ERROR) flag is set, the console program clears the TIMER INT flip-flop in the external control logic by negating the 8085A's serial output. The interval timer will continue to increment setting TIMER INT at the end of each specified count until the RUN bit in the ICCS is cleared. Whenever RUN is cleared, setting the SGL bit in the ICCS causes the console program to load a command code in the 9513 that single-clocks counter 1, incrementing the interval count by one.

<div align="center">

NOTE

During normal operation, the count in the NICR is decremented by three before it is loaded in the alarm registers. This is to compensate for the inherent external logic delay in clearing the low-order counter. (The undecremented NICR value is loaded when singlestepping the counter.) Also, the software delay in clearing the high-order counter prevents proper operation for very small time intervals. (See system specifications for exact values.)

</div>

Processor interrupts by the interval timer are not only generated when a time-out occurs with INT EN previously set in the ICCS, but also during a count if the INT bit is set in the ICCS and the ICCS is reloaded with INT EN = 1. The processor interrupt signal is cleared by the console program whenever the ICCS is loaded with INT = 1 or INT EN = 0.

When the ICCS or the ICR is read with an MFPR, the register data is transferred (in a data packet) from the console program to the CPU microcode. If the ICCS is referenced, the CPU microcode must unpack the one byte of register data (containing the six ICCS control bits) stored by the console program into the proper 32-bit format for transfer by the MFPR. If the ICR is referenced, the console program gets the current interval timer count by loading counters 1 and 2 into their respective hold registers.

The hold register data, a positive count, is then read out of the 9513 chip and sent to the CPU microcode (four bytes), where it must be converted to a negative count for transfer by the MFPR. To make the conversion, the microcode adds the positive count supplied by the console program to the last NICR value. This value, a negative count, is saved by the microcode every time the NICR is written by a MTPR and before it is converted to a positive count for storage by the console program.

2.4.4 Time of Year Clock (Counter Logic Groups 4 and 5)

The time of year clock, which is configured from counter logic groups 4 and 5 in the 9513, is a 32-bit binary up-counter that increments at a rate of 10 ms per count. It has no interrupt facility, acting only as a long-term elapsed time indicator for the operating system.

The counter mode registers for counter logic groups 4 and 5 are programmed as shown in Table 2-7. The 10 ms time of year clock source, which is the output from the 9513's FOUT pin and is generated by the frequency scaler from an external 1 kHz oscillator (Paragraph 2.4.2), connects to the 100 Hz stall latch. This latch allows the 8085 to stall a clock edge while the time of year clock is being accessed (read or write). The output of the 100 Hz stall latch connects to the SOURCE 1 pin and clocks counter 4. This counter provides the low-order 16 bits of the time of year count. Counter 5, the high-order 16 bits of the count, is clocked by the terminal count from counter 4.

Table 2-7 Counter Mode Selection for Counter Logic Groups 4 and 5

| Function | Mode Selected | |
	Counter 4	Counter 5
Gating control	None	None
Source selection	Active high, source 1	Active high, TCn-1
Output control	Inactive	Inactive
Count control	Disable special gate	Disable special gate
	Reload from load	Reload from load
	Count repetitively	Count repetitively
	Binary	Binary
	Up	Up

The time of year counter may be accessed by the CPU program at any time by referencing the time of year register (Figure 2-15) with an MTPR or MFPR. The register data is passed between the CPU microcode and the console program in data packets as when accessing other console-based processor registers such as NICR and ICR.

```
REGISTER  READ/WRITE  31                                                      00
  TODR       R/W       ┌──────────────────────────────────────────────────────┐
                       │                  TIME OF YEAR COUNT                   │
                       └──────────────────────────────────────────────────────┘
                                                                        TK-6509
```
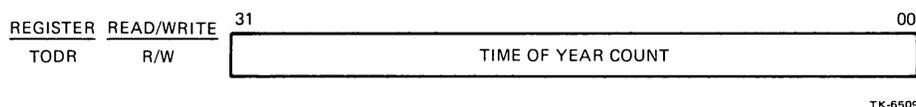
Figure 2-15 Time of Year Register Bit Format

When the time of year register is written with an MTPR, the console program takes the unsigned 32-bit binary count sent by the CPU microcode (four bytes) and writes it into the load registers for counters 4 and 5. It then loads the counters from the load registers.

When the time of year register is read by an MFPR, the console program loads the current count in counters 4 and 5 into their respective hold registers, reads the hold registers, and then sends the count to the CPU microcode (four bytes).

There is no instruction level on/off control for the time of year clock. Once the counter mode registers are programmed by the console program, the counters will increment continuously, as long as power is applied to the 9513 chip, the 1 kHz oscillator chip, and the chip containing the oscillator's output gate.

2-30

At system power-up, the console program checks the counter mode registers for counters 4 and 5 to verify that they are set correctly. If not, the time of year count (even though active during the power down period due to standby power backup) is assumed to be inaccurate and the console program stops the count, clears the counters, and sets the mode registers to their correct value. It then restarts the count from zero.

2.4.5 Power Fail Timer (Counter Logic Group 3)

Counter logic group 3 is reserved for use by the console program as a 2 ms timer. It is configured as a binary up-counter and uses the same clock source as the CPU interval timer, causing it to increment at a rate of 1 μs per count. The counter mode register is programmed as shown in Table 2-8.

Table 2-8 Counter Mode Selection for Counter Logic Group 3

Function	Mode Selected – Counter 3
Gating control	None
Source selection	Active high, source 2
Output control	Active low, TC pulse
Count control	Disable special gate
	Reload from load
	Count repetitively
	Binary
	Up

The power-fail timer is used when a system power-fail condition has been detected. (UNIBUS AC LOW asserts SEE AC LO, which interrupts the 8085A at its RST 7.5 input.) When a power-fail occurs, the console program asserts AC LO (to eliminate a transient condition). The 8085A checks a flag in console RAM. The flag indicates whether the 8085A was in program mode or console I/O mode when AC LO was asserted. If it was in console I/O module, the 8085A goes to its power-up routine (described below). However, if it was in program mode, the 8085A asserts the power-fail interrupt (level 1E) to the CPU, starts the power-fail timer (2 millisec), and goes back to the normal loop for program mode.

The CPU handles the power-fail interrupt very soon (due to high priority level of interrupt, 1E) and goes to the operating system power-fail routine. The power-fail routine ceases normal operation, saves all necessary information, and performs a branch self instruction (forever).

Meanwhile, the console processor is in its program mode loop. The 8085A checks the power-fail timer in this loop and will eventually sense the end of the two-millisecond period. The 8085A will then stop the CPU clock, disable main memory references, and go to its power-up routine. The 8085A will stay in this routine until power goes away completely (main ac power loss or keyswitch to off) or until ac power returns (keyswitch from STD BY to an on position). If ac power returns, the 8085A will attempt a restart if the AUTO-RESTART/BOOT switch is in the on position (to the left).

2.5 CONSOLE READ/WRITE OPERATIONS

As stated previously, 8085A operation, as controlled by the executing console program, consists essentially of a series of read/write operations to the various console processor devices, both memory and I/O. The memory devices are the ROM and the RAM. All other addressable console processor devices (such as the USARTs and 9513 interval timer) are I/O devices.

The general types of memory and I/O read/write operations are listed in Table 2-9. Addresses are also indicated. As can be seen, memory addresses are 16 bits, with the 6K ROM having addresses in the range of 0000 to 17FF, and the 16K RAM having addresses in the range 4000 to 7FFF. I/O addresses are eight bits, ranging from 00 to a maximum value of EC. The 6K ROM is the maximum configuration; the basic configuration is 4K.

Table 2-9 Console Read/Write Operations

Transfer Type	Address	Operation
Memory	0000-17FF	Read ROM
Memory	4000-7FFF	Read/write RAM
I/O	00-07	Read switch and power status
I/O	08-0A	Write control store write register
I/O	1C, 1D	Read/write interval timer registers
I/O	20	Read ready bit summary register
I/O	20-2F	Write console command register 0
I/O	30-3F	Write console command register 1
I/O	40-4B	Read/write USART registers
I/O	80-87	Read console read register, CPU status
I/O	A0-AF	Write console command register 2
I/O	CC	Write console write register
I/O	EC	Write console read register from Y bus

2.5.1 Read/Write Control Logic

The control logic associated with console read/write operations consists of decode logic that generates a number of read/write select levels, and miscellaneous control logic required for RAM and USART read/write operations. The RAM and USART control logic is detailed in Figure 2-16.
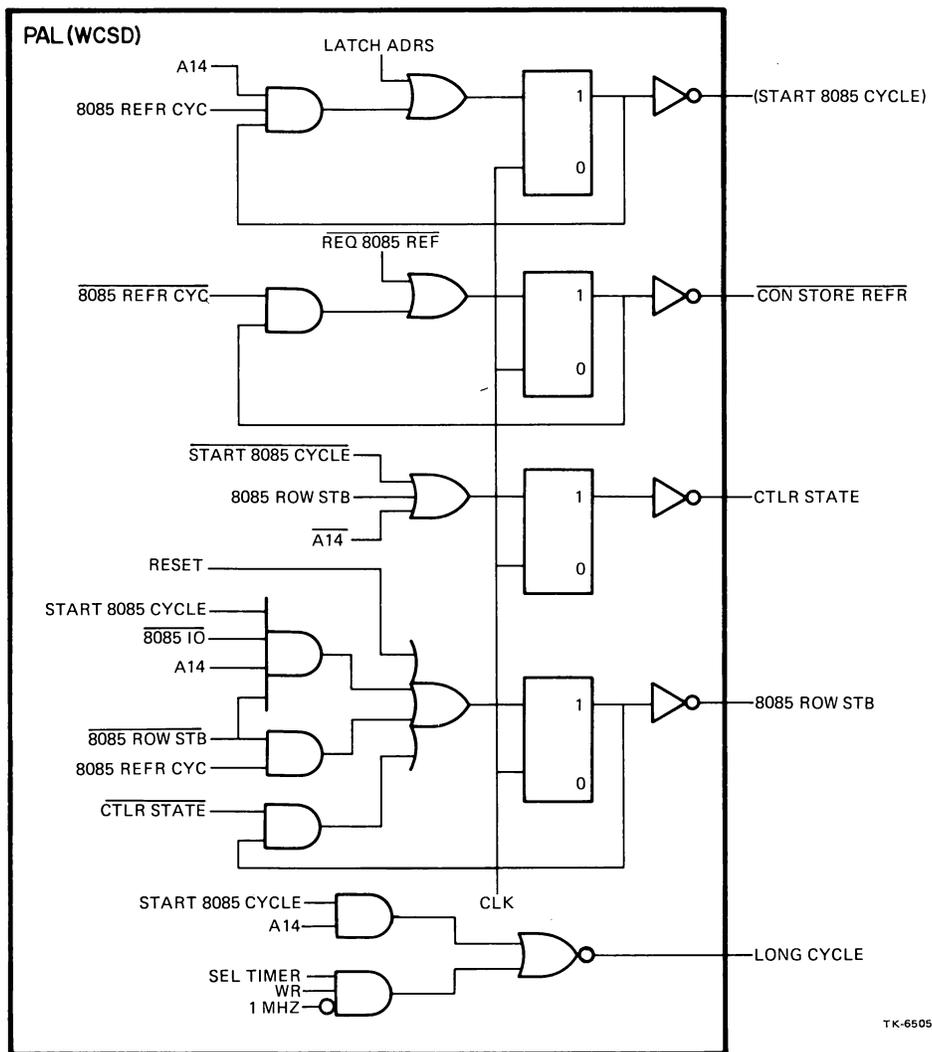
Figure 2-16   RAM/USART Control Logic (Sheet 1 of 3)
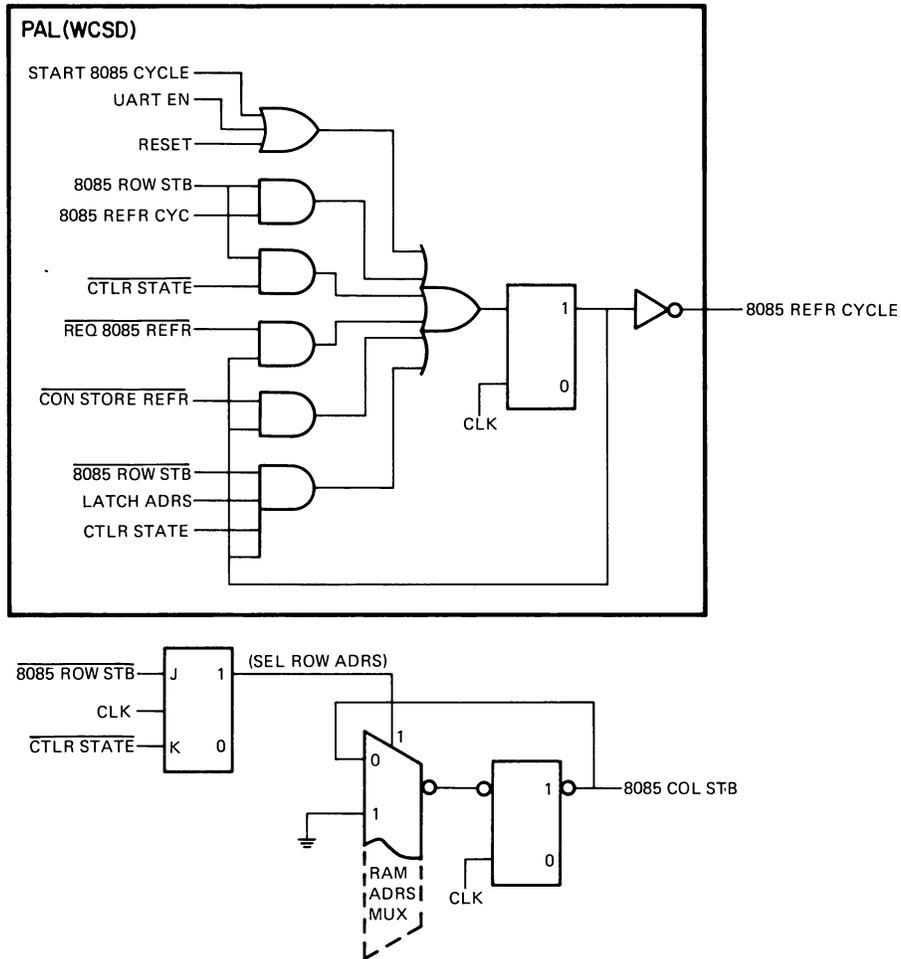
Figure 2-16   RAM/USART Control Logic (Sheet 2 of 3)

The majority of the console's read/write select levels (Table 2-10) are derived from the device address (on the A lines) and the control signals (RD, WR, 8085 IO) asserted by the 8085A during its machine cycle. For example, SEL TIMER is generated for an I/O reference (8085 IO = 1) to address 1C or 1D (A⟨15:11⟩ = 00011); that is, when the 9513 interval timer's control or data port is addressed. (SEL TIMER asserts the 9513's chip select level.) The various select levels and the RAM/USART control logic are discussed in Paragraphs 2.5.2 through 2.5.4.
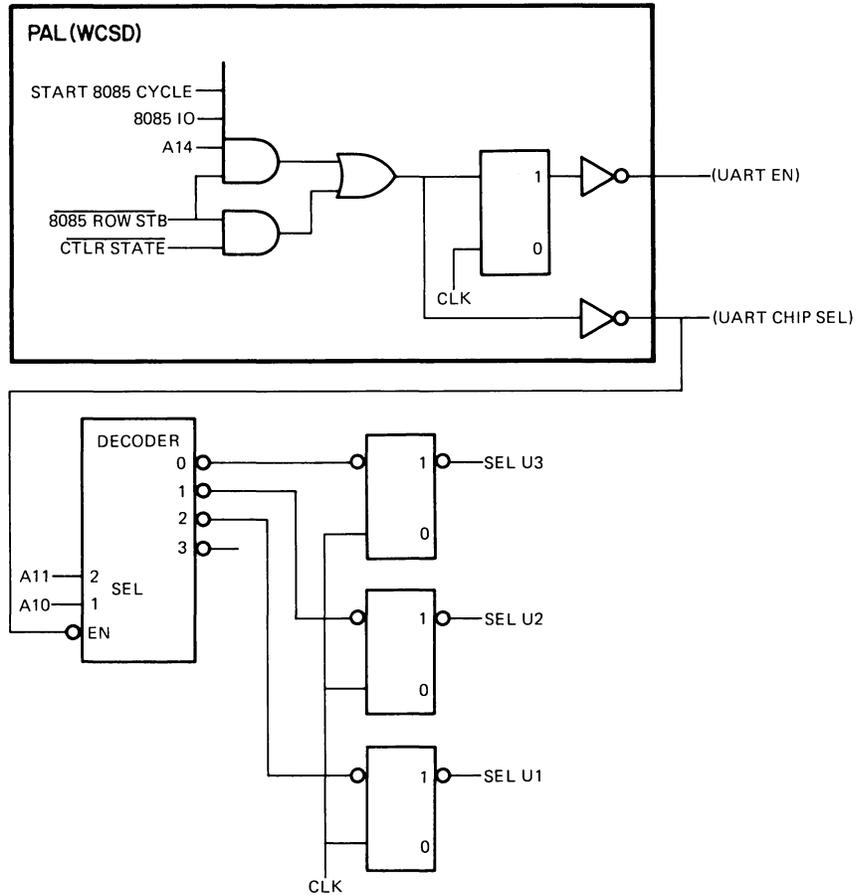
TK-6507

Figure 2-16   RAM/USART Control Logic (Sheet 3 of 3)

Table 2-10   Console Read/Write Select Levels
(Generated from A Lines)

8085 IO	RD	WR	A15	A14	A13	A12	A11	A10	A09	A08	Read/Write Select Signal
0	1	–	0	0	0	–	–	–	–	–	SEL ROM MUX
1	1	–	0	0	1	0	0	–	–	–	SEL ROM MUX
1	1	–	0	0	0	0	0	–	–	–	SEL STATUS
1	–	1	0	0	0	0	1	–	–	–	WRITE WWD REG
1	–	–	0	–	0	1	1	–	–	–	SEL TIMER
1	–	1	0	0	1	0	–	–	–	–	WRITE M0
1	–	1	0	0	1	1	–	–	–	–	WRITE M1
1	1	–	1	0	0	0	0	–	–	–	SEL CPU REGS
1	1	–	1	0	0	0	0	–	–	–	(DIR)
1	–	1	1	–	–	–	–	–	–	–	SEL CPU REGS

2.5.2 ROM Operations

The 4K (or 6K) × 8-bit ROM consists of four (or six) 2K × 4-bit read-only memory chips. The chips are configured into two (or three) 2K × 8-bit banks of two chips each. During an 8085A memory reference, the two most significant bits of ROM address on A lines 12 and 11 select a bank. The rest of the address bits on the A lines and the AD bus select a location within the selected bank.

A Lines	AD Bus	Addressing Function
⟨12:11⟩		Select 1 of 3 possible banks
⟨10:08⟩	⟨7:0⟩	Select 1 of 2K possible locations within the selected bank.

The AD bus lines do not address the ROM directly as shown on the console processor block diagram, Figure 2-1. Because address and data information is multiplexed on the AD bus, the eight low-order memory address bits which are asserted at the beginning of the 8085A machine cycle are stored in a latch circuit. (Actually, the buffered AD bus outputs, CONS DATA ⟨7:0⟩, connect to the latch.) The latch circuit outputs, which connect to the ROM chips, then assert the low order memory address for the entire machine cycle.

Timing for the ROM read operation is shown in Figure 2-17. The 8085A's ALE output (LATCH ADRS) is used to latch the low-order memory address on the AD bus into the latch circuit. After the 8085A has negated the bus address bits, SEL ROM MUX (a read select level) is asserted to gate the ROM data outputs through a multiplexer and onto the AD bus, where they are read by the 8085A. The SEL ROM MUX signal, generated for all ROM addresses when the 8085A's RD signal is asserted (Table 2-10), enables only the multiplexer. The appropriate multiplexer inputs (the ready bit summary register is also read through this multiplexer) are selected by A line 13, which is a zero for all ROM references.
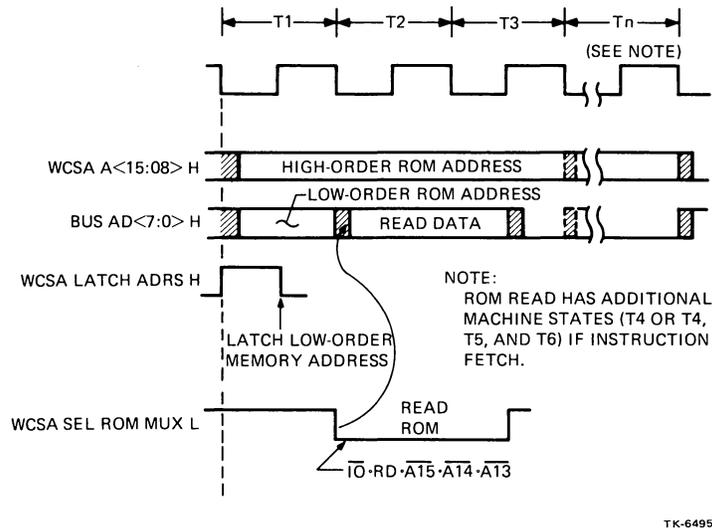
Figure 2-17   8085A ROM Read Operation Timing Diagram

2.5.3 RAM Operations

The 16K × 8-bit RAM in the console processor consists of eight parallel-connected 16K × 1-bit MOS chips. The MOS chips are dynamic RAMs and must be refreshed periodically.

The 16K (128 row × 128 column) RAM chips have seven address lines. During a RAM read or write operation, a 7-bit row address is transmitted on the RAM's address lines, followed by a 7-bit column address. The row and column addresses are generated by multiplexing the memory address asserted by the 8085A on the A lines and the AD bus. (The latched AD bus memory address bits are used as for ROM addressing.) The even-numbered address bits on the A lines and AD bus select a row in the RAM chips. The odd-numbered address bits select a column.

A Lines	AD Bus	Addressing Function
13,11,09	7,5,3,1	Select 1 of 128 possible row addresses
12,10,08	6,4,2,0	Select 1 of 128 possible column addresses

Timing for a RAM read or write operation is shown in Figure 2-18. One more machine state, a wait state (TW), is required to access the dynamic RAM chips, than when accessing the ROM chips. The wait state occurs when the 8085A's READY input is negated by LONG CYCLE asserted in the RAM/USART control logic (Figure 2-16, sheet 1). The LONG CYCLE signal is asserted when A line 14 is a 1 (as it is for RAM addresses), and when START 8085 CYCLE has been set by LATCH ADRS, which occurs at the beginning of every 8085A machine cycle.
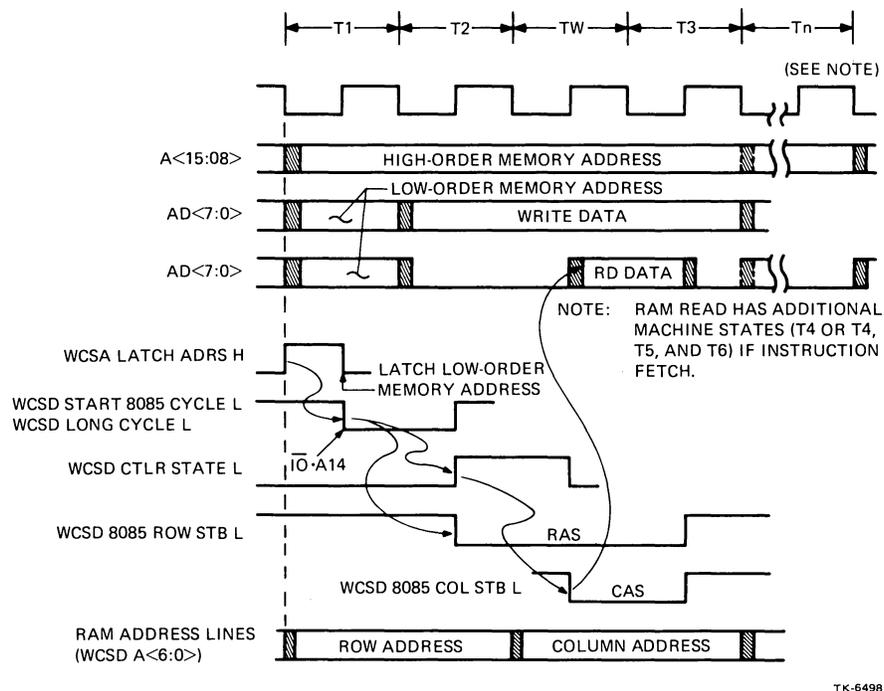
Figure 2-18   8085A RAM Read/Write Operations Timing Diagram

For a RAM address, START 8085 CYCLE also generates the first of two address strobes that load the row and column address into the dynamic RAM chips. The first strobe, 8085 ROW STB, is a flip-flop set directly by START 8085 CYCLE. The second strobe, 8085 COL STB, is a flip-flop held in the off state as long as a row address is being gated through the RAM address multiplexer. The row address is selected at the multiplexer inputs by a normally-asserted J-K flip-flop (Figure 2-16, sheet 2). Once 8085 ROW STB is asserted, the J-K flip-flop changes state to gate the column address through the multiplexer and allow 8085 COL STB to set.

When the column address strobe is generated for a RAM read operation, the data in the RAM location selected by the row and column addresses is transmitted by the RAM chips onto the AD bus and read by the 8085A. For a RAM write operation, the column strobe causes the write data asserted by the 8085A on the AD bus to be loaded into the selected RAM location. A write operation is specified by 8085 WRITE CYC, which is derived from the status lines asserted by the 8085A throughout the machine cycle. This signal connects directly to the RAM chip's write enable inputs.

To prevent loss of stored data, each storage cell in a dynamic RAM chip, such as those used in the console processor's RAM, must be recharged (refreshed) at least once during a specified time interval called the refresh interval (i.e.; specified at 2 ms or less for the 16K chips in the console processor). A single row address strobe refreshes all cells in a row address. Thus, the storage cells for all chips in the console processor's RAM (128 row $\times$ 128 column chips connected in parallel) may be refreshed by 128 strobes to all the possible row addresses. A refresh cycle is initiated every 12.8 μs, giving a refresh interval of approximately 1.7 ms.

RAM refresh timing is shown in Figure 2-19. A refresh cycle is initiated by REQ 8085 REF, which is one of the outputs from the free-running counter used to generate the baud clocks for the USARTS (Paragraph 2.2.3). REQ 8085 REF has a 12.8 μs period, and its positive-going edge sets 8085 REFR CYC in the RAM/USART control logic.

Once 8085 REFR CYC is set, it disables the RAM address multiplexer, and it enables the outputs of a 7-bit binary (wrap-around) up-counter, called the refresh counter, to drive the RAM chip's address lines. 8085 REFR CYC then generates a row address strobe, refreshing the row addressed by the refresh counter's current value. The trailing edge of 8085 REFR CYC increments the refresh counter so that it steps through all possible row addresses as refresh cycles continue to be initiated at the 12.8 μs rate.

When a refresh request is generated and a RAM read/write operation is in progress, the setting of 8085 REFR CYC and the resulting refresh cycle is delayed until the read/write completes.

NOTE
A RAM refresh is also delayed if a USART read/write operation is in progress. (See Paragraph 2.5.1.)

If a refresh request is generated (or pending) when a RAM read/write is started, the refresh is done with START 8085 CYCLE remaining set and asserting LONG CYCLE long enough to cause an extra wait state in the machine cycle. A normal read/write operation takes place immediately after the refresh cycle. Timing for the extended machine cycle is shown in the lower part of Figure 2-19.

2-38
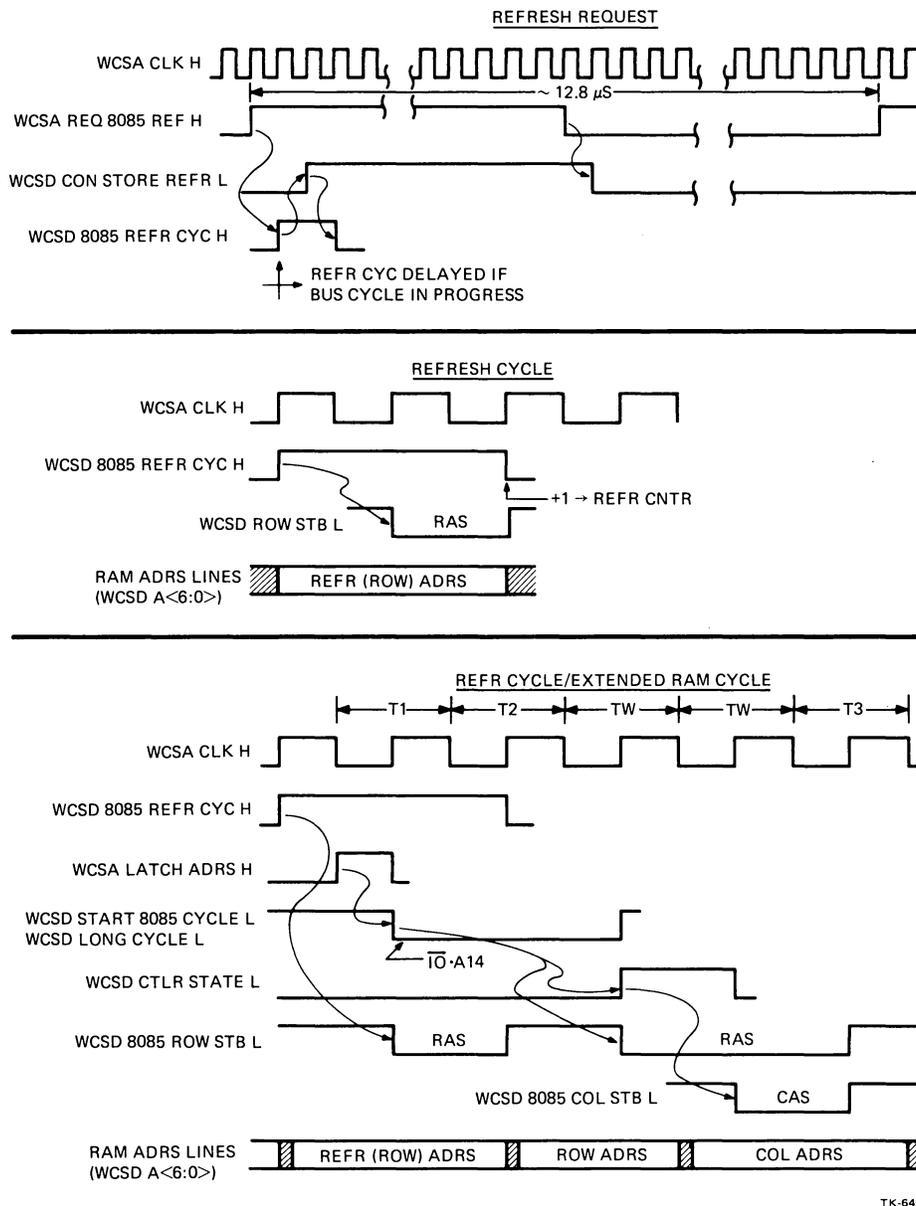
Figure 2-19   8085A RAM Refresh Operation Timing Diagram

A synchronizing flip-flop is used to allow 8085 REFR CYC to set and then cleared just after (and only after) the positive-going edge of the 12.8 μs REQ 8085 REFR signal. The flip-flop, which is set by 8085 REFR CYC and cleared when REQ 8085 REF goes false, is called CON STORE REFR and is also used to initiate the refresh cycle in basic control store. The control store's refresh cycle (Paragraph 4.3.5) is initiated by the negative-going edge of CON STORE REFR, at the same 12.8 μs rate as the console processor's refresh cycle.

2-39

2.5.4 I/O Operations

The control and data bits written and read for the range of I/O addresses in the console processor are shown in Figure 2-20.

The USART's control and data registers (I/O addresses 40 through 4B) and the 9513 interval timer's control and data registers (1C and 1D) have been discussed. Other registers include the console read and write registers (80 and CC), the control store write register (08 through 0A), and the ready bit summary register (20). In addition, individual bits in the console processor's command registers may be asserted or negated one at a time (20 through 3F, and A0 through AF), and one or two status bits may be read at a time via the console processor's input data multiplexers (00 through 07, and 82 through 87).

IO ADDRESS	READ/WRITE	07 00
00	R	μPC SHF OUT (μPC <14>)
01	R	MISC IN 2 ... BOOT SW
02	R	UNIBUS DC LOW ... HALT SW
03	R	DISABLE CTL-P
04	R	CSL SELF TEST EN SW ... APT
05	R	CPU +5 V OK ... REM S CARR (N.U.)
06	R	SLOW CLK (100 HZCP) ... REMOTE SW
07	R	CPU ± 15 V OK ... REM RING IND (N.U.)
08	W	CS WRITE REGISTER <07:00>
09	W	CS WRITE REGISTER <15:08>
0A	W	CS WRITE REGISTER <23:16>
0B-1B	–	NOT USED
1C	R/W	INTERVAL TIMER DATA
1D	R/W	INTERVAL TIMER CONTROL (COMMAND/STATUS)
1E, 1F	–	NOT USED
20	R	READY BIT SUMMARY REGISTER

READY BITS: 07 06 05 04 03 02 01 00

- PWR FAIL TIME-OUT
- TU58 RX RDY (DONE)
- TU58 TX RDY
- LOCAL TX RDY
- TIMER INT
- REMOTE RX RDY (DONE)
- LOCAL RX RDY (DONE)
- REMOTE TX RDY

TK-6484

Figure 2-20   8085A I/O Space (Sheet 1 of 4)

2-40

IO ADDRESS   READ/WRITE

IO ADDRESS	READ/WRITE	07 00
20	W	CLEAR CPU RUN
21	W	SET CPU RUN
22	W	CLR CPU SSTP (CLR CPU CLKS)
23	W	SET CPU SSTP (SET CPU CLKS)
24	W	CLR CSR SSTP (CLR CSR CLK)
25	W	SET CSR SSTP (SET CSR CLK)
26	W	CLR μPC SSTP (CLR UPC CLK)
27	W	SET μPC SSTP (SET UPC CLK)
28	W	CLR MCT CTL 0 (NOT USED)
29	W	SET MCT CTL 0 (NOT USED)
2A	W	CLR MCT CTL 1 (NOT USED)
2B	W	SET MCT CTL 1 (NOT USED)
2C	W	CLR START INTRVL TMR (STOP TIMER)
2D	W	SET START INTRVL TMR (START TIMER)
2E	W	CLR UNIBUS BBSY
2F	W	SET UNIBUS BBSY
30	W	SET UNIBUS DC LO
31	W	CLR UNIBUS DC LO
32	W	SET UNIBUS AC LO
33	W	CLR UNIBUS AC LO
34	W	CLR CINIT (UNIBUS INIT)
35	W	SET CINIT (UNIBUS INIT)
36	W	CLR WRITE WCS (END CS WRITE)
37	W	SET WRITE WCS (START CS WRITE)

07   00

TK-6486

Figure 2-20   8085A I/O Space (Sheet 2 of 4)

IO ADDRESS	READ/WRITE	07 00
38	W	CLR CSR SHF IN
39	W	SET CSR SHF IN
3A	W	CLR 100 HZ STALL
3B	W	SET 100 HZ STALL
3C	W	CLR RUN (TURN OFF RUN LIGHT)
3D	W	SET RUN (TURN ON RUN LIGHT)
3E	W	CLR AUTOTEST ACT (TURN OFF REMOTE LIGHT)
3F	W	SET AUTOTEST ACT (TURN ON REMOTE LIGHT)
40	R/W	USART 3 (REMOTE) DATA REGISTER
41	R	USART 3 (REMOTE) STATUS REGISTER
42	R/W	USART 3 (REMOTE) MODE REGISTERS
43	R/W	USART 3 (REMOTE) COMMAND REGISTER
44	R/W	USART 2 (TU58) DATA REGISTER
45	R	USART 2 (TU58) STATUS REGISTER
46	R/W	USART 2 (TU58) MODE REGISTERS
47	R/W	USART 2 (TU58) COMMAND REGISTER
48	R/W	USART 1 (LOCAL) DATA REGISTER
49	R	USART 1 (LOCAL) STATUS REGISTER
4A	R/W	USART 1 (LOCAL) MODE REGISTERS
4B	R/W	USART 1 (LOCAL) COMMAND REGISTER
4C-7F	—	NOT USED
80	R	CONSOLE READ REGISTER
81	—	NOT USED
82	R	CPU ACK
83	R	CPU ATTN

07   00

TK-6485

Figure 2-20   8085A I/O Space (Sheet 3 of 4)

IO ADDRESS	READ/WRITE	07 00
84	R	UPC SHF OUT (UPC <14>)
85	R	CSR <7>
86	R	CSR <15>
87	R	CSR <23>
88-9F	–	NOT USED
A0	W	SET HALT ON PE (ENABLE STALL ON PAR ERR)
A1	W	CLR HALT ON PE (DISABLE STALL ON PAR ERR)
A2	W	CLR PARAL LD CSR (SET CSR SHIFT MODE)
A3	W	SET PARAL LD CSR (SET CSR NORMAL MODE)
A4	W	SET INTRVL TIM INT
A5	W	CLR INTRVL TIM INT
A6	W	SET EN MEMORY REF (ENABLE MEM REQ)
A7	W	CLR EN MEMORY REF (DISABLE MEM REQ)
A8	W	SET CONS ACK
A9	W	CLR CONS ACK
AA	W	SET CONS ATTN
AB	W	CLR CONS ATTN
AC	W	SET PWR FAIL INT
AD	W	CLR PWR FAIL INT
AE	W	SET CONS HALT
AF	W	CLR CONS HALT
B0-CB	–	NOT USED
CC	W	CONSOLE WRITE REGISTER
CD-EB	–	NOT USED
EC	W	LOAD CONSOLE READ REG FROM Y BUS
ED-FF	–	NOT USED

07   00

TK-6487

Figure 2-20   8085A I/O Space (Sheet 4 of 4)

Timing for the read/operations made to the various I/O addresses is given in Figure 2-21. The I/O address is transmitted by the 8085A on the A lines and also on the multiplexed AD bus at the beginning of the machine cycle. The A lines connect to most of the I/O devices. However, the devices on the CPU's DAP module are accessed over the console bus, which is an extension of the AD bus. Thus, similar to low-order memory addressing, the AD bus address asserted at the beginning of the machine cycle is stored in a latch circuit by the LATCH ADRS signal. The latch circuit outputs can then be used to address the I/O devices on the DAP module for an entire I/O read or write cycle.

After the 8085A asserts the I/O address and LATCH ADRS (and the 8085 I/O signal), the console processor's read/write control logic causes the addressed I/O device to either transmit read data onto the AD bus (where it is read by the 8085A), or to load write data from the AD bus.

2.5.4.1 Reading and Writing the USART Registers – Timing for an I/O read or write to a USART register is shown in the upper half of Figure 2-21. Like RAM addresses, USART I/O addresses have A line 14 equal to 1, and START 8085 CYCLE asserts LONG CYCLE to cause a wait state during the machine cycle.

START 8085 CYCLE also asserts a decoder enable level, and the decoder's output sets one of three flip-flops (SEL U1, U2, or U3). (Refer to Figure 2-16, sheet 3.) The flip-flop that is set depends on which of the three USARTS in the console processor is referenced by the I/O address bits on A lines 11 and 10. The flip-flop's outputs connect directly to the USART's chip enable inputs.
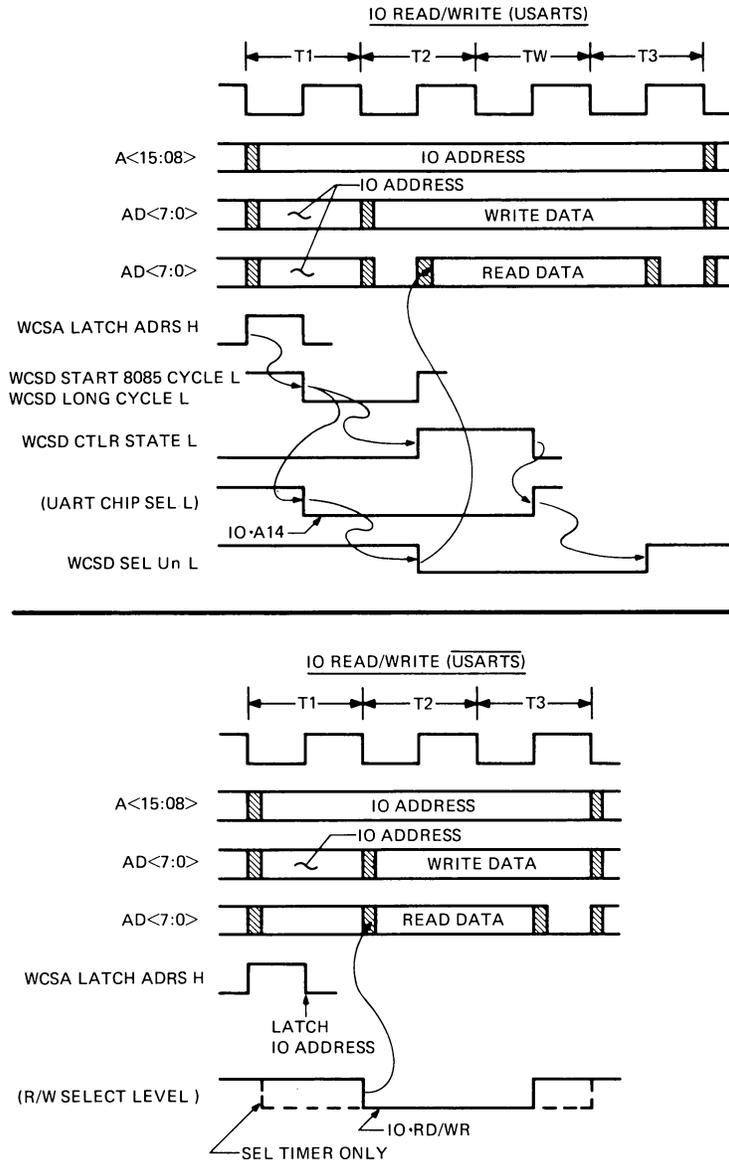
Figure 2-21   8085A I/O Read/Write Operations Timing Diagram

A lines 09 and 08 also connect directly to the USARTs. These I/O address bits specify the register to be accessed. When the chip enable line is asserted, the addressed register in the enabled USART is either written from, or read onto, the AD bus. As for RAM references, 8085 WRT CYC (which connects to the USART's write enable inputs) specifies the type of operation.

Because some of the read/write control logic is used for both USART and RAM operations (e.g.; LONG CYCLE), a RAM refresh request may extend a USART reference. As for an extended RAM cycle, the machine cycle is extended by one wait state when the refresh request occurs at the beginning of the USART read/write operation. Also, a RAM refresh will be delayed if the refresh request occurs when a USART read/write is in progress.

2.5.4.2 Reading and Writing the Other I/O Devices – Read/Write select levels for I/O devices other than the USARTs are included in Table 2-10. With one exception, the select levels are asserted when the 8085A generates either RD or WR during the machine cycle. (The leading edge of RD and the trailing edge of WR define when data is to be transferred to and from the AD bus.) The exception is SEL TIMER, the chip select level for the 9513 interval timer. It is asserted by 8085 I/O at the beginning of the machine cycle. However, the RD and WR signals connect directly to the 9513 chip to ensure correct timing for the I/O data transfer.

Two of the select levels generated for I/O read operations are SEL STATUS and SEL ROM MUX. (The latter is also asserted for a ROM read.) SEL STATUS allows the panel switches, the self-test switch, the power monitoring levels, and several other status bits to be read by a pair of input data multiplexers. Two status bits are read at a time, with one multiplexer connecting to AD bus line 7 and the other to AD bus line 0. (A lines 10 through 08 select the status bits at the multiplexer inputs.)

SEL ROM MUX is asserted during an I/O read of the ready bit summary register. It causes the USART's ready outputs plus the timer interrupt and power fail time-out flags to be gated onto the AD bus through the ROM multiplexer. A line 13, which is equal to 1 when reading the ready bit summary register, deselects the ROM outputs and selects the status bits at the ROM multiplexer inputs.

An I/O write to the control store write register asserts WRITE WWD REG. One byte in the 24-bit register is written at a time, with each 8-bit section of the register having its own separate clock. That is, WRITE WWD REG enables a decoder, causing one of its outputs (one of three used as the register clocks) to load the addressed byte. The decoder output asserted is selected by the I/O address bits on A lines 09 and 08. The register is loaded from the AD bus by the trailing edge of the decoder's output.

The WRITE M0 and M1 select levels cause a single bit to be written in command registers 0 and 1, respectively. When a command register (which is an 8-bit addressable latch circuit) is enabled by a select level, the output addressed by its select inputs is either asserted or negated, depending upon the state of its data input. The other outputs are not changed.

All outputs are latched to their current state when the select (enable) level is negated. The command registers are not written from the I/O write data asserted on the AD bus. Three of the four low-order I/O address bits on the A lines select the bit to be loaded while the fourth, the least significant I/O address bit, is used as the data input to set or clear the selected bit. For example, an I/O address of 20 sets output 0 of command register 0 (i.e., CPU RUN). An I/O address of 21 clears the same output.

I/O device selection for the console components on the DAP module is by the address latch outputs (not the A lines), and by the outputs from an address decoder that is enabled by the SEL CPU REG select level generated in the WCS module. Another select level generated in the WCS module controls the transceivers that gate AD bus data to and from the DAP module over the console bus. During an I/O write, this select level is negated, causing the transceiver to transmit AD bus data to the DAP module. The select level is asserted to change the direction of the data transfer only during an I/O read to a DAP module device.

Command register 2 on the DAP module is loaded much the same way as command registers 0 and 1. When command register 2 is addressed, the address decoder on the module asserts the register's enable, and the four low-order I/O address bits select and assert or negate an output.

An output from the address decoder also clocks the console write register (CWR) when it is addressed. The register is loaded with the I/O write data on the console bus by the trailing edge of the decoder's output. Another address decoder output reads the console read register (CRR) onto the console bus. The low-order register bit is gated to console bus line 0 through an input data multiplexer. The three low-order bits of I/O address, which are equal to 0 for the read of the CRR, select the appropriate multiplexer input. Other values for the low-order I/O address bits read a single DAP module status bit through the same multiplexer. Such status bits as CPU ACK and CPU ATTN may be read.

The CRR is normally loaded from the Y bus by the CPU microcode (Paragraph 2.6), but it can also be loaded by an I/O write operation. That is, the I/O address asserts an address decoder output that simply clocks the register. The current data on the Y bus (not the I/O write data on the console bus) is loaded into the register by the trailing edge of the decoder's output. This feature is used by the console-based microdiagnostics when verifying the console processor's basic data transfer capability to and from the CPU's data path. That is, data is loaded into the CWR where it is read onto the data path's D bus, through the data path's 2901As, and onto the data path's Y bus. The CRR is then clocked to load the data from the Y bus into the CRR, where it may be read and checked. Various data patterns are used to verify CWR and CRR operation before further testing of CPU components (via the CWR and CRR).

NOTE
The microdiagnostics load a MOVE micro-instruction (D ADRS field = FC) into the CPU's control store register to enable the transfer of CWR data through the data path's 2901As to the Y bus. The control store register is loaded one bit at a time as explained in Paragraph 4.3.3.

2.6 COMMUNICATIONS BETWEEN CONSOLE PROCESSOR AND DATA PATH

Communication between the console program running in the console processor and the CPU microcode controlling the data path in the CPU is by means of data transfers over the console bus. The bus transfers, one byte at a time, are through either the console write register (CWR) or the console read register (CRR) depending upon the direction of the transfer.

The console program sends data bytes to the data path over the console bus by writing the CWR. After a byte is loaded in the CWR, it can be read from the CWR into the data path by the CPU microcode. The read is by a MOVE microinstruction (discrete register address = FC) as described in Paragraph 6.9.

The CPU microcode sends data bytes from the data path to the console processor by first writing the CRR. The write is by a MISC microinstruction (function 2 field = 7). The data bytes are transferred over the console bus when the console program reads the CRR following each CRR write operation.

Data bytes are transferred over the console bus in groups (data packets). The number of bytes in a packet, as well as the number of packets interchanged during a transfer of information, are predefined and depend upon the machine mode (i.e., console mode or program mode) and the specific function being performed.

2.6.1 Communications in Console Mode

Except for a special case when a halt address packet is transferred during a program halt (Paragraph 2.6.2), communications over the console bus, when the console program is operating in console mode, are in response to console commands that require some action by the CPU microcode. For example, only the CPU microcode can access memory, and an examine memory command requires that the console program first send the CPU microcode a memory address over the console bus. The CPU microcode then makes the memory access and returns the memory data over the console bus so that it may be typed out by the console program.

Because they are the result of console commands, communications over the console bus in console mode are initiated by the console program. Furthermore, except for memory transfer commands (i.e., load memory command or an X command during APT), data packets sent by the console program or returned by the CPU microcode are a fixed size (10 bytes).

The types of transfers and the data packet formats in console mode are shown in Figure 2-22. The 10-byte packet sent by the console program to initiate an operation contains one byte of opcode, one byte reserved for use as an opcode modifier, four bytes reserved for address information, and four bytes reserved for data. Although an opcode modifier, an address, or data is not always required for an operation, a 10-byte packet is always sent over the bus.

NOTE: OPCODE MAY SPECIFY EXAMINE, DEPOSIT, INITIALIZE, CONTINUE, OR MEMORY TRANSFER.
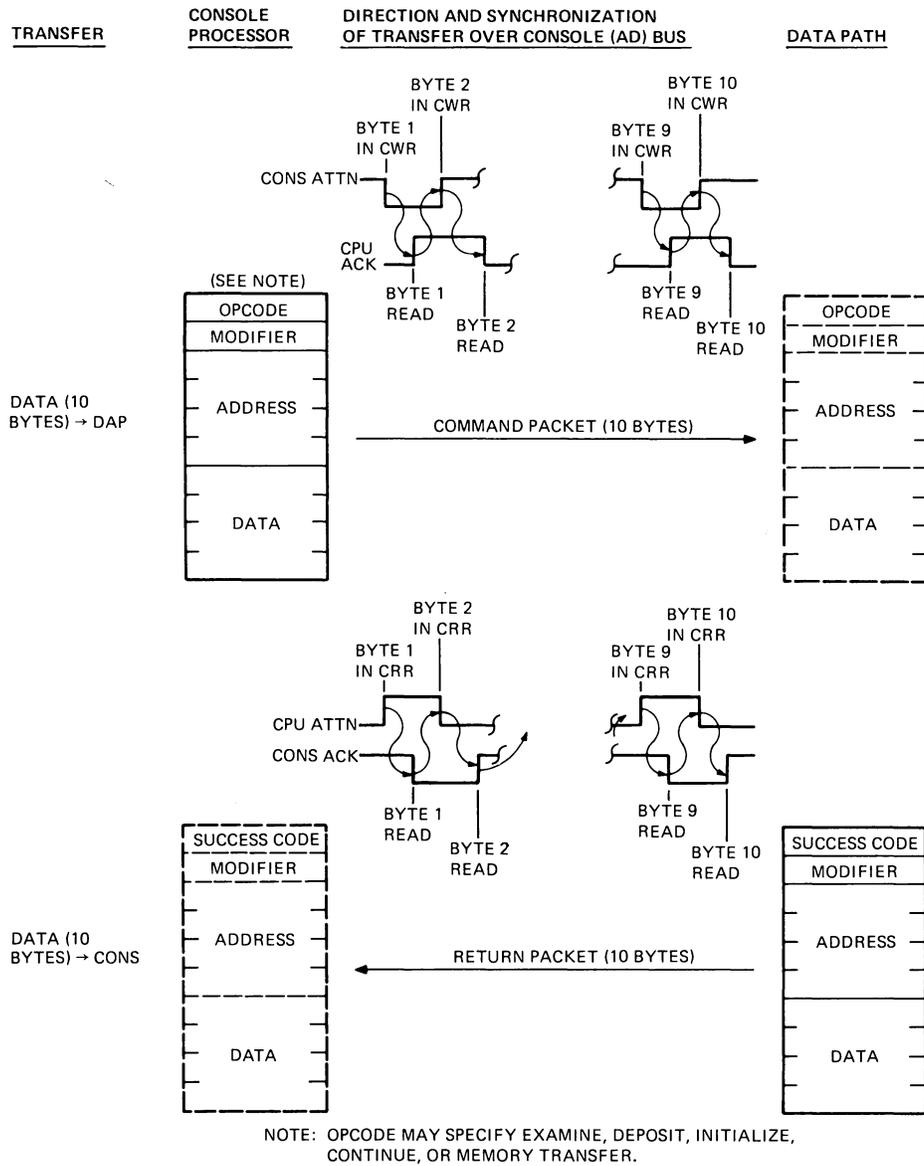
TK-6481

Figure 2-22   Communications Over Console Bus in Console Mode
(Sheet 1 of 2)
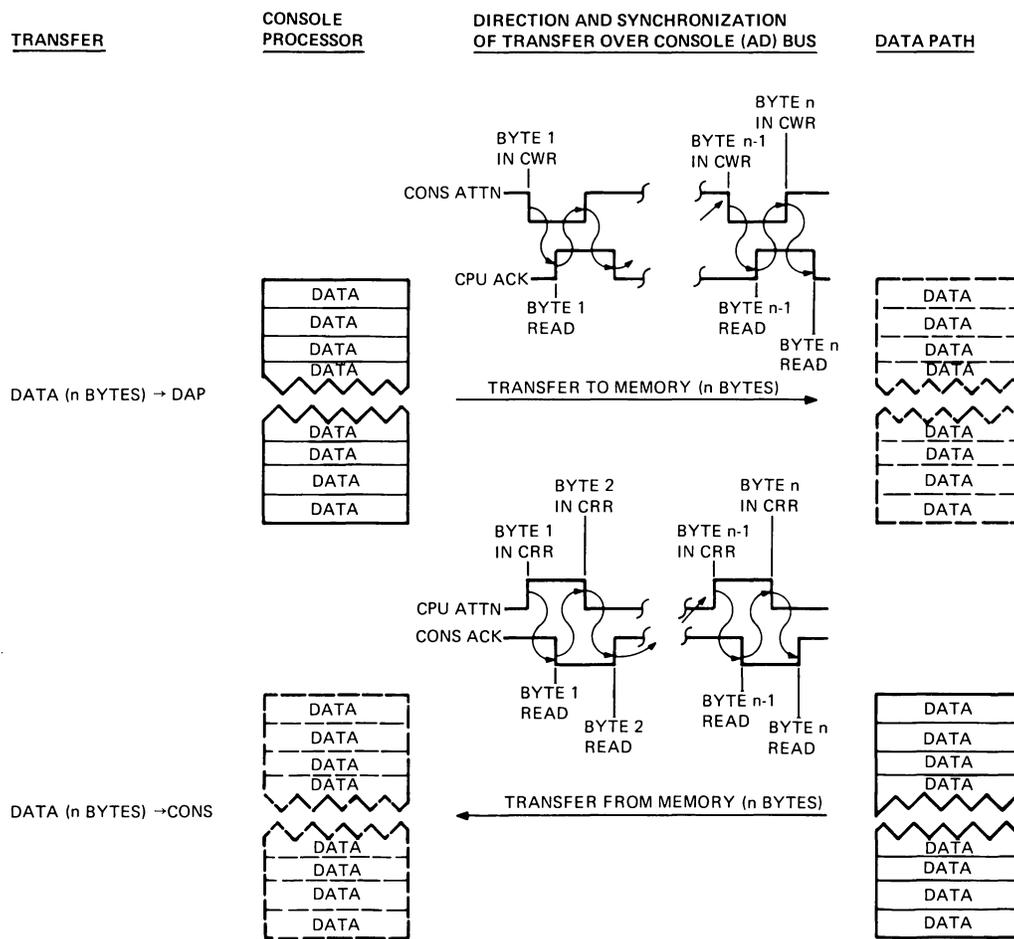
Figure 2-22 Communications Over Console Bus in Console Mode
(Sheet 2 of 2)

The CPU microcode responds to the 10-byte packet sent by the console program by first performing (or attempting to perform) the operation specified by the opcode. It then returns a 10-byte packet to the console program to indicate whether the operation was completed successfully or unsuccessfully, and to supply any necessary address and/or data information. Format for this return packet is similar to that sent by the console program. However, a success code rather than an opcode is contained in the packet.

For all operations except the memory transfer operations invoked by the L and X commands, the only console bus activity is the exchange of the two 10-byte packets; that is, the packet that initiates the operation and the packet that is returned to signal the end of the operation. However, if a memory transfer is specified by the first (command) packet, a variable length data packet containing the memory data is transferred next over the bus. (The data bytes in the command packet contain a byte count that specifies the length of the memory transfer.) This is followed by the 10-byte return packet that ends the operation. The variable length data packet is transferred to or from the data path depending upon the direction of the memory transfer. A positive byte count in the command packet specifies a transfer to memory. A negative count specifies a transfer from memory.

The console commands other than the memory transfer commands that initiate communications over the console bus are the examine, deposit, initialize, and continue commands. Examines and deposits may be made to physical memory, virtual memory, a GPR, an internal register address, or a machine specific register (i.e., PSL or CSR 0, 1, or 2 in the MCT). The type of access is specified in the low-order nibble of the opcode modifier. When applicable, the high-order nibble specifies the size of the transfer (i.e., byte, word, or longword).

The initialize command, which initializes the various CPU registers and generates masks and constants in local store, requires no opcode modifier nor any address or data information in the command packet. The continue (CPU program execution) command also requires no address or data information, but the modifier specifies either a normal start or the execution of a single instruction. (Single step mode is set previously by another console command.) Packet contents for the variable length memory transfer commands have been discussed. Modifier values and other data packet parameters for the various console bus operations in console mode are given in the *CPU Microcode Listing*.

The transfer of the individual data bytes in a packet is synchronized and controlled by the attention and acknowledge signals generated in both the console processor and data path.

For transfers to the data path, the console program asserts CONS ATTN to signal to the CPU microcode that the first data byte is in the CWR. The CPU microcode then reads the byte in the CWR and asserts CPU ACK to signal that the console program may reload the CWR with the next byte. When the next byte is loaded, the console program negates CONS ATTN. Correspondingly, the CPU negates CPU ACK when the byte is read. This alternate assertion and negation of the attention and acknowledge signals continues until all bytes in the packet have been transferred.

Transfers from the data path are synchronized and controlled in a similar fashion, except that CPU ATTN (not CPU ACK) is asserted and negated by the CPU microcode and CONS ACK (not CONS ATTN) is asserted and negated by the console program. A transition of CPU ATTN indicates the CPU microcode has loaded the CRR. The responding transition of CONS ACK indicates the byte in the CRR has been read by the console program.

Proper operation during console bus communications is checked by the console program. If the CPU microcode does not respond to a transition of CONS ATTN or CONS ACK by toggling its own synchronizing signal (CPU ACK or CPU ATTN), a time-out occurs in the console program which causes it to flag the communication error by typing an error message. The console program then enters the console idle loop.

2.6.2 Communications in Program Mode
Communications over the console bus in program mode are initiated by the CPU microcode. Data transfers are mainly in response to MTPRs or MFPRs directed to the console-based registers controlling the interval timer, the time of year clock, and the data transfers to and from the console terminal (local and remote) and the TU58. The types of transfers are shown in Figure 2-23.

Data packet size for the transfers in program mode is variable (one to five bytes) and depends upon the operation. The first (and sometimes only) byte sent by the CPU microcode is an opcode byte, which is used as an offset by the console program to index into the execution code for the specified operation.

Figure 2-23   Communications Over Console Bus in Program Mode
(Sheet 1 of 2)

TRANSFER CONSOLE
PROCESSOR DIRECTION AND SYNCHRONIZATION
OF TRANSFER OVER CONSOLE (AD) BUS DATA PATH

DATA
(BYTE 1)
IN CRR

DATA
(BYTE 3)
IN CRR

DATA
(BYTE 2)
IN CRR

DATA
(BYTE 4)
IN CRR

OFFSET
IN CRR

CPU ATTN

CONS ACK

OFFSET
READ

DATA
(BYTE 2)
READ

DATA
(BYTE 4)
READ

DATA
(BYTE 1)
READ

DATA
(BYTE 3)
READ

OFFSET → CONS
DATA (4 BYTES) → CONS

| OFFSET |
| DATA |
| DATA |
| DATA |
| DATA |

←

| OFFSET |
| DATA |
| DATA |
| DATA |
| DATA |

DATA
(BYTE 2)
READ

DATA
(BYTE 4)
READ

DATA
(BYTE 1)
READ

DATA
(BYTE 3)
READ

OFFSET
IN CRR

CPU ATTN

CONS ACK

OFFSET
READ

DATA
(BYTE 2)
IN CWR

DATA
(BYTE 4)
IN CWR

DATA
(BYTE 1)
IN CWR

DATA
(BYTE 3)
IN CWR

| OFFSET | ← | OFFSET |

OFFSET → CONS
DATA (4 BYTES) → DAP

| DATA |
| DATA |
| DATA |
| DATA |

→

| DATA |
| DATA |
| DATA |
| DATA |

TK-6483

Figure 2-23   Communications Over Console Bus in Program Mode
(Sheet 2 of 2)

Two operations for which only an offset is sent to the console program are those specifying a program halt and a UNIBUS INIT. (The program halt causes the console program to enter console mode, after which the CPU microcode sends a 10-byte halt address packet.) The other console bus communications in program mode are in response to interrupts or to MTPRs and MFPRs, and these transfer additional data. For example, a byte of data is sent in addition to the offset when interval timer control data is loaded by an MTPR (MTPR addressing the ICCS). Similarly, four bytes of additional data are sent to load a next interval count (MTPR addressing the NICR).

Other operations in program mode result in data being returned to the CPU microcode after it sends the offset. Of course, this is the case during the execution of MFPRs to the console-based registers. For example, one byte of register data is returned when interval timer status is read (MFPR addressing the ICCS). Other MFPRs cause two bytes or four bytes of register data to be returned.

An operation other than an MFPR that causes a return of data is an interrupt acknowledge sequence. Following the assertion of CONS ATTN by the console program (CONS ATTN interrupts the program executing in the CPU), the CPU microcode sends an offset which causes the console program to return a byte indicating the interrupting data transfer request. The interrupts occur during the transfer of terminal and TU58 data, as discussed in Paragraph 2.6.1.

The transfer of individual data bytes in program mode are synchronized and controlled by CPU ATTN and CONS ACK. When bytes are sent to the console processor from the data path, the CPU microcode asserts or negates CPU ATTN when it has loaded the CRR. The console program responds by asserting or negating CONS ACK when it has read the CRR. When bytes are returned to the data path from the console processor, CONS ACK is used to indicate the console program has loaded the CWR, whereas CPU ATTN is used to indicate the CPU microcode has read the CWR.

The console program checks CPU microcode response during console bus communications in program mode just at it does in console mode. When the CPU microcode fails to respond correctly to a transition of CONS ACK, a time-out occurs in the console program that causes it to enter the console mode idle loop. An error message is typed to flag the communication error.

3.1 INTRODUCTION

The CPU clock generator on the WCS module produces the 90 ns basic system clock and CPU clock phases 0, 1, and 2. Clock distribution to the CPU's DAP module and other system modules is shown in Figure 3-1.
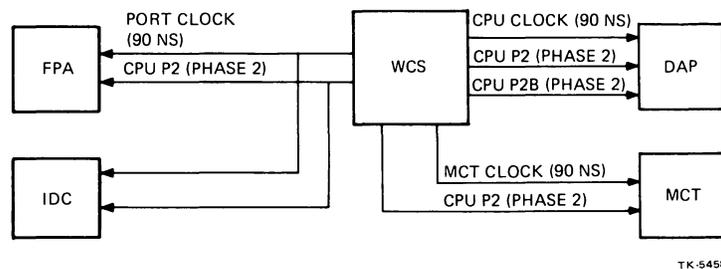
Figure 3-1   Clock Distribution

3.2 CLOCK GENERATOR CIRCUIT

The clock generator circuitry (Figure 3-2) consists of a 44.4 mHz oscillator and a divide-by-four frequency divider that produces the continuous 90 ns (11.1 mHz) basic clock, and a three-stage ring counter that produces the three CPU clock phases. Figure 3-3 shows clock generator output timing.

The first two stages of the ring counter generate CPU clock phases 0 and 1 (P0 and P1). These outputs are free-running to produce continuous clock trains. The phase 0 and phase 1 clocks are used mainly to generate the signals necessary to read, write, and refresh the dynamic RAMs in basic control store.

The third stage of the ring counter (CPU clock phase 2) actually has three independent outputs. These are not always free-running, and are gated by the clock generator's start/stop/step control. Each phase 2 output performs a different function. CPU P2 is used mainly to generate the CPU data path clocks, thus executing the current microinstruction. CPU P2B is used to generate the micro-PC clock. The third phase 2 output is used to generate CLK CSR, which normally loads the control store register (CSR) with the next microinstruction. The CPU microcycle is discussed in Paragraph 4.3.4.
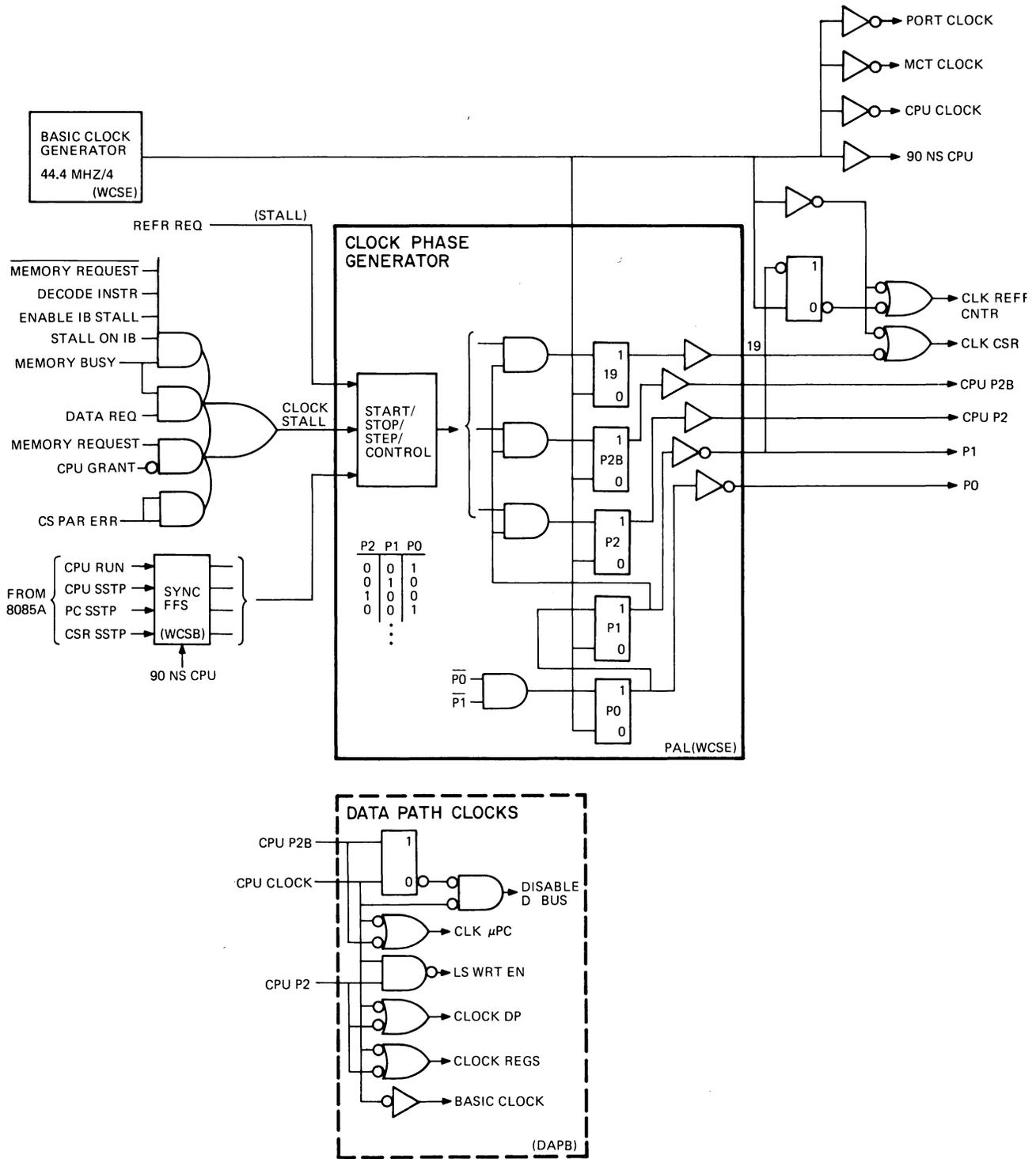
Figure 3-2   CPU Clock Generator Block Diagram

TK-5439
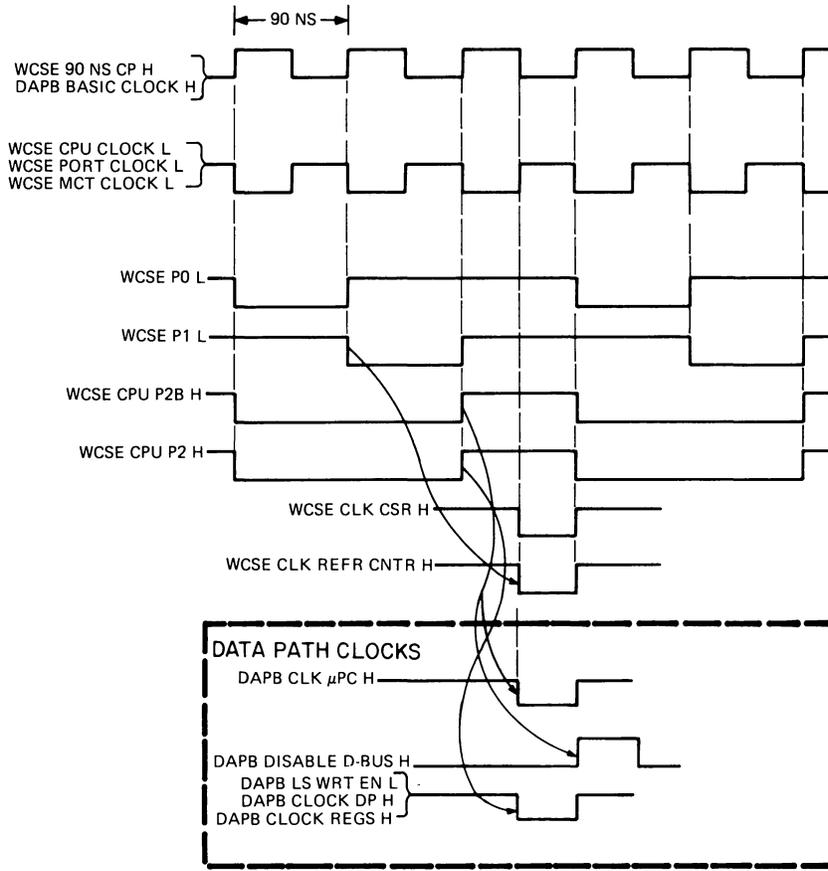
Figure 3-3 CPU Clocks Timing Diagram

3.3 CLOCK START/STOP/STEP CONTROL

The phase 2 clocks, by executing the current microinstruction and loading the next, are the controlling factor in sequencing CPU operation. Thus, these are the clocks that are gated to start, stop, and step the CPU.

3.3.1 Clock Control by the Console

The 8085A console processor provides the primary clock control. CPU RUN is asserted by the console program to start the phase 2 clocks, and it must remain asserted for normal full-speed CPU operations. If CPU RUN is negated, the phase 2 clocks are stopped, unconditionally halting all instruction level processing. The CPU is then considered to be in maintenance mode.

Whenever the CPU is in maintenance mode (CPU RUN = 0), the console program may assert signals to single-step the phase 2 clocks and thus CPU operations. Furthermore, all or parts of the CPU may be single-stepped depending on which clocks are generated. The signals asserted by the console program and the clocks they produce are as follows.

Signal	CPU P2 (DP Clocks)	CPU P2B (CLK μPC)	- (CLK CSR)
CPU SSTP	x	x	x
PC SSTP		x	
CSR SSTP			x

The CPU SSTP signal single-steps all three phase 2 clocks, causing a single CPU microinstruction to be executed. The signal is asserted by the 8085A console-based microdiagnostics when testing the CPU at single-step speeds.

The console program may also assert PC SSTP to single-step CPU P2B (and thus generate CLK μPC), but not the other phase 2 clocks. Similarly, CSR SSTP single-steps only one phase 2 clock to generate CLK CSR. The signals are asserted by the console program during microdiagnostics and system boot-strap; PC SSTP when parallel loading and shifting the micro-PC, and CSR SSTP when parallel loading and shifting the control store register (CSR).

3.3.2 Clock Stalls

Even when full-speed CPU operation has been enabled by the console program (CPU RUN = 1), it is sometimes necessary to temporarily stop CPU processing. This is accomplished by delaying (stalling) the generation of the phase 2 clocks. The clocks are delayed until the stall condition has been removed. Clock stalls are caused by control store refresh cycles, control store parity errors, and CPU interaction with memory.

A control store refresh cycle (REFR REQ = 1) stops the phase 2 clocks for just one 270 ns clock period. This inhibits just one microcycle in control store, allowing the basic control store's dynamic RAM storage array to be refreshed during the stall interval (Paragraph 4.3.5).

A control store parity error (CS PARITY ERR = 1) asserts CLOCK STALL and delays the phase 2 clocks until the error is removed. The error, which also interrupts the 8085A console processor, in-dicates that bad parity has been detected for the microinstruction in the CSR (Paragraph 4.3.7).

There are three types of clock stalls due to CPU interaction with memory. All three assert CLOCK STALL until the stall condition is removed.

One stall condition occurs when a memory request has been made by the CPU (MEMORY REQUEST = 1) and the memory controller does not respond immediately to the request (CPU GRANT = 0). Response can be delayed if the memory controller is performing a UNIBUS operation. The stall is released when the memory controller is ready to accept the memory request (CPU GRANT = 1). Memory requests are made by the MEM REQ microinstruction (Paragraphs 5.2.1 and 6.11) or by the DECODE microinstruction when an automatic refill of the prefetch register is initiated (Paragraph 5.2.1).

A second stall condition can occur for DECODE microinstructions (DECODE INSTR = 1) that are not making a memory request (MEMORY REQUEST = 0). The stall occurs when the DECODE microinstruction is to use prefetch register data (ENABLE IB STALL = 1) but the register does not yet contain valid instruction data (STALL ON IB = 1); that is, the memory controller is still busy doing a previously initiated instruction fetch (MEMORY BUSY = 1). The stall is released when the prefetch register is filled (MEMORY BUSY = 0).

The third stall condition due to interaction with memory occurs when the CPU is making a data transfer request (DATA REQ = 1) and the memory controller is not ready for the transfer of read/write data (MEMORY BUSY = 1). Stalls occur because of memory refresh cycles, soft read data errors, or because read data is unaligned. The stall is released when the data transfer is finally made (MEMORY BUSY = 0). For soft errors, the transfer is made after the data has been corrected. For unaligned data, it is made after the memory controller has aligned the information. Memory data transfers are initiated by the MOVE microinstruction (Paragraph 6.11).

NOTE
If the prefetch register cannot be filled or if a data transfer cannot be made due to a hard error (NXM, uncorrectable read error, etc.), the memory controller negates MEMORY BUSY when the hard error is detected to prevent an indefinite stall from occurring.

Clock stall conditions also affect the phase 2 clocks if they are being single-stepped in maintenance mode. That is, a control store refresh cycle will delay all clocks (for 270 ns) just as it normally does when the CPU is operating at full-speed. However, the other clock stall conditions only completely inhibit CPU P2 (data path clocks). The remaining phase 2 clocks, which clock the micro-PC and the CSR, are not completely inhibited and can still be single-stepped. This prevents conditions forced by the single-stepping microdiagnostics from blocking further operations; for example, the microdiagnostics shift data into the CSR one bit at a time, and the random parity errors produced would otherwise block further shift operations.

CHAPTER 4
CPU CONTROL STORE AND MICROSEQUENCER

4.1 INTRODUCTION

The execution of system-level instructions and operations in the VAX-11/730 system is sequenced and controlled by the microprogram contained in the CPU's writable control store (WCS). The control store, which is loaded during system bootstrap, provides storage for up to 20K microinstructions. (The basic control store is 16K but an additional 4K may be installed to support port devices and supply storage for user microcode.) Each microinstruction is 24 bits and contains several control fields, each of which control a specific CPU function.

Associated with the control store is a 24-bit control store register (CSR) which holds the micro-instructions as they are read from control store and executed. The control store, including the CSR, is contained on the WCS module.

The sequence of microinstructions read from the control store and loaded into the CSR is determined by the microsequencer located on the DAP module. The microsequencer presents a 15-bit next micro-address derived from either a micro-PC, a subroutine stack, or from the jump address field in the current microinstruction. (The current microinstruction is the microinstruction currently in the CSR.) The microaddress supplied by the micro-PC or stack can be incremented to provide a microprogram skip function. Also, part of the jump address may be OR'd with outputs from the operand specifier (OS) register located in the data path, or part of the jump address may be supplied by either of the two mapping ROMs in the instruction processing hardware.

4.2 MICROINSTRUCTION FORMATS

There are seven basic types of microinstructions used to execute the microcode in the CPU. These are the BASIC, MOVE, EXTENDED, MEM REQ, MISC/PORT, JUMP, and DECODE. Micro-instructions are 24 bits long, 23 bits of which are used for opcode and control information. One bit, the high order bit in each microinstruction (bit 23), is used to generate odd parity for the microword. Op-code and control bit formats for the various CPU microinstructions are shown in Figure 4-1.

An expanding opcode scheme is used to specify CPU microinstruction type. That is, a BASIC is speci-fied when a single op code bit is set (CSR ⟨22⟩ = 1). However, if this op code bit is not set, a second op code bit that is set (CSR ⟨21⟩ = 1) defines the next two types of microinstructions, depending upon the state of a third bit (CSR ⟨20⟩). (A MOVE has CSR ⟨20⟩ = 0; an EXTENDED has CSR ⟨20⟩ = 1.) The other microinstructions are defined in a similar fashion with four opcode bits required to specify a MEM REQ, MISC/PORT, JUMP, or DECODE. The advantage of an expanding opcode is that it allows more control bits in some microinstructions than would be possible with a fixed-length opcode field.

Control bits for the CPU microinstructions are defined in Tables 4-1 through 4-9. They are described in more detail in the appropriate sections of this manual; for example, the various microsequencer oper-ations specified by the microinstruction's skip and jump control fields (SCTL and JCTL) are discussed in Paragraphs 4.4.4 through 4.4.8.
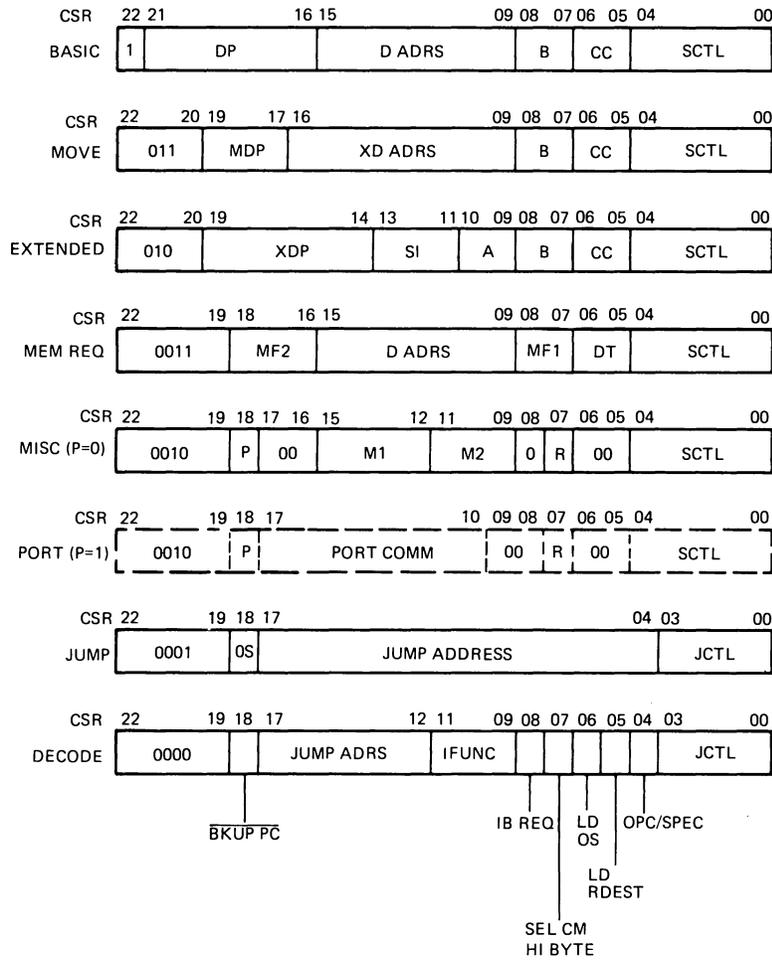
Figure 4-1 Bit Formats for CPU Microinstructions

Table 4-1 Control Bit Definitions for BASIC Microinstruction

CSR Bit(s)	Field	Function
⟨22⟩	OPCODE	Bit ⟨22⟩ = 1 defines microinstruction as a BASIC.
⟨21:16⟩	DP	Data path control. Controls 2901A data processor. Local store or discrete register written when DP = 20:3F.
⟨15:09⟩	D ADRS	Data address. Selects local store location or discrete register read and written.
⟨08:07⟩	B	B address. Selects 2901A working register read (from ports A and B) and written.

Table 4-1   Control Bit Definitions for BASIC Microinstruction (Cont)

CSR Bit(s)	Field	Function
⟨06:05⟩	CC	Condition code (and data type) control.

CC	Operation
00	Use longword, hold CCs
01	Use longword, load CCs
10	Use size, load CCs
11	Copy CCs

CSR Bit(s)	Field	Function
⟨04:00⟩	SCTL	Skip control. Defines microsequencer skip condition or special function.

Table 4-2   Control Bit Definitions for MOVE Microinstruction

CSR Bit(s)	Field	Function
⟨22:20⟩	OPCODE	Bits ⟨22:20⟩ = 011 define microinstruction as a MOVE.
⟨19:17⟩	MDP	Data path control. Controls 2901A data processor. Memory data request made when MDP = 0, 1, or 4. Local store location or discrete register read when MDP = 1, 2, 3, 5, 6, or 7 and written when MDP = 0, 4, 5, 6, or 7.
⟨16:09⟩	XD ADRS	Data address (extended). Selects local store location or discrete register read and written.
⟨08:07⟩	B	B address. Selects 2901A working register read (from ports A and B) and written. Working register address is B+4 when MDP = 2.
⟨06:05⟩	CC	Condition code (and data type) control.

CC	Operation
00	Use longword, hold CCs
01	Use longword, load CCs
10	Use size, load CCs
11	Copy CCs

CSR Bit(s)	Field	Function
⟨04:00⟩	SCTL	Skip control. Defines microsequencer skip condition or special function.

Table 4-3 Control Bit Definitions for EXTENDED Microinstruction

CSR Bit(s)	Field	Function
$\langle 22{:}20 \rangle$	OPCODE	Bits $\langle 22{:}20 \rangle$ = 010 define microinstruction as an EXTENDED.
$\langle 19{:}14 \rangle$	XDP	Data path control (extended). Controls 2901A data processor.
$\langle 13{:}11 \rangle$	SI	Shift input control. Selects shift data inputs to 2901A register (Q or working register).
$\langle 10{:}09 \rangle$	A	A address. Selects 2901A working register (read from port A).
$\langle 08{:}07 \rangle$	B	B address. Selects 2901A working register (read from port B). The address to which data is written.
$\langle 06{:}05 \rangle$	CC	Condition code (and data type) control.

CC   Operation

CC	Operation
00	Use longword, hold CCs
01	Use longword, load CCs
10	Use size, load CCs
11	Copy CCs

CSR Bit(s)	Field	Function
$\langle 04{:}00 \rangle$	SCTL	Skip control. Defines microsequencer skip condition or special function.

Table 4-4 Control Bit Definitions for MEM REQ Microinstruction

CSR Bit(s)	Field	Function
⟨22:19⟩	OPCODE	Bits ⟨22:19⟩ = 0011 define microinstruction as a MEM REQ.
⟨18:16,08:07⟩	MF2,1	Memory function. Specifies type of memory request.
⟨15:09⟩	D ADRS	Data address. Selects local store location containing memory address data.
⟨06:05⟩	DT	Data type control.
		DT **Operation**
		00 Byte
		01 Word
		10 Use size
		11 Longword
⟨04:00⟩	SCTL	Skip control. Defines microsequencer skip condition or special function.

Table 4-5 Control Bit Definitions for MISC/PORT Microinstruction

CSR Bit(s)	Field	Function
⟨22:19⟩	OPCODE	Bits ⟨22:19⟩ = 0010 define microinstruction as a MISC/PORT.
⟨18⟩	P	PORT/MISC select. P = 0 for MISC. P = 1 redefines MISC as a PORT.
⟨07⟩	R	Working Register Address. Selects WR 0 when R = 0. Selects WR 1 when R = 1.
⟨04:00⟩	SCTL	Skip control. Defines microsequencer skip condition or special function.

Table 4-5   Control Bit Definitions for MISC/PORT Microinstruction (Cont)

CSR Bit(s)	Field	Function
MISC (P = 0)		
⟨15:12⟩	M1	MISC function

M1	Operation	M1	Operation
0	Clear STATE 1:0	8	Set SEL ACC IN
1	Clear STATE 1, Set STATE 0	9	Clear SEL ACC IN
2	Set STATE 1, Clear STATE 0	A	Clear WSC PAGE
3	Clear STATE 0	B	Set WCS PAGE
4	Clear STATE 1	C	Clear CPU ATTN and CPU ACK
5	Set STATE 0	D	Set CPU ACK
6	Set STATE 1	E	Set CPU ATTN
7	Set RBKUP FLAG	F	NOP

CSR Bit(s)	Field	Function
⟨11:09⟩	M2	MISC function 2

M2	Operation	M2	Operation
0	NOP	4	Assert READ ACC μPC
1	Mask interrupts	5	Assert XFER GRANT
2	Assert CPU DATA AVAIL	6	Mask halt and T TRAP
3	Assert TRAP ACC	7	Write CRR register

CSR Bit(s)	Field	Function
PORT (P = 1)		
⟨17:15⟩	Device Select	IDC = 7
⟨14:13⟩	Operation	Read = 0 Write = 1 Control = 2

⟨12:10⟩	Hardware Select		
		⟨14:13⟩ = 0 or 1	⟨14:13⟩ = 2
	12:10 = 0	Control/status register	Clear FIFO Counter
	= 1	Disk address register	Reset BR
	= 2	Data byte	
	= 3	Data longword	Clear IDC
	= 4	Pattern (read only)	Set automode
	= 5	Position (read only)	Clear automode
	= 6		Select FIFO A
	= 7		Select FIFO B

Table 4-6 Control Bit Definitions for JUMP Microinstruction

CSR Bit(s)	Field	Function
⟨22:19⟩	OPCODE	Bits ⟨22:19⟩ = 0001 define microinstructions as a JUMP.
⟨18⟩	OS	Select OS. Causes OS⟨4:0⟩ to be ORed with five low-order bits of jump microaddress.
⟨17:04⟩	JUMP ADDRESS	Specifies 14-bit jump microaddress. Bits ⟨08:04⟩ ORed with OS⟨4:0⟩ if OS = 1.
⟨03:00⟩	JCTL	Jump control. Defines microsequencer jump condition or special functions.

Table 4-7 Control Bit Definitions for DECODE Microinstruction

CSR Bit(s)	Field	Function
⟨22:19⟩	OPCODE	Bits ⟨22:19⟩ = 0000 define microinstruction as a DECODE.
⟨18⟩	BKUP PC	Backup PC. When equal to zero, causes PC to be stored in WR 0 (IFUNC = even) or WR 4 (IFUNC = odd).
⟨17:12⟩	JUMP ADRS	Specifies six high-order bits of 14-bit jump microaddress. Eight low-order bits supplied by OPCODE or SPEC mapping ROM.
11:09⟩	IFUNC	Defines a set of dispatch addresses for the DECODE by supplying the high-order address bits for mapping ROMs. Instruction data supplies low-order address bits.
⟨08⟩	IB REQ	Enable byte of instruction data contained in PFR to drive IB bus. Increment PC. Refill PFR if last byte. Used in native mode.
⟨07⟩	SEL CM HI BYTE	Enables high-order byte of instruction data contained in PFR to drive IB bus. Used in compatibility mode.
⟨06⟩	LD OS	Load OS register from IB bus. Clear OS3 if compatibility mode and if LD RDEST = 1.
⟨05⟩	LD RDEST	Load RDEST (register destination) FLAG if mode specifier = 5 (native mode) or 0 (compatibility mode).
⟨04⟩	OPC/SPEC	Selects OPCODE mapping ROM when OPC/SPEC = 1. Selects SPEC mapping ROM when OPC/SPEC = 0.
⟨03:00⟩	JCTL	Jump control. Defines microsequencer jump condition or special function.

Table 4-8   SCTL Field Definitions

SCTL	Skip Condition	SCTL	Skip Condition
00	NOT ALU N	10	Return + 1 if NOT ERR SUM
01	NOT ALU Z	11	Loop if NOT ALU Z
02	NOT ALU V	12	Pop stack
03	ALU C	13	Loop if ALU C
04	NOT PSL C	14	Return
05	BRANCH FALSE	15	No skip (NOP)
06	GPR DEST	16	Return + 1
07	NOT INTERR REQ	17	Loop if NOT INTERR REQ and NOT ACC SYNC
08	ALU N	18	CONSOLE ATTN
09	ALU Z	19	CONSOLE ACK
0A	ALU V	1A	PORT INT
0B	NOT ALU C	1B	RBKUP FLAG
0C	STATE 0	1C	ALU N XOR ALU V
0D	STATE 1	1D	NOT ERR SUM
0E	NOT STATE 0	1E	Skip
0F	NOT STATE 1	1F	ACC SYNC

Table 4-9 JCTL Field Definitions

JCTL	Jump Condition	JCTL	Jump Condition
0	NOT ALU N	8	ALU N
1	NOT ALU Z	9	ALU Z
2	NOT ALU V	A	ALU V
3	ALU C	B	NOT ALU C
4	Jump	C	JSR
5	BRANCH FALSE	D	JSR if IB VALID
6	GPR DEST	E	No jump. Skip if IB VALID
7	NOT INTERR REQ	F	IB VALID

4.3 CONTROL STORE

A block diagram of control store is shown in Figure 4-2. The main components are the 16K basic control store, the additional (optional) 4K of user control store, a control store write register, and the CSR. Microinstructions are read from either the basic or user control store onto the CS bus and into the CSR. The CS bus is also used to load microinstructions into basic or user control store from the control store write register.

There are major differences between the basic and user control stores. The larger basic control store uses dynamic RAMs in its storage array which must be refreshed periodically. Associated circuitry includes address gates that generate a multiplexed row/column address, a refresh address counter, and a timing circuit that generates address select and strobe levels.

The smaller user control store uses RAMs (not dynamic RAMs) as storage elements. As a result, refreshing is not required. Also, multiplexed addressing is not necessary and the only additional addressing circuitry is a bank select decoder that selects the appropriate RAM chips in the storage array.

Addresses for basic control store are in the 0 to 16K range. User control store addresses are 16K to 20K.
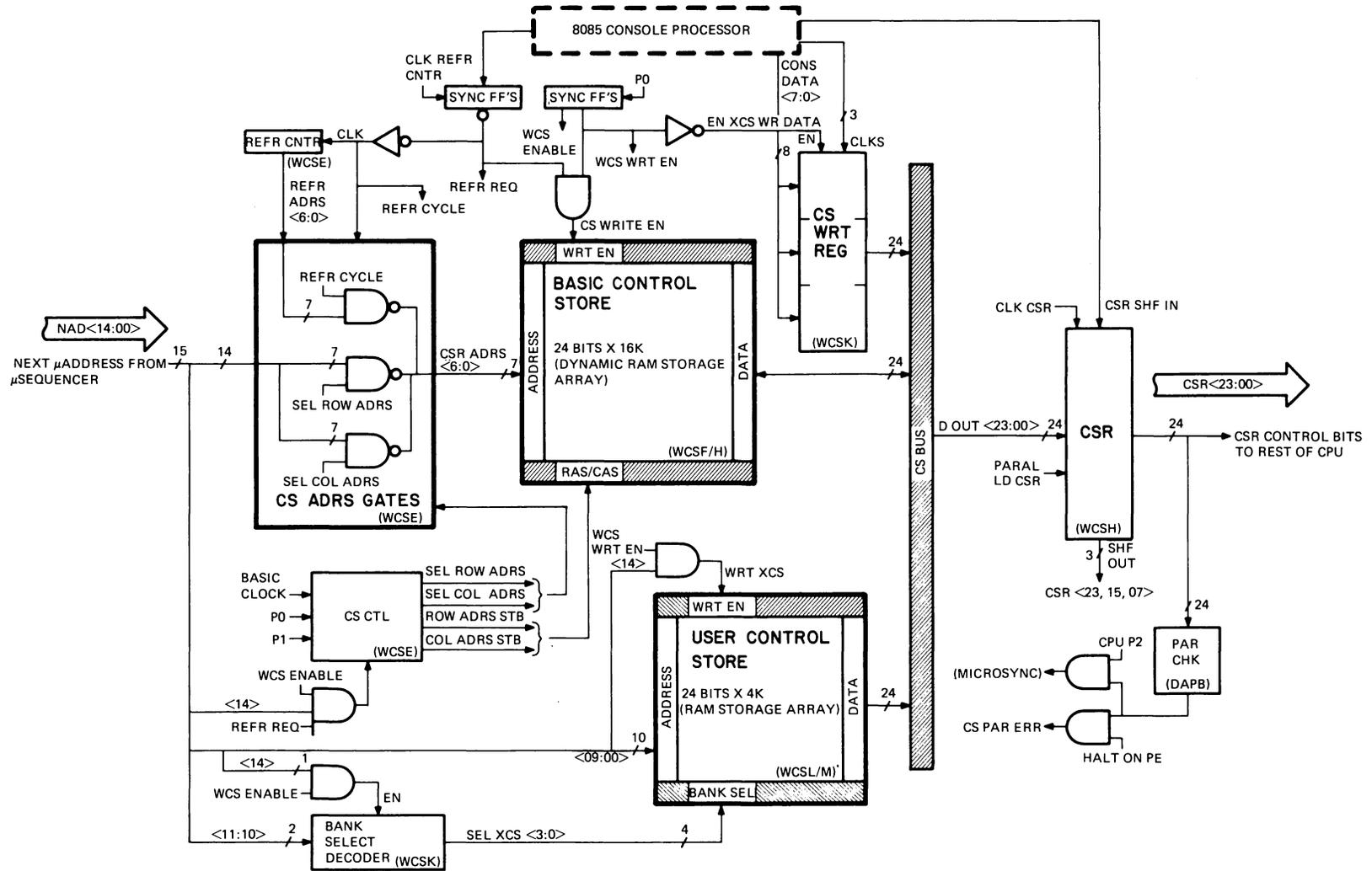
Figure 4-2   Control Store Block Diagram

TK-5440

4.3.1 Basic Control Store Storage Array

The dynamic RAMs used in basic control store are 16K (16,384) location × 1-bit MOS chips. There are 24 chips, one for each bit in the microword, to provide the total storage capacity of 16K 24-bit locations.

The basic control store has seven address lines CS ADRS ⟨6:0⟩ that parallel-connect to all 24 MOS chips. To access a location in the 16K (128 row × 128 column) chips, the address gates that drive the CS ADRS lines assert a 7-bit row address followed by a 7-bit column address. The row and column addresses are derived from the next microaddress presented by the microsequencer on the NAD lines.

NAD Lines	Addressing Function
⟨13:11, 03:00	Select 1 of 128 possible row addresses
10:04⟩	Select 1 of 128 possible column addresses

Also, because basic control store addresses are in the 0 to 16K range (NAD ⟨14:00⟩ = 0000 to 3FFF), the high-order NAD line (NAD 14 = 1) is used to deselect the basic control store by inhibiting the column address strobe for the storage array. (The column address strobe must be asserted before data can be read or written in the array's dynamic RAMs.) The generation of the address strobes is discussed in Paragraph 4.3.4.

4.3.2 User Control Store Storage Array

The RAMs in user control store are 1K (1024) location MOS chips. Each chip stores four bits, and 24 chips are used to provide the total storage of 4K (4096) 24-bit locations. The 24 chips in the storage array are configured into four 1K × 24-bit groups (banks) of six chips each. One of these banks is selected by one of the four outputs from the bank select decoder whenever user control store is addressed. The bank select signals (SEL XCS ⟨3:0⟩) connect to the CHIP EN inputs on the appropriate MOS chips.

The user control store is addressed directly by the NAD lines from the microsequencer. Two NAD lines select a 1K bank, and the 10 low-order NAD lines (which parallel-connect to all chips in the array) select a location within the selected bank.

NAD Lines	Addressing Function
⟨11:10⟩	Select 1 of 4 possible banks
⟨09:00⟩	Select 1 of 1K possible locations within the selected bank

Because user control store addresses are in the 16K to 20K range (NAD ⟨14:00⟩ = 4000 to 43FF), NAD 14 = 0 is used to disable the bank select decoder and thus deselect all chips in the array when user control store is not addressed.

4.3.3 Control Store Register (CSR)

The CSR is normally loaded directly from the CS bus to hold the current microinstruction. This is a parallel load of all 24 bits in the microinstruction from either the basic or user control store storage array. The CSR also has a shift capability. The CSR load mode is controlled by PARAL LD CSR, which is normally asserted by the 8085A console processor.

However, the console-based microdiagnostics may load and test the CSR by negating PARAL LD CSR and shifting data into the CSR's least significant bit position. CSR bits 07, 15, and 23 may then be examined as the test data is shifted through the register. CSR SHF IN, which may be set or cleared by the microdiagnostics, provides the shift data input. The CSR register clock, CLOCK CSR, is single-stepped by the microdiagnostics during the shift operations.

4.3.4 Basic Microcycle

Three CPU clock phases (0, 1, and 2) constitute a microcycle. The time that the CSR is clocked by CLK CSR to load the current microinstruction is defined at T0. This is the beginning of the microcycle and all other timing in the CPU is relative to this.

For example, the local store (LS) in the data path is written by a pulse (LS WRT EN) asserted from T225 to T270 of the microcycle. Basic microcycle timing is shown in Figure 4-3. Note that T270 (the end) of one microcycle coincides with T0 (the beginning) of the next.
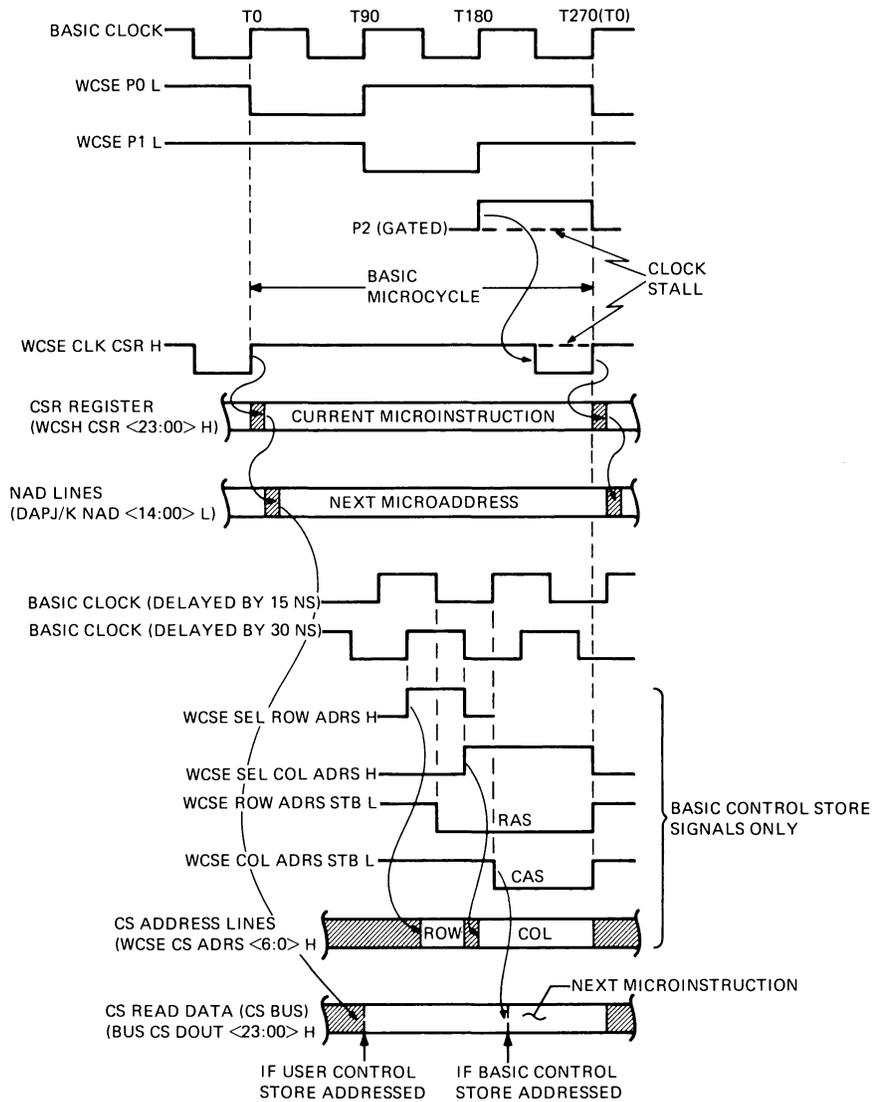
Figure 4-3   Basic Microcycle Timing Diagram

A microcycle is normally equal to a microstate; that is, the 270 ns interval from one CPU clock phase 0 to the next. However, during clock stalls, CPU clock phase 2 is inhibited, causing CLK CSR and the data path clocks not to be asserted for some length of time depending on the stall condition. As a result, T270 may be delayed, extending the microcycle by some multiple of 270 ns microstates (e.g., 270 ns, 540 ns, 810 ns, etc.).

With reference to the block diagram (Figure 4-2) and the timing diagram (Figure 4-3), control store operation during the basic microcycle is as follows.

At T0, the rising edge of CLK CSR loads the contents of the next microaddress from the CS bus directly into the CSR. This new microinstruction, now the current microinstruction, remains in the CSR for the entire microcycle.

Once the current microinstruction is loaded into the CSR, its control bits are asserted to set up and condition the hardware. The CPU functions specified by the microinstruction are then executed near or at the end of the microcycle. Most CPU functions are executed at T270.

The control bits in the current microinstruction also determine the next microaddress generated by the microsequencer. As a result, shortly after the CSR is loaded at the beginning of the microcycle, this new microaddress is transmitted to control store over the microsequencer's NAD lines NAD ⟨14:00⟩. The 15 NAD lines address the basic and user control store storage arrays as described in Paragraphs 4.3.1 and 4.3.2.

For basic control addresses (NAD 14 = 0), the control store address gates first allow a row address to the MOS chips in the storage array. This is followed by a column address. The multiplexer is controlled by two select levels, SEL ROW ADRS and SEL COL ADRS. Two other signals, ROW ADRS STB (RAS) and COL ADRS STB (CAS) latch the row and column addresses in the MOS chips. Once the column address has been latched, the contents of the next microaddress are read onto the CS bus.

For user control store addresses (NAD 14 = 1), no address line multiplexing is done. The NAD lines connect directly to the array chip's address inputs and the contents of the next microaddress are read onto the CS bus following the assertion of a bank select level by the bank select decoder.

After the contents of the next microaddress are read from either basic or user control store, the contents are loaded from the CS bus into the CSR to become the current microinstruction when the next CLOCK CSR occurs.

The basic control store's address select levels and address strobes are generated by a timing circuit. The timing circuit generates the address signals using the basic clock, the basic clock delayed, and CPU clock phases 0 and 1. A simplified block diagram showing basic timing is given in Figure 4-4. Note that more than one row and column address select level is generated. Those that do not coincide with the corresponding address strobe have no effect on basic control store operation.

As stated previously, during user control store references, NAD 14 = 1 disables the generation of the column address strobe. Also, extra row and column address select levels are generated but (again) this has no effect on basic control store operations.

The column address strobe is also inhibited before and after the write interval during a control store write operation (by WCS ENABLE = 0) as discussed in Paragraph 4.3.6. Furthermore, during a refresh cycle (REFR CYCLE = 1), all outputs from the timing circuit are inhibited except for the row address strobe. The refresh cycle is discussed in Paragraph 4.3.5.
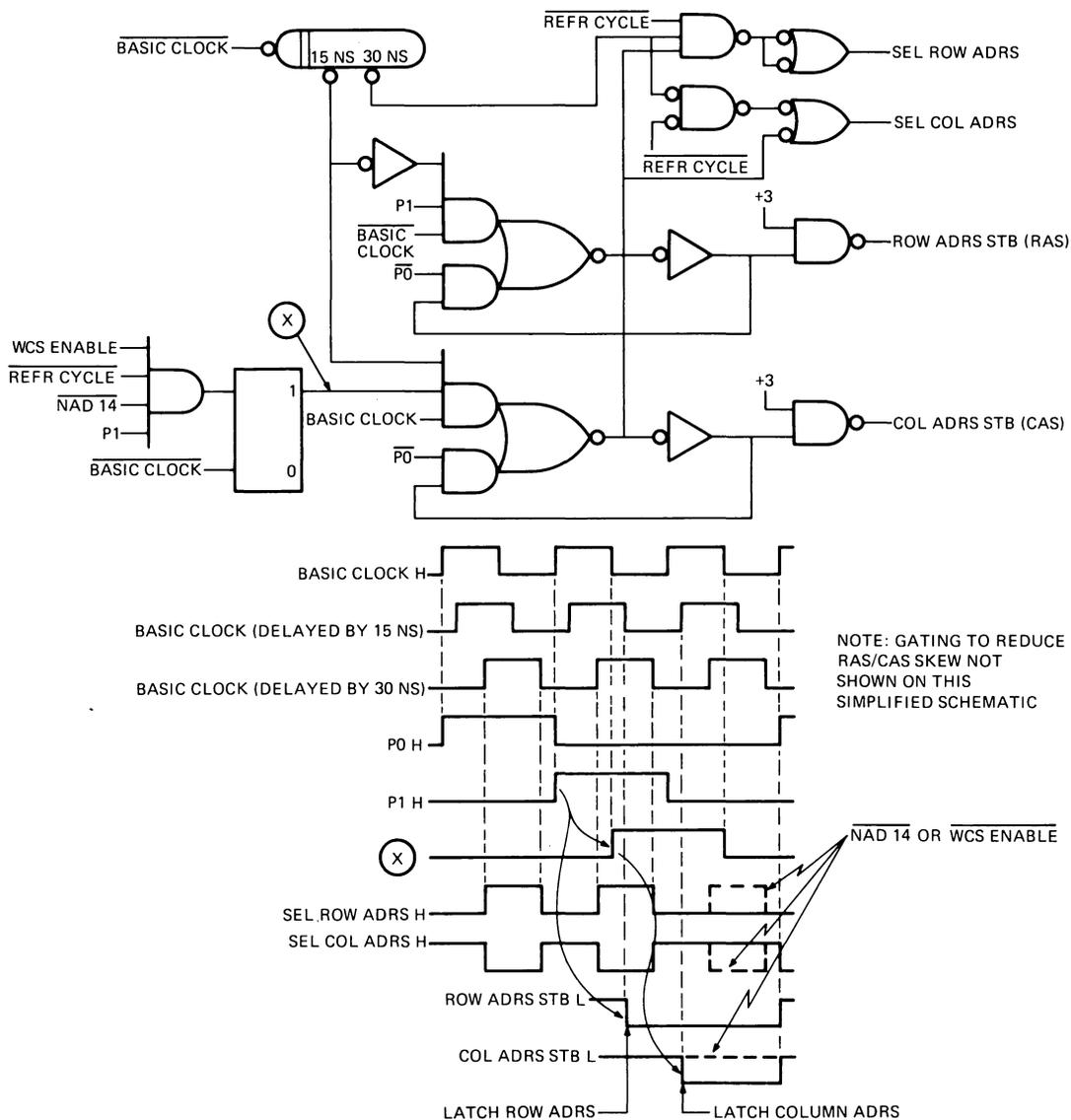
Figure 4-4   Control Store Timing Circuit (Simplified)

4.3.5   Control Store Refresh Cycle

Only the basic control store has to be refreshed. Like the dynamic RAM in the console processor which uses the same 16K (128 row × 128 column) MOS chips, the basic control store is refreshed by a series of row strobes to all 128 row addresses within the specified refresh interval. Also, a refresh cycle (which refreshes one row address) occurs every 12.8 μs.

The control store refresh cycle is initiated by the refresh control in the console processor. The refresh request signal is CON STORE REFR, (12.8 μs period), which is derived from a counter that is clocked continuously by the 8085A clock. The CON STORE REFR signal is synchronized to the control store microstate by three synchronizing flip-flops (clocked by CLK REFR CNTR) to generate REFR REQ when CON STORE REFR goes low. (Timing is shown in Figure 4-5, sheet 1.) REFR REQ, asserted for one microstate, then initiates a control store refresh cycle. It also initiates a memory refresh cycle by asserting MAIN MEM REFR. The memory refresh cycle, which occurs after the control store refresh cycle, is described in the *VAX-11/730 Memory System Technical Description*.
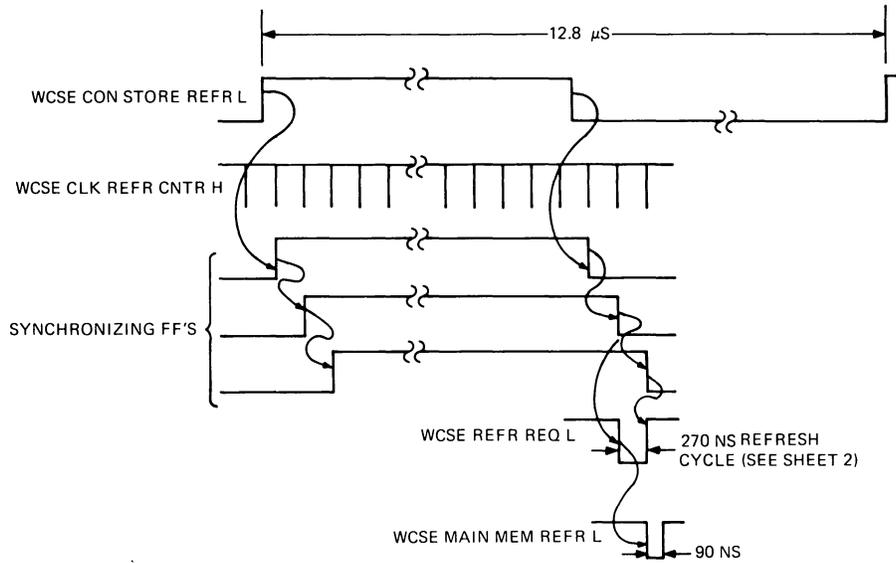
Figure 4-5 Control Store Refresh Operation Timing
Diagram (Sheet 1 of 2)
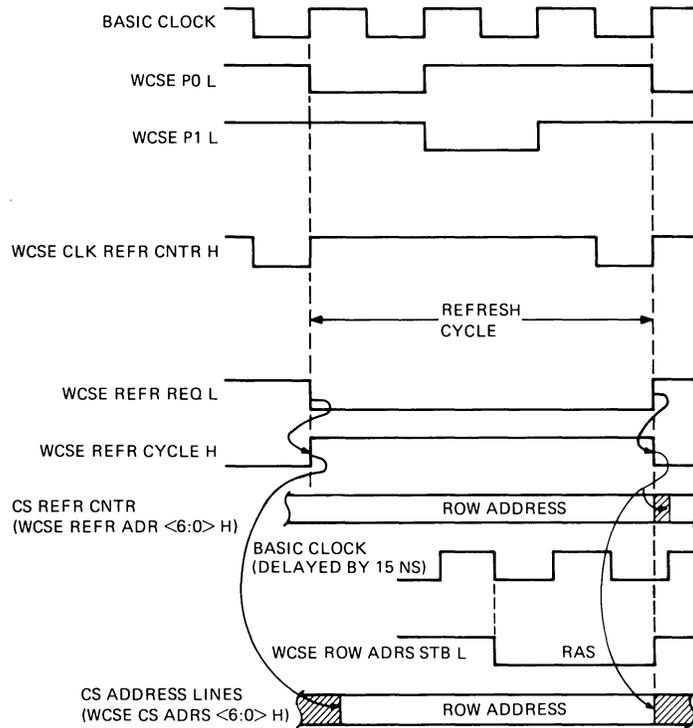
Figure 4-5 Control Store Refresh Operation Timing
Diagram (Sheet 2 of 2)

Timing for the control store refresh cycle initiated by REFR REQ is shown in Figure 4-5, sheet 2. REFR REQ first asserts REFR CYCLE. Then, during the microstate, REFR REQ stalls CPU clock phase 2 to prevent a microcycle, and REFR CYCLE conditions the control store address gates so that the outputs from a 7-bit refresh counter (a row address) are transmitted on the control store address lines. Also, REFR CYCLE inhibits all control store address select and strobe levels except for the row address strobe. This signal (ROW ADRS STB) is then asserted at its normal time to refresh the selected row address and end the refresh cycle.

The refresh counter (REFR ADRS ⟨6:0⟩) is incremented by the trailing edge of REFR CYCLE at the end of every refresh cycle and it increments through all the 128 possible row addresses during the 1.7 ms refresh interval.

> **NOTE**
> **SEE DC LO from the console processor asserts REFR CYCLE and forces continuous refresh cycles during system power-up and power-down sequencing. Otherwise, dc voltage transients that occur during the normal control store read cycle could cause the control store's contents to be modified.**

4.3.6 Control Store Write Operation

The control store is written by the 8085A console processor during system bootstrap, by certain console commands, and during microdiagnostics. The operation is performed with the CPU in maintenance mode (i.e., CPU clock phase 2 stopped). One 24-bit microword of control store data is written at a time.

The console program (or console-based microdiagnostic monitor) running in the 8085A console processor writes a single control store location by first shifting the microaddress into the micro-PC. (CLK μPC is single-stepped during this operation.) With the microinstruction in the CSR specifying a NOP function (previously loaded by the console processor), the microaddress in the micro-PC is transmitted by the microsequencer over the NAD lines to control store.

Next, the console program (or console-based microdiagnostic monitor) loads three bytes of write data into the control store write register. It then asserts WRITE WCS. This causes a set of four synchronizing flip-flops (clocked by CPU clock phase 0) to negate WCS ENABLE and to assert EN XCS WR DATA and WCS WRT EN. Timing is shown in Figure 4-6, sheet 1.

The normally asserted WCS ENABLE signal allows generation of the column address strobes for basic control store. In addition, it allows the assertion of the bank select levels for user control store. Thus, when WCS ENABLE is negated at the start of the control store write operation, it prevents either of the two storage arrays from driving the CS bus.

With the CS bus inactive, the other two signals generated by the synchronizing flip-flops do the following. EN XCS WRT DATA, asserted 270 ns after WCS ENABLE goes false, now enables the previously loaded control store write register to drive the CS bus. (The CS bus lines connect to the array chip's write data input pins as well as to the read data output pins.)

EN WCS WRT EN, generated at the same time as EN XCS WRT DATA, asserts the write enables for the storage arrays. (The enables parallel-connect to the array chip's write enable input pins.) The basic control store's write enable (CS WRITE EN) is conditioned by REFR REQ. The user control store's write enable (WRT XCS) is asserted only when NAD 14 = 1.

With the control store write register driving the CS bus, and with the array chips enabled to be written, WCS ENABLE is reasserted by the synchronizing flip-flops for as long as the console-generated write request is true. (The request will be true for several CPU microstates due to the comparatively slow speed of the 8085A console processor.) During the time WCS ENABLE is reasserted, called the write interval, it again allows column address strobes and bank select levels to be generated. This causes the console write register data transmitted on the CS bus to be written into the addressed control store location.

During the write interval, the same data is written at the same microaddress, one CPU microstate after the other. The timing for a single write cycle is shown in Figure 4-6, sheet 2. If a refresh request occurs when writing basic control store, REFR CYCLE inhibits the array's write enable level. Because the refresh cycle takes just one microstate to complete, and because the write to a single control store location repeats for several microstates, a refresh request cannot prevent a basic control store write operation from occurring.

A write to user control store, which does not use dynamic RAMs, does not need to be inhibited by a refresh cycle. Both a user control store write cycle and a basic control store refresh cycle may occur simultaneously.
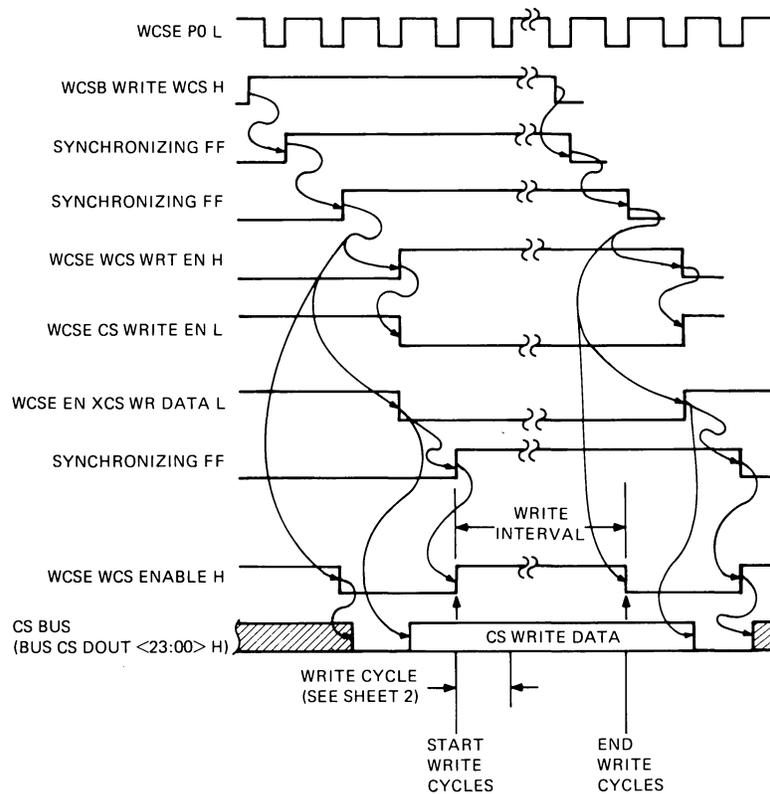
Figure 4-6   Control Store Write Operation Timing
Diagram (Sheet 1 of 2)
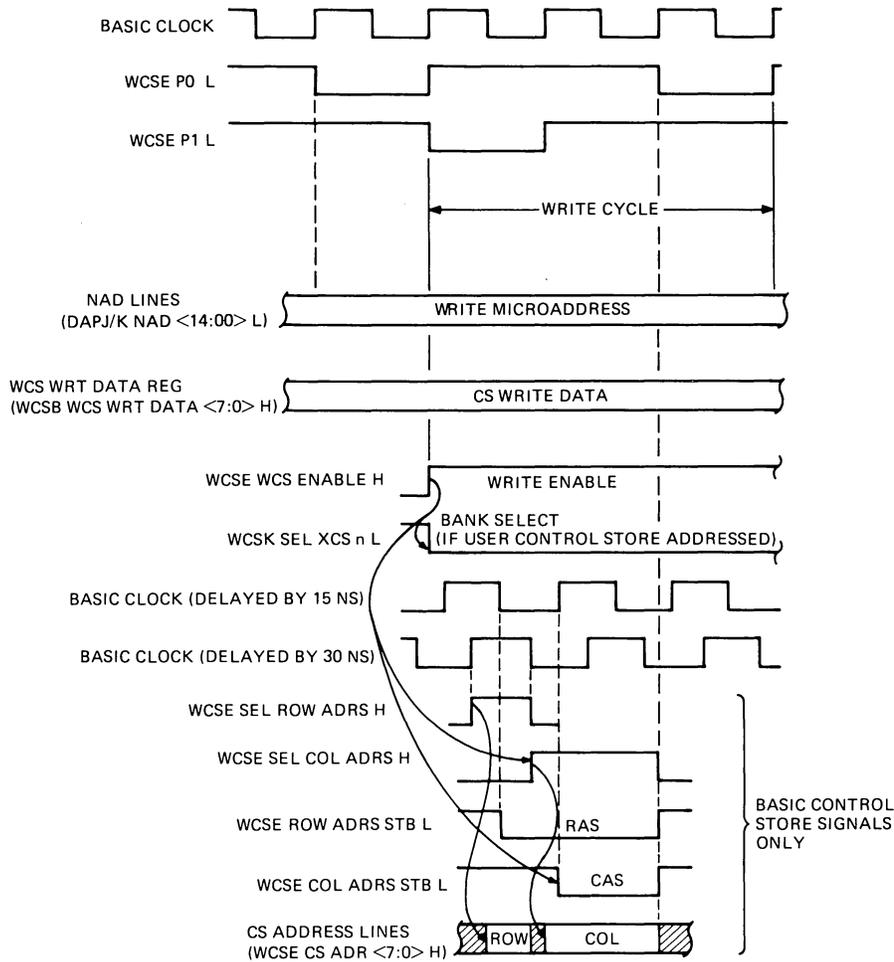
Figure 4-6   Control Store Write Operation Timing
Diagram (Sheet 2 of 2)

The control store write interval ends when the console processor negates WRITE WCS, which in turn negates WCS ENABLE. Chip write enable levels and the write register's output enable are then negated (270 ns later). With nothing driving the CS bus, WCS ENABLE is reasserted to its normal state, ending the control store write operation.

4.3.7   Control Store Parity and Microsync

All microinstructions in control store contain a parity bit (bit 23) which is generated by the microcode assembler program when the microcode is created. The correct parity is odd, and it is checked when the microinstruction is loaded into the CSR as the current microinstruction.

If the parity of the current microinstruction is bad (even), and if an error halt has been enabled by the 8085A console processor as it normally is (HALT ON PE = 1), CS PARITY ERR is asserted to stall CPU clock phase 2 before the microinstruction can be executed. The error signal, by asserting PAR ERR, also interrupts the 8085A console processor directly at one of its restart (RST) inputs. (The interrupt occurs at what would normally be T270 of the microcycle, had it been allowed to occur.) The console program may then read the micro-PC, which should be the address of the failing microinstruction + 1, and abort CPU operation.

A maintenance feature of the VAX-11/730 system is that the control store parity checking circuits may be used to generate a scope sync relative to a specific microinstruction. This microsync signal is generated by loading bad parity into the desired microaddress and by negating HALT ON PE to prevent error halts. (Both may be accomplished by means of console commands.) Then, with the CPU running at full-speed, a scope sync will be generated every time the microinstruction with bad parity is the current microinstruction.

The microsync signal is brought out to a test point (TP1) at the handle end of the DAP (M8390) module for easy accessibility. The signal is asserted during CPU clock phase 2. Thus, the execution of the microinstruction generating the sync signal may be viewed on the scope.

During the testing of the storage arrays in control store by the microdiagnostics, the CSR is inadvertently loaded with test data having bad parity. (Control store parity is not generated by hardware, only checked.) In these cases, the console processor may prevent clock stalls (if not single-stepping the clocks in console mode) by negating HALT ON PE.

4.4 MICROSEQUENCER
With reference to the block diagram (Figure 4-7), the basic logic elements in the microsequencer are the micro-PC, the subroutine stack including the stack pointer, the micro-PC bus and NAD bus, the incrementing logic connecting the two buses, and the NAD multiplexers that present the next microaddress to control store. There are also circuits to decode the current microinstruction, the main one being the microsequencer control which decodes the microinstruction's SCTL/JCTL fields, together with the various skip/jump/return conditions tested by the microprogram. The outputs from this circuit are the major signals used to sequence and control microsequencer operation.

4.4.1 Micro-PC
The micro-PC is a 15-bit register that drives the micro-PC bus (when EN μPC = 1) and is parallel-loaded from the NAD lines that present the next microaddress to control store. The load, which occurs at T270 of the microcycle, is not direct. That is, the micro-PC contains gating at its inputs which increments the NAD line value. As a result, during normal microsequencer operation, the micro-PC always contains the microaddress of the current microinstruction + 1.

The micro-PC is also a shift register. When the CPU is in maintenance mode, the 8085A console processor may load and examine register data one bit at a time by shifting data into the least significant bit position and examining the most significant bit position. This is similar to the shift facility built into the CSR. As a matter of fact, PARAL LD CSR (the signal controlling shift operation in the CSR) controls shift operation in the micro-PC.

To load data into the micro-PC, the console processor negates PARAL LD CSR and asserts or negates CONS ACK, which is the shift data input to the LSB. (CONS ACK, a jump or skip condition during normal microsequencer operation, can be used as the shift data input bit when the CPU is in maintenance mode.) The register clock, CLK μPC, is then single-stepped by the console processor to load and shift the data.
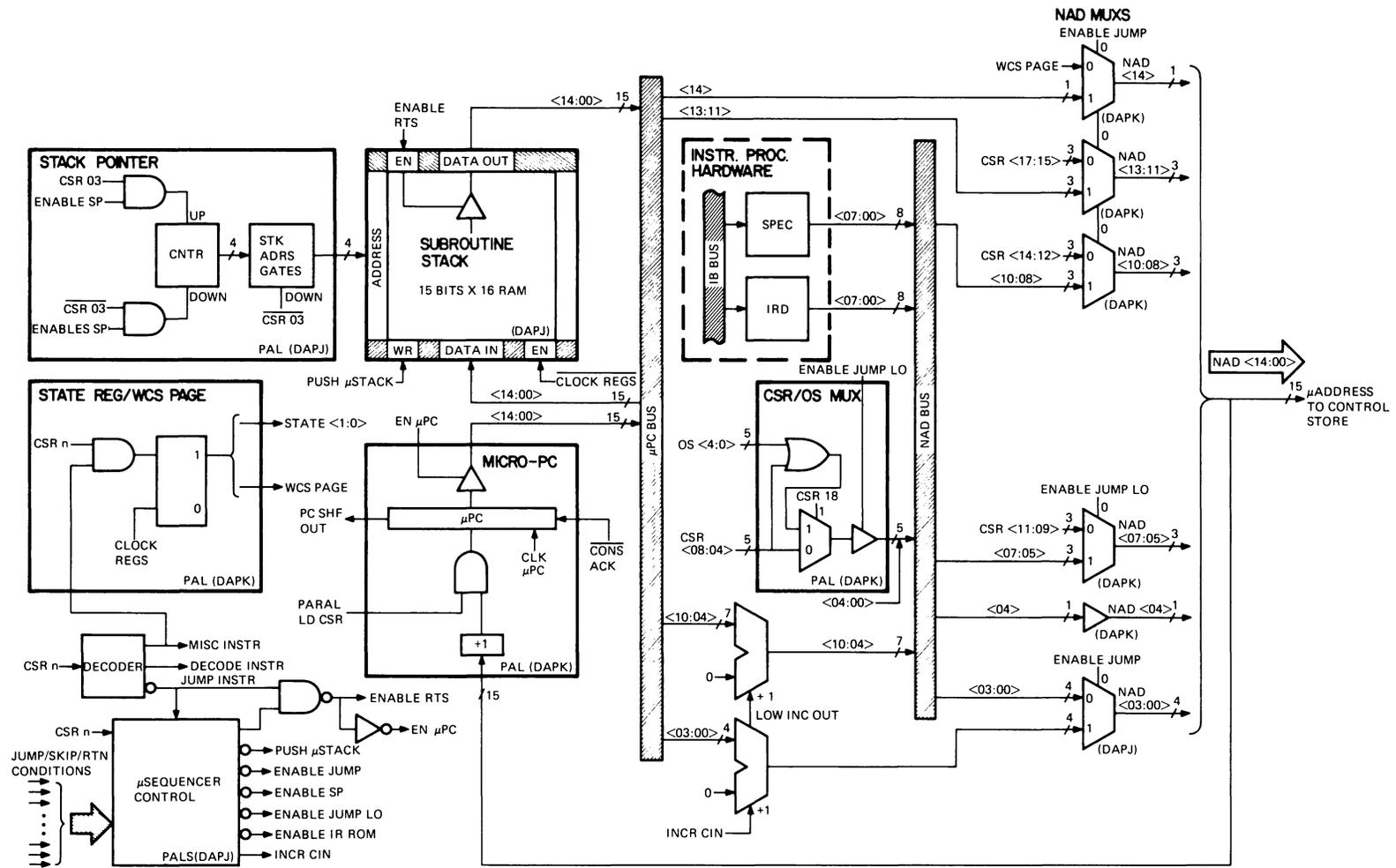
Figure 4-7   Microsequencer Block Diagram

4.4.2 Subroutine Stack

The subroutine stack is a 15-bit $\times$ 16 location static RAM, configured as a last in-first out (LIFO) stack. The stack saves microaddresses written from the micro-PC bus when a jump to subroutine (JSR) microinstruction is executed by the microsequencer. Saving the microaddress on the micro-PC bus (i.e., the microaddress of the JSR + 1) allows a return to the previous level of microprocessing once the subroutine has been executed. During a return, the stack drives the micro-PC bus to provide the return address.

The 16-location stack allows for subroutine nesting; that is, the jumping from one subroutine to another. Up to 16 return addresses may be saved (pushed) on the stack, thus allowing up to 16 JSRs before a return (pop) need take place.

Position in the stack during subroutine jumps and returns is determined by a stack pointer, as shown in Figure 4-8. The pointer addresses the 16-location RAM and consists of a 4-bit binary counter (enabled to count by ENABLE SP = 1) that supplies the RAM address through a set of stack address gates. The counter, which is down-counted by a JSR and up-counted by a return, normally addresses the top-of-stack. The top-of-stack is defined as the RAM location containing the last microaddress pushed.

NOTES

The stack pointer counter is not a standard binary counter. It counts through all 16 possible values, which is all that is required for addressing the 16-location stack, but (to contain the design within a single PAL) the count sequence does not give contiguous ascending or descending values. Consequently, the count direction ("down" for a pop/"up" for a push) is arbitrary. For reference purposes, the count sequence, and thus the stack address sequence, is given in Figure 4-8.

Note that the stack pointer counter is cyclic (wraparound). As a result, it is not necessary for the microprogram to pop all data off the stack at the end of an instruction execution; the current location entering an instruction execution becomes the new top-of-stack.
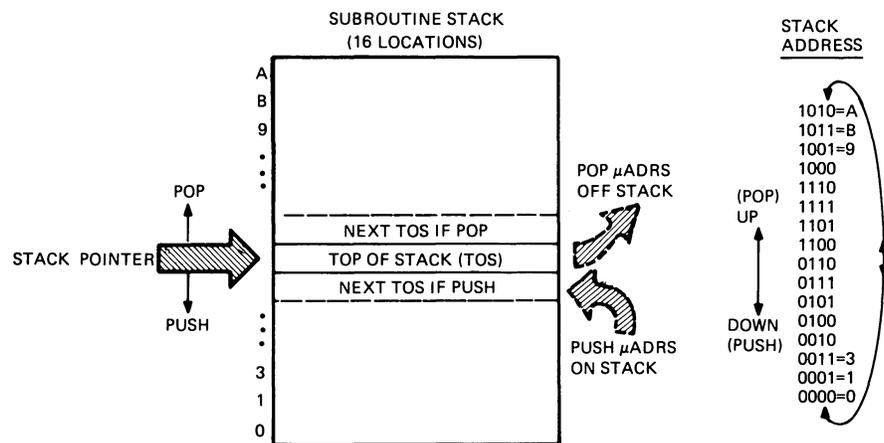
Figure 4-8   Subroutine Stack Addressing

4-21

During a JSR (PUSH μSTACK = 1), the stack is written from the micro-PC bus at T225 of the micro-cycle by CLOCK REGS. Because the stack pointer is not down-counted to the new top-of-stack address until T270 of the microcycle, the stack address gating forces a preliminary down-count of the current counter value for the stack write operation.

During a return (ENABLE RTS = 1), no preliminary up-count by the stack address gates is required because the counter is already at the current top-of-stack address. The microaddress in the top-of-stack location is transmitted on the micro-PC bus during the microcycle to select the next microinstruction (the return address). Then, at T270 of the microcycle, the counter is up-counted to the new top-of-stack value, provided the return is not a loop function. Detailed operation of the stack is described in Paragraphs 4.4.7 and 4.4.8.

4.4.3 State Register

The state register consists of two flip-flops set and cleared by the MISC microinstruction's function 1 field. The state register outputs, STATE 1 and STATE 0, are microsequencer skip conditions and may be used by the CPU microcode for internal housekeeping functions such as branch control and iteration (loop) count. STATE 1 and STATE 0 are negated during the class decode operation performed by the instruction processing hardware (i.e.; when IRD STATE = 1).

4.4.4 Microsequencer Control

The microsequencer control generates the basic control signals necessary to sequence and control microsequencer operation. The signals generated, which are listed in Tables 4-10 and 4-11, depend on the current microinstruction's SCTL or JCTL field, and the state of any skip, jump, or return conditions that have been specified by the SCTL/JCTL field.

Table 4-10   SCTL Field Decoding by Microsequencer Control

SCTL	Condition Met	Function	EN μPC	ENABLE RTS	ENABLE SP	INCR CIN
15	–	NOP	x			
17	No(INT.SYNC)	NOP	x			
00:0F,18:1D,1F	No	NOP	x			
1E	–	Skip	x			x
10	No	Skip	x			x
00:0F,18:1D,1F	Yes	Skip	x			x
14	–	Return		x	x	
16	–	Return + 1		x	x	x
10	Yes	Return + 1		x	x	x
11,13,17	Yes	Loop		x		
12	–	Pop	x		x	
11,13	No	Pop	x		x	
17	No(SYNC)	Pop	x		x	x

Table 4-11 JCTL Field Decoding by Microsequencer Control

JCTL	Condition Met	Function	EN μPC	PUSH μSTACK	ENABLE SP	INCR CIN	JUMP ENABLES*
0:3,5:B, D:F	No	NOP	x				
4	–	Jump	x				x
0:3,5:B, F	Yes	Jump	x				x
C	–	JSR	x	x	x		x
D	Yes	JSR	x	x	x		x
E	Yes	Skip	x			x	

*Jump enables == ENABLE JUMP/ENABLE JUMP LO (JUMP only) and ENABLE IR ROM (DECODE only)

During a microcycle, the microsequencer control outputs cause one of eight basic microsequencer functions as follows.

1. NOP – Only EN μPC is asserted to cause the micro-PC contents to be transmitted on the micro-PC bus. The unmodified micro-PC contents (i.e., the microaddress of the current microinstruction + 1) become the next microaddress presented to control store. Thus, the microinstruction immediately following the current one in the microprogram will be executed next. A NOP occurs as a result of a specified skip or jump condition not being met; or it can be specified unconditionally (SCTL = 15) to allow stepping through the microprogram in a sequential fashion.

2. Skip – Control signal INCR CIN is asserted in addition to EN μPC. This causes the incrementing logic between the micro-PC and NAD buses to increment the micro-PC contents by one. Thus, the next microaddress is that of the current microinstruction plus two, causing a skip in microprogram sequence.

3. Jump – EN μPC is asserted (to cause a NOP when a jump condition is false), but with a jump condition true; or if a jump has been specified unconditionally, the micro-PC is ignored, and jump enable levels are asserted to select all or part of the next microaddress from the current microinstruction's jump address field in the CSR. (The five least significant bits of the jump address field may be OR'd with OS ⟨4:0⟩, or the mapping ROMs in the instruction processing hardware can furnish the eight least significant bits.) The jump enable signals are ENABLE JUMP, ENABLE JUMP LO (JUMP microinstruction only), and ENABLE IR ROM (DECODE microinstruction only).

4. Jump to Subroutine (JSR) – Actually a jump, with the jump enable levels selecting the next microaddress all or in part from the current microinstruction; but PUSH μSTACK is also asserted to push the contents of the micro-PC (transmitted on the micro-PC bus by EN μPC) onto the subroutine stack. In addition, ENABLE SP is asserted to allow the stack pointer to down-count to a new top-of-stack address.

5. Return – ENABLE RTS causes the contents of the subroutine stack's top-of-stack address to be transmitted on the micro-PC bus. The unmodified contents then furnish the next (return) micro-address. ENABLE SP is also asserted to pop the stack; that is, up-count the stack pointer to the previous top-of-stack address.

6. Return + 1 – Same as a return function, except INCR CIN is also asserted to increment the return address by one.

7. Loop – Same as a return function except ENABLE SP is not asserted, resulting in no change to the stack pointer.

8. Pop – Essentially a NOP (or a skip in one case), but ENABLE SP is asserted in addition to EN μPC to up-count the stack pointer, thus discarding the top location on the stack.

NOTE
Only the low-order 11 bits of microaddress are incremented during a skip or return + 1 function. This effectively divides the microprogram in control store into 2K sections (pages); that is, even though the microprogram can jump or no-op to another section, it cannot move to another section by means of a skip or return + 1 function.

4.4.5 Skip (Or No-Skip) Operations

Skip operations provide for checking the various status bits in the machine. With one exception, skip operations are specified by the SCTL field of the current microinstruction, which can be any microinstruction but a JUMP or DECODE. (The exception is JCTL = E, which specifies a SKIP on IB VALID, and is used during instruction-stream decoding by the DECODE microinstruction.) Timing for a skip operation is shown in Figure 4-9.

As discussed in Paragraph 4.4.4, once a microinstruction specifying a skip is loaded as the current microinstruction, it invokes either a skip function (unconditionally or when the specified skip condition is met) or a NOP (skip condition not met). In both cases, microsequencer control output EN μPC is asserted to transmit the micro-PC contents (the microaddress of the current microinstruction + 1) on the micro-PC bus.

With no jump enable levels asserted, the four MSBs of microaddress (BUS μPC D $\langle 14:11 \rangle$) are gated directly through the NAD multiplexers to the corresponding NAD lines. However, the 11 LSBs of microaddress (BUS μPC D $\langle 10:00 \rangle$) are passed through the incrementing logic (two parallel-connected adder circuits), between the micro-PC and NAD buses before being gated to the NAD lines via the NAD multiplexers.

If the specified skip condition is met (e.g., ALU N = 0, STATE 0 = 1, etc.) or if an unconditional SKIP is specified (SCTL = 1E), INCR CIN asserts the carry-in input to the incrementing logic, adding one to the 11 LSBs of microaddress. The incremented value then becomes the skip address (the microaddress of the current microinstruction + 2) presented to control store.

If a specified skip condition is not true, INCR CIN is not asserted, and the low-order 11 bits of microaddress are gated through the incrementing logic unchanged. This results in a NOP function. (As stated previously, a NOP can be made to occur unconditionally by SCTL = 15.) Thus, a no-skip condition selects the next microinstruction in sequence.
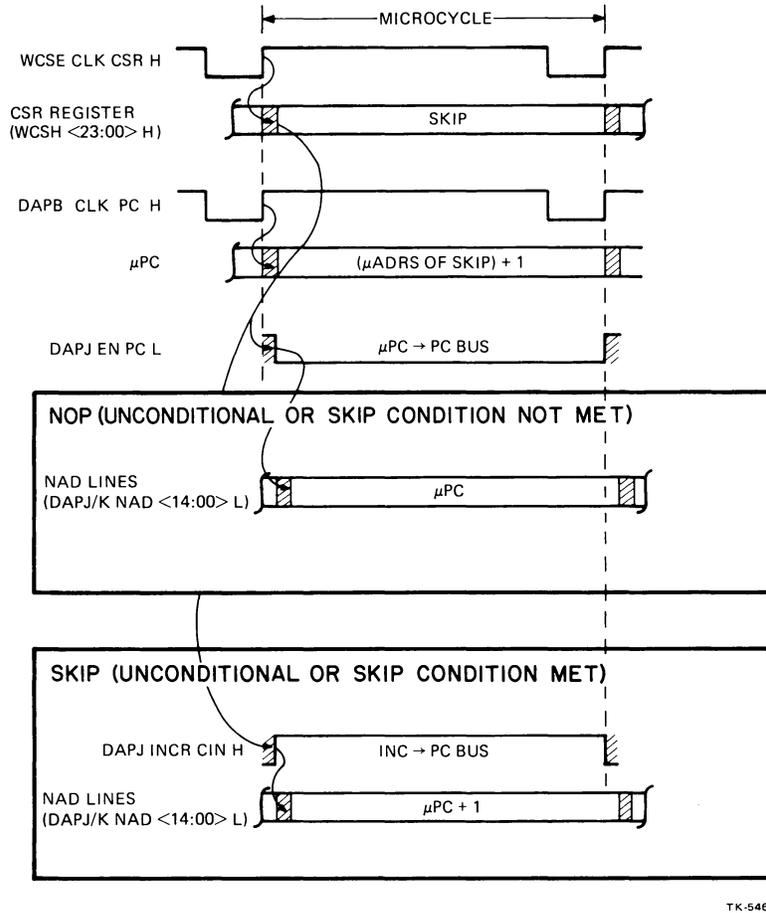
Figure 4-9   Skip (Or No-Skip) Timing Diagram

4.4.6   Jump (Or No-Jump) Operations

Jump operations, including the JSR, are specified by the JCTL field of the current microinstruction. The microinstruction can be either a JUMP or a DECODE. Timing is shown in Figure 4-10.

As discussed in Paragraph 4.4.4, when the current microinstruction specifies a jump, either a jump or a no-op function results. The jump occurs when it is specified as unconditional or when a jump condition is met. The no-op occurs when a jump condition is not met.

In both cases, EN μPC is asserted by the microsequencer control, placing the micro-PC contents on the micro-PC bus. This provides for loading the subroutine stack with a return address if the jump specified is a JSR. It also provides for executing the NOP when the jump condition is not met. The micro-PC contents are gated through the incrementing logic (unchanged) and NAD multiplexers to select the next microinstruction in sequence, just as happens when a skip condition is not met.

4-25
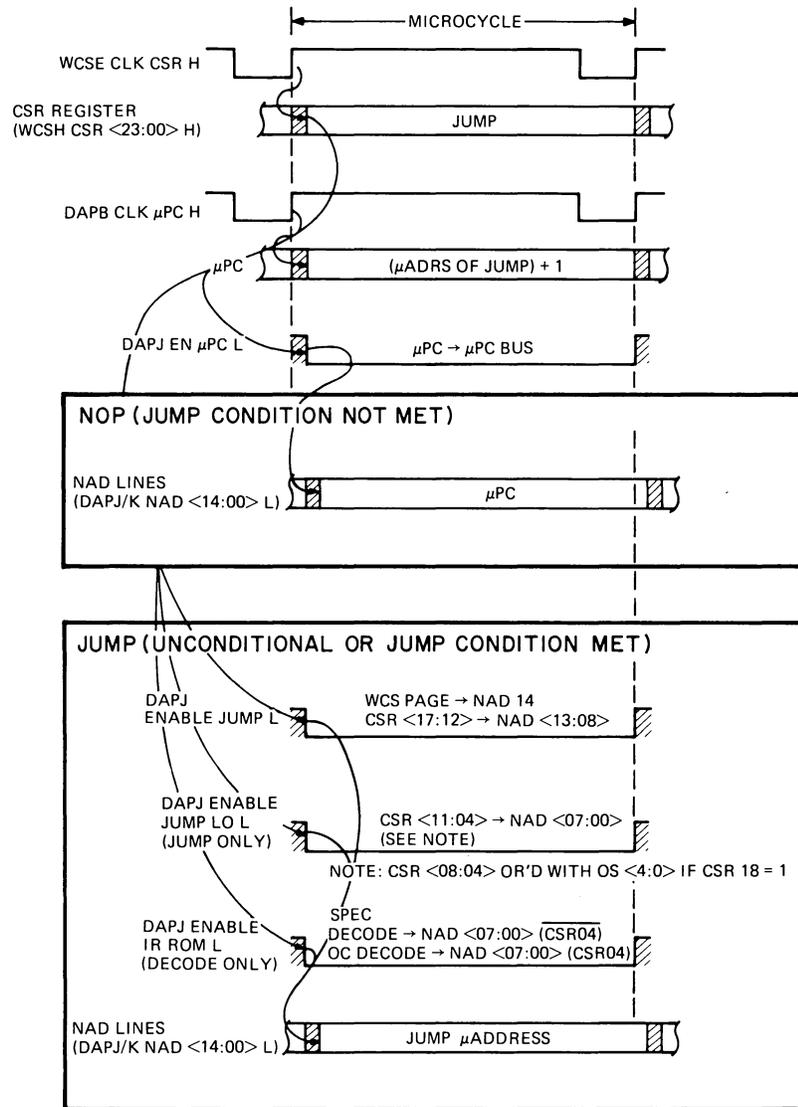
Figure 4-10   Jump (Or No-Jump) Timing Diagram

TK-5466

When the jump is unconditional (JCTL = 4 or C) or when the jump condition is met, two of three jump enable levels are asserted by the microsequencer control to deselect the micro-PC bus value at the NAD multiplexers, and select the next microaddress (also at the multiplexers) from the CSR and the CSR/OS MUX or mapping ROMs. The CSR bits are the current microinstruction's jump address field. Selection is as shown in Figure 4-11.

NOTE
Although the micro-PC and the subroutine stack are 15 bits wide and may address both user and basic control store, the jump address fields in the JUMP and DECODE provide for only 14 bits of micro-address. As a result, flip-flop WCS PAGE (which asserts NAD 14) is used as the high-order jump address bit at the input to the NAD multiplexers. The flip-flop is set before changing to user control store and while executing a microprogram in user control store. It is cleared before changing to basic control store and while executing a microprogram in basic control store. WCS PAGE is set and cleared by the MISC microinstruction.

One jump enable signal, ENABLE JUMP, is always asserted to select WCS PAGE and CSR ⟨17:12⟩ as the seven MSBs of jump address (NAD ⟨14:08⟩). Another enable, ENABLE JUMP LO, is asserted for a JUMP microinstruction only. It enables the CSR/OS MUX to drive the NAD bus which selects CSR 08:04, or CSR ⟨08:04⟩ OR'd with OS ⟨4:0⟩, as the low-order portion of the jump address (NAD ⟨04:00⟩).

NAD LINES

```
                14  13                              05  04                  00
                 ┌───┬──────────────────────────────┬────────────────────┐
JUMP (CSR 18 = 0)│WCS│        CSR<17:09>             │    CSR<08:04>       │
                 │PAG│                               │    (OS MUX)         │
                 └───┴──────────────────────────────┴────────────────────┘

                14  13                              05  04                  00
                 ┌───┬──────────────────────────────┬────────────────────┐
JUMP (CSR 18 = 1)│WCS│        CSR <17:09>            │CSR<08:04> + OS<4:0> │
                 │PAG│                               │    (OS MUX)         │
                 └───┴──────────────────────────────┴────────────────────┘

                14  13                   08  07                            00
                   ┌───┬──────────────────┬─────────────────────────────┐
DECODE (CSR 04 = 0)│WCS│   CSR<17:12>      │         SPEC DECODE          │
                   │PAG│                   │                              │
                   └───┴──────────────────┴─────────────────────────────┘

                14  13                   08  07                            00
                   ┌───┬──────────────────┬─────────────────────────────┐
DECODE (CSR 04 = 1)│WCS│   CSR<17:12>      │        OP CODE DECODE        │
                   │PAG│                   │                              │
                   └───┴──────────────────┴─────────────────────────────┘
```

TK-5464
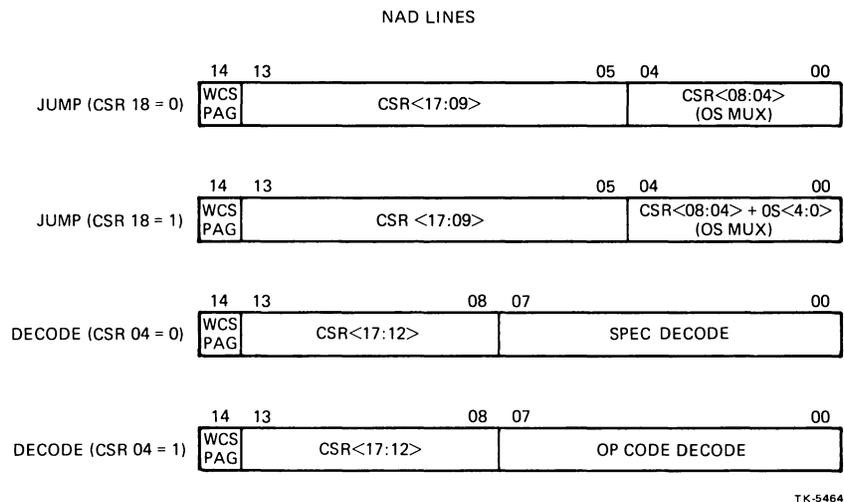
Figure 4-11   Jump Address Selection

4-27

The ORing of the CSR and OS bits occurs in the CSR/OS MUX when CSR 18 (the select OS bit) is equal to one in the JUMP microinstruction. This hardware feature allows multiple microprogram branch addresses to be generated, depending on the OS contents (e.g., register addresses, sign bits, etc.). The bits are OR'd to facilitate the masking of unused data; that is, a set bit in the microinstruction's jump field effectively masks the corresponding OS bit.

The remaining jump enable level, ENABLE IR ROM, is asserted for a DECODE microinstruction only. It enables either the SPEC or OPCODE dispatch ROMs to drive the NAD bus and provide the eight LSBs of jump address (NAD ⟨07:00⟩). Thus, multiple branch addresses may be generated in the microprogram when decoding the current system-level instruction. The ROM selected depends on CSR ⟨04⟩ (the OPC/SPEC bit) in the DECODE microinstruction. CSR ⟨04⟩ = 1 selects the OPCODE ROM; CSR ⟨04⟩ = 0 selects the SPEC ROM.

4.4.7 Subroutine Jumps and Returns

A jump to subroutine (JSR) is like any other jump in that it can be unconditional (JCTL = C) or conditional (JCTL = D). Also, when a jump function is invoked, jump enables are asserted, and a jump address is selected for the JUMP and DECODE microinstructions (as described in Paragraph 4.4.6 and shown in Figures 4-10 and 4-11). Similarly, a NOP occurs and operation is described in Paragraph 4.4.6 and shown in Figure 4-9 when the single jump condition (IB VALID) is not met.

The difference between a JSR and other jump functions is that the jump address is the microaddress of a subroutine in the microprogram. Thus, the micro-PC contents must be saved on the subroutine stack for a subsequent return to the current level of microprocessing. JSR timing is shown in the first part of Figure 4-12.
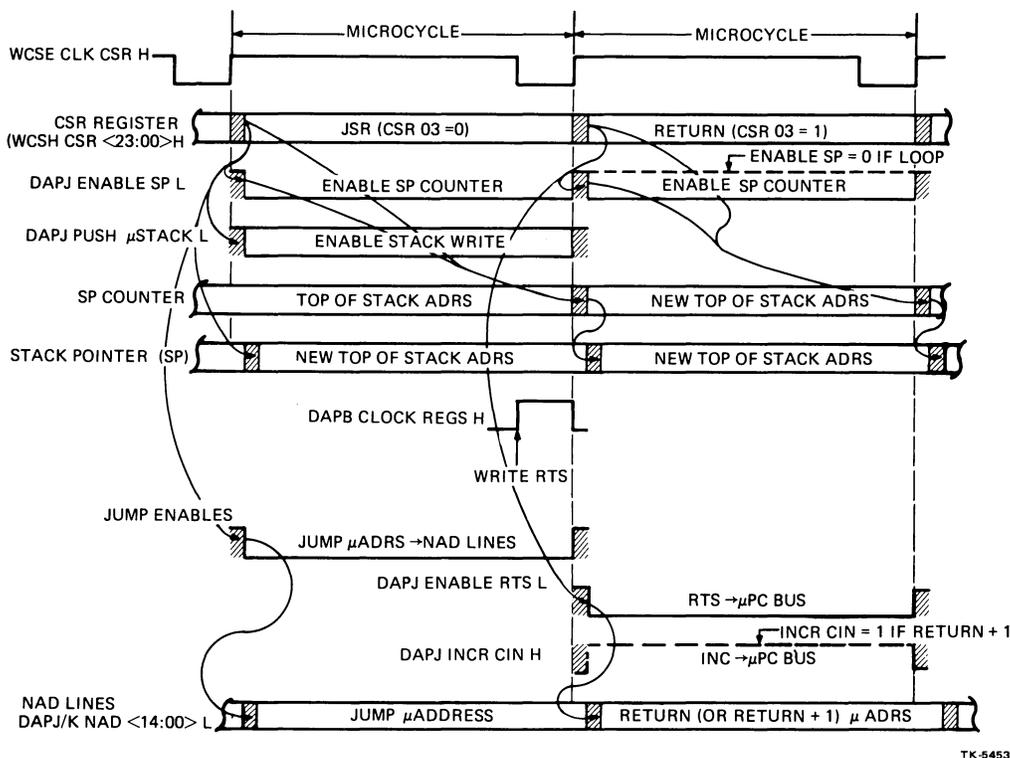
Figure 4-12   JSR/Return Timing Diagram

4-28

The microsequencer control outputs which control stack operation during a JSR are PUSH μSTACK and ENABLE SP. The PUSH μSTACK signal asserts the stack's write enable inputs, causing the micro-PC contents (transmitted on the micro-PC bus by μEN PC) to be loaded by the leading edge of CLOCK REGS. The ENABLE SP signal (also asserted during a return) then allows the stack pointer counter to be changed to the new top-of-stack address by the trailing edge of CLOCK REGS. For a JSR, CSR $\langle 03 \rangle$ is equal to zero, which causes the counter to be down-counted.

As explained in Paragraph 4.4.2, because the stack pointer is not down-counted to the new top-of-stack address until after the stack write, the stack address gates between the counter and the stack force a preliminary down-count during the write operation. Again, CSR 03 is the controlling signal; that is, CSR $\langle 03 \rangle$ = 0 conditions the address gates to temporarily change the value of the counter outputs, just as if the counter had actually been down-counted before.

Whereas JSRs are specified by the JCTL field of a JUMP or DECODE microinstruction, a return from the execution of a subroutine is specified by the SCTL field of any microinstruction other than a JUMP or DECODE. Returns are either to the address saved on the stack by the last JSR (the micro-address of the JSR + 1) or to that address incremented by one (the microaddress of the JSR + 2). Timing is shown in the lower part of Figure 4-12.

An unconditional return to the microaddress held in the top-of-stack is specified by SCTL = 14. The microsequencer control asserts ENABLE RTS, which transmits the top-of-stack contents onto the micro-PC bus to give the next microaddress. Also, as for a JSR, the microsequencer control asserts ENABLE SP to allow the stack pointer to be changed to a new value at the end of the microcycle. For a return, however, CSR $\langle 03 \rangle$ is equal to one and the counter is up-counted to the previous top-of-stack address.

A return to the top-of-stack contents + 1 (a return + 1 function) is the same as a return function, except that INCR CIN is also asserted by the microsequencer control. Like a skip function, the micro-PC bus value (the unmodified top-of-stack contents in this case) is incremented by the adders between the micro-PC and NAD buses to give the next microaddress. The return + 1 function may be unconditional (SCTL = 16) or conditional (SCTL = 10). If the single return + 1 condition (ERR SUM = 0) is not met, ENABLE RTS is not asserted. EN μPC and INCR CIN are asserted, however, to cause a standard skip function. The next microaddress will then be the micro-PC contents + 1.

4.4.8 Iteration Control (Loops and Pops)
The microsequencer provides a loop control feature for microinstructions (other than the JUMP and DECODE) that allows a single microinstruction to give an automatic jump to a loop address, following the test of certain machine status bits. Without this feature, two microinstructions (the test microinstruction followed by a "do-nothing" JUMP) would be necessary to perform the same function. The status bits tested are ALU Z = 0 (SCTL = 11), ALU C = 1 (SCTL = 13), and INTERR REQ = 0 AND ACC SYNC = 0 (SCTL = 17).

The loop control feature is implemented by holding the loop address on the top of the subroutine stack. The loop address is loaded on the stack by a JSR (to the loop address) in the control store location immediately preceding the loop address. A return function will then cause a jump to the loop address, provided the stack pointer remains at (or is at) the associated top-of-stack address. Thus, a loop is a return without the stack pointer being popped (ENABLE RTS = 1, ENABLE SP = 0).

NOTE
Because the only requirement of the iteration control feature is that the stack pointer point to the stack location holding the loop address when the loop is invoked, there may be subroutine calls and/or routines using this feature inside the loop.

The loop control feature is disabled by popping the associated top-of-stack address. This may be done on command (SCTL = 12) or automatically when the test conditions for jumping to the loop address are not met; for example, if ALU Z = 1 (SCTL = 11) or if ALU C = 0 (SCTL = 13), the micro-sequencer control asserts ENABLE SP and EN μPC. (The unconditional pop, SCTL = 12, asserts the same two signals.) With CSR 03 = 1, ENABLE SP up-counts (pops) the stack pointer, and (as for a NOP) EN μPC selects the micro-PC contents as the next microaddress taking the microprogram out of the loop.

When SCTL = 17, and either or both of the two test conditions for maintaining the loop are not met, operation differs depending on the status bit that has changed. If ACC SYNC = 1, the stack is popped (ENABLE SP = 1), and INCR CIN is asserted in addition to EN μ PC to cause a skip in the micro-program when exiting from the loop. If ACC SYNC still equals zero but a processor interrupt request has been made (INTERR REQ = 1), the stack is not popped. Only EN μPC is asserted to cause a NOP and select the next microinstruction for the exit. Operation of the SCTL = 17 loop function provides for efficient handling of interrupts while waiting (looping) on data from the FPA.

CHAPTER 5
INSTRUCTION PROCESSING HARDWARE

5.1 INTRODUCTION

The instruction processing hardware consists of a 32-bit prefetch register, an 8-bit opcode register, three mapping ROMs, and assorted control logic. A block diagram is shown in Figure 5-1.

The prefetch register (PFR) holds the instructions read from memory (over the MC bus) and executed by the CPU. During instruction evaluation and execution, the instruction data (op code, specifier, etc.) is removed from the PFR one byte at a time and transmitted on the instruction buffer (IB) bus. From the IB bus, the instruction data may be stored and/or used to generate dispatch (displacement) addresses for the microprogram. The manipulation of instruction data is done by the DECODE microinstruction.

Instruction data can be stored in one of two registers. Opcodes are stored in the opcode (OPC) register; specifiers and other instruction data are stored in the operand specifier (OS) register, which is part of the CPU's data path.

Instruction data on the IB bus generates microprogram dispatch addresses by accessing the OPCODE and SPEC mapping ROMs. That is, the DECODE microinstruction controlling the operation supplies the high order part of a jump address, and the mapping ROM outputs (asserted on the microsequencer's NAD bus) supply the low-order part of the jump address. The OPCODE ROM allows a dispatch on opcode; the SPEC ROM allows a dispatch on specifier information. Only one of these two ROMs is enabled during a DECODE.

The OPC CLASS mapping ROM, actually a ROM/PAL configuration, is not used for dispatch purposes. When accessed by the opcode, its outputs define the data type and condition code class for each instruction. The ROM/PAL also contains logic to test the PSL condition codes during branch instructions.

The PC, which is maintained in a local store location in the CPUs data path, is closely associated with the instruction processing hardware. The PC contents are used as a memory address when loading the PFR with instruction data over the MC bus. The native (VAX) mode PC is kept in local store location 10. The compatibility (PDP-11) mode PC is kept in local store location 47.
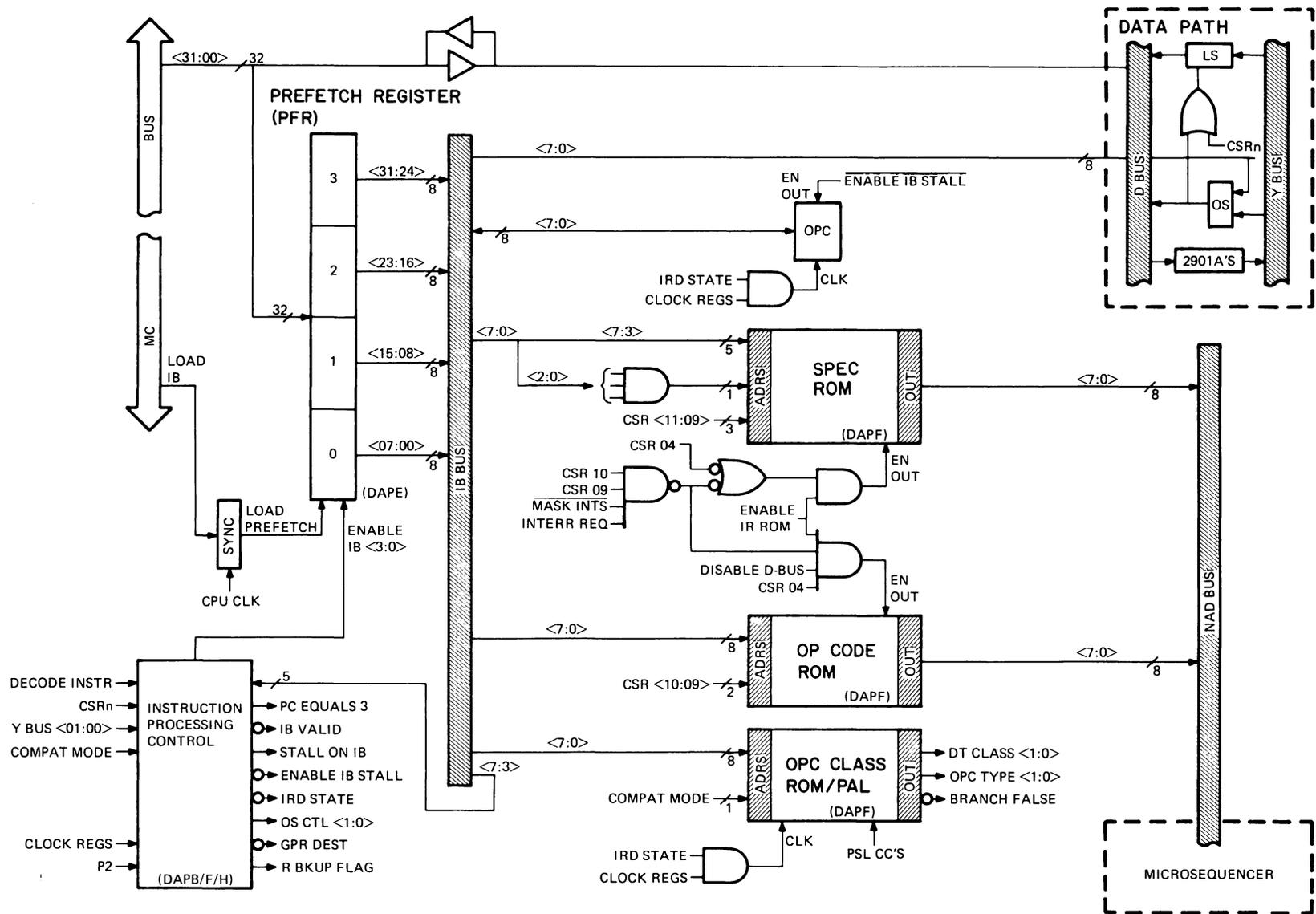
Figure 5-1   Instruction Processing Hardware Block Diagram

TK-5438

5.2 INSTRUCTION PREFETCH REGISTER (PFR)

The 32-bit PFR is an instruction buffer that holds four bytes of prefetched longword-aligned instruction data from memory. Because the CPU can operate in either native or compatibility mode, the instruction data loaded from memory into the PFR has two basic formats.

In native mode, the variable length instructions are stored in contiguous byte positions in memory and are aligned on byte boundaries. The contiguous bytes of instruction data are referred to as the instruction stream. In compatibility mode, the PDP-11 instructions are 16 bits, occupy two contiguous bytes, and are aligned on word boundaries. Basic formats are shown in Figure 5-2.

NATIVE MODE

OPERAND SPECIFIER N (1 OR 2 BYTES)	IMMEDIATE DATA (1,2,4, or 8 BYTES)	OPERAND SPECIFIER 2 (1 OR 2 BYTES)	SPECIFIER EXTENSION (1 TO 6 BYTES)	OPERAND SPECIFIER 1 (1 OR 2 BYTES)	OPCODE (1 OR 2 BYTES)

COMPATABILITY MODE

PDP-11 INSTRUCTIONS (2 BYTES)

INSTRUCTIONS

SINGLE OPERAND GROUP

15 OPCODE (10 BITS)	06 05 DEST (6 BITS) 00

DOUBLE OPERAND GROUP

15 OPCODE (4 BITS)	12 11 SRC (6 BITS)	06 05 DEST (6 BITS) 00

CONDITION CODE OPERATORS

15 0	0	0	2	4	05 04 03 CC'S (4 BITS) 00

REGISTER SOURCE OR DESTINATION

15	09 08 REG (3 BITS)	06 05 SRC/DEST (6 BITS) 00

SUBROUTINE RETURN

15 0	0	0	2	0	03 02 REG (3 BITS) 00

BRANCH

15 OP CODE (8 BITS)	08 07 OFFSET (8 BITS) 00

TK-5446

Figure 5-2 Basic Instruction Formats (Native and Compatibility Modes)

5.2.1 Loading the PFR

Instruction-stream data may be prefetched from memory and loaded in the PFR in two different ways. The PFR is loaded automatically in native mode when a DECODE microinstruction removes the last byte of prefetched instruction data contained in the register. The PFR may also be loaded on command by the MEM REQ microinstruction. The ability to prefetch and store instruction data greatly enhances overall operation of the CPU.

NOTE
The PFR cannot be loaded automatically when the CPU is operating in compatibility mode.

5-3

The automatic prefetch of instruction data in native mode occurs when a DECODE has the IB REQ bit (CSR ⟨08⟩) asserted (which causes a byte to be removed from the PFR and transmitted on the IB bus), and the two low-order bits of the PC are equal to three. It is the value of the PC, which is incremented automatically by every IB REQ and which points to the current (and in this case the last) byte of pre-fetched instruction data being removed from the PFR, that initiates the refill.

A memory request is made by CPU hardware and the next longword (the next contiguous four bytes) of instruction-stream data is loaded into the register. In order to address the next longword of instruction data, the memory controller (MCT) must increment the memory address (the current value of the PC) supplied by the CPU.

The PFR is loaded on command by a MEM REQ microinstruction specifying a READ WITH RCHK IFILL operation (CSR ⟨18:16, 8:7⟩ = 0E). Again, a memory request is made with the PC supplying the memory address, and another longword of instruction data is loaded in the PFR. Because all bytes of instruction data in the PFR have not necessarily been processed by the CPU, this operation is called a "flush and load." It is initiated in native mode when there is a jump in instruction-stream processing. It is the only way to fill the PFR in compatibility mode.

NOTE
In some cases in native mode, prefetched instruction data in the PFR is not strictly byte oriented (e.g., long literals, word or long displacement, etc.). In these cases, the data may be fetched by the CPU by means of a normal memory request (not a READ WITH RCHK IFILL) and processed in the data path independent of the instruction processing hardware. This requires that the PC be updated manually after the data is processed, and that a flush and load be initiated to refill the PFR.

5.2.1.1 Instruction Data in the PFR – In both types of PFR load operations, the PFR is always loaded with an aligned longword of instruction data. That is, if an opcode is in byte 0 of a memory longword, it is loaded into byte 0 of the PFR. This is illustrated in Figure 5-3, which shows native mode instruction data in memory, and the same data as it is loaded into the PFR.
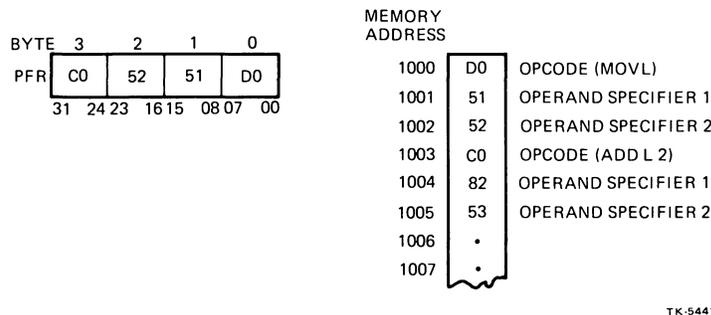
Figure 5-3   Instruction Data in Memory and PFR

To ensure that an aligned longword reference occurs during the flush and load operation, the MCT ignores the two low-order bits of memory address from the CPU. For example, if a native mode instruction changes the PC to 1003, the MCT uses address 1000 during the accompanying flush and load operation (which is part of instruction execution), and the PFR is loaded as shown in Figure 5-3.

During an automatic refill of the PFR, no action other than the normal incrementing of the memory address is necessary to ensure an aligned longword reference. This is because the two low-order bits of the PC, and thus the pre-incremented memory address, are always equal to three. As a result, the normal increment clears the two low-order bits of address to give an automatic aligned longword fetch.

5.2.1.2 Detailed Operation for the PFR Load – Timing for the PFR load operation is given in Figure 5-4. When a PFR load is initiated by either a DECODE or MEM REQ microinstruction, the CPU first asserts MEMORY REQ on the MC bus. This occurs at the beginning of the microcycle. Shortly afterwards, the CPU transmits the memory address (the PC) on the MC bus data lines.
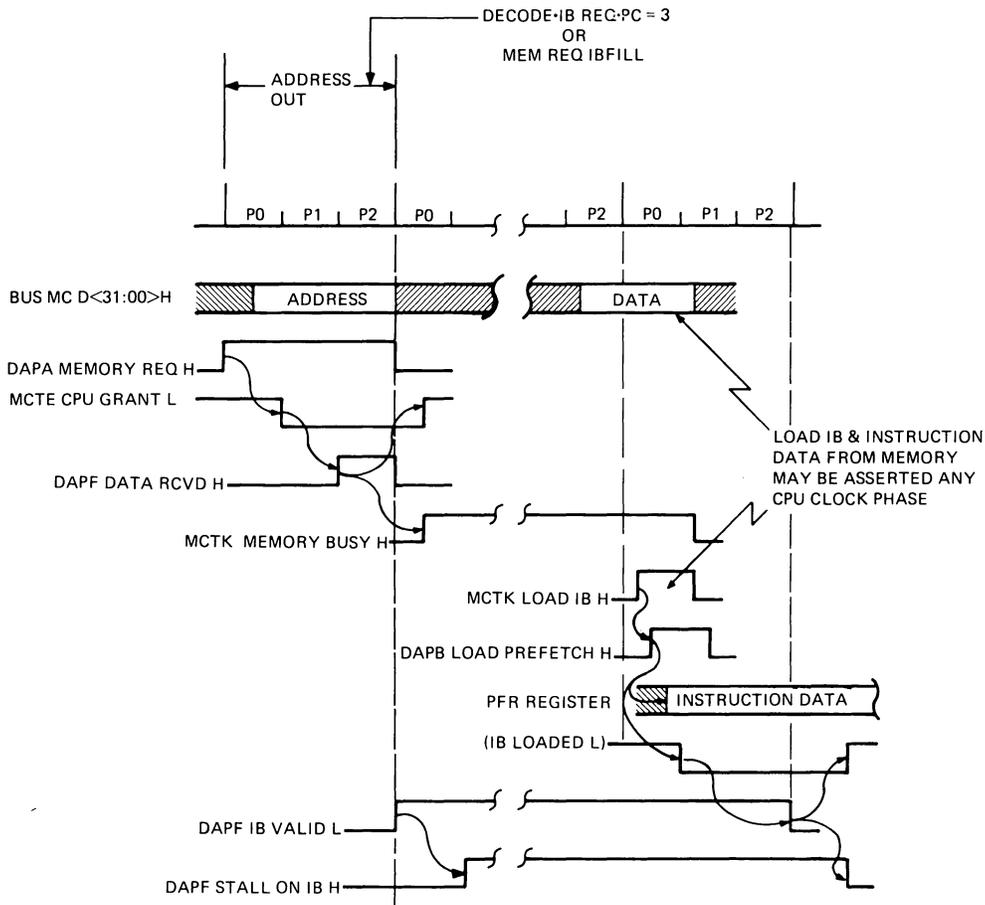
Figure 5-4   PFR Load Operation Timing Diagram

5-5

After receiving MEMORY REQ, the MCT asserts CPU GRANT on the MC bus when it is ready to accept the memory address and begin the memory reference. If CPU GRANT is not asserted immediately (during clock phase 1), the CPU clock is stalled. (CLOCK STALL is asserted in the CPU clock generator.) A stall can occur if the MCT is performing a UNIBUS operation.

Once CPU GRANT is asserted by the MCT (releasing a clock stall, if any), the CPU completes execution of the DECODE or MEM REQ microinstruction initiating the memory reference. IB VALID, a control flip-flop which (when set) indicates the PFR contains unprocessed instruction data, is cleared at this time. The IB VALID logic is shown in Figure 5-5.
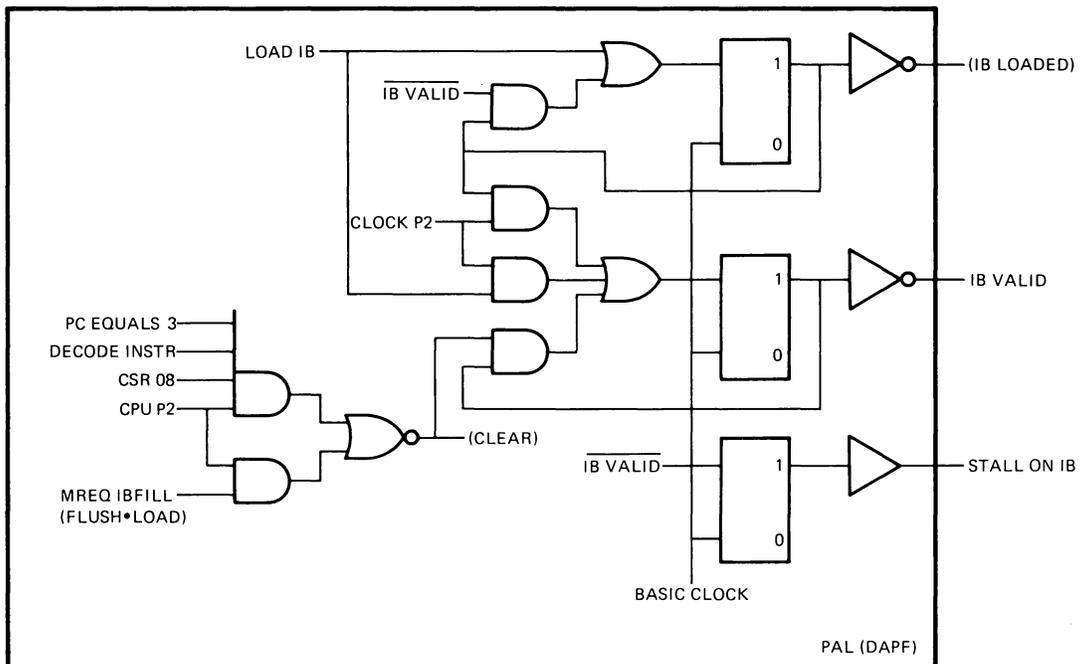
Figure 5-5   IB VALID Control Logic

Following the DECODE or MEM REQ that starts the memory reference, the CPU is free to execute microinstructions (not DECODEs) during the time that the instruction data is being fetched from memory. (Two microcycles may be executed before the instruction data can be fetched from memory and transmitted over the MC bus to the CPU.) When the data is valid on the MC bus data lines, the MCT asserts LOAD IB.

When received by the CPU, LOAD IB asserts the PFR clock (LOAD PREFETCH). This signal causes the 32-bits of instruction data to be strobed from the MC bus data lines directly into the PFR. IB VALID is also set at the end of the current microcycle to indicate that the PFR contains unprocessed instruction data. DECODE microinstructions may now be executed to remove and process the information.

As stated above, there is a two-microcycle delay before the PFR can be loaded from memory and a DECODE microinstruction executed to process the first byte of instruction data. However, this assumes no extra delay in retrieving the data due to correctable memory errors, memory refresh cycles, etc.

If a DECODE attempts to remove a byte from the PFR before the PFR is filled, the CPU clock is stalled. That is, with the memory still busy fetching the instruction data (MEMORY BUSY = 1), STALL ON IB will be set because IB VALID is cleared (refer to Figure 5-5), and ENABLE IB STALL will be asserted because data is being removed from the PFR. (ENABLE IB STALL is discussed in Paragraph 5.2.2.) These signals, together with DECODE INSTR = 1, assert CLOCK STALL in the CPU clock generator to delay execution of the DECODE microinstruction until the PFR is filled (MEMORY BUSY = 0).

If the PFR cannot be filled due to a hard error (NXM, uncorrectable read error, etc.), the MCT negates MEMORY BUSY, preventing or releasing a clock stall; but LOAD IB is not asserted and IB VALID stays cleared. Thus, DECODE microinstructions removing data from the PFR specify a jump (or skip) that is conditional on IB VALID. Then, if IB VALID is not set, a dispatch to the memory management microcode is made to recover from the error.

5.2.2 Unloading the PFR
The instruction data in the PFR is unloaded and processed by the DECODE microinstruction. In native mode, the DECODE's IB REQ bit (CSR ⟨08 = 1⟩) causes a single byte to be removed and transmitted on the 8-bit IB bus. If IB REQ is not asserted, no byte is removed and the op code register drives the IB bus. In compatibility mode, the unload is not conditional. A DECODE always removes a byte from the PFR.

Whenever the PFR is to be unloaded, CSR ⟨08⟩ = 1 or COMPAT MODE = 1 assert ENABLE IB STALL. This signal, in turn, allows one of four enable levels (ENABLE IB3 through ENABLE IB0) to be generated, causing the corresponding PFR byte to be transmitted on the IB bus. (ENABLE IB3 transmits byte 3 on the bus, ENABLE IB2 transmits byte 2, etc.) The single enable that is asserted depends mainly on the two low-order bits of the PC. This value is stored in control flip-flops, which in turn drive a decoder to generate the enables, as shown in Figure 5-6.

The flip-flops storing the two low-order bits of the PC are loaded by all MEM REQ microinstructions initiating a flush and load of the PFR (MREQ IBFILL = 1), and by all DECODE microinstructions in native mode removing a byte from the PFR (CSR ⟨08⟩ = 1). In both cases, the value of the PC + 1 is asserted on the Y bus (as explained in Paragraph 5.7), and the two low-order bus bits (BUS Y D ⟨01:00⟩) are loaded into the flip-flops at the end of the microcycle. The incremented PC is loaded at the end of the DECODE so that during the next DECODE, the value held in the flip-flops will be that of the current PC.
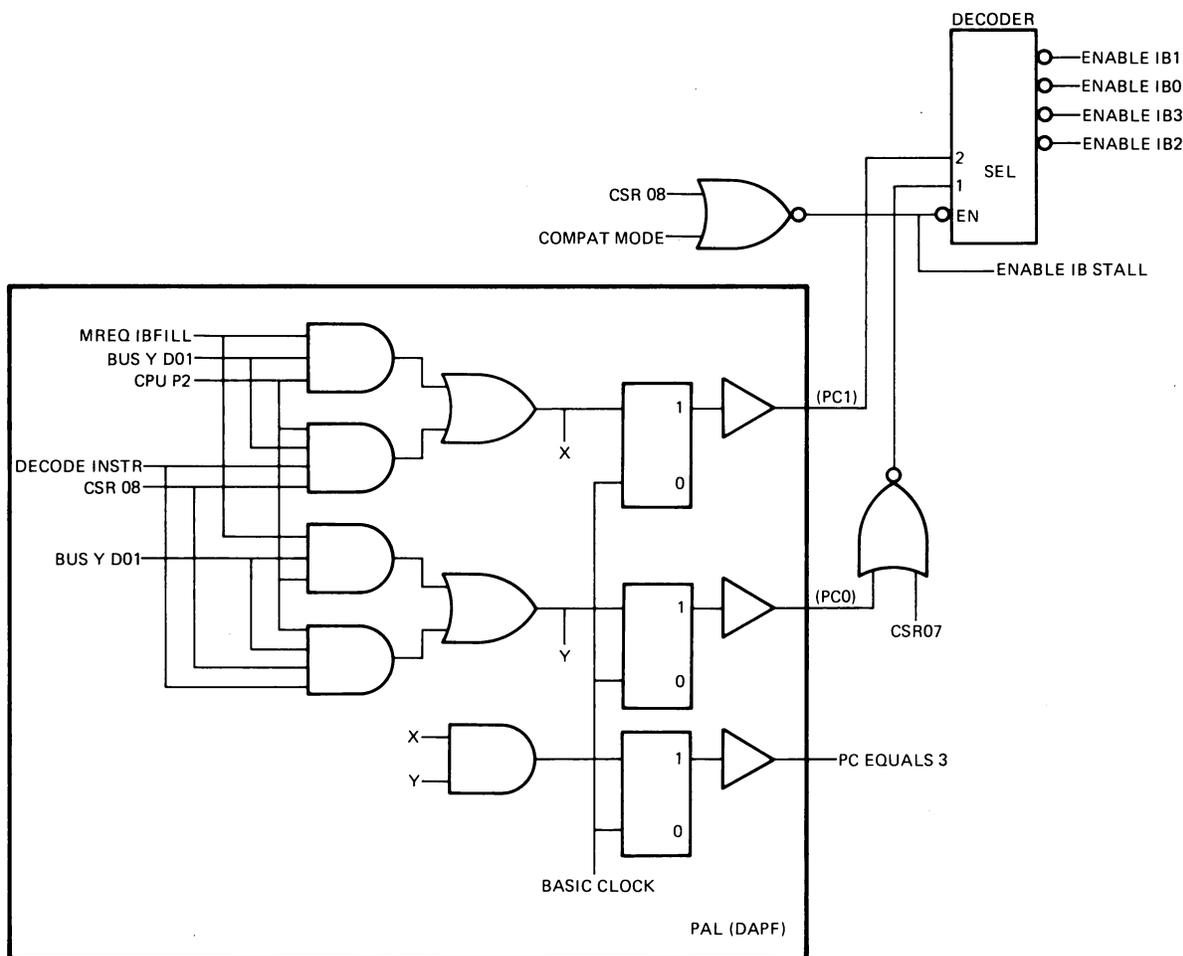
Figure 5-6   PFR Control Logic

In native mode, the ENABLE IB level asserted (and thus the byte removed from the PFR) corresponds directly to the PC value held in the flip-flops. This is also true in compatibility mode, but provision is made to select byte 1 (when the PC equals 0) and byte 3 (when the PC equals 2) if the SEL CM HI BYTE control bit (CSR ⟨07⟩) in the DECODE is asserted. Selection is shown below.

Control Flip-Flops

PC1	PC0	CSR07	PFR Enable
0	0	0	ENABLE IB 0
0	0	1 (Compat. mode only)	ENABLE IB 1
0	1	–	ENABLE IB 1
1	0	0	ENABLE IB 2
1	0	1 (Compat. mode only)	ENABLE IB 3
1	1	–	ENABLE IB 3

5-8

As the PC is incremented by successive DECODEs in native mode (i.e., CSR ⟨07⟩ always equal to zero), the corresponding PFR enables that are generated remove successive bytes from the PFR. That is, after byte 0 is removed, byte 1 is removed, followed by byte 2, etc. When byte 3 is removed, the current PC being equal to 3, control flip-flop PC EQUALS 3 (Figure 5-6) initiates an automatic refill of the PFR.

In compatibility mode, the instruction data is not byte oriented, and the PC has values equal to zero or two (instructions aligned on word boundaries). It is for this reason that the DECODE's SEL CM HI BYTE is provided, to remove and process bytes 1 and 3 of the PDP-11 instruction word.

5.3 OPCODE REGISTER (OPC)

The 8-bit OPC is loaded from the IB bus when an opcode is removed from the PFR. The load occurs during a class decode operation when the DECODE microinstruction specifies a PFR unload (EN-ABLE IB STALL = 1), the DECODE's OPC/SPEC bit (CRS ⟨04⟩) is asserted, and the DECODE's IFUNC field (CSR ⟨11:09⟩) is equal to one or three. (The class decode and other types of decode operations are discussed in Paragraph 5.7.) The OPC holds the opcode throughout the execution of an instruction.

Although loaded from the IB bus, the opcode register may also drive the IB bus. It then provides a dispatch on opcode by addressing the IRD ROM. (This is called an opcode decode operation.) The opcode register drives the IB bus whenever ENABLE IB STALL = 0; that is, whenever a DECODE is not unloading the PFR.

5.4 MAPPING ROMS

There are three mapping ROMs used in the CPU's instruction processing hardware. Two of them, the 1K × 8-bit OPCODE ROM and the 512 × 8-bit SPEC ROM, are used for microprogram dispatch purposes. The 512 × 4-bit OPC CLASS ROM (plus PAL) is used to generate data type and condition code class codes for each instruction. It also contains logic to implement the execution of branch on condition instructions.

During a DECODE microinstruction, one of the two dispatch ROMs is selected to drive the micro-sequencer's NAD bus ⟨07:00⟩. This provides the eight low-order bits of the DECODE's jump address when a jump or JSR is specified by the JCTL field. CSR ⟨17:12⟩, the DECODE's jump address field, provides the six high-order address bits.

By supplying part of the jump address, the OPCODE ROM is used to dispatch on an instruction's opcode. Similarly, the SPEC ROM is used to dispatch on an instruction's operand specifiers. This includes dispatches in compatibility mode on a PDP-11 instruction's opcode and destination fields.

With reference to the block diagram (Figure 5-1), the dispatch ROM that is selected to drive the NAD bus depends on the DECODE's OPC/SPEC bit (CSR ⟨04⟩). Microsequencer control output EN-ABLE IR ROM will be asserted when a jump is to occur, and this signal together with CSR ⟨04⟩ = 1 selects the OPCODE ROM. If ENABLE IR ROM is asserted and CSR ⟨04⟩ = 0, the SPEC ROM is selected.

In one special case, the SPEC ROM may be selected even though the OPCODE ROM has been speci-fied by the DECODE. When the DECODE's IFUNC field equals three (specifying a native mode class decode), and if an interrupt request of a priority level greater than the current CPU interrupt priority level is pending, the SPEC ROM is selected to force an 8-bit dispatch vector of all ones (FF) on the NAD bus. This provides an automatic dispatch to interrupt handler microcode after executing one in-struction, and before starting execution of the next.

This function may be disabled by MASK INTS, a signal set by a MISC microinstruction immediately preceding the DECODE. MASK INTS is set by the microprogram for a few special cases only; for example, during T-bit trap handling and certain memory management functions.

Both of the dispatch mapping ROMs are addressed by the instruction data on the IB bus and by the DECODE's IFUNC control bits. The OPC CLASS ROM/PAL is addressed by the IB bus and the COMPAT MODE control bit. Mapping ROM addressing is shown in Figure 5-7.
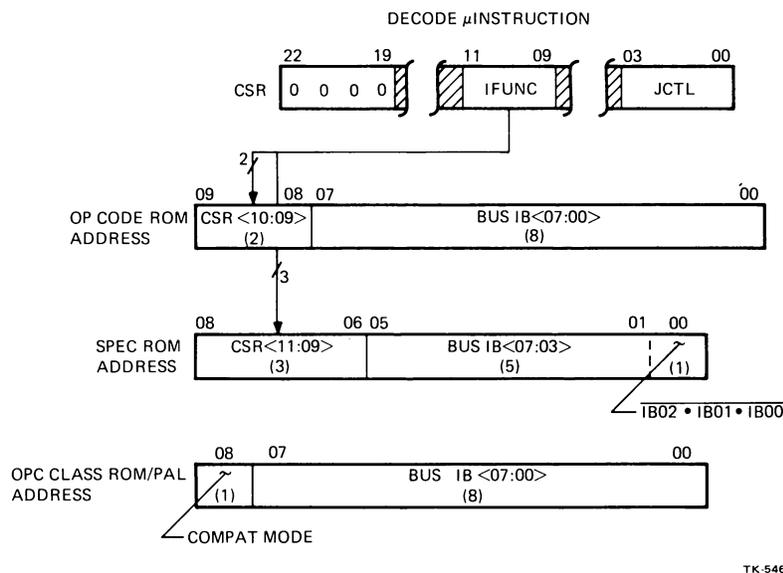
Figure 5-7   Mapping ROM Addressing

The IFUNC field provides the most significant address bits for the two dispatch ROMs. This structures the contents of the dispatch ROMs (Figures 5-8 and 5-9). In the 1K-word OPCODE ROM, the two low-order bits of the IFUNC field are used to partition the contents into four 256-word blocks. In the 512-word SPEC ROM, all three IFUNC bits are used to partition the contents into eight 64-word blocks.

Figure 5-8   OPCODE ROM

Figure 5-9   SPEC ROM

5-10

During instruction processing, the IFUNC field (which specifies the type of decode operation) selects a block of addresses in a dispatch ROM, and the opcode or specifier information on the IB bus selects a word in that block. For example, during a class decode in native mode, IFUNC = 3 selects a block of addresses in the OP CODE ROM from 300 to 3FF, labeled VAX.IRD in Figure 5-8.

The opcode on the IB bus then selects a word in the VAX.IRD block that dispatches the microprogram either to class code that is specific to that instruction (to fetch the necessary operands before instruction execution), or directly to execution code for the fast instructions (MOV, CMP, etc.). The contents of the dispatch ROMs are given in the *CPU Microcode Listing*. The types of decode operations corresponding to the various IFUNC field values are discussed in Paragraph 5.7.

The OPC CLASS ROM/PAL configuration is shown in Figure 5-10. The ROM, which is addressed by the opcode and COMPAT MODE during a class decode, outputs two 2-bit codes that define the implicit data type of the first operand, and the condition code class for each instruction. COMPAT MODE, the most significant address bit, partitions the 512-word ROM into two 256-word blocks. One block is dedicated for native mode class decodes; one for compatibility mode class decodes.
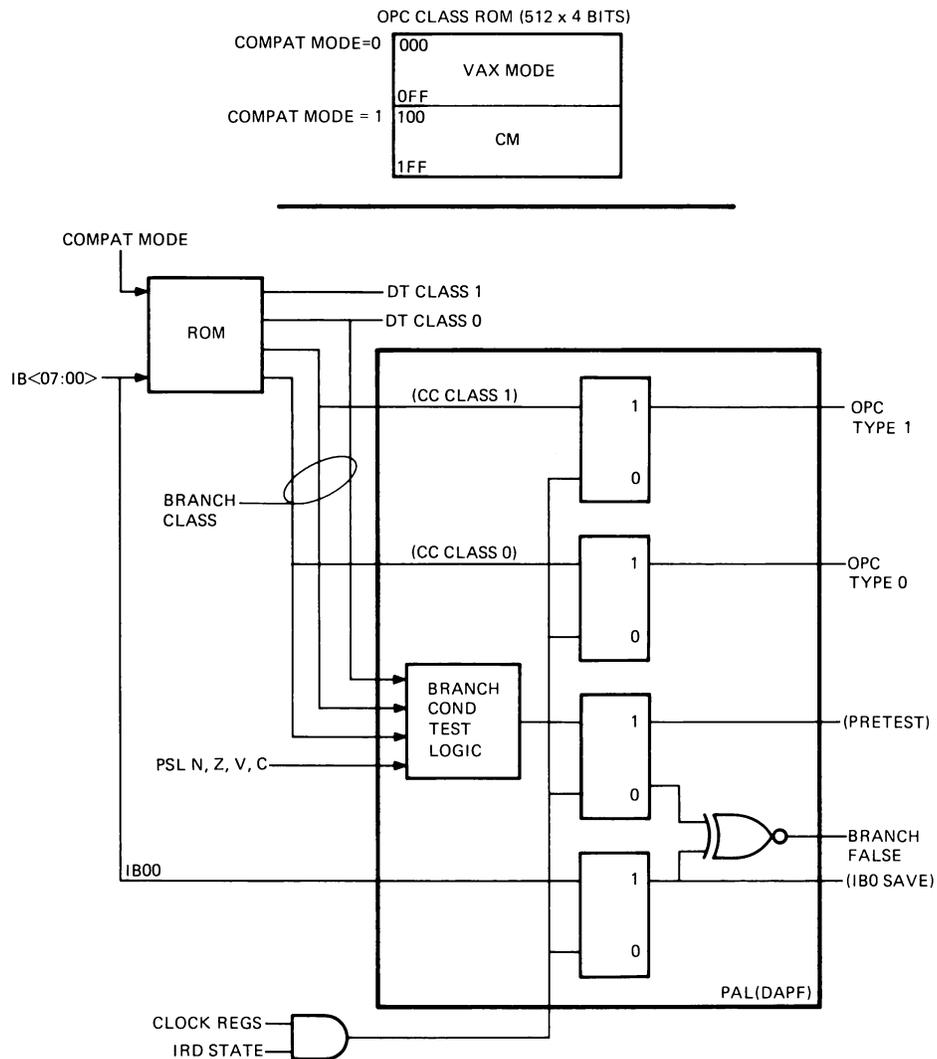
Figure 5-10   OPC CLASS ROM/PAL

5-11

Two of the four ROM outputs, DT CLASS 1 and DT class 0, specify the data type as follows.

DT CLASS 1	0	Data Type
0	0	Byte
0	1	Word
1	0	(Not used)
1	1	Longword

The DT CLASS code is stored in the size register (in the CPU's data path control) for use during execution of the instruction. The ROM outputs specifying condition code (CC) class are stored in flip-flops within the ROM/PAL itself. The PAL outputs, OPC TYPE 1 and OPC type 0, specify the CC class code as follows.

OPC TYPE 1	0	CC Class
0	0	Arithmetic ADD (copy N,Z,V,C)
0	1	Arithmetic SUB (copy N,Z,V/invert C)
1	0	CMP (XOR N with V/clear V/invert C)
1	1	Logical (clear V/load Z,N/keep previous C)

In addition to generating data type and CC class codes, the OPC CLASS ROM/PAL contains logic to test the PSL CCs during branch on condition (B) instructions. When a branch on condition op code is addressing the ROM during a class decode, three of the four ROM outputs (DT CLASS 0 and the two CC class outputs) specify the branch conditions to be tested. These are as follows.

DATA TYPE 0	CC CLASS 1	0	Branch Condition Tested
0	0	0	Z
0	0	1	C OR Z (less than or equal, unsigned)
0	1	0	V
0	1	1	C
1	0	0	N OR Z (less than or equal)
1	0	1	(N XOR V) OR Z (less than or equal, compatibility mode)
1	1	0	N
1	1	1	N XOR V (less than or equal, compatibility mode)

PAL logic then tests the condition specified against the current state of the PSL CCs. To do this, the state of the specified CC bit (or logical combination of bits) is loaded into a PAL flip-flop (PRETEST) at the end of the microcycle. Also, the low-order bit of the op code is stored in a flip-flop (IB0 SAVE) at the end of the microcycle. The op code bit is stored because it implicitly specifies on which state of the branch condition the branch is to occur. For example, an opcode of 12 specifies a branch on $Z = 0$ while an opcode of 13 specifies a branch on $Z = 1$.

To complete the test, the outputs of the two flip-flops are XORcd to negate BRANCH FALSE when a branch condition has been met. BRANCH FALSE is a microsequencer jump or skip condition which may be tested by the microcode.

5.5 REGISTER DESTINATION (GPR DEST) CONTROL BIT

The GPR DEST control bit, a microsequencer skip and jump condition, is incorporated into the instruction processing hardware in order to speed execution of instructions specifying a GPR as an operand destination (i.e., a register mode address).

With reference to Figure 5-11, GPR DEST is a flip-flop set at the end of the DECODE microinstruction when the DECODE's LD RDEST bit (CSR ⟨05⟩) is asserted, and provided REGISTER MODE is true. (GPR DEST is cleared otherwise.) REGISTER MODE will be true during a specifier decode operation when the instruction data specifies register mode addressing.
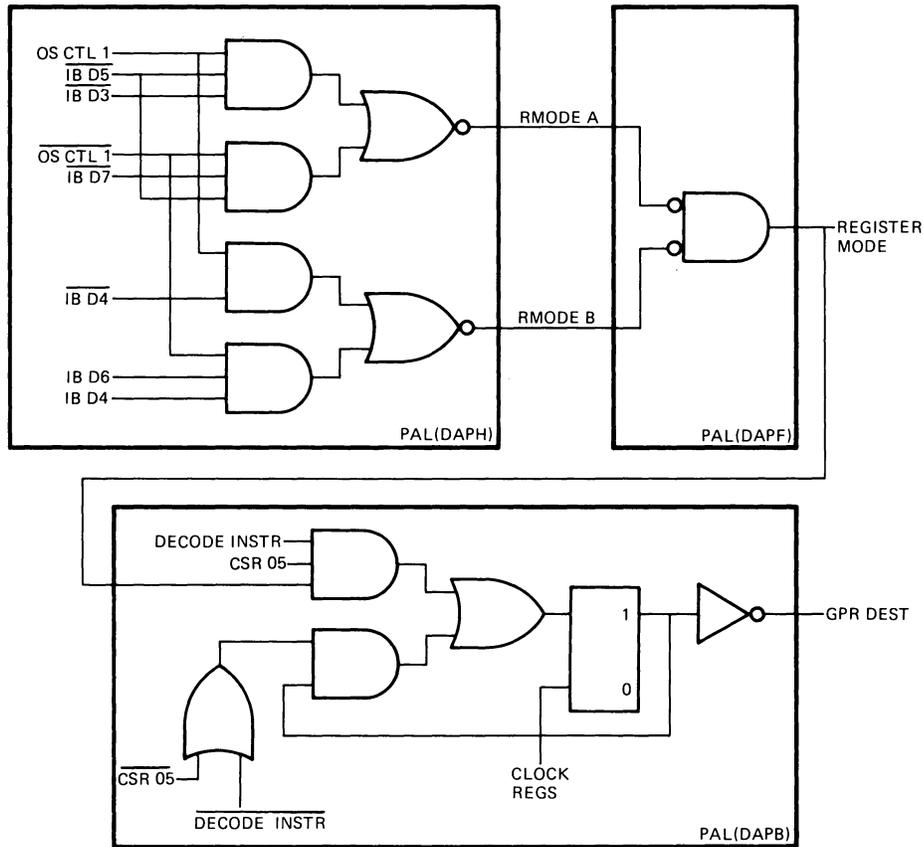
Figure 5-11   GPR DEST Control Logic

The instruction data specifying the address mode is sampled on the IB bus by RMODE A and B. Both of these signals will be true to assert REGISTER MODE when the specifier's address mode field is equal to five in native mode, or when the destination's address mode field is equal to zero in compatibility mode. One of the OS register control bits (OC CTL 1) is used to gate the IB bus data. During a specifier decode in native mode, OS CTL 1 = 0; in compatibility mode, OS CTL 1 = 1.

5.6 REGISTER BACKUP MASK FLAG

In VAX-11/730 systems, it is required that all instructions which evaluate specifiers be restartable. The register backup mask flag, which is a microsequencer skip function, aids the microprogram in restarting the instruction. The RBKUP FLAG flip-flop is shown in Figure 5-12.
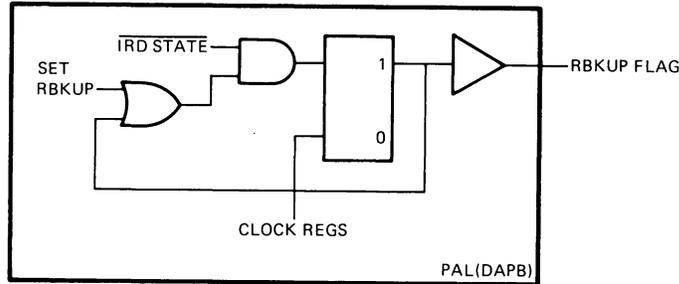
Figure 5-12   RBKUP FLAG Control Logic

RBKUP FLAG is set by a MISC microinstruction that has its function 1 field equal to seven. The microinstruction asserts SET RBKUP, which sets the flip-flop at the end of the microcycle. RBKUP FLAG is set by the microprogram the first time a bit in the register backup mask (2901A working register 3) is set; that is, the first time a GPR is modified by the microprogram during execution of the instruction. (When a GPR is modified during specifier evaluation, the previous contents are stored in local store in case a restart is necessary, and a bit corresponding to that GPR is set in the low order 16-bit portion of the register backup mask.)

Following an instruction's execution, RBKUP FLAG is cleared, indicating the register backup mask is no longer valid. It is cleared by IRD STATE during the class decode operation for the next instruction.

5.7 INSTRUCTION DECODE OPERATIONS

The instruction decode operations performed in the CPU are executed by the DECODE microinstruction. A flow diagram for the DECODE is shown in Figure 5-13.

Certain data path operations are performed by the DECODE no matter what control bits in the microinstruction are asserted. For example, the address of the native mode PC is forced in local store (address = 10) and the local store latches are enabled to read the contents onto the D bus and, in turn, the MC bus. (This provides the memory address for an automatic refill of the PFR if the DECODE's IB REQ bit is asserted and the last PFR byte is being removed.)

Also, the 2901s are set up to increment the native mode PC asserted on the D bus and transmit the incremented value on the Y bus. (The new PC value may then be written into local store and the low-order bits stored in flip-flops if IB REQ = 1.) The operations on the native mode PC are allowed to occur for any DECODE, even though the machine may be in compatibility mode, because IB REQ is equal to zero for all compatibility mode instruction decode operations.

All other operations done by the DECODE depend on which control bits it asserts. The IFUNC field, together with the OPC/SPEC bit, provide the main control and determine the basic types of decode operations that are executed. The basic operations, and the control bits asserted for these operations, are listed in Tables 5-1 and 5-2.
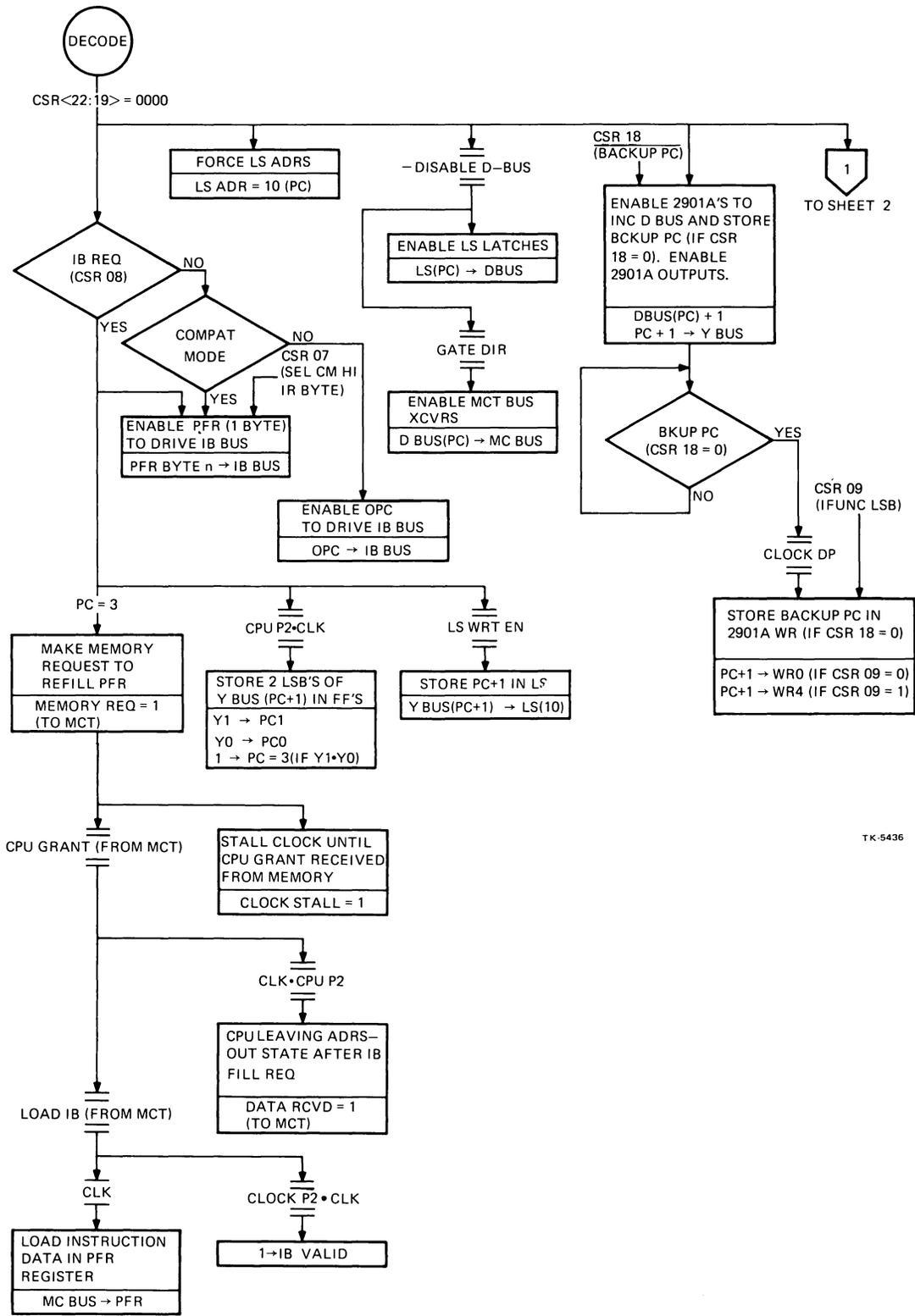
5-14

Figure 5-13 DECODE Microinstruction Flow Diagram
(Sheet 1 of 2)
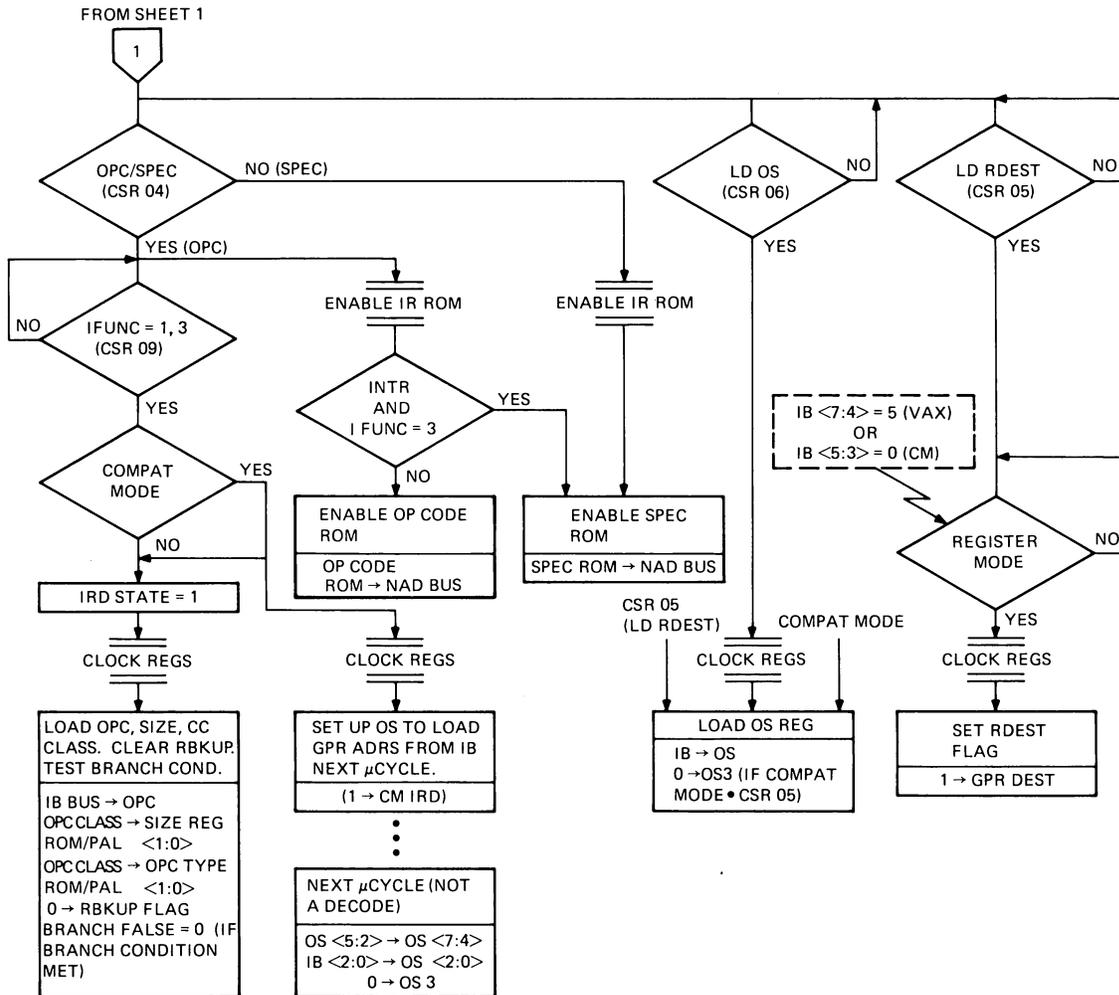
Figure 5-13   DECODE Microinstruction Flow Diagram
(Sheet 2 of 2)

Table 5-1   DECODE Control Bits for Native Mode Instruction Decodes

Decode Operation	BKUP PC (CSR<18>)	IB REQ (CRS<08>)	SEL CM HI BYTE (CSR<07>)	OPC/ SPEC (CSR<04>)	LOAD OS (CSR<06>)	LOAD RDEST (CSR<05>)	IFUNC (CSR<11:09>)
Class (VAX IRD)	0	1	0	1	0	0	3
Specifier	–	1	0	0	1	1	0/1/2/4/5
Opcode	1	0	0	1	0	0	2
Get-Byte	1	1	0	–	1	0	–

Table 5-2   DECODE Control Bits for Compatibility Mode Instruction Decodes

Decode Operation	BKUP PC (CSR<18>)	IB REQ (CRS<08>)	SEL CM HI BYTE (CSR<07>)	OPC/ SPEC (CSR<04>)	LOAD OS (CSR<06>)	LOAD RDEST (CSR<05>)	IFUNC (CSR<11:09>)
Class (CM IRD)	–	0	1	1	1	0	1
Single Operand	–	0	0	0	1	0	7
Destination	–	0	0	0	1	1	6
No Operand	–	0	0	1	1	0	0

5.7.1 Class Decodes

The class decode is the first instruction decode operation performed on a native mode or compatibility mode instruction. The DECODE microinstruction, a jump if IB VALID, removes the opcode from the PFR and uses it to address the OPCODE ROM and the OPC CLASS ROM/PAL. (The opcode is also loaded into the opcode register.) The OPCODE ROM then causes a dispatch to the appropriate class or execution flow for the instruction. Also, the OPC CLASS ROM/PAL supplies codes indicating the instruction's condition code class and the implicit data type for the first operand.

With reference to the DECODE flow diagram (Figure 5-13), IB REQ (CSR ⟨08⟩) is asserted for a class decode in native mode to remove a byte (the opcode) from the PFR at the beginning of the micro-cycle. Byte 0, 1, 2 or 3 may be removed, depending on the two low-order bits of the current PC.

IB REQ also updates the PC at the end of the microcycle by writing the PC value, now incremented and asserted on the Y bus, into local store. And, if the DECODE's BACKUP PC control bit is negated (CSR ⟨18⟩ =0), the incremented PC is stored in a 2901A working register location. Also, at this time, the low-order bits of the incremented PC are stored in the flip-flops that control PFR byte selection.

In compatibility mode, IB REQ is always negated. However, a DECODE always removes a byte from the PFR in this machine mode. Without SEL CM HI BYTE asserted (CSR ⟨07⟩ = 0), byte 0 or byte 2 would be removed from the PFR because 0 or 2 are the only values of the two low-order PC bits in compatibility mode. (PDP-11 instructions are aligned on word boundaries, and the PC is not updated automatically when a byte is removed from the PFR.) During a class decode, however, SEL CM HI BYTE is asserted (CSR ⟨07⟩ = 1) to select either byte 1 or byte 3, which contains all or part of the opcode for most PDP-11 instructions.

The IFUNC field equals three for native mode class decodes. It is equal to one for compatibility mode class decodes. The OPC/SPEC bit is asserted (CSR ⟨04⟩ = 1) in either mode. With the VAX or PDP-11 opcode (or other data identifying the type of PDP-11 instruction) removed from the PFR and asserted on the IB bus, OPC/SPEC = 1 normally selects the OPCODE ROM to provide part of the DECODE's jump address. The OPCODE ROM is addressed by the opcode and the IFUNC field to provide a dispatch address from the VAX.IRD or CM.IRD areas in the ROM (Figure 5-8).

The OPCODE ROM is not always selected during a class decode. If an interrupt is to be serviced in native mode, the SPEC ROM is selected to cause a dispatch to interrupt handling microcode. As for the OPCODE ROM, the opcode and IFUNC field provide the SPEC ROM address. Consequently, IFUNC = 3 selects the ROM's interrupt trap block (Figure 5-9). Furthermore, because interrupt handling is not dependent on the opcode (which selects a location within the interrupt trap block), all locations in the block contain the same dispatch address (FF).

Whereas the OPCODE and SPEC dispatch ROMs are enabled only during a DECODE (by the OPC/SPEC bit and ENABLE IR ROM from the microsequencer), the ROM outputs from the OPC CLASS ROM/PAL are always asserted. However, the ROM outputs are not stored in flip-flops until the class decode operation; that is, when the opcode is addressing the ROM, and when OPC/SPEC = 1 and FUNC = 1 or FUNC = 3 asserts IRD STATE. The IRD STATE signal loads SIZE REG 1 and 0 and OPC TYPE 1 and 0 from ROM outputs at the end of the microcycle. IRD STATE also causes BRANCH FALSE to be asserted at this time if the opcode is for a branch on condition instruction, and the branch condition has not been met.

Another flip-flop set by IRD STATE at the end of the microcycle (CM IRD) sets up the data path's OS register for a special function in compatibility mode. Ordinarily, the loading of the OS register from the IB bus is controlled by the DECODE's LOAD OS (CSR ⟨06⟩) and LD RDEST (CSR ⟨05⟩) control bits. Two control signals, OS CTL 1 and 0, are generated to perform the functions listed in Table 5-3. For example, during the compatibility mode class decode, LD OS is asserted and LD RDEST is negated, which causes OS CTL 1 and 0 to equal 00.

Table 5-3 OS Control by the DECODE Microinstruction

LOAD OS (CSR06)	LOAD RDEST (CSR05)	COMPAT MODE	OS CTL 1	OS CTL 0	Function
0	–	–	1	1	NOP OS⟨7:0⟩
1	1	1	1	0	IB⟨7:0⟩ to OS⟨7:0⟩ (Clear OS3)
1	–	0	0	0	IB⟨7:0⟩ to OS⟨7:0⟩
1	0	1	0	0	IB⟨7:0⟩ to OS⟨7:0⟩
NEXT STATE (CM IRD = 1 AND not DECODE)			0	1	OS⟨5:2⟩ to OS⟨7:4⟩ OS0 to OS2 IB⟨7:6⟩ to OS⟨1:0⟩ (Clear OS3)

This results in a direct load of the opcode information into OS at the end of the microcycle. It is at the end of the microcycle following the class decode, if the next microinstruction is not a DECODE, that the CM IRD flip-flop performs its special function. OS CTL 1 and 0 are forced to a value of 01 to cause the bits in OS ⟨5:2⟩ to be shifted into OS ⟨7:4⟩, the bit in OS ⟨0⟩ to be shifted into OS ⟨2⟩, the low-order bits in the PFR's low byte (IB bus bits 7 and 6) to be loaded in OS ⟨1:0⟩, and OS ⟨3⟩ to be cleared.

This operation on the data in OS has significance for only the double-operand class of PDP-11 instructions. As shown in Figure 5-14, the data is manipulated to assemble the source register address into the low-order bits of OS. Following the class decode, a JUMP microinstruction ORing the jump field with OS is done, masking out OS ⟨4⟩ and OS ⟨0⟩.
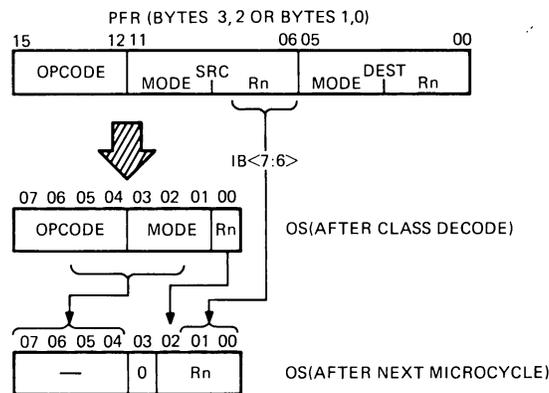
Figure 5-14 Assembly of GPR Number in OS Following Class Decode in Compatibility Mode

5-19

This results in a jump to microcode that evaluates the mode. At the end of the JUMP, the hardware shifts and reloads OS so that in the microcode's source evaluation flows, the specified GPR can be read from local store. OS ⟨3⟩ is cleared during the shift and reload operation because four OS bits (OS ⟨3:0⟩) are used to address local store, and the PDP-11 GPR number is only three bits long.

> **NOTE**
> When the special function to load the GPR number in OS is used, CSR ⟨07⟩ must be zero in the microinstruction following the DECODE. This is so that the GPR address bits in the low-order PFR byte will be loaded into OS. If CSR ⟨07⟩ were equal to one, the high order byte would be selected.

5.7.2 Specifier Decodes

Specifier decode operations are used mainly to dispatch from the class flows in the microcode to the specifier flows. The DECODE microinstruction removes the specifier information from the PFR and uses it to address the SPEC ROM. (The specifier information is also loaded in the data path's OS register.) The SPEC ROM outputs then provide a dispatch to the specifier flow that fetches (and stores) operands for that class of instruction.

As in the class decode in native mode, a DECODE doing a specifier decode in native mode has IB REQ asserted, which removes a byte from the PFR (any byte depending on the PC) and increments and restores the PC as previously described. Unlike a class decode, however, the SPEC/OPC bit is negated, and both LD OS and LD RDEST are asserted.

With reference to Figure 5-13, OPC/SPEC = 0 selects the SPEC ROM to provide part of the jump address for the decode. The jump address is selected from one of eight blocks of data in the ROM. The block selected depends on the DECODE's IFUNC field, as shown in Figure 5-9. The various IFUNC field values for the specific decode in native mode are 0, 1, 2, 4 and 5. The value is determined by the type of specifier information to be decoded (integer operand, address of operand, floating point operand, etc.). This is a function of the opcode, and thus the result of the previous class decode.

The assertion of LD OS during the native mode specifier decode causes a direct load of the specifier data on the IB bus into OS. If the specifier contains a GPR number, the register address then provides an index for addressing the GPR location in local store. LD RDEST = 1 sets GPR DEST, a microsequencer skip and jump condition, if the specified address mode is register mode.

During a specifier decode in compatibility mode (IB REQ = 0, OPC/SPEC = 0), either byte 0 or 2 is removed from the PFR. The byte selected depends solely on the PC, as SEL CM HI BYTE is equal to zero for specifier decodes. The IFUNC field values in compatibility mode are equal to either six or seven. IFUNC = 6 is for decoding the destination field of PDP-11 instructions; IFUNC = 7 dispatches on the low-order opcode bits (not the destination field) of the single-operand class of PDP-11 instructions.

As in native mode, LD OS is asserted for the compatibility mode specifier decode. Furthermore, LD RDEST is asserted to test for register mode if IFUNC = 6; that is, when the GPR number loaded into OS ⟨3:0⟩ is to be used for subsequent addressing of the GPR in local store. For this type of decode operation, OS⟨3⟩ (the low-order mode bit) is cleared because the PDP-11 GPR address is only three bits. OS CTL 1 and 0 have a value of 10 to perform this function (Table 5-3).

5.7.3 Other Decode Operations

There are three other basic decode operations, two occurring in native mode and one in compatibility mode.

The opcode decode in native mode does not remove a byte from the PFR (IB REQ = 0). Instead, the opcode stored in the opcode register during the class decode is used to address the OPCODE ROM (OPC/SPEC = 1). This type of decode operation (IFUNC = 2) is used to dispatch from the class flows to the execution flows for a specific opcode.

The get-byte decode operation in native mode removes a byte from the PFR (IB REQ = 1), but the data is not used to address the dispatch ROMs. (No jump or JSR is done.) Instead, the instruction data (displacements, immediate data, etc.) is loaded directly into OS (LD OS = 1). Once in OS, the instruction data may be processed in the data path.

The last basic decode operation is done in compatibility mode. It evaluates PDP-11 instructions having no operands, such as HALT, RTT, etc. (It is also used to evaluate the SWAB instruction which has one operand, and the RTS instruction which specifies a register number.) IFUNC = 0 and the instruction data in PFR bytes 0 or 2 are used to address the OPCODE ROM (OPC/SPEC = 1). The instruction data is also loaded into OS (LD OS = 1, LD RDEST = 0).

6.1 INTRODUCTION

The major components in the data path are the 2901A data processor, the local store, and the operand specifier (OS) register. They connect between the Y bus and the D bus, as shown in Figure 6-1. Other logic elements include the condition code registers, the data type control (DT CTL), the sign extension control (SXT CTL), the register read/write control (REG R/W CTL), and the D bus multiplexer.

The 2901A data processor consists of eight cascaded 2901A 4-bit microprocessor slices, configured for carry look-ahead and external shift control. The principal elements in the 2901As are a 16-location RAM, a high-speed ALU, and a separate shiftable holding register called the Q register. The RAM locations are used as CPU working registers. The ALU, in conjunction with the working registers and the Q register, performs the arithmetic and logical functions necessary to implement the CPU instruction set. Data enters the 2901As from the D bus; 2901A output data is transmitted on the Y bus. The output data is either the ALU output or the contents of a RAM (working register) location.

The local store (LS) is a large (256 × 32-bit) RAM that contains the GPRs visible to the program, the backup GPRs, several of the privileged processor registers, and the many constants and masks used by the microprogram. In addition, it contains several locations used for temporary data storage. The local store may be addressed directly by the microprogram. The OS register may supply the four or five low-order bits of address. This allows the OS register contents to be used as an index when accessing certain local store locations such as the GPRs, masks, etc. Data is read from local store onto the D bus; it is written into local store from the Y bus.

As with local store, the 8-bit OS register is read onto the D bus and loaded from the Y bus. However, it can also be loaded from the IB bus under control of the instruction processing hardware. It is this second load path that implements the principal function of the OS register; that is, it provides an entry point into the CPU's data path for instruction data removed from the PFR. From the OS register, the instruction data may be used to address local store, and it may be transferred to the 2901As for data processing.

The condition code registers consist of the PSL condition codes (N, Z, V and C) visible to the program running in the CPU, and the ALU condition codes (N, Z, V, and C) which are provided for testing at the microprogram level. The ALU CC register stores the result indicators generated by the 2901A data processor (sign, overflow, etc.), and they may be loaded at the end of all microinstructions that use the 2901A data processor for arithmetic and logical operations.

The ALU CCs are also microsequencer skip and jump conditions, and may be tested by the next microinstruction. At the end of an instruction's execution, the ALU CCs may be copied into the PSL CCs to indicate to the CPU program the result of the arithmetic or logical operation performed. Both the ALU and PSL CCs can be read onto the D bus. The ALU CCs, in addition to being conditioned by the 2901A data processor, can be loaded from the Y bus.
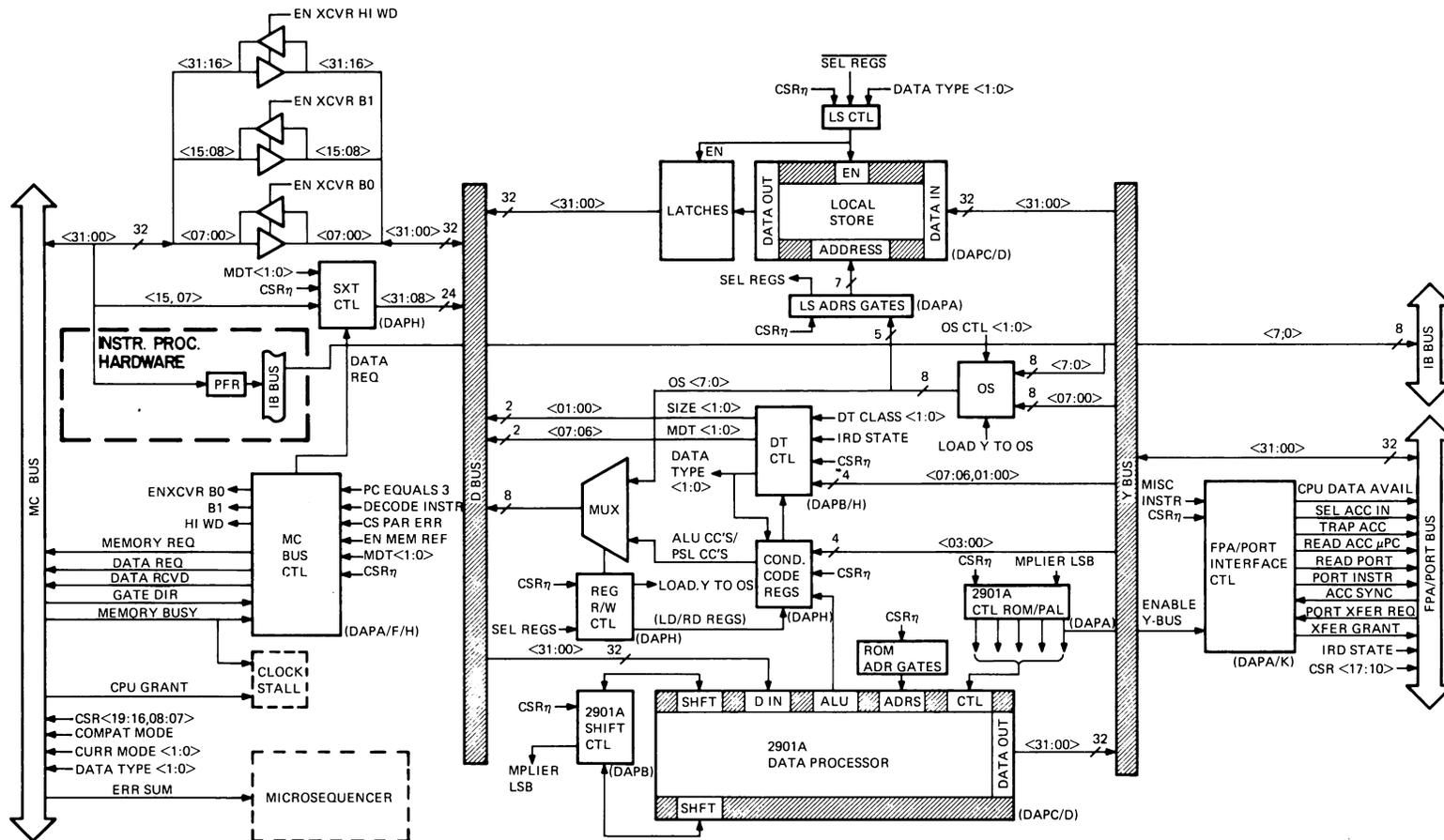
Figure 6-1 Data Path Block Diagram

The major logic element in the data type control is the size register, which is loaded with a code indicating the implicit data type (byte, word, or longword) of an instruction's first operand during the class decode operation. The size register may also be loaded with an arbitrary value at any time during an instruction's execution.

Although data transfers within the data path are generally 32 bits, the size of some transfers are limited when the size register and (in some cases) the current microinstruction specify a word or byte operation. For example, the size of the data written into local store is a function of the data type for some microinstructions. The size register in the data type control may be read onto the D bus and may be loaded from the Y bus.

The function of the sign extension control is to append sign bits to word or byte data received from the memory controller (MCT). Read data from the MCT is gated from the MC bus onto the D bus and into the 2901As for data processing. At this time, to aid in the data processing, the sign extension control samples the sign bit in the byte or word and transmits a copy onto all the high-order D bus data lines not carrrying data. The sign extension control also appends sign bits when OS register data is read onto the D bus and into the 2901A data processor.

The OS and condition code registers are read onto the D bus by way of the D bus multiplexer. Selection control is by the register read/write control, which also controls the direct read of the size register (and others) onto the D bus, as well as the loading of the OS, condition codes, and size register from the Y bus.

6.2 BASIC DATA PATH TRANSFERS
The basic transfers of data between the logic elements in the data path are shown in Figure 6-2.
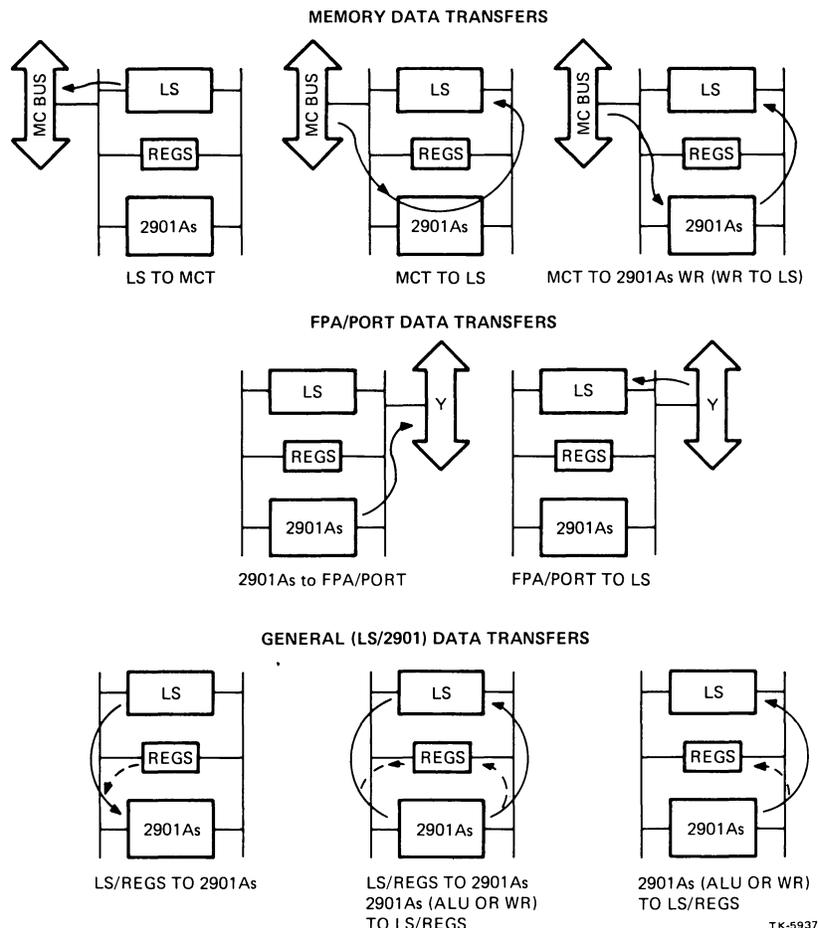
Figure 6-2   Basic CPU Data Transfers

6-3

Addresses and write data to be transferred to the the memory controller (MCT) are read from local store onto the D bus and transmitted over the MC bus. Read data from the MCT is gated from the MC bus onto the D bus and passed through the 2901A data processor (unmodified) to local store. The read data can also be stored in a 2901A working register. In the latter case, the current contents of the working register are transmitted on the Y bus and written into local store before data from the MCT is loaded.

The address information transferred to the MCT is read from local store by the MEM REQ micro-instruction. Other data transfers from and to the MCT are by the MOVE microinstruction.

Data transfers from and to the FPA or port device are over the Y bus. Transfers from the FPA or port device are made by the MOVE microinstruction, which loads the data directly into local store. Transfers to the FPA or port device are by the MISC/PORT microinstruction, which reads the data onto the Y bus from a 2901A working register location.

The other basic transfers in the CPU data path are between local store, or a discrete register such as the OS or a condition code register, and the 2901A data processor.

In one general type of transfer, D bus data from local store or a CPU discrete register is passed through the 2901A ALU, and optionally loaded into the 2901A's internal Q register or an internal working register. The path through the ALU allows the data to be processed independently, or in combination with the current contents of the Q register or a working register, during the data transfer.

When D bus data is processed, the ALU output may be transmitted on the Y bus. This allows the second general type of transfer. That is, from the Y bus, the processed data may be loaded back into the local store location or the discrete register that was read during the first part of the transfer. The 2901A data processor output may also be the contents of a working register, and this data may be loaded into local store or a discrete register as well.

During the third general type of data path transfer, D bus data is not processed in the 2901As. The contents of a previously loaded 2901A internal register is processed instead. For example, the Q register and/or a working register may be processed in the ALU and the ALU output (or a working register) transmitted on the Y bus. The 2901A output data is then loaded into local store or a discrete register. The three general types of data path transfers are initiated by the MOVE and BASIC micro-instructions.

Other data transfers in the data path are internal to the 2901A data processor only. No D bus data is processed, and no Y bus data is loaded into local store or a discrete register. Only working register and Q register data is processed in the 2901As. These operations are initiated by the EXTENDED micro-instruction.

6.3   BASIC DATA PATH TIMING
Basic data path timing is shown in Figure 6-3. DISABLE D-BUS, LS WRT EN, CLOCK REGS, and CLOCK DP perform the major clocking functions.

DISABLE D-BUS disables the transmission of all data on the D bus at the beginning of the microcycle (T0 to T45). This is to prevent overloads and possible damage to tri-state output circuits during the interval just after T0. At this time, one set of D bus output circuits (such as the local store latches) could be turning on, and another set (such as the MC bus transceiver outputs) could be in the process of turning off.

6-4
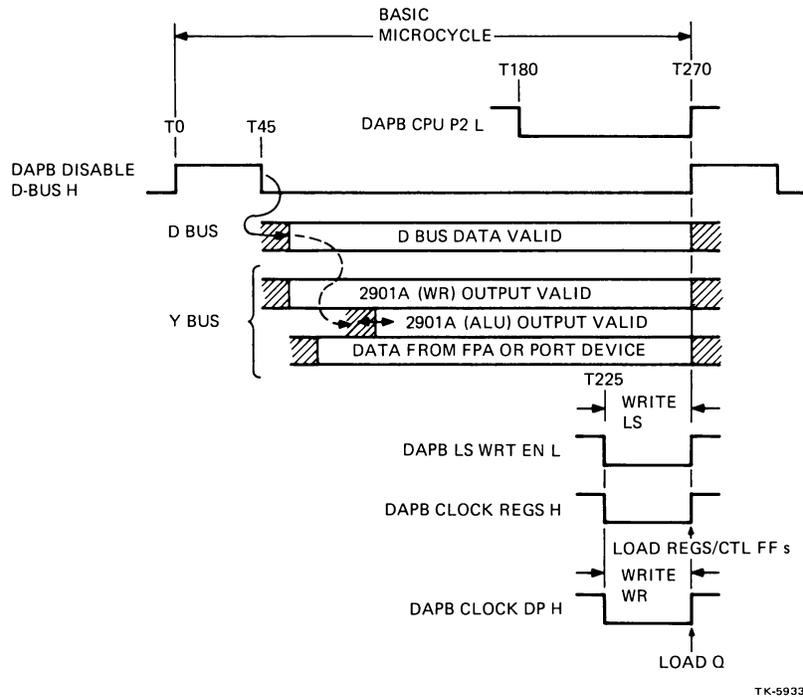
Figure 6-3 Basic Data Path Timing

No more than one tri-state output circuit is allowed to drive the D bus at any one time.

LS WRT EN is the local store write clock. If enabled by the current microinstruction, the write occurs at the end of the microcycle (from T225 to T270) when LS WRT EN is true (low). Also, at the end of the microcycle, the trailing edge of CLOCK REGS clocks the discrete registers in the CPU, as well as the majority of the data path control logic.

The 2901A data processor is clocked by CLOCK DP. When CLOCK DP is true (low), the 2901A ALU outputs may be written into a 2901A working register at the same time that local store is written. The ALU outputs may also be loaded into the 2901A's internal Q register. The Q register load occurs at the trailing edge of CLOCK DP. As for the other data path operations, the loading of the working registers and the Q register are controlled by the current microinstruction.

6.4 2901A DATA PROCESSOR
The CPU's data processor is made up of eight parallel-connected 2901A 4-bit data processor slices. Each 2901A contains a 16-word × 4-bit RAM, a high-speed ALU, a 4-bit Q register, and associated shifting, decoding, and multiplexing circuitry. A detailed block diagram of the 2901A is shown in Figure 6-4. Input/output pin definitions are given in Table 6-1.

6-5
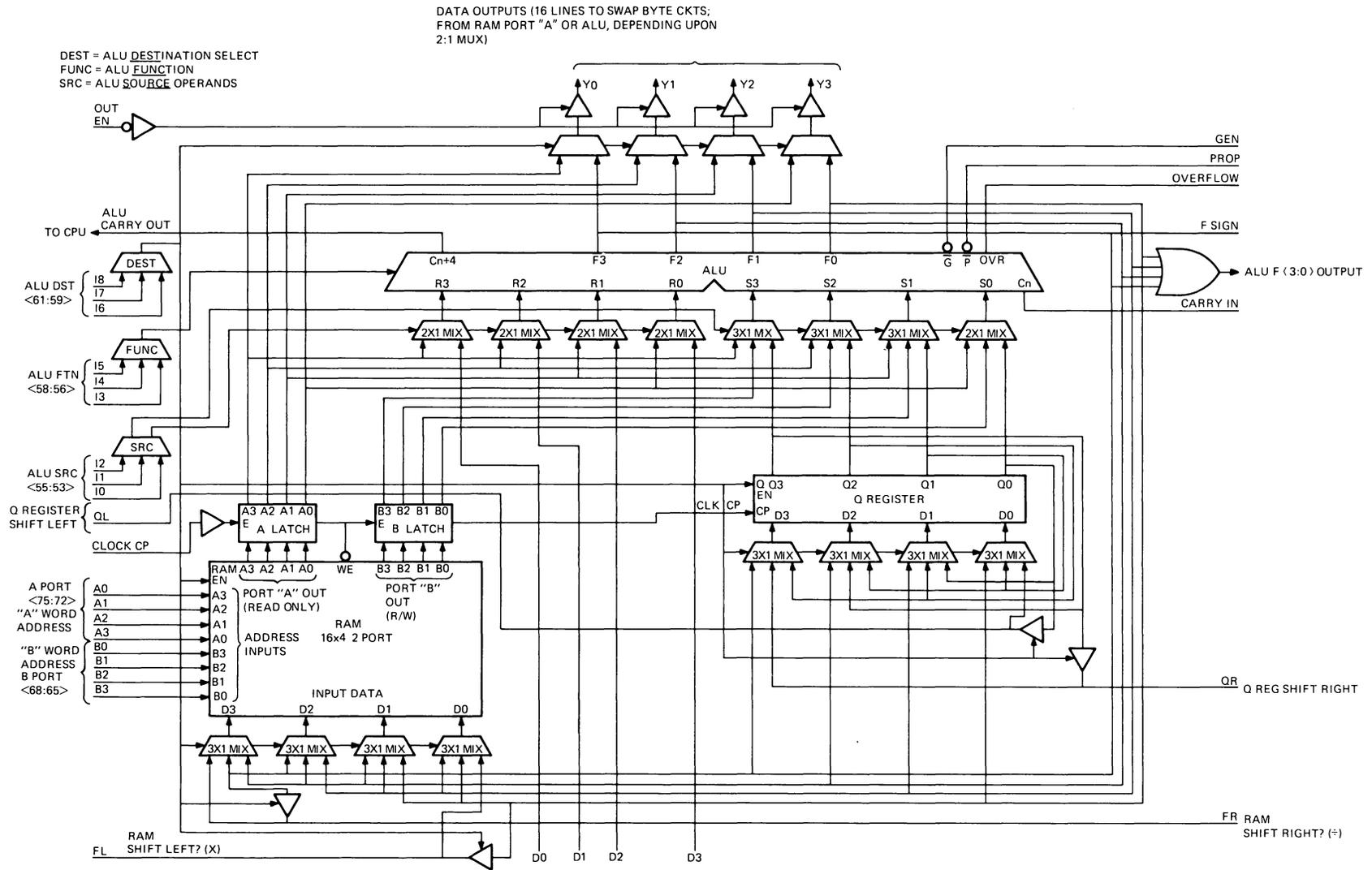
Figure 6-4 2901A Microprocessor Slice – Detailed Block Diagram

Table 6-1   2901A Input/Output Pin Definitions

Pin(s)	Function
D0⟨:3⟩	Data inputs. May be selected as data source for ALU depending on source control. D0 is the most significant bit.
Y⟨0:3⟩	Data outputs (tri-state). When enabled, output data is from either RAM (port A) or ALU depending on destination control. Y0 is the most significant bit.
OUT EN	Data output enable. When low, Y outputs are active (high or low). When high, Y outputs are off.
A⟨0:3⟩	RAM address inputs, port A.
B⟨0:3⟩	RAM address inputs, port B.
I⟨2:0⟩	Source control inputs. Select either D inputs, RAM (port A), or zeros as one set of data inputs to ALU; select either RAM (port A or port B), Q register, or zeros as the other set of data inputs to ALU.
I⟨5:3⟩	Function control inputs. Select the arithmetic or logical operation performed by the ALU.
I⟨8:7⟩	Destination control inputs. Select RAM (port A) or ALU as Y data outputs, and determine if (and what) data is deposited in the Q register and the RAM. Input to the Q register is from the ALU or the Q register itself (shifted left or right). Input to the RAM is from the ALU (unmodified or shifted left or right).
QR	Shift data input/output. Input to most significant bit of Q register during right shift operations; output from most significant bit of Q register during left shift operations.
QL	Shift data input/output. Input to least significant bit of Q register during left shift operations; output from least significant bit of Q register during right shift operations.
FR	Shift data input/output. Similar to QR, but at the most significant bit of the RAM.
FL	Similar to QL, but at the least significant bit of the RAM.
PROP,GEN	Carry propagate and generate outputs for use by carry look-ahead circuits.
CARRY IN	Carry input to ALU.
CARRY OUT	Carry output from ALU.
OVERFLOW	Overflow output from ALU. Indicates that the result of arithmetic two's complement operation has overflowed into the sign bit.

Table 6-1 2901A Input/Output Pin Definitions (Cont)

Pin(s)	Function
F = 0	Open collector output that indicates all four ALU outputs are equal to zero.
F SIGN	Sign bit output. The most significant ALU output bit.
CLOCK	Clock input. The RAM outputs are valid when the clock is high; the RAM is written when the clock is low. The Q register is loaded on the clock's low-to-high transition.

Connected together, the eight 2901As provide a 32-bit wide CPU data processing element. A simplified block diagram is shown in Figure 6-5.

Only six of the 16 32-bit RAM locations are currently used, and these are used only as working registers to hold data temporarily for a variety of functions. The 32-bit Q register is also used to hold data temporarily. This register, as well as the ALU outputs (at the input to the RAM), may be shifted right or left.

Data enters the 2901A array from the D bus. The input data is applied to the ALU, where it (as one operand), and zeros or other data previously stored in the RAM or Q register (as the second operand), may be operated on as specified by the 2901A's function control lines. The ALU performs three arithmetic functions and five logic functions.

The D bus inputs are not always one of the operands applied to the ALU. The other ALU data sources, including the zero operand, may be selected in various combinations by the 2901A's source control inputs. For example, the contents of a RAM location read from the RAM's A port may be selected as one operand; and zeros, the Q register, or the contents of a RAM location read from the RAM's B port selected as the other operand. The RAM locations read from the A and B ports are determined by the 2901A's A and B address inputs.

The ability to select an operand of zeros, which is implemented by turning off the appropriate ALU input (R or S in Figure 6-5), allows a single operand to be processed separately without having to set zeros into another operand source. The zero operand may be selected in combination with any one of the other ALU inputs (D, Q, or the RAM location read from port A or B).

The ALU outputs (the F outputs in Figure 6-5) may be loaded into the RAM and/or the Q register. When the RAM is loaded, the ALU data may be shifted one bit left (equivalent to multiplying by two) or one bit right (equivalent to dividing by two). The Q register may also be shifted one bit left or right. In addition, shift control logic external to the 2901A array allows shift data to be rotated within the Q register and RAM, or shifted and rotated from one to the other (i.e., Q to RAM, or RAM to Q).

Besides applying the input to the RAM registers and Q register, the ALU outputs may be selected as the output from the 2901A array. The contents of a RAM location read from port A can also be selected as the output. The output selected, as well as the inputs to the RAM and Q registers, are controlled by the 2901A's destination control inputs.

The control logic associated with the 2901As includes PAL logic to generate the RAM's A and B port addresses, a ROM/PAL configuration used to generate the carry input, the function, source, and destination control bits, and the external carry look-ahead and shift control circuits.
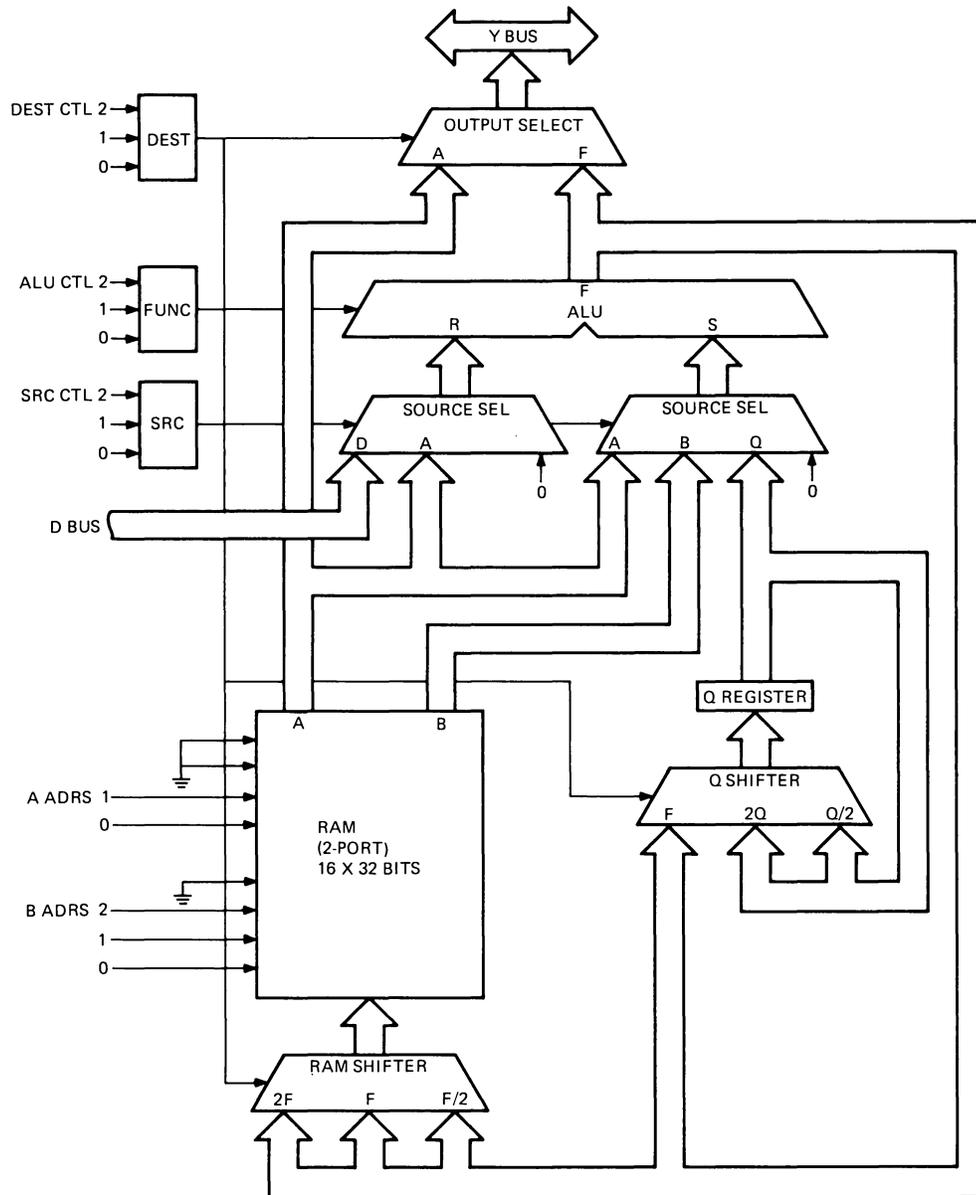
TK-5936

Figure 6-5   Data Processor (Eight 2901As) Simplified Block
Diagram

6.4.1   2901A RAM (Working Register) Addressing

The RAMs in the 2901A array are two-port devices in that two locations can be read simultaneously, one from the A port as specified by the A address, and the other from the B port as specified by the B address. If both A and B addresses are the same, the same location is read from both ports. At the end of a microcycle and when enabled by the 2901A destination control bits, new data is written into the RAM location defined by the B address.

Because only six of the 16 RAM locations (working registers) in the 2901As are used, address signals are generated for only three of the four B address inputs, and for only two of the four A address inputs. (The other 2901A address inputs are grounded.) PAL logic generates the address signals (B ADRS 2, 1, and 0; and A ADRS 1 and 0) from the control bits in the various microinstructions shown in Table 6-2.

6-9

Table 6-2   2901A RAM Addressing

Microinstruction	A ADRS 1	0	B ADRS 2	1	0	Remarks
DECODE	(Not used)		CSR<09>	0	0	B ADRS = 0/4
MEM REQ	(Not used)		1	0	1	B ADRS = 5
MOVE	CSR<08>	CSR<07>	MDP = 2	CSR<08>	CSR<07>	B ADRS = B field (B field + 4 if MDP = 2) A ADRS = B field
BASIC	CSR<08>	CSR<07>	0	CSR<08>	CSR<07>	B ADRS = A ADRS = B field
EXTENDED	CSR<10>	CSR<09>	0	CSR<08>	CSR<07>	B ADRS = B field A ADRS = A field
MISC/PORT	CSR<08>	CSR<07>	0	0	CSR<07>	B ADRS = A ADRS = 0/1 (CSR<08> must equal 0)

During the DECODE microinstruction, either working register (WR) 4 or 0 is addressed. WR 4 is used to store the PC during class decode operations (when CSR $\langle 09 \rangle$ = 1), in the event instruction execution must be aborted and then restarted due to errors or page failures when accessing memory. The PC can also be stored in location 0 (when CSR $\langle 09 \rangle$ = 0). This is done during specifier decode operations for branch type instructions.

In these cases, having the PC in a working register saves a microstate during the ensuing instruction execution.

During the MEM REQ microinstruction, WR 5 is addressed to store the memory address that is also transmitted on the MCT bus to the memory controller. Like the backup PC, the memory address may then be referenced following page failures, or during error recovery operations.

The MOVE and BASIC microinstructions have a B field (CSR $\langle 08{:}07 \rangle$) to address the working registers. The B field value is used to address both the A and B ports of the RAM. Ordinarily, only WR 0 through WR 3 can be addressed. (The B field is only two bits wide.) However, when a MOVE's MDP field is made equal to two, address line B ADR 2 is forced to a one allowing WR 4 and WR 5 to be addressed. This is the microinstruction used by memory management and error recovery microcode to access the backup-PC and the memory address stored in these two RAM locations.

The EXTENDED microinstruction has a 2-bit A field (CSR $\langle 10{:}09 \rangle$) in addition to a 2-bit B field (CSR $\langle 08{:}07 \rangle$). Thus, port A and B RAM addresses may have different values, allowing greater flexibility in the type of data processing done by this microinstruction. The EXTENDED is used to perform a variety of operations, some complex, and all occurring within the 2901A array itself.

One of the functions of the MISC/PORT microinstruction is to transfer data from a working register to the FPA or a port device. A single address bit (CSR $\langle 07 \rangle$) selects either WR 0 or WR 1. Two working registers are made available for FPA/port transfers to provide additional data buffering and facilitate the loading of data during diagnostic operations. Working register usage is summarized in Table 6-3.

Table 6-3   2901A Working Register Assignments

Working Register	Use
0	FPA/port data, general use
1	FPA/port data, general use
2	General use
3	Register backup mask, general use
4	Backup-PC
5	Memory address
6:F	Not used

6.4.2 2901A Control Bit Generation

The 2901A control ROM (a ROM/PAL combination) is used to generate the source, function, destination, and carry input control bits for the 2901As. Four of these control bits, the three ALU function bits and the low-order destination control bit, are read from the 512 × 4-bit ROM. The remaining six control bits are generated by the PAL. The ROM address, and all the PAL inputs (except for a control signal called MPLIER LSB), are control bits in the current microinstruction.

The three source control bits (SRC CTL 2, 1, and 0) generated by the 2901A control ROM specify the ALU source operands; that is, the data gated to the ALU's R and S inputs. Table 6-4 shows the eight combinations of inputs that can be selected. All 2901A register outputs [Q, working register read from port A, and working register read from a zero operand at the ALU inputs (SRC CTL codes 2, 3, 4, and 7)].

Table 6-4 2901A ALU Source Operand Control

SRC CTL 2	1	0	Code	Source Operands*	Remarks	
0	0	0	0	A.Q	R = WR(A)	S = Q
0	0	1	1	A.B	R = WR(A)	S = WR(B)
0	1	0	2	0.Q	R = 0	S = Q
0	1	1	3	0.B	R = 0	S = WR(B)
1	0	0	4	0.A	R = 0	S = WR(A)
1	0	1	5	D.A	R = D bus	S = WR(A)
1	1	0	6	D.Q	R = D bus	S = Q
1	1	1	7	D.0	R = D bus	S = 0

*Source operand notation corresponds to that in the *CPU Microcode Listing.*

Also, the working register output from port A may be processed with either the Q register or the working register outputs from port B (codes 0 and 1). Finally, the D bus inputs may be processed with the working register outputs from port A or the Q register (codes 5 and 6).

The three ALU function bits (ALU CTL 2, 1, and 0) specify the ALU operation to be performed on the operands selected by the source control bits. Three arithmetic and five logical operations may be selected as shown in Table 6-5. The arithmetic operations are add, S minus R, and R minus S (ALU CTL codes 0, 1, and 2). The logical operations are OR, AND, MASK, XOR, and XNOR (codes 3 through 7). Table 6-6 shows the ALU output (F) for the various values of the ALU function and source control bits. For the arithmetic operations, the ALU output also depends on the ALU carry input as indicated.

Table 6-5   2901A ALU Function Control

ALU CTL 2	1	0	Code	Function*	Symbol	Remarks
0	0	0	0	R plus S	R + S	Add R with S
0	0	1	1	S minus R	S – R	Subtract R from S
0	1	0	2	R minus S	R – S	Subtract S from R
0	1	1	3	R OR S	R V S	OR R with S
1	0	0	4	R AND S	R S	AND R with S
1	0	1	5	NOTR AND S	R S	Complement R, then AND with S (mask function)
1	1	0	6	R XOR S	R S	XOR R with S
1	1	1	7	R XNOR S	R S	XOR R with S, then complement result

*Function notation corresponds to that used in the *CPU Microcode Listing.*

Table 6-6   ALU Output (F) as a Function of ALU Source and Function Control

ALU Function	Carry In	0(A.Q)	1(A.B)	2(0.Q)	3(0.B)	ALU Source 4(0.A)	5(D.A)	6(D.Q)	7 (D.0)
0 (R plus S)	No	A+Q	A+B	Q	B	A	D+A	D+Q	D
	Yes	A+Q+1	A+B+1	Q+1	B+1	A+1	D+A+1	D+Q+1	D+1
1 (S minus R)	No	Q-A-1	B-A-1	Q-1	B-1	A-1	A-D-1	Q-D-1	-D-1
	Yes	Q-A	B-A	Q	B	A	A-D	Q-D	-D
2 (R minus S)	No	A-Q-1	A-B-1	-Q-1	-B-1	-A-1	D-A-1	D-Q-1	D-1
	Yes	A-Q	A-B	-Q	-B	-A	D-A	D-Q	D
3 (R OR S)	–	AVQ	AVB	Q	B	A	DVA	DVQ	D
4 (R AND S)	–	A Q	A B	0	0	0	D A	D Q	0
5 (NOTR AND S)	–	A Q	A B	Q	B	A	D A	D Q	0
6 (R XOR S)	–	A Q	A B	Q	B	A	D A	D Q	D
7 (R XNOR S)	–	A Q	A B	Q	B	A	D A	D Q	D

The three ALU destination control bits (DEST CTL 2, 1, and 0) generated by the 2901A control ROM determine where and how the ALU output is to be stored. As shown in Table 6-7, the ALU output may be stored unmodified in Q (code 0) or in the RAM (codes 2 and 3); or it may be shifted right or left and stored in the RAM without affecting Q (codes 5 and 7), or at the same time shifting Q right or left (codes 4 and 6). In addition, a NOP operation can be done (code 1) which does not load a 2901A register.

Table 6-7   2901A ALU Destination Control

DEST CTL 2	1	0	Code	Destination	Remarks
0	0	0	0	LOAD.Q	Load Q, output ALU (F)
0	0	1	1	NOP	Hold all registers
0	1	0	2	WRITE.B.A	Write WR (B), output WR (A)
0	1	1	3	WRITE.B.F	Write WR (B), output ALU (F)
1	0	0	4	RSHF.RAM.Q	Right shift RAM and Q, output ALU (F)
1	0	1	5	RSHF.RAM	Right shift RAM, output ALU (F)
1	1	0	6	LSHF.RAM.Q	Left shift RAM and Q, output ALU (F)
1	1	1	7	LSHF.RAM	Left shift RAM, output ALU (F)

*Destination notation corresponds to that in the *CPU Microcode Listing*.

In all cases except one, the ALU output is the output from the 2901As that is transmitted on the Y bus. The exception (code 2) writes the output of the ALU to a working register (as code 3 does), but passes the working register read from port A to the Y bus. This allows two 32-bit transfers in one microcycle when a local store write from the Y bus is done also.

The 2901A control ROM generates the control bits for the 2901A array by decoding the nine most significant bits in the current microinstruction (CSR ⟨22:14⟩). As shown in Table 6-8, these include the op code bits for all microinstructions, the data path control field for the ·BASIC, MOVE, and EX-TENDED, and the BPC (backup PC) bit for the DECODE. MPLIER LSB is also a control ROM input which can modify two operations by the EXTENDED, as explained in Paragraph 6.4.4.

Table 6-8   Decoding of Current Microinstruction by 2901A Control ROM

Microinstruction	22	21	20	CSR 19	18	17	16	15	14	2901A Operations	Specified By
BASIC	1	(6-BIT DP FIELD					)	--	–	64	DP field
MOVE	0	1	1	(MDP		)	–	--	–	8	MDP field
EXTENDED	0	1	0	(6-BIT XDP FIELD					)	64	XDP field
MEM REQ	0	0	1	1	--	--	–		–	1 (D to WR)	–
MISC/PORT	0	0	1	0	–	–	--	–	–	1 (WR to Y)	--
JUMP	0	0	0	1	–	–		–	·	1 (NOP)	
DECODE	0	0	0	0	BPC	–	--	–	–	2*	BPC

*D + 1 to Y, and D + 1 to WR if BPC = 1.

The 6-bit data path control fields of BASIC and EXTENDED allow up to 64 separate 2901A operations to be specified by each microinstruction; that is, one set of 2901A control bits are generated for each data path control field value. Similarly, the MOVE's 3-bit data path control field allows eight 2901A operations to be specified.

Except for the DECODE, only a single 2901A operation is done by the other microinstructions. For example, during a MEM REQ, the 2901A control ROM generates the control bits necessary to transfer the D bus inputs (memory address data) through the ALU and store them in WR 5.

During the DECODE, one of two 2901A operations can occur, depending on the microinstruction's BPC control bit (CSR⟨18⟩). If BPC = 0, the D bus inputs (the PC) are incremented by the ALU and asserted on the Y bus. This also occurs when BPC = 1, but the control bits generated by the 2901A control ROM specify that the ALU output (the incremented PC) must also be stored in a working register.

The 2901A control ROM outputs are given in the *CPU Microcode Listing*. The control ROM partition for the various microinstructions are shown in Figure 6-6. Due to the addressing scheme (Table 6-6), blocks of ROM locations having identical contents are required for all microinstructions but the EXTENDED. For example, the MOVE requires 64 ROM addresses (0C0 to 0FF), although only eight 2901A operations are specified by its data path control field. As a result, eight blocks of eight identical ROM entries are required to generate the 2901A control bits for this microinstruction.
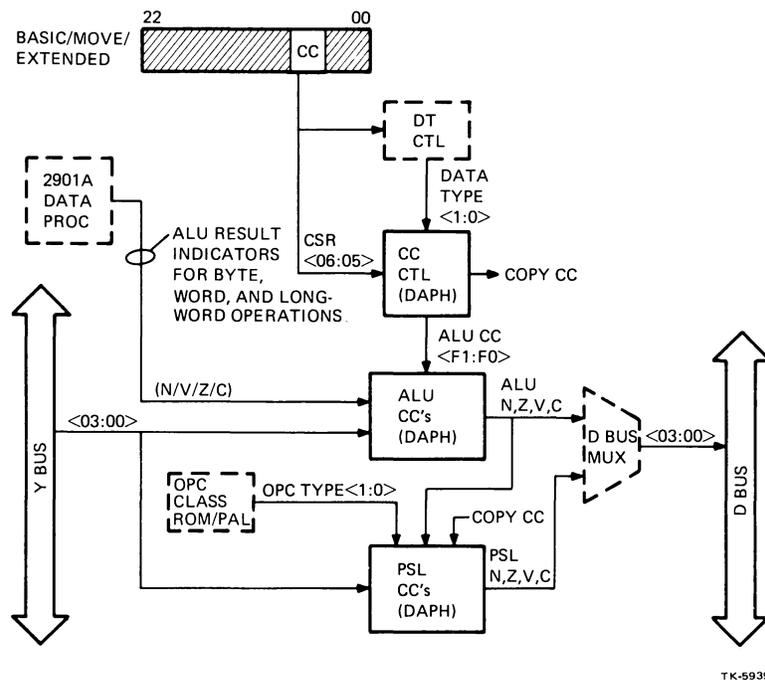
Figure 6-6 2901A Control ROM

An example of an entry in the microcode listing that shows the 2901A control ROM outputs is given below.

SRC/D.0, ALU/R.PLUS.S, DST/WRITE.B.F, CIN/CIN

The notation used to show the control bits generated is the same as in Tables 6-4, 6-5, and 6-7. The example chosen is for the DECODE, which causes the PC value at the D bus inputs of the 2901A to be incremented, asserted on the Y bus, and stored in a working register (BPC = 1).

Using Table 6-6, note that inputs of the 2901A are a source of "D.0" (code 7), an ALU function of "R.PLUS.S" (code 0), and a "CIN" (carry input) which causes the ALU to increment the D bus inputs by +1 as required. Using Table 6-7, note also that a destination of "WRITE.B.F" (code 3) causes the ALU output to be asserted at the 2901A outputs (on the Y bus), and stored in a working register to complete the operation.

6.4.3 Carry Logic

Carry look-ahead (skipper) circuits are used in the 2901A data processor to speed ALU operation. Connection to the 4-bit 2901A microprocessor slices is shown in Figure 6-7.

Figure 6-7   2901A Carry Logic

6-18

Each 2901A has a carry propagate output (P) and a carry generate output (G) for use by external carry logic. These are sampled by the carry skipper circuits to generate the carry inputs to all the 2901As except the low-order one. (The low-order 2901A is for bits $\langle 3:0 \rangle$ in the data path.)

The carry input to the low-order 2901A (ALU CARRY-IN), which is the carry input to the entire array, is generated by the 2901A control ROM. The carry input may be asserted or not, depending on the 2901A operation specified by the current microinstruction.

In lieu of carry look-ahead circuitry, a slower ripple-carry scheme would have to be used whereby the carry output from one 2901A would connect to the carry input of the next.

Carry outputs from the 2901As connect to the ALU condition code logic. Depending on the size of the data being processed in the ALU (longword, word, or byte), the state of either CARRY 32, CARRY 16, or CARRY 8 is set into carry status flip-flop ALU C at the end of the microcycle. ALU C is one of the ALU condition codes, and is a microsequencer jump or skip condition which may be tested by the microcode.

An additional carry status flip-flop, HALF CARRY, stores the carry output from the low-order 2901A (CARRY 4). It is used during instructions performing packed decimal operations (ADDP, SUBP, etc.); that is, together with ALU C, it is used to generate the appropriate decimal constant to convert 2901A ALU results (which are in binary) to the packed decimal format. The packed decimal format has two 4-bit decimal digits per byte. As described in Paragraph 6.9, the constant (00, 06, 60, or 66) is asserted on the D bus when a BASIC or MOVE microinstruction's D address field is equal to 7D.

6.4.4 Shift Control

The 2901A microprocessor slice has four shift data input/output pins. Two are for the Q register, one for shifting data in or out of the least significant bit (LSB), and the other for shifting data in or out of the most significant bit (MSB). The other two shift data input/output pins are for shifting the ALU output data written into a RAM location (working register). As for the Q register, one input/output pin is for the LSB (ALU 0), and one is for the MSB (ALU 31).

In the 2901A array, the most significant Q or RAM shift data input/output pin in one 2901A connects to the least significant input/output pin of the next. This allows 32-bit right or left shifts of the Q register, and a working register across the array.

Also, the least significant shift data input/output pins at the low-order 2901A, and the most significant input/output pins at the high-order 2901A, connect to an external shift control. By gating data shifted out of one end of the array into the other end, the external shift control allows both 32-bit and 64-bit shift and rotate operations to take place. Also, trailing sign bits, zeros, or other data may be inserted in the shift data stream.

Shift and rotate operations are controlled by the EXTENDED microinstruction. Control is by the microinstruction's data path control (XDP) field, working in conjunction with the shift input (SI) field. Only certain values of the XDP field initiate a shift operation.

The XDP field (CSR $\langle 19:14 \rangle$), by addressing the 2901A control ROM (as discussed in Paragraph 6.4.2), generates the control bits that specify the 2901A operation.

For most of the shift operations initiated by the XDP field, the ALU output is specified as the unmodified contents of the working register read from the B port of the RAM. The ALU destination specified (code 4, 5, 6, or 7) loads the working register contents back into the same RAM location, shifting it left or right. At the same time, the Q register may also be shifted left or right.

The 3-bit SI field (CSR $\langle 13:11 \rangle$), together with the 3-bit destination code generated by the XDP field, determine the operation of the shift control logic external to the 2901As. (A simplified diagram of the shift control is shown in Figure 6-8.) The SI field selects the shift data inputs to the high-order and low-order 2901As. That is, shift data may be gated into any one of the 2901A shift data input/output pins (Q SHF MSB and RAM SHF MSB at the high-order 2901A, and Q SHF LSB and RAM SHF LSB at the low-order 2901A).

The input data is the output from one of the shift data input/output pins at the other end of the array. The destination code is used to enable the gates driving the shift data input/output pins. Two of the four gates are enabled at a time. The gates that are enabled determine the direction of the shift.
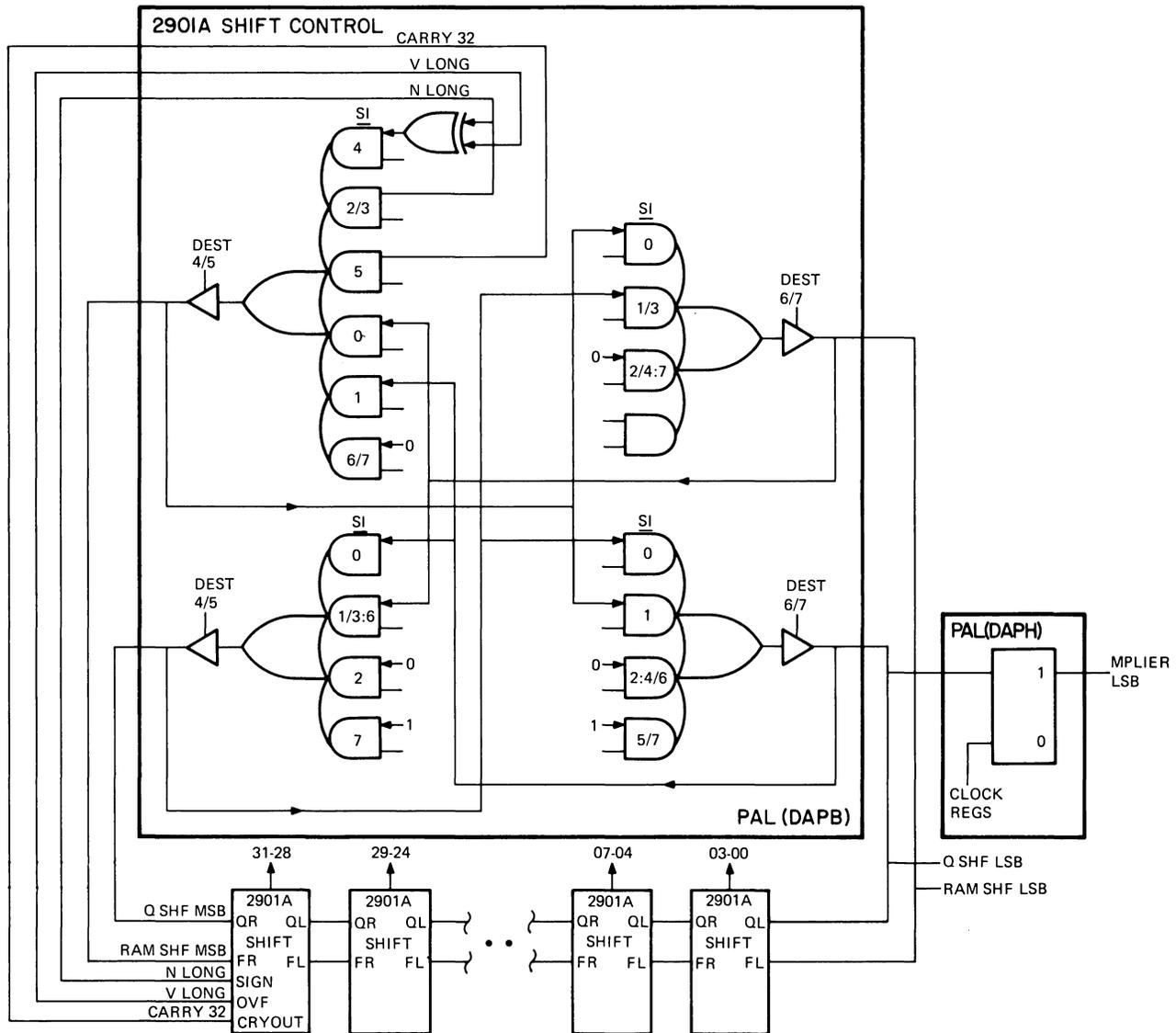
Figure 6-8   2901A Shift Control

6-20

Shift data input selection by the SI field is shown in Table 6-9. For example, for SI = 0, the input gated to RAM SHF MSB (ALU 31) is RAM SHF LSB (ALU 0), and the input gated to RAM SHF LSB (ALU 0) is RAM SHF MSB (ALU 31). Selection for the Q register's shift inputs is similar, with conditions set up for gating data shifted out of one end of the 2901A array through the shift control, and back into the other end of the array. Thus, working register and Q register data can be rotated either right or left, depending on the value of the destination code.

Table 6-9 Shift Data Inputs

SI	Input to SHF MSB(ALU 31)	Input to RAM SHF LSB (ALU 0)	Input to Q SHF MSB (Q31)	Input to Q SHF LSB (Q4)
0	RAM SHF LSB (ALU 0)	RAM SHF MSB (ALU 31)	Q SHF LSB (Q 0)	Q SHF MSB (Q 31)
1	Q SHF LSB (Q 0)	Q SHF MSB (Q 31)	RAM SHF LSB (ALU 0)	RAM SHF MSB (ALU 31)
2	N LONG (SIGN)	0	0	0
3	N LONG (SIGN)	Q SHF MSB (Q 31)	RAM SHF LSB (ALU 0)	0
4	N LONG XOR V LONG (SIGN XOR OVF)	0	RAM SHF LSB (ALU 0)	0
5	CARRY 32 (CRY OUT)	0	RAM SHF LSB (ALU 0)	1
6	0	0	RAM SHF LSB (ALU 0)	0
7	0	0	1	1

The meaningful combinations of SI field and destination code values, and the right or left shift or rotate operations they produce, are given in Figure 6-9. Other combinations give unspecified results.

For the preceding example where SI = 0, it can be seen that a destination code of 6, which left-shifts both RAM and Q data in the 2901As, causes data to be rotated within both a working register and Q register. Correspondingly, a destination code of 7, which left-shifts only RAM data in the 2901As, causes only working register data to be rotated.

Most of the shift operations initiated by the EXTENDED's XDP field pass the contents of a single RAM location through the ALU unmodified. This results in simple 1-bit shifts, left or right, of working register data. (Q register data is always shifted one bit left or right.) For example, with the usual ALU output, an SI field of two and a destination of seven results in a working register being shifted one bit left, with a trailing zero inserted by the shift control during the operation. (Refer again to Table 6-9 and to Figure 6-9.)
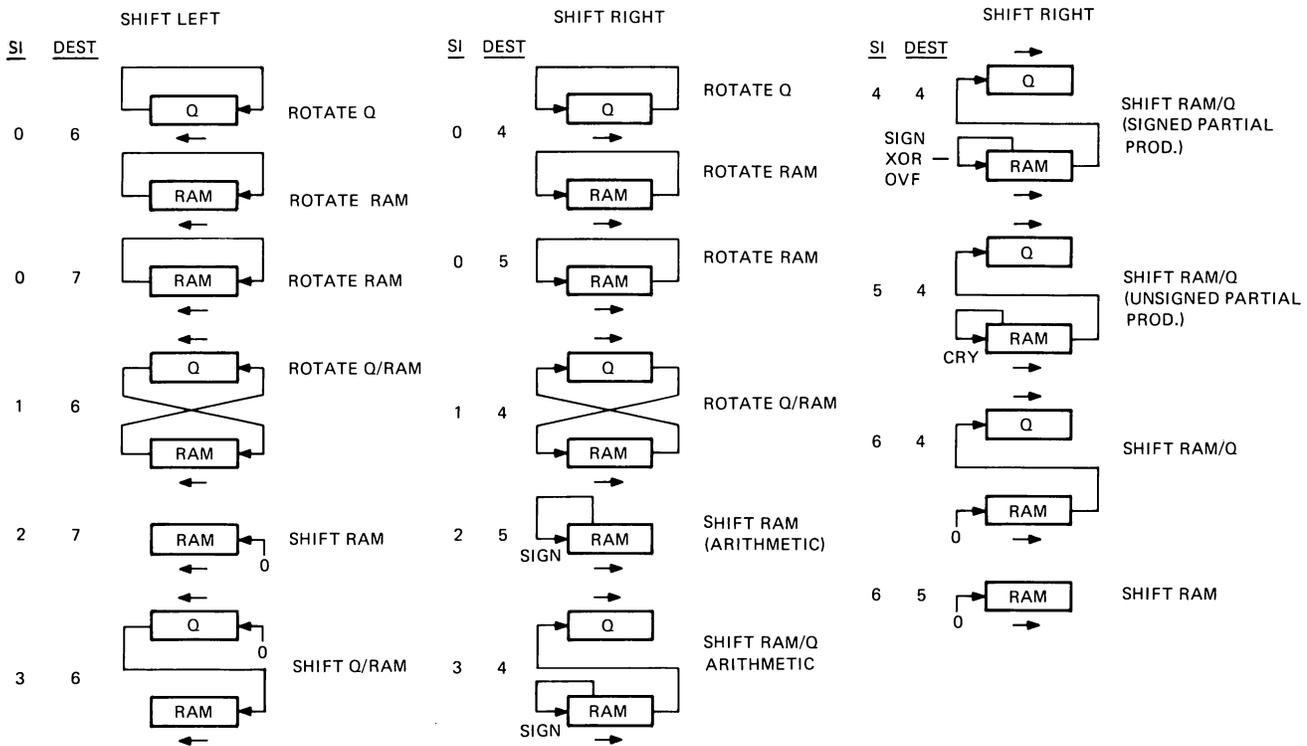
Figure 6-9   Shift Configurations

However, one XDP field value (8B) generates 2901A control bits that add a working register to themselves in the ALU, the destination code still being equal to 7. With the same SI field value of two, as in the preceding example, the result is not a 1-bit shift, but a 2-bit shift due to the "add before shift" operation specified by XDP.

Two other operations are not just simple shifts of unmodified working register data. One XDP field value (88) adds two different working registers together. Another (89) subtracts two working registers. Both generate a destination code of 4. In conjunction with SI field values of 4 and 5, these shift operations cause one cycle of a hardware multiply to execute.

An SI field value of 4 is used for signed partial products (2s complement); and an SI field value of 5 is used for unsigned partial products (floating point). In one case (SI = 4), the trailing bit inserted by the shift control is the sign bit out of the 2901A array (N LONG) XOR'd with the overflow bit out of the array (V LONG). In the other case (SI = 5), the trailing bit is the carry output from the high-order 2901A (CARRY 32).

Both the "add before shift" and "subtract before shift" operations used for the hardware multiply cycle are conditional. The add or subtract does not take place unless the LSB of the multiplier is set. The multiplier, which is held in the Q register during multiply operations, is right shifted by the microcode during the microcycle preceding each multiply cycle. (The multiplicand is stored in the working register read from A; the partial product is stored in the working register read from B.)

6-22

The bit shifted out of Q, the LSB, is stored in control flip-flop MPLIER LSB (Figure 6-9). MPLIER LSB is an input to the 2901A control ROM and controls the ALU source code during the upcoming hardware multiply cycle. IF MPLIER LSB is set, the source code will be 1 or 2 (depending on XDP), and the ALU selects the working registers read from A and B as the operands. This adds or subtracts the multiplicand and partial product. If MPLIER LSB is not set, the ALU source code will equal 3 and the working register read from B (the partial product) will be passed through the ALU unchanged. There is no hardware support for a divide operation.

6.5 LOCAL STORE (LS)

The local store is a 256 location RAM that contains the 16 GPRs, 16 backup GPRs used for restart purposes, several of the privileged processor registers, a number of masks and constants generated by the microcode during system initialization, and several locations used for temporary data storage. It consists of eight parallel-connected 256 × 4-bit RAM chips, four 8-bit latch circuits, a set of RAM address gates, and read/write control logic. The resulting 256 × 32-bit configuration is shown in Figure 6-10.
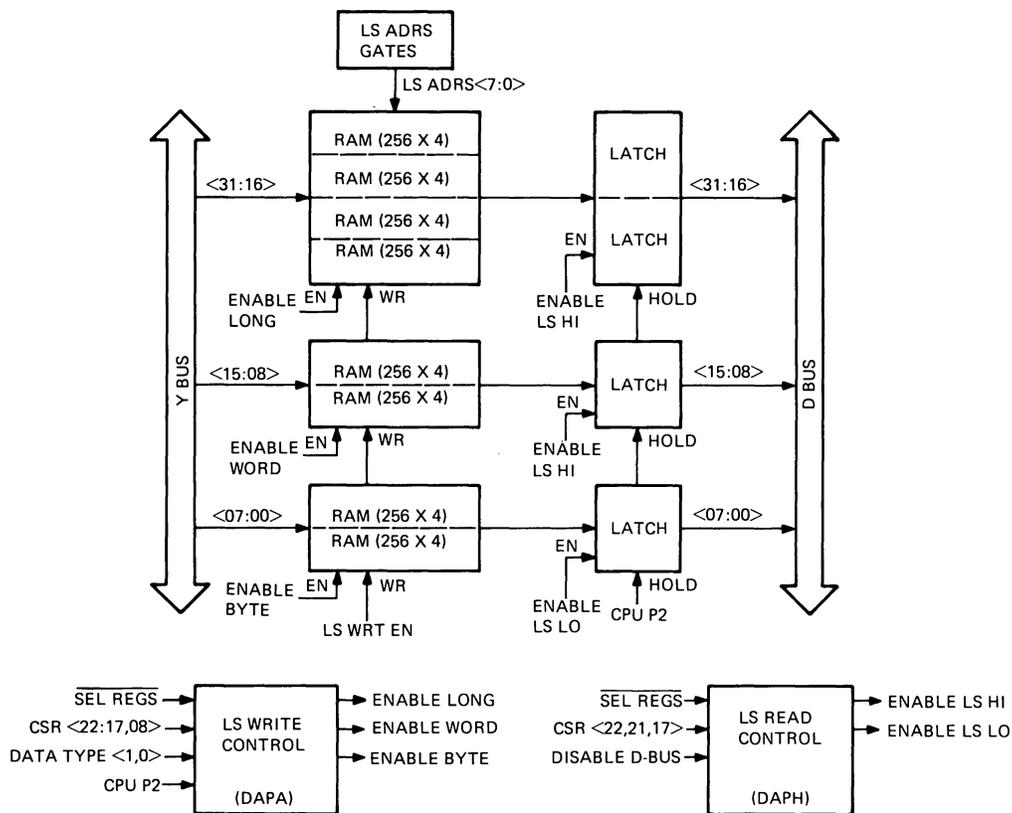
TK-5950

Figure 6-10   Local Store Configuration

6-23

As discussed in Paragraph 6.3, the local store is written from the Y bus with a pulse (LS WRT EN) asserted at the end of the microcycle (T225 to T270). The number of bits written is data-type dependent, and controlled by three signals connecting to the RAM's write enable inputs. These write enable signals are ENABLE BYTE, ENABLE WORD, and ENABLE LONG. Only bits 7 to 0 are written during byte operations (ENABLE BYTE = 1). A word operation writes bits 15 to 0 (ENABLE WORD = 1 and ENABLE BYTE = 1), and a longword operation writes bits 31 to 0 (ENABLE LONG = 1, ENABLE WORD = 1, and ENABLE BYTE = 1).

The local store outputs are enabled onto the D bus via the latch circuits. The latches have tri-state outputs that are turned on by two output enable signals, ENABLE LS HI and ENABLE LS LO. When local store is to be read, both of these enable signals are asserted (from T45 to T270), causing all 32 local store bits to be transmitted on the D bus. The latch circuits are transparent until clock phase 2 (CPU P2), at which time the outputs latch up for the remainder of the microcycle.

The local store is addressed by the 7-bit D ADRS (data address) fields of the MEM REQ and BASIC microinstructions CSR $\langle 15:09 \rangle$), and the 8-bit XD ADRS (extended data address) field of the MOVE microinstruction (CSR $\langle 16:09 \rangle$). The local store locations accessed by the various values of D ADRS and XD ADRS are given in Tables 6-10 and 6-11. Local store address assignments are shown in Figure 6-11.

Table 6-10   Local Store Addressing for MEM REQ/BASIC Microinstructions

D Field	LS ADR								Local Store Address	Remarks
	7	6	5	4	3	2	1	0		
00:77	0	(		D field				)	00:77	D address equals LS address
78	0	1	0	0	(OS$\langle 3:0 \rangle$			)	40:4F	Base address 40 indexed by OS$\langle 3 \rangle$ = 0
79	0	0	1	0	(OS$\langle 3:0 \rangle$			)	20:2F	Base address 20 indexed by OS$\langle 3:0 \rangle$
7A		–							–	Not used
7B	0	0	1	(	OS$\langle 4:0 \rangle$			)	20:3F	Base address 20 indexed by OS$\langle 4:0 \rangle$
7C:7F		–							–	Address assigned to discrete registers

Table 6-11 Local Store Addressing for MOVE Microinstruction

XD Field	LS ADR 7	6	5	4	3	2	1	0	Local Store Address	Remarks
00:77	(		XD Field					)	00:77	XD address equals LS address
78	0	1	0	0	(		OS⟨3:0⟩)		40:4F	Base address 40 indexed by OS⟨3:0⟩
79	0	0	1	0	(		OS⟨3:0⟩)		20:2F	Base address 20 indexed by OS⟨3:0⟩
7A				–					–	Not used
7B	0	0	1	(	OS⟨4:0⟩		)		20:3F	Base address 20 indexed by OS⟨4:0⟩
7C:7F				–					–	XD addresses assigned to discrete registers
80:F7	(		XD Field					)	80:F7	Address equals LS address
F8	1	1	0	0	(		OS⟨3:0⟩)		C0:CF	Base address C0 indexed by OS⟨3:0⟩
F9	1	0	1	0	(		OS⟨3:0⟩)		A0:AF	Base address A0 indexed by OS⟨3:0⟩
FA				–					–	Not used
FB	1	0	1	(	⟨OS4:0⟩		)		A0:BF	Base address A0 indexed by OS⟨4:0⟩
FC:FF				–					–	Addresses assigned to discrete registers

NOTE: DISCRETE REGISTER ADDRESS.

TK-5949
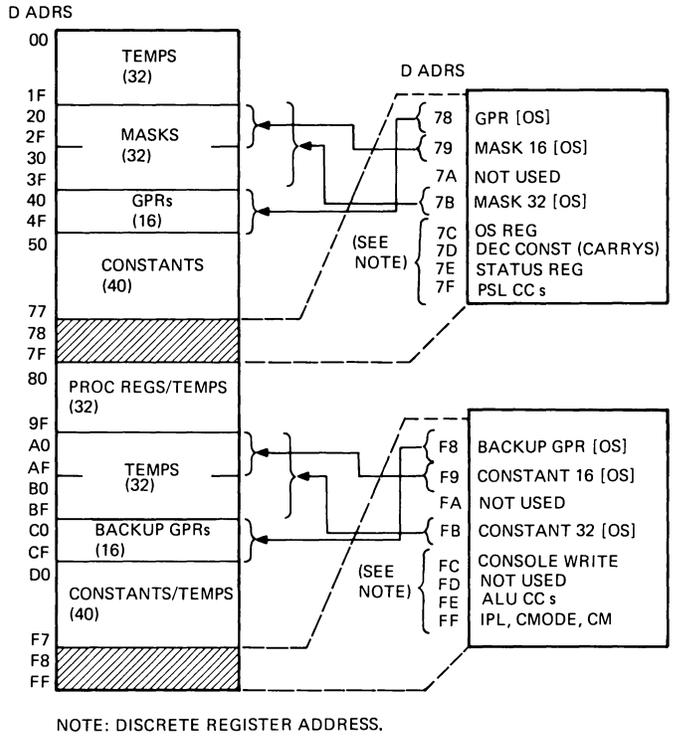
Figure 6-11   Local Store Address Assignments

Not all the RAM locations in local store are used to store data. D address values of 00 to 77, and 80 to F7, allow direct access to the corresponding addresses in local store; but addresses in the ranges 78 to 7F, and F8 to FF, do not. In this second range of addresses, half (78:7B and F8:FB) are pseudo addresses. The other half (7C:7F and FC:FF) are assigned to discrete registers such as the OS register or the condition codes. Discrete register addressing is discussed in Paragraph 6.9.

The pseudo addresses form the mechanism that allows the OS register to access local store. For example, a D address of 78 causes the local store address logic to generate a base address of 40 (LS ADRS 7:4 = 0100) and to gate the four low-order OS bits as the four low-order local store address bits (LS ADRS 3:0 = OS 3:0). The resulting local store address, 40 indexed by OS, accesses one of 16 locations in the range 40 to 4F, which is one of the GPRs. The local store address logic is shown in Figure 6-12.

Other pseudo addresses allow the OS register to access the MASKS, backup GPRs, and one of the temporary data storage areas in local store (see Figure 6-11). In some cases, the five low-order bits of OS are used to generate the local store address. This allows indexing into 32-location blocks of mask and temporary storage data.

The local store is written by the BASIC, MOVE, and DECODE microinstructions. The three write enable signals (ENABLE BYTE, ENABLE WORD, and ENABLE LONG) are generated as shown in Table 6-12.

Only certain values of the data path control fields in a BASIC or MOVE instruction cause the local store to be written. For the BASIC, the DP field values are 20 to 3F. For the MOVE, XDP equals 0 and 4 to 7. For both microinstructions, the local store write enables are inhibited by SEL REGS = 1. The SEL REGS signal is asserted by the local store address logic (Figure 6-12) when the D address is for a discrete register.

6-26

Table 6-12 Local Store Write Control

Microinstructions	DATA TYPE		ENABLE			Write LS
	1	0	LONG	WORD	BYTE	
BASIC						
DP = 20:3F/D ADRS = 00:7B	0	0	0	0	1	<7:0>
	0	1	0	1	1	<15:0>
	1	1	1	1	1	<31:0>
MOVE						
MDP = 0,4:7/XD ADRS = 00:7B, 80:FB	0	0	0	0	1	<7:0>
	0	1	0	1	1	<15:0>
	1	1	1	1	1	<31:0>
DECODE (IB REQ = 1)	–	–	1	1	1	<31:0>
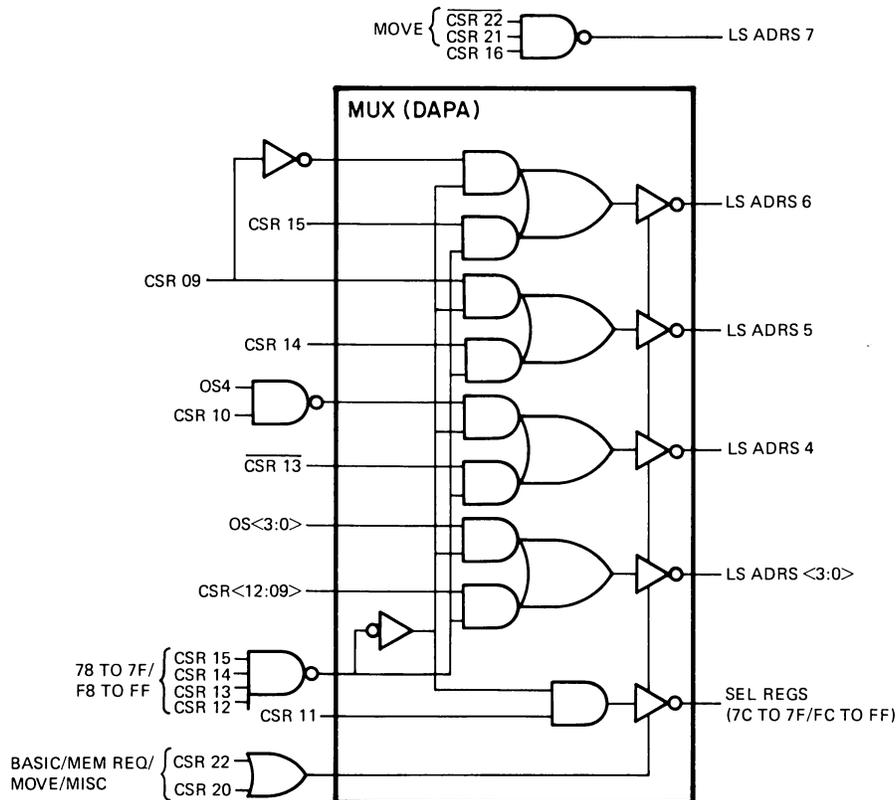
TK-5930

Figure 6-12   Local Store Address Logic

6-27

The number of bits written into local store during the BASIC and MOVE is determined by control signals DATA TYPE 1 and 0 from the data type control logic. For byte operations, DATA TYPE 1 and 0 equal 00 to assert ENABLE BYTE and write the low-order eight bits of local store. For word operations, DATA TYPE 1 and 0 equal 01, which asserts ENABLE WORD (in addition to ENABLE BYTE) to write the low-order 16 bits of local store. Finally, during longword operations, DATA TYPE 1 and 0 are equal to 11, which asserts ENABLE LONG (in addition to the other two write enables), and causes all 32 bits of local store to be written.

The type of write operation specified by DATA TYPE 1 and 0 is determined by the CC field of the BASIC and MOVE, the size register, and the compatibility mode control bit. The CC field, which controls both data type and the condition codes, may specify longword operations while holding or loading the ALU CCs (CC field = 00 and 01), or it may specify the data type previously loaded in the size register while loading or copying the ALU CCs (CC field = 10 and 11). When longword operation is specified by the CC field, either directly or indirectly by means of the size register, word operations will take place if the machine is in compatibility mode.

The DECODE microinstruction writes local store only when the incremented native mode PC is being restored following an unload of the PFR (i.e., IB REQ = 1). To address the PC, the DECODE simply disables the local store address gates generating an address of 10. The address is 10 instead of 00 because LS ADRS 4, unlike the other address gates, has inputs which are low when true. Thus, disabling the address gates is the equivalent to a D address of 10, which is the address used by the microcode when accessing the native mode PC by means of the BASIC and MOVE microinstructions. When the PC is written by the DECODE, all three write enables are asserted to write all 32 local store bits.

NOTE
Because the native mode PC is stored in local store location 10, location 4F in the GPR block is used for other purposes (e.g., temporary data storage).

Local store read control is shown in Table 6-13. The read enables, ENABLE LS HI and ENABLE LS LO, are asserted during all microinstructions except when a discrete register is being addressed during the BASIC and MOVE, and except for even values of the MOVE's data path control fields (MDP = 0, 2, 4, and 6). Even values of MDP prevent a local store read so that memory read data may be gated onto the D bus when MDP = 0 or 4.

Table 6-13   Local Store Read Control

Microinstruction	D Address	ENABLE LS HI	LS LLO	Read LS
BASIC	00:7B	1	1	<31:00>
MOVE(MDP = 1,3,5,7)/EXTENDED	00:7B, 80:FB	1	1	<31:00>
MEM REQ/MISC/JUMP/DECODE		1	1	<31:00>

6.6 OPERAND SPECIFIER (OS) REGISTER

The 8-bit OS register (OS$\langle 7:0 \rangle$), which can access local store, is loaded from both the Y bus and the IB bus. The register can be read onto the D bus when its contents are to be processed by the 2901A data processor.

The load from the IB is controlled by the DECODE microinstruction. Control is by the register's input select levels (OS CTL 1 and 0), as described in Paragraphs 5.7.1 and 5.7.2, and as summarized in Table 5-3.

The load from the Y bus, and the read onto the D bus, occur during data path transfers by the BASIC or MOVE microinstructions. LOAD Y TO OS, asserted by the CPU's register read/write control, loads an arbitrary value into OS from Y bus data lines 7 to 0 for certain values of the microinstruction's data path control fields, and when the OS register is addressed by the D address field. (The discrete register address for OS is 7C.)

The same value of the data path control field causes the OS register to be gated onto D bus data lines 7 to 0 via the D bus multiplexer. Again, control is by the register read/write control when the OS register is addressed by the microinstruction's D address field. Whenever the OS is read onto the D bus, its contents are sign extended. That is, D bus bits 31 to 8 are made equal to the state of OS$\langle 7 \rangle$ by the CPU's sign extension control.

Finally, during microdiagnostic operations, the four low-order bits of OS (OS$\langle 3:0 \rangle$) may be transmitted on the UNIBUS BR lines. This is done by the 8085A console processor during microdiagnostic operations. The OS register is first loaded and then a MISC microinstruction with a function 2 field equal to 4 is executed. The ability to assert the BR lines allows the microdiagnostics to test the CPU's interrupt logic.

6.7 DATA TYPE CONTROL

The data type control performs the following data path functions:

1. Determines whether a byte, word, or longword of data is written into local store.

2. Determines whether a byte, word, or longword of read data from the memory controller is gated from the MC bus onto the D bus.

3. Controls the appending of sign bits on D bus data by the sign extension logic.

4. Determines the set of 2901A ALU result indicators (i.e.; N, Z, V, and C for either a byte, word, or longword operation) that are loaded into the ALU CCs.

The main logic element in the data type control is the size register. The data type control also includes the memory data type (MDT) register plus other control logic, as shown in Figure 6-13.

Figure 6-13   Data Type Control

The 2-bit size register, SIZE REG 1 and 0, is detailed in Figure 6-14. It is loaded by the DECODE microinstruction during class decode operations (IRD STATE = 1) with a code generated by the OPC ROM/PAL (DT CLASS 1 and 0) that specifies the data type of the instruction's first operand. The code, once in the size register, specifies the data type as follows.

Size Register	Data Type
00	Byte
01	Word
10	(Not used)
11	Longword

If the data type in the size register must be changed after the start of an instruction's execution, the microcode may reload the size register from Y bus data lines 1 and 0. The load from the Y bus is made by a MOVE or BASIC microinstruction using a discrete register address of 7D or 7E (LOAD STAT = 1). If the discrete register address used is 7E, the same MOVE or BASIC that loads the size register reads the current value onto the D bus data lines 1 and 0. The size register bits are asserted on D bus data lines 1 and 0 (by EN STAT REG = 1).

How and if the size register controls data path operations is determined by the current micro-instruction. For example, during a BASIC, MOVE, or EXTENDED microinstruction, the CC field (CSR ⟨06:05⟩) may select the size register to generate control signals DATA TYPE 1 and 0 (Figure 6-13). As discussed in Paragraph 6.5, DATA TYPE 1 and 0 determine the size of any data written into local store. These signals also control the ALU CCs if they are being loaded. Finally, DATA TYPE 1 and 0 are asserted on the MC bus. When the current microinstruction is a MEM REQ, they indicate to the memory controller the size of the requested data transfer.

6-30

Figure 6-14   Size and MDT Registers

The other data type-dependent functions in the data path are controlled by the 2-bit MDT register (MDT 1 and 0). (The MDT register is shown with the size register in Figure 6-14.) The MDT register is loaded by the MEM REQ microinstruction with the data type code asserted by DATA TYPE 1 and 0 on the MC bus. The data type is specified by the MEM REQ's DT field which (like the CC field) is CSR⟨06:05⟩ in the microinstruction.

If the MEM REQ initiates a read operation in the memory controller, the MDT register outputs are used to control both the sign extension logic and the size of the data gated onto the D bus during the next MOVE, when the requested read data from the memory controller is transferred within the data path. (While the previously loaded MDT register controls the transfer of MC bus data during the MOVE, the MOVE's CC field generates DATA TYPE 1 and 0 to control any local store writes.) The MDT register cannot be loaded from the Y bus. However, it is read onto D bus ⟨7:6⟩ by the same MOVE or BASIC that reads the size register (discrete register address = 7E).

The generation of DATA TYPE 1 and 0 by the various microinstructions is shown in Table 6-14. DATA TYPE 1 and 0, when equal to 00, specifies byte operation; 01 specifies word operation; and 11 specifies longword operation. A value of 10 is not used.

6-31

Table 6-14   Generation of DATA TYPE Control Signals

Microinstruction		Data Size 1	0	Type 1	0	Data Type
BASIC/MOVE/EXTENDED						
CC(CSR<06:05>)	= 00 (Use longword context, hold CCs)	–	–	1	1	Longword*
	= 01 (Use longword context, set CCs)	–	–	1	1	Longword*
	= 10 (Use size, set CCs)	0	0	0	0	Byte
		0	1	0	1	Word
		1	1	1	1	Longword*
	= 11 (Copy CCs)	0	0	0	0	Byte
		0	1	0	1	Word
		1	1	1	1	Longword*
MEM REQ						
DT(CSR<06:05>)	= 00 (Byte)	–	–	0	0	Byte
	= 01 (Word)	–	–	0	1	Word
	= 10 (Use size)	0	0	0	0	Byte
		0	1	0	1	Word
		1	1	1	1	Longword*
	= 11 (Longword)	–	–	1	1	Longword*
DECODE/JUMP/MISC/PORT		–	–	–	–	N/A

*DATA TYPE<1:0> = 01 (Word) if COMPAT MODE = 1

Also shown in Table 6-14 is the dual function of the BASIC, MOVE, and EXTENDED micro-instructions' CC field, which controls both data type and the condition codes. (The condition codes are discussed in Paragraph 6.8.) CC field values of 00 and 01 assert DATA TYPE 1 and 0 to cause long-word operations (while holding or setting the CCs), provided the machine is not in compatibility mode. If the machine is in compatibility mode, these same CC field values generate DATA TYPE 1 and 0 to cause word operations. Only byte and word operations are allowed in compatibility mode.

CC field values 10 and 11 specify a data type equal to the size register (while setting or copying the CCs). Again, this is conditional on the machine mode. If the size register contains a longword code in compatibility mode, word operations result.

The MEM REQ microinstruction's DT field specifies only data type and has no other function. Conditional on machine mode, DT field values of 00, 01, and 11 generate corresponding values for DATA TYPE 1 and 0, allowing byte, word, and longword operations to be specified. Also, when DT is equal to 10, the data type is specified by the size register.

Data type 1 and 0 load the MDT register during a MEM REQ by generating control signals MDT CTL 1 and 0.

Operation is as follows.

DATA TYPE 1	0	MDT CTL 1	0	MDT 1	0	Data Type
0	0	0	0	0	0	Byte
0	1	0	1	0	1	Word
1	1	1	0	1	1	Longword

During all microinstructions other than a MEM REQ, MDT CTL 1 and 0 are equal to 11, which causes the MDT register to hold its current value.

6.8 CONDITION CODE (CC) LOGIC

The condition code (CC) logic consists of the ALU CCs, the PSL CCs, and the control logic to load them. The ALU CCs are loaded from the 2901A ALU result indicators, to indicate to the microprogram the result of a microinstruction's execution. The PSL CCs are loaded from the ALU CCs to indicate to the CPU program the result of a native or compatibility mode instruction's execution. A block diagram of the condition code logic is shown in Figure 6-15.
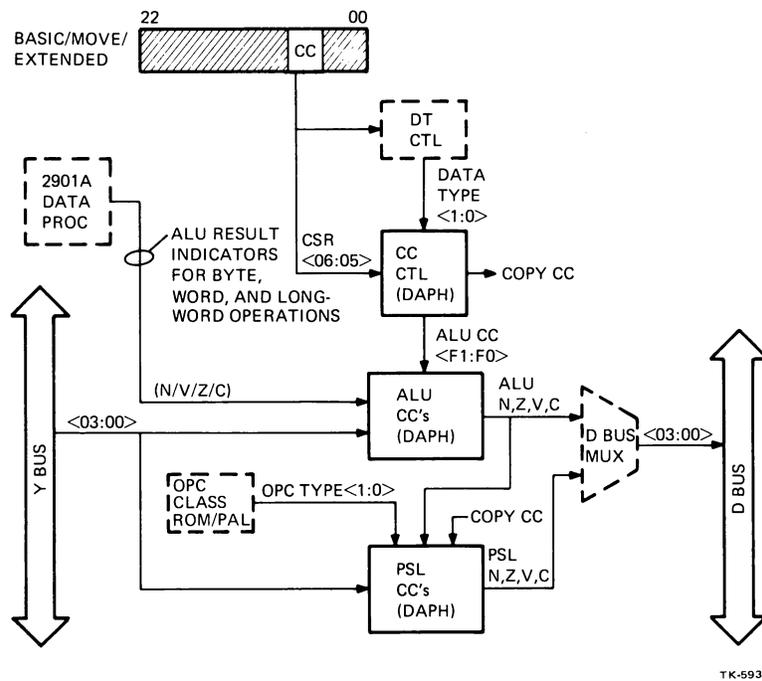
Figure 6-15   Condition Code Logic

6-33

The ALU CCs consist of four flip-flops (ALU N, Z, V, and C), as shown in Figure 6-16. They may be loaded by a BASIC, MOVE, or EXTENDED at the end of the microcycle. The ALU result indicators that are specified to be loaded depend on the size of the data being processed in the 2901As, as shown in Table 6-15.
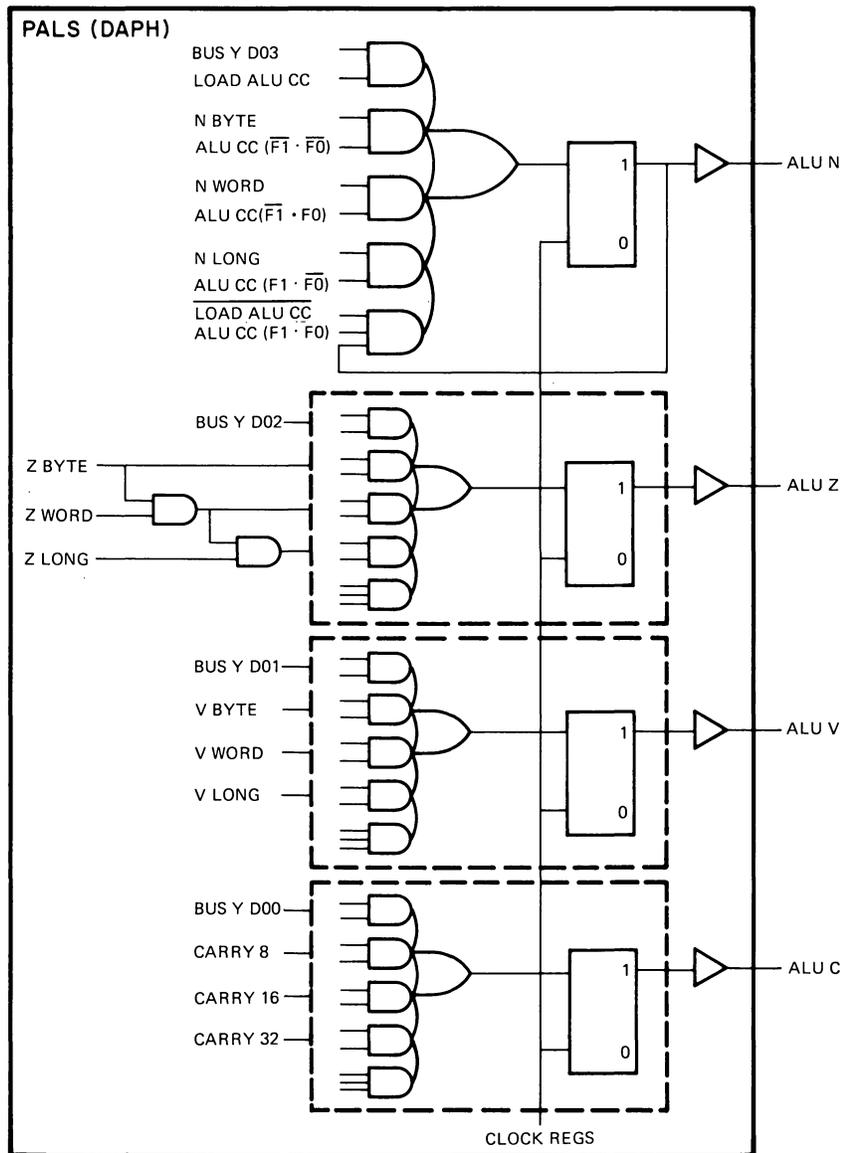
Figure 6-16   ALU Condition Codes

Table 6-15   ALU Indicator to ALU CC Transfer

ALU Operation	ALU Indicators Loaded into ALU CCs
Byte	N BYTE (ALU$\langle 7 \rangle$) to ALU N Z BYTE (ALU$\langle 7{:}0 \rangle$ = 0) to ALU Z V BYTE (OVF$\langle 7 \rangle$) to ALU V CARRY 8 to ALU C
Word	N WORD (ALU$\langle 15 \rangle$) to ALU N Z WORD AND Z BYTE (ALU$\langle 15{:}00 \rangle$ = 0) to ALU Z V WORD (OVF$\langle 15 \rangle$) to ALU V CARRY 16 to ALU C
Longword	N LONG (ALU$\langle 31 \rangle$) to ALU N Z LONG AND Z WORD AND Z BYTE (ALU$\langle 31{:}00 \rangle$ = 0) to ALU Z V LONG (OVF$\langle 31 \rangle$) to ALU V CARRY 32 to ALU C

The sign bit, which is the high-order bit of the ALU result, is loaded into condition code flip-flop ALU N. It is the high-order ALU output (F SIGN) from the appropriate 2901A chip, that is, it is N BYTE (ALU$\langle 7 \rangle$) for byte operations, N WORD (ALU$\langle 15 \rangle$) for word operations, and N LONG (ALU$\langle 31 \rangle$) for longword operations. The sign bit equals zero when the ALU result is positive. It is a one when the result is negative.

The ALU Z flip-flop stores the state of the ALU's zero result indicators. Each 2901A has an open collector output (F = 0) that is asserted when all four ALU outputs in the chip are zero. The open collector outputs are wired together to assert Z BYTE when ALU $\langle 7{:}0 \rangle$ are all zeros, Z WORD when ALU$\langle 15{:}8 \rangle$ are all zeros, and Z LONG when ALU$\langle 31{:}16 \rangle$ are all zeros.

These signals are gated as necessary at the input to ALU Z to indicate a zero result for each data size. For example, all three signals are ANDed to load a longword zero result indication.

ALU overflow status is stored in the ALU V flip-flop. An ALU overflow is indicated by V BYTE, V WORD, or V LONG. These signals are the output of a 2901A output pin (OVERFLOW) that is logically the XOR of the carry-in and carry-out of the high-order ALU output in the chip. The pins used as input to ALU V are for the high-order bit of the ALU result (ALU $\langle 31,15,7 \rangle$), which indicates that the result of an arithmetic operation (twos complement) has overflowed into the sign bit for either a byte, word, or longword operation.

The ALU C flip-flop is set by the carry-out signals from the appropriate 2901A. These are CARRY 8 for byte operations, CARRY 16 for word operations, and CARRY 32 for longword operations.

The PSL CCs are detailed in Figure 6-17. They are loaded at the end of a CPU instruction's execution by a BASIC, MOVE, or EXTENDED. As with the ALU CCs, the load of the PSL CCs is dependent on data size. Also, the way the PSL CCs are loaded is dependent on the CC class defined for the instruction. The CC class (one of four) is specified by flip-flops OPC TYPE 1 and 0 in the OPC ROM/PAL, as shown in Table 6-16. OPC TYPE 1 and 0 are loaded during the class decode operation for the instruction.
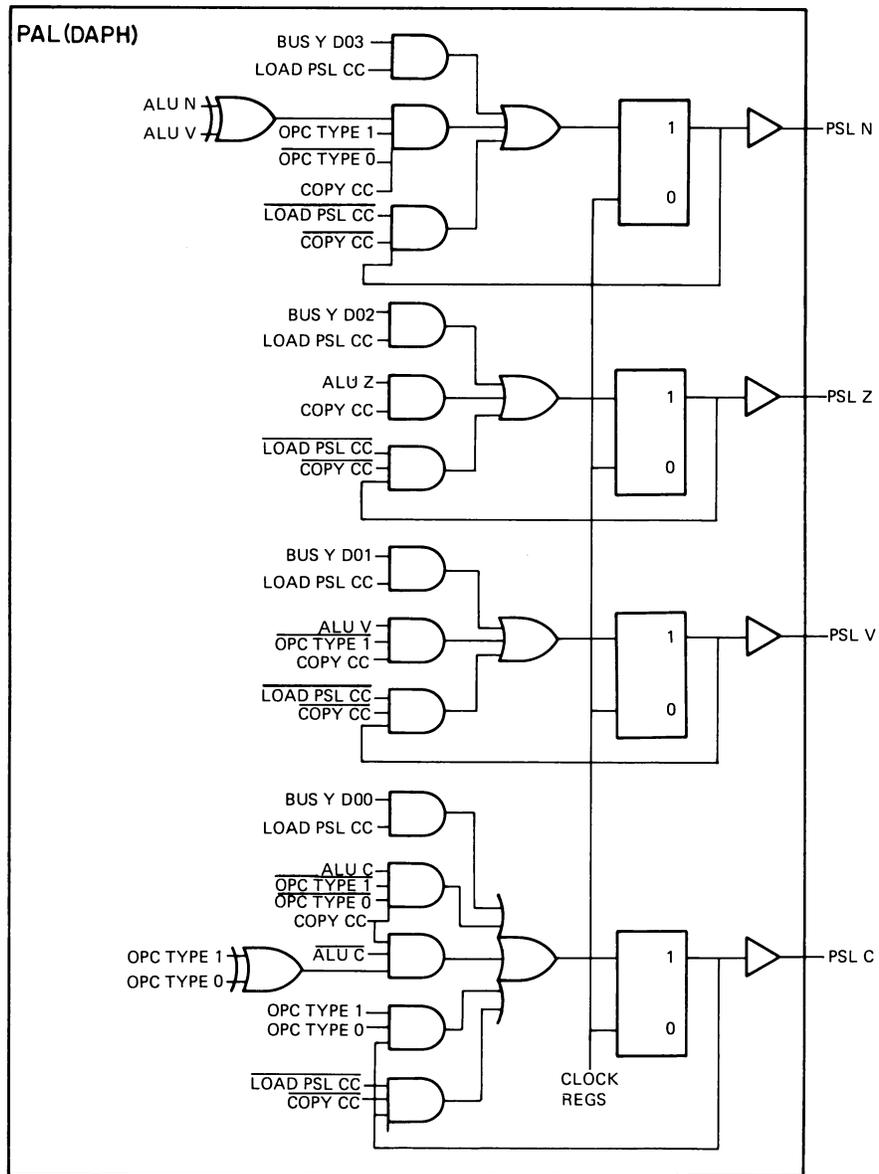
Figure 6-17   PSL Condition Codes

Table 6-16   ALU CC to PSL CC Transfer

Opcode Type	ALU CCs Loaded in PSL CCs
OPC TYPE ⟨1:0⟩ = 00 (Arithmetic add)	ALU N to PSL N ALU Z to PSL Z ALU V to PSL V ALU C to PSL C
= 01 (Arithmetic subtract)	ALU N to PSL N ALU Z to PSL Z ALU V to PSL V –ALU C to PSL C
= 10 (Compare)	ALU V XOR ALU N to PSL N ALU Z to PSL Z 0 to PSL V –ALU C to PSL C
= 11 (Logical)	ALU N to PSL N ALU Z to PSL Z 0 to PSL V Hold PSL C

If loading the PSL CCs, and the CC class as specified by OPC TYPE 1 and 0 is 00, a direct transfer of the ALU CCs to the PSL CCs takes place. This does not occur for the other CC classes, however. When OPC TYPE 1 and 0 are 01 and 10, the negation of ALU C is loaded into PSL C. Also, PSL C is not loaded at all for a CC class of 11; and ALU N is XOR'd with ALU V if loading PSL N when the CC class is 10. For both classes 11 and 10, PSL V is cleared during the CC transfer.

Two control signals, ALU CC F1 and F0, determine ALU CC operation. ALU CC F1 and F0 equal to 00, 01, and 10 load the ALU indicators for byte, word, and longword operations, respectively. When ALU CC F1 and F0 are 11, the ALU CCs are held at their current value. A third control signal, copy CC, enables the loading of the PSL CCs from the ALU CCs. The three control signals are generated as shown in Table 6-17.

CC operation is specified by the CC field of the BASIC, MOVE, and EXTENDED microinstructions. As previously discussed, this field (together with the size register and COMPAT MODE) also controls the data type logic generating DATA TYPE 1 and 0. As a result, in some cases, the DATA TYPE signals can be used in conjunction with the CC field to generate the required CC control signals.

A CC field value of 00 is a NOP. Both ALU CC F1 and F0 are asserted to hold the ALU CCs, and COPY CC is negated to hold the PSL CCs. A CC field of 01 is used to load the ALU CCs, following a longword operation in the ALU. CC equal to 10 uses the size register to define the loading of the ALU CCs. This is when the DATA TYPE signals are used to generate ALU CC F1 and F0. Because the DATA TYPE signals force word operation in compatibility mode when longword operation is specified by the size register, word operation is also forced when loading the ALU CCs in this instance.

Table 6-17 Generation of CC Control Signals

Microinstruction	DATA TYPE 1	DATA TYPE 0	ALU CC F1	ALU CC F0	COPY CC	CC Load Function
BASIC/MOVE/EXTENDED						
CC(CSR<06:05>)						
= 00 (Longword context, hold CCs)	–	–	1	1	0	No change
= 01 (Longword context, set CCs)	–	–	1	0	0	ALU CCs:longword
= 10 (Use size, set CCs)	0	0	0	0	0	ALU CCs:byte
	0	1	0	1	0	ALU CCs:word
	1	1	1	0	0	ALU CCs:longword*
= 11 (Copy CCs)	-		1	1	1	PSL CCs from ALU CCs
MEM						
REQ/DECODE/JUMP/						
MISC/PORT	–	–	1	1	0	NOP

*Cannot occur in compatibility mode. If size register specifies longword operation and COMPAT MODE = 1,
DATA TYPE <1:0> = 01 (word) causes ALU CC F1 AND F0 to equal 01 (Load ALU CCs:word).

When the CC field is 11, it asserts COPY CC to load the PSL CCs from the ALU CCs. The ALU CCs hold their current value following the transfer. Both the ALU CCs and PSL CCs hold their current value when microinstructions other than the BASIC, MOVE, or EXTENDED are executed.

The ALU CCs are microsequencer skip conditions and may be tested by the microcode. They may also be tested by reading them (through the D bus multiplexer) onto D bus⟨3:0⟩, and into the data path with a MOVE microinstruction using a discrete register address of FE. A MOVE can also write the ALU CCs (discrete register address FD or FE) from Y bus⟨3:0⟩ at the end of the microcycle (LOAD ALU CC = 1). This allows the microcode to set the ALU CCs to any value during an instruction's execution.

Of the PSL CCs, only PSL C is a microsequencer skip condition. However, like the ALU CCs, all four PSL CCs may be read into the data path through the D bus multiplexer and onto D bus⟨3:0⟩. They may also be loaded from Y bus ⟨3:0⟩ (LOAD PSL CC = 1). The PSL CCs may be read and written by both a BASIC and a MOVE. The discrete register address used is 7F or FF.

6.9 REGISTER READ/WRITE CONTROL

A number of the discrete registers in the CPU may be accessed by the BASIC and MOVE. (A discrete register is a hardware register that is not a local store or 2901A RAM location.) The discrete registers that may be accessed are the ALU and PSL CCs, and the OS, microstatus, PSL, and console write registers. Also, the HALF CARRY flip-flop may be accessed. When read, HALF CARRY is gated to generate a decimal constant during packed decimal arithmetic operations.

Discrete registers are read onto the D bus and written from the Y bus. Bus bit assignments are shown in Figure 6-18. Register read data is valid on the D bus shortly after the beginning of the microcycle when DISABLE-D BUS goes false, and the data remains valid for the rest of the microcycle. A register is written from the Y bus at the end of the microcycle when the register is clocked.
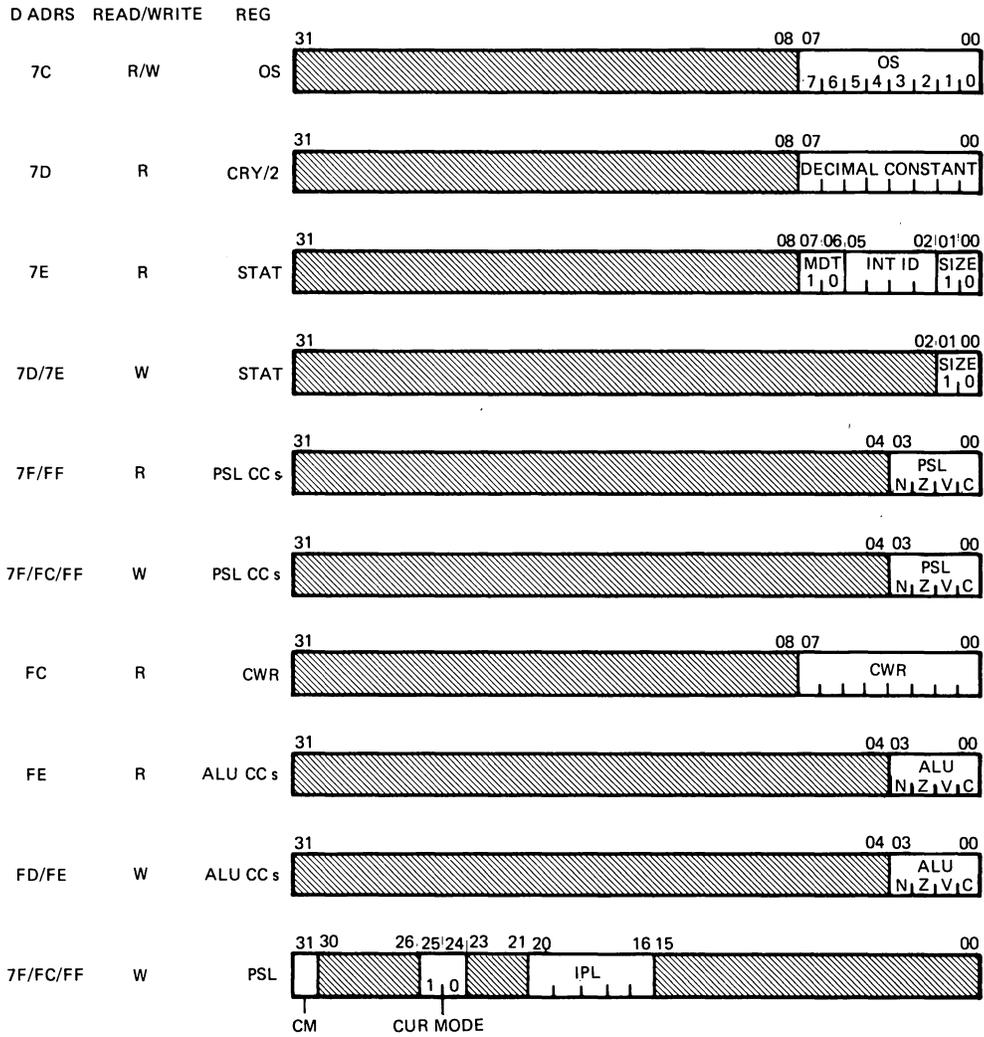
Figure 6-18 Discrete Register Read/Write Bit Assignments

The discrete registers are addressed by the D address fields of the BASIC and MOVE. As discussed previously, discrete register addresses are in the range 7C to 7F and FC to FF, causing SEL REGS to be asserted by the local store address logic. This signal, in conjunction with the D address and data path control fields of the BASIC and MOVE, asserts the control signals necessary to read and write the register.

The discrete register read/write control logic is shown in Figure 6-19. The control signals include three read enable levels (READ REGS, EN STAT REG, READ CONSOLE), a write enable (WRITE REGS), and two register select levels (REG SEL ⟨1:0⟩).

The MOVE or BASIC may read a register when one of the read enables is asserted, write a register when the write enable is asserted, or do both a read and write when a read enable is asserted together with a write enable. For registers which can be both read and written, register data read onto the D bus and into the data path may be examined and/or modified in the 2901A data processor during the BASIC or MOVE, and the unmodified or modified data written back into the register at the end of the same microcycle.
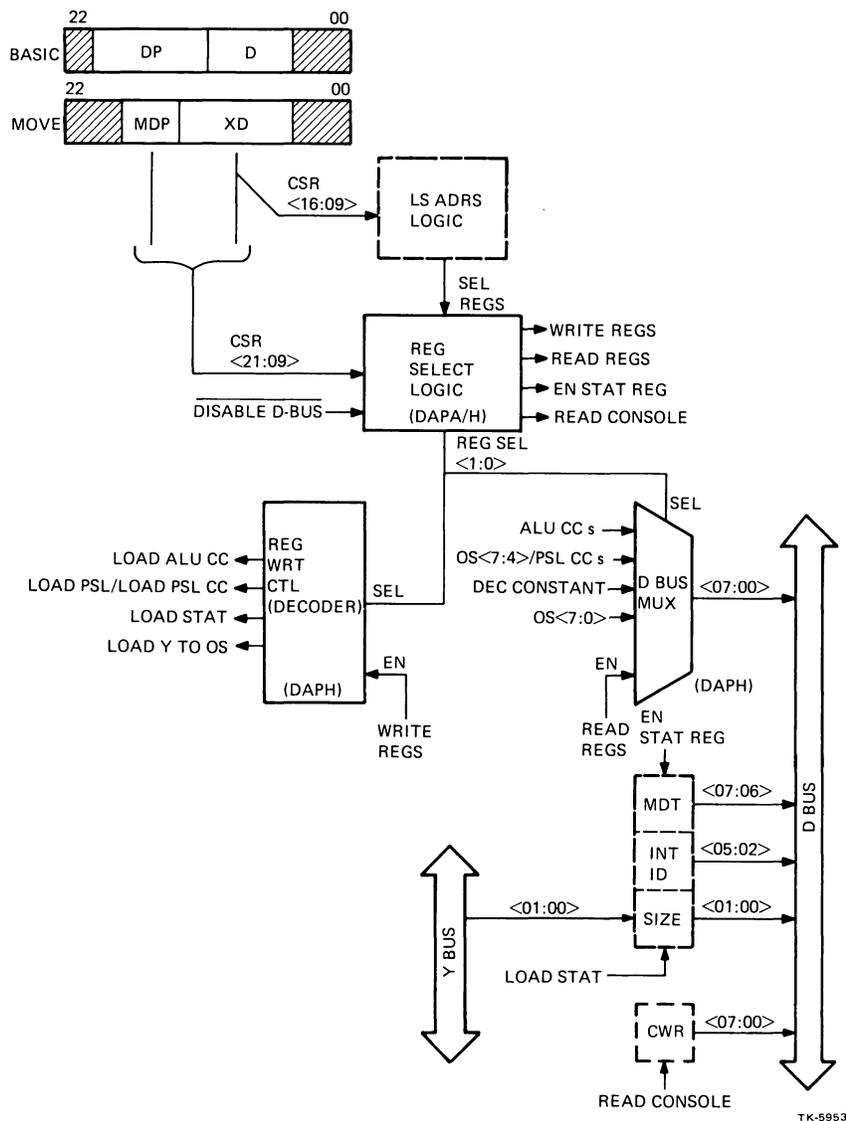
Figure 6-19   Register Read/Write Control

6-40

Except for the microstatus (STAT) and the console write register (CWR), register data is read onto the D bus through the D bus multiplexer. REG SEL ⟨1:0⟩ select the D bus multiplexer's inputs. READ REGS enables the multiplexer's outputs. The reading of the STAT and the CWR are controlled by EN STAT REG and READ CONSOLE, which cause the corresponding register flip-flops to drive the D bus directly. The STAT register consists of the MDT and size registers in the data type control, as well as the interrupt identifier code stored in the CPU's interrupt logic.

The same two levels that select register read data at the input to the D bus multiplexer also determine the register to be written. By selecting the output of a decoder, REG SEL ⟨1:0⟩ generate the appropriate write enable signal (LOAD Y TO OS, LOAD STAT, etc.) when WRITE REGS is asserted.

Table 6-18 shows the read/write control signals generated by the BASIC and MOVE. The BASIC's 7-bit D address may specify a discrete register address in the 7C to 7F range. As a result, it may access the OS and STAT registers, the PSL CCs, and the decimal constant. The MOVE can do the same, but with an additional D address bit, it may specify discrete register addresses in the range FC to FF, allowing access to the ALU CCs, the CWR, and the PSL.

Table 6-18   Register Read/Write Control Signal Generation

Microinstruction	D Address	REG SEL 1 0	Read Signal*	Register Read*	Write Signal†	Register Written†
BASIC	7C	1 1	–	OS	LOAD Y TO OS	OS
	7D	1 0	–	CRY/2	LOAD STAT	STAT
	7E	1 0	EN STAT REG	STAT	LOAD STAT	STAT
	7F	0 1	–	PSL CCs	LOAD PSL CC	PLS CCs
MOVE						
MDP = 0/1/3/4/5/7	7C	1 1	–	OS	LOAD Y TO OS	OS
	7D	1 0	–	CRY/2	LOAD STAT	STAT
	7E	1 0	EN STAT REG	STAT	LOAD STAT	STAT
	7F/FF	0 1	–	PSL CCs	LOAD PSL CC	PSL CCs
					LOAD PSL	PSL
	FC	0 1	READ CONSOLE	CWR	LOAD PSL CC	PSL CCs
					LOAD PSL	PSL
	FD	0 0			LOAD ALU CC	ALU CCs
	FE	0 0	–	ALU CCs	LOAD ALU CC	ALU CCs
MDP = 2/6	–	1 1	–	OS		

*The CWR (console write register) and the STAT register are not read by a MOVE when MDP = 0/2/4/6. Other registers are not read when MDP = 0/4. Register reads are enabled by REG SEL <1:0>, except when READ CONSOLE = 1 and EN STAT REG = 1.

†No register is written by a BASIC when DP = <00:1F>, or by a MOVE when MDP = 1/2/3.

The decimal constant and the CWR are read-only registers. The PSL is a write-only register, although the PSL CCs may be both read and written. When writing the PSL, an additional bit (T TRAP) is loaded, but not directly from the Y bus. T TRAP, which causes a hardware interrupt (Paragraph 7.5), is set by signal T or TP (trace enabled or trace pending). T TRAP is the OR of Y bus bits 30 and 4.

NOTES

There are two copies of the PSL, the discrete part which supports hardware processing and another which is kept in a local store location (1D). Those bits in the discrete PSL which are modified directly by the microcode (by the BASIC or MOVE) are also copied in the local store location. (These are compatibility mode, current mode, and IPL control bits.)

However, the PSL CCs are not updated in local store because these bits are set or cleared as a result of a hardware (ALU) calculation. Therefore, when the PSL is to be stored, it requires that the microcode first read the PSL CCs (with a BASIC or MOVE), and then assemble the current PSL from the PSL CCs and the contents of the local store location.

Whereas the discrete register address in the D address field of the BASIC or MOVE determines the register accessed, the microinstruction's data path control field determines whether the access is read-only, write-only, or both a read and write. Register writes are inhibited for the BASIC when DP = 00 to 1F, and for the MOVE when MDP = 1, 2, or 3.

These values of DP and MDP correspond to the data path operations that do not write local store from the 2901A data processor. Conversely, this means that any time a local store location can be addressed to store the 2901A outputs, a discrete register address may be used instead to store the data in a writable discrete register.

The only restrictions when reading discrete registers are for the MOVE. When MDP = 0 or 4, no register can be read. (Memory read data is present on the D bus in these situations.) Table 6-19 gives the register addresses and summarizes the various read/write restrictions when accessing the discrete registers with the BASIC and MOVE.

6.10 SIGN EXTENSION CONTROL

When a word or byte of read data from the memory controller is gated from the MC bus onto the D bus and into the 2901A data processor during a MOVE, the sign extension control transmits copies of the sign bit on the high-order D bus data lines to aid in the data processing. For example, if a byte of read data which is gated onto data lines 7 to 0 is negative (data line 7 = 1), ones are transmitted on the high-order data lines 31 to 8 by the sign extension control.

Similarly, for a word of read data which is gated onto data lines 15 to 0, copies of the sign bit (data line 15) are transmitted on high-order lines 31 to 16. The MC bus transceivers for the high-order data lines are disabled by the data type control during the read data transfers. This is so that the sign extension control may drive the D bus.

Table 6-19   Discrete Register Address and Read/Write Summary

Register	Microinstruction	D Address	Read Only	Write Only	Read/Write
OS	BASIC	7C	YES (DP = 00:1F)	NO	YES (DP = 20:3F)
	MOVE	7C	YES (MDP = 1/3)	YES (MDP = 0/4)	YES (MDP = 5/7)
	MOVE	–	YES (MDP = 2/6)	NO	NO
CRY/2	BASIC	7D	YES	NO	NO
	MOVE	7D	YES (MDP = 1/3/5/7)	NO	NO
STAT	BASIC	7D	NO	YES (DP = 20:3F)	NO
	BASIC	7E	YES (DP = 00:1F)	NO	YES (DP = 20:3F)
	MOVE	7D	NO	YES (MDP = 0/4/5/7)	NO
	MOVE	7E	YES (MDP = 1/3)	YES (MDP = 0/4)	YES (MDP = 5/7)
PSL CCs	BASIC	7F	YES (DP = 00:1F)	NO	YES (DP = 20:3F)
	MOVE	7F	YES (MDP = 1/3)	YES (MDP = 0/4)	YES (MDP = 5/7)
	MOVE	FC	NO	YES (MDP = 0/4/5/7)	NO
	MOVE	FF	YES (MDP = 1/3)	YES (MDP = 0/4)	YES (MDP = 5/7)
CWR	MOVE	FC	YES (MDP = 1/3/5/7)	NO	NO
ALU CCs	MOVE	FD	NO	YES (MDP = 0/4/5/7)	NO
	MOVE	FE	YES (MDP = 1/3)	YES (MDP = 0/4)	YES (MDP = 5/7)
PSL	MOVE	7F	NO	YES (MDP = 0/4/5/7)	NO
		FC	NO	YES (MDP = 0/4/5/7)	NO
		FF	NO	YES (MDP = 0/4/5/7)	NO

The sign extension control also transmits high-order sign bits when the OS register contents are read onto the D bus and into the 2901As during a BASIC or MOVE. The sign bit for the 8-bit OS register is OS⟨7⟩.

The sign extension control is shown in Figure 6-20. Multiplexers select the appropriate sign bit (OS⟨7⟩ or MC bus ⟨15⟩ or ⟨7⟩) depending on the data transfer, and sign-send logic enables tri-state output latch circuits to transmit the selected sign bit on the high-order D bus data lines. There are two sets of tri-state output latch circuits, one to transmit the sign bit on data lines 31 to 16, and another to transmit the sign bit on lines 15 to 8.
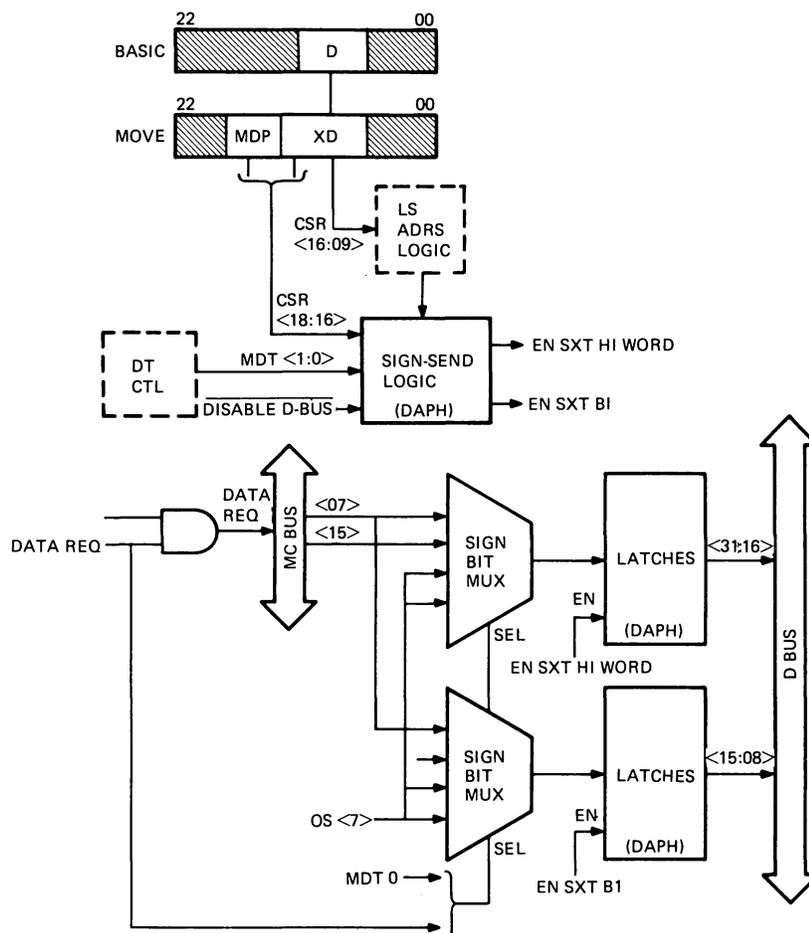
Figure 6-20   Sign Extension Control

Sign bit selection by the multiplexers is shown in Table 6-20. OS register data, not MC bus data, can be transferred by the BASIC; OS⟨7⟩ is selected as the sign bit for this microinstruction. During the MOVE, which transfers both OS and MC bus data, DATA REQ = 1 indicates a memory controller reference. Thus, DATA REQ is used to select MC bus ⟨15⟩ or ⟨7⟩, depending on the state of MDT 0 in the data type control. (The MDT register indicates the size of the data transfer, and MDT 0 is cleared for byte transfers and set for word transfers.) When DATA REQ is not asserted for a MOVE, OS ⟨7⟩ is selected as the sign bit.

Table 6-20   Sign Bit Selection

Microinstruction	DATA REQ (MOVE:MDP=0/1/4)	MDT 0	Sign Bit
BASIC	–	–	OS⟨7⟩
MOVE	0	–	OS⟨7⟩
1	0	MC⟨07⟩	
1	1	MC⟨15⟩	

Two signals control the latch circuits. EN SXT HI WORD enables the set of latches that transmit the selected sign bit on data lines 31 through 16; EN SXT B1 enables the latches that transmit the sign bit on lines 15 to 8. Both signals are asserted when the OS or a byte of MC bus data is transferred on the D bus. Only EN SXT HI WORD is asserted when a word of MC bus data is present on the D bus. The two latch enables are generated as shown in Table 6-21.

Table 6-21   Sign Send Control

Microinstruction	D Address	MDT 1	MDT 0	EN SXT H1 WORD	B1	Remarks
BASIC	00:7B	–	–	0	0	No sign send
	7C:7F	–	–	1	1	OS<7> to D D<31:08>
MOVE						
MDP = 0/4	–	0	0	1	1	MC D<07> to D D<31:08>
	–	0	1	1	0	MC D<15> to D D<31:16>
	–	1	1	0	0	No sign send
MDP = 3/7	7C:7F,FC:FF	–	–	1	1	OS<7> to D D<31:08>
MDP = 2/6	00:7F	–	–	1	1	OS<7> to D D<31:08>
EXTENDED/MEM REQ/						
MISC/JUMP/DECODE	–	–	–	0	0	No sign send

6-45

During a BASIC, both latch enables are asserted to transmit OS$\langle 7 \rangle$ as the sign bit (selected by the multiplexers) whenever the D address is for a discrete register. The microprogram ignores the trailing OS sign bits when the BASIC reads a discrete register other than OS over the D bus.

During a MOVE, both latch enables are asserted to transmit the OS sign bit for discrete register addresses, but only when the data path control field (MDP) is 2, 3, 6, or 7. Of these, only MDP equal to 3 and 6 actually move discrete register data from the D bus into the 2901As. (MDP = 6 always moves OS register data.)

For values of MDP that move read data from the MC bus, which are MDP = 0 or 4, the latch enables asserted depend on the size of the transfer as indicated by the MDT register. MDT 1 and 0 equal to 00 indicates a byte transfer and asserts both latch enables; MDT 1 and 0 equal to 01 indicates a word transfer asserting only EN SXT HI WORD. No latch enables are asserted, and no trailing sign bits need be transmitted, when MDT 1 and 0 are 11. In this case, all 32 data lines are being used to transfer a longword of MC bus data over the D bus.

6.11 MEMORY REFERENCES
A memory reference made to the memory controller (MCT) by the CPU usually requires the execution of at least two microinstructions. A MEM REQ microinstruction is executed to generate a memory request, and to transfer address information to the MCT over the MC bus. This is followed by a MOVE, which makes a data request and then transfers a longword of read/write data from/to the MCT over the MC bus.

Exceptions to the two-microinstruction (MEM REQ/MOVE) memory reference include quadword and octaword references and references to fill the PFR with instruction data. Quadword references require two MOVEs following the MEM REQ to transfer the two longwords of memory data. Octaword references require four MOVEs to transfer four longwords. When refilling the PFR (see Paragraph 5.2.1), a DECODE or MEM REQ makes the memory request, but a MOVE is not required. The PFR is loaded directly from the MC bus (by MC bus signal LOAD IB) without any data request by the CPU.

NOTE
Whereas octawords may be both read and written in the VAX-11/730 system, only a quadword read reference is implemented.

Timing for a CPU memory reference by the MEM REQ and MOVE microinstructions is shown in Figure 6-21. The MEM REQ microinstruction asserts MEMORY REQ on the MC bus at the beginning of the microcycle. (A flow diagram for the MEM REQ is given in Figure 6-22.) Also asserted on the bus are DATA TYPE 1 and 0, which are generated by the CPU's data type control; CSR $\langle 19 \rangle$, which is a one for a MEM REQ microinstruction; and CSR $\langle 18:16 \rangle$, and CSR $\langle 08:07 \rangle$, which are the MEM REQ's memory function fields.

The DATA TYPE signals specify the size of the memory data transfer (byte, word, or longword) and are controlled by the MEM REQ's DT field as discussed previously. (The DATA TYPE signals are sometimes ignored, such as when making a quadword or octaword reference.) The memory function lines specify the type of reference. For example, making CSR $\langle 18:16 \rangle$ and CSR $\langle 08:07 \rangle$ equal to 01000 specifies the read of a virtual memory address.
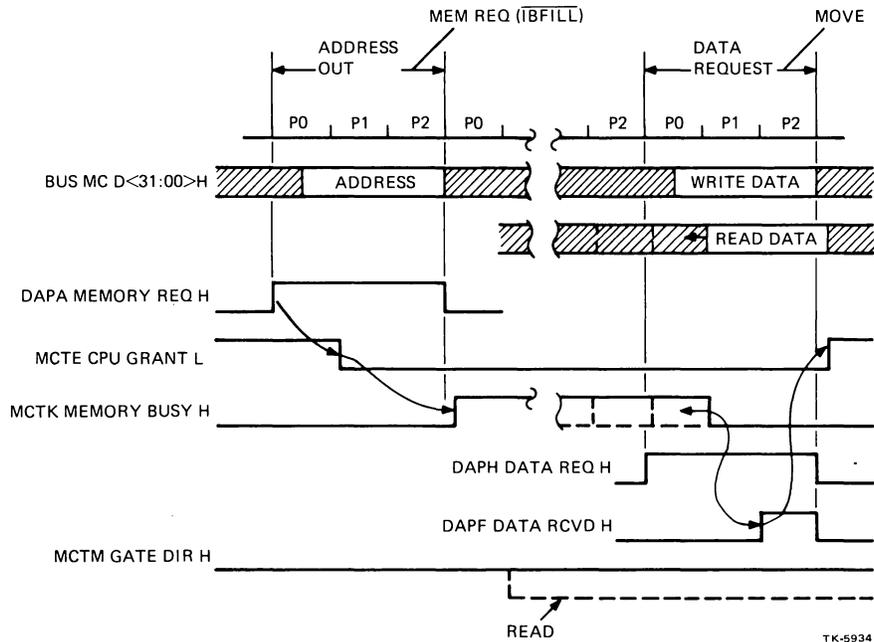
Figure 6-21   CPU Memory Reference Timing Diagram

Figure 6-22   MEM REQ Microinstruction Flow Diagram
(Sheet 1 of 2)

6-47

FROM SHEET 1

① 1

CPU GRANT (FROM MCT)

STALL CLOCK UNTIL
CPU GRANT RECEIVED
FROM MEMORY

CLOCK STALL = 1

MREQ
IBFILL

NO

CSR<18:16, 08:07> = 01110

YES

CLK·CPU P2

CPU LEAVING
ADRS-OUT
STATE AFTER
MREQ.IBFILL REQ

DATA RCVD = 1
(TO MCT)

LOAD IB (FROM MCT)

CLK

CLOCK P2·CLK

LOAD INSTRUCTION
DATA IN PFR
REGISTER

MC BUS → PFR

1 → IB VALID

TK-5942

Figure 6-22   MEM REQ Microinstruction Flow Diagram
(Sheet 2 of 2)

In addition to the DATA TYPE and memory function signals, three PSL status bits are transmitted on the MC bus to indicate machine mode. These are COMPAT MODE and CURR MODE 1 and 0. When the machine is in compatibility mode, COMPAT MODE prevents access to words in memory that are not aligned on word boundaries. The CURR MODE signals indicate the privilege level of the currently executing program. Depending on the access mode (e.g.; kernal, executive, etc.), reads and/or writes to certain memory addresses may not be allowed, as determined by the operating system and as specified by the protection bits in the MCT's translation buffer.

Following the assertion of MEMORY REQ, the MEM REQ microinstruction reads 32 bits of address information from local store, and enables all MC data line transceivers to gate the address from the D bus onto the MC bus. The local store location containing the address information is specified by the MEM REQ's D address field.

When the MCT is ready to accept the address transmitted on the MC bus and begin the operation specified by the memory function signals, it asserts CPU GRANT on the MC bus. If this signal is not asserted immediately (during CPU clock phase 1), CPU clock phase 2 is stalled, delaying further execution of the MEM REQ. (A stall can occur if the MCT is already active transferring UNIBUS NPR data.)

When CPU GRANT is asserted, the stall (if any) is released and the MEM REQ completes its execution by storing the address transmitted on the MC bus in 2901A working register 5, and by storing the state of the DATA TYPE signals in the MDT register in preparation for handling the read/write data during the upcoming MOVE (or MOVEs). After asserting CPU GRANT, the MCT asserts MEMORY BUSY on the MC bus to indicate it is busy processing the memory request.

Following the MEM REQ and with MEMORY BUSY equal to 1, the microprogram usually executes one non-memory related microinstruction before doing a MOVE. (The MCT cannot be ready to accept write data or to return read data any sooner than this.) When the MOVE is executed, its operation depends on its MDP function field; that is, whether it is to transfer write data to the MCT (MDP = 1), or whether it is to transfer read data from the MCT (MDP = 0 or 4). A flow diagram for the MOVE is shown in Figure 6-23.
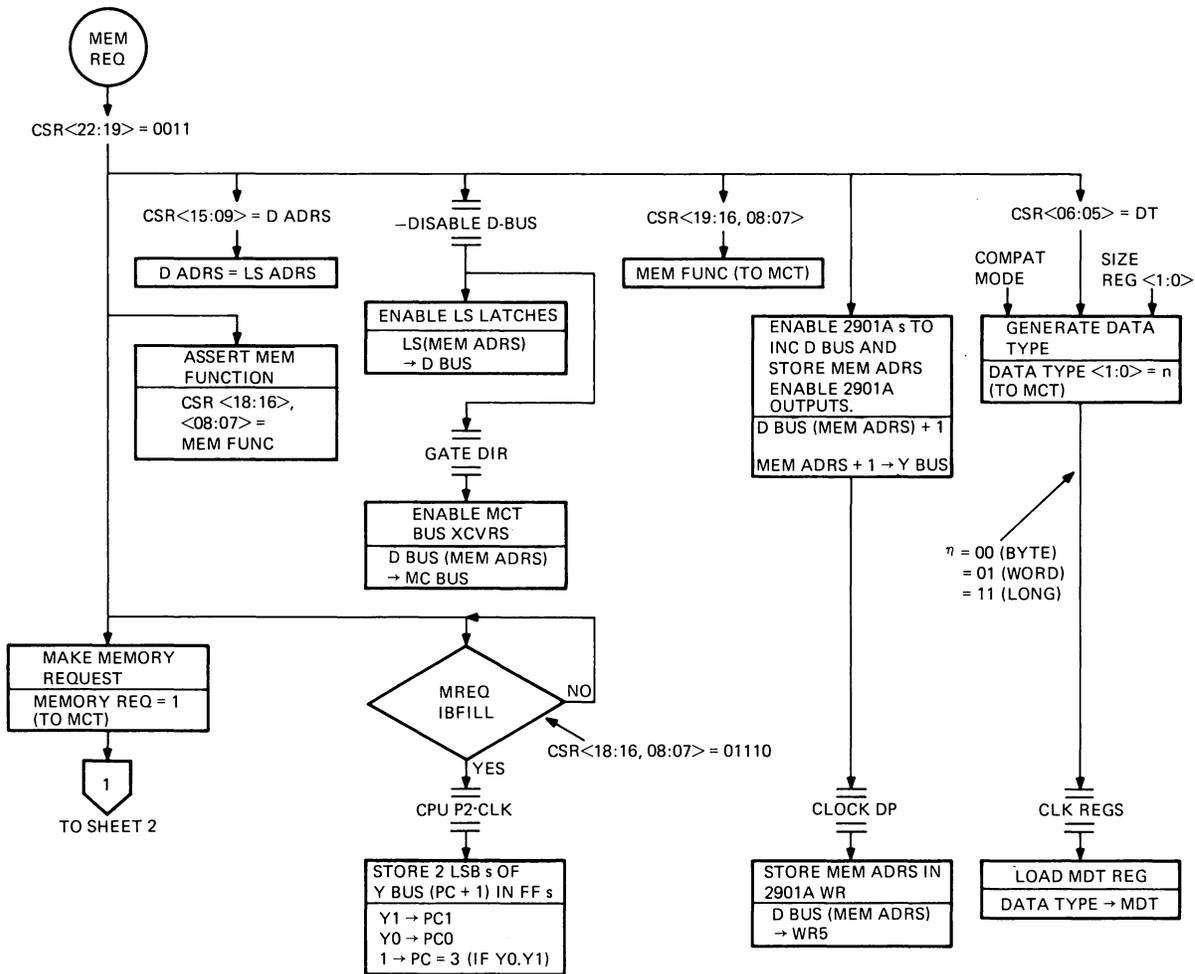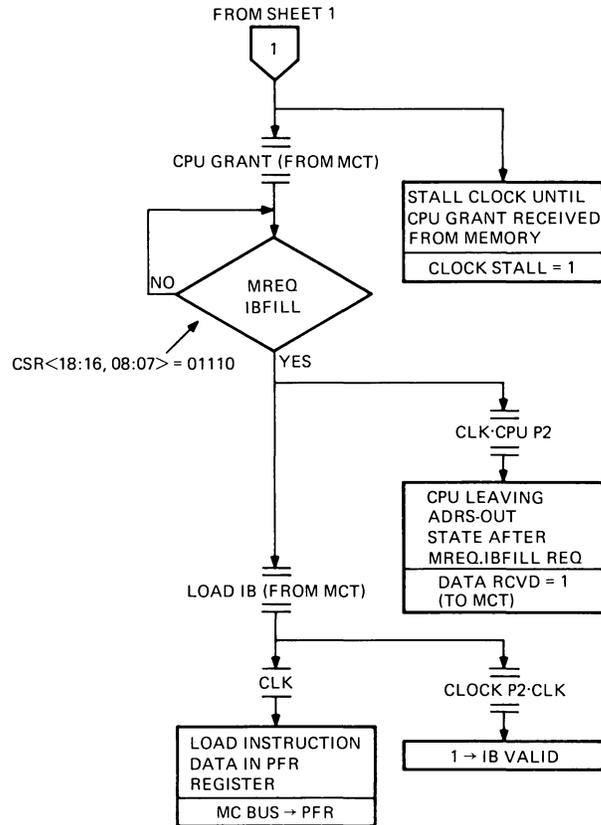

Figure 6-23   MOVE Microinstruction Flow Diagram (Sheet 1 of 3)

6-49

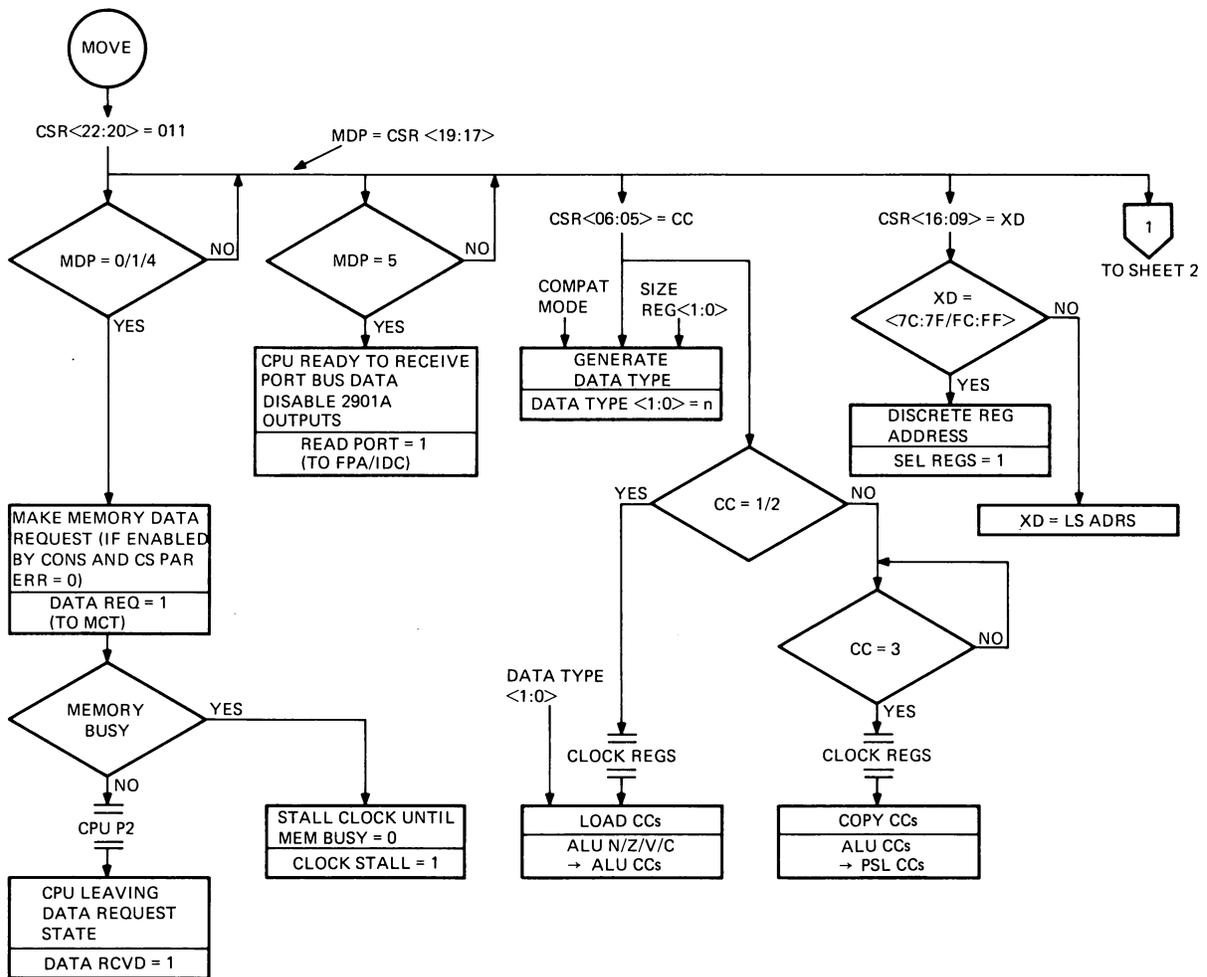Figure 6-23 MOVE Microinstruction Flow Diagram (Sheet 2 of 3)

FROM SHEET 2

2

MDP = 2   YES

NO

WR ADRS = B
(SEE NOTE)

WR ADRS = B + 4
(SEE NOTE)

ENABLE 2901A s FOR
OPERATION SPECIFIED
BY MDP.ENABLE
2901A OUTPUTS
UNLESS
MDP= 5 (PORT XFR)

CLOCK DP

MDP	2901A OPERATION
0	WR (B) → Y BUS, D BUS (MEM DATA) → WR(B)
1	NOP: WR(B) → Y BUS
2	WR (B + 4) → Q/Y BUS
3	D BUS → WR(B)/Y BUS
4	D BUS (MEM DATA) → Y BUS
5	(NO OUTPUT)
6	D BUS (OS) + WR(B) → Y BUS
7	WR(B) → Y BUS
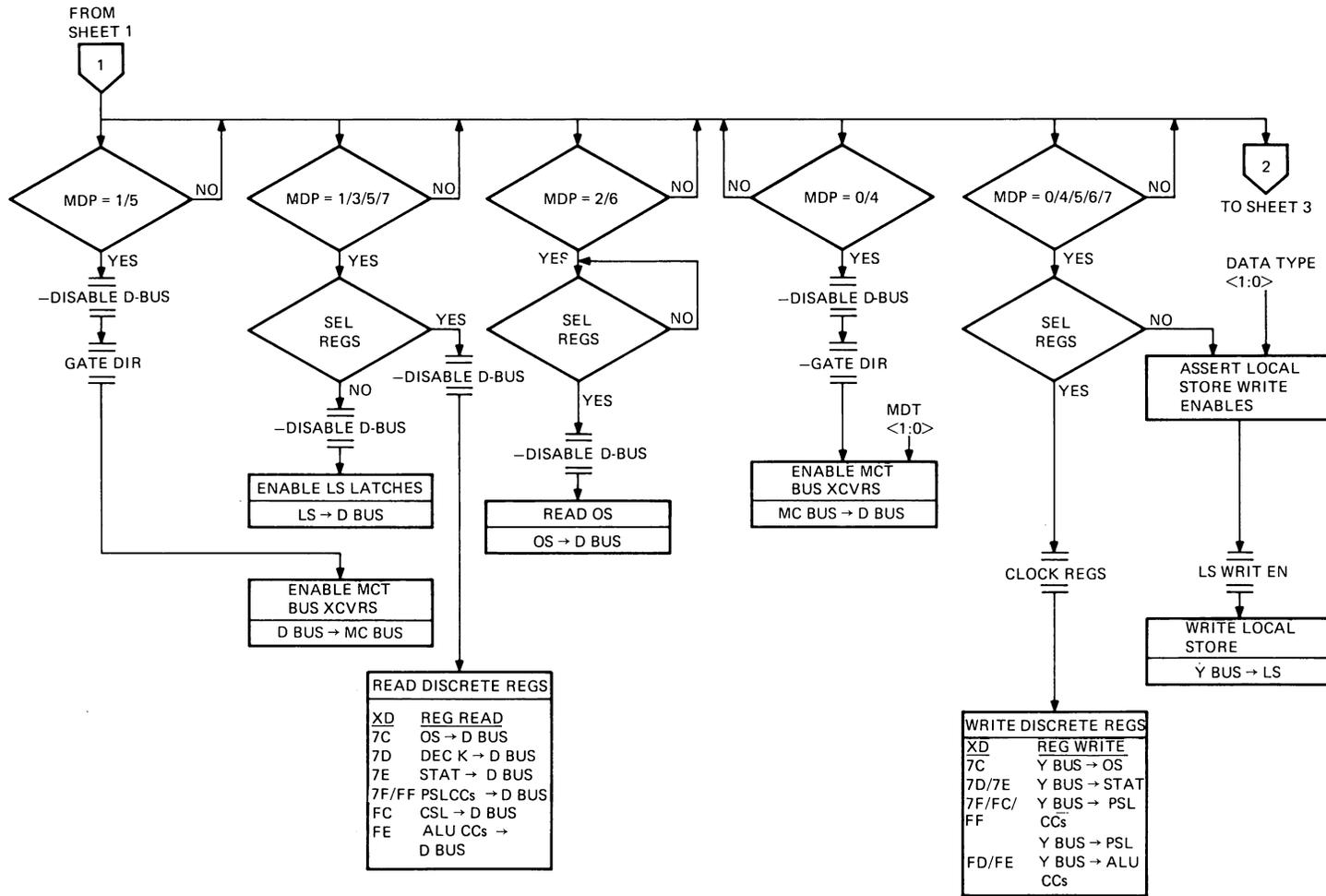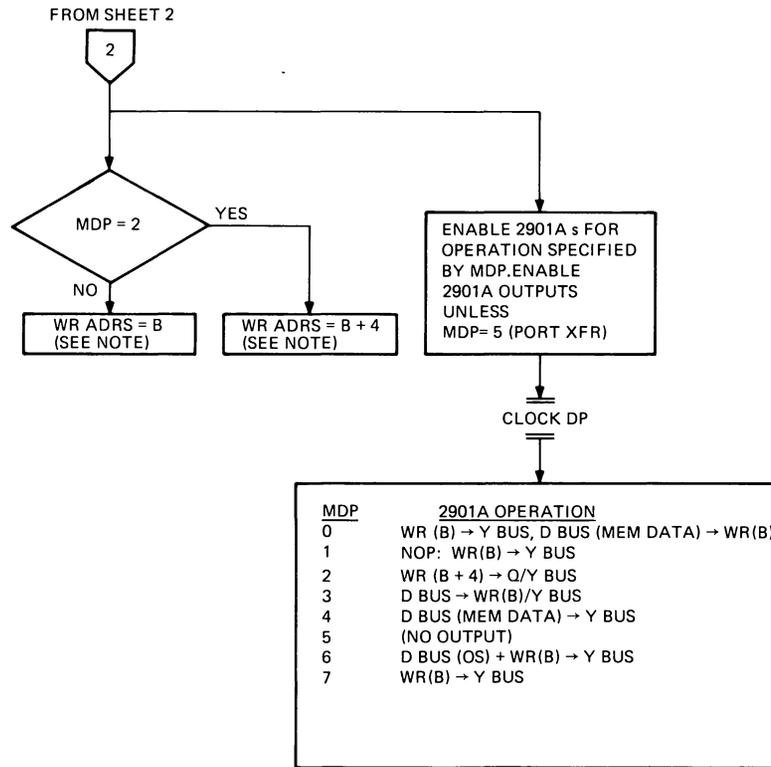
NOTE: WORKING REGISTER ADDRESS (B)
        SPECIFIED BY CSR <08:07>.

TK-5947

Figure 6-23   MOVE Microinstruction Flow Diagram (Sheet 3 of 3)

The MOVE requests the transfer of read/write data by asserting DATA REQ on the MC bus at the beginning of the microcycle. For a write transfer, the write data is also read from local store (the location specified by the MOVE's D address field), and the data line transceivers for the MC bus are enabled to gate the write data from the D bus to the MC bus. As when address information is transmitted to the MCT by the MEM REQ, all 32 bits read from the local store location are enabled onto the MC bus by the MOVE. The high-order data lines not carrying write data during byte and word transfers are ignored by the MCT.

If the data transfer is a read, the transceivers are also enabled, but only the transceivers for the lines carrying read data. That is, for byte and word transfers, the transceivers for the high-order lines are disabled so that the sign extension control may append trailing sign bits to the read data.

The transceiver enables are EN XCVR HI WD, which enables bus bits 31 to 16; EN XCVR B1, which enables bits 15 to 8; and EN XCVR B0, which enables bits 7 to 0. They are generated as shown in Table 6-22. All three enables are asserted to pass the 32 bits of address data during the MEM REQ (and also the DECODE when initiating a refill of the PFR), and during the transfer of write data by the MOVE.

Table 6-22 Generation of MC Bus and D Bus Transceiver Enables

Microinstruction	GATE DIR	MDT 1	MDT 0	EN H1 WD	XCVR B1	XCVR B0	REMARKS
MEM REQ/DECODE	1	–	–	1	1	1	Address to MCT
MOVE (MDP = 1)	1	–	–	1	1	1	Write data to MCT
MOVE (MDP = 0/4)	–	0	0	0	0	1	Read data (byte) to CPU
	–	0	1	0	1	1	Read data (word) to CPU
	–	1	1	1	1	1	Read data (longword) to CPU

However, during the transfer of read data by the MOVE, the MDT register outputs disable the appropriate high-order lines. The transceivers are also conditioned by GATE DIR. This MC bus signal is normally asserted by the MCT to allow address and write data to pass from the D bus to the MC bus. However, following a MEM REQ requesting a read transfer, the MCT negates BUS DIR, allowing data to pass from the MC bus to the D bus.

Following the assertion of DATA REQ by a MOVE during a write data transfer, the MCT, if not busy (MEMORY BUSY = 0), takes the write data off the MC bus and begins (or continues) the write reference. In the CPU, MEMORY BUSY = 0 allows clock phase 2 to be generated, which asserts DATA RCVD on the bus. This in turn causes the MCT to negate CPU GRANT, ending MC bus dialogue for the transfer of one longword of write data.

For a read transfer, the read data from the MCT is valid on the MC bus whenever the MCT negates MEMORY BUSY. The read data is gated onto the D bus and into the 2901As by the MOVE, and (again) MEMORY BUSY = 0 allows CPU clock phase 2 to be generated. Clock phase 2 completes execution of the MOVE and asserts DATA RCVD on the MC bus to signal the end of the bus transfer. The read data is stored in a working register when the MOVE's MDP field is zero; it is stored in local store if MDP is equal to four.

If the MCT is still busy (during clock phase 1) when a MOVE asserts DATA REQ and attempts to transfer data, CPU clock phase 2 is stalled, delaying the MOVE and the requested data transfer. The stall is released when MEMORY BUSY = 0.

Stalls are caused by memory refresh cycles, soft read data errors, or because read data is not aligned in memory. Hard errors (such as nonexistent memory and uncorrectable read errors) or conditions requiring intervention by the memory management microcode (such as TB entry not valid and access violation) may stall the MOVE; but when the condition is detected, the MCT asserts ERR SUM on the MC bus and negates MEMORY BUSY to release the stall. The ERR SUM (error summary) signal is a microsequencer skip condition, and MOVEs accessing the MCT usually skip on this error flag in order to dispatch to the memory management microcode when necessary.

To determine why ERR SUM has been asserted, the memory management microcode reads the CSRs (controller status registers) in the MCT, using a MEM REQ and MOVE as when making other types of memory references.

6.12 FPA/PORT DEVICE TRANSFERS

Data is transferred to and from the FPA and a port device over the Y bus. Control is by the CPU's MISC/PORT and MOVE microinstructions. (A flow diagram for the MISC/PORT is shown in Figure 6-24.) Opcodes are also transmitted to the FPA over the IB bus when a class decode operation is executed by a DECODE.
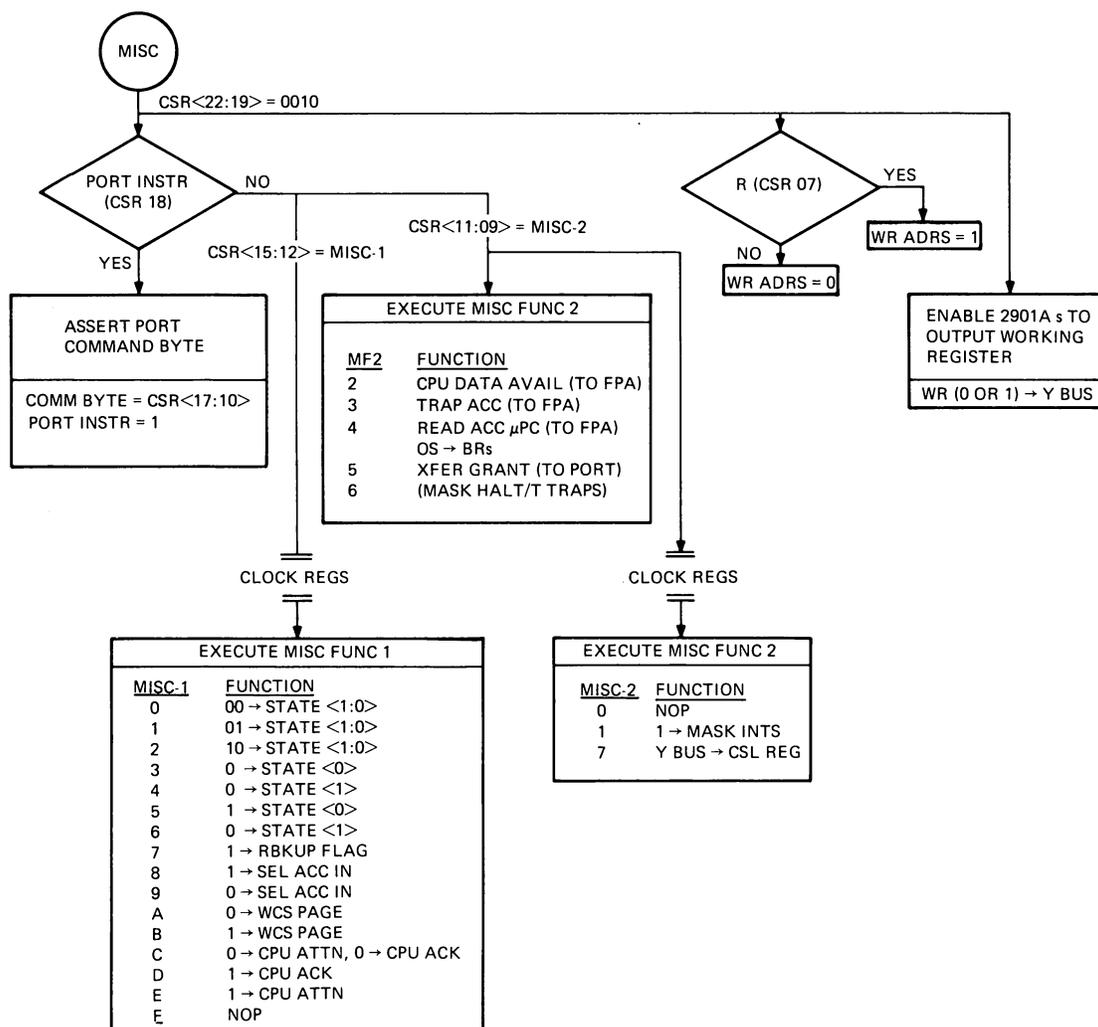
Figure 6-24   MISC/Port Microinstruction Flow Diagram

6-53

Timing for the transfers that take place between the CPU and the FPA are shown in Figure 6-25. The FPA examines the opcodes of all instructions being processed by the CPU by monitoring the IB bus during class decode operations when IRD STATE is true. When the opcode is for one of the instructions that are executed by the FPA in conjunction with the CPU (ADDF, POLY, EMOD, etc.), the FPA first accepts operand data from the CPU. It then performs the calculations specified by the instruction and sends the results back to the CPU.

The transfer of operand data to the FPA is made by the CPU's MISC microinstruction over the Y bus. Transfers are one longword at a time, and one MISC after the other is executed until all operand data has been sent. The MISC always transmits the contents of a 2901A working register on the Y bus, but if the MISC's function 2 field is equal to 2, it also asserts CPU DATA AVAIL to the FPA, indicating the Y bus data is operand data.
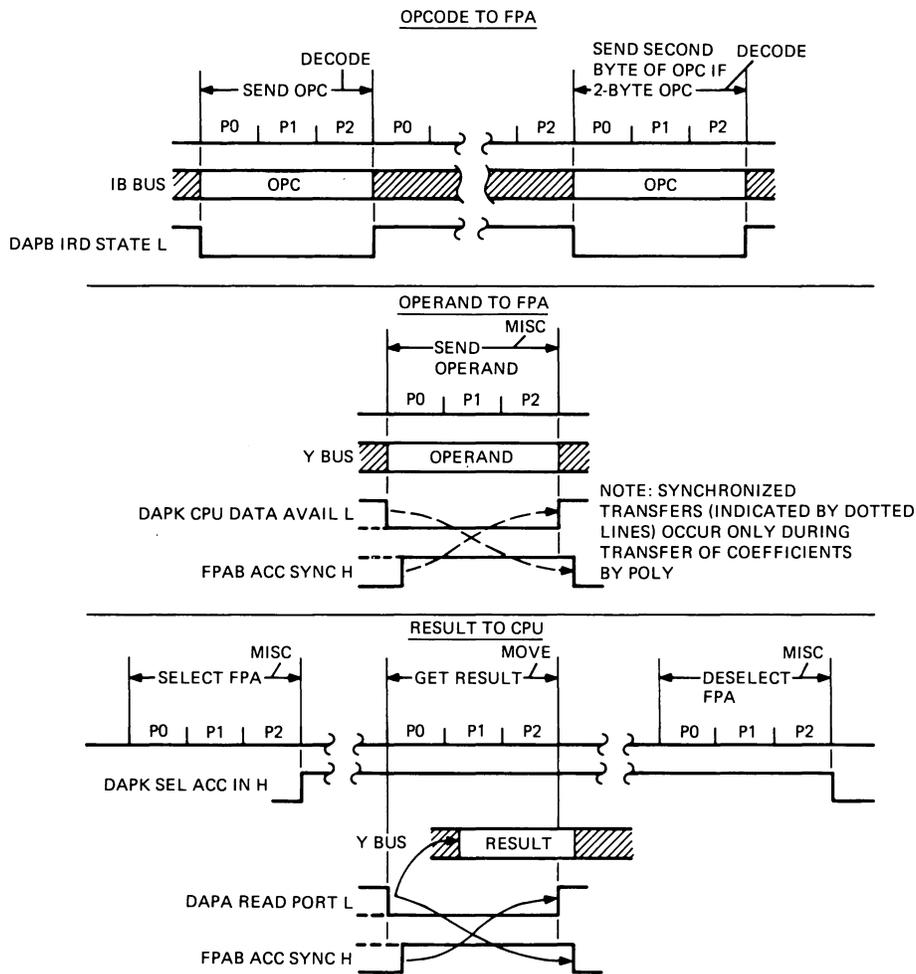
Figure 6-25 CPU/FPA Transfers Timing Diagram (Sheet 1 of 2)
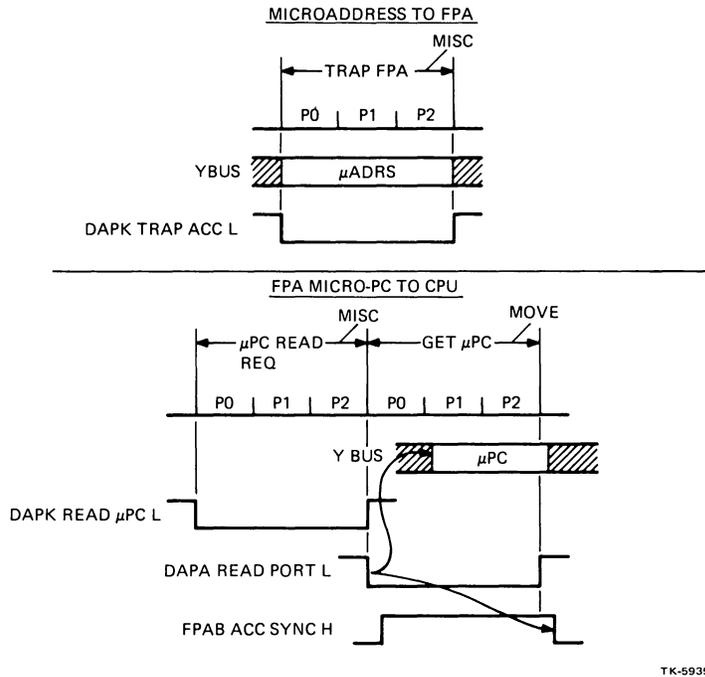
Figure 6-25   CPU/FPA Transfers Timing Diagram (Sheet 2 of 2)

For all instructions except the POLY, operand data is taken by the FPA as fast as it can be sent by the CPU. However, during the POLY, coefficients must be processed by the FPA between transfers, and (as a result) the FPA is not always ready to accept data when the CPU asserts CPU DATA AVAIL. To synchronize transfers, the FPA asserts ACC SYNC when it is ready for data. Also, the CPU micro-instruction transferring the data (the MISC) is contained in a loop address and enabled to loop on ACC SYNC = 0 (Paragraph 4.4.8).

When the FPA is asserting ACC SYNC waiting for data, it responds to CPU DATA AVAIL by taking the data and negating ACC SYNC at the end of the current CPU microcycle (the MISC). If the FPA is not yet ready for the data (ACC SYNC = 0 when CPU DATA AVAIL is asserted by the CPU), the CPU's MISC instruction loops (repeats) until ACC SYNC is asserted, indicating the data has been removed from the Y bus.

As can be seen, ACC SYNC and CPU DATA AVAIL work together to synchronize the transfer of each longword of operand data. That is, when only one synchronizing signal is asserted, the device as-serting the signal (either the FPA or CPU) will stall operations until the other device asserts its own synchronizing signal, allowing the data transfer to complete.

Once operands have been processed by the FPA, the results must be transferred to the CPU. Again, control is by the CPU and transfers are over the Y bus, one longword at a time. However, a MOVE microinstruction with its MDP field equal to 5 is used by the CPU instead of a MISC.

The MOVE disables the 2901A outputs in the CPU and asserts synchronizing signal READ PORT to the FPA. The FPA responds if it is ready for the transfer by placing the result data on the Y bus. The MOVE then stores the data in the local store location addressed by its D address field.

As for the transfer of coefficients during a POLY, ACC SYNC = 1 indicates the FPA is ready for a data transfer and the CPU microinstruction, which is asserting its own synchronizing signal (READ PORT in this case), loops on ACC SYNC = 0. Whenever both synchronizing signals are asserted, the transfer is made.

During the transfer of results by the MOVE, the FPA must be selected by the CPU generated signal, SEL ACC IN. This is a flip-flop controlled by the MISC and it must be set prior to the MOVE. It is set when the MISC has a function 1 field value of 8. It is cleared when the function 1 field value is 9. The select signal is necessary because the MOVE is also used to transfer data from a port device over the Y bus.

There are two other types of data transfers between the FPA and CPU over the Y bus. In one case, the FPA's micro-PC may be loaded by the CPU. In the second case, the FPA's current micro-PC may be read by the CPU.

When the FPA's micro-PC is loaded, it causes the FPA to abort its current operation and trap to the microaddress sent by the CPU. The trap in initiated by a MISC (function 2 field = 3) which asserts TRAP ACC to the FPA. The trap feature is used to abort the FPA during execution of memory management microcode, and to invoke microdiagnostic routines in the FPA.

Reading the FPA's micro-PC takes two consecutive microinstructions by the CPU. A MISC (function 2 field = 4) asserts READ μPC to the FPA. This is followed by a MOVE, again with the MDP field equal to 5 as when reading result data. When the FPA receives READ PORT, it asserts ACC SYNC and gates the current micro-PC onto the Y bus. The CPU then takes the data. The MOVE is not made to loop on ACC SYNC in this case. The FPA will always respond with the micro-PC data immediately following the MISC, and when READ PORT is first received.

Timing for port device transfers over the Y bus is shown in Figure 6-26. Both data and commands are transferred to a port device by a MISC with its port control bit CSR ⟨18⟩ asserted. CSR ⟨18⟩ = 1 redefines the MISC as a PORT microinstruction, which in turn redefines the rest of the control bits in the microinstruction. That is, eight of the bits defined as part of the function fields for the MISC are redefined as a command byte for the PORT. These are CSR ⟨17:10⟩, which connect directly to the port device over the FPA/port bus.

When the PORT microinstruction is executed, PORT INSTR is asserted by the CPU causing the port device to load the command byte. Also, the PORT microinstruction (like the MISC) transmits the contents of a 2901A working register onto the Y bus. This may be device write data, data to be loaded in a control status register, etc; the type of data is defined by the command byte.

A MOVE (MDP = 5) is used to transfer data from a port device, just as when collecting results from the FPA. READ PORT is asserted by the CPU, but there is no corresponding synchronizing signal generated by the port device. The port device always sends data when it first receives READ PORT. The data read by the CPU (device read data, status register contents, etc.) is specified by the command byte of a previous PORT microinstruction. The CPU MOVE instruction may follow the PORT microinstruction when the data is from a register or is a byte. When the data read is a longword, the CPU must delay at least one CPU cycle to allow the port device enough time to assemble the longword.
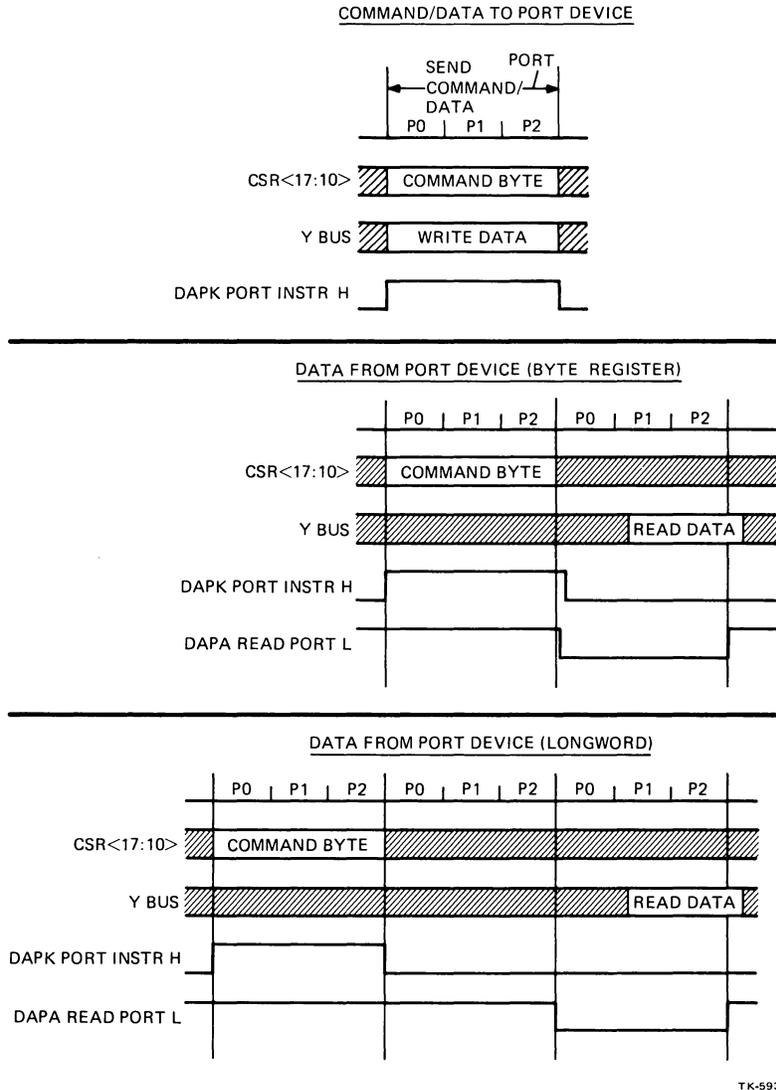
Figure 6-26   CPU/Port Device Transfers Timing Diagram

During fast interrupt operations by a port device (Paragraph 7.6), a PORT microinstruction does not have to precede every MOVE. The CPU can send the PORT microinstruction for AUTOMODE reads. Then successive MOVEs can read the port device's data buffer.

NOTE
When executing the MOVE or PORT micro-instructions to transfer port device data, the FPA must previously have been deselected by a MISC (function 1 field = 9). (Deselecting the FPA is equivalent to selecting a port device.) A port device also has an address assignment allowing more than one device to be connected to the Y bus. The device's address is specified in the high-order bits of the command byte.

CHAPTER 7
INTERRUPT PROCESSING HARDWARE

7.1 INTRODUCTION
The interrupt processing hardware in the CPU consists of an interrupt request register, a priority encoder circuit, and interrupt control logic as shown in Figure 7-1. The circuitry flags and identifies hardware generated interrupts and one type of exception (a trace fault), and it can cause an automatic dispatch to interrupt handling microcode in native mode. There is no dedicated CPU hardware for detecting software generated interrupts.

The interrupt processing hardware flags a processor interrupt condition by setting flip-flop INTERR REQ. As described in Paragraph 5.4, INTERR REQ causes the automatic dispatch to interrupt handling microcode in native mode by selecting the SPEC ROM in the instruction processing hardware during the next class decode; that is, following the execution of the current instruction and before execution of the next. (The automatic dispatch may be disabled by a mask bit, as discussed in Paragraph 7.2.5.)

INTERR REQ is also a microsequencer skip or jump condition allowing a dispatch to the interrupt handling microcode at defined points during the execution of long instructions. It also provides the means for flagging interrupts when the machine is in compatibility mode.

7.2 INTERRUPT DETECTION AND IDENTIFICATION
There are two classes of hardware generated interrupt requests, those which are assigned an interrupt priority level (IPL) and are serviced only when the processor's current IPL is at a lower value, and those which have no assigned IPL and are serviced on an immediate basis, regardless of the processor's current level of processing. Not included in either class are the processor interrupt requests generated (when T or TP is set in the PSL) by the trace fault control bits.

These trace bits have no assigned IPL, but service is still dependent on the current level of processing. As a matter of fact, servicing of the trace bits is dependent on both the processor's IPL and the IPL of any other interrupt requests waiting for service. The trace bits are serviced only when there are no other interrupts having an IPL higher than the processor's. The various interrupt requests and their assigned IPL (if any) are given in Table 7-1.

The 8085A console processor may interrupt the processor in four different ways. An AC low (power fail) condition, an interval timer time-out, and a console attention all have assigned IPLs. A console halt does not. It is one of the two interrupt requests that are serviced on an immediate basis, and are not compared to the processor's IPL.

UNIBUS devices interrupt the processor by means of the four UNIBUS bus request lines, BR7 through BR4. Each BR level has an assigned IPL with BR4 (which has the lowest priority) having the same IPL as a console attention. A port device, although it does not transfer read/write data over the UNIBUS, also uses a BR line to interrupt the CPU.
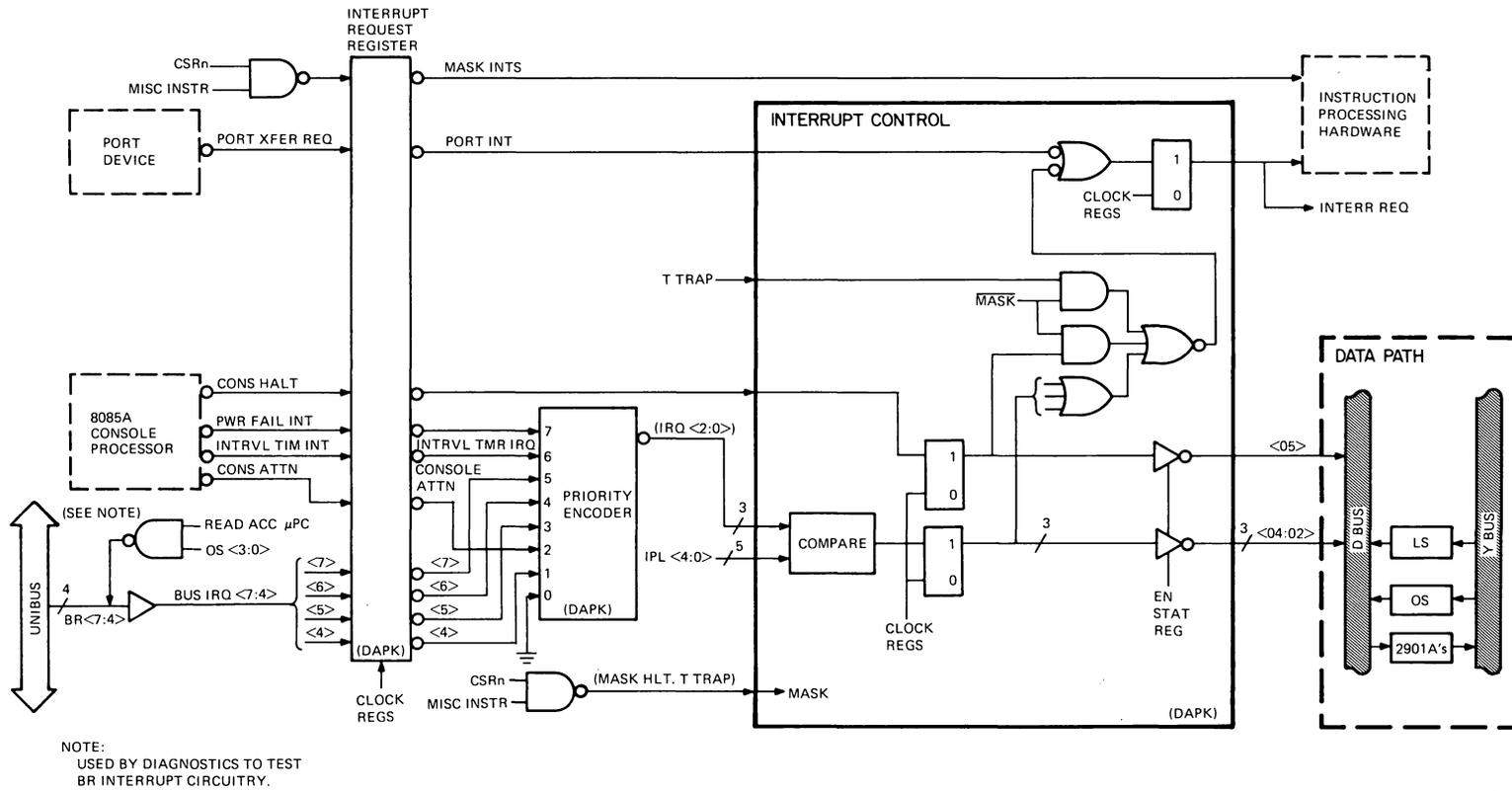
Figure 7-1   Interrupt Processing Hardware Block Diagram

Table 7-1 Interrupt Conditions

Interrupt Request	IPL	Interrupt Identifier	Port Interrupt Flag	Request Priority*
Port (fast) interrupt	–	- - - -	1	1
Console halt	–	0 - - -	–	2
Ac low	1E	- 0 0 0	–	3
Correctable memory error†	1A	- - - -	–	N/A
Interval timer time-out	18	- 0 0 1	–	4
BR7 (UNIBUS)	17	- 0 1 0	–	5
BR6 (UNIBUS)	16	- 0 1 1	–	6
BR5 (UNIBUS and/or port device)	15	- 1 0 0	–	7
Console attention (CTY or TU58)	14	- 1 0 1	–	8
BR4 (UNIBUS)	14	- 1 1 0	–	9
Trace bits (T or TP)	–	- 1 1 1	–	N/A

*Priority is determined by both hardware and microcode. A priority of one is the highest. A priority of nine is the lowest.

†Microcode initiated interrupt. Does not assert INTERR REQ in interrupt processing hardware. Serviced like a software interrupt during REI instruction.

In addition to asserting a BR line, a port device may generate a higher priority (fast) interrupt when read/write data is being transferred over the Y bus to and from the CPU. The fast interrupt, like a console halt condition, has no IPL and is serviced on an immediate basis.

A corrected (1-bit) memory read error (CRD) condition may interrupt the processor. This interrupt condition, which has an IPL of 1A, does not set INTERR REQ to interrupt the processor as the hardware generated interrupts (or the trace bits) do. Instead, the interrupt is initiated by the CPU's interrupt handling microcode. Also, this interrupt is serviced like a software interrupt and thus has no set priority in relation to the other interrupt conditions that set INTERR REQ.

Although the trace bits set INTERR REQ, they also have no set interrupt request priority. Because the trace bits are serviced only when there are no interrupt requests having an IPL higher than the processor's, trace bit priority at any one time is determined by the processor's current IPL, as compared to the IPL of any interrupt requests waiting for service.

The highest IPL value (1F) is not assigned to any interrupt request. When the processor is operating at this level, all interrupt requests with assigned IPLs are effectively disabled, preventing the UNIBUS devices or the console (except when requesting a halt) from interrupting the processor. The processor's IPL is raised to 1F, disabling further interrupt activity following the detection of serious system failures such as power fail, machine check exception, etc.

The servicing of hardware-generated interrupt conditions and trace bits by the CPU's interrupt handling microcode is shown in Figure 7-2. Port (fast) interrupts and console halts are serviced first, entirely by the microcode itself. The microcode causes a dispatch to an instruction level service routine for the other types of interrupts. Details of CPU microcode operation responding to various interrupt conditions are given in Paragraphs 7.3 through 7.6.

The system control block (SCB) in main memory supplies the dispatch addresses for the instruction level interrupt service routines. To access the SCB, the CPU microcode adds a 10-bit vector quantity to the contents of the system control block base register (SCBB), which is located in local store location 87. The high-order portion of the SCBB contains the base address of the two-page SCB.

The 10-bit vector quantity added to the SCBB supplies the low-order address bits (an offset) to specify a location within the SCB.

The offset added to the SCBB, and thus the SCB address generated, depends on the type of interrupt request being serviced. If CNSL/ATTN interrupts the processor, indicating a read/write data transfer request by a terminal or a TU58 tape, the console program supplies additional interrupt status information, and the microcode generates an offset for the highest priority transfer request enabled.

Also, if a UNIBUS device causes a processor interrupt, its device vector supplies the offset's nine low-order bits. (This is because more than one UNIBUS device may assert a BR line.) Because the locations corresponding to UNIBUS devices are in the second page of the SCB, the microcode first adds 200 to the device vector before adding the resulting 10-bit offset to the SCBB.

Once the address of a SCB location has been generated, the CPU microcode makes a memory reference to read its contents. The location that is read contains the virtual address of the appropriate interrupt service routine. In addition, because the address is longword aligned, the two low-order bits of the SCB location are used as control bits to specify how the interrupt is to be handled. Currently, interrupts are only handled on a stack (either the kernel or interrupt stack); the microcode pushes the PSL and the PC on the stack, writes a new PSL and PC (using the dispatch address in the SCB location), and dispatches to the interrupt service routine.

The IPL written into the new PSL indicates the new level of processing; that is, the IPL of the interrupt being serviced. (The current level of processing is maintained if a trace fault request is being serviced.) At the end of an interrupt service routine, the stack is popped by an REI instruction, and the PSL and PC are reloaded to return the processor to the level of processing prior to the interrupt.

To determine the type of interrupt request interrupting the processor, the CPU microcode can read a 4-bit interrupt identifier code generated by the interrupt processing hardware. (The identifiers for the various interrupt requests are listed in Table 7-1.)

Note that there is no identifier for a port (fast) interrupt. This is because the port interrupt request signal is a microsequencer skip condition and may be tested directly by the microcode (SCTL = 1A). Also, there is no identifier for a corrected memory error condition, as this interrupt is generated by the microcode and not the interrupt processing hardware. The 4-bit identifier, part of the STAT register, is read with a BASIC or MOVE (discrete register address = 7E).

INTERRUPT DURING
CLASS DECODE

PROC
INTR

INTERRUPT DURING
INSTRUCTION EXECUTION

PROC
INTR

END
REI

BIT 31 = 1
IN LS LOC 88.

FAST
INTERR —— YES

MASK HLT. T OR TP

TRANSFER PORT
READ/WRITE DATA

YES   CRD
INTERR

NO

OTHER
INTERR
REQ

IPL≥1A

YES

YES   NO

CLASS DECODE

YES

NO

CONSOLE
HALT —— YES

NO

A

NO   B

HALT
INSTR,
ETC.

TELL CONSOLE INSTRUC-
TION PROCESSING HAS
HALTED. ENTER HALT
LOOP.WAIT FOR CON-
SOLE COMMANDS.

PROG
HALT

RETURN TO PREVIOUS
LEVEL OF MICRO
PROCESSING (CLASS
DECODE OR INSTRUC-
TION EXECUTION).

INTERR
REQ >PROC
IPL —— YES

NO

AC LOW V INT TIM

BR LEVEL

CONSOLE ATTN

GET INTERRUPT
PRIORITY REGISTER
FROM CONSOLE.
GENERATE OFFSET
FOR HIGHEST PRIO-
RITY DATA TRANS-
FER REQUEST (TU58
RCV, TU58 XMIT, TTY
RCV, TTY XMIT.)
ADD TO SCBB.

TP —— YES

NO

YES   BR5

TRACE FAULT

YES   PORT
DEV   NO   NO

0 → TP

B

GET DEVICE VECTOR

T —— YES

NO

T → TP

GENERATE OFFSET
INTO SCB. (OFFSET
DEPENDS UPON TYPE
OF INTERRUPT REQ.)
ADD TO SCBB.

GENERATE OFFSET
INTO SCB. (DEVICE
VECTOR + 200.) ADD
TO SCBB.

CLASS DECODE (MASK
INTERR). BEGIN
INSTRUCTION
EXECUTION (FPD = 0).
CONTINUE INSTRUC-
TION EXECUTION
(FPD = 1).

NOTE:
PUSH TP=0 AND BACKUP-PC
IF INTERRUPT DURING LONG
INSTRUCTION.

READ LOCATION IN
SCB. USE CONTENTS
TO DISPATCH TO
APPROPRIATE
INSTRUCTION LEVEL
SERVICE ROUTINE.
(PUSH PSL & PC.
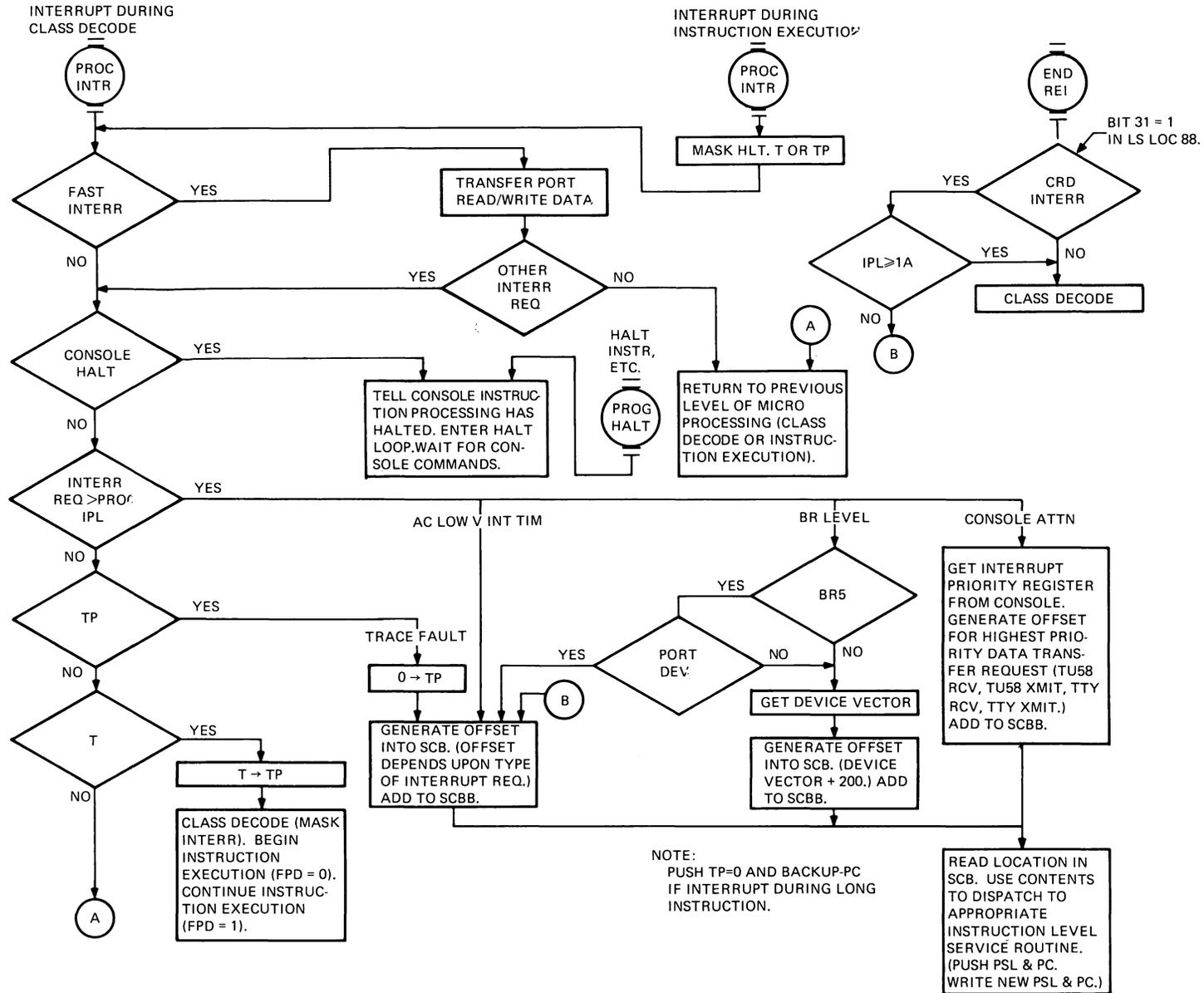WRITE NEW PSL & PC.)

A

TK-6644

Figure 7-2   Servicing of Hardware-Generated Interrupts and Trace
Bits by CPU Microcode

7-5

7.2.1 Interrupt Request Register

Except for the trace control bits, all of the conditions that cause a processor interrupt by setting IN-TERR REQ are stored in the interrupt processing hardware's interrupt request register. (The trace bits are stored in the PSL.)

A stored interrupt request remains in the register until the interrupting device itself negates the request. For example, during the servicing of a UNIBUS request, the CPU microcode makes a memory reference which causes the MCT to first assert the UNIBUS bus grant signal corresponding to the bus request that is being serviced, and then read the device's interrupt vector and transfer it to the CPU. During this operation, the bus grant line that is asserted causes the interrupting device to negate its bus request, removing the interrupt condition. This clears the UNIBUS request in the interrupt request register unless another UNIBUS device is asserting the same bus request signal.

7.2.2 Priority Encoder Circuit

The interrupt requests stored in the interrupt request register that have assigned IPLs connect to a priority encoder circuit. When more than one of these requests are asserted at any one time, the encoder circuit resolves interrupt request priority (i.e., which request is to be serviced first) by generating a 3-bit output code that specifies the highest priority request at its inputs. The request with the highest priority is the request with the highest IPL.

For example, the AC low condition, which has the highest assigned IPL and which connects to the highest priority input (7), generates an encoder output of 7 regardless of what other requests are asserted. Correspondingly, an interval timer time-out which has the next lowest assigned IPL and which connects to next lowest priority input (6), generates an output code of 6. Of course, this only occurs if there is not an AC low condition present. Output codes 5 to 1 are generated in a similar fashion to determine priority for the other requests with assigned IPLs. An output code of 0 indicates that no interrupt requests are asserted at the encoder inputs.

7.2.3 Interrupt Control

Whereas the priority encoder circuit resolves priority for interrupt requests with assigned IPLs, additional control logic is required to compare the encoder's output with the processor's IPL before a processor interrupt may be generated. The interrupt control logic also samples the other types of interrupt requests and unconditionally generates a processor interrupt for each unless the request is masked. (Interrupt requests due to a console halt or the trace bits may be inhibited by a mask bit, as discussed in Paragraph 7.2.5.)

In addition to generating the processor interrupt by setting the INTERR REQ flip-flop, the interrupt control also generates the 4-bit interrupt identifier that is read by the microcode. The identifier is held in four flip-flops that may be read onto the D bus when the STAT register is read (EN STAT = 1).

The INTERR REQ flip-flop is set in two different ways. In one case, if there is a port (fast) interrupt, or if a trace bit is set (and not masked), INTERR REQ is enabled directly and set at the end of the current microcycle. In the other case, if there is a console halt condition (not masked) or a request with an assigned IPL higher than the processor's, a nonzero identifier code is first loaded into the four interrupt identifier flip-flops at the end of the current microcycle. Then, with at least one identifier flip-flop set, INTERR REQ is set at the end of the following microcycle.

It can be seen that more than one interrupt condition can cause INTERR REQ to be set. Also, IN-TERR REQ remains set until all conditions causing a processor interrupt have been serviced, and the corresponding interrupt request signals negated.

The interrupt control compares a request's IPL with the processor's IPL by comparing the output of the priority encoder, which indicates the highest priority request at its inputs and thus (implicitly) the request's IPL, with the IPL held in the PSL IPL $\langle 4:0 \rangle$. When the request's IPL is higher and a processor interrupt is to be generated, the priority encoder output is loaded into the three low-order identifier flip-flops to first identify the interrupt condition and then set INTERR REQ. A console halt condition unconditionally sets the high-order identifier flip-flop to set INTERR REQ (if not masked).

NOTE
The identifier code as read by the microcode and given in Table 7-1 is the complement of the code held in the identifier flip-flops. That is, the negation of the flip-flop outputs are enabled onto the D bus when the STAT register is read.

7.2.4 Interrupt Priority

Interrupt priority for the conditions asserting INTERR REQ is determined by both hardware and microcode. A port (fast) interrupt, a console halt, and the trace bits all set INTERR REQ, regardless of the processor's IPL, so then the priority as determined by hardware is the same.

However (with reference to Figure 7-2), the CPU microcode services a fast interrupt before a console halt, and it services the trace bits (T or TP) only when there are no other requests to be serviced; that is, after a fast interrupt and a console halt, and only when there are no requests with assigned IPLs higher than the processor's IPL. This is indicated when the three low-order bits of the identifier (as read by the microcode) are all ones.

Although a request with an assigned IPL higher than the processor's is serviced by the microcode before the trace bits, it is serviced only after a fast interrupt or a console halt. Also, when more than one request with an assigned IPL is asserted, the hardware (the priority encoder circuit) determines the order in which they are serviced as discussed in Paragraph 7.2.2. Only one request with an assigned IPL may interrupt the processor at one time. The priority circuit asserts INTERR REQ, and selects and generates an output code that identifies the request with the highest IPL.

7.2.5 Interrupt Mask Functions

There are two interrupt mask functions. One is to prevent INTERR REQ from causing the automatic dispatch to the interrupt handling microcode between instructions in native mode. The other is to prevent either a console halt condition or the trace bits from setting INTERR REQ in the first place. Both are invoked by the MISC microinstruction.

Except during long instructions when INTERR REQ may be tested directly by the CPU microcode, the automatic dispatch (during the class decode) is the only dispatch to the interrupt handling microcode in native mode. As a result, the first of the two mask functions provides a convenient means in native mode to disable interrupt activity and allow processing to continue at the current level.

Masking of the automatic dispatch is accomplished by executing a MISC (with its function 2 field equal to 1) immediately before the DECODE microinstruction that performs the class decode. This sets a flip-flop in the interrupt request register (MASK INTS) that prevents selection of the SPEC ROM in the instruction processing hardware, should an interrupt occur. The flip-flop is set for one microcycle only. As a result, a MISC must be executed before every class decode for which interrupts are to be disabled.

The second of the two interrupt mask functions makes it possible to mask console halt and the trace bits during interrupt polling; that is, when the INTERR REQ bit is being tested by the microcode. The major reason this mask function is provided is because it is desirable to service interrupts during the execution of long instructions, but only if the console is not halting the machine and there is no request for a trace fault. Otherwise, when a console halt condition causes instruction level processing to stop, the execution of the long instruction would not complete.

A trace fault should never occur during the execution of an instruction. Trace faults are for tracing the sequence of program execution, and should interrupt the processor between instructions only.

The masking of a console halt and the trace bits is accomplished by executing a MISC (with its function 2 field equal to 6) immediately before the microinstruction which tests for interrupts. The MISC asserts a signal that prevents the console halt or the trace bits from enabling the INTERR REQ flip-flop. If there are no other interrupt requests waiting for service, INTERR REQ will then be clear when the bit is tested during the next microcycle.

7.3 UNIBUS INTERRUPTS
A UNIBUS device interrupts the processor by asserting its assigned BR line (one of BR7 through BR4). More than one BR level can be asserted at any one time by the various UNIBUS devices, and more than one device can assert the same BR level. For example, more than one terminal may be causing the asynchronous line controller on the UNIBUS to assert BR4.

Although the MCT arbitrates the transfer of data over the UNIBUS, the CPU determines BR signal priority as explained in Paragraph 7.1.4. (The BR signals have assigned IPLs.) BR7 has the highest priority; BR4 has the lowest.

The servicing of UNIBUS interrupt requests is as shown in Figure 7-3. The interrupt handling micro-code first makes a memory reference to the MCT in order to collect a device interrupt vector. In response to the MEM REQ microinstruction by the CPU, the MCT asserts the BG line on the UNIBUS (one of BG7 through BR4) that corresponds to the bus request being serviced. (The BG line to be asserted is specified by the CPU.)

When received by the interrupting device, the BG level indicates that the device may become bus master. The device then acknowledges selection as bus master (asserts SACK), negates its bus request, and assumes control of the UNIBUS (asserts BBSY). Once it is bus master, the device transmits its 9-bit device vector on the UNIBUS data lines and asserts a data strobe (INTR). It then relinquishes control of the UNIBUS when the MCT indicates it has stored the vector (MCT asserts SSYNC).

After the MCT has stored the device vector, a MOVE microinstruction executed by the interrupt handling microcode (to complete the memory reference) causes the vector to be transferred over the MC bus and collected by the CPU. The CPU clock will stall if the MOVE occurs before the MCT has received the vector from the device.

When the microcode has stored the device vector, it generates an offset, adds it to the SCBB, and reads an SCB location (Paragraph 7.2.1). Reading an SCB location, which is in main memory, requires another memory reference to the MCT. Following the second reference, the microcode uses the contents to dispatch to the interrupt service routine for the interrupting device.
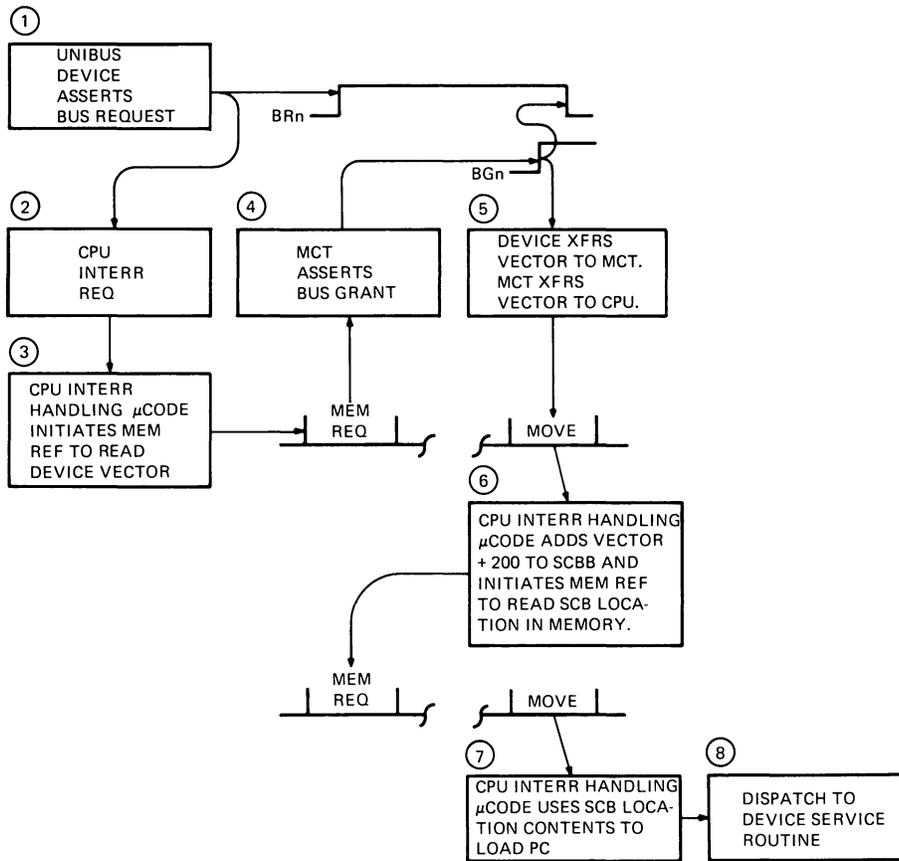
Figure 7-3   UNIBUS Interrupt Request Handling

7.4 CONSOLE INTERRUPTS

The console program running in the 8085A console processor can interrupt the processor in four different ways.

1.   PWR FAIL INT signals loss of AC power.

2.   INTRVL TIM INT signals an interval timer time-out condition.

3.   CONS ATTN signals that a data transfer to/from the console terminal (local or remote) or the TU58 tape device may be initiated.

4.   CONS HALT indicates that the program running in the CPU is to halt.

The first three interrupt requests have assigned IPLs. As a result, the interrupt handling microcode services each of these requests by causing a dispatch to the appropriate instruction level interrupt service routine as described in Paragraph 7.3.

The generation of the interrupt request due to a power fail condition (IPL = 1E) is discussed in Paragraph 2.4.5. The interval timer interrupt request (IPL = 18) is discussed in Paragraph 2.4.3. The assertion of CONS ATTN (IPL = 14) for a tape or terminal data transfer and the subsequent transfer of the console program's interrupt priority register contents are discussed in Paragraph 2.3.3.

The four interrupt priority register bits indicate to the microcode the type of data transfer request (or requests) causing the assertion of CONS ATTN. The types of requests listed in the order of data transfer priority (highest to lowest) are:

1. Received character (tape)
2. Ready for transmit character (tape)
3. Received character (terminal)
4. Ready for transmit character (terminal).

The microcode causes a dispatch to the interrupt service routine for the highest priority data transfer request when more than one request is causing the assertion of CONS ATTN.

The last console-generated interrupt request, CONS HALT, causes no dispatch to an interrupt service routine. The console asserts CONS HALT when a CNTL-P is received from the console terminal, indicating the machine is to stop instruction level processing; that is, the machine is to leave the program mode of operation and enter the console mode of operation.

When the console halt interrupt request is serviced, the interrupt handling microcode first sends a halt code to the console processor. This causes the console program to negate CONS HALT, turn off the RUN indicator on the front panel, and enter console mode. The microcode then sends a halt address (the CPU's PC) to the console processor (this address and the halt condition are typed at the console terminal) and enters a wait loop.

The microcode remains in the wait loop until it receives a command packet from the console program telling it to resume instruction level processing or to perform some other console command. (Communications between the CPU microcode and the console program are discussed in Paragraph 2.6.)

NOTE
The microcode also sends a halt code and halt address to the console processor when a HALT instruction is executed by the CPU (in kernel mode) or for certain error conditions such as an invalid SCB vector, CHMX from/to the interrupt stack, etc. Refer to the *CPU Microcode Listing* or *VAX-11/730 Diagnostic System Overview Manual* to identify the conditions that halt instruction level processing.

7.5 TRACING
A trace fault is an exception that occurs between instructions when tracing is enabled. Tracing is enabled by software during program debug and performance evaluation operations. It is designed so that one and only one trace fault occurs after the execution of one traced instruction and before the execution of the next instruction.

Tracing is controlled by two bits in the PSL. These are the T (trace enable) bit and the TP (trace pending) bit. If either the T or TP bit is set, it causes a processor interrupt unless the interrupt is masked as discussed in Paragraph 7.2.5.

To begin the tracing of a program, the system's debugging software sets the T bit (not the TP bit) in the PSL. Then, at the next class decode (as shown in the upper half of Figure 7-4), a processor interrupt is generated and the interrupt handling microcode sets the TP bit in the PSL, provided there are no other interrupt conditions. (Refer to the flow diagram of microcode operation in Figure 7-2.)
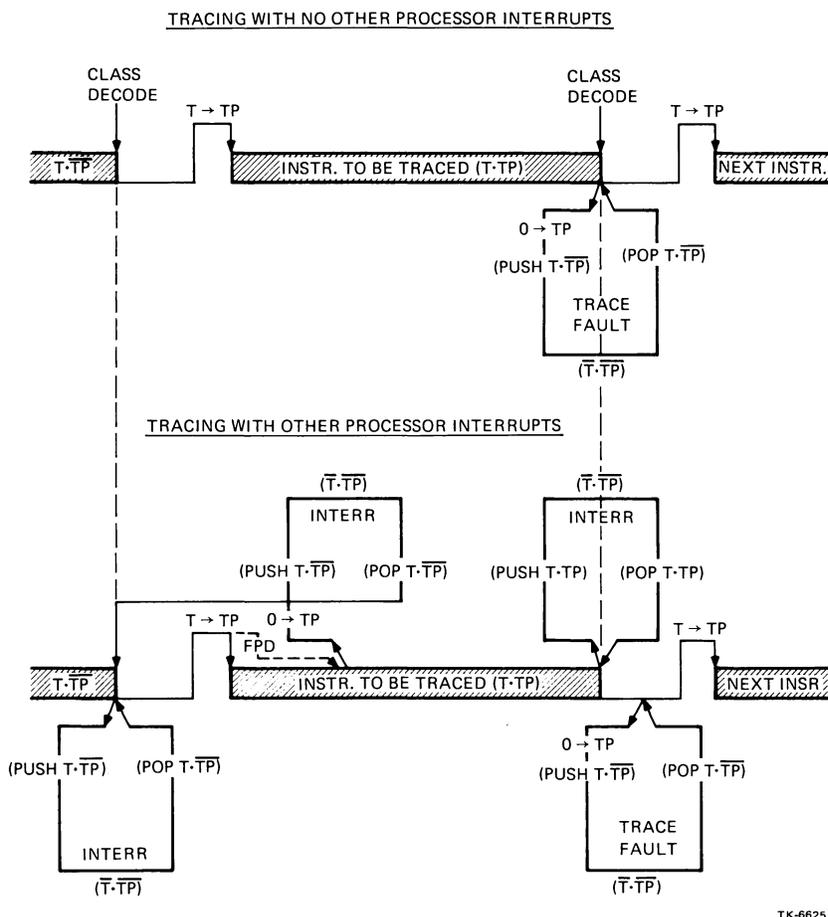
Figure 7-4   Trace Operations

After setting TP, the microcode masks all interrupts and dispatches back to the class decode, which starts the processing of the execution code for the instruction to be traced. All interrupts are masked in native mode by the disabling of the automatic dispatch function. In compatibility mode, the INTERR REQ flag is not polled.

7-11

Following execution of the instruction to be traced, T and (in this case) TP cause another processor interrupt during the class decode. If there are no other interrupts, TP equal to one then causes a dispatch to the appropriate instruction level service routine which (for a trace fault) is the debugging software.

When the microcode pushes the PSL and PC on the stack prior to the dispatch, TP = 0 is pushed so that another trace fault will not be generated following the return to normal processing. (The return, to the class decode of the next instruction, is made with an REI instruction which restores the previous PSL and PC.) Also, T and TP are made equal to zero in the PSL that is created prior to dispatch, so that no tracing occurs during the trace fault routine itself.

Operation when there are other processor interrupt conditions in addition to the trace bits is shown in the lower half of Figure 7-4. As indicated, all other interrupts are serviced first. (Refer again to the flow diagram in Figure 7-2.) That is, following an interrupt during a class decode, all other interrupts are serviced before TP is set at the start (or continuation) of instruction execution, and before the trace fault is generated at the end of instruction execution. Because the microcode pushes the current value of the TP bit when servicing the other types of processor interrupts between instructions, one and only one trace fault is generated for each instruction that is traced.

When interrupts are serviced during the execution of instructions being traced, the current value of TP (which is a one) is not pushed. Instead, TP = 0 is pushed because the return at the termination of the interrupt routine is to the class decode for the same instruction. (The PC is backed up before it is pushed with TP = 0.) If TP = 1 were pushed, it would cause two trace faults for the preceding instruction in the executing program.

As for trace faults, the new PSLs that are created when interrupts occur have both T and TP equal to zero, so that tracing does not occur during the interrupt service routine. Because the servicing of exceptions by the microcode during tracing is similar to that for interrupts, the majority of interrupts and exceptions that occur are totally transparent to the executing program.

7.6 PORT (FAST) INTERRUPTS

Fast interrupts, which are requested by a port device, are serviced entirely by the microcode before any other interrupt request. To test for a fast interrupt following a processor interrupt, the interrupt handling microcode first tests the port interrupt request flag (PORT INT). If it is not set, the microcode reads the interrupt identifier and services the highest priority interrupt condition generating the processor interrupt request. If the port interrupt flag is set, it means the port device is requesting write data from the CPU or it has read data for transfer to the CPU. The microcode then transfers the data over the Y bus, as described in Paragraph 6.12.

Following the transfer of the port read/write data, the interrupt handling microcode executes a MISC microinstruction (function 2 field equals 5) which transmits XFER GRANT to the port device. XFER GRANT clears the port interrupt request in the port device, removing the fast interrupt condition.

With the fast interrupt serviced, the microcode tests the INTERR REQ flag to see if there are any other interrupt conditions. If not, it returns to the previous level of microprocessing. If there are still other interrupts, the microcode reads the identifier and services a second interrupt condition.

7.7 PORT (SLOW) INTERRUPTS

In addition to the fast interrupt capability, a port device may interrupt the processor by asserting a BR line. (BR5 is currently used.) The BR line is normally asserted to signal the end of a read/write operation, caused either by normal termination when the correct number of longwords have been transferred, or by an error condition.

When a BR5 level is to be serviced, the interrupt handling microcode first reads port device status. (The slow interrupt condition asserts a status register bit in addition to BR5.) If the port device is interrupting, the microcode generates an offset, adds it to the SCBB, reads an SCB location, and dispatches to the appropriate interrupt service routine, as with other interrupt requests having an assigned IPL. If the port device is not interrupting, the UNIBUS device asserting BR5 is serviced (Paragraph 7.3).

7.8 CORRECTED MEMORY ERROR INTERRUPTS

An interrupt due to a corrected memory error is generated mainly to facilitate the instruction level reporting of the error condition. It differs from other hardware interrupts in that it is initiated by the microcode, not the INTERR REQ flip-flop in the interrupt processing hardware. Also, it is serviced like a software interrupt during the execution of an REI instruction.

When ERR SUM is asserted by the memory controller indicating some type of memory error, the CPU microcode normally reads and stores memory status to (first) determine the type of error, and (second) to supply other data useful for fault diagnosis (physical memory address, syndrome bits, etc.). If the error flagged is a corrected read error and error reporting is enabled, the microcode sets bit 31 in local store location 88. (The low-order half of this location holds the software interrupt summary register.)

Then, near the end of the next REI's execution with bit 31 in the local store location set, the microcode generates an offset and dispatches to the appropriate interrupt service routine, provided the IPL in the PSL is less than the IPL of the corrected memory error interrupt condition (1A). The interrupt service routine stores the error data previously read by the microcode on a disk so that it may be retrieved by error reporting software.
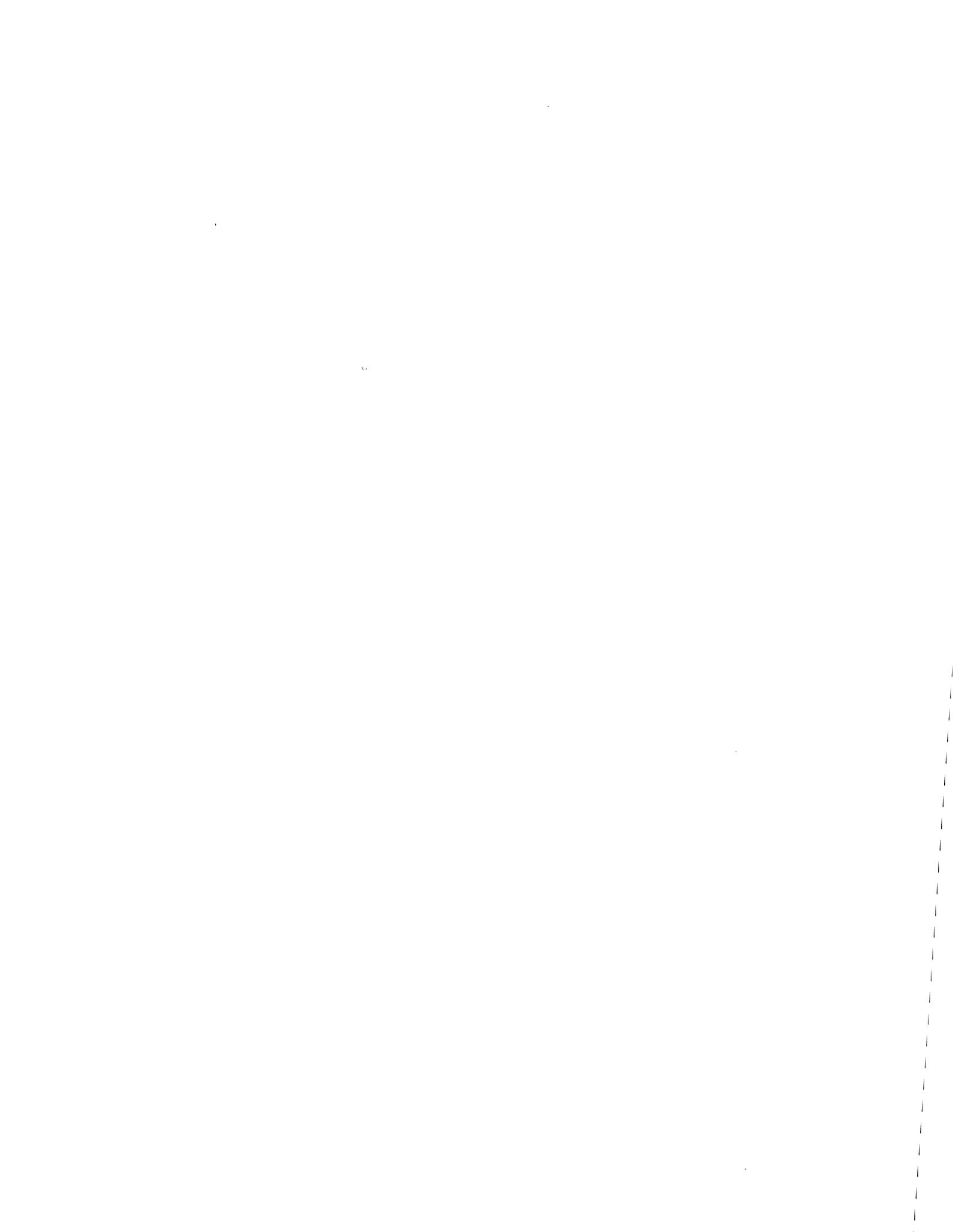
APPENDIX A
PROGRAMMED ARRAY LOGIC DEVICES (PALS)

A.1 INTRODUCTION

Programmed array logic devices (PALs) are logic arrays incorporating fusable-link technology, and which are manufactured on a chip using the TTL Schottky bipolar process used to make fusable-link PROMs. Like PROMs, PALS may be programmed to give a semi-custom designed chip unique to a specific requirement.

The basic logic configuration used in PALs is shown in Figure A-1. The circuitry consists of a programmable AND array connecting to a fixed OR array. The AND array shown in the basic logic configuration provides only four programmable (fusable-link) inputs for each of two fixed OR inputs. In the PAL circuits used in the VAX-11/730, up to 32 programmable AND inputs and up to 8 fixed OR inputs are used per output.
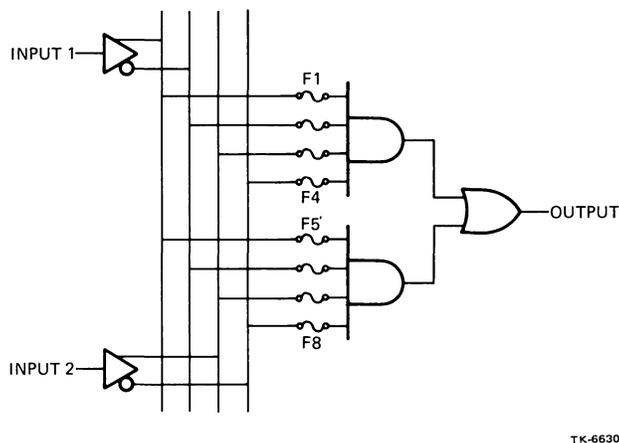

TK-6630

Figure A-1   Basic PAL Logic Configuration

An unprogrammed PAL has all fuses intact (Figure A-1). A PAL is programmed by first determining the AND inputs to be used, and then "blowing" the links for the unused AND inputs to give the desired AND before OR logic configuration. (A standard PROM programming device is used for this operation.) For example, the upper half of Figure A-2 shows the links blown to implement the XOR function AB V AB in the basic PAL logic configuration.
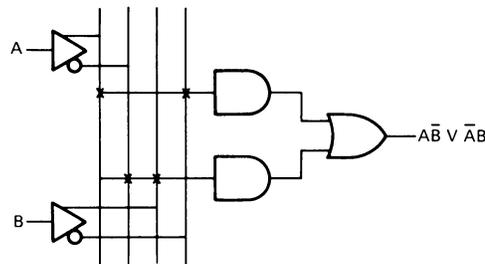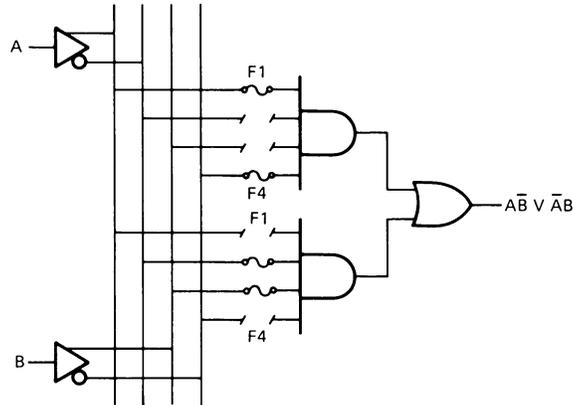
Figure A-2   XOR Logic Function Using PAL Logic

This same logic function may also be represented as shown in the lower half of the figure where an 'X' represents the links that are left intact to perform the logical AND. This last method of showing an AND array configuration is the convention used in the PAL plot listings provided in the VAX-11/730 microfiche set.

A.2   PAL DEVICE TYPES
The four types of PALs used in the VAX-11/730 are listed in Table A-1. Logic diagrams for each PAL are given at the end of this appendix.

With reference to the logical diagrams, it can be seen that the four PAL devices all use the basic AND before OR logic configuration discussed in Paragraph A.1. However, outputs from the 16L8 gate array chip are inverted and six of the eight outputs feed back to the AND arrays for added functionals.

In addition, the output inverters for these six outputs may be turned on and off by the AND arrays (programmable I/O). This provides added logic capability and (when the inverter is turned off) it also allows the corresponding output pin to be used as an input to the AND array, just like a designated input pin.

Table A-1 PAL Device Types Used in VAX-11/730

PAL Device Type	Inputs	Outputs	Prog. I/O	Register Outputs	Description
16L8	16	8	6	0	AND-OR gate array
16R8	16	8	0	8	AND-OR array with registers
16R6	16	8	2	6	AND-OR array with registers
16R4	16	8	4	4	AND-OR array with registers

Also note from the logic diagrams that the 16R8 chip has register outputs (D-type flip-flops) and no gate outputs. Again, outputs are fed back to the AND array, but not directly by way of the chip's output pins. Instead, the zero outputs of the flip-flops drive the array. As a result, the output pins, cannot be used as input pins, as for a 16L8. The other two PAL types, the 16R6 and the 16R4, have varying combinations of both gate and register outputs on the same chip.

A.3 PAL SYMBOLOGY
A typical PAL as represented in the VAX-11/730 Engineering Print Set is shown in Figure A-3. Information within the symbol includes the device type, part number, and chip location. For example, the PAL in the figure is type 16R4, and is located at E50 with part number 010K3. The PAL part number distinguishes one programmed PAL from another. Because PALs are programmed for specific applications, it is seldom that more than one PAL will have the same part number.
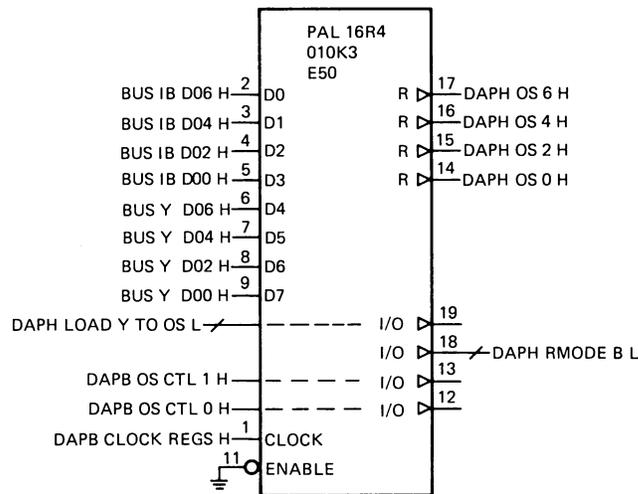
TK-6629

Figure A-3 Typical PAL Symbology

Inputs to the designated PAL input pins are shown at the left of the PAL symbol. Outputs appear at the right. When an output pin is used as an input pin (as discussed in Paragraph A.2), the input signal is entered at the left of the symbol, and a dotted line (drawn across the PAL symbol) is used to show the connection to the output pin on the right. Pins having both input and output capability are labeled as I/O on the PAL symbol. Gate outputs not having both input/output capability are labeled with an C (as for pins 12 and 19 of a 16L8). Register outputs are identified by an R. Finally, designated input pins are specified by a D.

A.4 READING THE PAL PLOT LISTING

An example of the PAL plot listing is shown in Figure A-4. The part number and PAL device type (a 16R6 in this case) are at the top of the listing. The input or output associated with each PAL pin is given next. (An NC indicates no connection; VCC indicates the +5 V power source.) A low assertion level for input/output signals on the listing is indicated by a slash ('/') immediately preceding the signal name. If there is no slash, the signal is asserted high. It should be remembered that input/output signal names on the listing are sometimes abbreviated and are not exactly the same as in the Engineering Print Set.

```
PART NUMBER: 23-004K4-0-0

DEVICE TYPE: PAL16R6

PIN NUMBER = SYMBOL TABLE:

1= CLOCK              8= SEL_9600_BAUD      15= STATE
2= ALE               9= RESET              16=/RAS
3= REQUEST_REFR      10= GROUND             17= REFRESH_DONE
4= IO                11= OUT_EN             18=/START_8085_CYC
5= A14               12=/UART_CHIP_SEL      19=/LONG_CYCLE
6= 9600_BAUD         13=/9600_300_BAUD      20= VCC
7= 300_BAUD          14= REFRESH_CYC


FUSE PLOT:           (X = FUSE INTACT , - = FUSE BLOWN)

OUTPIN 19   ---- ---- ---- ---- ---- ---- ---- ---- VCC
            ---- ---X ---- X--- ---- ---- ---- ---- START_8085_CYC*A14
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

OUTPIN 18   X--- ---- ---- ---- ---- ---- ---- ---- ALE
            ---- ---X ---- X--- ---- --X- ---- ---- REFRESH_CYC*START_8085_CYC*A14
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

OUTPIN 17   ---- -X-- ---- ---- ---- ---- ---- ---- /REQUEST_REFR
            ---- ---- ---X ---- ---- ---X ---- ---- /REFRESH_DONE*/REFRESH_CYC
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

OUTPIN 16   ---- ---- ---- --X- ---- --X- ---- ---- /RAS*REFRESH_CYC
            ---- ---- ---- ---X --X- ---- ---- ---- RAS*STATE
            ---- ---X -X-- X-X- ---- ---- ---- ---- START_8085_CYC*/RAS*/IO*A14
            ---- ---- ---- ---- ---- ---- X--- RESET
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
            XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

OUTPIN 15   ---- --X- ---- ---- ---- ---- ---- ---- /START_8085_CYC
```

Figure A-4   Sample PAL Plot Listing (Sheet 1 of 2)

```
               ---- ---- ---- ---X ---- ---- ---- ----  RAS
               ---- ---- ---- -X-- ---- ---- ---- ----  /A14
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

   OUTPIN 14   ---- ---X ---- ---- ---- ---- ---- ----  START_8085_CYC
               ---- ---- ---- ---X ---- --X- ---- ----  RAS*REFRESH_CYC
               ---- ---- ---- ---X --X- ---- ---- ----  RAS*STATE
               ---- -X-- ---- ---- ---- ---X ---- ----  /REFRESH_CYC*/REQUEST_REFR
               ---- ---- --X- ---- ---- ---X ---- ----  /REFRESH_CYC*REFRESH_DONE
               X--- ---- ---- --X- ---X ---X ---- ----  /REFRESH_CYC*/RAS*ALE*/STATE
               ---- ---- ---- ---- ---- ---- ---- X---  RESET
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

   OUTPIN 13   ---- ---- ---- ---- X--- ---- X--- ----  SEL_9600_BAUD*9600_BAUD
               ---- ---- ---- ---- X--- -X-- ---- ----  /SEL_9600_BAUD*300_BAUD
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

   OUTPIN 12   ---- ---- ---- ---- ---- ---- ---- ----  VCC
               ---- ---X X--- X-X- ---- ---- ---- ----  START_8085_CYC*IO*A14*/RAS
               ---- ---- ---- --X- --X- ---- ---- ----  /RAS*STATE
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
               XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
```

Figure A-4   Sample PAL Plot Listing (Sheet 2 of 2)

The rest of the listing consists of the AND array plots for each output pin. An 'X' represents the fusable links left intact; a dash ('–') represents a blown link. More importantly (in order to read the listing), to the right of each line in a plot is the list of signals selected by the intact links that make up the AND inputs. Because these individual AND terms are ORed by the PAL logic, the list of AND terms in the listing (ORed together) result in an easily read Boolean expression that represents the logic function performed. For example, output pin 12, which is a gate output (refer to Figure A-9) and the last plot in the listing, has the following input.

VCC
START_8085_CYC*IO*A14*/RAS
/RAS*STATE

The enable level for the gate output inverter (the top line) is connected to VCC, a logical 1. The dashes in the Boolean expression only represent a space (a blank character) in the signal name. An asterisk (*) between signal names specifies the logical AND operation. Discounting the enable level for the output inverter which in this case is always asserted, this input expression for output pin 12 (/UART_CHIP_SEL) may be read as follows.
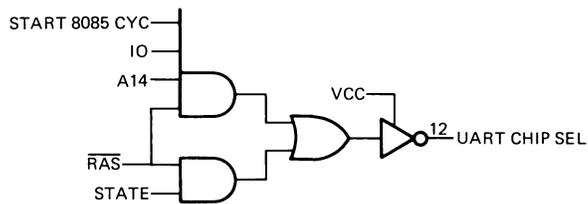
UART CHIP SEL L = START 8085 CYC H • IO H • A14 H • RAS H
**V**
RAS H • V STATE H

NOTE: • = AND
V = OR

The PAL circuitry for this output may be represented as shown in the upper half of Figure A-5. The same circuit using Engineering Print Set conventions for signal names is shown in the lower half of the figure.

A-5

PAL CIRCUIT FOR OUTPUT PIN 12

START 8085 CYC—
IO—
A14—

VCC—

RAS—
STATE—

12 UART CHIP SEL

THE SAME CIRCUIT USING PRINT SET CONVENTIONS

START 8085 CYC H—
IO H—
A14 H—

+3V H—

RAS L—
STATE H—

12 UART CHIP SEL L
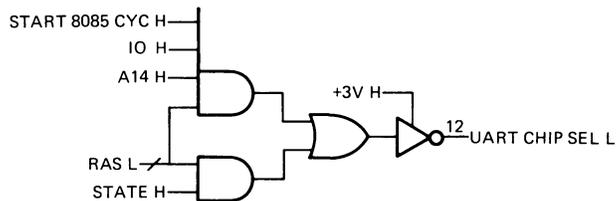
TK-6631

Figure A-5   PAL Circuit for Output Pin 12 on Sample Listing

For a register output, the Boolean expression read from the listing specifies the output signal just as for a gate output. Of course, the output pin is not asserted or negated until the register flip-flop is clocked. Flip-flops are clocked by the positive-going transition of the clock.

When the input statements given in the plot listing are read as AND terms ORed together as just described, they define the actual PAL circuit and the conditions to make the PAL output go low. (In the example given, the PAL output signal is also asserted when it is low.) When a PAL output signal is asserted at a high level, it is sometimes more convenient to think of the PAL's AND before OR circuit in terms of its equivalent OR before AND configuration. For example, the Boolean equation for register output pin 17 (REFRESH DONE) on the sample listing is as follows when read as AND before OR logic as done earlier in this section.

REFRESH DONE L = REQUEST REFR H
V
REFRESH DONE H • REFRESH CYC H

However, the listing may also be read as OR before AND logic, as shown in the following.

REFRESH DONE H = REQUEST REFR L

REFRESH DONE L V REFRESH CYC L

As can be seen, the second expression more clearly indicates that REFRESH DONE is set by the assertion of REQUEST REFR and REFRESH CYC, and that it remains set until REQUEST REFR is negated. The AND before OR circuit and the equivalent OR before AND circuit are diagramed in Figure A-6.

A-6

AND–OR PAL CONFIGURATION FOR OUTPUT PIN 17

REQUEST REFR H ⎯⎯⎯⎯

REFRESH CYC H ⎯⎯⎯⎯

VCC ⎯

17 (REFRESH DONE L)

REFRESH DONE H

CLOCK

EQUIVALENT OR–AND CONFIGURATION

REQUEST REFR L ⎯⎯⎯⎯

REFRESH CYC L ⎯⎯

VCC ⎯

17 REFRESH DONE H

CLOCK
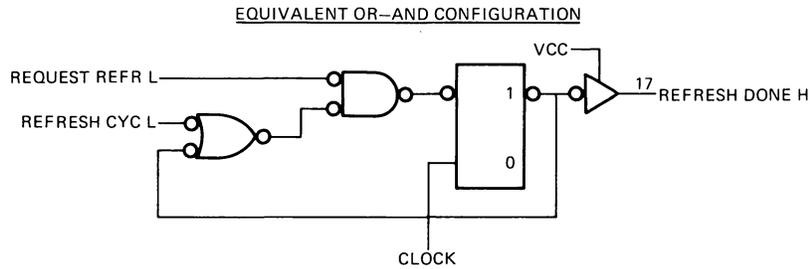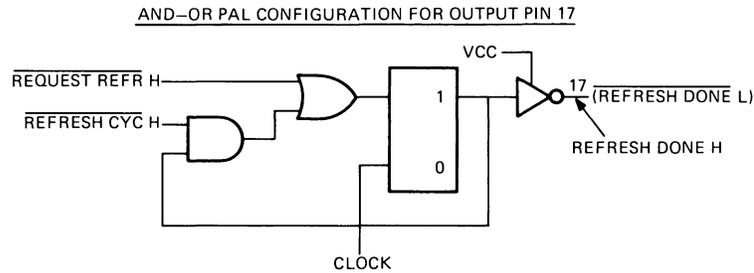
TK-6632

Figure A-6   PAL Circuit for Output Pin 17 on Sample Listing

To summarize:

1. When the plot listing is read as AND-OR, it specifies the input signal to give a low PAL output. The output line may or may not be asserted low.

2. If the PAL output line is asserted low, the AND-OR input expression is usually the best way to specify the output line.

3. If the PAL output line is asserted high, the equivalent OR-AND input expression is usually the best way to specify the output line.

A.5   PAL LOGIC DIAGRAMS
The logic diagrams for the 16L8, 16R4, 16R6, and 16R8 PAL devices are shown in Figures A-7 through A-10.
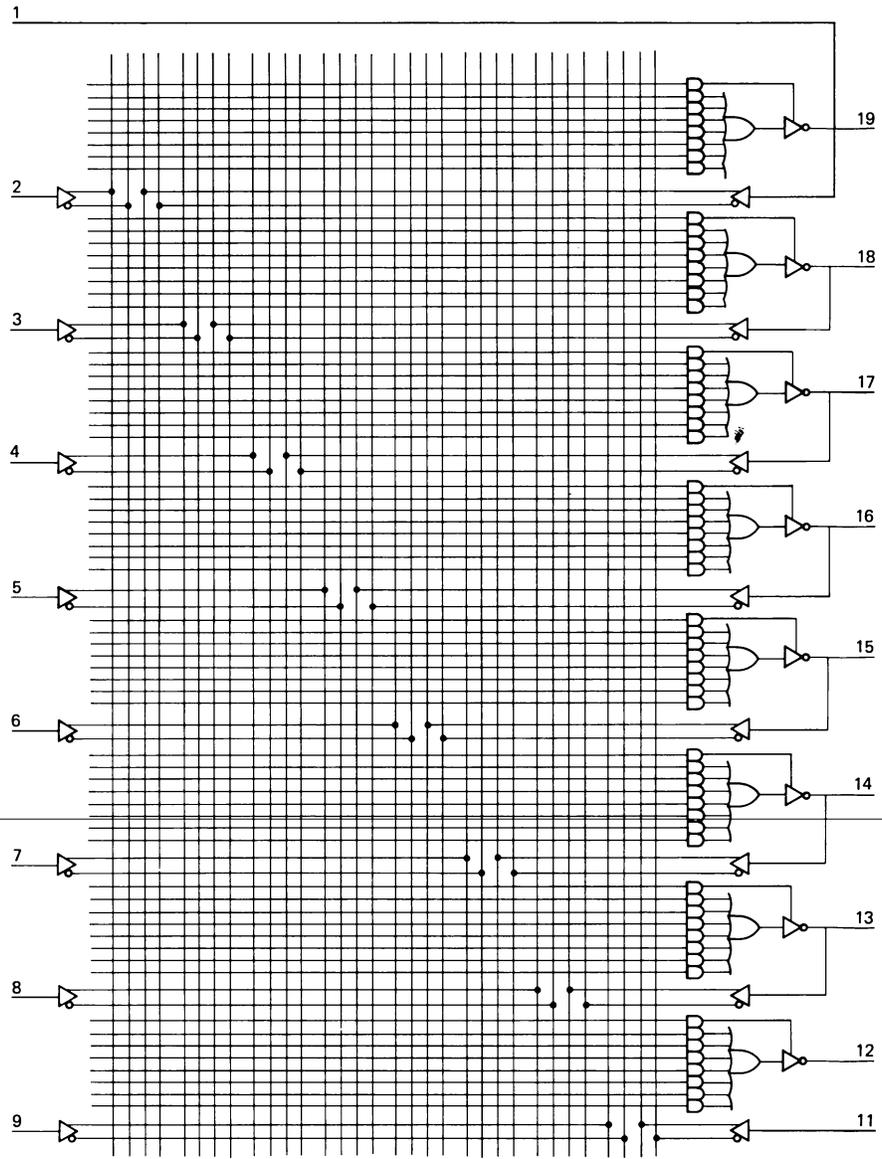
Figure A-7   16L8 PAL Device Logic Diagram
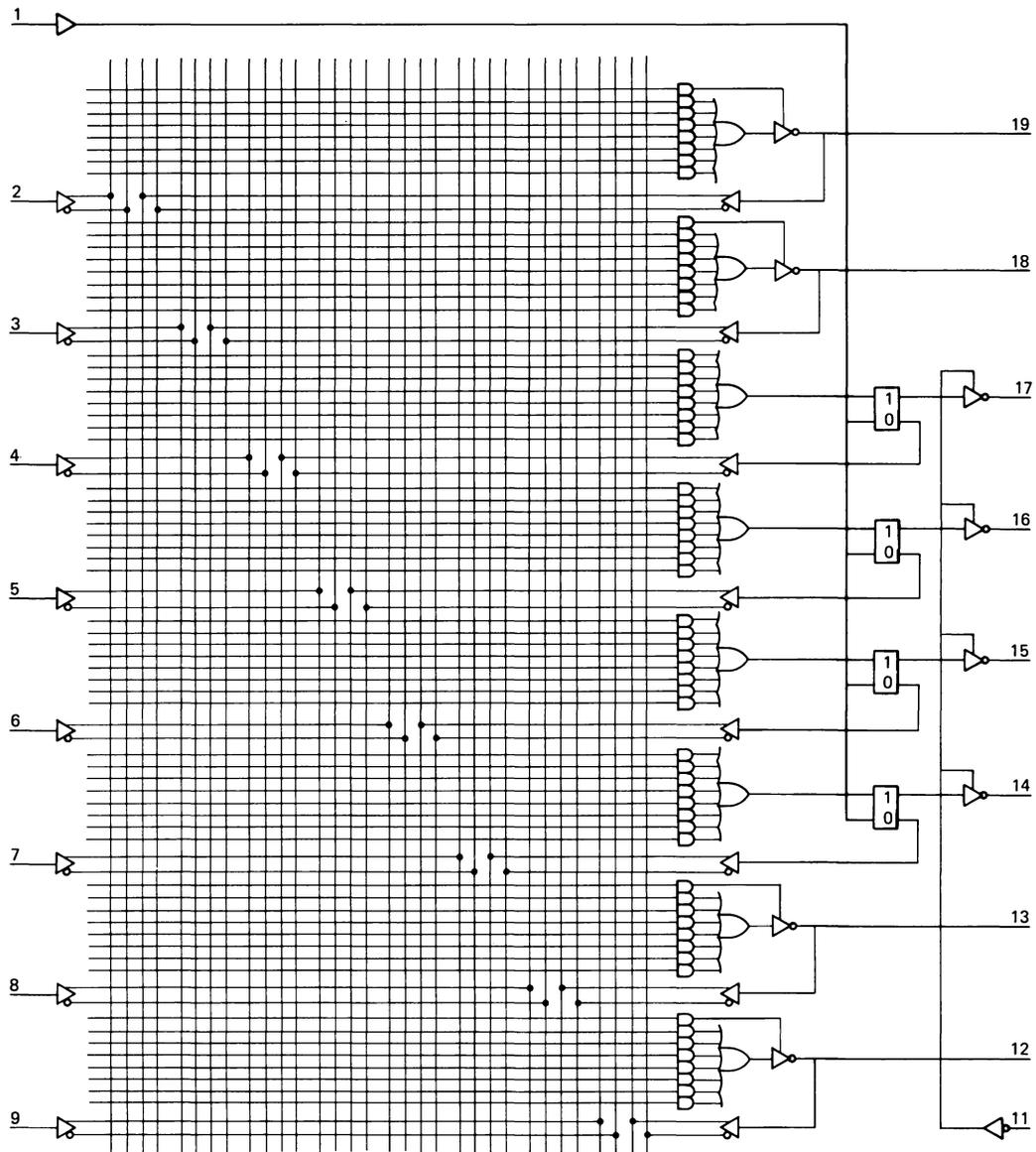
TK-6624
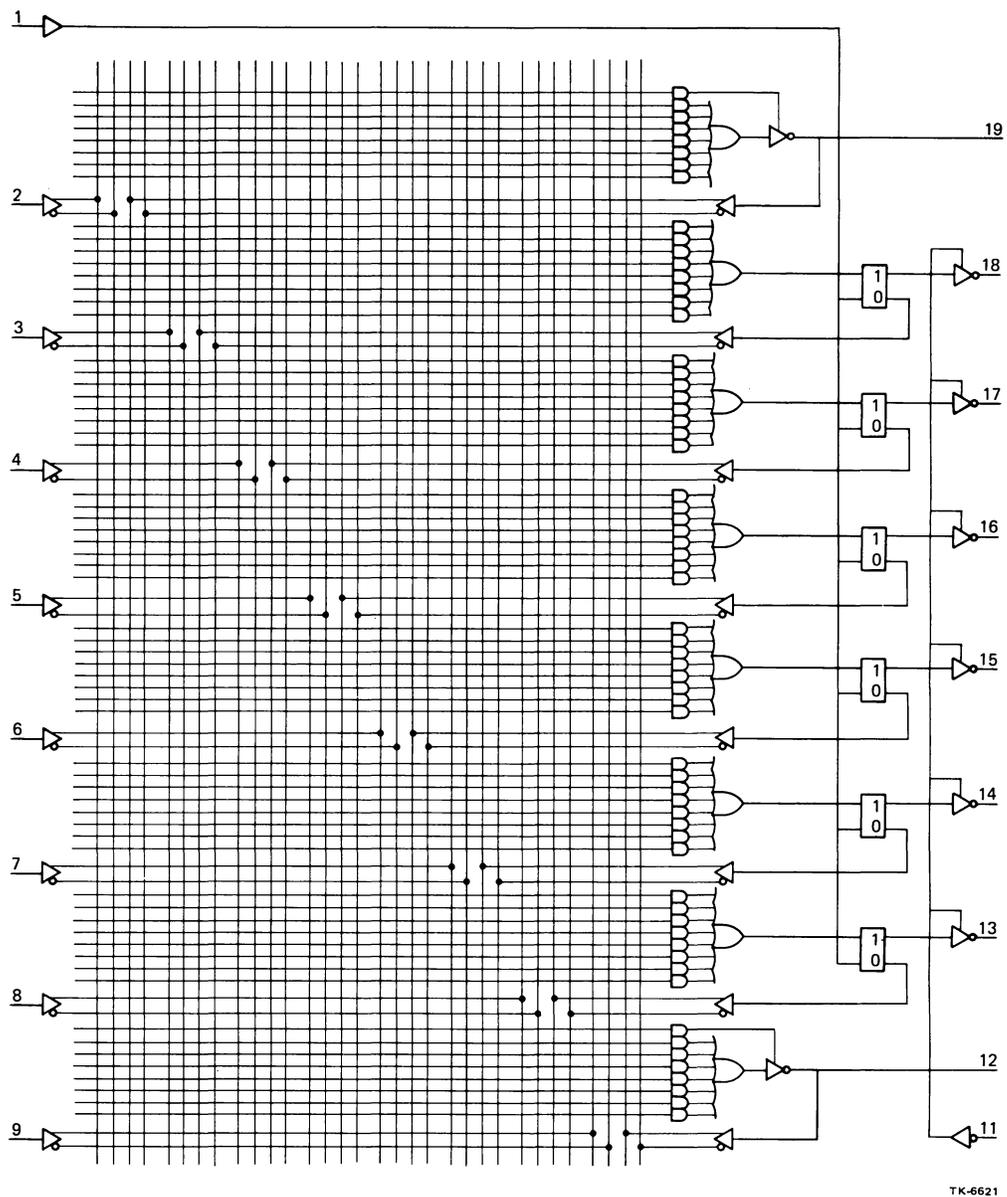
TK-6623

Figure A-8　16R4 PAL Device Logic Diagram

A-9

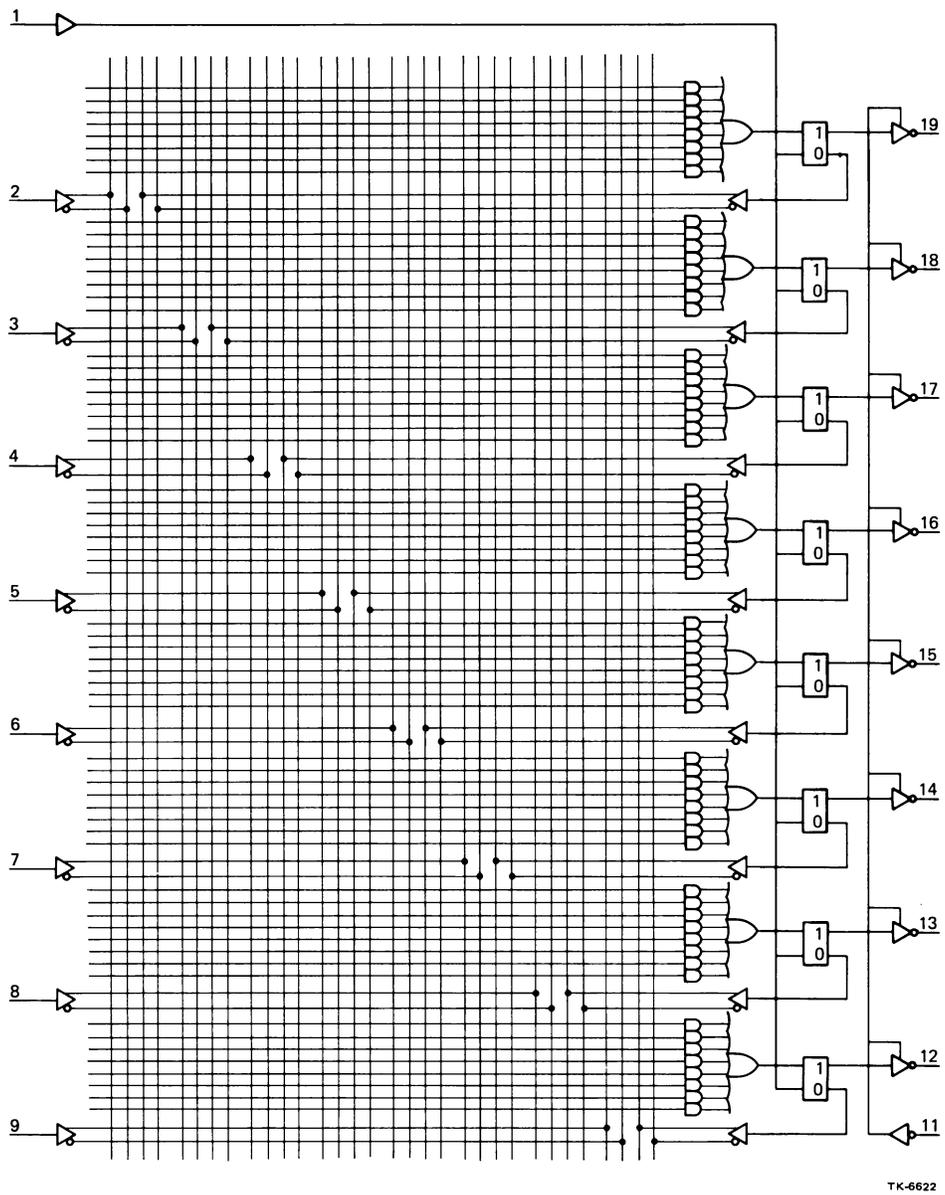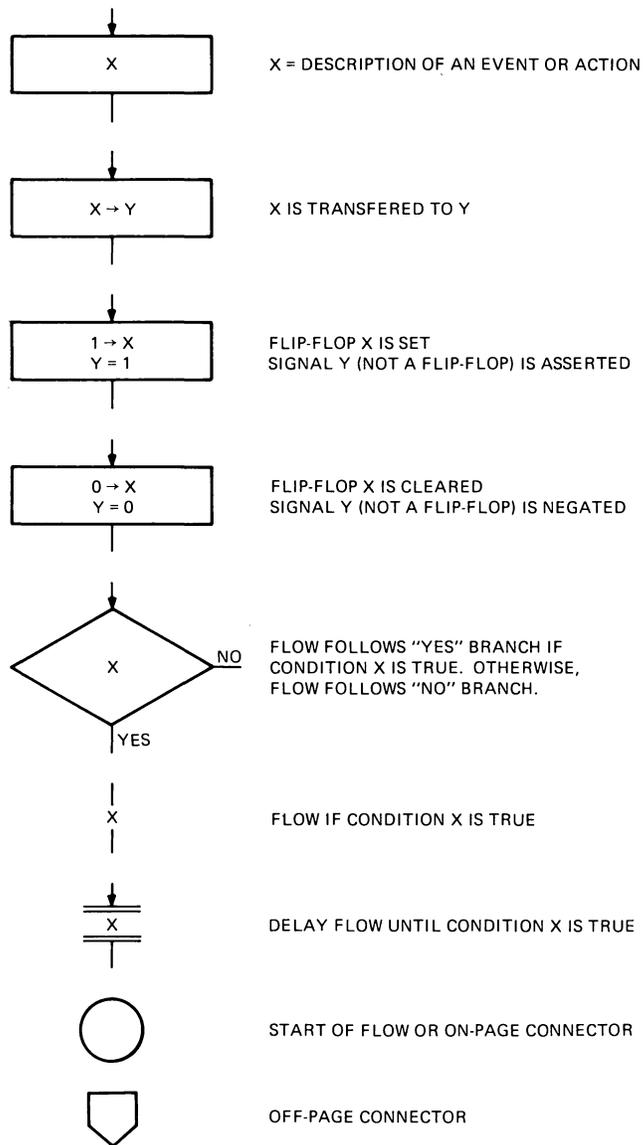Figure A-9   16R6 PAL Device Logic Diagram

TK-6621

Figure A-10   16R8 PAL Device Logic Diagram

TK-6622

The flow diagram symbols used in this technical description for the VAX-11/730 CPU are shown and described in Figure B-1.

Figure B-1   Flow Diagram Symbols

VAX-11/730   **Reader's Comments**
CENTRAL PROCESSING UNIT
TECHNICAL DESCRIPTION
EK-KA730-TD-001

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc? Is it easy to use? _____

_____

_____

_____

What features are most useful? _____

_____

_____

What faults or errors have you found in the manual? _____

_____

_____

Does this manual satisfy the need you think it was intended to satisfy? _____

Does it satisfy *your* needs? _____ Why? _____

_____

_____

Please send me the current copy of the *Documentation Products Directory,* which contains information on the remainder of DIGITAL's technical documentation.

Name_____ Street_____
Title_____ City_____
Company_____ State/Country_____
Department_____ Zip_____

Additional copies of this document are available from:

Digital Equipment Corporation
Accessories and Supplies Group
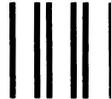P.O. Box CS2008
Nashua, New Hampshire 03061

Attention: Documentation Products
Telephone: 1-800-258-1710

Order No.__EK-KA730-TD-001_____

— Fold Here —