EK-KA750-TD-002

VAX-11/750 Central Processor Unit Technical Description

digital equipment corporation • maynard, massachusetts

First Edition, December 1980
Second Edition, March 1981

The material in this manual is for informational purposes and is subject to change without notice.

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

Printed in U.S.A.

CONTENTS

Page

CHAPTER 1 INTRODUCTION

CONTENTS (Cont)

Page

CHAPTER 2 THEORY OF OPERATION

CONTENTS (Cont)

Page

CONTENTS (Cont)

CONTENTS (Cont)

Page

FIGURES

Figure No. Title Page

FIGURES (Cont)

FIGURES (Cont)

FIGURES (Cont)

TABLES

TABLES (Cont)

TABLES (Cont)

CHAPTER 1
INTRODUCTION

1.1 MANUAL SCOPE

Chapter 1 of this manual provides a general description of the VAX-11/750. Chapter 2 provides a detailed functional description of the KA750 central processor. For a complete discussion of the KA750 central processor, this manual should be read in conjunction with the *VAX-11/750 Unibus Interface Technical Description* (EK-UI750-TD). This manual is a resource for appropriate branch and support level courses in the Field Service and Manufacturing training programs, and a field reference.

Detailed information concerning system components not covered in this manual can be found in the related literature listed in Table 1-2.

1.2 SYSTEM OVERVIEW

The VAX-11/750 is a 32-bit, high-speed, synchronous microprogrammed computer that represents a significant extension to the PDP-11 family of computers. The processor is capable of executing variable-length instructions in native mode, and nonprivileged PDP-11 instructions in compatibility mode. Compatibility mode enables existing user-mode PDP-11 programs to be run without modification.

The majority (90 percent) of the VAX-11/750 hardware logic design is implemented in custom large-scale integrated (LSI) circuits called gate arrays. These gate arrays are designed and manufactured specifically for the VAX-11/750. Gate array technology uses a fixed physical placement of 400 NAND gates (these gates are composed of bipolar circuit technology). Each gate array chip is configured during the manufacturing process to produce one of the 39 different types of gate array used in the VAX-11/750. These chips are used in the VAX-11/750 Central Processor Unit (CPU), floating-point accelerator, memory controller, and Massbus adapter.

Custom gate array technology has produced a positive impact on the VAX-11/750 design in a number ways.

- Increased speed per logic gate (5 to 10 ns)
- Lowered power consumption
- Fewer printed circuit boards due to LSI
- Increased reliability
- Lowered cost

For details on the preceding points see Table 1-1.

Table 1-1 Technology Specifications for the VAX-11/750

Implementation Technique – Gate Arrays
Circuit Technology – Low-Power Bipolar Schottky
Circuit Density – Large Scale Integration (LSI)

Die Size – .215 in × .244 in
Power Utilized per Die – 2 W max

Package Size – 1.44 in^2 (2.4 in × 0.6 in)
Number of Pins per Package – 48

I/O Circuits per Die – 44 I/O transceiver gates
Logic Gates – 400 identical 4-input NAND gates

Voltage Used – 2.5 V, 0.5 V

Speed per Gate – 5–10 ns

Unique Gate Array Types:

CPU and Memory Controller – 27
Floating Point Accelerator – 7
Massbus Adapter – 5

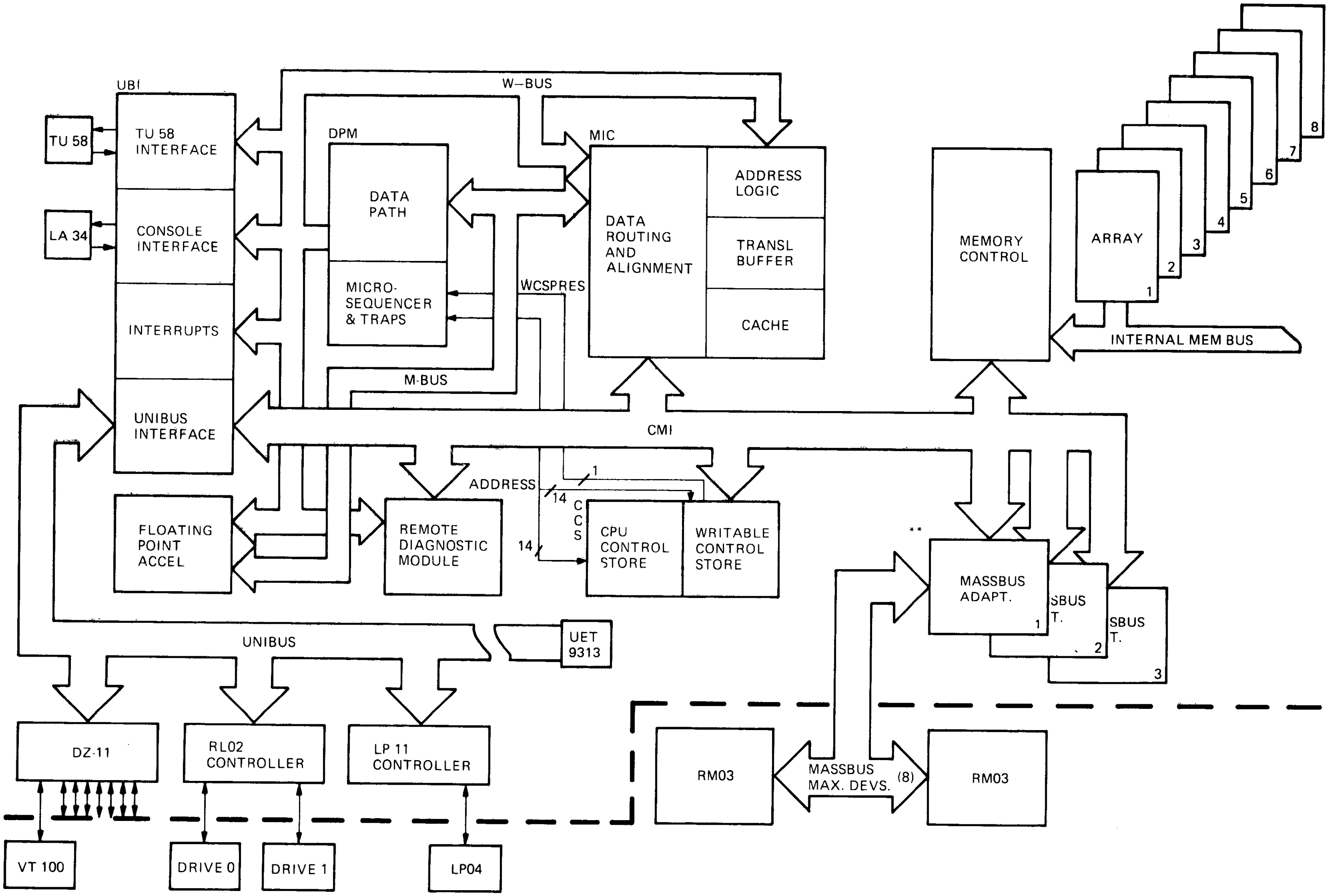
Total Number of Gate Arrays Used:

CPU and Memory Controller – 55
Floating Point Accelerator – 28
Massbus Adapter – 12

The major components of the VAX-11/750 system, shown in Figure 1-1, include the following.

Data Path Module (DPM)
Memory Interconnect Module (MIC)
CPU Control Store Module (CCS)
Unibus Interconnect Module (UBI) and peripherals
Memory Control
Massbus Adapter and Massbus peripherals
Floating-Point Accelerator (FPA) option
Remote Diagnostic Module (RDM) option
Writable Control Store (WCS) option

These major hardware components operate on clocked 320-ns cycles. Normal operations are synchronized by the system clock and each event occurs at defined points in time within the machine cycle.



*ONE MEMORY CONTROLLER CAN BE CONNECTED ALLOWING A MAXIMUM OF 8 X 256 BYTES = 2M BYTES.

**UP TO THREE MASSBUS ADAPTORS CAN BE CONNECTED.

Figure 1-1 VAX-11/750 System Block Diagram

1.2.1 VAX-11/750 Kernel Features

All VAX-11/750 system configurations are built around the VAX-11/750 "kernel." (See Figure 1-1.) The VAX-11/750 kernel consists of a central processing unit (CPU) with integral Unibus interfacing, integral TU58 and console terminal serial interfaces, a single-unit TU58 transport, and a memory controller with an initial 256K bytes of ECC MOS memory. The kernel also includes a single DZ11 (eight-line EIA with distribution panel) which is mounted in a nine-slot DD11 backplane. The standard VAX-11/750 kernel provides expansion capabilities in the form of mounting for optional WCS (writable control store), FPA (floating-point accelerator), and RDM (remote diagnosis module). The kernel allows slots for up to three Massbus adapters.

1.2.1.1 VAX-11/750 CPU – The VAX-11/750 CPU consists of the following four modules.

- Unibus Interface Module (UBI) – Contains a TU58 interface, console interface, interrupt logic, time-of-year clock, and Unibus interface.
- Data Path Module (DPM) – Includes the arithmetic logic, rotator logic, scratchpad logic (registers), interval timer, and the microsequencer logic.
- Memory Interconnect Module (MIC) – Holds address logic, translation buffer, execution buffer, cache, and data routing/alignment circuitry.
- CPU Control Store Module (CCS) – Contains the control store microcode ROMs. This module also houses the additional snap-on WCS module.

1.2.1.2 VAX-11/750 Memory Control – The VAX-11/750 allows the use of one memory controller. This memory controller contains its own microcode and performs as an interface between the CMI bus and up to 8 MOS ECC × 256K byte memory boards (2M bytes of main memory).

1.2.2 VAX-11/750 Internal Options

1.2.2.1 Floating-Point Accelerator (FPA) – An extended-hex module floating-point accelerator is available to increase system floating-point performance. The FPA feature is discussed in document EK-FP750-TD (Table 1-2).

1.2.2.2 Writable Control Store (WCS) – The WCS option provides customers with the capability of writing their own microcode for special applications.

1.2.2.3 Massbus Adapter (MBA) – An extended-hex module Massbus adapter option is available to allow incorporation of Massbus devices into the VAX-11/750. The Massbus adapter provides a high-speed, large-volume data path. Up to three Massbus modules may be installed on a system. Each Massbus adapter can accomodate up to eight devices.

1.2.2.4 Remote Diagnosis Module (RDM) – An extended-hex module remote diagnosis option is available for remote and local diagnosis of VAX-11/750 failures. The RDM is a Digital service tool that is not owned by the customer. This device is not functionally required for normal system operation.

1.2.2.5 Memory Arrays – Additional hex module memory arrays are available in 256K byte units up to the maximum system configuration of 2M bytes (8 hex modules).

1.2.2.6 Battery Backup (H7112) – An optional power supply is available to provide 10 minutes of battery backup for the fully configured memory.

1.2.2.7 Asynchronous Multiplexer (DZ11-A) – Up to four DZ11s and two H317 connectors can be supported in the VAX-11/750 cabinet. One DZ11-A with a connector panel is included in the base system.

Table 1-2 Related Manuals

Title	Document Number
Technical Descriptions:	
VAX-11/750 Unibus Interface (UBI)	EK-UI750-TD
MS750 Memory System	EK-MS750-TD
PS750 Power System	EK-PS750-TD
RH750 Massbus Adapter (MBA)	EK-RH750-TD
FP750 Floating-Point Accelerator (FPA)	EK-FP750-TD
Diagnostic System:	
VAX-11 Diagnostic System User's Guide	EK-VX11D-UG
VAX-11/750 Diagnostic System Overview	EK-VXD75-UG
User Documentation:	
Site Preparation Data Sheets	EK-CORP-SP
Installation/Acceptance	EK-SI750-IN
VAX-11 Architecture Handbook	EB-17580-18
VAX-11 Software Handbook	EB-15485-18
VAX-11 Hardware Handbook	EB-17281-20
VAX-11/750 Gate Array Chip Reference Manual	EK-GA750-RM

1.3 VAX-11/750 SYSTEM ARCHITECTURE

The majority of the VAX-11/750 system architecture is identical to that of the VAX-11/780. The system architecture is covered extensively in the *VAX-11 Architecture Handbook*, which is available from Digital Equipment Corporation (see Table 1-2).

This paragraph provides a quick reference, in table form, for data types and their representations, addressing modes, operand formats, and internal processor registers (IPRs).

1.3.1 Data Types and Their Representations

See Table 1-3 and Figure 1-2.

1.3.2 Addressing Modes

See Table 1-4.

1.3.3 Operand Formats

See Figures 1-3 through 1-19.

Table 1-3 Data Types

Data Type	Size	Range (Decimal)	
Integer		**Signed**	**Unsigned**
Byte	8 bits	-128 to $+127$	0 to 255
Word	16 bits	-32768 to $+32767$	0 to 65535
Longword	32 bits	-2^{31} to $+2^{31}-1$	0 to $2^{32}-1$
Quadword	64 bits	-2^{63} to $+2^{63}-1$	0 to $2^{64}-1$
Floating Point		$\pm 2.9 \times 10^{37}$ to 1.7×10^{38}	
F_floating	32 bits	Approximately seven decimal digits precision	
D_floating	64 bits	Approximately sixteen decimal digits precision	
Packed Decimal String	0 to 16 bytes (31 digits)	Numeric, 2 digits per byte Sign in low half of last byte	
Character String	0 to 65535 bytes	One character per byte	
Variable-length Bit Field	0 to 32 bits	Dependent on interpretation	
Numeric String	0 to 31 bytes (digits)	$-10^{31}-1$ to $+10^{32}-1$	
Queue	2 longwords/ queue entry	0–2 billion entries	

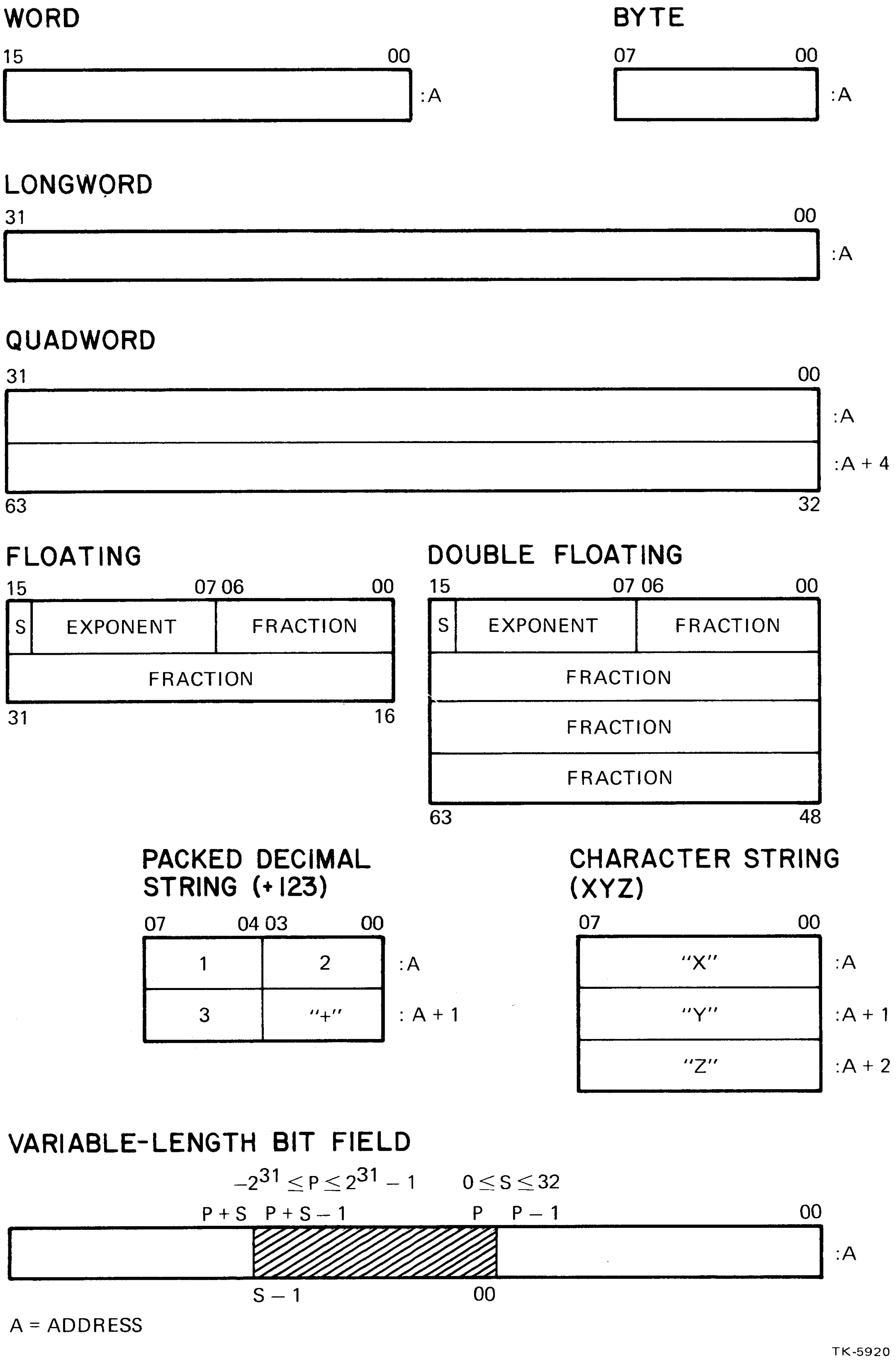


Figure 1-2 Data Type Representation

Table 1-4 Addressing Modes

Literal	S	# constant	
(Immediate)	I		
Register	R	n	
Register Deferred	(Rn)		
Autodecrement	−(Rn)		
Autoincrement	(Rn)+		
Autoincrement Deferred	@(Rn)+		
(Absolute)	@#address		Indexed [Rx]
Displacement	B W L	displacement (Rn)	
Displacement Deferred	@B W L	displacement (Rn) address	

Note:
n = 0 through 15
x = 0 through 14

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)

TK-0283

Figure 1-3 General Format of VAX-11 Instructions

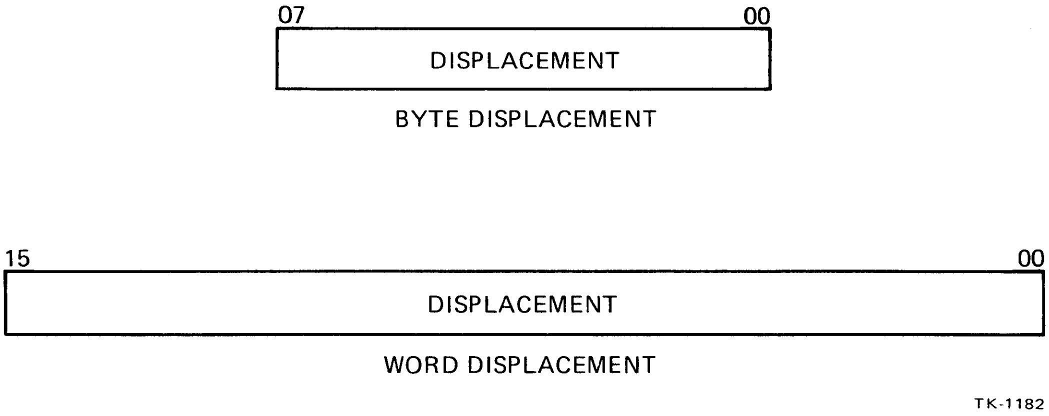


Figure 1-4 Operand Specifier Formats for Branch Mode Addressing

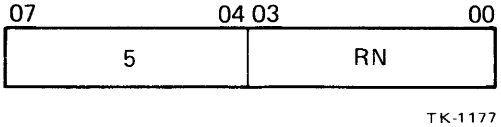


Figure 1-5 Operand Specifier Format in Register Mode

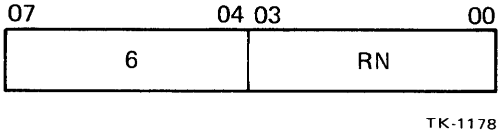


Figure 1-6 Operand Specifier Format in Register Deferred Mode

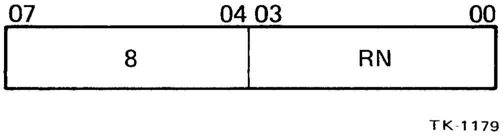


Figure 1-7 Operand Specifier Format in Autoincrement Mode

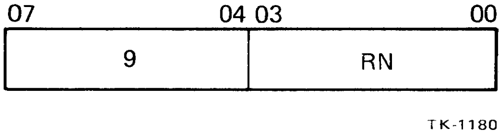


Figure 1-8 Operand Specifier Format in Autoincrement Deferred Mode

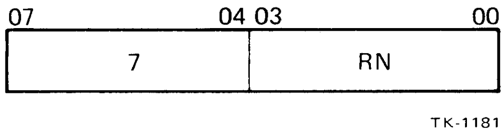


Figure 1-9 Operand Specifier Format in Autodecrement Mode

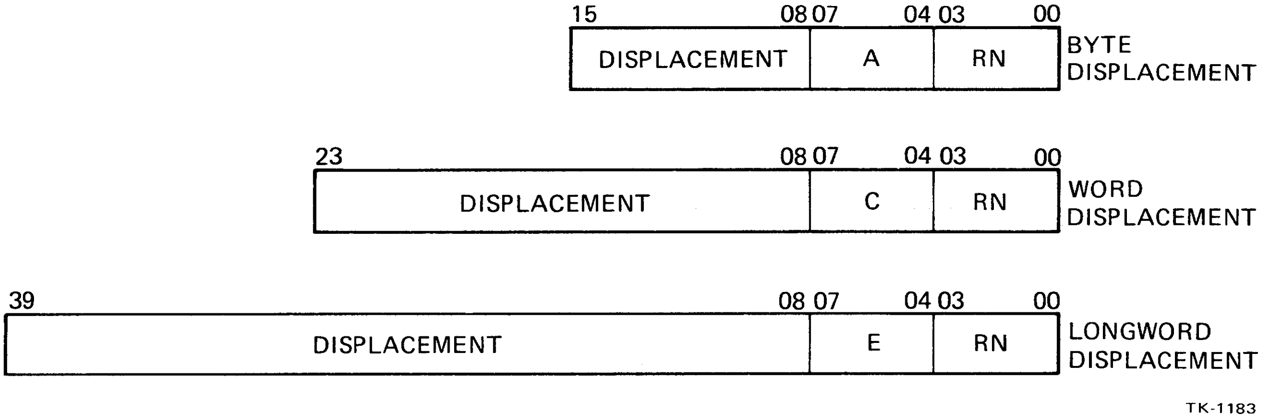


Figure 1-10 Operand Specifier Format in Displacement Mode

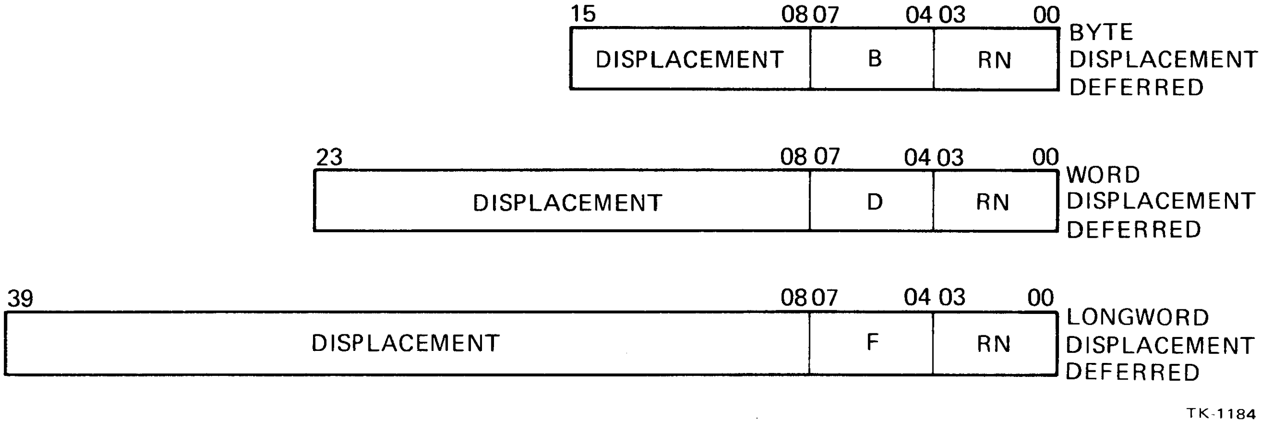


Figure 1-11 Operand Specifier Format in Displacement Deferred Mode

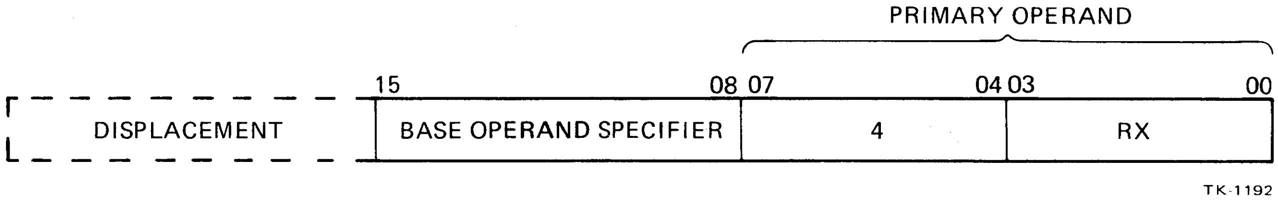
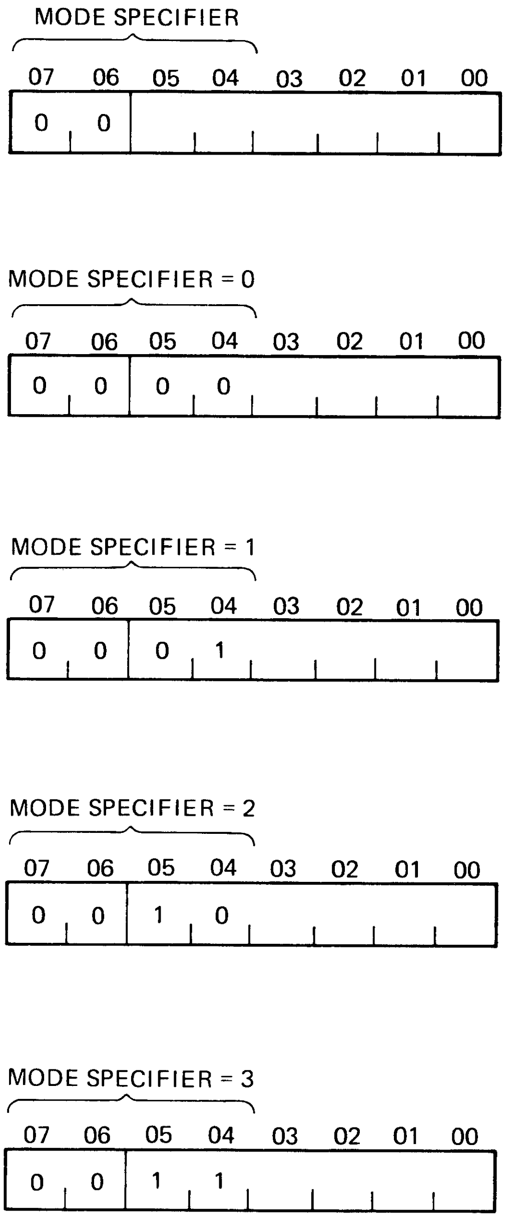
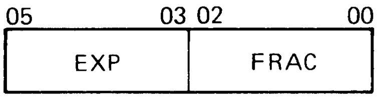


Figure 1-12 Operand Specifier Format in Index Mode



TK-1193

Figure 1-13 Operand Specifier Formats in Literal Mode



TK 1191

Figure 1-14 Floating Literal Format

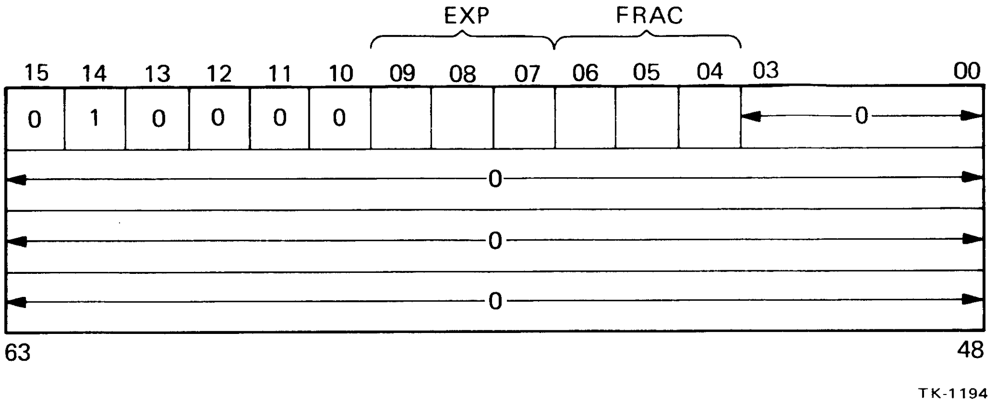


Figure 1-15 Literal Fields in Floating/Double Floating Operands

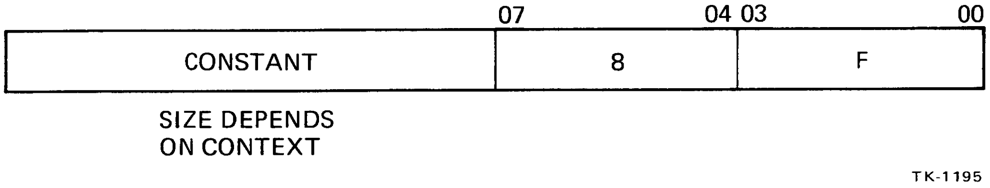


Figure 1-16 Operand Specifier Format in Immediate Mode

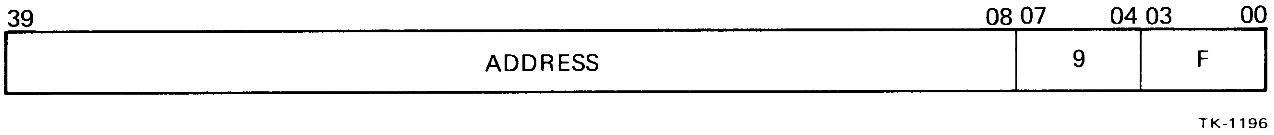


Figure 1-17 Operand Specifier Format in Absolute Mode

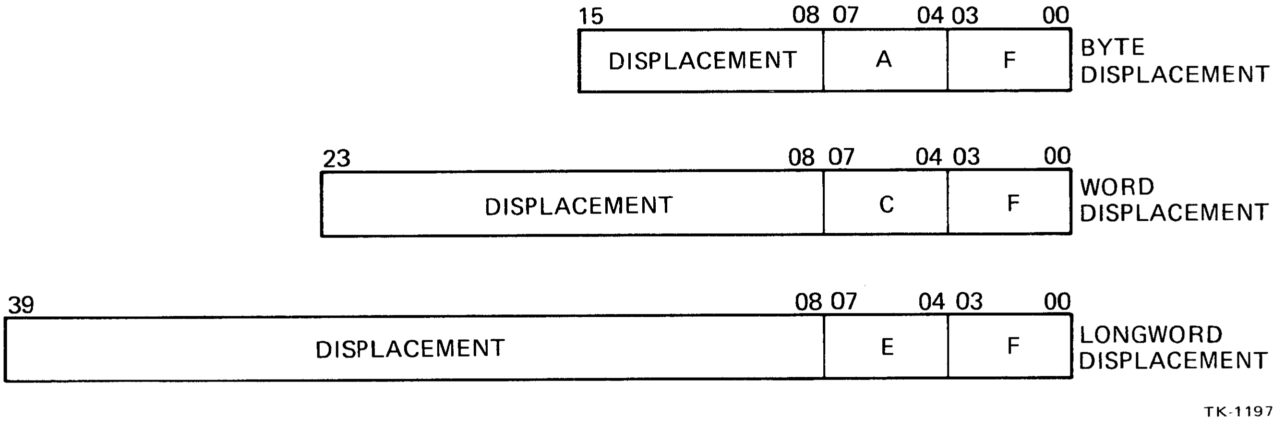


Figure 1-18 Operand Specifier Format in Relative Mode

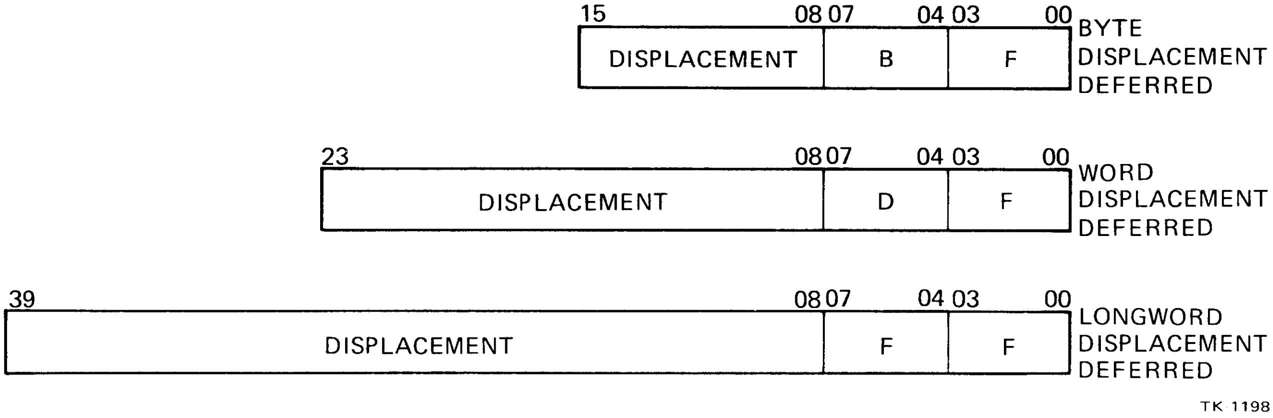


Figure 1-19 Operand Specifier Format in Relative Deferred Mode

1.3.4 Internal Processor Registers (IPRs)

VAX-11/750 IPRs may be accessed for a read or write operation by using the instructions Move to Processor Register (MTPR) and Move from Processor Register (MFPR). Another way to access the IPRs is to use examine/deposit commands while operating in console mode.

Accessing IPRs through MTPR and MFPR Instructions – See Table 1-5 and Figure 1-20.

Format:

Opcode src.rl, regnumber.rl MTPR
Opcode regnumber.rl, dst.wl MFPR

Operation:

If PSL <current-mode> NEQU kernel then (reserved instruction fault); PRS [regnumber]src;!MTPR
dstPRS [regnumber]; !MFPR

Condition Codes:

Ndst LSS O;
Zdst EQL O;
VO;
Cc;

Exceptions:

Reserved operand
Reserved instruction

Opcode:

DA MTPR Move to Processor Register
DB MFPR Move from Processor Register

Description:

The specified register is loaded or stored. The regnumber operand is a longword that contains the processor register number. Execution may have register-specific side effects.

NOTES

1. **A reserved operand fault occurs if the processor internal register does not exist or is read-only for MTPR or write-only for MFPR. It also occurs on some invalid operands to some registers.**
2. **A reserved instruction fault occurs if instruction execution is attempted in other than kernel mode.**

Table 1-5 lists and identifies the IPRs. The RW column indicates the read/write characteristics of each IPR. Figure 1-20 shows the bit structure of each of the IPRs.

Table 1-5 VAX-11/750 Internal Processor Registers (IPRs)

IPR No.	Mnemonic	RW*	Name
00	KSP	RW	Kernel Stack Pointer
01	ESP	RW	Executive Stack Pointer
02	SSP	RW	Supervisor Stack Pointer
03	USP	RW	User Stack Pointer
04	ISP	RW	Interrupt Stack Pointer
05	Reserved		
06	Reserved		
07	Reserved		
08	P0BR	RW	P0 Base Register
09	P0LR	RW	P0 Length Register
0A	P1BR	RW	P1 Base Register
0B	P1LR	RW	P1 Length Register
0C	SBR	RW	System Base Register
0D	SLR	RW	System Length Register
0E	Reserved		
0F	Reserved		
10	PCBB	RW	Process Control Block Base
11	SCBB	RW	System Control Block Base
12	IPL	RW	Interrupt Priority Level
13	ASTR	RW	AST Level Register
14	SIRR	WO	Software Interrupt Request Register
15	SIR	RW	Software Interrupt Summary Register
16	Reserved		
17	CMIERR	RO	CMI Error Register
18	ICCS	RW	Interval Clock Control/Status
19	NICR	WO	Next Interval Count Register
1A	ICR	RO	Interval Count Register
1B	TODR	RW	Time of Day Register
1C	CSRS	RW	Console Storage Receiver Status
1D	CSRD	RO	Console Storage Receiver Data
1E	CSTS	RW	Console Storage Transmit Status
1F	CSTD	WO	Console Storage Transmit Data
20	RXCS	RW	Console Receive Control/Status
21	RXDB	RO	Console Receive Data Buffer
22	TXCS	RW	Console Transmit Control/Status
23	TXDB	WO	Console Transmit Data Buffer
24	TBDR	RW	Translation Buffer Disable Register
25	CADR	RW	Cache Disable Register
26	MCESR	RW	Machine Check Error Summary Register
27	CAER	RW	Cache Error Register

*RO means read-only; WO means write-only. RW means both read and write.

Table 1-5 VAX-11/750 Internal Processor Registers (IPRs) (Cont)

IPR No.	Mnemonic	RW*	Name
28	ACCS	RO	Accelerator Control/Status Register
29	Reserved		
2A	Reserved		
2B	Reserved		
2C	Reserved		
2D	Reserved		
2E	Reserved		
2F	Reserved		
30	Reserved		
31	Reserved		
32	Reserved		
33	Reserved		
34	Reserved		
35	Reserved		
36	Reserved		
37	IO RESET	WO	Initialize Unibus
38	MME	RW	Memory Management Enable
39	TBIA	WO	Translation Buffer Invalidate All
3A	TBIS	WO	Translation Buffer Invalidate Single
3B	TB Data	RW	Translation Buffer Data
3C	Reserved		
3D	PMR	RW	Performance Monitor Register
3E	SID	RO	System Identification
3F	Reserved		

*RO means read-only; WO means write-only. RW means both read and write.

HEX	NAME	
00	KSP	**KERNEL STACK POINTER**
01	ESP	**EXECUTIVE STACK POINTER**
02	SSP	**SUPERVISOR STACK POINTER**
03	USP	**USER STACK POINTER**
04	ISP	**INTERRUPT STACK POINTER**

31   00
VIRTUAL ADDRESS OF TOP OF STACK

08	P0BR	**P0 BASE REGISTER** RESERVED OPERAND FAULT IF VLA < 2**31
0A	P1BR	**P1 BASE REGISTER** RESERVED OPERAND FAULT IF VLA < 2**31 - 2**21

31 02	01 00
VIRTUAL LONGWORD ADDRESS	MBZ

09	P0LR	**P0 LENGTH REGISTER** LENGTH OF P0PT IN LONGWORDS
0B	P1LR	**P1 LENGTH REGISTER** 2**21 - LENGTH OF P1PT IN LONGWORDS
0D	SLP	**SYSTEM LENGTH REGISTER** LENGTH OF SPT IN LONGWORDS RESERVED OPERAND FAULT IF MBZ ≠0

31 22	21 00
MBZ	LENGTH IN LONGWORDS

TK-1750

Figure 1-20 IPR Bit Structures
(Sheet 1 of 10)

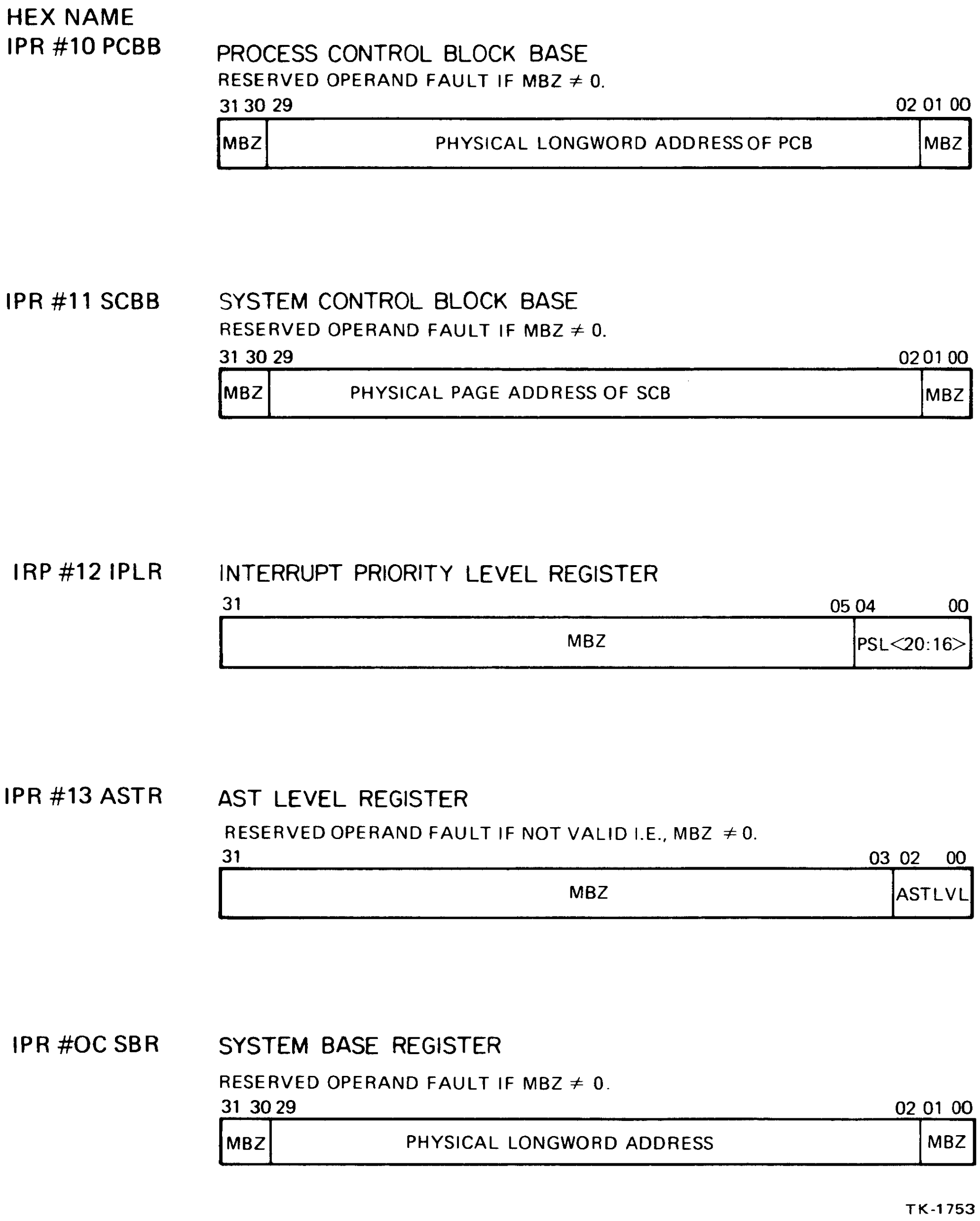

HEX NAME
IPR #10 PCBB
PROCESS CONTROL BLOCK BASE
RESERVED OPERAND FAULT IF MBZ ≠ 0.
31 30 29
02 01 00
MBZ
PHYSICAL LONGWORD ADDRESS OF PCB
MBZ
IPR #11 SCBB
SYSTEM CONTROL BLOCK BASE
RESERVED OPERAND FAULT IF MBZ ≠ 0.
31 30 29
02 01 00
MBZ
PHYSICAL PAGE ADDRESS OF SCB
MBZ
IRP #12 IPLR
INTERRUPT PRIORITY LEVEL REGISTER
31
05 04
00
MBZ
PSL<20:16>
IPR #13 ASTR
AST LEVEL REGISTER
RESERVED OPERAND FAULT IF NOT VALID I.E., MBZ ≠ 0.
31
03 02
00
MBZ
ASTLVL
IPR #OC SBR
SYSTEM BASE REGISTER
RESERVED OPERAND FAULT IF MBZ ≠ 0.
31 30 29
02 01 00
MBZ
PHYSICAL LONGWORD ADDRESS
MBZ
TK-1753

Figure 1-20 IPR Bit Structures
(Sheet 2 of 10)

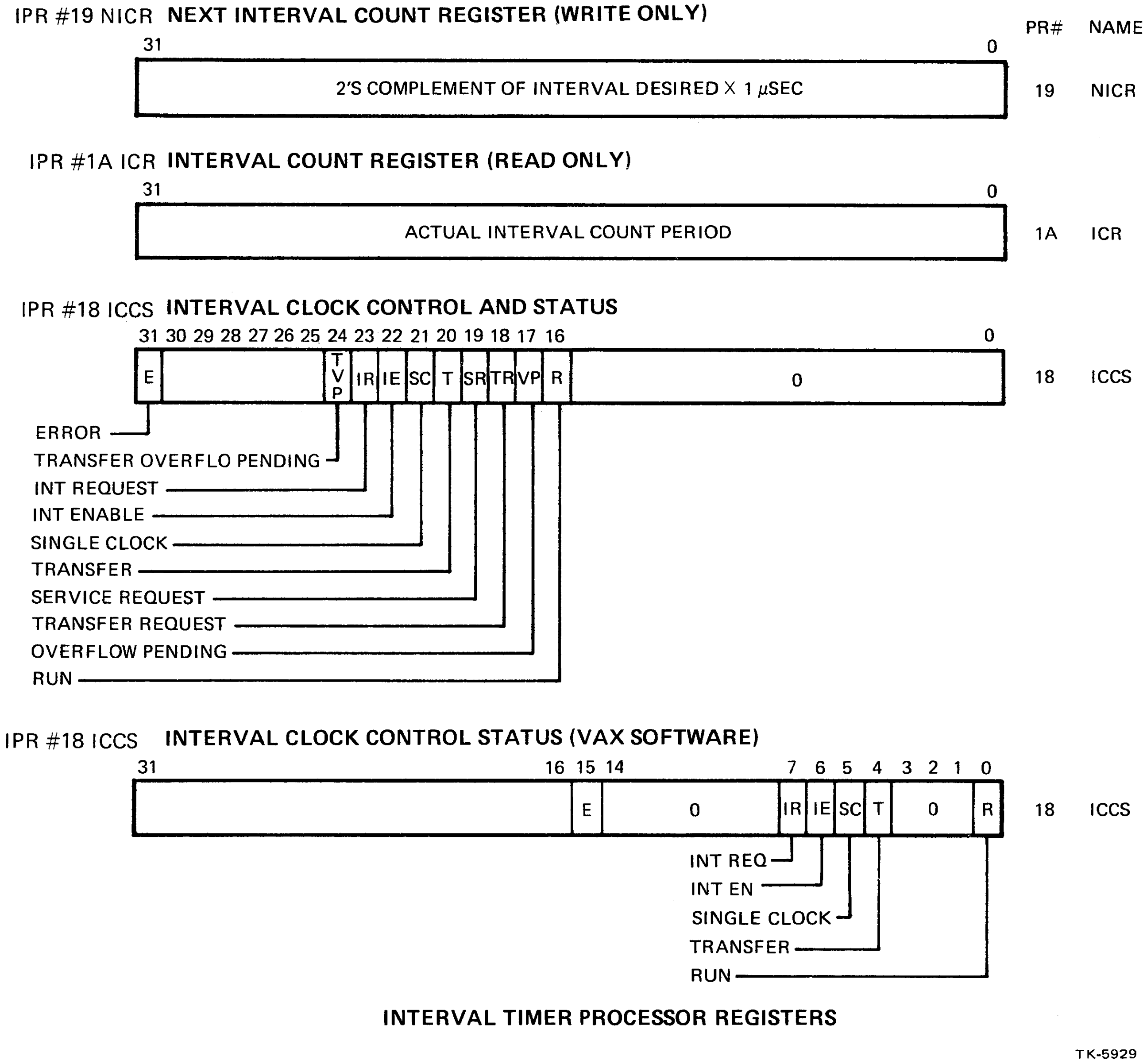

IPR #19 NICR NEXT INTERVAL COUNT REGISTER (WRITE ONLY)
PR#
NAME
31
0
2'S COMPLEMENT OF INTERVAL DESIRED X 1 μSEC
19
NICR
IPR #1A ICR INTERVAL COUNT REGISTER (READ ONLY)
31
0
ACTUAL INTERVAL COUNT PERIOD
1A
ICR
IPR #18 ICCS INTERVAL CLOCK CONTROL AND STATUS
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0
E
TVP
IR
IE
SC
T
SR
TR
VP
R
0
18
ICCS
ERROR
TRANSFER OVERFLO PENDING
INT REQUEST
INT ENABLE
SINGLE CLOCK
TRANSFER
SERVICE REQUEST
TRANSFER REQUEST
OVERFLOW PENDING
RUN
IPR #18 ICCS INTERVAL CLOCK CONTROL STATUS (VAX SOFTWARE)
31
16 15 14
7 6 5 4 3 2 1 0
E
0
IR
IE
SC
T
0
R
18
ICCS
INT REQ
INT EN
SINGLE CLOCK
TRANSFER
RUN
INTERVAL TIMER PROCESSOR REGISTERS
TK-5929

Figure 1-20 IPR Bit Structures
(Sheet 3 of 10)

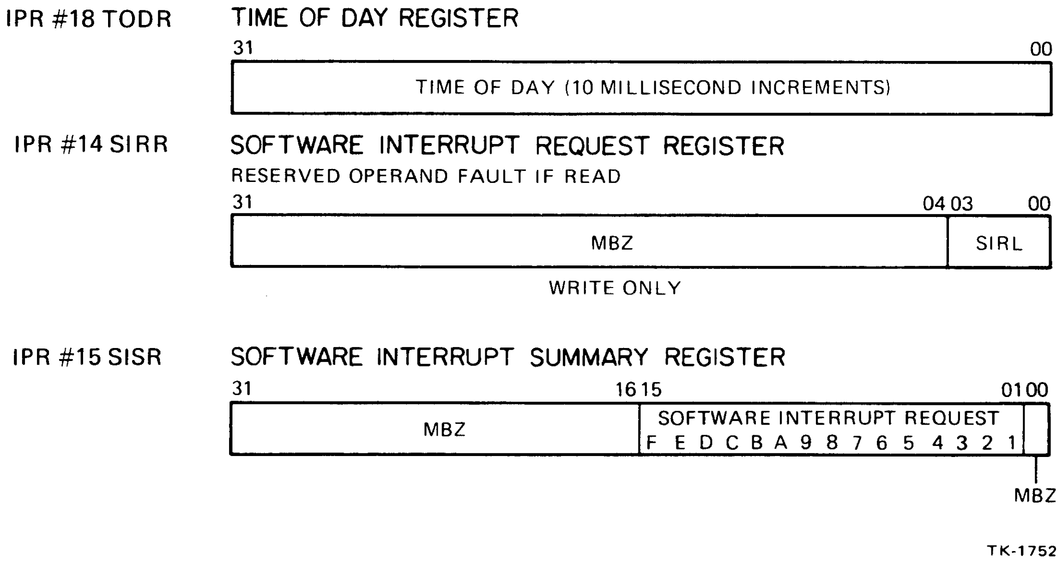

IPR #18 TODR
TIME OF DAY REGISTER
31
00
TIME OF DAY (10 MILLISECOND INCREMENTS)
IPR #14 SIRR
SOFTWARE INTERRUPT REQUEST REGISTER
RESERVED OPERAND FAULT IF READ
31
04 03
00
MBZ
SIRL
WRITE ONLY
IPR #15 SISR
SOFTWARE INTERRUPT SUMMARY REGISTER
31
16 15
01 00
MBZ
SOFTWARE INTERRUPT REQUEST
F E D C B A 9 8 7 6 5 4 3 2 1
MBZ
TK-1752

Figure 1-20 IPR Bit Structures
(Sheet 4 of 10))

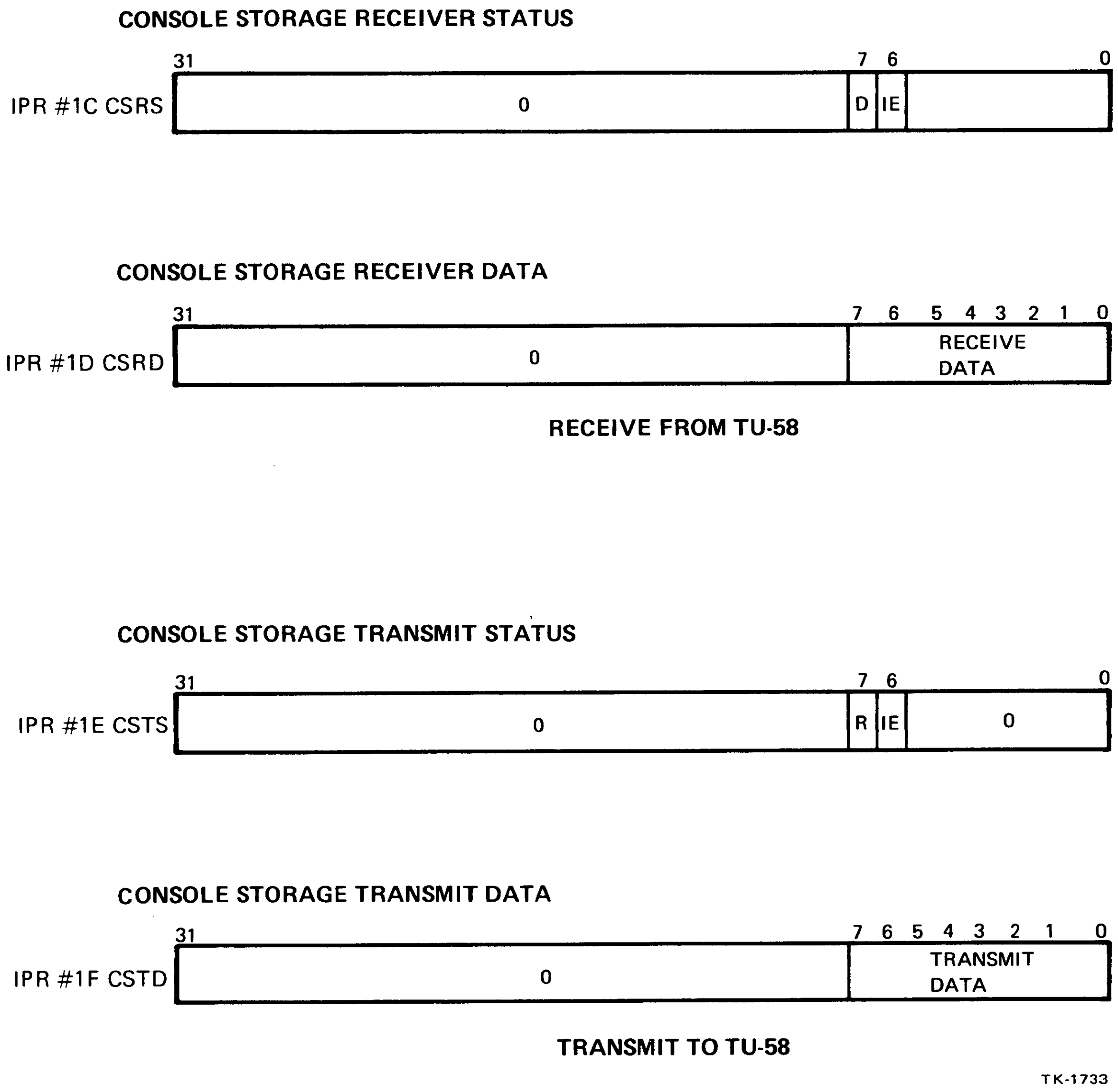
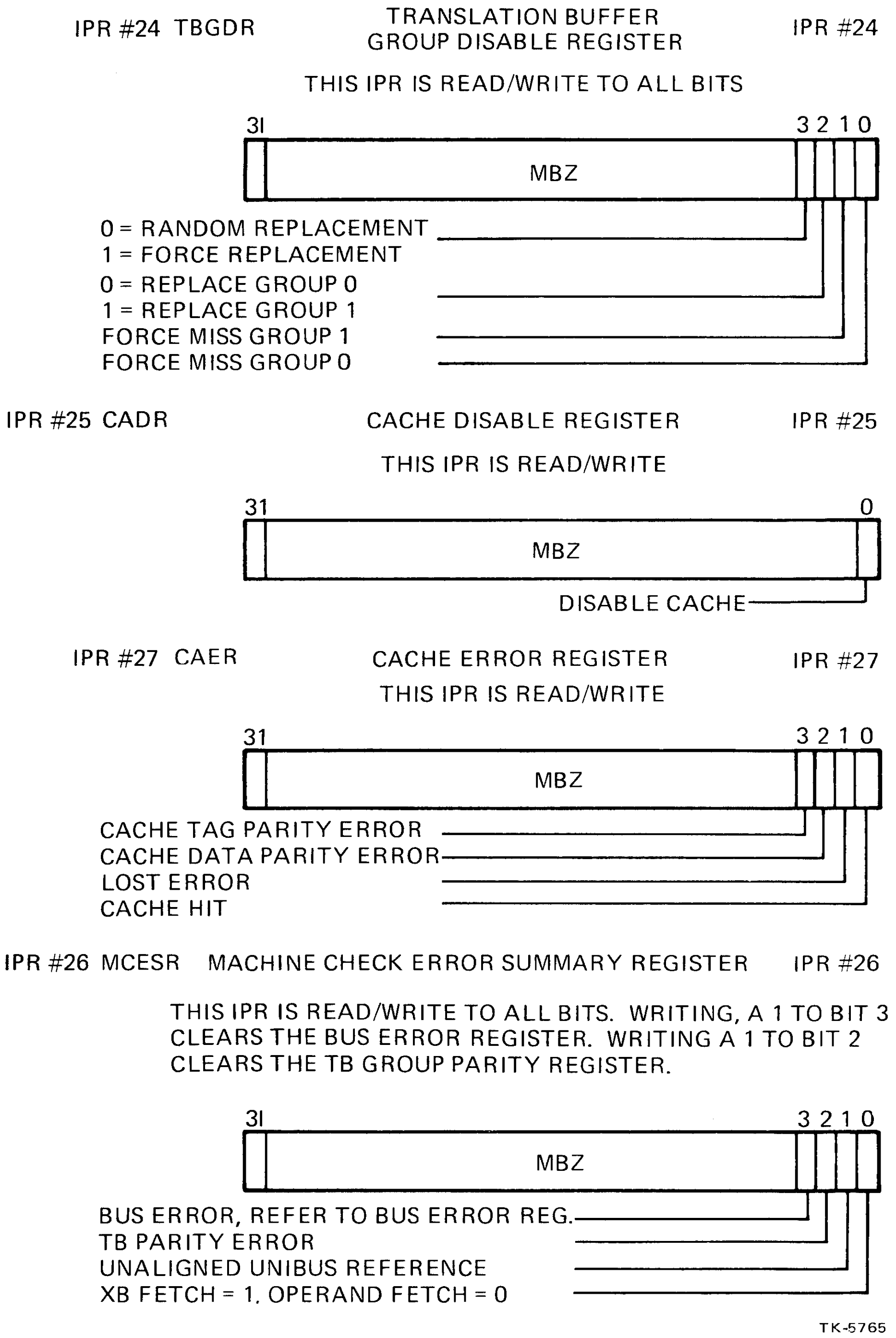


Figure 1-20 IPR Bit Structures
(Sheet 5 of 10)



THIS IPR IS READ/WRITE TO ALL BITS. WRITING, A 1 TO BIT 3 CLEARS THE BUS ERROR REGISTER. WRITING A 1 TO BIT 2 CLEARS THE TB GROUP PARITY REGISTER.

Figure 1-20 IPR Bit Structures
(Sheet 6 of 10)

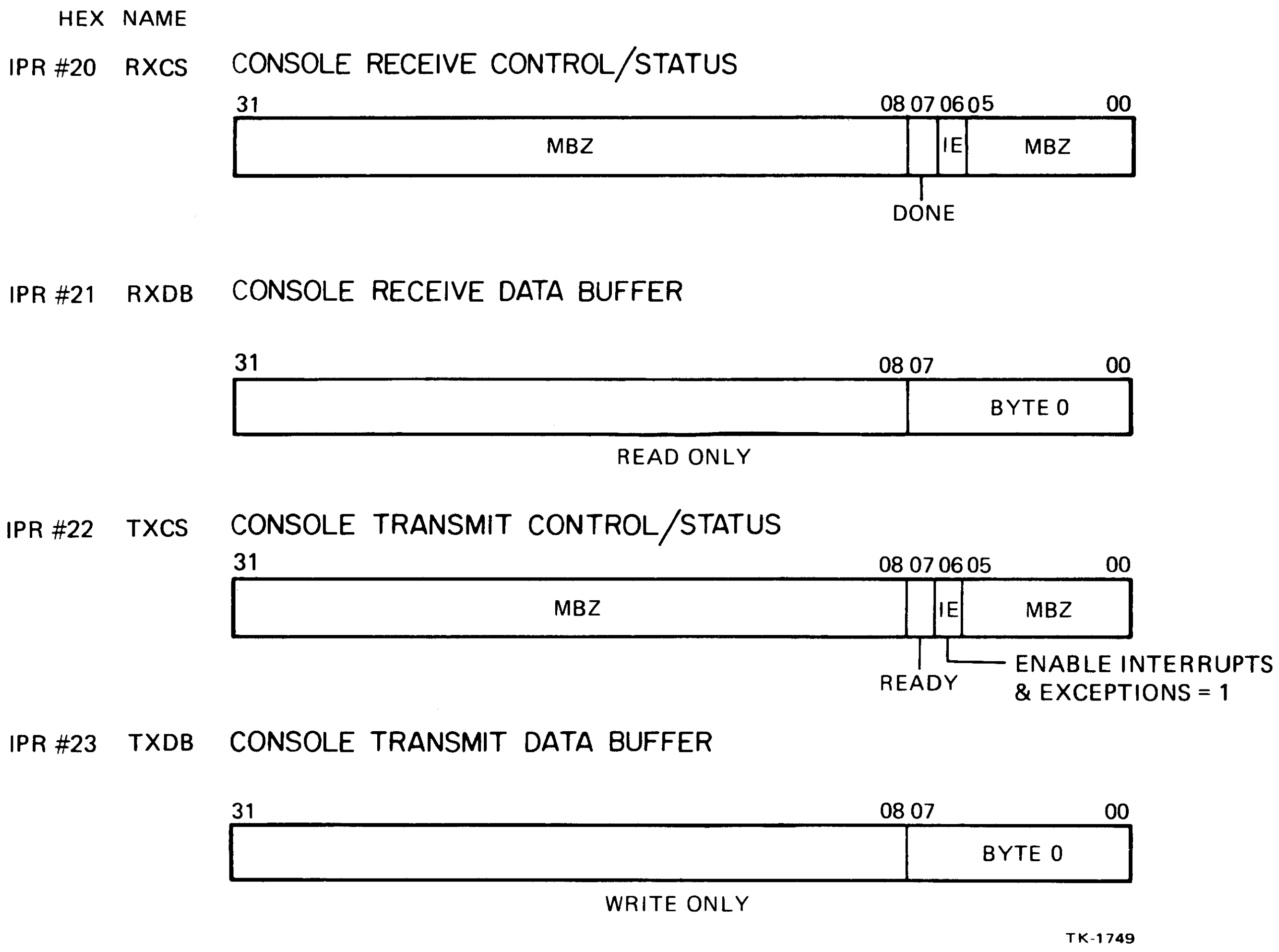

HEX NAME
IPR #20 RXCS CONSOLE RECEIVE CONTROL/STATUS
31
08 07 06 05
00
MBZ
IE
MBZ
DONE
IPR #21 RXDB CONSOLE RECEIVE DATA BUFFER
31
08 07
00
BYTE 0
READ ONLY
IPR #22 TXCS CONSOLE TRANSMIT CONTROL/STATUS
31
08 07 06 05
00
MBZ
IE
MBZ
READY
ENABLE INTERRUPTS & EXCEPTIONS = 1
IPR #23 TXDB CONSOLE TRANSMIT DATA BUFFER
31
08 07
00
BYTE 0
WRITE ONLY
TK-1749

Figure 1-20 IPR Bit Structures
(Sheet 7 of 10)

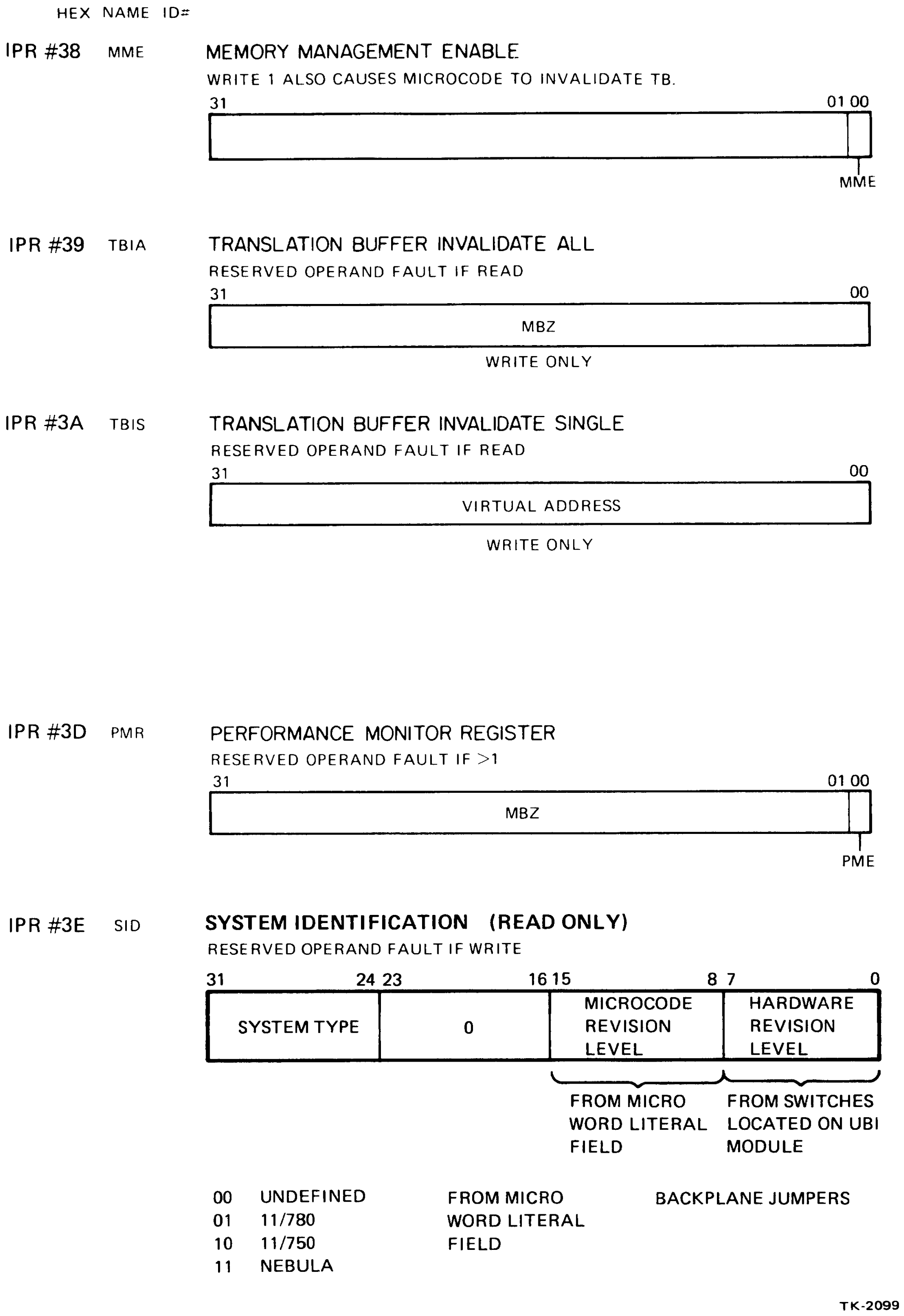
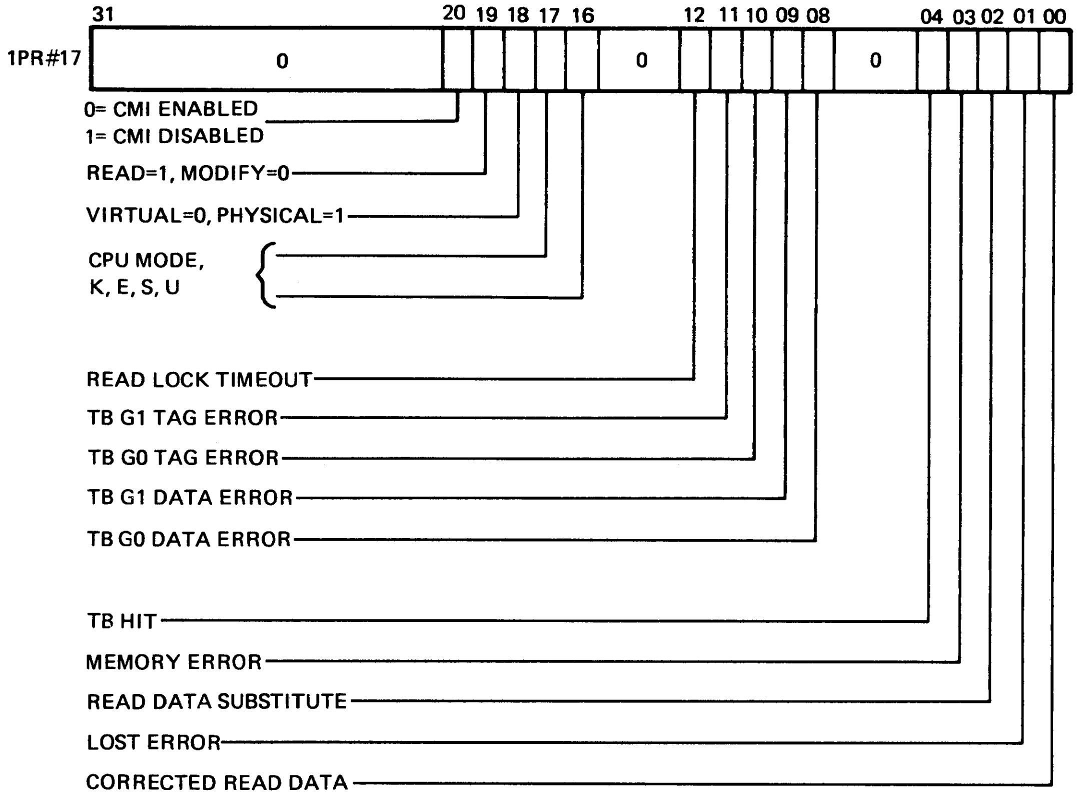

HEX NAME ID=
IPR #38 MME
MEMORY MANAGEMENT ENABLE
WRITE 1 ALSO CAUSES MICROCODE TO INVALIDATE TB.
31
01 00
MME
IPR #39 TBIA
TRANSLATION BUFFER INVALIDATE ALL
RESERVED OPERAND FAULT IF READ
31
00
MBZ
WRITE ONLY
IPR #3A TBIS
TRANSLATION BUFFER INVALIDATE SINGLE
RESERVED OPERAND FAULT IF READ
31
00
VIRTUAL ADDRESS
WRITE ONLY
IPR #3D PMR
PERFORMANCE MONITOR REGISTER
RESERVED OPERAND FAULT IF >1
31
01 00
MBZ
PME
IPR #3E SID
SYSTEM IDENTIFICATION (READ ONLY)
RESERVED OPERAND FAULT IF WRITE
31 24 23 16 15 8 7 0
SYSTEM TYPE
0
MICROCODE REVISION LEVEL
HARDWARE REVISION LEVEL
FROM MICRO WORD LITERAL FIELD
FROM SWITCHES LOCATED ON UBI MODULE
00 UNDEFINED
01 11/780
10 11/750
11 NEBULA
FROM MICRO WORD LITERAL FIELD
BACKPLANE JUMPERS
TK-2099

Figure 1-20 IPR Bit Structures
(Sheet 8 of 10)

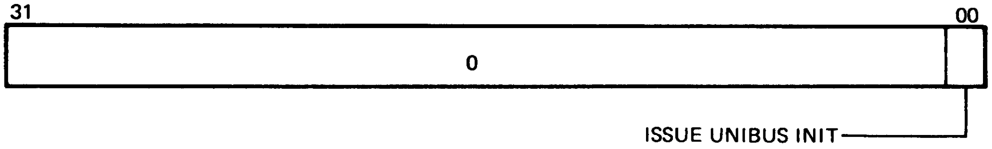


CMI ERROR PROCESSOR REGISTER

TK-3266

Figure 1-20 IPR Bit Structures
(Sheet 9 of 10)

IPR #37 IO RESET INITIALIZE UNIBUS



IO RESET PROCESSOR REGISTER

TK-3267

Figure 1-20 IPR Bit Structures
(Sheet 10 of 10)

1.4 VAX-11/750 CPU HARDWARE FUNCTIONAL OVERVIEW

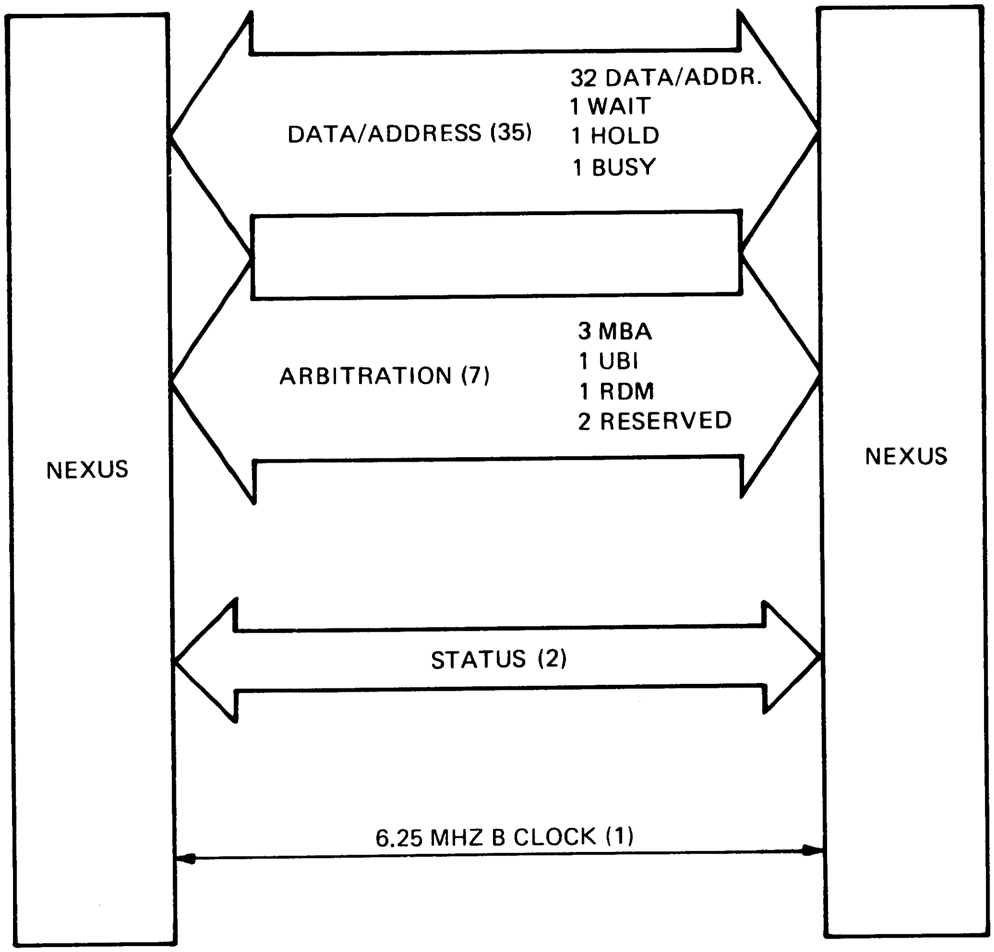
This section provides a functional description of the following circuitry.

- CPU – Memory Interconnect (CMI)
- MBus
- WBus
- Power Interface and Timing
- Data Path Module (DPM) Functionality
- CPU Control Store (CCS) Functionality
- Memory Interface and Control (MIC) Functionality
- Unibus Interface and Miscellaneous Hardware

Figure 1-1 provides a simplified overview of the VAX-11/750. The VAX-11/750 CPU is implemented on four modules: the data path module (DPM), the memory interconnect (MIC), the Unibus interface module (UBI), and the CPU control store (CCS) module. The DPM contains most of the arithmetic and logic functions, and the microsequencer. The MIC module consists of a translation buffer, execution buffer, data cache, and memory interface to the CMI. The UBI contains the integral Unibus interface along with the console and TU58 interfaces. The CCS module contains the microcode ROMs and interface for the optional writable control store (WCS). Functional block diagrams of each of these modules is provided in Figures 1-22 through 1-25.

1.4.1 CPU/Memory Interconnect (CMI)

The CMI consists of 45 bidirectional lines. These lines carry address, data, and priority arbitration between all subsystems on the backplane. The CMI relationship to the VAX-11/750 is shown in Figure 1-1. Figure 1-21 shows that the CMI signals are divided into four groups: bus clock (B CLK), data/address and control, priority arbitration, and status. Paragraph 2.5.9 describes the CMI signals and the timing and protocol involved in CMI operations.



TK-2064

Figure 1-21 The CMI Structure

1.4.2 MBus Overview

The MBus physically consists of 32 tri-state data lines. This bus is entirely under microcode control. The MBus acts as a major bus between three of the CPU modules: the FPA, DPM, and MIC module.

1.4.2.1 MBus Source Control – MBus data may be supplied from the following sources.

MTEMPs
Write Data Register (WDR)
Memory Data Register (MDR)
Virtual Address (VA) Register
Execution Buffer (XB)
PC Backup Register
Memory Address (MAD) Register
Translation Buffer (TB) Data

MBus data source is under control of the MSRC microfield.

1.4.2.2 MBus Destination Control – MBus data may be supplied to the ALP gate array chips, to the SRM (super rotator multiplexer), and to the FPA, when this option is present on the system.

MBus destination is under the control of several microfields. These fields are as follows: ALPCTL, FPA, MUX, and ROT.

1.4.3 WBus Overview

The WBus, like the MBus, consists of 32 tri-state data lines. This bus is also entirely under the control of microcode. The WBus provides a data path between sections of the DPM, MIC, UBI, FPA and RDM modules.

1.4.3.1 WBus Source Control – WBus data may originate from the following seven major sources.

Processor Status Longword (PSL)
Interval Timer
RNUM Register
Console and TU58 Interface Control
Time-of-Year (TOY) Clock
ALP Output
FPA Memory Status and Control Logic

Table 1-6 shows the microword fields that provide WBus source control.

Table 1-6 Microword Fields that Control the WBus

ALPCTL	
ALU	
ALUOD	
DQ1	
DQ2	ALU Group
DQ3	
LIT	
MUX	
WCTRL	
CCMISC	WCTRL Group
CCPSL	
FPA	
MSRC	Others

1.4.3.2 WBus Destination Control – Under microcode control, WBus data may be provided to the following destinations: the scratchpad registers, S and P latches, the microsequencer, condition code and PSL logic, RNUM, traps and interrupt logic, interval timer, console and TU58 interface control, address control logic, and finally to the FPA and RDM if these options are present on the system.

WBus data is supplied to the logic listed above under control of the following microcode fields: ALUSHF, BUS, BUT, CCPSL, FPA, MSRC, ROT, and WCTRL.

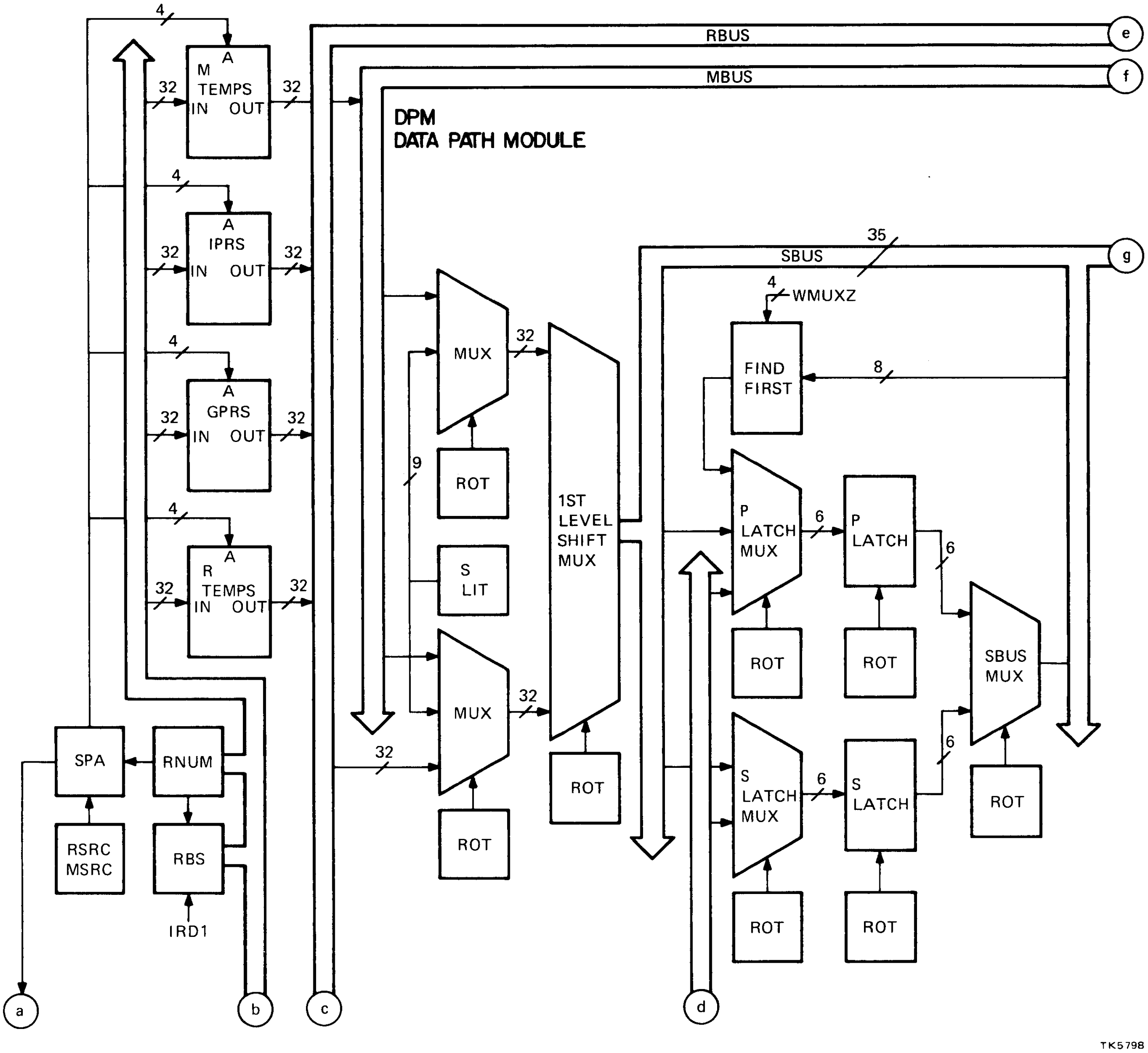


Figure 1-22 Data Path Module Functional Block Diagram
(Sheet 1 of 4)

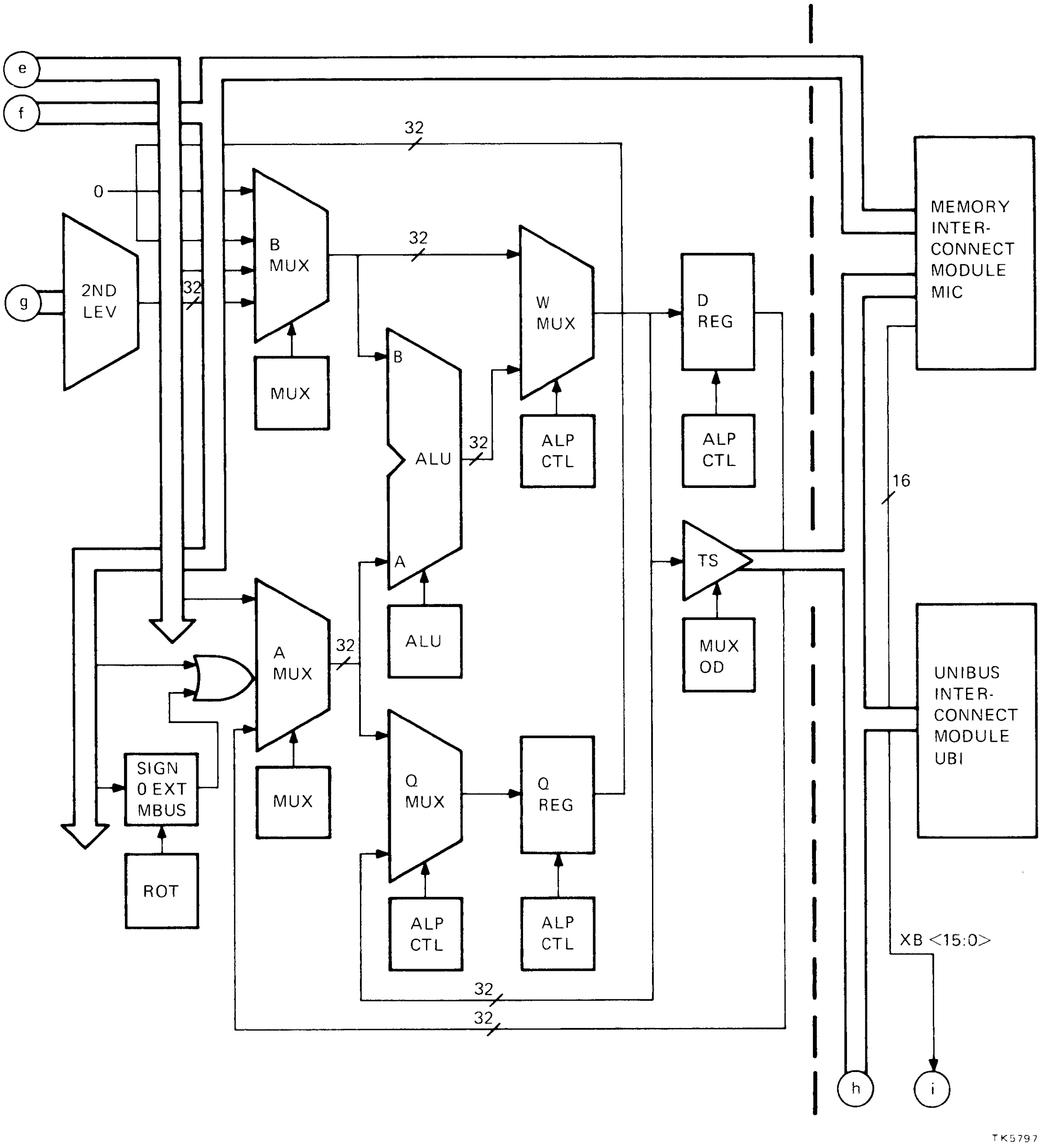


Figure 1-22 Data Path Module Functional Block Diagram (Sheet 2 of 4)

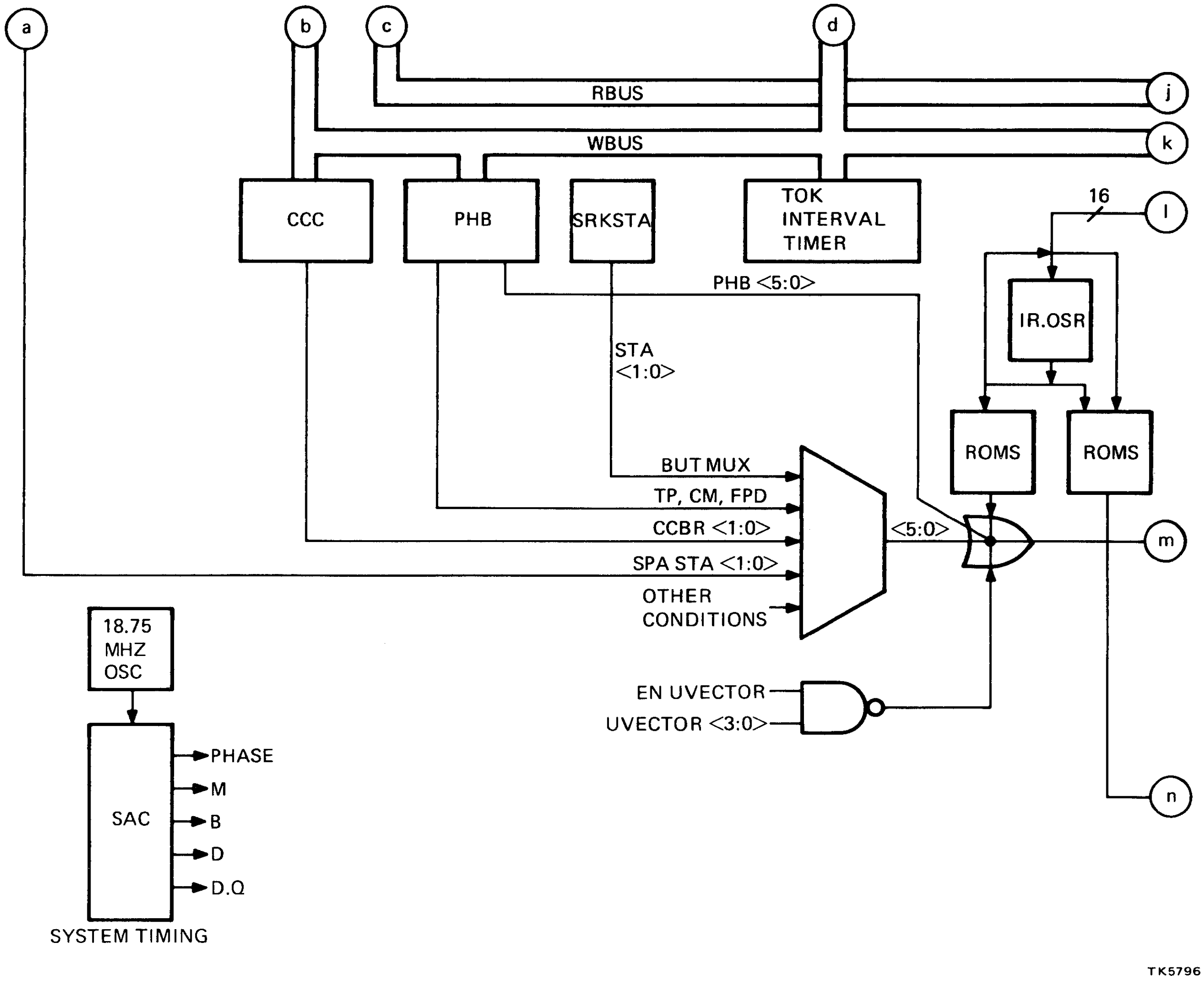


Figure 1-22 Data Path Module Functional Block Diagram
(Sheet 3 of 4)

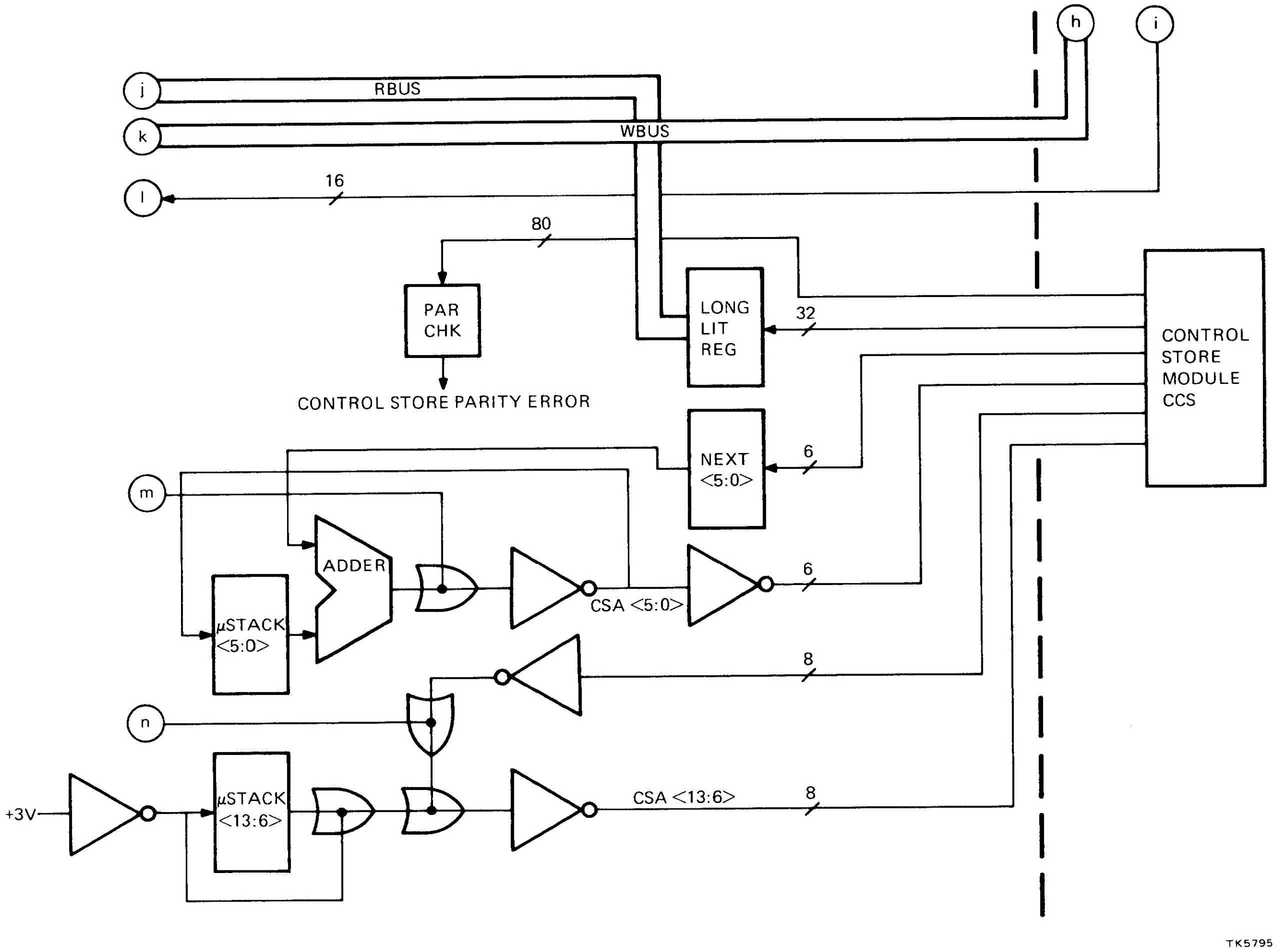


Figure 1-22 Data Path Module Functional Block Diagram (Sheet 4 of 4)

1.4.4 Power Interface and Timing

The power subsystem (not shown in the functional block diagrams) provides +5 Vdc, +2.5 Vdc, and the TOY clock battery. Power sequencing and control is accomplished by the power control section of the UBI module (see Figure 1-25). ACLO and DCLO interface to the UBI and microsequencer logic. MSEQ INIT is used to force a system reset and hold the microsequencer at ROM address 0000. Power sequencing is explained in detail in Chapter 2.

The system clock generation logic is represented by the blocks labeled OSC and SAC in Figure 1-22, the DPM functional block diagram. OSC represents an 18.75-MHz crystal that produces the basic time base for the system. This oscillator is physically located on the CCS module. SAC is physically located on the DPM module. The 18.75-MHz frequency is divided by 3 inside the service arbitration and control (SAC) gate array. The resultant divide-by-three output of the SAC gate array is used to produce a nonsymmetrical waveform, which is the time base for the whole system, called base clock. The duration of base clock is 160 nanoseconds. The SAC gate array produces other timing signals for use in the CPU and options. These signals are as follows.

1. B CLK is the basic clock signal. It is used to synchronize bus activities on the CMI. (Clock period is 160 ns.)

2. M CLK is the microsequencer clock and is used to load each new microinstruction. The normal duration of this clock is 320 ns (2 B CLK).

3. D CLK is the destination clock. This clock is used to write the scratchpads and registers with data at the end of the microinstruction. D clock occurs at the same rate as the M CLK and has a normal duration of 320 ns.

4. Phase clock is a symmetrical waveform with a cycle time of 320 ns. This clock is used to divide the microinstruction into two parts and test certain conditions at mid-microcycle time.

Depending on the hardware state of the CPU, the microsequencer may sometimes stretch out the clock period for M CLK, D CLK and PHASE to more than two B CLKs. Of the clock signals discussed above, all but B CLK are confined to the four CPU modules. B CLK is distributed to all system options via the CMI.

1.4.5 DPM Module Functionality

The DPM module microsequencer logic is shown on the lower half of Figure 1-22, below the WBus line. The microsequencer's function is to provide an address (control store address bus, CSA <13:0> to the CCS ROMs. This address selects the next microinstruction to be executed. The address provided on the CSA <13:0> lines may be sourced from one of several origins under control of the BUT microword field. These sources are as follows.

1. The NEXT microword field, bits <13:0> of the microinstruction, may be latched on M CLK L into latches contained on the DPM and CCS modules. This latched data is then used to provide CSA <13:0>.

2. CSA <13:0> can also be derived from instruction-dependent ROMs that are addressed by macrocode opcodes.

3. Conditional microbranching is also possible, using the microbranch multiplexer and wire-OR functions to drive CSA <5:0>.

4. Nesting of microsubroutines is possible to 15 levels, using the microstack mechanism to save calling microaddress. Return micro-orders can be specified to pop the microstack and add a positive or negative offset to the saved address.

The remainder of the DPM module is used to perform the arithmetic and logical functions of the CPU. This logic area, known as the data path, consists of the following three major subsystems.

1. Scratchpads
2. Super Rotator
3. Arithmetic Logic Unit (ALU)

Three primary buses are associated with these subsystems.

1. RBus is the register bus that interfaces the RTEMP scratchpads to the super rotator and ALU.

2. MBus interfaces the MTEMP scratchpads and the MIC interface registers to the ALU.

3. WBus conveys write data for most destination registers and scratchpads.

These are all tri-state buses.

The scratchpad section is functionally divided into four groups of 16 registers each.

1. RTEMPs for general microcode usage.
2. GPRs are macrocode general purpose registers.
3. IPRs are dedicated internal processor registers.
4. RTEMPs for general microcode usage.

Data is written into the scratchpads from the WBus on D CLK. Scratchpad data may be output to the RBus and MBus. RTEMPs 0–7 and MTEMPs 0–7 are dual ported. This means that both are always written from the WBus with the same data. Scratchpad operations are controlled primarily by the scratchpad address control (SPA) gate array and the RSRC and MSRC fields of the microword. Scratchpad outputs can go to either the super rotator or ALU.

The super rotator is shown functionally on Figure 1-22 as a barrel shifter implemented in gate arrays. Inputs to the rotator are the RBus, MBus, and the short literal field of the microword. The rotator outputs data on the SBus. The SBus is used as one of the ALU inputs. The rotator performs the following general functions.

1. Field extraction
2. Rotate and shift data on the MBus and RBus (nibble shifter)
3. Pack and unpack floating data

The final shift or rotate for rotator functions (bit shifter) is accomplished by the second level shifter, which is physically located in the ALU. The super rotator is controlled by the microword ROT field. The rotator output is supplied to the ALU subsystem.

The ALU subsystem is also implemented entirely within gate arrays. Functional blocks of the ALU shown in Figure 1-22 are all internal to the ALP gate arrays. Inputs to the ALU may be provided from two of four possible sources: the RBus, MBus, Zero, or the super rotator output. Data is input to the ALU through the A and B multiplexers under control of the MUX field of the microword. The ALU performs binary and BCD arithmetic functions as well as a series of logical functions. The ALU output is multiplexed to the WBus through the W MUX under control of the microword ALUOD field. The W MUX output is also provided to the D register and Q register. Both of these registers have general microcode usage and are used in multiply and divide functions.

The interval timer is implemented within a gate array and interfaces to the CPU WBus. The timer is controlled by the WCTRL field of the microword. The interval timer functions consistently with other VAX timers. The time base is provided from a crystal oscillator on the CCS module operating at 10 MHz. The crystal frequency is divided by 10 to generate the 1-MHz frequency for input to the timer. The timer itself is a 32-stage binary counter loaded with 2's complement of the desired interval in microseconds. When the counter overflows, a macro-level interrupt occurs. The timer is used by operating system software for scheduling and timing operations.

1.4.6 CPU Control Store Introduction

Figure 1-23 is a block diagram of the CCS control store. It is arranged in six 1K banks of 80 bits. There is circuitry to test the control store address for access to the unassigned regions and disable the address lines. A bank select decoder enables one of the six banks by decoding the CS ADD <12:10> lines to produce the bank select enable signal and allow the PROM data to go to the DPM module to be latched. Once the control store data is latched, the data is checked for correct data parity. The WCS also attaches to this module and is similar in design.

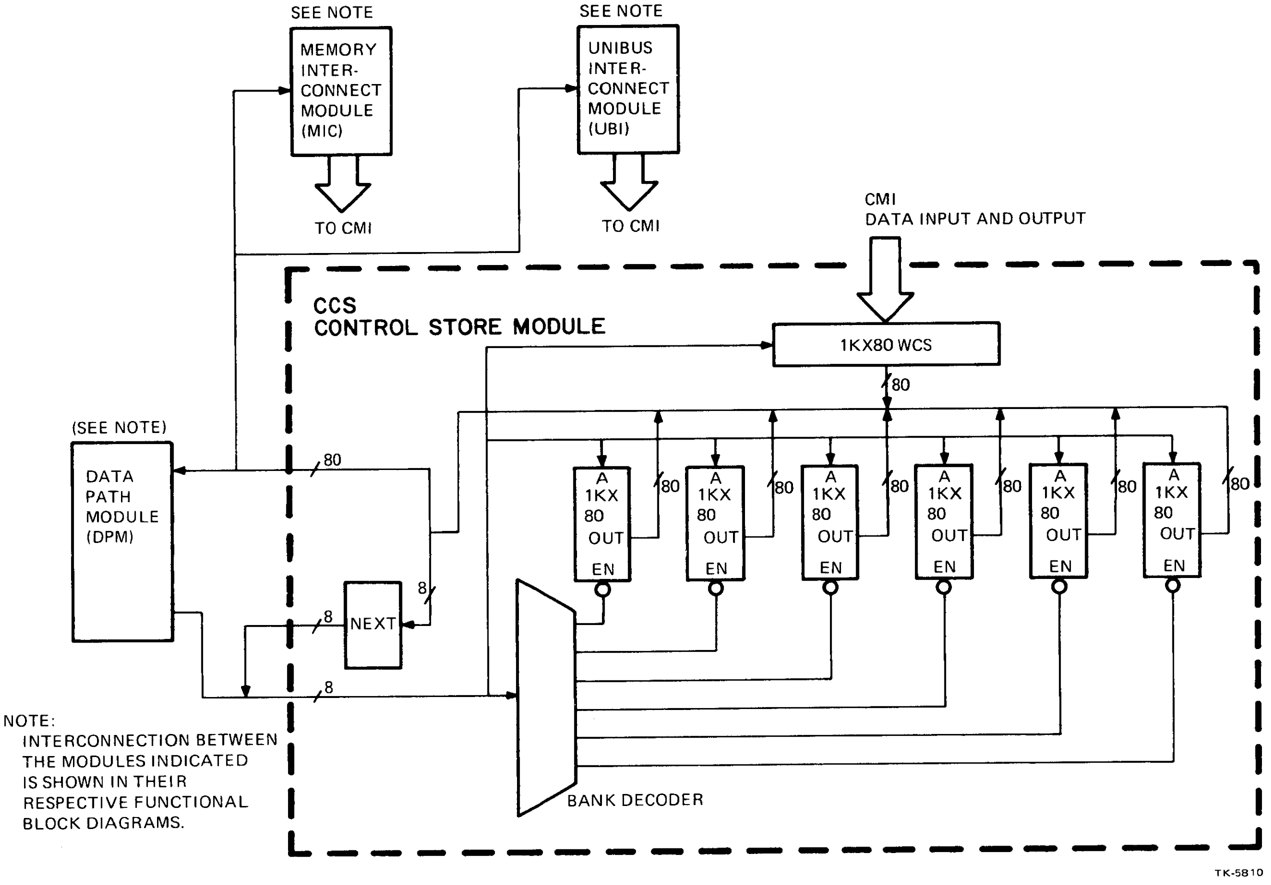

SEE NOTE
MEMORY INTER-CONNECT MODULE (MIC)
TO CMI
SEE NOTE
UNIBUS INTER-CONNECT MODULE (UBI)
TO CMI
CMI DATA INPUT AND OUTPUT
CCS CONTROL STORE MODULE
1KX80 WCS
80
(SEE NOTE)
DATA PATH MODULE (DPM)
80
8
NEXT
8
8
A 1KX 80 OUT EN
80
BANK DECODER
NOTE: INTERCONNECTION BETWEEN THE MODULES INDICATED IS SHOWN IN THEIR RESPECTIVE FUNCTIONAL BLOCK DIAGRAMS.
TK-5810

Figure 1-23 Control Store Module Functional Block Diagram

1.4.7 Memory Interface and Control (MIC) Functionality

The broad functionality of the MIC module is to interface the processor WBus and MBus with the CPU/memory interconnect (CMI). The MIC module consists of four functional sections.

1. Address control (ADD)
2. Translation buffer (TB)
3. Cache memory (Cache)
4. Memory data routing and alignment (MDR)

Memory address control functions are performed by four 8-bit ADD gate array chips (ADD section of Figure 1-24). Each chip processes one byte of an address longword from the WBus. The ADD section contains program counter (PC), virtual address (VA), and associated registers, plus adder and multiplexer circuits for address manipulation. The PC and VA registers hold addresses for operand and instruction stream references. The desired address source is multiplexed through the MA multiplexer to the MA register. Physical address information is directed to the MDR and virtual address information to the TB, on the memory address (MAD) lines. It should be noted that the ADD section is almost entirely under control of the WCTRL microword field.

The translation buffer (TB) is used to store previously translated virtual addresses. It consists of a 2 × 256 location two-way associative cache. The TB operates in conjunction with memory management microroutines that calculate physical addresses for any virtual address and then store the translated page frame number (PFN) in the translation buffer. The PFN is output to the 24-bit physical address bus (PA). The PA bus addresses the data cache and the main memory. Included in the TB is parity generation and checking logic. TB parity errors can be isolated to group tab or data storage from the machine check logout.

The data cache is used for both I-Stream and operand fetches (I-Stream data is also buffered in the XB). It consists of a 1K × 14 bit cache tag store, A=B address comparitor, 1K × 36 cache data store, and parity generation and checking logic. The cache is used for direct mapping of up to 4K bytes of data. This increases system operation speed by decreasing memory cycle time. The 1K × 14 bit cache tag store holds up to 1K 12-bit address plus parity and valid bit. The cache data store holds up to 1K × 32 bits of data plus four parity bits. Address input to the data cache is accomplished via the PA bus. Data input is via the data bus.

In general, operation of the cache is as follows. During a microinstruction memory reference, if the address on the PA bus is identical to an address stored in the cache tag store, a hit occurs. This is achieved by the A=B comparitor which looks at both the cache tag store output and the PA bus. For a hit, EN CACHE goes active and allows cache data onto the data bus. This data is routed to the operand rotator (OP ROT), which aligns it according to VA bits <1:0>. The OP ROT output is placed in MDR1, which is the interface to the MBus. When a cache miss occurs, data is placed in MDR1 from memory and the cache is updated simultaneously. The data cache can be invalidated from CMI when an I/O device modifies a memory location.

Memory data routing and alignment is performed by the OP ROT, XB and XB ROT logic. This logic is contained in eight 4-bit gate array chips. Each of these chips processes one bit per byte of data or address. This logic is used to interface the CMI to the DBus, MA bus, and PA bus. The XB contains two longword buffers that can be loaded from cache or through the CACHE INV ADD latch from memory. I-Stream prefetches are used to load the XB from memory. I-Stream prefetch is initiated by loading the PC and is completely transparent to the microcode. I-Stream data from the XB rotator can be sourced to both the MBus and the XB <15:0> bus.

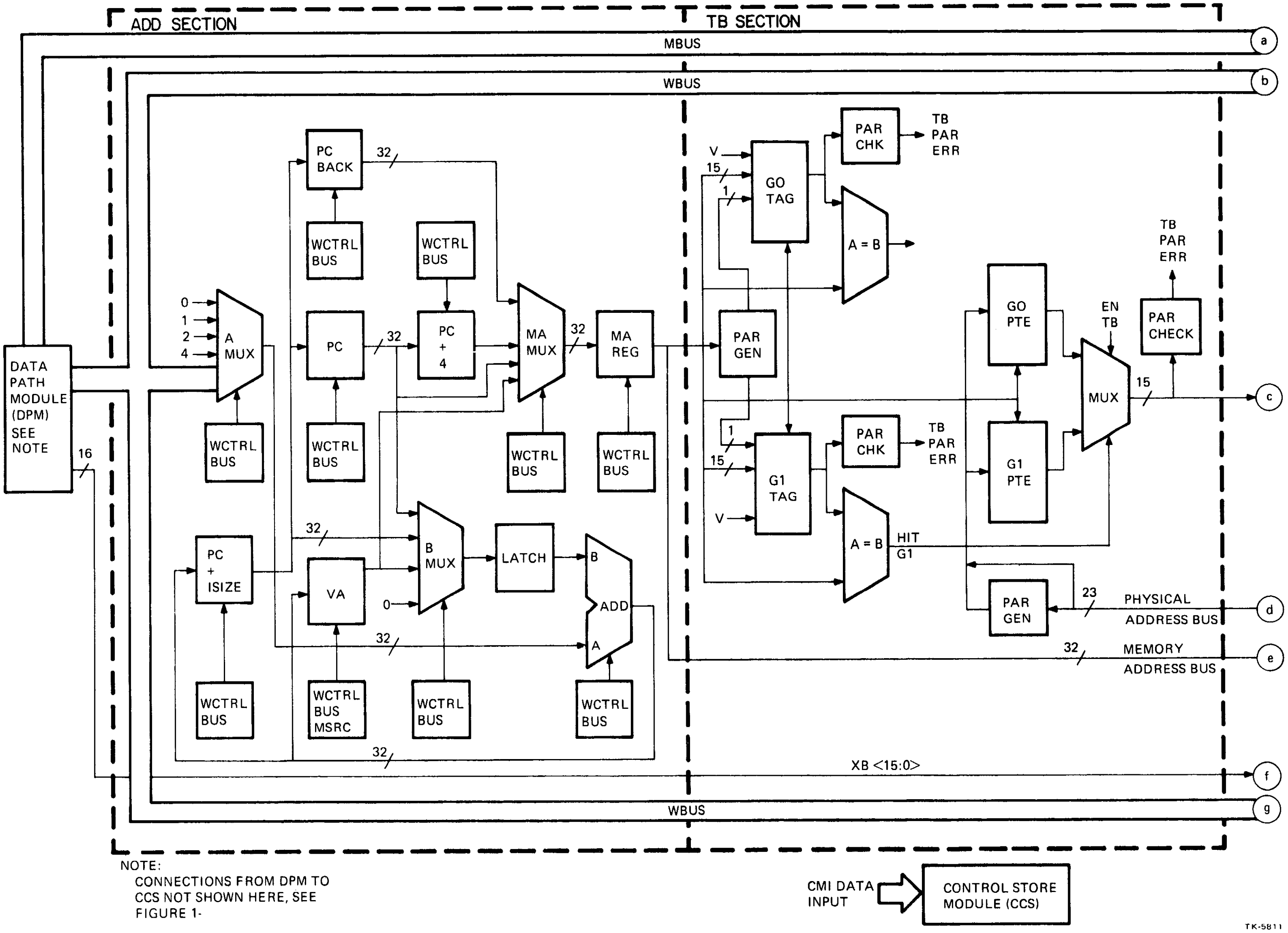


Figure 1-24 Memory Interconnect Module Functional Block Diagram (Sheet 1 of 2)

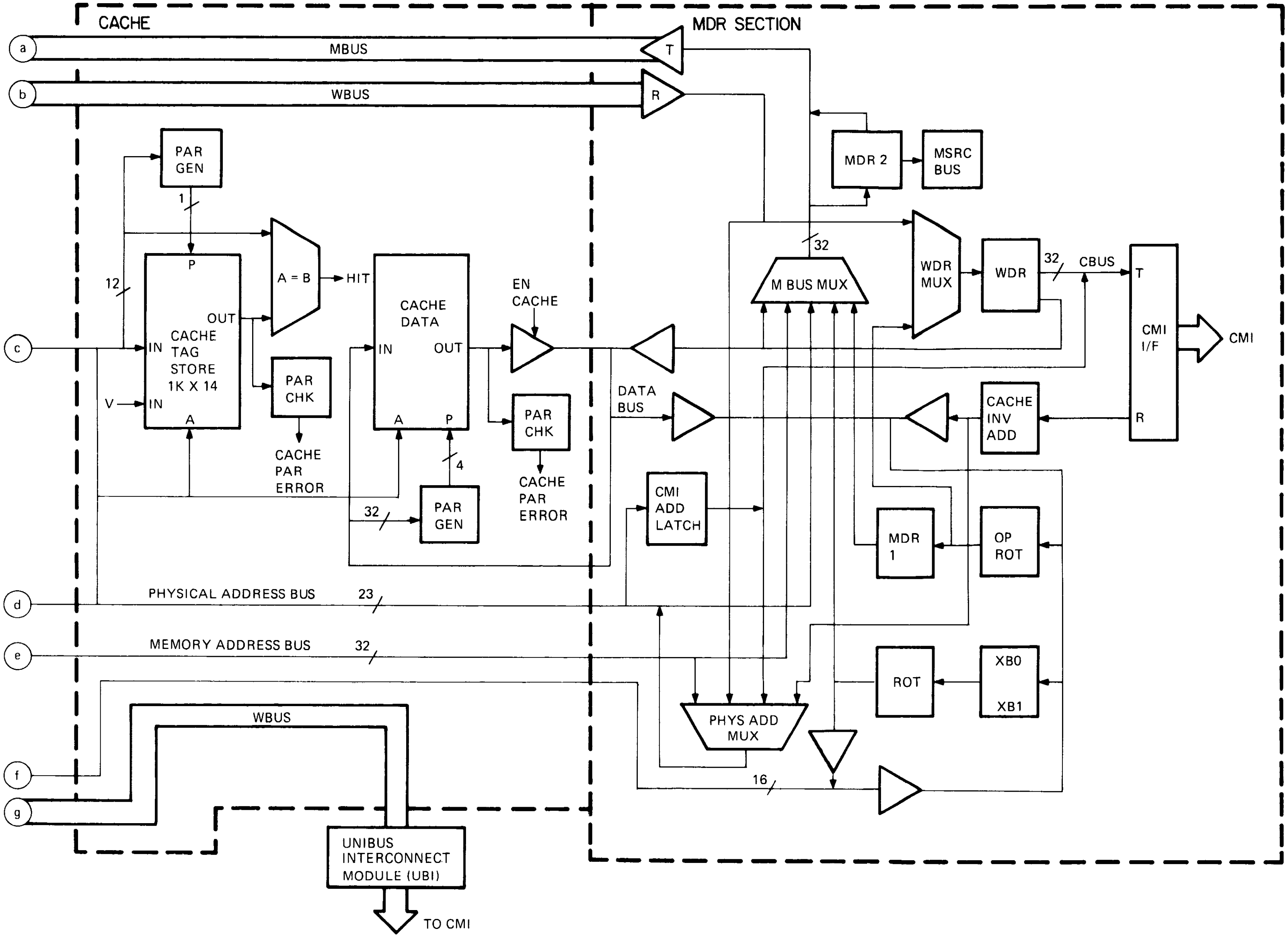


Figure 1-24 Memory Interconnect Module Functional Block Diagram (Sheet 2 of 2)

1.4.8 Unibus Interface and Miscellaneous Hardware

Figure 1-25 is a functional block diagram of the logic contained on the Unibus interconnect module. This logic functions as five separate subsystems.

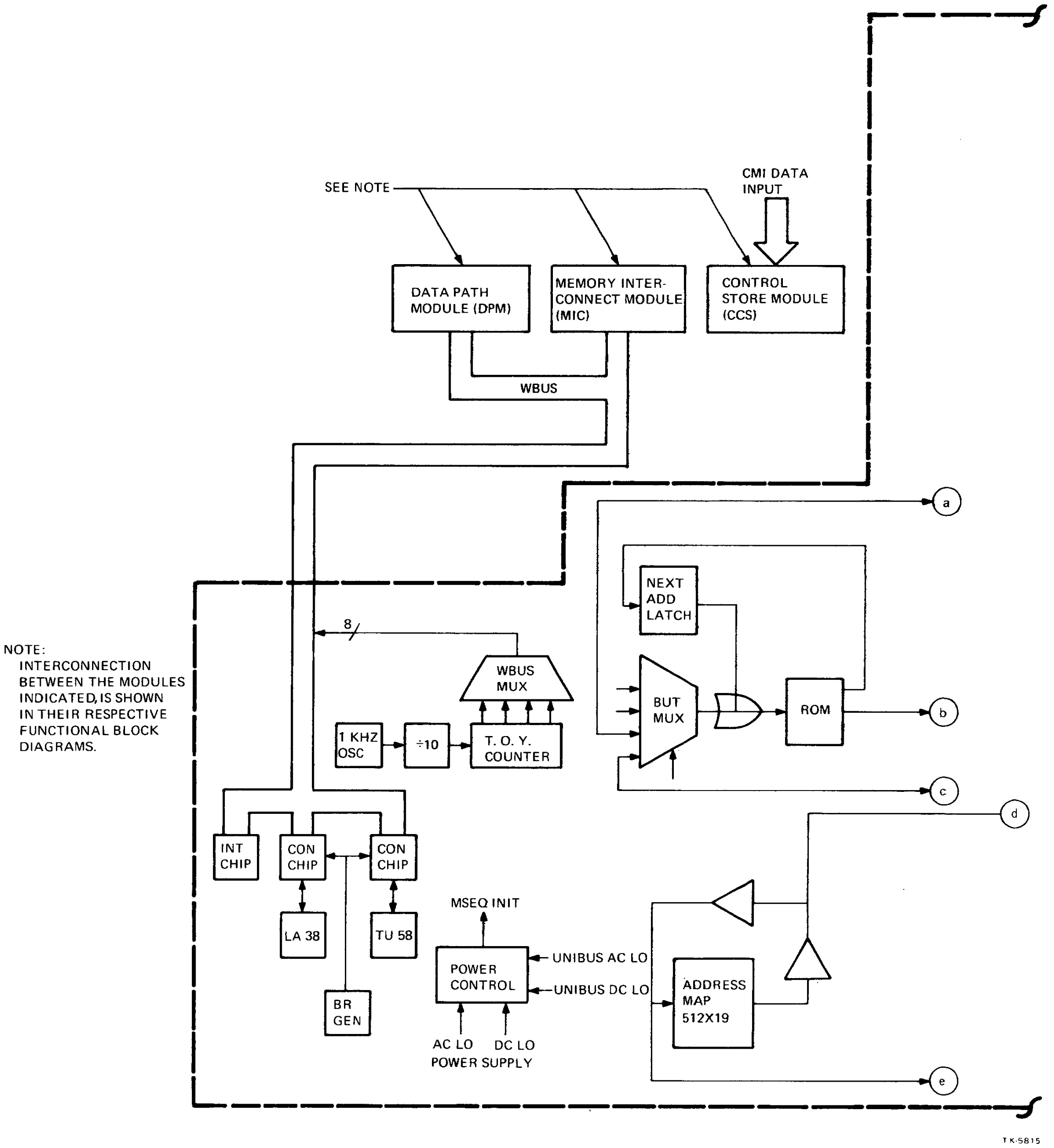


Figure 1-25 Unibus Interconnect Module Functional Block Diagram (Sheet 1 of 2)

1. Console interface
2. TU58 interface
3. Interrupt logic
4. Unibus interface
5. Time-of-year (TOY) clock

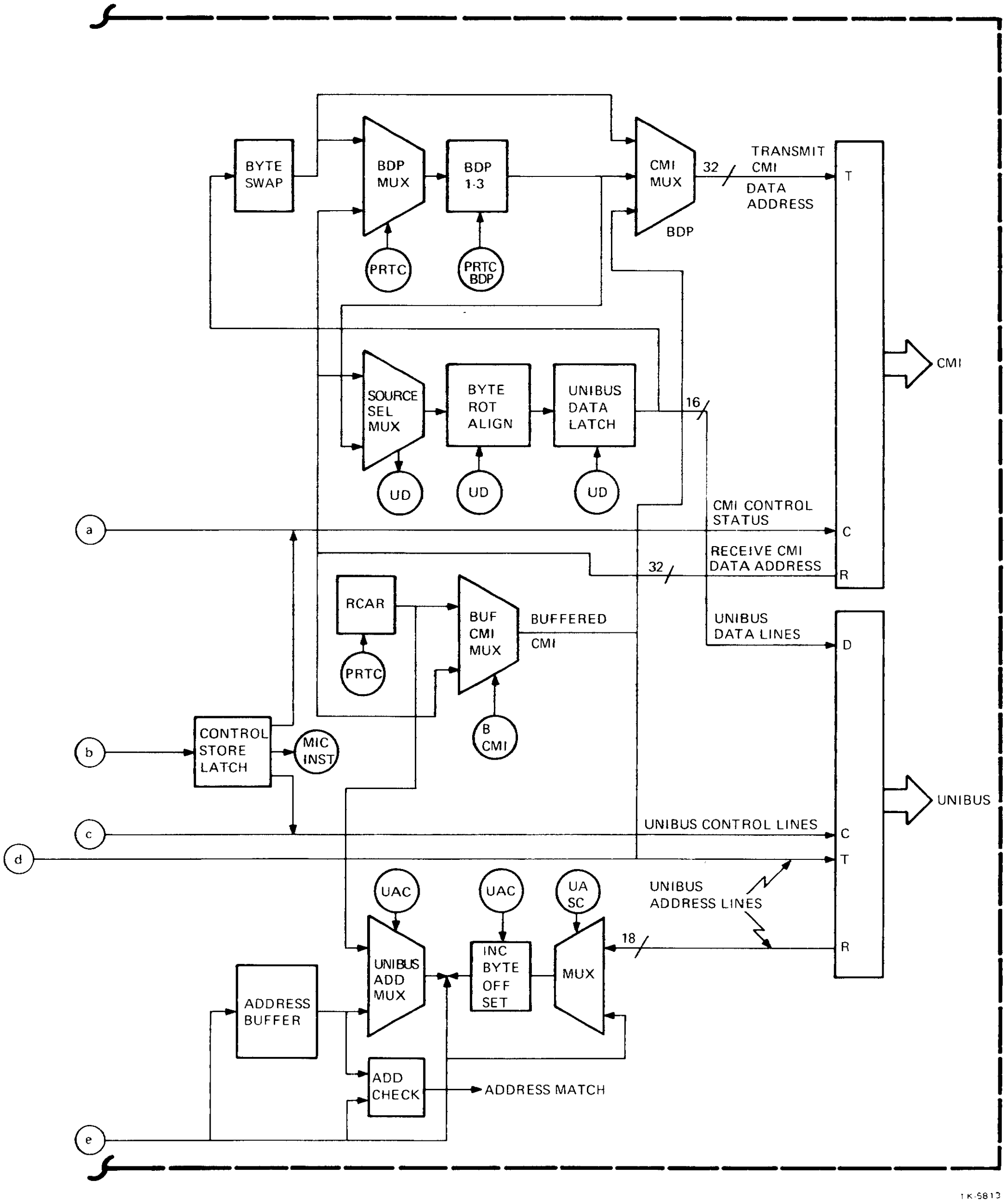


Figure 1-25 Unibus Interconnect Module Functional Block Diagram (Sheet 2 of 2)

1.4.8.1 Console Interface (CON) Overview – Interfacing between the console and CPU is provided by a CON gate array chip. This chip functions as an asynchronous serial line EIA interface. The console section of the microcode provides control for data exchanges between the console registers and the CPU (IPRs and GPRs) and memory. This functionality permits the console user to perform examine/deposit operations to certain CPU registers and to selected memory locations. The primary path for data exchanges between the console CON chip and the CPU is the WBus. As mentioned previously, the WBus is under control of the WCTRL field of the microword. The console interface operates at interrupt priority level 14 (IPL 14).

1.4.8.2 TU58 Interface – With few exceptions, the TU58 interface is identical to the console interface. A CON gate array chip functions as interface between the CPU and TU58. This chip is identical to and interchangeable with the one used as a console interface. This chip functions as an asynchronous serial line EIA interface. The console section of the microcode provides control for data exchanges between the TU58 interface and the CPU. The TU58 is accessed via IPRs at the macrocode level and requires macrocode drivers. The primary data path for data exchanges between the TU58 interface and CPU is the WBus. The TU58 interface operates at interrupt priority level 17 (IPL 17).

1.4.8.3 Interrupt Logic Introduction – The INT chip resides on the UBI module, as shown in Figure 1-25. Figure 1-26 provides a more detailed view of the INT chip, which handles all system interrupts, both hardware and software. The sources of interrupt requests are shown in Figure 1-26. More specifically, the INT chip can perform the following functions.

1. The INT chip stores three sections of the processor status longword: IPL (interrupt priority level), IS (interrupt stack flag), and CUR MODE (current mode). Also stored in INT is AST (asynchronous system trap level). The INT chip saves this data and returns it to the system on the WBus under control of the microword WCTRL <5:0> control field.

2. Another function of INT is receiving and storing the value of HSIPR (highest software interrupt pending request). This data is used in interrupt arbitration. The WBus, under control of WCTRL <5:0>, is used to receive this information.

3. The INT chip may place various data onto the MICROVECTOR <2:0> H lines. These lines are used to identify the highest priority interrupt present. They represent the three least-significant bits of microaddress to be supplied to the CPU control store (CCS) when servicing an interrupt (details provided in Paragraph 2.9.1).

4. The INT chip performs REI (return from exception or interrupt, check calculations). Here, the REI instruction uses IS, CUR MODE and IPL data.

5. The INT chip accomplishes arbitration of all interrupt requests, and encoding of the highest priority pending interrupt.

6. The INT chip handles Unibus arbitration within the group of bus request (BR) devices and issues highest priority bus grant (HPBG) to the Unibus interface. The SBR <7:4> lines convey bus requests to the INT chip from Unibus devices.

   The INT chip assigns an IPL level to the incoming SBR request as follows.

SBR				**IPL No.**
7	**6**	**5**	**4**	
1	0	0	0	17
0	1	0	0	16
0	0	1	0	15
0	0	0	1	14

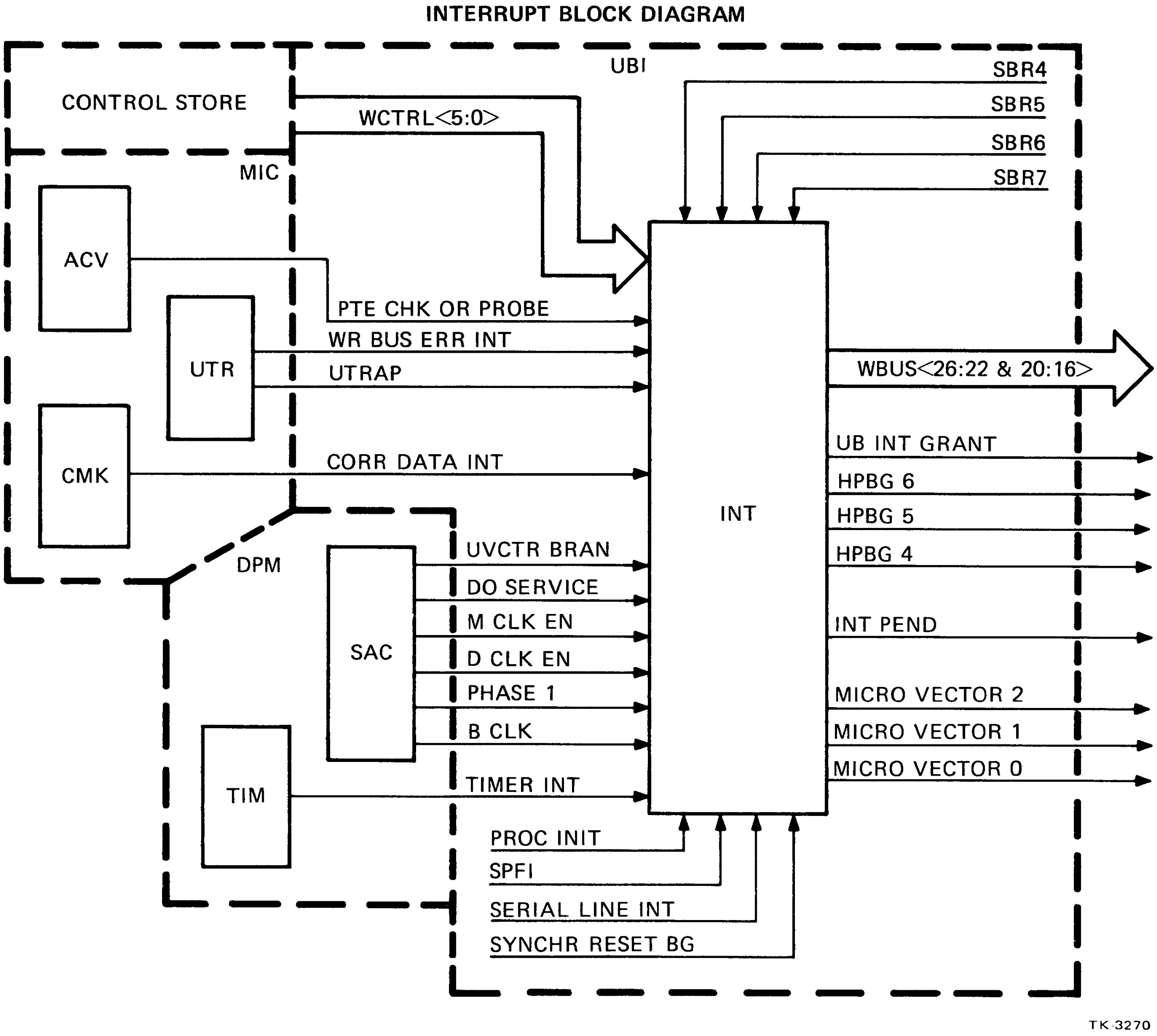


Figure 1-26 Interrupt Block Diagram

Note that the SBR lines are seen as interrupt inputs by the INT. Under control of the WCTRL <5:0> microcode field, the INT chip can issue a bus grant based on the IPL level of the bus request received previously. Bus grants to the Unibus are issued on the SBR 7 and HPGB <6:4> lines. Only one of these lines may be asserted at any one time.

More detailed information on the INT chip may be found in Paragraph 2.9.

1.4.8.4 Unibus Interface Overview – The Unibus to CMI interface section of the UBI module adheres to both CMI and Unibus protocols while monitoring and coordinating data transactions between these two buses. B CLK L, supplied by the CPU, is used for all timing functions and synchronization. Figure 1-25 shows all the functional blocks that make up the Unibus interface function of the UBI module: the Unibus data path (UDP), address map (MAP), Unibus control (UCN), UBI control store and Unibus arbitrator.

Unibus Data Paths (UDP) – The Unibus data path (UDP) section consists of four identical gate array UDP chips. Each chip processes two bits of each Unibus data/address and eight bits of CMI data/address. Note that Unibus address bits 0 and 1 do not go to the UDP, but rather to the UCN chip. The UDP section provides the necessary registers, gating, and alignment for data transfers between the Unibus, which is 16 bits wide, and the CMI, which is 32 bits wide. The UDP contains one direct data path (DDP) gating, and three buffered data path (BDP) registers and buffered address (BAR) registers. It also contains a SKEW register to temporarily latch address or data information received from the CMI (CMI latch), and the received CMI address register (RCAR) which stores CMI specified addresses for transfer to the Unibus address lines or to logic within the UBI.

Address Map (MAP) – The address map (MAP) (Figures 1-27 and 1-28) is the facility by which Unibus devices that make sequential DMA transfers are able to access noncontiguous pages of main memory. The 512 × 19-bit RAM is loaded by the software with the page frame numbers of main memory locations to be accessed, plus validity, offset, and data path information. Unibus NPR transfers take place on the direct data path or one of the three buffered data paths as designated by the map entry.

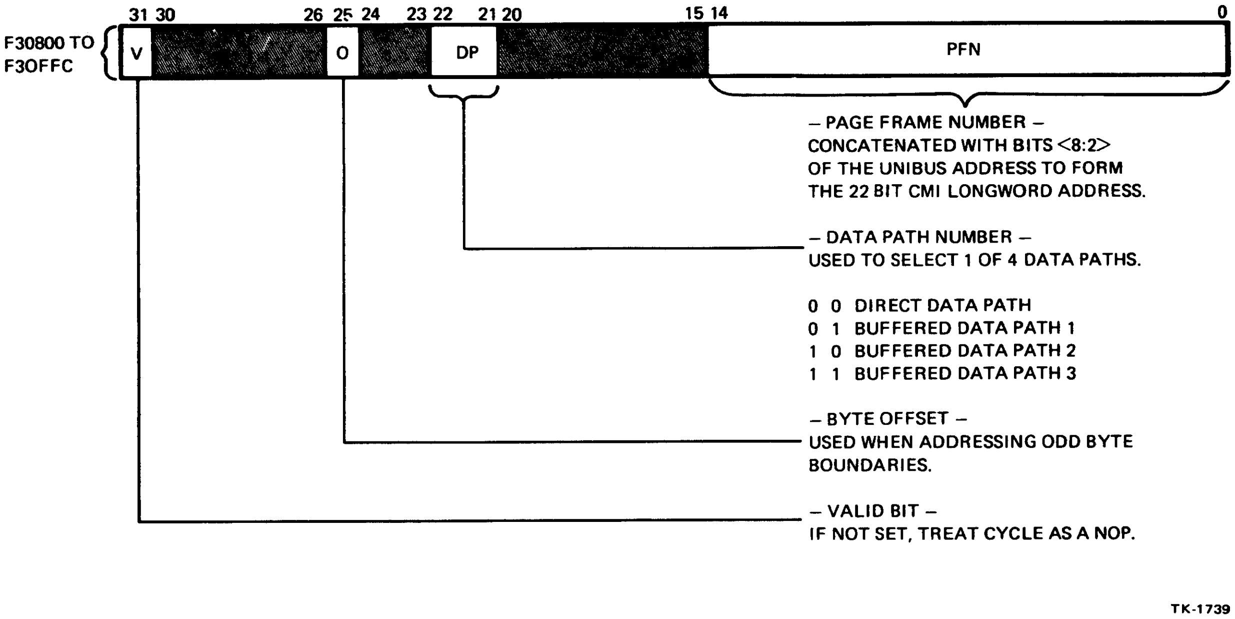


Figure 1-27 CMI Map Data Fields

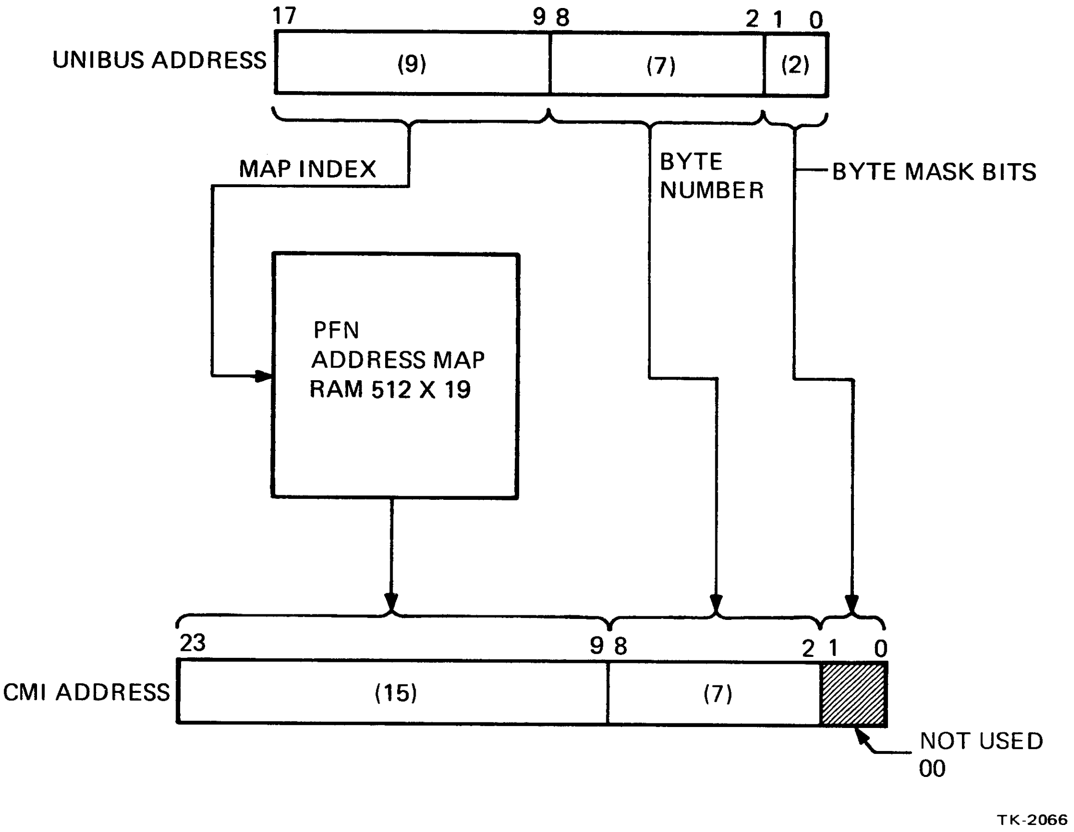


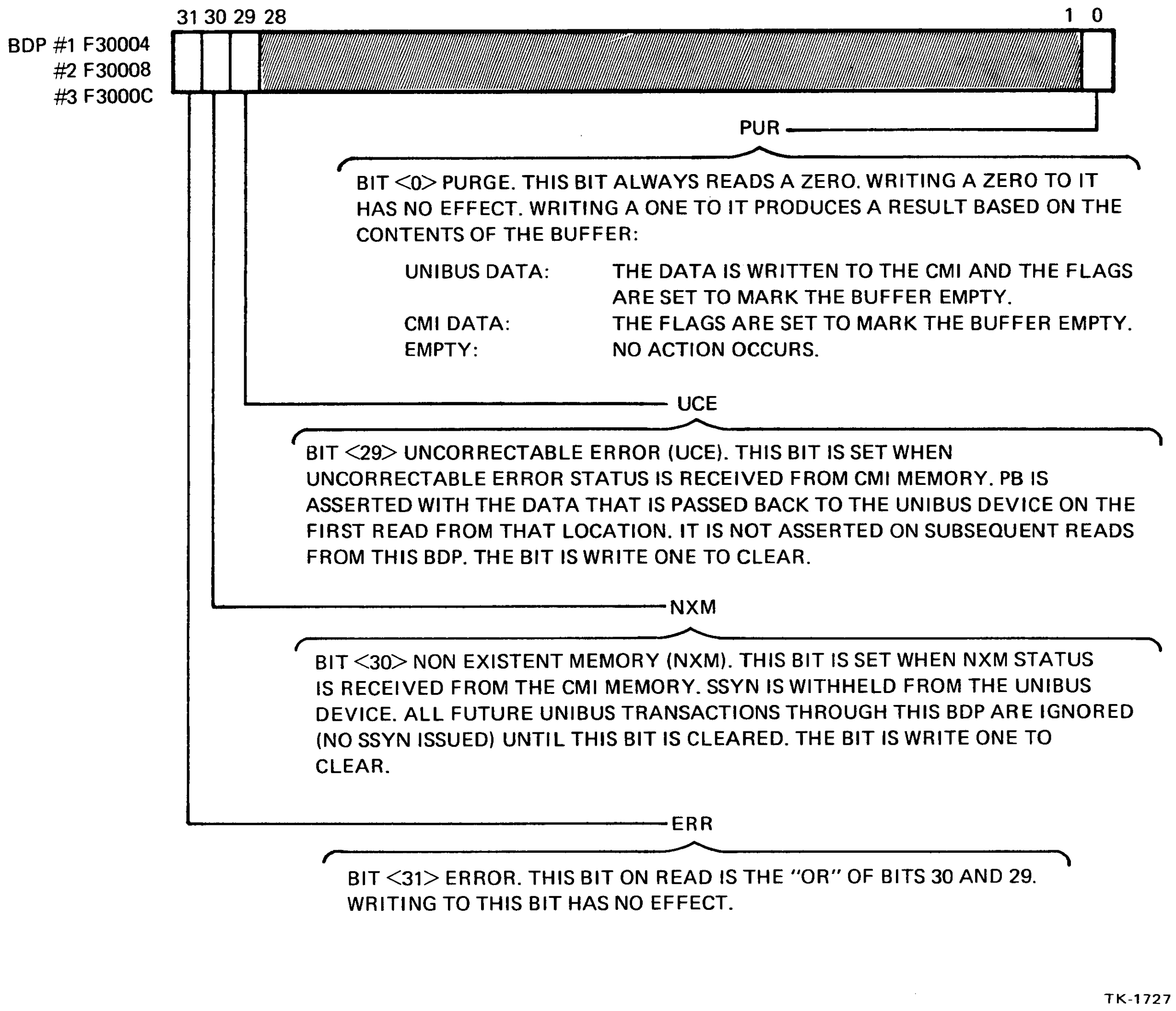
Figure 1-28 Unibus to CMI Address Translation

Unibus Control (UCN) – The UCN section, which is contained on a single gate array chip, accomplishes control signal interpretations for transactions between the CMI and the Unibus (Figures 1-29 and 1-30). The UCN contains error and byte flags for each of the three buffered data paths. The byte flags are enabled to determine which bytes are valid for transfer to main memory. The error flags store nonexistent memory and uncorrectable error status. The UCN generates the CMI byte mask and function codes for Unibus transactions to main memory. In addition, it contains the slave control logic that provides for access to MAP registers, buffered data path control/status registers and buffered data path diagnostic status registers.

UBI Control Store – The UBI control store consists of a 256 × 24-bit PROM array with outputs clocked to a buffer register. In conjunction with BUT field gating in the UCN, it performs microsequences that execute and direct UBI operations. Timing is provided by B CLK L, which is supplied by the CPU. The UBI microword generates control signals for the Unibus, the MAP, and for priority arbitration on the CMI. It also generates fields that determine address and data gating through the UDP.

NOTE

The UBI control store is resident on the UBI module and should not be confused with the control stores of the CPU.



BIT <0> PURGE. THIS BIT ALWAYS READS A ZERO. WRITING A ZERO TO IT HAS NO EFFECT. WRITING A ONE TO IT PRODUCES A RESULT BASED ON THE CONTENTS OF THE BUFFER:

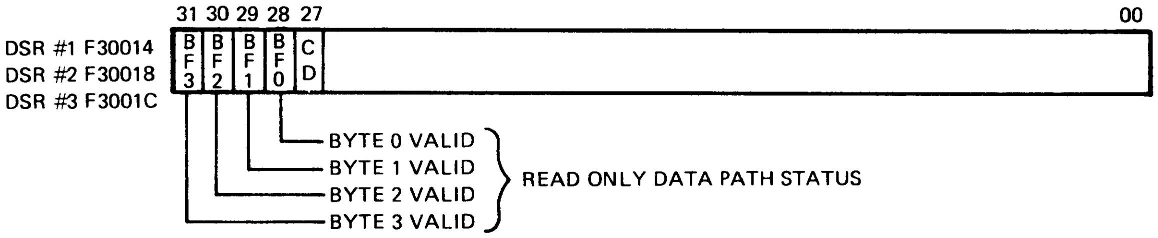
UNIBUS DATA:	THE DATA IS WRITTEN TO THE CMI AND THE FLAGS ARE SET TO MARK THE BUFFER EMPTY.
CMI DATA:	THE FLAGS ARE SET TO MARK THE BUFFER EMPTY.
EMPTY:	NO ACTION OCCURS.

BIT <29> UNCORRECTABLE ERROR (UCE). THIS BIT IS SET WHEN UNCORRECTABLE ERROR STATUS IS RECEIVED FROM CMI MEMORY. PB IS ASSERTED WITH THE DATA THAT IS PASSED BACK TO THE UNIBUS DEVICE ON THE FIRST READ FROM THAT LOCATION. IT IS NOT ASSERTED ON SUBSEQUENT READS FROM THIS BDP. THE BIT IS WRITE ONE TO CLEAR.

BIT <30> NON EXISTENT MEMORY (NXM). THIS BIT IS SET WHEN NXM STATUS IS RECEIVED FROM THE CMI MEMORY. SSYN IS WITHHELD FROM THE UNIBUS DEVICE. ALL FUTURE UNIBUS TRANSACTIONS THROUGH THIS BDP ARE IGNORED (NO SSYN ISSUED) UNTIL THIS BIT IS CLEARED. THE BIT IS WRITE ONE TO CLEAR.

BIT <31> ERROR. THIS BIT ON READ IS THE "OR" OF BITS 30 AND 29. WRITING TO THIS BIT HAS NO EFFECT.

Figure 1-29 BDP Control and Status Register



NOTE 1: THERE ARE FIVE FLAGS THAT KEEP TRACK OF THE DATA IN THE DATA BUFFER, NAMED CD AND BF3 THROUGH BF0. IF CD = 1, THEN THE BUFFER HAS FOUR BYTES OF DATA FROM THE CMI AND BF3 THROUGH BF0 ARE ALWAYS 0. IF CD = 0, THEN BF3 THROUGH BF0 INDICATE WHICH BYTES IN THE DATA BUFFER HAVE VALID UNIBUS DATA. IF THEY ARE ALL 0, THEN THE BUFFER IS CONSIDERED EMPTY.

NOTE 2: THIS IS A READ ONLY REGISTER THAT ALLOWS ONE TO CHECK THE FLAG BITS ASSOCIATED WITH EACH BDP. IT IS INTENDED ONLY FOR POSSIBLE DIAGNOSTIC USE AND NO REFERENCE TO IT IS REQUIRED FOR NORMAL USE OF THE BDP'S.

TK-1726

Figure 1-30 Diagnostic Status Register

Unibus Arbitrator – The Unibus arbitrator selects the next Unibus master, and generates the grant signal in response to an NPR or BR request. The CPU gains access to the Unibus through the arbitrator logic. BBSY is asserted when the CPU enables the CMI address longword for access to a Unibus device. Bus grant (BG) is issued after the processor determines that the BR request level is greater that the current PSL IPL level.

Unibus Initialize – Initialization logic monitors the ACLO and DCLO signals on the Unibus. DCLO initiates a process microsequence to discontinue operations and assert the initialize level on the Unibus. This also clears logic and devices on the Unibus during a power-up sequence. An ACLO condition asserts the sync power-fail interrupt (SPFI) signal to the INT chip. This generates a power fail interrupt to prepare for loss of power.

1.4.8.5 Time-of-Year Clock (TOY) and TOY Power Control – The TOY clock (Figure 1-25) and its power control are resident on the UBI module. The TOY clock is a binary 32-stage counter. The time base for the TOY is a precision 1-KHz crystal oscillator. The 1 KHz is divided by 10 in order to provide an increment pulse every 10 milliseconds. At this rate, counter overflow occurs in 1.3 years.

The counter is implemented in two parts. The first is a base time scratchpad that stores the time entered by the VMS system service. The second is a binary counter that is initially cleared and then maintains an offset from the base time. Software access to the TOY clock is achieved through the time-of-day register (TODR) (IPR No. 1B). TODR may be accessed in the console mode with examine or deposit commands. Under the VAX operating system, TODR is accessed with MTPR and MFPR functions.

Power backup to the counter circuitry is supplied by four 1.25-Vdc nickel-cadmium batteries. These batteries will sustain counter operation, and accuracy, for 100 hours under system power off or fail conditions.

1.4.9 Unibus Exerciser/Terminator (UET)

The M9313 UET module terminates the open collector lines of the Unibus. It also contains registers and features that allow the diagnostic software to perform checks and exercise Unibus functions. (See Figures 1-30, 1-31, and 1-32.) A Unibus device need not be present to make use of these features. The registers contained on the UET may be referenced using console examine and deposit commands. Some examples of these operations are as follows.

Console Prompt	Command	Operation
>>>	D FFF 460 0	; Address 0 in UET BAR
>>>	E FFF 462 1234	; Check UET DR
>>>	D FFF 464 1	; NPR GO, DATI Cycle

It should be noted that the M9302 terminator may be used on the VAX-11/750 system. However, the UBI macrodiagnostic will not run when this terminator is used.

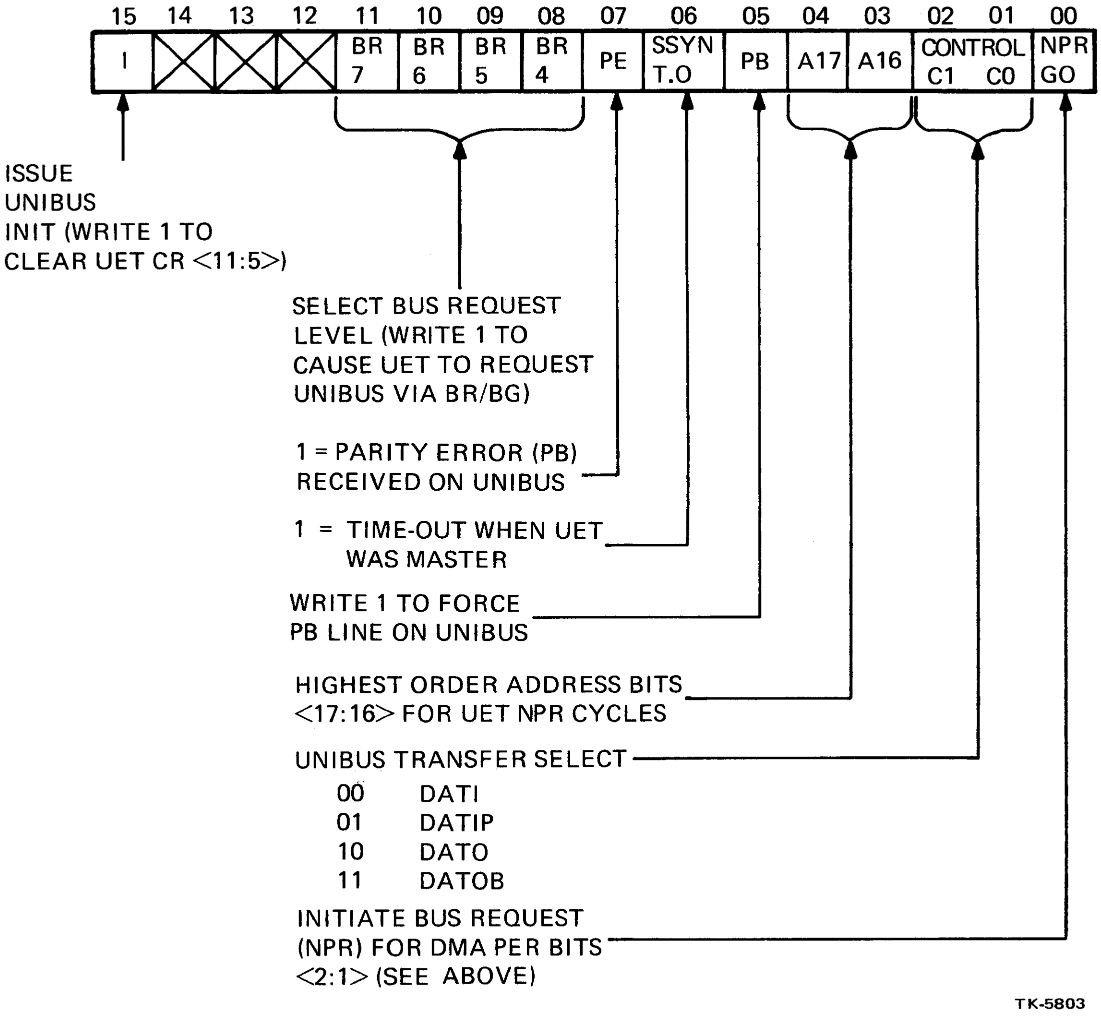

15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
I
BR 7
BR 6
BR 5
BR 4
PE
SSYN T.O
PB
A17
A16
CONTROL C1 C0
NPR GO
ISSUE UNIBUS INIT (WRITE 1 TO CLEAR UET CR <11:5>)
SELECT BUS REQUEST LEVEL (WRITE 1 TO CAUSE UET TO REQUEST UNIBUS VIA BR/BG)
1 = PARITY ERROR (PB) RECEIVED ON UNIBUS
1 = TIME-OUT WHEN UET WAS MASTER
WRITE 1 TO FORCE PB LINE ON UNIBUS
HIGHEST ORDER ADDRESS BITS <17:16> FOR UET NPR CYCLES
UNIBUS TRANSFER SELECT
00 DATI
01 DATIP
10 DATO
11 DATOB
INITIATE BUS REQUEST (NPR) FOR DMA PER BITS <2:1> (SEE ABOVE)
TK-5803

Figure 1-31 UET Control/Status Register

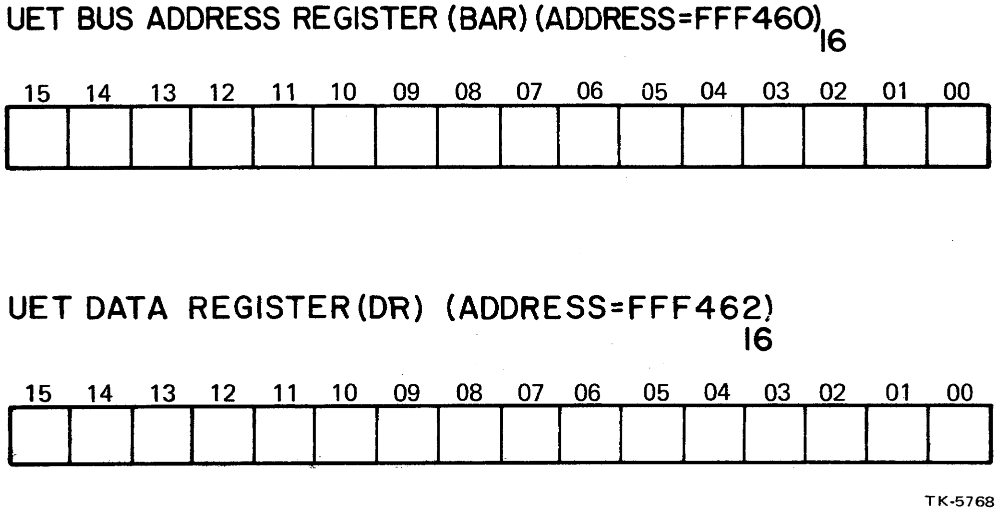


Figure 1-32 Unibus Exerciser/Terminator BAR and DR Register

1.5 VAX-11/750 DIAGNOSTICS

Diagnostics for the VAX-11/750 are broken down into five levels. Four of these levels are numbered 1 through 4. The remaining level is microdiagnostics.

Level **Description**

1. These diagnostics run under the VMS operating system without using the diagnostic supervisor; e.g., line printer diagnostic.

2. These diagnostics run under the diagnostic supervisor while the VMS system is still operating; e.g., reliability and acceptance tests.

3. These diagnostics run under the diagnostic supervisor, which must be running stand-alone and the VMS system not running; e.g., UBI diagnostic.

4. These diagnostics run stand-alone without the diagnostic supervisor or VMS operating; e.g., hardcore instruction.

MICROs The following diagnostics are loaded from the TU58 and run from the RDM RAM memory:

DPM microdiagnostic (data path)
MIC microdiagnostic (memory interconnect)

NOTE

Another diagnostic, named Micro-Verify, is resident in the machine CCS microcode. This diagnostic is run each time the front panel initialize button is indexed. Micro-Verify is a basic sanity check of the data path and MIC module. Micro-Verify is run before any other machine operation is performed.

Diagnostics run at micro-level:

VAX-11/750 Micro Data Path (DPM)

ECKAA.EXE	Microdiagnostic Monitor (MM)
ECKAB.EXE	Microdiagnostic DPM

VAX-11/750 Micro Memory Interconnect (MIC)

ECKAA.EXE	MM
ECKAC.EXE	Microdiagnostic MIC

Diagnostics run at levels other than micro:

VAX-11/750 Cache/TB;Memory;Cluster Excerciser

ECKAL.EXE	Cache/TB (Bootable;level 4)
ECKAM.EXE	Memory Diagnostic (level 3)
ECKAX.EXE	Cluster Excerciser (level 3)

VAX-11/750 DW750 (UBI);Diagnostic Supervisor

ESSAA.EXE	Diagnostic Supervisor
ECCBA.EXE	Diagnostic (level 3)

VAX-11/750 Hardcore Instruction

EVKAA.EXE	Hardcore Instruction (Bootable;level 4)

VAX-11 Instruction Tests

EVKAB.EXE	VAX Architectural Inst. (level 2 and 3)
EVKAC.EXE	VAX Floating-Point Inst. (level 3)
EVKAD.EXE	VAX Compatibility Mode Inst. (level 3)
EVKAE.EXE	VAX Privileged Architectural Inst. (level 3)

The following diagnostics are used to test options available on the VAX-11/750. These are the same diagnostics as are run on the VAX-11/780.

VAX CR/DISK User Mode

EVQDR	VAX Loadable Driver for RMOX/RM 80
EVQDM	VAX Loadable Driver for RK611-RK06/07
EVQDL	VAX Loadable Driver for RL11-RL01/02
EVABA	VAX CR11 CR Diagnostic
EVRAA	VAX RP/RK/RM/RX TU58 Reliability
EVRACX	VAX Disk Formatter

KMC11/DMC11/DZ11

EVDMA	VAX M8203 Repair Level
EVDXA	VAX COMM IOP Repair Level
EVDAA	VAX DZ11 8-Line ASYNC MUX

RK611 Diagnostics No. 1

EVREA	VAX RK611 Diagnostic, Part A
EVREB	VAX RK611 Diagnostic, Part B

RK611 Diagnostics No. 2

EVREC	VAX RK611 Diagnostic, Part C
EVRED	VAX RK611 Diagnostic, Part D
EVREE	VAX RK611 Diagnostic, Part E

RK611 Diagnostics No. 3

EVREF	VAX RK06/07 Drive Function Test, Part 1
EVREG	VAX RK06/07 Drive Function Test, Part 2

RM03/RM05

EVRDA	VAX RM03/RM05/RM80 Diskless
EVRDB	VAX RM03/RM05 Functional Test

TS11 Diagnostics

EVQTS	VAX Loadable Driver For TS11/TS04
EVMAA	VAX TM03/TE16/TU45
EVMAD	VAX TS11 Subsystem Repair

RL02/RM80 Diagnostics

EVRFA	VAX RL02 Subsystem Functional Diagnostics
EVRGA	VAX RM80 Formatter
EVRGB	VAX RM80 Functional Diagnostic

CHAPTER 2
THEORY OF OPERATION

2.1 CENTRAL PROCESSOR TIMING

This paragraph describes the VAX-11/750 central processor timing. Paragraph 2.1.1 provides a functional description of the power-up and power-down hardware sequencing. Paragraph 2.1.2 describes the generation of the CPU main timing signals.

2.1.1 CPU Power Sequencing

The hardware condition of the VAX-11/750 processor must be initialized to a defined state after power has been applied and stabilized. The following discussion is related to the schematic diagrams of the UBI module, the VAX-11/750 memory controller module (CMC), and the remote diagnosis module (RDM). The following schematics are referenced in the discussion that follows.

UBI Unibus Interface Module (D CS L0004-0-1 through D CS L0004-0-20 Rev C)

RDM Remote Diagnosis Module L0006 (D-CS-RDM01 TO RDM26)

MIC Memory Interface Module (D CS L0003-0-1 through D BD L0003-0-23 Rev B)

DPM Data Path Module (D CS L0002-0-1 through D BD L0002-0-26 Rev B)

CCS CPU Control Store (D CS L0005-0-1 through D CS L0005-0-16 Rev C)

CMC VAX-11/750 Memory Controller (D-CS-L0011-0-1 CMCA through CMCV Rev C)

Power sequencing timing diagrams are included in this document and related to the text and schematic diagrams. The operations discussed in Paragraphs 2.1.1.1–2.1.1.5 are as follows.

Power-up sequence
Power-down sequence
Power sequencing using INIT pushbutton
Power sequencing with RDM
Time-of-day battery power control circuit

2.1.1.1 Power-Up Sequence – The general sequence of events for the VAX-11/750 CPU power-up is similar to that of other processors. When the ac line voltage stabilizes, the power supply negates the signal ACLO L. When the dc output is reached, circuit power stabilizes and the signal DCLO L is negated.

As long as DCLO is asserted, the microsequencer is forced to the power-up location in control store, microaddress 0000, and the microcode does not execute until DCLO is negated. The CPU microcode runs through a wait period of 250 ms before any major operations are attempted. During this time the memory controller is writing zeros and the proper ECC for each location in all of the memory. The memory controller asserts ACLO until this is complete. The time required is approximately 830 ms to initialize all of memory. When the 250 ms wait interval expires, the microcode goes into a LOOP that tests ACLO L for negation. The microcode stays in this loop until ACLO L is negated.

If the RDM is present, it asserts ACLO and DCLO (under program control in the RDM) until its power-up sequence and self-verification is complete. The RDM does not receive ACLO and DCLO; it can only drive these signal lines. Once ACLO and DCLO are both negated the CPU microcode performs the Micro-Verify routine. Micro-verify tests the internal buses, data path, and prefetch mechanism. It also tests the initialization microroutine, which clears the cache, invalidates the translation buffer, and sets up the PSL. The microcode performs one of the following operations depending on the POWER ON ACTION switch located on the operator control panel.

1. Enter console mode.

2. Attempt warm restart. If restart fails, enter console mode.

3. Attempt warm restart. If restart fails, boot system in accordance with DEVICE switch.

4. Bootstrap system in accordance with DEVICE switch.

Power-Up Detailed Description – The following discussion is referenced to the UBI module schematics listed in Paragraph 2.1.1, and the timing diagrams contained in this chapter.

Refer to the UBI module schematic, pages 14 and 15. In accordance with the Unibus specification, the signal DCLO L is negated approximately 5 μs after dc voltage is applied. The power supplies drive Unibus ACLO and DCLO according to this specification. Again, ACLO L is negated when the line voltage is stable, but the memory controller holds ACLO asserted until it has written the zeros and ECC through all of memory.

Refer to the CMC module schematic, page CMCC. The internal clock logic on the CMC is designed to refresh the memory at intervals of 12.8 μs. The counter, E111 and E123, is the refresh/initialization row/column counter. At power-up, the negation of DCLO permits the counter to be incremented at a 12.8 μs rate. The initialize flip-flip, E128, is set by the first T1 clock after the negation of DCLO.

The initialize flip-flop clears when the most significant stage of the refresh/initialize counter sets. The time for this to occur is 12.8 μs × 65,536 for a total of 838 ms from DCLO negation. The initialize flip-flop drives the signal CMI ACLO which becomes Unibus ACLO.

Refer to UBI print 15. At the top center of the page is the Unibus transceiver (E105) that interfaces ACLO L and DCLO L to the UBI module from the power supplies via the Unibus. The signal RCVD DCLO H is true as long as DCLO is asserted.

After ACLO L is negated, DCLO is asserted for 5 μs. During this interval three things happen.

1. The CPU asserts Unibus BBSY L.
2. The CPU asserts Unibus INIT L.
3. The signal MSEQ INIT l is asserted – Refer to UBI print. In the lower left corner, the signal RCVD DCLO H is inverted and becomes DCLO BBSY L. This signal goes to three places. The first destination is the D-type latch at the left center of the UBI module schematic, to E132. This signal causes the latch to be cleared, which forces the signals MSEQ INIT L and INIT UB REQ H to be asserted. MSEQ INIT L holds the microsequencer logic at control store address 0000. DCLO BBSY L is also used to cause the assertion of BBSY and INIT. In the upper right corner of UBI 14, the signal DCLO BBSY L forces E119 reset, which generates the signal UB INIT H. UB INIT H goes to the Unibus transceiver on UBI 15 and drives Unibus INIT L true. DCLO BBSY L goes to E109 on the right side of UBI 13 and generates a signal called ASSERT BBSY H. This signal goes to the Unibus transceiver on UBI 15 and asserts Unibus BBSY L, preventing devices from becoming bus master.

Once DCLO L is negated, Unibus BBSY L is negated and INIT L is allowed to go away. The first microinstruction executed from control store issues a Unibus INIT micro-order in the bus field of the microword. This micro-order remains asserted during the entire 250 ms waiting period. When the signal INIT UB REQ is generated, it fires one-shot E133 (UBI 13) which generates a low pulse for 130 μs. The positive transition clocks E119 clear and the signal UB INIT H is negated. The signal INIT UB REQ L from UBI 14 goes to E77 at the bottom of UBI 12. The signal BBSY REQ H is generated and this goes to the bus busy flip-flop, which consists of E89 and E77. The signal at the output of this flip-flop is called CPU BBSY L, and this also goes to the gate E109, becoming ASSERT BBSY H. When INIT UB REQ L is gone, BBSY L is deasserted.

The microsequencer is allowed to run once DCLO is negated. The first part of the microcode routine from powerup forces Unibus BBSY for 250 ms and then checks ACLO at the end of this 250 ms period. A microbranch on ACLO is taken after the 250 ms. With ACLO asserted, the micromachine waits for the negation of ACLO. At this point it is still another 580 ms before ACLO is negated by the memory controller. Once ACLO is gone, then the Micro-Verify and initialization sequences are done and the system is restarted according to the POWER ON ACTION switch explained above.

2.1.1.2 Power-Down Sequence – When ac power is going down, ACLO is first asserted, which generates a macro-level power-fail interrupt request. This obtains the address of the power-down routine from vector SCBB+C. This routine saves the state of the CPU on the stack in memory. Typically, the amount of time between the assertion of ACLO and DCLO is 3–5 ms. This is sufficient time for the power-fail routine to run and save the CPU state before power is gone. The circuitry that controls the power-down sequence is also contained in the UBI module.

Power-Down Sequence Detailed Description – Refer to UBI module schematic UBI 15. The Unibus ACLO L signal is received at the tranceiver and becomes RCVD ACLO H. This signal is then synchronized to the CPU M clock. On the left side of UBI 15, RCVD ACLO is clocked into latch E127 and becomes SYNCHR ACLO H. This signal goes to UBI 14 latch E132. SYNCHR ACLO H is clocked into this latch by M CLK and the output of the latch goes to NAND gate E65. The output of E65 is called SPFI L (synchronous power fail interrupt) and it goes to the INT gate array E98 on UBI 15. This gate array is the interrupt arbitrator. It arbitrates the interrupt request which is IPL 1E (hex). At the next BUT SERVICE (IRD1+1 cycle), the interrupt service flows for the power fail are entered. The timing diagram in Figure 2-1 shows the relationship of these signals. When DCLO L is received, MSEQ INIT L is asserted. This forces the microsequencer to go to control store address 0000. UB INIT H on UBI 14 is forced true, and Unibus BBSY L is asserted. Approximately 5 μs after DCLO is asserted, the DC voltage should fall below specifications.

2.1.1.3 Power Sequencing With INIT Pushbutton – See Figure 2-2. The INIT pushbutton on the operator control panel initializes the VAX-11/750 processor by forcing ACLO L and BBSY L on the Unibus. This causes the power-down sequence explained previously. However, power is still present. Basically, pressing the INIT pushbutton causes ACLO to be asserted so that a power-fail interrupt request occurs. Seven ms later the CPU internal timing forces DCLO, which forces the microsequencer to 0000. After the DCLO pulse is gone, the microcode executes the power-up sequence. The microcode waits 250 ms and forces Unibus BBSY L. At the end of this interval, a microbranch on ACLO occurs. Pressing the INIT button does not force the memory to initialize by writing all zeros, so ACLO should be negated at the end of 250 ms, and the system is restarted, halted, or booted accoring to the setting of the POWER ON ACTION switch.

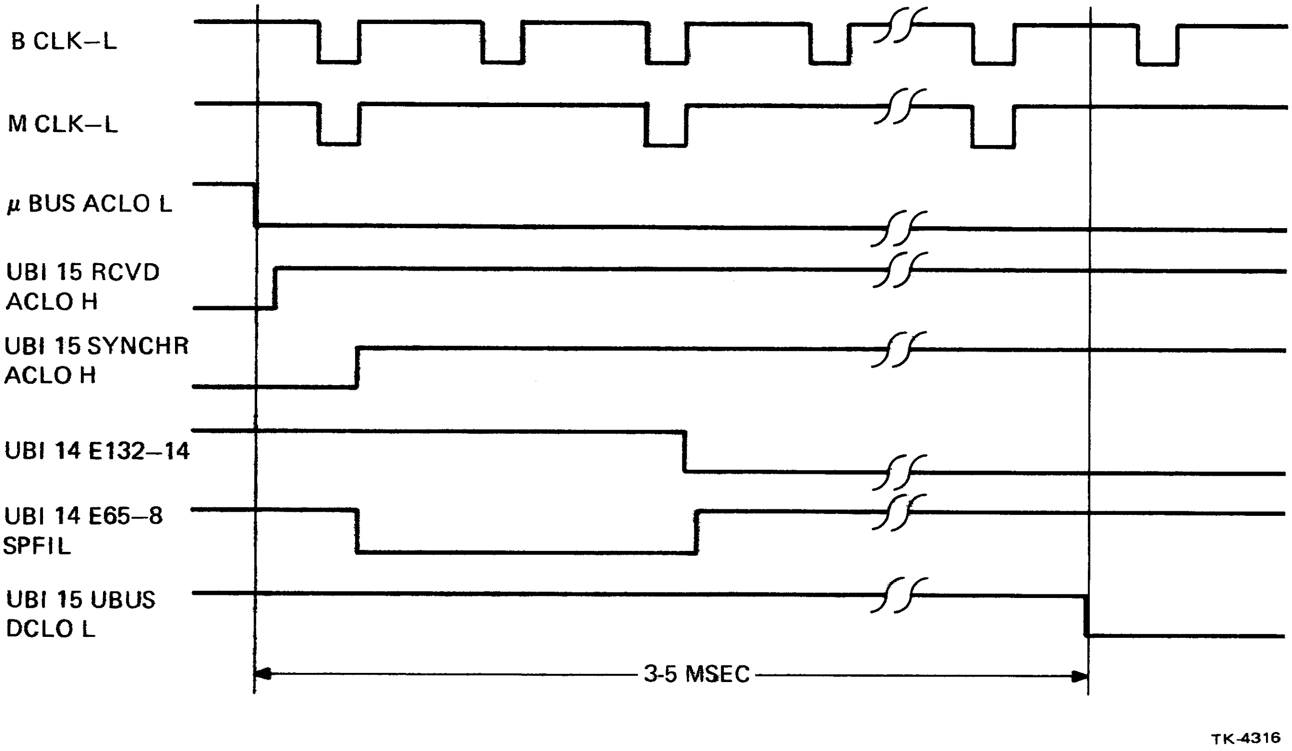


Figure 2-1 Power-Down Sequence Timing

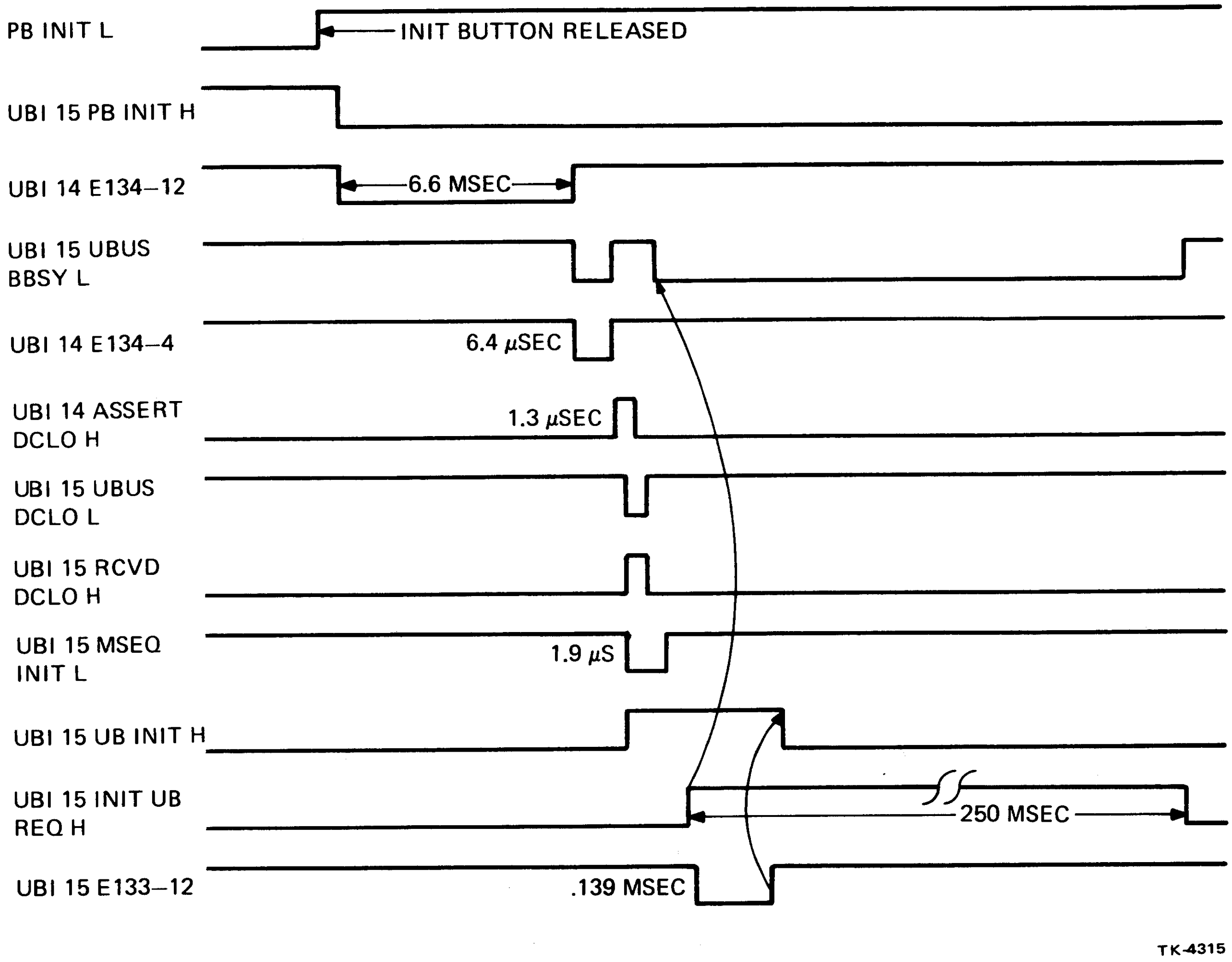


Figure 2-2 INIT Sequence Timing

Detailed Description of INIT Sequence – See Figure 2-2. The INIT pushbutton on the operator control panel is set up so that if the key switch is in either of the SECURE positions, INIT does not function. Pressing INIT connects ground to the backplane pin C7 shown in UBI module prints UBI 15. The signal is called PB INIT L. It directly drives UBUS ACLO L and generates a signal called PB INIT H. The signal PB INIT H is used to start a chain of one-shots that generate ASSERT DCLO H shown on UBI 14 after the operator releases INIT. At the same time, the signal RCVD ACLO H is true, causing the power-down sequence explained in Paragraph 2.1.1.2. This time DCLO is not asserted by the power supply, so the CPU has to force DCLO low. This is done using the one-shots E134A, E134B, and E133 on UBI 14. The first one-shot, E134A, is set for a 6.6 ms low pulse from pin 12. The second one-shot, E133B, produces a 6.4 μs low pulse, and the third one-shot, E132, produces a 1.3 μs high pulse. Refer to Figure 2-2. At the end of the first 6.6 ms interval, the signal ACLO BBSY L is asserted, which asserts BBSY L on the Unibus and fires the second one-shot. The second one-shot fires the third one-shot, and a 1.3 μs high pulse called ASSERT DCLO H is generated. This goes to the Unibus transceiver on UBI 15 and drives DCLO L for 1.3 μs.

When the signal DCLO is received from the transceiver, it forces the signals MSEQ INIT L, Unibus BBSY L, and Unibus INIT L to all be asserted during the 1.3 μs pulse. MSEQ INIT L holds the microsequencer at 0000 until the end of the 1.3 μs pulse, at which time the microcode begins executing again. The CPU microcode has a 250 ms wait loop where Unibus BBSY L remains asserted for the normal power-up sequence. At the end of the 250 ms interval, UBUS ACLO L is again tested. If it is inactive (high), the machine does Micro-Verify and INIT sequences, and restarts according to the position of the POWER ON ACTION switch.

2.1.1.4 Power Sequencing With RDM Installed – The remote diagnostic module (RDM) only drives ACLO and DCLO, and does not receive these signals. It is insensitive to power failures elsewhere in the system. At powerup, the RDM does a self-verification test. During this test, the RD FAULT light on the operator console is illuminated. When the self-test is complete, RD FAULT should be extinguished. While the RDM is performing the self-test, it asserts ACLO and DCLO to hold the processor microcode at location 0000 and force Unibus INIT and BBSY to be asserted. The RDM releases these signals at end of its self-verification test and allows a normal powerup to occur.

2.1.1.5 Time-of-Year Clock (TOY) Power Control – The time-of-year clock operation is described in Paragraph 2.7.5.1. This paragraph describes the toy clock circuitry on the UBI module.

Time-of-Year Battery Power Control Circuit – The power regulator/battery charging circuitry that controls the power to the CMOS logic is shown in the UBI module prints, sheet 1. The time-of-year clock power comes from four 1.35 Vac nominal nickel-cadmium rechargeable cells installed on the back cabinet frame. The TOY circuitry is implemented with CMOS elements that have very low power consumption characteristics. The TOY clock can run for up to 100 hours on battery power.

Detailed Time-of-Year Clock Power Control Description – Refer to the schematic diagram of the TOY clock on the UBI module print, page 1. This circuit is divided into two parts: a charging circuit, and a control circuit.

In discussing the TOY power control circuitry, three different time periods must be considered: when CPU power is down, CPU powerup, and when CPU power is up.

1. When CPU power is down, the TOY batteries provide power to the time-of-year clock CMOS circuitry. At this time diode D3 is used to block discharge of the battery through the power supply.

2. During CPU powerup, UBUS DCLO L is initially asserted. As a result, CPU DCLO H is asserted to comparator E24 A (pin 3). E24 A compares the voltage on pin 3 to a 2.5 V reference voltage on its pin 2. The voltage on pin 3 at this time is more positive than the 2.5 V on pin 2. As a result the E24 A pin 1 output holds transistor Q1 off. Q1 remains off while UBUS DCLO L is asserted. During this time, the battery level is sensed by a voltage divider comprised of resistors R27 and R7. The divided down voltage at the battery is compared to a 3 V reference voltage by comparator E24 B. If the battery voltage has fallen below 4.8 Vdc, the junction voltage of R27 and R7 are less than 3 Vdc and E24 B (pin 7) asserts BATT DCLO L (see Note below). BATT DCLO L active is input to gate E25. When the CPU power reaches a steady condition (indicated by the deassertion of UBUS DCLO L), the output of E25 (pin 11) clears TOY counter E26, and its Q0 output goes low. The 16 × 4 RAM (TOY offset memory) (E50) is disabled for write or read with the outputs <Q3:Q0> pulled high. Any attempt to read the RAM results in a value of 0 returned. The microcode interprets this as a TOY clock battery failure.

NOTE

TOY battery voltage below 4.8 Vdc is insufficient to maintain TOY memory data. BATT DCLO L is asserted to initialize invalidation of the TOY data.

3. When CPU power is up, CPU DCLO H becomes inactive. Comparator E24 A pin 1 allows transistor Q1 to be biased on, and the TOY battery is constantly charged through D3, R5, and Q1. Resistor R5 is used to limit TOY battery charge current to under 100 mA. Diodes D2, D4, and D5 are used to limit the charging voltage to approximatly 6 Vdc.

NOTE

As long as CPU power is up, the TOY circuitry receives dc power.

2.1.2 CPU Main Timing Generation

The VAX-11/750 processor timing circuitry is designed to execute microinstructions at a 320 ns rate. The CMI bus transactions are synchronized by a 160 ns bus clock. These intervals are derived from an 18.75-MHz TTL oscillator located on the CPU control store module (CCS) (slot 5). The oscillator is wired on the backplane to slot 2 of the data path module (DPM) where the service and arbitration control (SAC) gate array is located. The SAC gate array controls the following clock signals used by the CPU. The names of the clock outputs are explained.

Base Clock – Oscillator/3
B Clock – Oscillator/3 and not CS parity or remote clock halt (from RDM)
M Clock – Microsequencer Clock, Oscillator/6
D Clock – Destination Clock, Oscillator/6
Phase Clock – Oscillator/6
Q,D Clock – Oscillator/3 or /6

2.1.2.1 Detailed Analysis of CPU Timing Generation – The following discussion relates to the data path module (DPM) and CPU control store modules (CCS). It references the schematic diagrams for these modules. Timing diagrams to illustrate the clock generation are included in this document.

Refer to the CCS module schematic. On CCS 14 there are two oscillators. The one on the left is time base for the interval counter gate array (TOK) located on the DPM module. The oscillator on the right (E8) is the one that develops the time base for all CPU activity. The output called CPU OSC OUT H is connected via the backplane to slot 2 pin B28. The signal CPU OSC OUT H is connected to the input of the SAC gate array on DPM. Refer to the DPM module schematics. See the clock generation circuitry on DPM17. Backplane pin B28 is jumpered to backplane pin B27 on slot 2 so the signal called CPU OSC OUT H becomes CPU OSC IN H. B27 on the backplane is connected to pin 2 of the SAC gate array. The 18.75-MHz oscillator is divided by 3 within the SAC gate array to 6.25 MHz and appears at the output on pin 6. The signal called SETC goes to flip-flop E56 where it is resynchronized to the oscillator. The 0 output of E56 is BASE CLK L. It is a nonsymmetrical 6.25 MHz signal used to derive the other clock signals. Refer to Figure 2-3 for the phase relationship of the main timing signals. Note that BASE CLK L is present at all times.

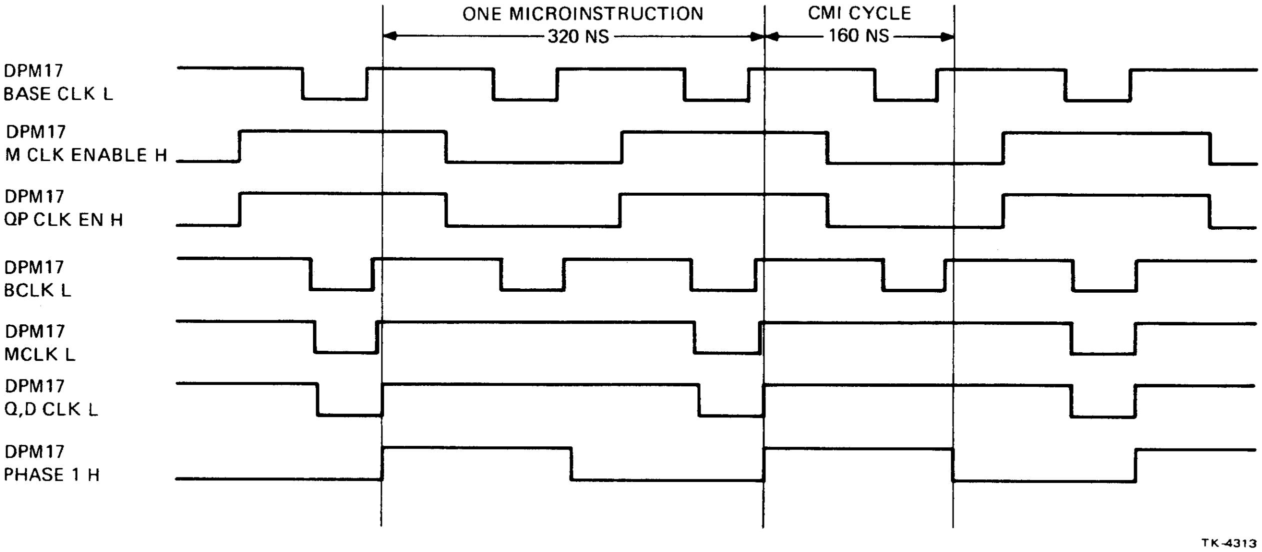


Figure 2-3 Main Timing Signals Phase Relationship

2.1.2.2 Derivation of B CLK L – The signal B CLK L is generated by gating BASE CLK L with a signal coming out of the SAC gate array called HALT L. HALT L can be asserted low under two conditions. The first condition is latching a control store parity error in the SAC gate array and then having a second control store parity error occur before an IRD 1. This causes HALT L to be true. The signal CS PARITY ERROR H is the output of the parity checking logic and enters the SAC gate array at pin 9 where it is latched internally. The second condition that stops B CKL L occurs when the RDM forces the clock to stop by driving the clock control lines called CLK CTRL 1 H and CLK CTRL 0 H low. The RDM can then control the clock and tick it or "step" one microinstruction at a time. The following table describes all the combinations of the clock control lines.

CLK CTRL 1	**CLK CTRL 0**	**Function**
L	L	Stop B CLK L
L	H	Generate 1 B CLK L and stop
H	L	Generate 1 M CLK L and stop
H	H	Run full speed

The RDM module controls these lines when the operator is single-ticking the clock or single-stepping microinstructions from the RDM console.

2.1.2.3 Derivation of M (Microsequencer) CLK L – The M CLK is the microsequencer clock. It used to load the next microinstruction into the control store output latches located on all CPU modules. The timing diagram in Figure 2-3 shows the phase relation of the M CLK to B CLK. Notice that the microinstruction is 320 ns and that is divided into 2 half cycles. M CLK L is used to load the control store output latches on the low-to-high transition. M CLK L occurs every other B CLK except when a stall condition occurs. Stalling the microsequencer is accomplished by inhibiting the M CLK L signal from being generated, thus holding the current microinstruction longer than 2 cycles. Stalling would be necessary when the microcode issued an MSRC/MDR micro-order and the data was not in the MDR yet, for example. M CLK L is derived by gating the BASE CLOCK H signal with M CLK ENABLE H which comes from E2 pin 13. The input to E2 11 and 12 are signals called MEM STALL H and MKEN L. MKEN L comes from the SAC gate array as MKEN H and is inverted through E4. The SAC gate array normally produces an output similar to the second waveform in Figure 2-3. If it is necessary to stall, the SAC gate array keeps MKEN H at low level until the stall condition is removed. The next B CLK generates the M CLK L and loads the next microinstruction. There are numerous conditions that can cause the M CLK to stall. Some of these conditions are listed below.

1. Memory Stall – Waiting for data or I-Stream and memory interface registers.

2. FPA Wait and FPA Stall.

3. Microtraps require an additional cycle set-up time.

4. Clock Extend Bit <15> of the microword extends microinstruction a 1/2 cycle.

5. Compatability Mode IRD 1 – The timing diagram in Figure 2-4 shows how the clocks are extended when the CLKX bit <15> of the microword is set to extend the microinstruction one B CLK.

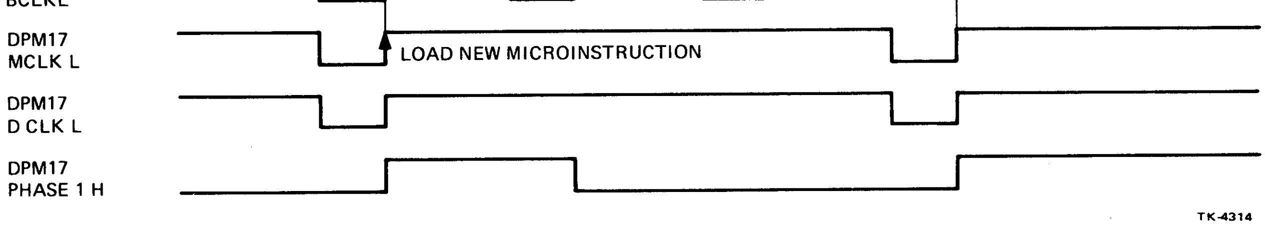


Figure 2-4 Clocks Extended 1/2 Cycle by CLKX

2.1.2.4 Derivation of the D (Destination) CLK – The D CLK L signal is used as write pulse to load data into registers and scratchpads at the end of a microinstruction. Note that the D CLK appears similar to the M CLK in the timing diagram and it coincides with the end of the microinsruction. The D CLK can also be stalled and inhibited under certain conditions. If the microsequencer clock is stalled, the D CLK must also be stalled until the stall condition is removed. The D CLK can be inhibited under certain conditions where the data could be erroneous due to a microtrap or transparent service routine. The D CLK can be inhibited in a service microroutine if the microroutine fetches the data and loads registers or scratchpads, by doing RET.DINH micro-order in the last microinstruction of the service routine. The memory interface control (MIC) module can force a destination inhibit under certain conditions, such as machine check or memory management microtraps.

2.1.2.5 Derivation of the Phase 1 Clock – The phase clock is generated to divide the microcycle into 2 parts. Generally, the first half of the microcycle is used to read scratchpad data which is to be operated on by the rotator and/or ALP logic. During the second half of the cycle, results on the WBus are written to the destination register or scratchpad. The PHASE 1 CLK signal is used to distinguish the two halves of a microcycle so that the correct operation on the scratchpad can be performed. The PHASE 1 CLK is derived from a signal called PHAS, pin 4 of the SAC gate array. This signal enables the J-K flip-flop E56 to set on the low-to-high transition of BASE CLOCK H. The signal PHAS goes low and simultaneously M CLK ENABLE H goes high, allowing the flip-flop to clear on the next BASE CLOCK H. Refer to Figure 2-3 for the phase relationship of these signals. PHASE 1 L is also subject to being stalled by the M CLK stall mechanism. Figure 2-4 shows a CLKX 1/2 cycle stall and the result of PHASE 1 L and H during a clock extend cycle.

2.1.2.6 Derivation of the Q,D Clock – The Q,D CLK signal is used to load and shift the Q and D registers in the data path. The Q,D clock appears the same as the D CLK in most cases. This clock can be modified to look like B CLK when the data path is doing the MULFAST+, MULFAST–, DIVFAST+, and DIVFAST– operations. These are 2-bit multiply and divides per cycle as opposed to the MULSLOW+, MULSLOW–, DIVSLOW+, and DIVSLOW– which are 1-bit multiply and divide operations per cycle.

2.1.2.7 Clock Distribution – The clock signals described above exit the DPM module from the drivers shown in the DPM 17 schematic diagram. The clock signals are connected via the backplane to other modules. Each module that receives the clock signals typically has an emitter follower to buffer the clock signal as shown on DPM 17. Transistors Q1, Q2, and Q3 are buffers for B CLK L, M CLK L, and PHASE 1 H respectively. The following table is list of the major clock signals and the backplane pin where the signal may be observed.

Clock Signal	Slot	Backplane Pin
CPU OSC OUT H	5	B31
CPU OSC OUT H	2	B28
CPU OSC IN H	2	B27
BASE CLK L	2	A73
B CLK L	2	B9
M CLK L	2	B5
D CLK ENABLE H	2	B25
M CLK ENABLE H	2	B15
PHASE 1 H	2	A78

2.2 VAX-11/750 FIRMWARE DESCRIPTION

The VAX-11/750 processor is a microprogrammed machine with a microword of 80 bits. This microword programs all the CPU activity during a single microinstruction cycle, which is 320 ns. All the CPU microinstructions are contained in a 6K × 80 bit control store PROM located on the CCS (CPU control store) module in slot 5 of the CPU backplane. There is also optional control store available in the form of a 1K × 80 bit WCS (writable control store) that attaches to the CCS module. The RDM module also has a writable control store that contains 64 locations for executing microdiagnostics.

A program called MICRO2 allows firmware designers to write individual microinstructions by writing statements or macro expansions in a readable form. The program includes a machine hardware definition that defines the function of every bit in the control store. With this hardware definition, statements can be written to generate individual microinstructions. The program analyzes the statement, indexes into the hardware definition file and produces a binary output that can be blasted into PROM. The CPU firmware routines and macroinstruction microcode was written using this microcode assembler.

Once the machine definition file is complete, a macro file can be built. A macro file is a list of statements that have specific microword fields defined to perform a specific CPU operation during a single microinstruction. The macro file is then expanded, thereby expanding the machine microprogramming language vocabulary. This eliminates defining each microword field in every single microinstruction. The next step is to write the microcode for the machine using macros. As the need for a specific operation occurs, new macros can be added to the microprogramming vocabulary.

The following discussion references the microcode listing of the VAX-11/750 CPU firmware. The microcode listing shows both the source code written by the firmware designer and also the binary output of the MICRO2 assembler program. The name of the listing is usually CMTXXX where XXX is the version number. The version number of the microcode listing is contained in the upper left corner of the listing at the filename. To determine the revision level of microcode in a processor, it is necessary to examine the system identification register, IPR 3E (hex). This register can be examined from the console terminal by typing the following.

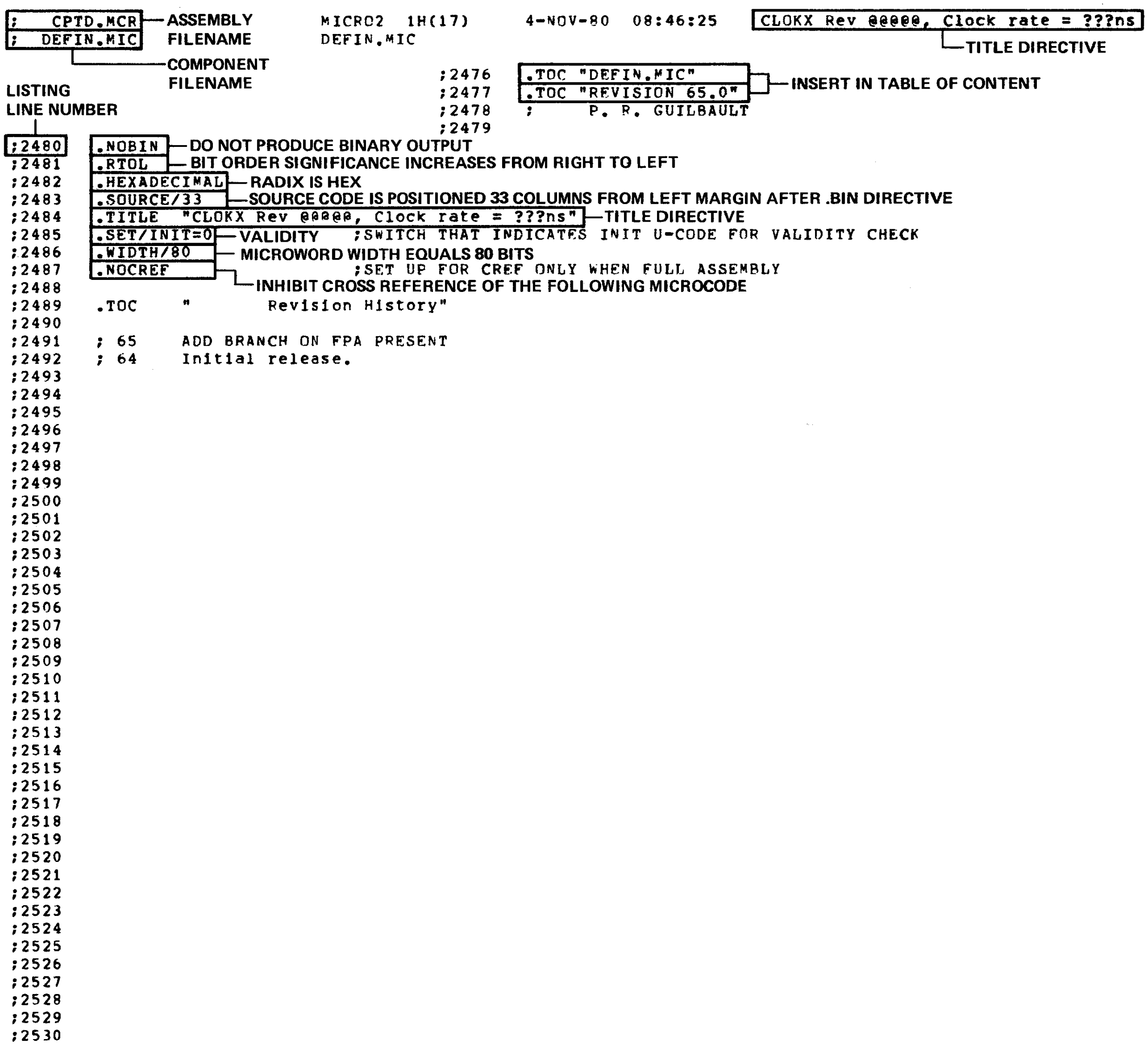
Console Prompt	Type	Console Prints
>>>	E/I 3E	
		I0000003E (IPR Contents)
>>>		

The hex revision level of the processor control store is contained in the third and fourth digits from the right.

2.2.1 Microcode

This paragraph is intended to provide enough background on microcode to ensure that the reader can understand future references to it.

2.2.1.1 Microcode Structure – The following discussion is about how to read microcode. You must have a microcode listing available for this discussion. The first subject is how the control store microword is defined to the MICRO2 assembler. The name of the machine definition file is DEFIN.MIC and contains the definition of each field and every function of that field. Contained in this document are examples from the CMT049 microcode listing to aid in learning to read microcode. Attempt to locate the same directives and statements in your listing. Some of the MICRO2 assembler directives you are likely to encounter are shown in Figure 2-5. A MICRO2 assembler directive is a statement preceded by a ".". In Figure 2-5, look at this page duplicated from the CMT049 listing and study each line within the boxes. At the top is the name of the entire listing CPTD.MCR. The line below called DEFIN.MIC is the name of the subfile that is appended together with all the other files in the listing. At the left there is a line number for every statement or directive. The directives are explained below.



```
;   CPTD.MCR          MICRO2  1H(17)      4-NOV-80   08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns
;  DEFIN.MIC          DEFIN.MIC
                                   ;2476   .TOC "DEFIN.MIC"
                                   ;2477   .TOC "REVISION 65.0"
                                   ;2478   ;       P. R. GUILBAULT
                                   ;2479
;2480   .NOBIN
;2481   .RTOL
;2482   .HEXADECIMAL
;2483   .SOURCE/33
;2484   .TITLE   "CLOKX Rev @@@@@, Clock rate = ???ns"
;2485   .SET/INIT=0            ;SWITCH THAT INDICATES INIT U-CODE FOR VALIDITY CHECK
;2486   .WIDTH/80
;2487   .NOCREF                ;SET UP FOR CREF ONLY WHEN FULL ASSEMBLY
;2488
;2489   .TOC    "        Revision History"
;2490
;2491   ; 65    ADD BRANCH ON FPA PRESENT
;2492   ; 64    Initial release.
;2493
;2494
;2495
;2496
;2497
;2498
;2499
;2500
;2501
;2502
;2503
;2504
;2505
;2506
;2507
;2508
;2509
;2510
;2511
;2512
;2513
;2514
;2515
;2516
;2517
;2518
;2519
;2520
;2521
;2522
;2523
;2524
;2525
;2526
;2527
;2528
;2529
;2530
```

ASSEMBLY FILENAME
COMPONENT FILENAME
LISTING LINE NUMBER
TITLE DIRECTIVE
INSERT IN TABLE OF CONTENT
DO NOT PRODUCE BINARY OUTPUT
BIT ORDER SIGNIFICANCE INCREASES FROM RIGHT TO LEFT
RADIX IS HEX
SOURCE CODE IS POSITIONED 33 COLUMNS FROM LEFT MARGIN AFTER .BIN DIRECTIVE
TITLE DIRECTIVE
VALIDITY
MICROWORD WIDTH EQUALS 80 BITS
INHIBIT CROSS REFERENCE OF THE FOLLOWING MICROCODE

Figure 2-5 MICRO2 Assembler Directives 1

.NOBIN
This directive instructs the assembler not to produce a binary output for the statements that follow this directive.

.RTOL
This directive tells the assembler that bit order of field definitions is from right to left. The LSB is at the right and more significant bits are to the left.

.HEXADECIMAL
This switches the default radix (octal) to hexadecimal.

.SOURCE/33
This directive tells the assembler the position of the source code left margin in the listing output. In this listing the source code is 33 columns from the left margin. This is necessary so the binary output will fit on the listing.

.TITLE "CLOK X REV @@@@@, CLOCK RATE = ??? ns"
This is used to print the title information at the top of each listing page.

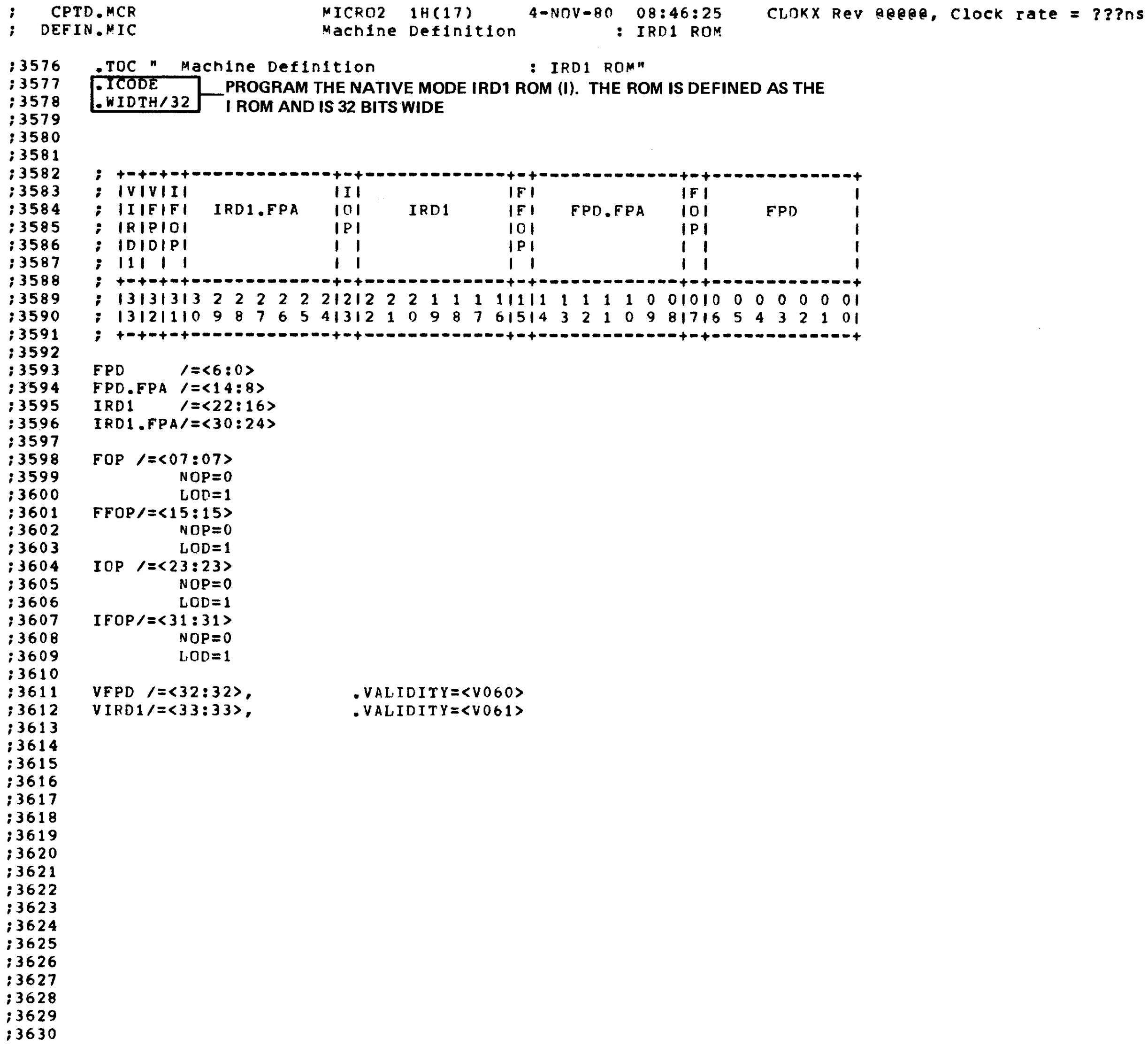
.SET/INIT = O
This is a validity argument that is used during the INIT microcode.

.WIDTH/80
This defines the microword width as 80 bits.

.NOCREF
The .NOCREF directive tells the MICRO2 assembler not to insert the following statements in the cross reference listing.

.TOC
This means to insert the text within the quotation marks into the table of contents at the beginning of the microcode listing.

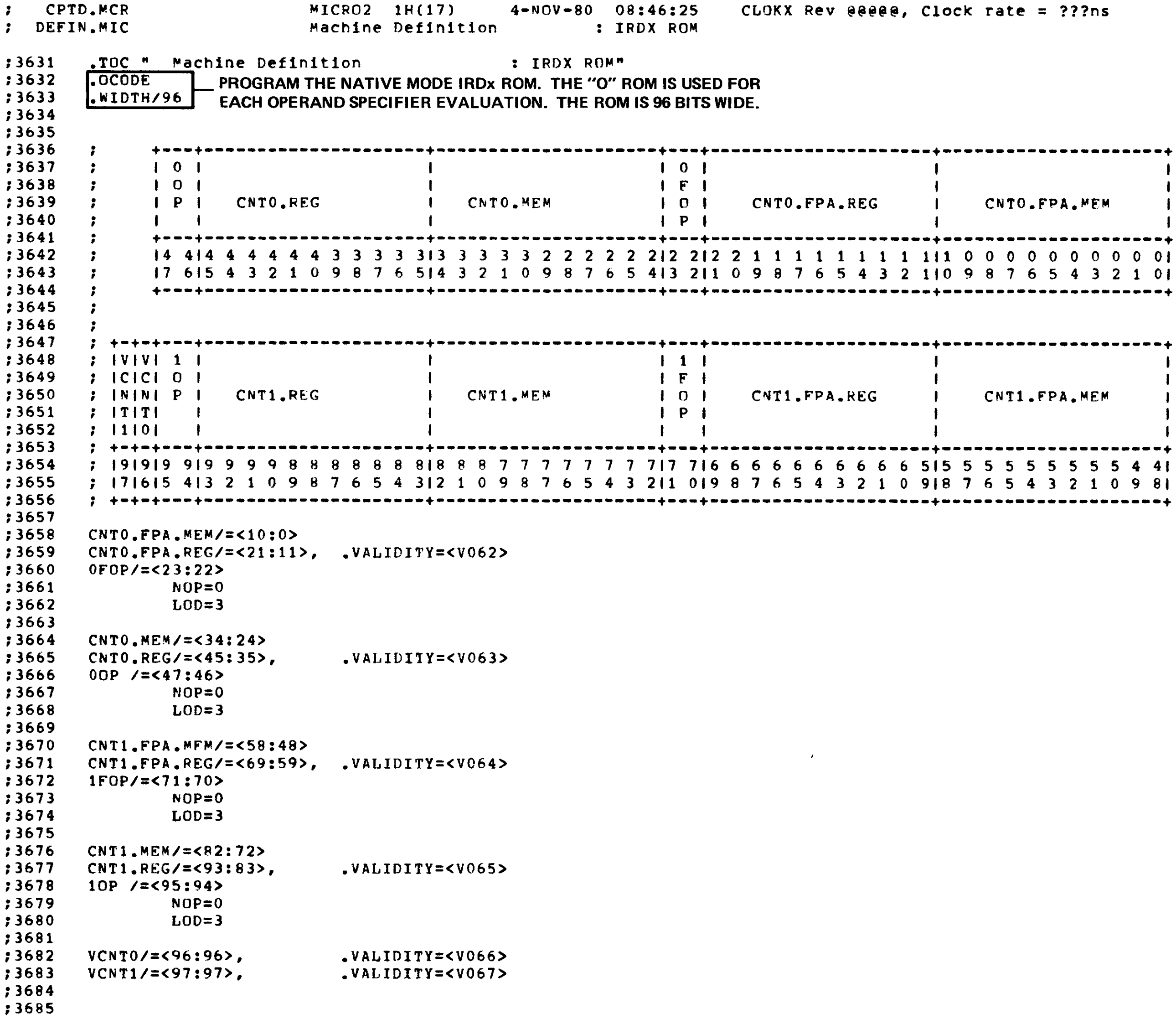
Following these directives is a revision history of the microcode and an explanation of each change. There is a revision history for each file in the listing. In addition to the control store microcode, the firmware designer has to program the IRD ROMs and the D-size ROM that programs the operand size for the individual VAX-11 macroinstructions. This means that the microprogrammer must specify the ROM being programmed. Figure 2-6 shows the .ICODE directive that directs the assembler to define the IRD1 ROM used at instruction decode to point to the operand specifier evaluation microroutine. The width directive defines the size of this ROM in bits. Figure 2-7 shows the .OCODE directive that defines the IRDx ROMs that are used at the first and second operand specifier evaluations after IRD1. The width directive (.WIDTH/96) indicates the width of the ROM being defined. To program the control store ROM, the firmware designer must use the .UCODE directive to insert microcode into the control store ROMs.



```
;   CPTD.MCR                 MICRO2  1H(17)     4-NOV-80  08:46:25     CLOKX Rev @@@@@, Clock rate = ???ns
;  DEFIN.MIC                 Machine Definition          : IRD1 ROM

        .TOC "  Machine Definition                  : IRD1 ROM"
        .ICODE
        .WIDTH/32




        ; +-+-+-+-------------+-+-------------+-+-------------+-+-------------+
        ; |V|V|I|             |I|             |F|             |F|             |
        ; |I|F|F|  IRD1.FPA   |O|    IRD1     |F|   FPD.FPA   |O|     FPD     |
        ; |R|P|O|             |P|             |O|             |P|             |
        ; |D|D|P|             | |             |P|             | |             |
        ; |1| | |             | |             | |             | |             |
        ; +-+-+-+-------------+-+-------------+-+-------------+-+-------------+
        ; |3|3|3|3 2 2 2 2 2 2|2|2 2 2 1 1 1 1|1|1 1 1 1 1 0 0|0|0 0 0 0 0 0 0|
        ; |3|2|1|0 9 8 7 6 5 4|3|2 1 0 9 8 7 6|5|4 3 2 1 0 9 8|7|6 5 4 3 2 1 0|
        ; +-+-+-+-------------+-+-------------+-+-------------+-+-------------+

        FPD      /=<6:0>
        FPD.FPA /=<14:8>
        IRD1     /=<22:16>
        IRD1.FPA/=<30:24>

        FOP /=<07:07>
                NOP=0
                LOD=1
        FFOP/=<15:15>
                NOP=0
                LOD=1
        IOP /=<23:23>
                NOP=0
                LOD=1
        IFOP/=<31:31>
                NOP=0
                LOD=1

        VFPD /=<32:32>,          .VALIDITY=<V060>
        VIRD1/=<33:33>,          .VALIDITY=<V061>
```

PROGRAM THE NATIVE MODE IRD1 ROM (I). THE ROM IS DEFINED AS THE I ROM AND IS 32 BITS WIDE

Figure 2-6 MICRO2 Assembler Directives 2



```
;   CPTD.MCR                   MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns
;  DEFIN.MIC                   Machine Definition          : IRDX ROM

.TOC "  Machine Definition               : IRDX ROM"
.OCODE
.WIDTH/96


;      +----+-------------------------------+-------------------------------+----+-------------------------------+-------------------------------+
;      |  O |                               |                               |  O |                               |                               |
;      |  O |                               |                               |  F |                               |                               |
;      |  P |    CNT0.REG                   |    CNT0.MEM                   |  O |    CNT0.FPA.REG               |    CNT0.FPA.MEM               |
;      |    |                               |                               |  P |                               |                               |
;      +----+-------------------------------+-------------------------------+----+-------------------------------+-------------------------------+
;      |4  4|4  4  4  4  4  4  3  3  3  3  3|3  3  3  3  3  2  2  2  2  2  2|2  2|2  2  1  1  1  1  1  1  1  1  1|1  0  0  0  0  0  0  0  0  0  0|
;      |7  6|5  4  3  2  1  0  9  8  7  6  5|4  3  2  1  0  9  8  7  6  5  4|3  2|1  0  9  8  7  6  5  4  3  2  1|0  9  8  7  6  5  4  3  2  1  0|
;      +----+-------------------------------+-------------------------------+----+-------------------------------+-------------------------------+
;
;
;  +-+-+----+-------------------------------+-------------------------------+----+-------------------------------+-------------------------------+
;  |V|V|  1 |                               |                               |  1 |                               |                               |
;  |C|C|  O |                               |                               |  F |                               |                               |
;  |N|N|  P |    CNT1.REG                   |    CNT1.MEM                   |  O |    CNT1.FPA.REG               |    CNT1.FPA.MEM               |
;  |T|T|    |                               |                               |  P |                               |                               |
;  |1|0|    |                               |                               |    |                               |                               |
;  +-+-+----+-------------------------------+-------------------------------+----+-------------------------------+-------------------------------+
;  |9|9|9  9|9  9  9  9  8  8  8  8  8  8  8|8  8  8  7  7  7  7  7  7  7  7|7  7|6  6  6  6  6  6  6  6  6  6  5|5  5  5  5  5  5  5  5  5  4  4|
;  |7|6|5  4|3  2  1  0  9  8  7  6  5  4  3|2  1  0  9  8  7  6  5  4  3  2|1  0|9  8  7  6  5  4  3  2  1  0  9|8  7  6  5  4  3  2  1  0  9  8|
;  +-+-+----+-------------------------------+-------------------------------+----+-------------------------------+-------------------------------+

CNT0.FPA.MEM/=<10:0>
CNT0.FPA.REG/=<21:11>,  .VALIDITY=<V062>
0FOP/=<23:22>
        NOP=0
        LOD=3

CNT0.MEM/=<34:24>
CNT0.REG/=<45:35>,      .VALIDITY=<V063>
0OP /=<47:46>
        NOP=0
        LOD=3

CNT1.FPA.MFM/=<58:48>
CNT1.FPA.REG/=<69:59>,  .VALIDITY=<V064>
1FOP/=<71:70>
        NOP=0
        LOD=3

CNT1.MEM/=<82:72>
CNT1.REG/=<93:83>,      .VALIDITY=<V065>
1OP /=<95:94>
        NOP=0
        LOD=3

VCNT0/=<96:96>,         .VALIDITY=<V066>
VCNT1/=<97:97>,         .VALIDITY=<V067>

```

PROGRAM THE NATIVE MODE IRDx ROM. THE "O" ROM IS USED FOR EACH OPERAND SPECIFIER EVALUATION. THE ROM IS 96 BITS WIDE.

Figure 2-7 MICRO2 Assembler Directives 3

2.2.1.2 Microword Field Definitions – The VAX-11/750 Microword Chart in the DEFIN.MIC file of the microcode listing shows the different fields of the microword. The microword has vertical functionality; that is, the same bit can have up to 5 functions. This means that some fields determine what others will be. The way to determine which field is used is explained in the hardware section that describes that field. ROT and ALPCTL field vertical functionality are described in Paragraph 2.6 of this document.

This discussion only indicates the purpose of each of the various fields. The following discussion deals with the vertical functionality and what each field does in the CPU. Bits <13:0> of the microword are called the NEXT address. It contains the address of the next microinstruction in the control store. Locate in the DEFIN.MIC file the defintion of the NEXT address field. The definitions of all the fields are arranged alphabetically to help you quickly locate them. The NEXT field definition looks like below.

```
NEXT/=<13:0>,.NEXTADDRESS
```

This definition defines the field name NEXT. The / indicates that bits <13:0> are equated to the NEXT field and the .NEXTADDRESS assembler directive instructs the assembler to insert the location of the label specified in the NEXT field into bits <13:0> of the control store. If the NEXT field is not specified, the assembler will point to the next microinstruction.

The following bit of the microword is called the JSR bit. Locate the JSR field description in DEFIN.MIC. The JSR bit is used in microsubroutine calls. If the field is = 1, the address of the current microinstruction is saved on a microstack. When the microsubroutine is complete, a return micro-order in the BUT field can be issued. This pops this microstack and ADDS bits <5:0> of the NEXT field in the return microinstruction to the address pushed on the microstack. It is possible to return to the location pushed on the microstack +31 or −32 decimal locations.

```
JSR/=<14:14>,.DEFAULT=0
             NOP=0
             PUSH=
```

The JSR bit is 1-bit field. A default value is specified so that if the field is not defined as a PUSH, the default value NOP is put into control store bit <14>.

The clock extend bit was mentioned briefly in Paragraph 2.1. This bit is used to extend the cycle time of the current microinstruction by one B CLK. There are some data path operations that require the extended cycle time, and the clock extend bit must be set. Clock extend is defined below.

```
CLKX/=<15:15>,.DEFAULT=0
             NOP=0
             XTND=1
```

This is also a single-bit field. The default value for this field if XTND is not specified is NOP.

The following group of bits in the microword are used for interfacing the FPA to the CPU. This field basically is used to pass data back and forth to the FPA via the MBus and WBus in the CPU. The VAX-11/750 CPU microroutines must fetch all the operands for the FPA and pass them via the MBus and WBus. When the FPA finishes a math operation, the result is passed back to the CPU, and the CPU must store the result in the destination specified by the operand specifiers. The FPA field is defined as follows.

```
FPA/=<19:16>,.DEFAULT=0
```

The comments indicate what each function does. The default value if the FPA field is not defined in the microinstruction is zero.

The bus field of the microword controls the CPU operation for reads and writes to the CMI bus. As the DEFIN.MIC file shows, the bus field is divided into three major groups of operations. These are:

Reads of memory
Writes to memory
Probes of various sorts of PTEs on different CPU buses.

The bus field definition is

BUS/=<24:20>,.DEFAULT=7

The bus field consists of bits 24 down to 20 of the microword. The default value is 7 when no bus operation is specified.

The following group of bits have vertical functionality to three levels. The WCTRL field is used to control the activity on the WBus. The CCMISC and CCPSL functions are combinations of certain CC and WCTRL micro-orders. The CC field defines how PSW condition codes are modified. The CCMISC field is a combination of the WCTRL field and the CC field. In DEFIN.MIC, note that the field is defined as follows.

CCMISC/=<32:25>

This includes both CC and WCTRL fields of the microword. If the microprogrammer wants to perform any of the functions listed in the definition for CCMISC, bits <32:25> then become CCMISC and the definitions for CC, CCPSL, and WCTRL are no longer valid in this microinstruction.

If the firmware designer does not specify a CCMISC function in the microword, he may specify CC and WCTRL or CC and CCPSL micro-orders. The CCPSL functions are really WCTRL micro-orders that affect the PSL. The CCPSL functions are defined in bit positions <30:25> as follows.

CCPSL/=<30:25>

If the microprogrammer does not specify a CCPSL function as described in the define file, the CCPSL definition is no longer valid and the WCTRL definition of bits <30:25> is then valid.

The WCTRL field controls the WBus activity as well as other activities. It is defined as follows.

WCTRL/=<30:25>,.DEFAULT=2

The WCTRL field has a default value of 2 if it is not specified in the microinstruction.

The CC field of the microword is defined below.

CC/=<32:31>,.DEFAULT=0

The CC field is used to set the PSL condition codes at the end of a VAX-11 macroinstruction. Typically, the CC field is set to CCOP1 or CCOP2 in the last microinstruction of the VAX-11 macroinstruction. If the microprogrammer had not specified any of the functions described above, bits <32:25> of the microword would have had the following default definitions.

<32:31> = CC/NOP.CCBR_SIGND	(Binary 00)
<30:25> = WCTRL/NOP	(Binary 000010)

The field of the microword above the CC field is the ISTRM. The ISTRM bit is used to allow the D-size bits <1:0> to determine the size of an operand, address, or displacement in the instruction stream. This means that the D-size ROM can determine the size as a function of the opcode of the VAX-11 macroinstruction. The ISTRM bit is defined as follows.

```
ISTRM/=<33:33>,.DEFAULT=0
            NOP=0
            ISIZE_DSIZE=1
```

The ISTRM definition is a single bit in location <33> of the control store.

The following part of the microword has vertical functionality. Note that above RSRC, ISTRM, and CC is a field called LITRL. Above that field is another field called long LONLIT. This vertical functionality is interpreted as follows. The two fields LITRL and LONLIT enable the firmware designer to enter constants or literal data into the data path from the control store microword. The LITRL field allows a 9-bit literal to enter the super rotator logic for manipulation, while LONLIT is a 32-bit constant that can be sourced onto the RBus in the data path logic. The choice of whether the LONLIT or the LITRL field is selected as an input to the data path is function of the field described in bits <77:76> of the microword. This field is called the LIT field and is defined as follows.

```
LIT/=<77:76>,.DEFAULT=0
            NOP=0
            LITRL=1
            FPAWAIT=2
            LONLIT=3
```

If the LIT field of the microword is 1, then bits <39:31> are interpreted as the LITRL field and not RSRC, ISTRM, and CC. If the LIT field equals 3, then bits <62:31> become the LONLIT field and the ROT, ALPCTL, BUT, DTYPE, RSRC, ISTRM, and CC fields are not valid during this particular microinstruction. The FPAWAIT micro-order in the LIT field is used in conjunction with the signal FPA STALL L to stall the CPU microcode until the FPA finishes a floating-point instruction. Knowing the vertical functionality of the microword in positions <39:34>, we can be certain that the LIT field must be 0 or 2 to interpret bits <39:34> as the RSRC field. As its name implies, the RSRC field of the microword controls the source of the data for the RBus in the data path. This field is defined as follows.

```
RSRC/=<39:34>,.DEFAULT=0
```

In the DEFIN.MIC file, the RBus data sources include all the RTEMP registers and the LONLIT register.

The DTYPE field occupies bit positions <41:40> of the microword. The DTYPE field is used to determine the width of the data path for each microinstruction. This field has 4 values described below.

```
DTYPE/=<41:40>,.DEFAULT=3
            BYTE=0
            WORD=1
            LONG=2
            IDEP=3
```

The width of the data path can be a byte, a word, or a longword. If the DTYPE field is not specified, the default is IDEP, which means let the D-size ROM select the size of the data path. The D-size ROM is programmed as function of the opcode of the VAX-11 macroinstruction currently executing.

The BUT field in bit positions <47:42> of the microword is used for conditional hardware microbranching, instruction decode, and microsubroutine returns. The BUT field selects a certain hardware condition as an input to a multiplexer whose output is inclusively ORed together with the lower bits of the NEXT address field of the current microinstruction. This means that there are two or more possible destination addresses as a result of this branch condition. The BUT field is also used to specify when to use the IRD ROMs rather the NEXT address field. The BUT field also specifies when to return from a microsubroutine. The operation on a return is to pop the microstack and ADD (not OR) the NEXT address field contained in that instruction to the microstack address saved by the last PUSH (JSR bit <14> =1). The BUT field is defined as follows.

```
BUT/=<47:42>,.DEFAULT=0
```

A very useful table is included in the BUT field definitions in the DEFIN.MIC file. Across the top of the table are 6 columns marked as follows.

```
CSA <5>   CSA <4>   CSA <3>   CSA <2>   CSA <1>   CSA <0>
```

Each column represents the control store address bit that is modified by a given hardware condition. For example, the BUT micro-order WX.EQ.0?=28 is used to test the result of an ALU operation for zero. The microprogrammer can have two targets as a result of this ALU operation. If the ALU output is zero, one target is used. If the ALU output is non-zero then the other target is used. In this case, bit <0> on the control store address lines is asserted to a 1 if the ALU output is 0. The microprogrammer must constrain the NEXT address field in the destination microinstruction such that bit <0> is clear so that the branch condition can be ORed into the control store address. If the NEXT field of the microinstruction were 1000, the microsequencer would read the microinstruction from location 1000 if the ALU output was non-zero, or it would read the microinstruction from location 1001 if the ALU output was zero.

The next part of the microword is the ALPCTL field. This field occupies bits <57:48> of the microword and programs the ALU operation during each microinstruction. This field has vertical functionality. ALPCTL may be interpreted as the MUX, ALU, and DQ fields in certain cases. The ALPCTL field programs the ALP and ALK gate arrays on the DPM module. The MUX, ALU and DQ fields are defined as follows. The MUX field selects the A and B inputs to the ALU. The ALU field defines the arithmetic or logical operation to be performed on the inputs selected by the MUX field. The DQ field programs the operation of the D and Q registers in the ALP gate arrays. Vertical functionality is determined by eliminating the ALPCTL functions. The ALPCTL is defined as follows.

```
ALPCTL/=<57:48>,.DEFAULT=364
```

All the definitions for ALPCTL operations are ALP special functions. If the microprogammer selects a function that is in the ALPCTL special functions table, the MUX, ALU, and DQ fields are not interpreted. If the ALU operation the microprogrammer wants to perform is not a special function described in the special function table, then the MUX, ALU and DQ micro-orders must be specified. Note in the DEFIN.MIC file that MUX field occupies bits <57:54> of the microword. This is part of the area defined by ALPCTL. The vertical functionality of the ALU and DQ fields is determined by the MUX input selection. If the MUX field selects D.R2 (A MUX gets MBus and B MUX gets RBus) or Z.S (A MUX gets 0 and B MUX gets the rotator output), then the ALU field is interpreted as the ALUOD field. If the MUX selects D.R2 or Z.S, the ALU output does not drive the WBus. The DQ field selection is also a function of the MUX input selection. The three DQ micro-order selections are defined below.

MUX Input Selection	**DQ Field**
MUX/M.R1, M.Q1, M.S, XM.R, XM.Q, XM.S, D.R1 D.Q1,D.S,Z.S,R.Q,R.S	DQ1
MUX/M.R2,M.Q2,D.Q2	DQ2
MUX/D.R2	DQ3

If the MUX selects D.R2, the DQ field is DQ3. If the MUX selects either M.R2, M.Q2, or D.Q2, the the DQ field used is DQ2. For all other MUX input selections, the DQ1 micro-order is used. The basic rule for defining the field for bits <57:48> is as follows. First, is an ALPCTL special function being specified? If the function is not an ALPCTL function, it must specify MUX, ALU, and DQ functions. Second, is the MUX field is selecting D.R2 or Z.S? If so, then the ALU field becomes ALUOD. Third, to determine the DQ micro-order, refer to the table above for MUX input selections and determine the proper DQ micro-order. The interpretation of the microword is explained in further detail in subsequent paragraphs.

The super rotator logic controls the shifting, packing, unpacking, and extraction of data from the MBus and RBus of the data path. The super rotator is also capable of extracting fields from combinations of the MBus and RBus data. The rotator can pack and unpack floating data types, BCD strings, and ASCII strings. The rotator is controlled by the ROT microword field. This field has vertical functionality. There are three possible definitions for bits <63:58> of the microword, excluding LONLIT. The rotator field interpretation can be summarized in two statements. The ROT field is interpreted as the ROT field if the microprogrammer uses micro-orders that do either of the following.

> Write the S or P latches in the ROT field. The ROT field is equal to any of the following. These are located in the definitions of the ROT field in the DEFIN.MIC file
>
> PL=2C
> SL=2E
> SL.PL_WB=2F
> OLIT0.PL43_WB=3F
> OLIT0.PL_LIT=3B
> PL.SL_WB=2D
> OLIT0.SL_LIT=3D

Select the super rotator as the input to the B leg of the ALU. The MUX field is equal to any of the following micro-orders. The MUX field is defined in the DEFIN.MIC file.

```
M.S=4
XM.S=7
D.S=C
Z.S=D
R.S=F
```

To summarize, bit <63:58> of the microword is interpreted as the ROT field, if the MUX is selecting the super rotator. Otherwise the S or P latch in the rotator is modified by specifying one of the above ROT micro-orders. If the above condition is not satisfied, the ROT field can become either ROTSRK or ALUXM, ALUCI, and ALUSHF. To specify bit <63:58> of the microword as the ROTSRK field, the BUT field must specify either SRKSTA or CCBR0.SRKSTA0 micro-orders. To enable microbranching on the result of the rotator operation, two status bits are generated by the SRK chip to indicate the status of every operation the rotator performs. These status bits are selected by the microsequencer BUT multiplexer when the BUT field selects the SRKSTA bits in the microbranch. So basically, the ROTSRK field is interpreted when the BUT micro-order specifies SRKSTA or CCBR0.SRKSTA. If the ROT field and the ROTSRK field are not interpreted, then bits <63:58> become ALUXM, ALUSHF, and ALUCI. ALUXM is a bit that determines whether to sign or zero-extend the MBUS input to the MUX depending on the DTYPE field size. ALUSHF is a 3-bit field that programs the shift input to the ALU and Q register. The ALUCI field programs the source of carry inputs to the ALU.

There is no more vertical functionality from here to the end of the microword. The next group of bits determines the source of data to the MBus. This is the MSRC field.

```
MSRC/=<68:64>,.DEFAULT=0
```

The next field is the SPW field. This field programs which set of scratchpad registers is written. If the RSIZE micro-order is specified, then the RTEMPs are written according to the D-size bits <1:0>. The other two writes to the scratchpads are longword writes.

```
SPW/=<70:69>,.DEFAULT=0
               NOP=0
               RSIZE=1
               RLONG=2
               MLONG=3
```

The names are indicativive of the definitions. The SPW determines the scratchpad that is written when the WBus is driven with the input data. If the SPW field specifies a write to scratchpad M, and the MSRC field indicates a nonscratchpad source such as VA or the PC, the scratchpad MTEMP0 will be written by default.

The MISC field of the microword programs the microprogramming aids such as the status flags <5:0>, the step counter, and parts of the PSL. The miscellaneous control field of the microword resides in bit positions <75:71>. The MISC field default value is 10 (hex).

```
MISC/=<75:71>,.DEFAULT=10
```

The LIT field is described earlier in this paragraph.

The most significant bits are the parity bits for the control store microword. Refer to the VAX-11/750 Microword Chart in the DEFIN.MIC file of the microcode listing. Above all the fields is the number 1 or the number 2. These numbers relate to the corresponding bit in the PAR field. PAR2 is a parity bit generated on all fields of the microword marked with a 2. When PAR 2 is included there is ODD parity. PAR1 is a parity bit generated on all fields marked with a 1. When PAR1 is included, the control store uses even parity.

2.2.1.3 Microcode Macro Expansions – This paragraph describes how the VAX-11/750 CPU microcode programming language vocabulary is made. The vocabulary is created by writing macro expansions that perform operations in the CPU. The following solution to a simple problem illustrates how to write a microcode macro expansion. The problem is as follows:

> Read the contents of MTEMP0, add 1 to the contents and store the result in MTEMP 0.

Determine first whether a path can be found by referring to the CPU functional block diagram (Figure 1-1 in Chapter 1). The MTEMP0 can be sourced onto the MBus. The constant 1 in the super rotator logic can be generated. The MBus is selected as the input to the A leg of the ALU and the super rotator output as the B leg data input. The ALU would have to do an A+B operation, and the result would appear on the WBus. The scratchpad write pulse should reload MTEMP0 with the result of the addition. The following list shows what each field value must be for this example. Fields not specified take on their default values.

Field Name	**Function**	**Binary**
MSRC/MTEMP0	Source MTEMP0 to MBus	0
ROT/ZLIT0	Zero Extend and rotate left 0	30
MUX/M.S	A leg gets MBus, B leg gets SR	4
ALU/A+B+CI	Add A plus B plus CI (CI=0)	4
SPW/MLONG	Write MTEMP0 long	3
LIT/LITRL	Enable LITRL field	1
LITRL/1	Put constant 1 in rotator	1

Stating the field name and value created a microinstruction that will read MTEMP0, add 1 to the contents of MTEMP0, and store the result back in MTEMP0. If the microinstruction is to be used again somewhere else, each field name must be stated and and assigned a value. The other alternative is to create a macro expansion to represent this function similar to the one below.

```
MTEMP0_MTEMP0+1
```

The macro shown above could be used again for the same operation after it is defined in the MACRO.MIC file. The method to define this macro is shown below.

```
MTEMP0_MTEMP0+1     "MSRC/0,ROT/ZLIT0,MUX/M.S,ALU/
                     A+B+CI,SPW/MLONG,LIT/LITRL,LITRL/1"
```

We have defined the name MTEMP0_MTEMP0. M+1 as all the fields specified in the macro. All other fields assume default values if not specifically stated. This macro must be placed in the MACRO.MIC file so that when MICRO2 assembles the source statement MTEMP0_MTEMP0+1, the MACRO.MIC file is referenced to produce the field values previously defined.

The microprogramming language was built in this way. Specific CPU operations that are used frequently are written as macros and placed in the MACRO.MIC file so that it is not necessary to write each field name and the value for it. In the VAX-11/750 Microcode Listing, these macros are classified into the following four groups.

1. Basic Group – This group contains combinations of the other types, and unusual cases. NOP is a basic macro for instance.

2. Register Transfer Group – Identified by underscore between source and destination. The example above is a register transfer macro because it reads a scratchpad and transfers the contents back to itself in this case. This type of macro always has an underscore in the statement somewhere.

   MTEMP_MTEMP0+1

   The underscore can be read as "gets"

   MTEMP0 "gets" MTEMP0+1

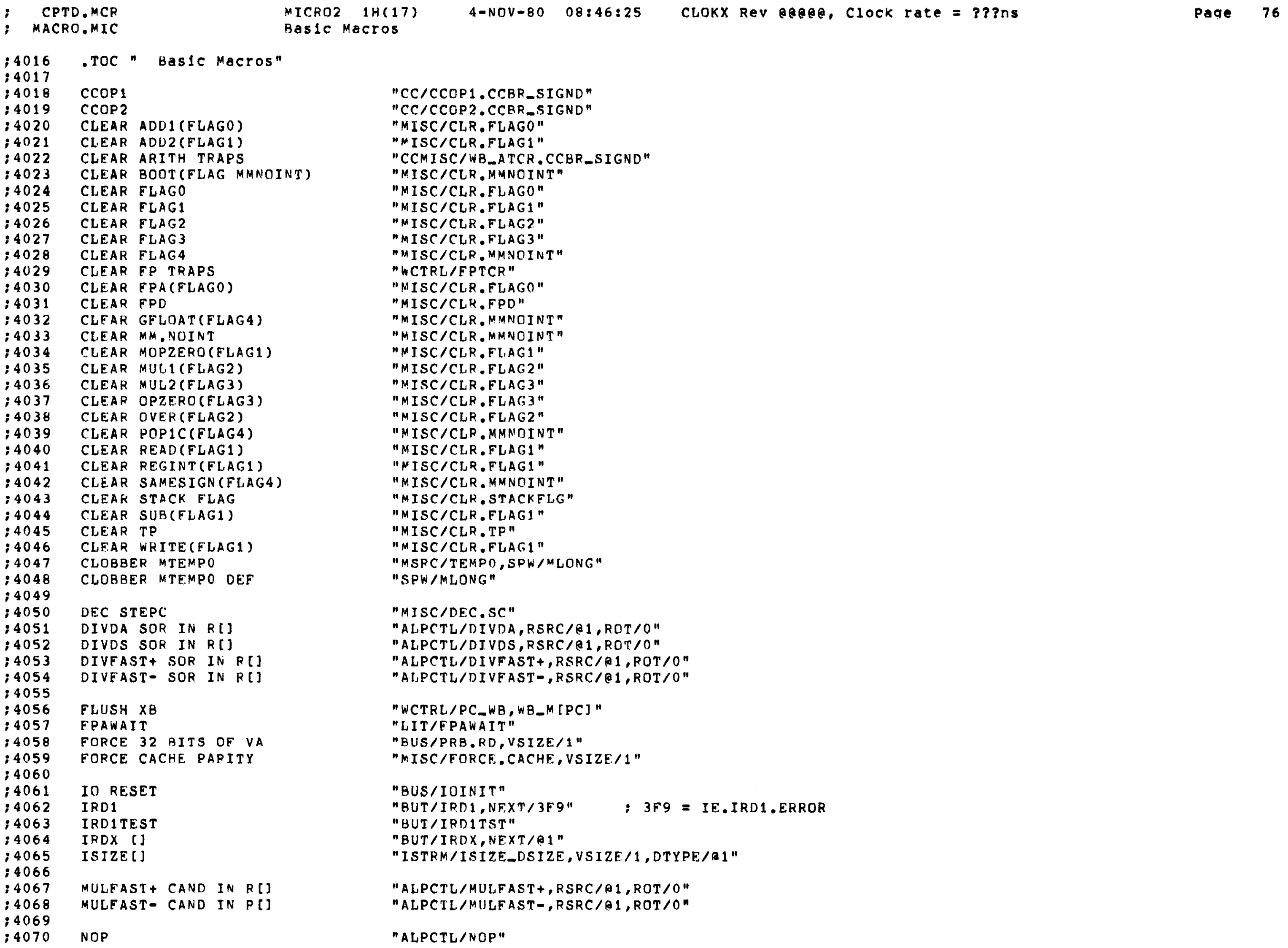
3. Bus Group – This group typically initiates reads and Writes to memory. It also tests PTEs and issues processor INIT. These macros contain the word read or write.

4. Branching Group – This group is used for microbranching and specifies a BUT micro-order. It can be recognized by the question mark (?).

   WX.EQ.0?

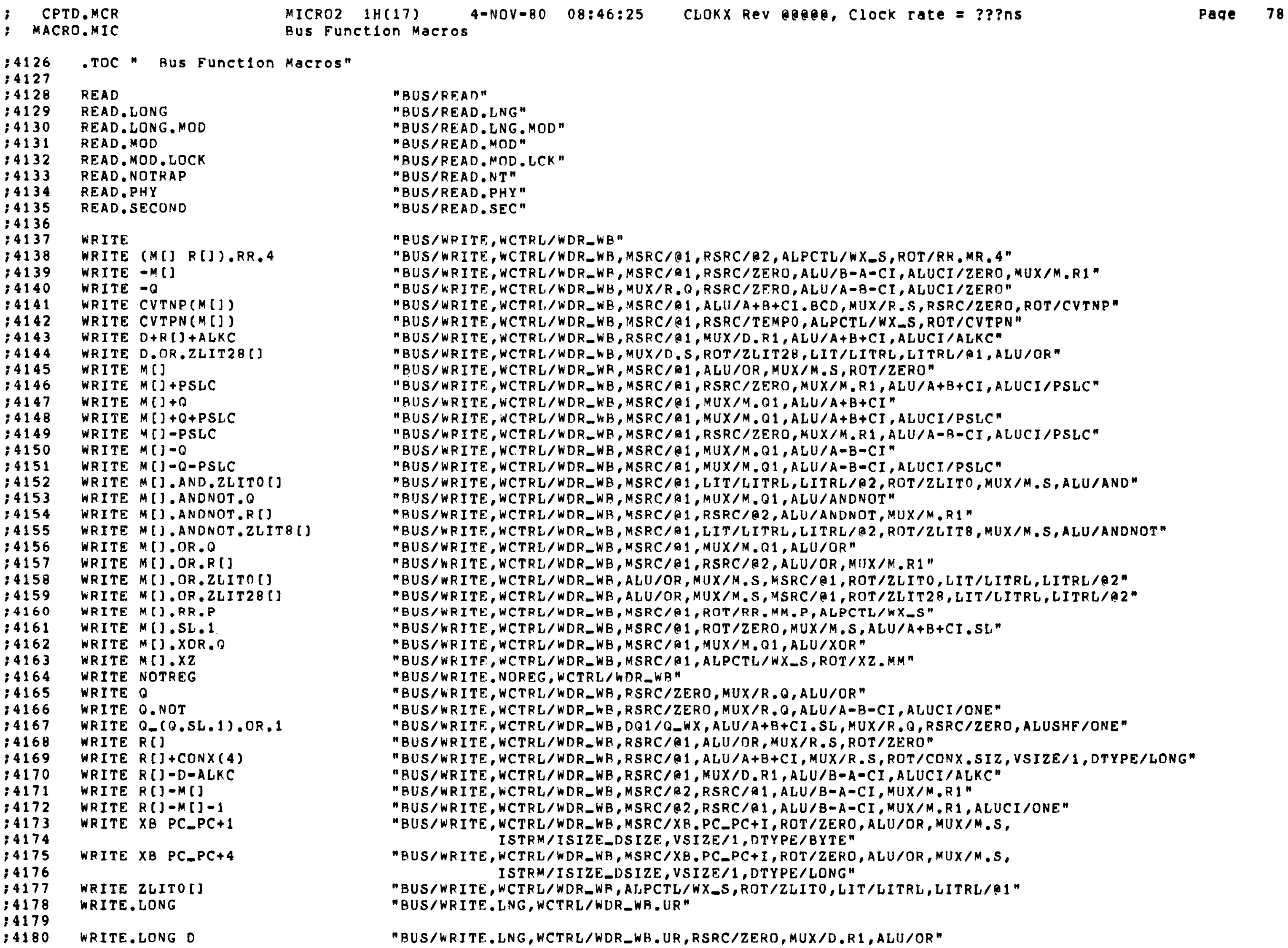
   This macro indicates a microbranch is done on the WMUX being equal to zero and the result is to modify bit <0> of the CS address of the next microinstruction.

Figure 2-8 shows examples of some of the Basic macros. Figure 2-9 illustrates some of the Bus Function macros. Figure 2-10 shows some of the Register Transfer macros and Figure 2-11 shows some Branching macros. Studying the four kinds of macros should enable you to determine what portion of the MACRO.MIC file to reference for any macro in the microcode listing.



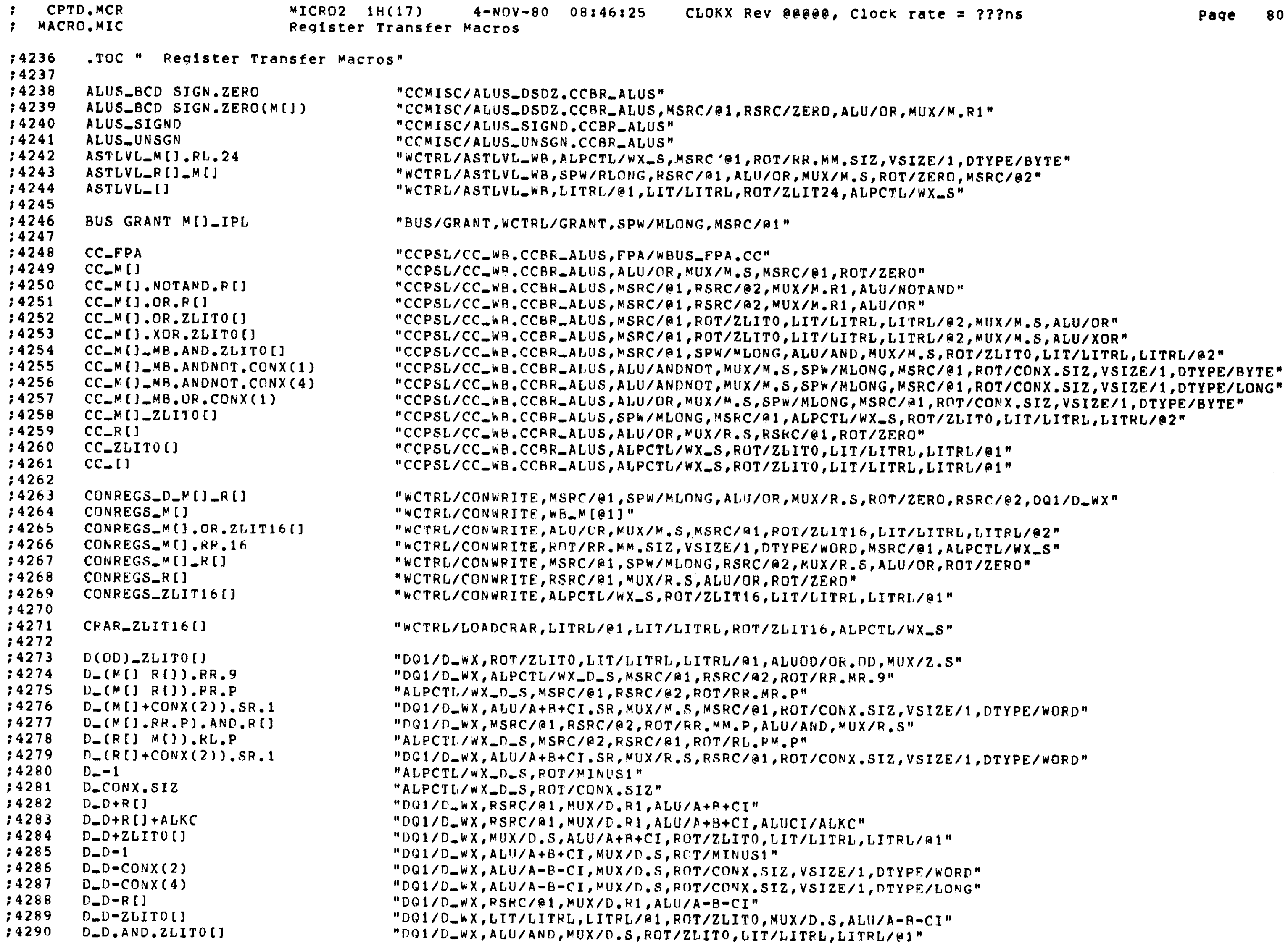
```
;   CPTD.MCR                 MICRO2  1H(17)     4-NOV-80  08:46:25     CLOKX Rev @@@@@, Clock rate = ???ns                    Page   76
;  MACRO.MIC                 Basic Macros

;4016   .TOC "  Basic Macros"
;4017
;4018   CCOP1                           "CC/CCOP1.CCBR_SIGND"
;4019   CCOP2                           "CC/CCOP2.CCBR_SIGND"
;4020   CLEAR ADD1(FLAG0)               "MISC/CLR.FLAG0"
;4021   CLEAR ADD2(FLAG1)               "MISC/CLR.FLAG1"
;4022   CLEAR ARITH TRAPS               "CCMISC/WB_ATCR.CCBR_SIGND"
;4023   CLEAR BOOT(FLAG MMNOINT)        "MISC/CLR.MMNOINT"
;4024   CLEAR FLAG0                     "MISC/CLR.FLAG0"
;4025   CLEAR FLAG1                     "MISC/CLR.FLAG1"
;4026   CLEAR FLAG2                     "MISC/CLR.FLAG2"
;4027   CLEAR FLAG3                     "MISC/CLR.FLAG3"
;4028   CLEAR FLAG4                     "MISC/CLR.MMNOINT"
;4029   CLEAR FP TRAPS                  "WCTRL/FPTCR"
;4030   CLEAR FPA(FLAG0)                "MISC/CLR.FLAG0"
;4031   CLEAR FPD                       "MISC/CLR.FPD"
;4032   CLEAR GFLOAT(FLAG4)             "MISC/CLR.MMNOINT"
;4033   CLEAR MM.NOINT                  "MISC/CLR.MMNOINT"
;4034   CLEAR MOPZERO(FLAG1)            "MISC/CLR.FLAG1"
;4035   CLEAR MUL1(FLAG2)               "MISC/CLR.FLAG2"
;4036   CLEAR MUL2(FLAG3)               "MISC/CLR.FLAG3"
;4037   CLEAR OPZERO(FLAG3)             "MISC/CLR.FLAG3"
;4038   CLEAR OVER(FLAG2)               "MISC/CLR.FLAG2"
;4039   CLEAR POP1C(FLAG4)              "MISC/CLR.MMNOINT"
;4040   CLEAR READ(FLAG1)               "MISC/CLR.FLAG1"
;4041   CLEAR REGINT(FLAG1)             "MISC/CLR.FLAG1"
;4042   CLEAR SAMESIGN(FLAG4)           "MISC/CLR.MMNOINT"
;4043   CLEAR STACK FLAG                "MISC/CLR.STACKFLG"
;4044   CLEAR SUB(FLAG1)                "MISC/CLR.FLAG1"
;4045   CLEAR TP                        "MISC/CLR.TP"
;4046   CLEAR WRITE(FLAG1)              "MISC/CLR.FLAG1"
;4047   CLOBBER MTEMP0                  "MSRC/TEMP0,SPW/MLONG"
;4048   CLOBBER MTEMP0 DEF              "SPW/MLONG"
;4049
;4050   DEC STEPC                       "MISC/DEC.SC"
;4051   DIVDA SOR IN R[]                "ALPCTL/DIVDA,RSRC/@1,ROT/0"
;4052   DIVDS SOR IN R[]                "ALPCTL/DIVDS,RSRC/@1,ROT/0"
;4053   DIVFAST+ SOR IN R[]             "ALPCTL/DIVFAST+,RSRC/@1,ROT/0"
;4054   DIVFAST- SOR IN R[]             "ALPCTL/DIVFAST-,RSRC/@1,ROT/0"
;4055
;4056   FLUSH XB                        "WCTRL/PC_WB,WB_M[PC]"
;4057   FPAWAIT                         "LIT/FPAWAIT"
;4058   FORCE 32 BITS OF VA             "BUS/PRB.RD,VSIZE/1"
;4059   FORCE CACHE PARITY              "MISC/FORCE.CACHE,VSIZE/1"
;4060
;4061   IO RESET                        "BUS/IOINIT"
;4062   IRD1                            "BUT/IRD1,NEXT/3F9"      ; 3F9 = IE.IRD1.ERROR
;4063   IRD1TEST                        "BUT/IRD1TST"
;4064   IRDX []                         "BUT/IRDX,NEXT/@1"
;4065   ISIZE[]                         "ISTRM/ISIZE_DSIZE,VSIZE/1,DTYPE/@1"
;4066
;4067   MULFAST+ CAND IN R[]            "ALPCTL/MULFAST+,RSRC/@1,ROT/0"
;4068   MULFAST- CAND IN P[]            "ALPCTL/MULFAST-,RSRC/@1,ROT/0"
;4069
;4070   NOP                             "ALPCTL/NOP"
```

Figure 2-8   Basic Macros



```
;   CPTD.MCR                 MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns              Page   78
;  MACRO.MIC                 Bus Function Macros

;4126   .TOC "  Bus Function Macros"
;4127
;4128   READ                            "BUS/READ"
;4129   READ.LONG                       "BUS/READ.LNG"
;4130   READ.LONG.MOD                   "BUS/READ.LNG.MOD"
;4131   READ.MOD                        "BUS/READ.MOD"
;4132   READ.MOD.LOCK                   "BUS/READ.MOD.LCK"
;4133   READ.NOTRAP                     "BUS/READ.NT"
;4134   READ.PHY                        "BUS/READ.PHY"
;4135   READ.SECOND                     "BUS/READ.SEC"
;4136
;4137   WRITE                           "BUS/WRITE,WCTRL/WDR_WB"
;4138   WRITE (M[] R[]).RR.4            "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/@2,ALPCTL/WX_S,ROT/RR.MR.4"
;4139   WRITE -M[]                      "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/ZERO,ALU/B-A-CI,ALUCI/ZERO,MUX/M.R1"
;4140   WRITE -Q                        "BUS/WRITE,WCTRL/WDR_WB,MUX/R.Q,RSRC/ZERO,ALU/A-B-CI,ALUCI/ZERO"
;4141   WRITE CVTNP(M[])                "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,ALU/A+B+CI.BCD,MUX/R.S,RSRC/ZERO,ROT/CVTNP"
;4142   WRITE CVTPN(M[])                "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/TEMP0,ALPCTL/WX_S,ROT/CVTPN"
;4143   WRITE D+R[]+ALKC                "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,MUX/D.R1,ALU/A+B+CI,ALUCI/ALKC"
;4144   WRITE D.OR.ZLIT28[]             "BUS/WRITE,WCTRL/WDR_WB,MUX/D.S,ROT/ZLIT28,LIT/LITRL,LITRL/@1,ALU/OR"
;4145   WRITE M[]                       "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,ALU/OR,MUX/M.S,ROT/ZERO"
;4146   WRITE M[]+PSLC                  "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/ZERO,MUX/M.R1,ALU/A+B+CI,ALUCI/PSLC"
;4147   WRITE M[]+Q                     "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A+B+CI"
;4148   WRITE M[]+Q+PSLC                "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A+B+CI,ALUCI/PSLC"
;4149   WRITE M[]-PSLC                  "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/ZERO,MUX/M.R1,ALU/A-B-CI,ALUCI/PSLC"
;4150   WRITE M[]-Q                     "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A-B-CI"
;4151   WRITE M[]-Q-PSLC                "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A-B-CI,ALUCI/PSLC"
;4152   WRITE M[].AND.ZLIT0[]           "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/ZLIT0,MUX/M.S,ALU/AND"
;4153   WRITE M[].ANDNOT.Q              "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/ANDNOT"
;4154   WRITE M[].ANDNOT.R[]            "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/@2,ALU/ANDNOT,MUX/M.R1"
;4155   WRITE M[].ANDNOT.ZLIT8[]        "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/ZLIT8,MUX/M.S,ALU/ANDNOT"
;4156   WRITE M[].OR.Q                  "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/OR"
;4157   WRITE M[].OR.R[]                "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/@2,ALU/OR,MUX/M.R1"
;4158   WRITE M[].OR.ZLIT0[]            "BUS/WRITE,WCTRL/WDR_WB,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZLIT0,LIT/LITRL,LITRL/@2"
;4159   WRITE M[].OR.ZLIT28[]           "BUS/WRITE,WCTRL/WDR_WB,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZLIT28,LIT/LITRL,LITRL/@2"
;4160   WRITE M[].RR.P                  "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,ROT/RR.MM.P,ALPCTL/WX_S"
;4161   WRITE M[].SL.1                  "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,ROT/ZERO,MUX/M.S,ALU/A+B+CI.SL"
;4162   WRITE M[].XOR.Q                 "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/XOR"
;4163   WRITE M[].XZ                    "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,ALPCTL/WX_S,ROT/XZ.MM"
;4164   WRITE NOTREG                    "BUS/WRITE.NOREG,WCTRL/WDR_WB"
;4165   WRITE Q                         "BUS/WRITE,WCTRL/WDR_WB,RSRC/ZERO,MUX/R.Q,ALU/OR"
;4166   WRITE Q.NOT                     "BUS/WRITE,WCTRL/WDR_WB,RSRC/ZERO,MUX/R.Q,ALU/A-B-CI,ALUCI/ONE"
;4167   WRITE Q_(Q.SL.1).OR.1           "BUS/WRITE,WCTRL/WDR_WB,DQ1/Q_WX,ALU/A+B+CI.SL,MUX/R.Q,RSRC/ZERO,ALUSHF/ONE"
;4168   WRITE R[]                       "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,ALU/OR,MUX/R.S,ROT/ZERO"
;4169   WRITE R[]+CONX(4)               "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,ALU/A+B+CI,MUX/R.S,ROT/CONX.SIZ,VSIZE/1,DTYPE/LONG"
;4170   WRITE R[]-D-ALKC                "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,MUX/D.R1,ALU/B-A-CI,ALUCI/ALKC"
;4171   WRITE R[]-M[]                   "BUS/WRITE,WCTRL/WDR_WB,MSRC/@2,RSRC/@1,ALU/B-A-CI,MUX/M.R1"
;4172   WRITE R[]-M[]-1                 "BUS/WRITE,WCTRL/WDR_WB,MSRC/@2,RSRC/@1,ALU/B-A-CI,MUX/M.R1,ALUCI/ONE"
;4173   WRITE XB PC_PC+1                "BUS/WRITE,WCTRL/WDR_WB,MSRC/XB.PC_PC+I,ROT/ZERO,ALU/OR,MUX/M.S,
;4174                                                   ISTRM/ISIZE_DSIZE,VSIZE/1,DTYPE/BYTE"
;4175   WRITE XB PC_PC+4                "BUS/WRITE,WCTRL/WDR_WB,MSRC/XB.PC_PC+I,ROT/ZERO,ALU/OR,MUX/M.S,
;4176                                                   ISTRM/ISIZE_DSIZE,VSIZE/1,DTYPE/LONG"
;4177   WRITE ZLIT0[]                   "BUS/WRITE,WCTRL/WDR_WB,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;4178   WRITE.LONG                      "BUS/WRITE.LNG,WCTRL/WDR_WB.UR"
;4179
;4180   WRITE.LONG D                    "BUS/WRITE.LNG,WCTRL/WDR_WB.UR,RSRC/ZERO,MUX/D.R1,ALU/OR"
```

Figure 2-9 Bus Function Macros



```
;   CPTD.MCR               MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns                Page   80
;  MACRO.MIC               Register Transfer Macros

;4236   .TOC "  Register Transfer Macros"
;4237
;4238   ALUS_BCD SIGN.ZERO                  "CCMISC/ALUS_DSDZ.CCBR_ALUS"
;4239   ALUS_BCD SIGN.ZERO(M[])             "CCMISC/ALUS_DSDZ.CCBR_ALUS,MSRC/@1,RSRC/ZERO,ALU/OR,MUX/M.R1"
;4240   ALUS_SIGND                          "CCMISC/ALUS_SIGND.CCBR_ALUS"
;4241   ALUS_UNSGN                          "CCMISC/ALUS_UNSGN.CCBR_ALUS"
;4242   ASTLVL_M[].RL.24                    "WCTRL/ASTLVL_WB,ALPCTL/WX_S,MSRC/@1,ROT/RR.MM.SIZ,VSIZE/1,DTYPE/BYTE"
;4243   ASTLVL_R[]_M[]                      "WCTRL/ASTLVL_WB,SPW/RLONG,RSRC/@1,ALU/OR,MUX/M.S,ROT/ZERO,MSRC/@2"
;4244   ASTLVL_[]                           "WCTRL/ASTLVL_WB,LITRL/@1,LIT/LITRL,ROT/ZLIT24,ALPCTL/WX_S"
;4245
;4246   BUS GRANT M[]_IPL                   "BUS/GRANT,WCTRL/GRANT,SPW/MLONG,MSRC/@1"
;4247
;4248   CC_FPA                              "CCPSL/CC_WB.CCBR_ALUS,FPA/WBUS_FPA.CC"
;4249   CC_M[]                              "CCPSL/CC_WB.CCBR_ALUS,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZERO"
;4250   CC_M[].NOTAND.R[]                   "CCPSL/CC_WB.CCBR_ALUS,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/NOTAND"
;4251   CC_M[].OR.R[]                       "CCPSL/CC_WB.CCBR_ALUS,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/OR"
;4252   CC_M[].OR.ZLIT0[]                   "CCPSL/CC_WB.CCBR_ALUS,MSRC/@1,ROT/ZLIT0,LIT/LITRL,LITRL/@2,MUX/M.S,ALU/OR"
;4253   CC_M[].XOR.ZLIT0[]                  "CCPSL/CC_WB.CCBR_ALUS,MSRC/@1,ROT/ZLIT0,LIT/LITRL,LITRL/@2,MUX/M.S,ALU/XOR"
;4254   CC_M[]_MB.AND.ZLIT0[]               "CCPSL/CC_WB.CCBR_ALUS,MSRC/@1,SPW/MLONG,ALU/AND,MUX/M.S,ROT/ZLIT0,LIT/LITRL,LITRL/@2"
;4255   CC_M[]_MB.ANDNOT.CONX(1)            "CCPSL/CC_WB.CCBR_ALUS,ALU/ANDNOT,MUX/M.S,SPW/MLONG,MSRC/@1,ROT/CONX.SIZ,VSIZE/1,DTYPE/BYTE"
;4256   CC_M[]_MB.ANDNOT.CONX(4)            "CCPSL/CC_WB.CCBR_ALUS,ALU/ANDNOT,MUX/M.S,SPW/MLONG,MSRC/@1,ROT/CONX.SIZ,VSIZE/1,DTYPE/LONG"
;4257   CC_M[]_MB.OR.CONX(1)                "CCPSL/CC_WB.CCBR_ALUS,ALU/OR,MUX/M.S,SPW/MLONG,MSRC/@1,ROT/CONX.SIZ,VSIZE/1,DTYPE/BYTE"
;4258   CC_M[]_ZLIT0[]                      "CCPSL/CC_WB.CCBR_ALUS,SPW/MLONG,MSRC/@1,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@2"
;4259   CC_R[]                              "CCPSL/CC_WB.CCBR_ALUS,ALU/OR,MUX/R.S,RSRC/@1,ROT/ZERO"
;4260   CC_ZLIT0[]                          "CCPSL/CC_WB.CCBR_ALUS,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;4261   CC_[]                               "CCPSL/CC_WB.CCBR_ALUS,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;4262
;4263   CONREGS_D_M[]_R[]                   "WCTRL/CONWRITE,MSRC/@1,SPW/MLONG,ALU/OR,MUX/R.S,ROT/ZERO,RSRC/@2,DQ1/D_WX"
;4264   CONREGS_M[]                         "WCTRL/CONWRITE,WB_M[@1]"
;4265   CONREGS_M[].OR.ZLIT16[]             "WCTRL/CONWRITE,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZLIT16,LIT/LITRL,LITRL/@2"
;4266   CONREGS_M[].RR.16                   "WCTRL/CONWRITE,ROT/RR.MM.SIZ,VSIZE/1,DTYPE/WORD,MSRC/@1,ALPCTL/WX_S"
;4267   CONREGS_M[]_R[]                     "WCTRL/CONWRITE,MSRC/@1,SPW/MLONG,RSRC/@2,MUX/R.S,ALU/OR,ROT/ZERO"
;4268   CONREGS_R[]                         "WCTRL/CONWRITE,RSRC/@1,MUX/R.S,ALU/OR,ROT/ZERO"
;4269   CONREGS_ZLIT16[]                    "WCTRL/CONWRITE,ALPCTL/WX_S,ROT/ZLIT16,LIT/LITRL,LITRL/@1"
;4270
;4271   CRAR_ZLIT16[]                       "WCTRL/LOADCRAR,LITRL/@1,LIT/LITRL,ROT/ZLIT16,ALPCTL/WX_S"
;4272
;4273   D(OD)_ZLIT0[]                       "DQ1/D_WX,ROT/ZLIT0,LIT/LITRL,LITRL/@1,ALUOD/OR.OD,MUX/Z.S"
;4274   D_(M[] R[]).RR.9                    "DQ1/D_WX,ALPCTL/WX_D_S,MSRC/@1,RSRC/@2,ROT/RR.MR.9"
;4275   D_(M[] R[]).RR.P                    "ALPCTL/WX_D_S,MSRC/@1,RSRC/@2,ROT/RR.MR.P"
;4276   D_(M[]+CONX(2)).SR.1                "DQ1/D_WX,ALU/A+B+CI.SR,MUX/M.S,MSRC/@1,ROT/CONX.SIZ,VSIZE/1,DTYPE/WORD"
;4277   D_(M[].RR.P).AND.R[]                "DQ1/D_WX,MSRC/@1,RSRC/@2,ROT/RR.MM.P,ALU/AND,MUX/R.S"
;4278   D_(R[] M[]).RL.P                    "ALPCTL/WX_D_S,MSRC/@2,RSRC/@1,ROT/RL.RM.P"
;4279   D_(R[]+CONX(2)).SR.1                "DQ1/D_WX,ALU/A+B+CI.SR,MUX/R.S,RSRC/@1,ROT/CONX.SIZ,VSIZE/1,DTYPE/WORD"
;4280   D_-1                                "ALPCTL/WX_D_S,ROT/MINUS1"
;4281   D_CONX.SIZ                          "ALPCTL/WX_D_S,ROT/CONX.SIZ"
;4282   D_D+R[]                             "DQ1/D_WX,RSRC/@1,MUX/D.R1,ALU/A+B+CI"
;4283   D_D+R[]+ALKC                        "DQ1/D_WX,RSRC/@1,MUX/D.R1,ALU/A+B+CI,ALUCI/ALKC"
;4284   D_D+ZLIT0[]                         "DQ1/D_WX,MUX/D.S,ALU/A+B+CI,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;4285   D_D-1                               "DQ1/D_WX,ALU/A+B+CI,MUX/D.S,ROT/MINUS1"
;4286   D_D-CONX(2)                         "DQ1/D_WX,ALU/A-B-CI,MUX/D.S,ROT/CONX.SIZ,VSIZE/1,DTYPE/WORD"
;4287   D_D-CONX(4)                         "DQ1/D_WX,ALU/A-B-CI,MUX/D.S,ROT/CONX.SIZ,VSIZE/1,DTYPE/LONG"
;4288   D_D-R[]                             "DQ1/D_WX,RSRC/@1,MUX/D.R1,ALU/A-B-CI"
;4289   D_D-ZLIT0[]                         "DQ1/D_WX,LIT/LITRL,LITRL/@1,ROT/ZLIT0,MUX/D.S,ALU/A-B-CI"
;4290   D_D.AND.ZLIT0[]                     "DQ1/D_WX,ALU/AND,MUX/D.S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
```

Figure 2-10 Register Transfer Macros

```
;5501   .TOC "  Branching Macros"
;5502
;5503   (M[TEMP3]-SL)BYTE RANGE CHECK?  "BUT/SRKSTA,MSRC/@1,MUX/M.S,ALU/A-B-CI,ROT/SL"
;5504   (PL+SL).GT.32?                  "BUT/SRKSTA,ROTSRK/VIELD.000"
;5505
;5506   ABSVAL M[]<7-0>?                "BUT/SRKSTA,ROT/MINUS1,MSRC/@1,MUX/M.S,ALU/AND"
;5507   ACLO FPLOCK?                    "BUT/FPS3"
;5508   ADD1(FLAGO)?                    "BUT/FLAGO"
;5509   ADD2(FLAG1) ADD1(FLAGO)?        "BUT/FLAG1TOO"
;5510   ALKC?                           "BUT/WBUS31TO30,ALPCTL/WB_ALUF"
;5511   ALLOW INT?                      "BUT/CCBR1.INT-TS"
;5512   ALUS?                           "BUT/CCBR,CC/NOP.CCBR_ALUS"
;5513   ALUS_UNSGN OLDALUS?             "BUT/CCBR,CCMISC/ALUS_UNSGN.CCBR_ALUS"
;5514   ASCII SIGN(M[])?                "BUT/SRKSTA,MSRC/@1,RSRC/ZERO,ALU/OR,MUX/M.R1,ROTSRK/ASCIISIGN.073"
;5515
;5516   BCD CHECK?                      "BUT/BCDCHK"
;5517   BCD CHECK M[]?                  "BUT/BCDCHK,MSRC/@1,ALU/A+B+CI.BCD,MUX/R.S,RSRC/ZERO,ROT/CVTNP"
;5518   BCD SIGN M[]?                   "BUT/SRKSTA,ROT/BCDSWP,MSRC/@1"
;5519   BCD SIGN.ZERO?                  "BUT/CCBR,CC/NOP.CCBR_ALUS"
;5520   BCD SIGN.ZERO(DEF)?             "BUT/CCBR"
;5521
;5522   BINARY LOAD?                    "BUT/CCBR,CC/NOP.CCBR_ALUS"
;5523   BOOT(FLAG MMNOINT)?             "BUT/MM.NOINT"
;5524   BRA ON ADD?                     "BUT/BRA.ON.ADD"
;5525
;5526   CCOP1 SIGND?                    "BUT/CCBR,CC/CCOP1.CCBR_SIGND"
;5527   CCOP2 SIGND CMP .NOT.IR0?       "BUT/CCBR1.CCBR0.IR0,CC/CCOP2.CCBR_SIGND"
;5528   CC_ZLITO[] ALUS?                "BUT/CCBR,CCPSL/CC_WB.CCBR_ALUS,LIT/LITRL,LITRL/@1,ROT/ZLITO,ALPCTL/WX_S"
;5529   CHECK INTERRUPTS?               "BUT/CCBR1.INT-TS,VSIZE/1,DTYPE/LONG"
;5530   CMP SIGNS?                      "BUT/CCBR,CCMISC/NOP.CCBR_CSIGNS"
;5531   COUNT OR INT TIMER?             "BUT/CCBR1.INT-TS"
;5532
;5533   DIVIDEND SIGN?                  "MSRC/TEMP1,BUT/FRO.FLTZ"
;5534   DBZ STEPC?                      "BUT/DBZ.SC"
;5535   DSIZE?                          "BUT/DSIZE"
;5536
;5537   EMODH(FLAG4)?                   "BUT/MM.NOINT"
;5538   EXPONENT RANGE?                 "BUT/SRKSTA"
;5539
;5540   FLAGO?                          "BUT/FLAGO"
;5541   FLAG1 (FLAG2.XOR.FLAG3)?        "BUT/F1.XOR23"
;5542   FLAG1?                          "BUT/FLAG1"
;5543   FLAG2?                          "BUT/FLAG2"
;5544   FLAG3?                          "BUT/FLAG3"
;5545   FLAG4?                          "BUT/MM.NOINT"
;5546   FLAG<1-0>?                      "BUT/FLAG1TOO"
;5547   FLAG<2-0>?                      "BUT/FLAG2TOO"
;5548   FPA PRESENT?                    "BUT/NO.FPA"
;5549   FPA(FLAGO)?                     "BUT/FLAGO"
;5550   FPD?                            "BUT/FPD"
;5551   FPS1?                           "BUT/FPS1"
;5552   FPS2?                           "BUT/FPS2"
;5553   FPS3?                           "BUT/FPS3"
;5554   FRO.FLTZ?                       "BUT/FRO.FLTZ"
;5555
```

Figure 2-11 Branching Macros

2.2.2 Macro Expansion Decoding

To repair the CPU, it may be necessary to translate the macro expansions back to binary data. The procedure to obtain the binary value consists of two steps. The first step is to determine the type of macro (Basic, Register Transfer, Branching, or Bus) and locate the macro in the MACRO.MIC file of the microcode listing. The second step is to trace each field value back to the DEFIN.MIC file and locate the binary data. This procedure is used to scope the logic to isolate a failure. An example is shown in the following figures, with the appropriate portion of the MACRO.MIC and DEFIN.MIC files reproduced.

Refer to the macro expansion in the box with the number 1 in Figure 2-12. This macro came from INIT microroutine as it appeared in CMT049. The macro is as follows.

LONLIT_[41F0000],

or

LONLIT "gets the constant" [41F0000]

Reproduced in Figure 2-13 is the portion the MACRO.MIC file that defines the macro LONLIT_[]. The macro is written so that the contents within the brackets "[]" is user-defined. The macro definition is set up so that the content of the LONLIT field is a dummy argument. When the user microprogrammer specifies the macro LONLIT_[], the LONLIT field can be anything.

LONLIT_[]   "LIT/LONLIT,LONLIT<.NOT[<LONLIT/@1>]>"

This macro defines the LIT field as LONLIT and the LONLIT field as the complement of the user-defined constant that is represented by the symbol "@1". The LONLIT register is loaded from the control store output in bit positons <62:31>. It is necessary to complement the data because the RBus is driven to true low. Knowing the macro definition, one can locate the binary data in the LIT field of the DEFIN.MIC file. A portion of the DEFIN.MIC file with the LIT and LONLIT field definitions is reproduced in Figure 2-14. The LIT and LONLIT fields are boxed. The binary data for the LIT field is shown below

LIT/LONLIT where LONLIT is equal to 3

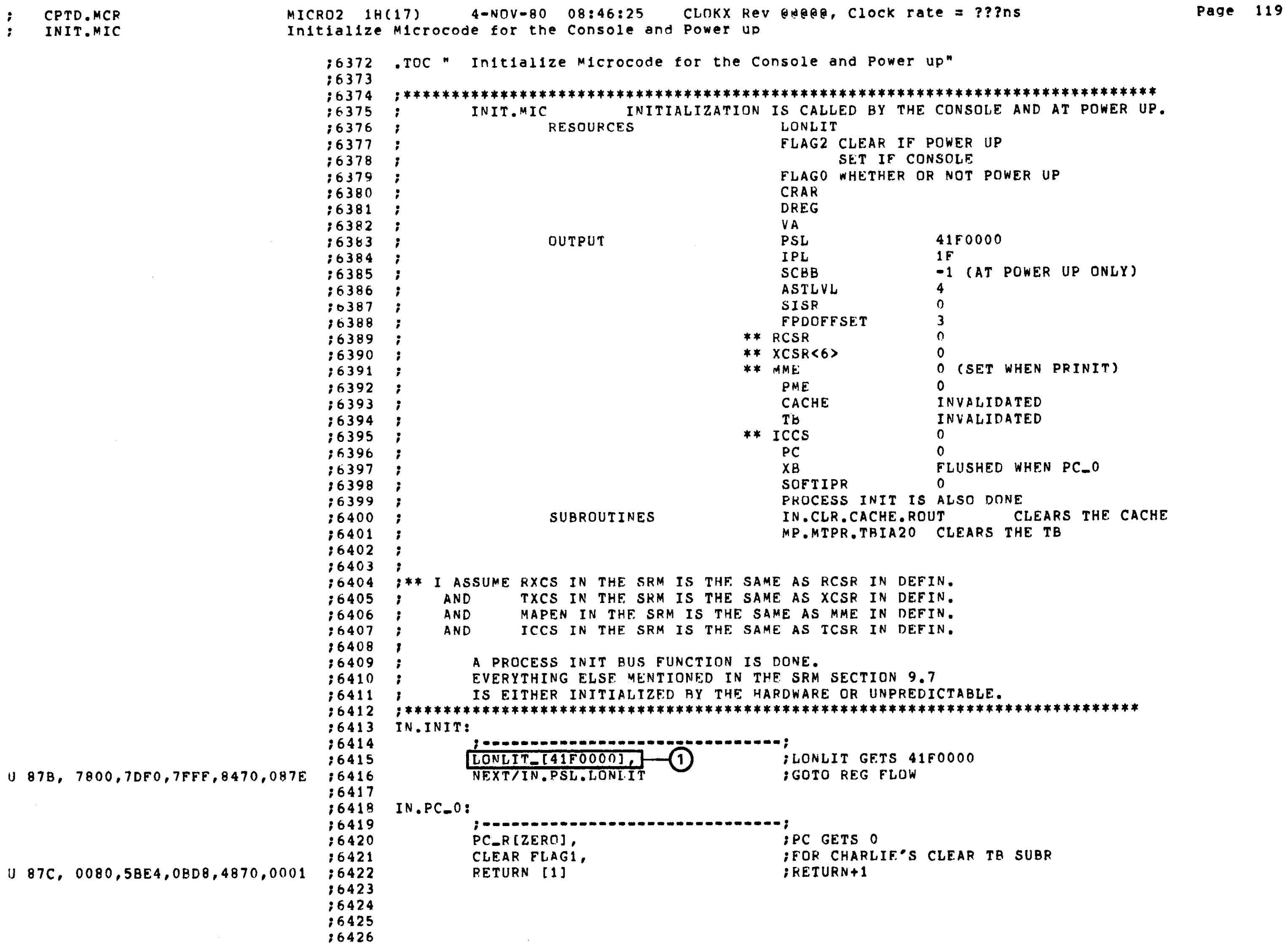
The LONLIT field would contain the complement of 041F0000, which is FBEOFFFF in bit positions <62:31>. This cannot be read directly from the binary shown in Figure 2-12 since LONLIT field is offset by 1 bit. Right-shifting FBEOFFFF one bit position yields 7DF87FFF, which is clearly visible in the binary output shown on the left side of Figure 2-12.

Figure 2-15 shows another macro. This can be identified as a Basic macro since it does not show the characteristics of Transfer, Bus, or Branching macros. The macro states the following.

CLEAR FLAG1,

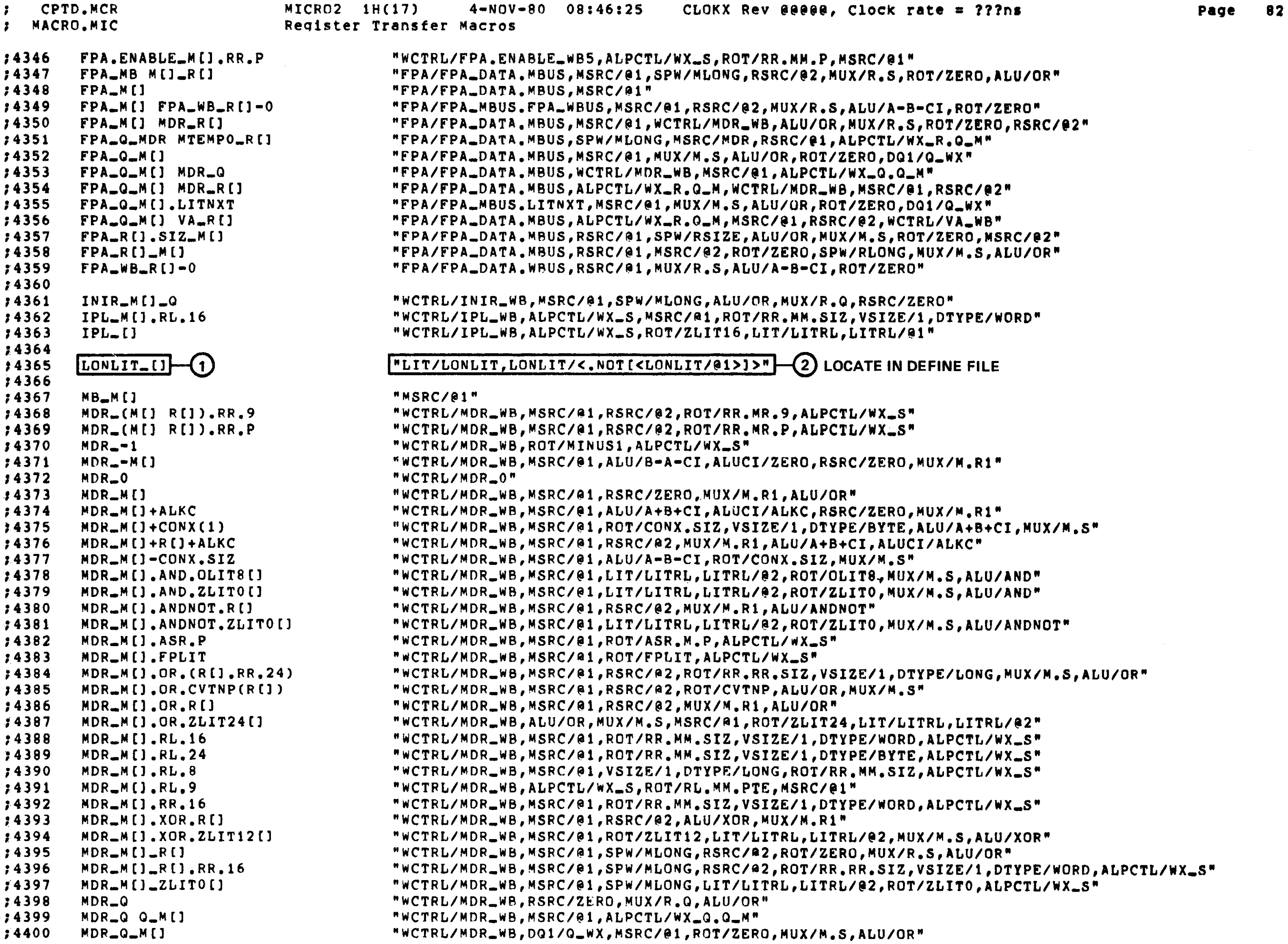
This clears status flag 1 in the microsequencer logic. Again, this is one of the firmware designer's microprogramming aids that can be used for microbranching tests. It instructs to clear status flag 1. Figure 2-16 shows the appropriate portion of the MACRO.MIC file, where the macro is defined as follows.

CLEAR FLAG1   "MISC/CLR."



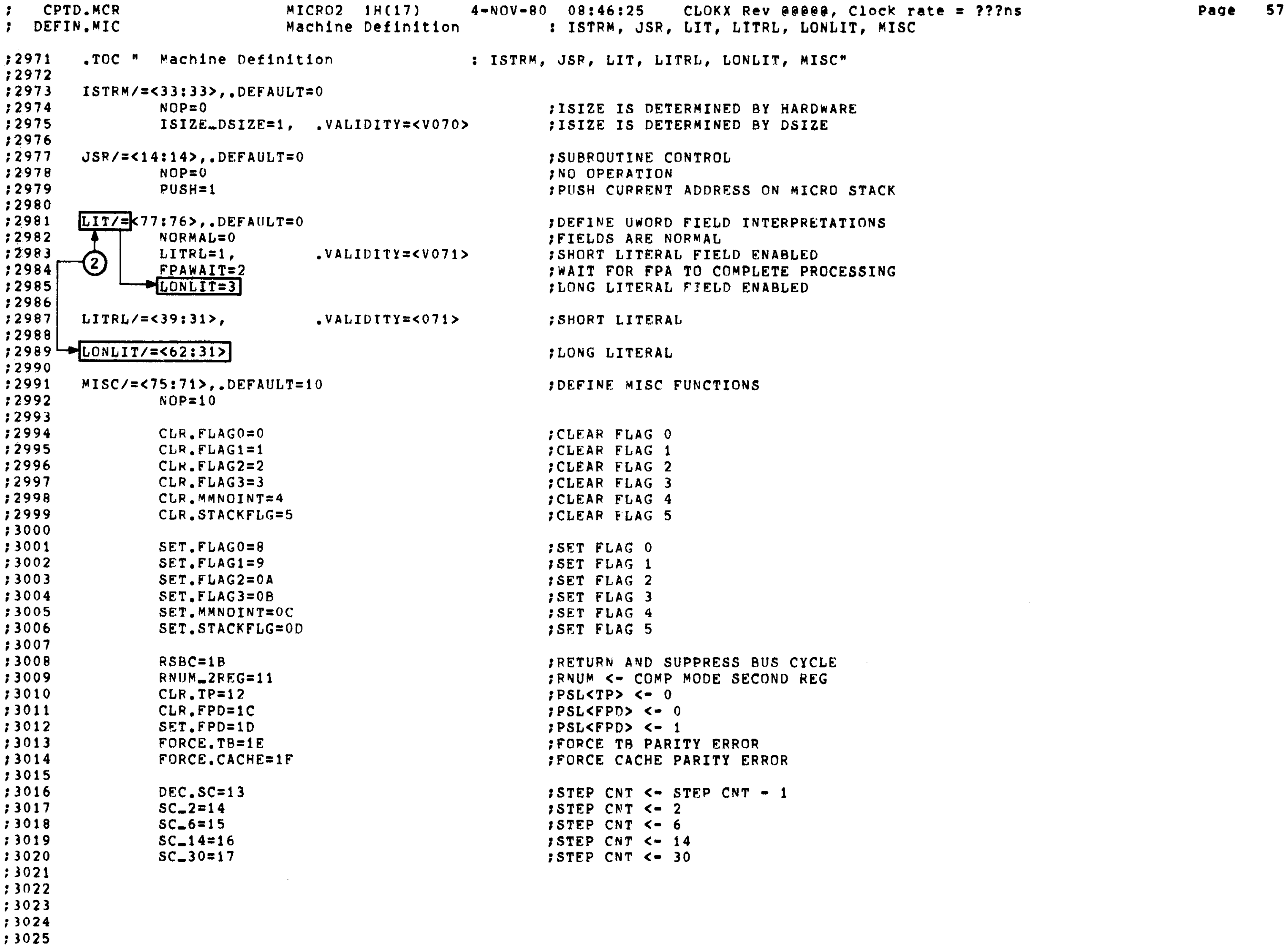
```
;   CPTD.MCR                MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @#@@@, Clock rate = ???ns                Page  119
;   INIT.MIC                Initialize Microcode for the Console and Power up

                              ;6372  .TOC "  Initialize Microcode for the Console and Power up"
                              ;6373
                              ;6374  ;*****************************************************************************
                              ;6375  ;       INIT.MIC        INITIALIZATION IS CALLED BY THE CONSOLE AND AT POWER UP.
                              ;6376  ;               RESOURCES                       LONLIT
                              ;6377  ;                                               FLAG2 CLEAR IF POWER UP
                              ;6378  ;                                                     SET IF CONSOLE
                              ;6379  ;                                               FLAGO WHETHER OR NOT POWER UP
                              ;6380  ;                                               CRAR
                              ;6381  ;                                               DREG
                              ;6382  ;                                               VA
                              ;6383  ;               OUTPUT                          PSL             41F0000
                              ;6384  ;                                               IPL             1F
                              ;6385  ;                                               SCBB            -1 (AT POWER UP ONLY)
                              ;6386  ;                                               ASTLVL          4
                              ;6387  ;                                               SISR            0
                              ;6388  ;                                               FPDOFFSET       3
                              ;6389  ;                                            ** RCSR            0
                              ;6390  ;                                            ** XCSR<6>         0
                              ;6391  ;                                            ** MME             0 (SET WHEN PRINIT)
                              ;6392  ;                                               PME             0
                              ;6393  ;                                               CACHE           INVALIDATED
                              ;6394  ;                                               TB              INVALIDATED
                              ;6395  ;                                            ** ICCS            0
                              ;6396  ;                                               PC              0
                              ;6397  ;                                               XB              FLUSHED WHEN PC_0
                              ;6398  ;                                               SOFTIPR         0
                              ;6399  ;                                               PROCESS INIT IS ALSO DONE
                              ;6400  ;               SUBROUTINES                     IN.CLR.CACHE.ROUT       CLEARS THE CACHE
                              ;6401  ;                                               MP.MTPR.TBIA20  CLEARS THE TB
                              ;6402  ;
                              ;6403  ;
                              ;6404  ;** I ASSUME RXCS IN THE SRM IS THE SAME AS RCSR IN DEFIN.
                              ;6405  ;    AND     TXCS IN THE SRM IS THE SAME AS XCSR IN DEFIN.
                              ;6406  ;    AND     MAPEN IN THE SRM IS THE SAME AS MME IN DEFIN.
                              ;6407  ;    AND     ICCS IN THE SRM IS THE SAME AS TCSR IN DEFIN.
                              ;6408  ;
                              ;6409  ;       A PROCESS INIT BUS FUNCTION IS DONE.
                              ;6410  ;       EVERYTHING ELSE MENTIONED IN THE SRM SECTION 9.7
                              ;6411  ;       IS EITHER INITIALIZED BY THE HARDWARE OR UNPREDICTABLE.
                              ;6412  ;****************************************************************************
                              ;6413  IN.INIT:
                              ;6414          ;-------------------------------;
                              ;6415          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
U 87B, 7800,7DF0,7FFF,8470,087E ;6416          NEXT/IN.PSL.LONLIT              ;GOTO REG FLOW
                              ;6417
                              ;6418  IN.PC_0:
                              ;6419          ;-------------------------------;
                              ;6420          PC_R[ZERO],                     ;PC GETS 0
                              ;6421          CLEAR FLAG1,                    ;FOR CHARLIE'S CLEAR TB SUBR
U 87C, 0080,5BE4,0BD8,4870,0001 ;6422          RETURN [1]                      ;RETURN+1
                              ;6423
                              ;6424
                              ;6425
                              ;6426
```

Figure 2-12   Labels and Macro Expansions



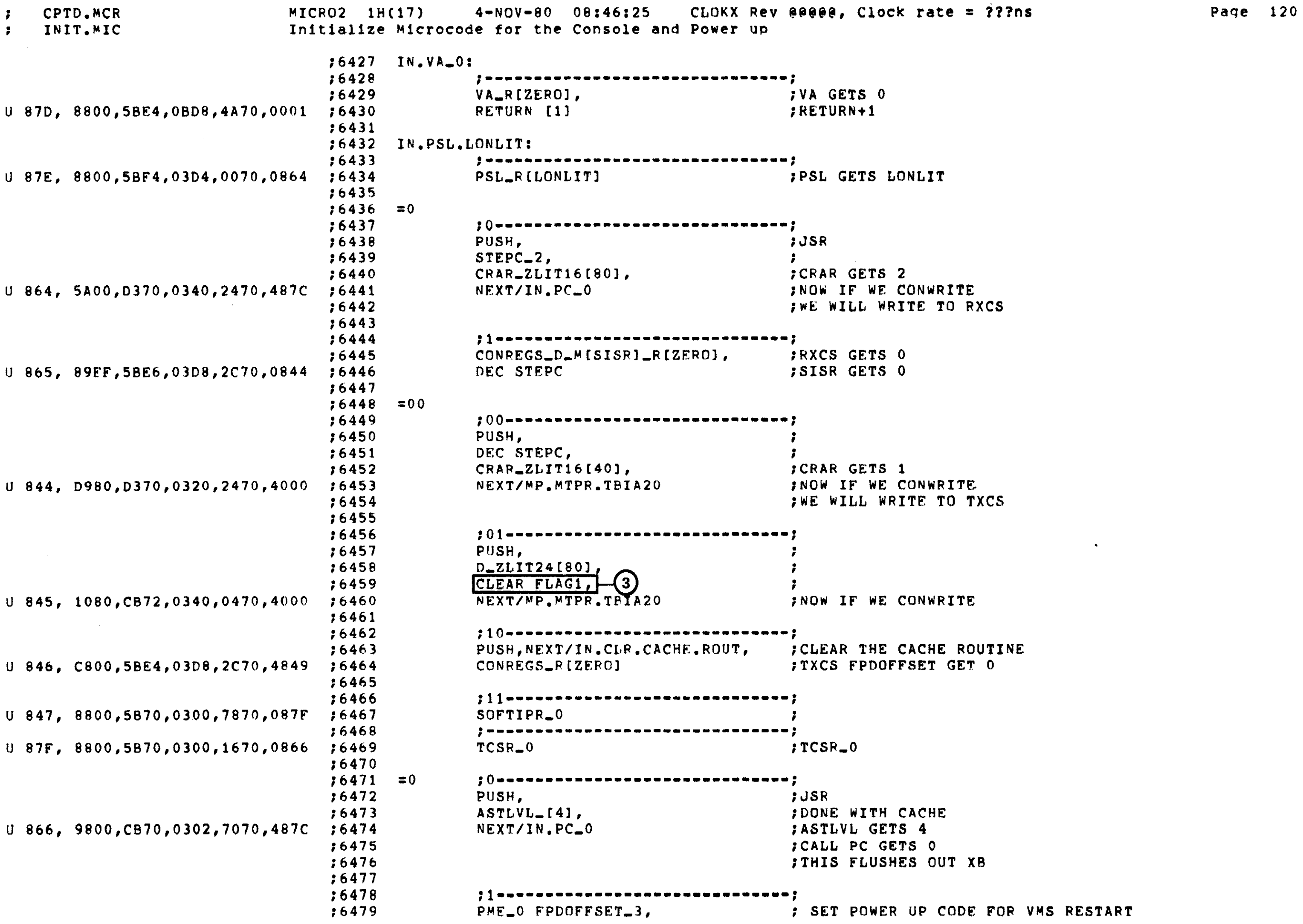
```
;  CPTD.MCR                 MICRO2  1H(17)     4-NOV-80  08:46:25     CLOKX Rev @@@@@, Clock rate = ???ns                Page   82
;  MACRO.MIC                Register Transfer Macros

;4346    FPA.ENABLE_M[].RR.P              "WCTRL/FPA.ENABLE_WB5,ALPCTL/WX_S,ROT/RR.MM.P,MSRC/@1"
;4347    FPA_MB M[]_R[]                   "FPA/FPA_DATA.MBUS,MSRC/@1,SPW/MLONG,RSRC/@2,MUX/R.S,ROT/ZERO,ALU/OR"
;4348    FPA_M[]                          "FPA/FPA_DATA.MBUS,MSRC/@1"
;4349    FPA_M[] FPA_WB_R[]-0             "FPA/FPA_MBUS.FPA_WBUS,MSRC/@1,RSRC/@2,MUX/R.S,ALU/A-B-CI,ROT/ZERO"
;4350    FPA_M[] MDR_R[]                  "FPA/FPA_DATA.MBUS,MSRC/@1,WCTRL/MDR_WB,ALU/OR,MUX/R.S,ROT/ZERO,RSRC/@2"
;4351    FPA_Q_MDR MTEMPO_R[]             "FPA/FPA_DATA.MBUS,SPW/MLONG,MSRC/MDR,RSRC/@1,ALPCTL/WX_R.Q_M"
;4352    FPA_Q_M[]                        "FPA/FPA_DATA.MBUS,MSRC/@1,MUX/M.S,ALU/OR,ROT/ZERO,DQ1/Q_WX"
;4353    FPA_Q_M[] MDR_Q                  "FPA/FPA_DATA.MBUS,WCTRL/MDR_WB,MSRC/@1,ALPCTL/WX_Q.Q_M"
;4354    FPA_Q_M[] MDR_R[]                "FPA/FPA_DATA.MBUS,ALPCTL/WX_R.Q_M,WCTRL/MDR_WB,MSRC/@1,RSRC/@2"
;4355    FPA_Q_M[].LITNXT                 "FPA/FPA_MBUS.LITNXT,MSRC/@1,MUX/M.S,ALU/OR,ROT/ZERO,DQ1/Q_WX"
;4356    FPA_Q_M[] VA_R[]                 "FPA/FPA_DATA.MBUS,ALPCTL/WX_R.Q_M,MSRC/@1,RSRC/@2,WCTRL/VA_WB"
;4357    FPA_R[].SIZ_M[]                  "FPA/FPA_DATA.MBUS,RSRC/@1,SPW/RSIZE,ALU/OR,MUX/M.S,ROT/ZERO,MSRC/@2"
;4358    FPA_R[]_M[]                      "FPA/FPA_DATA.MBUS,RSRC/@1,MSRC/@2,ROT/ZERO,SPW/RLONG,MUX/M.S,ALU/OR"
;4359    FPA_WB_R[]-0                     "FPA/FPA_DATA.WBUS,RSRC/@1,MUX/R.S,ALU/A-B-CI,ROT/ZERO"
;4360
;4361    INIR_M[]_Q                       "WCTRL/INIR_WB,MSRC/@1,SPW/MLONG,ALU/OR,MUX/R.Q,RSRC/ZERO"
;4362    IPL_M[].RL.16                    "WCTRL/IPL_WB,ALPCTL/WX_S,MSRC/@1,ROT/RR.MM.SIZ,VSIZE/1,DTYPE/WORD"
;4363    IPL_[]                           "WCTRL/IPL_WB,ALPCTL/WX_S,ROT/ZLIT16,LIT/LITRL,LITRL/@1"
;4364
;4365    LONLIT_[]—(1)                    "LIT/LONLIT,LONLIT/<.NOT[<LONLIT/@1>]>"—(2) LOCATE IN DEFINE FILE
;4366
;4367    MB_M[]                           "MSRC/@1"
;4368    MDR_(M[] R[]).RR.9               "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ROT/RR.MR.9,ALPCTL/WX_S"
;4369    MDR_(M[] R[]).RR.P               "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ROT/RR.MR.P,ALPCTL/WX_S"
;4370    MDR_-1                           "WCTRL/MDR_WB,ROT/MINUS1,ALPCTL/WX_S"
;4371    MDR_-M[]                         "WCTRL/MDR_WB,MSRC/@1,ALU/B-A-CI,ALUCI/ZERO,RSRC/ZERO,MUX/M.R1"
;4372    MDR_0                            "WCTRL/MDR_0"
;4373    MDR_M[]                          "WCTRL/MDR_WB,MSRC/@1,RSRC/ZERO,MUX/M.R1,ALU/OR"
;4374    MDR_M[]+ALKC                     "WCTRL/MDR_WB,MSRC/@1,ALU/A+B+CI,ALUCI/ALKC,RSRC/ZERO,MUX/M.R1"
;4375    MDR_M[]+CONX(1)                  "WCTRL/MDR_WB,MSRC/@1,ROT/CONX.SIZ,VSIZE/1,DTYPE/BYTE,ALU/A+B+CI,MUX/M.S"
;4376    MDR_M[]+R[]+ALKC                 "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/A+B+CI,ALUCI/ALKC"
;4377    MDR_M[]-CONX.SIZ                 "WCTRL/MDR_WB,MSRC/@1,ALU/A-B-CI,ROT/CONX.SIZ,MUX/M.S"
;4378    MDR_M[].AND.OLIT8[]              "WCTRL/MDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/OLIT8,MUX/M.S,ALU/AND"
;4379    MDR_M[].AND.ZLITO[]              "WCTRL/MDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/ZLITO,MUX/M.S,ALU/AND"
;4380    MDR_M[].ANDNOT.R[]               "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/ANDNOT"
;4381    MDR_M[].ANDNOT.ZLITO[]           "WCTRL/MDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/ZLITO,MUX/M.S,ALU/ANDNOT"
;4382    MDR_M[].ASR.P                    "WCTRL/MDR_WB,MSRC/@1,ROT/ASR.M.P,ALPCTL/WX_S"
;4383    MDR_M[].FPLIT                    "WCTRL/MDR_WB,MSRC/@1,ROT/FPLIT,ALPCTL/WX_S"
;4384    MDR_M[].OR.(R[].RR.24)           "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ROT/RR.RR.SIZ,VSIZE/1,DTYPE/LONG,MUX/M.S,ALU/OR"
;4385    MDR_M[].OR.CVTNP(R[])            "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ROT/CVTNP,ALU/OR,MUX/M.S"
;4386    MDR_M[].OR.R[]                   "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/OR"
;4387    MDR_M[].OR.ZLIT24[]              "WCTRL/MDR_WB,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZLIT24,LIT/LITRL,LITRL/@2"
;4388    MDR_M[].RL.16                    "WCTRL/MDR_WB,MSRC/@1,ROT/RR.MM.SIZ,VSIZE/1,DTYPE/WORD,ALPCTL/WX_S"
;4389    MDR_M[].RL.24                    "WCTRL/MDR_WB,MSRC/@1,ROT/RR.MM.SIZ,VSIZE/1,DTYPE/BYTE,ALPCTL/WX_S"
;4390    MDR_M[].RL.8                     "WCTRL/MDR_WB,MSRC/@1,VSIZE/1,DTYPE/LONG,ROT/RR.MM.SIZ,ALPCTL/WX_S"
;4391    MDR_M[].RL.9                     "WCTRL/MDR_WB,ALPCTL/WX_S,ROT/RL.MM.PTE,MSRC/@1"
;4392    MDR_M[].RR.16                    "WCTRL/MDR_WB,MSRC/@1,ROT/RR.MM.SIZ,VSIZE/1,DTYPE/WORD,ALPCTL/WX_S"
;4393    MDR_M[].XOR.R[]                  "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ALU/XOR,MUX/M.R1"
;4394    MDR_M[].XOR.ZLIT12[]             "WCTRL/MDR_WB,MSRC/@1,ROT/ZLIT12,LIT/LITRL,LITRL/@2,MUX/M.S,ALU/XOR"
;4395    MDR_M[]_R[]                      "WCTRL/MDR_WB,MSRC/@1,SPW/MLONG,RSRC/@2,ROT/ZERO,MUX/R.S,ALU/OR"
;4396    MDR_M[]_R[].RR.16                "WCTRL/MDR_WB,MSRC/@1,SPW/MLONG,RSRC/@2,ROT/RR.RR.SIZ,VSIZE/1,DTYPE/WORD,ALPCTL/WX_S"
;4397    MDR_M[]_ZLITO[]                  "WCTRL/MDR_WB,MSRC/@1,SPW/MLONG,LIT/LITRL,LITRL/@2,ROT/ZLITO,ALPCTL/WX_S"
;4398    MDR_Q                            "WCTRL/MDR_WB,RSRC/ZERO,MUX/R.Q,ALU/OR"
;4399    MDR_Q Q_M[]                      "WCTRL/MDR_WB,MSRC/@1,ALPCTL/WX_Q.Q_M"
;4400    MDR_Q_M[]                        "WCTRL/MDR_WB,DQ1/Q_WX,MSRC/@1,ROT/ZERO,MUX/M.S,ALU/OR"
```

Figure 2-13   Macro Expansions 2



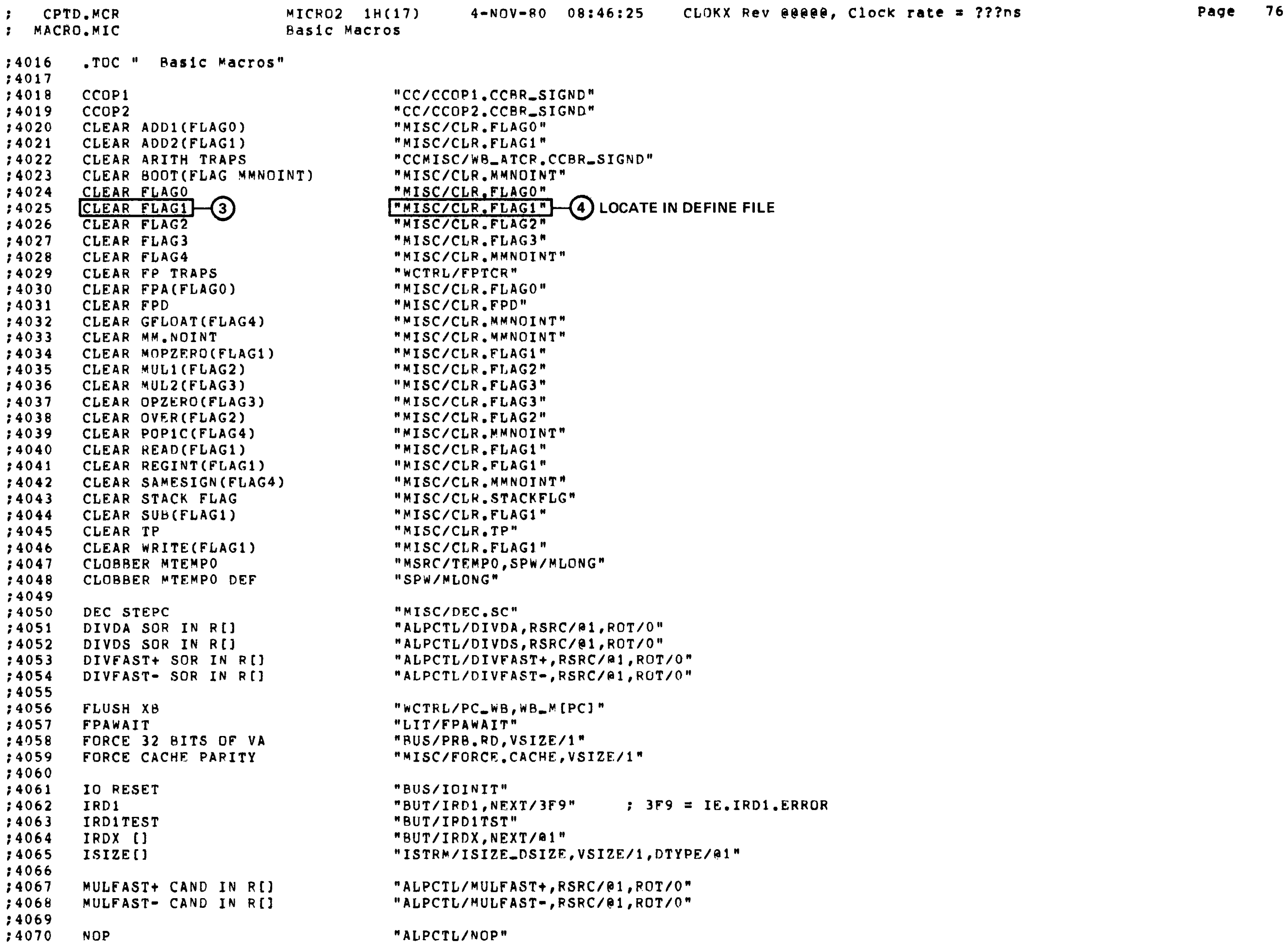
```
;   CPTD.MCR                MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns                 Page   57
;  DEFIN.MIC                Machine Definition         : ISTRM, JSR, LIT, LITRL, LONLIT, MISC

;2971   .TOC "  Machine Definition             : ISTRM, JSR, LIT, LITRL, LONLIT, MISC"
;2972
;2973   ISTRM/=<33:33>,.DEFAULT=0
;2974           NOP=0                                   ;ISIZE IS DETERMINED BY HARDWARE
;2975           ISIZE_DSIZE=1,  .VALIDITY=<V070>        ;ISIZE IS DETERMINED BY DSIZE
;2976
;2977   JSR/=<14:14>,.DEFAULT=0                         ;SUBROUTINE CONTROL
;2978           NOP=0                                   ;NO OPERATION
;2979           PUSH=1                                  ;PUSH CURRENT ADDRESS ON MICRO STACK
;2980
;2981   LIT/=<77:76>,.DEFAULT=0                         ;DEFINE UWORD FIELD INTERPRETATIONS
;2982           NORMAL=0                                ;FIELDS ARE NORMAL
;2983           LITRL=1,        .VALIDITY=<V071>        ;SHORT LITERAL FIELD ENABLED
;2984           FPAWAIT=2                               ;WAIT FOR FPA TO COMPLETE PROCESSING
;2985           LONLIT=3                                ;LONG LITERAL FIELD ENABLED
;2986
;2987   LITRL/=<39:31>,         .VALIDITY=<071>         ;SHORT LITERAL
;2988
;2989   LONLIT/=<62:31>                                 ;LONG LITERAL
;2990
;2991   MISC/=<75:71>,.DEFAULT=10                       ;DEFINE MISC FUNCTIONS
;2992           NOP=10
;2993
;2994           CLR.FLAG0=0                             ;CLEAR FLAG 0
;2995           CLR.FLAG1=1                             ;CLEAR FLAG 1
;2996           CLR.FLAG2=2                             ;CLEAR FLAG 2
;2997           CLR.FLAG3=3                             ;CLEAR FLAG 3
;2998           CLR.MMNOINT=4                           ;CLEAR FLAG 4
;2999           CLR.STACKFLG=5                          ;CLEAR FLAG 5
;3000
;3001           SET.FLAG0=8                             ;SET FLAG 0
;3002           SET.FLAG1=9                             ;SET FLAG 1
;3003           SET.FLAG2=0A                            ;SET FLAG 2
;3004           SET.FLAG3=0B                            ;SET FLAG 3
;3005           SET.MMNOINT=0C                          ;SET FLAG 4
;3006           SET.STACKFLG=0D                         ;SET FLAG 5
;3007
;3008           RSBC=1B                                 ;RETURN AND SUPPRESS BUS CYCLE
;3009           RNUM_2REG=11                            ;RNUM <- COMP MODE SECOND REG
;3010           CLR.TP=12                               ;PSL<TP> <- 0
;3011           CLR.FPD=1C                              ;PSL<FPD> <- 0
;3012           SET.FPD=1D                              ;PSL<FPD> <- 1
;3013           FORCE.TB=1E                             ;FORCE TB PARITY ERROR
;3014           FORCE.CACHE=1F                          ;FORCE CACHE PARITY ERROR
;3015
;3016           DEC.SC=13                               ;STEP CNT <- STEP CNT - 1
;3017           SC_2=14                                 ;STEP CNT <- 2
;3018           SC_6=15                                 ;STEP CNT <- 6
;3019           SC_14=16                                ;STEP CNT <- 14
;3020           SC_30=17                                ;STEP CNT <- 30
;3021
;3022
;3023
;3024
;3025
```

Figure 2-14 Macro Expansions 3



```
;   CPTD.MCR            MICRO2  1H(17)     4-NOV-80  08:46:25     CLOKX Rev @@@@@, Clock rate = ???ns                Page  120
;   INIT.MIC            Initialize Microcode for the Console and Power up

                                 ;6427  IN.VA_0:
                                 ;6428          ;------------------------------;
                                 ;6429          VA_R[ZERO],                    ;VA GETS 0
U 87D, 8800,5BE4,0BD8,4A70,0001  ;6430          RETURN [1]                     ;RETURN+1
                                 ;6431
                                 ;6432  IN.PSL.LONLIT:
                                 ;6433          ;------------------------------;
U 87E, 8800,5BF4,03D4,0070,0864  ;6434          PSL_R[LONLIT]                  ;PSL GETS LONLIT
                                 ;6435
                                 ;6436  =0
                                 ;6437          ;0-----------------------------;
                                 ;6438          PUSH,                          ;JSR
                                 ;6439          STEPC_2,                       ;
                                 ;6440          CRAR_ZLIT16[80],               ;CRAR GETS 2
U 864, 5A00,D370,0340,2470,487C  ;6441          NEXT/IN.PC_0                   ;NOW IF WE CONWRITE
                                 ;6442                                         ;WE WILL WRITE TO RXCS
                                 ;6443
                                 ;6444          ;1-----------------------------;
                                 ;6445          CONREGS_D_M[SISR]_R[ZERO],     ;RXCS GETS 0
U 865, 89FF,5BE6,03D8,2C70,0844  ;6446          DEC STEPC                      ;SISR GETS 0
                                 ;6447
                                 ;6448  =00
                                 ;6449          ;00----------------------------;
                                 ;6450          PUSH,                          ;
                                 ;6451          DEC STEPC,                     ;
                                 ;6452          CRAR_ZLIT16[40],               ;CRAR GETS 1
U 844, D980,D370,0320,2470,4000  ;6453          NEXT/MP.MTPR.TBIA20            ;NOW IF WE CONWRITE
                                 ;6454                                         ;WE WILL WRITE TO TXCS
                                 ;6455
                                 ;6456          ;01----------------------------;
                                 ;6457          PUSH,                          ;
                                 ;6458          D_ZLIT24[80],                  ;
                                 ;6459          CLEAR FLAG1,  3                ;
U 845, 1080,CB72,0340,0470,4000  ;6460          NEXT/MP.MTPR.TBIA20            ;NOW IF WE CONWRITE
                                 ;6461
                                 ;6462          ;10----------------------------;
                                 ;6463          PUSH,NEXT/IN.CLR.CACHE.ROUT,   ;CLEAR THE CACHE ROUTINE
U 846, C800,5BE4,03D8,2C70,4849  ;6464          CONREGS_R[ZERO]                ;TXCS FPDOFFSET GET 0
                                 ;6465
                                 ;6466          ;11----------------------------;
U 847, 8800,5B70,0300,7870,087F  ;6467          SOFTIPR_0                      ;
                                 ;6468          ;------------------------------;
U 87F, 8800,5B70,0300,1670,0866  ;6469          TCSR_0                         ;TCSR_0
                                 ;6470
                                 ;6471  =0      ;0-----------------------------;
                                 ;6472          PUSH,                          ;JSR
                                 ;6473          ASTLVL_[4],                    ;DONE WITH CACHE
U 866, 9800,CB70,0302,7070,487C  ;6474          NEXT/IN.PC_0                   ;ASTLVL GETS 4
                                 ;6475                                         ;CALL PC GETS 0
                                 ;6476                                         ;THIS FLUSHES OUT XB
                                 ;6477
                                 ;6478          ;1-----------------------------;
                                 ;6479          PME_0 FPDOFFSET_3,             ; SET POWER UP CODE FOR VMS RESTART
```

Figure 2-15 Macro Expansions 4



```
;   CPTD.MCR                  MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns               Page   76
;  MACRO.MIC                  Basic Macros

;4016   .TOC "  Basic Macros"
;4017
;4018   CCOP1                          "CC/CCOP1.CCBR_SIGND"
;4019   CCOP2                          "CC/CCOP2.CCBR_SIGND"
;4020   CLEAR ADD1(FLAG0)              "MISC/CLR.FLAG0"
;4021   CLEAR ADD2(FLAG1)              "MISC/CLR.FLAG1"
;4022   CLEAR ARITH TRAPS              "CCMISC/WB_ATCR.CCBR_SIGND"
;4023   CLEAR BOOT(FLAG MMNOINT)       "MISC/CLR.MMNOINT"
;4024   CLEAR FLAG0                    "MISC/CLR.FLAG0"
;4025   CLEAR FLAG1 ─③                 "MISC/CLR.FLAG1" ─④ LOCATE IN DEFINE FILE
;4026   CLEAR FLAG2                    "MISC/CLR.FLAG2"
;4027   CLEAR FLAG3                    "MISC/CLR.FLAG3"
;4028   CLEAR FLAG4                    "MISC/CLR.MMNOINT"
;4029   CLEAR FP TRAPS                 "WCTRL/FPTCR"
;4030   CLEAR FPA(FLAG0)               "MISC/CLR.FLAG0"
;4031   CLEAR FPD                      "MISC/CLR.FPD"
;4032   CLEAR GFLOAT(FLAG4)            "MISC/CLR.MMNOINT"
;4033   CLEAR MM.NOINT                 "MISC/CLR.MMNOINT"
;4034   CLEAR MOPZERO(FLAG1)           "MISC/CLR.FLAG1"
;4035   CLEAR MUL1(FLAG2)              "MISC/CLR.FLAG2"
;4036   CLEAR MUL2(FLAG3)              "MISC/CLR.FLAG3"
;4037   CLEAR OPZERO(FLAG3)            "MISC/CLR.FLAG3"
;4038   CLEAR OVER(FLAG2)              "MISC/CLR.FLAG2"
;4039   CLEAR POP1C(FLAG4)             "MISC/CLR.MMNOINT"
;4040   CLEAR READ(FLAG1)              "MISC/CLR.FLAG1"
;4041   CLEAR REGINT(FLAG1)            "MISC/CLR.FLAG1"
;4042   CLEAR SAMESIGN(FLAG4)          "MISC/CLR.MMNOINT"
;4043   CLEAR STACK FLAG               "MISC/CLR.STACKFLG"
;4044   CLEAR SUB(FLAG1)               "MISC/CLR.FLAG1"
;4045   CLEAR TP                       "MISC/CLR.TP"
;4046   CLEAR WRITE(FLAG1)             "MISC/CLR.FLAG1"
;4047   CLOBBER MTEMPO                 "MSRC/TEMPO,SPW/MLONG"
;4048   CLOBBER MTEMPO DEF             "SPW/MLONG"
;4049
;4050   DEC STEPC                      "MISC/DEC.SC"
;4051   DIVDA SOR IN R[]               "ALPCTL/DIVDA,RSRC/@1,ROT/0"
;4052   DIVDS SOR IN R[]               "ALPCTL/DIVDS,RSRC/@1,ROT/0"
;4053   DIVFAST+ SOR IN R[]            "ALPCTL/DIVFAST+,RSRC/@1,ROT/0"
;4054   DIVFAST- SOR IN R[]            "ALPCTL/DIVFAST-,RSRC/@1,ROT/0"
;4055
;4056   FLUSH XB                       "WCTRL/PC_WB,WB_M[PC]"
;4057   FPAWAIT                        "LIT/FPAWAIT"
;4058   FORCE 32 BITS OF VA            "BUS/PRB.RD,VSIZE/1"
;4059   FORCE CACHE PARITY             "MISC/FORCE.CACHE,VSIZE/1"
;4060
;4061   IO RESET                       "BUS/IOINIT"
;4062   IRD1                           "BUT/IRD1,NEXT/3F9"      ; 3F9 = IE.IRD1.ERROR
;4063   IRD1TEST                       "BUT/IPD1TST"
;4064   IRDX []                        "BUT/IRDX,NEXT/@1"
;4065   ISIZE[]                        "ISTRM/ISIZE_DSIZE,VSIZE/1,DTYPE/@1"
;4066
;4067   MULFAST+ CAND IN R[]           "ALPCTL/MULFAST+,RSRC/@1,ROT/0"
;4068   MULFAST- CAND IN R[]           "ALPCTL/MULFAST-,RSRC/@1,ROT/0"
;4069
;4070   NOP                            "ALPCTL/NOP"
```

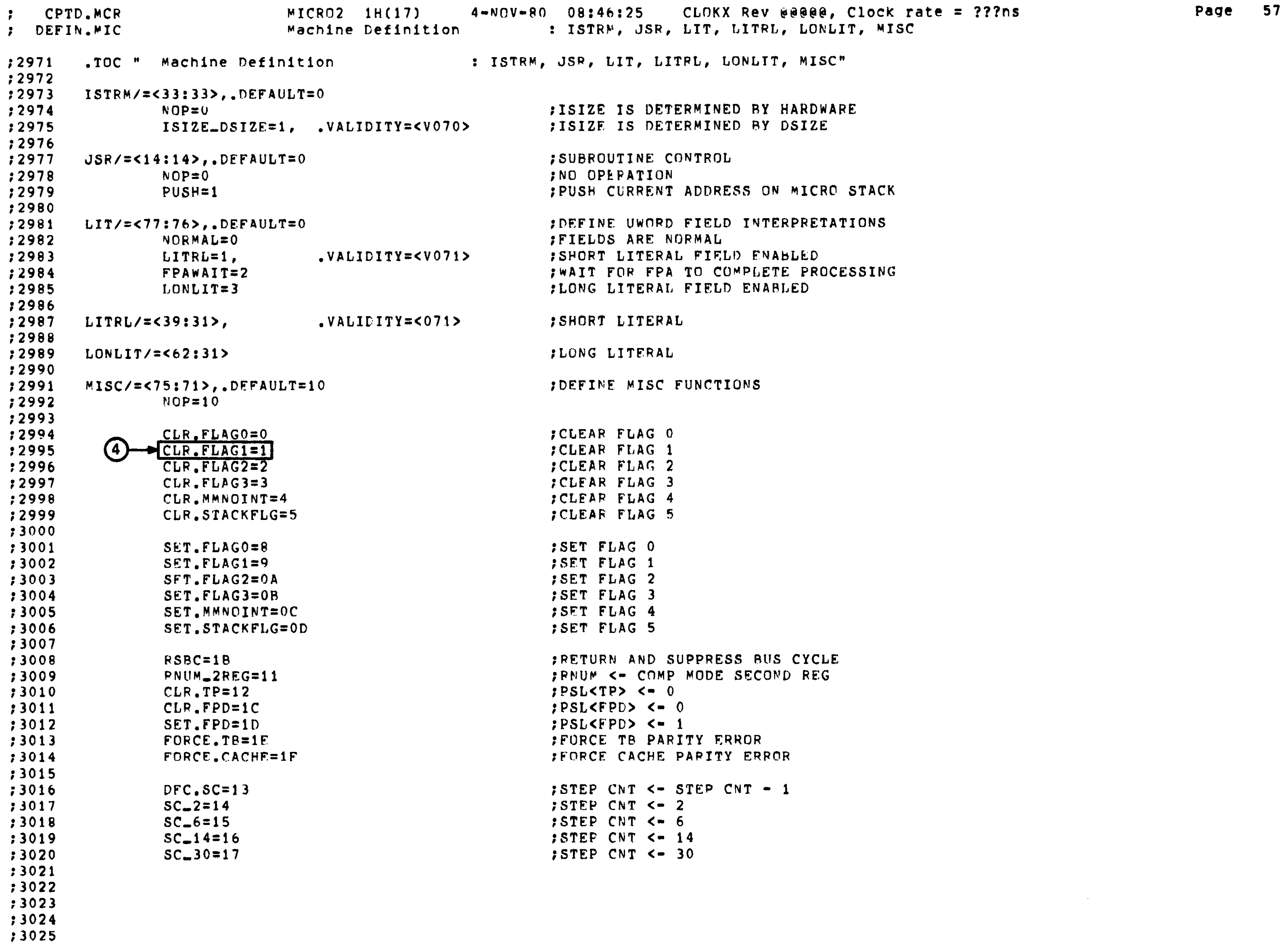
Figure 2-16 Macro Expansions 5

In the MISC field definition in the DEFIN.MIC file shown in Figure 2-17, the binary data in the MISC field of the microword is 00010. At this point we have defined the LIT, LONLIT, and MISC fields of the microword. All other fields assume their default values as defined in the DEFIN.MIC file. The NEXT field of the microinstruction points to the next microinstruction to be executed. If a NEXT field is not specified, the address of the next microinstruction is inserted into bits <13:0>. This is shown in Figure 2-18. The NEXT field in this example indicates (NEXT/IN.PSL.LONLIT).

If the NEXT field is specified, the MICRO2 assembler inserts the address of the label of the next microinstruction into the NEXT bits <13:0> of the microword. In this case the address in control store of the label IN.PSL.LONLIT is inserted into the NEXT field. All labels follow a convention where the first two letters indicate the file in which to find the label. The IN part of the label indicates that this label resides in the INIT microcode file. The list of label abbreviations is shown in the CHARTS.MIC file, called Microcode Label Prefixes.

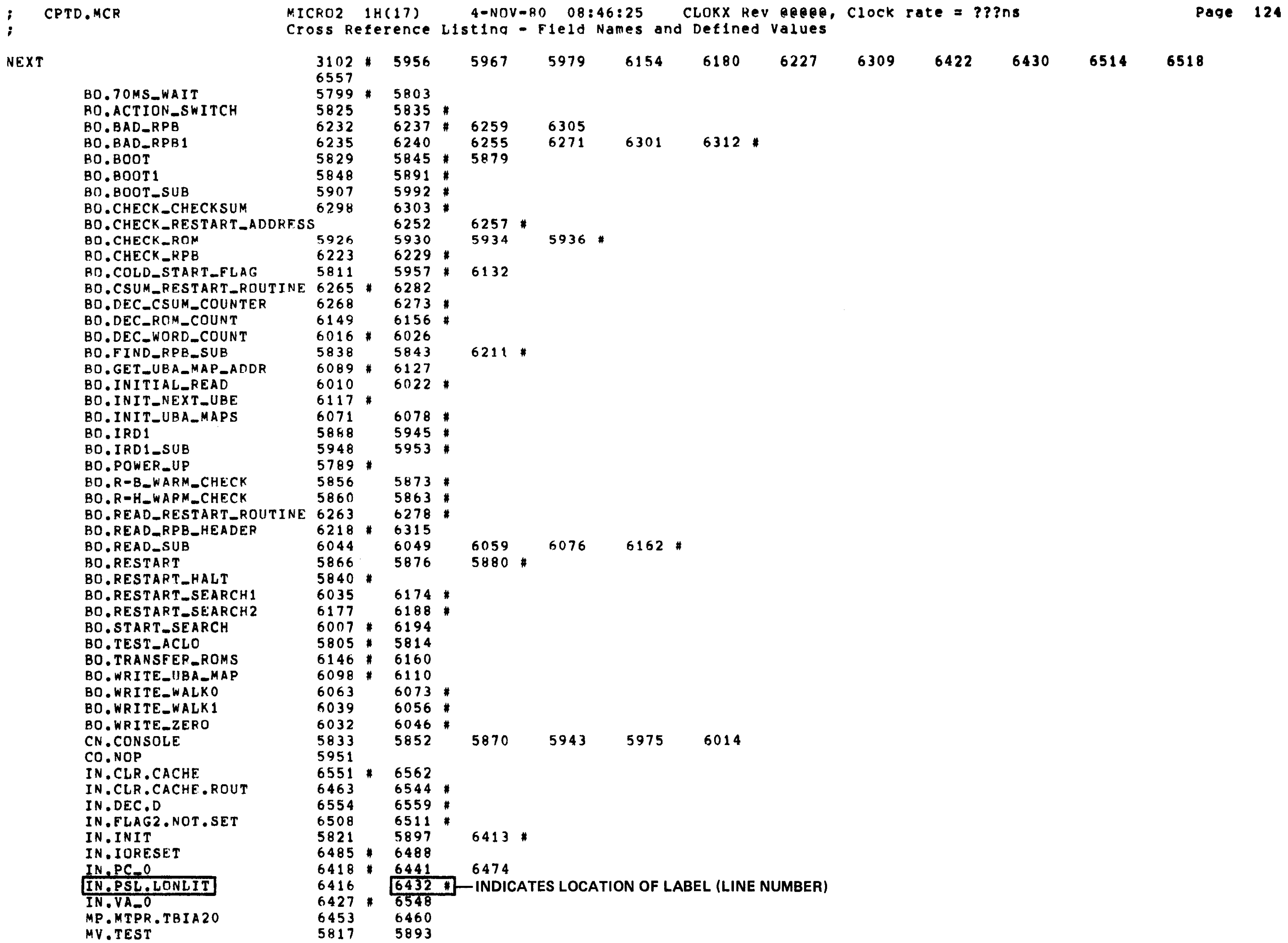
The microinstruction at the label IN.PSL.LONLIT shown on the same page. If it were not here, it would be necessary to cross reference either the location or the label to find the microinstruction at IN.PSL.LONLIT. The label IN.PSL.LONLIT would be cross referenced as follows. There is a file contained in this microcode listing called a CREF. This file is output by the MICRO2 assembler to cross reference labels, macros, and locations. In this case the CREF for Field Names and Defined Values is used. This CREF is located near the back of this listing. The labels are arranged alphabetically. Locate IN.PSL.LONLIT in the listing. Figure 2-18 shows a portion of this CREF. Observe that there are two numbers beside the label. These numbers are the line numbers in the listing where the microinstruction stored at the label IN.PSL.LONLIT is located. The line number with the "#" sign following it is the line number where the label IN.PSL.LONLIT is defined. Any other numbers are the line numbers of microinstructions whose NEXT field points to this label. Refer to Figure 2-19 to see that both these line numbers are on this page.

Another way to locate a microinstruction is to cross reference the NEXT field. The NEXT field can be read directly from the bottom four digits of the microword as shown in Figure 2-19. To locate the the control store address of this microinstruction, the location CREF at the back of the listing must be used. The location CREF cross references all the ROMs. Locate the U ROM location CREF which is for the main control store. The U ROM CREF is reproduced in Figure 2-20. The U ROM location CREF is laid out in 8 columns. To find location 087E, read to the right to the second-to-last column for 087E. The line number of the microinstruction is 6434. Figure 2-21 verifies that the line number is correct. An equal sign (=) that follows a line number indicates that the location is inside a constrained block of locations. MICRO2 control store address allocation is explained in Paragraph 2.2.3.



```
;   CPTD.MCR               MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns          Page   57
;  DEFIN.MIC               Machine Definition         : ISTRM, JSR, LIT, LITRL, LONLIT, MISC

;2971   .TOC "  Machine Definition              : ISTRM, JSR, LIT, LITRL, LONLIT, MISC"
;2972
;2973   ISTRM/=<33:33>,.DEFAULT=0
;2974           NOP=0                                   ;ISIZE IS DETERMINED BY HARDWARE
;2975           ISIZE_DSIZE=1,  .VALIDITY=<V070>        ;ISIZE IS DETERMINED BY DSIZE
;2976
;2977   JSR/=<14:14>,.DEFAULT=0                         ;SUBROUTINE CONTROL
;2978           NOP=0                                   ;NO OPERATION
;2979           PUSH=1                                  ;PUSH CURRENT ADDRESS ON MICRO STACK
;2980
;2981   LIT/=<77:76>,.DEFAULT=0                         ;DEFINE UWORD FIELD INTERPRETATIONS
;2982           NORMAL=0                                ;FIELDS ARE NORMAL
;2983           LITRL=1,        .VALIDITY=<V071>        ;SHORT LITERAL FIELD ENABLED
;2984           FPAWAIT=2                               ;WAIT FOR FPA TO COMPLETE PROCESSING
;2985           LONLIT=3                                ;LONG LITERAL FIELD ENABLED
;2986
;2987   LITRL/=<39:31>,         .VALIDITY=<071>         ;SHORT LITERAL
;2988
;2989   LONLIT/=<62:31>                                 ;LONG LITERAL
;2990
;2991   MISC/=<75:71>,.DEFAULT=10                       ;DEFINE MISC FUNCTIONS
;2992           NOP=10
;2993
;2994           CLR.FLAG0=0                             ;CLEAR FLAG 0
;2995           CLR.FLAG1=1                             ;CLEAR FLAG 1
;2996           CLR.FLAG2=2                             ;CLEAR FLAG 2
;2997           CLR.FLAG3=3                             ;CLEAR FLAG 3
;2998           CLR.MMNOINT=4                           ;CLEAR FLAG 4
;2999           CLR.STACKFLG=5                          ;CLEAR FLAG 5
;3000
;3001           SET.FLAG0=8                             ;SET FLAG 0
;3002           SET.FLAG1=9                             ;SET FLAG 1
;3003           SET.FLAG2=0A                            ;SET FLAG 2
;3004           SET.FLAG3=0B                            ;SET FLAG 3
;3005           SET.MMNOINT=0C                          ;SET FLAG 4
;3006           SET.STACKFLG=0D                         ;SET FLAG 5
;3007
;3008           RSBC=1B                                 ;RETURN AND SUPPRESS BUS CYCLE
;3009           RNUM_2REG=11                            ;RNUM <- COMP MODE SECOND REG
;3010           CLR.TP=12                               ;PSL<TP> <- 0
;3011           CLR.FPD=1C                              ;PSL<FPD> <- 0
;3012           SET.FPD=1D                              ;PSL<FPD> <- 1
;3013           FORCE.TB=1E                             ;FORCE TB PARITY ERROR
;3014           FORCE.CACHE=1F                          ;FORCE CACHE PARITY ERROR
;3015
;3016           DEC.SC=13                               ;STEP CNT <- STEP CNT - 1
;3017           SC_2=14                                 ;STEP CNT <- 2
;3018           SC_6=15                                 ;STEP CNT <- 6
;3019           SC_14=16                                ;STEP CNT <- 14
;3020           SC_30=17                                ;STEP CNT <- 30
;3021
;3022
;3023
;3024
;3025
```

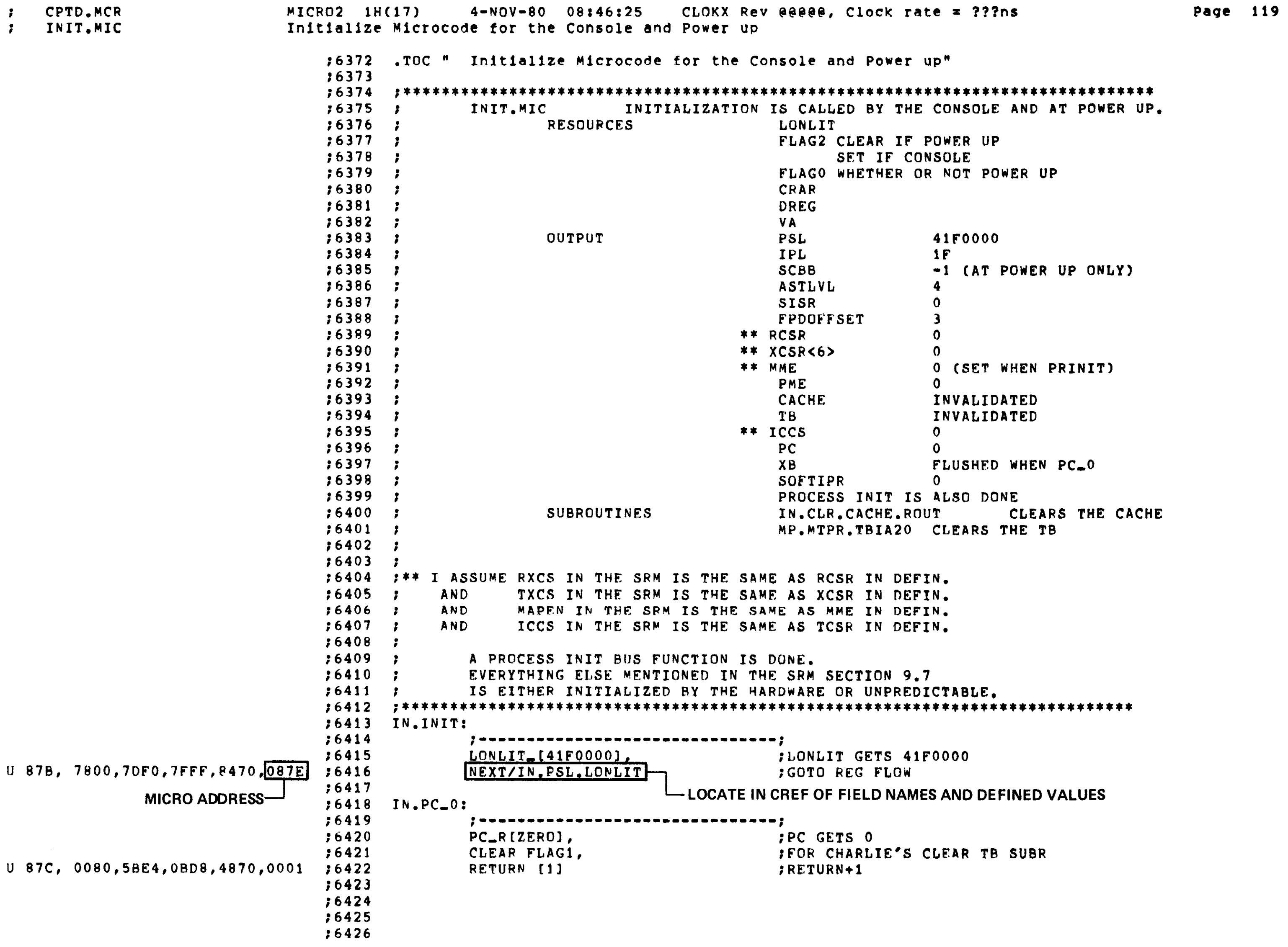
Figure 2-17 Macro Expansions 6



```
;   CPTD.MCR                  MICRO2  1H(17)     4-NOV-80  08:46:25     CLOKX Rev @@@@@, Clock rate = ???ns                  Page  124
;                             Cross Reference Listing - Field Names and Defined Values

NEXT                          3102 #  5956    5967    5979    6154    6180    6227    6309    6422    6430    6514    6518
                              6557
        BO.70MS_WAIT          5799 #  5803
        BO.ACTION_SWITCH      5825    5835 #
        BO.BAD_RPB            6232    6237 #  6259    6305
        BO.BAD_RPB1           6235    6240    6255    6271    6301    6312 #
        BO.BOOT               5829    5845 #  5879
        BO.BOOT1              5848    5891 #
        BO.BOOT_SUB           5907    5992 #
        BO.CHECK_CHECKSUM     6298    6303 #
        BO.CHECK_RESTART_ADDRESS      6252    6257 #
        BO.CHECK_ROM          5926    5930    5934    5936 #
        BO.CHECK_RPB          6223    6229 #
        BO.COLD_START_FLAG    5811    5957 #  6132
        BO.CSUM_RESTART_ROUTINE 6265 # 6282
        BO.DEC_CSUM_COUNTER   6268    6273 #
        BO.DEC_ROM_COUNT      6149    6156 #
        BO.DEC_WORD_COUNT     6016 #  6026
        BO.FIND_RPB_SUB       5838    5843    6211 #
        BO.GET_UBA_MAP_ADDR   6089 #  6127
        BO.INITIAL_READ       6010    6022 #
        BO.INIT_NEXT_UBE      6117 #
        BO.INIT_UBA_MAPS      6071    6078 #
        BO.IRD1               5868    5945 #
        BO.IRD1_SUB           5948    5953 #
        BO.POWER_UP           5789 #
        BO.R-B_WARM_CHECK     5856    5873 #
        BO.R-H_WARM_CHECK     5860    5863 #
        BO.READ_RESTART_ROUTINE 6263  6278 #
        BO.READ_RPB_HEADER    6218 #  6315
        BO.READ_SUB           6044    6049    6059    6076    6162 #
        BO.RESTART            5866    5876    5880 #
        BO.RESTART_HALT       5840 #
        BO.RESTART_SEARCH1    6035    6174 #
        BO.RESTART_SEARCH2    6177    6188 #
        BO.START_SEARCH       6007 #  6194
        BO.TEST_ACLO          5805 #  5814
        BO.TRANSFER_ROMS      6146 #  6160
        BO.WRITE_UBA_MAP      6098 #  6110
        BO.WRITE_WALK0        6063    6073 #
        BO.WRITE_WALK1        6039    6056 #
        BO.WRITE_ZERO         6032    6046 #
        CN.CONSOLE            5833    5852    5870    5943    5975    6014
        CO.NOP                5951
        IN.CLR.CACHE          6551 #  6562
        IN.CLR.CACHE.ROUT     6463    6544 #
        IN.DEC.D              6554    6559 #
        IN.FLAG2.NOT.SET      6508    6511 #
        IN.INIT               5821    5897    6413 #
        IN.IORESET            6485 #  6488
        IN.PC_0               6418 #  6441    6474
        IN.PSL.LONLIT         6416    6432 #
        IN.VA_0               6427 #  6548
        MP.MTPR.TBIA20        6453    6460
        MV.TEST               5817    5893
```

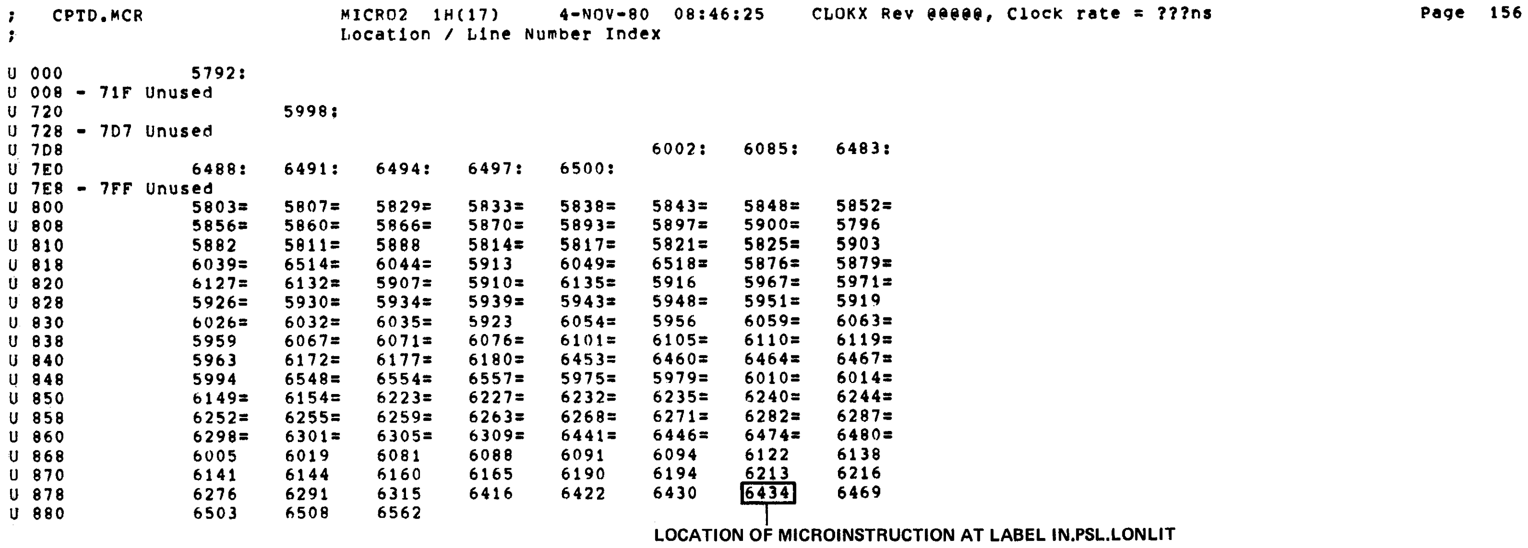
INDICATES LOCATION OF LABEL (LINE NUMBER)

Figure 2-18 Microinstruction Cross Reference 1



```
;   CPTD.MCR               MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns          Page  119
;   INIT.MIC               Initialize Microcode for the Console and Power up

                                  .TOC "  Initialize Microcode for the Console and Power up"

                                  ;****************************************************************************
                                  ;       INIT.MIC        INITIALIZATION IS CALLED BY THE CONSOLE AND AT POWER UP.
                                  ;               RESOURCES                   LONLIT
                                  ;                                           FLAG2 CLEAR IF POWER UP
                                  ;                                                 SET IF CONSOLE
                                  ;                                           FLAG0 WHETHER OR NOT POWER UP
                                  ;                                           CRAR
                                  ;                                           DREG
                                  ;                                           VA
                                  ;               OUTPUT                      PSL             41F0000
                                  ;                                           IPL             1F
                                  ;                                           SCBB            -1 (AT POWER UP ONLY)
                                  ;                                           ASTLVL          4
                                  ;                                           SISR            0
                                  ;                                           FPDOFFSET       3
                                  ;                                       ** RCSR            0
                                  ;                                       ** XCSR<6>         0
                                  ;                                       ** MME             0 (SET WHEN PRINIT)
                                  ;                                           PME             0
                                  ;                                           CACHE           INVALIDATED
                                  ;                                           TB              INVALIDATED
                                  ;                                       ** ICCS            0
                                  ;                                           PC              0
                                  ;                                           XB              FLUSHED WHEN PC_0
                                  ;                                           SOFTIPR         0
                                  ;                                           PROCESS INIT IS ALSO DONE
                                  ;               SUBROUTINES                 IN.CLR.CACHE.ROUT       CLEARS THE CACHE
                                  ;                                           MP.MTPR.TBIA20  CLEARS THE TB
                                  ;
                                  ;
                                  ;** I ASSUME RXCS IN THE SRM IS THE SAME AS RCSR IN DEFIN.
                                  ;    AND     TXCS IN THE SRM IS THE SAME AS XCSR IN DEFIN.
                                  ;    AND     MAPEN IN THE SRM IS THE SAME AS MME IN DEFIN.
                                  ;    AND     ICCS IN THE SRM IS THE SAME AS TCSR IN DEFIN.
                                  ;
                                  ;       A PROCESS INIT BUS FUNCTION IS DONE.
                                  ;       EVERYTHING ELSE MENTIONED IN THE SRM SECTION 9.7
                                  ;       IS EITHER INITIALIZED BY THE HARDWARE OR UNPREDICTABLE.
                                  ;****************************************************************************
                                  IN.INIT:
                                          ;-------------------------------;
                                          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
U 87B, 7800,7DF0,7FFF,8470,087E           NEXT/IN.PSL.LONLIT              ;GOTO REG FLOW

                                  IN.PC_0:
                                          ;-------------------------------;
                                          PC_R[ZERO],                     ;PC GETS 0
                                          CLEAR FLAG1,                    ;FOR CHARLIE'S CLEAR TB SUBR
U 87C, 0080,5BE4,0BD8,4870,0001           RETURN [1]                      ;RETURN+1
```

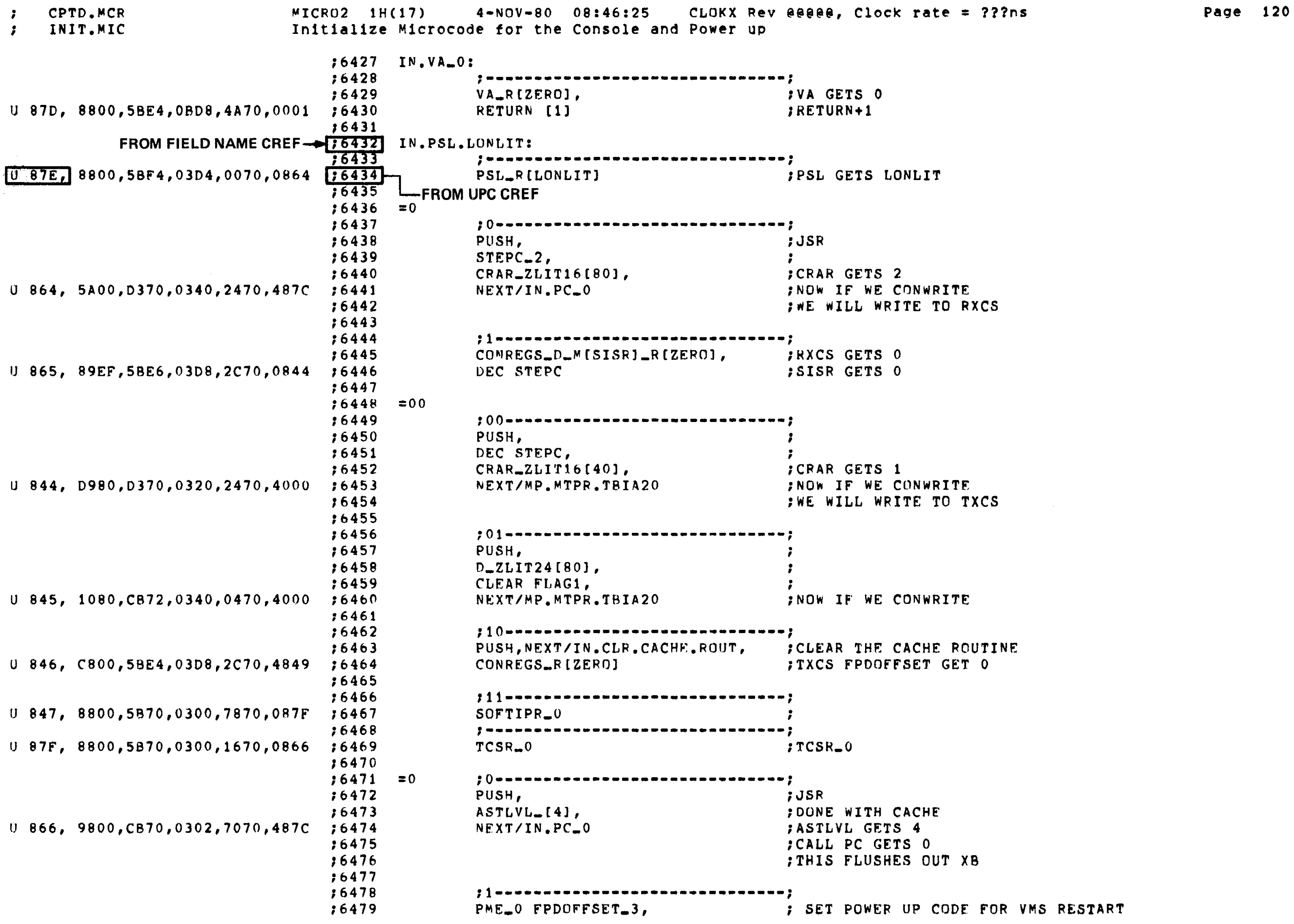
Figure 2-19   NEXT Address Field



```
;   CPTD.MCR                   MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns              Page  156
;                              Location / Line Number Index

U 000           5792:
U 008 - 71F Unused
U 720                   5998:
U 728 - 7D7 Unused
U 7D8                                                   6002:   6085:   6483:
U 7E0           6488:   6491:   6494:   6497:   6500:
U 7E8 - 7FF Unused
U 800           5803=   5807=   5829=   5833=   5838=   5843=   5848=   5852=
U 808           5856=   5860=   5866=   5870=   5893=   5897=   5900=   5796
U 810           5882    5811=   5888    5814=   5817=   5821=   5825=   5903
U 818           6039=   6514=   6044=   5913    6049=   6518=   5876=   5879=
U 820           6127=   6132=   5907=   5910=   6135=   5916    5967=   5971=
U 828           5926=   5930=   5934=   5939=   5943=   5948=   5951=   5919
U 830           6026=   6032=   6035=   5923    6054=   5956    6059=   6063=
U 838           5959    6067=   6071=   6076=   6101=   6105=   6110=   6119=
U 840           5963    6172=   6177=   6180=   6453=   6460=   6464=   6467=
U 848           5994    6548=   6554=   6557=   5975=   5979=   6010=   6014=
U 850           6149=   6154=   6223=   6227=   6232=   6235=   6240=   6244=
U 858           6252=   6255=   6259=   6263=   6268=   6271=   6282=   6287=
U 860           6298=   6301=   6305=   6309=   6441=   6446=   6474=   6480=
U 868           6005    6019    6081    6088    6091    6094    6122    6138
U 870           6141    6144    6160    6165    6190    6194    6213    6216
U 878           6276    6291    6315    6416    6422    6430    6434    6469
U 880           6503    6508    6562
```

LOCATION OF MICROINSTRUCTION AT LABEL IN.PSL.LONLIT

Figure 2-20 Microinstruction Cross Reference 2



```
;   CPTD.MCR              MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns              Page  120
;   INIT.MIC              Initialize Microcode for the Console and Power up

                                       ;6427  IN.VA_0:
                                       ;6428          ;-------------------------------;
                                       ;6429          VA_R[ZERO],                     ;VA GETS 0
U 87D, 8800,5BE4,0BD8,4A70,0001  ;6430          RETURN [1]                      ;RETURN+1
                                       ;6431
                                       ;6432  IN.PSL.LONLIT:
                                       ;6433          ;-------------------------------;
U 87E, 8800,5BF4,03D4,0070,0864  ;6434          PSL_R[LONLIT]                   ;PSL GETS LONLIT
                                       ;6435
                                       ;6436  =0
                                       ;6437          ;0------------------------------;
                                       ;6438          PUSH,                           ;JSR
                                       ;6439          STEPC_2,                        ;
                                       ;6440          CRAR_ZLIT16[80],                ;CRAR GETS 2
U 864, 5A00,D370,0340,2470,487C  ;6441          NEXT/IN.PC_0                    ;NOW IF WE CONWRITE
                                       ;6442                                          ;WE WILL WRITE TO RXCS
                                       ;6443
                                       ;6444          ;1------------------------------;
                                       ;6445          CONREGS_D_M[SISR]_R[ZERO],      ;RXCS GETS 0
U 865, 89EF,5BE6,03D8,2C70,0844  ;6446          DEC STEPC                       ;SISR GETS 0
                                       ;6447
                                       ;6448  =00
                                       ;6449          ;00-----------------------------;
                                       ;6450          PUSH,                           ;
                                       ;6451          DEC STEPC,                      ;
                                       ;6452          CRAR_ZLIT16[40],                ;CRAR GETS 1
U 844, D980,D370,0320,2470,4000  ;6453          NEXT/MP.MTPR.TBIA20             ;NOW IF WE CONWRITE
                                       ;6454                                          ;WE WILL WRITE TO TXCS
                                       ;6455
                                       ;6456          ;01-----------------------------;
                                       ;6457          PUSH,                           ;
                                       ;6458          D_ZLIT24[80],                   ;
                                       ;6459          CLEAR FLAG1,                    ;
U 845, 1080,CB72,0340,0470,4000  ;6460          NEXT/MP.MTPR.TBIA20             ;NOW IF WE CONWRITE
                                       ;6461
                                       ;6462          ;10-----------------------------;
                                       ;6463          PUSH,NEXT/IN.CLR.CACHE.ROUT,    ;CLEAR THE CACHE ROUTINE
U 846, C800,5BE4,03D8,2C70,4849  ;6464          CONREGS_R[ZERO]                 ;TXCS FPDOFFSET GET 0
                                       ;6465
                                       ;6466          ;11-----------------------------;
U 847, 8800,5B70,0300,7870,087F  ;6467          SOFTIPR_0                       ;
                                       ;6468          ;-------------------------------;
U 87F, 8800,5B70,0300,1670,0866  ;6469          TCSR_0                          ;TCSR_0
                                       ;6470
                                       ;6471  =0      ;0------------------------------;
                                       ;6472          PUSH,                           ;JSR
                                       ;6473          ASTLVL_[4],                     ;DONE WITH CACHE
U 866, 9800,CB70,0302,7070,487C  ;6474          NEXT/IN.PC_0                    ;ASTLVL GETS 4
                                       ;6475                                          ;CALL PC GETS 0
                                       ;6476                                          ;THIS FLUSHES OUT XB
                                       ;6477
                                       ;6478          ;1------------------------------;
                                       ;6479          PME_0 FPDOFFSET_3,              ; SET POWER UP CODE FOR VMS RESTART
```

Figure 2-21 Microinstruction Cross Reference 3

2.2.3 MICRO2 Address Allocation

The MICRO2 assembler assigns control store locations according to four priorities established by the firmware designer when a label, region, or constraint block for addresses is specified. The four control store allocation priorities are as follows.

1. Absolute Assignment – A label specifies an absolute control store address.
2. Region Directive – Allocates the control store microcode specific regions that are not absolutely assigned.
3. Constraint Block – Allocates sections of control store contiguous locations that are not absolutely assigned. The constraint block may be imbedded in a region.
4. Unconstrained – This is any location that is not absolutely assigned or constrained. It may be within a region. The assembler directive .NEXTADDRESS points the NEXT address field to the next microinstruction if no NEXT field is specified. The location of the unconstrained microinstruction is selected by the MICRO2 assembler after all absolute assignments and constraint blocks are determined.

An example of absolute assignment is shown in the Figure 2-22. Note that there is an absolute address assignment that forces the microinstruction at BO.POWER-UP to be stored at control store address 0000. You can verify this by looking at the U ROM binary shown on the left side of Figure 2-22. The control store address of BO.POWER-UP is absolute address 0000.

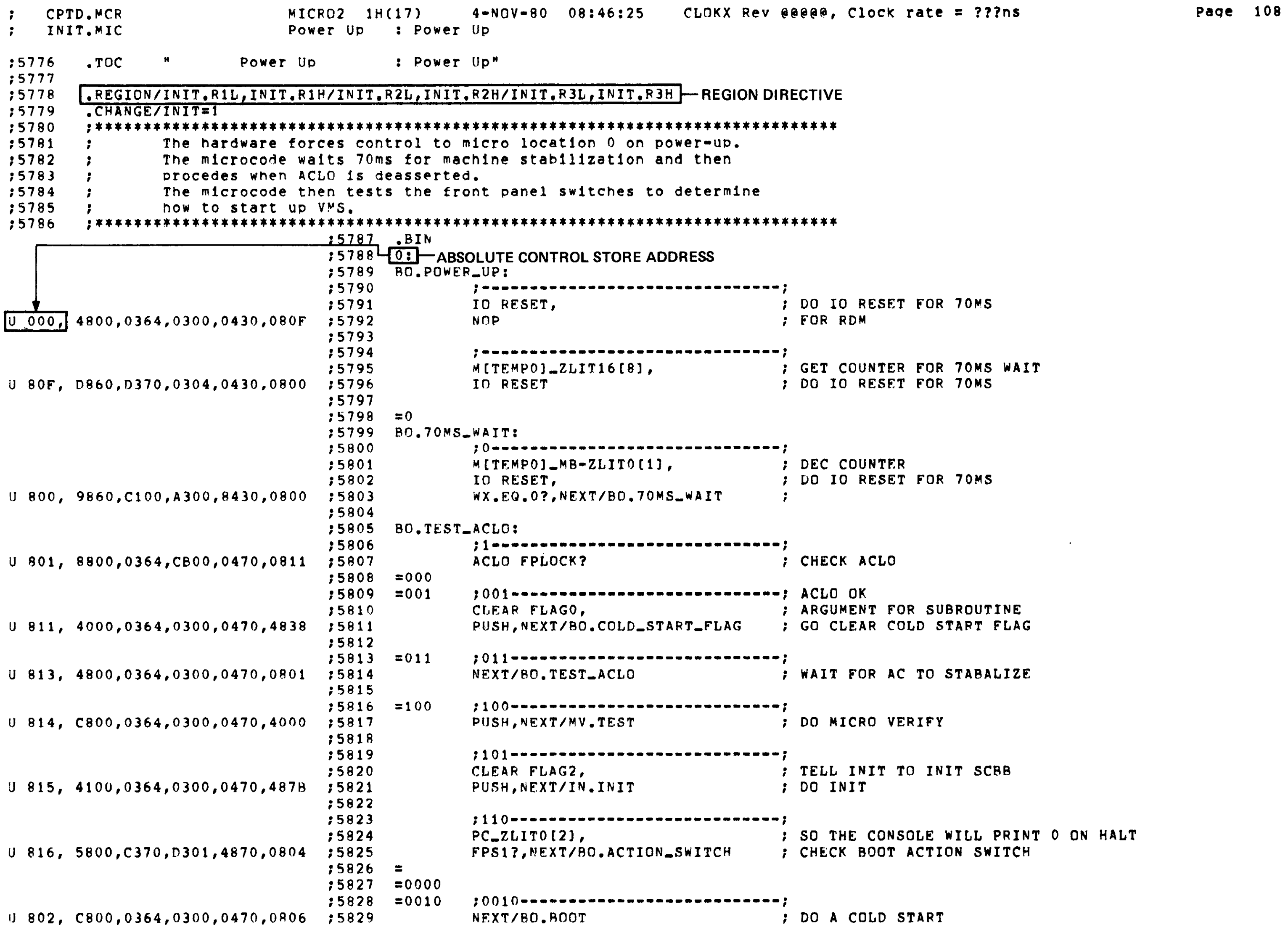
An example of the region directive is shown on Figure 2-22. This is a region directive macro that must be defined in the REGION.MIC file. Figure 2-23 shows how the region directive is developed. The SET directive equates values with the names in the table.

```
.SET/INIT.R1L=800
.SET/INIT.R1H=882
.SET/INIT.R2L=800
.SET/INIT.R2H=882
.SET/INIT.R3L=800
.SET/INIT.R3H=882
```

These values can be substituted for the expressions in Figure 2-23 to clarify the meaning. The region directive that is enclosed in the box could also be stated as:

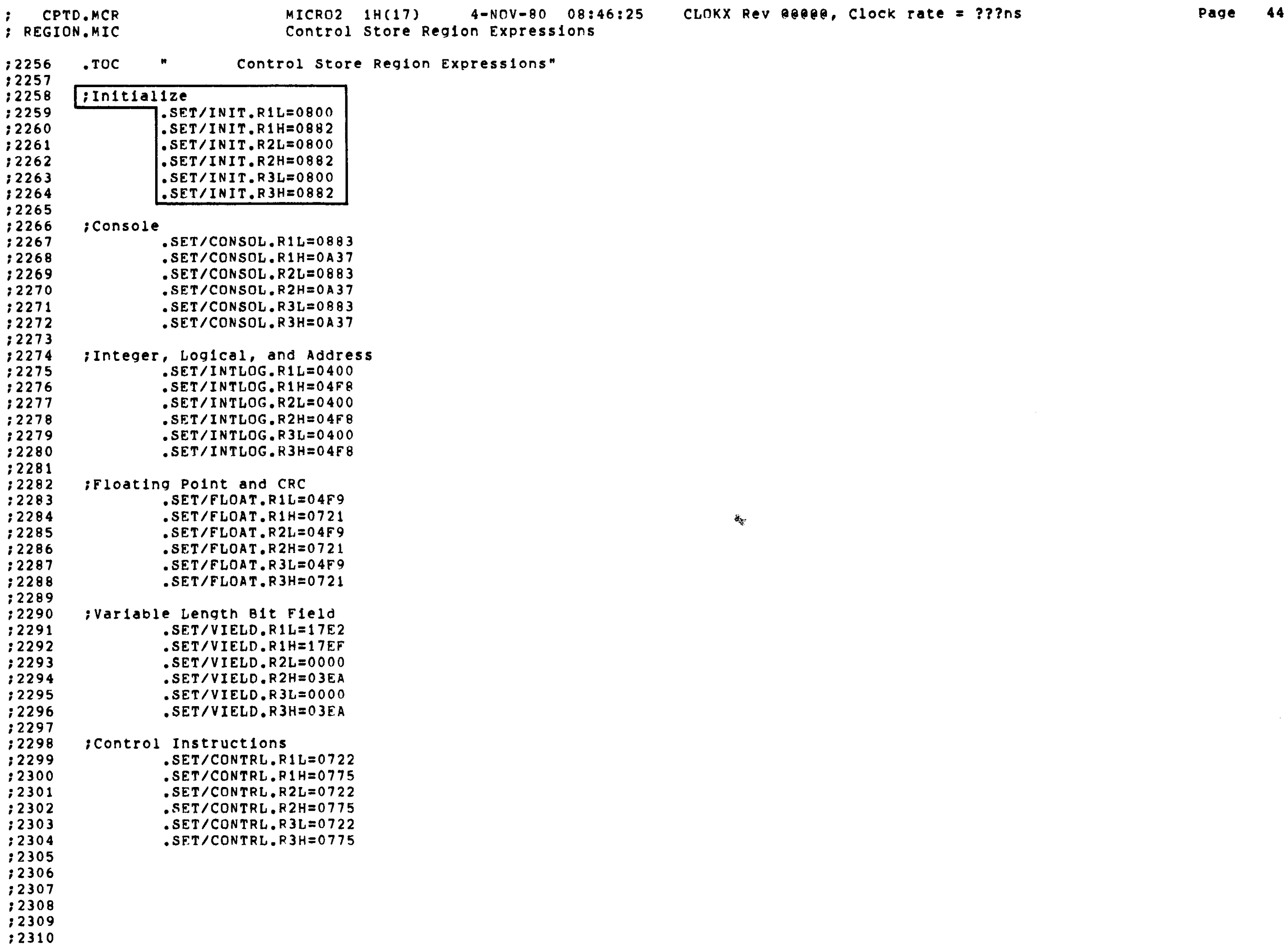
```
.REGION/800,882/800,882/800,882
```

This statement directs the MICRO2 assembler to store the microinstructions that follow this statement into the region of the control store from 800 to 882 (hex). Optionally, if there is not enough room in this region, it stores the balance in 800 to 882. And in the event there is still not enough room in 800 to 882, it stores the rest of the microcode in the region 800 to 882. Absolute assignments have priority over the region directive, so all locations that are not absolutely assigned are available within the region selected. The microinstruction that immediately follows the region directive at the label BO.70MS_WAIT is shown in Figure 2-22 at control store address 800 (hex). The region directive is particularly useful for debugging microcode and allocating patch space.



```
;   CPTD.MCR               MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns                Page  108
;   INIT.MIC               Power Up   : Power Up

;5776   .TOC    "       Power Up        : Power Up"
;5777
;5778   .REGION/INIT.R1L,INIT.R1H/INIT.R2L,INIT.R2H/INIT.R3L,INIT.R3H --REGION DIRECTIVE
;5779   .CHANGE/INIT=1
;5780   ;*************************************************************************
;5781   ;       The hardware forces control to micro location 0 on power-up.
;5782   ;       The microcode waits 70ms for machine stabilization and then
;5783   ;       procedes when ACLO is deasserted.
;5784   ;       The microcode then tests the front panel switches to determine
;5785   ;       how to start up VMS.
;5786   ;*************************************************************************
                                   ;5787  .BIN
                                   ;5788  0: --ABSOLUTE CONTROL STORE ADDRESS
                                   ;5789  BO.POWER_UP:
                                   ;5790          ;-----------------------------;
                                   ;5791          IO RESET,                     ; DO IO RESET FOR 70MS
U 000, 4800,0364,0300,0430,080F    ;5792          NOP                           ; FOR RDM
                                   ;5793
                                   ;5794          ;-----------------------------;
                                   ;5795          M[TEMPO]_ZLIT16[8],           ; GET COUNTER FOR 70MS WAIT
U 80F, D860,D370,0304,0430,0800    ;5796          IO RESET                      ; DO IO RESET FOR 70MS
                                   ;5797
                                   ;5798  =0
                                   ;5799  BO.70MS_WAIT:
                                   ;5800          ;0----------------------------;
                                   ;5801          M[TEMPO]_MB-ZLIT0[1],         ; DEC COUNTER
                                   ;5802          IO RESET,                     ; DO IO RESET FOR 70MS
U 800, 9860,C100,A300,8430,0800    ;5803          WX.EQ.0?,NEXT/BO.70MS_WAIT    ;
                                   ;5804
                                   ;5805  BO.TEST_ACLO:
                                   ;5806          ;1----------------------------;
U 801, 8800,0364,CB00,0470,0811    ;5807          ACLO FPLOCK?                  ; CHECK ACLO
                                   ;5808  =000
                                   ;5809  =001    ;001--------------------------; ACLO OK
                                   ;5810          CLEAR FLAG0,                  ; ARGUMENT FOR SUBROUTINE
U 811, 4000,0364,0300,0470,4838    ;5811          PUSH,NEXT/BO.COLD_START_FLAG  ; GO CLEAR COLD START FLAG
                                   ;5812
                                   ;5813  =011    ;011--------------------------;
U 813, 4800,0364,0300,0470,0801    ;5814          NEXT/BO.TEST_ACLO             ; WAIT FOR AC TO STABALIZE
                                   ;5815
                                   ;5816  =100    ;100--------------------------;
U 814, C800,0364,0300,0470,4000    ;5817          PUSH,NEXT/MV.TEST             ; DO MICRO VERIFY
                                   ;5818
                                   ;5819          ;101--------------------------;
                                   ;5820          CLEAR FLAG2,                  ; TELL INIT TO INIT SCBB
U 815, 4100,0364,0300,0470,487B    ;5821          PUSH,NEXT/IN.INIT             ; DO INIT
                                   ;5822
                                   ;5823          ;110--------------------------;
                                   ;5824          PC_ZLIT0[2],                  ; SO THE CONSOLE WILL PRINT 0 ON HALT
U 816, 5800,C370,D301,4870,0804    ;5825          FPS1?,NEXT/BO.ACTION_SWITCH   ; CHECK BOOT ACTION SWITCH
                                   ;5826  =
                                   ;5827  =0000
                                   ;5828  =0010   ;0010-------------------------;
U 802, C800,0364,0300,0470,0806    ;5829          NEXT/BO.BOOT                  ; DO A COLD START
```

Figure 2-22 Region Directive



```
;   CPTD.MCR               MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns              Page    44
; REGION.MIC               Control Store Region Expressions

        .TOC    "       Control Store Region Expressions"

        ;Initialize
                .SET/INIT.R1L=0800
                .SET/INIT.R1H=0882
                .SET/INIT.R2L=0800
                .SET/INIT.R2H=0882
                .SET/INIT.R3L=0800
                .SET/INIT.R3H=0882

        ;Console
                .SET/CONSOL.R1L=0883
                .SET/CONSOL.R1H=0A37
                .SET/CONSOL.R2L=0883
                .SET/CONSOL.R2H=0A37
                .SET/CONSOL.R3L=0883
                .SET/CONSOL.R3H=0A37

        ;Integer, Logical, and Address
                .SET/INTLOG.R1L=0400
                .SET/INTLOG.R1H=04F8
                .SET/INTLOG.R2L=0400
                .SET/INTLOG.R2H=04F8
                .SET/INTLOG.R3L=0400
                .SET/INTLOG.R3H=04F8

        ;Floating Point and CRC
                .SET/FLOAT.R1L=04F9
                .SET/FLOAT.R1H=0721
                .SET/FLOAT.R2L=04F9
                .SET/FLOAT.R2H=0721
                .SET/FLOAT.R3L=04F9
                .SET/FLOAT.R3H=0721

        ;Variable Length Bit Field
                .SET/VIELD.R1L=17E2
                .SET/VIELD.R1H=17EF
                .SET/VIELD.R2L=0000
                .SET/VIELD.R2H=03EA
                .SET/VIELD.R3L=0000
                .SET/VIELD.R3H=03EA

        ;Control Instructions
                .SET/CONTRL.R1L=0722
                .SET/CONTRL.R1H=0775
                .SET/CONTRL.R2L=0722
                .SET/CONTRL.R2H=0775
                .SET/CONTRL.R3L=0722
                .SET/CONTRL.R3H=0775
```

Figure 2-23 Region Directive Macros

The next highest priority is the constraint block. The microprogrammer must be able to direct the MICRO2 assembler to provide blocks of control store locations so that microbranch destinations will have the right bit set or clear for the particular microbranch condition. Figure 2-24 illustrates several constraint blocks in use. Line 5807 contains a branching macro that tests ACLO and front panel keyswitch position. The macro definition, which can be found in the branching macro file, is

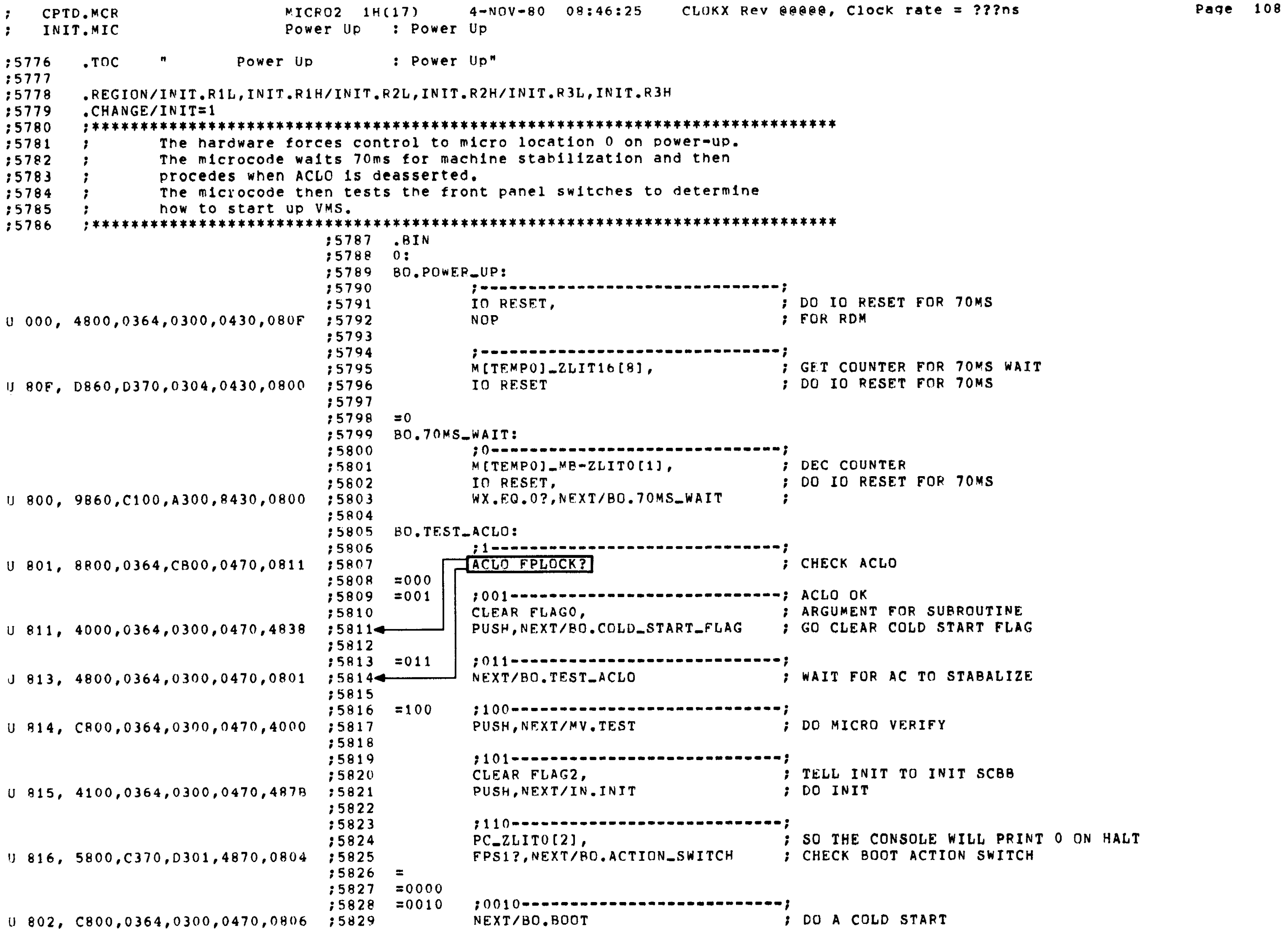
ACLO FPLOCK?   "BUT/FPS3"

where CSA 1 and CSA 0 are modified as follows.

CSA 1 = 1	if ACLO is asserted
CSA 0 = 1	if the 5 position keyswitch is in secure position

This microword is a NOP, other branching on the state of ACLO and front panel secure switch. The two targets are constrained such that control store address bit <0> is irrelevant. This allows only ACLO to be a microbranch condition. If ACLO is not asserted, control store address bit <1> is modified, changing the target address to 0813. This is the loop used to wait until ACLO is negated. The microsequence would be a loop from 0801 to 0813 and back to 0801 while ACLO is asserted. Once ACLO is negated, the microcode would execute the instruction at microaddress 0838. The constraint block allocates eight locations for this group of microwords. The first location (=000) is not used because bit <0> was not required.

The lowest priority address assignment is the unconstrained assignment. In this instance the control store address for the microinstruction is selected after all absolute assignments and constraint blocks have been allocated for the microcode in this particular region.



```
;   CPTD.MCR                 MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns                 Page  108
;   INIT.MIC                 Power Up    : Power Up

;5776   .TOC    "       Power Up        : Power Up"
;5777
;5778   .REGION/INIT.R1L,INIT.R1H/INIT.R2L,INIT.R2H/INIT.R3L,INIT.R3H
;5779   .CHANGE/INIT=1
;5780   ;*****************************************************************************
;5781   ;       The hardware forces control to micro location 0 on power-up.
;5782   ;       The microcode waits 70ms for machine stabilization and then
;5783   ;       procedes when ACLO is deasserted.
;5784   ;       The microcode then tests the front panel switches to determine
;5785   ;       how to start up VMS.
;5786   ;*****************************************************************************
                                  ;5787  .BIN
                                  ;5788  0:
                                  ;5789  BO.POWER_UP:
                                  ;5790          ;-------------------------------;
                                  ;5791          IO RESET,                       ; DO IO RESET FOR 70MS
U 000, 4800,0364,0300,0430,080F   ;5792          NOP                             ; FOR RDM
                                  ;5793
                                  ;5794          ;-------------------------------;
                                  ;5795          M[TEMP0]_ZLIT16[8],             ; GET COUNTER FOR 70MS WAIT
U 80F, D860,D370,0304,0430,0800   ;5796          IO RESET                        ; DO IO RESET FOR 70MS
                                  ;5797
                                  ;5798  =0
                                  ;5799  BO.70MS_WAIT:
                                  ;5800          ;0------------------------------;
                                  ;5801          M[TEMP0]_MB-ZLIT0[1],           ; DEC COUNTER
                                  ;5802          IO RESET,                       ; DO IO RESET FOR 70MS
U 800, 9860,C100,A300,8430,0800   ;5803          WX.EQ.0?,NEXT/BO.70MS_WAIT      ;
                                  ;5804
                                  ;5805  BO.TEST_ACLO:
                                  ;5806          ;1------------------------------;
U 801, 8800,0364,CB00,0470,0811   ;5807          ACLO FPLOCK?                    ; CHECK ACLO
                                  ;5808  =000
                                  ;5809  =001    ;001----------------------------; ACLO OK
                                  ;5810          CLEAR FLAG0,                    ; ARGUMENT FOR SUBROUTINE
U 811, 4000,0364,0300,0470,4838   ;5811          PUSH,NEXT/BO.COLD_START_FLAG    ; GO CLEAR COLD START FLAG
                                  ;5812
                                  ;5813  =011    ;011----------------------------;
U 813, 4800,0364,0300,0470,0801   ;5814          NEXT/BO.TEST_ACLO               ; WAIT FOR AC TO STABALIZE
                                  ;5815
                                  ;5816  =100    ;100----------------------------;
U 814, C800,0364,0300,0470,4000   ;5817          PUSH,NEXT/MV.TEST               ; DO MICRO VERIFY
                                  ;5818
                                  ;5819          ;101----------------------------;
                                  ;5820          CLEAR FLAG2,                    ; TELL INIT TO INIT SCBB
U 815, 4100,0364,0300,0470,487B   ;5821          PUSH,NEXT/IN.INIT               ; DO INIT
                                  ;5822
                                  ;5823          ;110----------------------------;
                                  ;5824          PC_ZLIT0[2],                    ; SO THE CONSOLE WILL PRINT 0 ON HALT
U 816, 5800,C370,D301,4870,0804   ;5825          FPS1?,NEXT/BO.ACTION_SWITCH     ; CHECK BOOT ACTION SWITCH
                                  ;5826  =
                                  ;5827  =0000
                                  ;5828  =0010   ;0010---------------------------;
U 802, C800,0364,0300,0470,0806   ;5829          NEXT/BO.BOOT                    ; DO A COLD START
```

Figure 2-24 Addressing Constraints

2.2.4 Microroutine Analysis

This paragraph analyzes microroutines, using the interpretations of microcode macro expansion and control store address allocations described in Paragraphs 2.2.1–2.2.4. This discussion is based on microcode listing version CMT047 or later of the INIT.MIC file. Several microinstructions executed during powerup are described.

The instant the operator applies power to the machine, the microcode begins execution from control store address 0000. The first microinstruction of the power microcode is as follows.

```
0:
BO.POWER_UP:
        ;-------------------   ;
        IO RESET,               ; DO IO RESET FOR 70MS
        NOP                     ; FOR RDM
```

The first microinstruction is assigned an absolute address of 0000. The macro IO RESET is a Basic macro that causes a Unibus INIT to be generated, and the macro NOP is a Basic macro that forces the default ALPCTL field value. This is the first microinstruction executed after the negation of DCLO. This microinstruction must always be located at absolute address 0000 because of the design of the microsequencer logic. The next microinstruction establishes a 250-ms wait loop to wait to test ACLO.

```
;-------------------   ; GET COUNTER FOR 70MS
M[TEMP0]_ZLIT16[8],     ; WAIT, DO IO RESET FOR
IO RESET                ; 70 MS
```

In the above microinstruction, MTEMP0 is loaded with the literal 8 zero-extended and rotated left 16 bit positions. The contents of MTEMP0 at the end of this microinstruction would be 00080000. 80000 (hex) times 480 ns is approximately 250 ms (despite what microcode listing indicates in the comment section). IO RESET is asserted again. This microinstruction sets up the memory initialization loop. The next microinstruction contains the microbranch to fall out of the memory initialization ROM state.

```
=0
BO.70MS_WAIT:
        ;0-------------------  ; DEC COUNTER
        M[TEMP0]_MB-ZLIT0[1],   ; DO IO RESET FOR 70MS
        IO RESET,
        WX.EQ.0?,NEXT/BO.70MS_WAIT
```

This microinstruction is in a constraint block because this is the microbranch on the WX.EQ.0? condition that modifies bit <0> of the CS address lines. The ROM address selected by the assembler was 800. The microcode reads MTEMP0, subtracts 1 from the contents, and microbranches to 801 if the WBus is zero. This loop is executed 80000 (hex) times, or (2**19)-1 times, or 524287 decimal iterations. 524287 times 480 ns is approximately 250 ms. At the end of the loop when MTEMP0 is equal to zero, the next microinstruction is executed.

```
BO.TEST_ACLO:
        ;1-------------------; CHECK ACLO
        ACLO FPLOCK?
```

This microinstruction is used to microbranch on ACLO. The next group of microinstructions are in a constraint block of eight words. The first location in the block the microprogrammer uses is 1. This essentially means that bit <0> of the BUT micro-order at BO.TEST_ACLO is excluded as a target in the microbranch. The BUT micro-order for ACLO FPLOCK? is BUT/FPS3 and this modifies bits <1:0> on the CS address lines as follows.

CSA <1>	CSA <0>
ACLO	FPLOCK

Bit <0> is asserted if the KEY switch on the operator control panel is in either of the SECURE positions. Since bit <0> is constrained out, it has no effect on the microbranch. If ACLO is asserted the next microinstruction executed is as follows.

```
=011   ;011------------------; WAIT FOR AC TO STAB-
       NEXT/BO.TEST_ACLO     ; ILIZE
```

This sends the microcode back to the microbranch at BO.TEST_ACLO. This is the loop the microcode uses until ACLO is negated. When ACLO is negated, approximately 838 ms after DCLO is negated for memory initialization, the next microinstruction is executed.

```
=001   ;001-------------------------------; ACLO OK
       CLEAR FLAG0,                       ; ARGUMENT FOR SUBROUT
       PUSH, NEXT/BO.COLD_START_FLAG      ; CLEAR COLD START FLAG
```

At this point after powerup, the 250 ms wait is done and ACLO has been tested. If ACLO is negated, the above microinstruction is executed. This instruction calls a subroutine that clears the cold-start flag, which is used to restart the system after a power fail. At powerup this flag is always clear. The address of this microinstruction is saved on the microstack. The last microinstruction of the clear cold-start flag microroutine does a RETURN [+3]. That microroutine is not traced here. When the push was done, address 0811 was written on the microstack. The last microinstruction in the cold-start flag routine does a return +3, which pops the 0811 off the microstack and ADDS 3. The return microaddress is 0814. The microinstruction at 0814 is as follows.

```
=100   ;100-------------------;
       PUSH,NEXT/MV.TEST      ; DO MICRO VERIFY
```

This microsubroutine call is to the Micro-Verify routine that checks CPU buses, registers, scratchpads, and memory interface logic. A percent sign (%) is printed at the console terminal at the beginning of Micro-Verify and at the successful completion. At the console terminal you should observe the two symbols.

```
%%
```

After the microverification of the processor is complete, the INIT microroutine is called. The return from Micro-Verify is a return +1 to address 0815.

```
;101-------------------------;
CLEAR FLAG2,                 ; TELL INIT TO INIT SCBB
PUSH, NEXT/IN.INIT           ; DO INIT
```

The INIT microroutine clears the data cache, invalidates all translation buffer locations, sets the PSL to 041F0000, sets the ASTLVL to 4, and does a CPU and I/O initialization. At the end of the INIT microroutine a return +1 is done to come back to 0816. At this point, the microverification and initialization routines are done and the next step is to restart the system based on the position of the POWER ON ACTION and DEVICE switches. There are four possible system start-up procedures.

ENTER CONSOLE MODE

ATTEMPT WARM RESTART, If restart fails enter console mode.

ATTEMPT WARM RESTART, If restart fails, boostrap system according to DEVICE switch.

BOOTSTRAP SYSTEM

The next microinstruction cases on the POWER ON ACTION switch to do one of the four procedures outlined above.

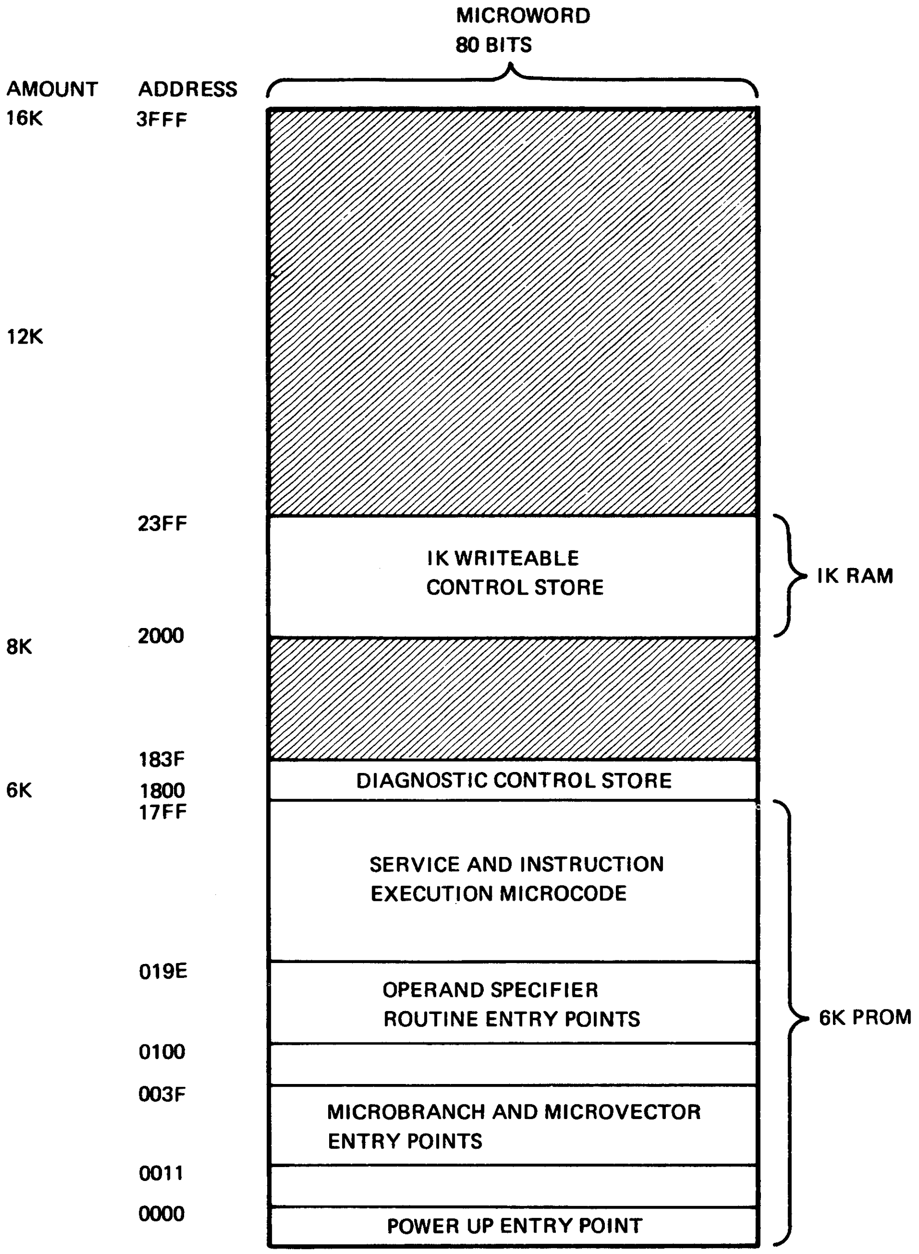
```
;110 -------------------       ; SO CONSOLE PRINTS 0
PC_ZLIT0[2],                   ; ON HALT
FPS1?, NEXT/BO.ACTION_SWITCH   ; CHECK BOOT ACTION
                               ; SWITCH
```

The program counter is loaded with 2 because the console subtracts 2 before typing the contents of the PC. At powerup the PC is cleared. The BUT micro-order is FPS1, which does a 4-way branch on the position of the POWER ON ACTION switch. At this point the flow can go in four ways.

2.3 MICROSEQUENCER AND CONTROL STORE SUBSYSTEM

The microsequencer and control store subsystem are interlocked with each other and are interdependent. The VAX-11/750 CPU microprogram subsystem consists of a microsequencer that addresses the control store for the next microinstruction and a PROM control store that contains the microinstructions. The microsequencer and control store subsystem address up to 16K locations of microinstructions. Figure 2-25 shows how the 16K locations are allocated in the current design of the CPU. Addresses 0 through 17FF are the PROM control store located on the CCS module in slot 5 of the CPU. Addresses 1800 to 183F are used for microcode execution only. The DCS is located on the RDM module. The RDM has its own microsequencer and timing logic and does not require the VAX-11/750 CPU microsequencer to be functional. Addresses 2000 through 23FF are assigned to the optional 1K WCS module that attaches as a daughter board to the CCS. At present, the rest of the control store address space is unassigned. The 6K × 80 CCS PROM functional allocation is shown at the bottom of Figure 2-25.

Figure 2-26 is a block diagram of the microsequencer logic showing the gate arrays implemented in the design. The four gate arrays are SAC, MSQ, PHB, and IRD. The most basic part of the microsequencer is shown at the upper right corner of the figure. This is bit <5:0> of the NEXT address from CCS going into the NEXT field latch. The output of the latch goes into the MSQ gate array adder to generate the control store address bits <5:0>. Bits <13:6> of the NEXT field from the CCS are latched on the CCS module. The output of that latch is recieved on the DPM module to generate bit <13:6> of the next control store address.



TK-1983

Figure 2-25 CCS Control Store Memory Allocation

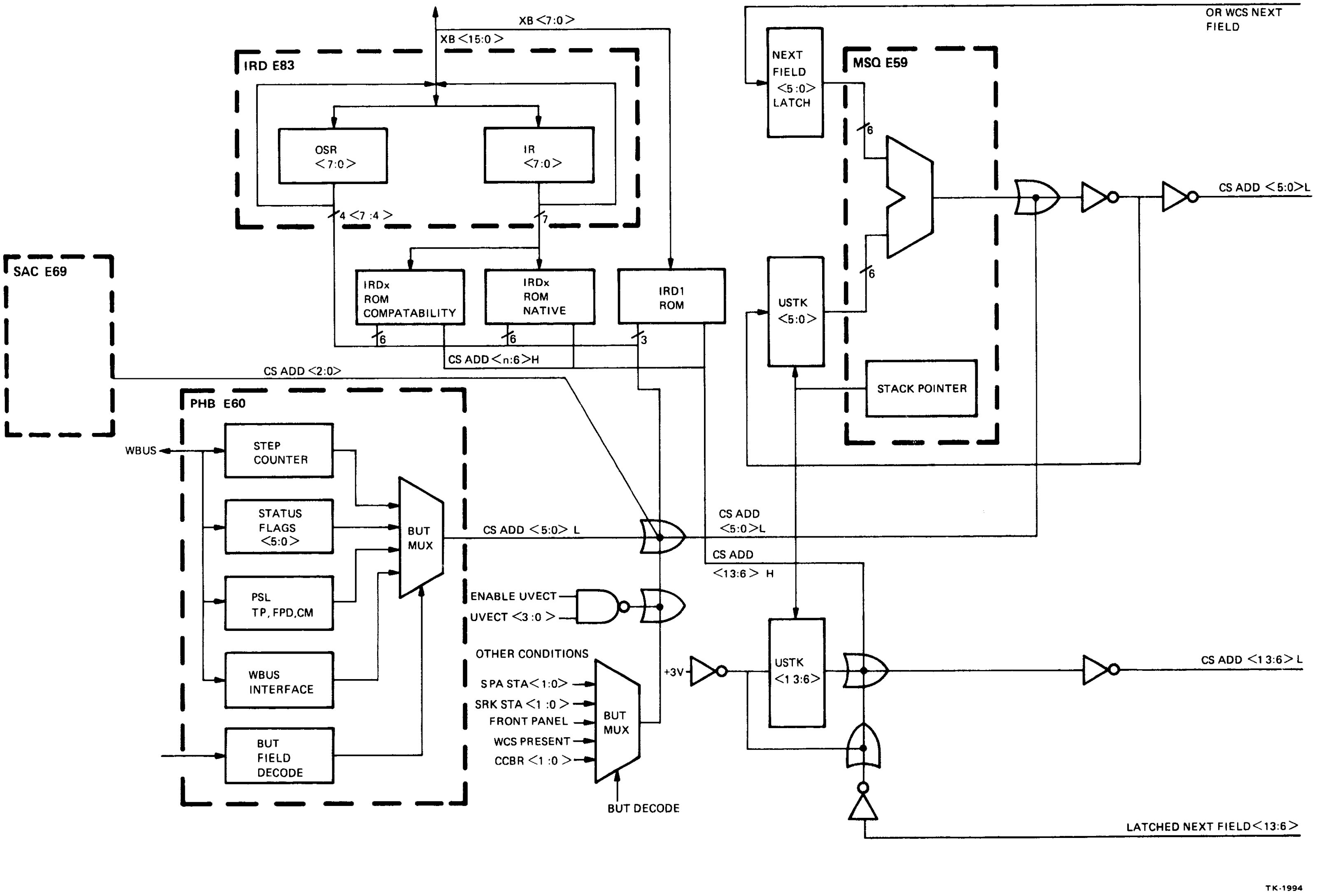


Figure 2-26 LSI Microsequencer Chip Functional Schematic

The rest of the logic in the microsequencer is used to perform microsubroutine calls and returns, microbranches on hardware state, and to decode the macroinstruction set. The basic operation of the microsubroutine-calling mechanism is the hardware-called microstack. This is a 16 × 13 bit RAM that is used to save control store addresses at the point another microroutine is called. The microstack mechanism allows up to 15 calls (JSR/PUSH) before a return (BUT/RETURN) has to be specified. The return micro-order pops the saved control store address off the microstack and ADDS the NEXT field <5:0> to the microstack address <5:0>. Carry to bit <6> is lost if there is one. Conditional microbranching is possible with the BUT micro-orders. The BUT micro-order selects a hardware condition and inclusively ORs the condition with selected control store address bits. The PHB gate array and discrete components accomplish this function in the microsequencer. The microsequencer also addresses the control store as a function of the VAX-11 macroinstruction on the XB lines or in the IRD gate array at instruction decode time. The IRD ROMS provide the control store starting address for macroinstruction execution.

Figure 2-27 is block diagram of the CCS control store. It is arranged into six 1K banks of 80 bits. There is circuitry to test the control store address for access to the unassigned regions and disable the address lines. A bank select decoder enables one of the six banks by decoding the CS ADD <12:10> lines to produce the bank select enable signal that allows the PROM data to go to the DPM module to be latched. Once the control store data is latched, the data is checked for correct data parity. The WCS attaches to this module and is similiar in design.

2.3.1 Microaddressing Modes

As seen in Figure 2-28, the address of the next microinstruction can be constructed in several ways. The method of generating the microaddress of the next microinstruction is referred to as the microaddressing mode. Figure 2-29 illustrates the seven microaddressing modes. Each mode is discussed below. A discussion of the associated control signals is provided in Paragraph 2.3.4.

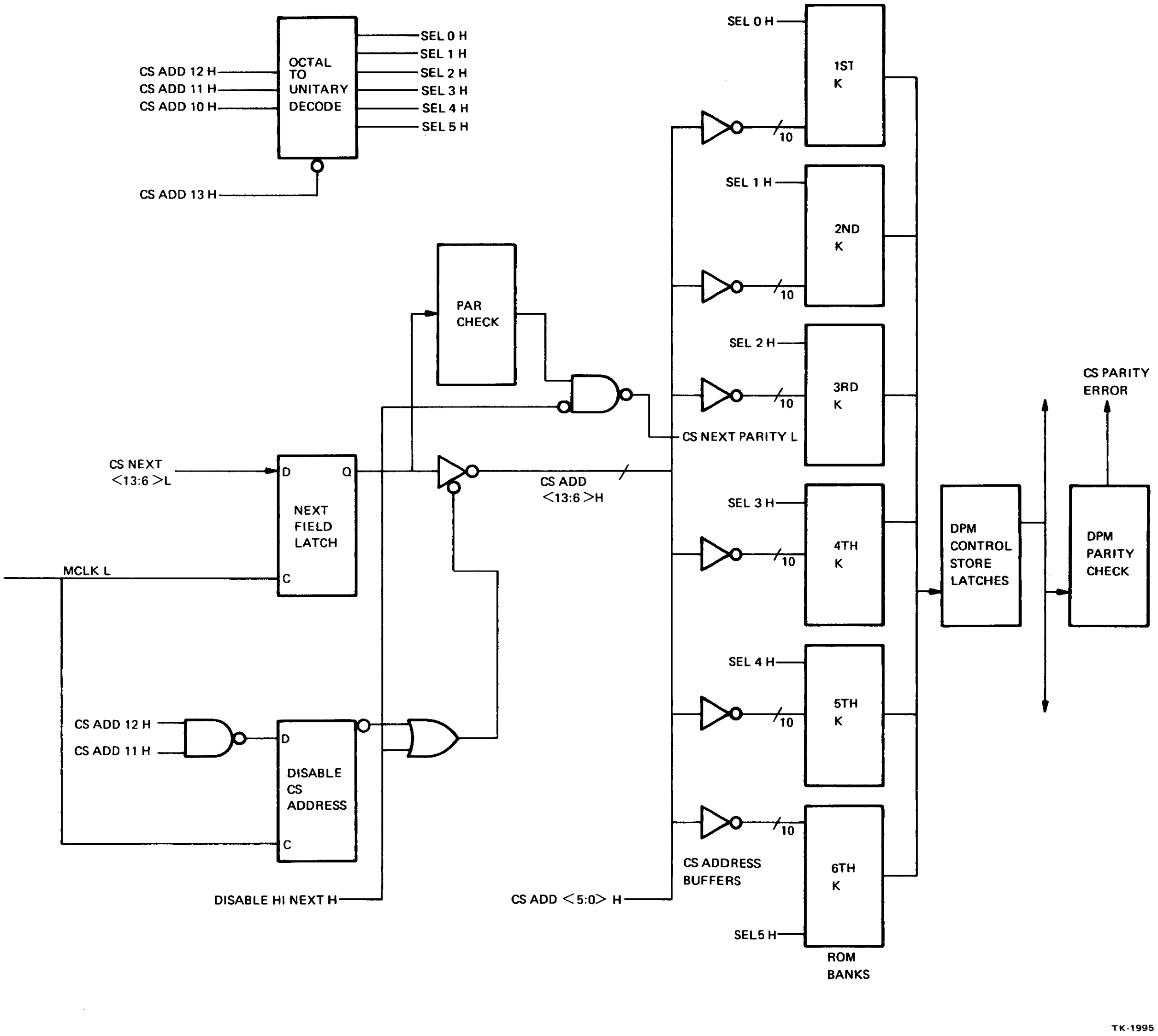


Figure 2-27 Control Store Simplified Diagram

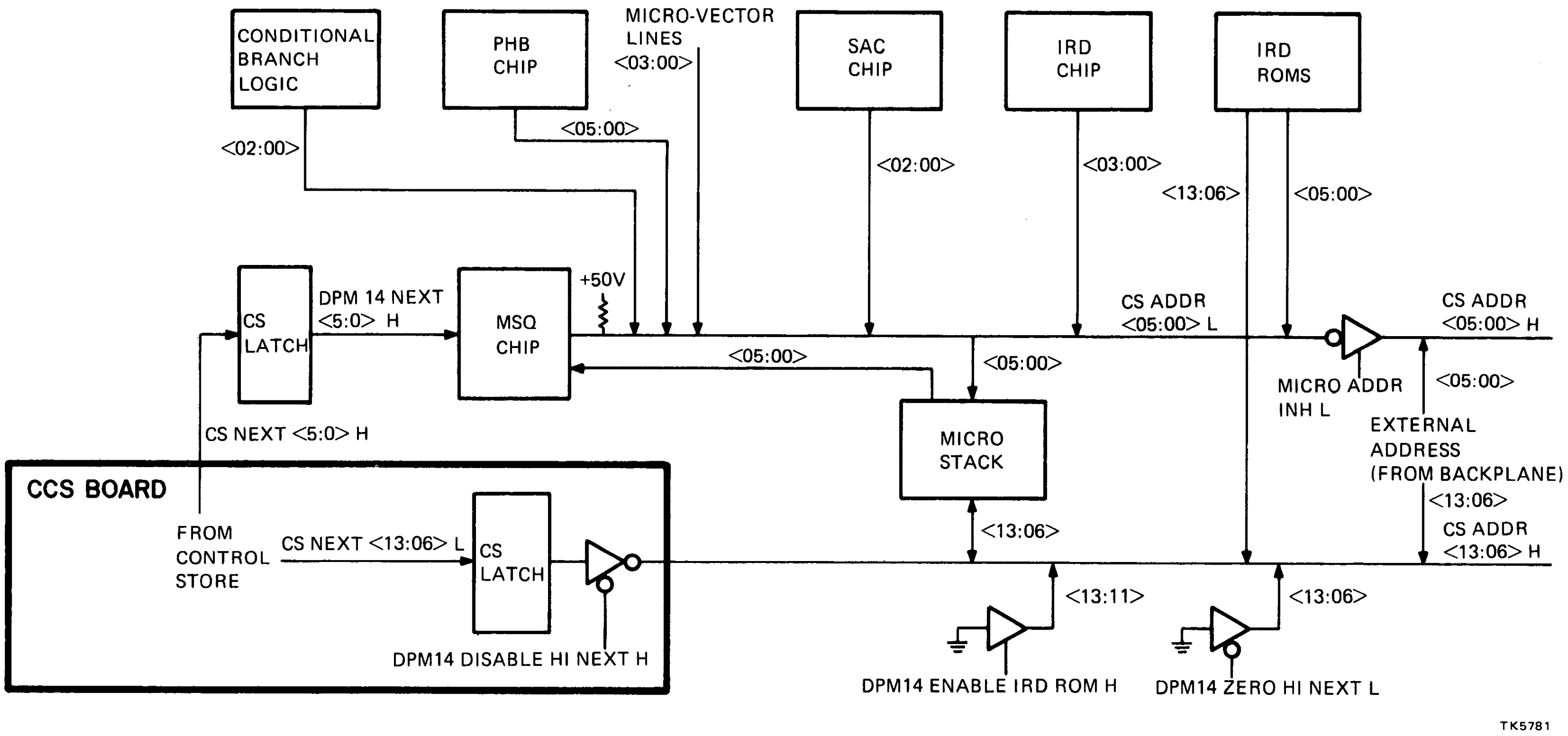


Figure 2-28 Microsequencer Block Diagram

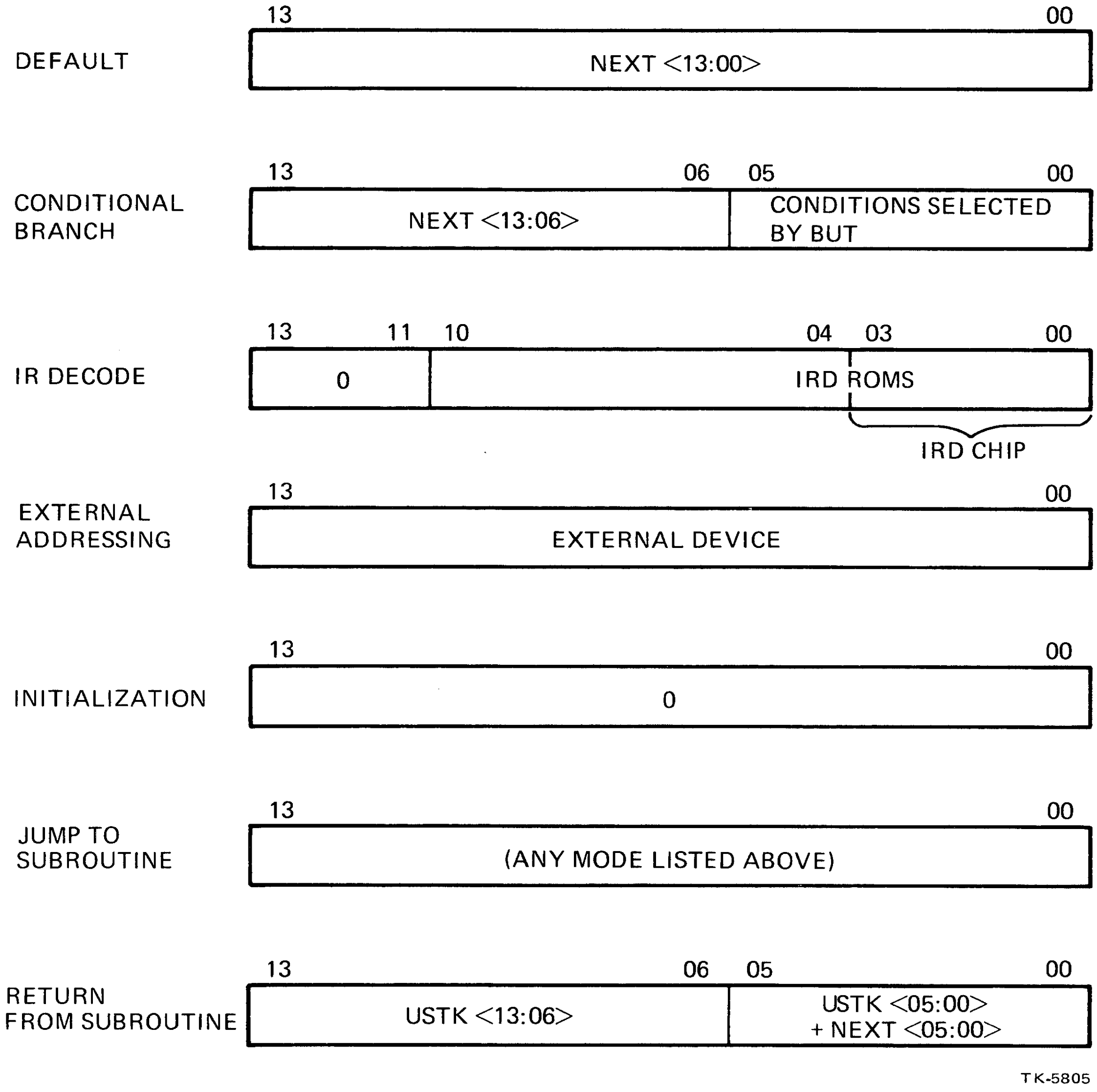


Figure 2-29 CS Address Generation for Each Microaddressing Mode

The default mode of microaddressing is where the address of the next microinstruction is specified by the NEXT microfield. The upper eight bits of the microaddress, CS ADDR <13:06>, are used directly from the control store latches. The lower six bits, CS ADDR <5:0>, are channeled through the MSQ chip. The BUT microfield must contain a NOP in this microaddressing mode.

For the conditional branch mode, the BUT microfield specifies conditions that generate the lower six bits of the microaddress. In this mode, the output of the MSQ chip is inhibited in order to allow an address to be ORed onto the CS ADDR lines by the PHB chip or conditional branch logic. The upper eight bits are specified by NEXT <13:06>.

In the IR decode mode the address of the next microinstruction is generated by an IRD ROM. The specific ROM and ROM location is a function of the macroinstruction. This mode is selected when the BUT microfield specifies an IRD 1 or IRDx. IR decode is further discussed in Paragraph 2.4.

An external addressing mode is provided to enable microaddress generation by the remote diagnosis option. This mode inhibits the microsequencer from generating the next microaddress. The signal MICRO ADDR INH L is asserted by the RD or another external device to disable the tri-state CS address drivers.

The initialization mode forces the next microaddress to zero. This mode is provided for the power-fail/power-up logic on the UBI module.

The jump to subroutine (JSR) mode is selected by the JSR microfield bit. When set, the address of the current microinstruction is pushed onto the microstack. The address of the next microinstruction can be generated by any of the addressing modes described above. A JSR is also forced by a microtrap or service condition (see Paragraphs 2.3.1.1 and 2.3.1.2).

The return from subroutine (return) mode is used at the end of a subroutine or error service routine to continue the original flow of the microprogram. This mode is selected when the BUT microfield specifies a RETURN, RET.DINH, or IRDX. When a return is specified, the address of the calling microinstruction is removed from the microstack. (The calling microinstruction is defined as the microinstruction that caused entrance into the subroutine.) Microaddress bits <5:0> are then generated by adding bits <5:0> from the stack to bits <5:0> of the NEXT microfield. NEXT <13:06> are ignored. The addition is performed within the MSQ chip. The resulting microaddress is always rewritten into the same microstack location.

Note that a JSR, microtrap, or service condition overrides the return mode. Note also that the LIT microfield cannot specify LONLIT for the conditional branch, IR decode, or Return microaddressing modes.

2.3.1.1 Microtraps – A microtrap is a microroutine initiated as a result of a microfault or error during a microinstruction. The microtrap enables the microinstruction to be completed successfully and is transparent to the microprogrammer.

The microsequencer performs the microtrap at the end of the microcycle in which the trap occurred. This is done by forcing a JSR to the appropriate microtrap routine. The microtrap routine corrects the problem and returns to the microinstruction by executing a return. The microinstruction is then reexecuted.

The appropriate microtrap routine is selected by a microvector address generated by the MIC logic. (Refer to Paragraph 2.3.1.3 for a description of microvector address generation.) This microvector overrides the addressing mode specified in the microinstruction. Following is a list of each microtrap and the vector address of its starting location in the control store.

Microtrap	Vector Address
Control Store Parity Error	0020
Unaligned Data, Read	0021
XB Miss	0022
XB ACV	0023
Unaligned Data, Write Unlock	0024
Unaligned Data, Write	0025
Write Unlock, Page Boundary	0026
Write, Page Boundary	0027
Machine Check Exceptions (see below)	0028
BUT XB Miss	0029
TB Miss, Read	002A
TB Miss, Write	002B
FPA Reserved Operand	002C
BUT XB ACV	002D
ACV, Read	002E
ACV, Write	002F

Note that a vector address of 0028 selects machine check exceptions. These include the following machine check errors. Refer to Paragraph 2.5 for details.

Machine Check Exceptions (0028)

XB TB Error
XB Bus Error
Bus Error
TB Error
BUT XB TB Error
BUT XB Bus Error
Cache Parity Error

Multiple microtrap conditions can occur during the same microcycle. Execution priority is handled by the ACV chip on the MIC module (Paragraph 2.5.2). Microtrap priority is assigned as follows.

Highest	Control Store Parity Error
	FPA Reserved Operand
	XB TB Error
	XB Bus Error
	Bus Error
	XB Miss
	XB ACV
	TB Error
	TB Miss, Read
	TB Miss, Write
	ACV, Read
	ACV, Write
	Write, Page Boundary
	Write Unlock, Page Boundary
	Unaligned Data, Read
	Unaligned Data, Write Unlock
	BUT XB TB Error
	BUT XB Bus Error
	BUT XB Miss
Lowest	BUT XB ACV

The microinstruction that caused the trap is reexecuted at the end of the microtrap routine. For this reason, destination registers and scratchpad registers must be inhibited for all but one execution cycle. The type of microtrap determines when the destinations are written. Table 2-1 lists each microtrap and indicates whether the destination is written during the microcycle in which the microtrap occurred, during the microcycle immediately following the microroutine, or not at all. Three groups of destinations are listed for the microtrap cycle and the retry cycle (cycle immediately following the microroutine). The first group of destinations includes the PC (program counter register in the ADD chip), the IR (instruction register in the IRD chip), and the OSR (operand specifier register in the IRD chip). As seen in Table 2-1 these registers are always inhibited during the microcycle in which the fault occurs. This is done in case an IR decode branch is specified in the faulted instruction (Paragraph 2.4). The bus cycle group includes any bus destinations. Bus cycles are inhibited when the microtrap condition makes it impossible for them to be successfully completed. The general destination group includes the scratchpad registers on the WBus (Paragraph 2.6.4.1).

Most register inhibits are performed by the hardware with the generation of clock inhibits. In certain instances, however, the inhibit must be specified by the microcode. Refer to Paragraph 2.6.4.4 for more details on register inhibits via microcode.

As shown in Table 2-1, the failing microinstruction may not need to be executed immediately following the microroutine. These types of microtraps are indicated by an X (no return). For the other types of microtraps in which a return must be immediately executed, three methods are available.

Return
Return and Inhibit Bus Cycles
Return and Inhibit Destinations

Table 2-1 Register Inhibits During Microtraps

	Microtrap Cycle			Retry Cycle		
Microtrap	**PC, IR, OSR**	**BUS CYC**	**GE DST**	**PC, IR, OSR**	**BUS CYC**	**GEN DST**
Control Store Parity	I	I	I	X	X	X
FPA Reserved Operand	I	I	I	X	X	X
XB TB Error	I	I	I	X	X	X
XB BUS Error	I	I	I	X	X	X
Bus Error	I	I	I	X	X	X
Unibus Unaligned	I	I	I	X	X	X
XB Miss	I	I	I	X	X	X
XB ACV	I	I	I	X	X	X
TB Error	I	I	I	X	X	X
TB Miss, Read	I	I	I	E	E	E
TB Miss, Write	I	I	I	E	E	E
ACV, Read	I	I	I	X	X	X
ACV, Write	I	I	I	X	X	X
Write, Page Boundary	I	I	I	E	I	E
WR Unlock, Page Boundary	I	I	I	E	I	E
Unaligned Data, Read	I	I	I	E	I	E
Unaligned Data, Write	I	I	I	E	I	E
Unaligned Data, Write Unlock	I	I	I	E	I	E
BUT XB TB Error	I	E	E	X	X	X
BUT XB BUS Error	I	E	E	X	X	X
BUT XB Miss	I	E	E	E	I	I
BUT XB ACV	I	E	E	X	X	X

Note: I = Inhibit, E = Execute, X = No Return

The return method is specified by the BUT microfield alone. A value of 02 results in the reexecution of the failing microinstruction with no inhibits. The Return and Inhibit Bus Cycles method is specified by a value of 02 in the BUT microfield and 1B in the MISC microfield. This method reexecutes the failing microinstruction, allowing the general destinations to be modified, while supressing bus cycles. The third method, Return and Inhibit Destinations, is specified when the BUT microfield contains a value of 03 (RET.DINH). In this case the original microinstruction is reexecuted, but all bus cycles and general destinations are inhibited. Note that the Return and Inhibit Destinations method does not inhibit the PC, IR, or OSR. The return methods are summarized below for each microtrap that requires immediate retry.

2.3.1.2 BUT Service – A hardware test called BUT Service is performed after each macroinstruction to determine if any traps or interrupts are pending. BUT Service is performed one microcycle after each macroinstruction to allow condition codes to become stable.

If a trap condition or interrupt is pending when BUT Service is performed, the appropriate service routine is initiated. This is referred to as DO Service and is initiated by the execution of a JSR. During this microcycle all destinations are inhibited. This includes the PC, IR, and OSR, bus cycles, and scratchpad registers.

The service routine is selected by a microvector address generated by the associated logic (refer to Paragraph 2.3.1.3 for a description of microvector address generation). This microvector overrides the addressing mode specified in the microinstruction. Following is a list of each service routine and the vector address of its starting location in the control store.

Service Condition	**Vector Address**
Arithmetic Trap	0011
FPA Integer Overflow Trap	0012
Interval Timer Overflow Trap	0014
T-Bit Trap	0015
Console Halt Trap	0016
Software Interrupt	0038
Console Interrupt	0039
Unibus Interrupt	003A
Interval Timer Interrupt	003B
Corrected Memory Data Interrupt	003C
Write Bus Error Interrupt	003E
Power-Fail Interrupt	003F

Multiple service conditions may exist when BUT Service is performed. Only one condition, however, may be serviced during each BUT Service. A priority decoder in the SAC chip determines which trap or interrupt to service (Paragraph 2.3.1.2.) Service priority is assigned as follows.

Highest	Arithmetic Trap
	FPA Integer Overflow Trap
	Interval Timer Overflow Trap
	Console Halt Trap
	Power-Fail Interrupt (IPL 1E)
	Write Bus Error Interrupt (IPL 1D)
	Corrected Memory Data Interrupt (IPL 1)
	Interval Timer Interrupt (IPL 18)
	Unibus Interrupt (IPL 14–17)
	Console Interrupt (IPL 14)
	Software Interrupt (IPL 01-0F)
Lowest	T-Bit Trap

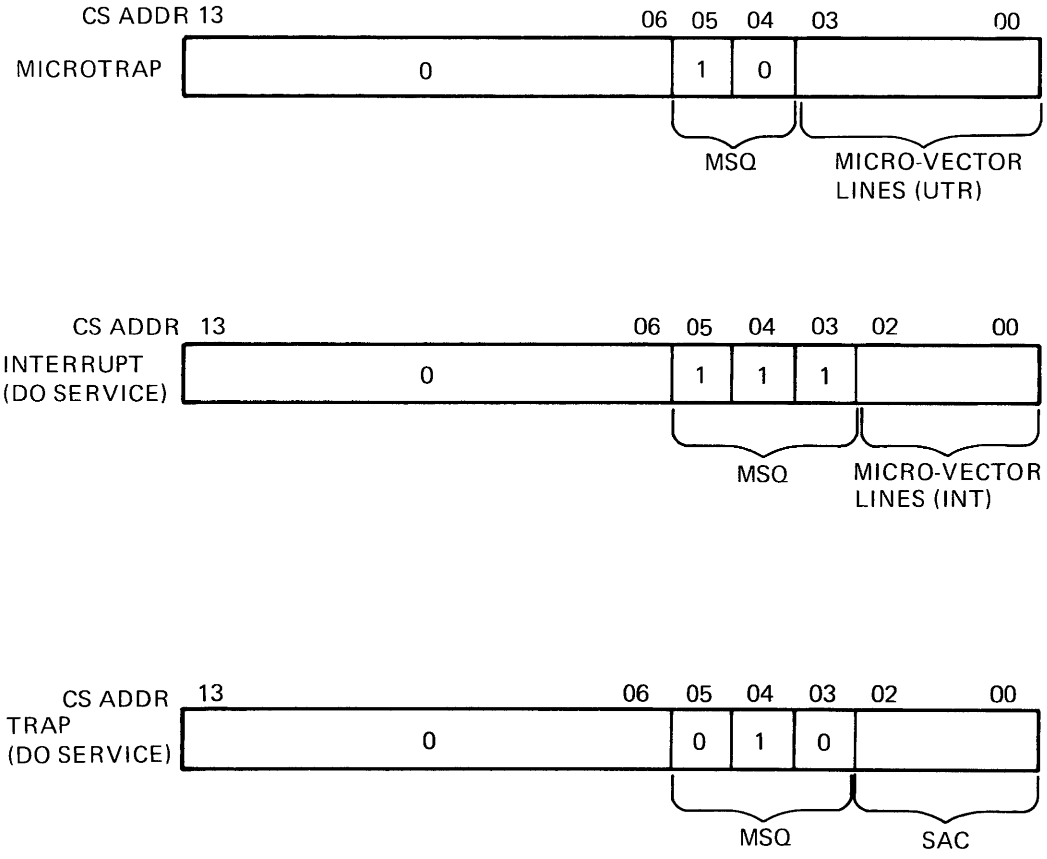
If a microtrap condition occurs during a microcycle in which a service condition is detected (during a BUT Service test), the service routine is performed and the microtrap is lost. Service routines have higher priority than microtraps. The only exception is the control store parity error microtrap, which has the highest priority.

During the execution of long macroinstructions, tests for interrupts can be performed by use of the BUT microfield. If an interrupt is detected, a microbranch to the appropriate service routine is executed.

2.3.1.3 Microvector Address Generation – A microvector is used to generate a CS address in four cases:

1. To generate the starting address of a microtrap routine when a microtrap occurs.
2. To generate the starting address of a service routine for an interrupt during BUT Service.
3. To generate the starting address of a service routine for a trap during BUT Service.
4. To generate a branch offset during a "BUT on microvector" operation (BUT = IE or 1F) in the conditional branch microaddressing mode.

The fourth case was briefly mentioned in Paragraph 2.3 and is further discussed below. Cases 1 through 3 are illustrated in Figure 2-30 and described below.



NOTES:
1. CS ADDR <13:06> ARE DRIVEN LOW BY A SIGNAL GENERATED BY MSQ, DPM14 ZERO HI NEXT L.

Figure 2-30 Microvector Address Generation

As seen in Figure 2-30, the microvector lines are used to OR a microvector onto the CS address lines when a microtrap or interrupt is to be serviced. In the case of a trap, however, the microvector is placed onto the CS address lines directly from the SAC chip. Note that in all three cases CS ADDR <13:06> are driven low by a signal from the MSQ chip. The MSQ chip also provides a base address which is ORed onto the lower six CS address lines. Paragraph 2.3.2 discusses the MSQ logic in detail.

The microvector lines are not used when a trap is being serviced. In this case the SAC chip drives the CS address lines directly. Figure 2-31 illustrates the BUT Service logic of the SAC chip. The SAC chip also includes the CPU clock generation logic (Paragraph 2.1.2). As seen in Figure 2-31, the input to the BUT Service flip-flop is asserted when the BUT decode logic detects an IRD 1 branch. This indicates the end of a macroinstruction and the appropriate time for BUT Service. At the following M clock, the BUT Service flip-flop is clocked. If a trap or interrupt is pending, DPM17 DO SRVC L is generated to indicate a service request is present. Note that this signal is inhibited if a CS parity error has occurred. (CS parity errors have priority over BUT Service.)

The priority decoder within the SAC chip monitors signals indicating trap and interrupt conditions. These signals include five specific trap indicators and one interrupt-pending indicator. If a trap is pending, the appropriate microvector is encoded by the SAC chip and placed directly on the CS address lines as CS ADDR <2:0> L. If an interrupt is pending, the appropriate microaddress is placed on the CS address lines via dedicated microvector address lines. For this case the CS address output of the SAC chip is inhibited and DPM17 ENABLE UVECT H is asserted. This signal is used to enable drivers on the microvector lines. Note that DPM17 ENABLE UVECT H can also be asserted if a microtrap occurs. For this, however, DO SRVC must not be asserted (i.e., DO SRVC has priority over microtraps).

The microvector lines are illustrated in Figure 2-32. When a microtrap or interrupt is to be serviced, DPM17 ENABLE UVECT is asserted by the SAC chip to enable the four drivers illustrated in this figure. These drivers are used to transfer a 4-bit vector from backplane pins onto the CS address lines. The vector is generated by the UTR chip on the MIC board if a microtrap is being serviced, or the INT chip on the UBI board if an interrupt is being serviced.

The microvector lines are also used during a "BUT on microvector" operation in the conditional branch microaddressing modes (case 4 listed above). For this case the vector is used as a branch offset. To accomplish this, DPM16 BUT UVECT L is generated to enable the vector line drivers when the BUT microfield equals IE or IF (IE = UVCTR, IF is undefined).

2.3.2 Microsequencer Control Signals

The MSQ chip provides most of the control signals for the microsequencer. These signals include the generation of the six low-order microaddress bits that are used as base address for most microaddressing modes. Figure 2-33 provides a simplified diagram of the logic contained in the MSQ chip. The three major areas of logic are the microaddress multiplexer logic, decode logic, and the microstack pointer logic. The microaddress multiplexer and decode logic are discussed here. The microstack pointer is discussed in Paragraph 2.3.3.

The microaddress multiplexer provides six low-order microaddress bits. These bits are used as a base address for one of the microaddressing modes or for the generation of a microvector. Table 2-2 lists the output of the microaddress multiplexer for each case. The reader should recall from Figure 2-30 that this CS address output is wire-ORed with other CS address sources. Therefore, it does not necessarily reflect the final CS address used.

The conditions listed in Table 2-2 are indicated by various signals monitored by the MSQ chip. Table 2-3 lists the signals that determine each condition.

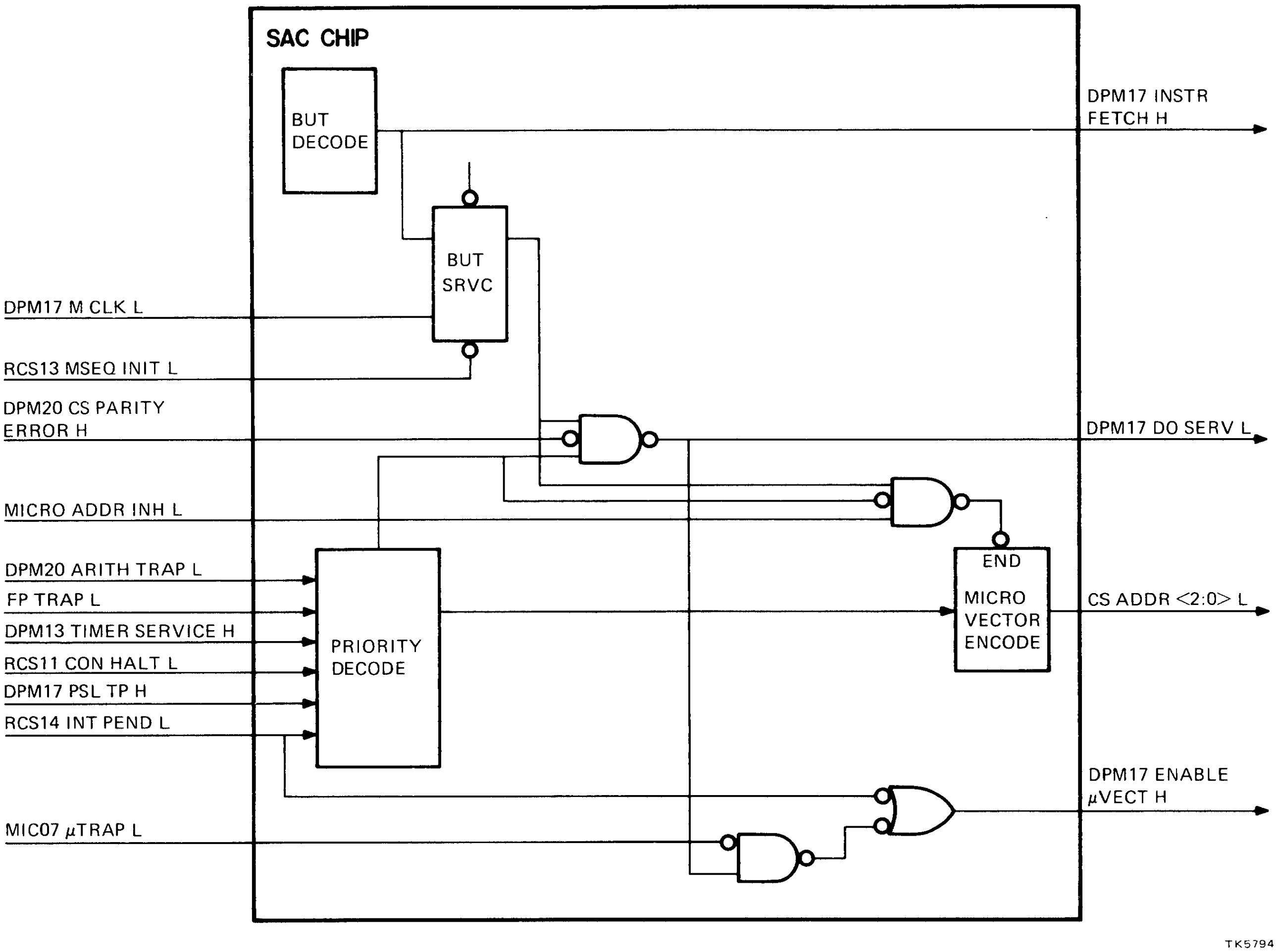
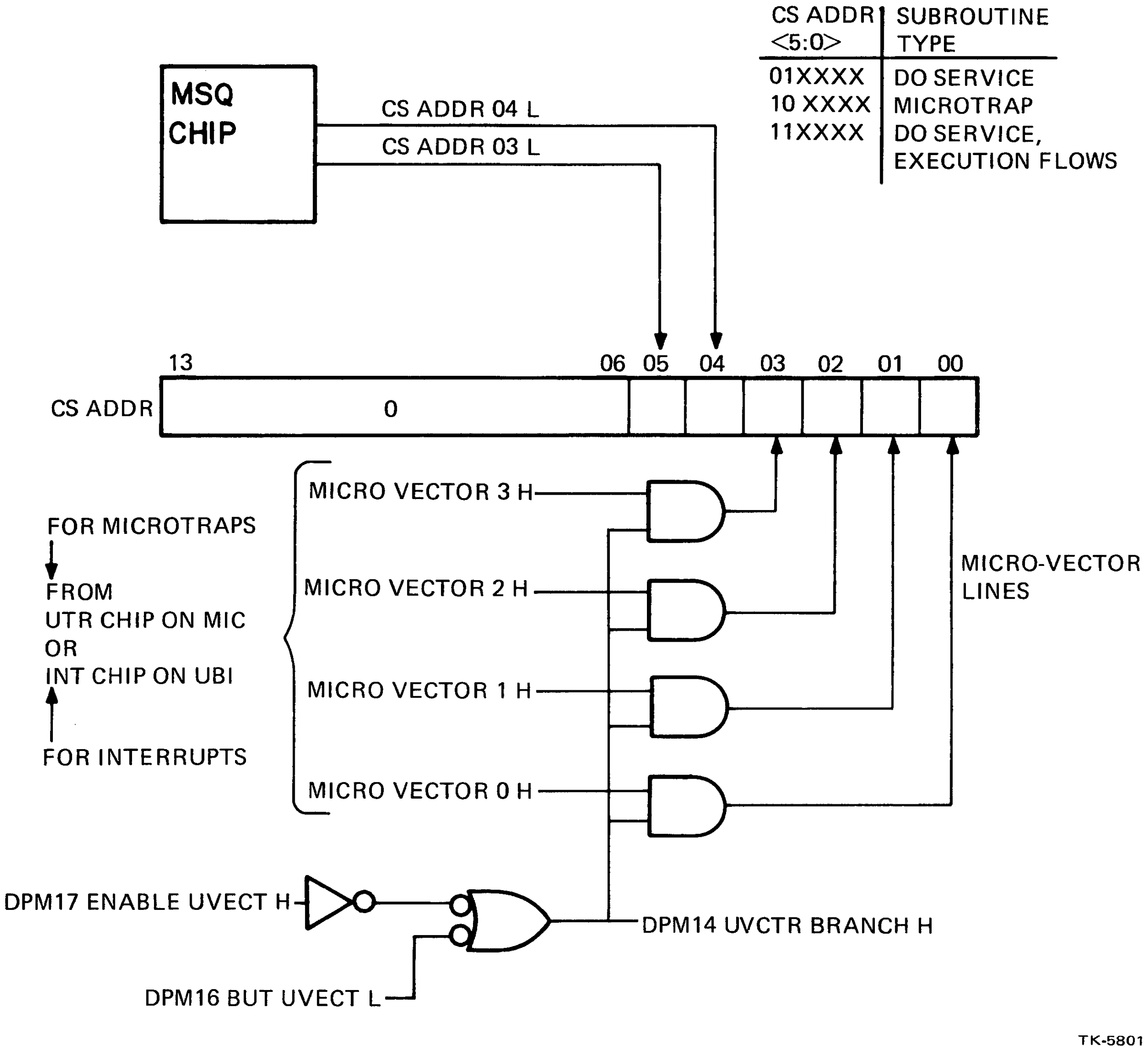


Figure 2-31 BUT Service Logic



CS ADDR <5:0>	SUBROUTINE TYPE
01XXXX	DO SERVICE
10 XXXX	MICROTRAP
11XXXX	DO SERVICE, EXECUTION FLOWS

Figure 2-32 Microvector Lines

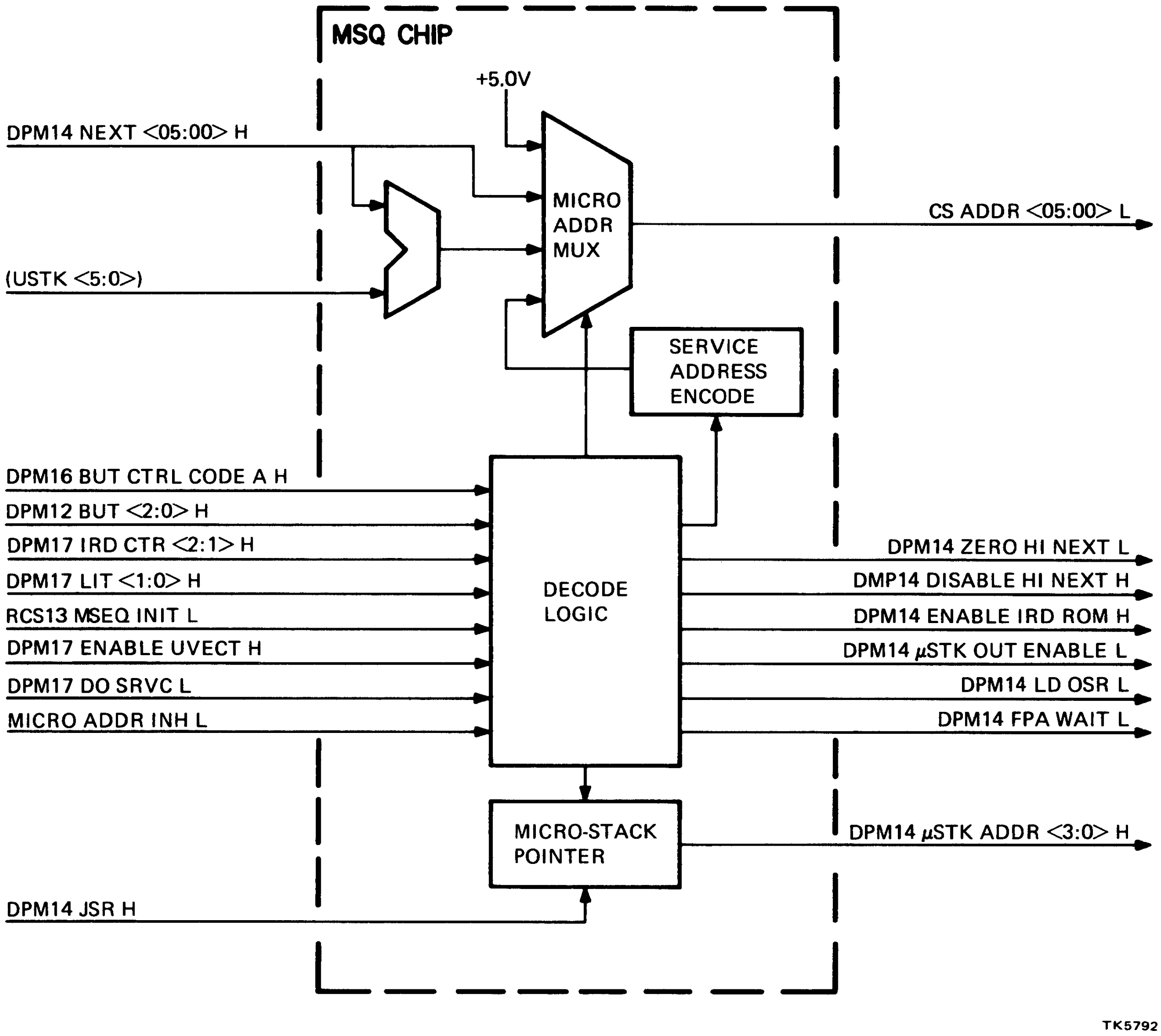


Figure 2-33 MSQ Logic

Table 2-2 Microaddress Multiplexer Outputs

Condition	Microaddress Multiplexer Output
Default	NEXT <5:0>
Conditional Branch	NEXT <5:0>
IR Decode	000000
External Addressing	000000
Initialization	000000
Return from Subroutine	NEXT <5:0> + USTK <5:0>
Microtrap	100000
Interrupt	111000
Trap	010000

Table 2-3 Condition Indicators for the MSQ Chip

Condition	Indicating Signal(s)
Default	–
Conditional Branch	DPM12 BUT <2:0>H DPM16 BUT CTRL CODE A H
IR Decode	DPM12 BUT <2:0> H DPM16 BUT CTRL CODE A H = H
External Addressing	MICRO ADDR INH L = L
Initialization	UBI13 MSEQ INIT L = L
Return from Subroutine	DPM12 BUT <2:0> H DPM16 BUT CTRL CODE A H = H
Microtrap	DPM16 ENABLE UVECT H = H DPM17 DO SRVC L = H
Interrupt	DPM17 ENABLE UVECT H = H DPM17 DO SRVC L = L
Trap	DPM17 ENABLE UVECT H = L DPM17 DO SRVC L = L

The microaddress multiplexer is controlled by decode logic within the MSQ chip. This logic decodes the signals listed in Table 2-3 to select the output of the microaddress multiplexer in addition to generating control signals for other CS address sources. Each of these control signals is described below.

The lower three bits of the BUT microfield are input to the decode logic of the MSQ chip. To minimize pin usage on the MSQ chip, bits <5:3> of the BUT microfield are decoded externally. If all three high-order bits are low, DPM16 BUT CTRL CODE A H is asserted.

In addition to the BUT microfield, bits <2:1> of the IRD counter are input to the MSQ chip. This is done for the following reason. When an IRDx is specified by the BUT microfield, a ROM branch or return function may be executed. The value of the IRD counter determines which occurs. If the counter contains a value less than 2, a ROM branch is performed. If the counter equals 2 or more, a return is performed.

Three output control signals of the MSQ chip are associated with the generation of the high-order CS address bits <13:06>. The three signals are:

DPM 14 ZERO HI NEXT L

DPM14 DISABLE HI NEXT H

DPM14 ENABLE IRD ROM H

DPM14 ZERO HI NEXT L is asserted to zero these bits during initialization or when a microvector is used. Initialization is detected by the assertion of the signal UBI14 MSEQ INIT L. Use of a microvector is detected by the assertion of DPM17 DO SRVC L or DPM17 ENABLE UVECT H. Note that DPM14 ZERO HI NEXT L is inhibited if MICRO ADDR INH L is asserted. This signal indicates the external microaddressing mode.

When MICRO ADDR INH L is asserted, the CS lines must be cleared for the assertion of a CS address by an external device. To clear the high-order CS address lines, DPM14 DISABLE HI NEXT H is asserted. This prevents the NEXT microfield from driving the CS lines (Figure 2-30). The microaddress multiplexer is likewise disabled. MICRO ADDR INH L also eliminates any effects of other low-order CS address sources by disabling drivers at the end of the CS address lines (Figure 2-30)

DPM14 DISABLE HI NEXT H is also asserted during the return from subroutine and IR decode microaddressing modes. In each of these cases the microaddress multiplexer is disabled. During the return from subroutine mode, DPM14 USTK OUT ENABLE L is generated to remove the microaddress from the microstack. During the IR decode mode, DPM14 ENABLE IRD ROM H is asserted to enable the IRD ROMs for the generation of the CS address. DPM14 ENABLE IRD ROM H also clears CS address bits <13:11> (Figure 2-30). Refer to Paragraph 2.4 for a complete description of IR decode. Paragraph 2.3.3 describes microstack operation.

2.3.3 Microstack Operation

The microstack is a 16-location stack within the microsequencer that provides the microprogrammer with the capability of subrouting and nesting. The address of the current microinstruction is always placed on top of the microstack. As long as a microstack function is not required (not a JSR or return) the stack pointer remains unchanged. For these microcycles, the stack location is always overwritten with the address of the new microinstruction.

The microstack pointer is contained within the MSQ chip. The stack pointer is incremented when a JSR is executed. For this case the address of the current microinstruction is stored in the new stack location. A JSR may be explicitly specified by the JSR microfield, or implicitly specified by an interrupt or exception. When a JSR is explicitly specified, DPM14 JSR H is generated and input to the MSQ chip to increment the stack pointer. Interrupts and exceptions are detected by the following signals.

DPM17 ENABLE UVECT H	**DPM17 DO SRVC L**	**Condition Indicated**
Asserted	Asserted	Interrupt
Asserted	Unasserted	Microtrap
Unasserted	Asserted	Trap

The MSQ chip decodes these signals to increment the microstack pointer and to generate the microaddress of the appropriate service routine.

At the end of a subroutine, a return microinstruction is executed. DPM14 USTK OUT ENABLE L is generated by the MSQ chip to enable the microstack output. This removes the microaddress indicated by the stack pointer and places it on the CS address lines. The microstack pointer is decremented at the end of the return microinstruction.

2.3.4 Control Store Module

The CPU control store module (CCS) occupies slot 5 of the backplane. The control store is a 6K × 80 bit PROM design. The circuitry is designed around 1K × 4 tri-state PROM. The design is implemented in six banks of 1K × 80 bits with bank-select logic that decodes the MSBs of the control store address. Figure 2-27 is the block diagram of the CCS module design, and it shows the major circuitry of the design.

The cycle time of the control store is the normal 320 ns microinstruction execution time, even though the PROM access time is approximately 60 ns. In some instances such as IRD 1, IRDx, and UTRAP, the cycle has to be extended because there is no I-Stream or because the hardware has to generate a microaddress by decoding certain conditions. The M CLK L signal is used to load a new microinstruction into the control store latches. The cycle time of each microinstruction begins on the low-to-high transition of the M CLK L.

The derivation of the M CLK L is explained in Paragraph 2.1. The control store timing for reading the next microinstruction from the NEXT field of the microword is shown in Figure 2-34. The signals referenced in the figure are from both the DPM module and CCS module print sets.

The NEXT address bits <5:0> are latched on DPM 14 on the low-to-high transition of M CLK L, and on CCS01 the NEXT address bits <13:6> are latched by the same M CLK L. Including the propagation delay, the next address bits <13:6> go to the CS ADD BUS, and reading of the control store commences. Bits <5:0> go straight through the MSQ gate array but are delayed slightly longer. The PROM data must be stable before the next M CLK L which latches the next microinstruction.

If a microinstruction has to be aborted because of a microtrap, the hardware must generate the control store address of the microinstruction to service the microtrap. Because of this, the cycle is extended 2 B clocks to obtain the necessary set-up time for the hardware to generate the control store address. Figure 2-35 illustrates the extended cycle for a control store parity error. The microvector for control store parity errors is 0020. The derivation of the microvector addressing is explained in Paragraph 2.3.1.1.

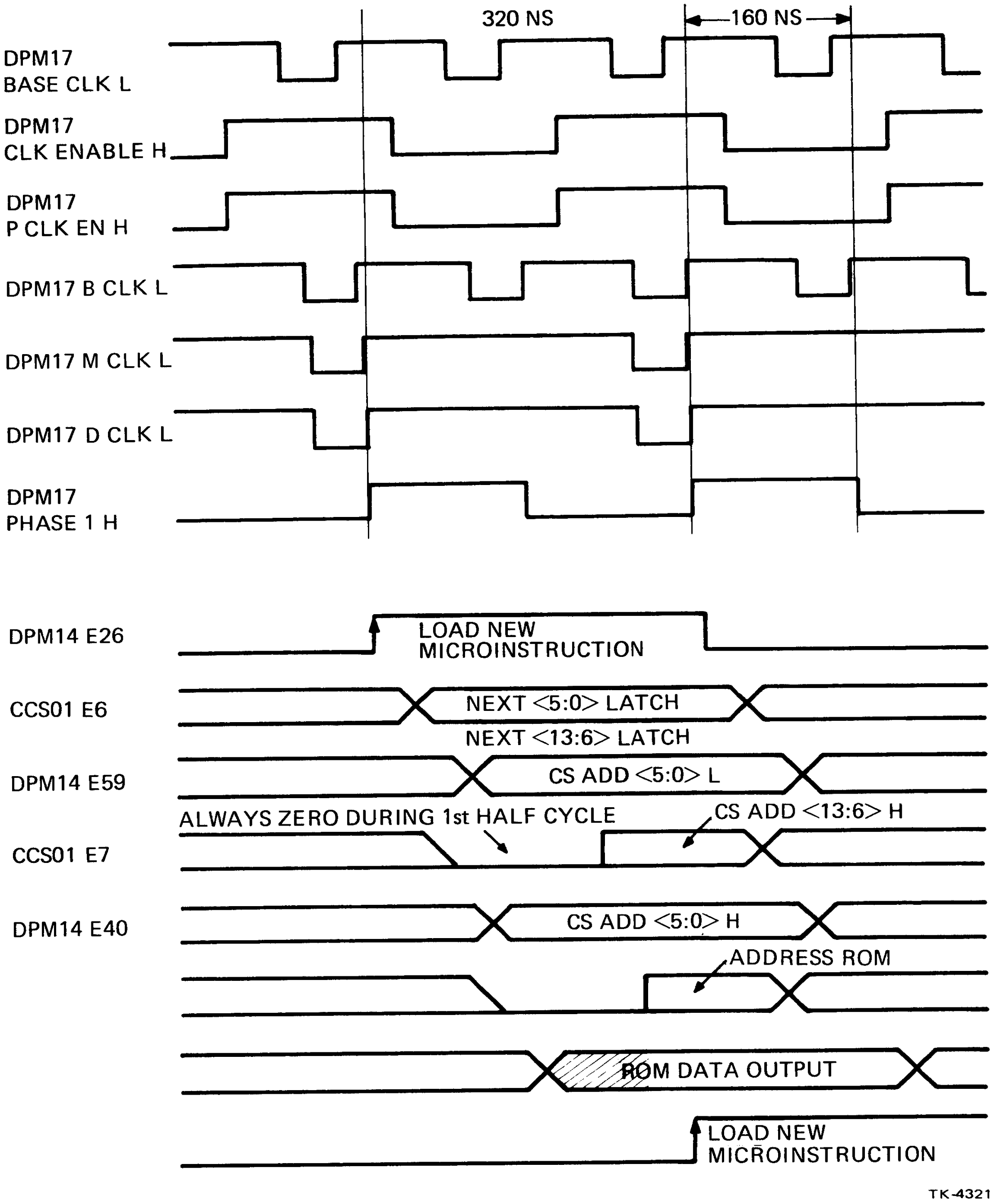
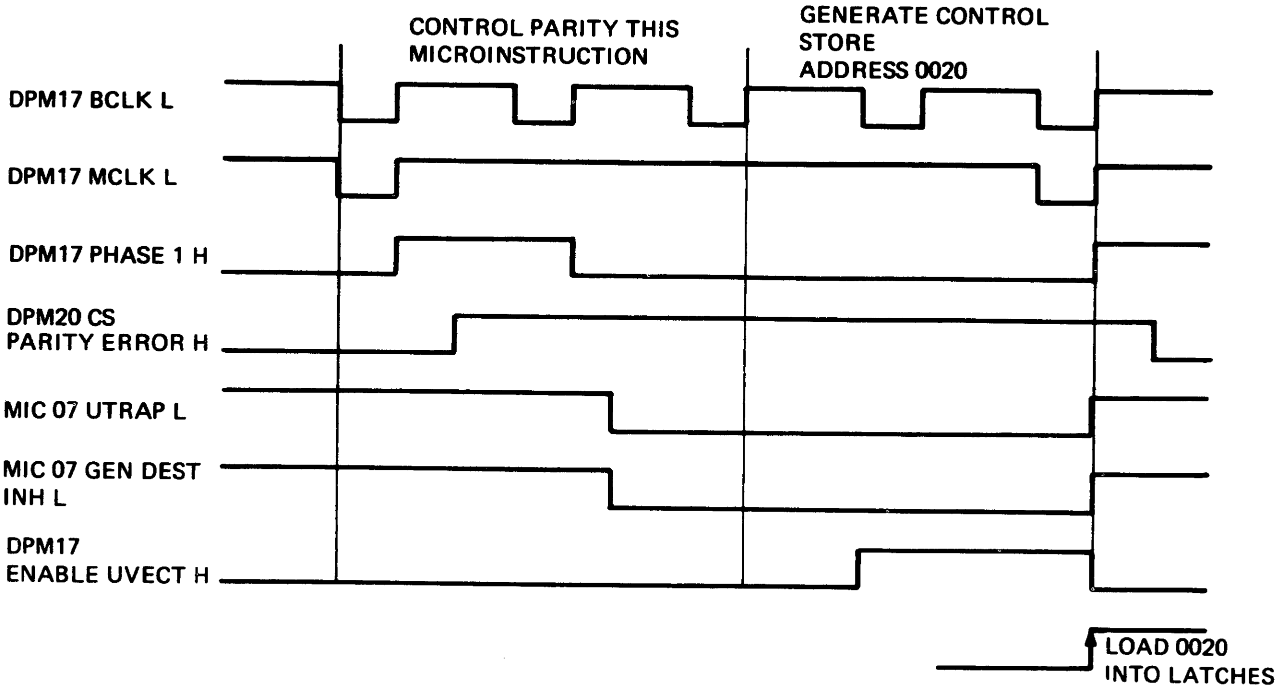


Figure 2-34 Control Store Timing (Reading Next Microinstruction from Microword NEXT Field)



TK-4322

Figure 2-35 Extend Clock Cycle for Control Store Parity Error

The latched microinstruction connects to parity checking circuitry distributed among the DPM, CCS and UBI modules. The parity checking logic generates a parity error if the microword is in error. This signal is called CS PARITY ERROR H and is located on DPM 20. CS PARITY ERROR H goes to the SAC gate array where it is latched in a flip-flop. If a second CS parity error occurs before the next IRD 1, the SAC gate array stops the B CLK signal and lights the CS PARITY ERROR indicator on the operator control panel. The CS parity error also forces a microtrap to divert the flow of the microcode to the CS parity error microroutine. This initiates a machine check exception that is serviced through the macrocode routine at SCBB+4. CS PARITY ERROR H goes to MIC7 to the ACV gate array where the CS parity error is encoded into a 3-bit number that is called ENC UTRAP <2:0> L. The encoded number is 7 and it enters the UTR gate array on MIC7. The UTR gate array generates the signal GEN DEST INH L that inhibits registers from being loaded with meaningless data.

The next B CLK L generates the signal from the UTR gate array called UTRAP L. UTRAP L goes back to the SAC gate array on DPM17 to extend the microcycle 2 B clocks to allow enough set-up time to enter the microtrap routine. The SAC gate array produces two outputs that go to the MSQ gate array so it can generate bits <5:4> of the microvector. These two outputs are called DO SRVC L and ENABLE UVEC H. DO SRVC L is only true if at BUT Service an interrupt or service request is pending. ENABLE UVEC H is true during microtraps and external interrupts at BUT Service. These signals are combined as shown in Figure 2-32 to produce the first two bits of the control store address.

The UTR gate array forms the microvector address in bits <3:0>. Gates E42 on DPM 14 are enabled to drive MICROVECTOR <3:0> H by the signal ENABLE UVEC H from the SAC gate array. The microaddress driven on the CS ADD lines then comes from the MSQ gate array for bits <5:4>. Bits <3:0> come from MICROVECTOR <3:0> H. Bits <13:6> of the CS ADD lines are zeroed by the MSQ gate array with a signal that goes to DPM 13 called ZERO HI NEXT L. Microaddress 0020 is formed by the hardware on the CS ADD lines. ROM access time is still from 60 ns and the contents of location 0020 should be stable by the time M CLK L is issued.

Some microinstructions may have to be extended to complete an operation that cannot be done in the normal 320 ns time. To extend the cycle for 1 B clock is the function of the CLKX bit <15> of the microword. Certain micro-orders must have the CLKX bit set in order to complete succesfully. The CLKX bit is set by a MICRO2 assembler post-processing program for certain micro-orders and the exact cycle time in nanoseconds is shown in the microcode listing in the binary data output. The time has an asterisk (*) following it. For example:

```
U  0800,1860,C100,A300,8430,8800    384*    ;5106

WX.EQ.0?,  NEXT/
```

The binary output shows the ROM address, the content, and the amount of time required to complete the ROM state. It takes 384 ns to execute this particular microinstruction, which is longer than the 320 ns normal cycle time. Figure 2-4 (in Paragraph 2.1.2.6) shows an extended microcycle timing diagram. As shown in this diagram, the normal 320 ns cycle becomes 480 ns.

2.3.5 Control Store Hardware Implementation

Refer to the control store schematic diagram CCS 01. The interface next address latch and CS ADDR <13:6> drivers are contained on this page.

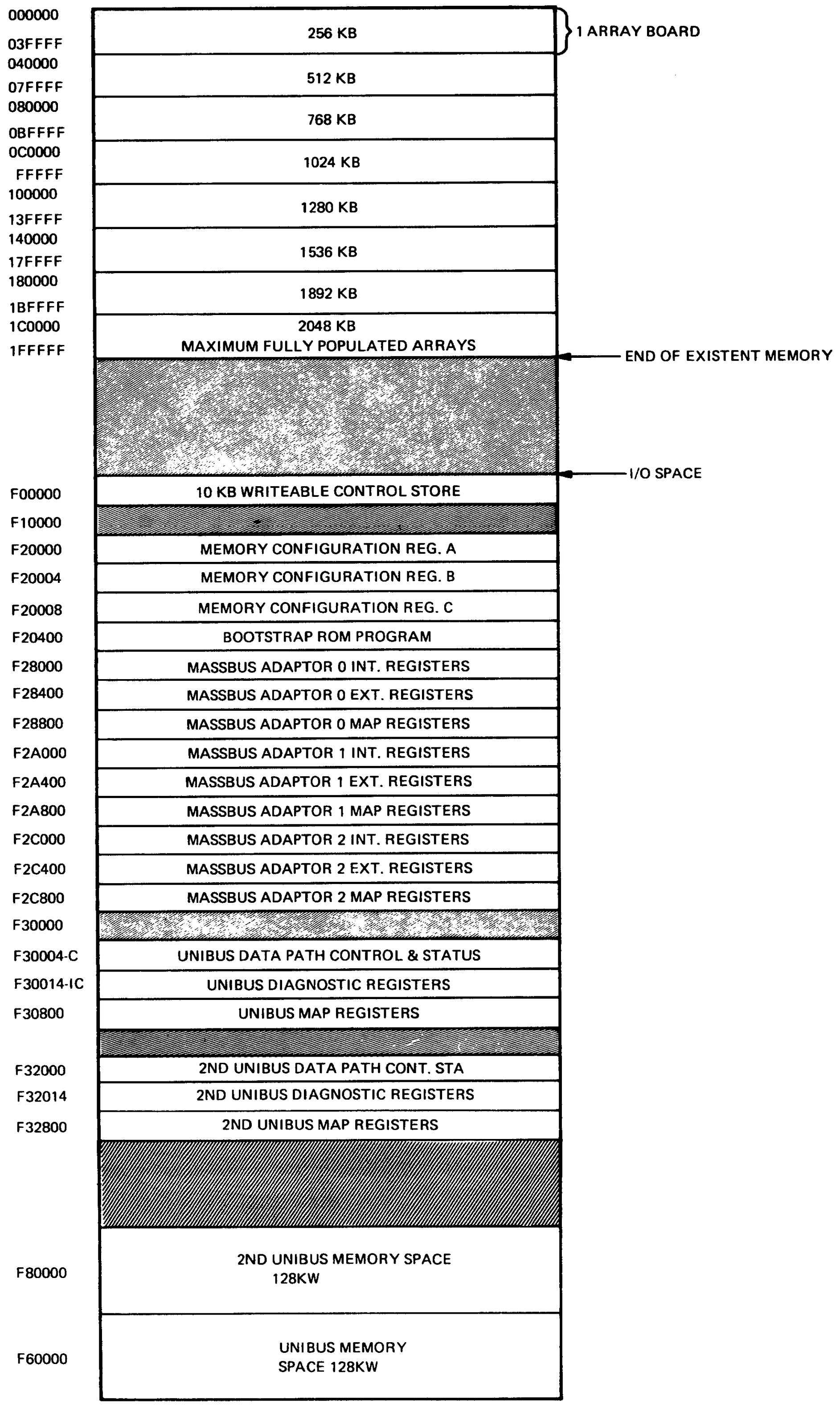
On the low-to-high transition of the M CLK L, a new microinstruction is loaded into the control store latches distributed among the UBI, CCS, and DPM modules. E6 latches bits <13:6> of the microword, which comprise high bits of the NEXT field. The output from the latch goes right to the CS ADD drivers to read the next microinstruction. Flip-flop E2 is there to prevent accesses to the unassigned seventh and eighth K of the control store. If a control store address to the unassigned area is latched, NAND gate E3 asserts a low output to clear E2 at the next M CLK L. The result is that E3 pin 2 disables the CCS module to drive the signal CS HNEXT PAR H, which should now be driven by the logic that contains the seventh or eighth K of the control store space (e.g., ROM). For a similar reason it also shuts off the drive for the tri-state drivers to CS ADDR <13:06>. In the upper left corner of CCS 01 is the bank select decoder that enables one of the six 1K banks by decoding bits <12:10> of the control store address. Note that bit <13> disables the decoder because bit <13> specifies the WCS address space or higher.

The tri-state control store address lines, CS ADDR <09:00>, are buffered on CCS 02 and CCS 03 before driving the address inputs to the ROMs. CCS 04 through CCS 08 show the lower 3K of the CCS control store, drawn in the order the microword is defined to MICRO2. Each ROM is a 1K × 4 bit tri-state part. Each bit of the microword has six possible sources on this board and two more sources externally (WCS and RDM). The upper 3K of the control store is shown on CCS 09 to CCS 13. The ROM output is latched on the FPA, DPM, MIC, UBI, and CCS CPU modules and a parity check is performed on the DPM module. The pinning for the daughter-board connectors that interface the WCS module to the CCS module is illustrated on CCS 14.

2.3.6 Writable Control Store

The writable control store (WCS) module is an optional module that attaches to the resident control store module (CCS) to provide the customer with the capability of executing application-specific microroutines. G and H floating math processors implement the G and H instruction set on the WCS module. The writable control store is 1K by 80 bits and has a data interface to the CMI bus. The WCS is loaded from the CMI and also can be read back over the CMI for write/read data comparison. The access time of the WCS RAMs is 55 ns. Timing for WCS operation is derived from B CLK L. Parity is not automatically generated when the microcode is written into WCS. The customer should either use the MICRO2 assembler, which computes parity to generate the microcode, or calculate the parity according to the hardware definition in the DEFIN.MIC file of the microcode listing. The data stored in WCS must have the correct parity, or control store parity errors will result when executing microcode from WCS.

2.3.6.1 WCS Detailed Description – This paragraph describes how the WCS is accessed via the CMI. It assumes the reader is familiar with the CMI concepts and protocol as described in Paragraph 2.5.9.1 of this manual. Refer to Figure 2-36, which illustrates the physical address space organization of the CMI. The VAX-11/750 physical address space is 16 megabytes in size, with the upper half being set aside for I/O registers and controllers. The first I/O address is F00000 (hex), the first longword of the WCS RAM. The WCS is designed as a 20-bit wide interface to the CMI. This means that four longword writes to sequential locations are required to pack one 80-bit microinstruction into WCS. CMI physical longword addresses F00000 through F0000C correspond to control store address 2000. Refer to Figure 2-25 for control store address allocation.



Loading a single 80-bit microinstruction into the WCS location 2000 could be accomplished as follows.

```
TABLE:      .LONG ↑X08800              ;bits <19:0>
            .LONG ↑X00843              ; <39:20>
            .LONG ↑X100A3              ; <59:40>
            .LONG ↑X11860C             ; <79:60>

LDWCS:      MOVAL TABLE, R0
            MOVL #4, R1
            MOVL #↑XF00000, R2
1$:         MOVL (R0)+, (R2)+
            SOBGTR R1, 1$
            HALT
```

The TABLE is the microcode binary to be loaded into WCS. Note that only the 20 lower bits of the longword location are meaningful. The last word in the table has an extra bit used to enable the WCS once the microcode is loaded. The first macroinstruction points R0 toward the table. The second macroinstruction sets up R1 as the loop counter and R2 is pointed to the first longword location in WCS. At 1$ is the MOVL which pulls a longword from the table and sends it to the WCS. After this, R0 and R2 are incremented to point to the next longword in their respective locations. The SOBGTR loops until R1 is equal to zero. This example program causes the four longwords from the table to be written to WCS locations F00000, F00004, F00008, and F0000C. A similar routine could be written that would read WCS back for data checking.

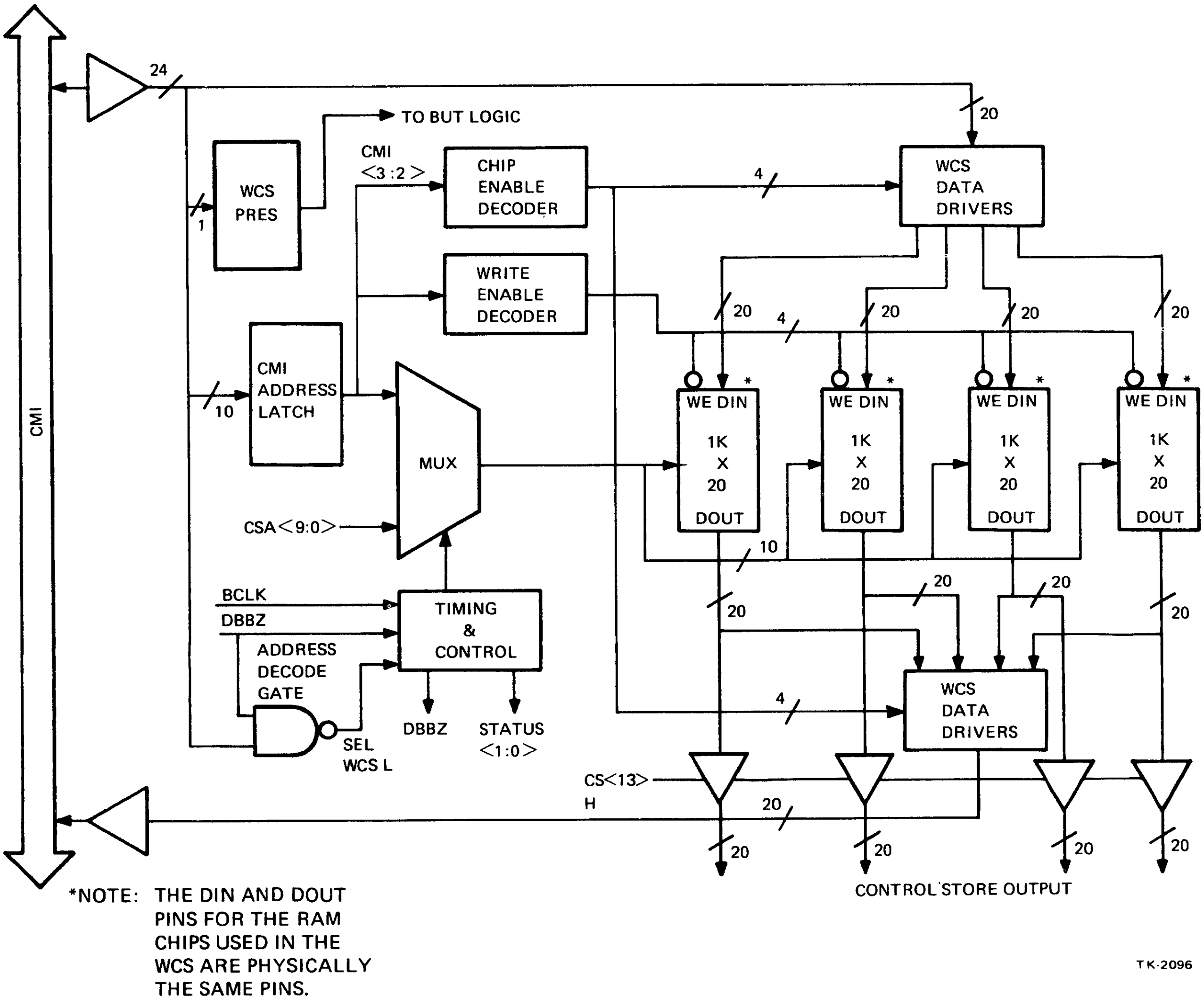
(See Figure 2-37, which is a block diagram of the WCS.) When MOVL (R0)+, (R2)+ from the previous example, is executed, it performs a CMI read for the source operand and CMI write to store to the destination, WCS. During the first write to the WCS, the address in R2 is F00000. When the CMI write occurs, address F00000 enables the NAND gate to generate the signal SEL WCS L. This signal indicates that the WCS is selected for a CMI transaction. Bits <3:2> of the CMI are used to select which 20-bit section of the WCS RAM is to be written. If bits <3:2> of the CMI address latch are 00, then the CMI data is written into bits <19:0> of the WCS location.

The following chart explains which section is enabled for bits <3:2>.

CMI Address <3:2>	**WCS RAM Written**	**CMI Data**
00	<19:00>	<19:00>
01	<39:20>	<19:00>
10	<59:40>	<19:00>
11	<79:60>	<19:00>

The output of the CMI address latch goes to the multiplexer that selects the address latch for writing and reading the WCS RAMs. When microcode executes from WCS, the same multiplexer selects the CS ADDR <9:0> lines from the microsequencer. The output of the RAMs goes to the other CPU modules where the microinstruction is latched on M CLK L. The WCS RAM data is also multiplexed back to the CMI during reads of the WCS, and the 20-bit RAM that is sourced back to the CMI is a function of address bits <3:2>.

Figure 2-38 shows the timing diagram for a CMI write cycle to the WCS. The figure shows the time the address is asserted on the CMI and time the write data is asserted. During the first B CLK L, when DBBZ L is asserted, the address and CMI are asserted. The WCS latches the address using the B CLK H signal so that the decode of the address is done in parallel. If the address is a WCS address, the signal SEL WCS L is asserted on WCS 01 in the module schematics. This causes the signal TIME 1 L and TIME 12 H to be asserted at the next B CLK L. The signal TIME 12 H prevents the CMI address latch on WCS 02 from being clobbered until the transaction is complete.



*NOTE: THE DIN AND DOUT PINS FOR THE RAM CHIPS USED IN THE WCS ARE PHYSICALLY THE SAME PINS.

Figure 2-37 1K × 80 Writable Control Store Block Diagram

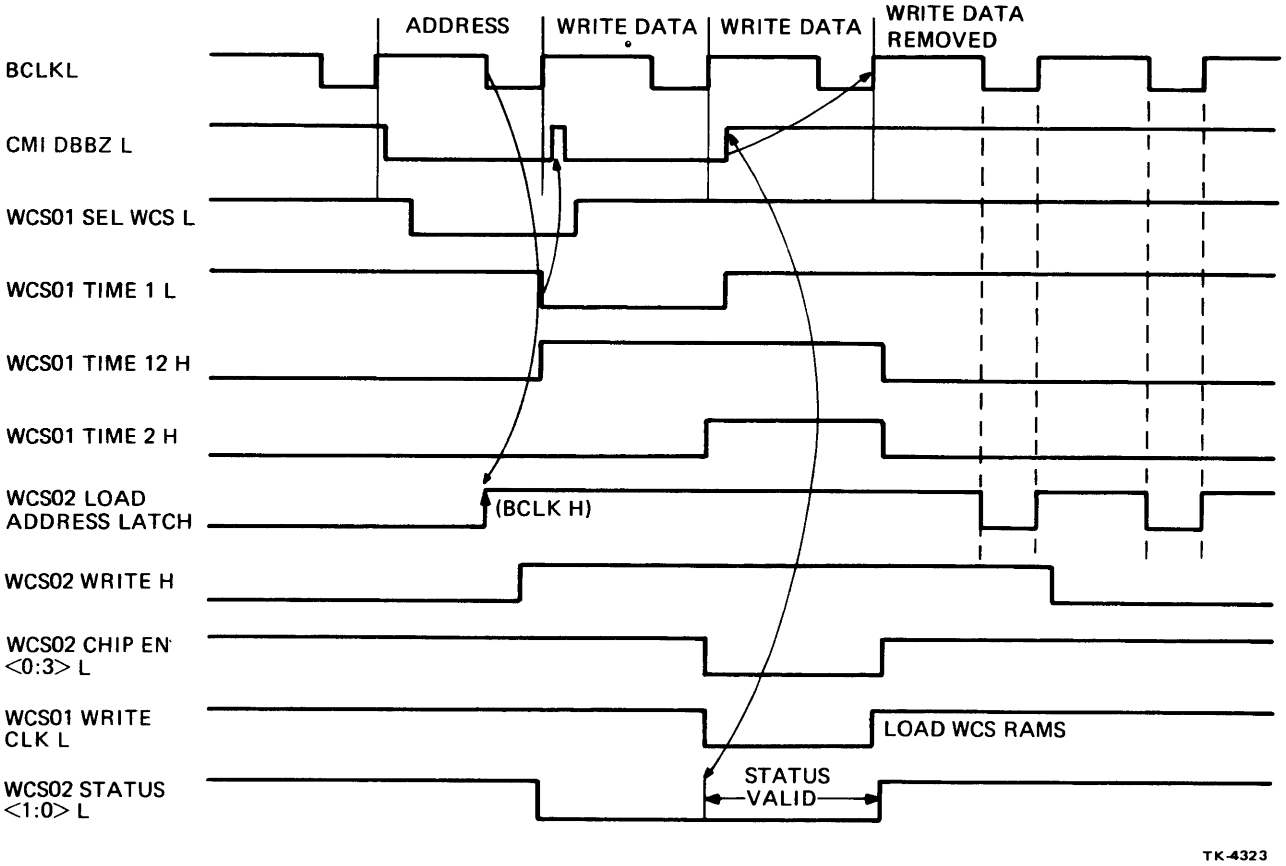


Figure 2-38 CMI Write Cycle Timing

The WCS interface logic also must decide if this is a read or write cycle. This is done by monitoring CMI DATA <27> which indicates read or write cycle. The signal WRITE H is the latched bit <27> and is used to set up the chip enables and write enables. The WCS must drive CMI DBBZ to keep the write data on this latch on WCS 02 from being clobbered until the transaction is complete. The WCS interface logic also must decide if this is a read or write cycle. This is done by monitoring CMI DATA <27> which indicates read or write cycle. The signal WRITE H is the latched bit <27> and is used to set up the chip enables and write enables. The WCS must drive CMI DBBZ to keep the write data on the bus for two cycles. The signal TIME 1 L drives CMI DBBZ L for one cycle after the address cycle so that the write data remains on the bus for two cycles. The signal TIME 12 H is used to enable the CMI status lines <1:0> which will be valid upon the negation of DBBZ L. TIME 2 H becomes the WCS RAM chip enable on writes to WCS, which occur during the second cycle that data is on the CMI. The write enable pulse that goes to all the RAM chips is generated from the signal WRT CLK L. The WCS microcode is written into the RAMs on the low pulse of WRT CLK L.

Reading the WCS requires some type of read of address F00000 to F03FFC. The program described above could be changed to read WCS address 2000 into memory.

```
WCS_DATA:        .BLKL 4

START:           MOVL #↑XF00000, R0
                 MOVL #4, R1
                 MOVAL WCS_DATA, R2
1$:              MOVL (R0)+, (R2)+
                 SOBGTR R1, 1$
                 HALT
```

This routine reads addresses F00000, F00004, F00008, and F0000C into the space allocated by the .BLKL directive called WCS_DATA. This routine could be modified to compare the write data to WCS with the data read back. During the execution of the MOVL (R0)+, (R2)+ instruction, when the source operand is fetched, a bus function micro-order causes a CMI read of the WCS. The timing diagram of the CMI read of WCS is shown in Figure 2-39. During the read transaction, after the CPU has arbitrated and won the CMI, the CMI address and CMI DBBZ L are asserted. The WCS latches address from the CMI on the low-to-high transition of B CLK H. In parallel to this the decode gate decides if this is a WCS address and asserts the signal SEL WCS L.

Again SEL WCS L is used to initiate the read cycle and prevent the address latch on WCS 02 from being clobbered during the read transaction. SEL WCS L also starts the generation of the signals TIME 1 L and TIME 12 H. On reads of WCS the WCS RAM data is available for the next CMI cycle. The signals SEL WCS L and TIME 1 L and NOT WRITE L allow the signal DRIVE CMI L to be generated for two cycles to allow the WCS to drive the 20 bits of RAM data onto the CMI for 2 cycles. During a read operation, bits <31:21> are not defined. These bits float on the CMI, and this is usually the same as receiving ones. The CMI master (CPU) clocks the read data on the next B CLK H. The read data remains on the CMI for an additional cycle after DBBZ is negated. The signal TIME 12 H is the chip enable signal on reads and so the RAMs are enabled for two CMI cycles to pass the content of the selected address to the CMI transceivers.

2.3.6.2 WCS Schematic Diagram Analysis – The timing diagram (Figure 2-39) can be used to study the schematic diagrams on WCS 01 and WCS 02. The rest of the logic is explained in the block diagram analysis. On WCS 01, in the lower left corner, is the NAND gate that determines whether or not the address on the CMI is a WCS address. This signal is called SEL WCS L and it goes to the latch E5, where at the next B CLK L, the signal TIME 1 L is asserted. On WCS 02, on the left side of the print, is the CMI address latch that is loaded at every B CLK H time. The latch is disabled if SEL WCS L generates TIME 12 H, preventing the latch from being overwritten during this CMI transaction. CMI DBBZ L is received and driven by the signal TIME 1 L for one cycle after the address has been asserted. The CMI transceivers are shown on WCS 03 and the direction of drive is a function of CMI bit <27> which indicates whether the cycle is a read or write. The signal DRIVE CMI L is asserted only during reads of WCS. Refer to Figure 2-39. WCS 03 shows the 2/1 multiplexer that selects the RAM address from either the CMI address latch or the control store microsequencer. The rest of the schematic diagrams are the RAMs themselves.

If the WCS module is added to the system after the initial delivery, it is important to remove a jumper on the backplane that disables any reference to WCS. This jumper grounds the signal CS ADD 13 H on the CCS module. The wire-wrapped jumper runs between B00548 and B00544.

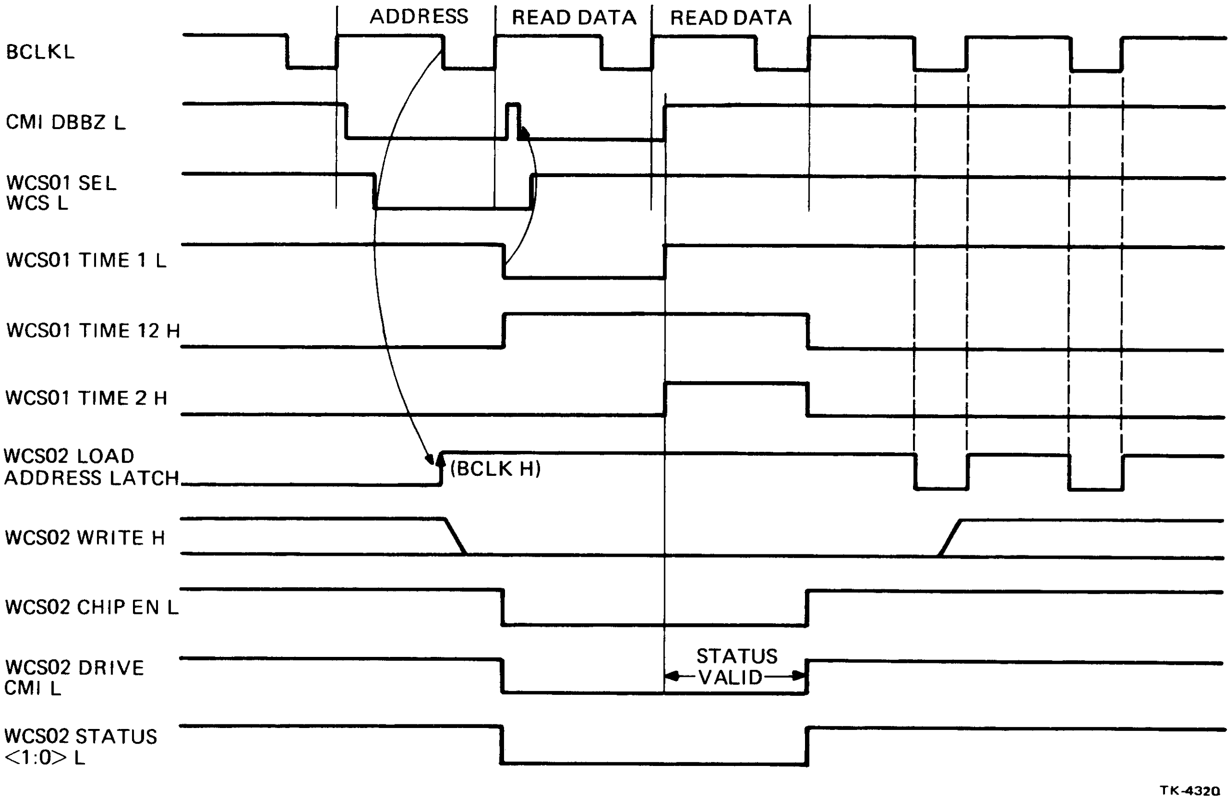


Figure 2-39 CMI Read of WCS (Timing Diagram)

2.4 INSTRUCTION DECODE OVERVIEW

Macroinstruction decode is performed by the data path module (DPM) instruction decode logic. This logic is illustrated in Figure 2-40. It consists of an instruction decode chip (IRD) and three groups of PROMs. The three PROM groups are as follows.

1. Native Mode IRD 1 PROMs (VAX instructions)

2. Native Mode IRDx PROMs (VAX instructions)

3. Compatibility Mode PROMs (for PDP-11 instructions)

Instruction stream data (ISTRM) is made available to the instruction decode logic via the memory interface and control (MIC) module execution buffer (XB). This data is received on the XBUF <15:0> H lines.

The function of native mode instruction decode is to decode a macroinstruction (i.e., MOVL R1, (R2)) to produce a base microaddress to the CCS PROMs corresponding to the macroinstruction opcode (MOVL) and an address mode offset for any operand specifiers (R1, (R2)). For native mode the opcode and first operand specifier (MOVL R1) are decoded during IRD 1 time and the second operand specifier, (R2), is decoded during IRDx time. If the instruction has more than two operand specifiers, each operand specifier is decoded in its turn. The IRD 1 PROM and IRD gate array chip decode the opcode and first operand specifier. At IRDx time the opcode, second, and third operand specifiers are decoded by the native IRDx PROM. For instructions having more than three operand specifiers, the microword BUT field specifies LOD.INC.BRA (BUT = 6). This BUT field micro-order brings in an additional operand specifier on XBUF <7:0> H. The IRD chip decodes this operand specifier and produces an address mode offset. This offset is then ORed with the microword NEXT field to provide an address for the next microinstruction to be executed.

Compatibility mode instruction decode is accomplished by the IRD gate array and the compatibility mode PROM. PDP-11 instructions have a varying format for opcodes and operands. This varying format makes it necessary for the IRD chip to encode each PDP-11 instruction opcode before using it to address the compatibility mode PROM. The PROM then produces a base microaddress to the CCS PROMs. The IRD chip, just as in native mode, provides an address mode offset to the CCS PROMs.

2.4.1 XBUF to Instruction Decode Data Transfer

See Figure 2-41. IRD 1 L and LD OSR L control the transfer of data from the MIC module execution buffer to the instruction decode logic (IRD chip and native IRD 1 PROM). Data may be transferred two bytes at a time on XBUF <15:0> H, or one-byte transfers may be done on XBUF <15:8> H or XBUF <7:0> H.

2.4.2 Instruction Decode Chip (IRD)

See Figure 2-40. The function of the IRD chip is to decode data received on XBUF <15:00> H and to output the following.

IR <7:0> H, used to address the native mode IRDx PROMs, compatibility mode PROMs, and D-size PROMs.

CS ADDR <03:00> L, used as an address mode offset to the CCS PROMs.

IRD RNUM <03:00> H, to the scratchpad address (SPA) gate array chip, selecting the general processor register to be used with the operand specifier being evaluated.

DISP ISIZE <01:00> H, used to indicate the size of an address displacement in the ISTRM.

All these outputs depend on the instruction mode (native or compatibility), instruction class (during compatibility mode), and addressing mode.

2.4.2.1 Instruction Register (INSTR REG) – See Figure 2-42. The instruction register is an 8-bit input register internal to the IRD chip. This register is loaded as specified in Table 2-4.

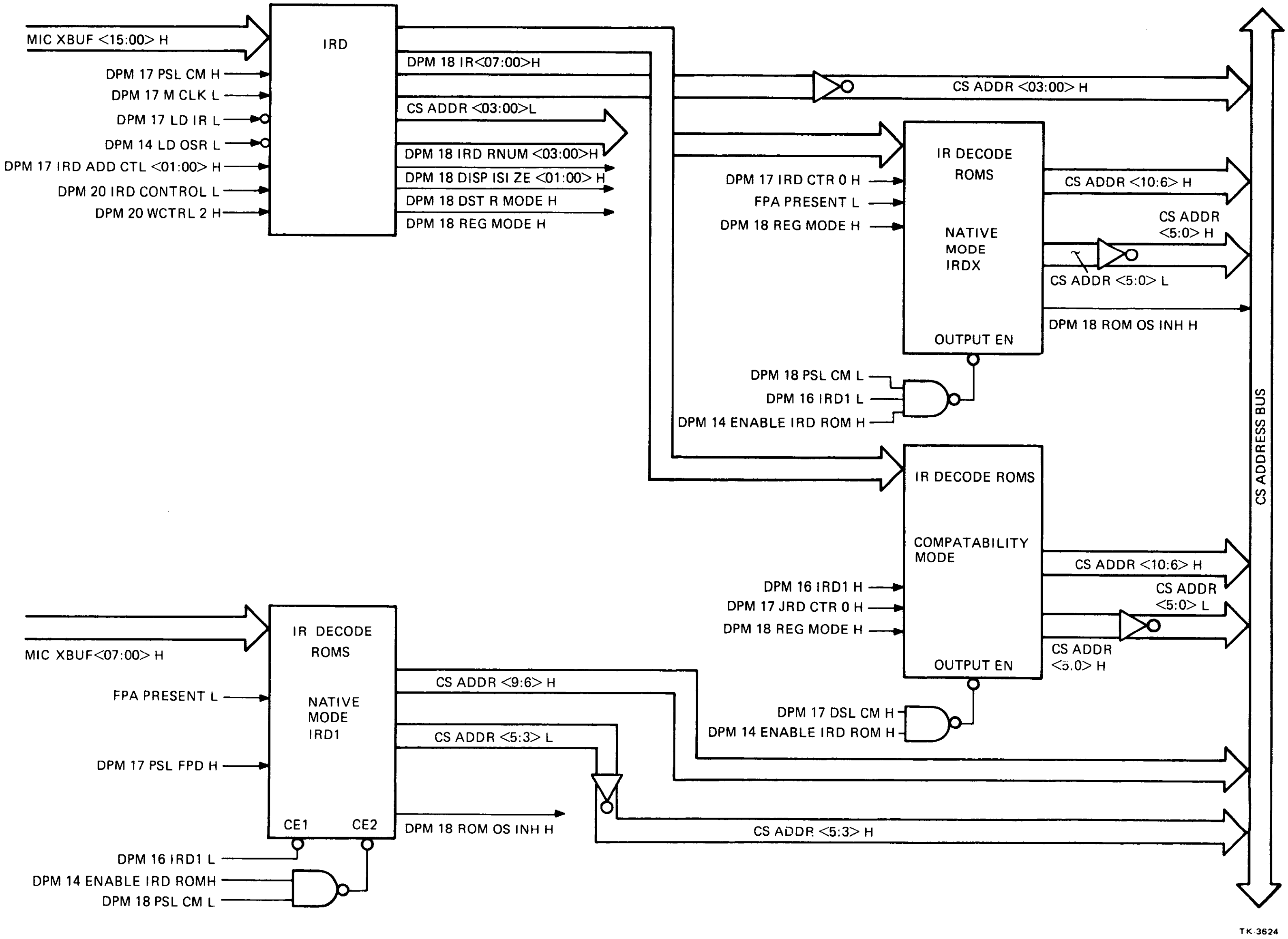
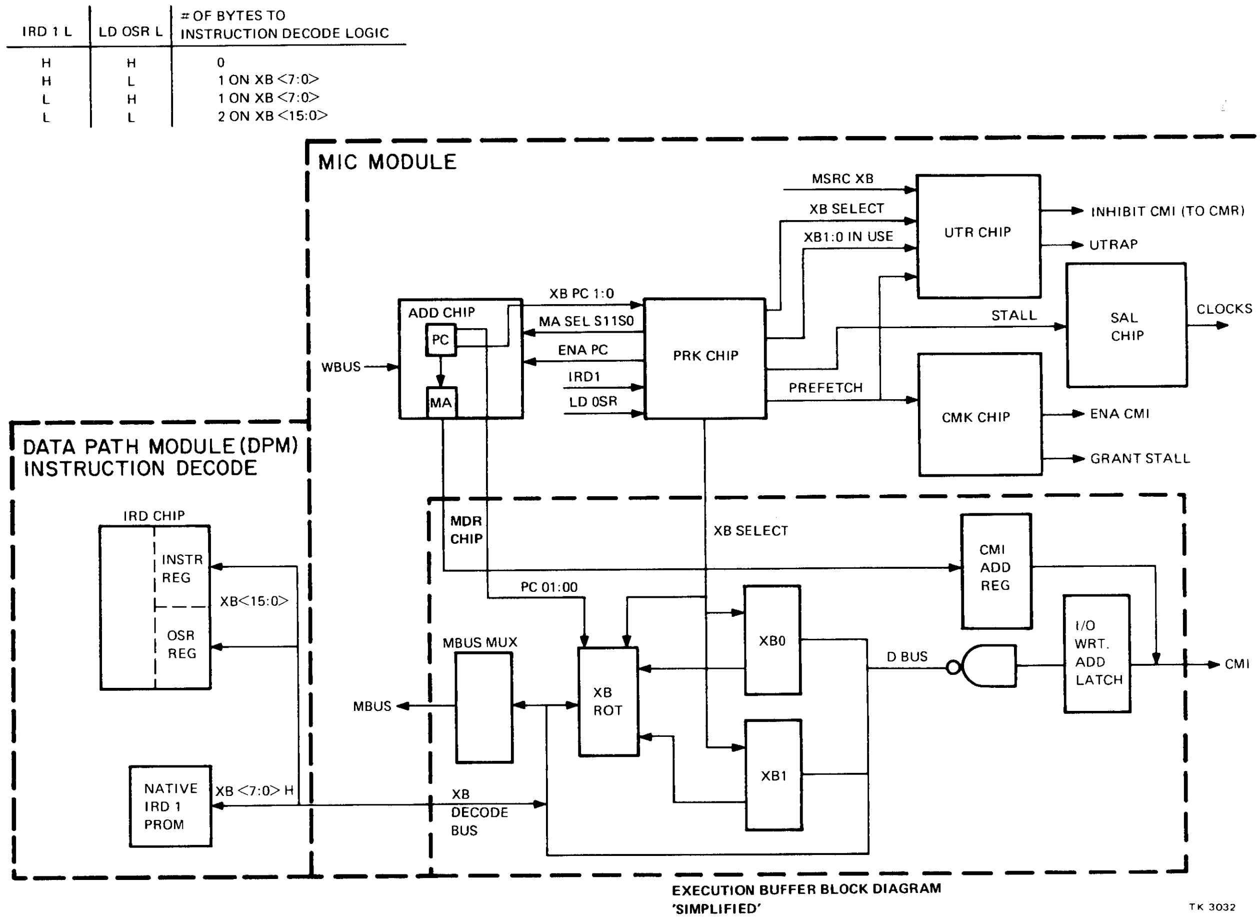


Figure 2-40 Instruction Decode Logic



IRD 1 L	LD OSR L	# OF BYTES TO INSTRUCTION DECODE LOGIC
H	H	0
H	L	1 ON XB <7:0>
L	H	1 ON XB <7:0>
L	L	2 ON XB <15:0>

Figure 2-41 Execution Buffer to Instruction Decode Transfer

Table 2-4 Loading the Instruction Register

M CLK L	PSL CM H	LD IR L	INSTR REG Loaded From
L	L	L	XBUF <07:00> H
L	H	L	XBUF <15:08> H
don't care	don't care	H	no load

For both native and compatibility modes, loading of the instruction register occurs when M CLK L is asserted. Table 2-4 shows that LD IR L must be low in order to load the instruction register. LD IR L is active when the microword BUT field specifies an IRD 1 (BUT = 4) or IRD 1 TST (BUT = 5) condition (See Tables 2-5, 2-6 and 2-7).

Load source is determined by the processor status longword (PSL) CM H bit. PSL CM H will be high for compatibility mode and low for native mode.

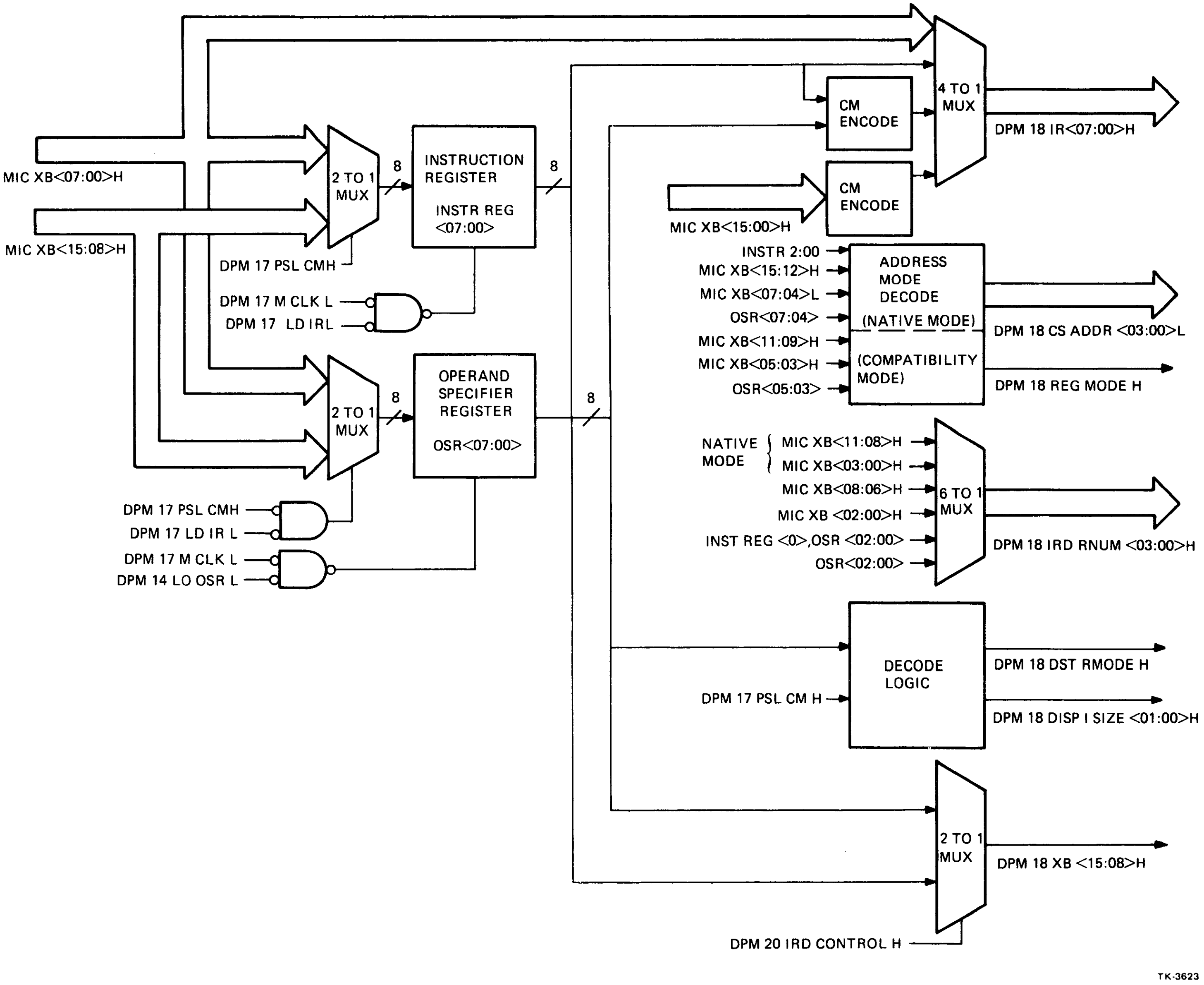


Figure 2-42 Instruction Decode Chip (IRD)

Table 2-5 Compatibility Mode Instruction Decode Hardware Conditions

BUT CODE <5:0>H	IRD CTR <2:0>H at Start of Microinstruction	Instruction Class (From Table 2-6	Control Store Address S ADDR<10:0>	INSTR REG	OSR REG	RNUM <3:0>	DSIZE LATCH	IRD CTR <2:0>H STATUS	PC	No. Bytes Requested From XB CS ADDR<3:0>L	Branch Offset Source CS ADDR<3:0>L
=4=IRDI	Don't Care	A, D2, B2 (Excluding XOR and SOB)	CM IRD ROM ORed with Table 2-13	Loaded	Loaded	Loaded with XB<2:0>	Loaded	7 During Instruction 0 at End	PC←PC+2	2	XB<5:3>
=4=IRDI	Don't Care	B1,XOR (From B2)OR SOB (From B2)	CM IRD ROM ORed with Table 2-13	Loaded	Loaded	Loaded with XB<8:6>	Loaded	7 During Instruction 0 at End	PC←PC+2	2	XB<11:09>
=4=IRDI	Don't Care	C, DI	CM IRD ROM ORed 1101	Loaded	Loaded	Loaded with 0	Loaded	7 During Instruction 0 at End	PC←PC+2	2	
=1=IRDX (OPSPEC Must Not Be Set)	0	BI, XOR (From B2)	CM IRD ROM ORed with Table 2-13	No Load	No Load	Loaded with OSR <2:0>	No Load	Increment	No Change	0	OSR <5:3>
=1=IRDX (OPSPEC Must Not Be Set	0	A, D2,B2 (Excluding XOR and SOB)	CM IRD ROM	No Load	No Load	Loaded with IR<0>OSR<7:6>	No Load	Increment	No Change	0	
=1= IRDX (OPSPEC Must Not Be Set	0	SOB	CM IRD ROM	No Load	No Load	Loaded with OSR<2:0>	No Load	Increment	No Change	0	
=1= IRDX (OPSPEC Must Not Be Set)	0	C, DI	CM IRD ROM	No Load	No Load	Loaded with 0	No Load	Increment	No Change	0	
=1= IRDX (OPSPEC Must Not Be Set)	1	Don't Care	CM IRD ROM	No Load	No Load	No Load	No Load	Increment	No Change	0	
=1=IRDX	2,3,4 5,6,7	Don't Care	Works Exactly Like a "Return"	No Load	No Load	No Load	No Load	No Change	No Change	0	
=18=BRA. ON.ADD	Don't Care	Don't Care	NXT WITH	No Load	No Load	No Load	No Load	No Change	No Change	0	OSR<5:3>

Table 2-6 Compatibility Mode Instruction Class Defined

Class A – 1 Operand
Opcode Mnemonic

Opcode			Mnemonic
00	4R	DD	JSR
00	50	DD	CLR
00	51	DD	COM
00	52	DD	INC
00	53	DD	DEC
00	54	DD	NEG
00	55	DD	ADC
00	56	DD	SBC
00	57	DD	TST
00	60	DD	ROR
00	61	DD	ROL
00	62	DD	ASR
00	63	DD	ASL
00	64	NN	MARK
00	65	SS	MFPI
00	66	DD	MTPI
00	67	DD	SXT
00	70	00	
			(Unused)
00	77	77	
10	40	00	
			EMT
10	43	77	
10	44	00	
			TRAP
10	47	77	
10	50	DD	CLRB
10	51	DD	COMB
10	52	DD	INCB
10	53	DD	DECB
10	54	DD	NEGB
10	55	DD	ADCB
10	56	DD	SBCB
10	57	DD	TSTB
10	60	DD	RORB
10	61	DD	ROLB
10	62	DD	ASRB
10	63	DD	ASLB
10	64	00	UNUSED
			(Unused)
10	64	77	

Class A – 1 Operand
Opcode Mnemonic (Cont)

Opcode			Mnemonic
10	65	SS	MFPD
10	66	DD	MTPD
10	67	00	
			(Unused)
10	77	77	

Class B1 – 2 Operand
Opcode Mnemonic

Opcode			Mnemonic
00	SS	DD	MOV
02	SS	DD	CMP
03	SS	DD	BIT
04	SS	DD	BIC
05	SS	DD	BIS
06	SS	DD	ADD
11	SS	DD	MOVB
12	SS	DD	CMPB
13	SS	DD	BITB
14	SS	DD	BICB
15	SS	DD	BISB
16	SS	DD	SUB

Class B2 – 1 1/2 Operand
Opcode Mnemonic

Opcode			Mnemonic
07	0R	SS	MU
07	1R	SS	DIV
07	2R	SS	ASH
07	3R	SS	ASHC
07	4R	DD	XOR
07	50	0R	FADD
07	50	1R	FSUB
07	50	2R	FMUL
07	50	3R	FDIV
07	50	40	
07			(Unused)
07	67	77	
07	7R	NN	SOB
17	00	00	
			Floating Point
17	77	77	

Table 2-6 Compatibility Mode Instruction Class Defined (Cont)

Class C – Branches

Opcode		Mnemonic
00	04	XXX BR
00	10	XXX BNE
00	14	XXX BEQ
00	20	XXX BGE
00	24	XXX BLT
00	30	XXX BGT
00	34	XXX BLE
10	00	XXX BPL
10	04	XXX BMI
10	10	XXX BHI
10	14	XXX BLOS
10	20	XXX BVC
10	24	XXX BVS
10	30	XXX BCC, BHIS
10	34	XXX BCS, BLO

Class D1 – Control

Opcode			Mnemonic
00	00	00	HALT
00	00	01	WAIT
00	00	02	RTI
00	00	03	BPT
00	00	04	IOT
00	00	05	RESET
00	00	06	RTT
00	00	07	
00	00	77	(Unused)

Class D2 – Control

Opcode			Mnemonic
00 1	01	DD	JMP
00	02	OR	RTS
00	02	10	(Unused)
00	02	27	
00	02	3N	SPL
00	02	40	NOP
00	02	41	Cond Codes
00	02	77	
00	03	DD	SWAB

Table 2–7 Native Mode Instruction Decode Hardware Conditions

D SIZE Latch	IRD CTR <2:0>H Status	PC	No. Bytes Requested From XB	Branch Offset Source CS ADDR<3:0>L	BUT Code <5:0>H	IRD CTR<2:0>H At Start of Microinstruction	Control Store Address CS ADDR<10:0>	INSTR Reg	OSR Reg
Loaded if ROM OS INH H=L	7 During Instruction 0 at End	PC←PC+2	1 If ROM OS INH H=H 2 If ROM OS INH H=L	XB<15:08>	=4=IRDI	Don't Care	IRDI ROM CSA ORed with Table 2–12 Unless ROM OS INH H=H	Loaded	Loaded if ROM OS INH H=L
Loaded if ROM OS INH H=L	Incremented if ROM OS INH=L	PC←PC+1 IF ROM CS INH H=L	1 If ROM OS INH H=L	XB<07:00>	=1= IRDX	0,1	IRDX ROM CSA ORed with Table 2–12 Unless ROM OS INH H=H	No Load	Loaded if ROM OS INH H=L
No Load	No Change	No Change	0		=1=IRDX	2,3,4,5,6	IRDX Works Exactly Like a "Return"	No Load	No Load
No Load	No Change	No Change	0	OSR<7:0>	=18=BRA.ON. ADD	Don't Care	NXT ORed with Table 2–12	No Load	No Load
Loaded	Increment	PC←PC+1	1	XB<07:00>	=6=LOD.INC. BRA	Don't Care	NXT ORed with Table 2–12	No Load	Loaded
No Load	No Change	PC←PC+1	1	XB<07:00>	=7=LOD.BRA	Don't Care	NXT ORed with Table 2-12	No Load	Loaded
Loaded	7 During 0 at End	PC←PC+2	2		=5=IRDITST	Don't Care	NXT	Loaded	Loaded

INSTR REG <7:0> data may be selected in whole or in part as a source for the following outputs.

1. XBUF <15:8> H receives INSTR REG <7:0> (see Paragraph 2.4.2.9).

2. IR <7:0> H receives INSTR REG <7:0> for native mode, or encoded INSTR REG <7:0> for CMODE (see Paragraph 2.4.2.3).

3. CS ADDR <2:0> receives INSTR REG <2:0> (see Paragraph 2.4.2.4).

4. IRD RNUM <2> receives INSTR <0> (see Paragraph 2.4.2.6).

2.4.2.2 Operand Specifier Register (OSR) – See Figure 2-42. The operand specifier register is internal to the IRD chip. It is an 8-bit register that is loaded under the conditions shown in Table 2-8.

Table 2-8 indicates that the OSR may be loaded during both native and compatibility modes. Data is loaded into OSR when the previous microword BUT field = 4 (IRD 1), 5 (IRD 1 TST), or 1 (IRDx).

OSR data can provide a source for the following outputs under conditions specified in the indicated sections.

1. XBUF <15:8> H gets OSR REG <7:0> (see Paragraph 2.4.2.9).

2. IR <7:0> H receives encoded OSR REG <7:0> (see Paragraph 2.4.2.3).

3. CS ADDR <3:0> L gets decoded OSR REG <7:4> or CS ADDR <2:0> L gets decoded OSR REG <5:3> (see Paragraph 2.4.2.4).

4. IRD RNUM <3:0> receives OSR REG <3:0>, or IRD RNUM <2:0> receives OSR REG <2:0>, or IRD RNUM <1:0> receives OSR REG <7:6> (see Paragraph 2.4.2.6).

5. REG MODE H = H if OSR REG <5:3> = 0 (see Paragraph 2.4.2.5).

2.4.2.3 IR <7:0> H – See Figure 2-42. IRD outputs IR <7:0> H are used for the following purposes.

1. During compatibility mode (PSL CM H = H) IR <7:0> H along with IRD CTR 0 H, and REG MODE H, are used as an address to the compatibility mode PROMs (See DPM 18, E25, E9, and E8) (See Table 2-9).

2. During native mode IRDx time IR <7:0> H, together with IRD CTR 0 H, FPA PRESENT L, and REG MODE H, are used to address the native mode IRDx PROMs (DPM 18, E27, E10, and E11) (See Table 2-10).

3. IR <7:0> H, PSL CM H, and IRD CTR <2:0> H provide an address to D-size PROM E7 (DPM 19).

4. IR <7:0> are decoded by the condition code logic (DPM 10 E13 and E70) in order to modify the state of PSL condition code bits N, Z, V, C.

5. IR <5, 3, 2, and 0> are provided to the BUT multiplexer circuitry (DPM 16, E57, and E46). Under control of certain BUT field micro-orders, these signals can be individually passed through the BUT multiplexer to the CS ADDR 00 L line. For example, BUT = 22 (IR2 is sourced to CS ADDR 00 L)

Table 2-8 Operand Specifier Register Source

M CLK L	PSL CM H	LD IR L	OSR Loaded From
L	L	H	XBUF <7:0> H
L	L	L	XBUF <15:8> H
L	H	don't care	XBUF <7:0> H

Table 2-9 Compatibility Mode ROM Addressing

	REG	IRD 1 Time		IRDx Time	
IR<7:0>	MODE	IRD 1 H=H,	IRD 1 H=L,	IRD 1 H=L,	Macro-
H	H	IRD CTR 0=1	IRD CTR 0=1	IRD CTR 0=1	instruction
	=1	REG [CM.OS.WRT]	[CM.JSR]	[IE.BAD.IRD],	;JSR,6
= OA1	=0	MEM [CM.OS.WRT]	[CM.JSR]	[IE.BAD.IRD]	
	=1	REG [CM.OS.WRT]	[CM.JSR]	[IE.BAD.IRD],	;JSR,7
=OB0	=0	MEM [CM.OS.WRT]	[CM.JSR]	[IE.BAD.IRD]	
	=1	REG [CM.OS.WRT]	[CM.JSR]	[IE.BAD.IRD],	;JSR,8
=OB1	=0	MEM [CM.OS.WRT]	[CM.JSR]	[IE.BAD.IRD]	
	=1	REG [CM.MFPD-REG]	[IE.BAD.IRD]	[IE.BAD.IRD],	;MFPD
=OAD	=0	MEM. [CM.OS.RED]	[CM.MFPD-MEM]	[IE.BAD.IRD]	

The IR <7:0> H outputs (Figure 2-42) are derived from various sources depending on the conditions shown in Table 2-11.

Table 2-11 shows that the IR <7:0> H outputs are affected by the state of PSL CM H and LD IR L. For native mode IRD 1 IR <7:0> H receives XB <07:00> H. During native mode IRDx IR <7:0> H gets INSTR REG <7:0> H. Due to the format of PDP-11 instruction opcodes, during compatibility mode instruction decode IR <7:0> H receives an encoded version of XB <15:00> H if LD IR L = L, or INSTR REG <7:0> H and OSR <7:0> H if LD IR L = H. Table 2-12 shows the encoding for compatibility mode IR <7:0> H.

2.4.2.4 CS ADDR <03:00> L – See Figure 2-40. CS ADDR <3:0> L are used to provide an address mode offset for both native and compatibility mode instructions. CS ADDR <3:0> L are inverted before being placed on the CS address bus. Address mode branch offsets are shown in Tables 2-13 (native mode) and 2-14 (compatability mode).

Table 2-10 Native IRD ROM Addressing

XB<7:0> H (Note 4)		FPA PRESENT L=H		FPA PRESENT L=L		
IR<7:0> H	IRD 1 L	REG MODE H=H	REG MODE H=L	REG MODE H=H	REG MODE H=L	Macro-instruction
		OPS REG	MEM	OPS FPA REG	FPA MEM	
=0DO	=L	FPD [NOP] [IE. OPCOD.DEC] IRD1 [LOD] [OS.RED]		[NOP] [IE.OPCOD.DEC] [LOD] [OS.RED]		:MOVL
	=H	CNTC [LOD] [IL.MOV.B.W.L.REG] CNT1 [NOP] [IL.MOV.B.W.L.MEM]	[OS.WRT2] [IL.MOV.B.W.L.MEM]	[LOD] [IL.MOV.B.W.L.REG] [NOP] [IL.MOV.B.W.L.MEM]	[OS.WRT2] [IL.MOV.B.W.L.MEM]	
=07D	=L	FPD [NOP] [IE.OPCOD.DEC.] IRD1 [LOD] [OS.QRED]		[NOP] [IE.OPCOD.DEC] [LOD] [OS.QRED]		:MOVQ
	=H	CNT0 [LOD] [IL.MOVQ] CNT1 [NOP] [IL.MOVQ]	[OS.WRT2] [IL.MOVQ]	[LOD] [IL.MOVQ] [NOP] [IL.MOVQ]	[OS.WRT2] [IL.MOVQ]	
=DBO	=L	FPD [NOP] [IE.OPCOD.DEC] IRD1 [LOD] [OS.RED]		[NOP] [IE.OPCOD.DEC] [LOD] [OS.RED]		:MOVW
	=H	CNT0 [LOD] [IL.MOV.B.W.L.REG] CNT1 [NOP] [IL.MOV.B.W.MEM]	[OS.WRT2] [IL.MOV.B.W.L.MEM]	[LOD] [IL.MOV.B.W.L.REG] [NOP] [IL.MOV.B.W.L.MEM]	[OS.WRT2] [IL.MOV.B.W.L.MEM]	

NOTES:

1. FPD (First Part Done), refers to the processor status longword (PSL) FDP H BIT. If PSL FPD H=H, the native IRD 1 ROM outputs the beginning microaddress for this field (IE.OPCOD.DEC).
2. IRD CTR 0 = 0, IRD 1 ROM outputs beginning microaddress for field name shown. = 1, IRD 1 ROM outputs beginning microaddress for field name shown.
3. OPS, refers to IRD DSR REG. NOP = Do Not Load, IRD 1 OUTPUT ROM OS INH H=H. LOD = Load, IRD 1 OUTPUT ROM OS INH H=L.
4. At IRD 1 time use XB <7:0> H

Table 2-11 IR <7:0> H Source Control

PSL CM H	LD IR L	IR <7:0> H Receives
L	L	XB <07:00> H
L	H	INSTR REG <7:0> H
H	L	See Table 2-12. Opcode <15:00> = XB <15:00> H.
H	H	See Table 2-12. Opcode <15:08> = INST REG <7:0> H. Opcode <07:00> = OSR <7:0> H.

Table 2-12 Compatibility Mode IR <7:0> H Encoding

Instr. Class from Table 2-6	IR 7	IR 6	IR 5	IR 4	IR 3	IR 2	IR 1	IR 0
A	H	L	Opcode <8>	Opcode <7>	Opcode <15>	Opcode <10>	Opcode <09>	Opcode <06>
B1	L	H	L	Opcode <7>	Opcode <15>	Opcode <14>	Opcode <13>	Opcode <12>
B2	L	H	H	Opcode <7>	Opcode <15>	Opcode <10>	Opcode <09>	Opcode <11>
C	L	L	L	H	Opcode <15>	Opcode <10>	Opcode <09>	Opcode <08>
D1	H	H	L	Opcode <7>	Opcode <15>	Opcode <02>	Opcode <01>	Opcode <00>
D2	H	H	H	Opcode <7>	Opcode <15>	Opcode <04>	Opcode <05>	Opcode <06>

Notes:

1. For each instruction class, certain IR <7:0> H bits are forced high (H) or low (L); e.g., Class A, IR 7 = H and IR 6 = L.

2. LD IR L determines how opcode <15:0> are to be defined as follows:

LD IR L	Opcode Definition
L	Opcode = corresponding XB 15:0 bit
H	Opcode <15:8> = INSTR REG <7:0> H Opcode <7:0> = OSR REG <7:0> H

Table 2-13 Native Mode Branch Offset to Operand Specifier Routines

CCS ADDR <3:0> Branch Offset	Operand Specifier			
	Mode	Register	Addressing Mode	
0000	5	O–F	Rn	Register Mode
0001	8	O–E	(Rn)+	Autoincrement Mode
0010	8	F	I↑#cons	Immediate Mode
0011	0–3	–	S↑#cons	Literal Mode
0100	7	O–F	−(Rn)	Autodecrement Mode
0101	A, C, E	F	Addr	Relative Mode
0110	A, C, E	O–E	D(Rn)	Displacement Mode
0111	9	F	@#Addr	Absolute Mode
1000	6	O–F	(Rn)	Register Deferred Mode
1001	B, D, F	F	@Addr	Relative Deferred Mode
1010	B, D, F	O–E	@D(Rn)	Displacement Deferred Mode
1011	9	O–E	@(Rn)+	Autoincrement Deferred Mode
1100	4	F	(Rn)[PC]	Index Mode PC
1101	4	O–E	(Rn)[Rx]	Index Mode

Table 2-14 Compatibility Mode Branch Offset to Operand Specifier Routines

CS ADDR <3:0> Branch Offset	Operand Specifier			
	Mode	Register	Addressing Mode	
0000	0	0–6	Rn	Register Mode
0001	0	7	PC	Register Mode PC
0010	1	0–6	(Rn)	Register Deferred Mode
0011	1	7	(PC)	Register Deferred Mode PC

Table 2-14 Compatibility Mode Branch Offset to Operand Specifier Routines (Cont)

CS ADDR <3:0> Branch Offset	Operand Specifier			
	Mode	Register	Addressing Mode	
0100	2	0–5	(Rn)+	Autoincrement Mode
0101	2	6	(SP)+	Autoincrement Mode SP
0110	3	0–6	@(Rn)+	Autoincrement Deferred Mode
0111	3	7	@#Addr	Absolute Mode
1000	4	0–5	−(Rn)	Autodecrement Mode
1001	4	6	−(SP)	Autodecrement Mode SP
1010	5	0–7	@−(Rn)	Autodecrement Deferred Mode
1011	4	7	−(PC)	Autodecrement Mode PC
1100	6	0–6	X(Rn)	Index Mode
1101	6	7	Addr X(PC)	Relative Mode
1110	7	0–7	@ADDR @X(Rn)	Index Deferred Mode
1111	2	7	#CONS	Immediate Mode

The CS ADDR <3:0> L branch offset source is dependent on a number of factors, as follows.

Table 2-15 shows the branch offset sources for both native and compatibility mode instructions. The CS ADDR <3:0> L branch offset source is determined by IRD ADD CTL <1:0> H, PSL CM H, and LD IR L. CS ADDR <3:0> L can be sourced from a decode of the XB data, OSR data or INSTR REG data. Table 2-15 references Table 2-16 (native mode) and Table 2-17 (compatibility mode) in order to show the decode for each instruction type or class. In Tables 2-16 and 2-17, AMODE is the data presented to the ADDR mode decode logic internal to the IRD chip. A decode of AMODE <3:0> and IRD RNUM <3:0> produces the CS ADDR <3:0> L address mode offset. CS ADDR <3:0> L is inverted before addressing the CCS PROMs.

2.4.2.5 REG MODE H - See Figure 2-40. REG MODE H is used to indicate that the instruction being decoded specifies register mode. REG MODE H is used as address bit 0 for both native IRDx and compatibility mode PROMs. Tables 2-9 and 2-10 show the effect of REG MODE H on the native IRDx and the compatibility mode PROM address.

The output state of REG MODE H is determined by the conditions shown in Table 2-18. Also see Figure 2-42.

Table 2-15 CS ADDR <3:0> L Source

IRD ADD CTL <1:0> H	PSL CM H	LD IR L	Instruction Class (from Table 2-6)	CS ADDR <3:0> L and TRD RNUM <3:0> H
0	X	X	X	CS ADDR <3:0> L = 1111 (No branch)
1	L	L	Native Mode	(See Table 2-16) here AMODE <3:0> = XB <7:4> H and IRD RNUM <3:0> = XB <11:08> H
1	L	H	Native Mode	(See Table 2-16) here AMODE <3:0> = XB <7:4> H and IRD RNUM <3:0> = <03:00> H
1	H	L	A,D2,B2 (XB <11:09> ≠4 or 7)	(See Table 2-17) here AMODE <2:0> = <5:3> H and IRD RNUM <2:0> = XB <2:0> H
1	H	L	B1,B2 (XB <11:09> = 4 or 7)	(See Table 2-17) here AMODE <2:0>= XB <11:09> H and IRD RNUM <2:0> = XB <08:06> H
1	H	L	C,D1	CS ADDR <3:0> L = 0001
2	L	X	Native Mode	(See Table 2-16) here AMODE <3:0> = OSR <7:4> H IRD RNUM <3:0> OSR <3:0> H
2	H	X	B1,B2 (INSTR REG <3:1> = 4)	(See Table 2-17) here AMODE <2:0> = OSR <5:3> H IRD RNUM <2:0> = OSR <2:0> H
2	H	X	Other	CS ADDR <3:0> L = 1111 (No branch)
3	X	X	X	CS ADDR <3>L = H CS ADDR <2:0> L ← INSTR REG <2:0> L

Note: X = Don't care; ≠ Not equal to

Table 2-16 Native Mode CS ADDR <3:0>

AMODE <3:0>	IRD RNUM <3:0>	(CS ADDR BUS) CS ADDR <3:0> H
0, 1, 2, 3	X	0011
4	0–14	1101
4	15	1100
5	0–15	0000
6	0–15	1000
7	0–15	0100
8	0–14	0001
8	15	0010
9	0–14	1011
9	15	0111
10, 12, 14	0–14	0110
10, 12, 14	15	0101
11, 13, 15	0–14	1010
11, 13, 15	15	1001

Note: X = IRD RNUM <3:0> not used.

Table 2-17 Compatibility Mode CS ADDR <3:0>

AMODE <3:0>	IRD RNUM <2:0>	(CS ADDR BUS) CS ADDR <3:0> H
0	0–6	0000
0	7	0001
1	0–6	0010
1	7	0011
2	0–5	0100
2	6	0101

Table 2-17 Compatibility Mode CS ADDR <3:0> (Cont)

AMODE <3:0>	IRD RNUM <2:0>	(CS ADDR BUS) CS ADDR <3:0> H
2	7	1111
3	0–6	0110
3	7	0111
4	0–5	1000
4	6	1001
4	7	1011
5	X	1010
6	0–6	1100
6	7	1101
7	X	1110

Note: X = IRD RNUM <2:0> not used.

Table 2-18 REG MODE H Output Source

PSL CM H	LD IR L	LD OSR L	IRD ADD CTL <1:0> H	Instruction Class (from Table 2-6)	REG MODE H
L	L	L	X		1 if XB <15:12> = 5
L	L	H	X	Native	0
L	H	L	X	Native	1 if XB <07:04> = 5
L	H	H	X	Native	0
H	L	X	X	A, D2, B2 (XB <11:09> ≠4 or 7)	1 if XB <05:03> = 0
H	L	X	X	B1, B2 (XB <11:09> = 4 or 7)	1 if XB <11:09> = 0
H	X	X	X	D1, C	0
H	X	X	2	X	1 if OSR <5:3> = 0
	Otherwise				0

Note: X = Don't care; ≠ means Not equal to.

2.4.2.6 IRD RNUM <3:0> H – See Figure 2-42. IRD RNUM <3:0> H specifies the number of the register associated with an operand specifier being evaluated. Here the value of IRD RNUM <3:0> H is loaded into the RNUM register located in the scratchpad address (SPA) gate array chip. RNUM register is loaded on the rising edge of M CLK L when LD RNUM H = H. The RNUM register contents are used to specify a source or destination register number during instruction execution.

IRD RNUM <3:0> H is also used internal to the IRD chip, along with AMODE <3:0> to determine the output on CS ADDR <3:0> H. This is shown in Tables 2-16 and 2-17.

IRD RNUM <3:0> H may be sourced from XB data, INSTR REG data, or OSR REG data. The data source depends on PSL CM H and LD IR L. (See Table 2-19.)

2.4.2.7 DST RMODE H – DST RMODE H is output to the MIC module. Here it is input to the cache controller (CAK), address controller (ADK) and the CPU memory controller (CMK) gate arrays. These gate arrays are located on MIC 6 and MIC 7 respectively. When asserted, DST RMODE H prohibits a data write operation to cache or memory. At this time data is written into the general processor register (GPR) specified by RNUM.

See Figure 2-42. During native mode the state of DST RMODE H is determined by OSR <7:4>. During compatibility mode the DST RMODE H output is determined by OSR <5:3>. These conditions are shown in Table 2-20.

2.4.2.8 DISP ISIZE <01:00> H – See Figure 2-42. DISP ISIZE <01:00> H are used for native displacement addressing mode instructions. They indicate the size of an address displacement in the I-Stream. DISP ISIZE <01:00> H are output to the DSIZE <1:0> H and ISIZE <1:0> L multiplexers on DPM 19. (See Table 2-21.)

2.4.2.9 XB <15:08> H – See Figure 2-42. XB <15:00> H may be used to transfer INSTR REG and OSR REG data from the IRD chip to the memory data register (MDR) gate array chips on MIC 1 and 2. This operation is necessary when an operand specifier input to the IRD chip is to be used as data instead of being used to specify address mode and register number (e.g., short literal mode and branch instruction destinations). This is necessary because by the time it is realized that a condition such as the above exists: the OSR or INSTR REG has already been loaded with the data, the PC has been incremented past the byte needed, and the byte of data is lost to the XB.

INSTR REG <7:0> H is transferred to the MDR when the microword WCTRL field = 2B (MDR_IR). OSR REG <7:0> H is sent to the MDR by a WCTRL (MDR_OSR.CCBR_BRATST) micro-order (WCTRL = 2F). In both cases the data is transferred to MDR and zero-extended.

When used as an IRD output, XB <15:08> H are enabled and driven as shown in Table 2-22.

2.4.3 IRD 1 (Native Mode) PROM

See Figure 2-40. The IRD 1 (native mode) PROM is composed of two 1K × 4 bit PROMs. These PROMs are enabled when PSL CM H indicates native mode and the microword BUT field specifies IRD 1. The native IRD 1 PROM output becomes the base address for a routine that is used to evaluate operand specifiers for the macroinstruction being decoded.

Table 2-19 IRD RNUM <3:0> H Source

PSL CM H	LD IR L	Instruction Class (From Table 2-6)	IRD RNUM <3:0> H
L	L	Native	XB <11:08> H
L	H	Native	XB <03:00> H
H	L	A, D2, B2(XB <11:09> ≠ 4 or 7)	0, XB <02:00> H
H	L	B1, B2(XB <11:09> ≠ 4 or 7)	0, XB <08:06> H
H	X	C, D1	0
H	H	A, D2, B2(INSTR REG <3:1> = 4 or 7)	0, INSTR REG <0> H OSR <7:6> H
H	H	B1, B2(INSTR REG <3:1> = 4 or 7)	0, OSR <2:0> H

NOTE: X = don't care; ≠ means not equal to.

Table 2-20 DST RMODE H Determination

PSL CM H	OSR 7 H	OSR 6 H	OSR 5 H	OSR 4 H	OSR 3 H	DST RMODE H
L	L	H	L	H	X	H
L		Otherwise			X	L
H	X	X	L	L	L	H
H	X	X		Otherwise		L

Note: X = Don't care; Otherwise = not register mode.

Table 2-21 DISP I-Size

PSL CM H	OSR <7:4> H	DISP I-Size <1:0> H
H	X	0
L	= 10, 11	1 (Byte)
L	= 12, 13	2 (Word)
L	= 14, 15	3 (Longword)
L	Other	0

Table 2-22 XB <15:08> H Output

IRD CONTROL H	WCTRL 2 H	XB <15:08> H
L	X	Z (High Impedance)
H	L	INSTR REG <7:0> H
H	H	OSR <7:0> H

2.4.3.1 Native IRD 1 PROM Enables – See Figure 2-40. The native IRD 1 PROM is enabled by the following signals.

IRD 1 L – This signal is produced by a decode of the BUT field. When BUT = 4 (IRD 1), circuitry on DPM 16 produces IRD 1 L.

ENABLE IRD ROM H – This signal is active during IRD 1 and IRDx time when IRD CTR <2:0> H = 7,0 or 1. ENABLE IRD ROM H is output from the MSQ gate array chip on DPM 14.

PSL CM L – This signal is derived by inverting PSL CM H (DPM 18). PSL CM H is output from the PHB gate array chip on DPM 17. PSL CM H is high for compatibility mode and low for native mode.

The IRD 1 native mode PROM is enabled when IRD 1 L = L and ENABLE IRD ROM H = H and PSL CM L = H.

2.4.3.2 Native IRD 1 PROM Addressing – The native IRD 1 PROM is addressed as follows.

ROM Address	Source	Comments
9:2	XB <7:0> H	Opcode of instruction
1	FPA PRESENT L	0 = Floating-point option present
0	PSL FPD H	1 if FPD bit in PSL set

Table 2-10 is a composite of both native IRD 1 and IRDx PROM addressing possibilities. This table shows a few of the routine look-ups resident in the native IRD 1 and IRDx PROMs. For example, if the macroinstruction being decoded specifies a MOV L. XB <7:0> H = 0D0, FPA PRESENT L = H, REG MODE H = H, and IRD 1 L = L, the IRD 1 PROM will output the base microaddress of the OS.RED operand specifier routine. This address is presented to the CCS PROMs on CS ADDR <9:3> H.

For most instructions the IRD gate array supplies an address mode offset on CS ADDR <3:0> L (See Table 2-13). However, for instructions that do not have operand specifiers, such as NOP (no operation), the IRD 1 PROM outputs a signal called ROM OS INH H. When ROM OS INH H is asserted, the MSQ gate array (DPM 14) forces LD OSR A L high. LD OSR A L being high causes the PHB gate array (DPM 17) to output 00 on IRD ADD CTL <1:0> H. Both IRD ADD CTL <1:0> H and ROM OS INH H = H are applied to the IRD chip. As a result, the IRD CS ADDR <3:0> L outputs go open. These lines are then pulled high by pull-up resistors on DPM 14. The CCS PROMs receive an address mode offset of 0 on CS ADDR <3:0> H.

2.4.4 IRDx (Native Mode) PROM

See Figure 2-40. The IRDx (native mode) PROM consists of three 2K × 4 bit PROMs used as a single 2K × 12 bit PROM. The native IRDx PROMs are enabled when the microword BUT field specifies IRDx (000001). The IRDx PROM output is used to provide a base address to the CCS PROMs for evaluation of the second and third operand specifiers of the macroinstruction being decoded.

2.4.4.1 Native IRDx PROM Enables – See Figure 2-40. The native IRDx PROMs are enabled by the following signals.

IRD 1 L = H
PSL CM L = H
ENABLE IRD ROM H = H

The origin of these signals and the conditions under which they are produced are detailed in Paragraph 2.4.3.1.

2.4.4.2 Native IRDx PROM Addressing – The native IRDx PROMs are addressed as shown below.

IRDx PROM Address	Source	Comments
10:3	IR <7:0> H	The opcode is latched into the IRD chip INSTR REG <7:0> during IRD 1. At IRDx this data is output as an address to the IRDx ROMs.
2	IRD CTR 0 H	This signal is the LSB of IRD CTR <2:0> H. IRD CTR <2:0> H are output from the SAC gate array on DPM 17. This count is forced to 7 at the beginning of IRD 1 and goes to 0 for the second operand specifier. (See Table 2-7.) IRD CTR <2:0> H is incremented by LD OSR L each time an operand specifier is loaded into the IRD chip OSR REG.
1	FPA PRESENT L	Low = Floating-point option is present. This signal comes from DPM 10, backplane pin <B17>.
0	REG MODE H	A signal output by the IRD chip under the conditions shown in Table 2-18. High = register mode operand specifier being evaluated.

Table 2-10 shows how the above signals address the IRDx ROM. Continuing with the example shown in Paragraph 2.4.3.2 (decoding a MOVL macroinstruction), for purpose of illustration assume that the complete instruction is MOVL R1, R2. In this case the following address is presented to the IRDx ROM.

IR <7:0> H = ODO
IRD 1 L = H
REG MODE H = H
FPA PRESENT L = H
IRD CTR 0 H = L

The IRDx ROM outputs a microaddress for the look-up corresponding to this address [IL. MOV. B.W.L. REG]. For this particular instruction decode IRDx supplies the entire CCS address (CS ADDR <10:0>). ROM OS INH H from the IRDx ROM is high. ROM OS INH H is generated in this case under the same conditions and for the same purposes detailed in Paragraph 2.4.3.2.

2.4.5 Compatibility Mode ROM

See Figure 2-40. The compatibility mode ROMs consist of three 2K × 4 bit PROMs used as one 2K × 11 bit ROM. The LSB output bit is not used. These ROMs perform the same function as the native mode ROMs in that they provide a microaddress to the CCS ROMs which is based on the opcode of the instruction being decoded.

2.4.5.1 Compatibility Mode ROM Enables – See Figure 2-40. The compatibility mode PROMs are enabled by the following signals and conditions.

PSL CM H = High – This signal, output by the PHB gate array (DPM 17), is latched high during compatibility mode.

ENABLE IRD ROM H = High – This signal is supplied by the MSQ gate array (DPM 14) during IRD 1 and IRDx time when IRD CTR <2:0> H = 7, 0 or 1.

2.4.5.2 Compatibility Mode ROM Addressing – The compatibility mode ROM during both IRD 1 and IRDx is addressed in the following way.

ROM Address	Source	Comments
10:3	IR <7:0>	This is the PDP-11 opcode after encoding in the IRD gate array (see Table 2-12).
2	IRD 1 H	This signal comes from DPM 16, and it is active (high) during IRD 1 time BUT <5:0> H = 4. At IRDx time, BUT <5:0> H = 1, and IRD 1 H goes low.
1	IRD CTR 0 H	This is the LSB of IRD CTR <2:0> H. At IRD 1 time the IRD CTR <2:0) H count equals 7, and it is changed to 0 at the first IRDx. The count increments by one each time the BUT field indicates IRDx.
0	REG MODE H	This output from the IRD chip indicates that the operand being decoded is register mode (see Table 2-18).

A JSR, 6 macroinstruction is used to illustrate how a compatibility mode instruction is decoded (see Table 2-9).

CM PS L H = H
IRD ROM H = H
IR <7:0> H addresses ROM location OA1
REG MODE H = H
IRD 1 H = H
IRD CTR 0 H = H

For the conditions shown above, the compatibility mode ROM outputs the base address for operand specifier routine REG [CM.OS.WRT]. This address is on CS ADDR <10:4>. The address mode offset is supplied by the IRD chip CS ADDR <3:0> L outputs. (See Table 2-15.)

2.4.6 BUT Field Conditions Used for Instruction and Operand Specifier Decode

See Tables 2-5 and 2-7. So far the only two BUT field conditions mentioned have been IRD 1 and IRDx. There are, however, four more BUT field conditions that are related to instruction decode. They are as follows.

BRA.ON.ADD – Used for decode of operand specifiers already loaded in OSR.

LOD.INC.BRA – Used for decode of operand specifiers not loaded in OSR.

LOD.BRA – Used to decode the operand specifier that specifies the base operand address for an index mode specifier.

IRD1TST – Used to test the loading of the IR and OSR.

Tables 2-5 and 2-7 show in detail the hardware condition existing during the occurrence of these BUT field conditions.

2.4.7 Decoding a MOVL R1, R2 and NOP Macroinstruction

MOVL R1, R2 is a native mode (VAX-11) macroinstruction. This instruction moves a longword from R1 to R2. MOVL R1, R2 is a good example of two-operand instruction decode. NOP means that no operation occurs. For the NOP instruction no operand specifiers are involved in the instruction decode.

2.4.7.1 MOVL R1, R2 Instruction Decode – See Figure 2-43. Decode of this macroinstruction is performed as follows.

IRD 1:

The BUT field of the last microword specified IRD 1 (BUT = 4) (Figure 2-43, Sheet 1, *1).

PSL CM H = L and PSL CM L = H (Figure 2-43, Sheet 1, *2).

The IRD chip INSTR REG is loaded with the opcode on XB <7:0> H (MOVL = 0D0) (Figure 2-43, Sheet 1, *3).

The native mode IRD 1 ROM is addressed as follows.

XBUF <7:0> H = 0D0
FPA PRESENT L = H
PSL FPD H = L

NOTE
The IRD 1 ROM is enabled by ENABLE IRD ROM H = H, PSL CM L = H, and IRD 1 L = L.

The native mode IRD 1 ROM outputs an address (CS ADDR <9:3>) to the CCS PROMs for OS.RED. This is the base address of the operand specifier routine. ROM OS INH H is low because an operand specifier, R1, is associated with this opcode. (See Figure 2-43, Sheet 1, *5.)

The IRD chip OSR REG is loaded from XB <15:8> H (see Figure 2-43, Sheet 1, *7). OSR REG <7:4> are decoded in order to produce CS ADDR <3:0> L. This is used as an address mode offset to the CCS PROMs.

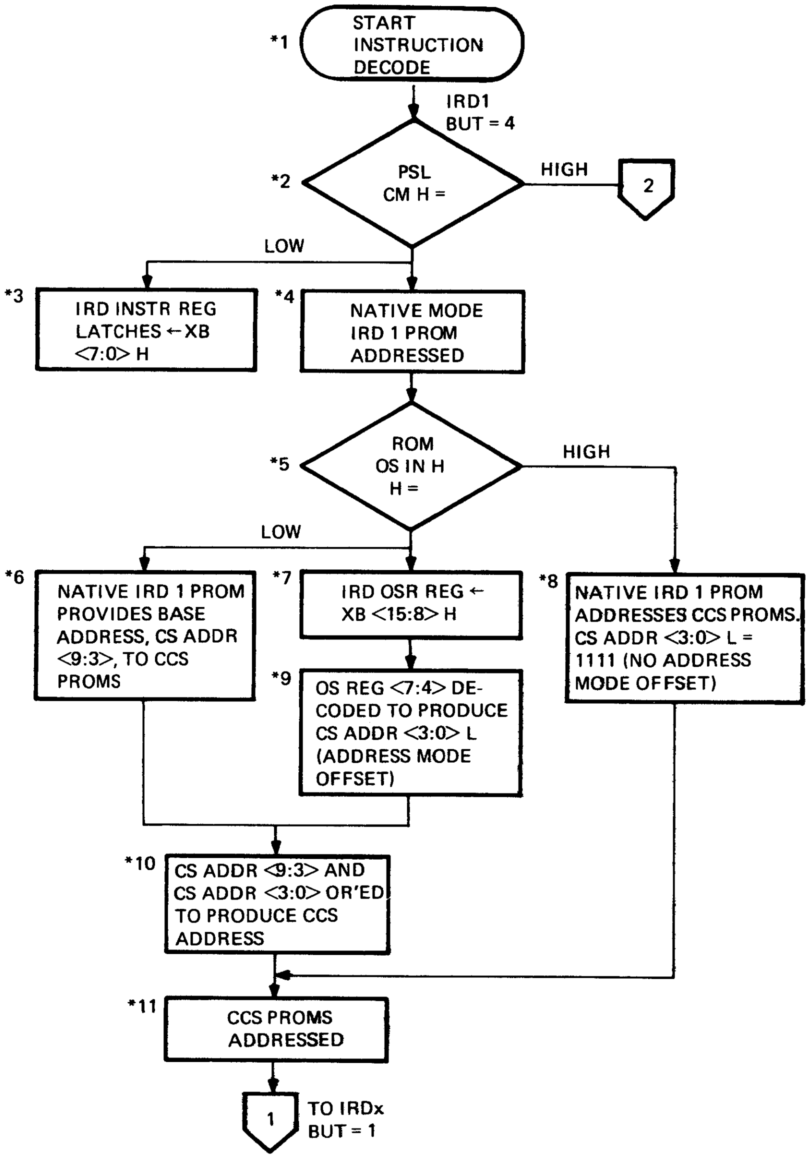

*1
START INSTRUCTION DECODE
IRD1 BUT = 4
*2
PSL CM H =
HIGH
2
LOW
*3
IRD INSTR REG LATCHES ←XB <7:0> H
*4
NATIVE MODE IRD 1 PROM ADDRESSED
*5
ROM OS IN H H =
HIGH
LOW
*6
NATIVE IRD 1 PROM PROVIDES BASE ADDRESS, CS ADDR <9:3>, TO CCS PROMS
*7
IRD OSR REG ← XB <15:8> H
*8
NATIVE IRD 1 PROM ADDRESSES CCS PROMS. CS ADDR <3:0> L = 1111 (NO ADDRESS MODE OFFSET)
*9
OS REG <7:4> DECODED TO PRODUCE CS ADDR <3:0> L (ADDRESS MODE OFFSET)
*10
CS ADDR <9:3> AND CS ADDR <3:0> OR'ED TO PRODUCE CCS ADDRESS
*11
CCS PROMS ADDRESSED
1
TO IRDx BUT = 1
TK5774

Figure 2-43 Instruction Decode Flows (Sheet 1 of 4)

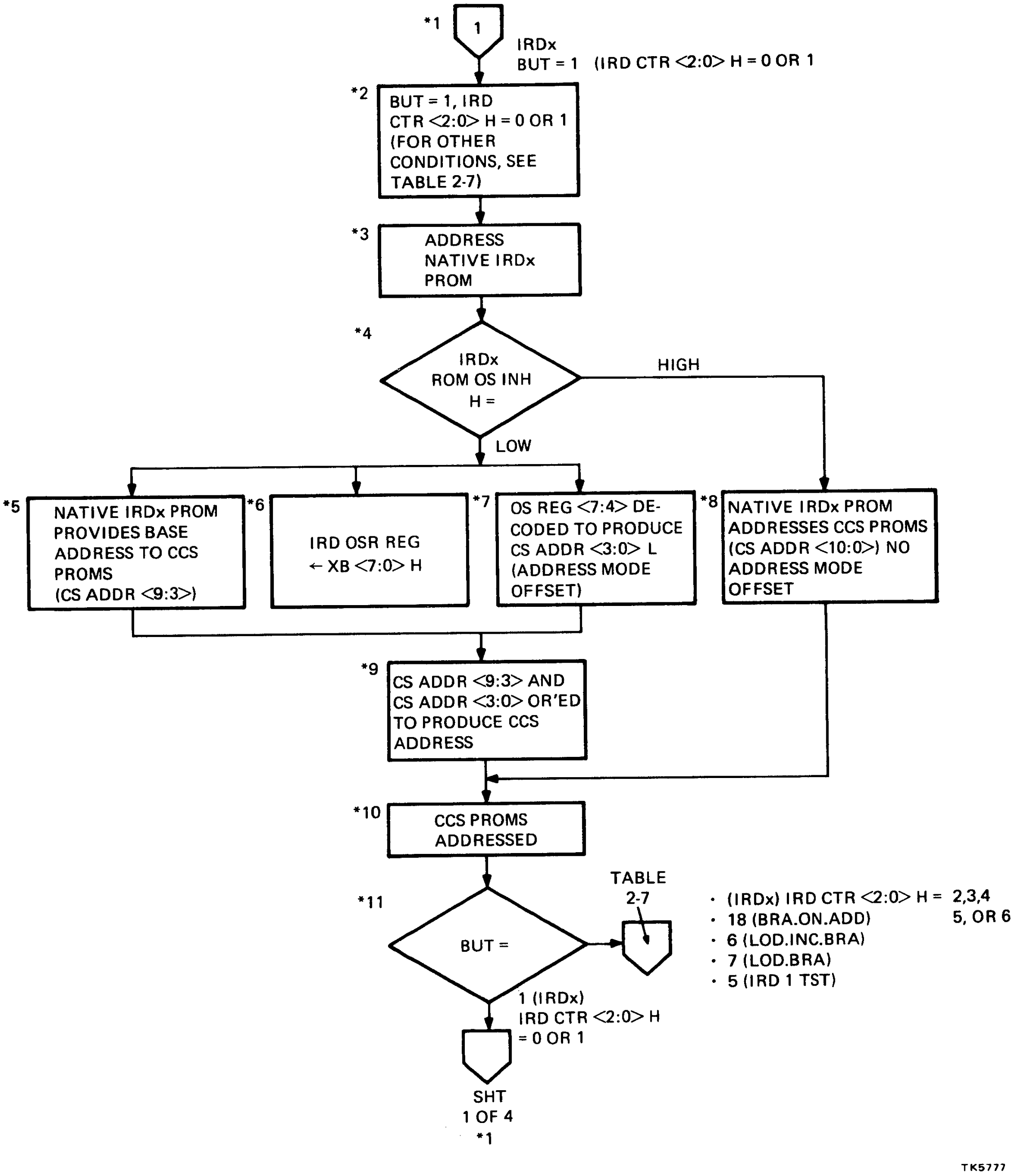

*1
1
IRDx
BUT = 1 (IRD CTR <2:0> H = 0 OR 1
*2
BUT = 1, IRD CTR <2:0> H = 0 OR 1 (FOR OTHER CONDITIONS, SEE TABLE 2-7)
*3
ADDRESS NATIVE IRDx PROM
*4
IRDx ROM OS INH H =
HIGH
LOW
*5
NATIVE IRDx PROM PROVIDES BASE ADDRESS TO CCS PROMS (CS ADDR <9:3>)
*6
IRD OSR REG ← XB <7:0> H
*7
OS REG <7:4> DE-CODED TO PRODUCE CS ADDR <3:0> L (ADDRESS MODE OFFSET)
*8
NATIVE IRDx PROM ADDRESSES CCS PROMS (CS ADDR <10:0>) NO ADDRESS MODE OFFSET
*9
CS ADDR <9:3> AND CS ADDR <3:0> OR'ED TO PRODUCE CCS ADDRESS
*10
CCS PROMS ADDRESSED
*11
BUT =
TABLE 2-7
· (IRDx) IRD CTR <2:0> H = 2,3,4 5, OR 6
· 18 (BRA.ON.ADD)
· 6 (LOD.INC.BRA)
· 7 (LOD.BRA)
· 5 (IRD 1 TST)
1 (IRDx) IRD CTR <2:0> H = 0 OR 1
SHT 1 OF 4
*1
TK5777

Figure 2-43 Instruction Decode Flows (Sheet 2 of 4)

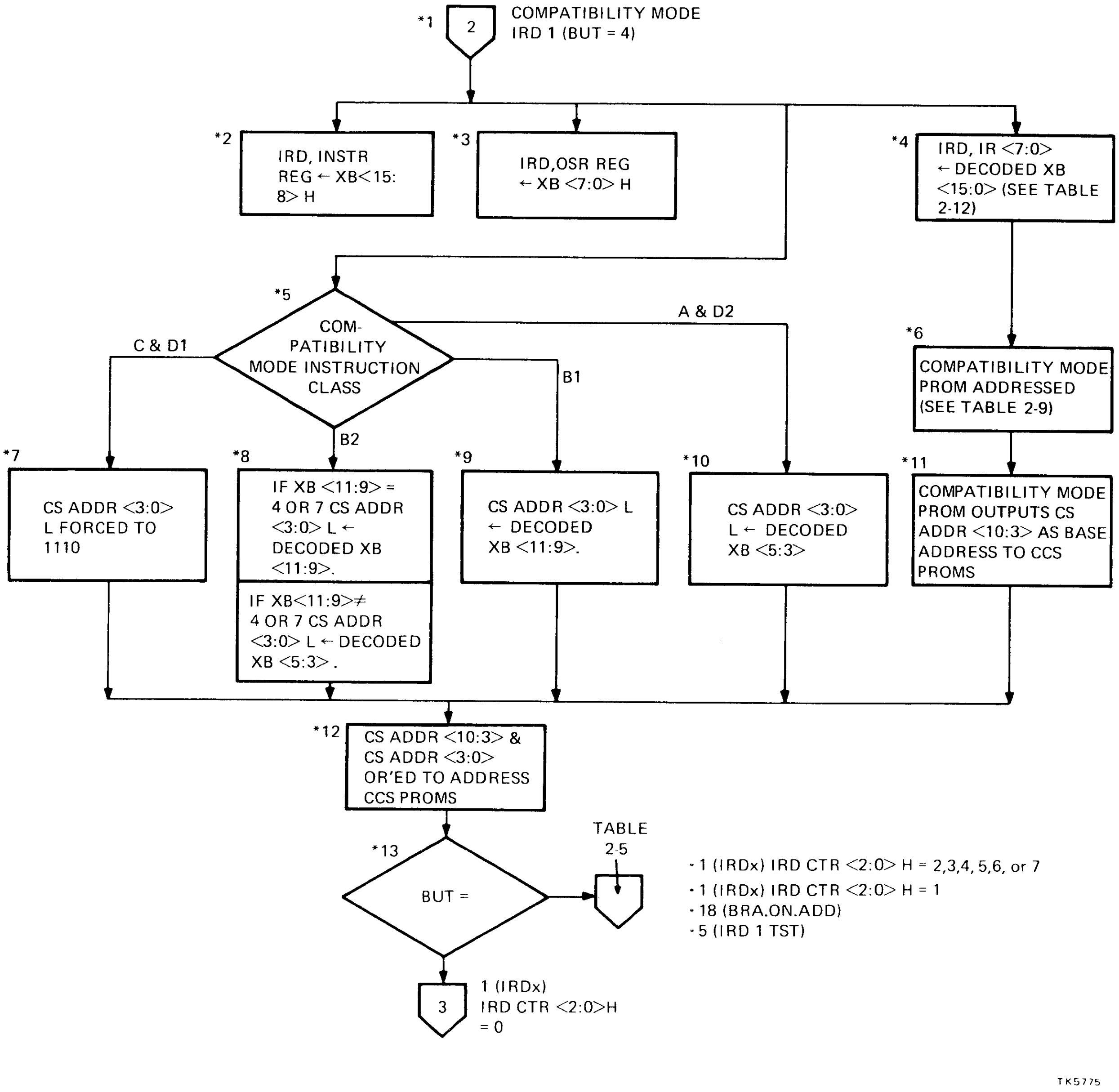

*1
2
COMPATIBILITY MODE
IRD 1 (BUT = 4)
*2
IRD, INSTR REG ← XB<15:8> H
*3
IRD,OSR REG ← XB <7:0> H
*4
IRD, IR <7:0> ← DECODED XB <15:0> (SEE TABLE 2-12)
*5
COMPATIBILITY MODE INSTRUCTION CLASS
C & D1
A & D2
B1
B2
*6
COMPATIBILITY MODE PROM ADDRESSED (SEE TABLE 2-9)
*7
CS ADDR <3:0> L FORCED TO 1110
*8
IF XB <11:9> = 4 OR 7 CS ADDR <3:0> L ← DECODED XB <11:9>.
IF XB<11:9>≠ 4 OR 7 CS ADDR <3:0> L ← DECODED XB <5:3>.
*9
CS ADDR <3:0> L ← DECODED XB <11:9>.
*10
CS ADDR <3:0> L ← DECODED XB <5:3>
*11
COMPATIBILITY MODE PROM OUTPUTS CS ADDR <10:3> AS BASE ADDRESS TO CCS PROMS
*12
CS ADDR <10:3> & CS ADDR <3:0> OR'ED TO ADDRESS CCS PROMS
*13
BUT =
TABLE 2-5
· 1 (IRDx) IRD CTR <2:0> H = 2,3,4, 5,6, or 7
· 1 (IRDx) IRD CTR <2:0> H = 1
· 18 (BRA.ON.ADD)
· 5 (IRD 1 TST)
3
1 (IRDx) IRD CTR <2:0>H = 0
TK5775

Figure 2-43 Instruction Decode Flows (Sheet 3 of 4)

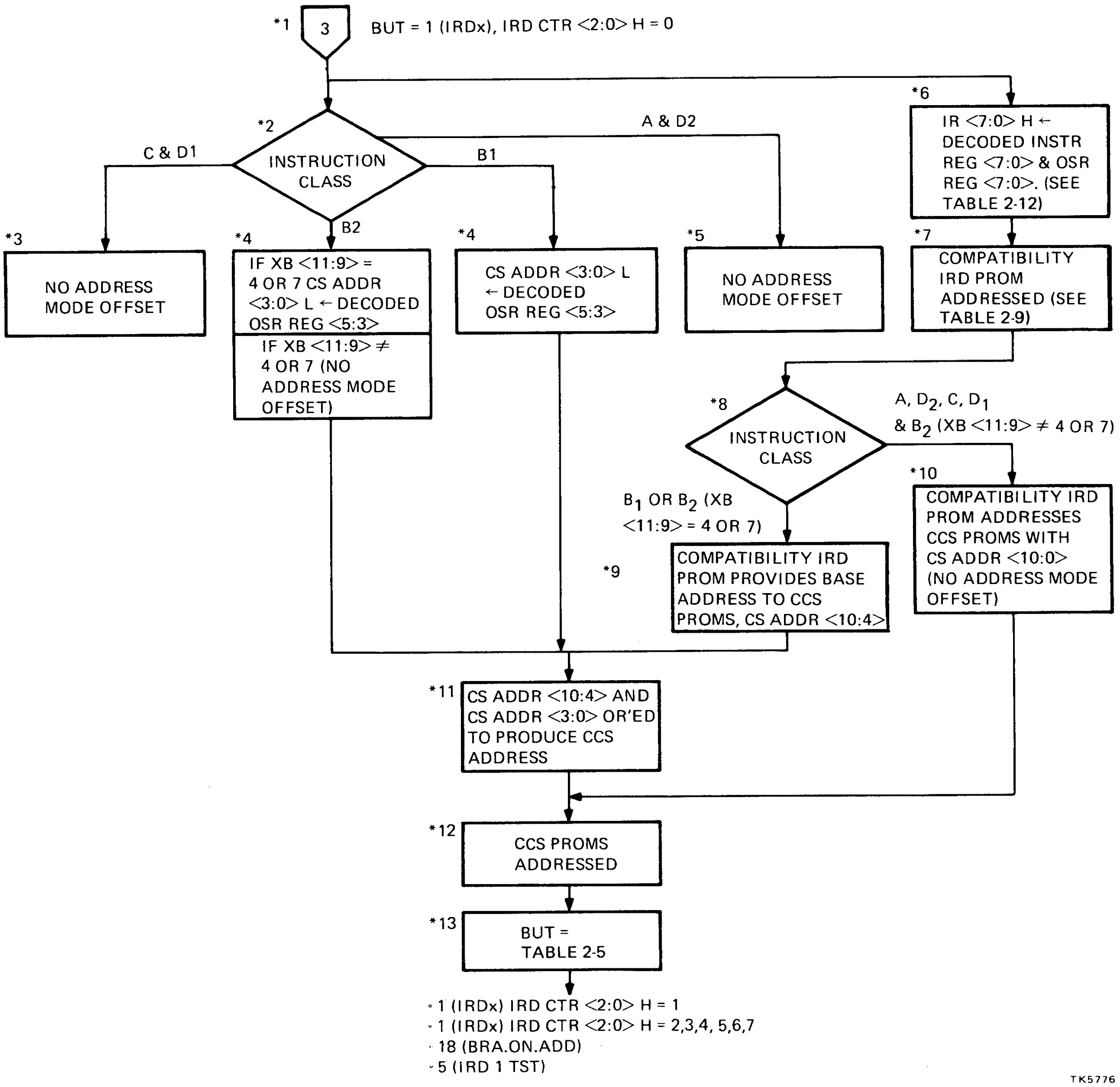

*1
3
BUT = 1 (IRDx), IRD CTR <2:0> H = 0
*2
INSTRUCTION CLASS
C & D1
B1
A & D2
B2
*3
NO ADDRESS MODE OFFSET
*4
IF XB <11:9> = 4 OR 7 CS ADDR <3:0> L ← DECODED OSR REG <5:3>
IF XB <11:9> ≠ 4 OR 7 (NO ADDRESS MODE OFFSET)
*4
CS ADDR <3:0> L ← DECODED OSR REG <5:3>
*5
NO ADDRESS MODE OFFSET
*6
IR <7:0> H ← DECODED INSTR REG <7:0> & OSR REG <7:0>. (SEE TABLE 2-12)
*7
COMPATIBILITY IRD PROM ADDRESSED (SEE TABLE 2-9)
*8
INSTRUCTION CLASS
A, D2, C, D1 & B2 (XB <11:9> ≠ 4 OR 7)
B1 OR B2 (XB <11:9> = 4 OR 7)
*9
COMPATIBILITY IRD PROM PROVIDES BASE ADDRESS TO CCS PROMS, CS ADDR <10:4>
*10
COMPATIBILITY IRD PROM ADDRESSES CCS PROMS WITH CS ADDR <10:0> (NO ADDRESS MODE OFFSET)
*11
CS ADDR <10:4> AND CS ADDR <3:0> OR'ED TO PRODUCE CCS ADDRESS
*12
CCS PROMS ADDRESSED
*13
BUT = TABLE 2-5
• 1 (IRDx) IRD CTR <2:0> H = 1
• 1 (IRDx) IRD CTR <2:0> H = 2,3,4, 5,6,7
• 18 (BRA.ON.ADD)
• 5 (IRD 1 TST)
TK5776

Figure 2-43 Instruction Decode Flows (Sheet 4 of 4)

The CCS ROMs receive the complete microaddress of the microinstruction needed. See below.

Base address supplied by native IRD 1 PROM CS ADDR <9:3>
Address mode offset, output by IRD chip CS ADDR <3:0>

```
100:
OS.RED:
     ;0000-----------------------          ; RN REGISTER MODE
     FPA_Q_M[MDR] MDR_R[GPR.R],            ; PLACE OPERAND
                                             (GPR(RNUM)) IN MDR
     CLOBBER MTEMP0 DEF,IRDX [1]           ; SAVE MDR IN Q BEFORE
                                             CLOBBERING IT
```

After execution of this microinstruction an IRDx time is specified. The BUT field of this microinstruction = 1 (IRDx). (See Figure 2-43, Sheet 2, *1.)

Shown above is a microcode excerpt taken from microcode listing Rev 5.01, Page 1045. The complete microaddress of this instruction is 100. The functionality of this microword is to place the contents of a general processor registor (GPR), in this case R1, into the memory data register (MDR). The BUT field of this microword equals 1 (IRDx). The next operation to occur is an IRDx instruction decode. (See Figure 2-43, Sheet 2, *1.)

IRDx:

The microword BUT field equals 1 (IRDx)

PSL CM H = L and PSL CM L = H indicating native mode operation.

In the IRD chip the contents of INSTR REG <7:0> are sourced to the IR <7:0> H output to provide a portion of the address to the native mode IRDx ROMs as follows.(See Figure 2-43, Sheet 2, *3.)

Signals	**ROM ADDR**
IR <7:0> H = 0D0	<10:3>
IRD CTR 0 H = L	<2>
FPA PRESENT L = H	<1>
REG MODE H = H	<0>

NOTE
The IRDx PROM is enabled by ENABLE IRD ROM H = H, IRD 1 L = H, and PSL CM L = H.

The output of the native mode IRDx ROM becomes CS ADDR <10:0> to the CCS ROMs for IL.MOV.B.W.L.REG. This is the next microinstruction to be executed (see below). ROM OS INH H is low (see Figure 2-43, Sheet 2, *4), allowing an operand specifier, R2, to be loaded into the OSR.

```
IL.MOV.B.W.L.REG:                          ;
IL.MOVA.B.W.K.REG:                         ;
     ;--------------------------------     ;
     R[DST.R].SIZ_M[MDR],SIZE[IDEP],       ;
     WRITE NOTREG,CCOP2,IRD1               ;
```

After execution of this microinstruction an IRD 1 time is specified. The BUT field of this microword is 4 (IRD 1). (See Figure 2-43, Sheet 1, *1)

The microcode excerpt shown above is taken from microcode listing Rev 5.01, Page 195. The microaddress of this instruction is OOEE. This microword takes the data stored in MDR and places it in a destination (GPR), in this case R2. The BUT field of this microword specifies IRD 1 (BUT = 4). Execution of the MOVL R1, R2 instruction is now complete. The instruction decode logic is now prepared to decode the next macroinstruction.

2.4.7.2 NOP Instruction Decode – (See Figure 2-43, Sheet 1, *1.) Decode of a native mode (VAX-11) NOP macroinstruction is performed as follows.

IRD 1:

The last microword BUT field specified IRD 1 (BUT = 4) (Figure 2-43, Sheet 1, *1).

PSL CM H = L and PSL CM L = H (Figure 2-43, Sheet 1, *2).

The IRD chip INSTR REG is loaded with the opcode on XB <7:0> H (NOP = 001) (Figure 2-43, Sheet 1, *3).

The native mode IRD 1 PROM is addressed as follows.

Signals	**ROM ADDR**
XBUF <7:0> H = 001_{16}	<9:2>
FPA PRESENT L = H	<1>
PSL FPD H = L	<0>

NOTE
The IRD 1 PROM is enabled by ENABLE IRD ROM H = H, PSL CM L = H, and IRD 1 L = L.

The native mode IRD 1 PROM outputs an address CS ADDR <9:3> to the CCS PROMs for MS.NOP. Since the NOP instruction has no operand specifier associated with it, the IRD 1 PROM also outputs ROM OS INH H = H. Because ROM OS INH = H, the IRD chip CS ADDR <3:0> L output goes open. CS ADDR <3:0> L is pulled high by resistors on DPM 14. The CCS PROMs then receive a microaddress of 300. This address contains microcode that performs the operation shown below.

```
MS.NOP:  ;------------------;
         PC_M[PC]-XLITO[1]    ;
         NEXT/GL.NOP.IRD1     ;
```

Here the NEXT field contains an address for the next microword to be used. At this microaddress, the BUT field equals 4 (IRD 1). All the other fields of this microword specify a default condition.

The microcode excerpt shown above was taken from microcode listing Rev 5.01, Page 485. The microaddress of this instruction is 300. The only function performed here is to subtract 1 from the PC. This is necessary because during IRD 1 time the PC is incremented by 2. Since the NOP instruction has no operand specifier, the next byte must be a new opcode, so the PC must be backed up by 1. The NEXT field of this microinstruction gives the microaddress of the next microinstruction to be executed. The next microinstruction BUT field = 4 (IRD 1) and all other fields in this instruction are in default condition. Decode of the NOP instruction is completed.

2.4.8 Instruction Decode Timing

Figures 2-44 and 2-45 show instruction decode timing for native mode and compatibility mode instruction decode.

2.4.8.1 Native Mode Instruction Decode Timing – Figure 2-44 shows native mode decode timing. The timing shown relates to a MOVL R1, (R2) macroinstruction. This instruction moves a longword from R1 to the memory address pointed to by R2.

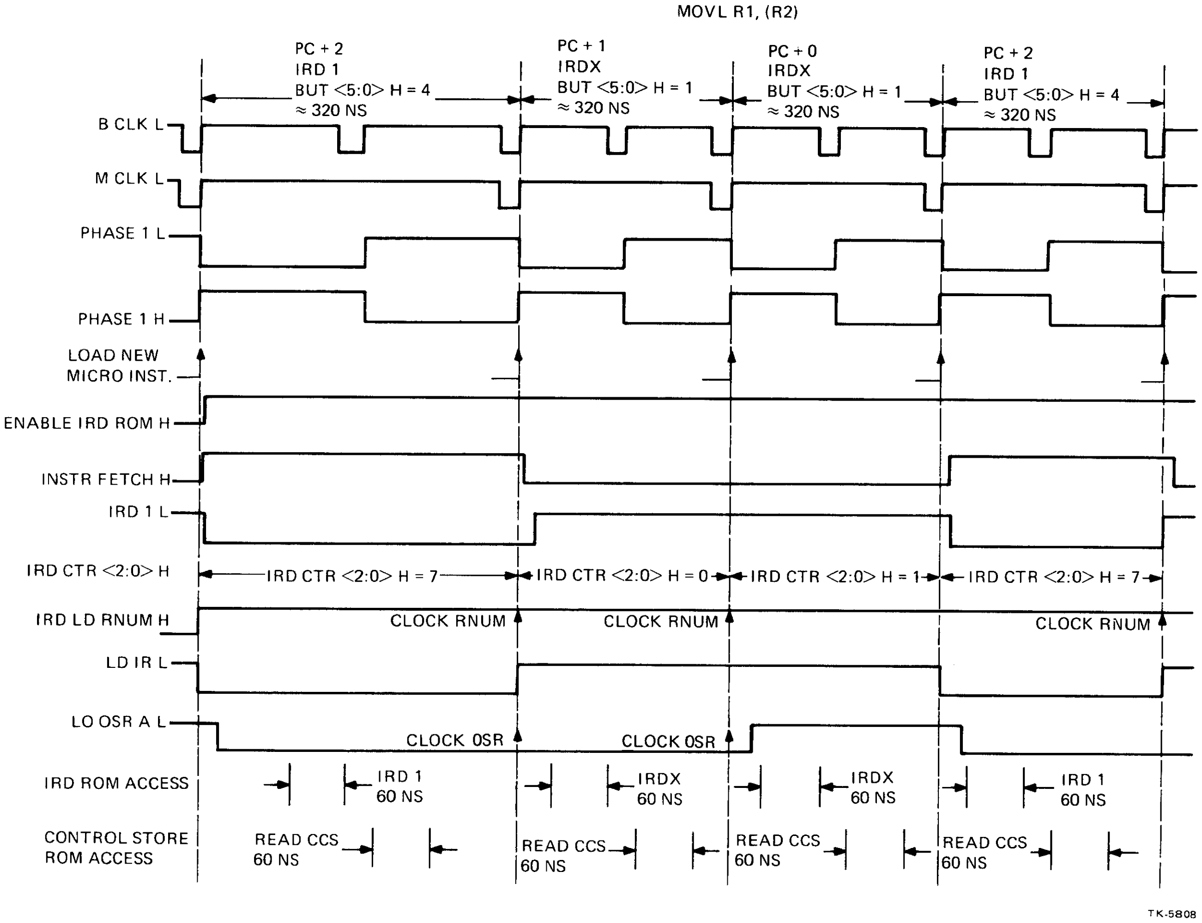


Figure 2-44 Native Mode Instruction Decode Timing

2.4.8.2 Compatibility Mode Instruction Decode Timing – Figure 2-45 shows compatibility mode decode timing. The timing shown is for a MOV,2 PDP-11 macroinstruction. Some of the basic differences between this timing chart and the native mode timing chart are as follows.

1. The IRD 1 time for compatibility mode is lengthened to 480 ns by extending the M CLK L cycle by 160 ns. This extension is necessary to allow PDP-11 macroinstructions to be encoded inside the IRD chip. IRD outputs IR <7:0> H are then used to address the compatibility IRD ROM. For native mode instructions, the native IRD 1 ROM is addressed directly by XBUF <7:0> H – no encoding is necessary. The compatibility ROM address is delayed approximately 60 ns.

2. The OSR REG need be loaded only once even though the instruction shown (MOV,2) is a two-operand instruction. The entire PDP-11 instruction is loaded into the INSTR REG and OSR REG at IRD one time.

3. For the third microcycle the BUT <5:0> H field = 2E (SPASTA). Here the compatibility IRD ROM is not addressed. The next microaddress (to CCS) is specified by the BUT field and NEXT field of the last microinstruction.

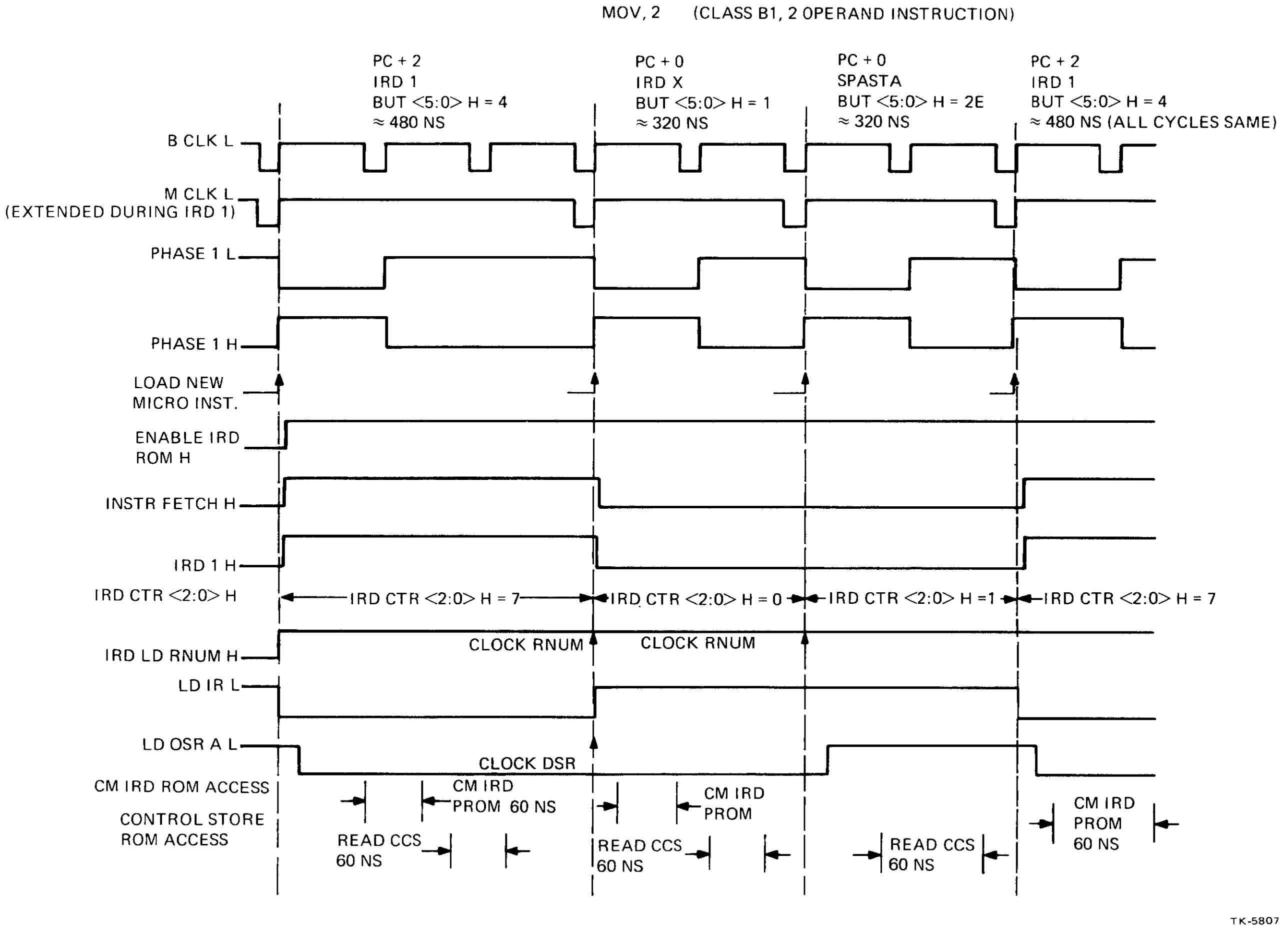


Figure 2-45 Compatibility Mode Instruction Decode Timing

2.5 MEMORY INTERCONNECT (MIC) MODULE

The memory interconnect module (MIC) performs the following functions for the processor.

- As CPU interface to the CMI, the MIC transmits the CMI address for access to memory or I/O, then receives or transmits CMI data.
- Performs instruction prefetch, maintains 2 longwords of I-Stream data from memory in the execution buffer (XB).
- Translation buffer stores page table entries for virtual to physical address translation.
- Cache memory stores most recent or frequently accessed data.
- Monitors CMI writes to main memory by other subsystems to invalidate cache.
- Generates stall to CPU clocks for microtraps and wait conditions.
- Makes access checks under microsequencer control. Generates microtraps on access violations, unaligned memory reference, error detection, etc.
- Decodes CPU-generated addresses to the Unibus.
- Read-lock timeout circuitry.

The MIC module functions in these two basic fashions.

1. It performs microcoded orders.
2. It monitors and generates nonmicrocoded functions:

   Prefetches I-Stream data.
   Responds to microtrap conditions.

Microcoded Functions – The MIC performs microcoded functions under direct control of the bus function, MSRC, and WCTRL fields of the CPU control store (CCS). Some examples of microcoded functions are:

1. Read or write to memory or to an I/O device.
2. Source data from the MDR to the MBus, WBus data to the MDR.
3. Probe translation buffer for access violations.
4. Read or Write MIC status/control registers.

Nonmicrocoded Functions – Some nonmicrocoded functions are directly related to microcode operations. With memory management enabled (the MME bit is set), the MIC monitors microcoded memory references for TB hits or misses and for access violations, all independent of the microcode. The access control violation chip (ACV) and microtrap chip (UTR) work interdependently to monitor those conditions whenever memory management is enabled. When an improper condition is detected, a microtrap is raised to the microcode. A microaddress is generated to place the machine in the routine that services the condition. The ACV also monitors parity conditions for the CCS.

I-Stream data is fetched from memory by the processor independent of the microcode. The MIC first loads the execution buffers with initial data. This is the flushing of the execution buffer (XB) that takes place whenever the PC is loaded with a new address. The new PC contents are used to retrieve two longwords from memory (or cache) and to store them in the XB registers (XB0 and XB1). As I-Stream data is used during execution of a machine instruction (macroinstruction), it is monitored by the prefetch control chip (PRK). The PRK determines when an XB is empty and must be refilled with another longword from memory. The MIC accomplishes a flush or refill of the XB by performing a non-microcoded read to cache or main memory called the prefetch operation. Since a prefetch and a microcoded reference to memory use the same data path, they are performed at different times.

Data stored in memory that is not part of the I-Stream, but is requested by the operand, is not stored in the XBs and may not be in cache. To retrieve the data from memory requires more time than the microcode takes to execute, so the MIC generates a stall condition to the DPM. This holds off the microword from completing its function until the requested data is available. A stall is generated only when the microword needs data that is not available, not as an unconditional result of a fetch to memory. The stall condition, access violation checks, and cache and TB hits and misses are monitored during prefetch and microcoded memory references.

2.5.1 MIC Organization

Figure 2-46 is a basic diagram of the four main functional sections of MIC logic.

Address Control (ADD)
Memory Data Routing and Alignment (MDR)
Translation Buffer (TB)
Cache Memory

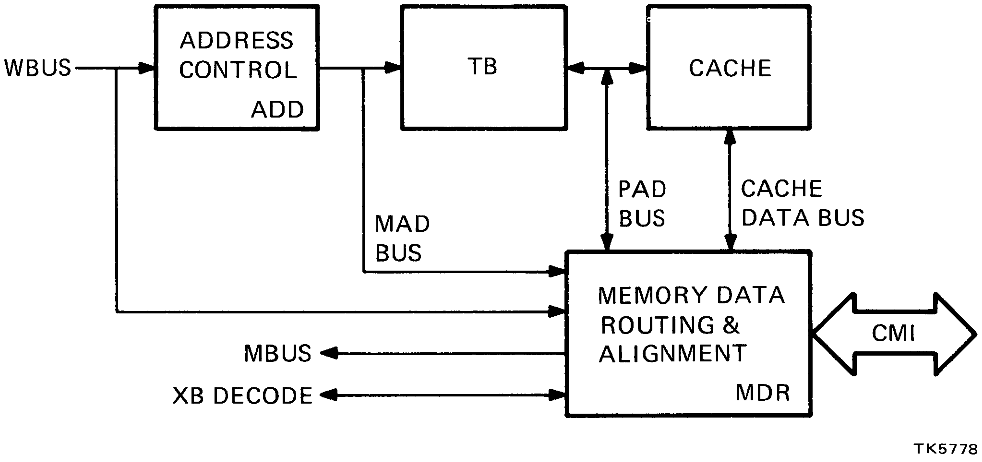


Figure 2-46 Basic MIC Diagram

2.5.1.1 Address Control – Memory address functions are performed by four 8-bit ADD chips. Each chip processes one byte of address information from the WBus. The ADD section contains program counter (PC), virtual address (VA), and associated registers, plus adder and multiplexer circuits for address manipulation. The memory address (MAD) lines direct physical addresses to the MDR, or virtual addresses to the TB, depending on the state of the memory management enable (MME) bit.

2.5.1.2 Memory Data Routing and Alignment – The MDR block is composed of eight 4-bit chip. Each chip processes one bit for each of the four bytes of CMI data. It performs data routing and alignment for transfers between the data path module and cache or main memory. It also contains the execution buffer (XB) that stores two longwords of I-Stream data prefetched from memory.

2.5.1.3 Translation Buffer – The TB consists of two sets of RAMs with 256 index locations each for 2-way set association. It stores page table entries (PTEs) for virtual to physical address translation in conjunction with microcode translation routines. TB data is divided into two fields. The address field stores virtual translation address (tag) information. The data field contains the translated physical page frame number (PFN) for each PTE with associated page protection bits. Physical address information is transferred between the MDR and TB or cache on the physical address (PAD) lines.

2.5.1.4 Cache Memory – Cache consists of 1,024 (1K) longword locations for direct mapping of 4K bytes. Cache data is longword-aligned with the CMI; rotation for data path alignment is accomplished in the MDR. Writes to memory on the CMI not generated by the CPU are checked for hits on the corresponding cache location. A hit causes invalidation of the location.

2.5.2 Address Control (ADD) Block

The memory address logic (Figure 2-47) supplies physical address information to the MDR (memory management disabled), or virtual address to the TB (memory management enabled).

Address manipulation takes place for the following functions.

CPU references (via microcode) to all CMI address space
Prefetch of processor references to main memory (or cache) for I-Stream data
Normal program counter increments
Branch Offsets for I-Stream lengths or special displacement functions.

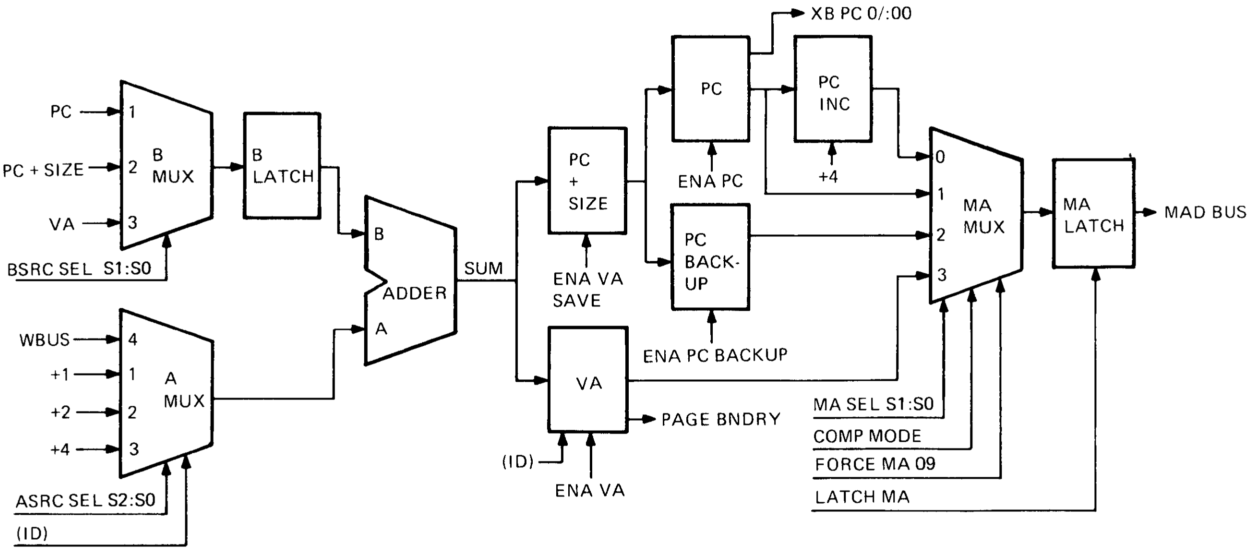


Figure 2-47 Address Control (ADD)

2.5.2.1 MA Latch and Multiplexer – Memory address (MAD) lines are driven from the tri-state drivers of the MA latch which is transparent to MA MUX outputs. The LATCH MA level is asserted by the PRK to close the latch during a memory reference. This holds the address until it is tested for microtraps and clocked to the MDR. It also allows the VA register to be updated in the same microstep that specified the memory function.

The latch is also closed on a microtrap to capture the address for microcode reference. In the event the bus function is a memory cycle (including bus grant) or an access probe, the MA may contain a prefetch address that must be saved.

The FORCE MA 09 level is driven by WXTRL code 29 (hex) from the ACV. It is used to facilitate TB addressing during invalidation microroutines.

Compatability mode (COMP MODE) from the ADK forces MA MUX bits <31:16> to zeros except for these bus functions.

Read, no microtrap
Write, no microtrap
Write longword, no microtrap
Read physical address
Write physical address
TB access probe

MA SEL <S1:S0> bits are driven by the PRK as shown in Table 2-23 to select MA MUX inputs.

Table 2-23 MA Multiplexer Input Select

MA SEL S1	 S0	MA Inputs Select Register
0	0	PC INC (PC Increment)
0	1	PC BACKUP
1	0	PC (Program Counter)
1	1	VA (Virtual Address)

2.5.2.2 ADD Registers and Adder

Program Counter (PC) – The PC provides addresses for instruction (I-Stream) fetches from memory. It is incremented by +2 during IRD 1 fetch (opcode and first OSR). It is then incremented by the I-size value (+1, +2, +4) determined by the OSR (operand specifier) as the I-Stream is used.

New addresses are entered into the PC directly from the WBus. This flushes the XB by prefetching 2 longwords of I-Stream information at the new address. The PC may also receive branch offsets added to its contents from the WBus. The ENA PC level from the PRK enables the PC to clock the updated address.

XB PC <01:00> bits are sent to the MDR to determine byte offset (rotation) gating from the XB. They are used by the PRK to determine MA MUX steering by the MA SEL <S1:S0> bits. Used with the IRD 1 and LD OSR signals, they also determine which XB0/XB1 bytes are filled from cache or main memory as I-Stream information is used by the CPU.

PC + Size and B Latch – The transparent B latch closes on the negative transition of B CLK L. This holds, on the B input to the adder, the prior value from the input register whose outputs change as it clocks new or incremented data. The SUM output from the adder is passed through the PC + Size latch which is transparent while B CLK L is low and ENA VA SAVE is true from the PRK. The PC receives PC + Size contents if I-Stream data is used during the microstep. The PC and PC + Size registers are closed to hold the new address on the positive transition of B CLK L. The B latch is opened, and is again transparent, on the positive transition of B CLK L.

PC BACKUP – The PC backup register receives PC + 2, the address following the opcode/first operand specifier (OSR) address during an IRD 1 microstep. The PC backup register retains its value for the duration of the macroinstruction. A recoverable trap or fault that occurs during macroinstruction execution may require the processor to back up and retry the instruction, starting at the address specified by the contents of the PC backup register. The register can be sourced onto the MBus by direction of the MSRC microfield.

PC Increment – The PC INC is an increment register that constantly reflects the PC register value incremented by four. It is used during a prefetch operation to retrieve a longword of I-Stream data at the next longword address.

Virtual Address (VA) – The VA register, with memory management disabled, provides physical address information to the MDR. With memory management enabled, it supplies virtual address to the TB for translation to physical address. Under direction of the microcode from the WCTRL field, the VA receives addresses from the WBus, or it may be incremented by a longword value (+4) independent of the WBus. It may also receive PC + Size register contents plus offset values from the WBus or the I-Stream. It is opened to receive address information by the ENA VA level from the ADK.

2.5.2.3 ADD Chip Identify (ID) – The ID pin on the ADD chip for address byte 0 (bits <07:00>) is grounded, and is connected to +3 on the other chips. The ground on byte 0 enables I-Size constants that are added to the low-order address byte when selected on the A MUX. They are disabled for bytes <3:1>.

The PAGE BNDRY level from address byte <1:0> chips are connected together. This allows the level to go high when VA register bits <8:3> are all ones. Bits <7:3> of the VA register, (byte 0 chip) when all ones, allow the PAGE BNDRY level to go high. The level is asserted high when bit 0 of the byte 1 chip (VA address bit <8>) is also a one.

2.5.2.4 Adder Inputs – The adder sections of each ADD chip, with carry look-ahead circuitry between the chips, make up a full 32-bit adder stage.

B MUX inputs are selected as shown in Table 2-24 by the BSRC SEL codes from the ADK.

A MUX inputs are selected as in Table 2-25 by the ASRC SEL codes, selected by MIC gating that monitors IRD 1, I-Size, LDOSR, and MSRC XB states.

Table 2-24 B Multiplexer Input Select

BSRC SEL S1	S0	B MUX Input Selection
0	0	Constant of 0
0	1	PC Register
1	0	PC + Size Register
1	1	VA Register

Table 2-25 A Multiplexer Input Select

ASRC SEL S2	 S1	 S0	A MUX Input Selection
0	0	0	Constant of 0
0	0	1	I-Size = Byte (+1)
0	1	0	I-Size = Word (+2)
0	1	1	I-Size = Longword (+4)
1	X	X	WBus address or offset

2.5.3 Memory Data Routing and Alignment (MDR)

The MDR block, Figure 2-48, performs all data routing and alignment functions for the CPU.

Alignment of data between the CMI or cache and the data path section of the CPU. A basic description of the CMI and CMI transfer formats is found in Paragraph 2.5.8.

CMI latch is transparent to CMI data. It closes to capture an address generated by another subsystem (snapshot CMI) that is performing a write to memory in order to invalidate the corresponding cache location on a cache hit.

Execution buffer (XB) maintains eight bytes of I-Stream information for the prefetch function.

Many functions are directed by combinations of the WCTRL, MSRC, and bus function fields of the microprocessor in conjunction with decisions made by the control chips. Basic examples of memory transfer functions are provided to illustrate address and data routing and alignment through the MDR block.

The MDR block contains the following registers and multiplexers.

CMI Address register holds physical longword address.

Write Data Register (WDR) holds data for a write to memory and/or cache.

Memory Data Register (MDR) receives read data from cache or memory.

CMI Latch closes for the snapshot CMI function to hold an address transmitted by another subsystem.

DBus Rotator (DBUS ROT) aligns DBus data clocked to the MDR or WDR.

A MUX sources physical addresses to the bidirectional physical address (PAD) bus and to the CMI address register.

M MUX sources data/address information to the MBus.

Execution Buffer (XB) stores eight bytes of I-Stream information (four bytes each, XB0 and XB1).

XB Rotator (XB ROT) rotates XB data for alignment to the XB decode bus or MBus.

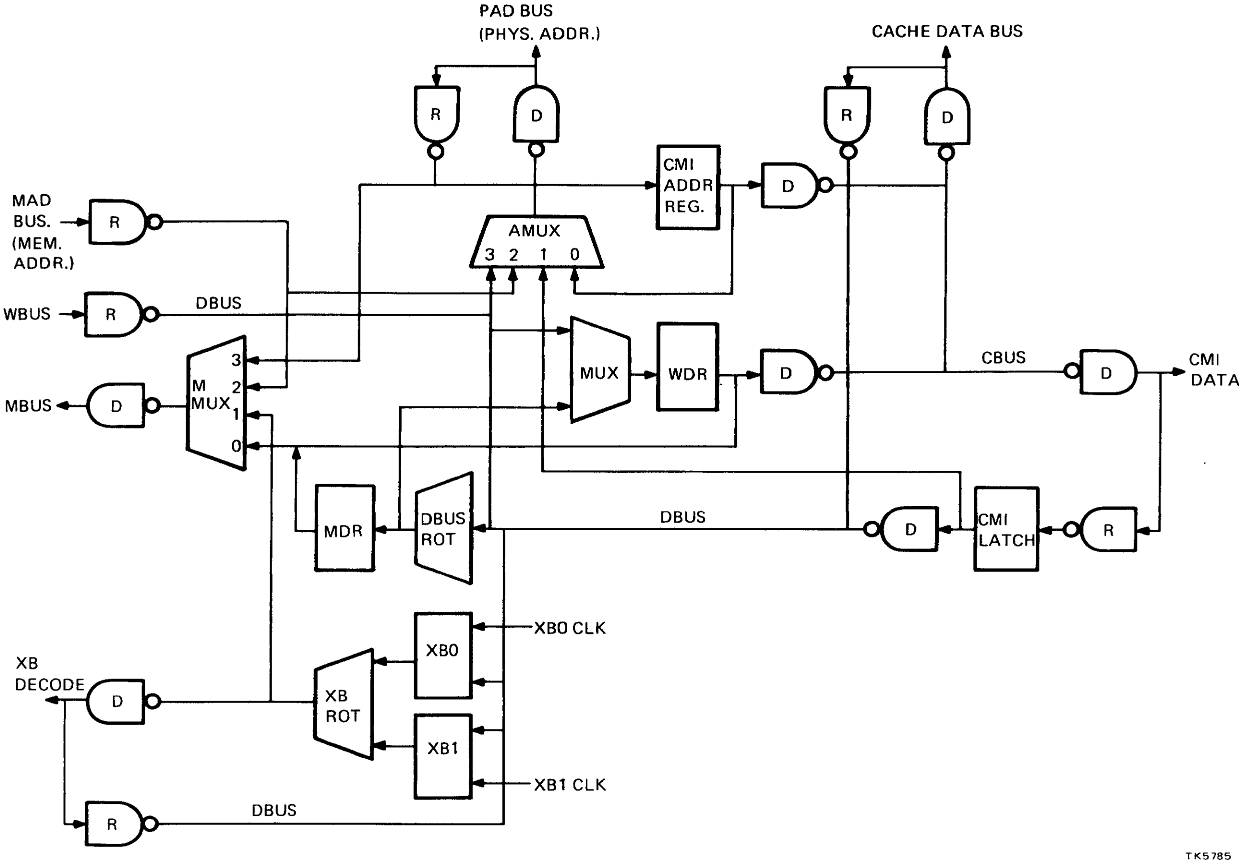


Figure 2-48 Memory Data Routing and Alignment (MDR)

2.5.3.1 MDR Address Functions

CMI Address Register – The content of this register is the physical longword address transmitted by the CPU to access CMI address space. It is continually sourced to the CBus in preparation for a CMI address cycle. During the CMI address cycle, CMI data drivers for bytes <2:0> are enabled from the MDR. Byte 3, with byte mask and function code bits, is enabled by the CMK which also asserts DBBZ. Byte 3 drivers from the MDR block are not enabled unless the WDR (write data register) is sourced to the CBus to source data onto the CMI for a write cycle.

Memory Management – With memory management disabled, physical address bits <23:02> from the MAD (memory address) lines of the ADD block are all sourced to the PAD (physical address) bus drivers from the A MUX. The PAD bus provides cache addressing if cache is enabled. The CMI address register is enabled by ADD REG ENA from the CMK to clock all 22 bits from the PAD receivers, since access to main memory is required for a write to memory, a cache miss on a read, or if cache is disabled.

With memory management enabled, only PAD drivers for virtual address bits <08:02> from the MAD lines are enabled. MAD bits <31:09> of a virtual address directly access the TB from the ADD block.

On a TB hit, the translated physical PFN is driven to PAD lines <23:09> from the TB. With byte address bits <08:02> from the A MUX, the complete physical longword address is asserted on the PAD lines to address cache. The physical address is also clocked to the CMI address register from the PAD receivers in the event that access to main memory is necessary.

MA latch contents on the MAD bus from the ADD block can also be sourced to the MBus from the M MUX by the microcode.

Physical Address (PAD) Bus – In addition to supplying the physical address for cache access, the PAD bus is used by the microcode to read or write address translation information to the TB or make access checks. Read data bits <23:09> from the TB are sourced to the MBus via the M MUX. MBus bits <31:24> are not used. TB write data from the WBus is sourced to the DBus for the A MUX.

The DBus sources data from the CMI on a TB miss. When a microtrap retrieves a PTE from main memory, the PTE is sourced onto the PAD bus from the WBus to check access privileges before being written to the TB.

MDR Chip Identify (ID) – The previous subsection illustrates MDR use of the physical address bus. All PAD drivers are normally enabled to drive physical address information. All are disabled when the receivers are sourced to the MBus.

When driving virtual address (memory management enabled), the drivers for byte 0 remain enabled, while the drivers for bytes 1 and 2 are disabled to allow TB contents to be asserted on the PAD lines. The ID pin for each MDR chip is grounded except for the chip that drives bit <08>. This allows the driver for that bit to remain enabled with byte 0 (bits <7:2>) to assert the VA byte address field onto the PAD lines.

Address Multiplexer (A MUX) – The A MUX sources all physical address information to the PAD bus and CMI address register. Its inputs are selected by the A MUX SEL <S1:S0> levels from the ADK as shown in Table 2-26, directed by the bus function.

Table 2-26 A Multiplexer Source Select

A MUX SEL		PAD Bus
S1	**S0**	**Driver Source**
0	0	CMI Address Register
0	1	CMI Data Latch
1	0	MAD Bus
1	1	DBUS

2.5.3.2 MDR Data Transfers

Write Data Register (WDR) – Write data from the WBus is sourced to the DBus. DBus rotator outputs are clocked to the WDR from the WDR MUX. The WDR is sourced to the CBus to write data to memory and cache. It may also be sourced to the MBus for storage on a microtrap.

For a write to memory, CMI address register contents are sourced to the CBus for transmission on the CMI DATA lines. CMI address longword bytes <2:0> are asserted for one B CLK cycle while DBBZ and byte 3 (byte mask and function code) are asserted by the CMK. The CMI address register is also sourced to the PAD lines to select cache and check for a hit. The CMI address register, after one B CLK cycle, is deasserted from the CBus. Instead, the WDR is sourced to the CBus and driven onto CMI DATA <31:00>. For a cache hit, the data is also written to cache. Cache data bus drivers are always enabled except when read data is sourced from cache to the DBus.

DBus Write Data Alignment – For alignment of write data to cache and the CMI, the DBus rotator left-rotates DBus data sourced from the WBus to inputs of the WDR MUX shown as in Table 2-27. The CAK produces DBUS ROT <S1:S0> selection from a decode of the WCTRL and bus function fields.

Table 2-27 DBUS Left Rotate Select

DBUS ROT S1	S0	WDR Bytes <3:0> Receive DBUS Data Bytes				
0	0	3	2	1	0	(no rotation)
1	1	2	1	0	3	(1 byte)
1	0	1	0	3	2	(2 bytes)
0	1	0	3	2	1	(3 bytes)

All bytes are clocked to the WDR. Bytes valid for transfer to memory are determined by the CMK, which transmits the byte mask to the CMI. The CAK disables invalid bytes to cache via ENA BYTE <3:0> levels.

When high-order bytes of offset (rotated) write data cross the longword boundary, a CMI write to memory is generated for the valid low-order bytes. A write, second reference occurs to transmit the valid high-order bytes at the next longword address, unless the page boundary is crossed. In this case, the microcode performs a PTE access check on the next page table entry before the write is allowed to continue. In the case of a TB miss, the microcode sources the WDR to the MBus via M MUX gating for MTEMP storage, while the next PTE is retrieved from memory and an access check made.

CMI Data Latch/Snapshot CMI – The CMI latch is normally transparent to information received from the CMI. The CMK monitors the CMI for writes to memory by an I/O device. When a write function code is detected, the CMK asserts the snapshot CMI level. This closes the CMI latch, capturing the write address generated by the device. HOLD is asserted on the CMI by the CMK, while CMI latch contents are sourced from the A MUX to the PAD bus to address cache. On a cache hit, HOLD is held asserted, preventing additional CMI activity, until the cache location is invalidated.

Memory Data Register (MDR) – The MDR register receives cache or CMI read data from the DBus rotator. It may also receive WBus data to load internal registers that can be sourced to the MBus. It can be cleared by microcode, or can receive IR or OSR contents from the XB Decode bus.

DBus Read Data Alignment – The MDR receives all data as shown in Table 2-28 from the DBus rotator, which produces right byte rotation of CMI data from the DBus. Bytes <3:0> are all clocked to the MDR with DBUS ROT <S1:S0> directed by the CAK.

When high-order bytes of a memory read cross the longword boundary, an unaligned read microtrap occurs and a read, second reference is performed to retrieve data at the next longword address. A longword of data is returned from memory and rotated as for the first longword. Clocking for the MDR is enabled as shown in Table 2-29 to complete the word or longword of rotated data in the register.

Table 2-28 DBUS Right Rotate Select

DBUS ROT S1	 S0	MDR Bytes <3:0> Receive DBUS Data Bytes				
0	0	3	2	1	0	(no rotation)
0	1	0	3	2	1	(1 byte)
1	0	1	0	3	2	(2 bytes)
1	1	2	1	0	3	(3 bytes)

Table 2-29 MDR Clock Second Reference

DBUS ROT S1	 S0	MDR Byte Clocks Enabled			
0	1	3	X	X	X
1	0	3	2	X	X
1	1	3	2	1	X

NOTE: X indicates the byte clock is disabled in conjunction with the CLK SEL <S1:S0>/DBUS destination signals of Table 2-30.

DBus Data Select – Data is sourced to the DBus as shown in Table 2-30, selected by DBUS SEL <S1:S0> from the ADK. The receiving register is clocked on the positive transition of B CLK L as selected by CLK SEL <S1:S0> from the ADK.

MBus Multiplexer (M MUX) – The MBus drivers are enabled as shown in Table 2-31 by latched MSRC bits decoded in MIC module discrete logic (MBUS ENA). The MMUX SEL <S1:S0> inputs are respectively driven by MMUX SEL S1 from the PRK, and the latched MSRC 2 bit.

Table 2-30 DBUS Data Select

CLK SEL S1	 S0	Clock DBUS Destination	DBUS S1	SEL S0	Select DBUS Source
0	0	(None)	0	0	Cache Data Receivers
0	1	MDR	0	1	CMI Data Latch
1	0	XB Registers	1	0	WBUS
1	1	WDR	1	1	XB Decode Bus

Table 2-31 M Multiplexer Source Select

M MUX SEL S1	 S0	MBUS Source
0	0	MDR/WDR
0	1	XB Rotator
1	0	MAD Bus
1	1	PAD Bus

2.5.3.3 Execution Buffer (XB) – The execution buffer consists of two 4-byte first in–first out buffer registers, XB1 and XB0. They function under PRK control to maintain two longwords (8 bytes) of I-Stream data available for the CPU; two bytes to the XB decode bus, and four bytes to the MBus for the data paths.

Prefetch Function – Independent of the microsequencer, the PRK keeps track of machine cycles using such signals as bus functions, XB PC <01:00> from the ADD, and ISIZE <01:00>, IRD 1, and LD OSR from the DPM.

Whenever the PC is loaded from the WBus, the PRK flushes the XB by prefetching two longwords of I-Stream information at the new address.

Table 2-32 illustrates that, depending on the state of the XB SEL (XB Select) level, byte 0 of one register is concatenated to byte 3 of the other. This allows the contents of both to be rotated as a quadword for sourcing to the XB decoder or the MBus.

Table 2-32 XB Rotation

XB SEL	XB 01	PC 00	MBUS Byte 3	MBUS Byte 2	XB Decoder Byte 1 MBUS Byte 1	XB Decoder Byte 0 MBUS Byte 0
0	0	0	XB1 B3	XB1 B2	XB1 B1	XB1 B0
0	0	1	XB0 B0	XB1 B3	XB1 B2	XB1 B1
0	1	0	XB0 B1	XB0 B0	XB1 B3	XB1 B2
0	1	1	XB0 B2	XB0 B1	XB0 B0	XB1 B3
1	0	0	XB0 B3	XB0 B2	XB0 B1	XB0 B0
1	0	1	XB1 B0	XB0 B3	XB0 B2	XB0 B1
1	1	0	XB1 B1	XB1 B0	XB0 B3	XB0 B2
1	1	1	XB1 B2	XB1 B1	XB1 B0	XB0 B3

XB Rotation – While the XB SEL level alternately designates the outputs of one register as currently active for the XB decode bus, it enables the inputs to the other to clock new prefetch data as the I-Stream contents are used.

If, for example in Table 2-32, the instruction begins with XB SEL and XB PC <01:00> equal to zeros, bytes <B1:B0> of XB1 are sourced to the XB decoder for the opcode and first OSR. For IRD 1, the PC is always incremented by +2 and XB PC is equal to 10_2. If the source OSR designates a longword-immediate, bytes <B1:B0> of XB0 and bytes <B3:B2> of XB1 are sourced to the MBus as a longword. Since all XB1 data is utilized, the XB SEL is set to a one and a prefetch to the next longword address (PC+4 from the ADD) clocks new data to XB1. With XB PC still equal to 10_2, bytes <B3:B2> of XB0 are now aligned with the XB decode bus for the second OSR.

XB decode bus drivers for byte 0 are always enabled. The drivers for byte 1 are normally enabled except when XB data for byte 1 is sourced back to byte 0 of the DBus. Bytes <3:1> of the DBus and the DBUS ROT <S1:S0> levels are all zero to source the information to the MBus via the MDR and M MUX.

2.5.4 Translation Buffer (TB)

A linear array of over four billion bytes of virtual address space is available on the VAX-11/750. All user virtual space is mapped (allocated) by the system software to physical main memory. The TB is a 2-way set associative cache memory that provides fast access to address translation and protection information. If the TB does not contain a valid translation when a virtual address reference is attempted, a microtrap occurs. The translation information is retrieved from memory or disk, stored in the TB, and the reference is retried.

2.5.4.1 TB Organization – Figure 2-49 is a basic block diagram of the TB. The TB consists of two identical sets of RAM matrices. Each is accessed by a virtual address from the memory address (MAD) bus. They are designated group 1 and group 0 with 256 locations each for 2-way set association. The PTE data matrix for each group is identified by a corresponding translation (tag) address matrix. The output of the group producing a TB hit is gated from the multiplexer to the physical address (PAD) bus for the address translation.

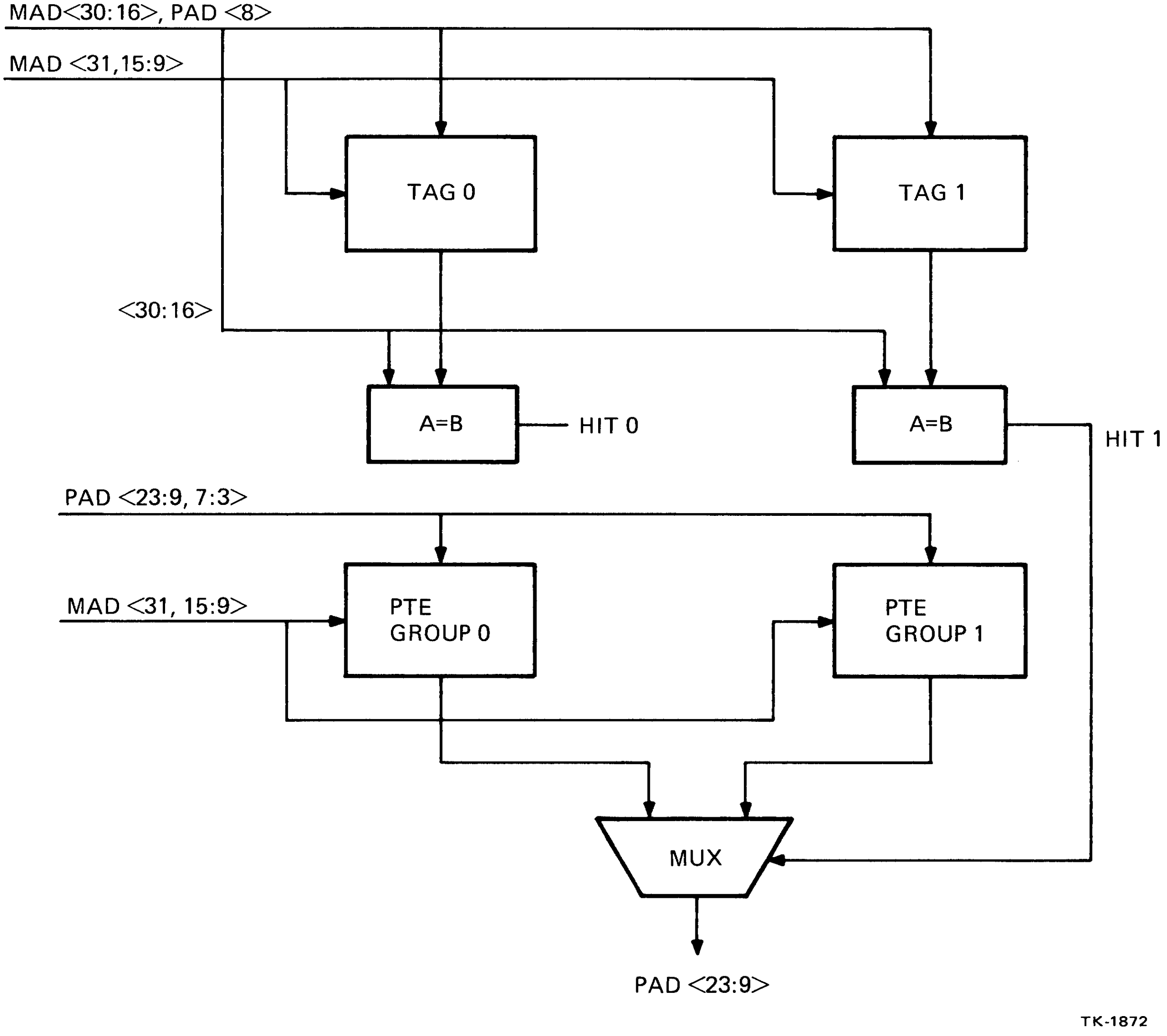


Figure 2-49 Translation Buffer

The 256 locations of each group are further divided into two parts. The upper 128 locations are reserved for system PTEs (MAD <31> = 1). Process PTEs occupy the lower 128 locations (MAD <31> = 0). The upper or lower areas are selected by MAD <31>. This facilitates invalidation of only the process PTEs on a context swap.

Virtual Addressing – Figure 2-50 illustrates how the tag and index fields of the VA access the TB and other translation functions. Virtual address (VA) bits <31:09> on the memory address (MAD) lines from the ADD block access the TB and are broken down into two fields.

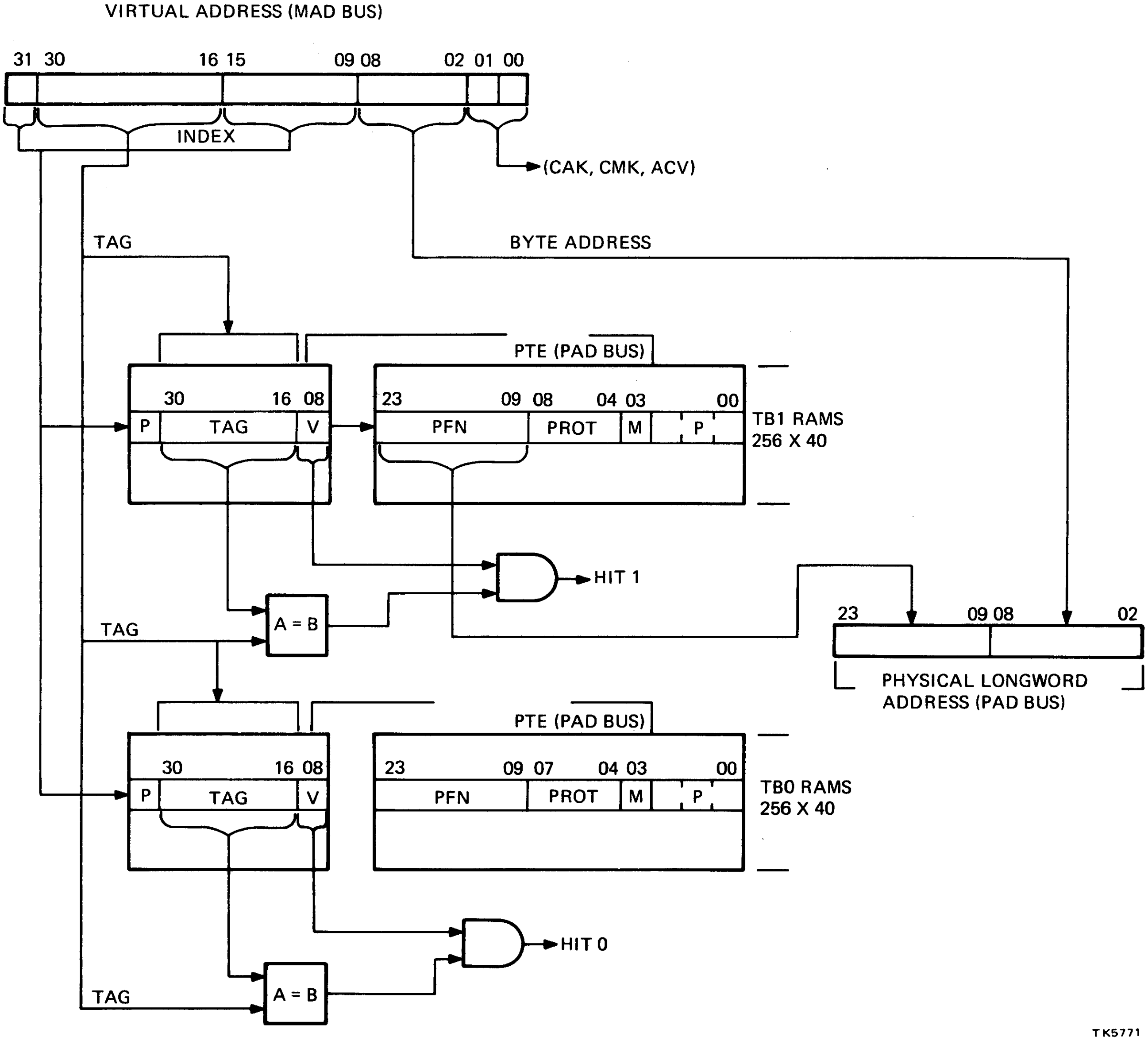
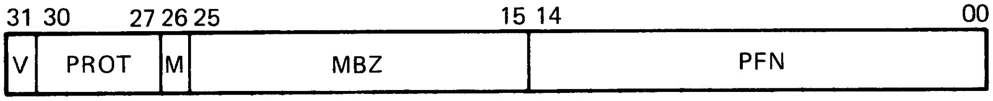


Figure 2-50 TB Functions

Index field, bits <31> and <15:09>, selects corresponding addresses in both tag and PTE data store groups. Bit <31> selects the lower or upper 128 locations for access to process or system PTEs. On a context switch, only the process PTEs in the lower half of the TB are invalidated.

Tag field, bits <30:16>, is written to the tag store while the corresponding PTE with translation data is written into the data store. PTE data is received from the MDR register on the physical address (PAD) lines. Bit <08>, the valid bit, is stored as part of the tag matrix that generates and stores one parity bit.

PTE Rotation – When a PTE is generated, it is stored in memory in the format shown in Figure 2-51. When retrieved from memory, it is rotated by the microcode nine places to the left as shown in Figure 2-52 for assertion on the WBus to the MDR and is stored in the TB. This places the PFN field into PAD bits <23:09> as shown in Figure 2-50. During an address translation the PFN is concatenated with bits <08:02> of the virtual address on the PAD bus to provide the physical longword address. VA bits <01:00> direct byte offset functions in the CAK, CMK, and ACV chips (the odd or even address of a byte, word, or longword).

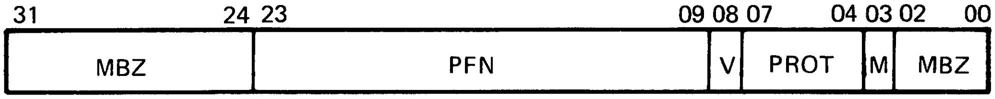


VALID BIT (V)	GOVERNS VALIDITY OF M BIT AND PFN FIELD
	V = 1; PAGE CAN BE ACCESSED BY EXECUTING PROCESS
	V = 0; PAGE CANNOT BE ACCESSED BY EXECUTING PROCESS

PROTECTION FIELD (PROT)	ALWAYS VALID AND USED BY HARDWARE EVEN WHEN V = 0
MODIFY BIT (M)	M = 1 IF PAGE HAS ALREADY BEEN RECORDED AS MODIFIED M = 0 IF PAGE HAS NOT BEEN RECORDED AS MODIFIED USED BY HARDWARE ONLY IF V = 1
BITS <25:15> (MBZ)	MUST BE ZEROS RESERVED FOR SYSTEM SOFTWARE
PAGE FRAME NUMBER (PFN)	UPPER 15 PHYSICAL ADDRESS BITS OF THE PAGE LOCATION USED BY HARDWARE ONLY IF V = 1.

TK5772

Figure 2-51 Page Table Entry Format



TK5773

Figure 2-52 PTE After Rotation

TB Hit – When the TB is presented with a virtual address reference, index bits <31> and <15:09> select the same location in both groups of matrices. Whichever group generates equality between the tag field of the incoming virtual address and the tag store contents must also have the V bit set to provide a TB hit. This indicates that the related data matrix location contains the correct page frame number for the address translation.

TB Miss – If the valid bit is clear (page invalid) or if no match exists between the TB tag of the indexed location and the VA tag field, a TB miss microtrap occurs. The PTE is read from memory to the TB and the reference is retried.

Only valid PTEs are loaded to the TB. Invalidation of TB entries is performed by the operating system when initialized or when it removes a page from the working set. When a PTE is read from memory into the MDR register on a TB miss, it is asserted onto the PAD bus via the WBus. The M bit, V bit and access privileges are checked by the logic before it is written into the TB.

The entire TB is invalidated upon system initialization to provide invalid PTEs with good parity. Process PTEs are invalidated during context switching since mapping is different for each process. Processes may have access privileges to common areas.

When a PTE is written to the TB, three parity bits (PAR <2:0>) are generated and stored in the data matrix. Each parity bit monitors the following data bits on the PAD bus.

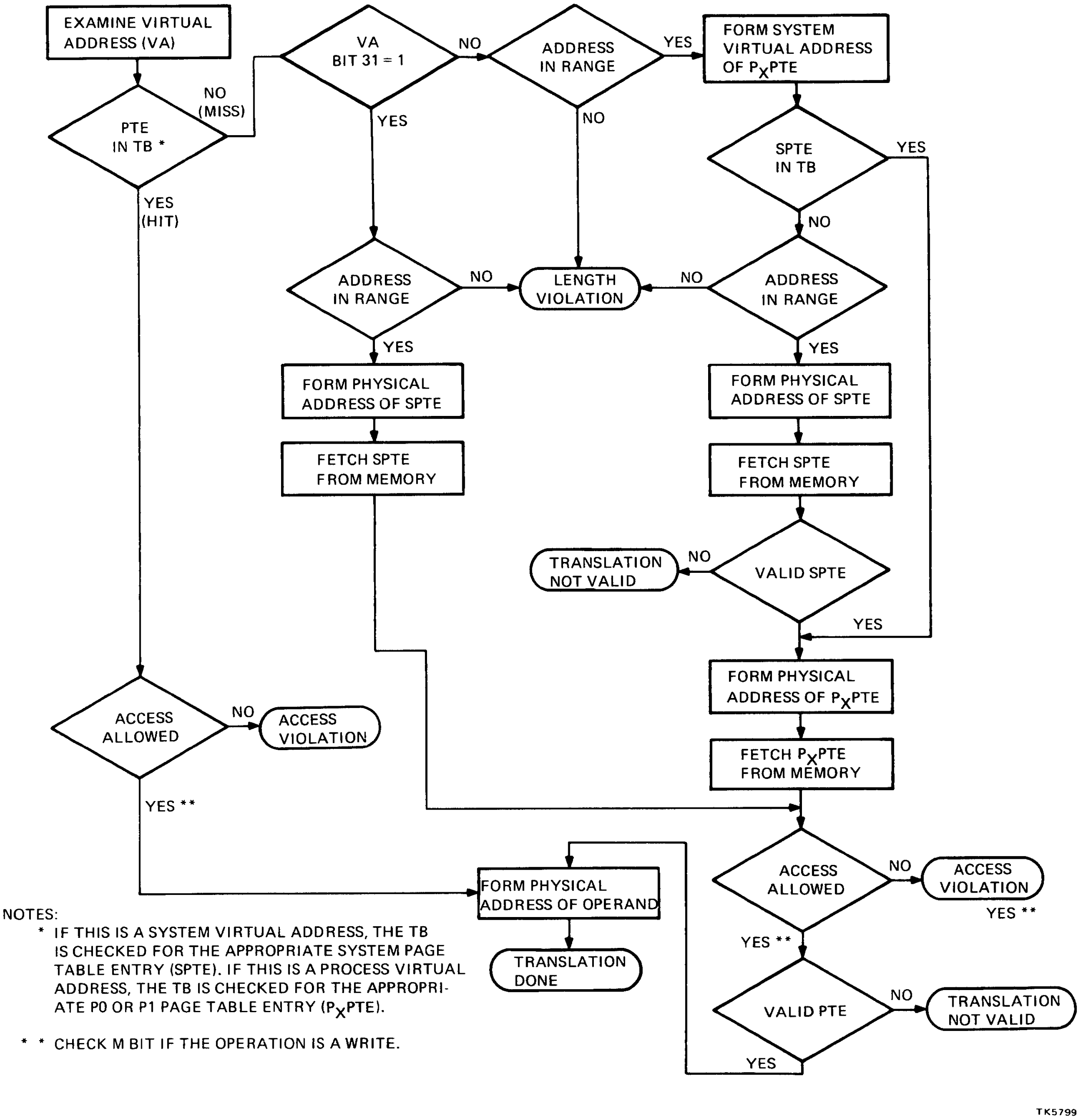
PAR 2 = PAD <23:18>
PAR 1 = PAD <17:11>
PAR 0 = PAD <10:09>, M Bit, and PAD <07:04> (access protection bits)

2.5.4.2 Address Translation – To support TB functions within memory management, a series of checks and responses are incorporated in the firmware as shown in Figure 2-53. If the translation information in the TB is not valid, a TB miss microtrap occurs and the translation discontinues. If the information cannot be found, or if a length or access violation occurs, a fault to the operating system takes place for software intervention.

TB Hit – A TB hit occurs when the tag field of the VA (MAD <30:16>) is equal to the tag contents of the PTE and the valid (V) bit is set at the location selected by the VA index field (MAD <31, 15:09>). Figure 2-54 shows a reference to P0, P1, or S0 space that results in a TB hit. (Bits <31:30> of the virtual address are equal to 00, 01, or 10, code 11 is unused.) VA bits <01:00> are not used in the translation since cache and memory information is longword-aligned. Contents of the TB hit address are output and the PFN points to the base address of the page in main memory. The byte offset field of the VA selects a longword within the page. This is the physical longword address of the data in memory.

For a TB hit to either system or process space (VA bit <31> = 1 or 0), the translation is completed unless an access violation occurs (Figure 2-53). No check is made for a length violation since the PTE could not be in the TB. A length violation on the first reference to the page does not load its PTE to the TB.

The M bit is checked during a write reference. If not set, a microtrap occurs. The PTE is fetched from main memory and the M bit is set. The PTE is rewritten to memory and the TB. The write is then completed.



NOTES:

- * IF THIS IS A SYSTEM VIRTUAL ADDRESS, THE TB IS CHECKED FOR THE APPROPRIATE SYSTEM PAGE TABLE ENTRY (SPTE). IF THIS IS A PROCESS VIRTUAL ADDRESS, THE TB IS CHECKED FOR THE APPROPRIATE P0 OR P1 PAGE TABLE ENTRY (P_XPTE).

- * * CHECK M BIT IF THE OPERATION IS A WRITE.

Figure 2-53 Address Translation Flow

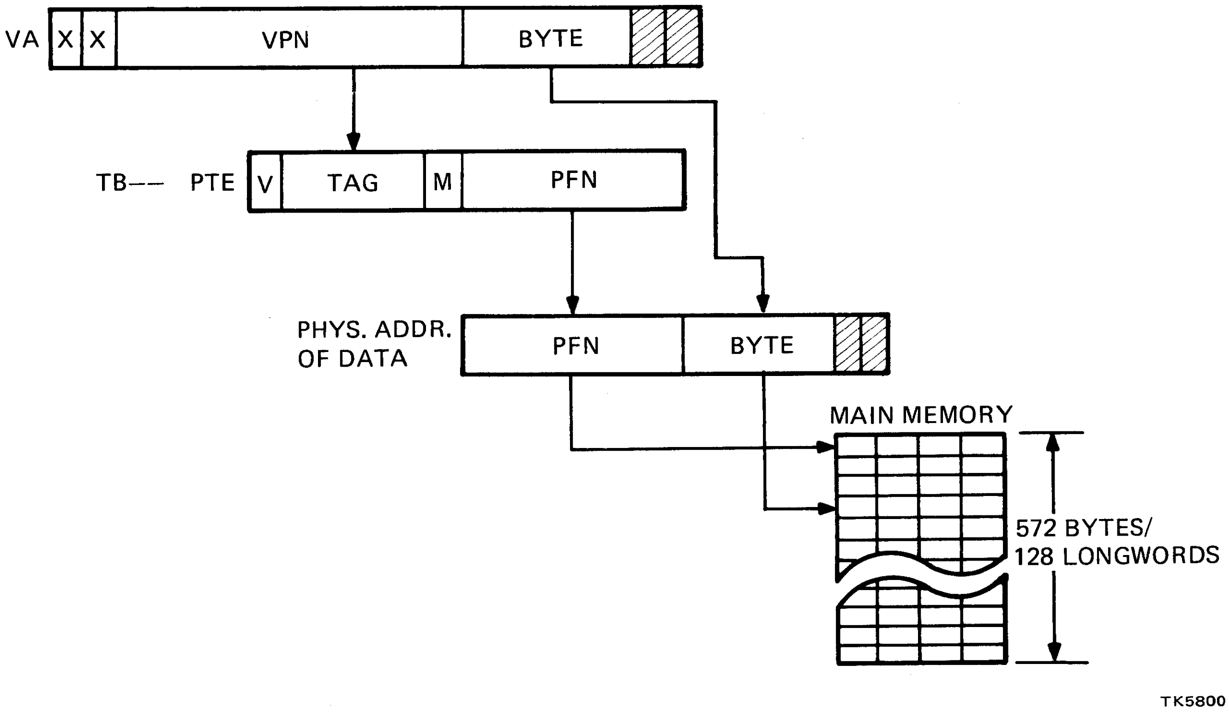


Figure 2-54 TB Hit-System or Process Space

System TB Miss – A TB miss on a PTE in the system region (VA bit <31> = 1) causes a microtrap response as shown in Figure 2-55. After a page length check (Figure 2-53), the physical address of the system PTE (SPTE) is formed by aligning and adding VPM bits <29:09> of the VA to the contents of the system base register (SBR, bits <23:02>). Bits <31:30> of the SBR are 00 since the contents are a physical address.

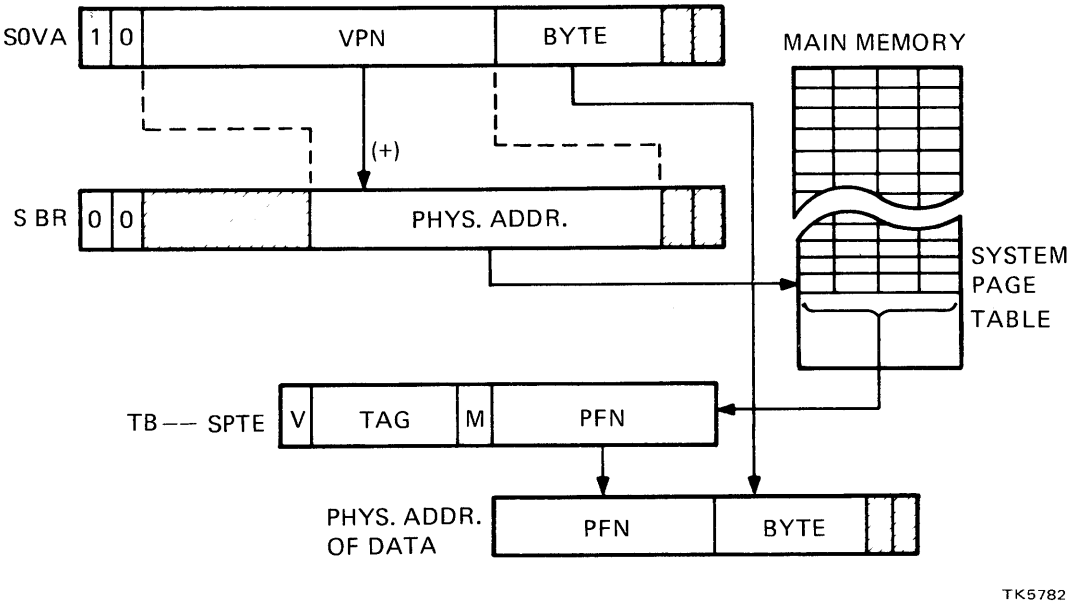


Figure 2-55 System TB Miss

The SPTE is retrieved from cache or main memory by the microcode, which does an access check before writing it to the TB. The protection code is checked before the V bit to avoid the overhead of writing a PTE to which access is not allowed into the TB. After an M-bit check for a write, the translation continues and the physical longword address is formed. If the M bit must be set, another branch of the microcode is selected to accomplish all tasks rather than allowing a microtrap to occur during a microtrap.

If the SPTE from memory is not valid (V bit is clear), a translation-not-valid fault calls for software intervention. The page is faulted and read into memory from disk, along with its corresponding SPTE. A retry on the reference loads the valid SPTE to the TB and the translation completes with a TB hit.

Process TB Miss – A TB miss on a PTE in the control or program region causes a microtrap response as shown in Figure 2-56. After a page length check of he virtual reference against the process length register (P0LR or P1LR), the VPN of the process VA is added to the contents of the process base register (P0BR or P1BR). The resulting virtual address makes access to memory from the SPTE in the TB.

The physical PFN from the TB is used with a byte offset from the P0BR or P1BR to retrieve the PxPTE from the process page table in cache or main memory. If the V bit is set, the PxPTE is written to the TB after the access code and M bit are checked. The reference is then retried. If the V bit in the PxPTE fetched from memory is clear, a translation-not-valid fault occurs to the operating system. The page and its valid PTE are then faulted from disk to memory.

When reference is made to a process page for a write and the M bit is clear in the PxPTE, the M bit of the SPTE is also checked. If clear, the M bit of the SPTE is set in the TB to avoid an M bit microtrap when the updated PxPTE is written to memory.

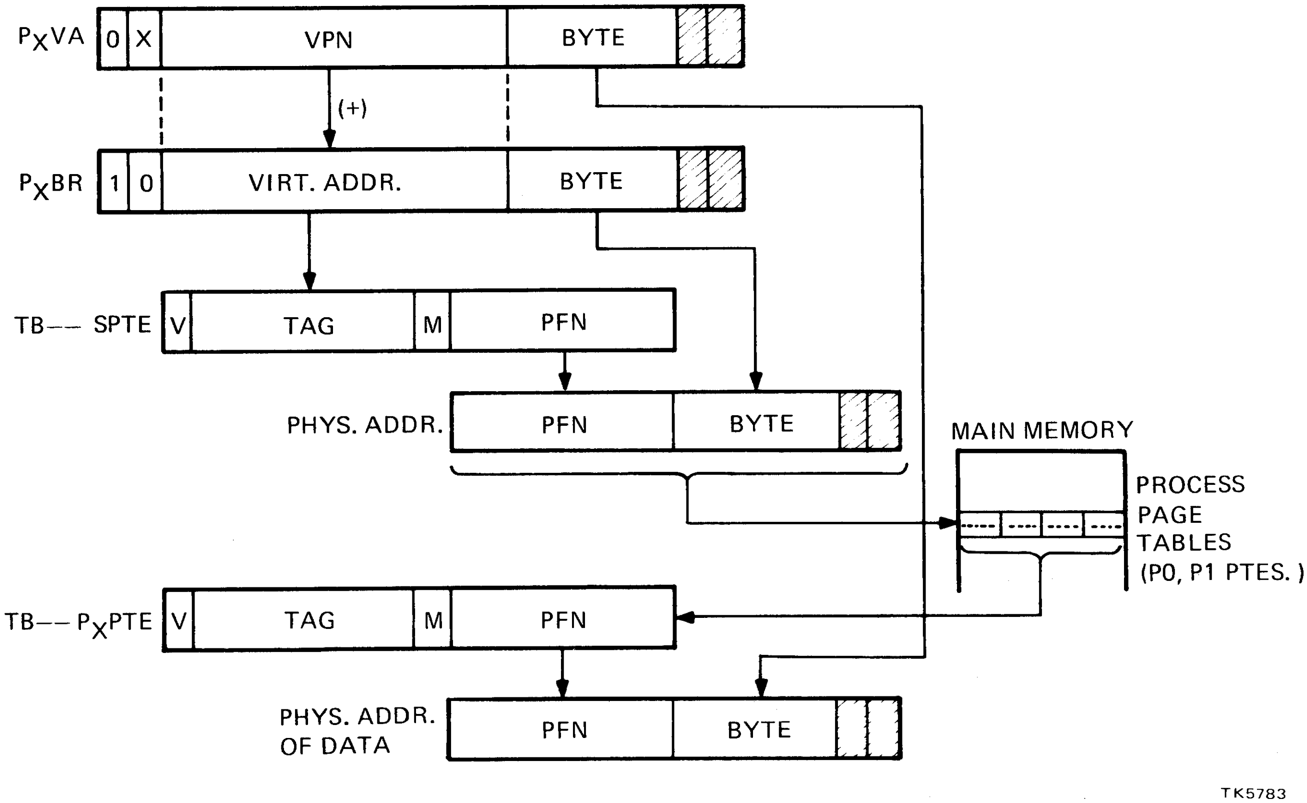


Figure 2-56 Process TB Miss

Process TB Double Miss – If the SPTE for the process PTE is also not in the TB, it must be retrieved from memory first. In Figure 2-57, the process VPN added to PxBR contents produces a virtual address. This VA, unable to be translated by the TB, is aligned and added to the SBR. This provides the physical address to retrieve the SPTE from memory. Once the SPTE is in the TB, the PxPTE is loaded as for a single miss, and the translation continues.

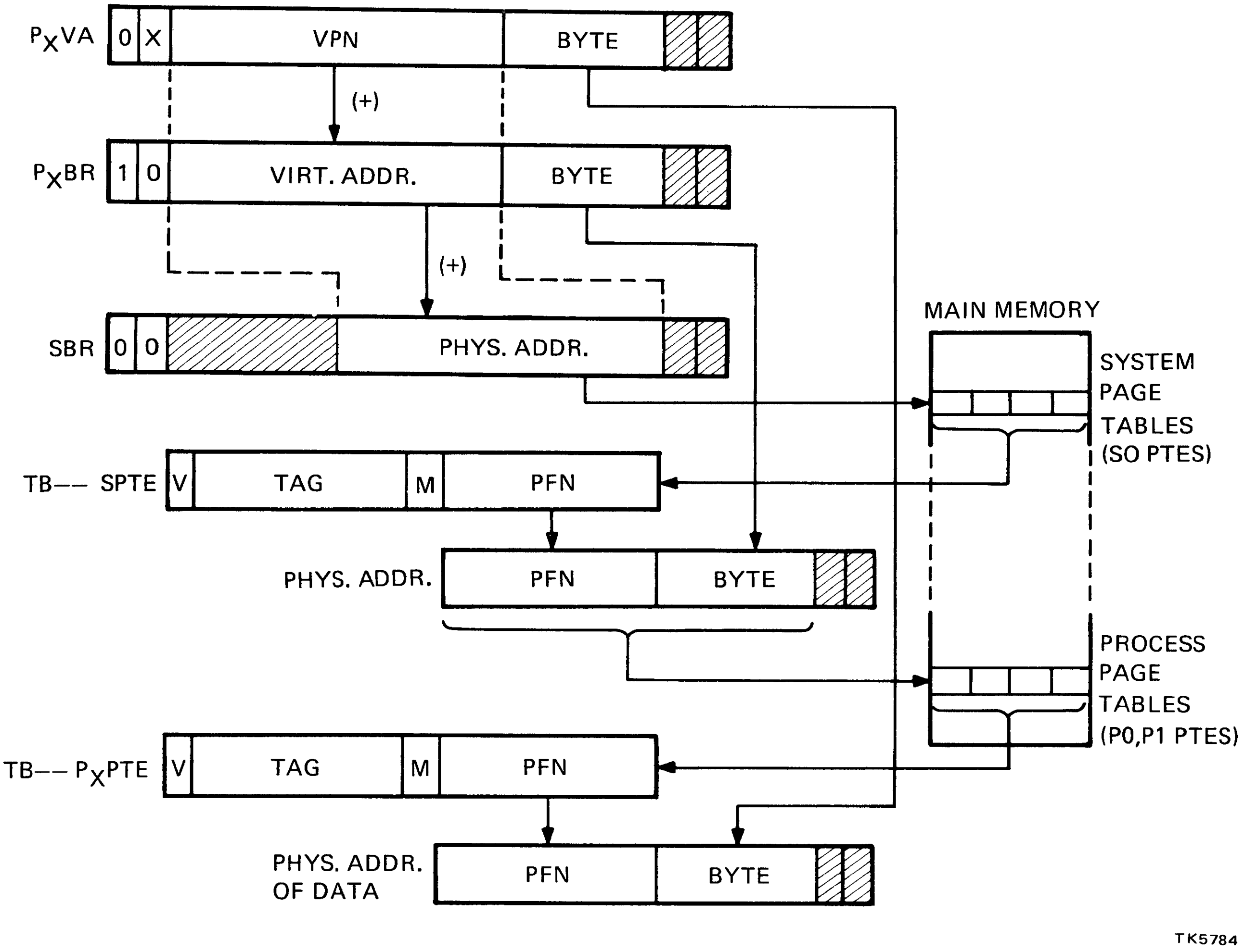


Figure 2-57 Process TB Double Miss

Memory Management Exceptions – An access violation occurs for two cases. A protection code violation occurs when the intended access request (read, write, or read modify) is not allowed for the current processor access mode.

A length violation occurs when the virtual page number of a P0 VA or S0 VA is greater than or equal to the contents of the P0LR or S0LR. Since P1 space grows toward lower addresses, a length violation fault occurs when the VPN is less than the contents of the P1LR.

A translation-not-valid fault occurs when the V bit is clear in the PTE fetched from memory by the microcode. Control is passed to an executive routine called the pager. The pager uses the information from the invalid PTE to locate the page on disk. It then adds it to the working set of the requesting process.

Since process page tables are mapped by system PTEs, a process VA may incur page faults to retrieve both the process PTE and the system PTE for the process. For any of these faults, the PSL and PC are pushed onto the kernel stack, followed by the faulting virtual address and a status longword describing the violation. Control and status register bits are described in Paragraph 2.5.6.

2.5.5 Cache Memory

Cache is a high-speed memory buffer for the storage of up to 4K bytes of data in 1,024 index locations. Its purpose is to reduce memory access time by storing data most likely to be required by the process(es) currently executing on the system. The most significant reduction is in the execution time for localized programs and frequently used routines or program loops.

VAX-11/750 cache uses the direct mapping technique. A physical address reference is compared to a stored address to access the stored data. If the data is not in cache, it is fetched from memory and loaded to cache for possible reuse.

2.5.5.1 Cache Organization – Like the TB, cache consists of an address matrix and a data matrix as shown in Figure 2-58. The index field, bits <11:02> of the physical address from the PAD bus, selects one of 1,024 index locations.

The tag field, bits <23:12> of the physical address, is stored in the address matrix along with one parity bit and the cache valid bit from the cache control chip (CAK). All cache locations are invalidated by the microcode when the machine is initially turned on.

A longword of data from the MDR block is stored in the corresponding index location of the data matrix. Four parity bits are generated and are stored in the data matrix, one for each byte of the data longword.

2.5.5.2 Cache Operation – Cache data is longword-aligned with the CMI. Alignment of cache or CMI data with the DPM takes place within the MDR block (Paragraph 2.5.3).

Cache Hit – A cache hit for a CPU memory reference results when the tag field of the physical address is equal to the contents of the address matrix, and the valid (V) bit is set. This indicates that valid data for the operation is stored in the corresponding index location of the data matrix. Cache can only be accessed by the CPU. A read or write cycle on the CMI originated by an I/O device does not have access to cache information.

Cache Miss – A cache miss results when the tag address bits do not agree or when the V bit is clear on a CPU memory reference. This indicates that the data is not in cache for the referenced address (tag fields are unequal), that cache does not contain the most recent data for the operation (V bit is clear), or both.

Read Hit – A cache hit on a CPU read to memory results in cache data being transferred to the MDR block (Paragraph 2.5.3). Any byte rotation takes place for the DPM as the data is clocked to the MDR register from the DBus rotator. With the data available in cache, no reference to slower main memory is necessary.

Read Miss – When a CPU read reference results in a miss, a CMI cycle is initiated to retrieve the data from main memory and to store it in cache.

Write Hit – A CPU write to memory that causes a hit in cache causes the new data to be written both to cache and to main memory. This is the write-through technique. Although extra time is required to write the data to main memory, this technique allows both main memory and cache to contain the updated information.

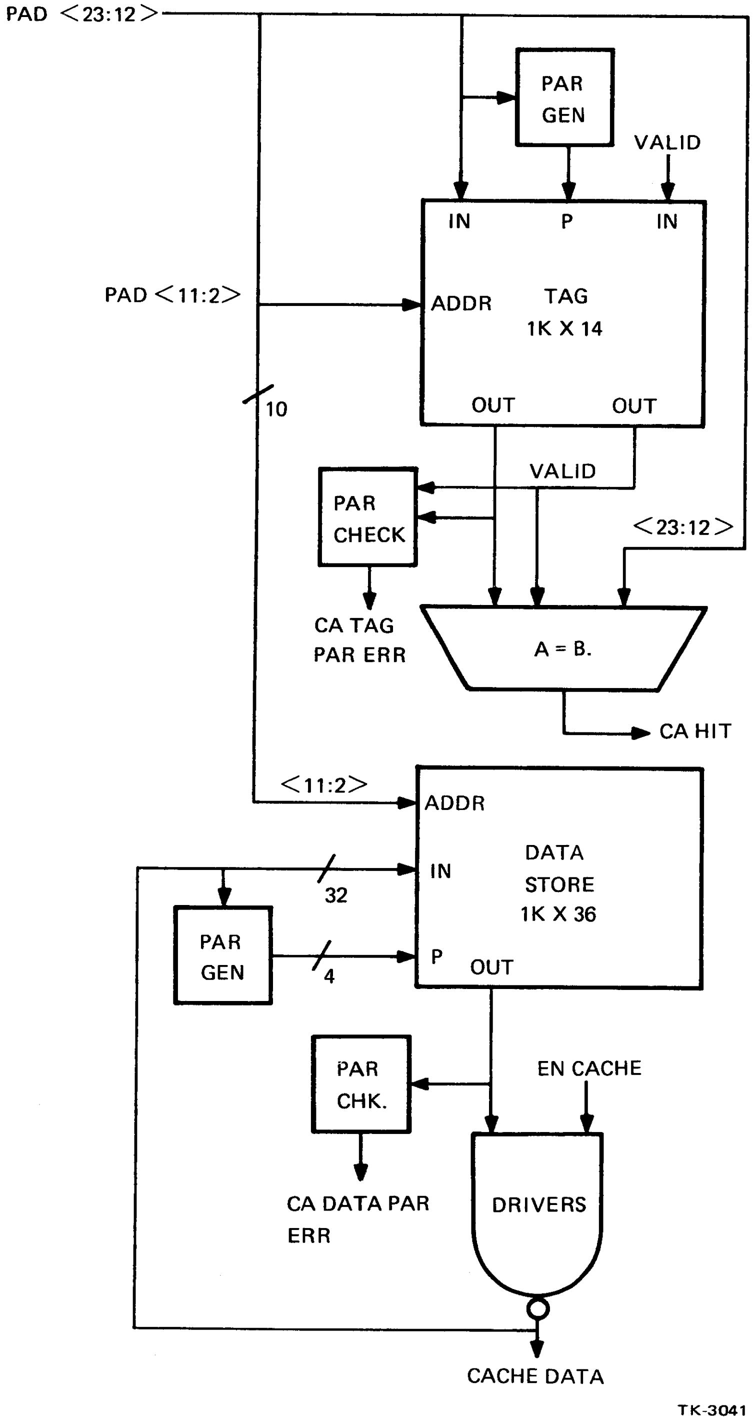


Figure 2-58 Cache Memory

Writes to memory by I/O devices are monitored on the CMI and checked for hits on cache addresses. A hit by an I/O device causes the cache location to be invalidated. A read reference to that location by the CPU then causes the updated information to be loaded to cache by a read miss.

Write Miss – An aligned longword write to memory by the CPU is written to cache as it is for a hit. If the CPU data is unaligned, or is less than a longword in length, cache is not written. If the information is later retrieved by the CPU, cache is then updated by a read miss. A cache write miss by an I/O device updates the main memory location and does not alter cache.

2.5.6 Memory Status/Control Registers

MIC status and control registers are accessed by the software or from the console as internal processor registers (IPRs). They are read or written on WBus bits <27:24> under WCTRL field control by the microcode. The memory status/control address register (MEMSCAR) is loaded from WBus <27:24> with a register address. The selected register is then affected by the source or destination WCTRL code. Figures 2-59 through 2-61 illustrate bit functions of the TB, cache, and memory management registers contained in the MIC control chips. All registers are initially zero. Also shown are the bit positions and IPR numbers (in hexadecimal), as well as the MEMSCAR number and chips that contain the registers.

Memory Management Enable (MME) – Bit <0> of IPR 38 is a read/write bit. When set, memory management is enabled and the address from the ADD block is virtual. When clear, memory management is disabled and the address is physical for direct access to cache or main memory.

TB Hit Register (TBHR) – Bit <4> of IPR 17 is a read-only bit that saves the status of the last microcode reference made to the TB for an address translation.

TB Group Disable Register (TBGDR) – The TBGDR, IPR 24, is a read/write register. If bit <3> is a one, bit <2> selects which group is replaced. When zero, bit <3> designates random replacement to either group when a PTE is loaded from memory. Bits <1,0> are set to disable either group by forcing a miss.

TB Group Parity Register (TBGPR) – TB parity error bits <11:08> of IPR 17 are read-only. If any bit is set, bit <2> of the MCESR reads as a one. Writing a one to bit <2> (TB error) of the MCESR, from the console or the software, clears all bits.

Cache Error Summary Register (CAER) – All bits are read/write. Bit <0> of IPR 27 saves the status of the last microcode reference to cache. Bits <3,2> hold parity error status of cache tag and data fields. Bit <1> is set by an access to cache if an error condition is encountered before a previous one is serviced.

Cache Disable Register (CADR) – Bit <0> of IPR 25 is read/write. When set, cache hits are disabled.

Cache Write-Only Register – Bit <20> of IPR 17 is read/write only by the microcode. When set (diagnostic mode only), CPU writes to the CMI are disabled and only cache is written.

Machine Check Error Summary Register (MCESR) – The MCESR, IPR 26, is read/write. Writing a one to bit <3> or bit <2> from the console or the software clears the summary register for bus errors or TB errors.

Bus Error Summary Register – Bus error bits <3:0> of IPR 17 are read-only. If any bit is set, bit <3> of the MCESR reads as a one. Writing a one to bit <3> (bus error) of the MCESR, from the console or the software, clears all bits.

Saved Mode Register – Saved mode bits <19:16> of IPR 17 are read/write and reflect the processor access mode and memory management states during the last microcode reference to memory.

Write Vector Occurred Register – Bit <12> of IPR 17 is read/write. It is first cleared when a bus grant is issued, then set in response to the write vector transaction on the CMI. It is also set by a read lock timeout (and NXM status is returned to the CPU).

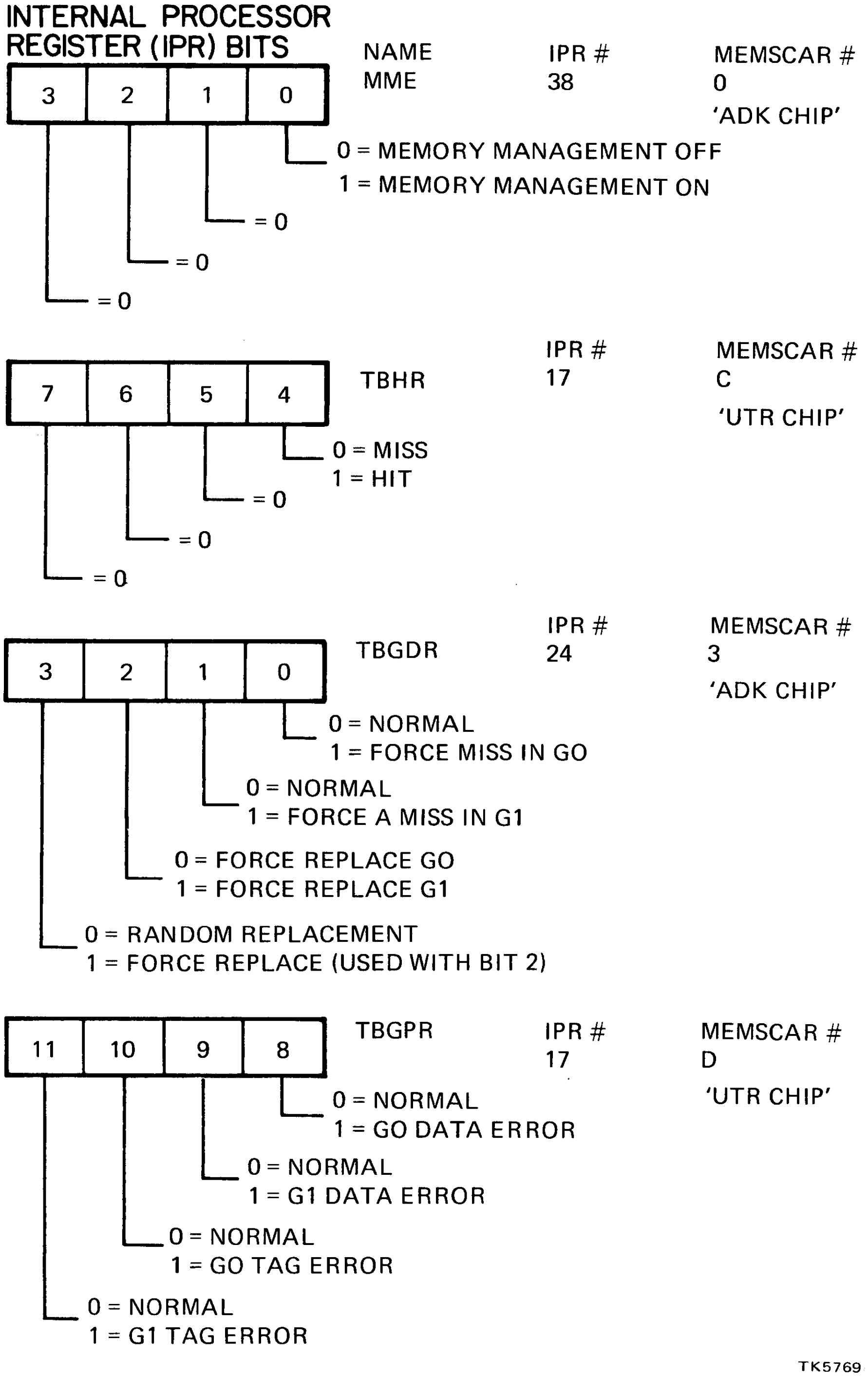


Figure 2-59 TB Registers

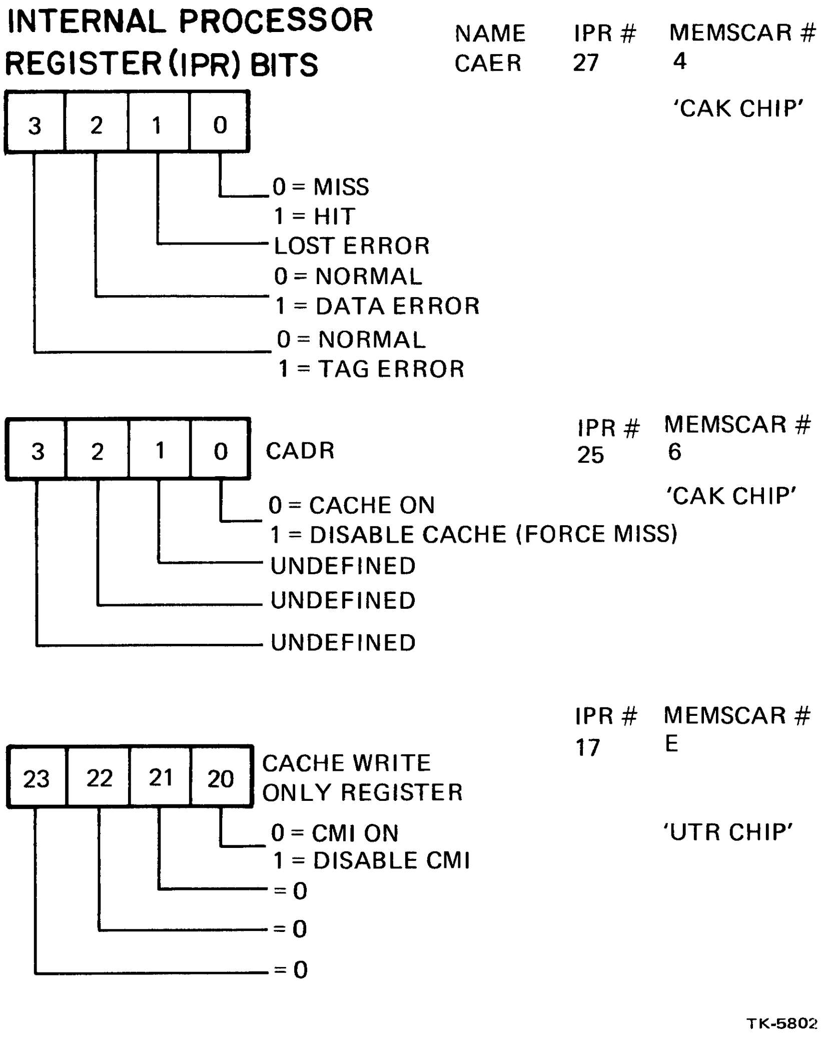


Figure 2-60 Cache Registers

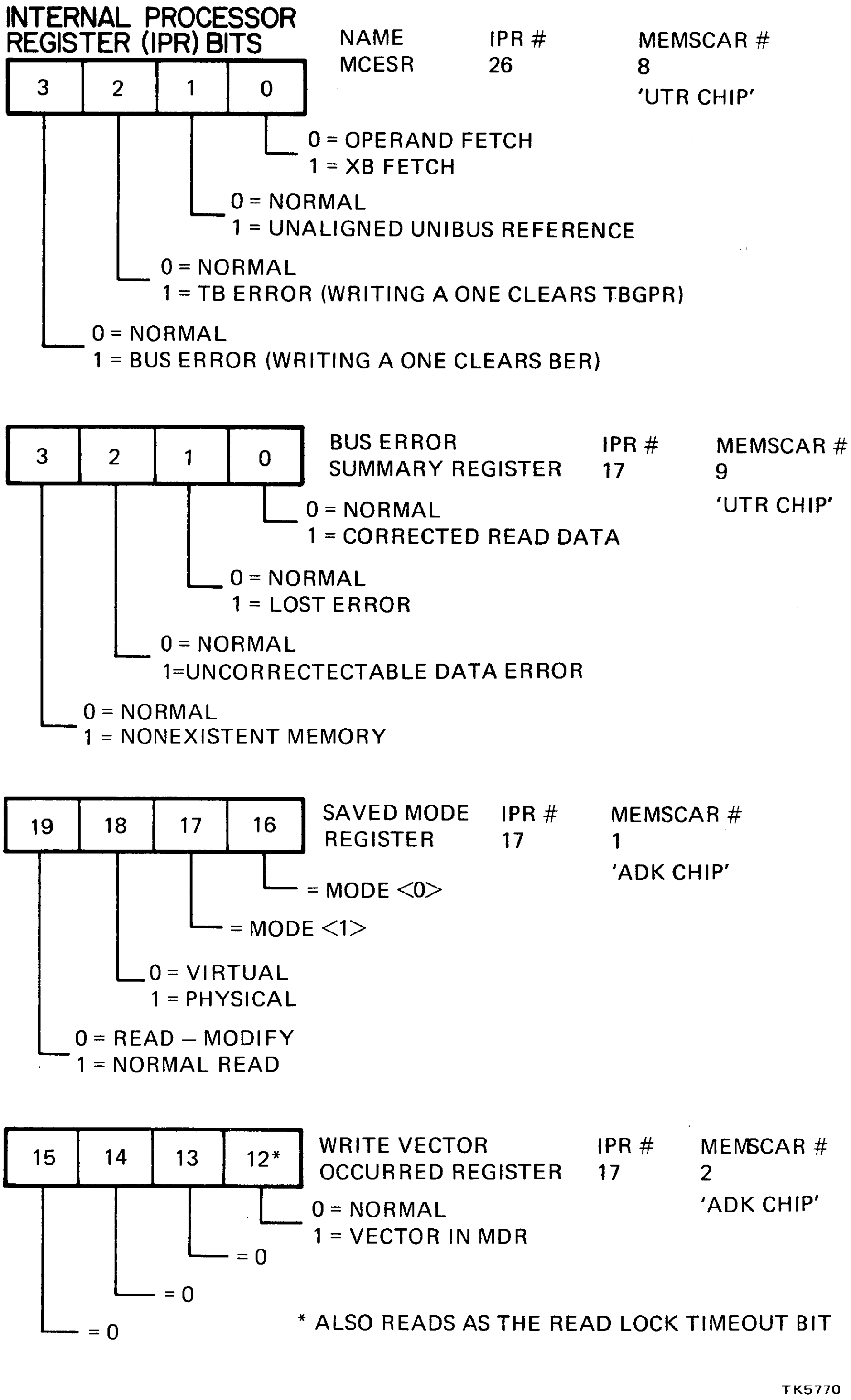


Figure 2-61 Status/Control Registers

2.5.7 Memory Interface Micro-Orders

This paragraph describes bus function code assignments, WCTRL codes, and MSRC codes. These codes are all in hexadecimal.

2.5.7.1 Bus Function Codes – The following is a list of the code assignments (in hex) for the bus function microfield. The functions are further defined following the list.

Code	Function
00	Read Physical Address
01	Processor Initialize
02	Read, No Microtrap
03	I/O Initialize (Not Used by MIC)
04	Read Lock Timeout Test
05	NOP
06	Read, Second Reference
07	NOP
08	Write Physical Address
09	REI Check (Not Used by MIC)
0A	Write, Second Reference
0B	Write Unlock, Second Reference
0C	Write, No Microtrap
0D	NOP
0E	Write Longword, No Microtrap
0F	Bus Grant
10	Read
11	Read Longword
12	PTE Access Check, Write
13	Read Lock
14	Read with Modify Intent
15	Read Longword with Modify Intent
16	PTE Access Check, Read
17	PTE Access Check, Read, Kernel Mode
18	Write
19	Write Longword
1A	Write If Not R Mode
1B	Write Unlock
1C	Probe Access, Write, Mode Specified
1D	Probe Access, Write
1E	Probe Access, Read, Mode Specified
1F	Probe Access, Read

The following is a brief description of the memory interface bus functions.

(10) Read – Replace the contents of the MDR register with the contents of the memory location specified by the virtual address presently in the VA and D-size.

(14) Read with Modify Intent – Checked for Write access. Otherwise, same as Read unless the resulting physical address is in Unibus space. In this case the Unibus must perform an interlocked operation (DATIP).

(11) Read Longword – Same as Read, except the two least significant bits of the address are ignored (for field instructions).

(15) Read Longword with Modify Intent – See Read Longword and Read with Modify Intent.

(02) Read, No Microtrap – Same as Read, but suppress ACV (access violation) and unaligned data microtraps.

(13) Read Lock – Same as Read; checked for Write access. In addition, signifies to other masters on the CMI that they must not perform Read Lock operations until a Write Unlock operation has taken place. If the CPU is unable to perform a Read Lock within approximately 64 μs of the time it was initiated, a Read Lock Timeout occurs. The Read Lock operation is aborted, a nonexistent memory machine check occurs, and the write vector occurred bit is set in the appropriate status/control register.

(00) Read Physical Address – Same as Read except that the address in the VA is to be used as a physical address instead of a virtual address and the two least significant bits are ignored.

(06) Read, Second Reference – Indicates to the memory interface control logic that a previous Read crossed a longword boundary. Therefore, only the portion of data fetched from memory that was not previously fetched should be clocked into the MDR.

(04) Read Lock Timeout Test – Special function for testing timeout counter in MDR chips.

There are three categories of write bus functions.

1. Those that load the write size latch. This category includes the following functions.

   a. Write
   b. Write if Not R Mode
   c. Write Unlock
   d. (Write Longword)

NOTE
Write Longword causes the write size latch to be loaded with D-size, but always writes all four bytes.

2. Those that use the latched size. This category includes the following functions.

   a. Write, Second Reference
   b. Write Unlock, Second Reference
   c. Write, No Microtrap

3. Those that always write all four bytes regardless of D-size. This category includes the following functions.

   a. Write Physical Address
   b. Write Longword, No Microtrap
   c. Write Longword

The write size latch is loaded with D-size during any microstep that specifies a category 1 write bus function, regardless of any destination inhibits or microtraps that might occur during that microstep.

(18) Write – Replace the contents of the memory location specified by the virtual address presently in the VA and D-size with the contents of the WDR register.

(1A) Write if Not Register Mode – Same as Write unless R Mode (register mode) from the microsequencer is asserted, in which case do nothing.

(1B) Write Unlock – Same as Write. In addition, releases the interlock set by a Read Lock operation.

(0A) Write, Second Reference – Indicates to the memory interface control logic that a previous write crossed a longword boundary. Therefore only the portion of the data in the WDR that was not previously stored should be written into the specified memory location.

(0B) Write Unlock, Second Reference – See Write Unlock and Write, Second Reference.

(0C) Write, No Microtrap – Same as Write, but suppress ACV (access violation), unaligned data, and page boundary crossing microtraps.

(08) Write Physical Address – Same as Write except that the address in the VA is to be used as a physical address instead of a virtual address and the two least significant address bits are ignored.

(0E) Write, No Microtrap, Long – Same as Write, No Microtrap, except that a longword is written ignoring the latched write size. Used for writing the M bit during mapping subroutines.

(19) Write Longword – Same as Write, except the two least significant bits of the address are ignored (for field instructions).

(1F) Probe Access, Read – Check the translation buffer entry corresponding to the address presently in the VA against the current mode for validity and read access. Indicate the results of the check on the microvector lines as follows.

NOTE

The following signal name abbreviations are used to define the state of the microvector lines during Probe and PTE Check micro-orders.

M	= PTE modify bit
V	= 1 if valid PTE
AC	= 1 if access allowed
PBOK	= 1 if not crossing a page boundary
PA	= 1 if memory mapping is not enabled (physical address)

On Probe the microvector lines are as follows.

MICROVECTOR <3> = (PBOK .AND. V .AND. AC) .OR. PA

MICROVECTOR <2> = M .AND. [(V .AND. AC) .OR. PA]

MICROVECTOR <1> = V .OR. PA

MICROVECTOR <0> = (AC .AND. V) .OR. PA

On PTE Check the microvector lines are:

MICROVECTOR <3> = 0
MICROVECTOR <2> = M .AND. V .AND. AC
MICROVECTOR <1> = V .AND. AC
MICROVECTOR <0> = AC

(1E) Probe Access, Read, Mode Specified – Same as Probe Access, Read except that access is checked against WBUS <25:24> instead of the current mode.

(16) PTE Access Check, Read – Same as Probe Access, Read except that a PTE image on the WBus is checked instead of a translation buffer entry. Note that the valid bit and the protection code bits must occupy the same positions on the WBus as they would if the PTE were to be loaded into the translation buffer.

(17) PTE Access Check, Read, Kernel Mode – Same as PTE Access Check, Read except that access is checked against kernel mode instead of current mode.

(1D) Probe Access, Write – Check the translation buffer entry corresponding to the address presently in the VA against the current mode for validity and write access. Indicate the results of the check on the microvector lines.

(1C) Probe Access, Write, Mode Specified – Same as Probe Access, Write except that access is checked against WBUS <25:24> instead of the current mode.

(12) PTE Access Check, Write – Same as Probe Access, Write except that a PTE image on the WBus is checked instead of a translation buffer entry. Note that the valid bit and the protection code bits must occupy the same positions on the WBus as they would if the PTE were to be loaded into the translation buffer.

(01) Processor Initialize – Generates a reset signal that initializes status/control registers.

(OF) Bus Grant – Causes a bus grant to be issued on the Unibus in response to the highest level Bus Request. After the grant is issued, memory interface logic stalls the procesor clock until the grantee releases the Unibus. During the time the processor is stalled, a Write Vector transaction may take place on the CMI, which causes an interrupt vector to be written into the MDR. If this happens, the status register write vector occurred bit is set.

Microtraps and Interrupts – In addition to the microtrap and interrupt pending lines from the memory interface control to the microsequencer, there are four microvector lines that describe the microtrap or interrupt. These lines can be used as a branch offset by the microsequencer.

As a result of a microtrap, certain functions in the microstep are inhibited and the normal flows are exited. Upon completion of the microtrap routine, the microcode returns to the microstep that caused the microtrap, and the functions that were previously inhibited are allowed to execute.

In TB miss microtrap subroutines, the microcode must probe ahead on memory references to avoid nested microtraps.

2.5.7.2 WCRTL Codes – The following WBus control codes (in hex) are required for the memory interface.

Code	Function

20	VA ← PC + ISIZE + (WBUS)
	PC ← PC + ISIZE

21	Reserved
22	VA ← VA + 4
23	MDR ← (WBUS)
24	PC ← (WBUS)
25	VA ← (WBUS)
26	MBUS ← WDR
27	MDR ← 0
28	TB DATA ← (WBUS)
29	TB Valid bit ← 0 VA ← (WBUS) (Invalidate both groups at the index position addressed by VA.)
2A	WDR ← (WBUS) Unrotated
2B	MDR ← OSR, Zero-extended
2C	PC ← PC + (WBUS)
2D	Cache Valid bit ← 0 VA ← (WBUS) (Invalidate cache at the index position addressed by VA. The address in the VA register is interpreted as a physical address.)
2E	WDR ← (WBUS)
2F	MDR ← IR, Zero-extended
30	Status/Control register ← WBUS <27:24>
31	Previous Mode register ← WBUS <23:22>
32	WBUS <27:24> ← Status/Control register
33	Bus Grant WBUS <20:16> ← IPL of current Unibus grantee
34	Status/Control Address register← <WBUS27:24>
35	Previous Mode register ← Current Mode register, then IS/Current Mode register ← WBUS <26:24>
37	REI Check
38	ASTLVL register ← WBUS <26:24>
39	Reserved
3A	WBUS <26:24> ← ASTLVL register
3B	Reserved
3C	Highest software IPR Register ← WBUS <20:16>
3D	IPL register ← WBUS <20:26>
3E	Reserved
3F	WBUS <20:16> ←IPL of last Unibus grantee

2.5.7.3 MSRC Codes – The MSRC codes required for the memory interface (in HEX) are as follows.

Code	Function
12	MBUS ← MDR register
13	MBUS ← WDR register
17	MBUS ← XB register (See Paragraph 2.5.3.3)
18	MBUS ← MA
19	MBUS ← PC Backup
1A	MBUS ← PC
1B	MBUS ← VA
1F	MBUS ← TB Data (address in VA is virtual, PAD bits <31:24> read as ones to the WBUS.)

2.5.8 CPU Memory Interconnect (CMI) Description

The CPU memory interconnect (CMI) consists of 45 bidirectional lines that carry address, data, and priority arbitration between all subsystems on the backplane. The signals of the CMI are divided into four groups: timing, data/address and control, priority arbitration, and status. Figure 2-62 and Table 2-33 provide descriptions of the CMI signals.

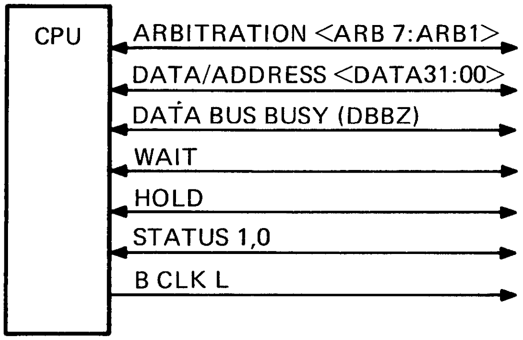


Figure 2-62 CMI Signals

Table 2-33 CMI Signal Description

Signal Line	Description
Timing	
B CLK L	B CLK L is generated by the CPU to synchronize system activity. One B CLK cycle is considered to be from one rising edge of B CLK L to the next. B CLK L is low for one-third of the cycle.
Data/Address and Control Group	
CMI Data <31:00>	The CMI data lines are first asserted by a device that has assumed control as master. The master transmits control and address information to the slave (CMI address). The lines are then enabled for the transfer of data (CMI data). Bits <01> and <00> of the CMI address are ignored since four bytes (one longword) of data are represented on the lines.
Data Bus Busy (DBBZ)	DBBZ is first asserted by the master for one CMI cycle while it places the CMI address on the CMI data lines. DBBZ is then asserted by the slave until data transfer is completed, except for a write operation where the slave is immediately ready to receive data.
HOLD	HOLD is used to temporarily suspend activity on the CMI.
WAIT	WAIT is asserted by a subsystem to initiate a processor interrupt. It is held until a write vector operation is performed. **NOTE** **CMI data signals are asserted at +3 V (high); all other signals are asserted at ground (low).**

Table 2-33 CMI Signal Description

Signal Line	Description
Priority Arbitration Group	
<ARB7:ARB1>	An ARB level is assigned to each subsystem and is used to gain control of the CMI. If a higher priority bit is not set and the CMI is idle (DBBZ and HOLD are not asserted), a subsystem asserts its own priority bit and assumes control of the CMI data lines on the following B CLK cycle. If a higher priority bit is set, the subsystem asserts its own priority bit to hold off lower priority subsystems until it gains control. Priority levels on the CMI are assigned as to the following devices: ARB 7 RDM – highest priority ARB 6 Reserved ARB 5 Reserved ARB 4 UBI (UBI 0) ARB 3 MBA 0 (or optional UBI 1) ARB 2 MBA 1 ARB 1 MBA 2 CPU – lowest priority
Status Group	
STATUS <1:0>	Status is transmitted by a slave to indicate the conditions under which data is returned to the master. Status bit combinations are defined as follows: **Status Bit** **1 0** 0 0 No response. Master attempted to access nonexistent memory (NXM) for read or write operation. 0 1 Data returned to master carries uncorrectable error (UCE). 1 0 Data is corrected. 1 1 Data has no errors.

CMI Transfer Formats – Information is transferred between subsystems on the CMI by two operations. Each operation consists of transmitting a separate format on the CMI data lnes. A master subsystem gains access to a slave by transmitting the physical longword address of the slave in the CMI address format (Figure 2-63) and asserting the DBBZ level for one B CLK cycle. A longword (four bytes of data) is then transferred to or from the slave in the CMI data format (Figure 2-64). If the slave is not immediately ready to receive write data or return status, it asserts DBBZ until it is.

Bits <01:00> of the physical longword address are not meaningful because data on the CMI is longword-aligned. The position of a byte in the CMI data longword is the effective address of the byte in relation to the physical longword address.

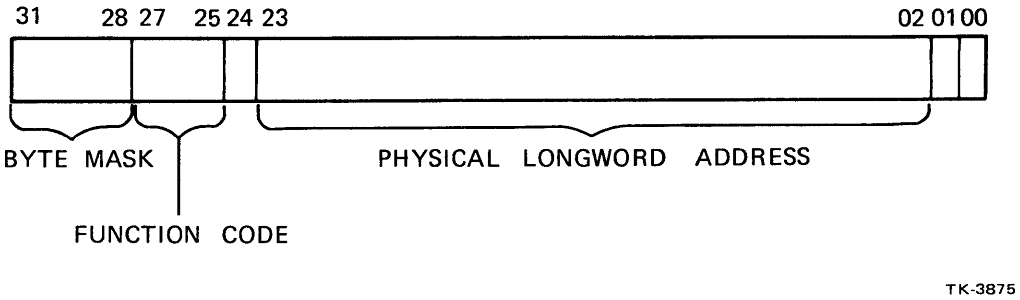


Figure 2-63 CMI Address Format

31 24	23 16	15 08	07 00
BYTE 3	BYTE 2	BYTE 1	BYTE 0

TK-3876

Figure 2-64 CMI Data Format

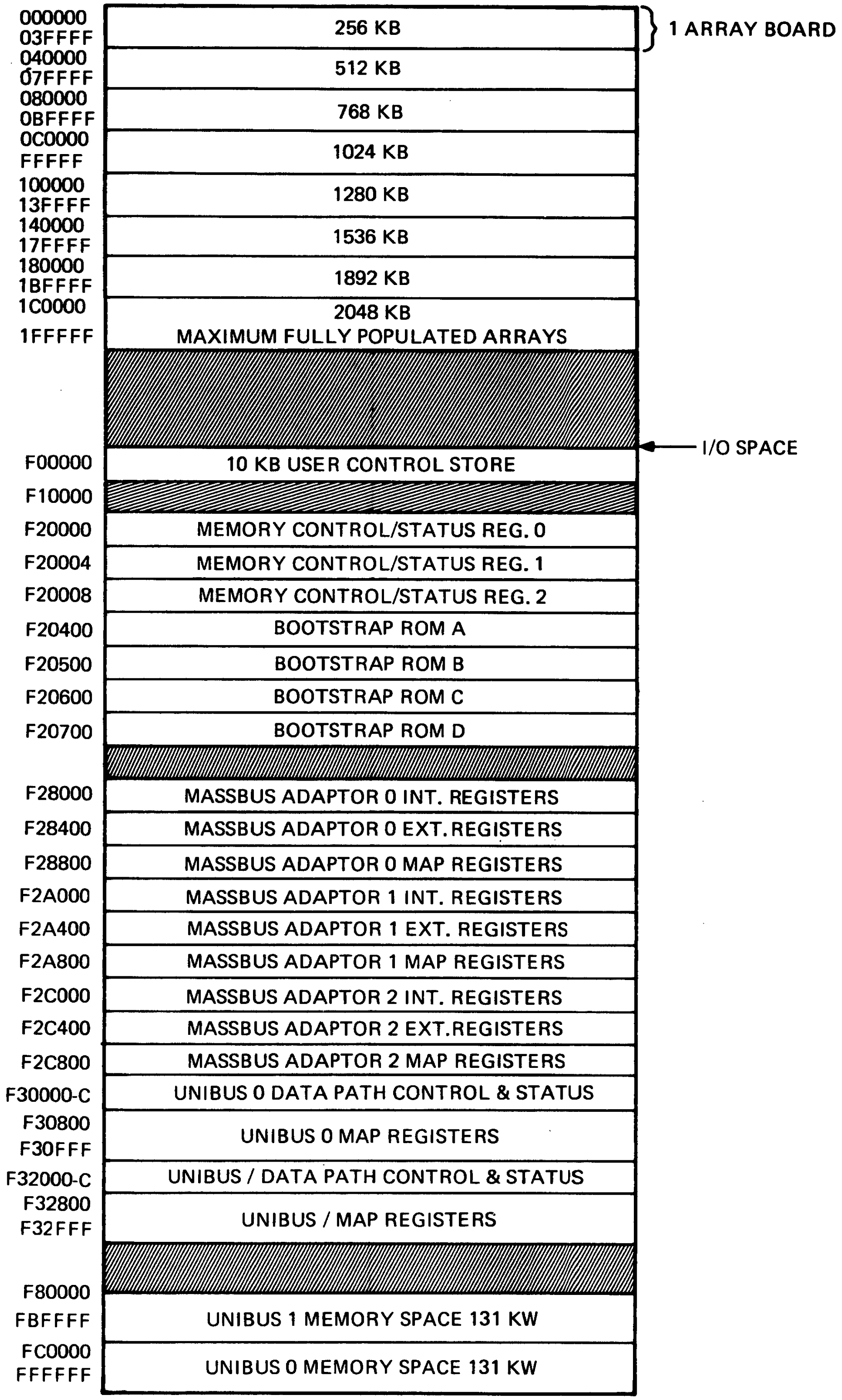
The byte mask bits of the CMI address (Figure 2-63) designate which bytes are valid for transfer.

Byte Mask Bit	Byte(s) Valid for Transfer
Bit 28	Byte 0 valid
Bit 29	Byte 1 valid
Bit 30	Byte 2 valid
Bit 31	Byte 3 valid

The function code field (Figure 2-63) designates the operation that is being performed by the master:

Function Bit			
27	26	25	CMI Operation
0	0	0	Read
0	0	1	Read Lock
0	1	0	Read with Modify Intent
0	1	1	Undefined
1	0	0	Write
1	0	1	Write Unlock
1	1	0	Write Vector
1	1	1	Undefined

CMI Physical Address Map – Figure 2-65 is a map of assigned physical address space on the CMI.



000000 03FFFF	256 KB
040000 07FFFF	512 KB
080000 0BFFFF	768 KB
0C0000 FFFFF	1024 KB
100000 13FFFF	1280 KB
140000 17FFFF	1536 KB
180000 1BFFFF	1892 KB
1C0000 1FFFFF	2048 KB MAXIMUM FULLY POPULATED ARRAYS
	▨
F00000	10 KB USER CONTROL STORE
F10000	▨
F20000	MEMORY CONTROL/STATUS REG. 0
F20004	MEMORY CONTROL/STATUS REG. 1
F20008	MEMORY CONTROL/STATUS REG. 2
F20400	BOOTSTRAP ROM A
F20500	BOOTSTRAP ROM B
F20600	BOOTSTRAP ROM C
F20700	BOOTSTRAP ROM D
	▨
F28000	MASSBUS ADAPTOR 0 INT. REGISTERS
F28400	MASSBUS ADAPTOR 0 EXT. REGISTERS
F28800	MASSBUS ADAPTOR 0 MAP REGISTERS
F2A000	MASSBUS ADAPTOR 1 INT. REGISTERS
F2A400	MASSBUS ADAPTOR 1 EXT. REGISTERS
F2A800	MASSBUS ADAPTOR 1 MAP REGISTERS
F2C000	MASSBUS ADAPTOR 2 INT. REGISTERS
F2C400	MASSBUS ADAPTOR 2 EXT.REGISTERS
F2C800	MASSBUS ADAPTOR 2 MAP REGISTERS
F30000-C	UNIBUS 0 DATA PATH CONTROL & STATUS
F30800 F30FFF	UNIBUS 0 MAP REGISTERS
F32000-C	UNIBUS / DATA PATH CONTROL & STATUS
F32800 F32FFF	UNIBUS / MAP REGISTERS
	▨
F80000 FBFFFF	UNIBUS 1 MEMORY SPACE 131 KW
FC0000 FFFFFF	UNIBUS 0 MEMORY SPACE 131 KW

TK-5814

Figure 2-65 CMI Physical Address Map

CMI Read/Write Cycles – Figure 2-66 is a timing diagram of read and write operations on the CMI. A minimum of three B CLK cycles is normally required to transfer one longword of data. These cycles are as follows.

1. Arbitration cycle (DBBZ and HOLD are not asserted, the CMI is idle).
2. CMI address cycle, CMI address and DBBZ asserted by master.
3. CMI data cycle, DBBZ asserted by slave if the slave is not ready to complete the transactions.

   a. Read cycle, slave deasserts DBBZ and returns data and status.

   b. Write cycle, slave clocks data, deasserts DBBZ and returns status.

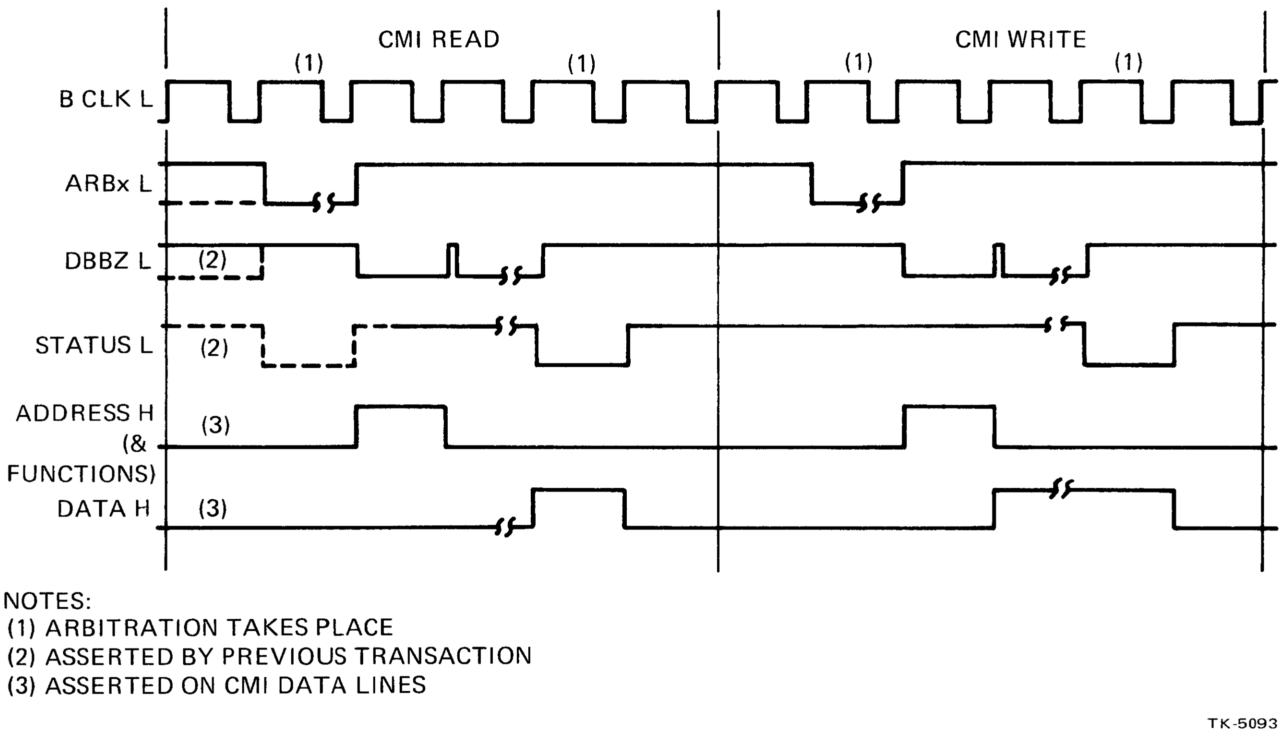


Figure 2-66 CMI Read/Write Cycles

Actual time required for a transfer varies with the ability of a slave subsystem to return data or status. If a slave is immediately ready to receive write data, it does not assert DBBZ and only two cycles are required as for the write vector function in Figure 2-67.

A subsystem may assert its arbitration level at any time. Arbitration takes place when DBBZ and HOLD are not asserted. The subsystem with the highest priority arbitration level asserted holds off lower priority subsystems. On the next positive transition of B CLK L, the new master asserts the physical longword address of the slave along with DBBZ. All other subsystems recognize that an address longword is present on the CMI and the addressed slave responds as in Figure 2-66.

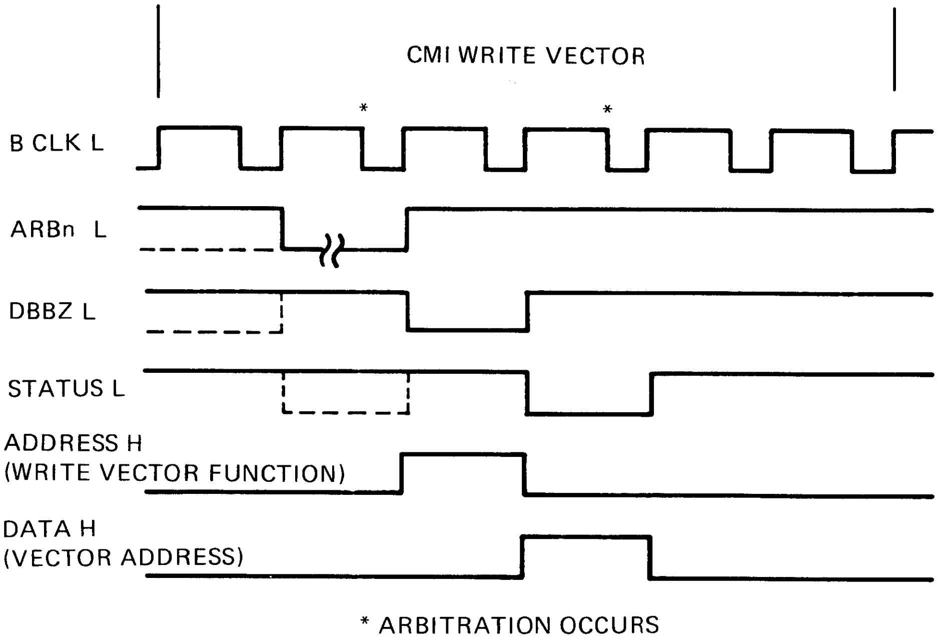


Figure 2-67 CMI Write Vector Cycle

All CMI subsystems contain a PREV DBBZ flip-flop that retains the asserted or deasserted state of DBBZ from the previous B CLK cycle. Arbitration takes place during a cycle with DBBZ not asserted and the highest priority subsystem with an arbitration level asserted wins access to the CMI. On the following cycle, the subsystem asserts a CMI address with DBBZ. The combination of the PREV DBBZ flip-flop cleared with DBBZ asserted indicates to all other subsystems that an address is present on the CMI.

Figure 2-67 illustrates a write vector cycle on the CMI generated by a Unibus or Massbus device. Function bits <27:25> of the CMI address specify the write vector function; all other bits are not meaningful. The vector address is asserted during the CMI data cycle. Typical response to a bus request (BR interrupt) is as follows.

BR Interrupt – A BR priority level generated by an I/O device is latched by the M CLK signal and asserted as the appropriate SBR level to the INT chip in the UBI. The INT chip compares the SBR <7:4> level to an IPL <17:14> level. When the SBR is higher than the current processor IPL, the following occurs.

1. INT PEND signal is updated at each trailing edge of M CLK and sent to the DPM and MIC modules.

2. INT chip selects MICROVECTOR <2:0> lines to identify the type of interrupt pending. The value is 2 for a Unibus-originated interrupt.

INT PEND is used by the CPU to generate remaining MICROVECTOR <5:3> lines to select the microvector address that services the incoming interrupt.

1. INT PEND is received by the SAC chip on the DPM while macrocode is running, but is not interpreted for one microinstruction following an IRD 1 cycle.

2. The SAC chip generates the DO SERVICE and ENABLE microvector signals to the MSQ chip which selects MICROVECTOR <5:4> bits.

3. DO SERVICE to the UTR chip on the MIC selects MICROVECTOR <3> bit.

Selected MICROVECTOR bits <5:3> with bits <2:0> from the UBI direct the CCS to the interrupt handling microroutine. The first function of the microroutine is to send a 33 (hex) on the WCRTL <5:0> lines to the INT logic, which enables the bus grant (BGn) level to be returned to the device. UB INT GRANT is also sent to the CMK chip on the MIC module. The CMK chip generates GRANT STALL, which stalls the CPU microcode until the vector is written to the MIC module or WAIT is deasserted.

When SACK is returned by the requesting device, WAIT is asserted on the CMI. WAIT is received by the MIC module and replaces UB INT GRANT to hold the CPU stalled. When the device can assert BBSY and the vector address, it then asserts INTR which holds WAIT asserted on the CMI to maintain the CPU in the stalled state. The INTR level then directs the UBI to perform a write vector operation on the CMI. Two B CLK cycles are required for a write vector on the CMI. DBBZ is not asserted by the CPU, the vector address is clocked directly, and status is returned.

Passive Release – The interrupt/write vector operation described above constitutes an active release of the Unibus device since the write vector operation was completed normally. A passive release is a condition caused by a device that raises a BR level and then, because of a malfunction or because of software or hardware limitations, loses it. If the BR level is lost after being synchronized by the arbitrator, BUS GRANT is asserted and held to await the return of SACK. A NO SACK timeout normally causes the arbitrator to assert SACK in order to release the bus grant level.

In order to prevent a passive release from holding the processor in a stall for the duration of the SACK timeout delay, a method is provided to release the CCS from the stall. With no requesting level present while in the interrupt service microroutine, the INT chip can interpret the requesting level as lower than the current IPL. The bus grant enable flip-flop is set for one B CLK cycle (fake grant), releasing the stall when it deasserts. Since a BR level is no longer asserted, no grant is issued to the Unibus.

BR Data Transfer – Some devices are designed to transfer data under the authority of a BR request. BR arbitration takes place as usual with one exception: once the device asserts BBSY, it then asserts address and data as it would for an NPR, asserting MSYN instead of INTR. A UBI microsequence is selected as for an NPR to process the transaction.

2.5.9 MIC Functions and Controls

Memory is read by the MIC under a read bus function (microcode-dependent) or by a prefetch cycle (independent of the microcode). Memory is only written by microcode under direction of a write bus function.

Since microcode functions use common circuitry, they are performed in logical sequence. A macroinstruction (machine instruction) is executed in microcoded steps, each step consisting of a single microinstruction.

During execution of the macroinstruction, MIC control logic monitors operating conditions that could cause a trap out of the main microcode sequence (a microtrap). When a microtrap condition is encountered, MIC control logic alters the CCS microaddress. This redirects the microsequencer to the routine that handles the condition and pushes the base return microaddress onto the microstack.

Some chip functions are decoded directly from the microinstruction. Others include conditions that are monitored on the input lines to the chips. Interaction between the chips synchronizes the non-microcoded with the microcoded functions. A pending function is held off while a prior function completes. In some cases, as for microtraps, functions and events occur in priority sequence. (CCS bus function bit <4>, when set, holds off prefetch cycles.)

Bus Cycle – All data transfer bus functions, including bus grant, decode a transfer sequence within the CMK chip called a bus cycle. A bus cycle begins with assertion of the ADD REG ENA signal and ends with the assertion of STATUS VALID from the CMK. It may or may not include a CMI read or write cycle to memory. A read bus function that results in a cache hit, for example, does not require a CMI read cycle to main memory. A bus cycle is generated by the following bus function codes.

Read Physical Address
Read, No Microtrap
Read Lock Timeout Test
Read, Second Reference
Write Physical Address
Write, Second Reference
Write Unlock, Second Reference
Write, No Microtrap
Write Longword, No Microtrap
Bus Grant
Read
Read Longword
Read Lock
Read with Modify Intent
Read Longword with Modify Intent
Write
Write Longword
Write Unlock

MIC Control Logic – Figure 2-68 is a block diagram of MIC data and address paths and registers. The six control chips described below work together to monitor and respond to operational conditions. Timing is provided by B CLK L. Under direction of the bus function, WCTRL, and MSRC fields of the microcode, the chips provide clocking, gating, and multiplexer selection for MIC operation. Major functions of the chips are as follows.

PRK – Prefetch Control chip, independent of the microcode, generates the prefetch function to memory for I-Stream data. PRK keeps track of machine I-Stream cycles and controls some ADD section gating in conjunction with the ADK chip.

CMK – CMI Control chip, in conjunction with the MDR, transmits and receives CMI control signals DBBZ, HOLD, and STATUS <1:0> bits. It drives the byte mask and function code for CPU access to the CMI, monitors CMI cycles for writes to memory by an I/O device, and initiates the snapshot CMI function. It generates the corrected data interrupt. In response to a Unibus or Massbus interrupt, it asserts the grant stall to the microcode during the CMI write vector operation.

ADK – Address Control chip drives multiplexer gating of the ADD and MDR sections for address manipulation. It controls write data inputs, physical address outputs, and group disables for the TB, and contains memory status/control registers and gating.

CAK – Cache Control chip contains cache status/control registers. It enables and disables writes to cache, controls data transfers between the MDR section and cache, and drives MDR rotator multiplexer for cache or CMI data alignment to the data paths. It monitors the snapshot CMI function from the CMK to check for cache hits by CMI I/O writes to memory.

ACV – Access Control Violation chip encodes microtrap conditions to UTR in priority sequence for these conditions.

CCS parity error
FPA reserved operand
Reserved
Write crossing page boundary
Write unlock crossing page boundary
Unaligned data, write unlock
Unaligned data

It generates the ACV signal for access violations, and the translation not valid signal on TB references or on PTE checks and probes from the WBus.

UTR – Microtrap Generator monitors microtrap conditions for microinstruction errors or violations from the ACV, TB misses or TB parity errors. For microtraps, the UTR encodes and asserts microvector bits <3:0>, shutting them off from the MSQ chip in the DPM. These bits are used in conjunction with bits <5:4> from the MSQ to specify the six low-order bits of the microaddress. The resulting address points to the microroutine that services the microtrap condition detected by the UTR. The UTR monitors CMI status from the CMK chip and generates the write bus error interrupt to the INT chip on the UBI module.

Common Input Signals – A number of signals are common to MIC control chip inputs. MIC timing and synchronization is obtained from the DPM via these signals.

B CLK L – The basic timing clock used throughout the processor.

M CLK ENABLE – Deasserted to provide a stall to the microsequencer.

D CLK ENABLE – Deasserted on some errors and microtraps to prevent clocking bad or incomplete data.

PHASE 1 – Provides two event times to execute a microinstruction. PHASE 1 asserted is the first event; PHASE 2 (PHASE 1 deasserted) is the second. M CLK and D CLK occur at half the rate of B CLK. Phase 1 is synchronized with the assertion of M CLK. The MIC module latches control store bus function <4:0>, MSRC <4:0>, and WCTRL <5:0> fields. The registers are clocked at M CLK time. Bus function bit <4> is also connected to a flip-flop within the ADK, CMK, PRK, and ACV chips, where it is examined prior to M CLK time. This allows these chips to determine in advance the type of upcoming bus function and holds off the prefetch cycle.

WBUS <27:24> lines are bidirectional. They are used for reading/writing bits <3:0> of MIC memory status/control registers discussed in Paragraph 2.5.5. Activity to these registers takes place under direction of the WCTRL field.

DST RMODE – Destination Register Mode from the DPM indicates to the CMK, ADK, and CAK that the destination register designated by the operand specifier is a GPR. Any write bus function decoded from the bus field is inhibited. The PRK is signaled, however, that it may generate a CMI cycle for a prefetch.

PSL CM – Compatability Mode bit of the processor status longword from the DPM causes the ADK to force MA MUX bits <31:16> to zeros in the ADD section as described in Paragraph 2.5.2.1. PSL CM forces the PRK to invalidate any prefetched I-Stream information for all writes. Compatability mode allows writes into the I-Stream directly ahead, and allows the modified instructions to be executed.

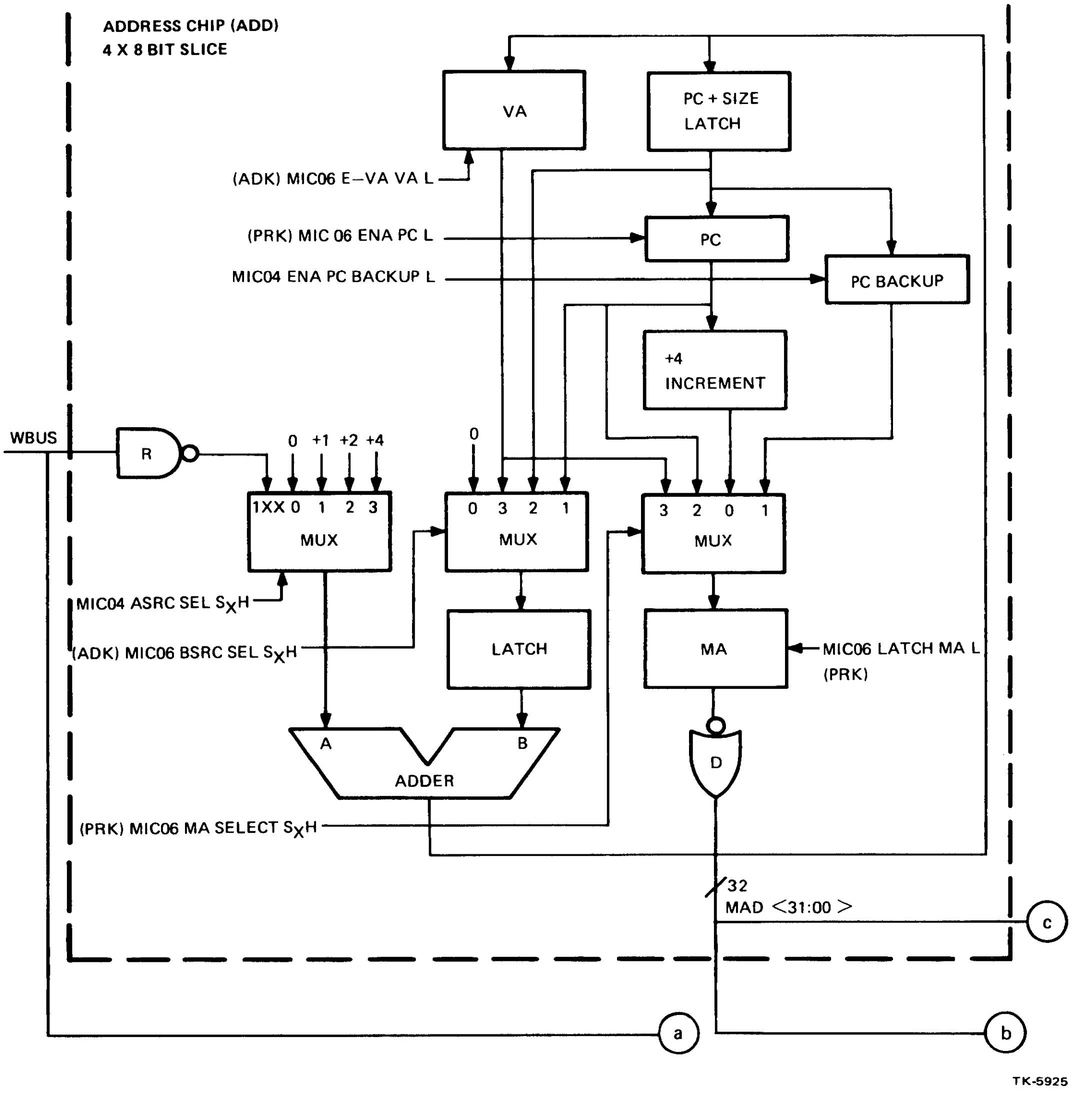


Figure 2-68 MIC Block Diagram
(Sheet 1 of 4)

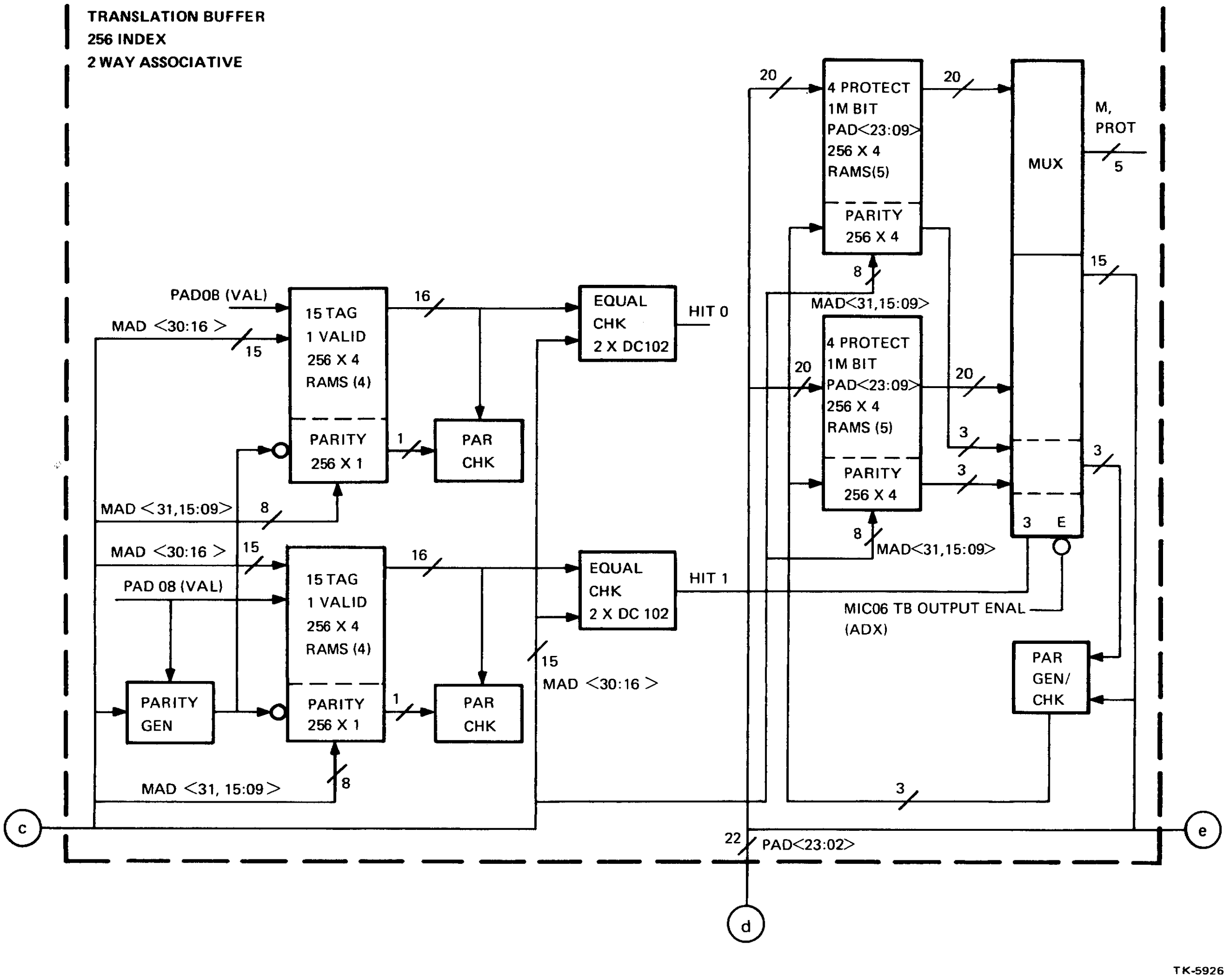


Figure 2-68 MIC Block Diagram
(Sheet 2 of 4)

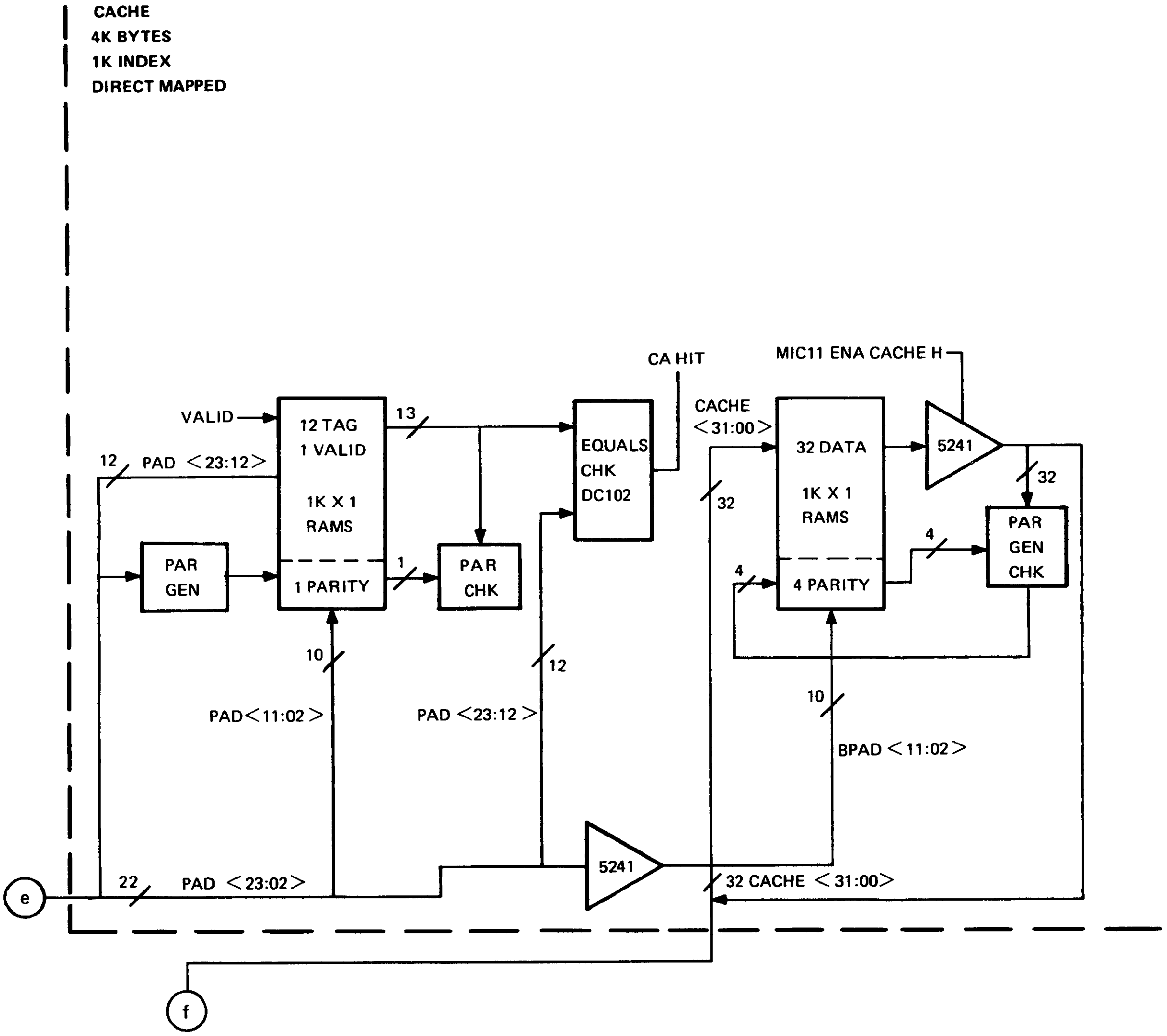
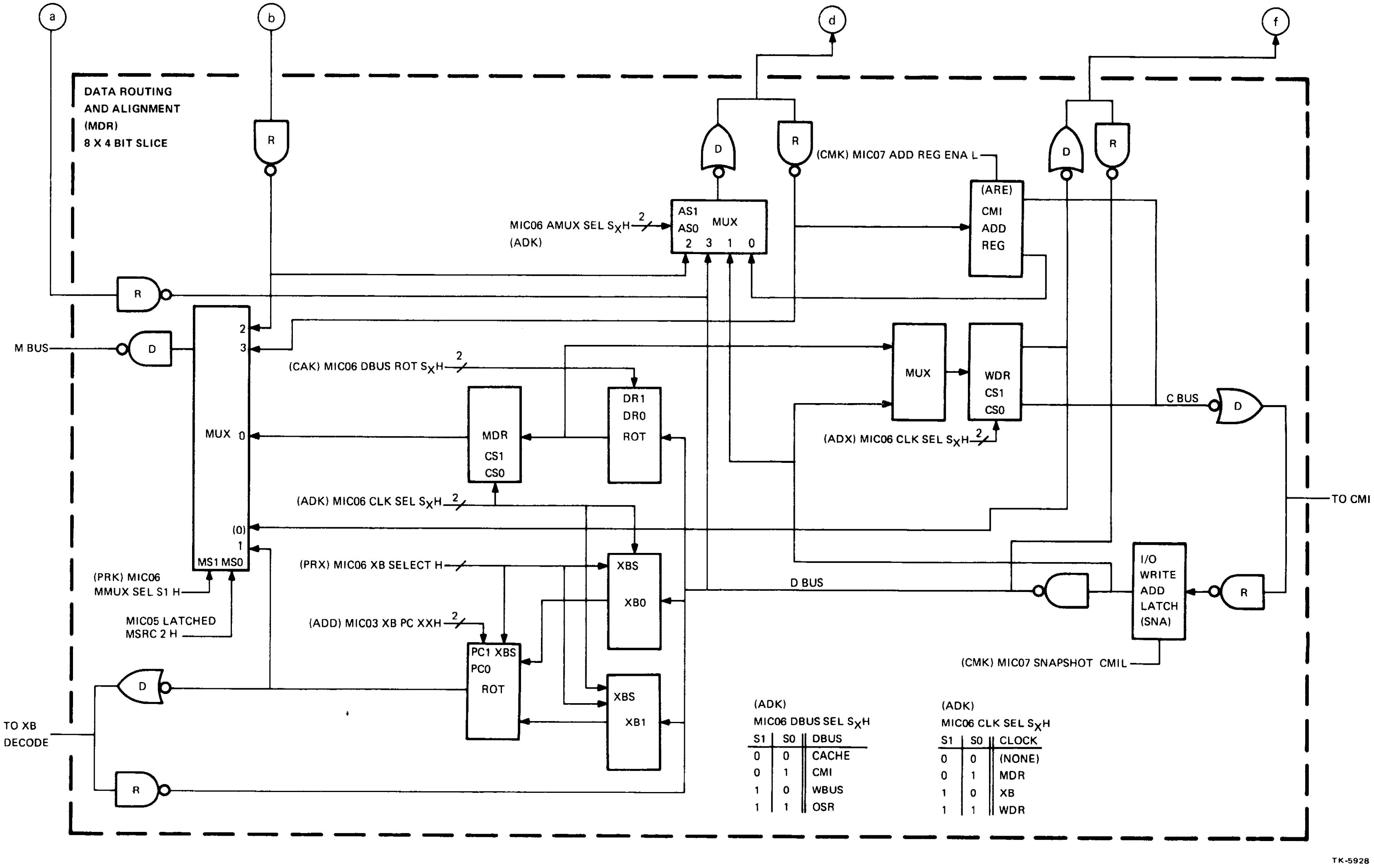


Figure 2-68 MIC Block Diagram
(Sheet 3 of 4)



(ADK)
MIC06 DBUS SEL S_XH

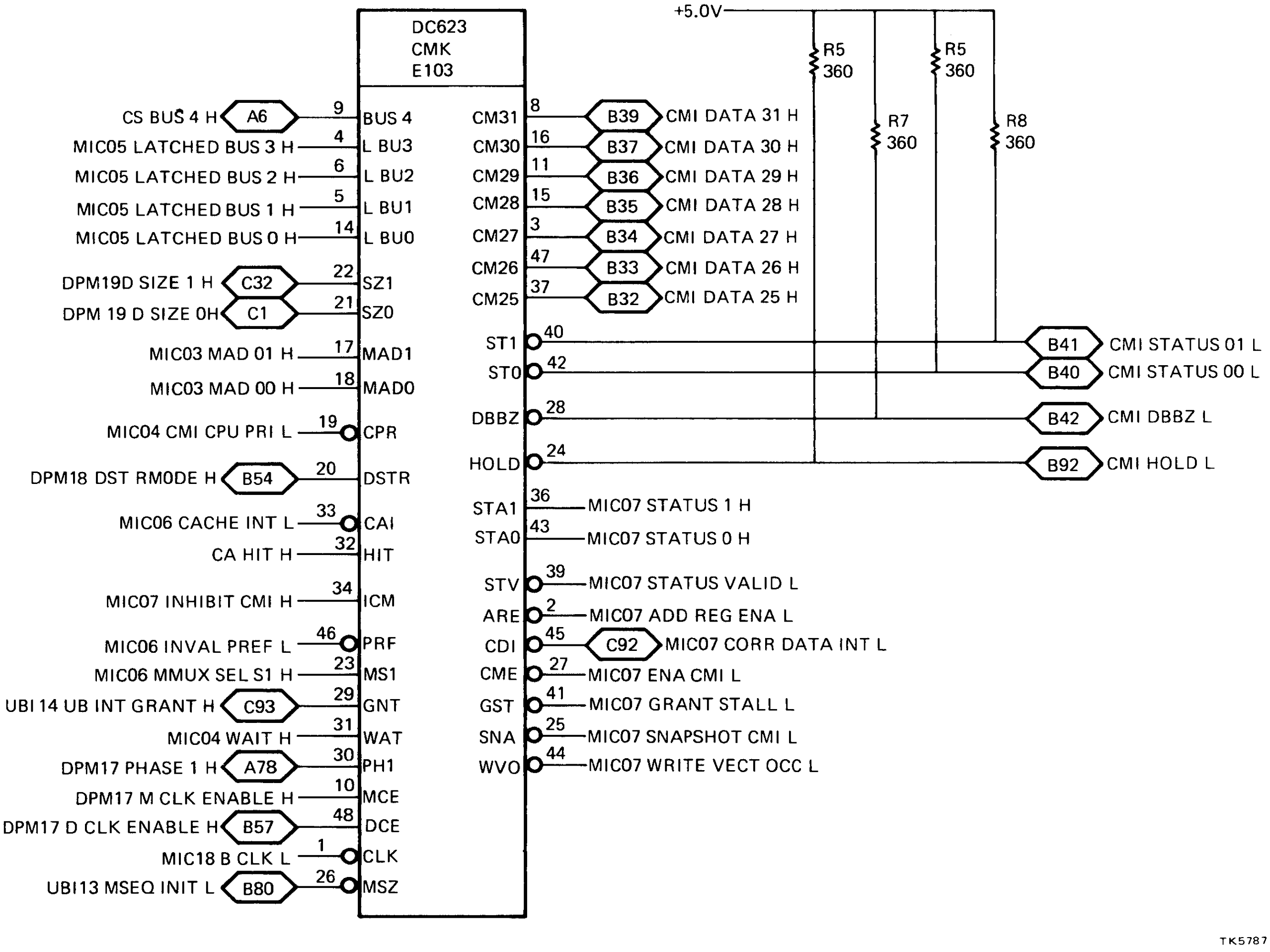
S1	S0	DBUS
0	0	CACHE
0	1	CMI
1	0	WBUS
1	1	OSR

(ADK)
MIC06 CLK SEL S_XH

S1	S0	CLOCK
0	0	(NONE)
0	1	MDR
1	0	XB
1	1	WDR

Figure 2-68 MIC Block Diagram
(Sheet 4 of 4)

2.5.9.1 CMI Control (CMK) – The CMK, Figure 2-69, monitors bus functions and responds to those that generate bus cycles. For the prefetch function and for bus cycles that require access to the CMI, the CMK initiates the CMI read or write cycles described in Paragraph 2.5.8. It generates the byte mask and function code fields of the CMI address shown in Figure 2-63 and asserts the DBBZ signal.



DC623
CMK
E103
+5.0V
R5 360
R5 360
R7 360
R8 360
CS BUS 4 H A6 9 BUS 4
MIC05 LATCHED BUS 3 H 4 L BU3
MIC05 LATCHED BUS 2 H 6 L BU2
MIC05 LATCHED BUS 1 H 5 L BU1
MIC05 LATCHED BUS 0 H 14 L BU0
DPM19D SIZE 1 H C32 22 SZ1
DPM 19 D SIZE 0H C1 21 SZ0
MIC03 MAD 01 H 17 MAD1
MIC03 MAD 00 H 18 MAD0
MIC04 CMI CPU PRI L 19 CPR
DPM18 DST RMODE H B54 20 DSTR
MIC06 CACHE INT L 33 CAI
CA HIT H 32 HIT
MIC07 INHIBIT CMI H 34 ICM
MIC06 INVAL PREF L 46 PRF
MIC06 MMUX SEL S1 H 23 MS1
UBI 14 UB INT GRANT H C93 29 GNT
MIC04 WAIT H 31 WAT
DPM17 PHASE 1 H A78 30 PH1
DPM17 M CLK ENABLE H 10 MCE
DPM17 D CLK ENABLE H B57 48 DCE
MIC18 B CLK L 1 CLK
UBI13 MSEQ INIT L B80 26 MSZ
CM31 8 B39 CMI DATA 31 H
CM30 16 B37 CMI DATA 30 H
CM29 11 B36 CMI DATA 29 H
CM28 15 B35 CMI DATA 28 H
CM27 3 B34 CMI DATA 27 H
CM26 47 B33 CMI DATA 26 H
CM25 37 B32 CMI DATA 25 H
ST1 40 B41 CMI STATUS 01 L
ST0 42 B40 CMI STATUS 00 L
DBBZ 28 B42 CMI DBBZ L
HOLD 24 B92 CMI HOLD L
STA1 36 MIC07 STATUS 1 H
STA0 43 MIC07 STATUS 0 H
STV 39 MIC07 STATUS VALID L
ARE 2 MIC07 ADD REG ENA L
CDI 45 C92 MIC07 CORR DATA INT L
CME 27 MIC07 ENA CMI L
GST 41 MIC07 GRANT STALL L
SNA 25 MIC07 SNAPSHOT CMI L
WVO 44 MIC07 WRITE VECT OCC L
TK5787

Figure 2-69 CMI Control CMK

The CMK monitors CMI signals and does the following.

Generates the snapshot CMI function during I/O writes on the CMI to invalidate cache
Starts the read lock timer when it detects a read lock function
Responds as slave to a write vector

CMK functions and signals are as follows.

CMI DATA <31:28> – These lines are transmitted only to drive the byte mask field of the CMI address shown in Figure 2-63. They are asserted as ones during prefetch cycles and for the following bus functions.

Read Physical Address
Read, Second Reference
Bus Grant
Write Longword, No Microtrap
Write Physical Address
Write Longword
Read Longword
Read Longword with Modify Intent

For all other cases, CMI DATA <31:28> are produced as shown in the following three charts, encoded by MAD <1:0> and D-Size <1:0> combinations.

For all other Reads:

MAD <1:0>			
00	**01**	**10**	**11**
1111	1110	1100	1000

For all other Writes except write, second reference and write Unlock, Second Reference:

D-Size	**MAD <1:0>**			
<1:0>	**00**	**01**	**10**	**11**
00	0001	0010	0100	1000
01	0011	0110	1100	1000
10	1111	1110	1100	1000
11	1111	1110	1100	1000

For Write, Second Reference and Write Unlock, Second Reference:

D-Size	**MAD <1:0>**			
<1:0>	**00**	**01**	**10**	**11**
00	0001	0001	0001	0001
01	0001	0001	0001	0001
10	0001	0001	0011	0111
11	0001	0001	0011	0111

The byte mask bits are generated by the CMI master to indicate which bytes of the CMI data longword are valid for transfer.

Byte Mask Bit	**Byte(s) Valid for Transfer**
Bit 28	Byte 0, bits <7:0> of the CMI data longword
Bit 29	Byte 1, bits <15:08>
Bit 30	Byte 2, bits <23:16>
Bit 31	Byte 3, bits <31:24>

CMI DATA <27:25> – These bidirectional lines drive and monitor the function code field of the CMI address shown in Figure 2-63. For CPU functions they are asserted as zeros (read) during prefetch cycles; or are asserted as follows for the indicated bus function.

Function Bit			
27	**26**	**25**	**Bus Function**
0	0	0	Read
0	0	1	Read Lock
0	1	0	Read with Modify Intent
0	1	1	Undefined
1	0	0	Write
1	0	1	Write Unlock
1	1	0	Write Vector not generated by CPU
1	1	1	Undefined

CMI STATUS <01:00> – These bidirectional lines are driven by the CMK to return no error status during a write vector operation by an I/O device. They are driven by the slave (memory or I/O device) to indicate the conditions under which data is returned to the CPU (master).

CMI Status		
01	**00**	**Error Status**
0	0	NXM – CPU attempted access to nonexistent memory. (Read or Write)
0	1	UCE – Uncorrectable data
1	0	Corrected data
1	1	No error

CMI DBBA – DBBZ is asserted by the CMK during a CPU-initiated CMI address cycle for a prefetch or one of the read/write bus functions.

SNAPSHOT CMI/CMI HOLD – The CMK contains a flip-flop that retains the state of DBBZ from the previous B CLK cycle. A combination of this flip-flop cleared (DBBZ was not asserted on the previous B CLK cycle), with DBBZ now asserted, indicates that an I/O device has a CMI address asserted on the CMI data lines. If the function code field of the CMI address is a write or write unlock, The CMK asserts CMI HOLD to prevent additional CMI activity. It asserts snapshot CMI to the MDR section, and to the ADK, CAK, and PRK chips. The physical address field, bits <23:02> of the CMI address, is passed through the MDR section to the PAD lines to access cache. If a cache hit results (CA HIT), the cache location is invalidated.

ADD REG ENA – Address register enable causes the CMI address to be latched in the CMI address register of the MDR. The CMI address register is also loaded with CPU-generated addresses for access to cache, or for transmission with the CMI address, to access main memory or I/O.

STATUS VALID – This indicates to the ADK, CAK, PRK, and UTR chips the end of the current bus cycle. Received CMI STATUS <01:00> is clocked to the STATUS <1:0> flip-flops in the CMK.

STATUS <1:0> – Received CMI STATUS <01:00> are clocked to the UTR chip during Status Valid at the end of every bus cycle.

ENA CMI – Enable CMI, to the MDR, allows CMI address, byte mask, and function code to be transmitted on the CMI with DBBZ to initiate a CMI cycle. The CA HIT (cache hit) signal causes ENA CMI to be deasserted to prevent a CMI cycle. ENA CMI is also deasserted by Inhibit CMI from the UTR when a microtrap condition is encountered during a TB function.

CORR DATA INT – Corrected Data Interrupt is sent to the INT chip on the UBI when Corrected Data status is received from the CMI during a bus function. Software may use this feature to cause a macro-level interrupt.

GRANT STALL, UB INT GRANT, WAIT – During a BR interrupt, the bus grant bus function directs the UBI to issue BGx to the highest requesting device. It also asserts Grant Stall and stops the microcode. UB INT GRANT received from the INT chip on the UBI indicates that the BRx level was still asserted and BGx was issued to the device. The release is considered active and the microcode is held stalled. The wait level, which was asserted by the UBI when it received the bus grant, releases UB INT GRANT and holds the microcode stalled until the vector address is clocked into the MDR.

In the event of a passive release, Wait is not asserted since no BGx is issued. The INT chip now interprets the requesting level as lower than the current IPL. The UB INT GRANT flip-flop is set for one B CLK cycle, unstalling the M CLK upon its release. Otherwise, the microcode would remain stalled until the NO SACK timeout occurs on the UBI.

WRITE VECT OCC – The Write Vector Occurred bit of IPR 17 is set when the vector address is clocked to the MDR (WAIT is deasserted). Checked by the microcode when M CLK is unstalled, it indicates whether the release was active or passive.

It is also set after 64 μs as the read lock timeout status bit if the CPU attempts to access the CMI during a read lock condition. It also causes nonexistent memory status to be transmitted to the UTR to initiate a bus error machine check.

INHIBIT CMI – From the UTR, this signal inhibits CMK bus cycle access to the CMI for certain microtrap conditions such as access violations.

CMI CPU PRI – This tells the CMK that no subsystem is arbitrating for the CMI and the CMK may have access.

2.5.9.2 Address Control (ADK) – The ADK, Figure 2-70, controls multiplexer selection and register clocking of the ADD and MDR logic as described in Paragraphs 2.5.2 and 2.5.3. It contains memory status/control registers described in Paragrapph 2.5.6 (MEMSCARs 0, 1, 2, and 3), and their associated read/write gating. The following paragraphs describe ADK signals and functions.

TB HIT <1,0> are driven from the <1,0> bits of the TB GDR (Figure 2-59 in Paragraph 2.5.6). They are wire-ORed to their corresponding outputs of the TB equality circuits to force TB misses for either group when set.

A MUX SEL <1,0> select MDR address multiplexer (A MUX) inputs to the PAD bus as shown in Table 2-26, Paragraph 2.5.3.1.

B SRC SEL <1,0> select B MUX inputs to the B side of the ADD adder as shown in Table 2-24, Paragraph 2.5.2.4.

CLK SEL <S1,S0> control clocking of the DBus destination registers as shown in Table 2-30, Paragraph 2.5.3.2.

DBUS SEL <S1,S0> enable source drivers to the DBus as shown in Table 2-30.

ENA VA is asserted to the ADD section to allow the SUM output of the address to be clocked to the VA register. ENA VA is selected by WCTRL field codes.

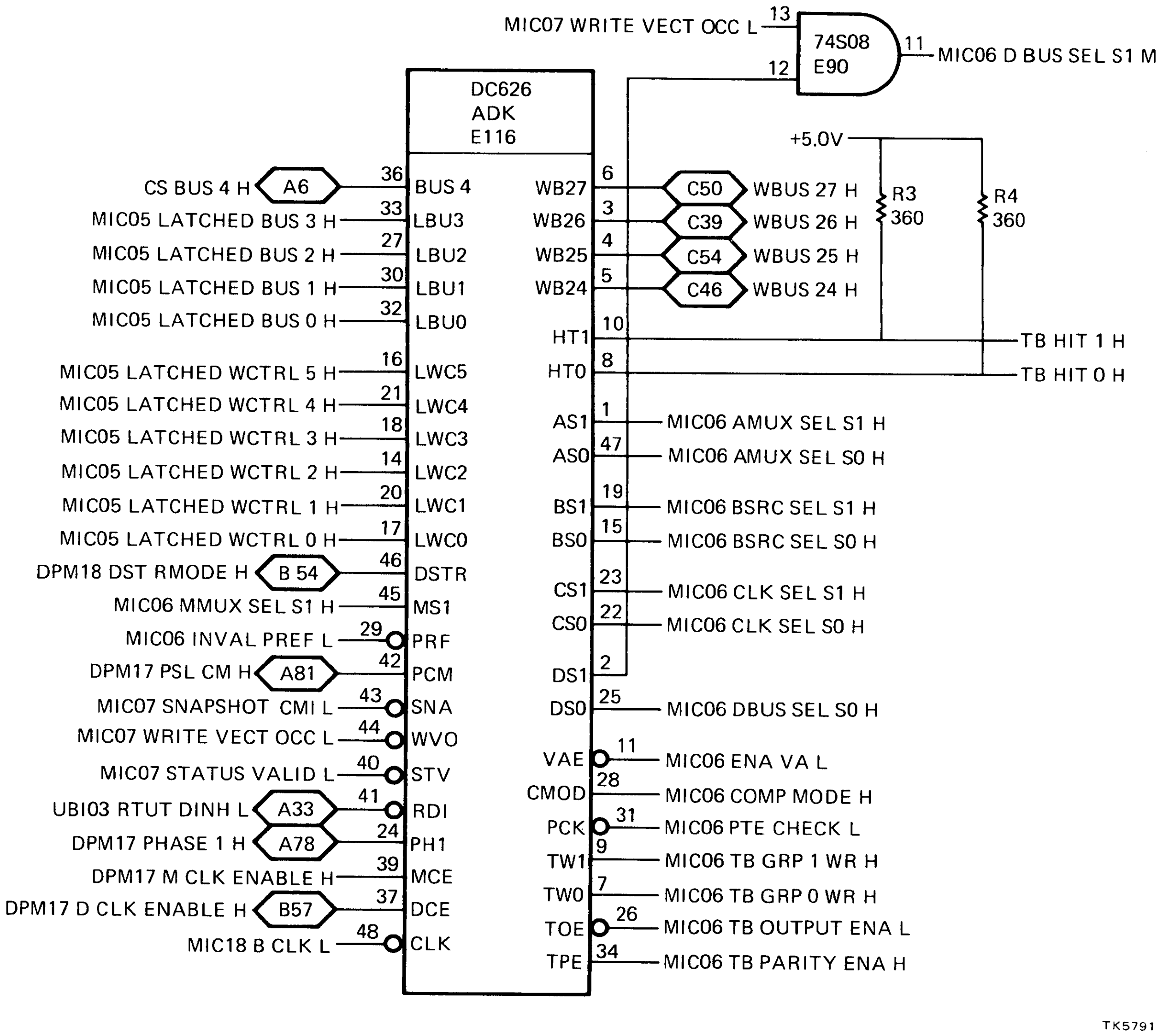


Figure 2-70 Address Control (ADK)

COMP MODE (compatibility mode) level is generated by monitoring the CM bit of the PSL and the prefetch or WCTRL lines for partial control of the ADD section MA MUX as described in Paragraph 2.5.2.1.

PTE CHECK is generated by WCTRL codes during functions other than prefetch, is used by the TB control to output valid bit, M bit, and access protection bits.

TB GRP <1,0> WR are generated by WCTRL codes while monitoring TB HIT <1,0> bits and bit <2> of the TB GDR (Figure 2-59, Paragraph 2.5.6). It is used to enable writes to TB tag and data stores.

TB OUTPUT ENA is used when MME is set to assert physical address (PTE) from the TB data store onto the PAD bus.

TB PARITY ENA is used when MME is set to enable monitor of TAG 0 and TAG 1 parity. It is enabled during prefetch or any bus function except a bus grant, or read or write physical.

WRITE VECT OCC – Write Vector Occurred from the CMK is the status bit set as a result of a completed write vector operation or because of a read lock timeout.

SNAPSHOT CMI – From the CMK, this directs the ADK to source the I/O generated CMI address through the MDR section to access cache.

2.5.9.3 Cache Control (CAK) – The CAK (Figure 2-71) contains the cache status control registers described in Figure 2-60 of Paragraph 2.5.6 (MEMSCARs 4 and 6). CAK signals and functions are as follows

CA HIT – Cache Hit from the cache equality logic controls ENA BYTE <3:0>, CACHE GRP 0 WR, CACHE VALID, and CACHE INT outputs.

ENA BYTE <3:0> – The cache byte <3:0> enable levels control writes to specific bytes on cache replacement functions.

DBUS R07 <S1:S0> – DBus rotator select <S1:S0> bits control rotation of DBus data as described in Paragraph 2.5.3.2, Tables 2-27 and 2-28.

CACHE GRP 0 WR – This signal controls writes to cache tag and data stores for replacement or invalidation.

CACHE VALID – Cache Valid is input to the cache tag store to write valid or invalid status to the selected cache location.

CACHE INT – Cache interrupt signals the CMK that the cache tag location generated a parity error on a hit. This generates Bus Error with uncorrectable data status.

STATUS VALID – Status Valid from the CMK develops Cache Valid, Cache GRP 0 Write, as it ends the memory write bus function.

SNAPSHOT CMI – Snapshot CMI from the CMK sets up CAK outputs to invalidate cache location at CMI-specified address.

M MUX SEL S1 – M MUX select <S1> from the PRK disables cache invalidation until the data path used for the snapshot CMI is free.

I/O ADDRESS – I/O Address disables writes to cache when an I/O device address is decoded.

MAD <01:00>, D-SIZE <1:0> – These signals decode to assert ENA BYTE <3:0> outputs during write to cache. ENA BYTE <3:0> results are equivalent to the charts for CMI DATA <31:28> defined for the CMK, Paragraph 2.5.9.1.

CA TAG PAR ERR, CA DATA PAR ERR – These parity error bits from the cache tag and data stores develop Cache INT to the CMK.

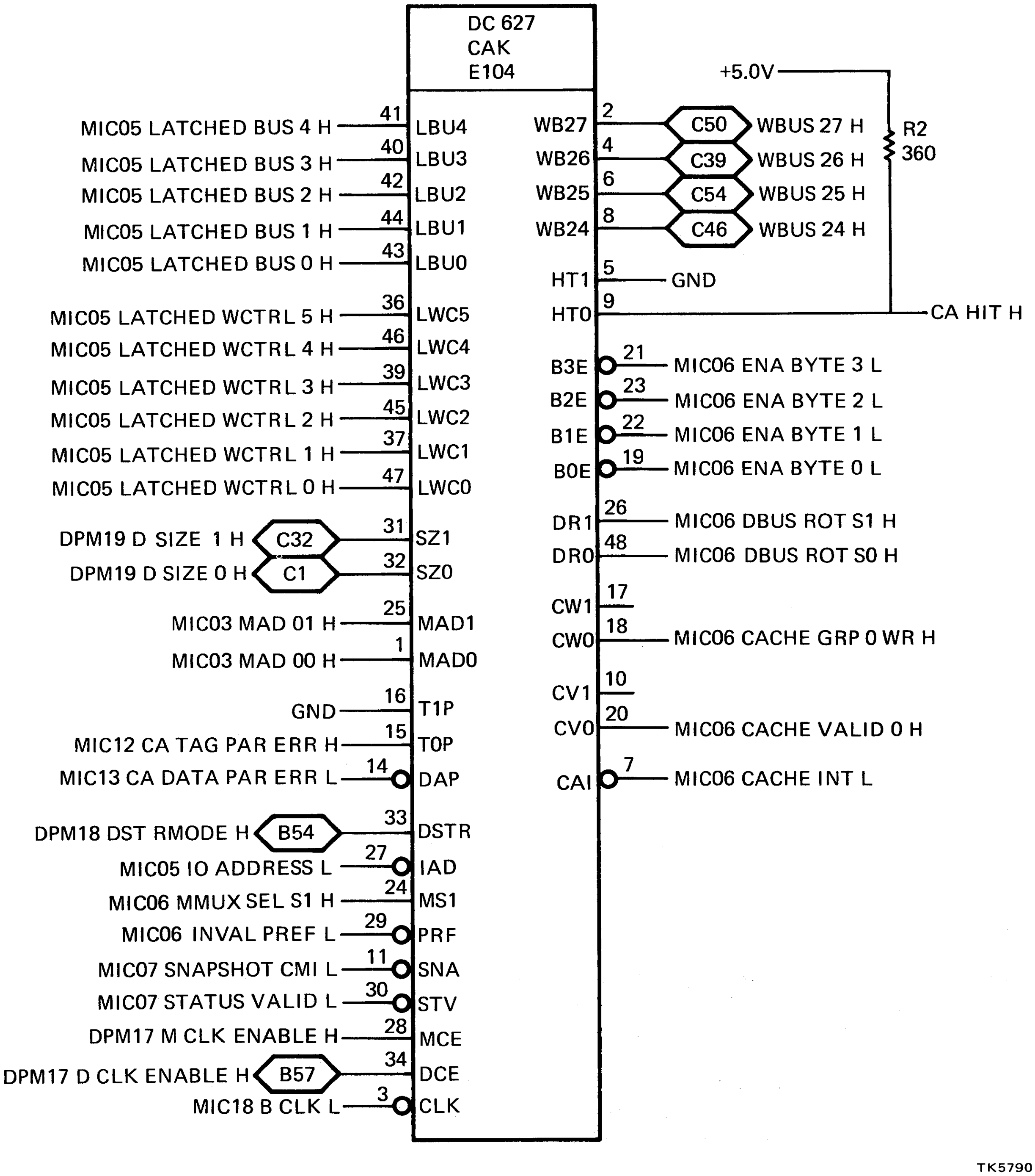

DC 627
CAK
E104
MIC05 LATCHED BUS 4 H 41 LBU4
MIC05 LATCHED BUS 3 H 40 LBU3
MIC05 LATCHED BUS 2 H 42 LBU2
MIC05 LATCHED BUS 1 H 44 LBU1
MIC05 LATCHED BUS 0 H 43 LBU0
MIC05 LATCHED WCTRL 5 H 36 LWC5
MIC05 LATCHED WCTRL 4 H 46 LWC4
MIC05 LATCHED WCTRL 3 H 39 LWC3
MIC05 LATCHED WCTRL 2 H 45 LWC2
MIC05 LATCHED WCTRL 1 H 37 LWC1
MIC05 LATCHED WCTRL 0 H 47 LWC0
DPM19 D SIZE 1 H C32 31 SZ1
DPM19 D SIZE 0 H C1 32 SZ0
MIC03 MAD 01 H 25 MAD1
MIC03 MAD 00 H 1 MAD0
GND 16 T1P
MIC12 CA TAG PAR ERR H 15 T0P
MIC13 CA DATA PAR ERR L 14 DAP
DPM18 DST RMODE H B54 33 DSTR
MIC05 IO ADDRESS L 27 IAD
MIC06 MMUX SEL S1 H 24 MS1
MIC06 INVAL PREF L 29 PRF
MIC07 SNAPSHOT CMI L 11 SNA
MIC07 STATUS VALID L 30 STV
DPM17 M CLK ENABLE H 28 MCE
DPM17 D CLK ENABLE H B57 34 DCE
MIC18 B CLK L 3 CLK
+5.0V
R2
360
WB27 2 C50 WBUS 27 H
WB26 4 C39 WBUS 26 H
WB25 6 C54 WBUS 25 H
WB24 8 C46 WBUS 24 H
HT1 5 GND
HT0 9 CA HIT H
B3E 21 MIC06 ENA BYTE 3 L
B2E 23 MIC06 ENA BYTE 2 L
B1E 22 MIC06 ENA BYTE 1 L
B0E 19 MIC06 ENA BYTE 0 L
DR1 26 MIC06 DBUS ROT S1 H
DR0 48 MIC06 DBUS ROT S0 H
CW1 17
CW0 18 MIC06 CACHE GRP 0 WR H
CV1 10
CV0 20 MIC06 CACHE VALID 0 H
CAI 7 MIC06 CACHE INT L
TK5790

Figure 2-71 Cache Control (CAK)

2.5.9.4 Prefetch Control (PRK)

Prefetch is a hardware operation controlled by the PRK chip, Figure 2-72. Independent from the microcode, the PRK initiates a CMI read cycle to memory for I-Stream data the program is most likely to need. The PRK maintains the I-Stream data in two longword execution buffer registers, XB0 and XB1, as determined by the PC. The PRK monitors these registers, and when the contents of one have been used by the program, it attempts to reload it. The prefetch operation is conducted as follows.

1. The PRK determines use of I-Stream data by monitoring the MSRC field of the microcoode, and the LD OSR and IRD 1 signals in conjunction with PC bits XB PC <01:00>.

2. The PRK monitors instruction size (ISIZE <1:0>) for steering the data upon retrieval.

3. It monitors WCTRL and bus functions to determine when the circuitry is available for the prefetch.

4. The WCTRL field is also monitored for direct loading of the PC. This generates a flush of the XB registers by prefetching two longwords of data at the new address.

   It monitors for the write bus function in compatability mode, which also flushes the XB.

5. From these monitored conditions it initiates a prefetch cycle and performs these functions:

   a. It enables MA SELECT <S1:S0> lines to steer PC or PC + 4 onto the MAD bus from the ADD section.

   b. It asserts the prefetch signal to all other chips to set up paths to receive I-Stream data.

   c. It asserts or deasserts XB SELECT to clock data to the empty XB register. This also selects the outputs of the other register for use by the program (see Paragraph 2.5.3.3).

   d. It stalls the microcode (asserts the STALL signal to take priority) if data needed by the microcode is not available, or if the data paths are in use by other than a bus function CMI cycle.

The following paragraphs further describe PRK functions and signals.

PREFETCH – Initiates the prefetch cycle. CMI cycle is generated by the CMK to retrieve XB0 or XB1 data when a bus cycle is not decoded or a cache invalidation is completed.

MA SELECT <S1:S0> – Memory address signals select ADD registers to the MAD bus as shown in Table 2-23, Paragraph 2.5.2.1. An MA SEL value of 00 sources the PC increment register to the MAD bus to be used as the prefetch address to memory.

LATCH MA – Asserted on a microtrap, the MA latch closes to capture the address being generated at the time the microtrap occurs. MA contents at this time may be a prefetch address or may be the result of a bus function that caused a memory cycle.

ENA PC – Enables PC to be clocked with incremented information as I-Stream is used or is loaded with new information.

M MUX SEL S1 from the PRK is one of the MBUS MUX control lines to the MDR chips. It is used during cache invalidation on CMI writes.

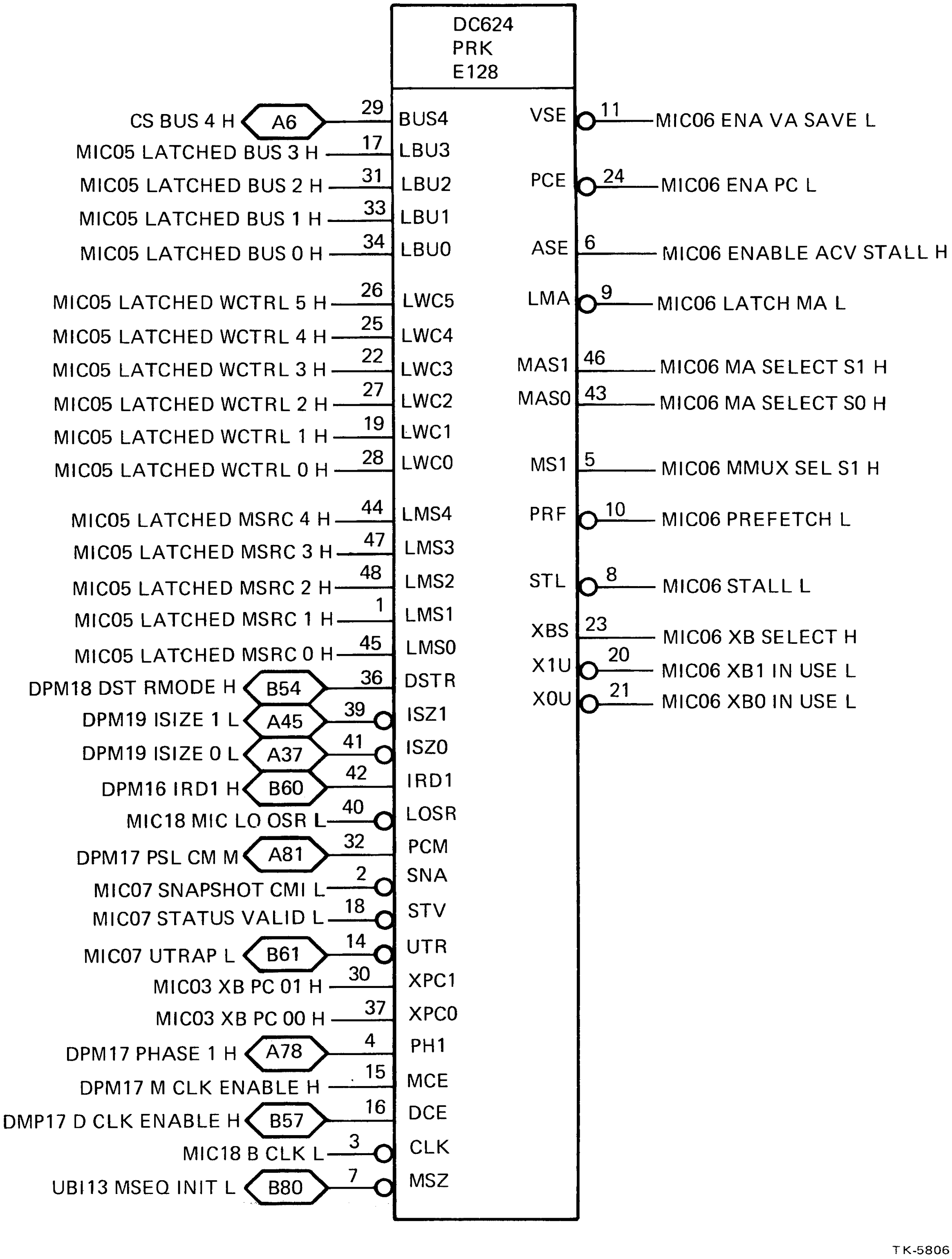


Figure 2-72 Prefetch Control (PRK)

ENA VA SAVE – PC + size latch is opened in ADD section, transparent to updated value it passes to the PC.

MSEQ INIT – From the UBI, this initializes control state elements on a power-up.

STALL – This is the signal that stalls the microcode (stops M CLK).

The following are examples of conditions that generate STALL.

Cache does not contain read data, generate CMI cycle to memory.
Prefetch cycle is in progress and MDR data path is in use.
Microcode attempts write to WDR register and last bus cycle is not completed.

XB SELECT – Steered by XB PC <01:00> from the ADD, this signal deasserted selects XB1 data outputs for use by the XB decode bus, or MBus, and XB0 inputs to receive I-Stream data from memory. XB1 inputs and XB0 outputs are selected when the signal is asserted as shown in Table 2-32, Paragraph 2.5.3.3.

<XB1:XB0> IN USE – These are used by the UTR with XB SELECT during a prefetch or XB MSRC to determine whether a microtrap conditon exists.

ENABLE ACV STALL – Used by MIC discrete logic for stall timing during TB parity generation.

ISIZE <1:0> – These signals come from the DPM to indicate the size of the instruction:

00	= 00
01	= Byte
10	= Word
11	= Longword

IRD 1 – From the DPM, IRD 1 signals that an operation code (opcode) of one byte is required for instruction fetch. It is also used to develop XB SELECT, and with LD OSR outputs, select needed byte(s) from XB.

LD OSR – From the DPM, load OSR requests another operand specifier (OSR) be output from XB1 or XB0.

UTRAP – From the UTR, UTRAP (microtrap) inhibits any prefetch from occurring until the microtrap routine is completed.

INVAL PREF – INVAL Prefetch simulates a prefetch cycle to the CMK, ADK, and CAK for one B CLK period to clear flip-flops within those chips when a cache invalidate function and bus grant occur simultaneously.

2.5.9.5 Access Control Violation (ACV)

Microtraps – The ACV (Figure 2-73) monitors and identifies microtrap conditions for the microtrap chip (UTR). It encodes ENC UTRAP <2:0> levels to the UTR in priority order as in the following chart:

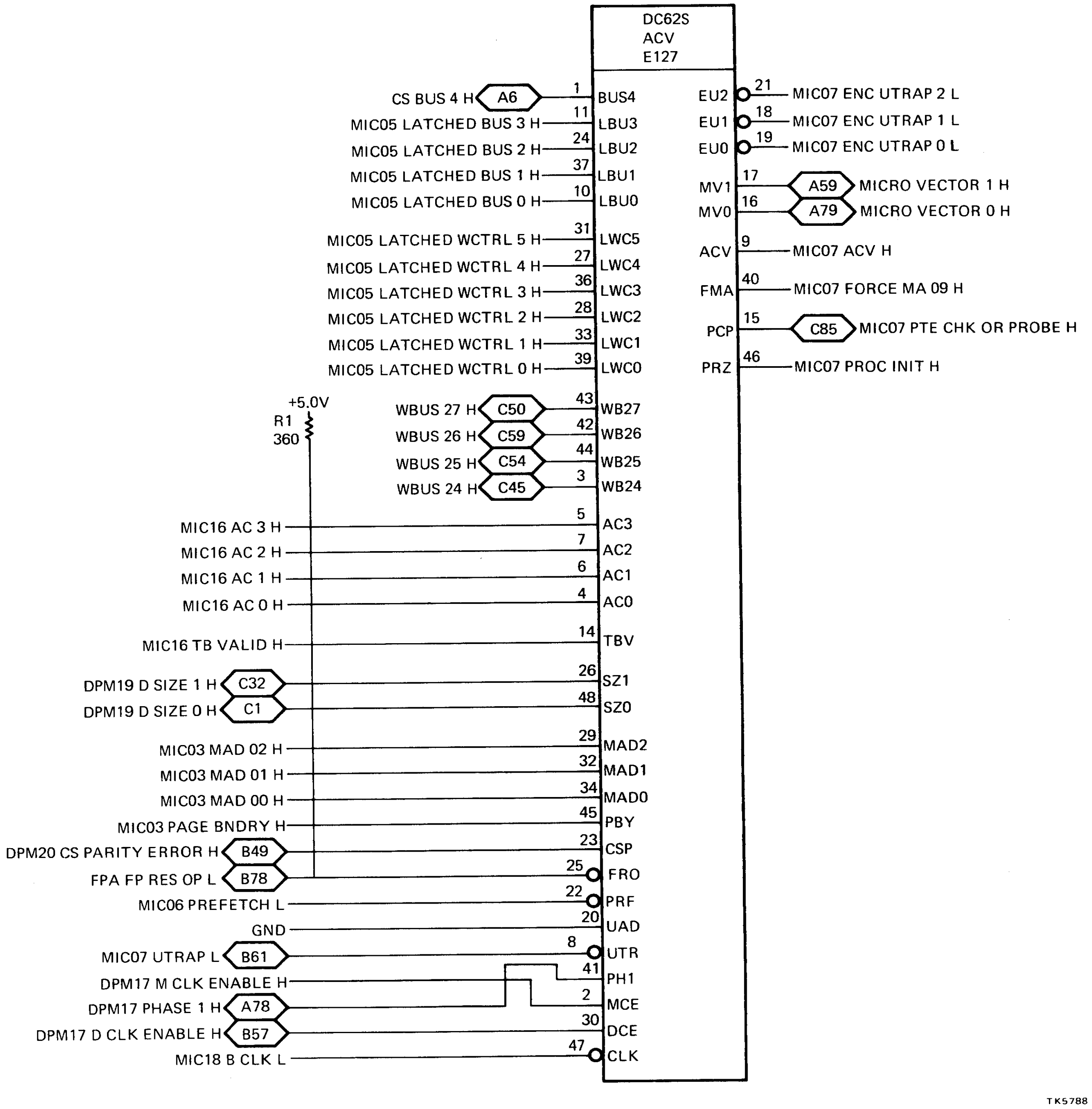


Figure 2-73 Access Control Violation (ACV)

Microtrap	ENC UTRAP			
Level	**2**	**1**	**0**	**Microtrap Condition**
1	1	1	1	Control Store Parity Error
2	1	1	0	FPA Reserved Operand
3	1	0	0	Reserved
4	0	1	0	Write Crossing Page Boundary
5	0	1	1	Write Unlock Crossing Page Boundary
6	1	0	1	Write Unlock Unaligned Data
7	0	0	1	Unaligned Data

The value of these levels is all zeros unless a microtrap is detected. The following paragraphs further describe microtrap conditions for this chart.

Unaligned Data microtrap is detected when the bus function is one of those listed below, coincident with the MAD <01:00> and D-Size <1:0> combinations marked "UNAL" on the chart.

Write
Write Unlock (Microtrap is Write Unlock, Unaligned Data)
Write if Not R Mode
Read with Modify Intent
Read
Read Lock
Probe Access, Read
Probe Access, Read, Mode Specified
Probe Access, Write
Probe Access, Write, Mode Specified
PTE Access Check, Read
PTE Access Check, Write
PTE Access Check, Read, Kernel Mode

D-Size	**MAD <01:00>**			
<1:0>	**00**	**01**	**10**	**11**
00				
01				UNAL
10		UNAL	UNAL	UNAL
11		UNAL	UNAL	UNAL

Two microtrap conditions are detected on ACV inputs: CS Parity Error and FP RES OP (FPA reserved operand).

Cross Page – This is a condition generated internal to the ACV chip. It is used to monitor and detect conditions common to these microtraps:

Unaligned Unibus Data
Write Unlock Crossing Page Boundary
Write Crossing Page Boundary

The ACV monitors the WBus and WCTRL fields to determine when MME is set and maintains an internal MME flip-flop.

Cross Page gating is enabled when MME and PAGE BNDRY from the ADD section are true and Prefetch and FPA RES OP are false during one of the bus functions listed below. Cross Page is then true when MAD <02:00> and D-Size <1:0> coincide to designate end of page (EOP) as indicated on the chart.

Write
Write Unlock
Write if Not R Mode
Probe Access, Read
Probe Access, Read, Mode Specified
Probe Access, Write
Probe Access, Write, Mode Specified

D-Size	**MAD <02:00>**							
<1:0>	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
00								
01								EOP
10						EOP	EOP	EOP
11		EOP	EOP	EOP	EOP	EOP	EOP	EOP

Two microtraps are detected when Cross Page is true during one of these indicated bus functions.

Write Crossing Page Boundary:
- Write
- Write if Not R Mode
- Probe Access, Read
- Probe Access, Read, Mode Specified
- Probe Access, Write
- Probe Access, Write, Mode Specified

Write Unlock Crossing Page Boundary:
- Write Unlock
- Probe Access, Read
- Probe Access, Read, Mode Specified
- Probe Access, Write
- Probe Access, Write, Mode Specified

Violation Detection – Other ACV chip signals have violation detection functions described in the following paragraphs.

ACV – Access violation to the UTR is generated when the access code monitored on the AC <3:0> inputs violates the access protection code for the current processor mode. The ACV chip contains the current mode (CM) register of the PSL. CM <1:0> are read and written on WBUS <25:24> under WCTRL direction.

Access Checks – The following bus functions are checked for read access.

Read
Read, Second Reference
Read Longword
Read Physical Address
Probe Access, Read
Probe Access, Read, Mode Specified
PTE Access Check, Read
PTE Access Check, Read, Kernel Mode

If a prefetch cycle is not in progress during PTE Access Check, Read, Kernel Mode, then CM <1:0> are forced to Kernel Mode (00). All other bus functions are checked for write access except for the following codes. No access check is made on these functions.

Read, No Microtrap
Write, No Microtrap
Write Longword, No Microtrap

PTE CHK or PROBE – Asserted when UTRAP is false from the UTR and the specified bus function is PTE Access Check or Access Probe.

UTRAP – Asserted by the UTR to hold off PTE Access Check or Access Probe bus functions until a microtrap is completed.

MICROVECTOR <1:0> – These tri-state lines are asserted from the ACV if the bus function is PTE Access Check or Access Probe and UTRAP is not asserted by the UTR. They are wire-ORed into the CS NEXT outputs to generate branching on the NEXT field of the microcode. They are asserted for these conditions.

> MICROVECTOR 1 – Access Probe with TB valid or MME disabled; or for TB valid with ACV false.
>
> MICROVECTOR 0 – Access Probe with MME disabled; or for ACV false with TB valid or PTE Access Check enabled.

FORCE MA 09 – This is deasserted during phase 1 of WCTRL code 29 (clear TB Valid bit) and asserted during phase 2. It is used by TB invalidation routines to clear two index locations of both TB groups in a single microinstruction. The TB index location is specified by the WBus value and loaded to the VA register.

PROC INIT – Processor initialize is generated by bus function I/O initialize.

2.5.9.6 Microtrap Generator (UTR) – The UTR, Figure 2-74, monitors conditions that may cause a microtrap during execution of a machine instruction. When a microtrap occurs, the UTR turns off microvector <3:0> lines from the MSQ to assert them to direct the microsequencer to the microroutine that handles the condition.

Microvector <3:0> lines from the UTR generate the low-order hexadecimal digit of the control store address. The MSQ chip on the DPM drives the 2X code onto control store address <5:4> for microtraps listed below, while the UTR drives the 0 through F values of the least-significant hexadecimal digit.

Microvector Address	**Microtrap Name**
20	Control Store Parity Error
21	Read Unaligned Data
22	MSRC XB Miss
23	MSRC XB ACV
24	Write Unlock Unaligned Data
25	Write Unaligned Data
26	Write Unlock Crossing Page Boundry
27	Write Crossing Page Boundry
28	Machine Check Exception (See Below)
29	BUT XB Miss
2A	Read TB Miss
2B	Write TB Miss
2C	FPA Reserved Operand
2D	BUT XB ACV
2E	Read ACV
2F	Write ACV

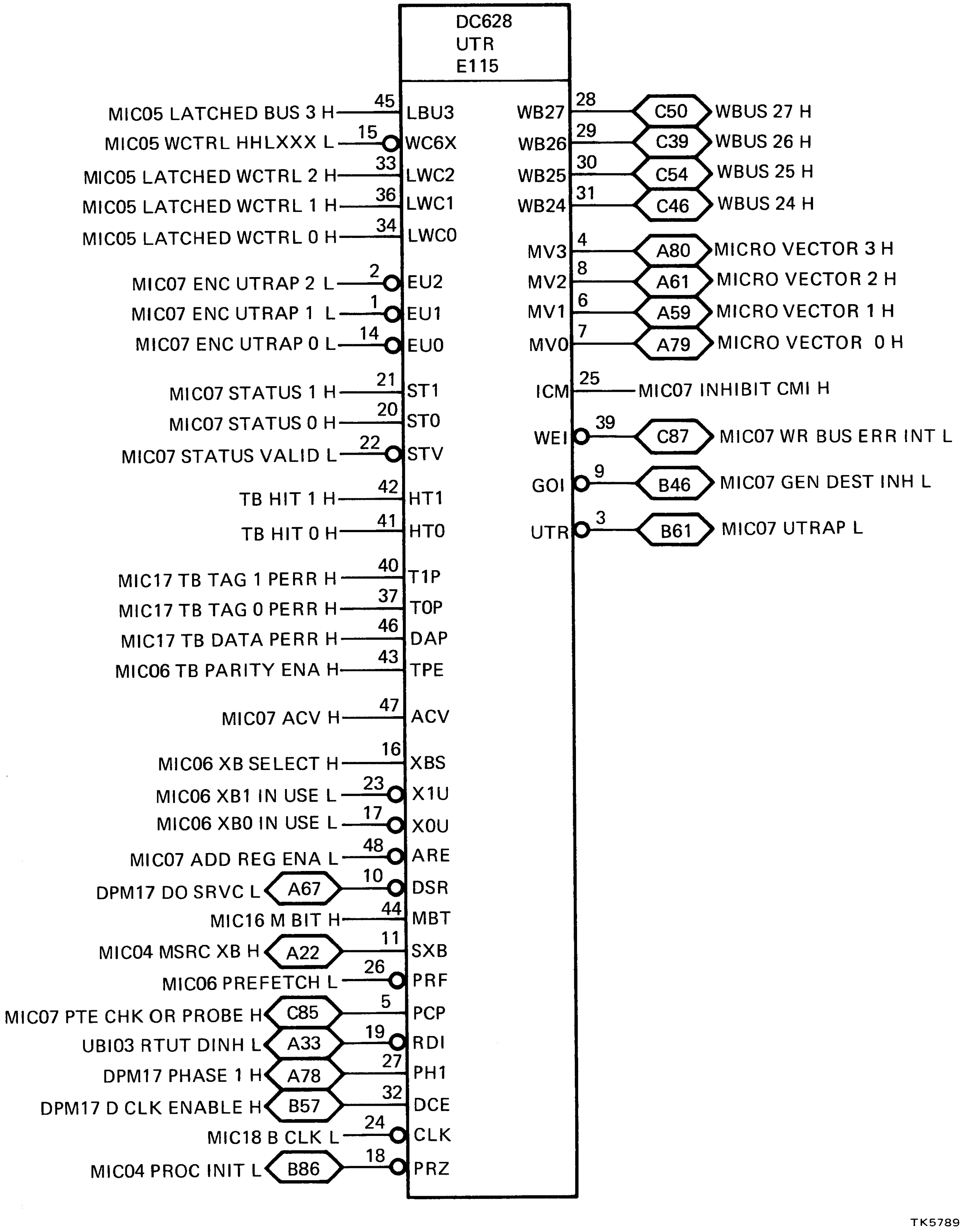

DC628
UTR
E115
MIC05 LATCHED BUS 3 H 45 LBU3
MIC05 WCTRL HHLXXX L 15 WC6X
MIC05 LATCHED WCTRL 2 H 33 LWC2
MIC05 LATCHED WCTRL 1 H 36 LWC1
MIC05 LATCHED WCTRL 0 H 34 LWC0
MIC07 ENC UTRAP 2 L 2 EU2
MIC07 ENC UTRAP 1 L 1 EU1
MIC07 ENC UTRAP 0 L 14 EU0
MIC07 STATUS 1 H 21 ST1
MIC07 STATUS 0 H 20 ST0
MIC07 STATUS VALID L 22 STV
TB HIT 1 H 42 HT1
TB HIT 0 H 41 HT0
MIC17 TB TAG 1 PERR H 40 T1P
MIC17 TB TAG 0 PERR H 37 T0P
MIC17 TB DATA PERR H 46 DAP
MIC06 TB PARITY ENA H 43 TPE
MIC07 ACV H 47 ACV
MIC06 XB SELECT H 16 XBS
MIC06 XB1 IN USE L 23 X1U
MIC06 XB0 IN USE L 17 X0U
MIC07 ADD REG ENA L 48 ARE
DPM17 DO SRVC L A67 10 DSR
MIC16 M BIT H 44 MBT
MIC04 MSRC XB H A22 11 SXB
MIC06 PREFETCH L 26 PRF
MIC07 PTE CHK OR PROBE H C85 5 PCP
UBI03 RTUT DINH L A33 19 RDI
DPM17 PHASE 1 H A78 27 PH1
DPM17 D CLK ENABLE H B57 32 DCE
MIC18 B CLK L 24 CLK
MIC04 PROC INIT L B86 18 PRZ
WB27 28 C50 WBUS 27 H
WB26 29 C39 WBUS 26 H
WB25 30 C54 WBUS 25 H
WB24 31 C46 WBUS 24 H
MV3 4 A80 MICRO VECTOR 3 H
MV2 8 A61 MICRO VECTOR 2 H
MV1 6 A59 MICRO VECTOR 1 H
MV0 7 A79 MICRO VECTOR 0 H
ICM 25 MIC07 INHIBIT CMI H
WEI 39 C87 MIC07 WR BUS ERR INT L
GOI 9 B46 MIC07 GEN DEST INH L
UTR 3 B61 MIC07 UTRAP L
TK5789

Figure 2-74 Microtrap Generator (UTR)

Machine check exception may be the result of any of these conditions:

MSRC XB TB Error
MSRC XB Bus Error
Bus Error
TB Error
But XB TB Error
But XB Bus Error

For a machine check, a macroroutine examines all conditions pushed onto the stack by the microroutine starting at location 28, and examines the necessary IPRs to determine the problem.

The machine check error codes are as follows:

0 = Unused
1 = Control Store Parity Error
2 = Memory Error
3 = Cache Parity
4 = Write Bus Error
5 = Corrected Memory Data
6 = Unused
7 = Bad IRD

The above error codes are developed in the DPM and pushed onto the stack at the stack pointer (SP) address plus four. The following data is pushed onto the stack by a machine check microtrap.

(SP)	Length Parameter = 28 (hex)
(SP) + 4	Error Code (from above list)
(SP) + 8 VA	Virtual Address Register (operand address)
(SP) + C PC	PC at time of exception (OSR address)
(SP) + 10 MDR	Memory Data Register (Data to or from memory)
(SP) + 14 SMR	Saved Mode Register (CPU mode during fault, MME, R/W)
(SP) + 18 RLTO	Read Lock Timeout Register (Bit 0 = 1, timeout)
(SP) + 1C TBGPR	Translation Group Parity Register
(SP) + 20 CAER	Cache Error Register
(SP) + 24 BER	Bus Error Register
(SP) + 28 MCESR	Machine Check Error Summary Register
(SP) + 2C PC BACKUP	(Opcode address)
(SP) + 30 PSL	Processor Status Longword

Microtraps are tested by priority gating in the following priority sequence (1 is highest priority and 22 is lowest).

Priority Sequence	Microtrap Name	Microvector Code
1	Control Store Parity Error	(20)
2	FPA Reserved Operand	(2C)
3	MSRC XB TB Error	(28)
4	MSRC XB Bus Error	(28)
5	Bus Error	(28)
6	Unaligned Unibus Data	(28)
7	MSRC XB TB Miss	(22)
8	MSRC XB ACV	(23)
9	TB Error	(28)
10	TB Miss, Read	(2A)
11	TB Miss, Write	(2B)
12	ACV, Read	(2E)
13	ACV, Write	(2F)
14	Write Crossing Page Boundary	(27)
15	Write Unlock Crossing Page Boundary	(26)
16	Unaligned Data, Read	(21)
17	Unaligned Data, Write	(25)
18	Unaligned Data, Write Unlock	(24)
19	BUT XB TB Error	(28)
20	BUT XB BUS Error	(28)
21	BUT XB TB Miss	(29)
22	BUT XB ACV	(20)

Memory Status/Control Registers – The UTR contains five registers, described in Paragraph 2.5.6 and illustrated in Figures 2-59 through 2-61.

XB Status – UTR also contains two 9-bit status registers, one each for XB1 and XB0. Their purpose is to latch error or status conditions during a prefetch cycle. These chip inputs are monitored:

TB HIT <1:0>
STATUS <1:0>
ACV
TB TAG <1:0> PERR
TB DATA PERR
<2:0> code from ACV chip)

The XB1 and XB0 error registers are opened when enabled as follows, and latch the indicated states internal to the UTR.

XB1 ERR ENA:

XB1 TB HIT 1
XB1 TB HIT 0
XB1 STATUS 1
XB1 STATUS 0
XB1 ACV
XB1 TAG 1 PERR
XB1 TAG 0 PERR
XB1 DATA PERR

XBO ERR ENA:

XBO TB HIT 1
XBO TB HIT 0
XBO STATUS 1
XBO STATUS 0
XBO ACV
XBO TAG 1 PERR
XBO TAG 0 PERR
XBO DATA PERR

TB HIT <1:0> from each register are checked for a multiple hit and ORed to the TB error bit in XB microtraps.

STATUS <1:0> are each decoded and ORed to the bus error register for corrected data, uncorrectable data, or for nonexistent memory bits.

ACV from either register generates XB microtrap for prefetch access violation.

TAG <1:0> and DATA PERR are ORed from each register to the TBGPR for a machine check.

2.6 CPU DATA PATH

2.6.1 Data Path Overview

The VAX-11/750 data path is 32 bits wide. Its main components are five different types of LSI gate chips and two arrays of scratchpad registers. Functionally the data path consists of three major logic sections.

Scratchpad logic – This logic is composed of 64 × 32 bits registers. These registers are divided into four groups: 16 GPRs (general purpose registers), 16 IPRs (internal processor registers), 16 RTEMPs (R-type temporary registers), and 16 MTEMPs (M-type temporary registers). The scratchpad logic also includes an SPA (scratchpad address control) section that provides address control to the scratchpad registers. The WBus is used to write data into the scratchpad registers. The RTEMPs, GPRs, and IPRs output data on the RBus. MTEMP data is output on the MBus.

Rotator logic – The rotator is conceptually a 64-bit in, 32-bit out barrel shifter combined with a data-shuffling multiplexer.

There are three sources of data into the rotator.

1. MBUS, denoted by M, is normally used as the input data <63:32> to the rotator.

2. RBUS, denoted by R, is normally used as the input data <31:00> to the rotator.

3. LITRL are 9-bit input data directly from the following microfields: RSRC, ISTRM and CC. The 9-bit LITRL can be zero or one extended to 32-bit and rotated by 0, 1, 2 ... 7 nibbles.

The barrel-shifting operation is implemented in two levels. The first level shifts the 64-bit inputs right by 0, 4, 8 ... 28 bits and outputs a 35-bit intermediate result. This level shifts the SBus data right by 0, 1, 2, or 3 bits. Outputs from the second level shifter are denoted by S <31:00>. By a proper combination of the two level shifts, the 64-bit input data can be shifted right 0 through 31 bits and left 1 through 31 bits.

The SBus data can also be masked off starting from an arbitrary bit position. This, combined with the barrel-shifting operation, effectively executes a variable length bit field extract, and zero-extended operation.

The data shuffling multiplexer implements some VAX-peculiar functionality such as BCD swapping, convert from BCD format to ASCII, etc.

ALP (Arithmetic Logical Processor) Logic – The ALP is made up of eight identical slices of gate array chips connected to perform 32-bit binary and 8-digit BCD arithmetic with carry look-ahead logic. Two internal registers are provided for intermediate storages.

There are seven major sections associated with the ALP logic:

1. ALU Input MUX, A MUX and B MUX
2. ALU
3. Output MUX, W MUX
4. Q Register
5. D Register
6. WBus Control
7. Status Logic

The ALU performs three binary arithmetic operations, two quasi-BCD arithmetic operations, and five logical operations.

The three binary arithmetic operations are:

A plus B plus CIN (A + B + CIN)
A plus .NOT.B plus CIN (A – B – CIN)
B plus .NOT.A plus CIN (B – A – CIN)

In this mode, two carry look-ahead signals (P and G) are calculated based on 16.

The two quasi-BCD arithmetic operations are:

A plus B plus CIN (A + B + CIN, BCD)
A plus .NOT.B plus CIN (A – B – CIN,BCD)

In this mode, the output of the ALU is the same as for binary arithmetic, but the P and G signals are calculated based on 10. Extra logic is used to adjust the 4-bit ALU output to a true BCD result.

The file logical operations are:

A.AND.B
A.OR.B
A.ANDNOT.B
B.ANDNOT.A
A.XOR.B

The ALP logic receives its inputs from the MBus, RBus, and super rotator. The ALP outputs data to the WBus. ALP logic is controlled by the ALP CTL and ROT microfields.

2.6.2 Data Path Control

The VAX-11/750 data path is under control of the following microfields.

Field	Width (in bits)
LIT	2
SPW	2
MSRC	5
ROT	6
ALPCTL	10
RSRC	6
ISTRM	1
DTYPE	2

The function of each of these microfields is discussed extensively in Paragraphs 2.6.4 through 2.6.6.5 on the Data Path.

2.6.3 I-Size and D-Size Source

I-Size <1:0> L and D-Size <1:0> H are generated on DPM 19.

I-Size <1:0> L are output to the MIC module. Here they are used to define the size of fetches (number of bytes) from the execution buffer (XB) to the MBus and to the DPM for instruction and operand specifier decode. I-Size is also used to specify the increment value added to the PC during each use of the XB. I-Size <1:0> L are input to the following MIC module logic sections.

ASRC select logic (E89, E88, E87, and E92) (located on MIC 04)

PRK (prefetch control) gate array (see MIC 06)

Details of I-Size <1:0> functionality are covered in the MIC module Section 2.5.9.

D-Size <1:0> H are used by both the DPM and MIC modules. On the DPM module, D-Size <1:0> H define data size (e.g., byte, word, longword, quad) to the following logic.

ALK (arithmetic logic control) gate array (located on DPM 10)

A 2 × 4 multiplexer, E64 (located on DPM 3)

ALP (arithmetic logic processor) gate array (located on DPM 5, 6, 7, and 8)

SRK (super rotator control) gate array (located on DPM 9)

CCC (condition code logic) gate array (located on DPM 10)

SPA (scratchpad address) gate array (located on DPM 11)

On the MIC module, D-Size <1:0> H supply data size information to the following destinations.

CAK (cache control) gate array (located on MIC 6)

CMK (CMI control) gate array (located on MIC 7)

ACV (access control violation) gate array (located on MIC 7)

Actual implementation of D-Size <1:0> H is discussed more fully in each of the individual logic sections. For detailed information see the relevant paragraphs in this chapter.

2.6.3.1 I-Size <1:0> L Generation – (See Table 2-34 and DPM 19.) Table 2-34 shows the hardware conditions for generation of I-Size <1:0> L. I-Size <1:0> L may be derived from the following three possible sources.

1. D-Type <1:0> H – This is made available to the I-size multiplexer from the CS latch (E44) on DPM 12. These two bits are part of the control store microword latched into the CS latch. The D-Type <1:0> H microword field can have four possible values:

   0 = Byte
   1 = Word
   2 = Longword
   3 = IDEP

   The first three values (0, 1, 2) can be used directly to specify data size, or, when decoded by the I-size logic (E31, E32, and E33), to specify I-Size <1:0> L. If D-Type <1:0> H = 3 (IDEP), data size will be instruction-dependent. E.g., MOVL (data size = long), MOVB (data size = byte), etc.

2. D-Size multiplexer out – When D-Type <1:0> H = 3 (IDEP), I-Size <1:0> L is determined by a decode of D-Size Latch <1:0> <0> H and D-Size Latch 0 <1> H. The states of these D-size latch (E30) signals are determined by the output of D-Size MUX E6 B <1:0>. This multiplexer receives its input from D-size PROM E7. The D-size PROM and D-size latch theory of operation is described in Paragraph 2.6.3.3.

3. DISP I-Size <1:0> H – This is supplied by the IRD gate array chip (DPM 18). DISP I-Size <1:0> H are produced by a decode of OSR (operand specifier register) <7:4>. These bits specify I-size when the general register addressing mode equals A, B, C, D, E, or F (relative or displacement mode) (see Table 2-21).

The I-size multiplexer (E33 and E32) output source is controlled by ISTRM H, LIT <1:0> and D-Type <1:0> H. These are all part of the microword latched into the CS latch (DPM 12). The interpretation and control function of these fields is as follows.

ISTRM H = 0 = NOP – In this case LIT <1:0> and D-Type <1:0> H have no significance. I-Size <1:0> L are sourced from DISP I-Size <1:0> H.

ISTRM H = 1 = I-Size_D-Size (I-Size is determined by D-Size). Here the I-Size <1:0> L source depends on the values of LIT <1:0> and D-Type <1:0> H (see below).

LIT <1:0> = 1 or 3 (ISTRM H = 1). The interpretation of these values in the LIT field are as follows: 1 = LITRL = short literal field enabled, and 3 = LONLIT = long literal field enabled. In both of these cases I-Size <1:0> L is sourced from DISP I-Size <1:0> H.

LIT <1:0> = 0 or 2 (ISTRM H = 1 and D-Type <1:0> H are not equal to 3). The LIT field is interpreted as follows: 0 = normal = the relevant microword fields are not used as part of a literal value; 2 = FPA WAIT = wait for the FPA to complete processing. For both of these LIT field values, I-Size <1:0> L are derived by a decode of D-Type <1:0> H:

D-Type <1:0> H	**I-Size <1:0> L**
= 0	= 1 (byte)
= 1	= 2 (word)
= 2	= 3 (longword)

LIT <1:0> = 0 or 2, ISTRM H = 1, and D-Type <1:0> H = 3 (IDEP). As was discussed earlier when D-Type <1:0> H = 3, I-Size <1:0> L are produced from a decode of D-Size Latch <1:0> <0> H and D-Size Latch <0> <1> H.

Table 2-34 Hardware Conditions for I-Size <1:0> L Generation (see DPM 19)

Microword Fields					
ISTRM H	LIT <1:0>	D-Type <1:> H	DISP I-Size Decode from OSR	D-Size MUX Out (E6)	I-Size <1:0> L
0	X	X	0	X	0 (0 bytes)
0	X	X	1	X	1 (1 byte)
0	X	X	2	X	2 (2 bytes)
0	X	X	3	X	3 (4 bytes)
1	1, 3	X	0	X	0
1	1, 3	X	1	X	1
1	1, 3	X	2	X	2
1	1, 3	X	3	X	3
1	0, 2	0	X	X	1
1	0, 2	1	X	X	2
1	0, 2	2	X	X	3
1	0, 2	3	X	0	1
1	0, 2	3	X	1	2
1	0, 2	3	X	2	3
1	0, 2	3	X	3	3 (QUAD maps to LONG)

Notes: In general if I-Size is non-zero and MSRC does not specify ←XB, the PC is not updated and no microtrap associated with the XB occurs.
X = not applicable.

2.6.3.2 D-Size <1:0> H Generation – (See Table 2-35 and DPM 19.) Table 2-35 gives the hardware conditions for generation of D-Size <1:0> H. D-Size <1:0> H can be derived from three possible sources.

1. D-Type <1:0> H – see Paragraph 2.6.3.1

2. DISP I-Size <1:0> H – see Paragraph 2.6.3.1

3. D-Size Latch <1:0> <1> H – The value of these two bits is equivalent to the output of the D-Size MUX E6 B <1:0> (see Table 2-36).

The D-size multiplexer (E34) output source is controlled by four microword fields: ISTRM H, MSRC, LIT <1:0>, and D-Type <1:0> H. These control functions are implemented as follows.

Condition 1 – ISTRM H = 0, MSRC not equal to 17 (MBus does not receive execution buffer data), LIT <1:0> = 0 (fields are normal), 1 (short literal field enabled) or 2 (wait for FPA to complete processing), D-Type <1:0> H = 0, 1, or 2 – MSRC not equal to 17 results in MSRC XB H (DPM 19, location A8) = L, LIT is not equal to 3 (LONG LIT) so LONG LIT L (DPM 19, location A8) = H. With these hardware conditions on DPM 19, D-Size <1:0> H is sourced from D-Type <1:0> H.

Condition 2 – ISTRM H = 0, MSRC not equal to 17, LIT <1:0> = 0, 1, or 2 and D-Type 1:0 H = 3 (IDEP) – MSRC XB H (DPM 19, location A8) = L, LONG LIT L (DPM 19, location A8) = H. Because D-Type equals 3, both D-Type 1 H and D-Type 0 H (DPM 19, location B8) now are high. D-Size <1:0> H is now sourced from D-Size Latch <1:0> <1> H.

Condition 3 – For this condition LIT <1:0> = 3 (LONG LIT). As a result LONG LIT L (DPM 19, location A8) goes low. Since MSRC XB H = H, D-Size <1:0> H are now forced to (2).

Condition 4 – The MSRC field is now equal to 17 (the MBus receives XB data) which results in MSRC XB H (DPM 19, location A8) going high, ISTRM H = L brings FORCE D-Type L (DPM 19, location A8) high. D-Size <1:0> H is now derived from a decode of DISP I-Size <1:0> H.

Condition 5 – MSRC is not equal to 17 (MBus does not get XB) so MSRC XB H is now low. ISTRM H = H, LIT <1:0> = 1, and D-Type <1:0> H = 0, 1, or 2. Under these conditions the D-Size <1:0> H multiplexer (E34) output is sourced from D-Type <1:0> H.

Condition 6 – The change here from Condition 5 is that D-Type <1:0> H now equals 3 (IDEP). D-Size <1:0> H receives D-Size Latch <1:0> <1> H.

Condition 7 – Here the key point is that LIT <1:0> = 3 (LONG LIT). This being the case, LONG LIT L (DPM 19, location A8) is low. MSRC XB H is low (MBus does not receive XB). The state of LONG LIT L and MSRC XB H cause D-Size <1:0> H to be forced to (2).

Condition 8 – ISTRM H = H, LIT <1:0> = 1 or 3. This signal combination causes FORCE D-Type L (DPM 19, location A2 and A8) to go high. MSRC = 17 which brings MSRC XB H high. D-Size <1:0> H are now derived from a decode of DISP I-Size <1:0> H.

Conditions 9 and 10 – For both of these conditions, FORCE D-Type L = L. If D-Type <1:0> H equals 0, 1, or 2, D-Size <1:0> H are sourced from D-Type <1:0> H. If D-Type <1:0> H = 3 (IDEP) (Condition 10), D-Size <1:0> H are sourced from D-Size Latch <1:0> <1> H.

Table 2-35 Hardware Conditions for D-Size <1:0> H Generation (see DPM 19)

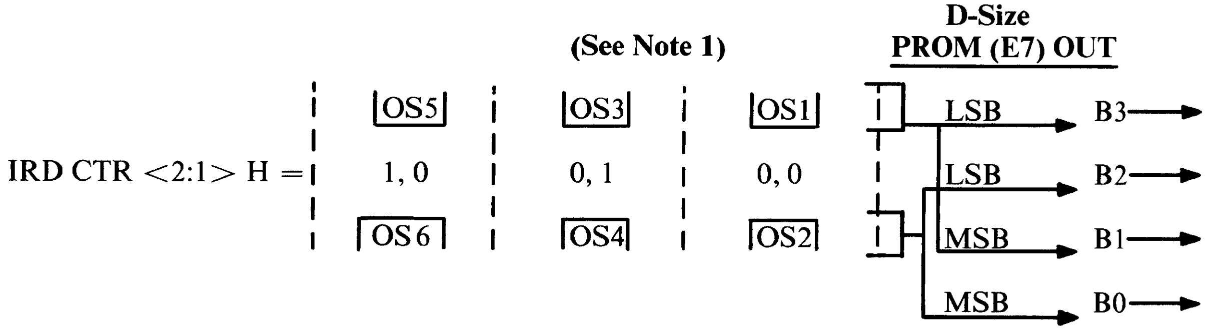
	Microcode Fields					
Condition No.	ISTRM H	MSRC	LIT <1:0>	D-Type <1:0> H	DISP I-Size Decode From OSR	D-Size <1:0> H
	0	NOT ← XB	0, 1, 2	0	X	0
1	0	NOT ← XB	0, 1, 2	1	X	1
	0	NOT ← XB	0, 1, 2	2	X	2
2	0	NOT ← XB	0, 1, 2	3	X	D-Size latch
3	0	NOT ← XB	3	X	X	2
	0	← XB	X	X	0	1
	0	← XB	X	X	1	0
4	0	← XB	X	X	2	1
	0	← XB	X	X	3	2
	1	NOT ← XB	1	0	X	0
5	1	NOT ← XB	1	1	X	1
	1	NOT ← XB	1	2	X	2
6	1	NOT ← XB	1	3	X	D-Size latch
7	1	NOT ← XB	3	X	X	2
	1	← XB	1, 3	X	0	1
	1	← XB	1, 3	X	1	0
8	1	← XB	1, 3	X	2	1
	1	← XB	1, 3	X	3	2
	1	X	0, 2	0	X	0
9	1	X	0, 2	1	X	1
	1	X	0, 2	2	X	2
10	1	X	0, 2	3	X	D-Size latch

Notes: Condition nos. are used for illustrative purposes (see Paragraph 2.6.3.2).
X = not applicable.

2.6.3.3 IDEP, D-Size Circuit Description - See DPM 19 and Table 2-36. The IDEP, D-Size circuitry consists of a 2K × 4 bit PROM (D-size PROM) (E7), a 4 × 2 multiplexer (E6), two inverters (E45), and D-size latch (E30). This circuitry supplies data size information when microword field D-Type <1:0> H = 3 (IDEP). The data size in this case is instruction-dependent; e.g., MOVL (data size is longword) ADC (data size is word). See note 1 in Table 2-36).

The D-size PROM is addressed by PSL CM H, IR <7:0> H, and IRD CTR <2:1> H. PSL CM H and IR <7:0> H are used to address a 12-bit location in the D-size PROM which corresponds to the present macroinstruction being decoded by the IRD circuitry (DPM 18). Each 12-bit location is divided into 6 × 2 bit locations which may be selected by IRD CTR <2:1> H. These locations are titled OS1–OS6. They are selected as shown in Table 2-36. The example shown in Table 2-36 is for a compatibility mode ADC (add carry) macroinstruction. For this instruction (see note 1, Table 2-36) PSL CM H = 1 (H), IR <7:0> H = A3. Data at this address is output four bits at a time on D-size PROM outputs B <3:0>. IRD CTR <0> is used to select the D-size multiplexer output B <1:0>. D-size data is loaded into the D-size latch on the rising edge of BUF M CLK L when LD OSR A L = low and INDEX MODE BUT L = H. INDEX MODE BUT L = L when BUT <5:0> H = 3 (RET.DINH) or 7 (LOD.BRA) and LIT <1:0> are not equal to 3 (LONG LIT).

Table 2-36 D-Size Latch Hardware Conditions (see DPM 19)



D-Size MUX (E6) input	IRD CTR <0>	D-Size MUX (E6) OUT B <1:0>
→ D00 → D01	= 0	D00 → B0 D10 → B1
→ D10 → D11	= 1	D01 → B0 D11 → B1

NOTES: 1. e.g.:

Macro Inst	PSL CM H	IR <7:0> H	D-Size for OS No.					
			1	2	3	4	5	6
ADC	= 1	= A3	word	0	0	0	0	0

IRD CTR <2:0> H = 0
(see Notes 2 and 3)

Table 2-36 D-Size Latch Hardware Conditions (see DPM 19) (Cont)

2.

IRD CTR			OS
2	1	0	No.
0	0	0	= 1
0	0	1	= 2
0	1	0	= 3
0	1	1	= 4
1	0	0	= 5
1	0	1	= 6

OS (operand specifier)

3. Byte = 0
Word = 1
Long = 2
Flot = 2
Quad = 3
DBLE = 3

2.6.4 Scratchpad Section

The scratchpad section of the data path consists of the scratchpad register sets and the scratchpad address logic (SPA chip). Figure 2-75 illustrates the associated logic.

2.6.4.1 Scratchpad Register – The scratchpad of the data path consists of two RAM arrays: RAM-M and RAM-R. RAM-M is a register set containing 16, 32-bit locations. These locations provide temporary storage for addresses, operands, and other data during the execution of the microprogram.

RAM-R is a register set containing 48, 32-bit locations. These locations are divided into three general groups as follows.

Location	Mnemonic	General Usage
0–15	RTEMP0-RTEMP15	Microcode temporaries (similar to RAM-M)
16–31	GPR0-GPR15	General purpose registers (GPR15 is actually a microcode temporary)
32–47	IPR0-IPR15	Internal processor registers or microcode temporaries.

Locations 0 through 7 of both RAM-M and RAM-R are implemented to function as a dual-port RAM. That is, a write to any of these locations in one RAM simultaneously results in a write to the same location in the other RAM (Paragraph 2.6.4.4). This feature provides some flexibility to the microcode.

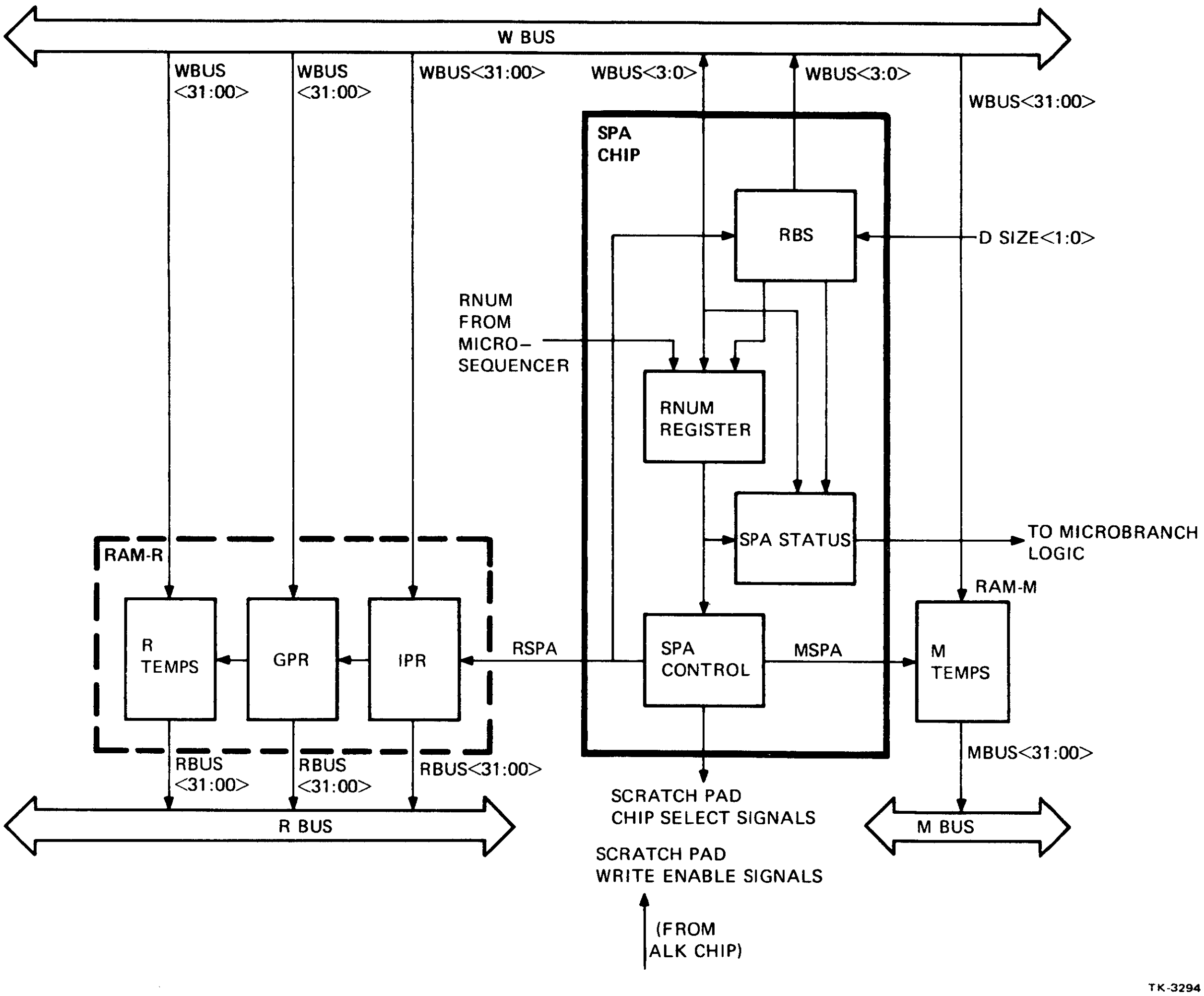


Figure 2-75 Scratchpad Logic

2.6.4.2 Scratchpad Address Selection – The microcode is capable of addressing a scratchpad location either explicitly as a number in a microfield or indirectly through the RNUM register (Paragraph 2.6.4.6). The RSRC and MSRC fields of the microword select a scratchpad location in RAM-R and RAM-M, respectively. These fields are also used to control various internal operations in the SPA chip and for selection of MBus and RBus sources. Table 2-37 shows the relationship between the RSRC field value and the RAM-R location or function selected. Table 2-38 shows the relationship between the MSRC field value and the RAM-M location or function selected. In these tables, R is a 4-bit register (RNUM) discussed in Paragraph 2.6.4.6. GPR.R denotes a general purpose register indexed by the RNUM register. MM.TEMP denotes a temporary register used for memory management. Temporaries listed in parentheses are defaults.

Table 2-37 RSRC Assignments

RSRC <5:0> (Hex)	RAM-R Register	Operation
00–0D	TEMP0–TEMP13	–
0E	MM.TEMP5	–
0F	MM.TEMP1	–
10–1D	R0–R13	–
1E	SP	–
1F	RTMPGPR	–
20	KSP	–
21	ESP	–
22	SSP	–
23	USP	–
24	ISP	–
25	PCBB	–
26	MM.TEMP2	–
27	MM.TEMP3	–
28	P0BR	–
29	P0LR	–
2A	P1BR	–
2B	P1LR	–
2C	SBR	–
2D	SLR	–
2E	SPNICR.SPICR	–
2F	MM.TEMP4	–
30	TEMP.R	–
31	DST.R	–
32	IPR.R	–
33	GRP.R	–
34	(TEMP0)	–
35	(TEMP7)	LONLIT
36	(TEMP0)	ZERO
37	(TEMP0)	ZERO.CLRRBSP
38	TEMP.ROR1	–
39	DST.ROR1	–
3A	IPR.ROR1	–
3B	GPR.ROR1	–
3C	TEMP.R+1	–
3D	DST.R+1	–
3E	IPR.R+1	–
3F	GPR.R+1	–

When the RSRC field specifies either DST.R, DST.R+1, or DST.ROR1, the location addressed is the same as GPR.R, GPR.R+1, and GPR.ROR1 respectively. DST.R, DST.R+1, and DST.ROR1 are used to conditionally inhibit the writing of the general purpose registers (Paragraph 2.6.4.4).

In Table 2-37, ROR1 is interpreted as the RAM-R location specified by:

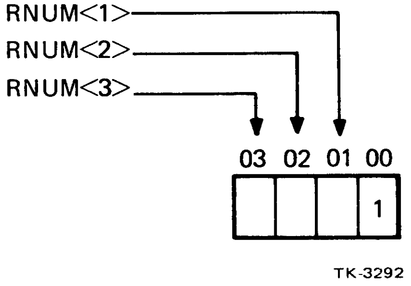


Table 2-38 MSRC Assignments

MSRC <4:0> (Hex)	RAM-M Register	Operation	Description
00–0A	TEMP0–TEMP10	–	Microcode Temporaries
0B	ERRCOD	–	Error Code
0C	FPDOFFSET	–	FPD Pack Routine Offset
0D	MM.TEMP0	–	Memory Management Temp
0E	SCBB	–	System Control Block Base
0F	SISR	–	Software INT Summary Reg.
10	TEMP.R	–	MTEMP Indexed by RNUM
11	TEMP.R+1	–	MTEMP Indexed by RNUM+1
12	(TEMP0)*	MDR	MBUS ← MDR
13	(TEMP0)*	WDR	MBUS ← WDR
14	(TEMP0)	PSHSUB	Write – to RBS
15	(TEMP0)	PSHADD	Write + to RBS
16	(TEMP0)	WBUS_RNUM	WBUS ← RNUM
17	(TEMP0)*	XB.PC_PC+1	MBUS ← XB, PC ← PC+I
18	(TEMP0)*	MA	MBUS ← MA
19	(TEMP0)*	PC BACK	MBUS ← PC BACK
1A	(TEMP0)*	PC	MBUS ← PC
1B	(TEMP0)*	VA	MBUS ← VA
1C	(TEMP0)	READRBS	Read RBS
1D	(TEMP0)	RNUM_WBUS	RNUM ← WBUS
1E	(TEMP0)	WB_RBSP	WBUS ← RBSP
1F	(TEMP0)*	TB	MBUS ← TB Data

*Write-Only

Most of the operations listed in Table 2-38 are self-explanatory. The less obvious ones, PSHADD, PSHSUB, and READRBS are explained in Paragraph 2.6.4.5. The operations listed in Table 2-38 are briefly described below.

LONLIT – This operation is used to source the contents of the long literal register onto the RBus. (Refer to Paragraph 2.2.1.2.)

ZERO – This operation is used to source a constant of all zeros onto the RBus.

ZERO.CLRRBSP – This operation is used to clear the RBS pointer (RBSP) under microcode control. Clearing the RBSP effectively clears the RBS. This operation also sources a constant of all zeros onto the RBus.

As mentioned previously, locations 0–7 of RTEMP and MTEMP are implemented as a dual port. This implies that a write to one of these locations in one RAM simultaneously results in a write to the same location in the other RAM. This simultaneous write is always performed even though the scratchpad location is not explicitly specified in the RSRC or MSRC microfields.

2.6.4.3 Scratchpad Address Generation – During each microcycle, the scratchpad address chip (SPA) decodes the MSRC and RSRC microfields to generate the appropriate scratchpad address and chip select. The chip select is asserted to enable the appropriate register set – RTEMP, GPR, IPR, or MTEMP. The scratchpad address selects one of the 16 locations within the selected register set.

Figure 2-76 illustrates the areas of the scratchpad associated with each chip select signal and scratchpad address. As this figure shows, the SPA logic provides separate address lines for RAM-M and RAM-R.

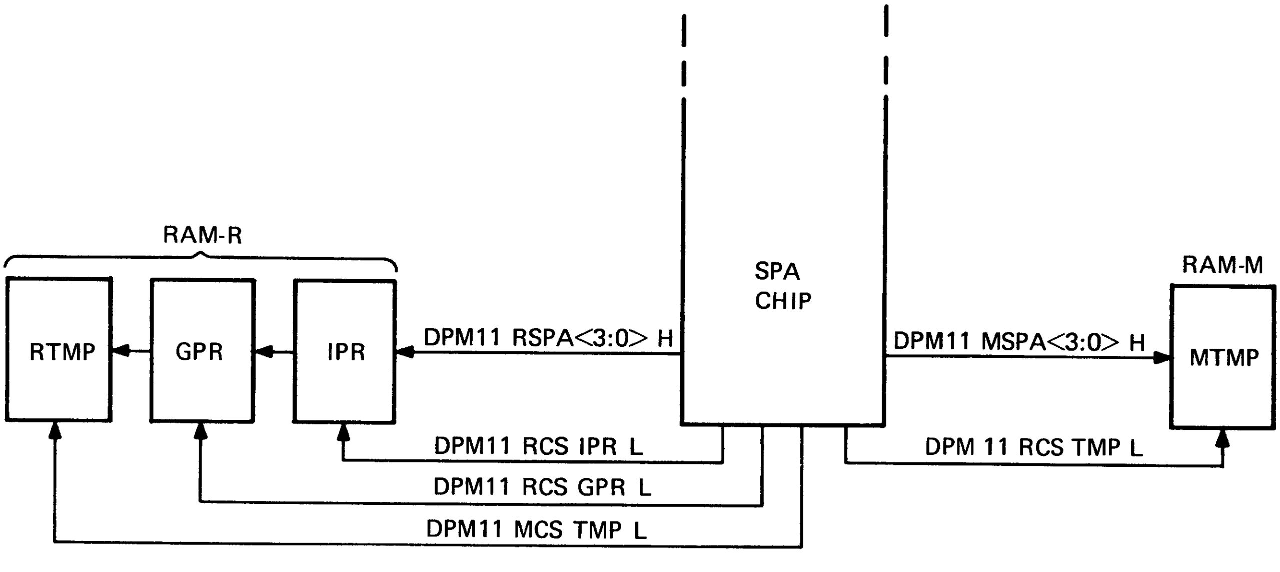


Figure 2-76 Scratchpad Address and Chip Select

2.6.4.4 Scratchpad Read/Write Control – The scratchpad locations selected by the MSRC and RSRC microfields are read during the first half of every microcycle. Contents from the selected location in RAM-R are output onto the RBus. Likewise, if the MSRC microfield selects a location in RAM-M, its contents are output onto the MBus. The SPA chip generates the appropriate chip selects and scratchpad addresses.

Scratchpad writes (SPW) are executed only during the second half of a microcycle. The SPW microfield determines whether or not a write to the location specified by the MSRC or RSRC microfield is to occur, and, if so, whether the write is D-size dependent. The values of the SPW microfield are interpreted as follows.

SPW <1:0>	Mnemonic	Location (For Chip Select)	Length of Write (For Write Enable)
00	NOP	No Write	–
01	RSIZE	RAM-R location specified by RSRC	Specified by D-size <1:0>
10	RLONG	RAM-R location specified by RSRC	Longword
11	MLONG	RAM-M location specified by MSRC	Longword

Normally, when the SPW microfield specifies a write to RAM-M, the scratchpad location is explicitly specified by the MSRC microfield. For cases in which the SPW microfield specifies a write to RAM-M but the MSRC microfield does not explicitly specify a scratchpad location, the write is defaulted to MTEMP0. Writes to RAM-R are handled similarly under the same conditions.

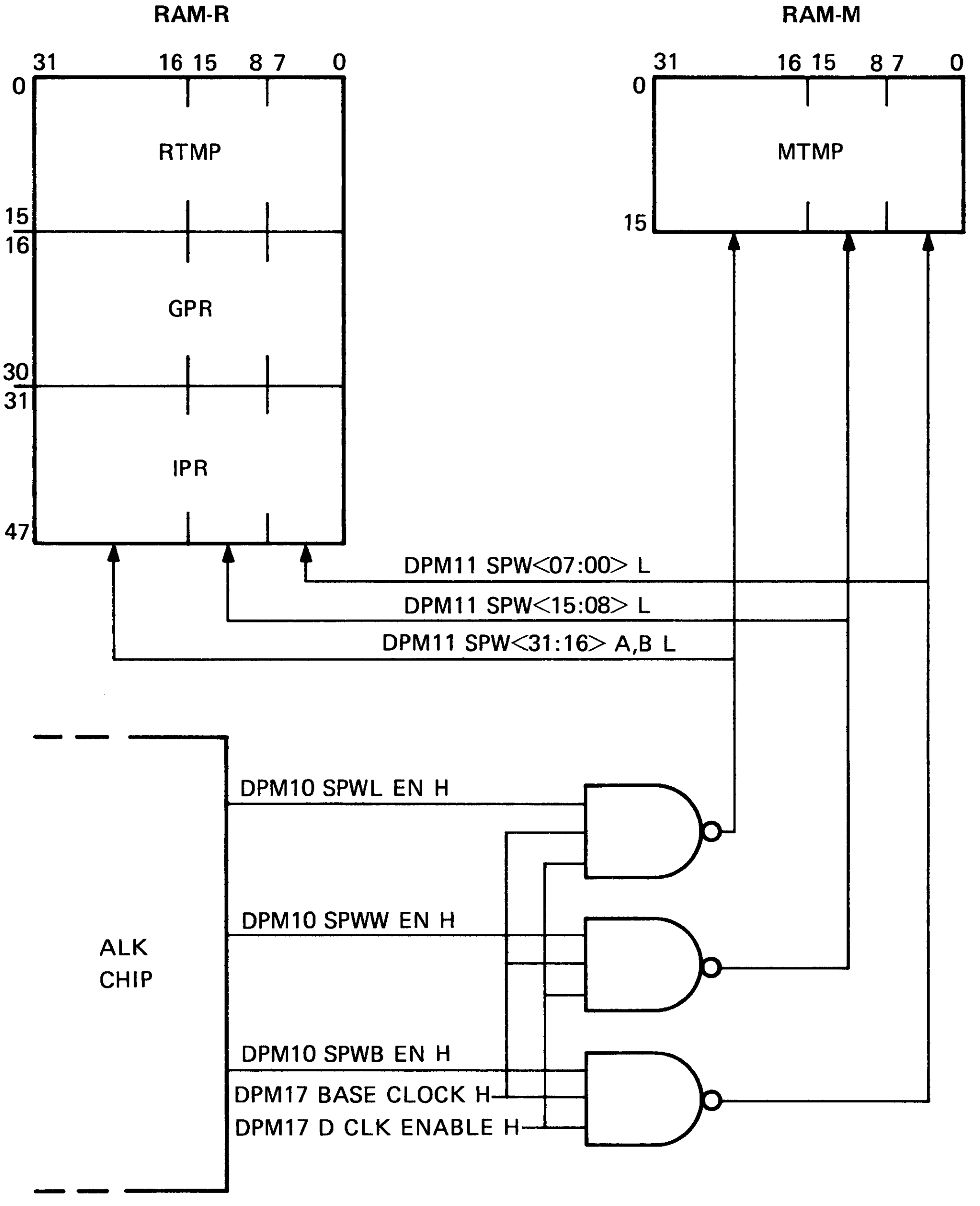
As an example of the default situation, assume the following microfields contain the indicated values.

MSRC <4:0> 1E	RSRC <5:0> 10	SPW <1:0> 11
Output RBSP onto WBus	Output GPR0 onto RBus	Write longword to RAM-M location specified by MSRC

The SPW microfield specifies a longword write during the second half of the microcycle to the RAM-M location specified by the MSRC microfield. For this microcycle, however, the MSRC microfield specifies an operation rather than a scratchpad location. The write is therefore defaulted to TEMP0 in RAM-M.

The SPW microfield is decoded by the SPA chip to generate the appropriate chip select signals. These signals are used to implement the dual-port write feature. If the SPW microfield equals 11 (MLONG) and the MSRC specifies (whether directly or indirectly through RNUM) an MTEMP location 0 through 7, the chip selects for MTEMP and RTEMP are both asserted. If MSRC does not specify a location 0 through 7, then only the chip select for the MTEMP location is asserted. When the SPW microfield contains a value other than 11, the scratchpad location is determined by RSRC instead of MSRC.

In addition to determining whether a write is to be executed, the SPW microfield specifies the amount of data to be written into the selected location. The SPW microfield is decoded by the ALK chip (Paragraph 2.6.5.3) to control the length of the write by generating the appropriate write enable signals. Figure 2-77 illustrates the areas of the scratchpad associated with each write enable signal. An area of the scratchpad is enabled for the write only if the corresponding write enable and chip select signals are asserted.



TK-3299

Figure 2-77 Write Enable Signals

If the SPW microfield equals 01, the number of bytes to be written is determined by two signals from the microsequencer, D-Size <1:0>. These signals are interpreted as follows.

D-Size <1:0>	Interpretation
00	Byte 0
01	Bytes 0 and 1
10	Bytes 0, 1, 2, and 3
11	Bytes 0, 1, 2, and 3

Refer to Paragraph 2.6.5.3 for a complete description of the D-size signals and their use.

2.6.4.5 Register Backup Stack (RBS) – The register backup stack (RBS) is located in the SPA chip. The RBS contains six 7-bit locations. These locations provide a means to save information required to restart an instruction. If an instruction causes a fault requiring a macro-level trap, it is necessary to restore the general purpose registers to their original state. The information stored in the RBS allows reconstruction of the register contents so that the instruction can be restarted.

Each RBS entry contains a bit that specifies whether a GPR was modified by an autoincrement or autodecrement operand specifier. It also contains two D-size bits that specify the data size for the current microcycle, and four bits that specify the register being modified. Figure 2-78 illustrates the RBS and the format of an RBS entry. Table 2-39 shows the interpretation of the D-size signals.

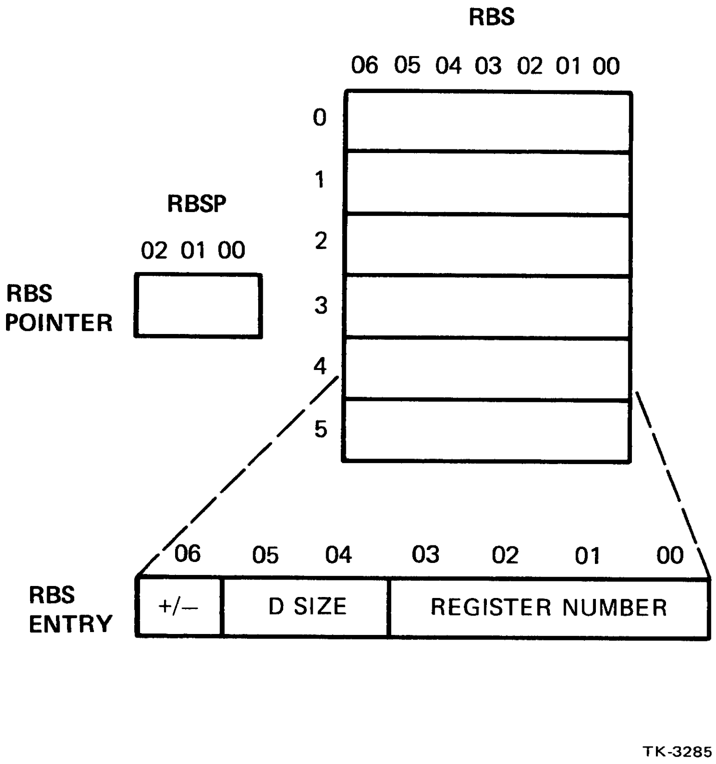


Figure 2-78 RBS Entry Format

Table 2-39 D-Size Interpretation

D-Size 5	 4	Data Size
0	0	Byte
0	1	Word
1	0	Longword
1	1	Quadword

The RBS operates more like a silo rather than a stack (i.e., first in-first out operation rather than last in-first out). Each time a macroinstruction is fetched, the RBS is emptied by clearing the RBSP (RBS pointer). The RBSP is incremented after each read or write to the RBS so that the value of the RBSP always represents the depth of the RBS. When information is to be removed from the RBS, the value of the RBSP can be saved in a temporary register before RBS is cleared. The appropriate number of reads is then performed to back up to the correct register.

Reads and writes to the RBS are controlled by the MSRC field of the microinstruction. When the MSRC microfield specifies a write to the stack (PSHADD or PSHSUB), information is pushed onto the stack before the RBSP is incremented. When the MSRC microfield specifies a read from the stack (READRBS), the location is likewise read before the RBSP is incremented. Table 2-40 shows the relationship between the MSRC microfield value and RBS operation.

Table 2-40 RBS Operations

RBS Operation	MSRC <4:0> (Hex)	Result
PSHADD	15	RBS <3:0> ← register number RBS <5:4> ← D-Size <1:0> RBS <6> ← 1 RBSP incremented
PSHSUB	16	RBS <3:0> ← register number RBS <5:0> ← data type D-Size <1:0> RBS <6> ← 0 RBSP incremented
READRBS	17	RNUM ← RBS <3:0> WBUS <3:0> ← encode RBS <5:4> SPASTA <1:0>* ← 0, RBS <6> RBSP incremented

*Refer to Paragraph 2.6.4.7 for a complete description of SPASTA <1:0>.

As indicated in Table 2-40, on a READRBS operation, the D-size field (RBS <5:4>) is encoded as follows and output onto the WBus:

D-Size RBS	<5:4>	Encoded Value
0	0	0001
0	1	0010
1	0	0100
1	1	1000

In addition, Table 2-40 indicates that during a READRBS operation, two status signals are generated. Refer to Paragraph 2.6.4.7 for a complete description of these status signals.

2.6.4.6 Register Number Register (RNUM) – The RNUM register is a 4-bit register contained within the SPA chip. The RNUM register is used to indirectly address a scratchpad register. As seen in Figure 2-75, the RNUM register can be loaded with a 4-bit number from the microsequencer (register number), the register backup stack (RBS), or the WBus. Loading is enabled by the MSRC field of the microword or a load signal (DPM17 IRD LD RNUM H) directly from the microsequencer. When the load signal is asserted by the microsequencer, the RNUM register is loaded with a number specified by DPM18 IRD RNUM <3:0> H. Otherwise the RNUM register is loaded as follows.

MSRC <4:0> (Hex)	Operation Specified	RNUM Contents
1D	RNUM WBUS	WBUS <3:0>
1C	READRBS	Register field of RBS

2.6.4.7 Scratchpad Status Signals – The SPA chip generates two status signals, SPASTA <1:0>, for microbranching. These signals are generated as a function of the MSRC and RSRC microfields.

When a GPR location is explicitly specified in the RSRC microfield or implicitly specified through the RNUM register, the status signals indicate the contents of the RNUM register as follows.

SPASTA <1:0>		RNUM Register Contents	GPR General Use
0	1	E	VAX mode SP
1	0	7	Compatibility mode PC
1	1	6	Compatibility mode SP
0	0	all other values	–

This makes it possible to identify the program counter (PC) and stack pointer (SP) from all other general purpose registers. The status signals are undefined for this case if the MSRC microfield specifies a READRBS, RNUM_WBUS, OR WB_RBSP operation (MSRC = 1C, 1D, or 1E).

When a GPR location is not specified by the RSRC microfield, the status signals are defined for the following operations only. These operations are specified in the MSRC microfield.

MSRC <4:0> (Hex)	Operation
1C	READRBS
1D	RNUM_WBUS
1E	WB_RBSP

If the MSRC microfield specifies a READRBS operation, and a GPR location is not specified by the RSRC microfield, the status signals are used to indicate an autoincrement or autodecrement mode. They specifically indicate bit 6 of the RBS location pointed to by RBSP.

SPASTA <1:0>		RBS <6>	Indicated Mode
0	0	0	autodecrement
1	0	1	autoincrement
0	1	–	–
1	1	–	–

If the MSRC microfield specifies a RNUM_WBUS operation, and a GPR location is not specified by the RSRC microfield, the status signals are used to identify particular IPR locations for the MTPR and MFPR instructions. They specifically indicate the scratchpad address that is loaded into the RNUM register as follows.

SPASTA <1:0>		WBUS <3:0>	IPR General Use
1	1	0–4	Processor control stack pointers
1	0	5–7, E, F	Reserved locations
0	0	8–D	All others
0	1	–	–

If the MSRC microfield specifies a WB_RBSP operation (and a GPR location is not specified by the RSRC microfield) the status signals are used to detect an empty RBS condition. For this case the RBSP is monitored and the status signals encoded as follows.

SPASTA <1:0>		RBSP	RBS Condition
0	1	0	Empty
0	0	All other values	Not empty
1	0	–	–
1	1	–	–

2.6.5 Arithmetic Section

The arithmetic section of the data path consists of the arithmetic/logical processor and associated control logic. Contents from the MBus, RBus, and SBus are input to the arithemtic/logical processor to allow the required arithmetic and logic operations to be performed during the execution of the macroinstructions. Results can be output on the WBus.

The arithmetic/logical processor (ALP) consists of eight ALP chips that perform the ALU functions of the data path. Each ALP chip processes a 4-bit slice to perform 32-bit arithmetic or logical operations. The ALP is discussed as a single unit throughout this chapter, unless otherwise specified.

The CLA chip (carry look-ahead chip) provides the appropriate carry or borrows for each of the cascaded ALUs within the ALP chips. The CLA hardware is transparent to the microcode. Refer to Paragraph 2.6.5.2 for a brief description of its functionality.

All functions within the ALP are controlled by the ALK chip (arithmetic/logical control chip). Refer to Paragraph 2.6.5.3 for a description of the ALK. Basically, the ALK decodes the 10-bit ALPCTL microfield to generate control signals for the ALP.

2.6.5.1 Arithmetic/Logical Processor (ALP) – Figure 2-79 illustrates a functional block diagram of the ALP. As seen in this figure, the ALP contains input latches, the S shifter, the ALU and its input and output multiplexers, BCD adjust logic, and the D and Q registers. These are discussed in the following paragraphs.

2.6.5.1.1 ALP Input Latches – Data is latched from the tri-state RBus and MBus and input to the ALU input multiplexers by dedicated feed-through latches. The latches are simultaneously clocked by the signal DPM11 DP PHASE H. Figure 2-80 illustrates the clock waveform.

2.6.5.1.2 S Shifter – The S shifter provides the second level shifting in association with the rotator section (see Paragraph 2.6.6). Although physically located in the ALP chips, the S shifter is functionally part of the rotator section. Data from the SBus is shifted right 0, 1, 2, or 3 bits by the S shifter and input to the B multiplexer. The number of bit positions to be shifted is determined by two signals (DPM09 SHF <1:0> L) from the rotator control logic (SRK chip). These signals are generated from the value of the ROT field in the microword that defines the rotator function. The number of bits to be shifted is specified as follows.

DPM09	SHF <1:0> L	Number of Bits Shifted Right
H	H	0
H	L	1
L	H	2
L	L	3

Note that the S shifter does not latch data from the SBus.

2.6.5.1.3 ALU A and B Input Multiplexers (A MUX and B MUX)

A MUX – The A input to the ALU is controlled by the A MUX. The A MUX is capable of selecting data from RAM-M or memory control interface registers (VA, PC, MDR) via the MBus, RAM-R via the RBus, or the D register. These registers hold data for use during instruction execution. When the contents of these registers must be manipulated or used in an ALU operation, the A MUX selects the correct source.

If the required data on the MBus is less than 32 bits, the data can be sign- or zero-extended. For this case, the A MUX selects the sign/zero-extended version of the MBus. The type of extension, sign or zero, is selected by bit <63> of the microword (0 = zero, 1 = sign). This bit defines the ALUXM subfield of the microword (Paragraph 2.6.6.1).

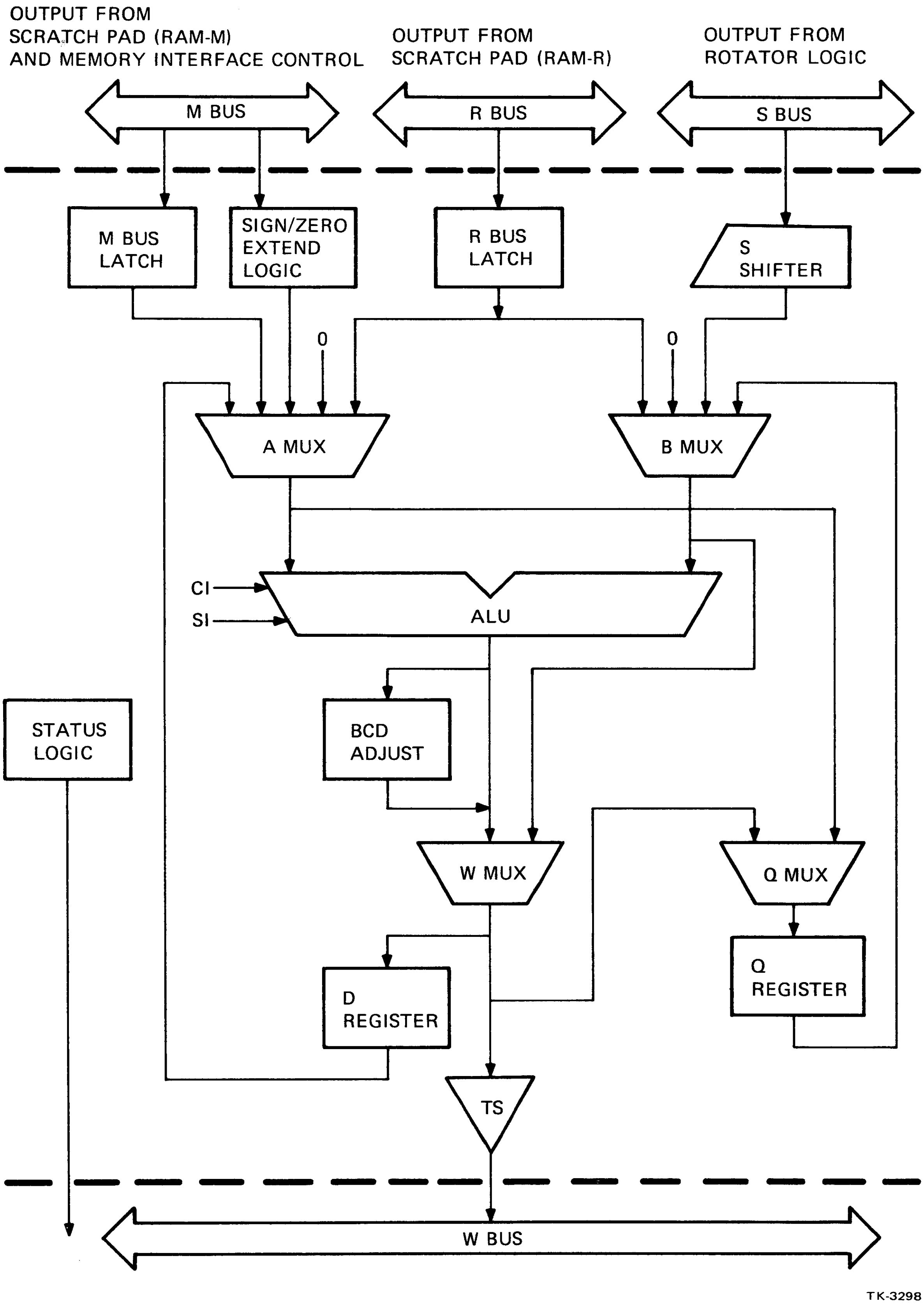


Figure 2-79 Arithmetic and Logical Processor (ALP)

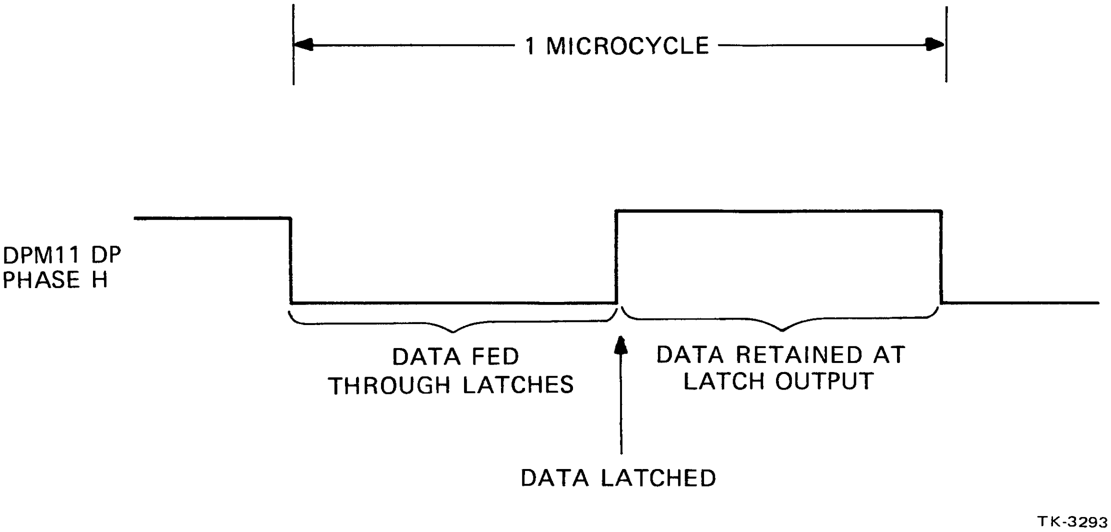


Figure 2-80 ALP Input Latch Timing

B MUX - The B input to the ALU is controlled by the B MUX. The B MUX is capable of selecting information from RAM-R via the RBus, the rotator section via the SBus and S shifter, or the Q register for use during instruction execution. If the required data is present on the RBus, the B MUX selects the R latch output.

A and B MUX Control - The A and B input multiplexers are usually controlled by signals from the ALK chip (Paragraph 2.6.5.3) as specified by the value in ALPCTL <9:6> of the microword. These bits define the MUX subfield of the microword. (Refer to Paragraph 2.2.1.2 for an explanation of subfields.) Table 2-41 lists the A MUX and B MUX selection for each subfield value.

2.6.5.1.4 Extended/Nonextended MBus Data - If the MBus data required for an ALP operation is less than 32 bits (i.e., a byte or word), the data can be sign- or zero-extended by the ALP logic.

Figure 2-81 illustrates the logic associated with extension of MBus data. As seen in this figure, both extended and nonextended versions of the latched MBus data are presented to the A MUX. The A MUX performs the appropriate selection. Refer to Paragraph 2.6.5.1.3 for a description of the A MUX.

The construction of the extended version of the MBus is controlled by several signals, as seen in Figure 2-81. These signals are directly related to the data size on the MBus. DPM19 D SIZE 1 H is one of two signals generated by the microsequencer to indicate data size (Paragraph 2.6.5.3). When this signal is low (the data size is a word or byte), the extended data input (DPM03 EXT DATA L) is used for the generation of bits 31:16. In addition, if the data type is a byte, DPM10 X <15:08> EN L is asserted to select the extended data input for the generation of bits <15:08>. DPM10 X <15:08> EN L is asserted by the ALK chip (Paragraph 2.6.5.3) when the D-size signals are both low (i.e., data size = byte).

Figure 2-82 illustrates the multiplexer used in the selection of the extended data input. Note that this multiplexer is external to the ALP.

Table 2-41 A and B Multiplexer Control

MUX Subfield ALPCTL <9:6> (Hex)	A MUX Data	B MUX Data
0	MBus	RBus
1	MBus	RBus
2	MBus	Q Register
3	MBus	Q Register
4	MBus	S Shifter
5	Extended MBus	RBus
6	Extended MBus	Q Register
7	Extended MBus	S Shifter
8	D Register	RBus
9	D Register	RBus
A	D Register	Q Register
B	D Register	Q Register
C	D Register	S Shifter
D	0	S Shifter
E	RBus	Q Register
F	RBus	S Shifter

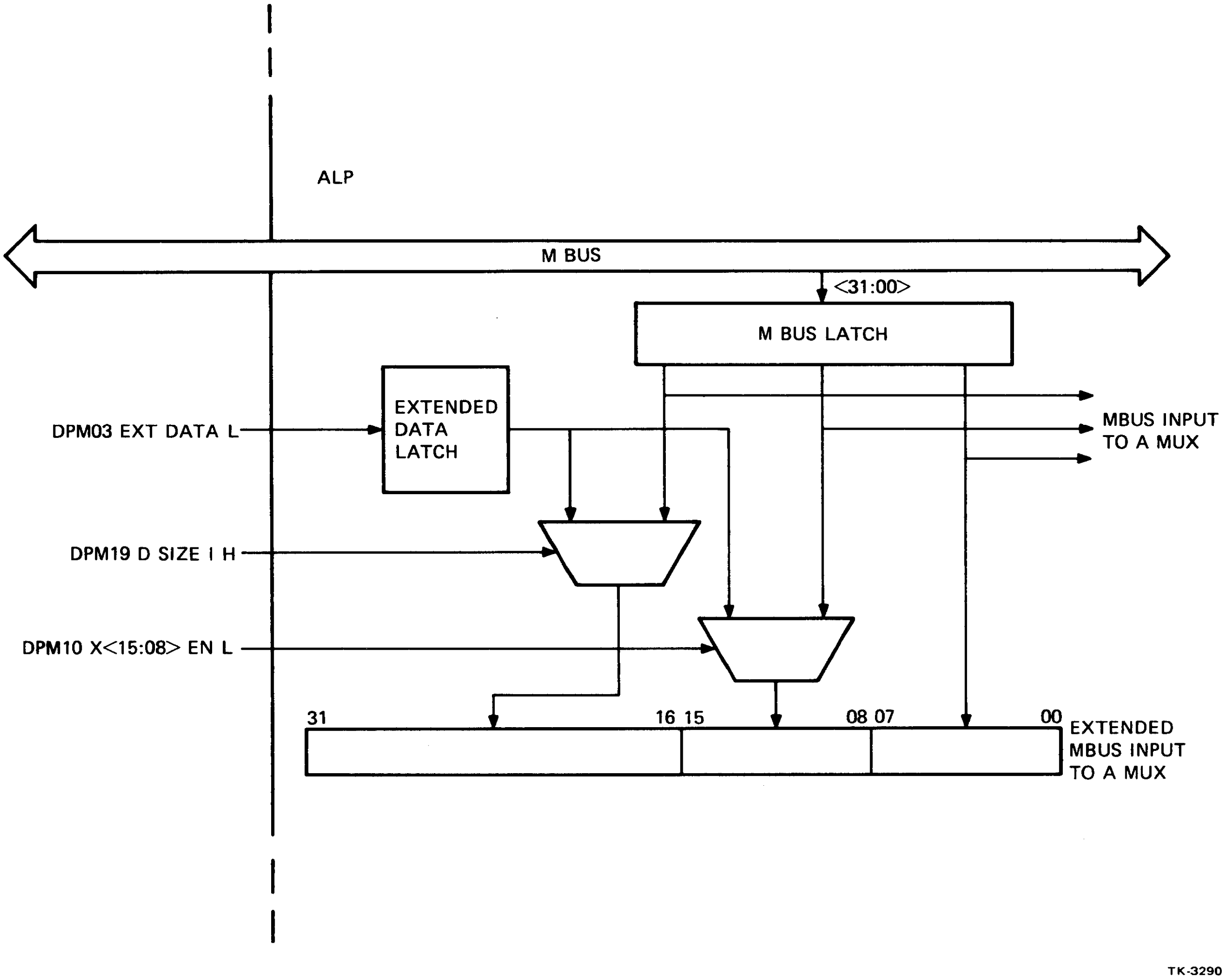
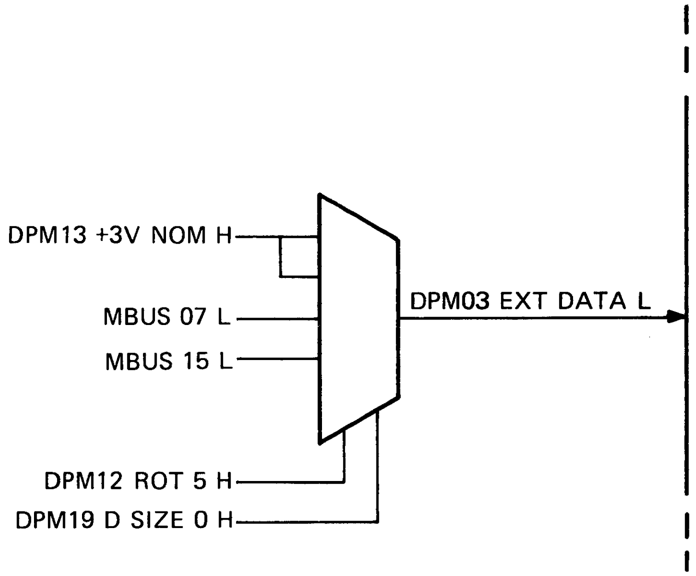


Figure 2-81 Extended MBus Data



ALUXM (ROT 5)
L – ZERO EXTEND
H – SIGN EXTEND

D SIZE 0
L – BYTE
H – WORD

ALUXM (ROT 5)	D SIZE 0	EXT DATA
L	L	0
L	H	0
H	L	MBUS 07
H	H	MBUS 15

TK-3287

Figure 2-82 Extended Data Selection

The type of extension, sign or zero, is determined by bit <63> of the microword. This bit defines the ALUXM subfield of the microword. It is cleared to indicate zero-extend and set to indicate sign-extend. If sign-extend is indicated, the sign value (plus or minus) must be derived from the most significant bit of the data type. For this case, a D-size signal is used to select bit 07 if the data type is a byte, or bit 15 if the data type is a word. (The D-size signals indicate data type and are generated by the microsequencer, Paragraph 2.6.5.3.) The selected bit is then input to the ALP for the sign extension. If a zero extension is selected, a zero (DPM13 +3 V NOM H) is input to the ALP.

2.6.5.1.5 Arithmetic and Logical Unit (ALU) – The ALU is the main processing unit of the ALP logic. It performs 32-bit arithmetic or logical functions.

The ALU operation is usually selected by ALPCTL <5:2> of the microword. These bits define the ALU or ALUOD subfield of the microword depending on their value and the value of the MUX subfield (ALPCTL <9:6>). (Refer to Paragraph 2.2.1.2 for an explanation of subfields.) Table 2-42 shows the subfield interpretation of ALPCTL <5:2> and the selected ALU function for each value. Terms listed under ALU function are defined in Table 2-43.

Table 2-42 ALU Control

ALPCTL <5:2> (Hex)	Subfield Interpretation	ALU Function
0	ALU	A−B−CI
1	ALU	A−B−CI, BCD
2	ALU	(A−B−CI).SR
3	ALU	(A−B−CI).SL
4	ALU	A+B+CI
5	ALU	A+B+CI, BCD
6	ALU	(A+B+CI).SR
7	ALU	(A+B+CI).SL
8	ALU or ALUOD*	A.AND.B
9	ALU or ALUOD‡	A.OR.B
A	ALU or ALUOD†	(A.AND.B).SR
B	ALU or ALUOD†	(A.AND.B).SL
C	ALU or ALUOD*	B−A−CI
D	ALU or ALUOD‡	A.XOR.B
E	ALU	A.AND.(.NOT.B)
F	ALU	(.NOT.A).AND.B

*ALUOD only if ALPCTL <9:6> = 9 (hex)
†ALUOD only if ALPCTL <9:6> = D (hex)
‡Either of the above

Table 2-43 ALU Mnemonic Definitions

Mnemonic	Definition
A	A input
B	B input
CI	Carry input
BCD	Binary coded decimal
SR	Shift right
SL	Shift left

ALU Carry-In – Specification of the ALU carry-in depends on the B MUX selection. As long as the MUX subfield (ALPCTL <9:6>) does not contain a value of 4, 7, C, D, or F, the carry-in is specified by bits <59:58> of the microword. These bits define the ALUCI subfield of the microword. The ALUCI subfield specifies the ALU carry input as follows.

ALUCI Subfield (ROT <1:0>)	ALU Carry-In
00	0
01	ALKC flag
10	1
11	PSL <C>

The ALKC flag is located in the ALK chip and is used to save the carry or borrow from the ALU during an add or subtract. PSL <C> refers to the C bit of the processor status longword (bit 00).

If the MUX subfield contains a value of 4, 7, C, D, or F, the carry-input is defaulted to a hard-wired zero. These values indicate that a rotator function must be specified by bits <63:58> of the microword (the ROT microfield). Bits <59:58> can therefore not specify the carry-in. The carry input is also defaulted to a hardwired zero if the ROT microfield (ROT <5:0>) specifies a function that modifies the P latch or S latch. This condition exists when ROT <5:0> = 27, 2D, 2F, 3B, 3D, or 3F.

The ALK chip decodes ROT <5:0> and enables the appropriate carry input for the ALP. (Refer to Paragraph 2.6.5.3 for a complete description of the ALK chip.)

ALU Shift-In – As seen in Table 2-42, the ALU can shift the result of an add, subtract, or AND operation by one bit. Specification of the ALU shift-in depends on the B MUX selection. As long as the MUX subfield (ALPCTL <9:6>) does not contain a value of 4, 7, C, D, or F, the shift-in is specified by bits <62:60> of the microword. For this case these bits define the ALUSHF subfield of the microword. Table 2-44 lists the shift-in for each value of the ALUSHF subfield. As seen in this table, the ALUSHF subfield also specifies the shift-in for Q register shifts. (See Paragraph 2.6.5.1.7 for a description of the Q register.)

Table 2-44 ALU and Q Register Shift-In

ALUSHF Subfield (ROT <4:2>)	ALU Shift-In	Q Register Shift-In
000	0	0
001	1	1
010	(Note 1)	(Note 1)
011	(Note 2)	(Note 2)
100	0	1
101	1	0
110	WBUS <30>	WBUS <30>
111	PSL <C>	PSL <C>

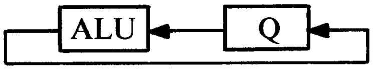
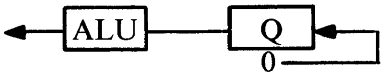
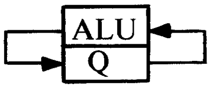
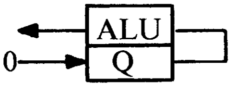
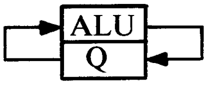
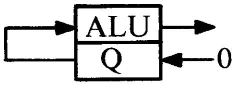
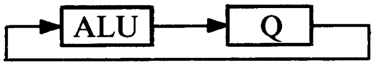
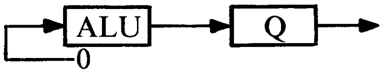
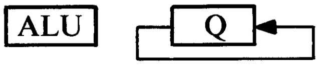
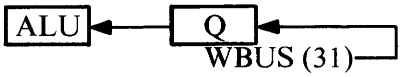
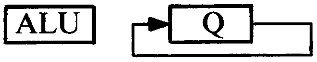
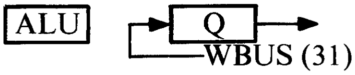
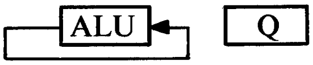
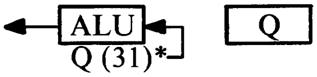
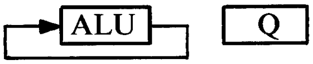
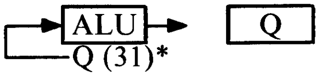
NOTES

1. **This shift-in depends on the shift operation of both the ALU and Q register as shown in Table 2-45 under shift.**

2. **This shift-in depends on the shift operation of both the ALU and Q register as shown in Table 2-45 under rotate.**

When the value of the ALUSHF subfield equals 010 or 011, the shift-in for the ALU and Q register depends on the type of shift (right or left) specified for each. For this case the shift-in is determined as shown in Table 2-45. As mentioned above, the type of shift for the ALU is selected by the ALU or ALUOD subfield of the microword (ALPCTL <5:2>). The type of shift for the Q register is selected by the DQ subfield (ALPCTL <1:0>).

Table 2-45 ALU and Q Shift-in Special Cases

ALU Shift	Q Register Shift	ALUSHF = 011 (Rotate)	ALUSHF = 010 (Shift)
Left	Left		
Left	Right		
Right	Left		
Right	Right		
None	Left		
None	Right		
Left	None		
Right	None		

*Q <31> is undefined for any load Q function.

Just as for the ALU carry-inputs, the shift inputs are defaulted to 0 when either of the following conditions exists.

1. The B MUX selects the rotator (S shifter) for input to the ALU. This condition is specified when the MUX subfield (ALPCTL <9:6>) = 4, 7, C, D, or F.

2. The ROT microfield specifies a function that modifies the P latch or S latch. This condition exists when ROT <5:0> = 27, 2D, 2F, 3B, 3D, or 3F.

2.6.5.1.6 BCD Adjust Logic – When the ALU subfield (ALPCTL <5:2>) specifies a BCD operation, the output of the ALU may or may not have to be adjusted to form legal BCD digits (0 through 9). Dedicated logic in each ALP chip automatically computes the appropriate adjustment for the corresponding 4-bit ALU output.

2.6.5.1.7 D and Q Registers – The D register is a 32-bit holding register for the intermediate result of an ALU operation. The D register is loaded from the W MUX and provides data to the A MUX. Similarly, the Q register is a 32-bit holding register that is capable of a right or left shift by one bit. The Q register is loaded from the Q MUX, which can select the output from the W MUX or A MUX. The output of the Q register is input to the B MUX.

The loading of the D and Q registers is controlled by two bits in the microword, ALPCTL <1:0>. These two bits define one of three types of DQ subfields providing a special function is not specified by ALPCTL <9:0>. (See Paragraph 2.6.5.4.) The type of DQ subfield is determined by the value of the MUX subfield (ALPCTL <9:6> and is selected as shown in Table 2-46.

Table 2-47 shows the relationship between the DQ subfield value and register control. As seen in this table, the DQ subfield not only controls the loading of the D and Q registers, but also determines whether the Q register is to be shifted. The direction of the shift is also specified.

Table 2-46 DQ Subfield Types

MUX Subfield (ALPCTL <9:6>	**Subfield Interpretation of ALPCTL <1:0>**
0, 2, 4, 5, 6, 7, 8, A, C, E, F	DQ1
1, 3, B	DQ2
9	DQ3

Table 2-47 D and Q Register Control

DQ Subfield	**Subfield Values**			
	0	**1**	**2**	**3**

DQ1	NOP	Q ← W MUX	D ← W MUX	Q ← W MUX
				D ← W MUX

DQ2	SHF Q LEFT	SHF Q RIGHT	SHF Q LEFT D ← W MUX	SHF Q RIGHT D ← W MUX
DQ3	SHF Q LEFT D ← W MUX	SHF Q RIGHT D ← W MUX	Undefined	Undefined

If the DQ subfield specifies a Q-register shift, the value to be shifted into the vacant position is selected by bits <62:60> of the microword. These bits define the ALUSHF subfield, which is also used to select the shift-in for ALU functions. Table 2-45 lists the shift-in for the Q register as well as the ALU.

2.6.5.1.8 W Multiplexer (W MUX) – The W MUX selects the output of the ALU or B MUX for input to the Q MUX, D register, and WBus. The W MUX is controlled by the ALPCTL field of the microword, which is used to define an ALP special function. (See Paragraph 2.6.5.4 for a description of special functions.) The ALU output is selected for most values of this field. The B MUX output is only selected when the following special functions are specified.

ALPCTL <9:0> (Hex)	ALP Special Function Mnemonic
047	WX_R.Q_M
0C7	WX_Q.Q_M
147	WX_R.Q_XM
1C7	WX_S.Q_XM
247	WX_R.Q_D
2C7	WX_Q.Q_D
347	WX_S.Q_D
3C7	WX_S.Q_R
057	WX_D_R.Q_M
0D7	WX_D_Q.Q_M
157	WX_D_R.Q_XM
1D7	WX_D_S.Q_XM
257	WX_D_R.Q_D
2D7	WX_D_Q.Q_D
357	WX_D_S.Q_D
3D7	WX_D_S.Q_R

When the ALPCTL field specifies one of the functions listed above, the ALU output is ignored. The ALP status signals, however, remain valid.

The output of the W MUX is normally routed onto the WBus. This sourcing, however, is inhibited when an ALUOD function is specified. Refer to Paragraph 2.6.5.1.5 for a description of ALUOD functions.

2.6.5.1.9 ALP Status Logic – Three types of status signals are generated by the ALP logic to set condition codes and execute microbranches. Table 2-48 lists the status signals according to type and gives a brief description of their meaning. Note that the precise definitions of the overflow and carry signals depend on the ALU operation performed. The conditions for the assertion of a carry signal are listed in Table 2-49. Likewise, Table 2-50 lists the conditions for each overflow signal.

2.6.5.2 Carry Look-Ahead (CLA) Functionality – The carry look-ahead (CLA) chip provides the necessary carry or borrows between each of the cascaded ALUs within the ALP chips. The CLA function should not be confused with the ALU carry-in described in Paragraph 2.6.5.1.5 or the ALU carry status described in Paragraph 2.6.5.1.9. These sections are concerned with the carry result of an arithmetic or logical operation rather than the carry or borrow generated between each 4-bit ALU slice.

When the A and B inputs have been selected for an arithmetic operation, each ALP chip generates signals to indicate which adjoining slices require a borrow or carry. The CLA chip monitors these signals and generates a carry input for the appropriate slices.

The signals monitored by the CLA chip consist of two types – propagate and generate. Each ALP chip has its own propagate and generate line to the CLA chip. The CLA chip determines the proper carry input (condition of the ALUC signal) for each ALP chip by decoding the signals on these lines.

Table 2-48 ALP Status Signals

Status Signal Type	Status Signals	Interpretation
WMUXZ	WMUXZ B0 H WMUXZ B1 H WMUXZ B2 H WMUXZ B3 H	Indicates the corresponding byte of the W MUX output is all zeros.
ALU Overflow	ALUV 07 H ALUV 15 H ALUV 31 H	Indicates the result of the arithmetic operation cannot be represented by the corresponding data type (i.e., overflow condition).
ALU Carry	ALUC 07 L ALUC 15 L ALUC 31 L	Indicates a carry has been generated for the corresponding data type as a result of the ALU operation.

Table 2-49 Conditions for Carry Status

ALU Operation	Carry Status Signal ALUC <31> L	ALUC <15> L	ALUC <07> L
Binary Add	ALUC <n> L if (A <n:00> + B <n:00> + CI)* • $2^{(n+1)}$		
Binary Subtract	ALUC <n> L if A <n:00> • (B <n:00> + CI)*		
BCD Add	Asserted if A+B+CI • 99,999,999	Undefined	Undefined
BCD Subtract	Asserted if A•(B+CI)*	Undefined	Undefined
Logical (any)	Undefined	Undefined	Undefined

*Unsigned arithmetic

Table 2-50 Conditions for Overflow Status

ALU Operation	Overflow Status Signals		
	ALUV <31> H	**ALUV <15> H**	**ALUV <07> H**
Binary (Any)	$C_{31} + C_{30}$	$C_{15} + C_{14}$	$C_7 + C_6$
BCD (Any)	Not Asserted	Not Asserted	Not Asserted
Logical (Any)	Not Asserted	Not Asserted	Not Asserted

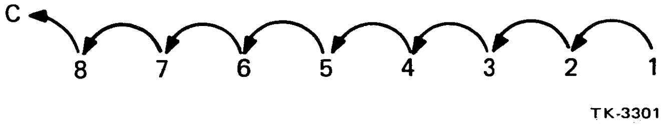
Table 2-51 lists the conditions for the generation of a propagate signal and generate signal. Note that the assertion of a signal depends on the selected ALU operation and the relationship of the A and B ALU inputs. These A and B inputs refer only to the associated 4-bit slice.

Table 2-51 Propagate/Generate Signals

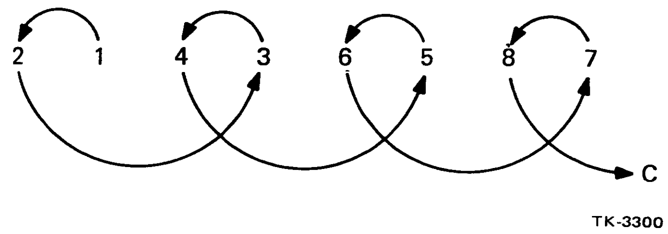
ALU Operation	Propagate Signal Asserted If:	Generate Signal Asserted If:
Binary Add	$A+B=F_{16}$	$A+B \rangle F_{16}$
Binary Subtract	$A=B$	For $A-B$, $A \rangle B$ For $B-A$, $B \rangle A$
BCD Add	$A+B=9$	$A+B \rangle 9$
BCD Subtract	$A=B$	$A \rangle B$
Logical (any)	Undefined	Undefined

The CLA chip also monitors a BCD indicator signal from the ALK chip. This signal indicates whether the current ALP operation is BCD or not. The CLA chip uses this information to propagate the correct carries for the ALP chips. The following two examples illustrate the carry propagation for a BCD and non-BCD operation. Each numeric digit represents a nibble (4 bits). The most significant digit is 8; the least significant digit is 1.

The carry propagation for a non-BCD number is performed as follows.

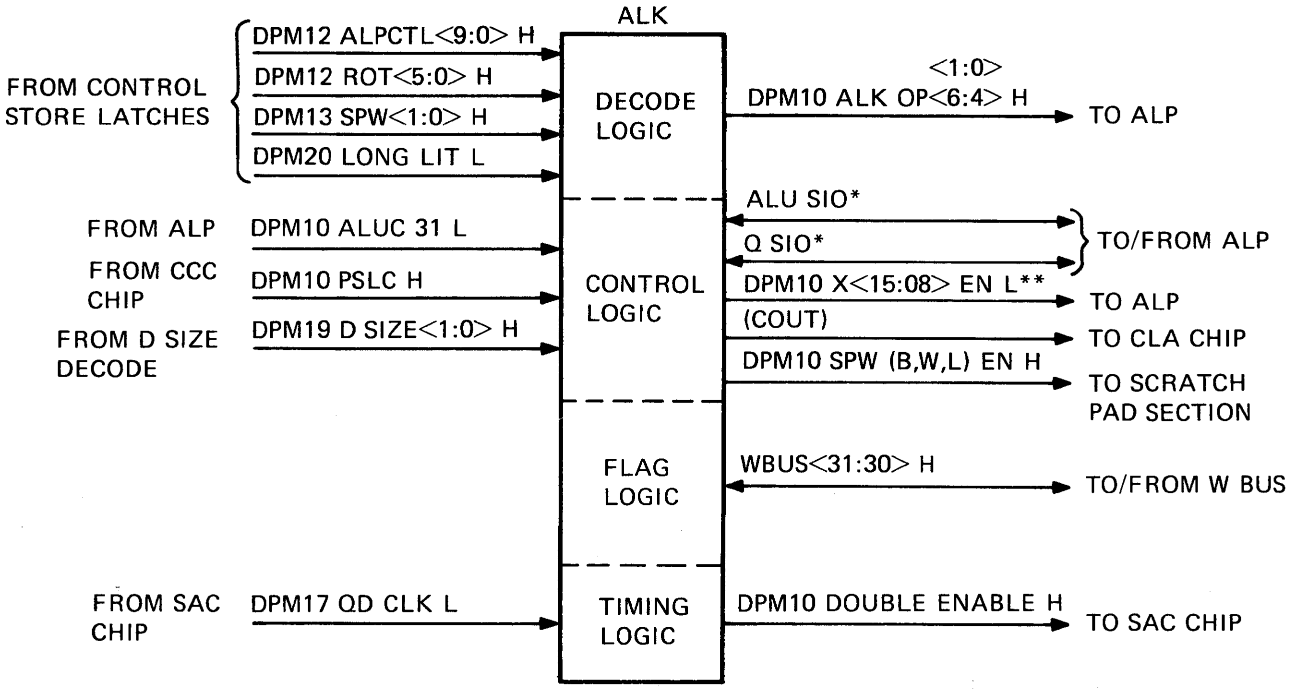


The carry propagation for a BCD number is performed as follows.



2.6.5.3 ALK Logic – The arithmetic/logical control (ALK) chip controls all functions within the ALP. Among these functions are data input selection, ALU operation, carry input selection, and shift input selection.

The logic of the ALK chip can be divided into four sections as illustrated in Figure 2-83. Each input and output of the ALK can be associated with one of these sections. The sections are decode, control, flag, and timing (see Paragraphs 2.6.5.3.1–2.6.5.3.4).



*ALU SIO IS ACTUALLY: ALU SIO 31 L, ALU SIO 00 L.
Q SIO IS ACTUALLY: Q SIO 31 L, Q SIO 15 L, Q SIO 07 L, Q SIO 00 L.
**DPM10 X<15:08> EN L IS ONE SIGNAL. (15:08 IS PART OF THE SIGNAL NAME)

Figure 2-83 ALK Chip

2.6.5.3.1 Decode Logic – The ALK decodes various microfields to specify ALP operations and to control its own internal operations. DPM12 ALPCTL <9:0> H are decoded to generate the basic opcode for the ALP (DPM10 ALK OP <6:4, 1:0> H). These signals are sent to each ALP chip to specify the basic ALP operation. DPM12 ROT <5:0> H are also decoded to specify the shift-in and carry-in for the ALU and Q register of the ALP. The selection of a shift-in and carry-in is described in Paragraph 2.6.5.1.5. DPM13 SPW <1:0> H is decoded to enable writes to the scratchpad (Paragraph 2.6.4.4). Note that normal decoding of the ALPCTL and ROT microfields is disabled in the ALK chip when the LIT microfield specifies a long literal operation (LIT <1:0> = 11). For this case the following conditions are forced.

1. DPM10 ALK OP <6:4, 1:0> H is set to LHHLL. This disables D and Q register operations and ALU shifts.

2. BCD operations are disabled.

3. The ALK flags are affected as follows.

   ALKC flag remains intact
   ALUSO flag remains intact
   Loop flag is cleared
   TOG flag is undefined

2.6.5.3.2 Control Logic – The results of the decode enable operations in the control section of the ALK chip. These operations include the control of shift inputs for the ALU and Q register, sign/zero extension of MBus data, and the enabling of scratchpad writes.

D-Size Signals – The ALP can execute operations on byte, word, and longword data types. The specific data type for an operation is defined by two signals generated by the microsequencer, D-Size <1:0> H. The D-size signals are input to the ALK chip for this reason. Here they are used to determine the data size for writes in addition to the bit position for sign/zero-extension of MBus data in the ALP chips. The D-size signals are also used to specify the data type for bus functions and to set condition codes. Refer to Paragraph 2.6.3 for a more complete description of D-size signals and their use.

Write Enable Signals – The ALK chip generates three signals that control the amount of data written into a specified scratchpad location. The three signals are DPM10 SPWB EN H, DPM10 SPWW EN H, and DPM10 SPWL EN H. The generation of these signals is determined by the value in the SPW microfield and the D-size signals. Refer to Paragraph 2.6.4.4 for a complete description.

Extend Enable Signal – DPM10 X <15:08> EN L is a single signal generated by the ALK chip to enable sign/zero-extension of MBus data. It is generated as a result of decoding the D-size signals. Refer to Paragraph 2.6.5.1.4 for a complete description of MBus extension.

Shift-In/Out Control – The ALK controls the selection of the shift inputs to the ALU and Q register of the ALP. The appropriate bit positions are available to the ALK via dedicated lines. This concept is illustrated in Figure 2-84. With these lines, the shift input can be selected and transferred to the ALP. Selection of the shift input is determined by the ALUSHF subfield of the microword (ROT <5:2>) as described in Paragraph 2.6.5.1.5. Shift inputs include 0, 1, WBUS <30>, and PSL <C>. The ALK is also capable of interconnecting the transfer lines to execute the rotate functions described in Paragraph 2.6.6.

In addition to providing a path for shift inputs to the ALP, these bidirectional lines make it possible to store shift-outs. Whenever an ALU shift is performed, the lost bit is transferred to the ALK to be stored in the ALUSO flag.

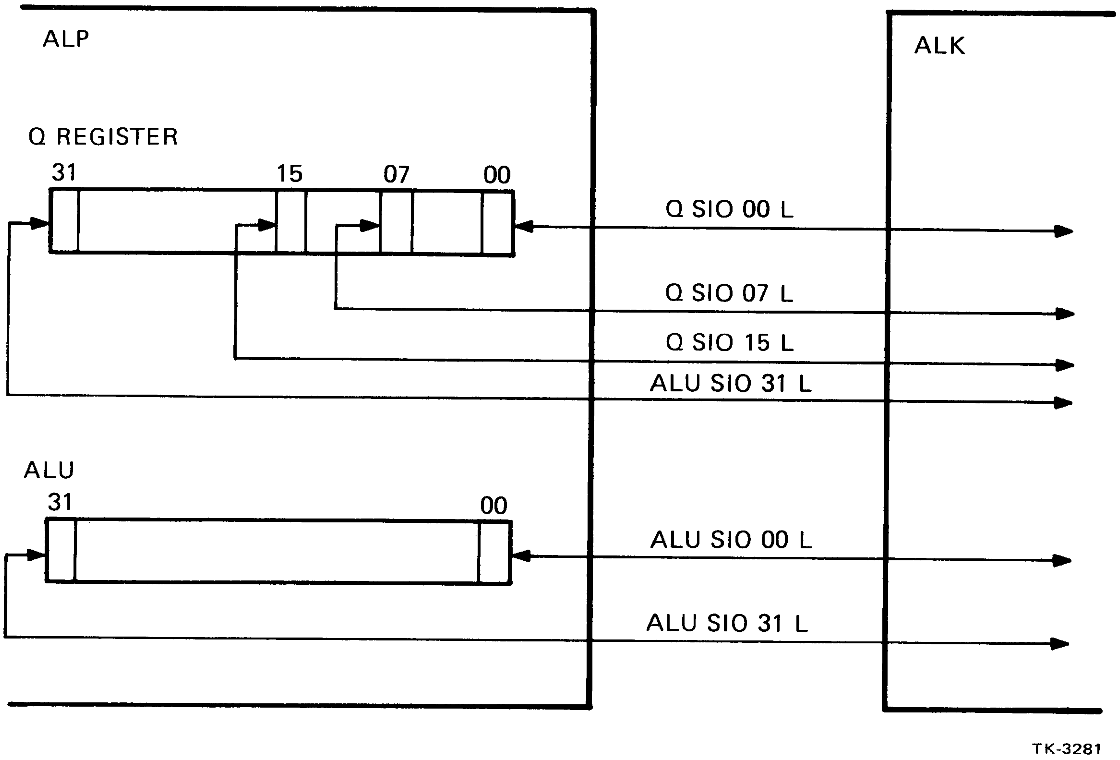


Figure 2-84 Shift-In/Out Lines

2.6.5.3.3 Flag Logic – The ALK logic includes four flags for use during the execution of certain arithmetic operations: the ALKC flag, ALUSO flag, Loop flag, and TOG flag. Each of these flags, except TOG, can be directly accessed via microcode. When enabled, the appropriate flag is set at the end of the microcycle. Each of these four flags is described below.

ALKC Flag – The ALKC flag is loaded with the resultant carry or borrow when the ALU subfield of the microword specifies an ALU add or subtract operation.

During a multiple-length add, the carry output from each ALU operation is saved by the ALKC flag to provide a carry input to the subsequent add. For example, during a 64-bit add, the resultant carry from the first 32-bit add is retained by the ALKC flag. This flag is then used as the carry input for the second 32-bit add. The ALKC flag is likewise used to retain the resultant borrow during each iteration of a multiple-length subtract operation. Note that for both add and subtract, the carry or borrow is always derived from the most signficant bit position.

The ALKC flag is sourced onto the WBus (WBUS <30>) when the ALPCTL microfield equals 37C, 37D, 37E, or 37F (Paragraph 2.6.5.4).

ALUSO Flag – The ALUSO flag is loaded with the bit shifted out of the ALU when an ALU shift function is specified by the ALU subfield of the microword. On an ALU shift left operation, the ALUSO flag is loaded with the data shifted out from ALU <31>. During an ALU shift right operation, the ALUSO flag is loaded with the data shifted out from ALU <00>.

The ALUSO flag is also loaded during the shifting associated with various special functions. During each iteration of the MULFAST and MULSLOW functions, the ALUSO flag is loaded with ALU <00>. Likewise, the ALUSO flag is loaded with ALU <31> during each iteration of the DIVFAST and DIVSLOW function. Note, however, that the flag remains intact for the special divide functions. Refer to Paragraph 2.6.5.4 for a complete description of each of these special functions.

The ALUSO flag is sourced onto the WBus (WBUS <31>) when the ALPCTL microfield equals 37C, 37D, 37E, or 37F.

Loop Flag – The Loop flag is set when the ALPCTL microfield specifies a multiplication or division operation. For the execution of these operations, an ALU function is repeated several times consecutively. The Loop flag indicates that a multiplication or division loop is in progress and must not be interrupted. Refer to Paragraph 2.6.5.4 for a complete description of the multiply and divide operations.

The Loop flag is sourced onto the WBus (WBUS <30>) when the ALPCTL microfield equals 378, 379, 37A, or 37B. When this occurs, the condition of the Loop flag remains intact.

TOG Flag – The TOG flag is used to control the ALU during multiplication and division. During each iteration of a multiply function, the TOG flag is loaded with bit 00 of the Q register. In this case, the TOG flag is used to control the ALU inputs. Similarly during each iteration of a divide function, the TOG flag is loaded with the ALU carry bit (or 1's complement). In this case, the TOG flag is used to select an ALU add or subtract. Refer to Paragraph 2.6.5.4 for a complete description of the multiply and divide functions.

Note that unlike the ALKC, ALUSO, and Loop flags, the TOG flag cannot be sourced onto the WBus.

2.6.5.3.4 Timing Logic – The ALK and ALP chips are clocked by the signal DPM17 QD CLK L. This signal is generated by the logic associated with the SAC chip located on the DPM module.

The QD clock pulse is usually generated once every microcycle. If the ALK decodes a MULFAST on DIVFAST special function, however, DPM10 DOUBLE ENABLE H is asserted. This enables the number of QD clock pulses to 2 per microcycle.

2.6.5.4 ALP Special Functions – The ALP logic provides the capability of executing special functions in addition to the basic ALU and data routing operations. The capability allows the ALP to execute a complex operation at the specification of a single microfield value. (A complex operation is defined as an operation that involves several ALU and/or data routing operations, such as multiply, divide, etc.)

Special functions are selected by the ALPCTL microfield (bits <9:0>). When a special function is specified in this microfield, all corresponding subfields are ignored (i.e., the ALU, ALUOD, MUX, DQ1, DQ2, and DQ3 subfields are ignored.) Each logic element involved in the specified operation is implicitly controlled by the value in ALPCTL <9:0>. For the execution of some special functions, a single value must remain in this microfield for several microinstructions.

The special functions can be divided into five groups. Table 2-52 lists each function according to group and gives a brief description of each.

The multiply and divide special functions are described in the following paragraphs because of their relative complexity. Many of the concepts discussed can be applied to both types of functions. Among these concepts are: the definition of iteration, the difference between FAST and SLOW, sign consideration, etc.

2.6.5.4.1 Multiply Algorithm – The special functions of the mutliply group are used to perform unsigned multiplication of two integers, each containing up to 32 bits. The multiplicand, however, is treated as positive or negative, depending on the type of multiply function invoked (Table 2-52).

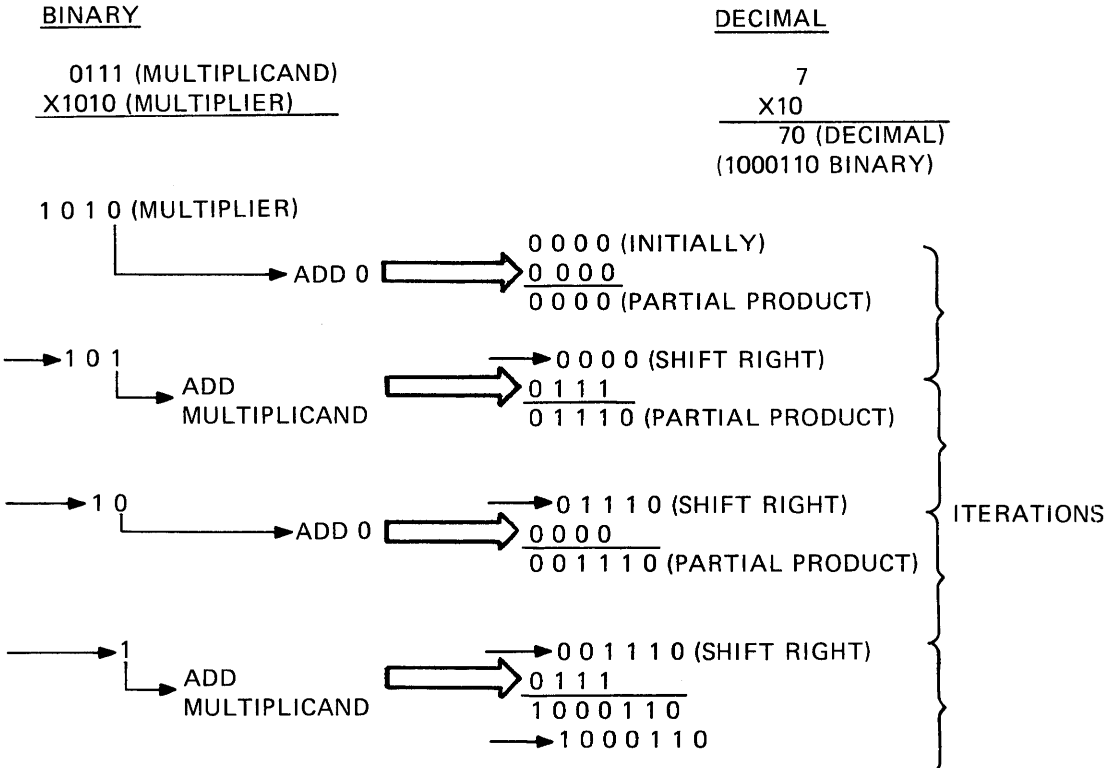
Table 2-52 ALP Special Functions

Special Function Group	ALPCTL (57:48) Hex	Mnemonic	Description
Data Routing	047	WX_R.Q_M	W MUX ← RBus Q ← MBus
	0C7	WX_Q.Q_M	W MUX ← Q (old) Q ← MBus
	147	WX_R.Q_XM	W MUX ← RBus Q ← S/Z MBus
	1C7	WX_S.Q_XM	W MUX ← SBus Q ← S/Z MBus
	247	WX_R.Q_D	W MUX ← RBus Q ← D
	2C7	WX_Q.Q_D	W MUX ← Q (old) Q ← D
	347	WX_S.Q_D	W MUX ← SBus Q ← D
	3C7	WX_S.Q_R	W MUX ← SBus Q ← RBus
WBus Disable	057	WX_D_R.Q_M	W MUX & D ← RBus Q ← MBus
	0D7	WX_D_Q.Q_M	W MUX & D ← Q (old) Q ← MBus
	157	WX_D_R.Q_XM	W MUX & D ← RBus Q ← S/Z MBus
	1D7	WX_D_S.Q_XM	W MUX & D ← SBus Q ← S/Z MBus
	257	WX_D_R.Q_D	W MUX & D ← RBus Q ← D (old)
	2D7	WX_D_Q.Q_D	W MUX & D ← Q (old) Q ← D (old)
	357	WX_D_S.Q_D	W MUX & D ← SBus Q ← D (old)
	3D7	WX_D_S.Q_R	W MUX & D ← SBus Q ← RBus
SBus Output	370	WX_S	W MUX ← SBus
	371	WX_Q_S	W MUX & Q ← SBus
	372	WX_D_S	W MUX & D ← SBus
	373	WX_D_Q_S	W MUX & D & Q ← SBus
	360	WX_.NOT.S	W MUX ← SBus
	361	WX_Q_.NOT.S	W MUX & Q ← SBus
	362	WX_D_.NOT.S	W MUX & D ← SBus
	363	WX_D_Q_.NOT.S	W MUX & D & Q ← SBus
Flag Output	378	WB_LOOPF	WB(31) ← 0, WB(30) ← LOOP
	379	WB_LOOPF.Q_0	WB(31) ← 0, WB(30) ← LOOP, Q ← 0
	37A	WB_LOOPF.D_0	WB(31) ← 0, WB(30) ← LOOP, D ← 0
	37B	WB_LOOPF.Q_D_0	WB(31) ← 0, WB(30) ← LOOP, Q & D ← 0
	37C	WB_ALUF	WB(31) ← ALUSO, WB(30) ← ALKC
	37D	WB_ALUF.Q_S	WB(31) ← ALUSO, WB(30) ← ALKC, Q ← S
	37E	WB_ALUF.D_S	WB(31) ← ALUSO, WB(30) ← ALKC, D ← S
	37F	WB_ALUF.Q_D_S	WB(31) ← ALUSO, WB(30) ← ALKC, Q & D ← S
Multiply	279	MULFAST+	Multiply +RBus by Q (2 iterations/cycle)
	27B	MULSLOW+	Multiply +RBus by Q (1 iteration/cycle)
	269	MULFAST−	Multiply −RBus by Q (2 iterations/cycle)
	26B	MULSLOW−	Multiply −RBus by Q (1 iteration/cycle)
Divide	26C	DIVFAST+	Divide Q by +RBus (2 iterations/cycle)
	26E	DIVSLOW+	Divide Q by +RBus (1 iteration/cycle)
	27C	DIVFAST−	Divide Q by −RBus (2 iterations/cycle)
	27E	DIVSLOW−	Divide Q by −RBus (1 iteration/cycle)
	26A	REM	Assemble Remainder (RBus = 0)
	27F	DIVDA	Double Divide, + Divisor
	26F	DIVDS	Double Divide, − Divisor

The multiply algorithm employed results in the generation of several intermediate partial products before the generation of the final product. The operation repeated for the generation of each partial product is referred to as a multiply iteration.

During each iteration of the multiply algorithm a basic add and shift is performed. Figure 2-85 illustrates an example of the multiply algorithm. Note that a 4-bit data type is used strictly for discussion purposes (i.e., ALU operations cannot be executed on data types of this length). As seen in this figure, an iteration is executed for each bit of the multiplier. Basically, during an iteration, the least significant bit of the multiplier is examined. If the bit equals 1, the magnitude of the multiplicand is added to the partial product. If the bit equals 0, zero is added. The new partial product and multiplier are then shifted to the right in preparation for the next iteration.

The same sequence is repeated for each iteration. Because of this, the multiply function is often referred to as a loop. Once initiated, this loop must not be interrupted until the multiply function is complete. The multiply loop is maintained as long as the ALPCTL microfield specifies the multiply function.



NOTE:
THIS FIGURE ILLUSTRATES THE MULTIPLY ALGORITHM AND IS NOT INTENDED TO DEMONSTRATE AN EXECUTABLE EXAMPLE.

TK-3282

Figure 2-85 Example of Multiply Algorithm

2.6.5.4.2 Hardware Implementation of Multiply – This paragraph describes the hardware implementation of the multiply algorithm described in Paragraph 2.6.5.4.1. Each of the following subsections of the paragraph describes an aspect of the multiply operation.

For the execution of any multiply function, the magnitude of the multiplier is loaded into the Q register and the multiplicand is placed on the RBus. With these inputs, the magnitude of the product is accumulated in the D and Q registers.

The data on the RBus (multiplicand) is treated as positive or negative, depending on the type of multiply function selected. For a positive multiplicand, a MULFAST+ or MULSLOW+ function is selected. For a negative multiplicand, a MULFAST– or MULSLOW– function is selected.

MULFAST vs. MULSLOW – The distinction between MULFAST and MULSLOW is time-related. During the execution of a MULSLOW function, one multiply iteration is executed every microcycle. For this type of multiply operation, the D clock is used. The MULFAST function executes two consecutive multiply iterations every microcycle. For this type of multiply operation, the B clock is used. Figure 2-86 illustrates this concept.

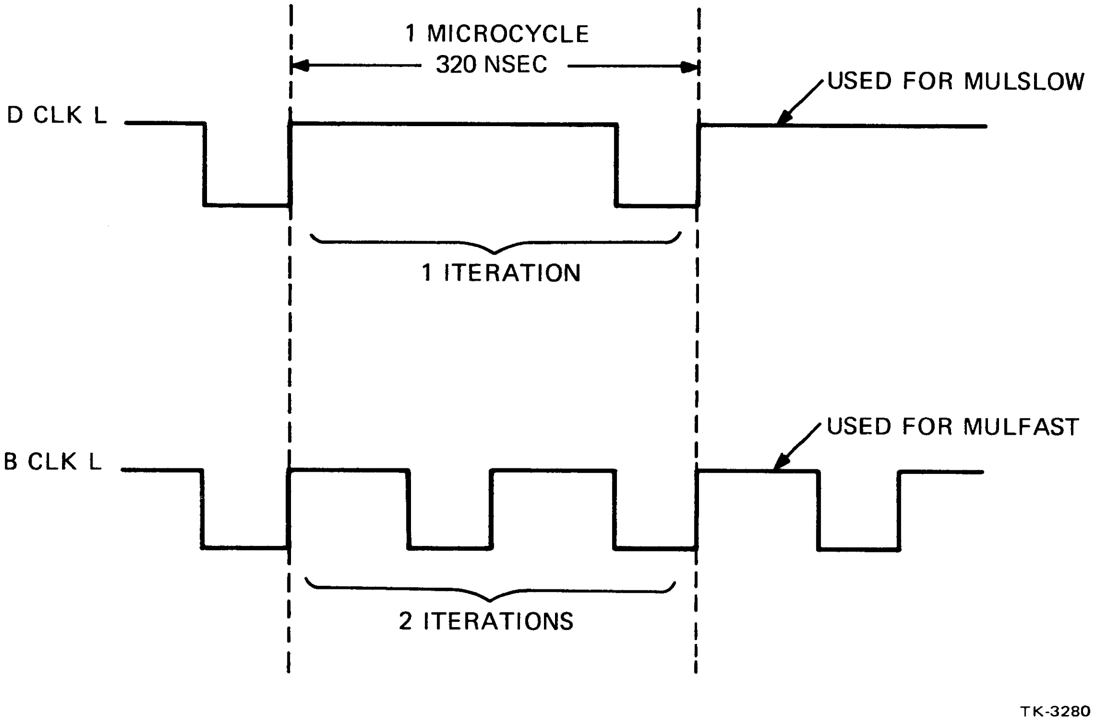


Figure 2-86 MULFAST vs. MULSLOW Timing

Set-Up Cycle – To set up initial conditions, a set-up cycle is always executed during the first microcycle in which the ALPCTL microfield specifies a multiply function. During the set-up cycle for a multiply function, the following events occur.

Loop flag ← 1
TOG flag ← Q <00>
Shift Q register right, shift input = 0
Clear D register
ALKC ← 0

The Loop flag is set to indicate that a multiplication loop has been entered and must not be interrupted. This flag remains set until the multiply function is complete. The TOG flag is loaded with the least significant bit of the multiplier from the Q register. This is done to reserve the bit for examination during the first iteration. With the least significant bit of the multiplier reserved, the Q register is shifted right in preparation for the examination of the next bit of the multiplier. The D register and ALKC flag are also cleared.

Multiply Flow – Figure 2-87 summarizes the events executed during each type of multiply function. As seen in this figure, the operations performed are basically identical for each type of function. The flow is entered when the ALPCTL microfield specifies a multiply function. With the Loop flag unasserted at this time, a set-up cycle is triggered. During this microcycle, the Loop flag is set to indicate that the multiply iterations can begin in the following microcycles.

When a sufficient number of iterations have been executed, the ALPCTL microfield is changed and the multiply loop is terminated. Note that the Loop flag is cleared only if the ALPTCTL microfield specifies a function other than a multiply, divide, or a WB ALUF function.

Although each type of multiply function is unique, similar events are executed for each type of iteration. During each iteration, the multiply algorithm must be performed. Figure 2-88 illustrates the events of the MULSLOW+ iteration in order to demonstrate the implementation of the basic algorithm structure. Comments in this figure relate to the description of the multiply algorithm in Paragraph 2.6.5.4.1.

As seen in Figure 2-88, the TOG flag is used to reserve the least significant bit of the multiplier. This is done so that the multiplier can be shifted during the same microcycle in preparation for the next iteration. The shift is performed to place the next multiplier bit into the least significant bit position of the Q register. By shifting the next multiplier bit into this position during each iteration, the TOG flag can always be loaded from bit 00 of the Q register. The Q register is shifted during the set-up cycle for the same reason.

The ALKC flag is loaded with the ALU <00> during each iteration of the multiply function. When the function has been completely executed, the ALKC flag contains the most significant bit of the low-order production from the Q register. The condition of the flag at this time is typically used for overflow detection.

Termination of a Multiply Loop – The ALPCTL microfield must specify the multiply function for the duration of the multiply loop. The step counter physically located in the PHB chip of the microsequencer can be used to determine when to terminate the loop. The step counter is initially loaded with the number of microcycles required for the operation. Using the MULSLOW function, a multiplication of N bits by N bits requires N + 1 microcycles (1 microcycle for the set-up cycle). The same multiplication operation requires N/2 + 1 microcycles using the MULFAST function. During each microcycle of the operation the counter is decremented. When the counter equals 0, the ALPCTL microfield is changed and the multiply loop is terminated.

2.6.5.4.3 Divide Algorithm – The special functions of the divide group are used to perform unsigned division of two integers. The divisor, however, is treated as positive or negative, depending on the type of divide function invoked (Table 2-52). The dividend can contain up to 64 bits; the divisor can contain up to 32 bits.

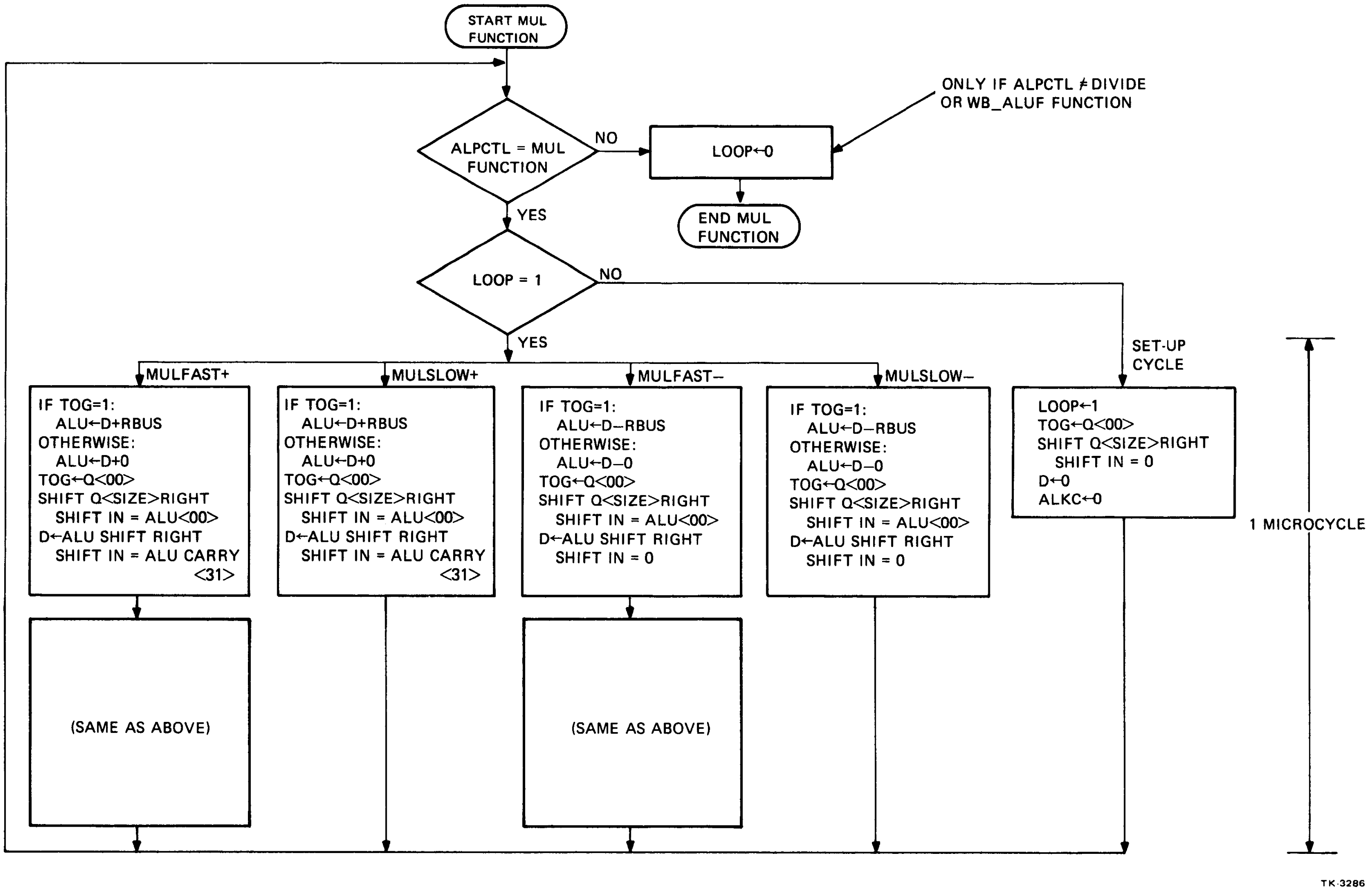


Figure 2-87 Multiply Flow

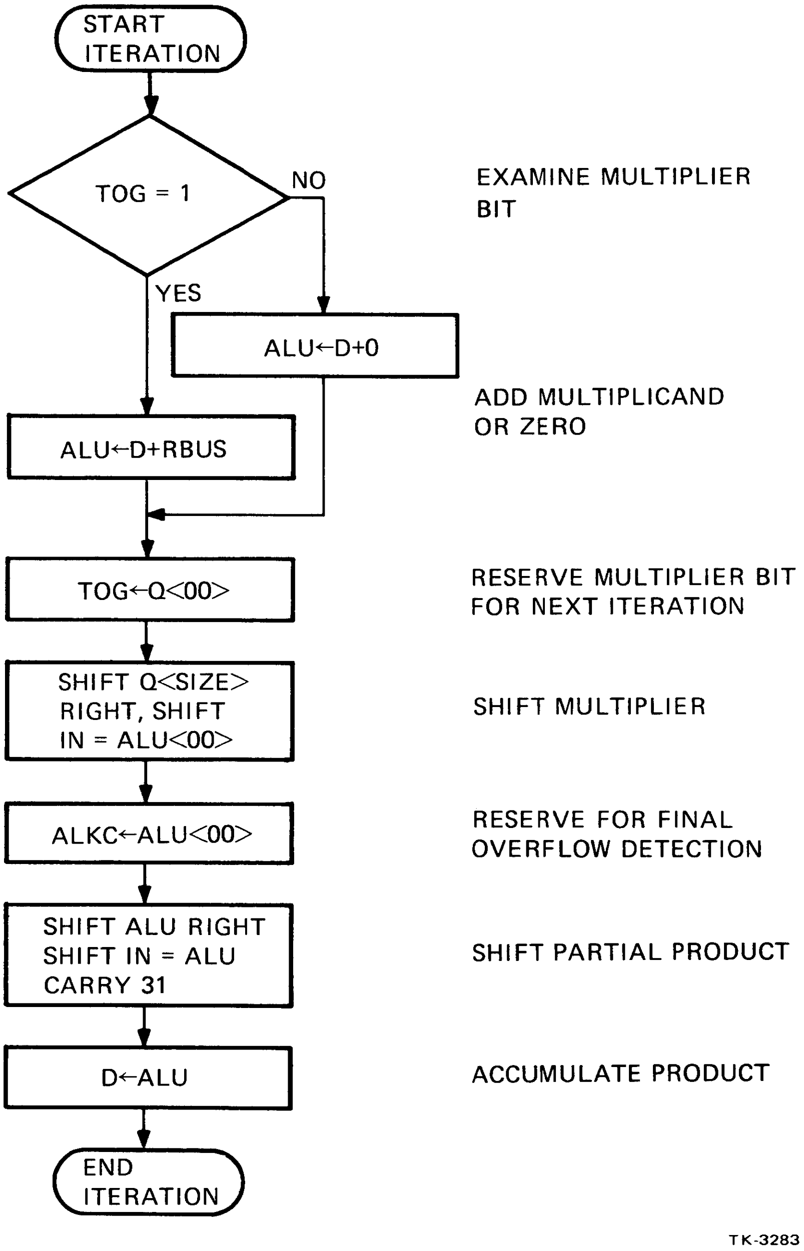


Figure 2-88 Multiply Iteration; Positive Multiplicand

The divide algorithm employed is nonrestoring. To understand nonrestoring division, consider the case of restoring division. For the first iteration the high-order bit of the dividend is compared to the divisor. When dealing with positive numbers in restoring division, this is done by subtracting the divisor from the high-order bit of the dividend. If the subtraction is successful (indicated by a positive remainder), a 1 is entered in the quotient ending the iteration. If the subtraction is unsuccessful, a 0 is entered in the quotient and the remainder is restored back to its original value. This is done by adding the divisor to the remainder. The disadvantage to this process is that two arithmetic operations (a subtraction and addition) are required during the same iteration when the comparison is unsuccessful.

The chief advantage of nonrestoring division over restoring division is that the remainder need not be restored with an extra operation if the subtraction result is unsuccessful. Figure 2-89 illustrates an example comparing the restoring and nonrestoring divide algorithms. Note that an arithmetic operation is eliminated during each iteration of the nonrestoring divide algorithm. (Divide iteration is defined as the operation repeated for the generation of each quotient bit.)

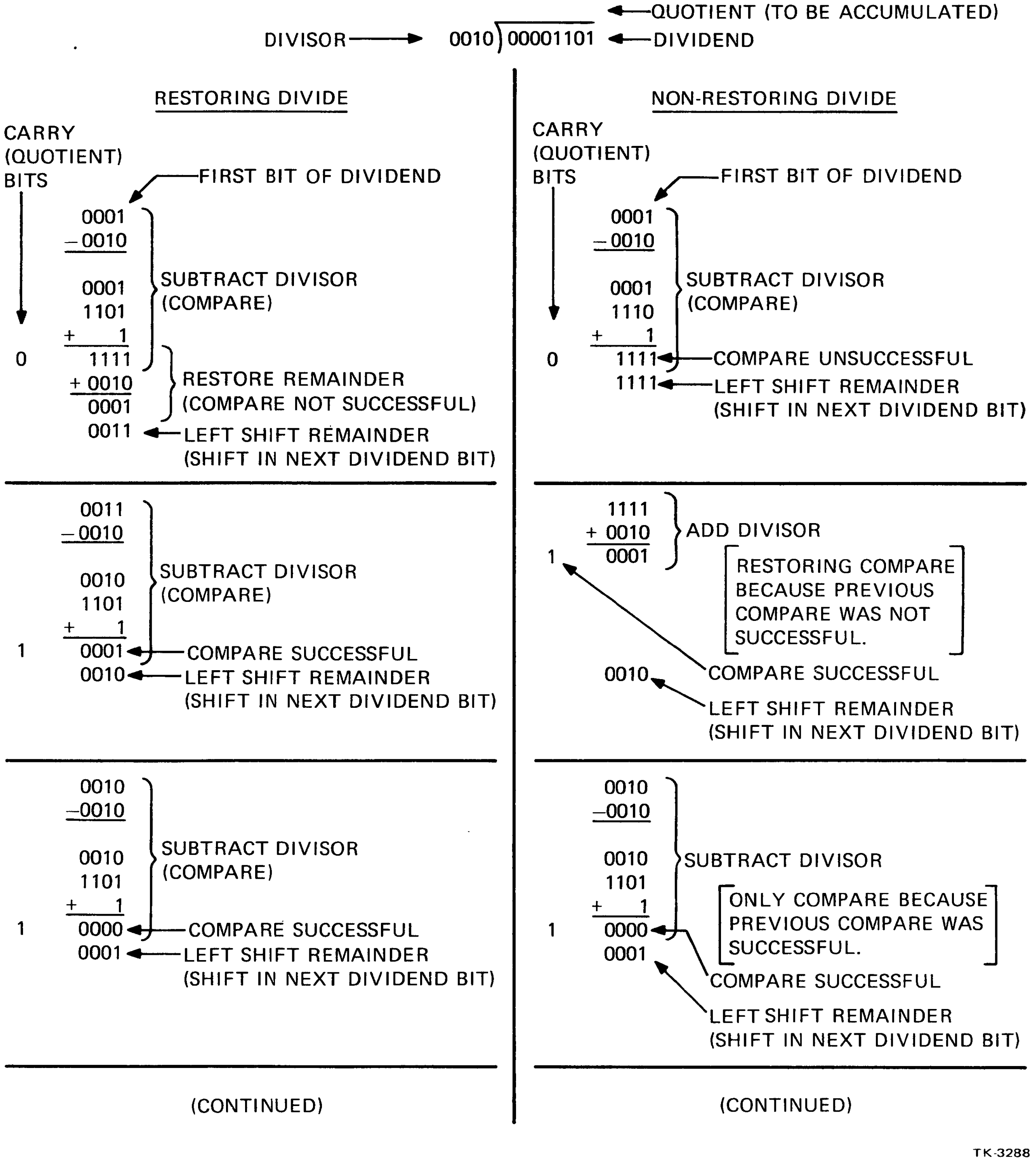
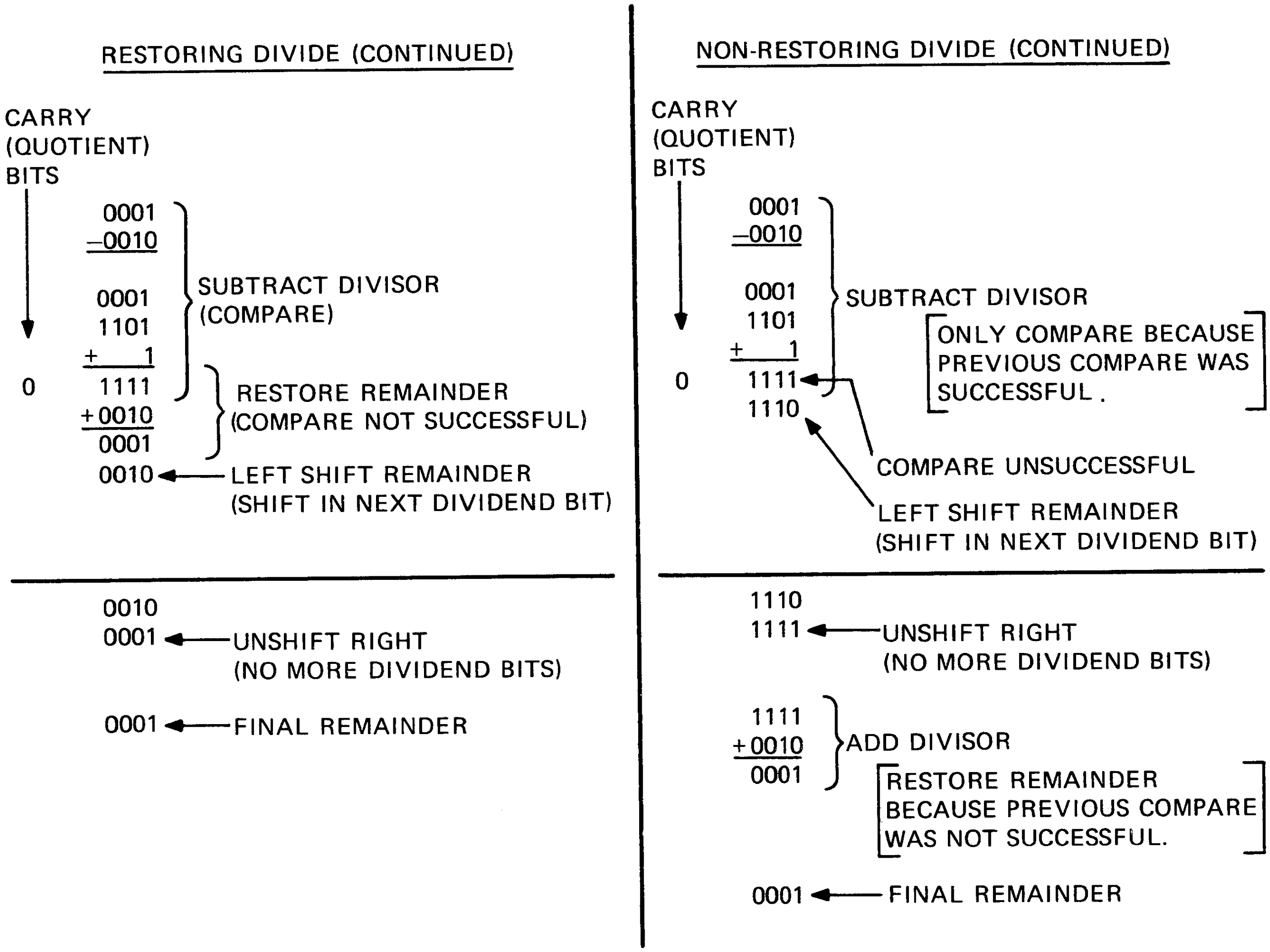

QUOTIENT (TO BE ACCUMULATED)
DIVISOR
0010)00001101
DIVIDEND
RESTORING DIVIDE
CARRY (QUOTIENT) BITS
FIRST BIT OF DIVIDEND
0001
−0010
0001
1101
+ 1
0 1111
SUBTRACT DIVISOR (COMPARE)
+ 0010
0001
RESTORE REMAINDER (COMPARE NOT SUCCESSFUL)
0011
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
0011
−0010
0010
1101
+ 1
1 0001
SUBTRACT DIVISOR (COMPARE)
COMPARE SUCCESSFUL
0010
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
0010
−0010
0010
1101
+ 1
1 0000
SUBTRACT DIVISOR (COMPARE)
COMPARE SUCCESSFUL
0001
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
(CONTINUED)
NON-RESTORING DIVIDE
CARRY (QUOTIENT) BITS
FIRST BIT OF DIVIDEND
0001
−0010
0001
1110
+ 1
0 1111
SUBTRACT DIVISOR (COMPARE)
COMPARE UNSUCCESSFUL
1111
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
1111
+ 0010
1 0001
ADD DIVISOR
RESTORING COMPARE BECAUSE PREVIOUS COMPARE WAS NOT SUCCESSFUL.
COMPARE SUCCESSFUL
0010
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
0010
−0010
0010
1101
+ 1
1 0000
SUBTRACT DIVISOR
ONLY COMPARE BECAUSE PREVIOUS COMPARE WAS SUCCESSFUL.
COMPARE SUCCESSFUL
0001
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
(CONTINUED)
TK-3288

Figure 2-89 Restoring vs. Nonrestoring Divide (Sheet 1 of 2)


RESTORING DIVIDE (CONTINUED)
CARRY (QUOTIENT) BITS
0001
−0010
0001
1101
+ 1
0
1111
+0010
0001
0010
SUBTRACT DIVISOR (COMPARE)
RESTORE REMAINDER (COMPARE NOT SUCCESSFUL)
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
0010
0001
UNSHIFT RIGHT (NO MORE DIVIDEND BITS)
0001
FINAL REMAINDER
NON-RESTORING DIVIDE (CONTINUED)
CARRY (QUOTIENT) BITS
0001
−0010
0001
1101
+ 1
0
1111
1110
SUBTRACT DIVISOR
ONLY COMPARE BECAUSE PREVIOUS COMPARE WAS SUCCESSFUL.
COMPARE UNSUCCESSFUL
LEFT SHIFT REMAINDER (SHIFT IN NEXT DIVIDEND BIT)
1110
1111
UNSHIFT RIGHT (NO MORE DIVIDEND BITS)
1111
+0010
0001
ADD DIVISOR
RESTORE REMAINDER BECAUSE PREVIOUS COMPARE WAS NOT SUCCESSFUL.
0001
FINAL REMAINDER

NOTES:
1. HORIZONTAL LINES REPRESENT THE END OF AN ITERATION.
2. ALL SUBTRACTION IS EXECUTED IN 2'S COMPLEMENT FORM.

TK-3289

Figure 2-89 Restoring vs. Nonrestoring Divide (Sheet 2 of 2)

2.6.5.4.4 Hardware Implementation of Divide – This paragraph describes the hardware implementation of the divide algorithm described in the previous paragraph. Each subsection of this paragraph describes an aspect of the divide operation.

For the execution of a divide function, the magnitude of the dividend is loaded into the D and Q registers and the divisor is placed on the RBus. The execution of the divide operation results in a quotient and final remainder. The magnitude of the quotient is accumulated in the Q register. The final remainder, however, must be derived from the final contents of the D register and ALUSO flag. This derivation is accomplished by the execution of the REM special function. (Details of the REM function are described below.)

The data on the RBus (divisor) can be treated as positive or negative, depending on the type of divide function selected. For a positive divisor, a DIVFAST+ or DIVSLOW+ function is selected. For a negative divisor, a DIVFAST− or DIVSLOW− function is selected.

DIVFAST vs. DIVSLOW – The distinction between DIVFAST and DIVSLOW is time-related, just as with MULFAST and MULSLOW. For an explanation of the timing involved, refer to Paragraph 2.6.5.4.2.

Set-Up Cycle – A set-up cycle is executed during the first microcycle of any divide function. This is done to set up initial conditions. During the set-up cycle for a divide function, the following events occur.

For a Positive Divisor

ALU ← D – RBus
TOG flag ← ALU CARRY 31
Loop flag ← 1
Shift ALU left, shift-in
= Q <MSB>
Shift Q register left,
    shift-in = ALU CARRY 31
ALUSO flag ← ALU <31>
D register ← ALU

For a Negative Divisor

ALU ← D + RBus
TOG flag ← ALU CARRY 31
Loop flag ← 1
Shift ALU left, shift-in
= Q <MSB>
Shift Q register left,
    shift-in = ALU CARRY 31
ALUSO flag ← ALU <31>
D register ← ALU

The nonrestoring divide algorithm illustrated in Figure 2-89 explains the events listed above. Because a positive divisor is used in the example of Figure 2-89, the divisor is subtracted from the dividend to determine whether the dividend is divisible. If the result is negative (indicating that the divisor is larger than the dividend), a 0 is entered into the quotient. If the result is positive, a 1 is entered into the quotient. The result is also reserved for examination during the first iteration by loading the ALU carry bit (sign bit) into the TOG flag. Note that the inverse is loaded in this case. (Refer to the section below on Divide Flow.

The Loop flag is set to indicate a division loop has been entered and must not be interrupted. This flag remains set until the divide function is complete.

The ALU is shifted left during the set-up cycle to shift in the next dividend bit in preparation for the iteration to follow. The Q register is likewise shifted to store the resultant quotient bit. The data associated with the partial remainder is accumulated in the D register and ALUSO flag.

Divide Flow – Figure 2-90 summarizes the events executed during each basic type of divide function. (The DIVDA and DIVDS are described in Paragraph 2.6.5.4.6.) As seen in this figure, the operations performed are similar for each type of function. The flow is entered when the ALPCTL microfield specifies a divide function, and terminated when the microfield is changed. This method of entrance and termination is identical to that described in Paragraph 2.6.5.4.2 for multiply. The Loop flag is likewise used in the same way.

Figure 2-91 illustrates a DIVSLOW+ iteration as an example of the implementation of the divide algorithm. Comments in this figure relate to the description of the divide algorithm in this paragraph. Note that the TOG flag is used just as it is used during the execution of a multiply function. During an iteration, it is first examined to determine the arithmetic operation to be performed. Once executed, the TOG flag is loaded with the results for examination during the following iteration.

Termination of a Divide Loop – The step counter, located in the PHB chip of the microsequencer (Paragraph 2.3) can be used to determine when to terminate a division loop. The step counter is initially loaded with the number of microcycles required for the operation. The value to be loaded is calculated by the same process described in Paragraph 2.6.5.4.2 for a multiply loop. Once loaded, the step counter is decremented for each microcycle of the operation. When the counter is decremented to zero, the ALPCTL microfield is changed and the loop is terminated.

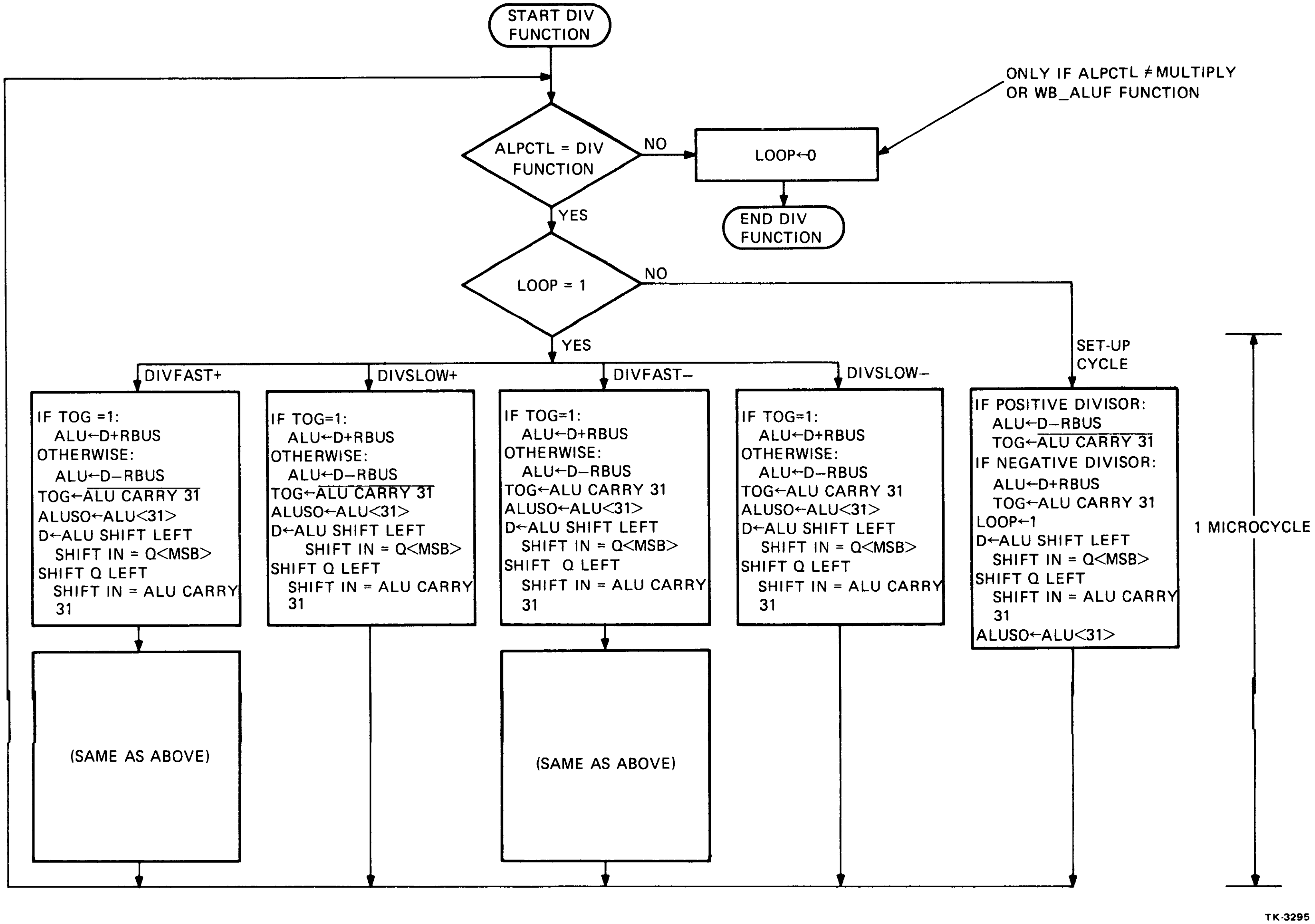
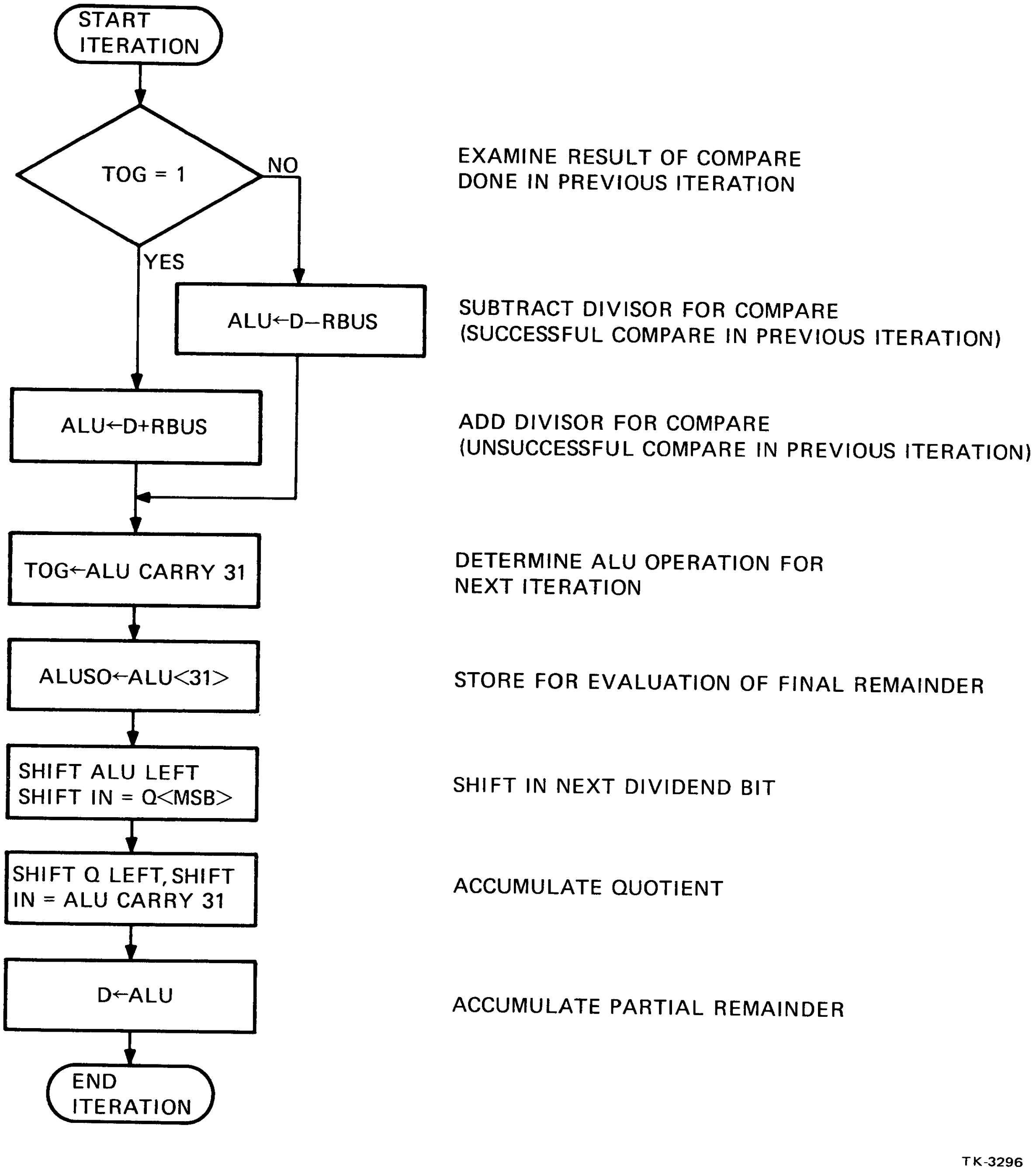


Figure 2-90 Divide Flow



EXAMINE RESULT OF COMPARE DONE IN PREVIOUS ITERATION

SUBTRACT DIVISOR FOR COMPARE (SUCCESSFUL COMPARE IN PREVIOUS ITERATION)

ADD DIVISOR FOR COMPARE (UNSUCCESSFUL COMPARE IN PREVIOUS ITERATION)

DETERMINE ALU OPERATION FOR NEXT ITERATION

STORE FOR EVALUATION OF FINAL REMAINDER

SHIFT IN NEXT DIVIDEND BIT

ACCUMULATE QUOTIENT

ACCUMULATE PARTIAL REMAINDER

Figure 2-91 Nonrestoring Divide Iteration; Positive Divisor

2.6.5.4.5 REM – According to the nonrestoring divide algorithm, the partial remainder is not restored between iterations. In addition, the algorithm results in a surplus shift in the final remainder. Because of this, the final remainder is not readily available at the end of a divide operation. Additional operations must be performed. These operations basically unshift the remainder in the D register, and restore the divisor if the compare in the final iteration was unsuccessful. (Refer to the last part of Figure 2-89.)

The REM special function is used to unshift the remainder in the D register. The events performed during the REM special function are listed below.

ALU ← D register – RBus
Shift ALU right, shift input = ALUSO flag
D register ← ALU
ALUSO flag ← ALU <00>

For proper execution of the REM function, the RBus must be cleared (set to 0). Basically, during the execution of the REM function, an ALU operation is performed to transfer the contents of the D register to the ALU. The ALU is then shifted and loaded back into the D register.

Figure 2-92 illustrates an example of a complete divide flow for a positive divisor and negative divisor. The flows are basically identical. The sign of the divisor must only be considered during the divide itself and during remainder restore. With this flow, the magnitude of the quotient is accumulated in the Q register and the magnitude of the remainder is accumulated in the D register.

As seen in Figure 2-92, the remainder is only restored if the compare during the last iteration is unsuccessful. An unsuccessful compare is indicated by the unasserted condition of ALU CARRY 31. For this reason, ALU CARRY 31 is stored in ALUS <1>. ALUS <1> is the ALU state latch described in Paragraph 2.6.5.3. For a positive divisor, the remainder is restored by adding the divisor. For a negative divisor, the remainder is restored by subtracting the divisor.

2.6.5.4.6 DIVDA and DIVDS – The DIVDA and DIVDS special functions are used during double precision divide operations. The events performed during each function are listed below.

DIVDA	**DIVDS**
ALU ← D + RBus + ALKC flag	ALU ← D – RBus – ALKC flag
Shift ALU left shift input = ALKC flag	Shift ALU left shift input = ALKC flag
D register ← ALU	D register ← ALU
Shift Q register left shift input = ALU CARRY 31	Shift Q register left shift input = ALU CARRY 31

Note that the DIVDS function is similar to the DIVDA function except that a subtract operation is performed instead of an addition. The ALKC flag and ALUSO flag remain intact for both functions. In addition to the events listed above, the data loaded into the D register is also channeled onto the WBus during each function.

Figure 2-93 illustrates a sample flow of double precision divide using the DIVDA and DIVDS special functions. As mentioned in this figure, the high-order magnitude of the dividend is loaded into the D register, with the low-order magnitude in MTEMP register 2 of the scratchpad. The high- and low-order magnitudes of the divisor are likewise loaded into RTEMP registers 1 and 0 of the scratchpad, respectively. With the divisor and dividend loaded, the flow is executed and the high-order 32-bit quotient is accumulated in the Q register. This result is then saved before the step counter is reset to 32_{10} and the flow is reexecuted to compute the low-order 32-bit quotient. The low-order 32-bit quotient is likewise accumulated in the Q register.

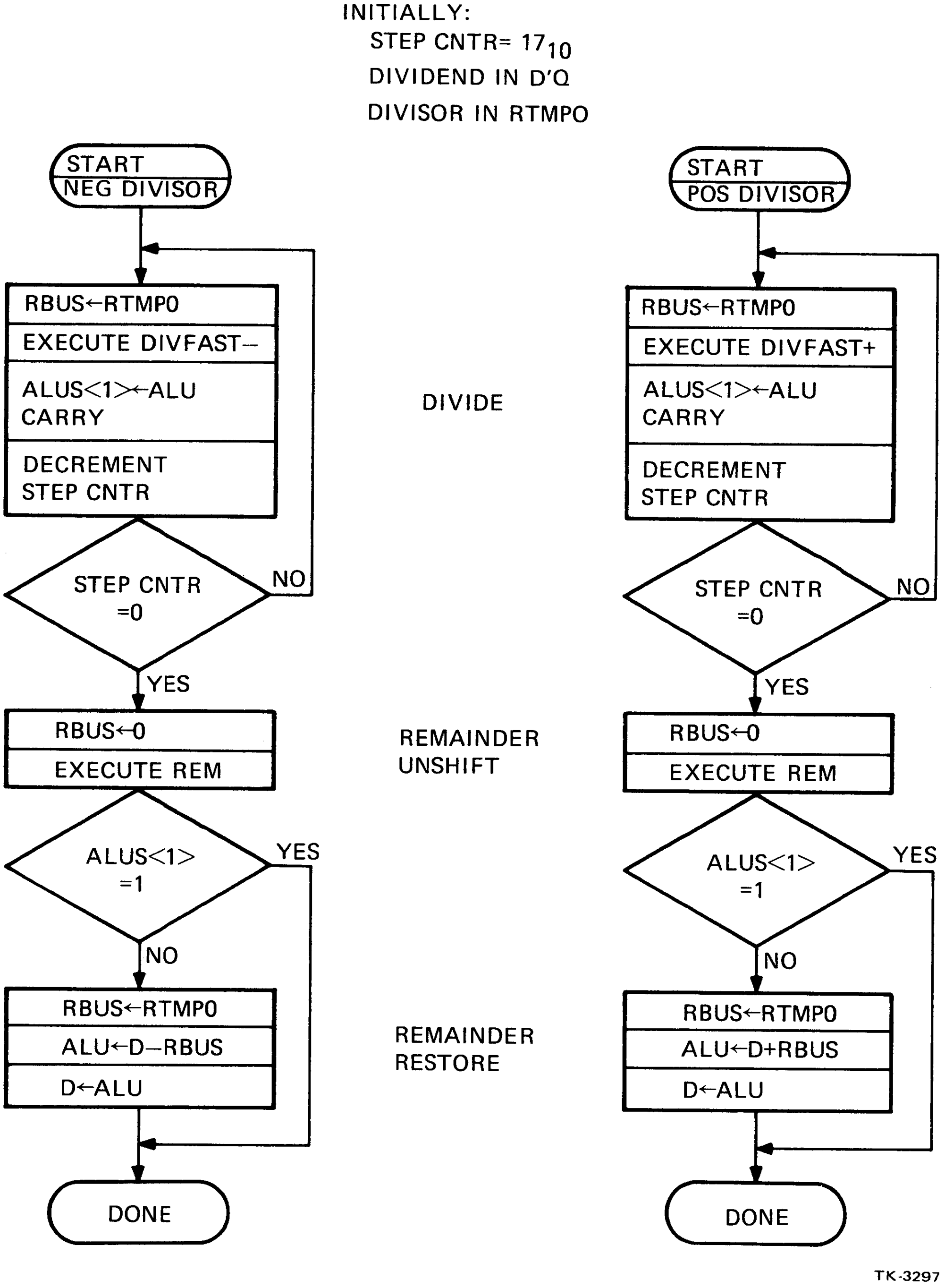


Figure 2-92 Example Flow of 62 × 32 Bit Divide

For the flow illustrated in Figure 2-93, half an iteration is executed each microcycle. Also, note that the reference to ALUS <1> refers to the ALU state latch in the condition code chip (CCC), not the ALUSO flag.

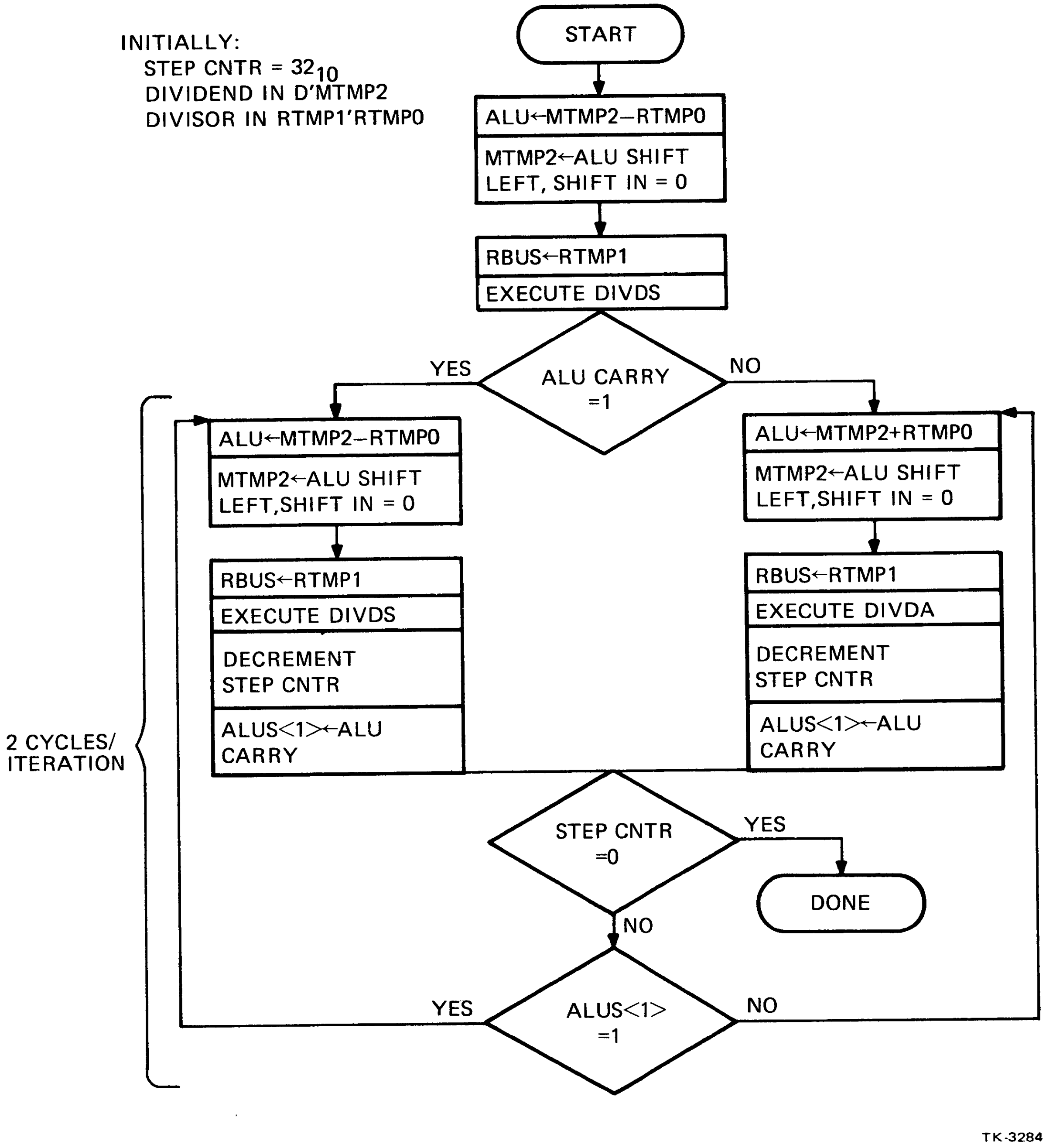


Figure 2-93 Double Precision Divide Example Using DIVDA and DIVDS

2.6.6 Rotator Section

The rotator section provides the data path with the capability of various bit shifting and shuffling operations. The circuitry consists of a rotator and rotator control logic. The rotator is implemented with four SRM chips and a section from each of the eight ALP chips in the arithmetic section. The SRK chip contains the rotator control logic.

2.6.6.1 Interpretation of the ROT Microfield – All rotator operations are specified by the ROT microfield, bits <63:58> of the microword. These bits may also be encoded for the following purposes.

1. To generate SRK status signals for microbranching.

2. To specify a carry-in for the ALU, a shift-in for the ALU and Q register, and selection of extended MBus data.

The interpretation of these bits depends on the content of the current microinstruction. Figure 2-94 illustrates this concept. As seen in this figure, these bits can define the ROT microfield, the ROTSRK subfield, or the ALUXM, ALUSHF, and ALUCI subfields of the microword depending on their intended purpose in the microword. In addition, these bits sometimes have two interpretations. The interpretation specifically depends on the following conditions.

Condition	**Indication**
Rotator output is used	MUX subfield specifies M.S, XM, D.S, Z.S or R.S
P latch or S latch is loaded	Bits <63:58> of the microword equal 2D, 2F, 3B, 3D, or 3F
Status signals are used for	BUT microfield specifies SRKSTA microbranching

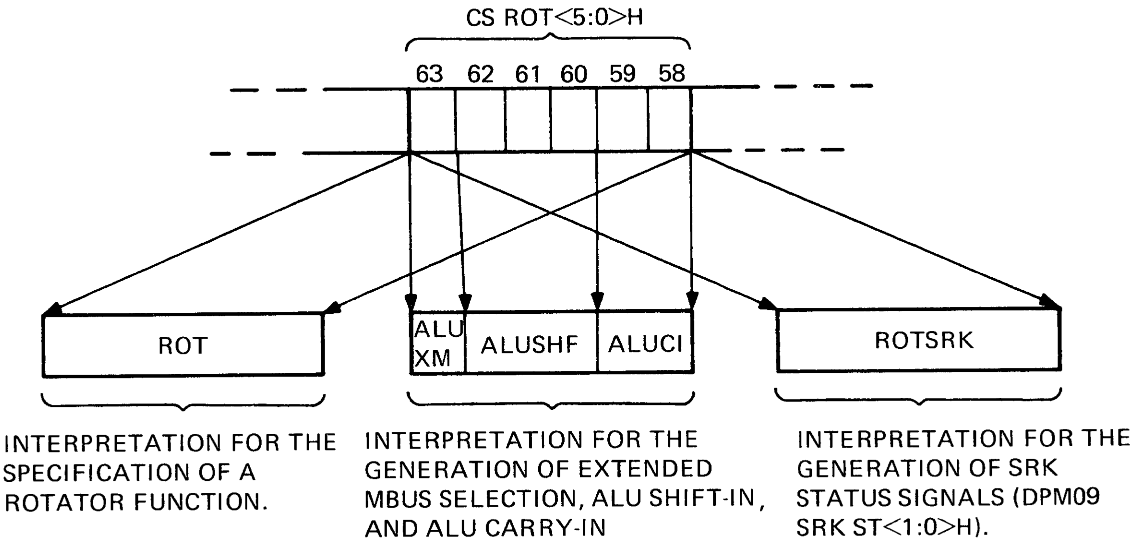
If the rotator output is to be used during the microcycle, or the S latch or P latch is modified, bits <63:58> of the microword, define the ROT microfield (and ALUXM subfield) only. For this case the value on the ROT lines specifies a rotate function (Paragraph 2.6.6.3). The ALU shift-in and ALU carry-in for this case are defaulted to a hard-wired zero (the ALUSHF <2:0> and ALUCI <1:0> interpretations are not used). Note that ROT <5>, however, defines the ALUXM subfield for selection of extended MBus data, as it always does (i.e., ALUXM <0> has no hard-wired default value). Refer to Paragraph 2.6.5.1.5 for descriptions of the ALUXM, ALUSHF, and ALUCI subfields.

If the SRK status signals are to be used during the microcycle (i.e., the BUT microfield specifies SRKSTA), bits <63:58> of the microword also define the ROTSRK subfield. For this case, the value on the ROT lines specifies a microtest (Paragraph 2.6.6.4.2).

Note that the SRK status signals are generated during every microcycle even though they may not be used for a microbranch. Likewise, the rotator performs the rotator function specified by ROT <5:0> even though the resultant output may not be used.

2.6.6.2 The Rotator (SRM and S Shifter) – Figure 2-95 illustrates the basic architecture of the rotator. The shifting is accomplished in two levels. The first level shift is executed by logic contained in the SRM chips. This level shifts the 64-bit input 0, 4, 8, 12, 16, 20, 24, or 28 positions to the right. A 35-bit result is then output on the SBus. The second level shifting is executed by logic in the ALP chips of the arithmetic section. For this shift, data from the SBus is shifted by 0, 1, 2, or 3 positions to the right by the S shifter. Note that the S shifter is considered part of the rotator even though it resides in the ALP chips of the arithmetic section.

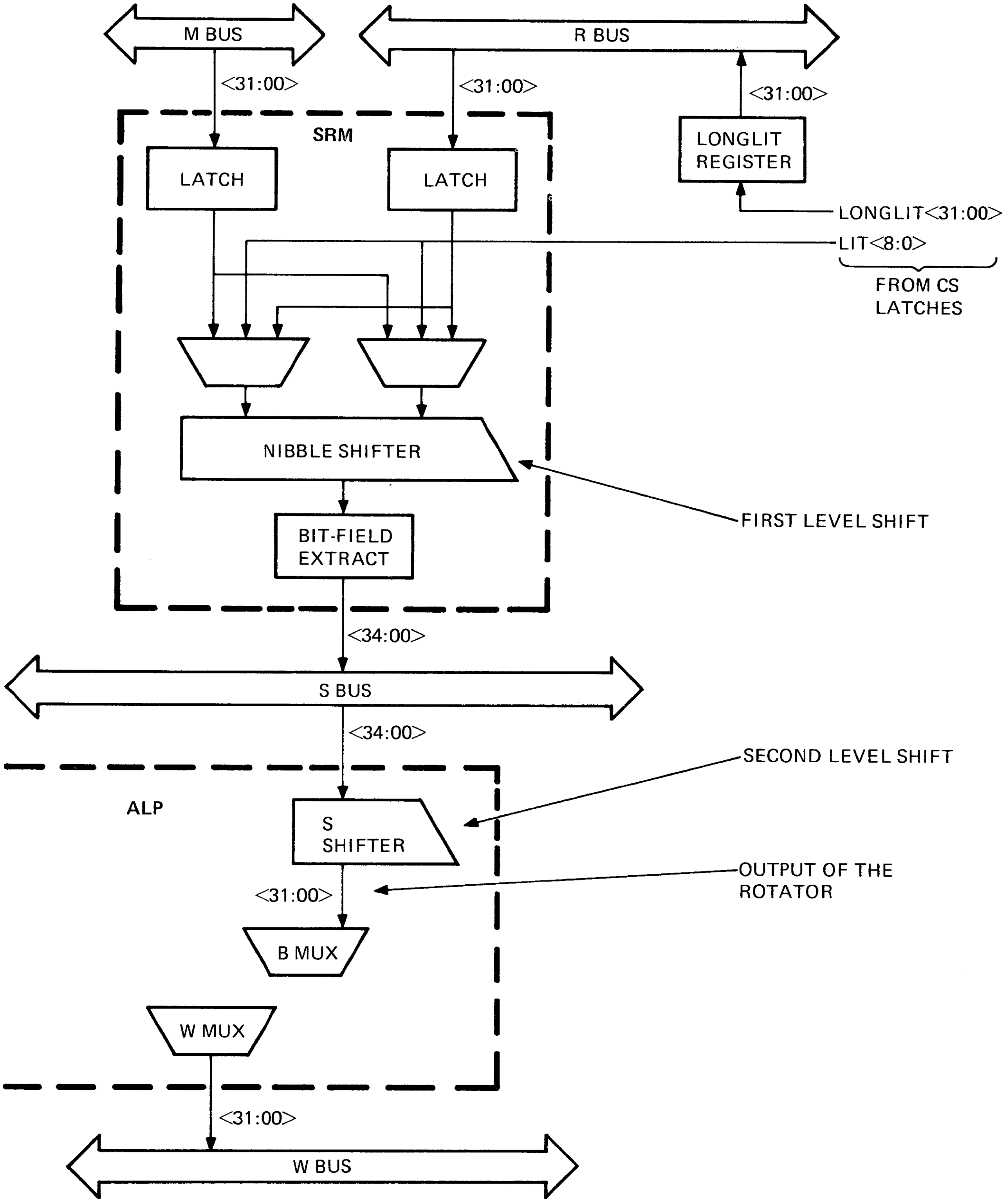
The reader should observe the distinction between the SBus and rotator output. Note, however, that if no S-shifter operation is specified by DPM09 SHF <1:0> L, the output of the rotator and SBus are equal.


CS ROT<5:0>H
63
62
61
60
59
58
ROT
ALU XM
ALUSHF
ALUCI
ROTSRK
INTERPRETATION FOR THE SPECIFICATION OF A ROTATOR FUNCTION.
INTERPRETATION FOR THE GENERATION OF EXTENDED MBUS SELECTION, ALU SHIFT-IN, AND ALU CARRY-IN
INTERPRETATION FOR THE GENERATION OF SRK STATUS SIGNALS (DPM09 SRK ST<1:0>H).

ROTATOR OUTPUT USED OR S LATCH OR P LATCH LOADED	STATUS SIGNALS USED	INTERPRETATION OF BITS<63:58> OF THE MICROWORD
NO	NO	ROT, ALUXM, ALUSHF, ALUCI
YES	NO	ROT, ALUXM
NO	YES	ROTSRK, ALUXM, ALUSHF, ALUCI
YES	YES	ROT, ROTSRK, ALUXM

TK-3324

Figure 2-94 Interpretation of the ROT Microfield



TK-3337

Figure 2-95 Rotator

With the proper combination of shifting at each of the two levels, the rotator can shift input data 0–31 positions to the right or 1–31 positions to the left. The SRM chips of the rotator are also capable of masking bit positions to extract a bit-field (any length with zero extension). In addition to the shifting capabilities, groups of bits can be shuffled to execute BCD swapping, conversion of BCD to ASCII, etc. Paragraph 2.6.6.3 describes each of these functions.

2.6.6.3 Rotator Functions – The rotator function is determined by the value of the ROT microfield. These bits, ROT <5:0>, are decoded by the SRK chip to generate control signals for the SRM chips. In addition to specifying control of the rotator, the ROT microfield specifies the internal operation of the SRK itself. These internal operations include loading latches, interpreting condition signals (data size and zero indicators), and generating status signals. The SRK chip is discussed in Paragraph 2.6.6.4.

Table 2-53 shows the selected rotator function for each value of the ROT microfield. (Note that these bits also define the ALUCI, ALUSHF, ALUXM, and ROTSRK subfields of the microword. Refer to Paragraph 2.2.1.2 for a description of subfields.) A brief description of each rotator function is also provided in Table 2-53. Details of various functions are discussed below.

Table 2-53 Rotator Functions

ROT <5:0> (Hex)	Function Mnemonic	Description (See Note for Notation)
00	XZ.MR	EXTZ M'R, POS = PL, Size = SL
01	XZ.MM	EXTZ M'M, POS = PL, Size = SL
02	XZ.RR	EXTZ R'R, POS = PL, Size = SL
03	ASR.M.P	Arithmetic Shift M Right, No. Bits = PL
04	RR.MR.P	Rotate M'R Right, No. Bits = PL
05	RR.MM.P	Rotate M'M Right, No. Bits = PL
06	RR.RR.P	Rotate R'R Right, No. Bits = PL
07	RR.MR.S	Rotate M'R Right, No. Bits = SL
08	RL.RM.4	Rotate R'M Left, No. Bits = 4
09	RR.MR.4	Rotate M'R Rotate, No. Bits = 4
0A	RR.RR.SIZ	Rotate R'R Right by 1, 2, 3, 0 Bytes
0B	RR.MR.9	Rotate M'R Right, No. Bits = 9
0C	XZ.PTX	EXTZ M'M, POS = 07, Size = 23
0D	XZ.VPN	EXTZ M'M, POS = 09, Size = 21
0E	RR.MM.SIZ	Rotate M'M Right by 1, 2, 3, 0 Bytes
0F	GETNIB	Get 0'MBUS <3:0>
10	GETEXP	EXTZ M'M POS = 7, Size = 8
11	RL.MM.PTE	Rotate M'M Left, No. Bits = 9
12	CLR2BM	CLR M <15:00>
13	CLR1BM	CLR M <07:00>
14	CLR3BM	CLR M <23:00>

NOTE: WB = WBUS low byte; M = MBUS; R = RBUS
EXTZ = Extract/zero-extend functions
P = P latch; S = S latch, both on SRK CHIP.
POS = starting bit position of a bit field to be extracted.
Size = size of bit field.

Table 2-53 Rotator Functions (Cont)

ROT <5:0> (Hex)	Function Mnemonic	Description (See Note for Notation)
15	ASL.R.7	Arithmetic Shift R Left By 7 Bits
16	ZERO	Constant 0
17	ASL.R.SIZ	Arithmetic Shift R Left By 0, 1, 2, 3 bits
18	BCDSWP	BCD Swap, M
19	GETFPF	Unpack FP Fraction, M'R
1A	FPACK	Pack FP DATA, M = FRAC R = EXP
1B	CVTPN	Convert Packed to Numeric, M
1C	CONX.SIZ	Constant 1, 2, 4, 8 on Size
1D	ASR.M.3	Arithmetic Shift M Right, No. Bits = 3
1E	FPLIT	Expand Floating-Point LIT, M
1F	CVTNP	Convert Numeric to Packed, M'R
20	RL.RM.PS	Rotate R'M Left, No. Bits = P+SL
21	RL.MM.P	Rotate M'M Left, No. Bits = PL
22	RL.RR.P	Rotate R'R Left, No. Bits = PL
23	RL.RM.P	Rotate R'M Left, No. Bits = PL
24	RR.MR.PS	Rotate M'R Right, No. Bits = PL+SL
25	RR.MM.PS	Rotate M'M Right, No. Bits = PL+SL
26	RR.RR.PS	Rotate R'R Right, No. Bits = PL+SL
27	PL_MSS	Find Most Significant Bit, Set MBUS
28	ASL.R.P	Arithmetic Shift R Left, No. Bits = PL
29	ASL.M.P	Arithmetic Shift M Left, No. Bits = PL
2A	ASR.M.-P	Arithmetic Shift M Right, No. Bits = −PL
2B	ZLITPL	0 EXT LIT and Rotate Left PL Bits
2C	PL	SBUS ← PL
2D	PL.SL_WB	SL ← WBUS <5:0>, SBUS ← PL
2E	SL	SBUS ← SL
2F	SL.PL_WB	PL ← WBUS <5:0>, SBUS ← SL
30	ZLIT0	0 EXT LIT and Rotate Left 00 Bits
31	ZLIT28	0 EXT LIT and Rotate Left 28 Bits
32	ZLIT24	0 EXT LIT and Rotate Left 24 Bits
33	ZLIT20	0 EXT LIT and Rotate Left 20 Bits
34	ZLIT16	0 EXT LIT and Rotate Left 16 Bits
35	ZLIT12	0 EXT LIT and Rotate Left 12 Bits
36	ZLIT8	0 EXT LIT and Rotate Left 08 Bits
37	ZLIT4	0 EXT LIT and Rotate Left 04 Bits
38	OLIT0	1 EXT LIT and Rotate Left 00 Bits
39	MINUS1	Constant of All 1's

NOTE: WB = WBUS low byte; M = MBUS; R = RBUS
EXTZ = Extract/zero-extend functions
P = P latch; S = S latch, both on SRK CHIP.
POS = starting bit position of a bit field to be extracted.
Size = size of bit field.

Table 2-53 Rotator Functions (Cont)

ROT <5:0> (Hex)	Function Mnemonic	Description (See Note for Notation)
3A	OLIT24	1 EXT LIT and Rotate Left 24 Bits
3B	OLIT0.PL_LIT	PL ← LIT
3C	OLIT16	1 EXT LIT and Rotate Left 16 Bits
3D	OLIT0.SL_LIT	SL ← LIT
3E	OLIT8	1 EXT LIT and Rotate Left 08 Bits
3F	OLIT0.PL43_WB	PL <4:3> ← WBUS <1:0>

NOTE: WB = WBUS low byte; M = MBUS; R = RBUS
EXTZ = Extract/zero-extend functions
P = P latch; S = S latch, both on SRK CHIP.
POS = starting bit position of a bit field to be extracted.
Size = size of bit field.

References to PL and SL in Table 2-53 denote the P latch and S latch of the SRK chip. These latches are described in Paragraph 2.6.6.4. The term POS denotes the starting bit position of a bit field to be extracted. SIZE denotes the size of the bit field.

Figure 2-96 illustrates the EXTZ M,R function. For this type of function, data from the MBus and RBus are concatenated to form a 64-bit data structure with the MBus data in the most significant bit positions. A bit field is then extracted from this data structure and zero-extended onto the SBus.

The bit field to be extracted is implicitly specified by the ROT microfield. The SRK decodes this microfield to generate control signals for the SRM. These signals determine the first and last bits of the bit field to be extracted. Refer to Paragraph 2.6.6.4.1 for a description of these control signals.

The EXTZ M,M and EXTZ R,R functions are the same as the EXTZ M,R function except that:

1. For the EXTZ M,M function, data from the MBus is concatenated with itself to form the 64-bit data structure.

2. For the EXTZ R,R function, data from the RBus is concatenated with itself to form the 64-bit data structure.

Note that for all three types of EXTZ functions, the bit field to be extracted is defined by control signals from the SRK chip.

Arithmetic Shift Functions – The ASR and ASL are examples of the arithmetic shift functions. For these functions, data from the RBus or MBus is shifted and output onto the SBus. The selection of data and direction of shift is explicitly specified by the ROT microfield. The SRK decodes this microfield to generate the appropriate control signals for the rotator. The shift count is specified for two shift functions: ASL.R.7 and ASR.M.3. The shift count may also be indirectly specified through P latch or D-size signals from the microsequencer: ASR.M.P, ASL.R.SIZ, ASL.R.P, ASL.M.P, ASR.M.–P.

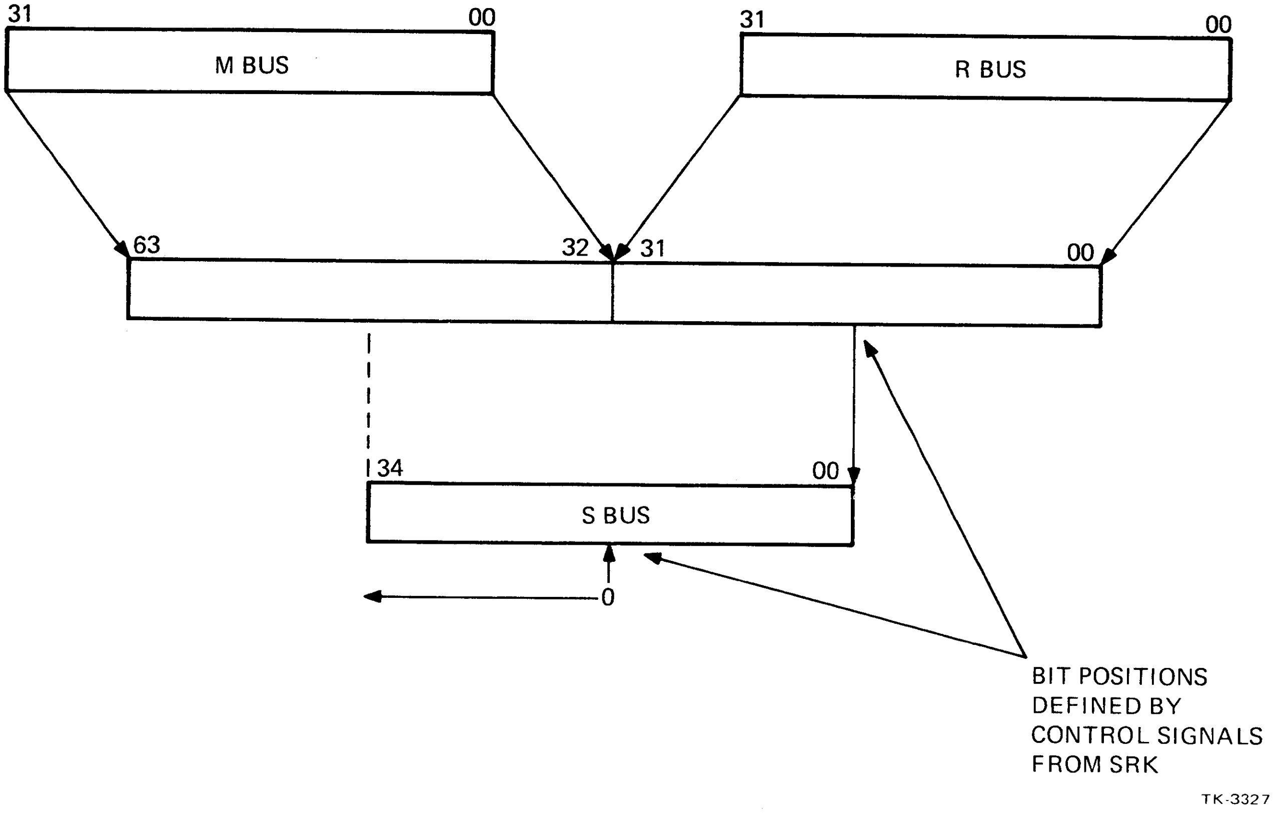


Figure 2-96 EXTZ M,R Function

Table 2-54 lists each of the arithmetic shift functions and describes their general use.

Table 2-54 Use of Arithmetic Shift Functions

ROT <5:0> (Hex)	Function Mnemonic	General Use
03	ASR.M.P	Used to align a floating-point fraction when the exponent difference is positive and stored in the P latch.
15	ASL.R.7	Used to unpack the low-order fraction of a double precision floating-point datum.
17	ASL.R.SIZ	Used in the index mode operand specifier routine.
1D	ASR.M.3	Used to convert a bit position to a byte position in the field instructions.
28	ASL.R.P	Used for the ASHL instructions.
29	ASL.M.P	Used for the ASHL instructions.
2A	ASR.M.-P	Used to align a floating-point fraction when the exponent difference is negative and stored in the P latch. Also used for ASH type instructions.

Rotate Functions – As seen in Table 2-53, there are 17 rotate functions selectable by the ROT microfield. These functions are denoted by mnemonics that begin with R. Each of these functions explicitly specifies the direction of rotation, the data to be rotated (MBus, RBus, or both), and the number of bits to be rotated. In some cases, the number of bits is indirectly specified by a reference to the P latch or S latch of the SRK, or the D-size signals from the microsequencer.

Get Functions – Three types of get functions can be selected by the ROT microfield. Figure 2-97 illustrates each of these functions. The GETNIB function extracts and zero-extends the low-order nibble (4 bits) of the MBus. This function is provided for general functionality. The GETEXP function extracts and zero-extends bits <14:07> of the MBus. This function is used to extract the exponent field from a floating-point datum. The GETFPF function extracts and merges fields from the MBus and RBus. This is done to unpack the fraction field of a floating-point datum.

FP Functions – The FPACK function is used to assemble a floating-point data format. During this function, data from the RBus and MBus are merged on the SBus as shown in Figure 2-98. As seen in this figure, the fraction bits must be placed on the upper 23 bits of the MBus. Likewise, the exponent bits must be placed on the lower eight bits of the RBus.

The FPLIT function is used to expand a floating-point literal. For this function, the literal must be placed on the MBus as shown in Figure 2-99. The FPLIT function places the literal in the correct format.

Clear Functions – Three types of clear functions can be selected by the ROT microfield. Each of these functions clears one or more lower bytes of MBus data and outputs the result onto the SBus. The three clear functions are CLR1BM, CLR2BM, and CLR3BM.

Constant Functions – Three types of constant functions can be selected by the ROT microfield. Each of these functions generates a constant for input to the B multiplexer. The zero function outputs a constant of all zeros. The MINUS1 function outputs a constant of all ones. A constant specified by the D-size signals from the microsequencer is output onto the SBus when the CONX.SIZ function is selected. For the CONX.SIZ function, the constant is specified as follows.

D-Size <1:0>	**Constant**
00	1
01	2
10	4
11	8

The CONX.SIZ function is used in the autoincrement and autodecrement modes of the operand specifier routines.

Convert Functions – Three types of convert functions can be selected by the ROT microfield. The purpose of each function is listed below.

BCDSWP – This function is used to arrange the bytes of a BCD string into correct arithmetic order.

CVTPN – This function is used to convert four BCD digits to four numeric digits.

CVTNP – This function is used to convert eight numeric digits to eight BCD digits.

The BCDSWP function reorganizes bytes of MBus data to convert a BCD string in memory format to the correct arithmetic order.

BCDSWP is primarily provided for the CVTPL instruction in which each BCD digit is serially examined. For other types of BCD instruction, BCD add or BCD subtract is used to perform decimal arithmetic. In these cases, no prior shifting is required.

As an example of BCDSWP, consider the memory storage of the number +12345678. Figure 2-100 illustrates the consecutive memory byte locations.

If the longword containing this decimal number was accessed from memory, it would be placed on the MBus in the format shown in Figure 2-101. The BCDSWP function places the data onto the SBus in the correct order as shown in Figure 2-101.

The CVTPN function converts a packed decimal (BCD) format to a numeric string. For this function, a constant must be placed on the RBus as shown in Figure 2-102. Note that only four BCD digits can be converted to numeric during each CVTPN function.

The CVTNP function is the complement of the CVTPN function described above. In this case, however, eight numeric digits (instead of four) can be converted to eight BCD digits during each operation. As an example, consider the decimal number 12345678. The eight numeric digits are loaded onto the MBus and RBus as shown in Figure 2-103. The data formats shown in this figure are easily accomplished because of the way a numeric string is stored in memory (Figure 2-104).

With the numeric digits properly placed on the MBus and RBus, the CVTNP can be performed to properly align the BCD digits.

Latch Functions – Eight types of latch functions can be selected by the ROT microfield. These functions control the loading and reading of the S and P latches in the SRK chip (Paragraph 2.6.6.4.1). The functions used strictly for loading the latches are listed below:

ROT <5:0>	**Function Mnemonic**
27	PL_MSS
3B	OLIT0.PL_LIT
3D	OLIT0.SL_LIT
3F	OLIT0.PL43_WB

For the second and third functions, the P latch or S latch is loaded with LITRL <5:0> of the literal subfield of the microword. The entire 9-bit contents of this subfield is also one-extended and output onto the SBus during both functions.

The PL_MSS function locates the MSB (most significant bit that is equal to 1) on the MBus and loads the number of the bit position into the P latch. To accomplish this, the SRK examines the WMUXZ signals (zero byte indicators) from the ALP to determine the left-most non-zero byte on the MBus. The byte is then rotated onto SBUS <7:0> by the SRM chips. The P latch is finally loaded with a value decoded from the WMUXZ signals and SBUS <7:0> as follows.

WMUXZ <3:0>	**P Latch <4:3>**
0XXX	11
10XX	10
110X	01
111X	00

SBUS <7:0>	**P Latch <2:0>**
1XXXXXXX	111
01XXXXXX	110
001XXXXX	101
0001XXXX	100
00001XXX	011
000001XX	010
0000001X	001
0000000X	000

Paragraph 2.6.6.4.1 describes the control signals generated by the SRK for the SRM. Note that for the proper execution of this function, the ALP must be selected to output the MBus data.

The PL_MSS function can be used for software interrupt arbitration, floating-point normalization, and certain macroinstructions such as CALL, PUSHR, CVTLP, FFS, and FFC.

The OLIT0.PL43_WB function merely loads bits <4:3> of the P latch with bits <1:0> of the WBus. Bits <5,2:0> of the P latch remain unchanged. This function is used for address calculations in the field instructions.

The PL and SL functions are used strictly for reading the P and S latches, respectively. The contents of the latches are zero-extended and output onto the SBus.

The remaining two latch functions are associated with reading one latch while loading the other. The PL.SL_WB function loads the S latch with WBUS <5:0> and outputs the contents of the P latch onto the SBus (zero-extended). The SL.PL_WB function similarly loads the P latch with WBUS <5:0> and outputs the contents of the S latch (zero-extended).

Literal Functions – The literal functions are associated with manipulation of the 9-bit literal subfield of the microword. For each of these functions the literal subfield is zero- or one-extended, rotated left, and output onto the SBus. The number of bits to be rotated is explicitly specified for almost all of these functions. The only exception is the ZLITPL function. For this function the number of bits is specified by the P latch in the SRK chip.

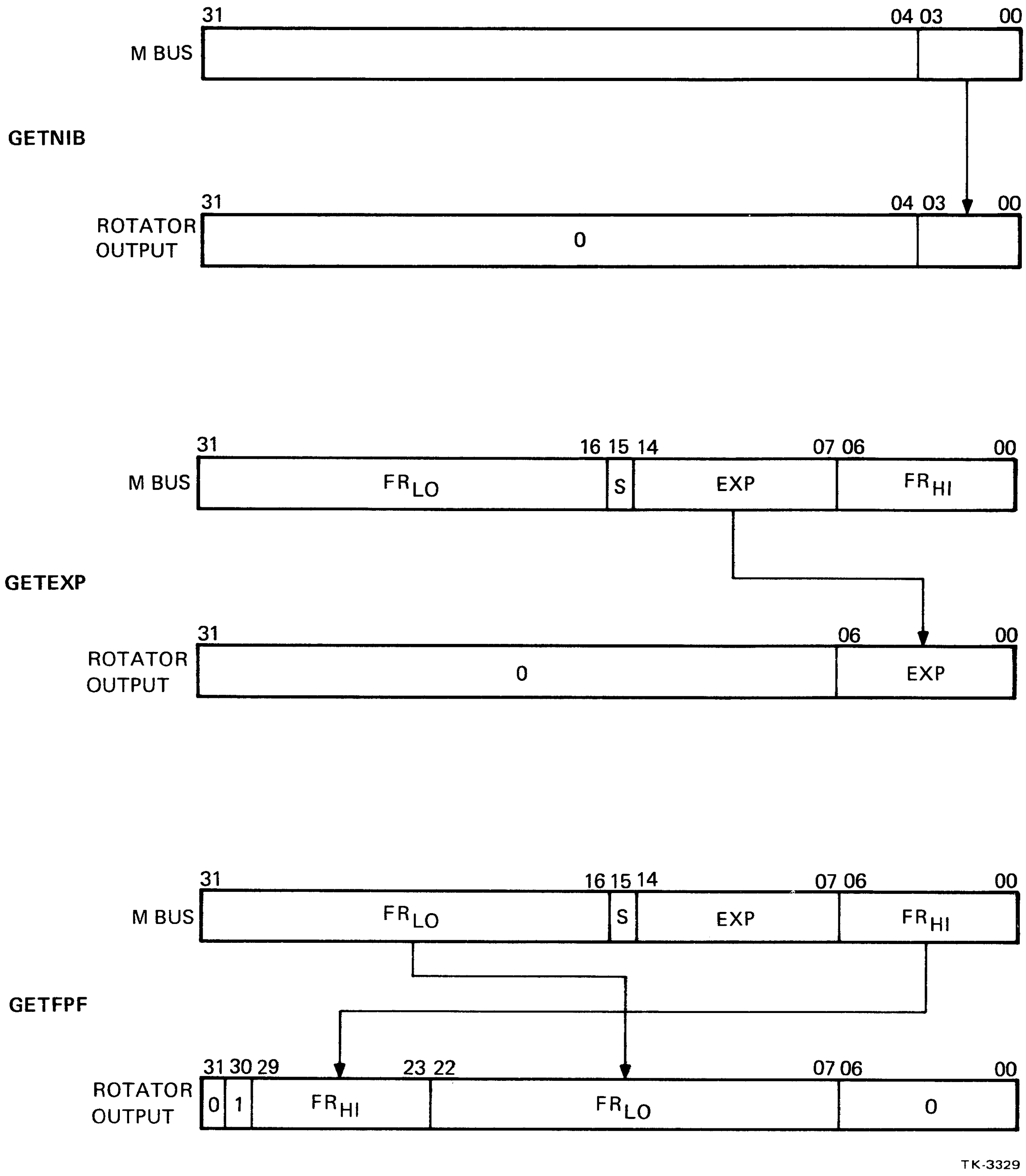
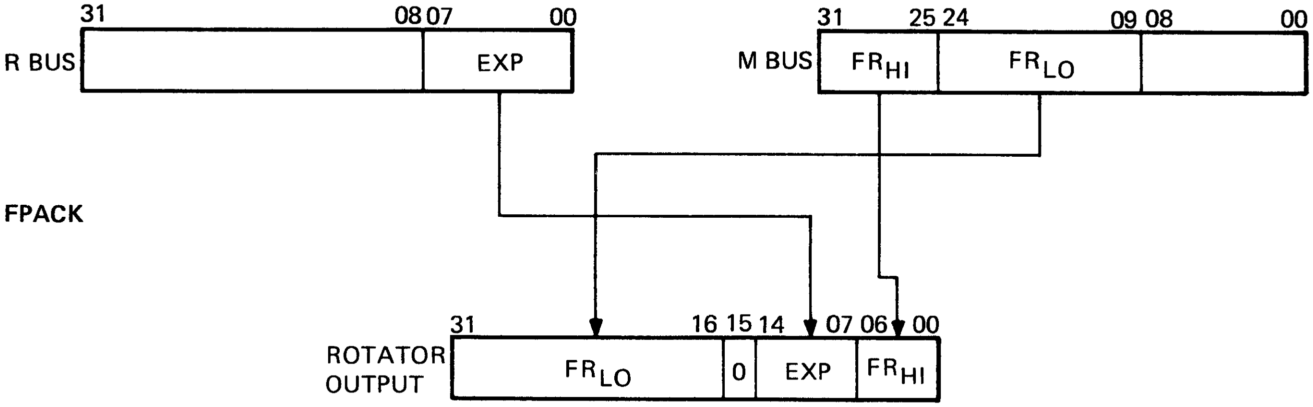
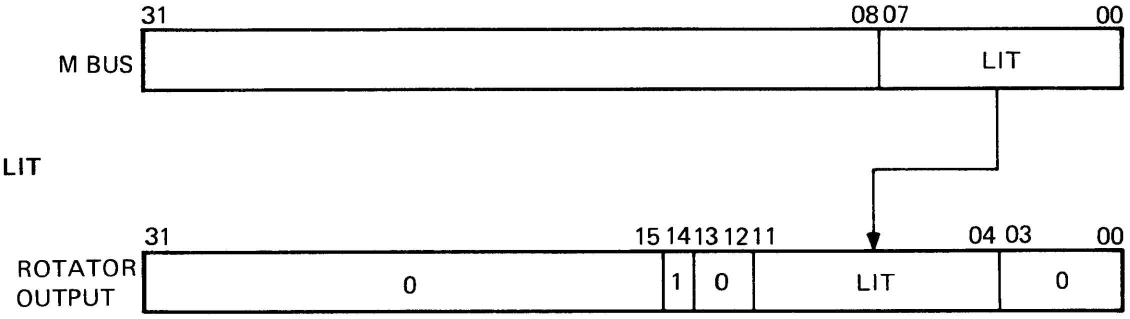


Figure 2-97 Get Functions



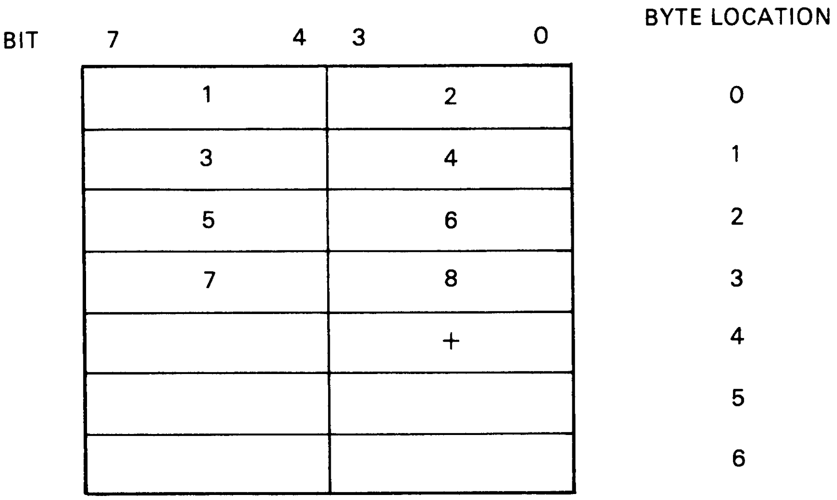
TK-3326

Figure 2-98 FPACK Function



TK-3328

Figure 2-99 FPLIT Function



BIT 7 4	3 0	BYTE LOCATION
1	2	0
3	4	1
5	6	2
7	8	3
	+	4
		5
		6

TK-0251

Figure 2-100 Memory Storage of a Decimal Number

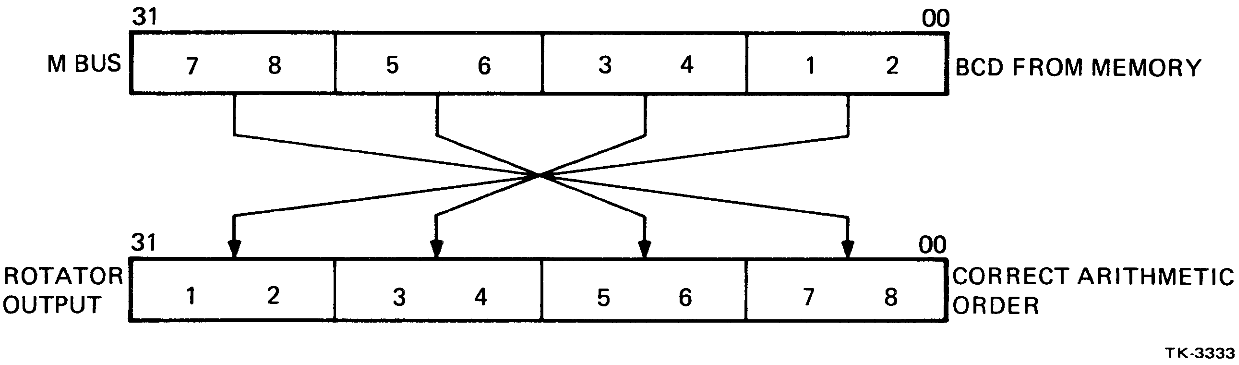


Figure 2-101 BCDSWP Function

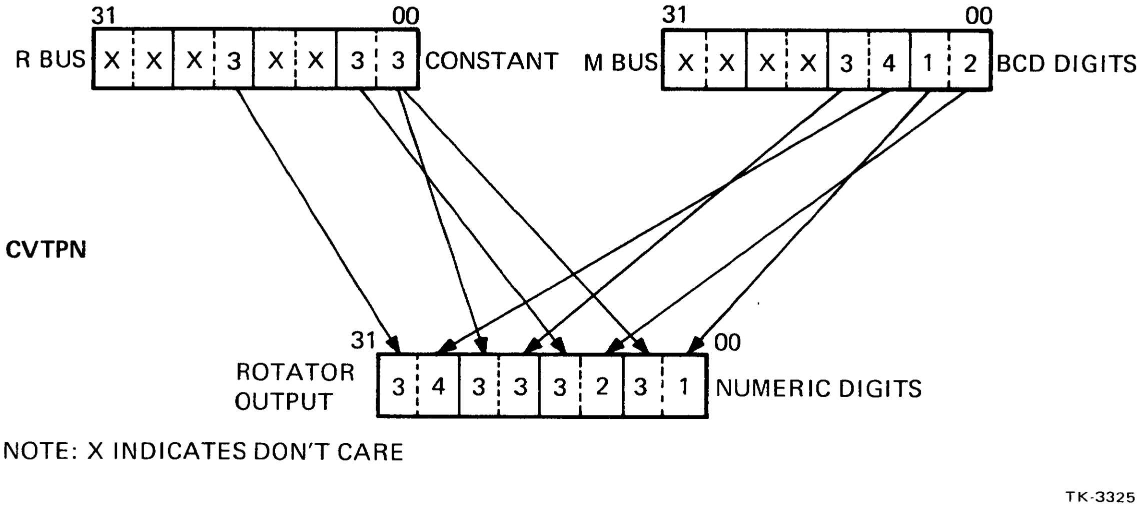


Figure 2-102 CVTPN Function

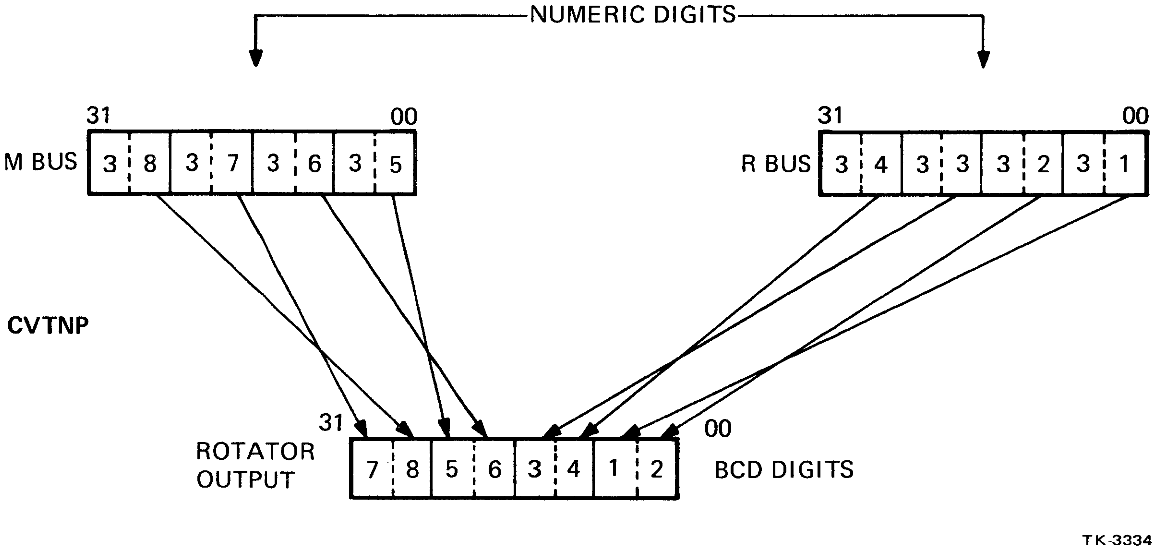
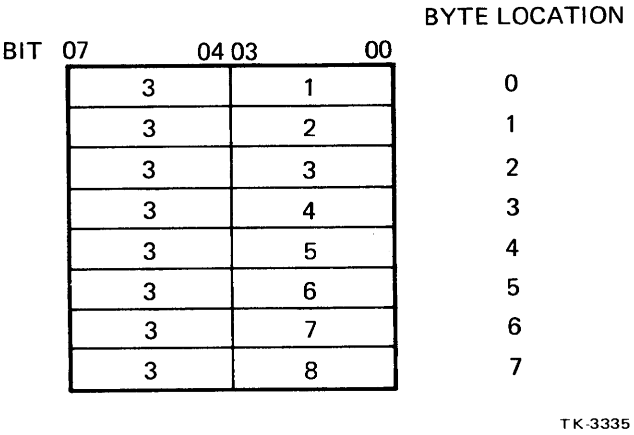


Figure 2-103 CVTNP Function



BIT 07–04	03–00	BYTE LOCATION
3	1	0
3	2	1
3	3	2
3	4	3
3	5	4
3	6	5
3	7	6
3	8	7

TK-3335

Figure 2-104 Memory Storage of a Numeric String

2.6.6.4 Rotator Control (SRK) – The super rotator is controlled by the super rotator control chip, SRK. The SRK decodes the ROT microfield to generate control signals for the super rotator. In addition, the SRK generates status signals for microbranches.

Figure 2-105 illustrates the basic logic structure of the SRK chip. As seen in this figure, the SRK contains two 6-bit latches: the P latch (position latch), and S latch (size latch). These latches are generally used to specify the size of a bit field (S latch) and the number of bit positions for the shift (P latch). The SRK contains three additional areas of logic: position logic, function logic and status logic. The position and function logic areas generate the output control signals for the super rotator. These control signals are described in Paragraph 2.6.6.4.1. The status signals generated by the status logic are discussed in Paragraph 2.6.6.4.2.

The ROT microfield is used to explicitly specify a rotator function. In addition, the ROT microfield is used to load or read the P and S latches. These latches may be loaded from the SBus or WBus as specified by the ROT microfield. Likewise, the contents of either latch may be read onto the SBus. Table 2-53 (in Paragraph 2.6.6.3) lists each value of the ROT microfield and the selected function.

Data from the S latch or P latch is output onto the SBus in the format illustrated in Figure 2-106. Note that for this case the SRM chips will output all zeros in bit positions <31:08>.

2.6.6.4.1 Control Signals – The SRK uses the ROT microfield to encode three groups of control signals for the super rotator (refer to Figure 2-105). The three groups of signals are:

Primary Function Signals – PRI <1:0>
Secondary Function Signals – SEC <5:0>
Shift Signals – SHF <4:0>

As seen in Figure 2-105, these signals are also dependent on the D-size signals from the microsequencer, the zero indicator signals from the ALP, and the contents of the S and P latches.

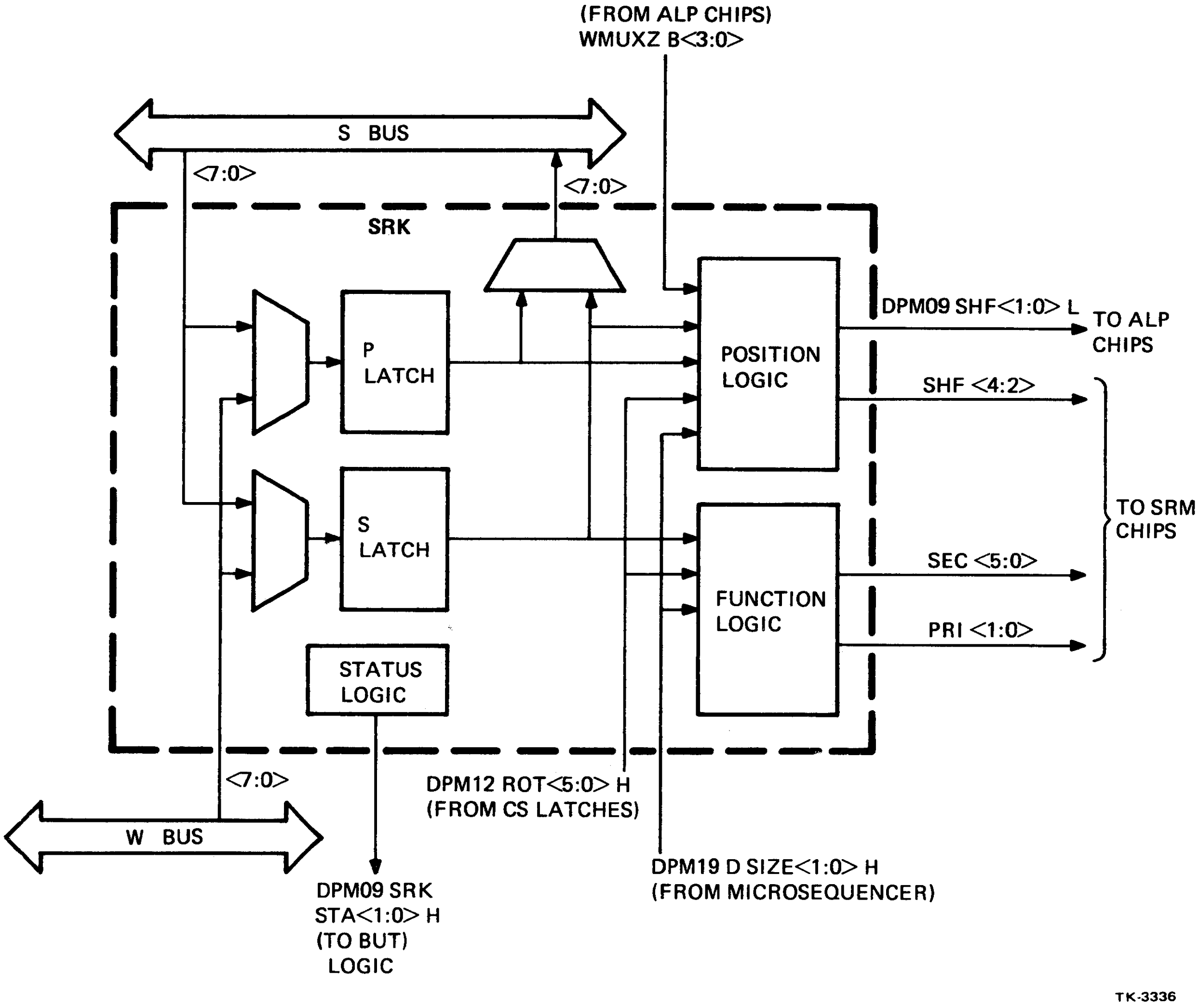


Figure 2-105 SRK Logic

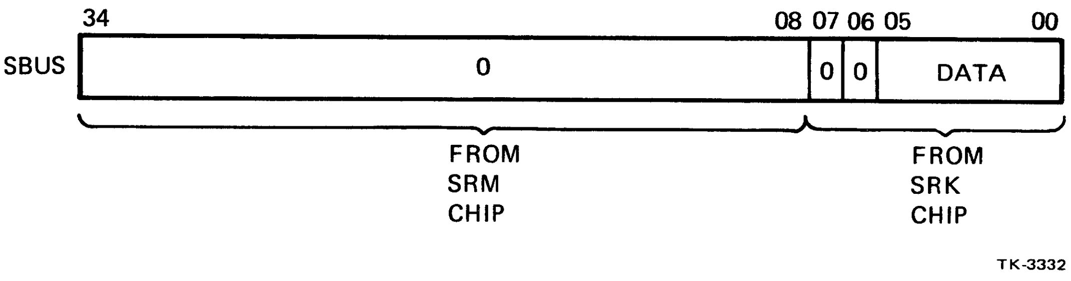


Figure 2-106 Data from S or P Latch

The primary function signals, PRI <1:0> are used to select one of the three primary function types as follows.

PRI <1:0>		**Primary Function Type**
H	H	EXTZ M,R
H	L	EXTZ M,M
L	H	EXTZ R,R

These functions basically extract and zero-extend a bit field. When one of these functions is specified by the PRI signals, the SHF and SEC signals are used to indicate the first and last bits of the bit field to be extracted. These extract/zero-extend functions are described below in this paragraph and in Paragraph 2.6.6.3.

If both primary function signals are low, SEC <3:0> are used to specify a secondary function type as follows.

SEC <3:0>				**Secondary Function Type**
H	H	H	H	CLR 1 BYTE
H	H	H	L	CLR 2 BYTE
H	H	L	H	LO BYTE OFF
H	H	L	L	CLR 3 BYTE
H	L	H	H	ASL R
H	L	H	L	ASL M
H	L	L	H	LIT ONE
H	L	L	L	LIT ZERO
L	H	H	H	FP FRACT
L	H	H	L	BCD SWAP
L	H	L	H	CVTPN
L	H	L	L	FP PACK
L	L	H	H	ASR M
L	L	H	L	CONSTANT 8
L	L	L	H	CVTNP
L	L	L	L	FP LIT

Note that only four of the six SEC signals are used to define a secondary function type. Each secondary function type is briefly described below.

CLR 1 BYTE – The three high-order bytes from the MBus are transferred onto the SBus. The low byte is forced to zero. SBUS <34:32> are also forced to zero.

CLR 2 BYTE – Same as CLR 1 BYTE except that the lower two bytes are cleared.

LO BYTE OFF – SBUS <34:08> are output as all zeros. SBUS <07:00> are in a high impedance state.

CLR 3 BYTE – Same as CLR 1 BYTE except that the lower three bytes are cleared.

ASL R – Data from the RBus is shifted left by the number of positions specified by control inputs SHF <4:0>. Zeros are shifted into the vacant bit positions.

ASL M – Same as ASL R, except data from the MBus is used instead of data from the RBus.

LIT ONE – This function is dependent on the control input bit SEC <4>. If SEC <4> is high, the nine bits of data from the LITRL subfield of the microword are one-extended and rotated to the right by the number of positions specified by control inputs SHF <4:0>. If SEC <4> is low, a constant of all ones is generated.

LIT ZERO – The nine bits of data from the LITRL subfield of the microword are zero-extended and rotated right by the number of positions specified by SHF <4:0>.

FP FRACT – Extracts the fraction field of a floating point datum. Refer to Paragraph 2.6.6.3.

CVTPN – Converts a 4-digit BCD string to a 4-digit numeric string. Refer to Paragraph 2.6.6.3.

FP PACK – Assembles an exponent field and fraction field into a floating-point datum format. Refer to Paragraph 2.6.6.3.

ASR M – Data from the MBus is shifted right by the number of positions specified by control inputs SHF <4:0>. Zeros are shifted into the vacant bit positions.

CONSTANT 8 – Generates a constant of 8, 4, 2, or 1 to autoincrement or autodecrement a register. Refer to Paragraph 2.6.6.3.

CVTNP – Converts an 9-digit numeric string to a 8-digit BCD string. Refer to Paragraph 2.6.6.3.

FP LIT – Expands a floating-point short literal. Refer to Paragraph 2.6.6.3.

Table 2-55 lists the output control signals of the SRK for each rotator function selected. The values for the three groups of output signals are given in hexadecimal. Primary and secondary function types are listed in parentheses under the corresponding PRI and SEL columns.

For most of the functions listed in Table 2-55, two values are shown under the SEC column. The second value indicates the secondary function type defined by SEC <3:0>. The first value has no effect on the function, but is indicated for completeness.

For the PL_MSS function, the SRK generates control signals to rotate the left-most non-zero byte from the MBus onto SBUS <7:0>. To accomplish this, the SRK monitors zero-byte indicators (WMUXZ signals) from the ALP. These signals determine SHF <4:3> as follows.

WMUXZ <3:0>	**SHF <4:3>**
0XXX	11
10XX	10
110X	01
111X	00

SHF <2:0> are always encoded as all zeros for this function. The PRI and SEC signals are encoded as shown in Table 2-55.

Table 2-55 SRK Control Signal Output

ROT <5:0> (Hex)	Function Mnemonic	PRI <1:0>	SEC <5:0>	SHF <4:0>
0	XZ.MR	0 (EXTZ M,R)	Note 2	PL
1	XZ.MM	1 (EXTZ M,M)	Note 2	PL
2	XZ.RR	2 (EXTZ R,R)	Note 2	PL
3	ASR.M.P	3	3,C (ASR M)	PL
4	RR.MR.P	0 (EXTZ M,R)	3F	PL
5	RR.MM.P	1 (EXTZ M,M)	3F	PL
6	RR.RR.P	2 (EXTZ R,R)	3F	PL
7	RR.MR.S	0 (EXTZ M,R)	3F	SL
8	RL.RM.4	0 (EXTZ M,R)	3C	1C
9	RR.MR.4	0 (EXTZ M,R)	3C	4
A	RR.RR.SIZ	2 (EXTZ R,R)	3C	(DSIZE+1)*8
B	RR.MR.9	0 (EXTZ M,R)	3E	9
C	XZ.PTX	1 (EXTX M,M)	19	7
D	XZ.VPN	1 (EXTZ M,M)	15	9
E	RR.MM.SIZ	1 (EXTZ M,M)	39	(DSIZE+1)*8
F	GETNIB	1 (EXTZ M,M)	3	0
10	GETEXP	1 (EXTZ M,M)	A	7
11	RL.MM.PTE	1 (EXTZ M,M)	3A	17
12	CLR2BM	3	3,1 (CLR 2 BYTE)	0
13	CLR1BM	3	3,0 (CLR 1 BYTE)	10
14	CLR3BM	3	0,3 (CLR 3 BYTE)	0
15	ASL.R.7	3	2,4 (ASL R)	19
16	ZERO	3	3,5 (ASL M)	0
17	ASL.R.SIZ	Note 2	Note 2	Note 2
18	BCDSWP	3	0,9 (BCD SWAP)	0
19	GETFPF	3	3,8 (FP FRACT	1
1A	FPACK	3	2,B (FP PACK)	1
1B	CVTPN	3	2,A (CVTPN)	0
1C	CONX.SIZ	3	0,D (CONSTANT 8)	(3 –DSIZE)
1D	ASR.M.3	3	0,C (ASR M)	3
1E	FPLIT	3	2,F (FP LIT)	0
1F	CVTNP	3	0,E (CVTNP)	0

NOTES:

1. EXTZ = Extract/zero-extended functions, M = MBUS, R = RBUS, WB = WBUS low byte.
2. See description in text.
3. LIT input forced to all ones.

Table 2-55 SRK Control Signal Output (Cont)

ROT <5:0> (Hex)	Function Mnemonic	PRI <1:0>	SEC <5:0>	SHF <4:0>
20	RL.RM.PS	0 (EXTZ M,R)	3F Note 2	−(PL + SL)
21	RL.MM.P	1 (EXTZ M,M)	3F	−(PL)
22	RL.RR.P	2 (EXTZ R,R)	3F	−(PL)
23	RL.RM.P	0 (EXTZ M,R)	3F	−(PL) Note 2
24	RR.MR.PS	0 (EXTZ M,R)	3F	(PL + SL)
25	RR.MM.PS	1 (EXTZ M,M)	3F	(PL + SL)
26	RR.RR.PS	2 (EXTZ R,R)	3F	(PL + SL)
27	PL_MSS	1 (EXTZ M,M)	3F	Note 2
28	ASL.R.P	3	3,4 (ASL R)	−(PL) Note 2
29	ASL.M.P	3	3,5 (ASL M)	−(PL) Note 2
2A	ASR.M.-P	3	3,C (ASR M)	−(PL)
2B	ZLITPL	3	3,7 (LIT ZERO)	−(PL)
2C	PL	3	1,2 (LO BYTE OFF)	C
2D	PL.SL_WB	3	1,2 (LO BYTE OFF)	8
2E	SL	3	3,2 (LO BYTE OFF)	18
2F	SL.PL_WB	3	0,2 (LO BYTE OFF)	Same as ROT = 27
30	ZLIT0	3	0,7 (LIT ZERO)	0
31	ZLIT28	3	3,7 (LIT ZERO)	4
32	ZLIT24	3	3,7 (LIT ZERO)	8
33	ZLIT20	3	3,7 (LIT ZERO)	C
34	ZLIT16	3	0,7 (LIT ZERO)	10
35	ZLIT12	3	2,7 (LIT ZERO)	14
36	ZLIT8	3	3,7 (LIT ZERO)	18
37	ZLIT4	3	3,7 (LIT ZERO)	1C
38	OLIT0	3	0,6 (LIT ONE)	0
39	MINUS1	3	3,6 (LIT ONE) Note 3	0
3A	OLIT24	3	2,6 (LIT ONE)	8
3B	OLIT0.PL_LIT	3	2,6 (LIT ONE)	0
3C	OLIT16	3	0,6 (LIT ONE)	10
3D	OLIT0.SL_LIT	3	0,6 (LIT ONE)	0
3E	OLIT8	3	2,6 (LIT ONE)	18
3F	OLIT0.PL43_WB	3	0,6 (LIT ONE)	0

NOTES:

1. EXTZ = Extract/zero-extended functions, M = MBUS, R = RBUS, WB = WBUS low byte.
2. See description in text.
3. LIT input forced to all ones.

The control signals for the ASL.R.SIZ, ASL.M.P, ASL.R.P, RL.RM.PS, and RL.RM.P functions also require further discussion. For the ASL.R.P function, all output control signals are dependent on the value of D-size as follows.

D-Size <1:0>	**PRI <1:0>**	**SEC <5:0>**	**SHF <4:0>**
0	2 (EXTZ R,R)	34	0
1	3	3,4 (ASL R)	1F
2	3	3,4 (ASL R)	1E
3	3	3,4 (ASL R)	1D

Note that when D-size equals zero, the function performed is extract/zero-extend rather than arithmetic shift. For the ASL.M.P and ASL.R.P functions, a constant of 0 results if PL <4:0> equals zero.

If PL <4:0> equals zero during the RL.RM.P function, the RBus is sourced onto the SBus. The same sourcing operation is performed if PL <4:0> and SL <4:0> equal zero during the RL.RM.PS.

For the explicit extract/zero-extend functions (ROT = 0, 1, or 2), the SHF and SEC signals are encoded to position a bit field and define its length, respectively. Refer to Figure 2-96 for an illustration of this type of function.

Figure 2-107 illustrates the encoding of the SHF and SEC signals. SHF <4:2> are encoded to specify the number of nibbles to shift. These signals are defined by bits <4:2> of the P latch. SHF <1:0> are encoded to specify the final bit shift (0, 1, 2, or 3). These bits are defined by bits <1:0> of the P latch. (Note that SHF <1:0> are sent to the ALP for the second level shift instead of the SRM.)

The encoding of the SEC signals is somewhat more complicated than the encoding of the SHF signals as shown in Figure 2-107. For these signals, an arithmetic operation is performed within the SRK chip. Bits <1:0> of the P latch are added to the contents of the S latch minus 1. Bits <5,3:0> of the result are directly used to generate SEC <5,3:0>. The encoding of SEC <4>, however, is dependent on bit <5> of the arithmetic result. If bit <5> is asserted, SEC <4> is asserted. If this bit is not asserted, SEC <4> is set to the condition of bit <4> of the result. Refer to Figure 2-107.

2.6.6.4.2 SRK Status Signals – The status logic of the SRK chip generates two status signals during each microcycle. These signals, DPM09 SRK ST <1:0> H, are used for microbranches (Paragraph 2.2.1.2).

If the BUT field specifies SRKSTA (Paragraph 2.2.1.2), bits <63:58> of the microword define the ROTSRK subfield. Table 2-56 lists the conditions that set the status signals for each value of ROTSRK <5:0>. The use of these signals for various microbranches is discussed below. (Refer to Paragraph 2.2.1.2 for a complete description of the microbranches.)

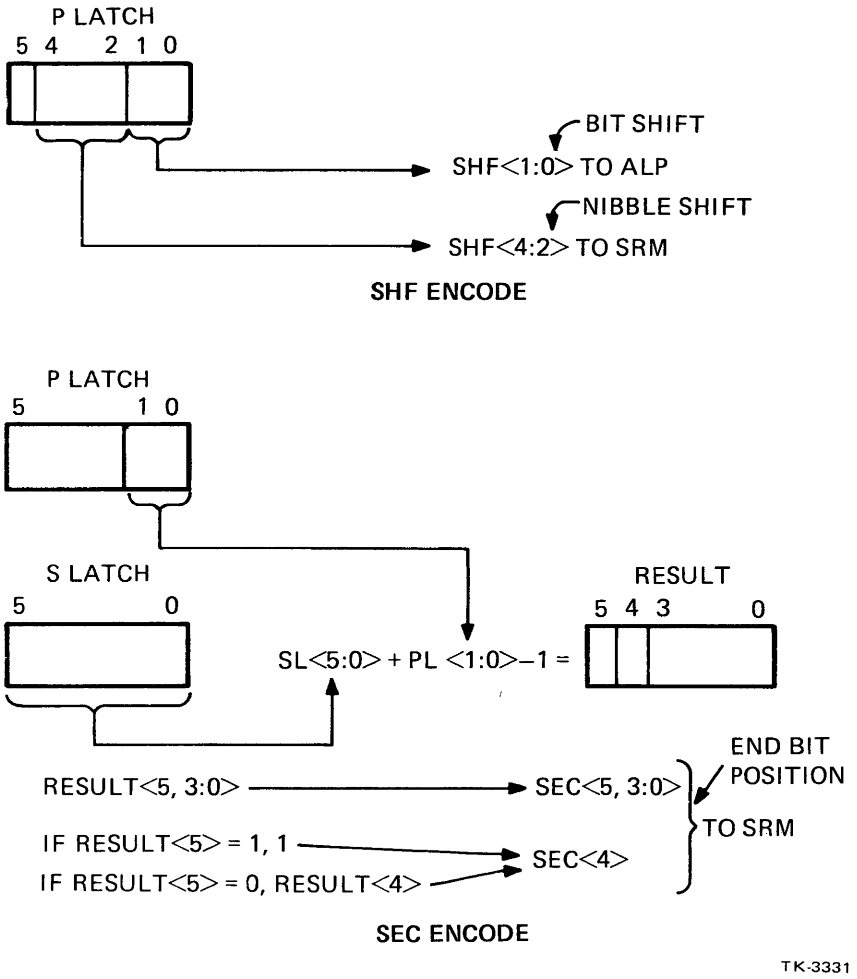


Figure 2-107 Control Signal Encoding for the Extract/Zero Extended Functions

Table 2-56 SRK Status Signals

ROTSRK <5:0> (Hex)	Test Mnemonic	Conditions that Set SRKSTA<1> (See Note 1)	Conditions that Set SRKSTA<0> (See Note 1)
0	VIELD.000	SL.EQ.0	(PL<4:0>+SL).GT.32
1	VIELD.001	SL.EQ.0	(PL<4:0>+SL).GT.32
2	VIELD.002	SL.EQ.0	(PL<4:0>+SL).GT.32
3	PL5.003	0	PL<5>
4	VIELD.010	SL.EQ.0	(PL<4:0>+SL).GT.32
5	VIELD.011	SL.EQ.0	(PL<4:0>+SL).GT.32
6	VIELD.012	SL.EQ.0	(PL<4:0>+SL).GT.32
7	PL5.013	0	PL<5>

Table 2-56 SRK Status Signals (Cont)

ROTSRK <5:0> (Hex)	Test Mnemonic	Conditions that Set SRKSTA<1> (See Note 1)	Conditions that Set SRKSTA<0> (See Note 1)
8	DSIZE.020	DSIZE<1>	DSIZE<0>
9	DSIZE.021	DSIZE<1>	DSIZE<0>
A	DSIZE.022	DSIZE<1>	DSIZE<0>
B	DSIZE.023	DSIZE<1>	DSIZE<0>
C	DSIZE.030	DSIZE<1>	DSIZE<0>
D	DSIZE.031	DSIZE<1>	DSIZE<0>
E	DSIZE.032	DSIZE<1>	DSIZE<0>
F	DSIZE.033	DSIZE<1>	DSIZE<0>
10	BCDSIGN.040	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,3)
11	BCDSIGN.041	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,13)
12	BCDSIGN.042	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,13)
13	BCDSIGN.043	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,13)
14	ASCIISIGN.050	Note 2	Note 2
15	ASCIISIGN.O51	Note 2	Note 2
16	ASCIISIGN.052	Note 2	Note 2
17	ASCIISIGN.O53	Note 2	Note 2
18	BCDSIGN.060	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,13)
19	BCDSIGN.061	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,13)
1A	BCDSIGN.062	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,13)
1B	BCDSIGN.063	SBUS<3:0>.NE.0	SBUS<3:0>.NE.(11,13)
1C	ASCIISIGN.070	Note 2	Note 2
1D	ASCIISIGN.071	Note 2	Note 2
1E	ASCIISIGN.072	Note 2	Note 2
1F	ASCIISIGN.073	Note 2	Note 2
20	SL.EQ.0.100	SL.EQ.0	Undefined
21	SL.EQ.0.SIGN.101	SL.EQ.0	PL<5>
22	SL.EQ.0.SIGN.102	SL.EQ.0	PL<5>
23	WX.NE.0.103	WMUX<31:16>.NE.0	WMUX<15:00>.NE.0
24	VIELD.110	SL.EQ.0	(PL<4:0>+SL).GT.32
25	VIELD.111	SL.EQ.0	(PL<4:0>+SL).GT.32
26	VIELD.112	SL.EQ.0	(PL<4:0>+SL).GT.32
27	WX.NE.0.113.D	WMUX<31:16>.NE.0	WMUX<15:00>.NE.0
28	PL.EQ.0.SIGN.120	PL<4:0>.EQ.0	PL<5>
29	PL.EQ.0.SIGN.121	PL<4:0>.EQ.0	PL<5>
2A	PL.EQ.0.SIGN.122	PL<4:0>.EQ.0	PL<5>
2B	PL.EQ.0.123	PL<4:0>.EQ.0	0
2C	WBRANGE.130	Note 3	Note 3
2D	WBRANGE.131.D	Note 3	Note 3
2E	WBRANGE.132	Note 3	Note 3
2F	WBRANGE.133.D	Note 3	Note 3

Table 2-56 SRK Status Signals (Cont)

ROTSRK <5:0> (Hex)	Test Mnemonic	Conditions that Set SRKSTA<1> (See Note 1)	Conditions that Set SRKSTA<0> (See Note 1)
30	ABSVAL.140	Note 4	Note 4
31	ABSVAL.141	Note 4	Note 4
32	ABSVAL.142	Note 4	Note 4
33	ABSVAL.143	Note 4	Note 4
34	ABSVAL.150	Note 4	Note 4
35	ABSVAL.151	Note 4	Note 4
36	ABSVAL.152	Note 4	Note 4
37	ABSVAL.153	Note 4	Note 4
38	ABSVAL.160	Note 4	Note 4
39	ABSVAL.161	Note 4	Note 4
3A	ABSVAL.162	Note 4	Note 4
3B	ABSVAL.163.D	Note 4	Note 4
3C	ABSVAL.170	Note 4	Note 4
3D	ABSVAL.171.D	Note 4	Note 4
3E	ABSVAL.172	Note 4	Note 4
3F	ABSVAL.173.D	Note 4	Note 4

NOTES:

1. All values are listed in decimal.
2. ASCII Sign Check; see Table 2-57.
3. WBus Range Check; see Table 2-57.
4. Absolute Value Check; see Table 2-57.
5. SL must be in the range of (1, 32), otherwise results are undefined.
6. Note that the rotator function implied by the value in ROT <5:0> is performed even though the rotator output is not used. (Refer to Table 2-53 for a list of the rotator functions.)

As mentioned in Paragraph 2.6.6.1, the ALUXM, ALUSHF, and ALUCI subfields are also defined when the ROTSRK subfield is defined. The test mnemonics shown in Table 2-56 illustrate this concept. Each test mnemonic indicates the basic check specified by ROTSRK <5:0> in addition to the values of the ALUXM, ALUSHF, and ALUCI subfields. For example, BCDSIGN.063 is interpreted as follows:

```
BCDSIGN. 063
                ALUCI <1:0> = 3
Type of         ALUSHF <2:0> = 6
Check           ALUXM <0> = 0
```

Note that for this reason more than one value of the ROTSRK subfield can specify one type of check (i.e., the values for ALUXM, ALUSHF and ALUCI may be different.) Refer to Paragraph 2.6.5.1.5 for a complete description of the ALUXM, ALUSHF, and ALUCI subfields.

The type of check shown in the mnemonics in Table 2-56 is read as follows:

Type of Check	Meaning
VIELD	Field Violation Check
D-SIZE	D-Size Check
BCDSIGN	BCD Sign Check
ASCIISIGN	ASCII Sign Check
WBRANGE	WBus Range Check
ABSVAL	Absolute Value Check

For the field violation check (VIELD), SRKSTA <0> is used to indicate whether the bit field to be extracted overlaps a longword boundary. For this type of check, the SRK performs the arithmetic operation shown in Table 2-56. If the result is greater than 32, a longword boundary has been violated and SRKSTA <0> is set.

A BCD sign check can be performed during a BCD instruction by monitoring the SRKSTA signals. For this check (BCDSIGN) SRKSTA <1> indicates whether SBUS <3:0> is zero; SRKSTA <0> indicates whether SBUS <3:0> contains a negative sign.

The SRKSTA signals can also be used to indicate the context size of the current instruction. To accomplish this, a D-size check is invoked to cause the SRK to output the two D-size signals from the microsequencer.

For the ASCII sign check, WBus range check, and absolute value check, SRKSTA <1> and <0> must be interpreted together. Table 2-57 lists the condition indicated for each value of the status signals. The ASCII sign check is typically used during decimal string instructions for a numeric sign test. For this check, a value of 00 indicates ASCII- , 01 indicates ASCII+ or space, and an 11 indicates that WBus data is not in ASCII format.

Table 2-57 ASCIISIGN, WBRANGE, ABSVAL

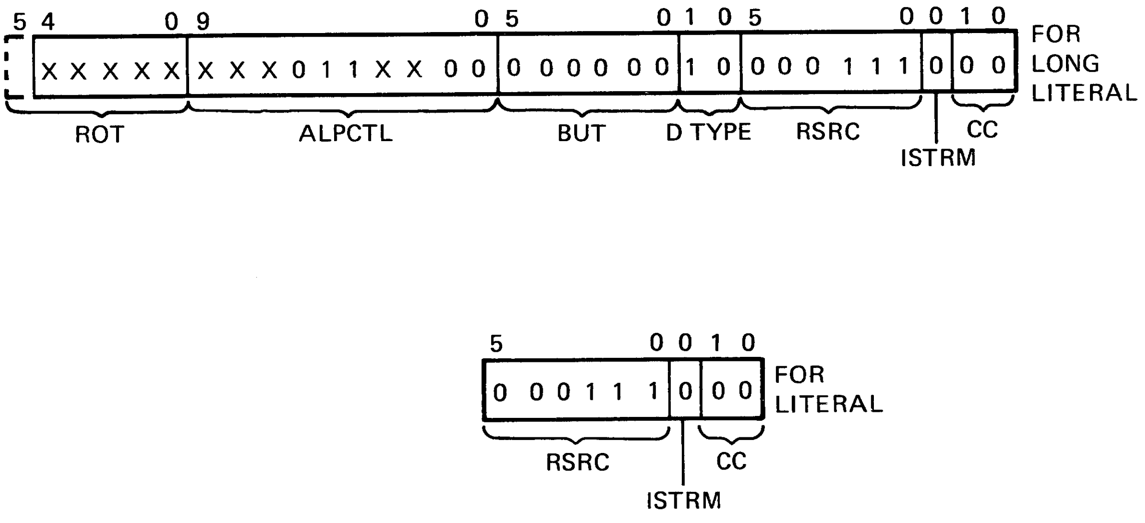
Basic Check	SRKSTA <1:0>	Indication (All Values in Decimal)
ASCII Sign	0 0	WBUS<7:0>.EQ.45
	0 1	WBUS<7:0>.EQ.(32,43)
	1 0	Machine Check
	1 1	WBUS<7:0>.NE.(32,43,45)
WBUS Range (unsigned)	0 0	WBUS<7:0>.EQ.(1 to 31)
	0 1	WBUS<7:0>.EQ.0
	1 0	WBUS<7:0>.EQ.32
	1 1	WBUS<7:0>.GT.32
Absolute Value	0 0	WBUS<7:0>.EQ.(−1 to−31)
	0 1	WBUS<7:0>.LT.−31
	1 0	WBUS<7:0>.EQ.(0 to 31)
	1 1	WBUS<7:0>.GT.31

2.6.6.5 Literal/Long Literal Control – A 9-bit or 32-bit literal can be entered into the data path directly from the microword. This mode is selected by a 2-bit field in the microword, LIT <1:0>. The LIT microfield specifies the interpretation of other microfields as shown in Table 2-58.

Table 2-58 Interpretation of the LIT Microfield

LIT <1:0>	Mnemonic	Description
0 0	NORMAL	NOP, all microfields are interpreted normally.
0 1	LITRL	The RSRC, ISTRM, and CC microfields define a 9-bit literal.
1 0	FPAWAIT	Used by microsequencer to sync with FPA.
1 1	LONLIT	The ROT <4:0>, ALPCTL, BUT, DTYPE, RCRC, ISTRM, and CC microfields define the 1;s complement of a 32-bit literal.

When the LIT microfield specifies either LITRL or LONLIT, the original control functions of the corresponding microfields are void. For the control of the associated logic sections, these microfields take on a hardwired default value. The defaulted values are illustrated in Figure 2-108. Note that these values do not represent the literal itself, but rather the value of the microfields as seen by the hardware they control.



NOTES:

1. ROT<5> MUST BE ZERO TO ENSURE THAT THE P OR S LATCH IS NOT MODIFIED. MICROCODE MUST ENFORCE THIS RESTRICTION.

2. X = NO DEFAULT.

TK-3330

Figure 2-108 Defaulted Literal and Long Literal Values Used by Control Logic

The value of a literal (short) taken from the microword is directly input to the SRM from the CS latches. These microfields are input to the SRM whether or not a literal is specified in the microword. Figure 2-109 illustrates the literal/long literal control logic.

As seen in Figure 2-109, the microfields associated with the long literal are input to the long literal register from the CS latches. The output of this register is connected to the RBus. When the LIT microfield specifies a long literal, DPM20 LONG LIT L is asserted. This signal selects the QD clock for the clocking of the long literal register. Note that the QD clock is selected by a multiplexer in the CLA chip. The assertion of another signal from the SPA chip (DPM11 LITREG EN L) enables the contents of the long literal register onto the RBus. Note that the long literal placed on the RBus is the 1's complement of the 32-bit literal in the microword.

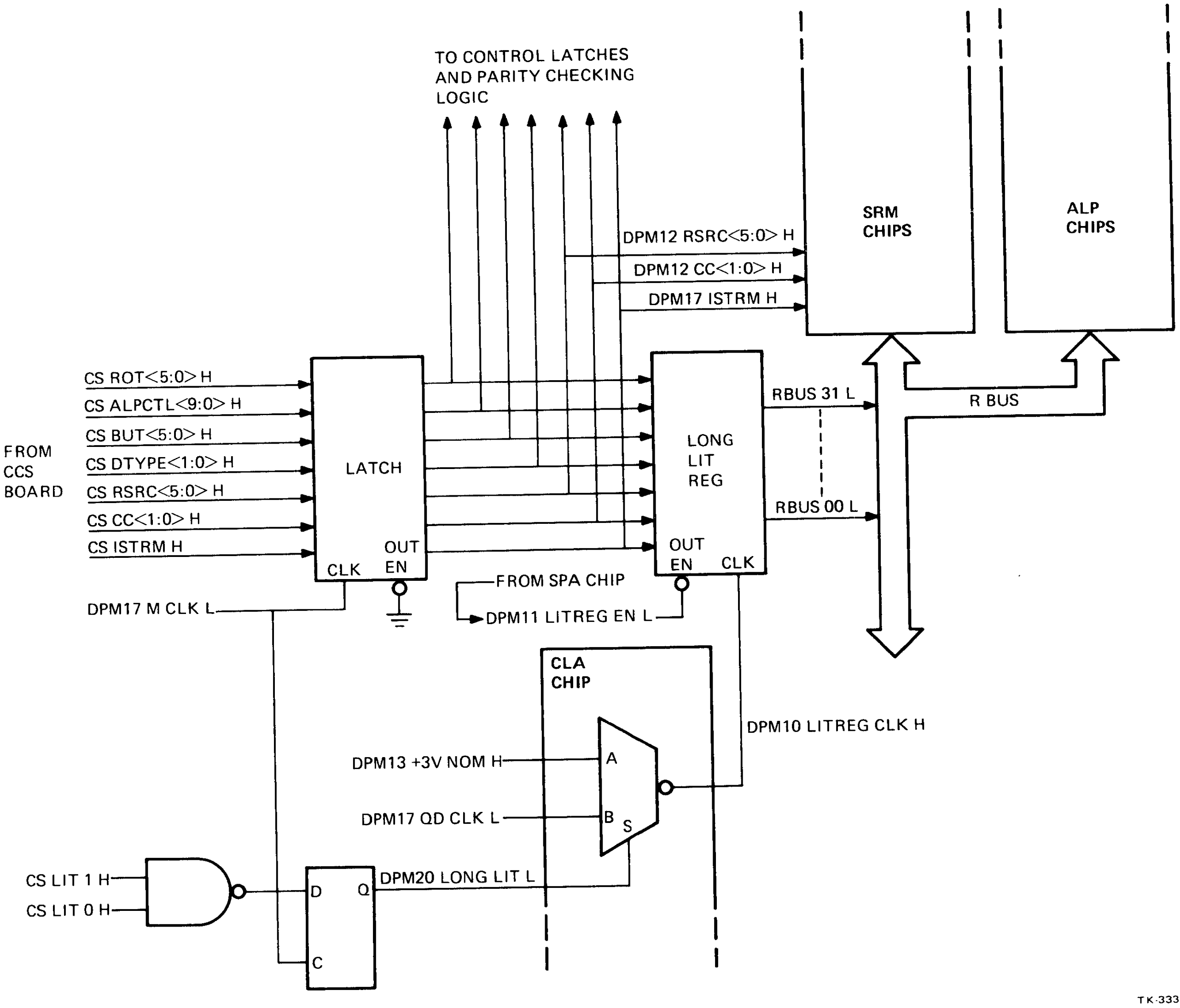


Figure 2-109 Literal/Long Literal Control

2.7 INTERVAL TIMER AND TIME-OF-YEAR CLOCK

2.7.1 Introduction to Interval Timer

The interval timer is used primarily to schedule events and control the amount of time a particular task can operate. The operation of the VAX-11/750 interval timer from the software level is consistent with other VAX processors. Most of the timer is implemented within a gate array called TOK. The timer is implemented using a 10-MHz TTL oscillator, a divide by 10, and the TOK gate array. The timer is incremented at 1- μs intervals, which makes the operation consistent with other VAX timers. The maximum interval then could be expressed as (((2**32)-1)*.000001)/60 which works out to be approximately 71 hours or 3 days. External dedicated scratchpads are required to maintain the interval count. The interval timer is accessible to the VAX-11 macrocode through internal processor registers (IPRs). These IPRs can be accessed with MTPR and MFPR macroinstructions, and also from the console terminal: Internal processor registers are described in Paragraph 2.7.3.

The interval timer operates as follows. The operating system loads the timer with 2's complement of the desired interval a particular task must run. The timer is started with an MTPR instruction. When the timer overflows at the end of the desired interval, a macro-level interrupt is requested. If the IPL level of the timer interrupt request (IPL 18) is greater than the current PSL IPL, the timer service macroroutine is entered via SCBB+C0. This terminates the current task, if an event of higher priority has not already done so.

2.7.2 Detailed Description of the Timer Circuitry

Refer to the module schematic schematic page CCS 14. E5 on CCS 14 is the 10 MHz TTL oscillator that provides the time base for the interval timer gate array (TOK) on the DPM module. The output from E5 goes to E4 IC, which is a decade divider. The output of E4 is a symmetrical 1 MHz signal that provides the increment interval of 1 μs. The signal TOK OSC OUT H is wired from slot 5 (CCS module) to slot 2 (DPM module). The TOK gate array is shown in the lower left corner of DPM 13. The signal TOK OSC OUT H enters the DPM module and goes to pin 45 of the TOK gate array. The other inputs to the TOK gate array are PROC INIT L, which clears any interrupt requests left in the gate array and sets the logic to a known state. B CLK L and D CLK ENABLE H are used internally to form a D CLK to load the timer control and data registers.

Access to the gate array is entirely controlled by the WCTRL field of the microword which is used in the MTPR and MFPR macroinstructions and the interval timer service microroutines. There is a bidirectional interface to the CPU WBus for reading and writing the timer control and data registers. The signal TIMER SERVICE H that exits the TOK gate array is used to signal the microcode that a microroutine to update the high half of the interval count, or a transfer of data to the ICR register from the NICR register, is necessary. The signal TIMER INT L is the timer interrupt request that is generated when the interval timer overflows. This goes to the INT gate array on UBI so the interrupt request can be arbitrated among the other requests. Timer functionality is verified with the hardcore instruction test EVKAA.

2.7.3 Interval Timer Firmware Requirements

Figure 2-110 shows the VAX-11/750 interval timer internal processor registers (IPRs) as they appear to the software. There are three registers associated with the interval timer. IPR 19 is the next interval count register (NICR). This register is loaded with the 2's complement of the desired interval. The number loaded into this register is the 2's complement of the desired interval in seconds divided by 1 μs. The IPR 1A is the interval count register (ICR). It contains the current count of the timer at all times. The ICR is loaded from the NICR and the value in the NICR does not change unless an MTPR instruction writes new data into it. IPR 18 is the interval counter control and status register (ICCS). This register controls the operation of the interval timer. The functions of the bits in the ICCs are explained below.

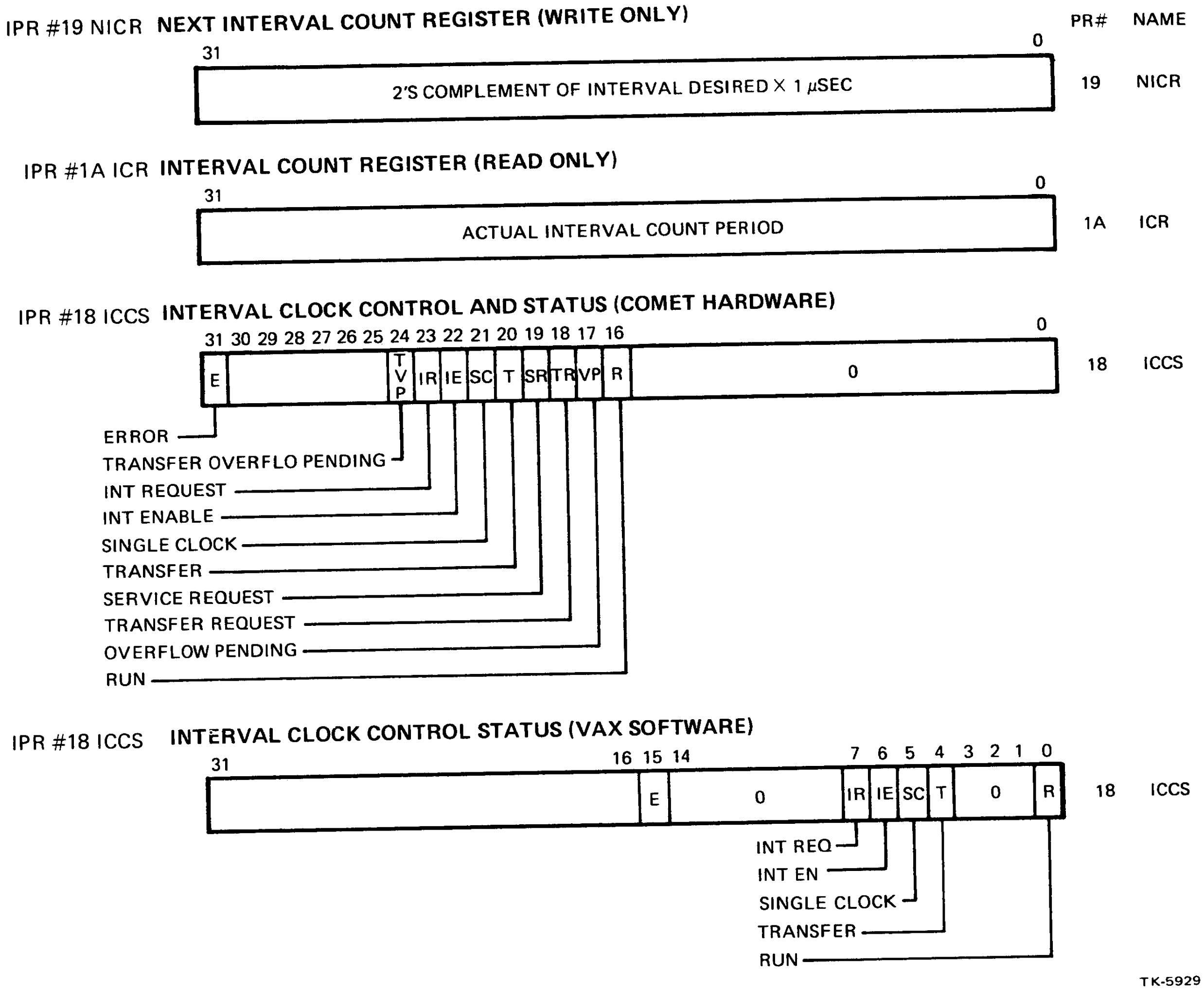


Figure 2-110 Interval Timer Processor Registers

ICCS Bit	Name	Function
<15>	ERROR	This bit is set if an improper operation is attempted: for example, start the timer without clearing the interrupt request (IR) from the previous timer overflow.
<7>	IR	Interrupt request is set when the timer overflows.
<6>	IE	Interrupt enable must be set by the VAX macrocode to enable timer interrupt requests at ICCS.
<5>	SC	This is a write-only bit that the macroprogrammer can use to step the interval clock one count at a time. Each write to the ICCS with bit <5> = 1 steps the interval timer one count.
<4>	TR	Transfer moves the NICR contents to the ICR.
<0>	RUN	This bit starts the interval counter incrementing until it overflows. This bit is set after the NICR is transferred to the ICR.

Figure 2-111 shows how the hardware is implemented. The TOK gate array does not contain all the circuitry to make the timer function. The first register in Figure 2-111 shows the TOK control bits in the high half of the WBus bits. The lower 15 bits of the TOK gate array can be read either as bits <15:0> of the NICR or as <15:0> of the ICR. The high halves of both the NICR and ICR are maintained in an RTEMP scratchpad dedicated to the timer. This means that when the lower 16 bits of the ICR are going to overflow, a carry from bit 15 must be added to the contents of the scratchpad that contains the high half of the ICR. This is accomplished by forcing a timer service trap at BUT Service to microvector to control store address 0014. Location 0014 contains the microservice routine that updates the scratchpad portion of the ICR.

The RTEMP scratchpad that contains the high half of the ICR is a single 32-bit location called R[SPNICR.SPICR]. The scratchpad location contains the high 16 bits of the NICR in bit positions <31:16>, and the high half of the ICR is stored in bits <15:0> of R[SPNICR.SPICR] (see Figure 2-111). The timer service microcode has to access the scratchpad by rotating the contents. The NICR is scratchpad memory in bits <31:16> and bits <15:0> actually reside in the TOK gate array. The same is true of the ICR. The ICCS shown in the bottom register reside in the TOK gate array, and interface to WBus bits <31:16>. The MTPR and MFPR instructions have to rotate the write and read data to the ICCS 16 bits to the left. TOK control bits are as follows.

TOK Bit	Function
VP (WBUS <17>)	This bit is set by the microcode in the interval timer service microroutine to indicate that the contents of the scratchpad ICR (SPICR) is all ones. This informs the TOK gate array that the next ICR overflow should set TIMER INT L.
TR (WBUS <18>)	TR is set in the TOK gate array after an MTPR initiates a transfer to the NICR. TR is not the same as transfer (WBUS <20>) which is set by the macroprogram to initiate the transfer of the NICR data to the ICR.
SR (WBUS <19>)	SR means service request. SR is set by the TOK gate array to request service from the timer service microroutine to update the SPICR after the ICR overflows.
TVP (WBUS <24>)	This bit is set by the microcode to tell the TOK gate array that the SPNICR is equal to −1. This enables the VP to set when the a transfer to the ICR is done and it prevents the ICR from being auto-loaded after interrupt.

2.7.4 Timer Service and Interrupts

The signal TIMER SERVICE H from the TOK gate array is asserted for two conditions. The first is if SR is set, indicating an overflow from ICR <15:0>, and the second is if TR is set, indicating that the previous macroinstruction was an MTPR that set the transfer bit (WBUS <20>). At the next BUT Service, the timer service request, if honored, invokes the timer service microroutine that begins at control store address 0014. This routine has to determine if there is a service request (SR) or transfer request (TR) and do the appropriate service. A service request (SR) means the microcode has to increment the SPICR. A transfer request (TR) causes the SPNICR to be moved to SPICR. Once the service request is completed the microroutine backs up the PC and does IRD 1 on the VAX-11/750 macroinstruction pre-empted by the timer service request.

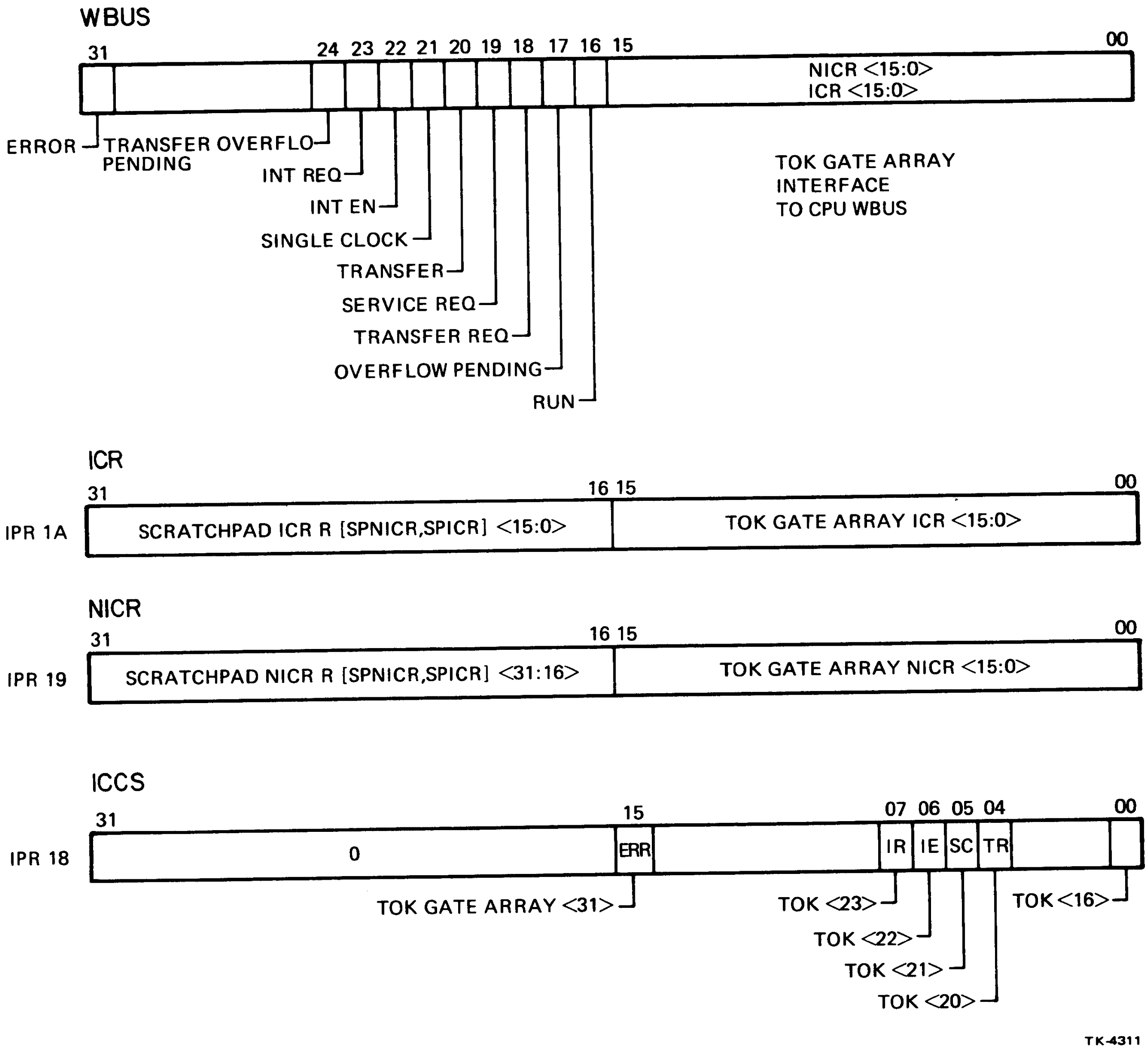


Figure 2-111 TOK Control, ICR, MCR, and ICCS Registers

Timer interrupt requests operate in a similiar fashion. At BUT Service, if any interrupts are pending, the INT gate array has already completed arbitration and it drives the microvector address lines <2:0> with the highest priority request encoded into a microaddress. The complete microaddress of the timer interrupt service routine is formulated by the SAC, MSQ, and INT gate arrays. The control store address of the first microinstruction of the timer interrupt service routine is 003B. The microcode transfers control of the macroprogram to the timer service routine pointed to by contents of SCBB+C0. This routine must clear the IR bit of the ICCS before using the timer again or an interval timer error occurs.

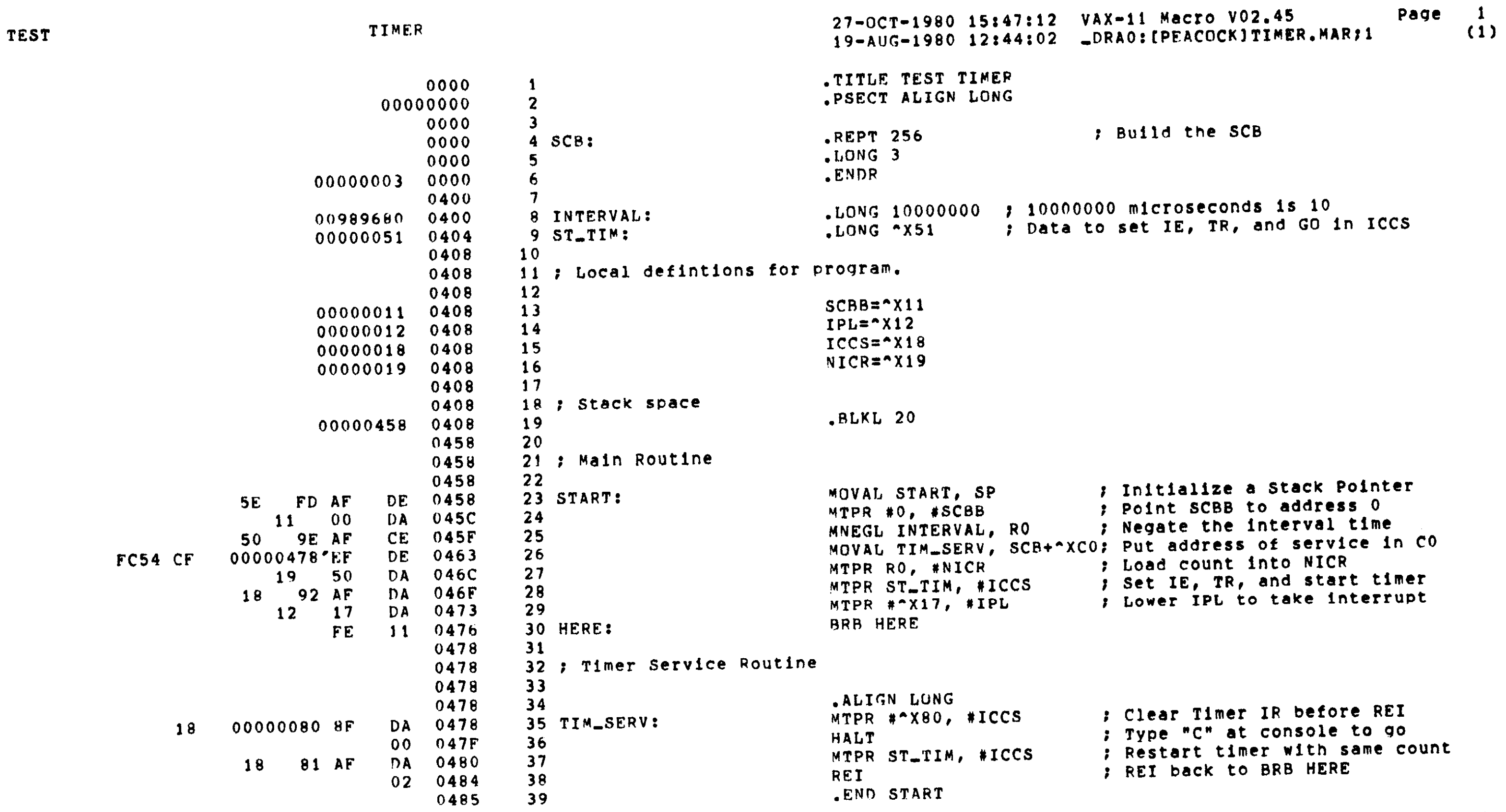
2.7.5 Timer Macrocoding Example

Figure 2-112 is an example of a macroprogram that activates the interval timer. The program shows the mechanism by which the timer establishes intervals of execution time for programs in a timesharing environment. This is a standalone program and could not operate under VMS. This routine sets up the interval timer with a 10-second interval. The timer is started and the CPU waits for the interrupt that occurs 10 seconds later when the counter overflows. When the counter overflows, the interrupt service routine is entered via SCB+C0, where it halts the CPU. If C is typed at the console, the program reloads the timer and waits for another 10 seconds until the counter overflows. This illustrates how to load the timer, start it, and handle the interrupt at vector SCBB+C0.

Lines 4, 5, and 6 are assembler directives that build the SCB in the low two pages of memory (0 to 3FC). The value associated with label INTERVAL is the test interval in μs. 10000000 μs is the same as ten seconds. The label ST_TIM has the value 51 (hex) associated with it. This is used to set bit <6> interrupt enable, bit <4> transfer NICR to ICR, and bit <0> the GO bit that starts the timer running. Lines 13 to 16 are local symbol definitions for internal processor registers. At line 19 is a directive to allocate 20 longwords for the stack space. Line 23 is the beginning of the main program. The first instruction sets up the stack pointer. The next instruction points the SCBB to address 0 in memory. At line 25 the interval value defined at line 8 is negated (2's complement) and put in R0. The address of the service routine (TIM_SERV) is moved into the SCB so that timer interrupt vectors to relative address 478. At line 27, the NICR is loaded with the 2's complement of the interval (10 seconds). The instruction at line 28 transfers the data pattern defined in line 9 to set IE, transfer the NICR to the ICR, and start the timer. The IPL of the machine is lowered to 17 to take the timer interrupt when the timer overflows. The next instruction waits for the interrupt.

When the interval timer overflows, the interrupt request at IPL 18 is generated. If it is granted, the macrocode resumes at the label called TIM_SERV. The interrupt service routine must clear bit <7> in the ICCS. Otherwise when the REI is executed, the IPL 18 interrupt request is immediately generated again. The HALT instruction prints out the PC at the end of 10 seconds. If the program is continued by typing C at the console, the timer is restarted with the same interval. Therefore the timer can be reloaded from the NICR continuously.

2.7.5.1 Time-of-Year (TOY) Clock Introduction – The time-of-year clock is loaded once with the binary time, and this clock is not disturbed as long as the system is operating. This clock is powered by NI-CAD batteries and is designed to operate for 100 hours without power being applied to the rest of the system. The circuit implementation uses CMOS logic elements which have very low power consumption characteristics. Refer to Paragraph 2.1.1.5 for a detailed explanation of the battery-charging circuit and battery interface. The time-of-year clock contains the time of year at all times and is used by the operating system for automatic system boot. The timer is loaded with binary time of year in 10-ms increments by operating system services through IPR 1B. IPR 1B is called the time-of-day register (TODR) and provides read/write access to the time-of-year clock. The clock circuitry is physically located on the UBI module in slot 4.



```
TEST                      TIMER                               27-OCT-1980 15:47:12  VAX-11 Macro V02.45          Page    1
                                                              19-AUG-1980 12:44:02  _DRA0:[PEACOCK]TIMER.MAR;1          (1)

                                 0000                                   .TITLE TEST TIMER
                             00000000                                   .PSECT ALIGN LONG
                                 0000
                                 0000   SCB:                            .REPT 256               ; Build the SCB
                                 0000                                   .LONG 3
                       00000003  0000                                   .ENDR
                                 0400
                       00989680  0400   INTERVAL:                       .LONG 10000000  ; 10000000 microseconds is 10
                       00000051  0404   ST_TIM:                         .LONG ^X51      ; Data to set IE, TR, and GO in ICCS
                                 0408
                                 0408   ; Local defintions for program.
                                 0408
                       00000011  0408                                   SCBB=^X11
                       00000012  0408                                   IPL=^X12
                       00000018  0408                                   ICCS=^X18
                       00000019  0408                                   NICR=^X19
                                 0408
                                 0408   ; Stack space
                       00000458  0408                                   .BLKL 20
                                 0458
                                 0458   ; Main Routine
                                 0458
         5E    FD AF     DE      0458   START:                          MOVAL START, SP         ; Initialize a Stack Pointer
            11    00     DA      045C                                   MTPR #0, #SCBB          ; Point SCBB to address 0
         50    9E AF     CE      045F                                   MNEGL INTERVAL, R0      ; Negate the interval time
FC54 CF  00000478'EF     DE      0463                                   MOVAL TIM_SERV, SCB+^XC0; Put address of service in C0
            19    50     DA      046C                                   MTPR R0, #NICR          ; Load count into NICR
         18    92 AF     DA      046F                                   MTPR ST_TIM, #ICCS      ; Set IE, TR, and start timer
            12    17     DA      0473                                   MTPR #^X17, #IPL        ; Lower IPL to take interrupt
                  FE     11      0476   HERE:                           BRB HERE
                                 0478
                                 0478   ; Timer Service Routine
                                 0478
                                 0478                                   .ALIGN LONG
      18  00000080 8F    DA      0478   TIM_SERV:                       MTPR #^X80, #ICCS       ; Clear Timer IR before REI
                         00      047F                                   HALT                    ; Type "C" at console to go
         18    81 AF     DA      0480                                   MTPR ST_TIM, #ICCS      ; Restart timer with same count
                         02      0484                                   REI                     ; REI back to BRB HERE
                                 0485                                   .END START
```

Figure 2-112   Macroprogram that Activates Interval Timer

2.7.5.2 Time-of-Year Clock Detailed Description – Figure 2-113 is a block diagram of the time-of-year circuitry. The time-of-year clock circuitry is powered from the four NI-CAD batteries attached to the frame of the cabinet. The time-of-year circuitry is accessed via the CPU WBus interface under control of a WCTRL micro-order. The WCTRL field is latched on the UBI module and all six bits of the WCTRL field go to decode logic that is enabled for only two WCTRL micro-orders.

WCTRL Field	Binary	Function
WCTRL/TODCLK_WB	D	TOY <22:16>_WBUS
WCTRL/TODCLK	9	WBUS_TOY <27:16>

The block marked Decode Logic produces the enabling signals to drive the WBus with the time-of-year clock data and also receives the WBus data in the control latch. The time-of-year clock in the CPU is not designed as a parallel loadable binary counter. Rather, the TOY scratchpad is loaded with binary time of year (the initial time). The binary counter is cleared and contains delta time. The TOY scratchpad is only four bits wide. When the MTPR macroinstruction is executed, it must reset the time-of-year binary counter, and it must also pack the initial time of year into the 16 × 4 RAM one nibble at a time. The writing of the TOY scratchpad is controlled by the WBus data during the MTPR. The output of the TOY scratchpad interfaces to the WBus in bit positions <27:24>. After the MTPR loads the TOY scratchpad with the time, the binary counter is incremented every 10 ms by the 1-KHz oscillator divided by ten. The actual time is obtained by reading the TOY scratchpad, which contains the initial correct time, and then reading the binary counter data, which is delta time. The initial time and the delta time are added together to form the binary time of year.

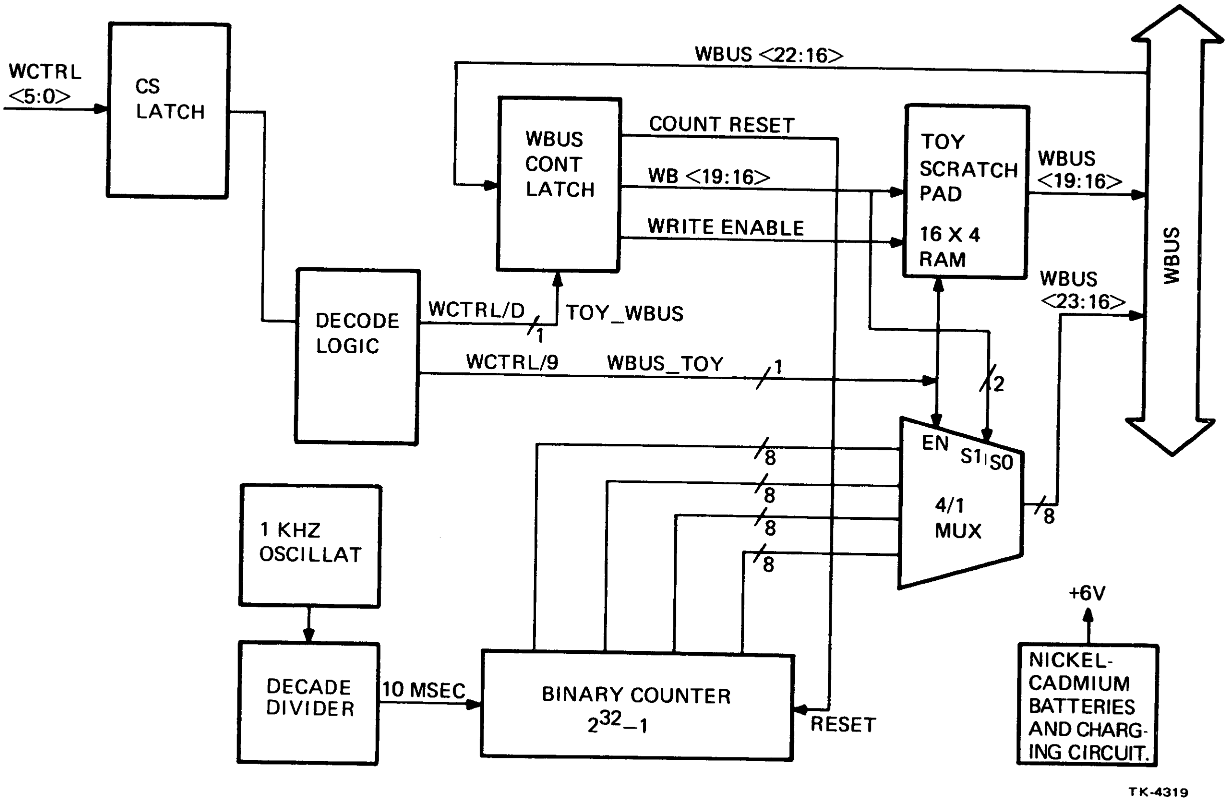


Figure 2-113 Time-of-Year Clock Block Diagram

When the macroprogram reads the time-of-year clock by executing an MFPR of IPR 1B, the MFPR microcode must unpack the TOY scratchpad time and store it. After the initial time is stored, the microcode must read the binary counter delta time one byte at time and unpack it as well. At this point the initial time is added to the delta time and stored in the destination specified by the MFPR.

Figure 2-114 shows the WBus interface to the time-of-year clock. The top part of the drawing shows the appearance of the logic when a write to the time-of-year clock is done. The functions of the bits are described below.

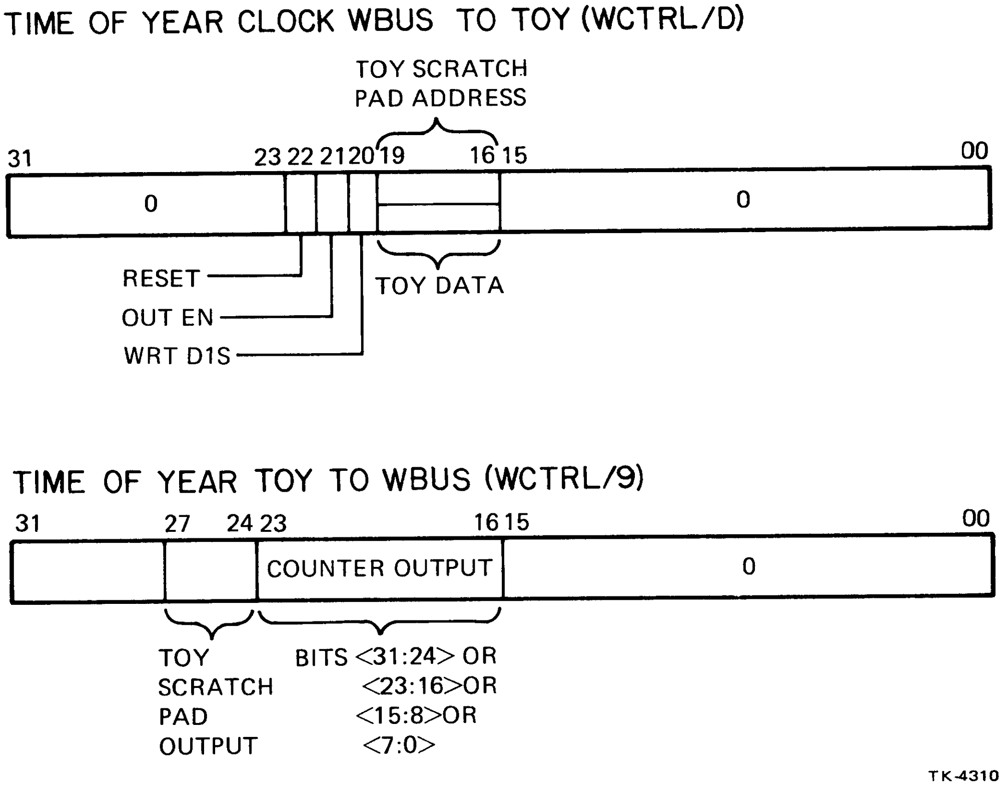


Figure 2-114 WBus Data for Time-of-Year Clock Write/Read

WBus Bit	Name	Function
<22>	RESET	Clears the binary counter, used in MTPR to TODR
<21>	OUTPUT EN	Enables the TOY scratchpad to drive the WBUS <27:24>, used in MFPR
<20>	WRITE DIS	Disables writes to the TOY scratchpad, used in the MFPR
<19:16>	TOY Scratchpad Address and Data	Scratchpad address and data

Reading the time-of-year clock yields WBus data that resembles the lower part of Figure 2-114.

WBus Bit	**Name**	**Function**
<27:24>	TOY Scratchpad Data	Initial time loaded by MTPR
<23:16>	Binary Counter	The output of the counter is passed in bit positions <23:16> according to the following table.

WBUS <23:16> = Binary Counter	<7:0>	If Control Latch <17:16> =	3
	<15:8>		2
	<23:16>		1
	<31:24>		0

The schematic diagram of the TOY circuitry is shown in the UBI module schematic on pages UBI 1 and UBI 2. On the left side of UBI 1 is the WCTRL field interface that is used to decode the two WCTRL micro-orders explained above. When the WCTRL specifies the TODCLK_WB, the control latch E39 is clocked with the WBus data. If the WCTRL micro-order is TODCLK, the tri-state drivers at the output of the TOY scratchpad E50 are enabled to drive the WBus. Simultaneously on UBI 2, if WCTRL specifies TODCLK, the 4/1 multiplexers are enabled to pass the selected portion of the binary counter to the WBus bits <23:16>. In the bottom left corner of UBI 2 is the 1-KHz oscillator that provides the time base for the clock. The output of the oscillator goes to the decade divider circuit E27. At the output of E27 is a symmetrical 100-Hz waveform that establishes the 10-ms increment rate for the binary counter stages E28, E52, and E14. The output of the counter goes to the 4/1 multiplexers E28, E51, E42, and E41 where it can be multiplexed onto the WBus by the MFPR instruction microcode.

The oscillator, decade divider, scratchpad, and counter are all powered from the battery so that when power fails, they keep functioning. The microcode tests the batteries during every MFPR from the TODR. If the batteries can no longer be charged, the microcode returns zero from the TODR and the software can do a macrobranch on that condition to have the system operator enter the time at system boot. If it is necessary to re-enter the time after booting twice, there is a strong possibility that the batteries can no longer hold a charge. This can be verified by examining the TODR (IPR 1B). If it contains zeros after being loaded with a non-zero value, there is a malfunction in the time-of-year power circuitry or the batteries are dead.

2.8 CONDITION CODE LOGIC

The condition code logic in the VAX-11/750 CPU performs the following three functions.

1. Sets or clears the PSL N, Z, V, and C bits according to the architectural definition of each macroinstruction and the result of the data path operation.
2. Determines whether or not conditional branch instructions are satisfied so the microcode can microbranch properly.
3. Initiates all arithmetic traps.

2.8.1 Condition Code Logic Description

Most of the logic circuitry to perform the three condition code functions is implemented within a gate array called CCC, located on the DPM module in slot 2. This gate array is controlled by a secondary encoding of the CC field and the WCTRL field of the microword called CC CTRL <3:0>. The PSW resides in the CCC gate array, while the copies of the CM bit <31> exist in PHB and CCC. PSL FPD bit <27> is contained in the PHB gate array, which is part of the microsequencer logic. The PSL IS bit <26>, CUR MOD <25:24>, PREV MOD <23:22>, and the IPL <20:16> all are part of the INT gate array located on UBI. When a CCPSL WB_PSL micro-order is issued, the entire PSL is sourced to the WBus on a read from all three gate arrays. Writing the PSL is also accomplished from the WBus, so all three gate arrays are enabled when the CCPSL function is PSL_WB. This discussion is limited to the PSW in the CCC gate array.

The CCC gate array is controlled by the CC and WCTRL fields of the microword, after they are reencoded by the CC control (E15) ROM on the DPM module (DPM 20). This ROM is not defined in the microcode listing. Figure 2-119 (see Paragraph 2.8.2) shows the ROM content for the various CC and WCTRL field functions. The vertical functionality of the microword is explained in Paragraph 2.2.1.2. The CCMISC field of the microword is true if any of the following combinations of the CC and WCTRL fields is desired by the microprogrammer.

CCMISC	**CC Binary**	**WCTRL Binary**
NOP.CCBR_BRATST	11	000111
NOP.CCBR_CSIGN	01	000110
WB_ATCR.CCBR_SIGND	00	000111
ALUS_DSDC.CCBR_ALUS	00	000110
ALUS_SIGND.CCBR_ALUS	11	000110
ALUS_UNSGN.CCBR_ALUS	10	000110
SETV.CCBR_SIGND	01	000111

The WCTRL field of the microword during the CCMISC is either 6 or 7. There is no WCTRL field definition for 6 or 7, which means that CCMISC micro-orders are unique operations. The CCPSL field of the microword is true if the microprogrammer specifies one of the following operations in the microinstruction.

CCPSL	**WCTRL BINARY**
WB_PSL.CCBR_SIGND	000100
CC_WB.CCBR_ALUS	000101
PSL_WB.CCBR_ALUS=0	000000
PSL_WB.CCBR_ALUS=1	000001
MDR_OSR.CCBR_BRATST	101111

The above field definitions are variations on the WCTRL micro-orders that are not defined as WCTRL functions. In both the CCMISC and CCPSL functions, the name of the definition has the CCBR microbranch bits defined also. The CCBR bits are two microbranch status bits that are defined in the microinstruction that specifies a BUT micro-order BUT/CCBR, BUT/CCBR.CCBR0.IR0, or BUT/CCBR0.SRKSTA0. The definition of CCBR <1:0> is contained in the CCPSL or CCMISC micro-order of the microword. (See Figure 2-115.) For example, the CCPSL micro-order WB_PSL.CCBR_SIGND indicates that the WBus gets the PSL from the INT, PHB, and CCC gate arrays. Additionally, the CCBR bits <1:0> assume their default values, which are as follows.

CCBR <1>		<0>	
0 =	WBus greater than or equal to 0.	0 =	WBus not equal to 0.
1 =	WBus less than 0.	1 =	WBus equal to 0.

These bits are useful for microbranching on the result of ALU operations or WBus data. The CCBR bits assume different functions depending on the CCMISC, CCPSL, or CC micro-order. An example of this is the CCMISC micro-order NOP.CCBR_BRATST. The CCBR bits take on a new function.

CCBR <1>	<0>	
0	0 =	Conditional branch not satisfied.
	1 =	Conditional branch condition is true.

This micro-order is specified in the microcode that executes the VAX-11 macroconditional branch instructions. It decodes the opcode of the branch instruction and compares the PSL N, Z, V, and C bits to the branch condition. For example, a BNEQ macroinstruction would assert CCBR <0> if the PSL Z bit was clear during the execution. Figure 2-115 is reproduced from the microcode listing. It defines the CCBR bits <1:0> for each of the CCMISC, CCPSL and CC micro-orders. The CCBR bits <1:0> are generated in the CCC gate array under control the redefined CC and WCTRL fields.

The CC field of the microword also can affect the CCBR bits <1:0> as shown in the chart. The CC field also has the two fields that set the PSL condition codes according to the architectural requirements and data path operation results. The CC field is defined as follows.

```
CC/=<32:31>,.DEFAULT=0
                NOP.CCBR_SIGND=0,
                NOP.CCBR_ALUS=3
                CCOP1.CCBR_SIGND=1,
                CCOP2.CCBR_SIGND=2,
```

The first two micro-orders are NOPs as far as the PSL condition codes are concerned, but they do affect the CCBR bits. The microprogrammer can use either of the NOP micro-orders with a BUT/CCBR micro-order to microbranch on the default signs explained above, or the ALU STATE bits <1:0> that are part of the ALU. The CCOP1 and CCOP2 micro-orders are used to set the PSL condition codes. The CCOP1 micro-order is used for about half of the macroinstruction set to set the condition codes. The CCOP2 micro-order is used to set the condition codes for the remainder of the macroinstruction set. Figures 2-116 to 2-118 include charts reproduced from the microcode listing to show which CC micro-order must be specified for a particular instruction in the far right column. The four columns across the page describe how each PSL condition code bit is affected when the CCOP1 or CCOP2 micro-order is specified.

```
        .TOC "  Micro Level Charts            : BUT/CCBR Chart"


;           +--------------------------+----------------------------------+---------------------+
;           |                          |                                  |     CCBR CONTROL    |
;           |       MICRO ORDER        |            OPERATION             +----------+----------+
;           |                          |                                  | CCBR<1>  | CCBR<0>  |
;  +--------+--------------------------+----------------------------------+----------+----------+
;  |        | NOP.CCBR<-SIGND          |                                  | LSS 0    | EQL 0    |
;  |   C    | NOP.CCBR<-ALUS           |                                  | ALUS<1>  | ALUS<0>  |
;  |   C    | CCOP1.CCBR<-SIGND        | CC OP 1                          | LSS 0    | EQL 0    |
;  |        | CCOP2.CCBR<-SIGND        | CC OP 2                          | LSS 0    | EQL 0    |
;  +--------+--------------------------+----------------------------------+----------+----------+
;  |   C    | NOP.CCBR<-BRATST         |                                  | 0        | BRA TST  |
;  |   C    | NOP.CCBR<-CSIGNS         |                                  | ALUS<1>  | LSS 0    |
;  |   M    | WB<-ATCR.CCBR<-SIGND     | WB<3:0> <-ATCR                   | LSS 0    | EQL 0    |
;  |   I    | ALUS<-DSDZ.CCBR<-ALUS    | ALUS<1:0> <- (BCD SIGN)'(BCD 0)  | ALUS<1>  | ALUS<0>  |
;  |   S    | ALUS<-SIGND.CCBR<-ALUS   | ALUS<1:0> <- (LSS 0)'(EQL 0)     | ALUS<1>  | ALUS<0>  |
;  |   C    | ALUS<-UNSGN.CCBR<-ALUS   | ALUS<1:0> <- (LSSU 0)'(EQL 0)    | ALUS<1>  | ALUS<0>  |
;  |        | SETV.CCBR<-SIGND         | PSL<V> <- 1                      | LSS 0    | EQL 0    |
;  +--------+--------------------------+----------------------------------+----------+----------+
;  |   C    | WB<-PSL.CCBR<-SIGND      | WB<31:0> <-PSL                   | LSS 0    | EQL 0    |
;  |   C    | CC<-WB.CCBR<-ALUS        | CC<-WB<3:0>                      | ALUS<1>  | ALUS<0>  |
;  |   P    | PSL<-WB.CCBR<-ALUS       | PSL<-WB<31:0>  ***               | ALUS<1>  | ALUS<0>  |
;  |   S    | PSW<-WB.CCBR<-ALUS       | PSW<-WB<15:0>  ***               | ALUS<1>  | ALUS<0>  |
;  |   L    | MDR_OSR.CCBR_BRATST      | MDR <- ZEXT OSR                  | 0        | BRA TST  |
;  +--------+--------------------------+----------------------------------+----------+----------+
;
;     WHENEVER THE OPERATION ALTERS ALUS, CCBR <- ALUS OLD
;
;     UNLESS OTHERWISE NOTED ALL VALUES ARE SIZE DEPENDENT
;
;         LSS 0     : WBUS<SIGN>.XOR.ALU 'OVERFLOW' (THIS WILL PRODUCE 'LSS 0' FOR A-B OR A+(-B))
;
;         OVERFLOW : XOR OF CIN AND COUT OF MSB(SIZE) ALU
;
;         EQL 0     : WMUX = 0
;
;         LSSU 0    : .NOT.(ALU CARRY) (THIS WILL PRODUCE 'LSSU 0' FOR A-B OR A+(-B))
;
;         BCD 0     : (WMUX<31:8>.EQ.0).AND.(WBUS<7:4>.EQ.0)
;
;         BCD SIGN : IF WBUS<3:0> > 9 THEN, ALUS<1> IS SET IF SIGN IS NEGATIVE
;
;         BRA TST   : FOR BRANCH INSTRUCTIONS CCBR<0> IS SET IF THE BRANCH CONDITION IS TRUE
;
;         *** IF V GFTS SET NO TRAP WILL OCCUR
```

Figure 2-115   BUT/CCBR Chart

```
        .TOC "  Micro Level Charts                 : Compatability Mode Condition Codes"

        ; CCOPS are not valid During any micro cycle that follows an IRD1. Because of this, they cannot be used in any of the micro
        ; instructions that are pointed to by the IRD1 entries in the Compatibility mode rom.

        ; NOTE 1 : 'SIGN', 'WX', 'OV', 'CRY', ARE ALL FUNCTIONS OF DSIZE
        ; NOTE 2 : (SIGN.XOR.OV).OR.CRY

        ; +---------------+--------+-------------------+--------------+-----------------+-------+
        ; | INSTRUCTION   |   N    |         Z         |      V       |        C        | CCOPx |
        ; +---------------+--------+-------------------+--------------+-----------------+-------+
        ; | ADC(B)        | SIGN   | WX.EQ.0           | OV           | CRY             |   1   |
        ; | ADD           | SIGN   | WX.EQ.0           | OV           | CRY             |   1   |
        ; | ASH           | SIGN   | WX.EQ.0           | 0            | 0               |   1   |
        ; | ASHC          | SIGN   | WX.EQ.0           | 0            | 0               |   1   |
        ; |               | SIGN   | (WX.EQ.0).AND.Z   | 0            | C               |   2   |
        ; | ASL           | SIGN   | WX.EQ.0           | N.XOR.C(IN)  | WB<31:16>.NE.0  |   2   |
        ; | ASLB          | SIGN   | WX.EQ.0           | N.XOR.C(IN)  | WB<31:8>.NE.0   |   2   |
        ; | ASR(B)        | SIGN   | WX.EQ.0           | N.XOR.C(IN)  | WB<31>          |   1   |
        ; | BI(T,S,C)(B)  | SIGN   | WX.EQ.0           | 0            | C               |   1   |
        ; | CLR(B)        | SIGN   | WX.EQ.0           | 0            | 0               |   1   |
        ; | CMP(B)        | SIGN   | WX.EQ.0           | OV           | .NOT.CRY        |   2   |
        ; | COM(B)        | SIGN   | WX.EQ.0           | OV           | .NOT.CRY        |   2   |
        ; | DEC(B)        | SIGN   | WX.EQ.0           | OV           | C               |   1   |
        ; | DIV           | SIGN   | WX.EQ.0           | OV           | CRY             |   1   |
        ; | INC(B)        | SIGN   | WX.EQ.0           | OV           | C               |   1   |
        ; | MFP(I,D)      | SIGN   | WX.EQ.0           | 0            | C               |   1   |
        ; | MTP(I,D)      | SIGN   | WX.EQ.0           | 0            | C               |   1   |
        ; | MOV(B)        | SIGN   | WX.EQ.0           | 0            | C               |   1   |
        ; | MUL           | SIGN   | WX.EQ.0           | 0            | 0               |   1   |
        ; |               | NOTE 2 | (WX.EQ.0).AND.Z   | 0            | WB<L>.NE.0      |   2   |
        ; | NEG(B)        | SIGN   | WX.EQ.0           | OV           | .NOT.CRY        |   2   |
        ; | ROL           | SIGN   | WX.EQ.0           | N.XOR.C(IN)  | WB<31:16>.NE.0  |   2   |
        ; | ROLB          | SIGN   | WX.EQ.0           | N.XOR.C(IN)  | WB<31:8>.NE.0   |   2   |
        ; | ROR(B)        | SIGN   | WX.EQ.0           | N.XOR.C(IN)  | WB<31>          |   1   |
        ; | SBC(B)        | SIGN   | WX.EQ.0           | OV           | .NOT.CRY        |   2   |
        ; | SUB           | SIGN   | WX.EQ.0           | OV           | .NOT.CRY        |   2   |
        ; | SWAB          | SIGN   | WX.EQ.0           | 0            | 0               |   1   |
        ; | SXT           | SIGN   | WX.EQ.0           | 0            | C               |   1   |
        ; | TST(B)        | SIGN   | WX.EQ.0           | 0            | 0               |   1   |
        ; | XOR           | SIGN   | WX.EQ.0           | 0            | C               |   1   |
        ; +---------------+--------+-------------------+--------------+-----------------+-------+
```

Figure 2-116 Compatibility Mode Condition Codes

```
;   CPTD.MCR              MICRO2  1H(17)     4-NOV-80  08:46:25    CLOKX Rev @@@@@, Clock rate = ???ns           Page    26
; CHARTS.MIC              Micro Level Charts          : Native Mode Condition Codes Part 1

;1266   .TOC "  Micro Level Charts            : Native Mode Condition Codes Part 1"
;1267
;1268
;1269   ; CCOPS are not valid During any micro cycle that follows an IRD1. Because of this, they cannot be used in any of the micro
;1270   ; instructions that are pointed to by the IRD1 rom.
;1271   ; NOTE 1 : 'SIGN', 'WX', 'OV', 'CRY', ARE ALL FUNCTIONS OF DSIZE
;1272   ; NOTE 2 : WB<15> + [(WX<15:0>.EQ.0).AND.CRY]
;1273
;1274     ; +-------------------+-------------+---------------------+--------------+------------+--------+
;1275     ; | INSTRUCTION       |      N      |          Z          |      V       |     C      | CCOPx  |
;1276     ; +-------------------+-------------+---------------------+--------------+------------+--------+
;1277     ; | ACB(B,W)          | SIGN        | WX.EQ.0             | OV           | C          |   1    |
;1278     ; | ACBL              | SIGN        | WX.EQ.0             | OV           | C          |   2    |
;1279     ; | ACB(F,D)          | WB<15>      | WX.EQ.0             | 0            | C          |   2    |
;1280     ; | ADAWI             | SIGN        | WX.EQ.0             | OV           | CRY        |   2    |
;1281     ; | ADD(B,W,L)(2,3)   | SIGN        | WX.EQ.0             | OV           | CRY        |   1    |
;1282     ; | ADD(F,D)(2,3)     | WB<15>      | WX.EQ.0             | 0            | 0          |   1    |
;1283     ; | ADWC              | SIGN        | WX.EQ.0             | OV           | CRY        |   1    |
;1284     ; | AOB(LEQ,LSS)      | SIGN        | WX.EQ.0             | OV           | C          |   2    |
;1285     ; | ASHL              | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1286     ; | ASHQ              | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1287     ; |                   | SIGN        | (WX.EQ.0).AND.Z     | 0            | C          |   2    |
;1288     ; | BIC(B,W,L)(2,3)   | SIGN        | WX.EQ.0             | 0            | C          |   2    |
;1289     ; | BIS(B,W,L)(2,3)   | SIGN        | WX.EQ.0             | 0            | C          |   2    |
;1290     ; | BIT(B,W,L)        | SIGN        | WX.EQ.0             | 0            | C          |   2    |
;1291     ; | CASE(B.W.L)       | SIGN.XOR.OV | WX.EQ.0             | 0            | .NOT.CRY   |   1    |
;1292     ; | CLR(B,W,L)        | SIGN        | WX.EQ.0             | 0            | C          |   2    |
;1293     ; | CLRD              | SIGN        | WX.EQ.0             | 0            | C          |   1    |
;1294     ; | CLRF              | SIGN        | WX.EQ.0             | 0            | C          |   2    |
;1295     ; | CLRQ              | SIGN        | WX.EQ.0             | OV           | C          |   1    |
;1296     ; | CMP(B,W,L)        | SIGN.XOR.OV | WX.EQ.0             | 0            | .NOT.CRY   |   1    |
;1297     ; | CMP(V,ZV)         | SIGN.XOR.OV | WX.EQ.0             | 0            | .NOT.CRY   |   1    |
;1298     ; | CMPC(3,5)         | SIGN.XOR.OV | WX.EQ.0             | 0            | .NOT.CRY   |   1    |
;1299     ; | CMPD              | SEE NOTE 2  | WX.EQ.0             | 0            | 0          |   2    |
;1300     ; |                   | .NOT.CRY    | WX.EQ.0             | 0            | 0          |   1    |
;1301     ; | CMPF              | WB<15>      | WX.EQ.0             | 0            | 0          |   1    |
;1302     ; |                   | SEE NOTE 2  | WX.EQ.0             | 0            | 0          |   2    |
;1303     ; | CRC               | SIGN        | WX.EQ.0             | 0            | C          |   2    |
;1304     ; | CVT(BW,BL)        | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1305     ; | CVT(FB,DB,FW,DW)  | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1306     ; |                   | N           | Z                   | WX<L>.NE.0   | C          |   2    |
;1307     ; | CVT(FD,DF,BF,BD,  | WB<15>      | WX.EQ.0             | 0            | 0          |   1    |
;1308     ; |      WF,WD,LD,LF) |             |                     |              |            |        |
;1309     ; | CVT(FL,DL,        | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1310     ; |      RFL,RDL)     |             |                     |              |            |        |
;1311     ; | CVT(LP,PL)        | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1312     ; | CVT(WB,LB,LW)     | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1313     ; |                   | N           | Z                   | WX<L>.NE.0   | C          |   2    |
;1314     ; | CVTWL             | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1315     ; | DEC(B,W,L)        | SIGN        | WX.EQ.0             | OV           | .NOT.CRY   |   1    |
;1316     ; | DIV(B,W,L)(2,3)   | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1317     ; | DIV(F,D)(2,3)     | WB<15>      | WX.EQ.0             | 0            | 0          |   1    |
;1318     ; | EDIV              | SIGN        | WX.EQ.0             | 0            | 0          |   1    |
;1319     ; | EMOD(F,D)         | WB<15>      | WX.EQ.0             | 0            | 0          |   1    |
;1320     ; +-------------------+-------------+---------------------+--------------+------------+--------+
```

Figure 2-117   Native Mode Condition Codes Part 1

```
.TOC "  Micro Level Charts                   : Native Mode Condition Codes Part 2"

; 'SIGN', 'WX', 'OV', 'CRY', ARE ALL FUNCTIONS OF DSIZE

; +------------------+-------------+-----------------+------------+-----------+-------+
; | INSTRUCTION      |     N       |        Z        |     V      |     C     | CCOPx |
; +------------------+-------------+-----------------+------------+-----------+-------+
; | EMUL             | SIGN        | WX.EQ.0         | 0          | 0         |   1   |
; |                  | SIGN        | (WX.EQ.0).AND.Z | 0          | C         |   2   |
; | EXT(V,ZV)        | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | FF(S,C)          | 0           | WX.EQ.0         | 0          | 0         |   1   |
; | INC(B,W,L)       | SIGN        | WX.EQ.0         | OV         | CRY       |   1   |
; | INDEX            | SIGN        | WX.EQ.0         | 0          | 0         |   2   |
; | INSQUE           | SIGN.XOR.OV | WX.EQ.0         | 0          | .NOT.CRY  |   1   |
; | INSV             | SIGN        | WX.EQ.0         | OV         | C         |   2   |
; | LOCC             | 0           | WX.EQ.0         | 0          | 0         |   1   |
; | M(T,F)PR         | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | MATCH            | 0           | WX.EQ.0         | 0          | 0         |   1   |
; | MCOM(B,W,L)      | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | MNEG(B,W,L)      | SIGN        | WX.EQ.0         | OV         | .NOT.CRY  |   1   |
; | MNEG(F,D)        | WB<15>      | WX.EQ.0         | 0          | 0         |   1   |
; | MOV(B,W,L)       | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | MOV(F,D)         | WB<15>      | WX.EQ.0         | 0          | C         |   2   |
; | MOVA(B,W,L)      | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | MOVAQ            | SIGN        | WX.EQ.0         | 0          | C         |   1   |
; | MOVC(3,5)        | 0           | WX.EQ.0         | 0          | 0         |   2   |
; |                  | SIGN.XOR.OV | WX.EQ.0         | 0          | .NOT.CRY  |   1   |
; | MOVQ             | SIGN        | WX.EQ.0         | OV         | C         |   1   |
; |                  | SIGN        | (WX.EQ.0).AND.Z | 0          | C         |   2   |
; | MOVTC            | SIGN.XOR.OV | WX.EQ.0         | 0          | .NOT.CRY  |   1   |
; | MOVTUC           | SIGN.XOR.OV | WX.EQ.0         | 0          | .NOT.CRY  |   1   |
; | MOVZ(BW,BL)      | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | MOVZWL           | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | MUL(B,W,L)(2,3)  | SIGN        | WX.EQ.0         | 0          | 0         |   1   |
; |                  | N           | Z               | WX<L>.NE.0 | C         |   2   |
; | MUL(F,D)(2,3)    | WB<15>      | WX.EQ.0         | 0          | 0         |   1   |
; | POLY(F,D)        | WB<15>      | WX.EQ.0         | 0          | 0         |   1   |
; | PROBE(R,W)       | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | PUSHA(B,W,L)     | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | PUSHAQ           | SIGN        | WX.EQ.0         | 0          | C         |   1   |
; | PUSHL            | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | REMQUE           | SIGN.XOR.OV | WX.EQ.0         | 0          | .NOT.CRY  |   1   |
; |                  | N           | Z               | WX<L>.EQ.0 | C         |   2   |
; | ROTL             | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; | S(C,P)ANC        | 0           | WX.EQ.0         | 0          | 0         |   2   |
; | SBWC             | SIGN        | WX.EQ.0         | OV         | .NOT.CRY  |   1   |
; | SKPC             | 0           | WX.FQ.0         | 0          | 0         |   1   |
; | SOB(GEQ,GTR)     | SIGN        | WX.EQ.0         | OV         | C         |   2   |
; | SUB(B,W,L)(2,3)  | SIGN        | WX.EQ.0         | OV         | .NOT.CRY  |   2   |
; | SUB(F,D)(2,3)    | WB<15>      | WX.EQ.0         | 0          | 0         |   1   |
; | TST(B,W,L)       | SIGN        | WX.EQ.0         | 0          | 0         |   1   |
; | TST(F,D)         | WB<15>      | WX.FQ.0         | 0          | 0         |   1   |
; | XOR(B,W,L)(2,3)  | SIGN        | WX.EQ.0         | 0          | C         |   2   |
; +------------------+-------------+-----------------+------------+-----------+-------+
```

Figure 2-118   Native Mode Condition Codes Part 2

The following discussion traces the microcode executed for a VAX-11 macroinstruction to illustrate how the condition codes are set. It begins with a review of the operation of the D-size ROM and how to read the microcode macro expansion. The D-size ROM is blasted by the microprogrammer that wrote the microcode for the macroinstruction being executed. The VAX-11 macroinstruction that is traced here is:

```
        ADDL2       R0, R1              ;Where R0 is 7FFFFFFF and R1
                                        ;is equal to 00000001
```

This is an integer add type instruction. The microcode for this macroinstruction is found in the INTLOG.MIC file of the microcode listing. The D-size ROM macros are typically the last section of one of these files. Locate the D-size ROM macro for the ADDL2 instruction. The hex opcode for an ADDL2 is C0. The D-size ROM macro should appear as below.

```
0D0:      SIZE  [LONG]  [LONG]  [ 0]  [ 0]  [ 0]  [ 0]  ;ADDL2
```

Read this macro from the left column. The number 0D0 is address input to the D-size ROM. The IRD counter output also addresses the D-size ROM, so that for one opcode, there are six locations in the ROM. There are six locations because the VAX-11 macroinstructions can have up to six operand specifiers that must program the size of the data path during each execution phase. In the ADDL2 macroinstruction, there are only 2 operands, so the D-size ROM must be blasted with data size for first and second operand specifier evaluations. The size of the data path for each operand specifier evalution is contained within the brackets. The first operand specifier evaluation is in the next column. The data size for each of the six operand specifier evaluations from 1 to 6 is read from left to right. Instructions that have less than 6 operands contain 0 in unused locations. The ADDL2 instruction contains the size [LONG] in the first and second operand specifier evaluations. The DEFIN.MIC file for the D-size ROM definition indicates that the data size definitions are as follows.

```
    IF D-Size = [BYTE] Then D-Size <1:0> = 0
    IF D-Size = [WORD] Then D-Size <1:0> = 1
    IF D-Size = [LONG] Then D-Size <1:0> = 2
    IF D-Size = [QUAD] Then D-Size <1:0> = 3
```

The D-size ROM would be blasted with a 2-bit binary size code for every execution phase of the macroinstruction. The D-size ROM output is used only if the D-type field of the microword specifies IDEP (data size is instruction-dependent). The D-size bits <1:0> go to the CCC gate array, so the PSL condition codes are set according to the data size of the macroinstruction.

To trace the ADDL2 macroinstruction through the microcode, refer to the IRD 1 and IRDx ROM macros located at the end of the INTLOG.MIC file. The IRD 1 and IRDx ROM macros appear as below.

```
.ICODE:                 OPS                         OPS
        0C0: FPD        [NOP][IE.OPCOD.DEC]         [NOP][IE.OPCOD.DEC]
        IRD1            [LOD][OS.RED]               [LOD][OS.RED]
```

This is the IRD 1 ROM macro definition for ADDL2. The IRD 1 ROM is addressed by the opcode of the instruction to be executed and the FPD and the signal FPA PRESENT.

The macroinstruction opcode provides the base target address in the ROM of which there are four locations. This macro allows the microprogrammer to blast all four locations with the address in control store of the microroutine to evaluate the first operand specifier. The FPD bit should not be set at IRD 1 of an ADDL2 instruction because it is not interruptable. If it is set, the machine will vector to location

SCBB+10 and execute the reserved to DEC opcode instruction fault service routine. FPA PRESENT is a signal used to change the flow depending on whether an FPA is present. The IRD 1 ROM macro has 2 targets across the page: one with FPA and one without FPA. The OPS bit is used to load the OSR at IRD 1 and IRDx. The IRD 1 ROM macro could be changed to show how the ROM is addressed as follows.

```
0D0:        FPD          NOT FPA        FPA
        NOT FPD          NOT FPA        FPA
```

This shows that at base IRD 1 ROM address 0D0, the four locations that are blasted are all the possible combinations of FPD and FPA PRESENT. The contents of the brackets is the label of a microroutine that is entered for each of the four possible combinations. In the example, an ADDL2 does not use the FPA; FPD should be clear; and both the source and destination operands are in registers. This discussion assumes that the FPA is not present, even if the FPA was installed in the CPU; the operand specifier routine address is the same; [OS.RED]. PSL FPD is false; and REG MODE is true for both the source and destination operands. This means the microcode will microbranch on the addressing mode and enter the OS.RED flows at the microinstruction that fetches the source operand from a register.

The IRDx ROM macro is similiar to the IRD 1 macro except that the IRD COUNTER output addresses these ROMs.

```
.OCODE              OPS      REG                        MEM
    0C0:    CNT0 [LOD][IL.ADD2.B.W.L.REG]      [OS.MOD]
            CNT1 [NOP][IL.ADD2.B.W.L.MEM]      [IL.ADD2.B.W.L.MEM]
```

The combinations of REG MODE and FPA PRESENT are used as address input to the IRDx ROM along with the IRD counter output. There are eight possible targets at IRDx (CNT0 has four combinations and so does CN 1). CNT0 address is used at the first IRDx, and the CNT1 address is used at the second IRDx. Since this is register mode for both the source and destination, the control store address at CNT0 is [IL.ADD2.B.W.L.REG] and the CNT1 control store address is [IL.ADD2.B.W.L.MEM]. In register mode the CNT1 address is meaningless. If the destination were not a register, the MEM flows would have been followed and the microcode would have gone to the following control store addresses.

```
[OS.MOD]                           VA_GPR
[IL.ADD2.B.W.L.MEM]                WRITE MEMORY AT VA
```

To summarize the flow of the ADDL2 R0, R1, the microcode goes to the following two ROM addresses.

```
IRD 1                              [OS.RED]
IRDx                               [IL.ADD2.B.W.L.REG]
```

The following discussion traces the microinstructions. They are reproduced below from the OSR.MIC and INTLOG.MIC files respectively.

```
100:
OS.RED:
    ;0000- - - - - - - - - - - - - - - - - - - - ; Rn          REGISTER MODE
    FPA_Q_M[MDR] MDR_R[GPR.R],   ; PLACE OP (GPR(RNUM)) IN MDR
    CLOBBER MTEMP0 DEF, IRDX [1]  ; SAVE MDR IN Q
```

This moves the source operand from R0 into the MDR and Q gets the old MDR data. The IRDx address is [IL.ADD2.B.W.L.REG] and at this IRDx, the next control store address is [IL.ADD2.B.W.L.REG]. This is the microinstruction stored at IL.ADD2.B.W.L.REG.

```
IL.ADD2.B.W.L.REG
     ;-------------------------; 80 A0 C0
     R[GPR.R].SIZ_M[MDR]+RB,CCOP1,    ;
     SIZE [IDEP], IRD1                ;
```

This microinstruction specifies that the GPR pointed to by the RNUM latch <R1> is the destination. The MDR <R0> is added to the destination GPR <R1>, which is selected by RNUM, and GPR <R1> is modified. The PSL condition codes are set with the CCOP1 micro-order. The condition codes are set according to the D-size which is specified with the SIZE [] macro. The SIZE being equal to IDEP means the D-size ROM specifies the data size, and the D-size ROM macro explained above indicates the data size of the source operand is [LONG] and the data size of the destination is also [LONG]. The result of adding 7FFFFFFF and 00000001 is 80000000. This is an integer overflow. As a result the PSL N, Z, V, and C bits should be set as follows for an ADDL2.

PSL N	Z	V	C
ALU <31>	WX <31:0> = 0	ALU <31> V	ALU <31> CO
1	0	1	0

2.8.2 Branch Instruction Implementation

The CCC gate array is used to decide whether a macrobranch instruction is satisfied. If the branch condition is not satisfied, the hardware must bump the PC to the next sequential instruction and do the IRD 1. If the branch condition is satisfied, the sign-extended displacement is added to the PC. Writing the PC flushes the XB and initiates prefetch for the new instruction stream data. This discussion traces a VAX-11 macrobranch instruction called BNEQ. This macroinstruction branches if the PSL Z bit is clear. The BNEQ instruction is located in the CONTRL.MIC file. The IRD 1 ROM macro for a BNEQ in the back of the CONTRL.MIC file appears below.

```
.ICODE                   OPS        REG
      012:               FPD [NOP][IE.OPCOD.DEC]
                         IRD1[LOD][CO.BRCND]


                         OPS        FPA REG
                         [NOP][IE.OPCOD.DEC]
                         [LOD][CO.BRCND]



.OCODE
      012:               CNT0[NOP][IE.BAD.IRD]
                         CNT1[NOP][IE.BAD.IRD]


                         [NOP][IE.BAD.IRD]
                         [NOP][IE.BAD.IRD]
```

The IRD 1 macro specifies that the address of the BNEQ microcode is CO.BRCND, which is the target address for all the conditional branch instructions. This instruction will not do an IRDx. The address for a fault is [IE.BAD.IRD], which initiates a machine check exception. The microcode sequence for the BNEQ is shown below.

```
=1000
CO.BRCND:
    ;1111- - - - - - - - - - - - - - - - - - - - -;
    MDR_ZEXT(OSR) BRATST?,          ; GET DISPLACEMENT FROM OSR
                                    ; TEST FOR BRANCH
    NEXT/CO.BRCND-DECIDE            ; GO TO DECISION BLOCK
```

This microinstruction moves the branch displacement from the OSR to the MDR, zero-extending from bit <8> to bit <31> in the MDR. In the same macro, the BRATST? implies that the BUT micro-order is BUT/CCBR and the CCPSL micro-order is CCPSL/MDR_OSR.CCBR_BRATST. This can be verified by locating this macro in the MACRO.MIC file. This microinstruction has two possible destinations. If the PSL Z bit is set, the microcode reads the microinstruction at CO.BRCND-DECIDE. If the PSL Z bit is clear, the microcode executes the microinstruction at CO.BRCND-DECIDE+1.

If the PSL Z bit is set, the branch condition is not satisfied and the next microinstruction is shown below.

```
=0
CO.NOP:
CO.BRCND-DECIDE:
    ;- - - - - - - - - - - - - - - - - - - - - - - ; NO BRANCH IF CONTROL COMES
    IRD1                                          ; HERE,GO DO NEXT INSTRUCTION
```

This is an instruction to do IRD 1 and execute the next sequential instruction. If the PSL Z bit is clear, the CCBR bits <1:0> are equal 01, according to the CCPSL micro-order at location CO.BRCND. The following microinstructions are executed.

```
CO.BRCND-BRANCH:                           ; BRANCH IF CONTROL COMES
    ;1- - - - - - - - - - - - - - - - - - - - - - ; HERE, CALCULATE NEW PC
    PC_PC+SEXT(M[MDR]),                    ; WASTE CYCLE TO LET PC CATCH
    SIZE [IDEP], NEXT/CO.NOP               ; UP
```

The PC gets the sign-extended MDR if the branch condition is satisfied. Writing a new value in the PC causes the XB to be invalidated, and prefetch for the new I-Stream begins, If the XB is not full at IRD 1, the micromachine is stalled until the XB is filled. The next microinstruction is at CO.NOP, as shown above.

The third function of the CCC gate array is to generate the signals that cause an arithmetic trap at the BUT Service following an arithmetic operation, The PSW bits <7:5> are the trap enable bits that must be set by a macroroutine. The functions of these bits are described below.

PSW <7>	Decimal Overflow Trap Enable.
PSW <6>	Floating Underflow Trap Enable.
PSW <5>	Integer Overflow Trap Enable.

If an arithmetic operation causes one of the trap conditions, the CCC gate array asserts the signal ARITH TRAP L. At the next BUT Service, the arithmetic trap is arbitrated with console halt, interrupt pending, etc. and the trap flows are entered. The type of arithmetic trap is logged into the arithmetic trap code register (ATCR) contained in the CCC gate array. The arithmetic trap results in aborting the next macroinstruction and performing the trap service from SCBB+30. The trap microcode pushes the PSL, PC of the NEXT instruction, and the ATCR on the stack.

2.8.3 Hardware Implementation of Condition Code Logic

The condition code logic is on the DPM module print set. Refer to DPM 20. The CCC gate array is controlled by 4-bit field called CC CTRL <3:0>. This field comes from the output of ROM E15 on DPM 20. The address input to this ROM is the CC and WCTRL fields of the microword that is latched on DPM 20 and DPM 12. The output is called CC CTRL <3:0> H. These four signals go to the CCC gate array shown on DPM 10. Figure 2-119 shows how the CC CTRL lines and the "good samaritan" ROM are programmed for various combinations of the WCTRL and CC fields. The signal LIT 0 H is present because if the LIT field is 1 or 3, the CC field is not interpreted and becomes part of the short or long literal. Lines CC CTRL <3:0> on DPM 10 are the control input to the gate array. The VAX-11 or compatibilty macroinstruction opcode is latched in E13 and is the input to combinational logic that sets the PSL condition codes according to the architectural definitions and data path results. The D-size bits <1:0> enter the CCC gate array and are used to select the correct data path sign, C bit, and V bit. The sign can be either WBUS <31>, WBUS <15>, or WBUS <7> depending on the D-size bits <1:0>. The same is true of the sources of the C bit and V bit. The C and V bits are also selected as a function of data size. FPA Z and V are interfaced to CCC so that FPA divide by zeros and overflow can force the appropriate arithmetic trap condition. CCC generates the trap for FPA instruction traps also. The bidirectional interface to the WBus connects the PSW (−TP) to the rest of PSL when the CCPSL micro-order specifies WB_PSL. Writing the PSW from the WBus is accomplished with the PSL_WB micro-order. The PSL C bit goes to the BUT multiplexer on DPM 16 for microbranching on the state of the C bit. ARITH TRAP L goes to the SAC gate array on DPM 17 for initiating the arithmetic trap at BUT Service. The CCBR bits <1:0> go to the BUT multiplexer on DPM 15 and 16 for microbranching on their state.

The functionality of the CCC gate array is tested with microdiagnostics and indirectly with macrodiagnostics. Figure 2-119 shows the programming of the CC CTRL ROM and the good samaritan ROM that are not blasted by the microprogrammers.

2.9 INTERRUPTS AND EXCEPTIONS

During operation of the VAX-11/750, certain critical events can occur that require execution of software outside the explicit flow of control. Events that occur as a result of the process currently being executed by the CPU are called exceptions. Events that occur as a result of the system as a whole (external to the process being executed) are called Interrupts. Associated with each type of interrupt is an interrupt priority level (IPL). IPLs are used to arbitrate the servicing of multiple hardware and software interrupts. Table 2-59 shows all IPL used in this system. All exceptions (E) listed in this table carry an IPL of 1F.

Table 2-59 lists the system control block (SCB) for this system. The following is a expanded discussion of interrupts and exceptions and some terminology used when dealing with this subject.

An interrupt is an event other than an exception, branch, jump, case, or call instruction that changes the normal flow of instruction execution. They are generally external to the process executing when the interrupt occurs. Interrupts occur one cycle after IRD 1 or are explicitly tested by microcode.

Good Samaritan Encoding

	Good Samaritan Inputs			Good Samaritan Outputs
WCTRL Function	**WCTRL**	**CC**	**LIT 0 H**	**CC CTRL <3:0>**
WRITE PSL	00	X	X	9
WRITE PSW	01	X	X	B
READ PSL	04	X	X	3
WRITE CC	05	X	X	A
CC MISC 1	06	0	0	5
CC MISC 1	06	1	0	8
CC MISC 1	06	2	0	7
CC MISC 1	06	3	0	6
CC MISC 1	06	X	1	0
CC MISC 2	07	0	0	2
CC MISC 2	07	1	0	F
CC MISC 2	07	2	0	0
CC MISC 2	07	3	0	1
CC MISC 2	07	X	1	0
Any other WCTRL function		0	0	0
Any other WCTRL function		1	0	C
Any other WCTRL function		2	0	E
Any other WCTRL function		3	0	4
Any other WCTRL function		X	1	0

Figure 2-119 Good Samaritan Encoding

Table 2-59 Interrupts and Exceptions IPL Levels and System Control Block Format

Vector	Description	IPL	I/E
SCBB+0	Not Used	–	–
SCBB+4	Machine Check CS Parity Bad IRD Memory Error Cache Parity	1F	E

Table 2-59 Interrupts and Exceptions IPL Levels and System Control Block Format (Cont)

Vector	Description	IPL	I/E
SCBB+8	Kernel Stack Invalid	1F	E
SCBB+C	Power Fail	1E	I
SCBB+10	Reserved Opcode	*	E
SCBB+14	Customer Opcode XFC	*	E
SCBB+18	Reserved Operand	*	E
SCBB+1C	Reserved Address Mode	*	E
SCBB+20	Access Violation	*	E
SCBB+24	Translation Invalid	*	E
SCBB+28	Trace Trap	*	E
SCBB+2C	Breakpoint Opcode	*	E
SCBB+30	Compatability Mode	*	E
SCBB+34	Arithmetic Trap	*	E
SCBB+40	CHMK	*	E
SCBB+44	CHME	*	E
SCBB+48	CHMS	*	E
SCBB+4C	CHMU	*	E
SCBB+54	Corrected Read Data	1A	I
SCBB+60	Write Bus Error	1D	I
SCBB+84	Soft Interrupt	1	I
SCBB+88	Soft Interrupt	2	I
SCBB+8C	Soft Interrupt	3	I
SCBB+90	Soft Interrupt	4	I
SCBB+94	Soft Interrupt	5	I
SCBB+98	Soft Interrupt	6	I
SCBB+9C	Soft Interrupt	7	I
SCBB+A0	Soft Interrupt	8	I
SCBB+A4	Soft Interrupt	9	I
SCBB+A8	Soft Interrupt	A	I
SCBB+AC	Soft Interrupt	B	I
SCBB+B0	Soft Interrupt	C	I
SCBB+B4	Soft Interrupt	D	I
SCBB+B8	Soft Interrupt	E	I
SCBB+BC	Soft Interrupt	F	I
SCBB+C0	Interval Timer	18	I
SCBB+F0	TU58 Receive	17	I
SCBB+F4	TU58 Transmit	17	I
SCBB+F8	Console Receive	14	I
SCBB+FC	Console Transmit	14	I
SCBB+160	Massbus Adapter 0	15	I
SCBB+164	Massbus Adapter 1	15	I
SCBB+168	Massbus Adapter 2	15	I

*Current IPL not changed for these exceptions.

Table 2-59 Interrupts and Exceptions IPL Levels and System Control Block Format (Cont)

Vector	Description	IPL	I/E
SCBB+200	Unibus (SCBB+200+Unibus Vector)	14–17	I

*Current IPL not changed for these exceptions.

An exception is an event detected by the hardware other than an interrupt, jump, branch, case, or call instruction that changes the normal flow of instruction execution. An exception is always caused by the execution of an instruction or set of instructions. Exceptions occur anytime during execution. Examples are as follows.

1. Attempts to execute a privileged or reserved instruction
2. Trace traps
3. Compatibility mode faults
4. Breakpoint instruction execution
5. Arithmetic traps

The three types of hardware exceptions are as follows.

1. Trap – An exception condition that occurs at the end of the instruction that caused the exception. The PC saved on the stack is the address of the next instruction that would normally have been executed (arithmetic trap).

2. Fault – A hardware condition that occurs in the middle of an instruction and leaves the registers and memory in a consistent state that allows the instruction to restart, thus allowing for correct results once the fault has been cleared or eliminated (reserved address mode).

3. Abort – An exception that occurs in the middle of an instruction and leaves the registers and memory in an indeterminate state that may prohibit an instruction restart (machine check).

The interrupt priority level (IPL) is the interrupt level at which the processor executes when an interrupt is granted. There are 31 possible priority levels. IPL 1 is the lowest and IPL 1F is the highest. (Only 24 levels are used.)

An interrupt or exception vector is an offset from the SCBB that contains the starting address of a procedure to be executed when a given interrupt or exception occurs.

The system control block base register (SCBB) is a processor register containing the base address of the system block (IPR 11 (MTEMP 4).

The interrupt block diagram contains the following.

1. The interrupt chip (INT) is mounted on the UBI board with inputs from chips on the DPM board, MIC board, and other circuits on the UBI board.

2. Interrupt chip inputs WCTRL <5:0> – Comes from the control store (CS) and is used to issue commands to the INT chip, such as the following.

   a. Read or write status data to or from the WBus.

   b. Issue Unibus grants.

   c. Place the results of a return from exception or interrupt (REI) check onto the microvector lines.

   d. Place certain status data onto the microvector lines.

Table 2-60 lists the types of interrupts, traps, and microtraps that can occur in the VAX-11/750. Also listed are the initial CPU control store microaddresses for each of these conditions (see Add). These microaddresses are divided into three major categories as follows.

1. Traps – micro Add = 1n (hex)
2. Microtraps – micro Add = 2n (hex)
3. Interrupts – micro Add = 3n (hex)

NOTE
n may equal 1 through F (hex).

Each of the above conditions results in a microaddress being generated to the CCS on the CS ADDR <5:0> H lines. Paragraphs 2.9.1 through 2.9.3 describe how these addresses are generated.

Table 2-60 Fixed Control Store Address

ADD	Function of Vector	Method of Initiation
0000	Power-Up	
0011	Arithmetic Trap	DO Service
0012	FPA Integer Overflow Trap	DO Service
0014	Timer Service	DO Service
0015	T-Bit trap	DO Service
0016	Console P Trap	DO Service
0020	Control Store Parity Error	Microtrap
0021	Read Unaligned Data	Microtrap
0022	MSRC XB Miss	Microtrap
0023	MSRC XB ACV	Microtrap
0024	Write Unlock Unaligned Data	Microtrap
0025	Write Unaligned Data	Microtrap
0026	Write Unlock Crossing Page Boundry	Microtrap

Note: MSRC XB TB Error
MSRC XB Bus Error
Bus Error
Unaligned Unibus Data
TB Error
BUT XB TB Error
BUT XB Bus Error

Table 2-60 Fixed Control Store Address (Cont)

ADD	Function of Vector	Method of Initiation
0027	Write Crossing Page Boundry	Microtrap
0028	Machine Check Exceptions (See Note)	Microtrap
0029	BUT XB Miss	Microtrap
002A	Read TB Miss	Microtrap
002B	Write TB Miss	Microtrap
002C	FPA Reserved Operand	Microtrap
002D	BUT XB ACV	Microtrap
002E	Read ACV	Microtrap
002F	Write ACV	Microtrap
0038	Soft Interrupt	DO Service, Execution Flows
0039	Console Interrupt	DO Service, Execution Flows
003A	Unibus Interrupt	DO Service, Execution Flows
003B	Interval Timer Interrupt	DO Service, Execution Flows
003C	Corrected Memory Interrupt	DO Service, Execution Flows
003E	Write Bus Error Interrupt	DO Service, Execution Flows
003F	Power Fail	DO Service, Execution Flows

Note: MSRC XB TB Error
MSRC XB Bus Error
Bus Error
Unaligned Unibus Data
TB Error
BUT XB TB Error
BUT XB Bus Error

2.9.1 Interrupt Microaddress Generation

All interrupts are generated from the interrupt chip (INT) located on the UBI module. A microaddress in the range 38 through 3F (hex) is conveyed to the CCS. The exact microaddress depends on the interrupt type to be serviced. The microaddress is made up of three pieces of logic in the CPU (see Figure 2-120), consisting of these bits.

Bits 0, 1 and 2 from INT gate array chip (UBI module)
Bit 3, from Microtrap (UTR) gate array (MIC module)
Bits 4 and 5, from microsequencer MSQ chip (DPM module)

As the system is running macrocode, a request for an interrupt is sent to the INT chip. Here a console interrupt is used as an example. However, all interrupts are handled in basically the same way. The one factor that distinguishes one interrupt type from another is the output of the INT chip on MICRO-VECTOR <2:0> H (see Table 2-61). The INT chip compares the interrupt priority level (IPL) of the request to the IPL already present in its IPL register. (The IPL register is internal to the INT chip.) If the requested IPL is higher, the signal INT PENDING is sent to the SAC chip on the DPM module. System response to INT PENDING is delayed until one microcycle after the IRD 1 time of a macroinstruction. When IRD 1 is decoded by the SAC chip from the microword BUT field, the INT chip, MSQ chip and UTR chip respond at the same time. One microcycle after the IRD 1 cycle, if INT PENDING is asserted, the SAC chip generates DO SRVC L and ENABLE UVECT H. These signals allow the three previously mentioned chips to produce the needed microaddress (39 hex) on CS ADDR <5:0> H for console interrupt. This microaddress is created as follows (see Figure 2-120).

Table 2-61 INT Chip MICROVECTOR <2:0> H Output Microvector Value Chart

IPL	Name	Microvector <2:0> H
00	No Interrupt Request Present	000
01–0F	(HSIPR) Highest Software Interrupt Pending Request	000
14	(SLINE INT) Serial Line Interrupt	001
14–17	(SBR) Synchronous Bus Request (4–7)	010
18	(TIMER INT) Interval Timer Interrupt	011
1A	(CDIR) Corrected Data Interrupt Request	100
1B	Reserved	101
1D	(WEIR) Write Bus Error Interrupt Request	110
1E	(SPFIR) Synchronous Power Fail Interrupt Request	111

INT chip – When a console interrupt is requested, the lower three bits (0, 1 and 2) of the needed microaddress are sent to tri-state drivers on the MICROVECTOR <2:0> H lines (these drivers are not presently enabled). When DO SRVC L is generated by SAC, the INT chip allows bits <2:0> to be driven onto the MICROVECTOR <2:0> H lines. These bits are driven to 001 for console interrupt.

MSQ chip – When DO SRVC L = L, ENABLE U VECT H = H, MICRO ADD INH L = H, and MSEQ INIT L = H, the MSQ chip outputs a low on CS ADDR 5L and CS ADDR 4L. This action always occurs for an interrupt (see Table 2-62). Note that this table is also used for traps and microtraps.

UTR chip – DO SRVC L is also supplied to the UTR chip. Here it disables the tri-state driver to MICROVECTOR line 3. MICROVECTOR line 3 is driven high by the UTR chip. ENABLE UVECT H is high at this time, thus enabling the NAND gates shown in Figure 2-120. This permits inputs from the MSQ, UTR, and INT chips to be ORed together to produce a microaddress of 39 (hex) on CS ADDR <5:0> H.

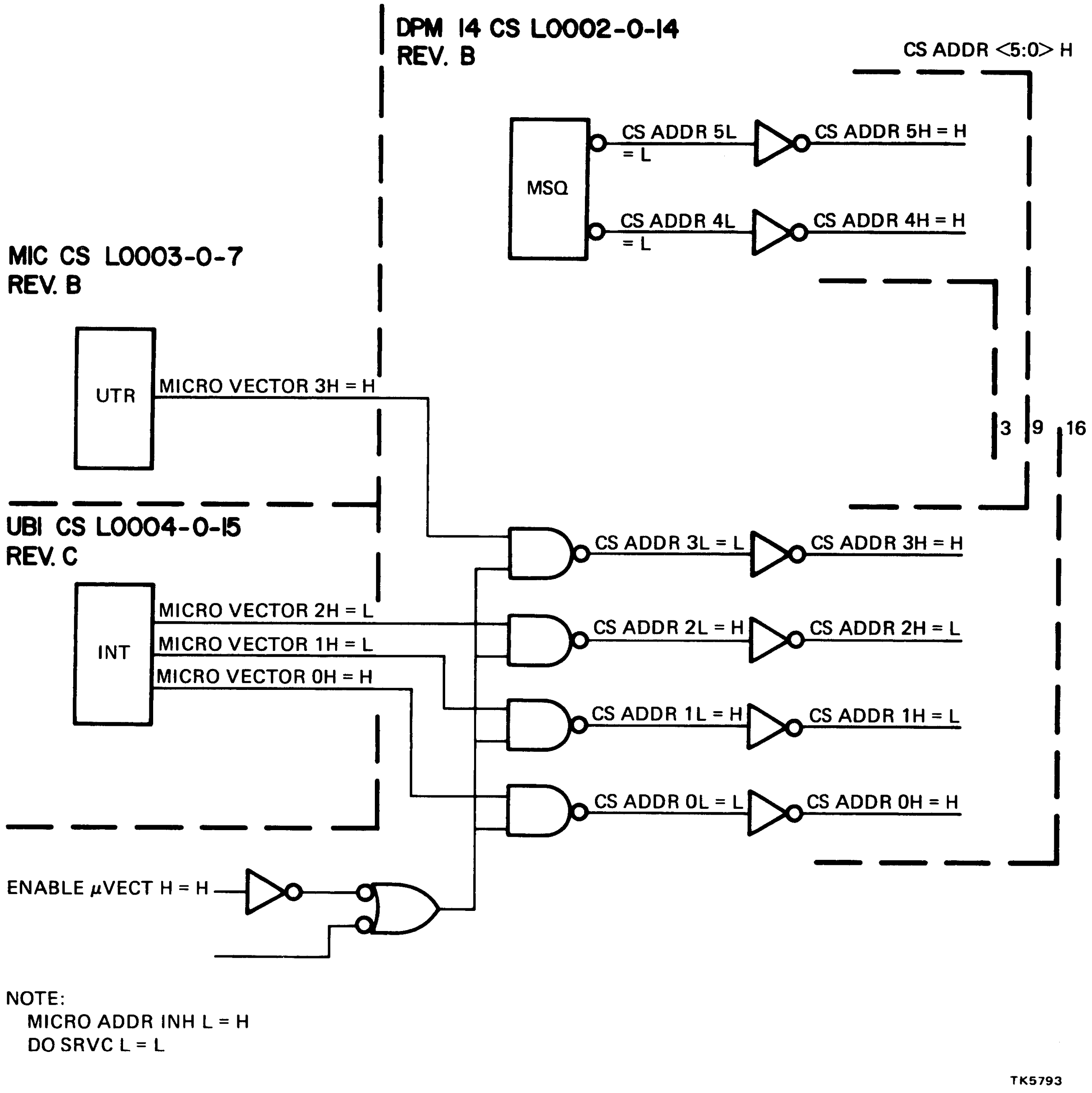


Figure 2-120 Microaddress Generation for Interrupt (CONSOLE INT)

NOTE

Figure 2-120 shows the source of CS ADDR <5:0> H, but does not show the various enables and controlling signals involved with the above operation. For details, see the appropriate schematics, indicated on Figure 2-120, and Table 2-62.

Table 2-62 MSQ CS ADDR L <5:4> L Output

	MICRO ADDR INH L	ENABLE UVECT H	DO SRVC L	Internal MSQ INIT L	CS ADDR L* 5 4
Microtrap	H	H	H	H	L H
Trap	H	L	L	H	H L
Interrupt	H	H	L	H	L L

*L = True (1), H = False (0).

2.9.2 Trap Condition Microaddress Generation

Producing a microaddress for a trap condition is a less complicated process than that for interrupts. Table 2-60 shows that all traps are 1n (hex). (See Figure 2-121.) Two chips are responsible for producing the microaddress to the CPU control store: SAC outputs CS ADDR <2:0> L and MSQ outputs CS ADDR <5:3> L. Figure 2-121 shows the five possible trap signals input to the SAC chip.

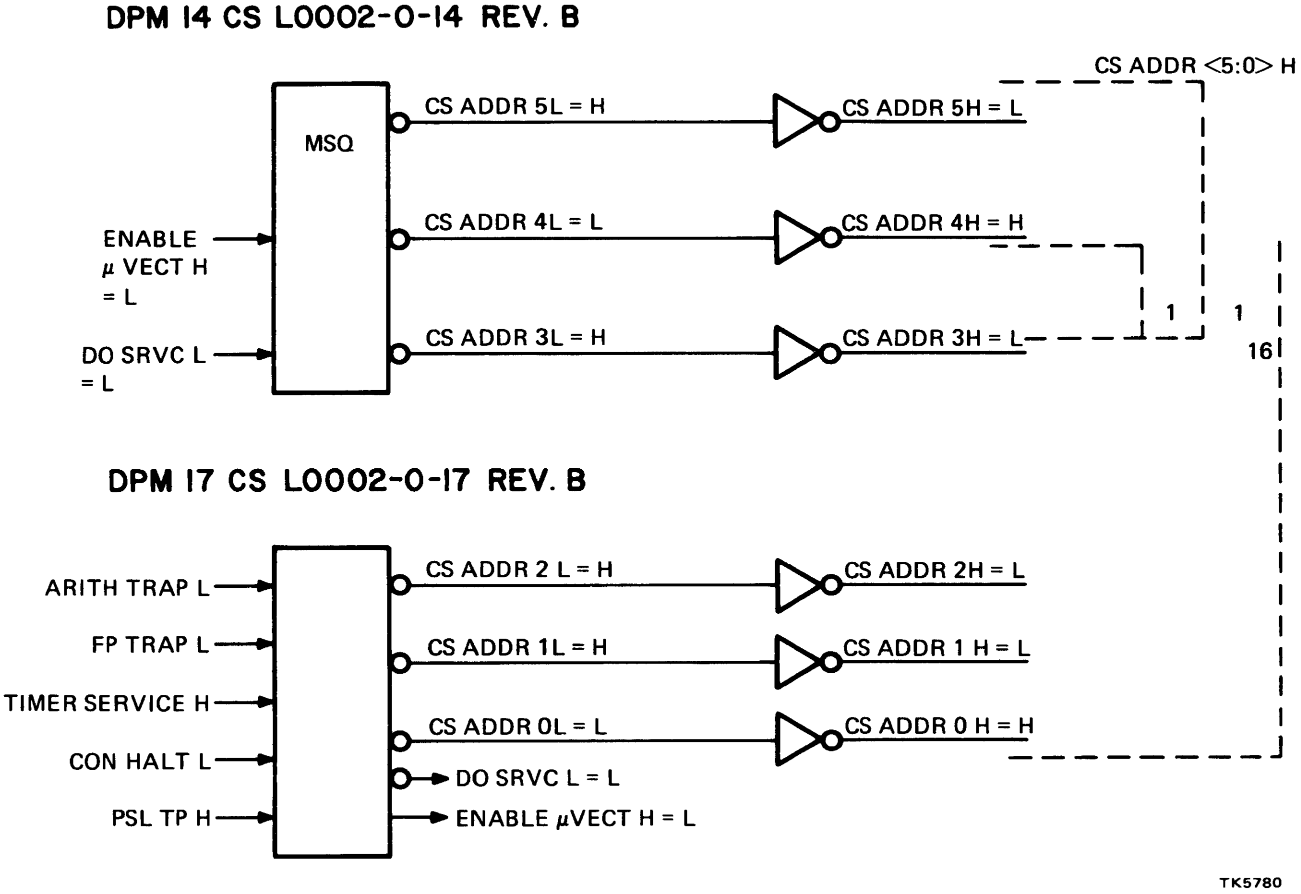


Figure 2-121 Microaddress Generation for Trap (Arithmetic Trap)

1. Arithmetic Trap (ARITH TRAP L)
2. FPA Integer Overflow Trap (FP TRAP H)
3. Timer Service (TIMER SERVICE H)
4. T-Bit Trap (PSL TP H)
5. Console – P Trap (CON HALT L)

An arithmetic trap is used in the following example (see Figure 2-121).

> SAC – the signal ARITH TRAP L = L. The SAC chip responds to ARITH TRAP L by asserting low, high, and low respectively on CS ADDR <2:0> L. (See Table 2-63.) The SAC chip also outputs DO SRVC L to the MSQ chip. ENABLE UVECT H (output by SAC) is inactive, low, at this time.
>
> MSQ – DO SRVC L = L and ENABLE UVECT H = L are recognized by the MSQ chip, which then outputs a high and low on CS ADDR <5:4> respectively (see Table 2-62).

The output of SAC and MSQ are ORed together as shown in Figure 2-121, thus producing a microaddress on the CS ADDR <5:0> H lines of 11 (hex).

2.9.3 Microtrap Condition Microaddress Generation

Microaddress generation for microtraps is accomplished by the UTR and MSQ gate arrays (see Figure 2-122). However, the SAC is also instrumental in this operation. Table 2-60 shows the microaddress used for the various microtrap conditions. For this discussion a READ TB MISS microtrap operation is used as an example. Table 2-60 shows that the microaddress for this operation is 2A (hex). This microaddress is generated as follows.

> UTR Chip – This chip constantly monitors events occurring during each microinstruction. If a translation buffer (TB) miss occurs during a read microinstruction, the instruction cannot be completed. Microcode flows (starting at microaddress 2A (hex) must be executed in order to fetch the needed PTE from memory. In response to TB PARITY ENA H = H and the absence of a TB HIT (TB HIT 1 H = L and TB HIT 0 H = L), UTRAP L is generated. In addition, MICROVECTOR <3:0> H are set to H, L, H, L (see Table 2-64 for other microtrap conditions) These lines are driven on the rising edge of MCLK L.
>
> SAC Chip – When this chip receives UTRAP L it generates ENABLE UVECT H. DO SRVC L stays inactive (high).
>
> MSQ Chip – The MSQ recieves ENABLE UVECT H and outputs low and high on CS ADDR 5 L and CS ADDR 4 L respectively (see Table 2-62). The outputs of the MSQ and UTR chips are ORed together as shown in Figure 2-122. This produces an address of 2A (hex) on CS ADDR <5:0> H.

Table 2-63 SAC Chip CS ADDR <2:0> L (Output Conditions for Traps)

Trap Condition	CS ADDR <2:0> L		
	2	1	0
Arithmetic Trap	H	H	L
FPA Integer Overflow	H	L	H
Timer Service	L	H	H
T-Bit Trap	L	H	L
Console P Trap	L	L	H

Note: L = True (0), H = False (1).

Table 2-64 UTR Chip MICROVECTOR <3:0> H Output

Microtrap Condition	Priority	Microvector <3:0> H			
		3	2	1	0
Control Store Parity Error	1	L	L	L	L
FPA Reserved Operand	2	H	H	L	L
MSRC XB TB Error	3	H	L	L	L
MSRC XB Bus Error	4	H	L	L	L
Bus Error	5	H	L	L	L
Reserved	6	H	L	L	L
MSRC XB Miss	7	L	L	H	L
MSRC XB ACV	8	L	L	H	H
TB Error	9	H	L	L	L
Read TB Miss	10	H	L	H	L
Write TB Miss	11	H	L	H	H
Read ACV	12	H	H	H	L
Write ACV	13	H	H	H	H
Write Crossing Page Boundary	14	L	H	H	H
Write Unlock Crossing Page Boundary	15	L	H	H	L
Read Unaligned Data	16	L	L	L	H
Write Unaligned Data	17	L	H	L	H
Write Unlock Unaligned Data	18	L	H	L	L
BUT XB TB Error	19	H	L	L	L
BUT XB Bus Error	20	H	L	L	L
BUT XB Miss	21	H	L	L	H
BUT XB ACV	22	H	H	L	H

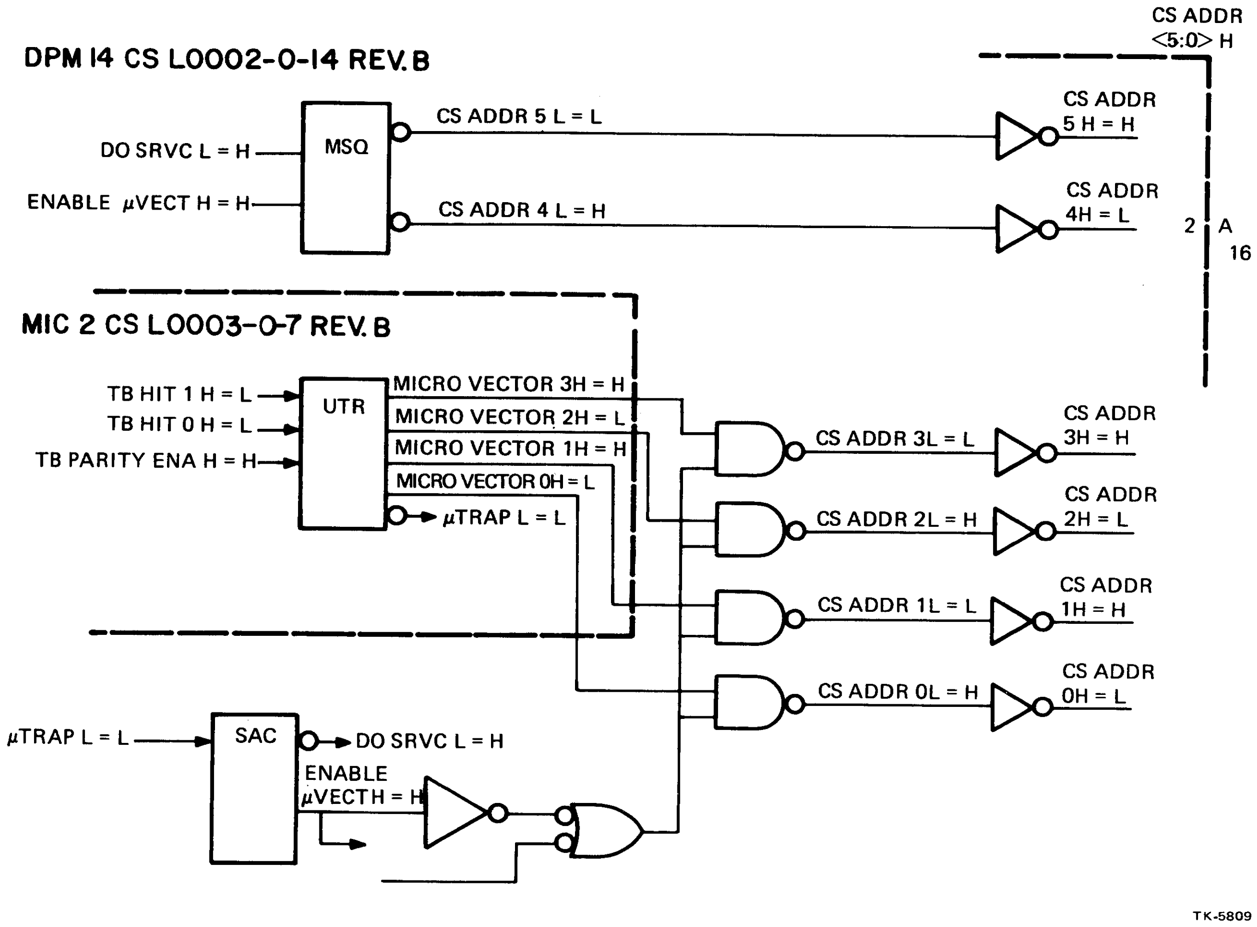


Figure 2-122 Microaddress Generation for Microtrap (READ TB MISS)

APPENDIX A
LIMITED GLOSSARY OF MNEMONICS

ACCS – accelerator control and status register

ACV – access control violation

ADD – address control

ADK – address controller

ALK – arithmetic logic control

ALP – arithmetic and logical processor

ALU – arithmetic logic unit

AMUX – address multiplexer

AST – asynchronous system trap

ASTR – AST - level register

ATCR – arithmetic trap code register

BAR – buffered address register

BCD – binary coded decimal

BCLK – base clock

BDP – buffered data path

BR – bus request

BUT – branch under test

CADR – cache disable register

CAER – cache error summary register

CAK – cache controller

CCC – condition code logic (gate array)

CCS – CPU control store

CI – carry input

CLA – carry look-ahead

CM – compatibility mode/current mode

CMC – memory controller

CMI – CPU memory interconnect

CMIERR – CMI error register

CMK – CPU memory controller

CMOS – complementary metal-oxide semiconductor

CON – console interface

CSA – control store bus address

CSRD – console storage receiver data

CSRS – console storage receiver status

CSTD – console storage transmit data

CSTS – console storage transmit status

CUR MODE – current mode

DCLK – destination clock

DDP – direct data path

DPM – data path module

ESP – executive stack printer

FPA – floating-point accelerator

GPR – general purpose register

HPBG – highest pending bus grant

HSIPR – highest software interrupt pending request

ICCS – internal counter control and status/internal clock control and status

ICR – interval count register

INT – interrupt logic

IPL – interrupt priority level

IPR – internal processor register

IR – instruction register/interrupt request

IRD – instruction decode (chip)

IS – interrupt stack (flag)

ISP – interrupt stack pointer

ISTRM – instruction stream data

JSR – jump to subroutine

KSP – kernel stack pointer

MA – memory address

MAD – memory address

MAP – memory address map

MBA – Massbus adapter

MCESR – machine check error summary register

MCLK – microsequencer clock

MDR – memory data register/memory data routing and alignment

MEMSCAR – memory status and control register

MIC – memory interface and control/memory interconnect

MME – memory management enabled

MSQ – microsequencer (chip or gate array)

MSRC – MBUS data source (microfield)

MTEMP – M-type temporary registers (output to MBus)

MFPR – move from processor register instruction

MTPR – move to processor register instruction

NICR – next interval count register

NPR – non-processor request

OSR – operand specifier register

P latch – position latch

PA – physical address

PAD – physical address (lines)

PBR – process base register

PC – program counter

PCBB – process control block base

PFN – page frame number

PHB – practically half the BUTs (microsequencer chip or gate array)

PLR – process length register

PMR – performance monitor register

PRK – prefetch control chip

PSL – processor status longword

PTE – page table entries

Q,D CLK – loads and shifts Q and D register

RBS – register backup stack

RBSP – RBus pointer

RBSP – register backup stack pointer

RCAR – received CMI address register

RDM – remote diagnostic module

REI – return from exception or interrupt (check)

RNUM – register number register

ROT – (refers to super rotator)

RTEMP – temporary registers (output to RBus)

RXCS – console receive and status

RXDB – console receive data buffer

S latch – size latch

SAC – service arbitration and control (gate array)

SBR – system base register

SCB – system control block

SCBB – SCB base register

SID – system identification

SIR – software interrupt summary register

SIRR – software interrupt request register

SL – shift left

SLR – system length register

SPA – scratchpad address

SSP – supervisor stack pointer

SPFI – sync power fail interrupt

SPICR – scratchpad interval count register

SPNICR – scratchpad next interval count register

SPTE – system page table entry

SPW – scratchpad write

SR – shift right/service request

SRK – super rotator control

TAG – virtual translation address

TB – translation buffer

TBDR – translation buffer disable register

TBGDR – TB group disable register

TBGPR – TB group parity register

TBHR – TB hit register

TBIA – translation buffer invalidate all

TBIS – translation buffer invalidate single

TODR – time-of-day register

TOK – interval timer (gate array)

TOY – time-of-year (clock)

TR – transfer request

TXCS – console transmit control and status

TXDB – console transmit data buffer

UBI – Unibus interconnect

UCN – Unibus control

UDP – Unibus data path

UET – Unibus exerciser/terminator

USP – user stack pointer

UTR – microtrap

V bit – valid bit

VA – virtual address

WCS – writable control store

WDR – write data register

XB – execution buffer

VAX-11/750
Central Processor Unit
Technical Description
EK-KA750-TD-002

Reader's Comments

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 *Technical Documentation Catalog*, 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
444 Whitney Street
Northboro, MA 01532
Attention: Printing and Circulating Service (NR2/M15)
Customer Services Section

Order No. EK-KA750-TD

Fold Here

Do Not Tear – Fold Here and Staple

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Educational Services Development and Publishing
1925 Andover Street (TW/B01)
Tewksbury, MA 01867