VAX-11/750 LEVEL II

Student Workbook

For Internal Use Only

Copyright © 1980, Digital Equipment Corporation.
All Rights Reserved.

The reproduction of this material, in part or whole, is strictly prohibited. For copy information, contact the Educational Services Department, Digital Equipment Corporation, Bedford, Massachusetts 01730.

Printed in U.S.A.

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may not be used or copied except in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital.

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

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

SYSTEM INTRODUCTION

INTRODUCTION

The 11/750 system is an extension of the VAX Family System with many of the same characteristics of the VAX 11/780.

The 11/750 system allows users up to 4.3 billion virtual address while only using 2 MEG of physical memory. To do this the 11/750 has mass storage devices on a Mass bus or Unibus for quick and easy access by the CPU.

The 11/750 may run in two modes of operation:

1. NATIVE (VAX VMS)
2. COMPATABILITY (PDP-11)

There is also the capability of Remote diagnostics to help both the user and the field service technician. With this Introduction Module we will attempt to give you the basic facts and concepts of the 11/750 System including:

1. Basic Architecture
2. Analysis of Block Diagram
3. Physical Characteristics
4. Diagnostic Overview

11/750 SUPPORT COURSE

MODULE I: SYSTEM INTRODUCTION

SYNOPSIS

The system introduction module consists of 11/750 system characteristics and block diagram analysis.

OBJECTIVES

Provided with a blank 11/750 system block diagram and a list of 11/750 component names, correctly label the 11/750 system block diagram.

Given a list of 11/750 characteristics, correctly indicate as True/False the characteristics that make 11/750 a unique system.

SAMPLE TEST ITEM

Identify the following 11/750 unique characteristics as True or False.

1. The 11/750 Processor Microword is 99 bits. ____

2. The 11/750 has a Virtual Memory System. ____

3. The 11/750 data path is 16 bits. ____

4. There is no Remote Diagnosis capability with the 11/750 system. ____

RESOURCES

11/750 Specification
New Product Data Sheet
11/750 Block Diagram
11/750
11/750 Pocket Reference Guide

MODULE OUTLINE

I. SYSTEM INTRODUCTION

- A. Course Overview
- B. Basic Architecture
  - 1. 11/750 Specifications
  - 2. 11/750 Physical Characteristics
- C. Block Diagram Analysis
  - 1. CPU
    - a. Data Path Module
    - b. Memory Interconnect Module
    - c. 11/750 Control Store
    - d. Unibus Interface
    - e. Major Buses
  - 2. Memory
    - a. Controller (1)
    - b. Array Boards (up to 8)
  - 3. Options
    - a. CPU Options
      - 1. Writable Control Store
      - 2. Floating-Point Accelerator
    - b. Mass Bus Options
    - c. Unibus Options
    - d. Remote Diagnostic Module

D. 11/750 Physical Inspection

   1. Front Panel

   2. Card Locations

   3. Backplanes

   4. Power

E. Diagnostic Overview

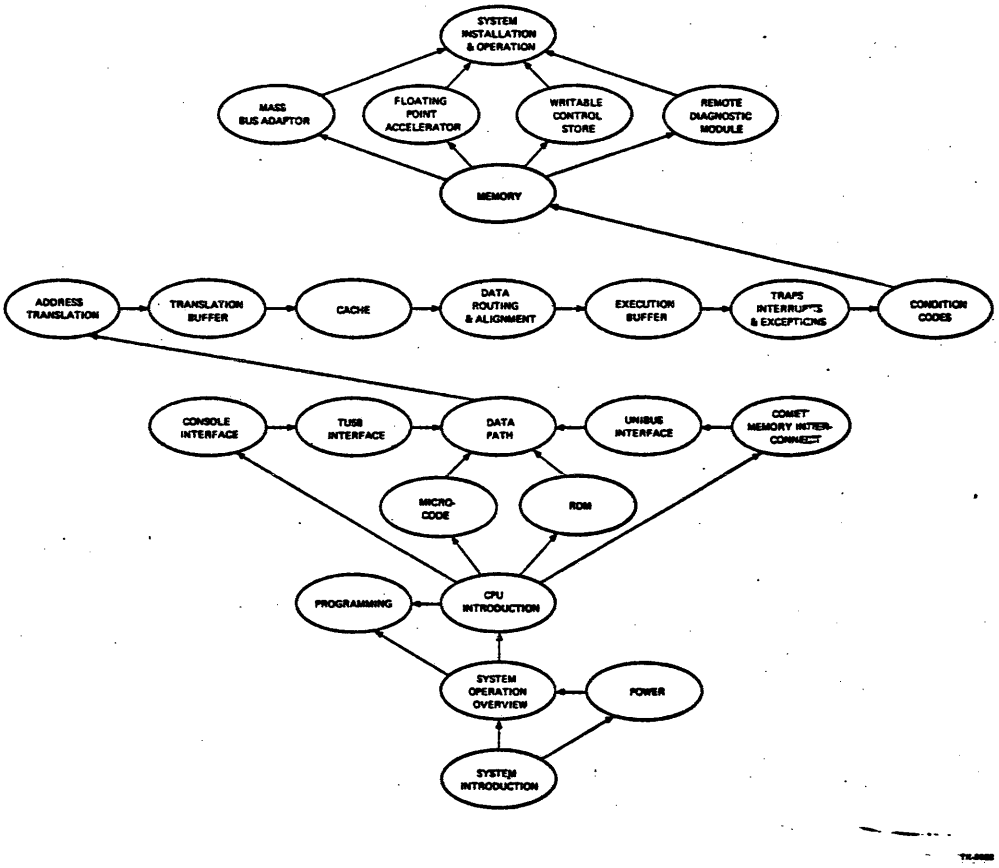


Figure 1-1 Course Map

Basic Architecture

1. Comet Specifications

    a. Micro-controlled machine

    b. Similar 32-bit architecture except:

        (1) Use of LSI curcuits (gate arrays)
        (2) Increases reliability
        (3) Decreases size

    c. Virtual 32-bit addressing (hexidecimal)

        (1) 4.3 billion virtual processes
        (2) 2 meg max physical (moss - batt. backup 10 min.)

    d. Two modes of operatiton

        (1) Native (VAX VMS)
        (2) Compatibility (PDP-11)

    e. Remote Diagnostic Capability

        (1) Company owner module

            (a) Increase level of service
            (b) Improves field service efficiency

    f. Power

        (1) 115 or 230 volts

2. Physical Characteristics

    a. 40 inches high, 30 inches deep, 29 inches wide

    b. Five basic sections

        (1) CPU - 4 boards - 3 major buses
        (2) Options - CPU and I/O
        (3) Front Panel
        (4) Backplanes - Comet and Unibus
        (5) Power

What Is an 11/750 Gate Array?

400 identical two transistor cells which can:

1. Be connected to form 4 input NAND
2. Together with a neighboring cell be connected to form 4 input NAND or AND.

44 identical transceiver cells which can:

1. By disconnecting the receiver, be a TTL, Tri-state or open collector drives (Internal array to outside world)
2. By disconnecting the driver, be a TTL receiver (high impedance)
3. Both

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

Die Size - .215 inches X .244 inches
Power Utilized per Die - 2 watts max

Package Size - 1.44 sq. in. (2.4 inches X 0.6 inches)
Number of Pins/Package - 48

I/O Circuits/die - 44 I/O transceiver gates
Logic Gates - 400 identical 4 input NAND gates

Voltage Used - +2.5 volts, +5 volts
Speed per Gate - 5-10 nanoseconds

Unique Gate Array Types

CPU and Memory Controller - 27
Floating-Point Accelerator - 7
Mass Bus Adaptor - 5

Total Number of Gate Arrays Used:

CPU and Memory Controller - 55
Floating-Point Accelerator - 28
Mass Bus Adaptor - 12

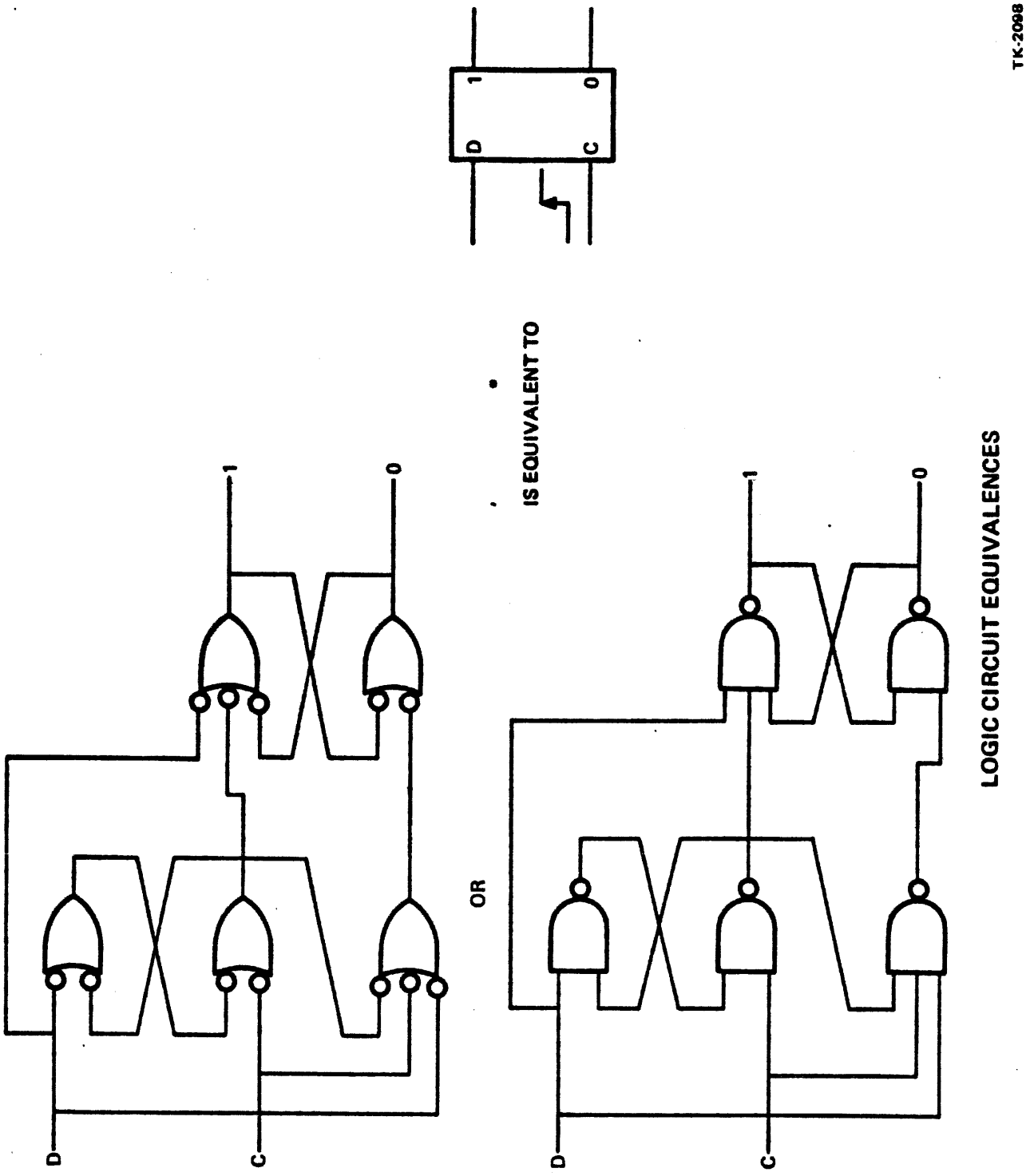
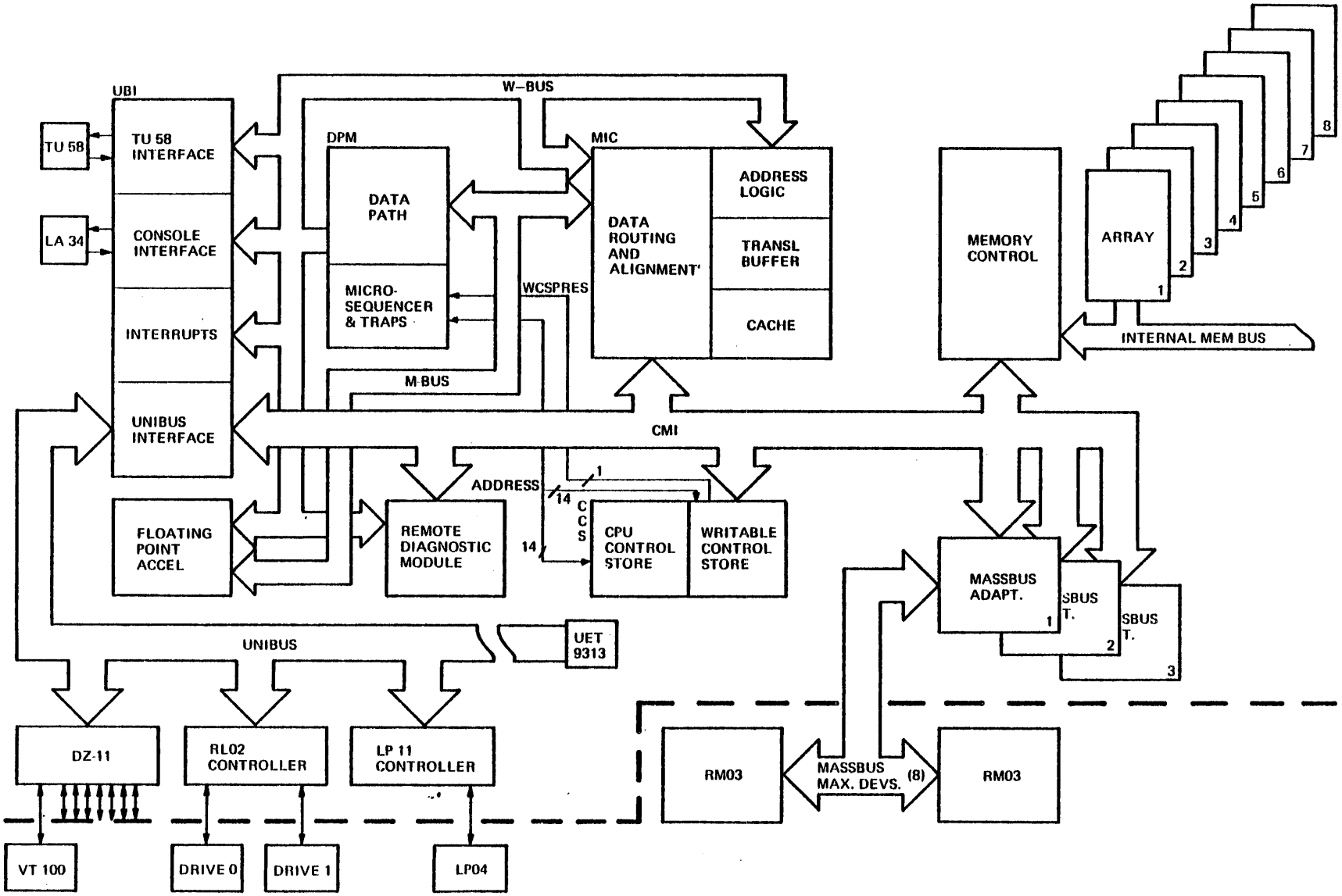


Figure 1-2 Logic Equivalent Circuits



COMET SIMPLIFIED SYSTEM BLOCK DIAGRAM

TK 2079

Figure 1-3 11/750 Simplified Block

Block Diagram Analysis

1. CPU - Four Boards

   a. Data path module (slot 2) DPM

      (1) Functions

         (a) Control Microsequencing
         (b) Arithmetic actions
         (c) Generate basic CLK from OSC on CCS

      (2) Contents

         (a) Super Rotator
         (b) ALU
         (c) 64 GPRs
         (d) Interval timer and basic CLK
         (e) Microsequencer

   b. Memory Interconnect Module (Slot 3) MIC

      (1) Functions

         (a) Acts as memory management by making physical address from virtual address

         (b) Checks for that physical address to find if it is located in memory

         (c) Stores data in a 1K cache for quick use by the CPU

         (d) Holds "PC" (program counter) to allow CPU greater efficiency. (Updates the PC without CPU microintervention.)

         (e) Due to VAX data and instruction storage it aligns data or instructions to useable positions for DPM.

         (f) Generates microtraps for needed interrupts and exceptions

      (2) Contents

         (a) Address Logic

         (b) Translation Buffer - 2-way set associative cache

(c) Cache - 1K direct for data

(d) Data routing and alignment holds PC and execution buffer.

c. CPU Control Store (Slot 5) CSS

(1) Functions

(a) Hold proms (microprogram)
(b) Mother board for WCS option
(c) Works in conjunction with microsequencer

(2) Contents

(a) Proms for 80 bit microword (1K)
(b) Snap on for WCS

d. CPU Unibus Interface Module (Slot 4) UBI

(1) Functions

(a) Interface the TU58 cartridge tape to operating system. TU58 used possibly for booting the system or loading of diagnostics. One chip, serial data between TU58 and interface. Parallel between interface and CPU.

(b) Interface the console terminal so operator may talk to system. May be used as user input. Once chip serial data between console and interface. Parallel between interface and CPU.

(c) Interface for all data to be passed between CPU and unibus.

(d) Generates all interrupts from unibus, massbus devices, TU58 and console terminal.

(e) Acts as generator for time of year (TOY) clock. To keep system informed as to correct time. Battery run. (Batteries included)

e. Buses: M, W, CMI

(1) Functions - to interconnect via etching on backplane all sections of CPU to allow them to communicate.

(a) M Bus - used to transmit data to and from scratch pad registers, memory data registers, PC, virtual address registers, PC, virtual address registers and data path module. Also included is data to and from FPA.

(b) W Bus - originates at ALU in DPM and xmits data to data routing and alignment, address logic, unibus interface and FPA.

(c) CMI Bus - (CPU Memory Interconnect) etch on backplane that connects CPU to all I/O buses and memory for data exchange. Synchronous interlocked 160 nanosec cycle.

(2) Contents

(a) M Bus - data lines <31:0>
(b) W Bus - data lines <31:0>
(c) CMI Bus - 45 lines

i. 32 data/address
ii. 1 wait
iii. 1 hold
iv. 1 busy
v. 7 arbitration lines
vi. 2 status
vii. 1 8MHZ clock

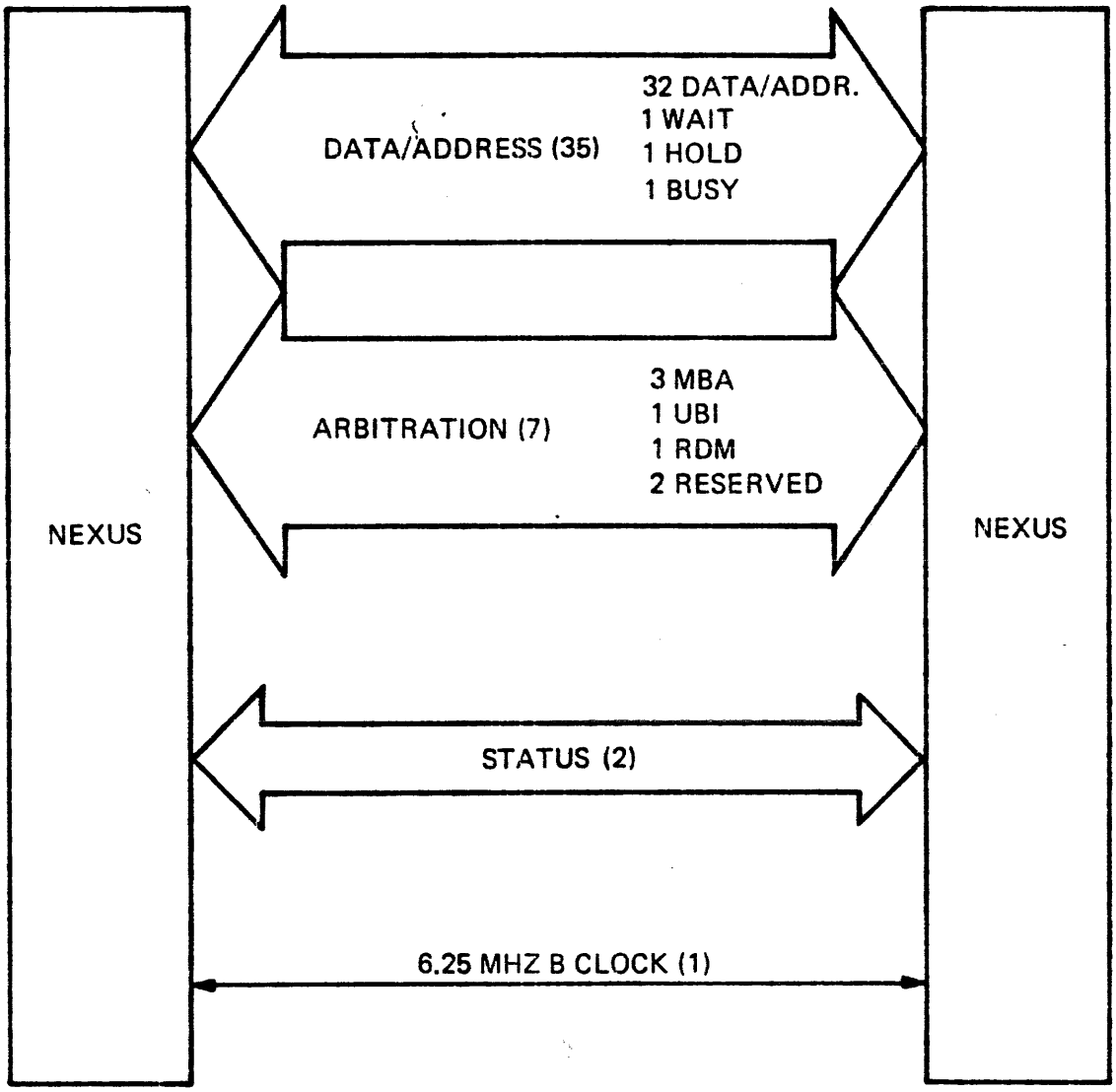


Figure 1-4 CMI Structure

2. Memory - two to nine modules

   a. Controller module (Slot 1Ø)

      (1) Functions

         (a) Controls data moving to and from CMI and memory

         (b) Controls refresh circuitry for moss memory

         (c) Performs error correction for 1 bit

         (d) Has boot ROMs (up to 4)

      (2) Contents

         (a) Two clocks

            i. fast - used between CMI and controller

            ii. slow - used between controller and memory

         (b) Error correcting circuitry

         (c) Up to 4 boot ROMs

         (d) refresh circuitry

   b. Array Boards - up to 8 (slots 1 -> 8 in hex)

      (1) Function - hold data for storage - up to 2 meg

      (2) Content

         (a) 256K of mos on each board
         (b) max 8 boards

   c. Memory Internal Bus

      (1) Function - carry data addresses and control signals between controller and memory

      (2) Contents

         (a) 39 data lines

(b) 7 multiplexed chip address lines
(c) Two address lines for 16K 39 grp select

(d) One ROM address strobe or column address strobe

(e) One read/write control

3. Options

a. CPU options - 2 boards

(1) Writeable control store (slot 5 in ex-hex)

(a) Function - allow programmer to write his own microcode. Ex: subroutine

(b) Contents-RAMs, board is plugged to CCS

(2) Floating-point accelerator (slot 1 in ex-hex)

(a) Function - used for working arithmetic functions which have large numbers of many decimal places

b. Massbus Options - maximum of 3 adaptors (slots 7 -> 9 in ex-hex)

(1) Function - to interface data between one of 8 devices possible on each adaptor to the CMI bus. Devices could be used for storage of operating system, space for user programs or overall virtual memory space.

(2) Contents - made up of LSI arrays and standard logic

Note

Interrupts from massbus devices are given a BR4 level and passed to CPU via unibus interface board. To receive BG back it must have the BG jumper removed if adaptor is present in a slot.

c. Unibus Options (slots 1 -> 9 unibus backplane)

(1) Function - many and varied according to what devices are purchased. DZ11 is normally bought to interface up to 8 user terminals.

(2) Contents - relative to what is purchased, but will always have M9313 for end of the bus termination and diagnostics.

d. Remote Diagnostic Module (slot 6 ex-hex) RDM

(1) Functions

(a) Needed to run microdiagnostics

(b) Run macrodiagnostics from remote site

(c) If macros won't run, differentiate between CPU and memory

(d) Utilize TU58 as backup source for hardcore, cache/TB diagnostic supervisor in case mass medium is down

(e) Down line load of micros is not a goal

(2) Contents

(a) RAMs - to hold microdiagnostic monitor

(b) DCS (Data Control Store) to hold microsequencers loaded by monitor

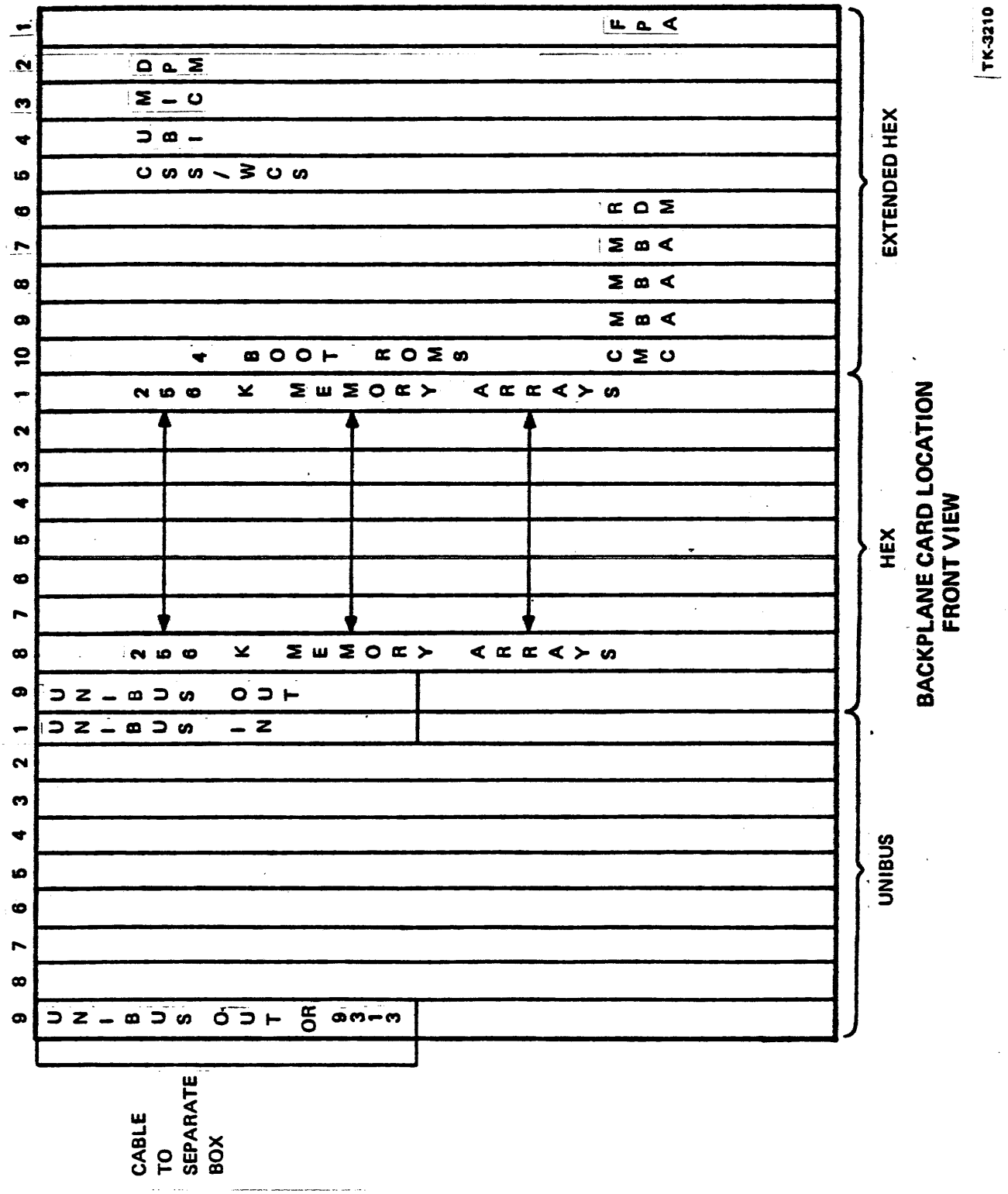


Figure 1-5 Backplane Card Location Front View

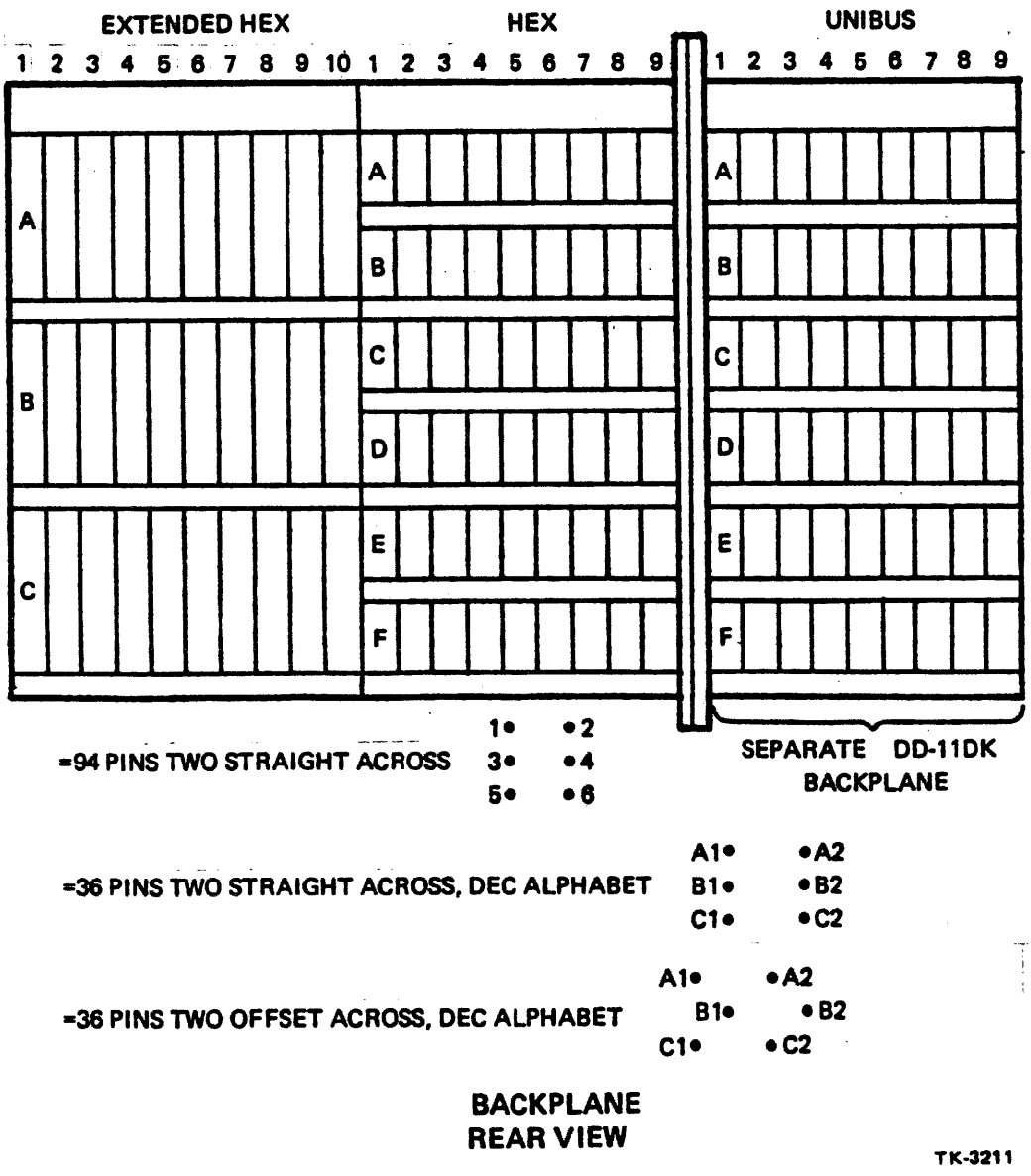


Figure 1-6 Backplane Rear View

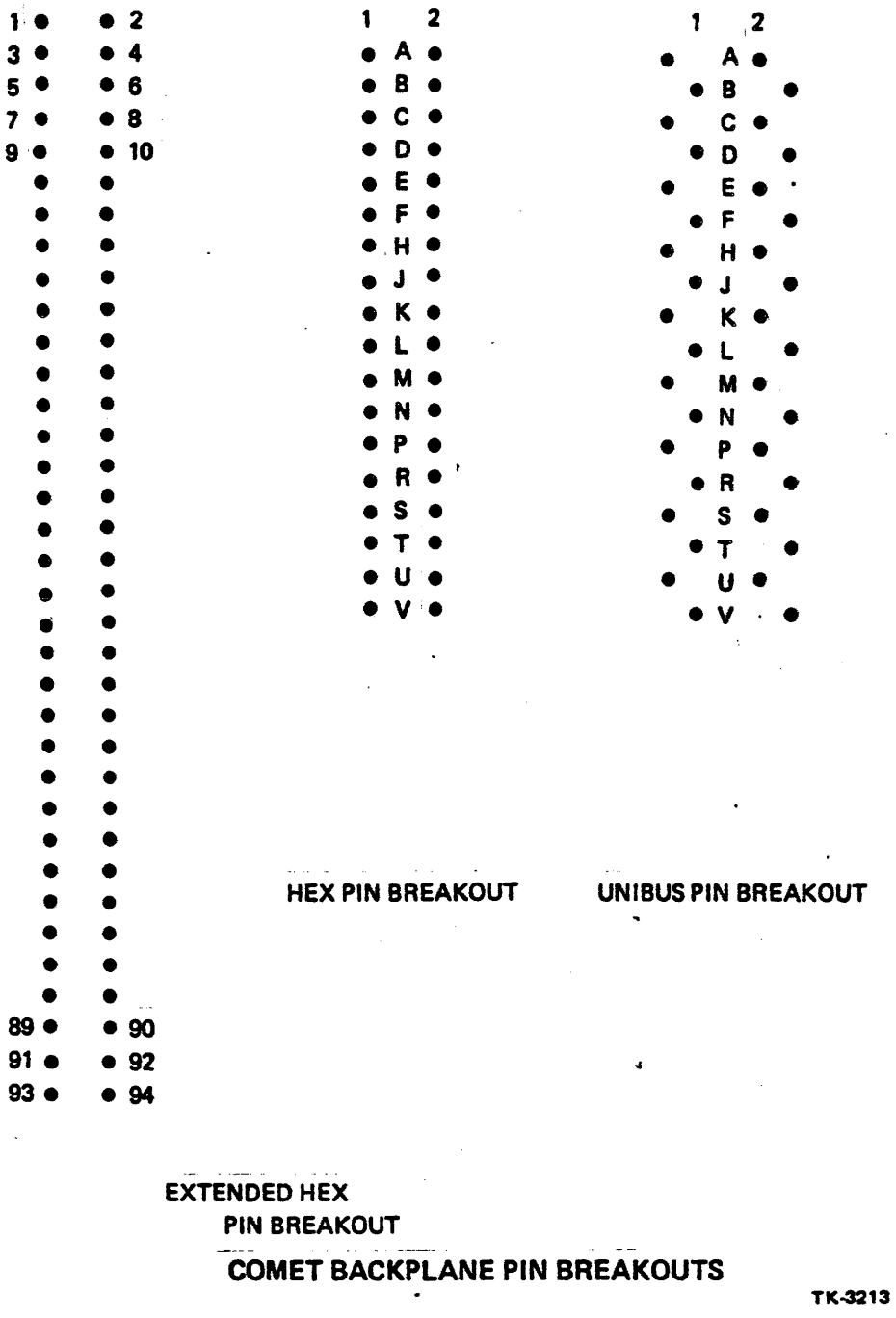
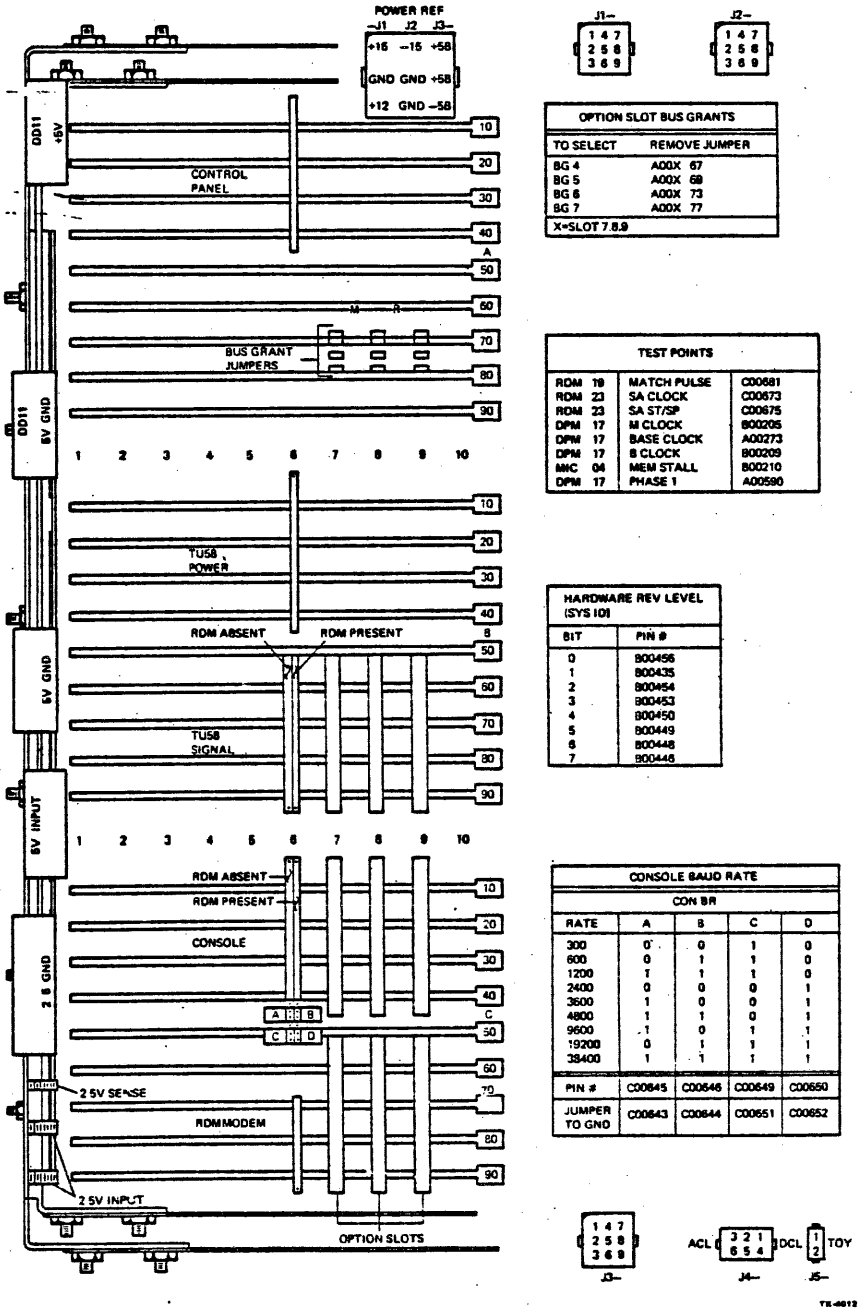


Figure 1-7 11/750 Backplane Pin Breakouts



OPTION SLOT BUS GRANTS	
TO SELECT	REMOVE JUMPER
BG 4	A00X 67
BG 5	A00X 69
BG 6	A00X 73
BG 7	A00X 77
X=SLOT 7,8,9	

TEST POINTS		
RDM 19	MATCH PULSE	C00681
RDM 23	SA CLOCK	C00673
RDM 23	SA ST/SP	C00675
DPM 17	M CLOCK	B00205
DPM 17	BASE CLOCK	A00273
DPM 17	B CLOCK	B00209
MMC 04	MEM STALL	B00210
DPM 17	PHASE 1	A00590

HARDWARE REV LEVEL (SYS I01)	
BIT	PIN #
0	B00456
1	B00455
2	B00454
3	B00453
4	B00450
5	B00449
6	B00448
7	B00446

CONSOLE BAUD RATE				
CON BR				
RATE	A	B	C	D
300	0	0	1	0
600	0	1	1	0
1200	1	1	1	0
2400	0	0	0	1
3600	1	0	0	1
4800	1	1	0	1
9600	1	0	1	1
19200	0	1	1	1
38400	1	1	1	1
PIN #	C00645	C00646	C00649	C00650
JUMPER TO GND	C00643	C00644	C00651	C00652

Figure 1-8 Jumper and Cable Connections

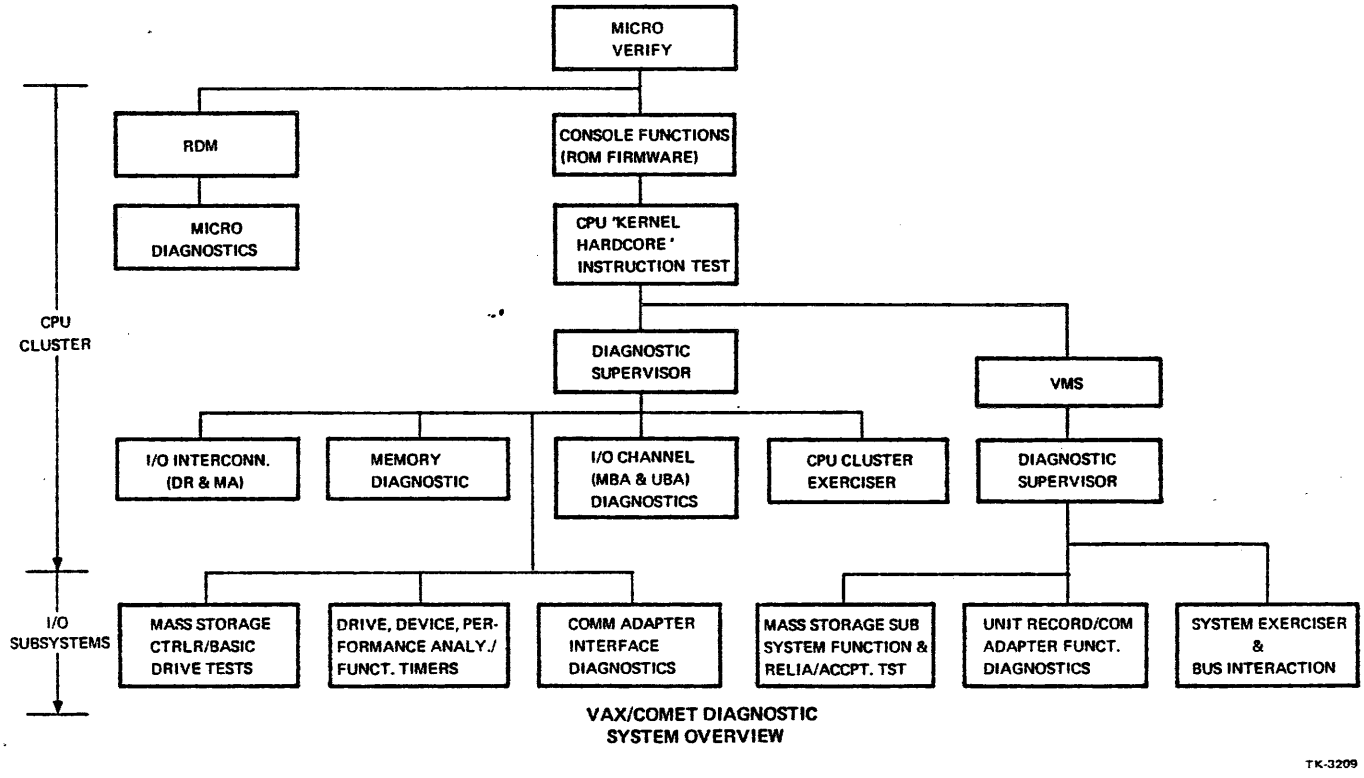


Figure 1-9 VAX Diagnostic Overview

The next section describes the diagnostics available on the 11/750 their different levels of usage. The names and locations of all diagnostics be found on micro fiche under ZZ-EVNDX. There is an other tape [TU58] lable to the field that is not concerned with diagnostics; that being CONSOLE tape which has the BOOT 58 program and BOOT command files locates it. That tape is not listed here and will be discussed later.

This is the beginning of diagnostic overview.

Diagnostics are broken down into five levels, four of which are numbered 1-4. The remaining level is microdiagnostics.

LEVEL 1. These are diagnostics that run under the VMS operating system and not using the diagnostic supervisor. EX. UETP (not a diagnostic, an excersiser).

LEVEL 2. These are diagnostics that run under the diagnostic supervisor while the VMS system is still operating. EX. Reliability and acceptance tests, line printer.

LEVEL 3. These are diagnostics that run under the diagnostic supervisor while the VMS system is not running. The diagnostic supervisor must be running stand alone. EX. UBI DIAGNOSTIC.

LEVEL 4. These are diagnostics that are run stand alone without the diagnostic supervisor or VMS operating. EX. Hardcore instruction.

MICROs These are diagnostics that are loaded from the TU58 and run from the RDM RAM memory. There will be a total of four;

1. DPM micro [data path]
2. MIC micro [memory interconnect]
3. CMC micro [memory controller]
4. FPA micro [floating point]

Of these four only the first two are available as of August 1, 1980.

There is another diagnostic that is run every time the machine is powered up or the Initialize button is pushed. This is called micro verify. This is resident in the machine inside the microcoded CSC

module and checks the basic sanity of the data path and mic module before any other operations are performed. This is discussed in its entirety in a later section.

There are some diagnostics that may be run under level 2 or 3 and should not be thought of as just level 2 or 3. These will be discussed as we reach them.

The following is a list of the diagnostics that are available and which TU58 tape they are distributed on.

The following four tapes are run at the micro level to check the CPU. They are not to be run in their numerical order for troubleshooting purposes. Order for troubleshooting will be discussed later.

```
TU58 TAPE #1: VAX 11/750 MICRO DATA PATH [DPM]
     ECKAA.EXE  MICRODIAGNOSTIC MONITOR [MM FROM NOW ON]
     ECKAB.EXE  MICRODIAGNOSTIC DPM

TU58 TAPE #2: VAX 11/750 MICRO MEMORY INTERCONNECT [MIC]
     ECKAA.EXE  MM
     ECKAC.EXE  MICRODIAGNOSTIC MIC

TU58 TAPE #3:  VAX 11/750 MICRO COMET MEMORY CONTROLLER [CMC]
     ECKAA.EXE  MM
     ECKAD.EXE  MICRODIAGNOSTIC CMC

TU58 TAPE #4:  VAX 11/750 MICRO FLOATING POINT [FPA]
     ECKAA.EXE  MM
     ECKAE.EXE  MICRODIAGNOSTIC FPA
  [TAPE 3 AND 4 NOT RELEASED AS OF AUG. 1ST 1980]
```

The following four tapes are used to test the CPU levels other than MICRO.

```
TU58 TAPE #5: VAX 11/750 CACHE/TB;MEMORY;CLUSTER EXCERSISOR
     ECKAL.EXE  CACHE/TB [BOOTABLE;LEVEL 4]
     ECKAM.EXE  MEMORY DIAGNOSTIC [LEVEL 3]
     ECKAX.EXE  CLUSTER EXCERSISOR [LEVEL 3]

TU58 TAPE #6: VAX 11/750 DW 750 [UBI];DIAGNOSTIC SUPERVISOR
     ESSAA.EXE  DIAGNOSTIC SUPERVISOR [ONLY TAPE TO CONTAIN
                                       THIS BOOTABLE]
     ECCBA.EXE  UBI DIAGNOSTIC [LEVEL 3]
```

TU58 TAPE #7: VAX 11/750 HARDCORE INSTRUCTION
EVKAA.EXE HARDCORE INSTRUCTION [BOOTABLE;LEVEL 4]

TU58 TAPE #8: 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]

Remaining tapes that follow are to be used to test options available on the 11/750. These will be [as the previous tapes #7 and 8] the same diagnostics that are run on the 11/780. To determine which level the diagnostics will be run at you will need to read the associated manual.

TU58 TAPE #9: 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

TU58 TAPE #10: KMC11/DMC11/DZ11
EVDMA VAX M8203 REPAIR LEVEL
EVDXA VAX COMM IOP REPAIR LEVEL
EVDAA VAX DZ11 8 LINE ASYNC MUX

TU58 TAPE #11: RK611 DIAGNOSTICS #1
EVREA VAX RK611 DIAGNOSTIC PART A
EVREB VAX RK611 DIAGNOSTIC PART B

TU58 TAPE #12: RK611 DIAGNOSTICS #2
EVREC VAX RK611 DIAGNOSTIC PART C
EVRED VAX RK611 DIAGNOSTIC PART D
EVREE VAX RK611 DIAGNOSTIC PART E

TU58 TAPE #13: RK611 DIAGNOSTICS #3
EVREF VAX RK06/07 DRIVE FUNCTION TEST PART 1
EVREG VAX RK06/07 DRIVE FUNCTION TEST PART 2

TU58 TAPE #14: RM03/RM05
EVRDA VAX RM03/RM05/RM80 DISKLESS
EVRDB VAX RM03/RM05 FUNCTIONAL TEST

TU58 TAPE #15: TS11 DIAGNOSTICS
EVQTS VAX LOADABLE DRIVER FOR TS11/TS04
EVMAA VAX TM03/TE16/TU45
EVMAD VAX TS11 SUBSYSTEM REPAIR

TU58 TAPE #16: RLØ2 SUBSYSTEM FUNCTIONAL DIAGNOSTICS

EVRFA	VAX RLØ2 SUBSYSTEM FUNCTIONAL DIAGNOSTICS
EVRGA	VAX RM8Ø FORMATTER
EVRGB	VAX RM8Ø FUNCTIONAL DIAGNOSTICS

As of August 1, 198Ø the above were the only diagnostics proven compatible with both the 11/78Ø and 11/75Ø. The following are the remaining diagnostics that are planned.

POSSIBLE ONE TAPE:

ESDRB	VAX DR11W DIAGNOSTIC
ESDRE	VAX DR11W REPAIR LEVEL

POSSIBLE 2ND TAPE:

ESDBA	VAX M82Ø1/2 REPAIR LEVEL DIAGNOSTIC
ESDBB	VAX DMC11 EXCERSISOR PROGRAM

POSSIBLE 3RD TAPE:

ESDUP	VAX DUP11 REPAIR LEVEL PART 1
ESDUQ	VAX DUP11 REPAIR LEVEL PART 2

Please note that all diagnostic (not including the micro diag. or EVKAA and ECKAL) that relate to your system will be sent with the system pack as part of the system on whatever medium your VMS is incorporated in.

VAX 11-/750 LEVEL II

Console Command Language and Bootstrap Process

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

OUTLINE

II. A. Console Command Language

1. Control Characters
2. Console Command Symbols
3. Console Commands
4. Errors and Illegal Characters

B. Bootstrap Process

1. Definition
2. Different Boot Methods
3. How Boot is Accomplished
4. Boot 58
5. Automatic Boot
6. System Shutdown
7. Copy Console Device Files

INTRODUCTION

This lesson introduces the student to the VAX console commands needed to communicate with the VAX-11/750. After becoming familiar with the commands and their functions in the classroom, a lab session will be provided to utilize each command. The lab session will allow the student to initialize the system and perform deposits and examinations to various registers and memory locations.

This lesson also covers the VAX-11/750 bootstrap process. By using flowcharts, the process will be covered from device selection to error indications. Once the process has been covered by lecture, a lab session will be utilized to reinforce the concepts and to demonstrate error conditions.

OBJECTIVES

1. Using console commands, initialize the system.
2. Using console commands, deposit data to a register.
3. Using console commands, examine data in a register.
4. Boot the system.
5. Given the console printout indicating a boot failure, clear the fault by locating the problem.

SAMPLE TEST ITEM

This console printout occurred while booting the system with FPS1 set to boot.

```
% %
XXXXXXXX  13
>>>
```

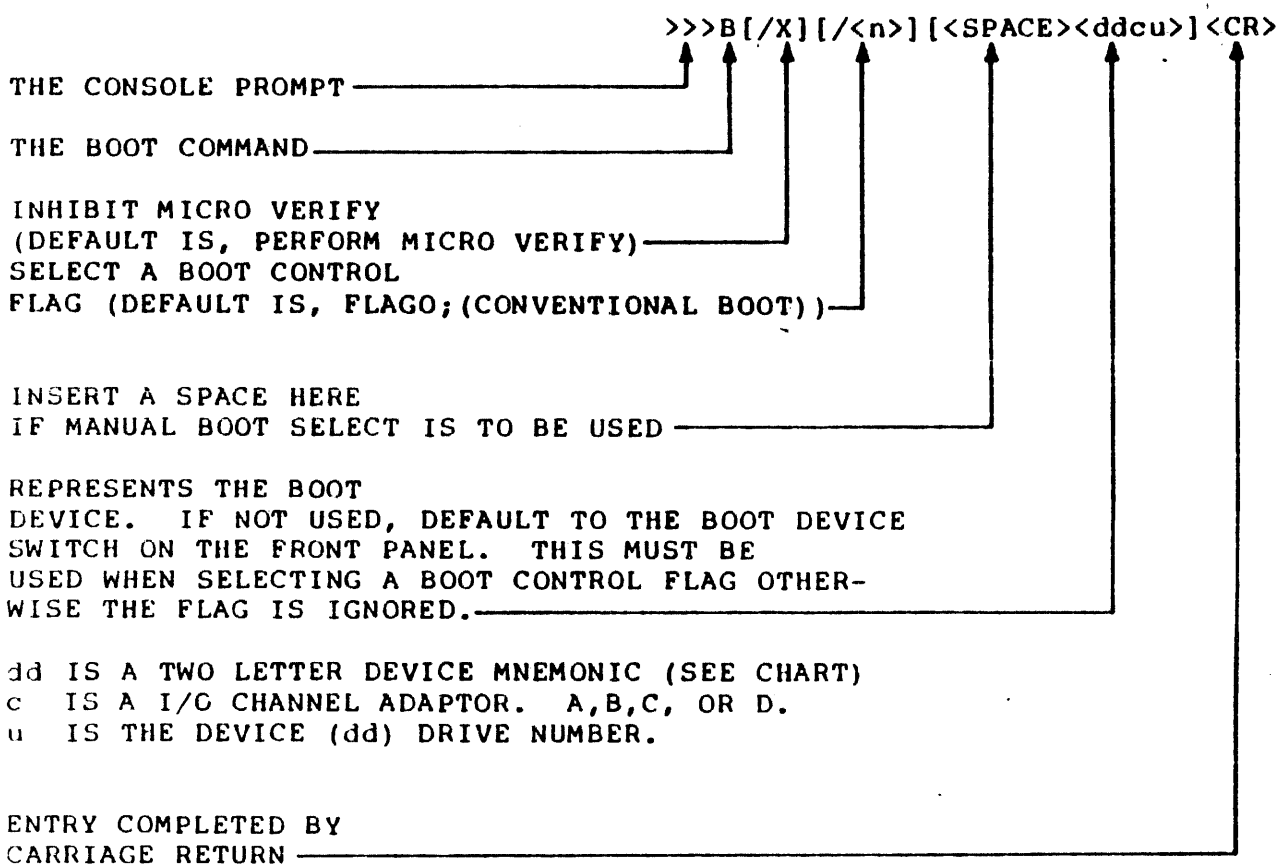
What is indicated to the operator?

a. A 64K bytes of good memory not found
b. A nonexistent boot ROM
c. A HALT was executed
d. Wrong Rev. level

RESOURCES

1. VAX 11/750 RDM Maintenance Card
2. VAX 11/750 Diagnostic System Overview Manual

THE BOOT COMMAND



dd	
MNEMONIC	DEVICE
DL	RL02
DM	RK06/7
DB	RP04/5/6
DR	RM03
DD	TU58

BOOT CONTROL FLAG	FUNCTION	/<N>
Ø	CONVERSATIONAL BOOT.	1
1	DEBUG	2
2	INITIAL BREAKPOINT	4
3	NOT USED WITH VAX 11/75Ø	8
4	DIAGNOSTIC BOOT	1Ø
5	BOOTSTRAP BREAKPOINT	2Ø
6	IMAGE HEADER	4Ø
7	MEMORY TEST INHIBIT	8Ø
8	FILE NAME	1ØØ
9	HALT BEFORE TRANSFER	2ØØ

BOOT CONTROL FLAG FUNCTIONS

THE DEPOSIT COMMAND

>>>D[<qualifier-list>][<space><address>]<space><data><cr>

CONSOLE PROMPT

DEPOSIT COMMAND

SIZE & SPACE

/B	/V
/W	/P
/L	/I
	/G

TO SELECT A
DEPOSIT ADDRESS
OTHER THAN Ø

<nnnnnnnn> -- HEX ADDRESS
<*> -- LAST LOCATION
<P> -- PSL
<+> -- NEXT LOCATION

YOU MUST SELECT A
HEX VALUE (1-8 DIGITS)

OPERATION COMPLETED

THE BOOT/BOOTSTRAP AS DEFINED BY THE DEC DICTIONARY - PG 28

boot (boots, booting, booted)* v. (See also bootstrap.) To bring a device or system to a defined state where it can operate on its own. EXAMPLE(S): The operator boots the system before starting operation.

boot (boots)* n. A protective housing, usually made from a resilient material, used to protect connectors or other terminals from moisture. EXAMPLE(S) : Pull the boot up over the plug to make the connection waterproof.

bootstrap (bootstraps, bootstrapping, bootstrapped)* v. (See also boot.) To bring a device or system to a defined state where it can operate on its own. EXAMPLE(S): You must bootstrap the system before logging on.

bootstrap (bootstraps)* n. A technique or device designed to bring a system or device into a desired state by means of its own action, e.g., a machine routine whose first few instructions are sufficient to bring the rest of itself into the computer from an input device. EXAMPLE(S): Using the bootstrap saves time.

CONSOLE MICROCODE EXAMINES THE BOOT DEVICE AND POWER ON ACTION SWITCHES ON THE FRONT PANEL;

- When you initially apply power by turning the front panel keyswitch
- When recovering from a power failure
- When the operator pushes the front panel initialize switch.
- After a software "CRASH"

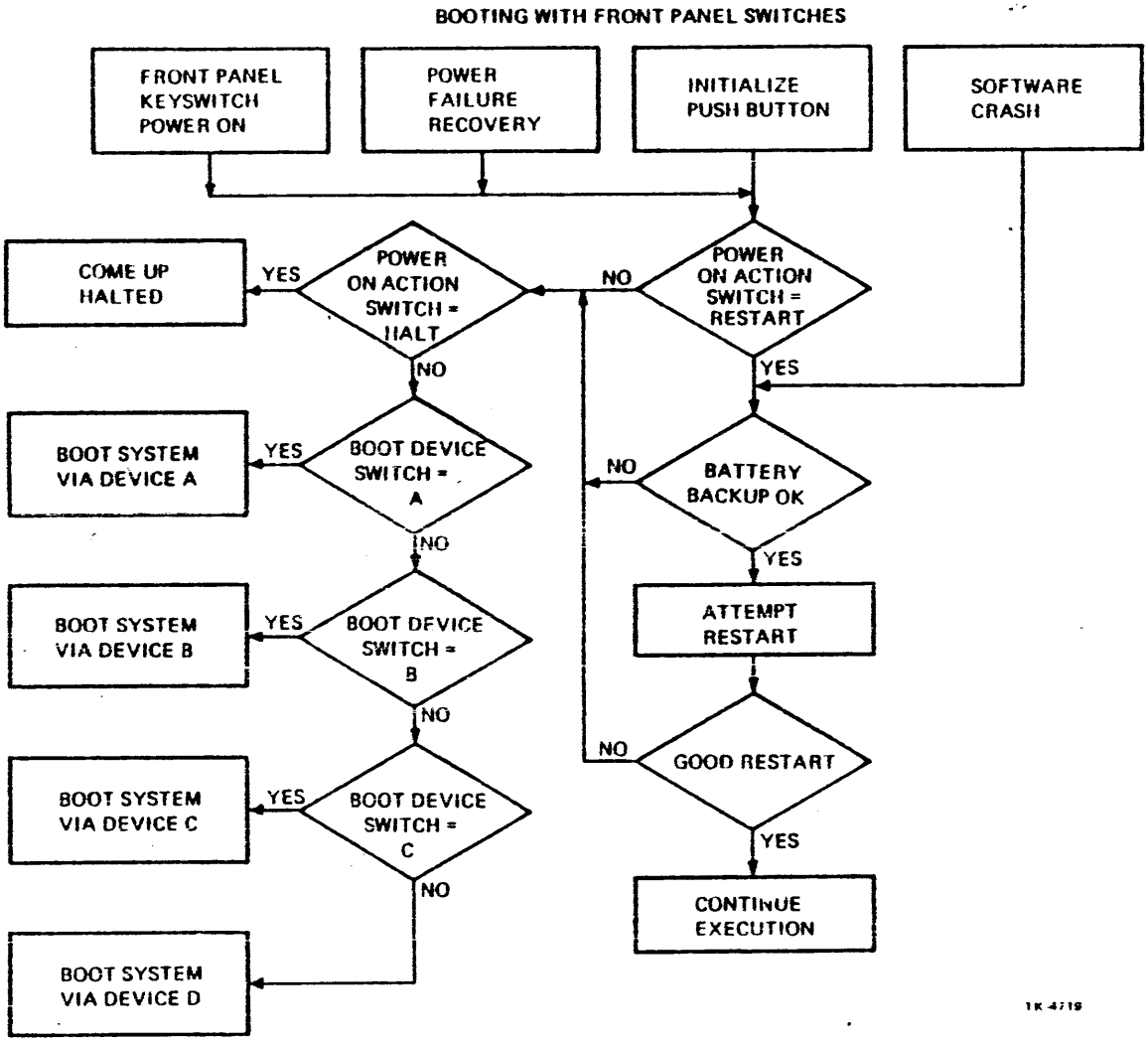

BOOTING WITH FRONT PANEL SWITCHES
FRONT PANEL KEYSWITCH POWER ON
POWER FAILURE RECOVERY
INITIALIZE PUSH BUTTON
SOFTWARE CRASH
POWER ON ACTION SWITCH = RESTART
NO
YES
COME UP HALTED
YES
POWER ON ACTION SWITCH = HALT
NO
BOOT SYSTEM VIA DEVICE A
YES
BOOT DEVICE SWITCH = A
NO
BATTERY BACKUP OK
NO
YES
BOOT SYSTEM VIA DEVICE B
YES
BOOT DEVICE SWITCH = B
NO
ATTEMPT RESTART
BOOT SYSTEM VIA DEVICE C
YES
BOOT DEVICE SWITCH = C
NO
GOOD RESTART
NO
YES
BOOT SYSTEM VIA DEVICE D
CONTINUE EXECUTION
1K-4719

Figure 2-1

SOFTWARE BOOT CONTROL FLAGS

(1 of 2)

Flag	Hex Value	Function
Ø	1	Conversational boot. At various points in the system boot procedure, parameters and other inputs will be solicited from the console.
1	2	Debug. This flag is passed through to VMS and causes the code for the executive debugger to be included in the running system.
2	4	Initial breakpoint. If this flag is set, and the executive debugger code is included (flag bit 1), then a breakpoint will occur immediately after the exec enables mapping.
3	8	Not used on the VAX-11/75Ø.
4	1Ø	Diagnostic boot. This flag causes a boot by file name for the diagnostic supervisor.

SOFTWARE BOOT CONTROL FLAGS

(2 of 2)

Flag	Hex Value	Function
5	20	Bootstrap breakpoint. This flag causes the bootstrap to stop at a breakpoint after performing necessary initialization.
6	40	Image Header. If this flag is set, the transfer address from the image header of the boot file will be used. Otherwise control will transfer to the first byte of the boot file.
7	80	Memory test inhibit. This flag inhibits the testing of memory during bootstrapping.
8	100	File name. Causes the bootstrap to solicit the name of the boot file.
9	200	Halt before transfer. Causes a HALT instruction to be executed prior to the transfer to the secondary boot file. This option is useful for debugging purposes.

BOOT DEVICE CODES (ddcu)

DEVICE CODE (dd)*	DEVICE TYPE
DL	RLØ2
DM	RKØ6/Ø7
DB	RPØ4/Ø5/Ø6
DR	RMØ3/RPØ7
DD	TU58

* Identifies the device that is storing the boot block.

CHANNELS ADAPTER (C)

A, B, C, D — To which port is the Device (dd) channeled to.

DRIVE NUMBER (u)

Ø, 1 — on which drive of our device (dd) is our boot block located.

CONSOLE COMMAND ERROR CODES

If an illegal console command is attempted or command is aborted because of a microtrap or some other condition a two digit error code is typed out and the console waits for new input. For example...

```
>>>E P<CR>          !Examine PSL
>>>E<CR>            !Implies Examine Next Location, this is illegal.
?11                 !Question Mark and error code is typed by console
>>>                 !At this point ready for new command
```

Error Codes

20= Deposit or Examine of Memory Failed (Access Violation, Translation not valid, Bus Error, TB Parity Error, or Control Store Parity E

11= Illegal access of an ipr

30= Apt Loading Checksum error

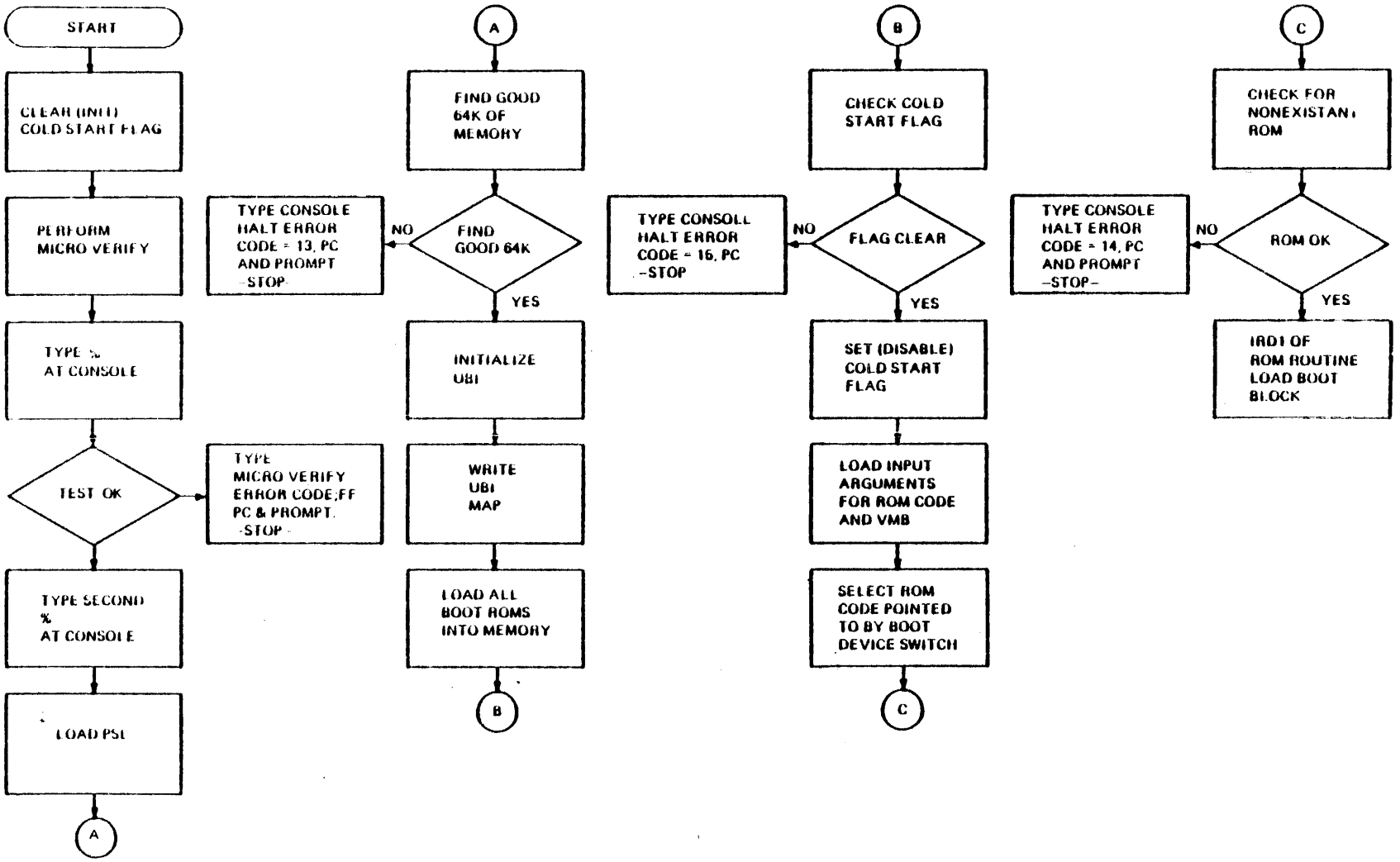
33= Attempt to Boot from unknown Device type (DM,DL,DO)

34= Boot Device Controller not "A","B","C", OR "D"

ROM STARTING ADDRESSES

DEVICE ROM	STARTING ADDRESS
A	FAØ2
B	FBØ2
C	FCØ2
D	FDØ2

CONSOLE SUBSYSTEM ACTION ON BOOT



INPUT ARGUMENTS

The general registers receive the input arguments from the console subsystem.

R1 - system bus address of a Massbus adapter (MBAØ unless otherwise specified in the Boot command).

R2 - physical address of the Unibus I/O page associated with a Unibus adapter (UBIØ unless otherwise specified in the Boot command).

R3 - device unit number (Ø unless otherwise specified in the Boot command).

R5 - software boot control flags (Ø unless otherwise specified in the Boot command).

SP - <base address + ^X2ØØ> of the 64K bytes of good memory.

C(SP)- transfer address of the boot block code.

FUNCTIONS AVAILABLE UNDER BOOT 58

- Load and start level 4 diagnostic programs.
- Bootstrap from the Massbus adapter
- Bootstrap from a disk whose boot block is bad.
- Bootstrap from a disk whose error rate prohibits ROM and boot block loading of a primary bootstrap.
- Boot the diagnostic supervisor instead of VMS.
- Deposit and examine data in physical memory, general registers, and internal processor registers.
- Load and start a program from a magtape drive on a Massbus.
- Store and invoke indirect command files on the TU58 cartridge to perform any of the above functions automatically as well as interactively.

POWER UP AND BOOT ERROR REPORTS

xxxxxxxx 13 >>>	This indicates that a good 64KB section of memory was not found and return to console mode
xxxxxxxx 14 >>>	This indicates a failure or nonexistance of the boot ROM
xxxxxxxx Ø6	If a halt instruction is executed after typing a console boot command, this indicates a failure of the read of logical block Ø from the selected boot device, the PC should be equal to the base address of the first good 64KB of memory plus FX16 for TU58 or FX2Ø for RKØ6. This failure occurs in the Boot ROM routine.

VMB PRIMARY BOOT FAILURES

BOOT is the program name for VMB.EXE
The "F" indicates a fatal error and the type of error is reported.

%BOOT-F-Unknown processor	This indicates that CPU is not a Comet or 11/780, check SID register for proper jumpering in the CPU type field on the Backplane.
%BOOT-F-Unexpected Exception	This indicates that one of the following exceptions occurred. 1. Access Violation 2. Breakpoint Opcode 3. Reserved Operand 4. TBit Trap 5. Page Fault (TNV)
%BOOT-F-Unexpected Machine Check	This indicates some sort of machine Check occurred. Check all adaptors using console examine and deposit commands. Probably a timeout.
%BOOT-F-Nonexistant Drive	Self explanatory, Check DEFBOO.CMD on 11/780 and insure system disk is drive being booted.
%BOOT-F-Unable to locate BOOT file	VMB can't find [SYSEXE]SYSBOOT.EXE or if bit 4 in R5 is set, VMB can't find [SYSMAINT]DIAGBOOT.EXE
%BOOT-F-Bootfile not contiguous	Indicates that [SYSEXE]SYSBOOT.EXE or [SYSMAINT]DIAGBOOT.EXE is not contiguous on system disk. Recopy or rebuild
%BOOT-F-I/O error reading boot file	Indicates problem reading boot file from disk by $QIO service (VMS System Service).

VAX-11/750 LEVEL II

System Overview

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

This lesson is developed to give you a basic understanding of location of Gate Arrays in 11/750 prints and overall understanding of how each board (DPM, MIC, UBI, and CCS) is laid out. All of the data given out in this section will be reiterated when each board is gone over in detail. You should start to form concepts of how the machine works functionally and how the prints are set up. There will be very little in the Student Guide so listen, take notes in your prints or Functional Block.

OBJECTIVES

The student will be able to locate all gate arrays in the prints relating to the four basic LPU boards, DFM, MIC, UBI and CCS.

The student will be able to take a MOVL LONG instruction and follow the path of the data from beginning to end.

SAMPLE TEST ITEM

The access control violation chip is located on which board?

1. DPM
2. MIC
3. UBI
4. CCS

RESOURCES

11/750 Print Set
11/750 Functional Block

A. The CPU Overview

1. Board common to all CPUs four (4) each

a. Unibus Interface (UBI Module - Slot 4 of extended hex section)

1) TU58 Data in and out
2) Console Data in and out
3) Handles interrupts
4) Interfaces Unibus data and CPU data with each other

b. Data Path Module (DPM - Slot 2 of extended hex section)

1) Contains the arithmetic logic
2) Contains the rotator logic
3) Houses the Scratch Pad logic (Registers)
4) Also houses the Microsequencer logic

c. Memory Interconnect (MIC Module - Slot 3 of the extended hex section)

1) Contains the address logic (PC)
2) Houses the Translation Buffer which translates virtual addresses to physical addresses
3) Cache
4) Contains the Data Routing and alignment which handles the routing of data in and out of or to and from memory and the data path.

d. CPU Control Store (CCS Module - Slot 5 of extended hex section)

1) Contains the Control Store ROMs for the Microcode

2) Houses the optional snap on WCS Module

B. Component Analysis

1. CPU Control Store (CCS)

a. 6K x 8Ø Bits, no gate arrays

2. Data Path (DDPM) - 22 gate arrays

a. Gate Array Chips

1) The microsequencer (MSQ) - Sequences the CPU microcode that controls most operations (NOT SHOWN AS ONE CHIP ON BLOCK SHOW IN PRINTS)

2) Practically Half the Buts (PHB) - Contains some of the bits of the PSL, the status flags and the step counter. It also contains the logic to generate half of the but micro orders.

3) Service Arbitration and Clock (SAC) - Deals with the IRD counter, service arbitration, and the system clock.

4) Condition Code Chip (CCC) - Deals with condition codes. It determines the condition codes for both VAX and compatability mode instructions, stores the PSL bits <FU,IV,DV,N,Z,V,C>, reads the bits out at Ucode request.

5) Instruction Register Decode (IRD) - Handles the IR Decode. It receives an opcode and operand specifier from the execution buffer (XB), decodes it and creates the signals needed by the microsequencer to process the appropriate routine.

6) Super Rotator Control (SRK) - It controls the functions of the Super Rotator (SR). The info it needs to control the SR comes from the 6 bit ROT field of the microcode. (NOT ON BLOCK IN TOTAL, PRINTS.)

   a. SPK chip contains the S and P latches and their associated mux in and out. Controls the super rotator via the ROT field of the microword and certain Wbus inputs.

7) Super Rotator Multiplex (SRM) - 4 ea - Perform 64 different operations via control of the SRK chip.

8) Scratch Pad Addressing (SPA) - Controls the operating of the 64 scratch pad registers, and it provides a mechanism to undo the auto decrementing and auto incrementing of the general purpose registers.

9) Timed Operation Control (TOK) - Implements the architecturally defined programmable interval time clock.

1Ø) Carry Look Ahead (CLA) - An array of combinational logic used to propagate and generate carries for up to 8 ALU slices. (NOT ON BLOCK, SHOW IN PRINTS.)

11) Arithmetic and Logic Control (ALK) -. (NOT ON BLOCK, SHOW IN PRINTS.)

a) Reencodes the ALP control field of the microword for special functions.

b) Controls the carry input and shift inputs for the ALP chips.

c) Decodes the scratch pad write enable.

d) Decodes miscellaneous signals.

12) Arithmetic Logic Processors (ALP) - 8 ea - Each chip is 4 bits wide. They form the circuit that performs the majority of the data manipulating when executing macro instructions.

3. Memory Interconnect Module (MIC) 18 Gate Arrays

a. 1) Memory Data Registers (MDRs) - 8 ea - Major portion of the data routing and alignment circuit. They receive and hold data in/out to/from the Mbus.

2) Prefetch Control (PRK) chip - Prefetch 8 bytes of instruction data starting with the PC address and replace used bytes as execution progresses.

3) Address (ADD) chips - Contain the VA, PC, PC backup and VA save circuits.

4) Address Control (ADK) chip - Is the control for the address logic and also works in conjunction with the prefetch control and memory data regs.

5) Access Violation (ACV) chip - Besides detecting access violations it monitors and detects.

a. Control store parity errors
b. FPA reserved operands
c. Unaligned data, including unibus data.
d. Crossing of page boundaries.

It then generates the appropriate Utrap signals to the Microtrap chip (UTR).

6) Microtrap (UTR) chip - Monitors machine conditions that can cause a microtrap.

7) The Cache control (CAK) chip - Controls the enabling and disabling of cache, controls the transfer of data to/from the MDR chips. Works in conjunction with the CMK chip to invalidate cache on CMI writes.

8) CPU Memory Interconnect Control (CMK) - Monitors and transmits control signals to/from the CMI bus. (Busy and HOLD.) Stalls the microcode for certain conditions.

4. Unibus Interface Module (UBI) 8 Gate Arrays

a. 1. The TU58 Interface consists of a Gate Array Chip (CON) and some associated logic that allows communication between the CPU.

2. The console Interface consists of a Gate Array chip (CON) and some associated logic that enables communication between the CPU and its console.

3. The interrupts circuit consists of a Gate Array chip (INT) and some associated logic that enable the handling of interrupts.

4. The Unibus interface consists of five (5) Gate Array chips, a ROM and Unibus Map.

a. The Unibus Data Path (UDP) chips make up the data path for the unibus interface, four (4) ea.

1) Areas that represent UDP Chips.

   a. 3 buffered data paths for data and addresses
   b. 1 direct data path for data and addresses
   c. Byte swapping and rotating circuits to align data

b. The Unibus Data Path Control - Controls UDC chips and Microcode (UCN) chip.

c. Unibus map for translating Unibus addresses to CMI addresses.

d. ROM for controlling UBI functions independent of CPU. (Note circles controlling UDP Chips are fields from ROM.)

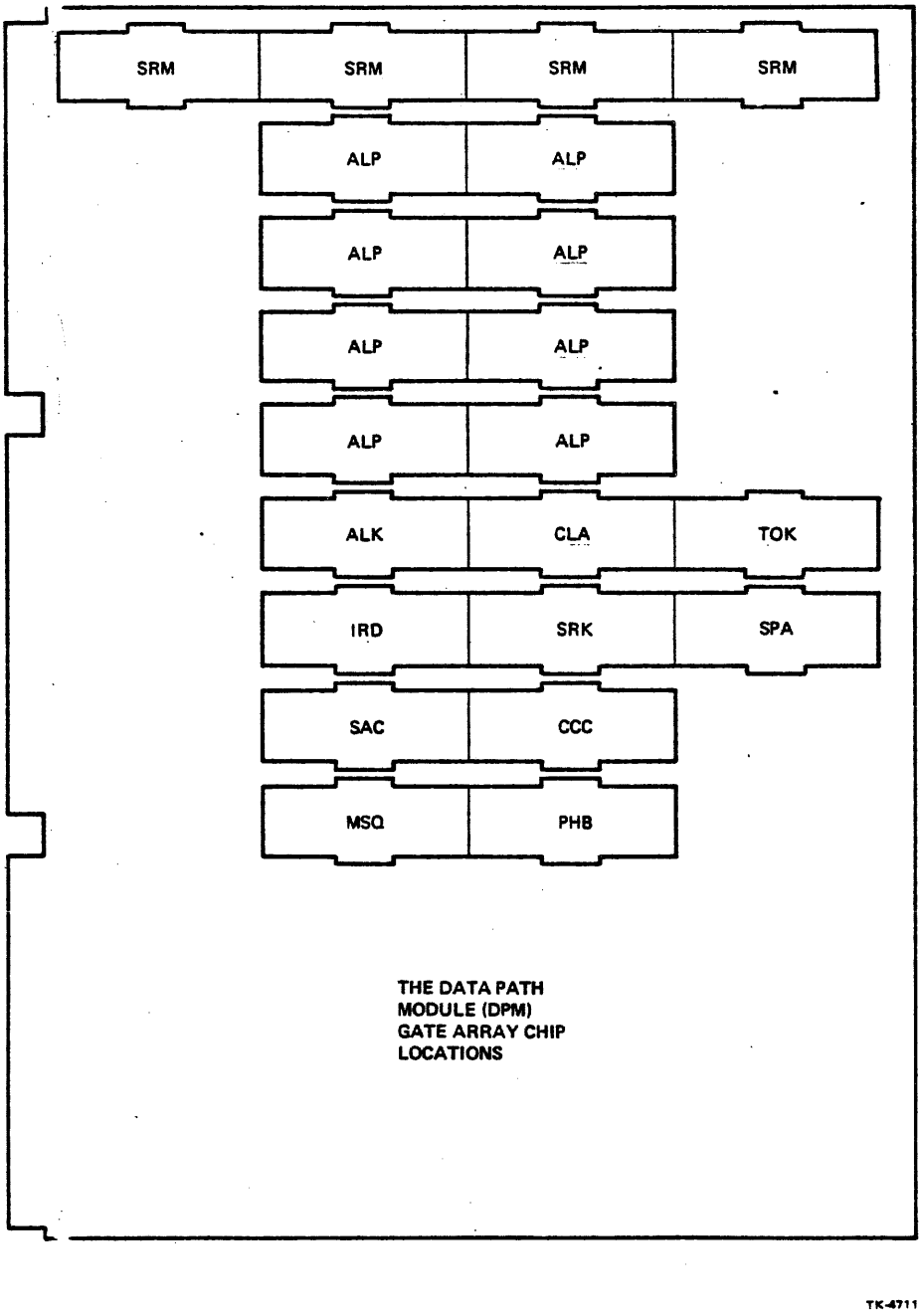


Figure 4-1 DPM Gate Array Locations

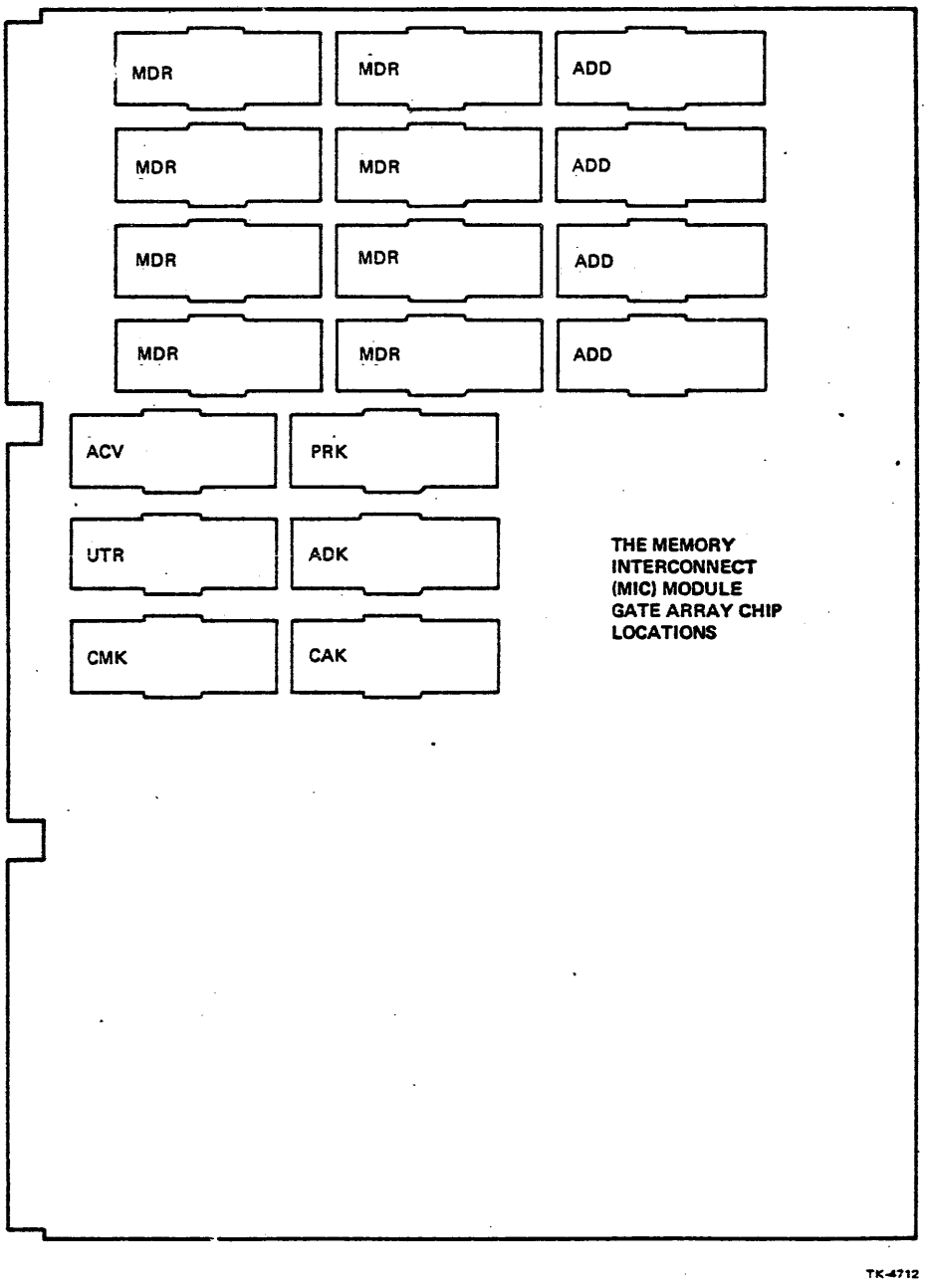


Figure 4-2 MIC Gate Array Locations

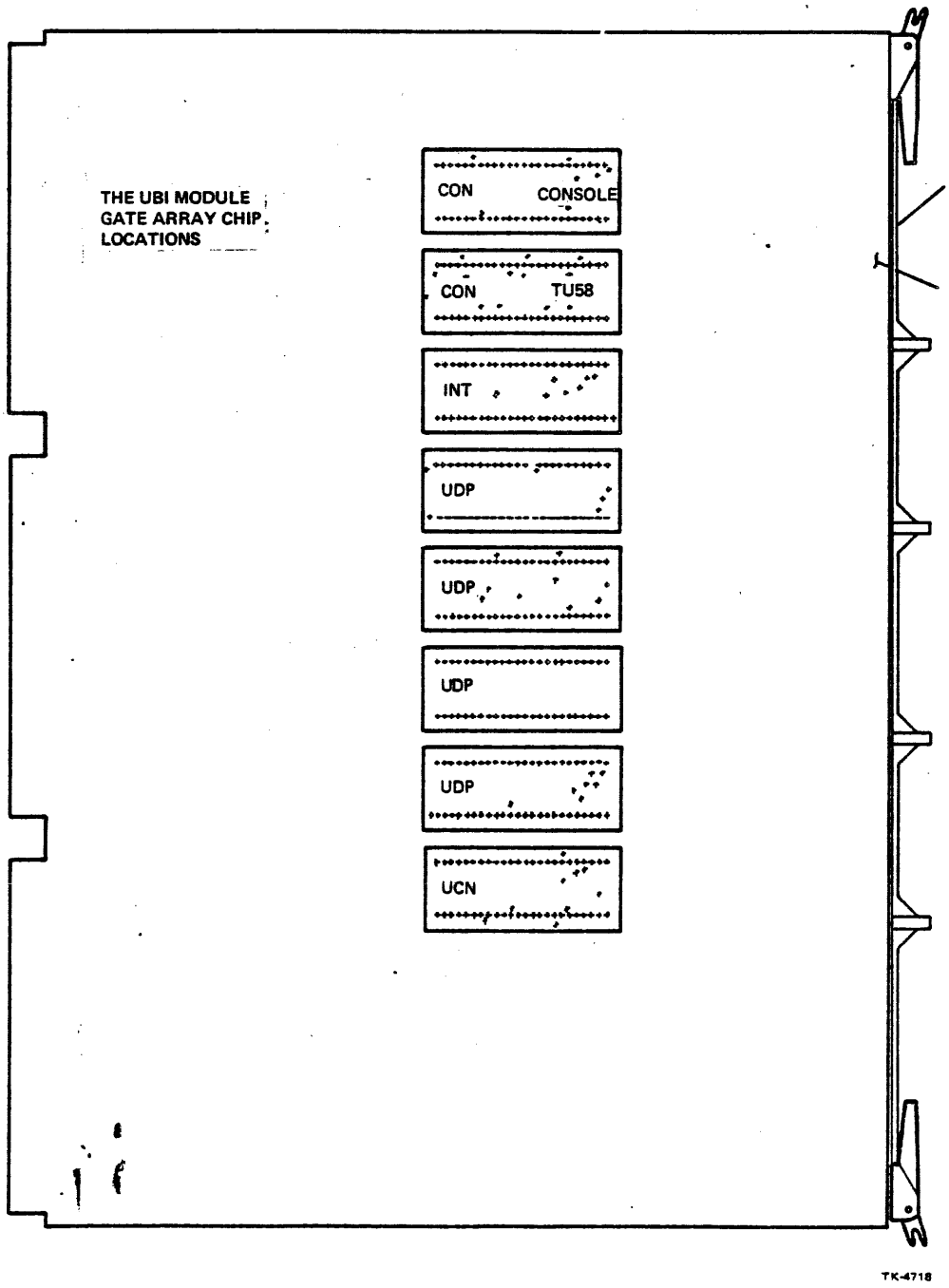


Figure 4-3 UBI Gate Array Locations

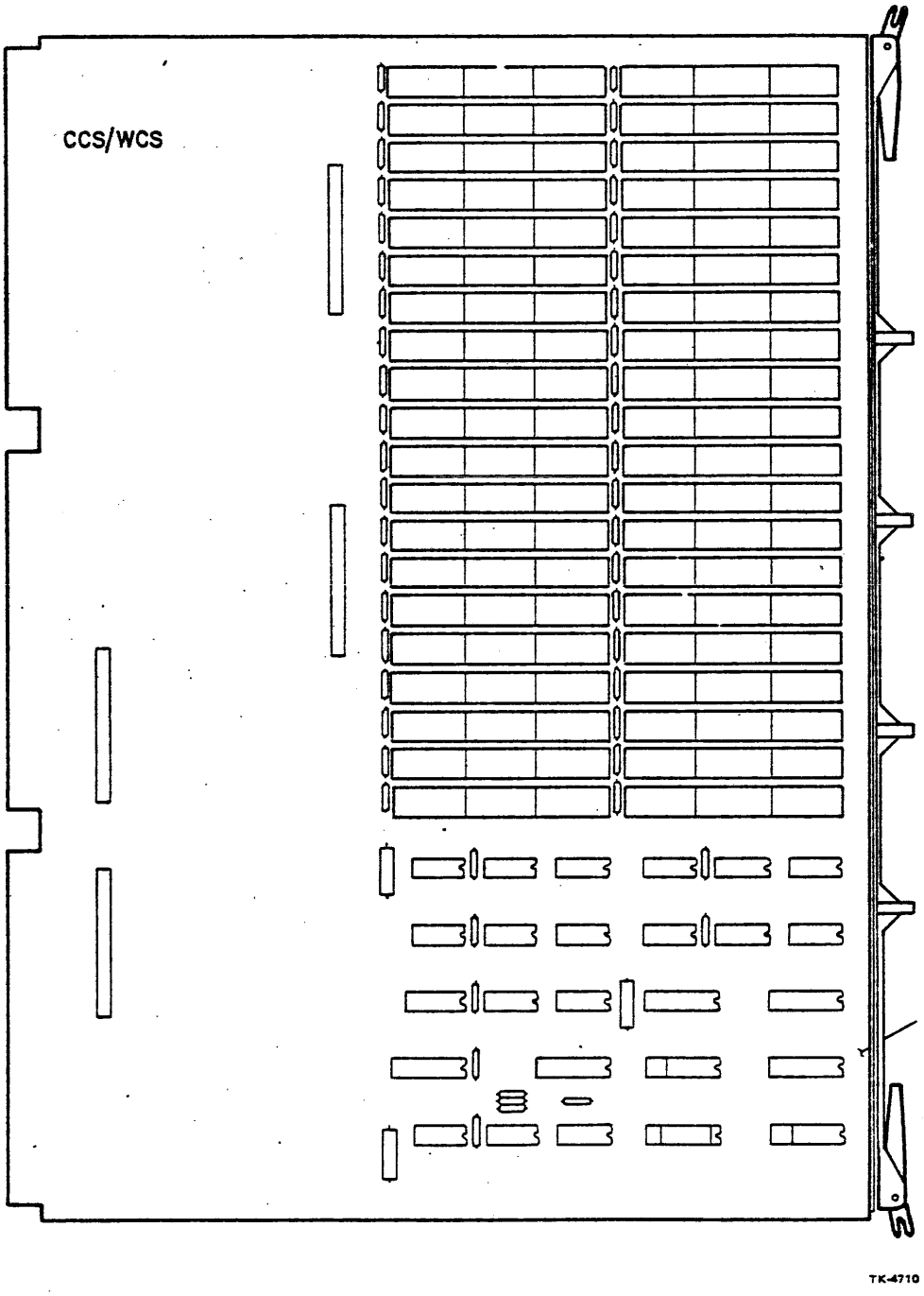


Figure 4-4 CCS Module

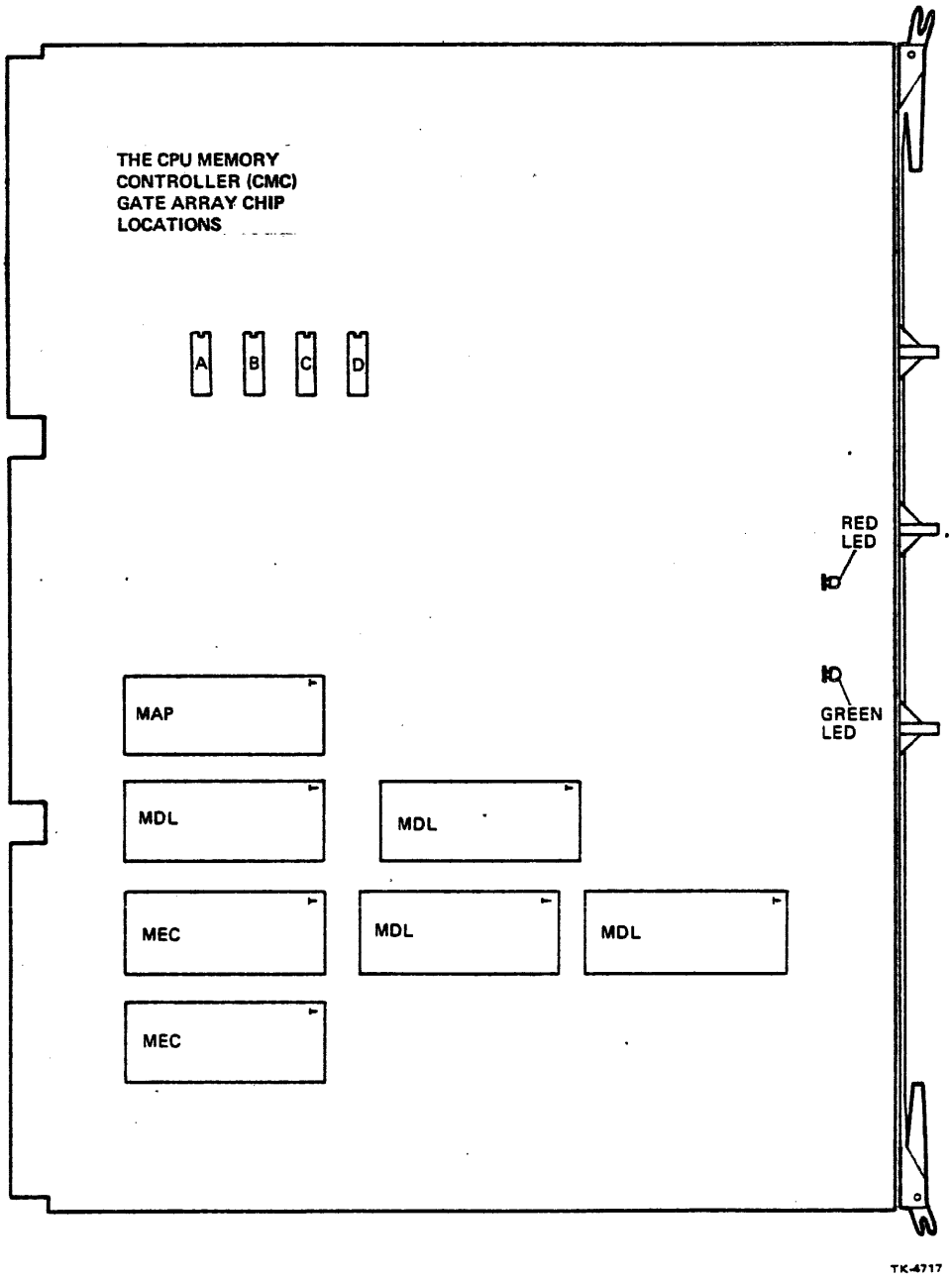


Figure 4-5 Memory Controller

VAX-11/750 LEVEL II

Programming

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

OUTLINE

V. PROGRAMMING

A. VAX Instruction Set
   1. Operand and Instruction Formats
   2. VAX Addressing Modes
   3. VAX Integer and Logical Instructions
   4. VAX Branching Instructions

B. Laboratory Exercise 3
   Write a routine to convert packed hex data to an ASCII string utilizing VAX 11 Programming Tools

C. VAX Instruction Set
   1. VAX Floating-Point Instructions
   2. VAX Subroutine and Procedure Calling Instructions

D. Laboratory Exercise 4
   Modify routine written in previous lab to be called as a procedure utilizing a CALLS or CALLG instruction

E. VAX Instruction Set
   1. VAX Character String, Packed Decimal and Field Instructions
   2. VAX Privileged Instructions
   3. Programming Examples

F. Laboratory Exercise 5
   Write a standalone program for the Comet CPU to communicate between the local console and a terminal on the Unibus

G. Summary

OBJECTIVES

Utilizing the VAX-11/780 Programming card, Architecture handbook and any class notes, write two (2) programs that perform the following:

a) Packed hex to ASCII conversion
b) 2 way communication between CPU Console and a terminal on the Unibus.

Load and execute the previously written programs and the instructor will verify operation.

SAMPLE TEST ITEM

Using the Comet system, load and execute the two (2) programs previously written in class. The instructor will verify proper operation by witnessing program execution.

LAB EXERCISE

a) Utilizing the VAX program development tools, write a packed hex to ASCII conversion routine in VAX-11 Macrocode.

b) Again, utilizing the same VAX programming tools, write a standalone program to communicate between the console terminal and a Unibus terminal and copy it on to a TU58 tape cartridge for console loading.

RESOURCES

VAX-11/780 Architecture Handbook
VAX-11/780 Software Handbook
Terminals and Communications Handbook
Program Development Listing

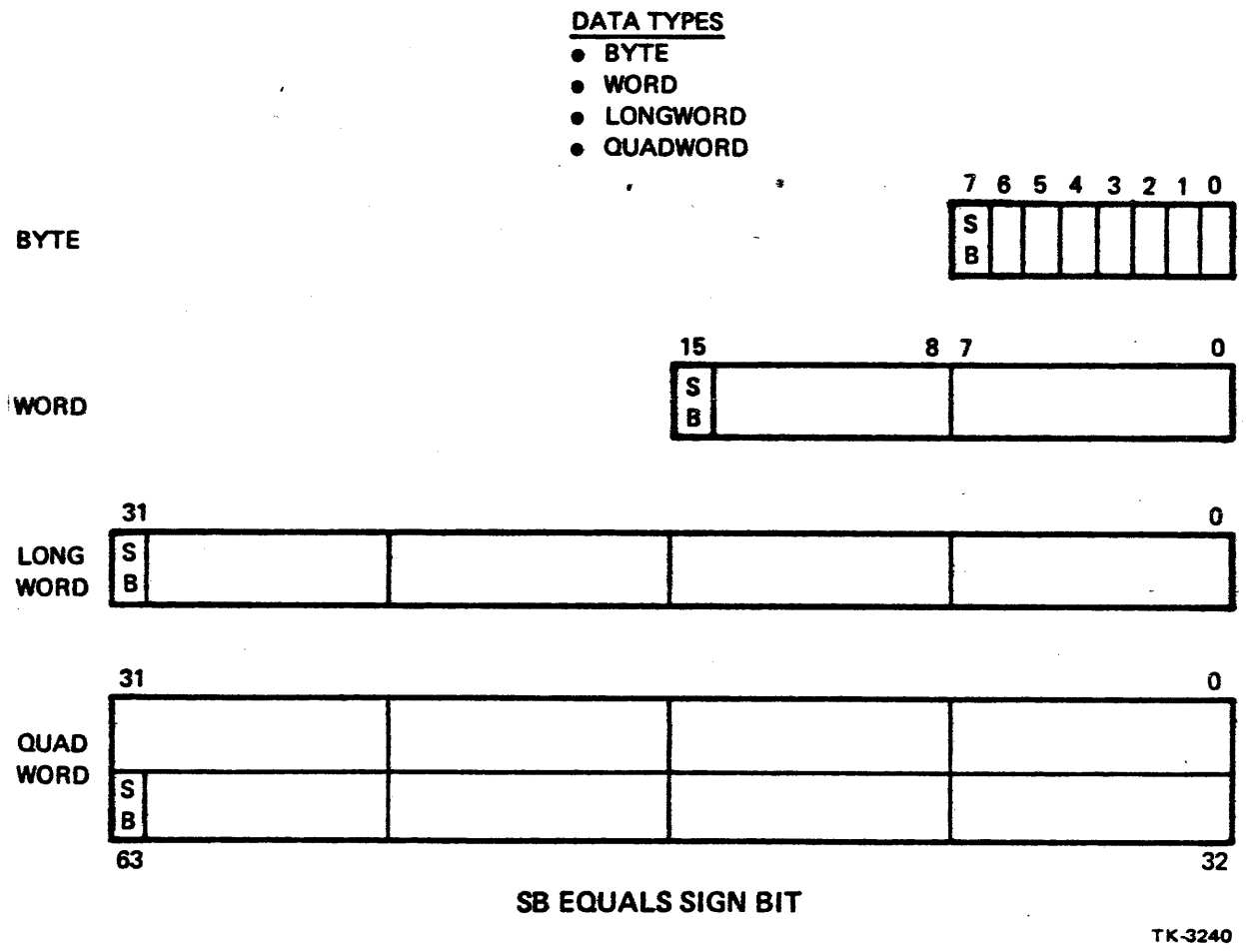

DATA TYPES
• BYTE
• WORD
• LONGWORD
• QUADWORD
BYTE
7 6 5 4 3 2 1 0
S B
WORD
15 8 7 0
S B
LONG WORD
31 0
S B
QUAD WORD
31 0
S B
63 32
SB EQUALS SIGN BIT
TK-3240

Figure 5-1

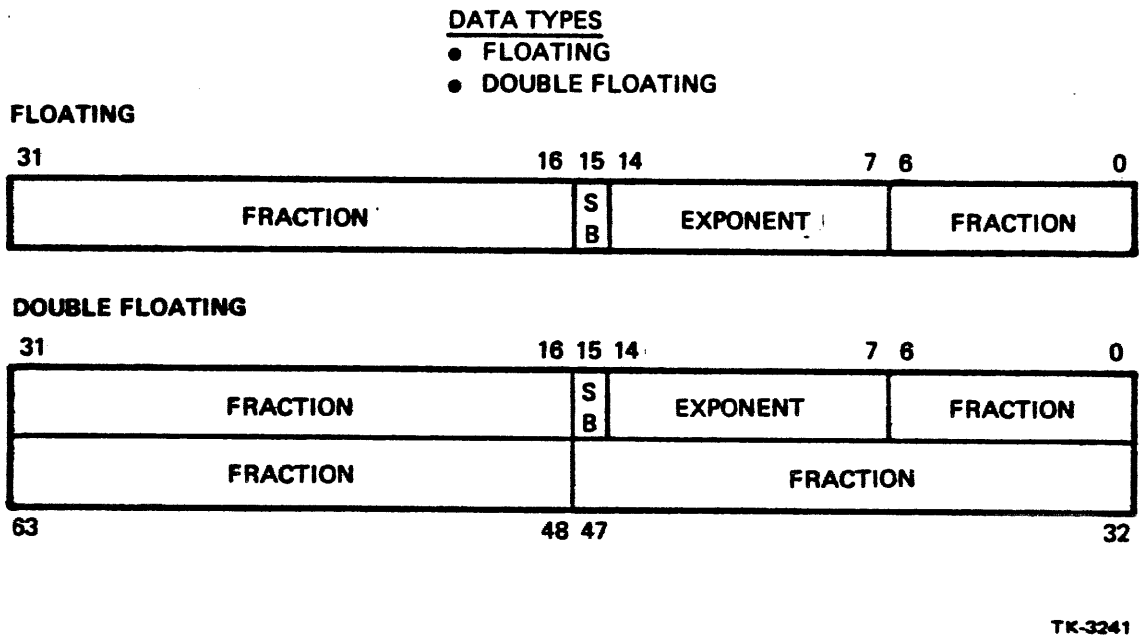
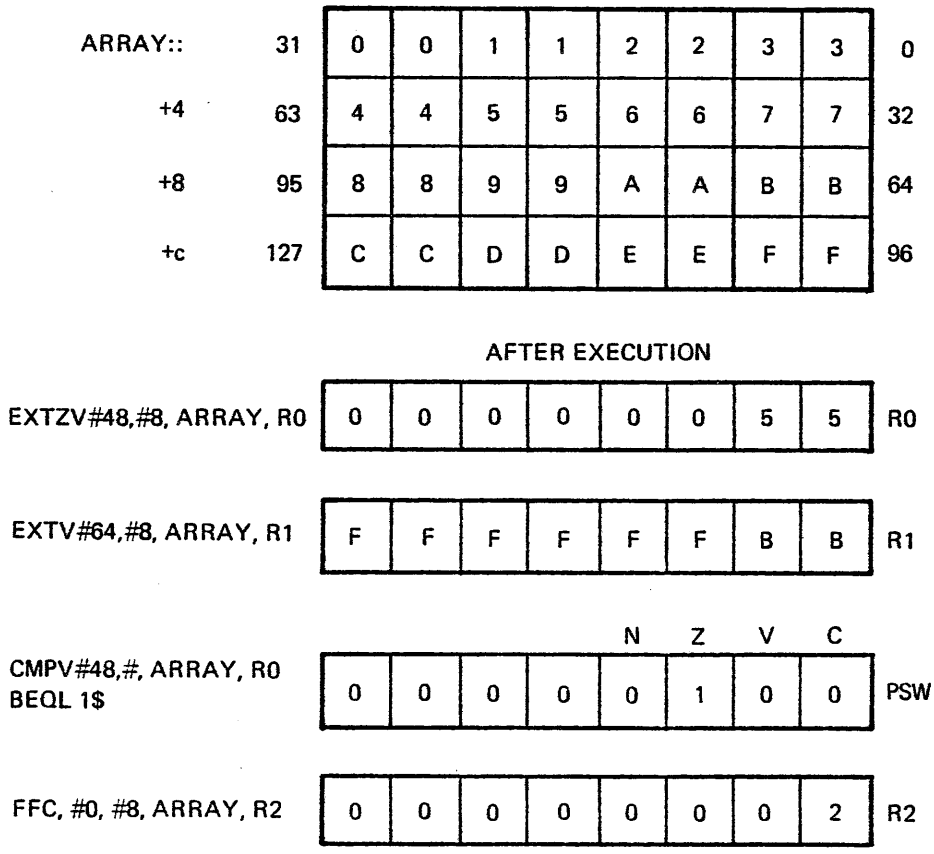

DATA TYPES
• FLOATING
• DOUBLE FLOATING
FLOATING
31
16 15 14
7 6
0
FRACTION
S B
EXPONENT
FRACTION
DOUBLE FLOATING
31
16 15 14
7 6
0
FRACTION
S B
EXPONENT
FRACTION
FRACTION
FRACTION
63
48 47
32
TK-3241

Figure 5-2



FIELD INSTRUCTION EXECUTION EXAMPLES

TK-3238

Figure 5-3

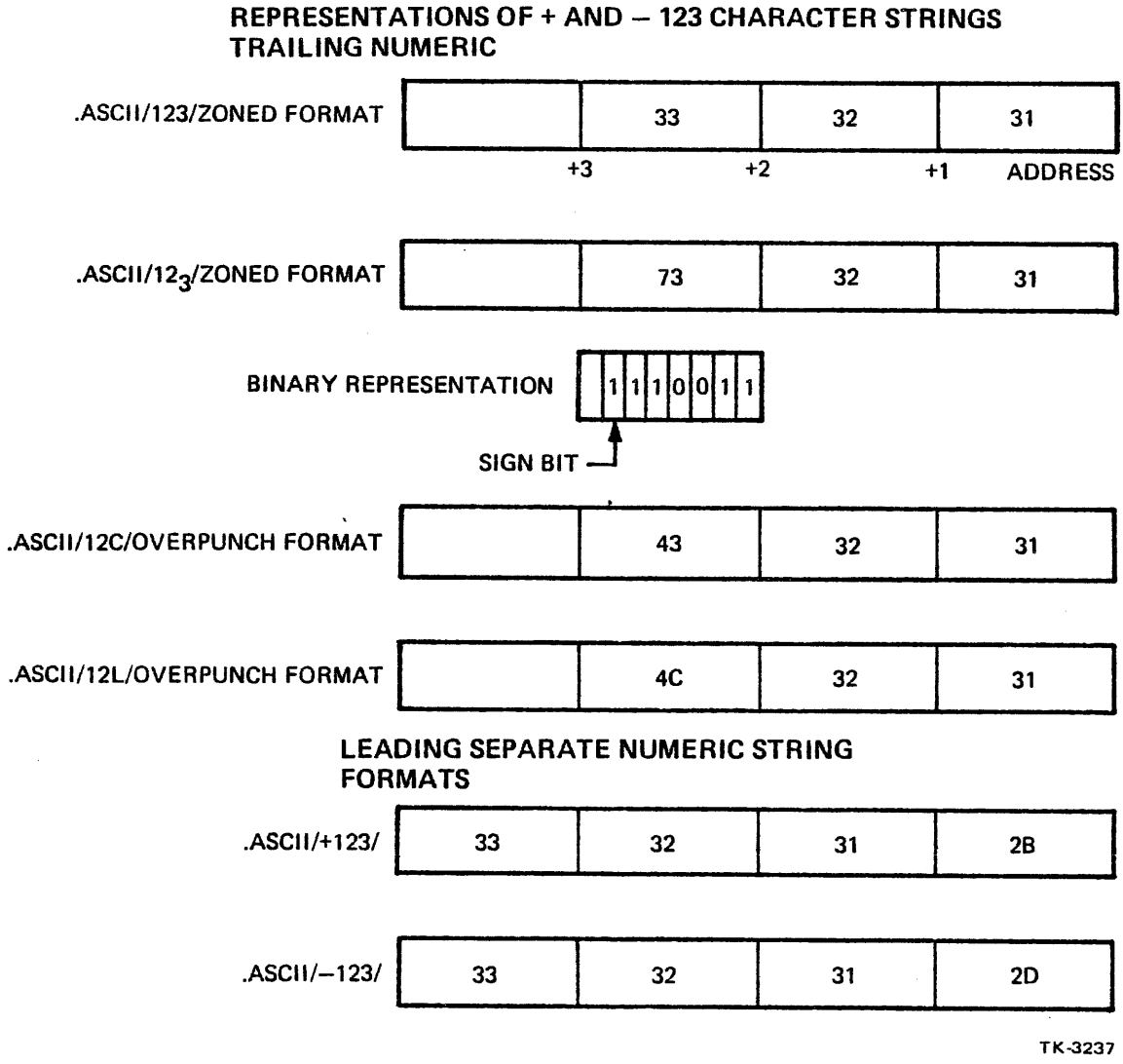

REPRESENTATIONS OF + AND – 123 CHARACTER STRINGS
TRAILING NUMERIC
.ASCII/123/ZONED FORMAT
33
32
31
+3
+2
+1
ADDRESS
.ASCII/12$_3$/ZONED FORMAT
73
32
31
BINARY REPRESENTATION
1 1 1 0 0 1 1
SIGN BIT
.ASCII/12C/OVERPUNCH FORMAT
43
32
31
.ASCII/12L/OVERPUNCH FORMAT
4C
32
31
LEADING SEPARATE NUMERIC STRING
FORMATS
.ASCII/+123/
33
32
31
2B
.ASCII/–123/
33
32
31
2D
TK-3237

Figure 5-4

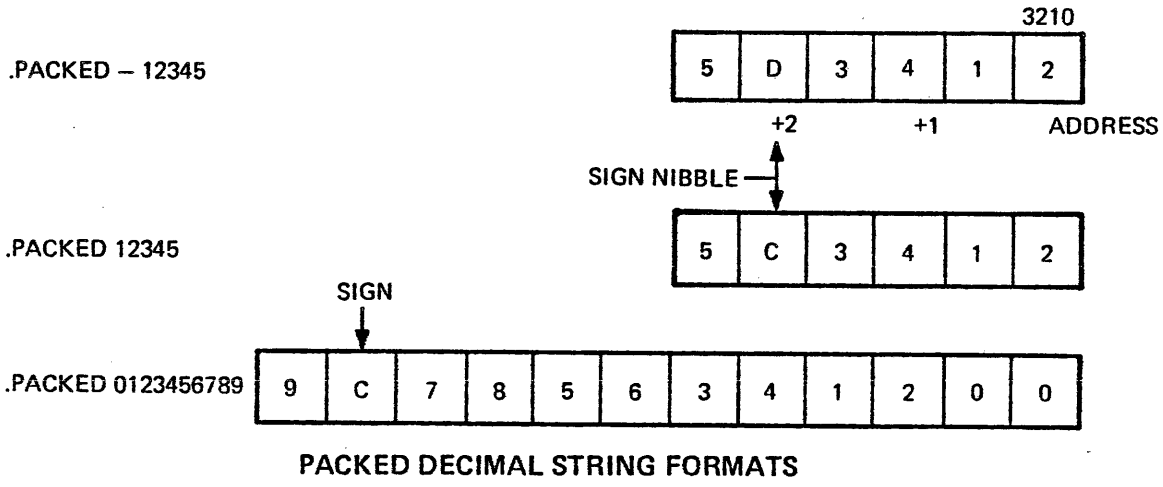

3210
.PACKED – 12345
5 D 3 4 1 2
+2
+1
ADDRESS
SIGN NIBBLE
.PACKED 12345
5 C 3 4 1 2
SIGN
.PACKED 0123456789
9 C 7 8 5 6 3 4 1 2 0 0
PACKED DECIMAL STRING FORMATS
TK-3239

Figure 5-5

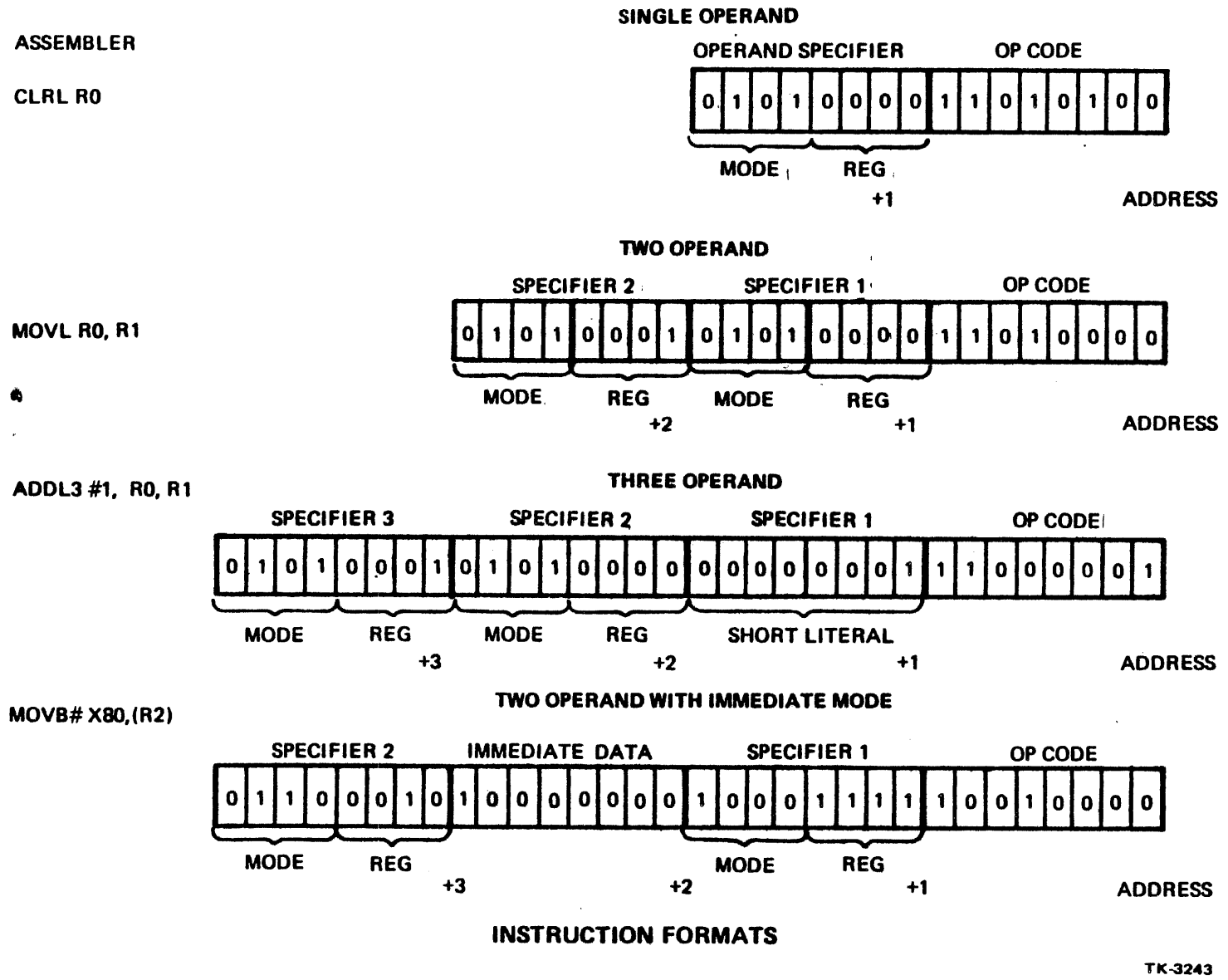


Figure 5-6 INSTRUCTION FORMATS

TK-3243

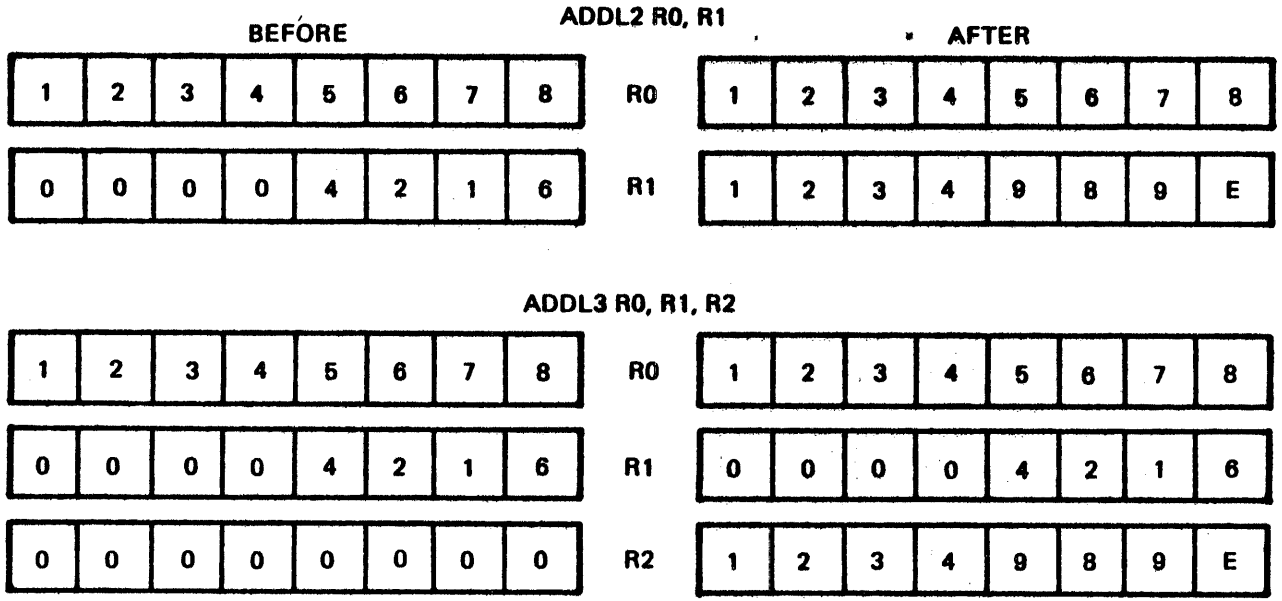
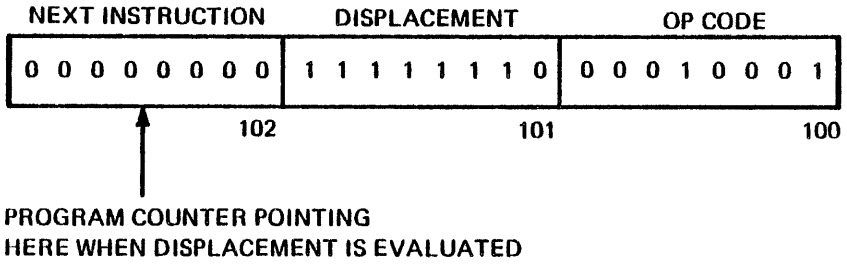
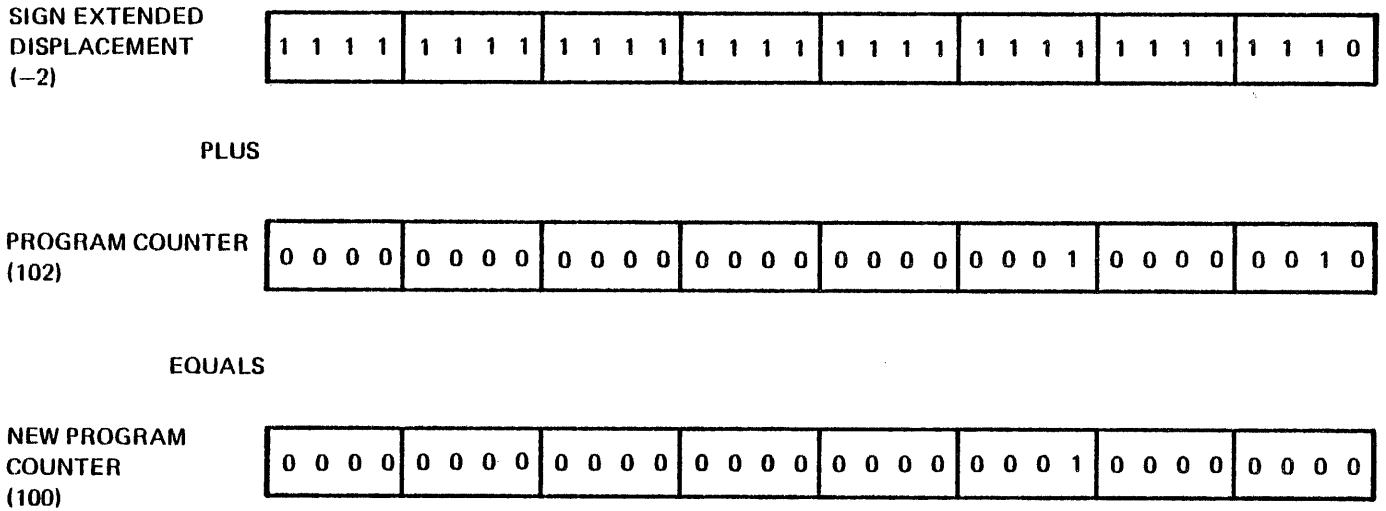


Figure 5-7
TWO AND THREE OPERAND INSTRUCTION
EXECUTION IN REGISTER MODE

ASSEMBLER SYNTAX

```
.=^X100
SELF: BRB SELF
```





BRANCH OFFSET CALCULATION

TK-3225

Figure 5-8

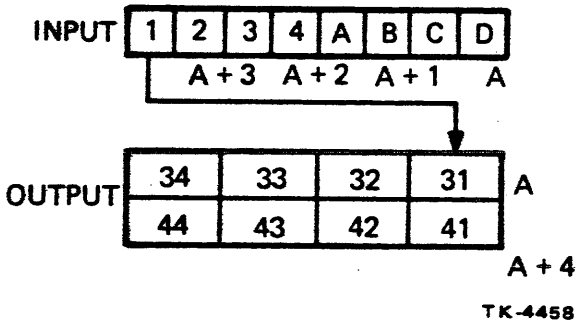
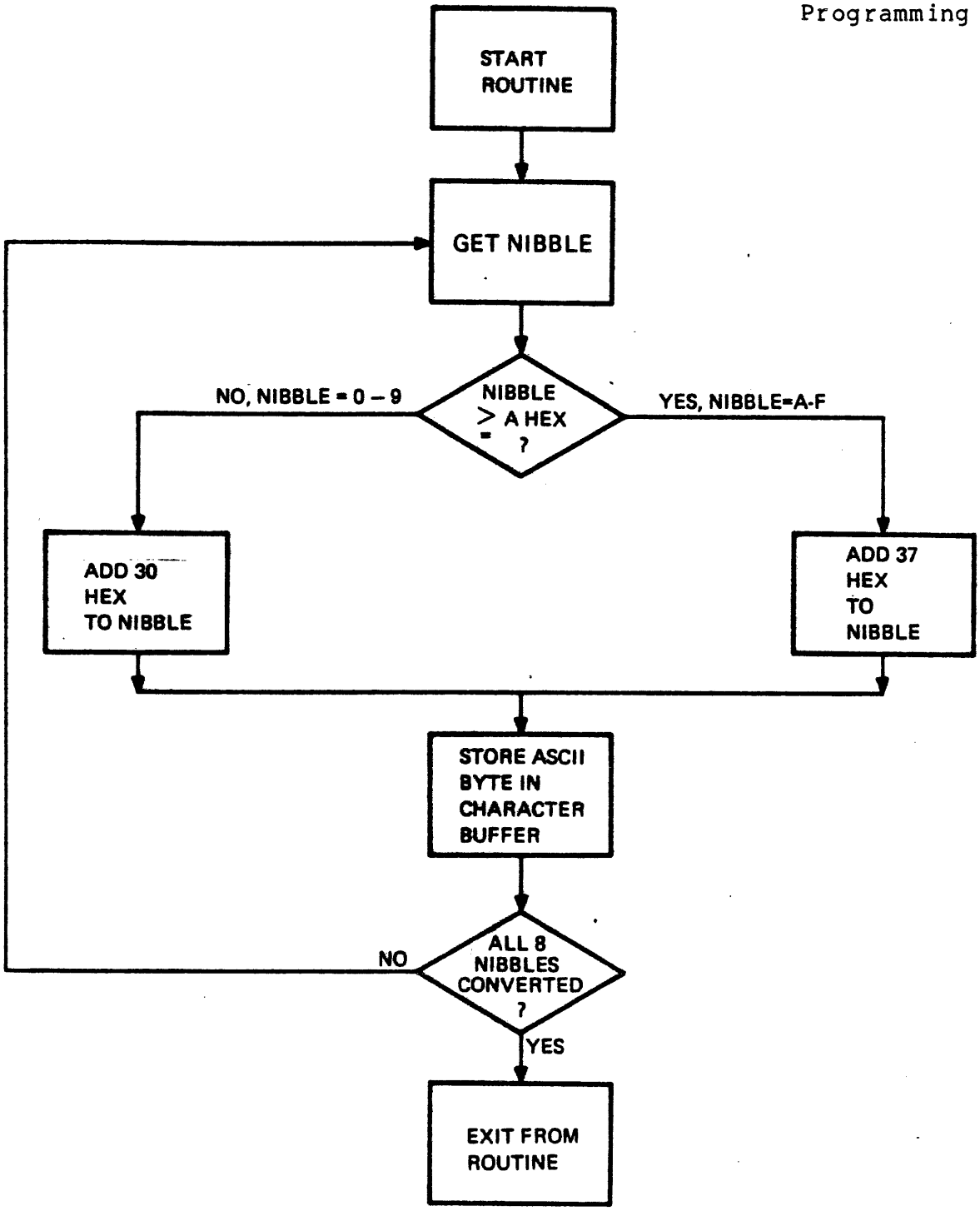

INPUT
1
2
3
4
A
B
C
D
A+3
A+2
A+1
A
OUTPUT
34
33
32
31
A
44
43
42
41
A+4
TK-4458

Figure 5-9



FLOW DIAGRAM FOR PACKED HEX TO ASCII CHARACTER CONVERSION ROUTINE

Figure 5-10

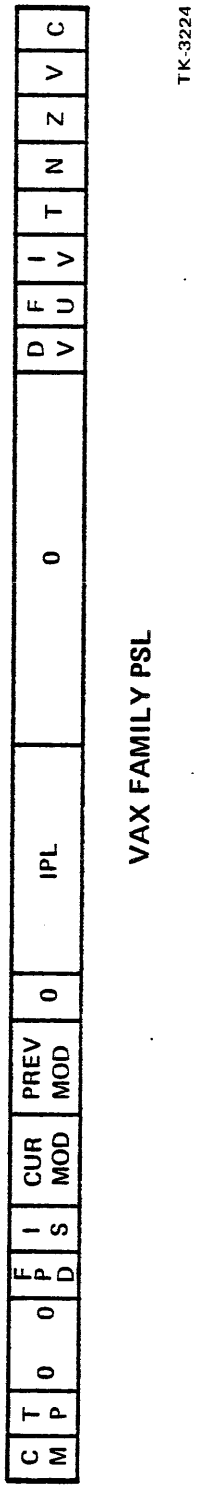

CM
TP
0
0
FPD
IS
CUR MOD
PREV MOD
0
IPL
0
DV
FU
IV
T
N
Z
V
C
TK-3224
VAX FAMILY PSL

Figure 5-11

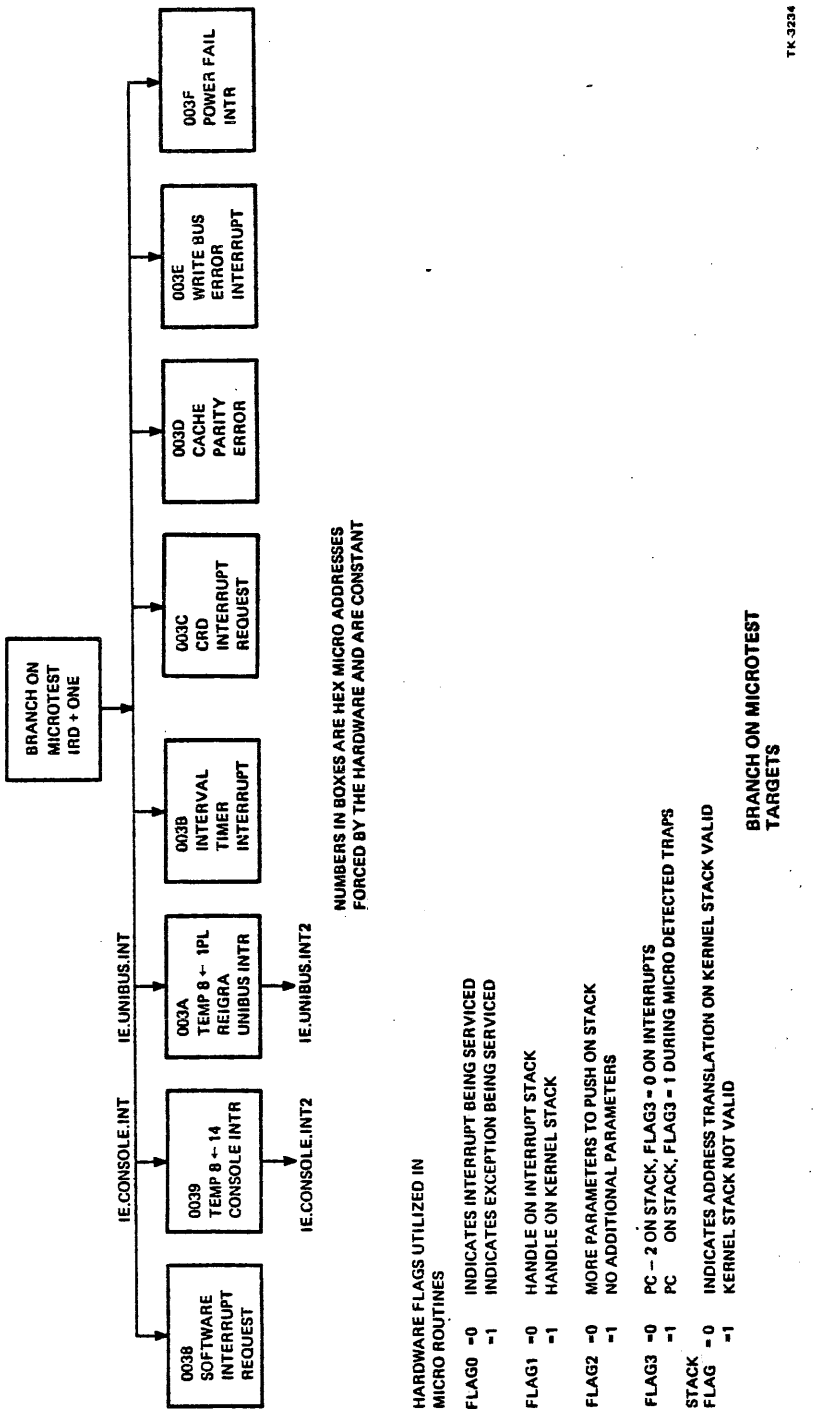

BRANCH ON MICROTEST IRD + ONE
0038 SOFTWARE INTERRUPT REQUEST
0039 TEMP 8 ← 14 CONSOLE INTR
003A TEMP 8 ← 1PL REIGRA UNIBUS INTR
003B INTERVAL TIMER INTERRUPT
003C CRD INTERRUPT REQUEST
003D CACHE PARITY ERROR
003E WRITE BUS ERROR INTERRUPT
003F POWER FAIL INTR
IE.CONSOLE.INT
IE.UNIBUS.INT
IE.CONSOLE.INT2
IE.UNIBUS.INT2
NUMBERS IN BOXES ARE HEX MICRO ADDRESSES FORCED BY THE HARDWARE AND ARE CONSTANT
HARDWARE FLAGS UTILIZED IN MICRO ROUTINES
FLAG0 =0 INDICATES INTERRUPT BEING SERVICED
=1 INDICATES EXCEPTION BEING SERVICED
FLAG1 =0 HANDLE ON INTERRUPT STACK
=1 HANDLE ON KERNEL STACK
FLAG2 =0 MORE PARAMETERS TO PUSH ON STACK
=1 NO ADDITIONAL PARAMETERS
FLAG3 =0 PC – 2 ON STACK, FLAG3 = 0 ON INTERRUPTS
=1 PC ON STACK, FLAG3 = 1 DURING MICRO DETECTED TRAPS
STACK FLAG =0 INDICATES ADDRESS TRANSLATION ON KERNEL STACK VALID
=1 KERNEL STACK NOT VALID
BRANCH ON MICROTEST TARGETS
TK-3234

Figure 5-12

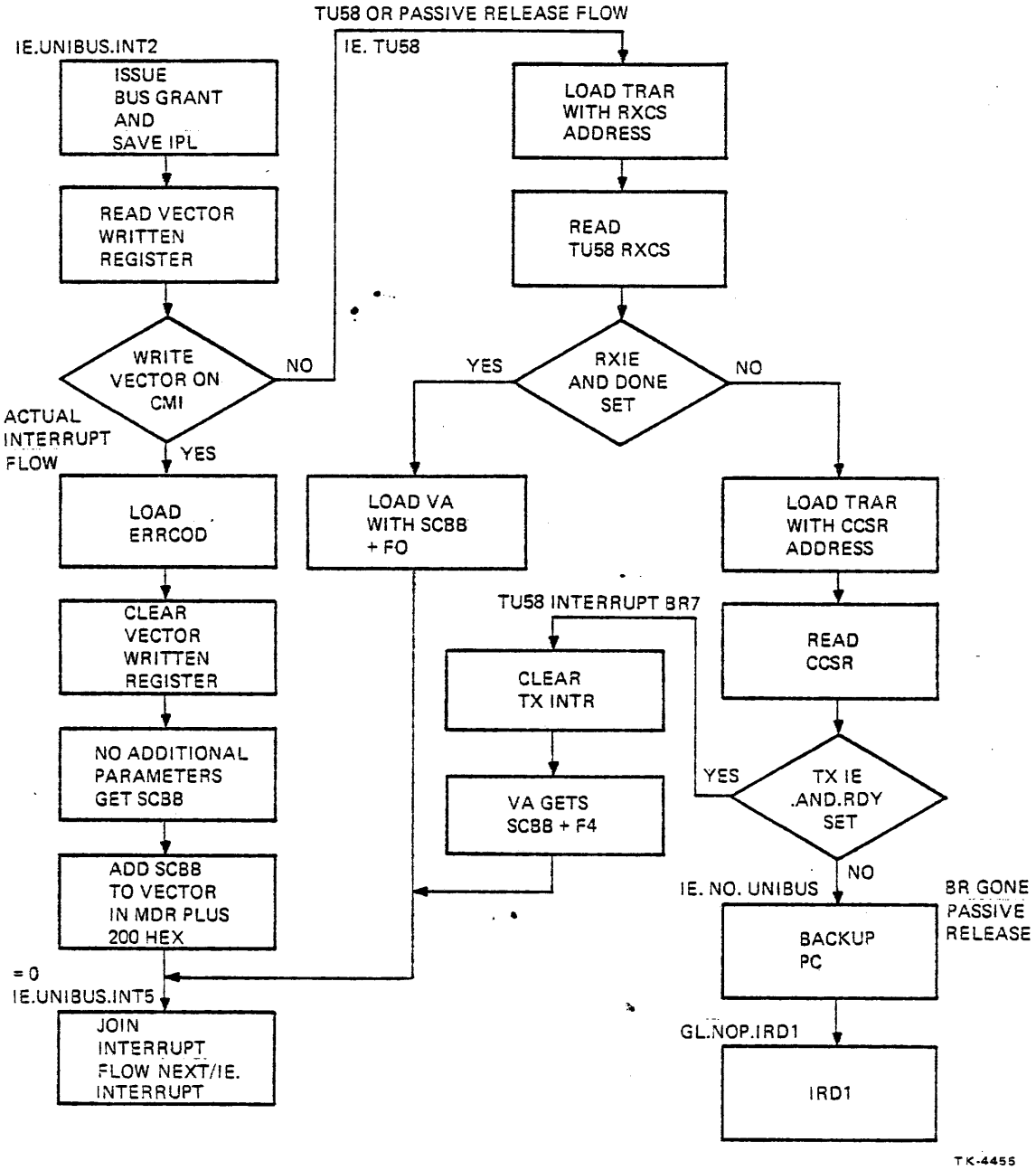

TU58 OR PASSIVE RELEASE FLOW
IE.UNIBUS.INT2
ISSUE BUS GRANT AND SAVE IPL
READ VECTOR WRITTEN REGISTER
WRITE VECTOR ON CMI
NO
YES
ACTUAL INTERRUPT FLOW
LOAD ERRCOD
CLEAR VECTOR WRITTEN REGISTER
NO ADDITIONAL PARAMETERS GET SCBB
ADD SCBB TO VECTOR IN MDR PLUS 200 HEX
= 0
IE.UNIBUS.INT5
JOIN INTERRUPT FLOW NEXT/IE. INTERRUPT
IE. TU58
LOAD TRAR WITH RXCS ADDRESS
READ TU58 RXCS
YES
RXIE AND DONE SET
NO
LOAD VA WITH SCBB + F0
TU58 INTERRUPT BR7
CLEAR TX INTR
VA GETS SCBB + F4
LOAD TRAR WITH CCSR ADDRESS
READ CCSR
YES
TX IE .AND.RDY SET
NO
IE. NO. UNIBUS
BACKUP PC
BR GONE PASSIVE RELEASE
GL.NOP.IRD1
IRD1
TK-4455

Figure 5-13

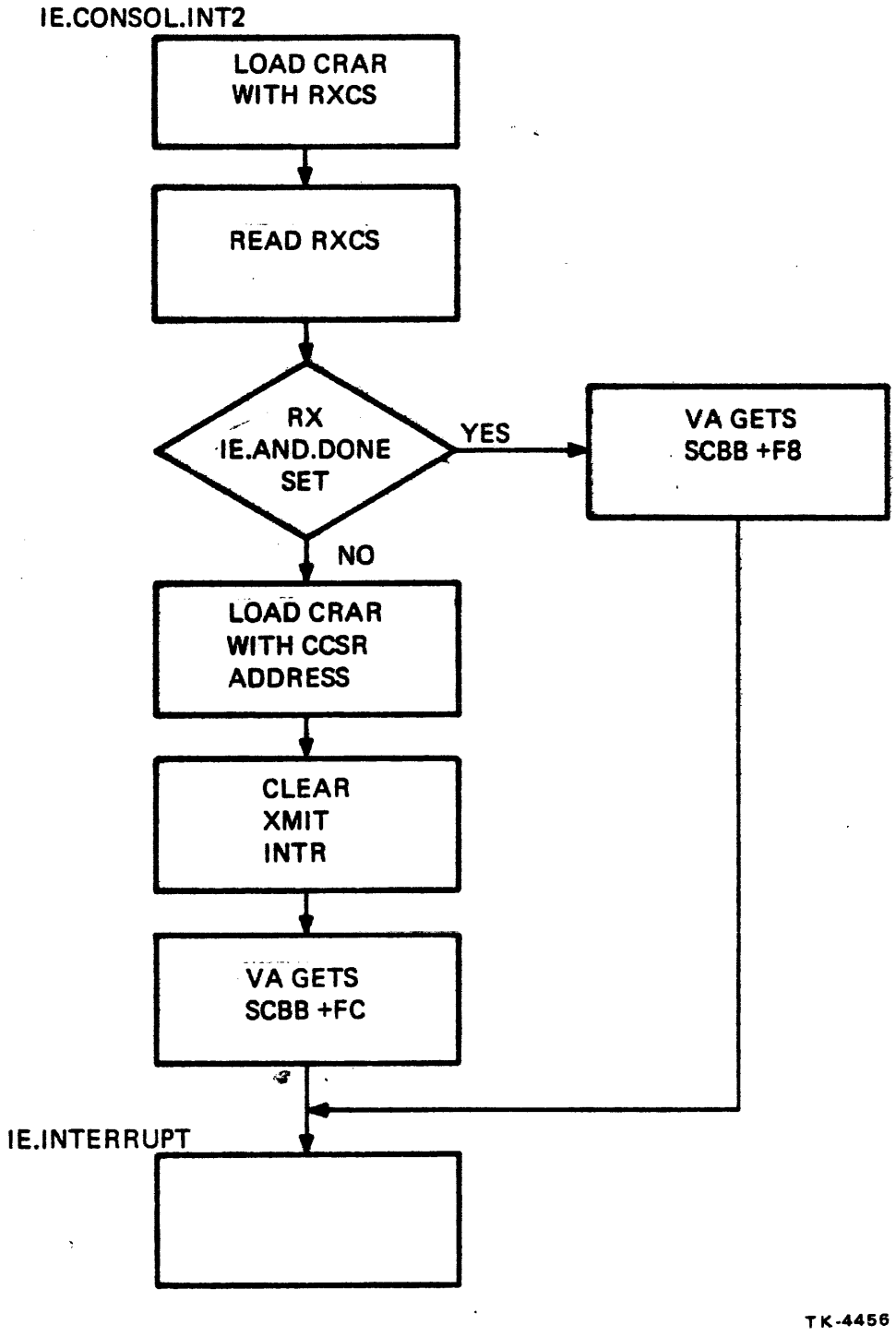

IE.CONSOL.INT2
LOAD CRAR WITH RXCS
READ RXCS
RX IE.AND.DONE SET
YES
VA GETS SCBB +F8
NO
LOAD CRAR WITH CCSR ADDRESS
CLEAR XMIT INTR
VA GETS SCBB +FC
IE.INTERRUPT
TK-4456

Figure 5-14

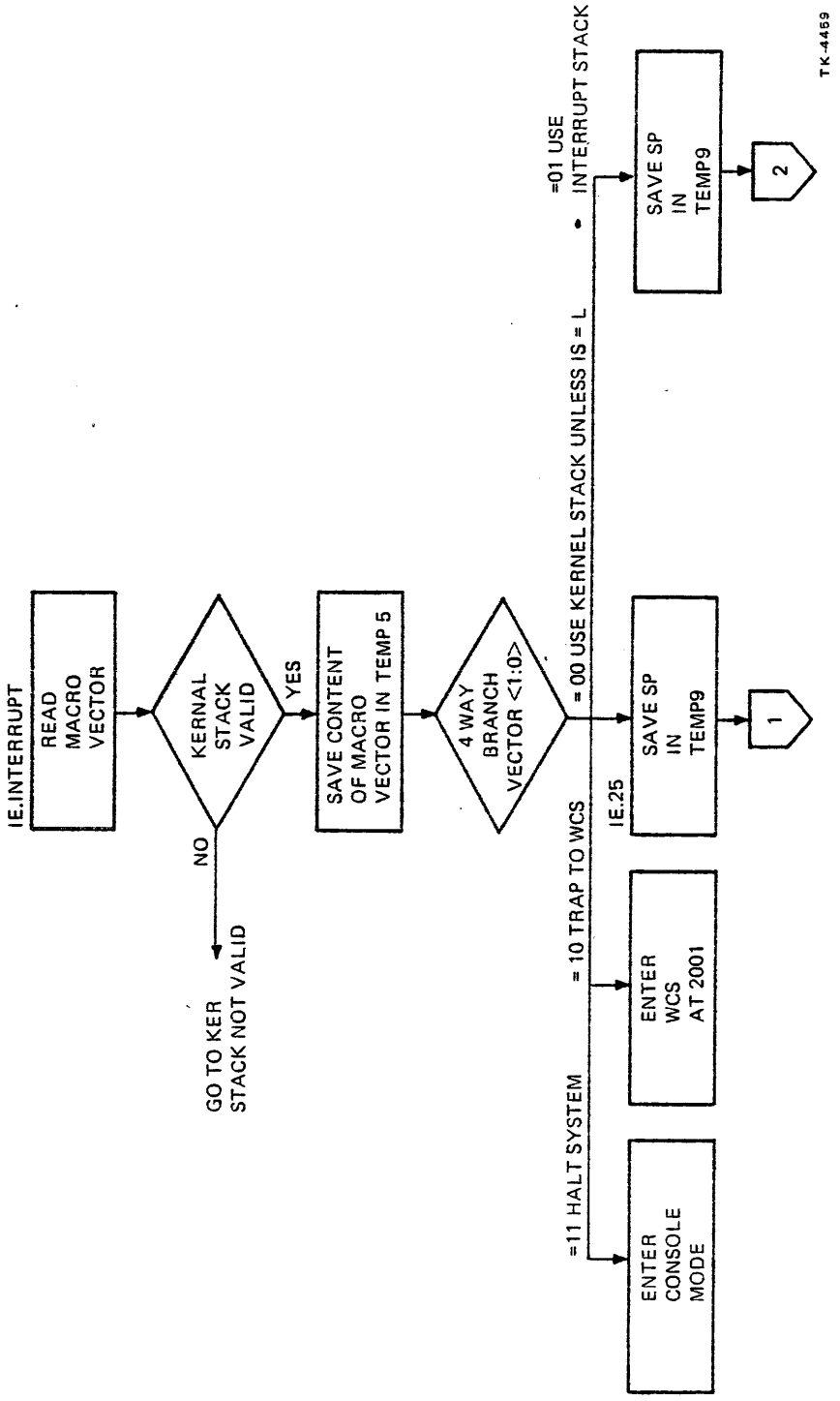

IE.INTERRUPT
READ MACRO VECTOR
KERNAL STACK VALID
NO
YES
GO TO KER STACK NOT VALID
SAVE CONTENT OF MACRO VECTOR IN TEMP 5
4 WAY BRANCH VECTOR <1:0>
=00 USE KERNEL STACK UNLESS IS = L
=01 USE INTERRUPT STACK
SAVE SP IN TEMP9
2
IE.25
SAVE SP IN TEMP9
1
=10 TRAP TO WCS
ENTER WCS AT 2001
=11 HALT SYSTEM
ENTER CONSOLE MODE
TK-4469

Figure 5-15

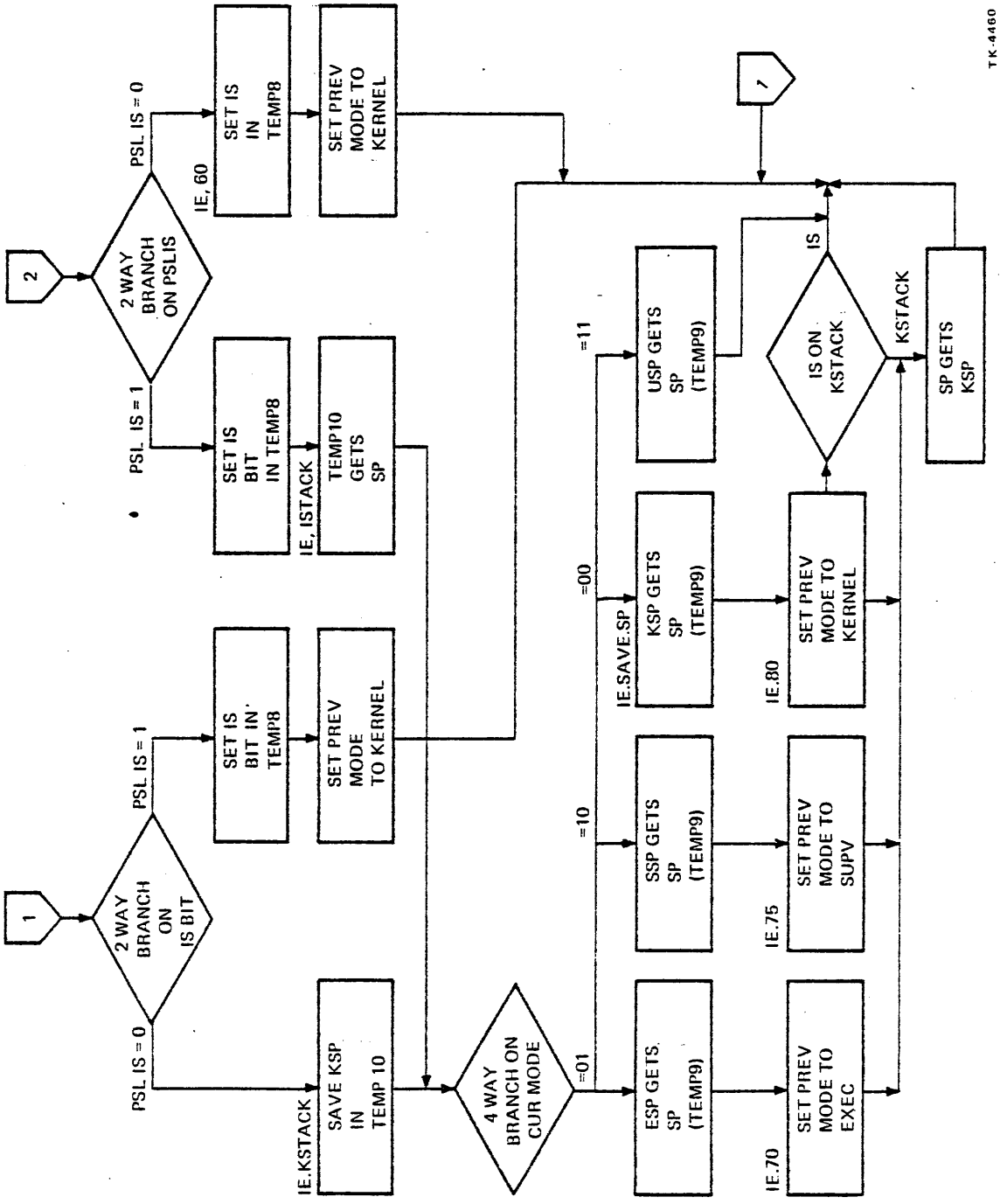
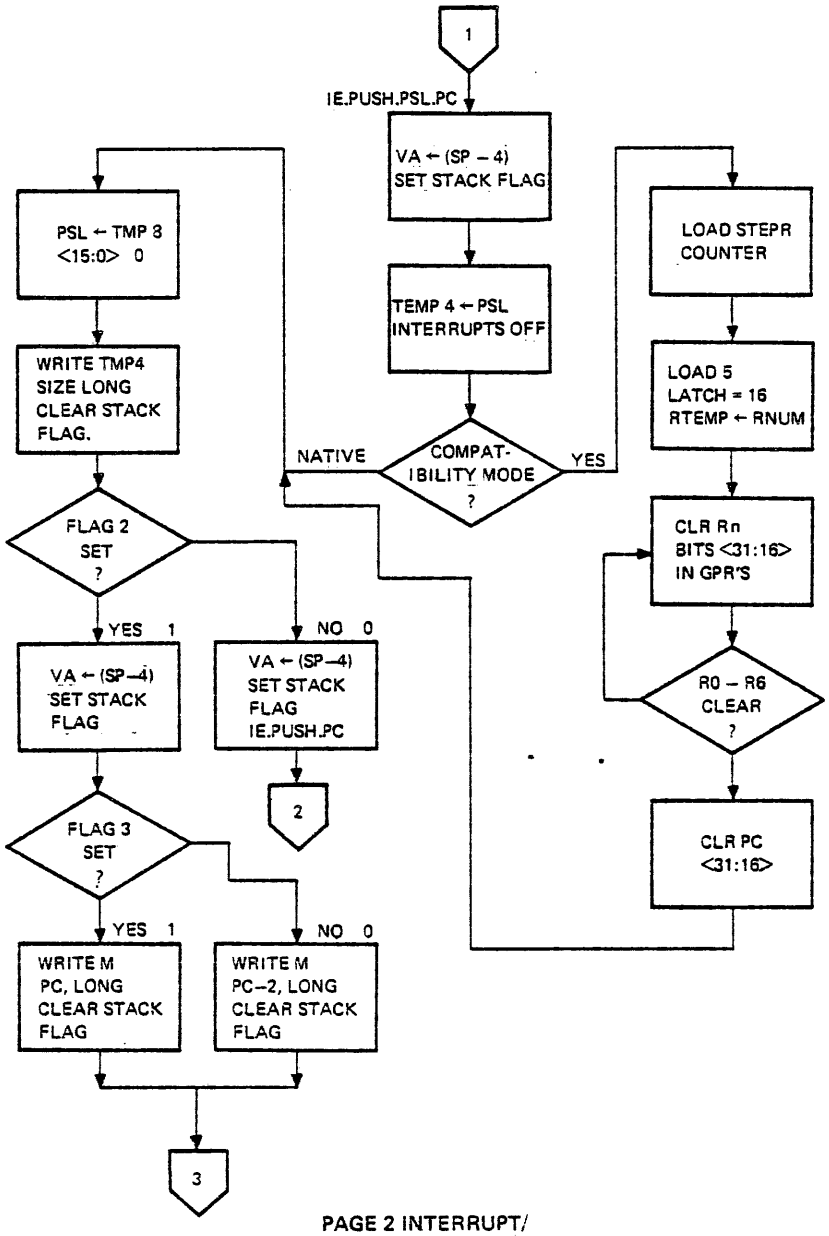
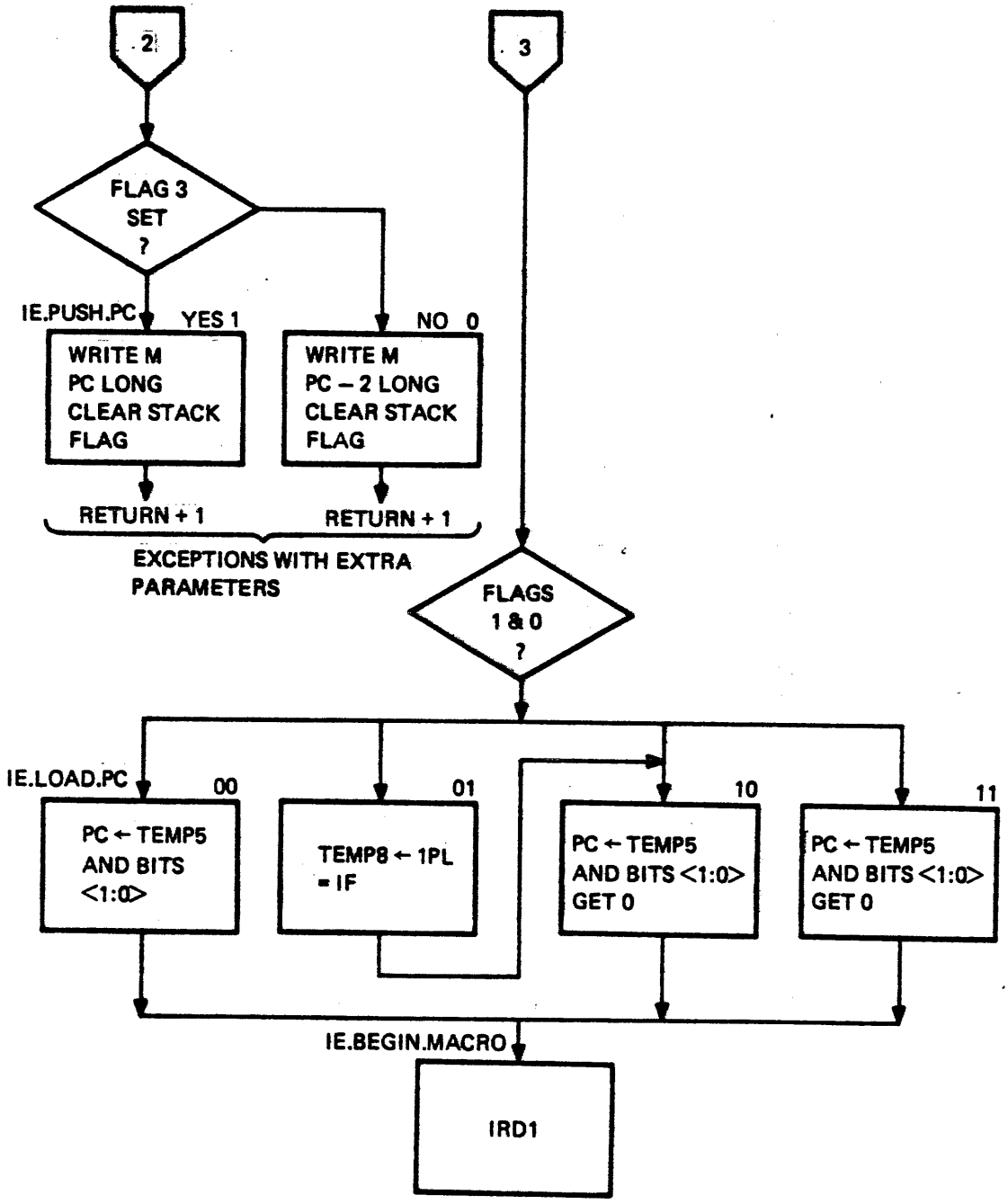

1
2 WAY BRANCH ON IS BIT
PSL IS = 1
SET IS BIT IN TEMP8
SET PREV MODE TO KERNEL
PSL IS = 0
IE.KSTACK
SAVE KSP IN TEMP 10
2
2 WAY BRANCH ON PSLIS
PSL IS = 0
IE, 60
SET IS IN TEMP8
SET PREV MODE TO KERNEL
PSL IS = 1
SET IS BIT IN TEMP8
IE, ISTACK
TEMP10 GETS SP
4 WAY BRANCH ON CUR MODE
=01
ESP GETS SP (TEMP9)
IE.70
SET PREV MODE TO EXEC
=10
SSP GETS SP (TEMP9)
IE.75
SET PREV MODE TO SUPV
=00
IE.SAVE.SP
KSP GETS SP (TEMP9)
IE.80
SET PREV MODE TO KERNEL
=11
USP GETS SP (TEMP9)
1
IS
IS ON KSTACK
KSTACK
SP GETS KSP
TK-4460

Figure 5-15A



PAGE 2 INTERRUPT/
EXCEPTION FLOWS

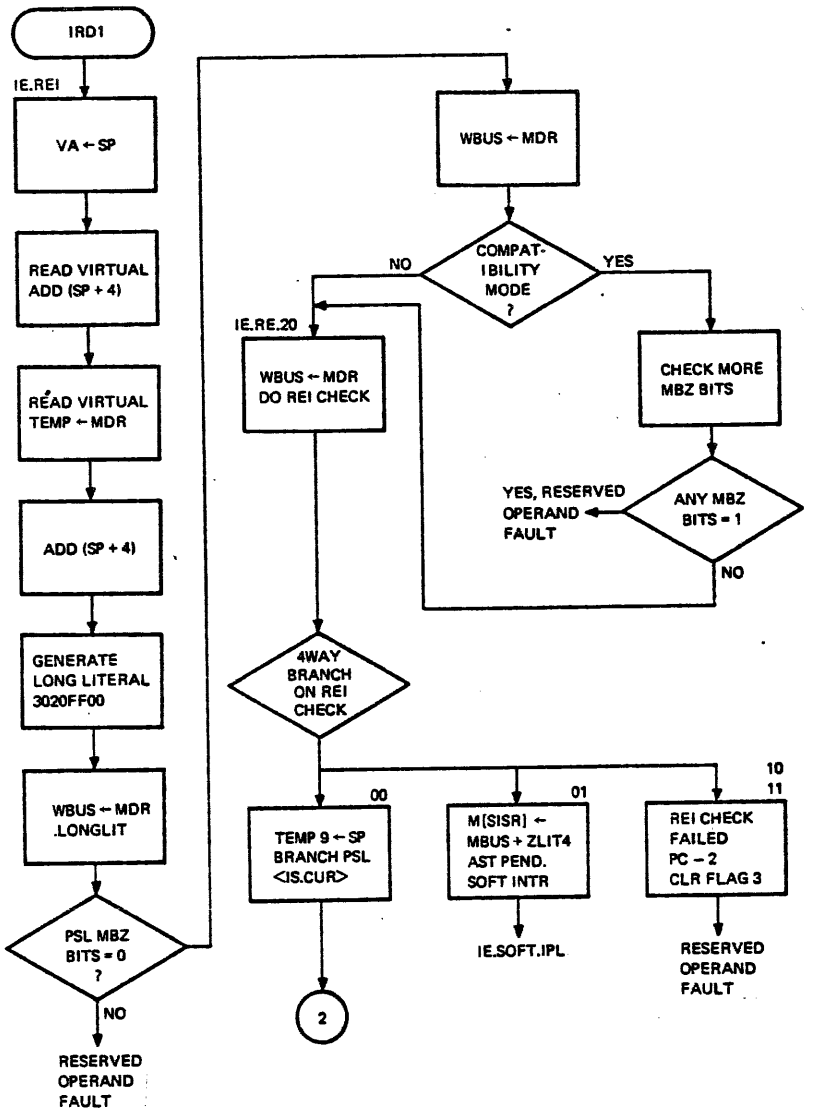
Figure 5-16



PAGE 3 GENERALIZED INTERRUPT/EXCEPTION FLOWS

TK-3230

Figure 5-17



REI INSTRUCTION GENERALIZED MICROFLOWS

Figure 5-18

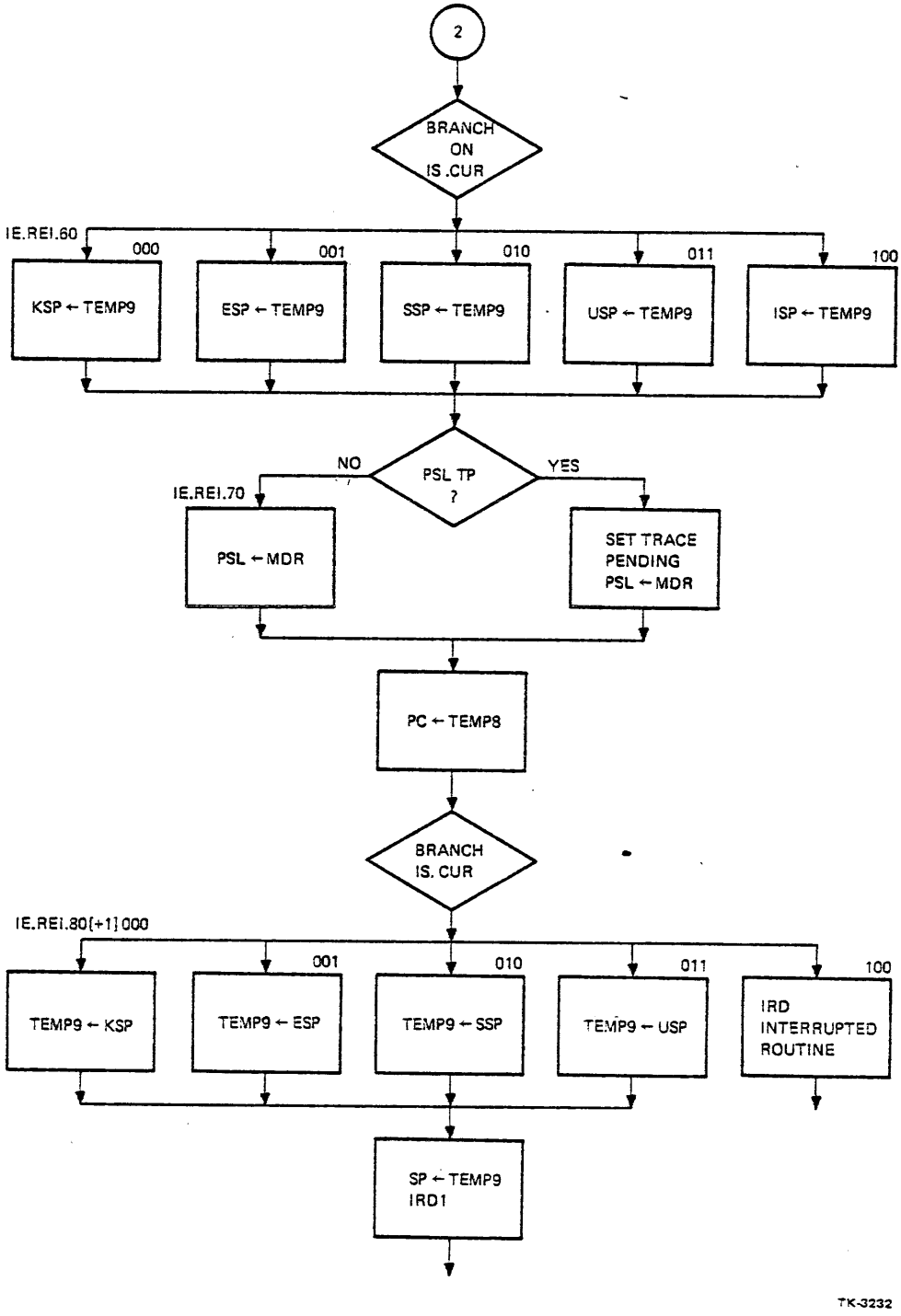
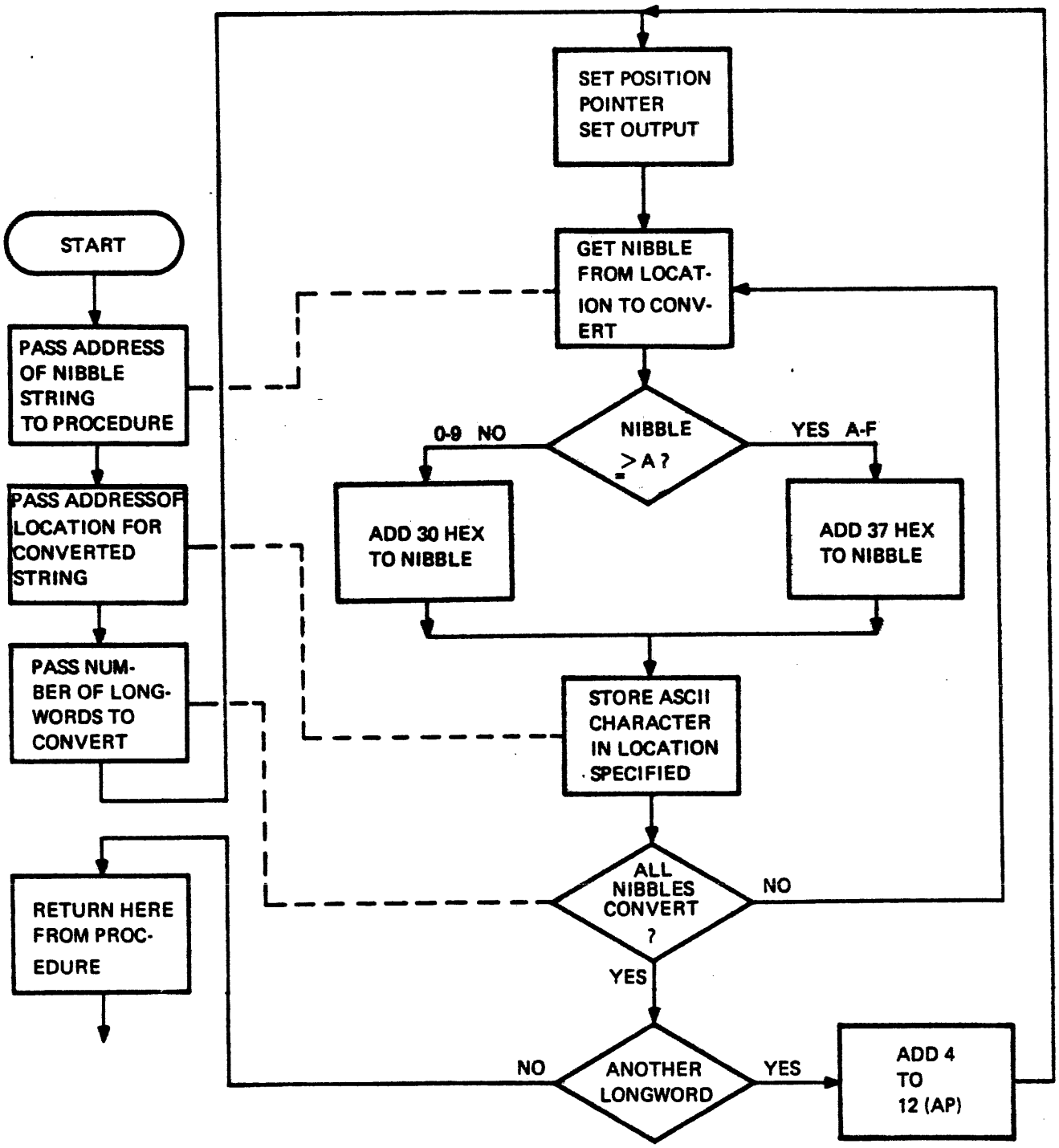

2
BRANCH ON IS.CUR
IE.REI.60
000
001
010
011
100
KSP ← TEMP9
ESP ← TEMP9
SSP ← TEMP9
USP ← TEMP9
ISP ← TEMP9
NO
PSL TP ?
YES
IE.REI.70
PSL ← MDR
SET TRACE PENDING
PSL ← MDR
PC ← TEMP8
BRANCH IS. CUR
IE.REI.80[+1]000
001
010
011
100
TEMP9 ← KSP
TEMP9 ← ESP
TEMP9 ← SSP
TEMP9 ← USP
IRD INTERRUPTED ROUTINE
SP ← TEMP9
IRD1
TK-3232

Figure 5-19



FLOW DIAGRAM FOR PACKED HEX TO ASCII CHARACTER CONVERSION PROCEDURE

TK-3229

Figure 5-20

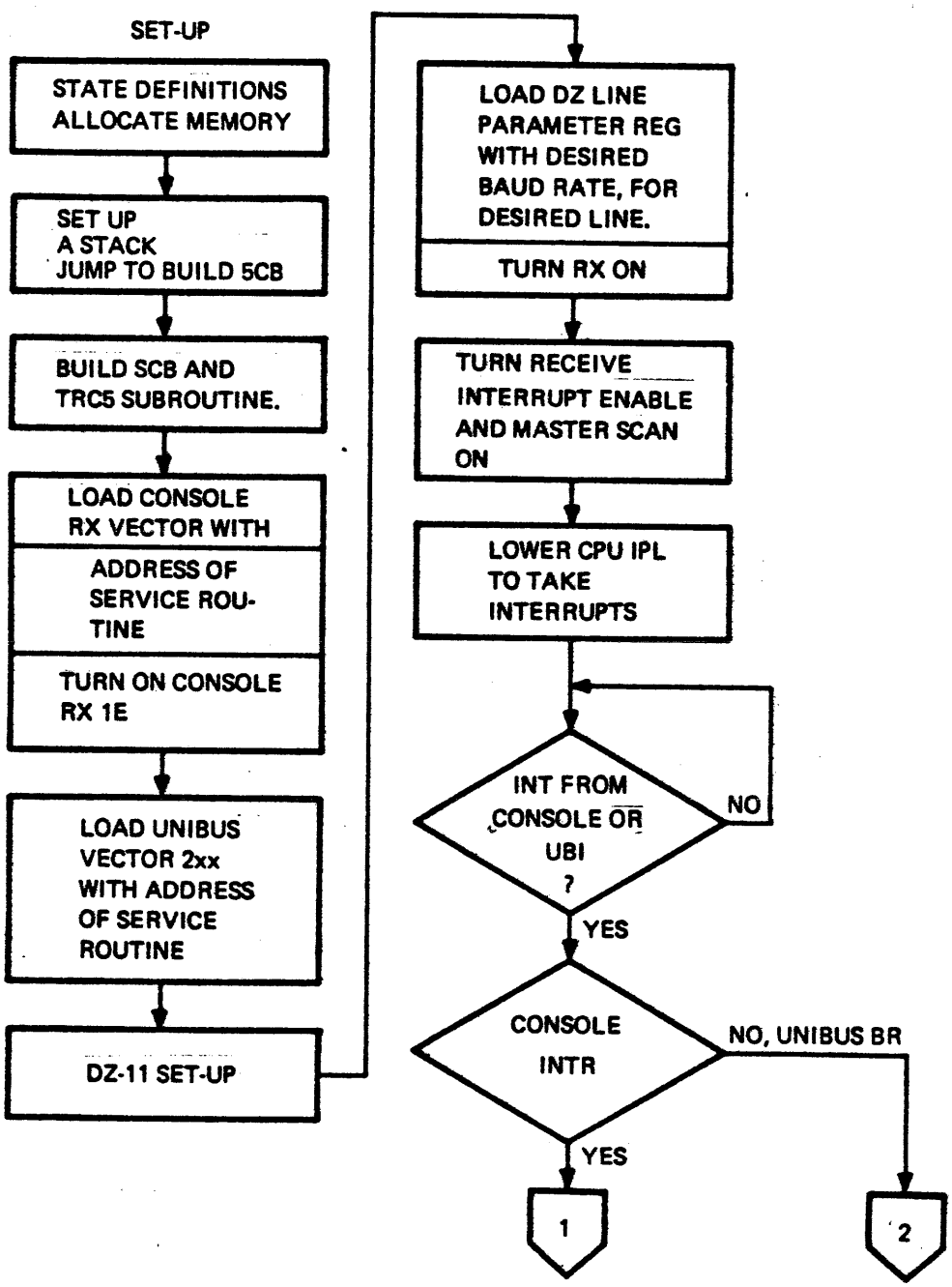

SET-UP
STATE DEFINITIONS
ALLOCATE MEMORY
SET UP
A STACK
JUMP TO BUILD SCB
BUILD SCB AND
TRC5 SUBROUTINE.
LOAD CONSOLE
RX VECTOR WITH
ADDRESS OF
SERVICE ROU-
TINE
TURN ON CONSOLE
RX 1E
LOAD UNIBUS
VECTOR 2xx
WITH ADDRESS
OF SERVICE
ROUTINE
DZ-11 SET-UP
LOAD DZ LINE
PARAMETER REG
WITH DESIRED
BAUD RATE, FOR
DESIRED LINE.
TURN RX ON
TURN RECEIVE
INTERRUPT ENABLE
AND MASTER SCAN
ON
LOWER CPU IPL
TO TAKE
INTERRUPTS
INT FROM
CONSOLE OR
UBI
?
NO
YES
CONSOLE
INTR
NO, UNIBUS BR
YES
1
2
LABORATORY EXERCISE 5
PROGRAM FLOW DIAGRAM
TK-3227

Figure 5-21

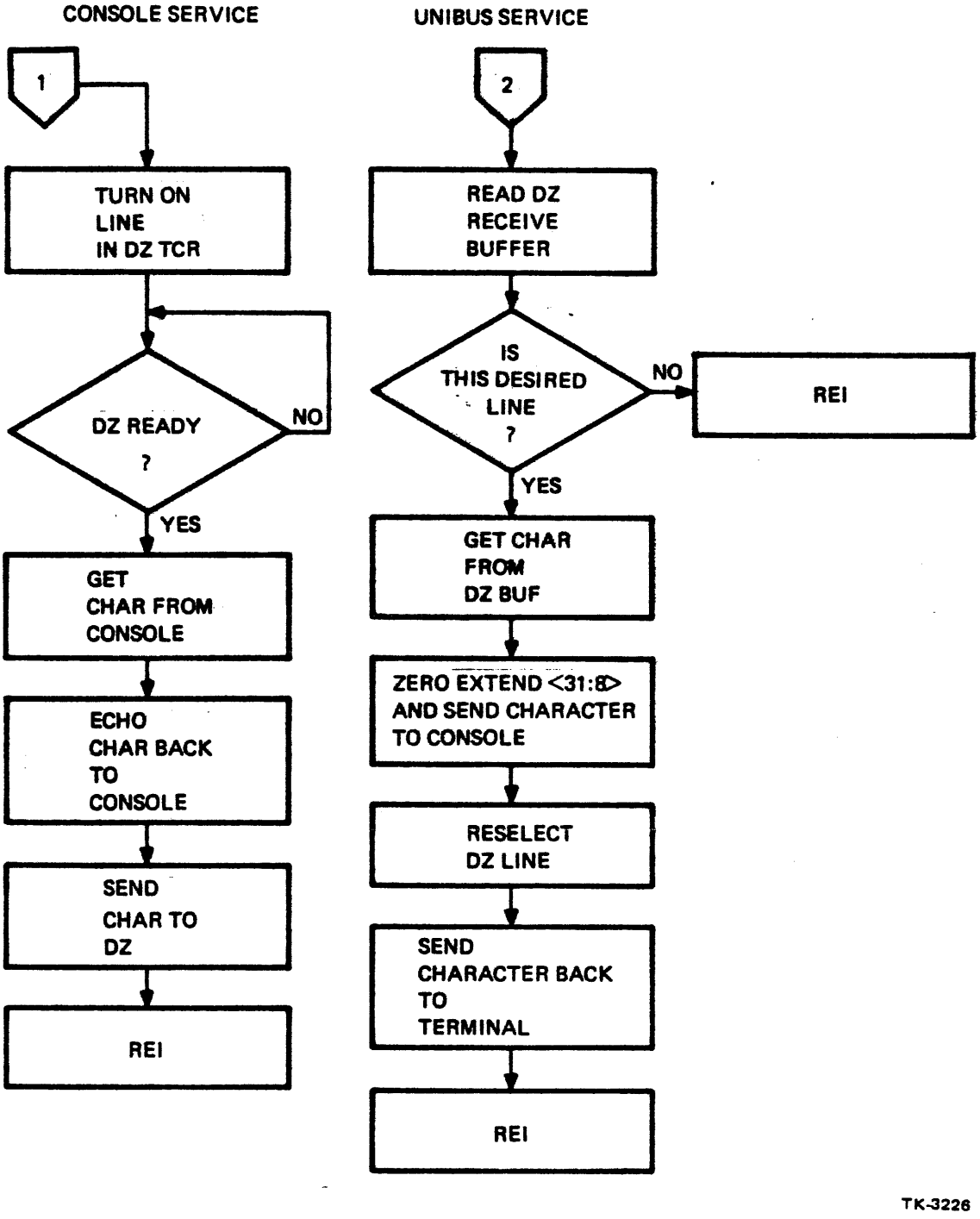
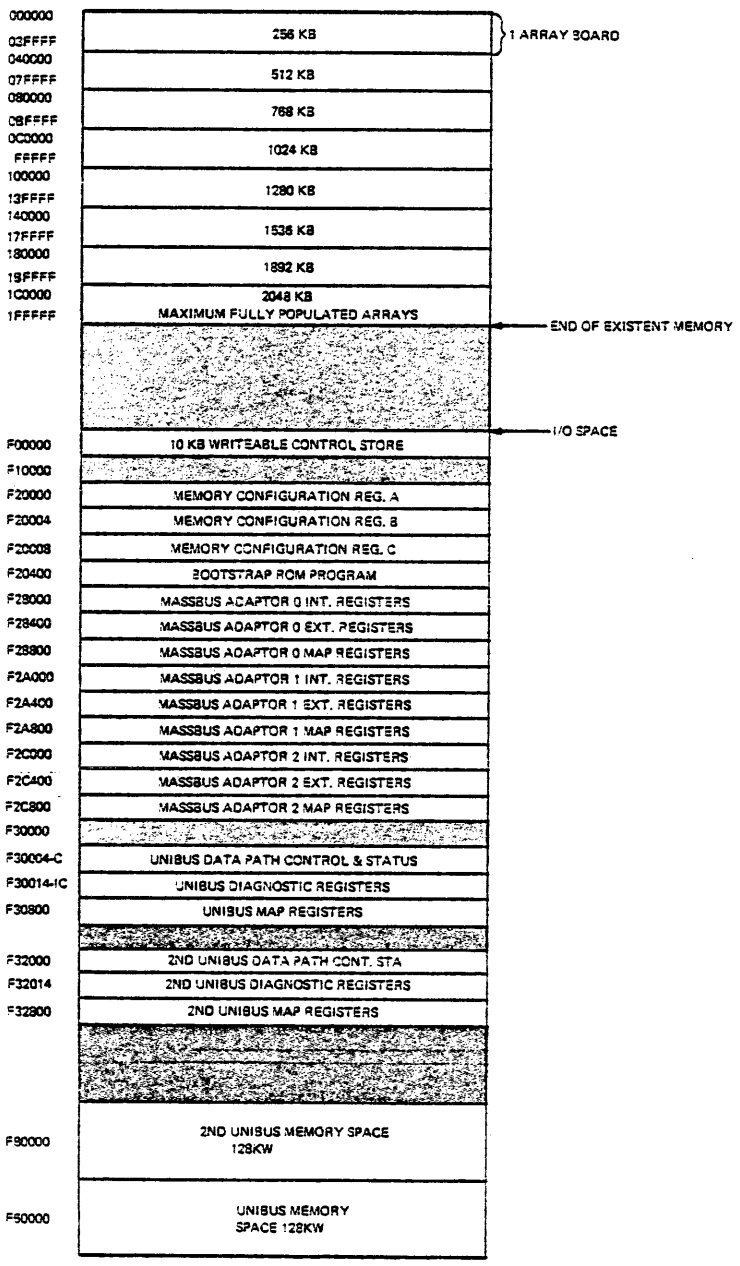

CONSOLE SERVICE
UNIBUS SERVICE
1
2
TURN ON LINE IN DZ TCR
DZ READY ?
NO
YES
GET CHAR FROM CONSOLE
ECHO CHAR BACK TO CONSOLE
SEND CHAR TO DZ
REI
READ DZ RECEIVE BUFFER
IS THIS DESIRED LINE ?
NO
REI
YES
GET CHAR FROM DZ BUF
ZERO EXTEND <31:8> AND SEND CHARACTER TO CONSOLE
RESELECT DZ LINE
SEND CHARACTER BACK TO TERMINAL
REI
TK-3226

Figure 5-22



COMET PHYSICAL MEMORY ORGANIZATION

TK-1735

Figure 5-23

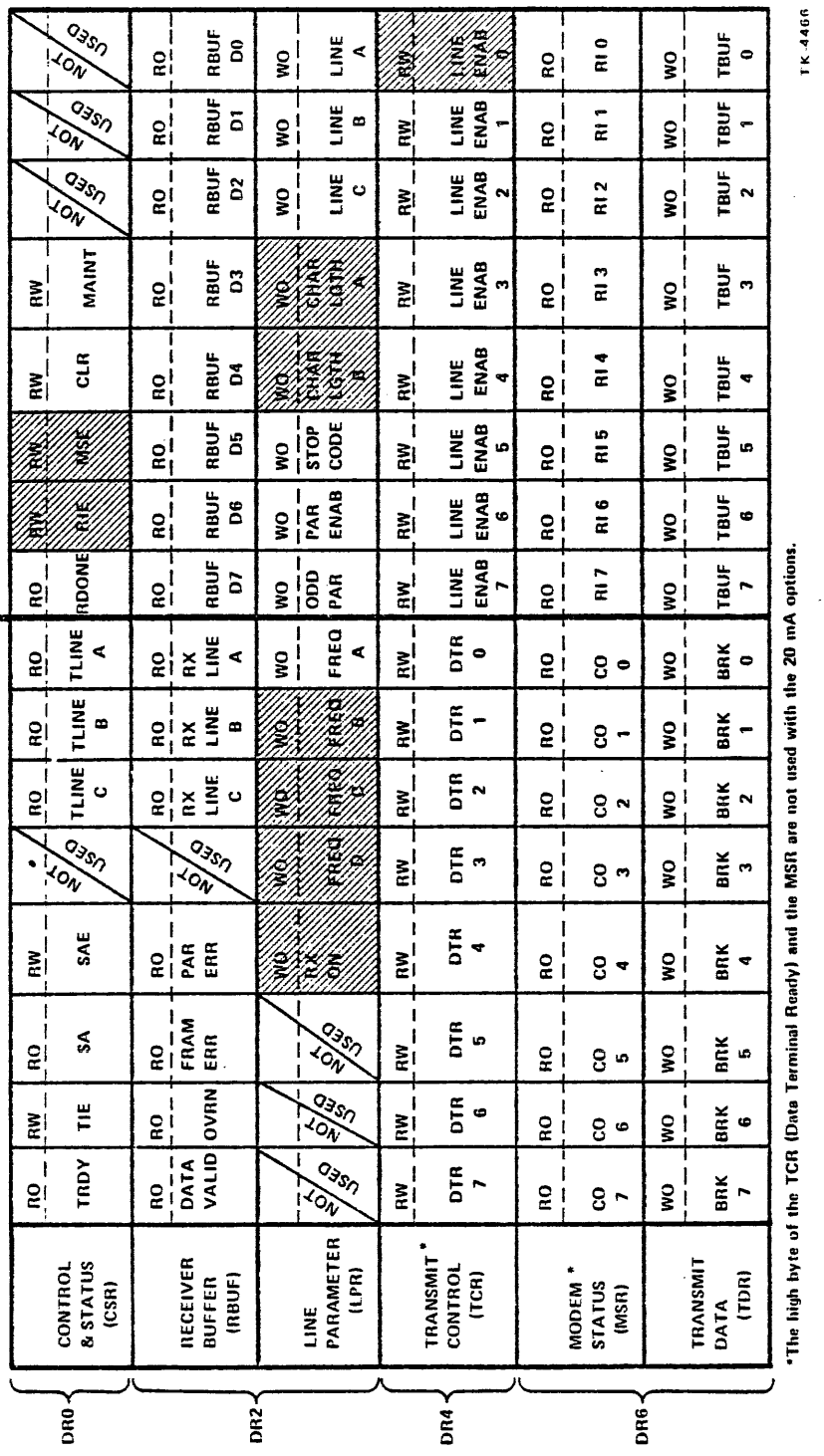

BYTES
HIGH
LOW
MSB
LSB
CONTROL & STATUS (CSR)
RECEIVER BUFFER (RBUF)
LINE PARAMETER (LPR)
TRANSMIT* CONTROL (TCR)
MODEM * STATUS (MSR)
TRANSMIT DATA (TDR)
DR0
DR2
DR4
DR6
*The high byte of the TCR (Data Terminal Ready) and the MSR are not used with the 20 mA options.
TK-4466

Figure 5-24

Figure 5-25

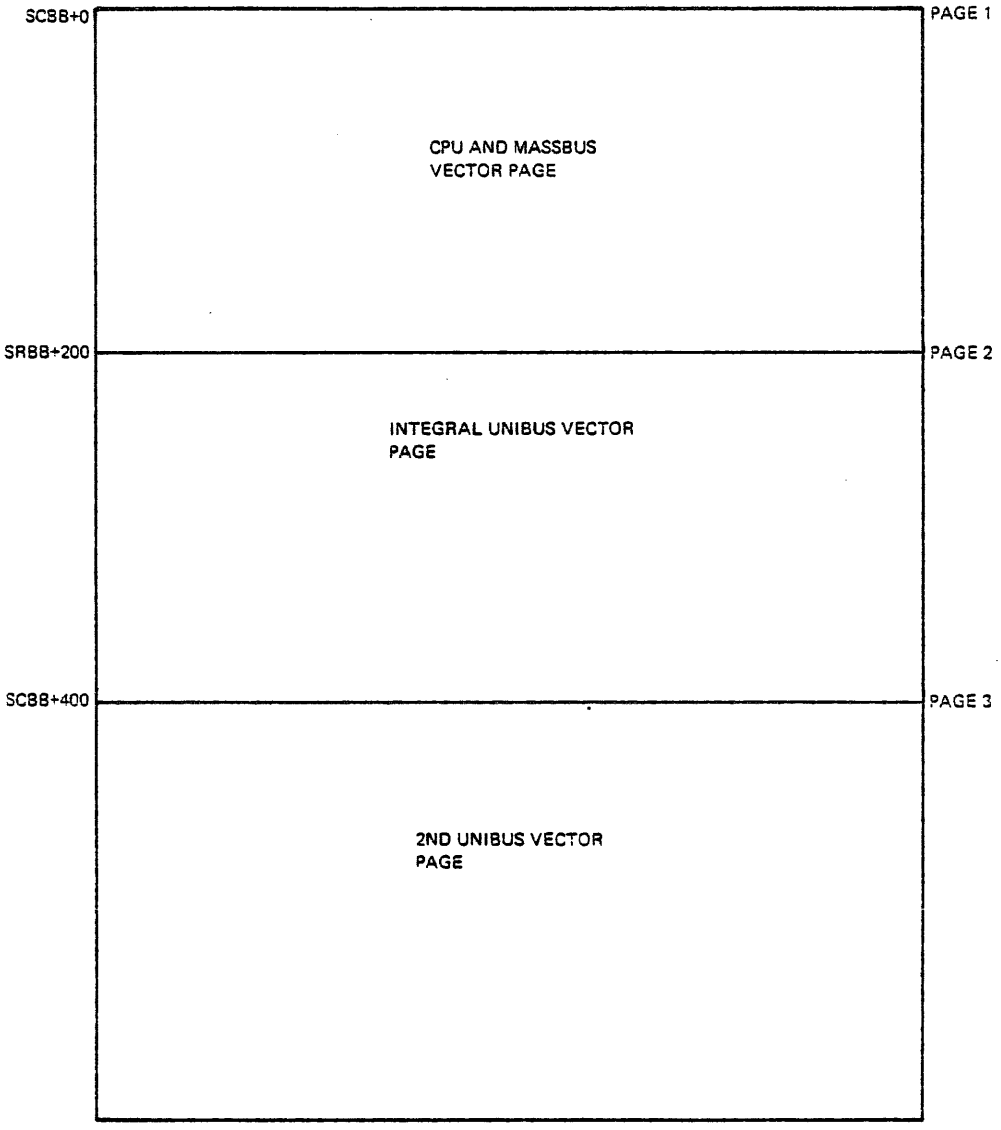

SCBB+0
PAGE 1
CPU AND MASSBUS
VECTOR PAGE
SRBB+200
PAGE 2
INTEGRAL UNIBUS VECTOR
PAGE
SCBB+400
PAGE 3
2ND UNIBUS VECTOR
PAGE
COMET SYSTEM CONTROL BLOCK
TK-1740

Figure 5-26

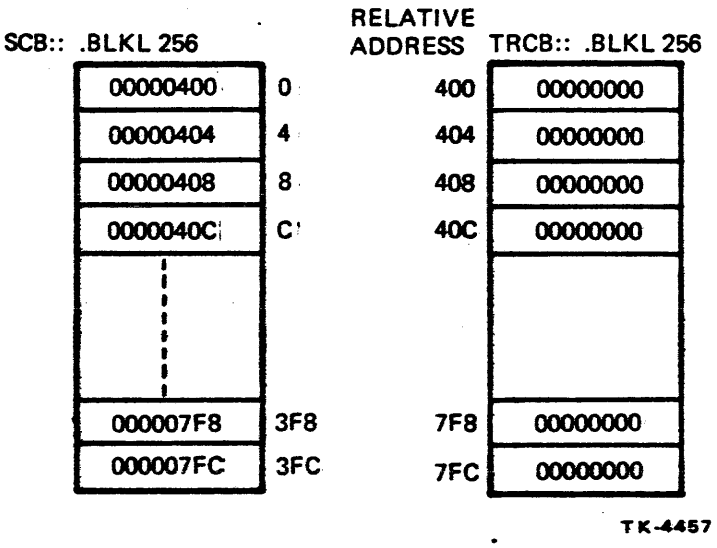
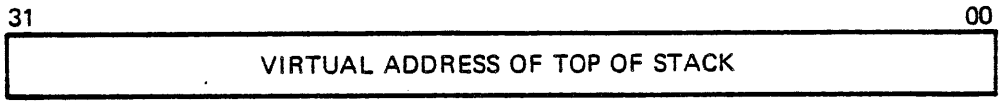

SCB:: .BLKL 256
00000400
00000404
00000408
0000040C
000007F8
000007FC
0
4
8
C
3F8
3FC
RELATIVE ADDRESS
400
404
408
40C
7F8
7FC
TRCB:: .BLKL 256
00000000
00000000
00000000
00000000
00000000
00000000
TK-4457

Figure 5-27

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



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

09 P0LR **P0 LENGTH REGISTER**

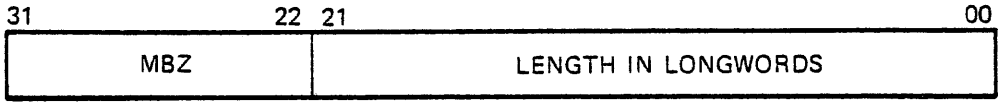
LENGTH OF P0PT IN LONGWORDS

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



TK-1750

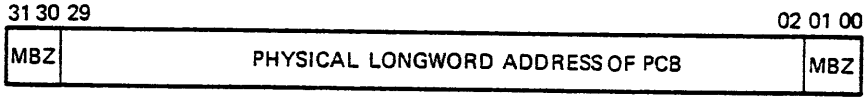
Figure 5-28

HEX NAME

10 PCBB

PROCESS CONTROL BLOCK BASE

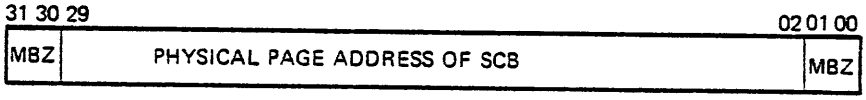
RESERVED OPERAND FAULT IF MBZ ≠ 0.



11 SCBB

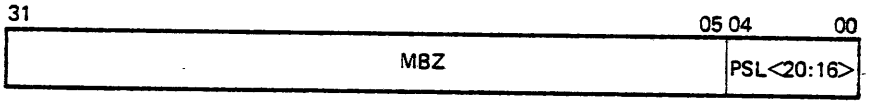
SYSTEM CONTROL BLOCK BASE

RESERVED OPERAND FAULT IF MBZ ≠ 0.



12 IPLR

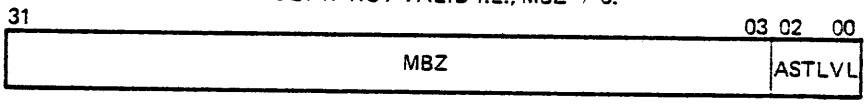
INTERRUPT PRIORITY LEVEL REGISTER



13 ASTR

AST LEVEL REGISTER

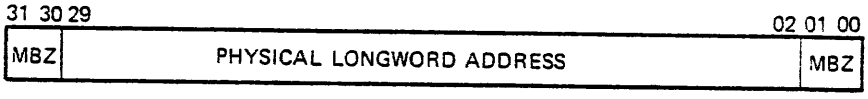
RESERVED OPERAND FAULT IF NOT VALID I.E., MBZ ≠ 0.



0C SBR

SYSTEM BASE REGISTER

RESERVED OPERAND FAULT IF MBZ ≠ 0.



TK-1753

Figure 5-29

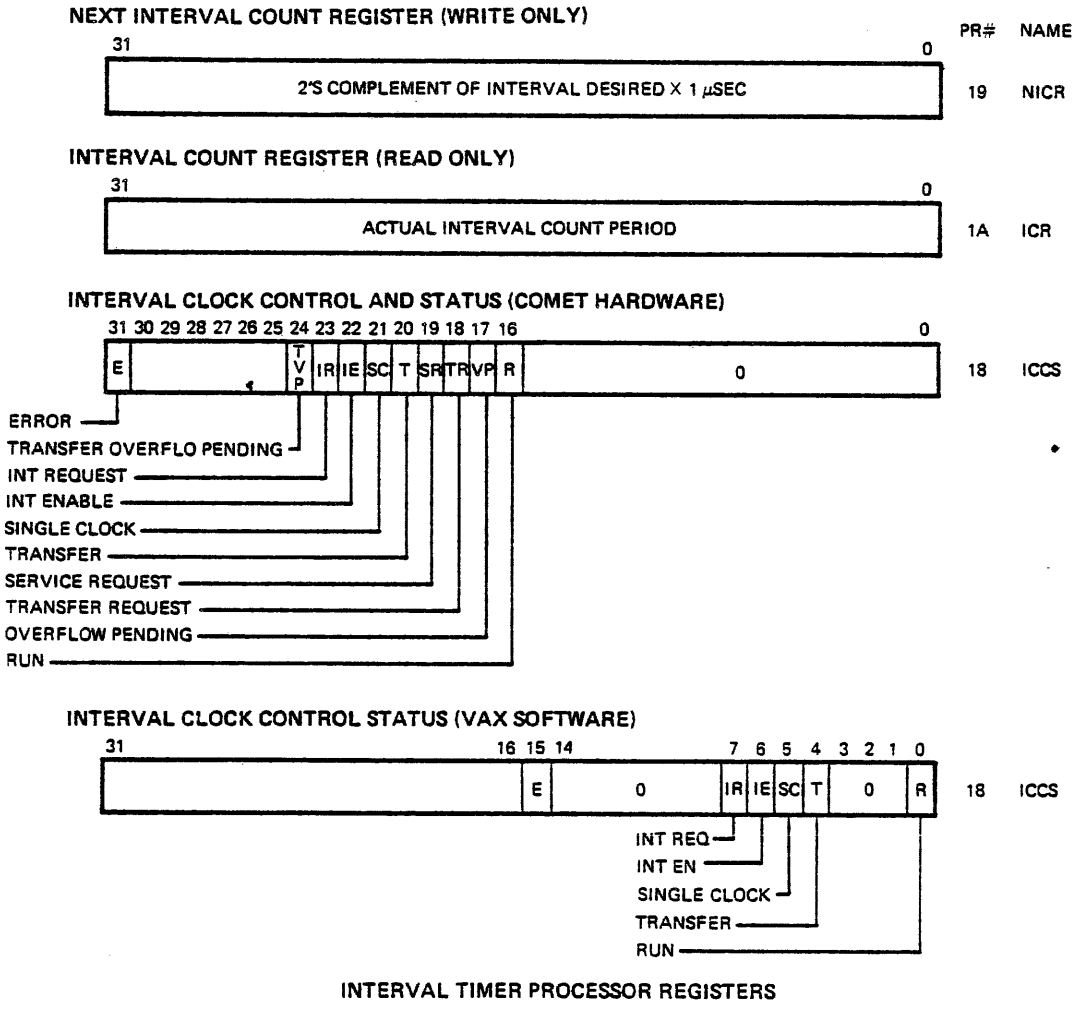

NEXT INTERVAL COUNT REGISTER (WRITE ONLY)
PR#
NAME
31
0
2'S COMPLEMENT OF INTERVAL DESIRED X 1 μSEC
19
NICR
INTERVAL COUNT REGISTER (READ ONLY)
31
0
ACTUAL INTERVAL COUNT PERIOD
1A
ICR
INTERVAL CLOCK CONTROL AND STATUS (COMET HARDWARE)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0
E
TVP
IR
IE
SC
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
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-1724

Figure 5-30

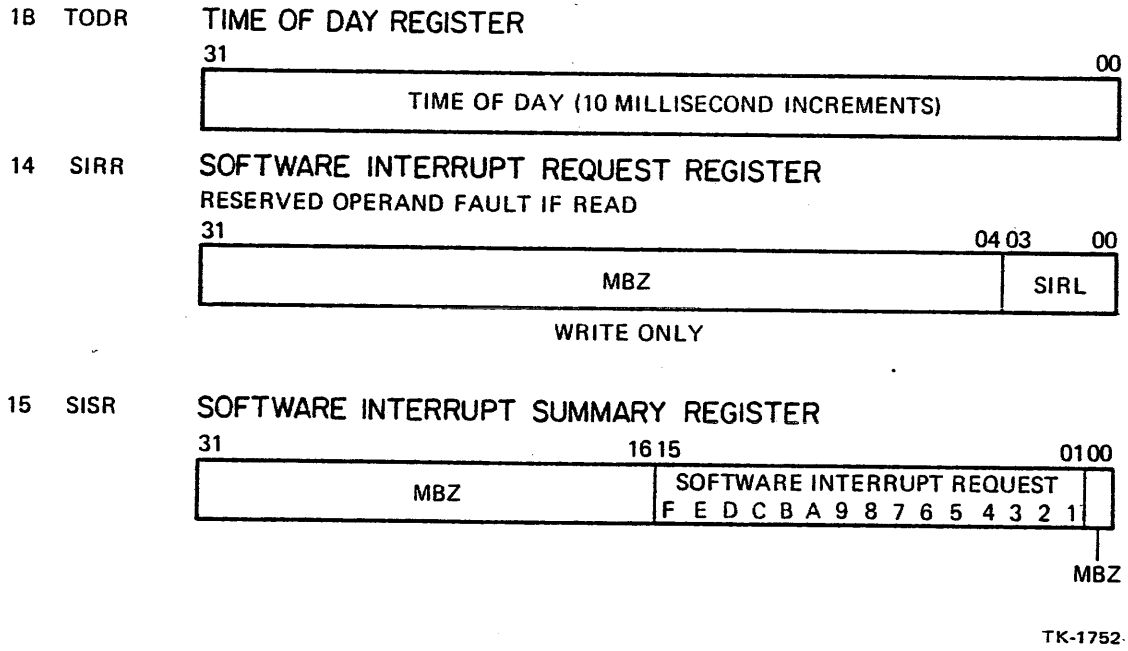
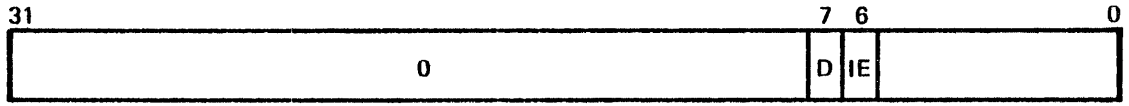

1B TODR
TIME OF DAY REGISTER
31
00
TIME OF DAY (10 MILLISECOND INCREMENTS)
14 SIRR
SOFTWARE INTERRUPT REQUEST REGISTER
RESERVED OPERAND FAULT IF READ
31
04 03
00
MBZ
SIRL
WRITE ONLY
15 SISR
SOFTWARE INTERRUPT SUMMARY REGISTER
31
16 15
01 00
MBZ
SOFTWARE INTERRUPT REQUEST
F E D C B A 9 8 7 6 5 4 3 2 1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

Figure 5-31

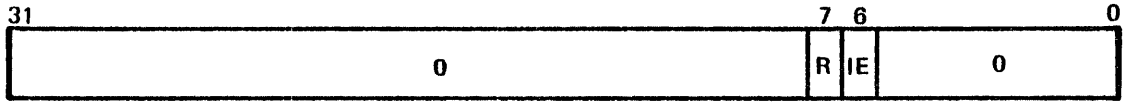
CONSOLE STORAGE RECEIVER STATUS



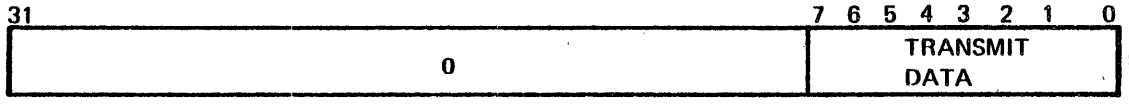
CONSOLE STORAGE RECEIVER DATA

RECEIVE FROM TU-58

CONSOLE STORAGE TRANSMIT STATUS



CONSOLE STORAGE TRANSMIT DATA

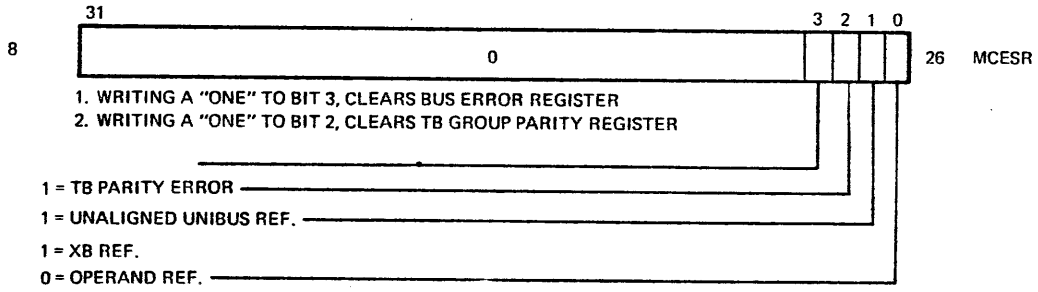


TRANSMIT TO TU-58

TK-1733

Figure 5-32

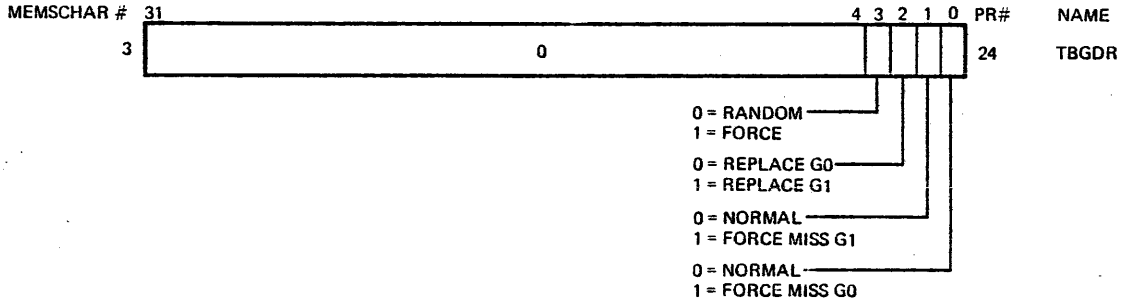
MACHINE CHECK ERROR SUMMARY REGISTER (REPORTS TYPE OF MACHINE CHECK)



BUS ERROR AND MACHINE CHECK ERROR AND SAVED MODE REGISTER

TK-1742

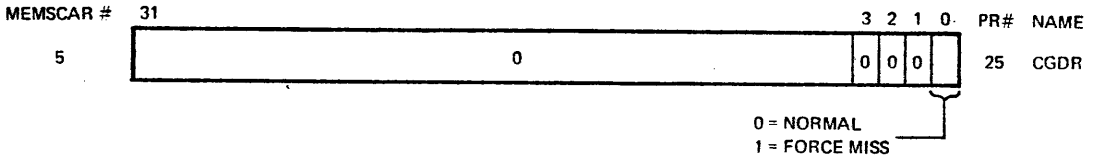
TRANSLATION BUFFER GROUP DISABLE REGISTER
(CONTROLES 2 WAY ASSOCIATIVE MEMORY OPERATION,
NORMAL LOADING IS RANDOM PLACEMENT OF DATA
IN PTE CACHE)



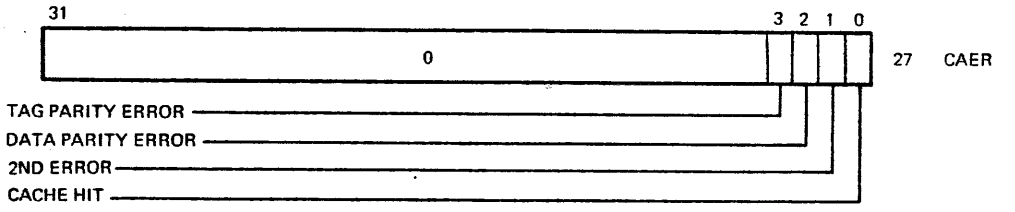
TRANSLATION BUFFER CONTROL AND STATUS REGISTERS

TK-1732

CACHE GROUP DISABLE REGISTER (CONTROLS REPLACEMENT OF DATA
INTO CACHE) 1 X 1K LONGWORD



CACHE ERROR REGISTER



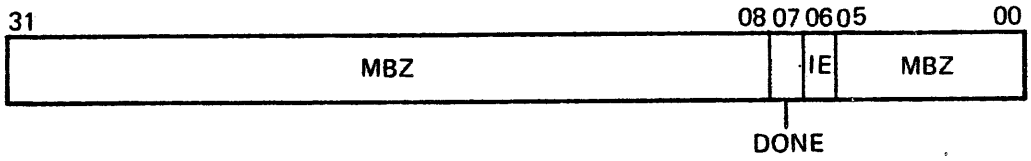
UNIBUS, CACHE CONTROL AND STATUS REGISTERS

TK-1734

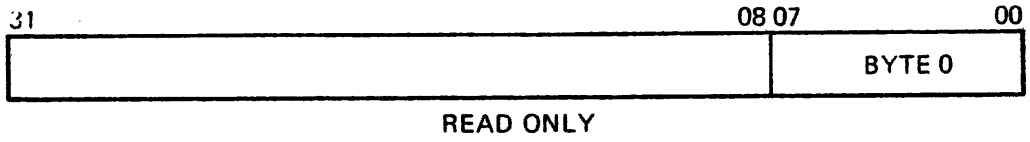
Figure 5-33

HEX NAME

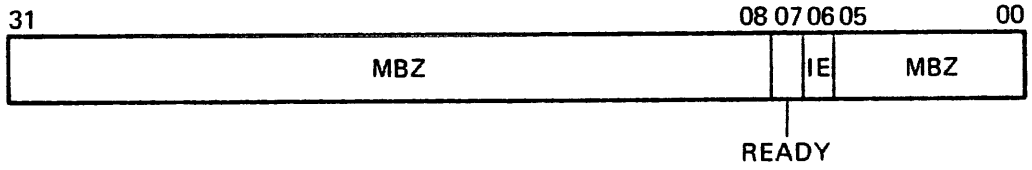
20 RXCS CONSOLE RECEIVE CONTROL/STATUS



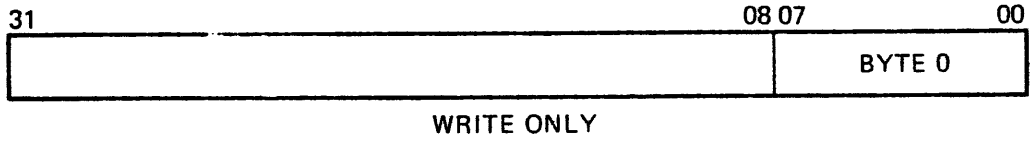
21 RXDB CONSOLE RECEIVE DATA BUFFER



22 TXCS CONSOLE TRANSMIT CONTROL/STATUS



23 TXDB CONSOLE TRANSMIT DATA BUFFER



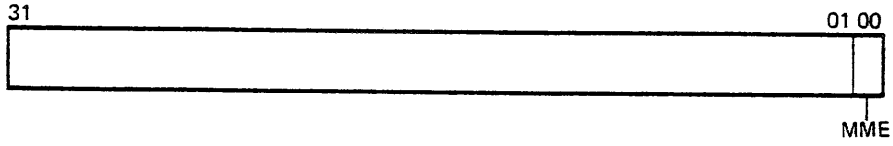
TK-1749

Figure 5-34

HEX NAME ID=

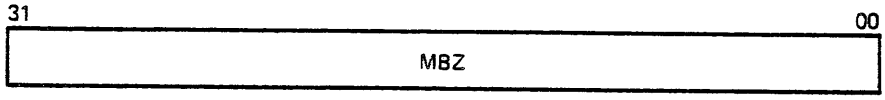
38 MME MEMORY MANAGEMENT ENABLE

WRITE 1 ALSO CAUSES MICROCODE TO INVALIDATE TB.



39 TBIA TRANSLATION BUFFER INVALIDATE ALL

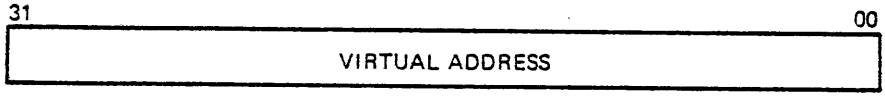
RESERVED OPERAND FAULT IF READ



WRITE ONLY

3A TBIS TRANSLATION BUFFER INVALIDATE SINGLE

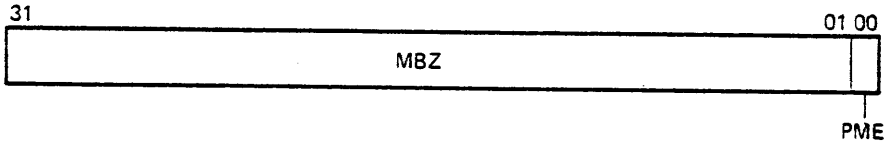
RESERVED OPERAND FAULT IF READ



WRITE ONLY

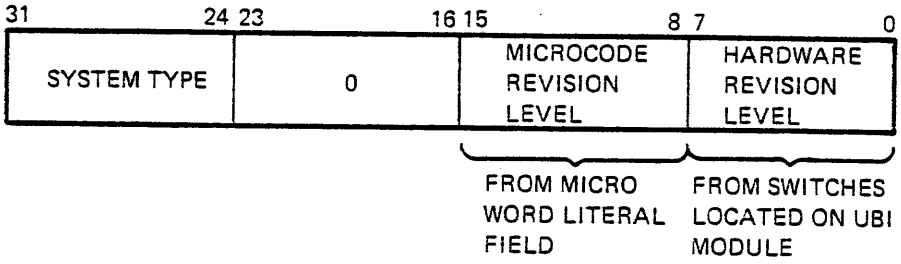
3D PMR PERFORMANCE MONITOR REGISTER

RESERVED OPERAND FAULT IF >1



3E SID **SYSTEM IDENTIFICATION (READ ONLY)**

RESERVED OPERAND FAULT IF WRITE



TK-2099

Figure 5-35

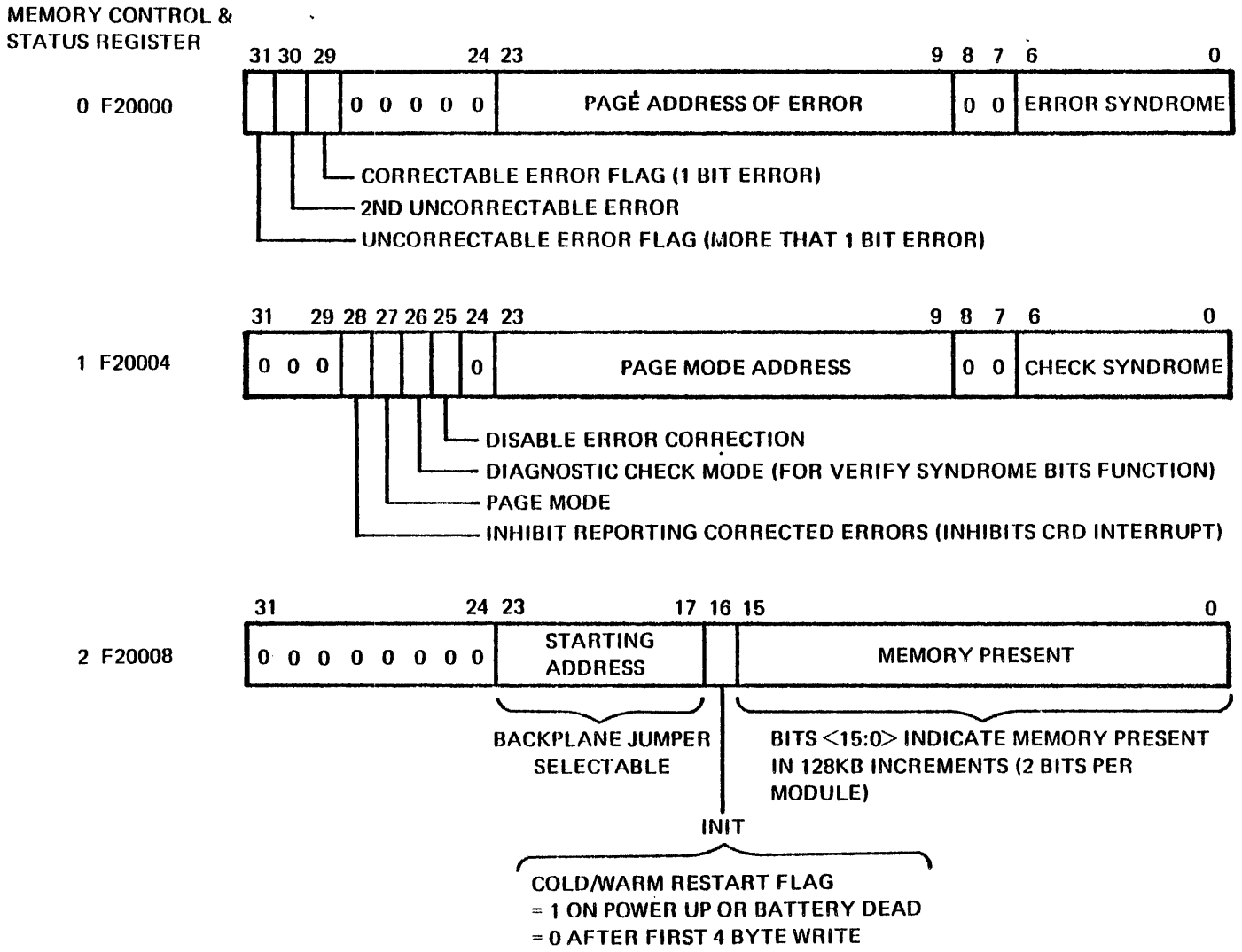

MEMORY CONTROL &
STATUS REGISTER
0 F20000
31 30 29
24 23
9 8 7 6
0
0 0 0 0 0
PAGE ADDRESS OF ERROR
0 0
ERROR SYNDROME
CORRECTABLE ERROR FLAG (1 BIT ERROR)
2ND UNCORRECTABLE ERROR
UNCORRECTABLE ERROR FLAG (MORE THAT 1 BIT ERROR)
1 F20004
31 29 28 27 26 25 24 23
9 8 7 6
0
0 0 0
0
PAGE MODE ADDRESS
0 0
CHECK SYNDROME
DISABLE ERROR CORRECTION
DIAGNOSTIC CHECK MODE (FOR VERIFY SYNDROME BITS FUNCTION)
PAGE MODE
INHIBIT REPORTING CORRECTED ERRORS (INHIBITS CRD INTERRUPT)
2 F20008
31
24 23
17 16 15
0
0 0 0 0 0 0 0 0
STARTING ADDRESS
MEMORY PRESENT
BACKPLANE JUMPER SELECTABLE
BITS <15:0> INDICATE MEMORY PRESENT IN 128KB INCREMENTS (2 BITS PER MODULE)
INIT
COLD/WARM RESTART FLAG
= 1 ON POWER UP OR BATTERY DEAD
= 0 AFTER FIRST 4 BYTE WRITE
TK-1725

Figure 5-36

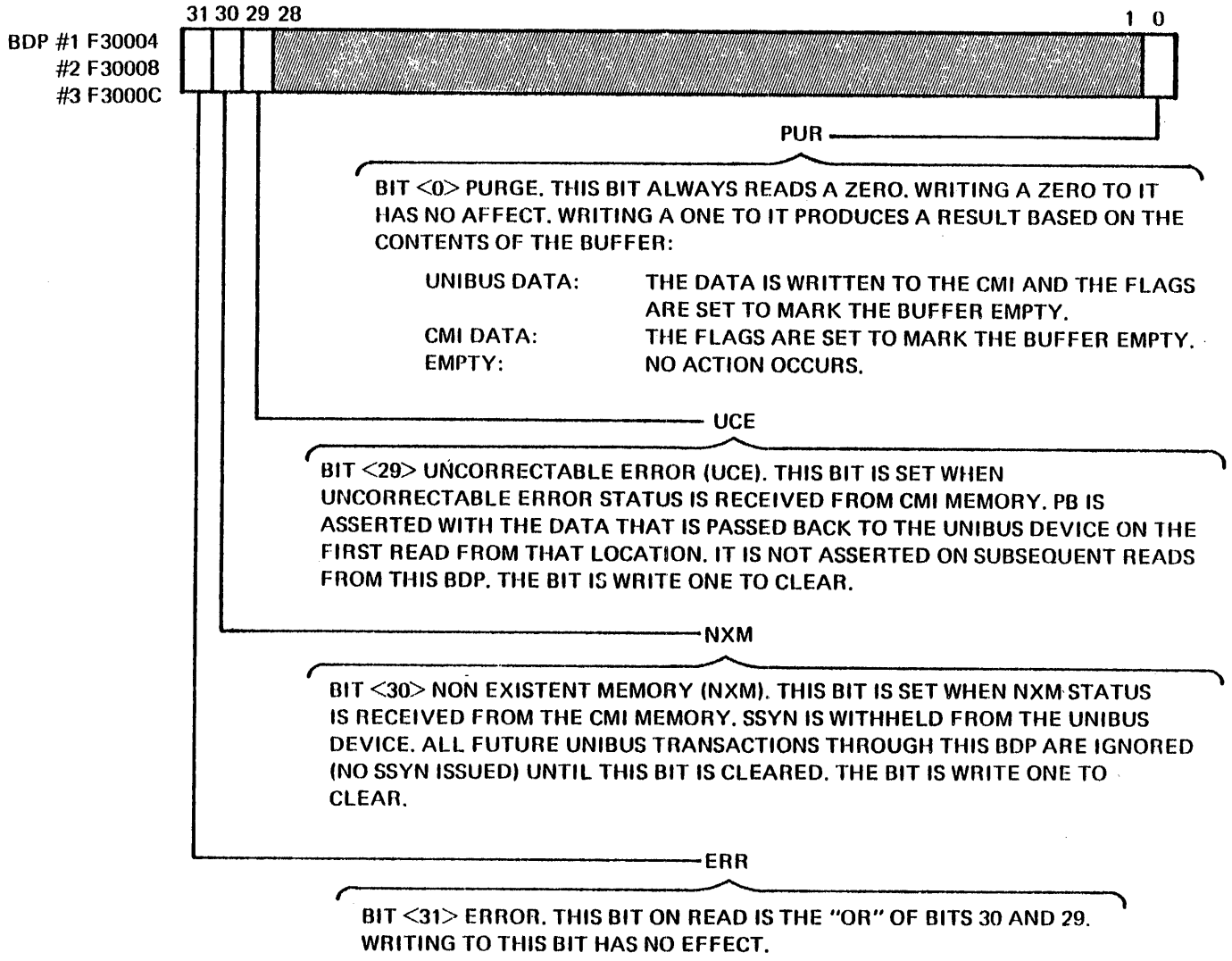
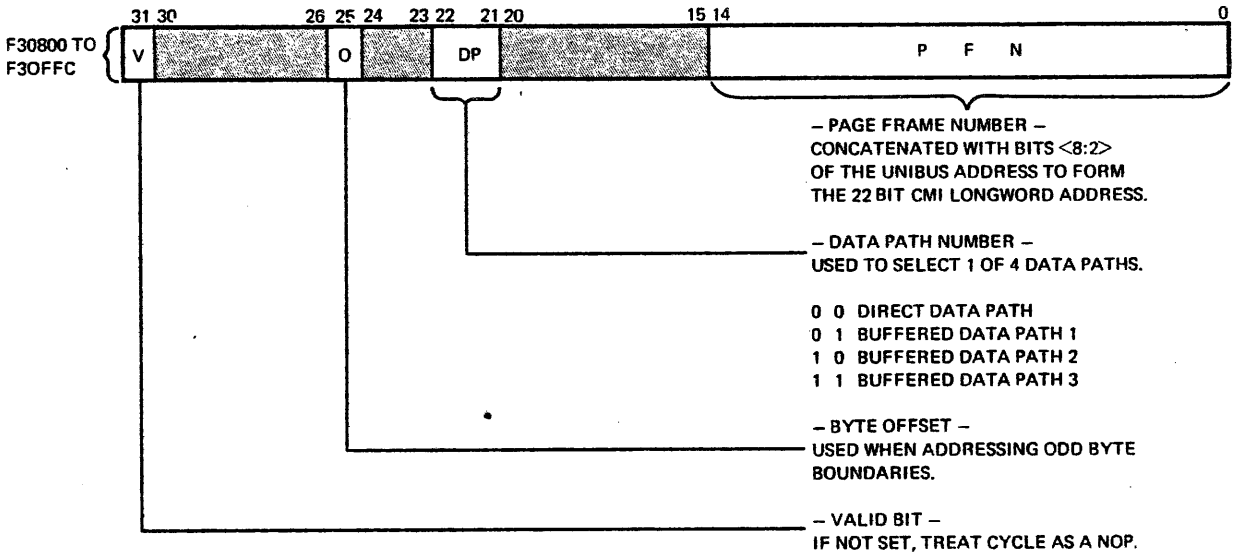
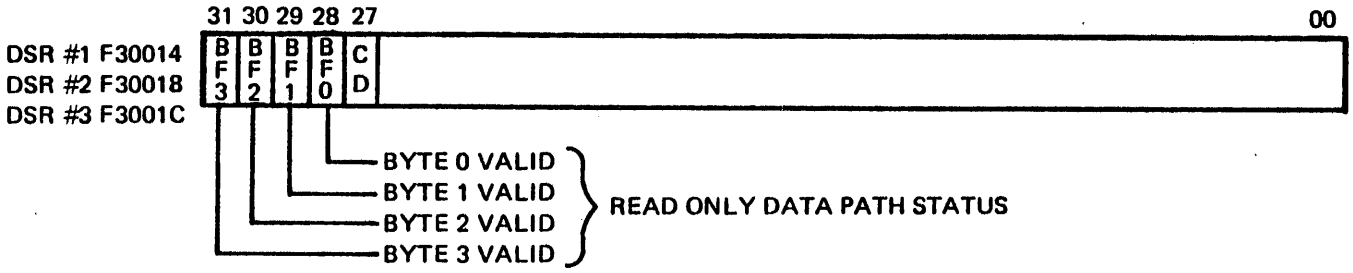

31 30 29 28
1 0
BDP #1 F30004
#2 F30008
#3 F3000C
PUR
BIT <0> PURGE. THIS BIT ALWAYS READS A ZERO. WRITING A ZERO TO IT HAS NO AFFECT. WRITING A ONE TO IT PRODUCES A RESULT BASED ON THE CONTENTS OF THE BUFFER:
UNIBUS DATA: THE DATA IS WRITTEN TO THE CMI AND THE FLAGS ARE SET TO MARK THE BUFFER EMPTY.
CMI DATA: THE FLAGS ARE SET TO MARK THE BUFFER EMPTY.
EMPTY: NO ACTION OCCURS.
UCE
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.
NXM
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.
ERR
BIT <31> ERROR. THIS BIT ON READ IS THE "OR" OF BITS 30 AND 29. WRITING TO THIS BIT HAS NO EFFECT.
BDP CONTROL AND STATUS REG.
TK-1727

Figure 5-37



UNIBUS TO CMI MAP DATA FIELDS ADDRESS

TK-1739

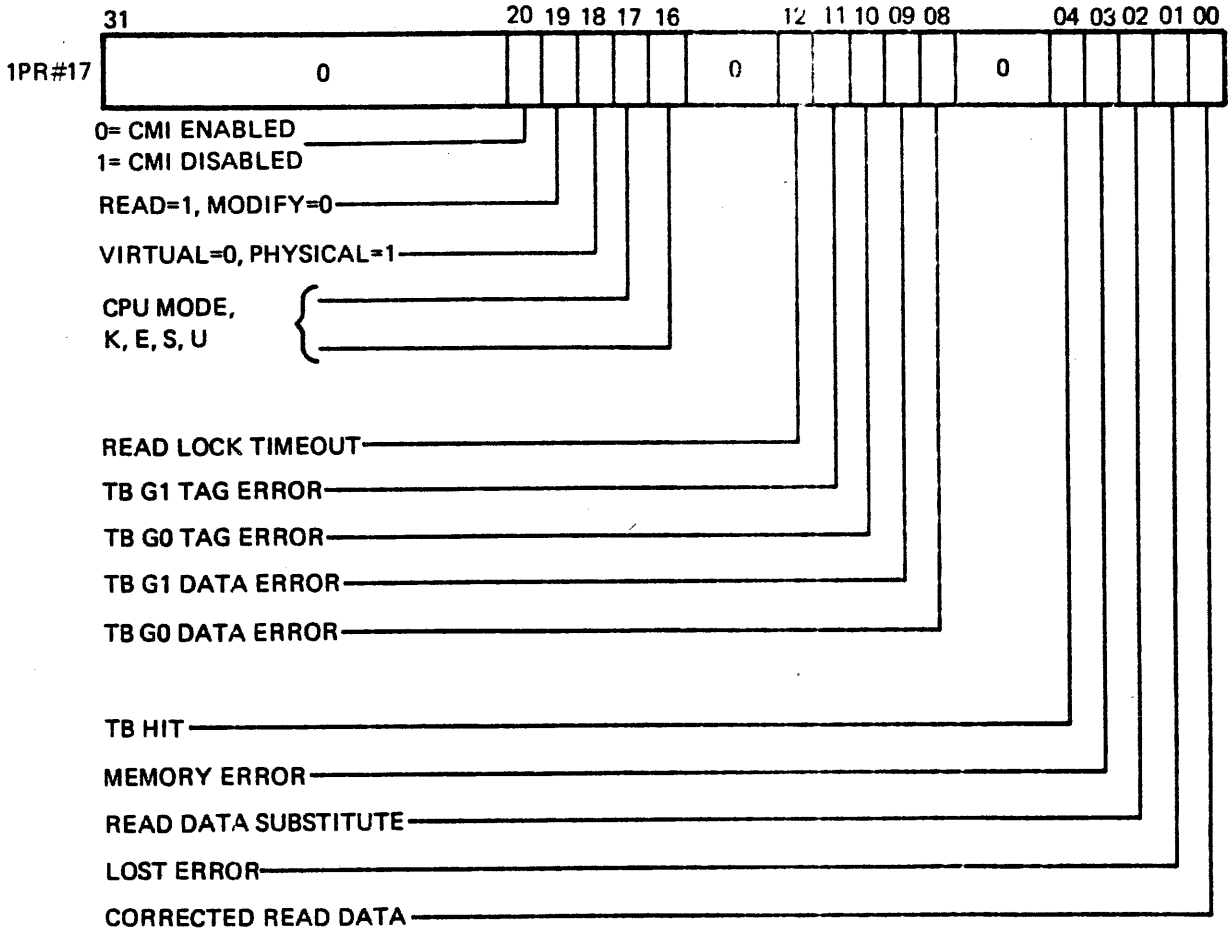


NOTE 1: THERE ARE FIVE FLAGS THAT KEEP TRACK OF THE DATA IN THE DATA BUFFER, NAMED CD AND BF3 THROUGH BF0. IF CD = 1, THEN THE BUFFER HAS FOUR BYTES OF DATA FROM THE CMI AND BF3 THROUGH BF0 ARE ALWAYS 0. IF CD = 0, THEN BF3 THROUGH BF0 INDICATE WHICH BYTES IN THE DATA BUFFER HAVE VALID UNIBUS DATA. IF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

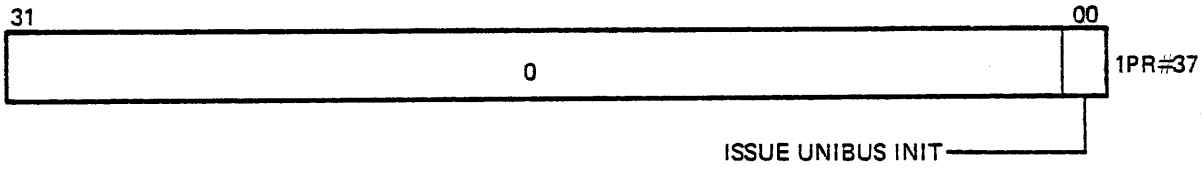
CUI DIAGNOSTIC STATUS REGISTER

TK-1726



CMI ERROR PROCESSOR REGISTER

TK-3266



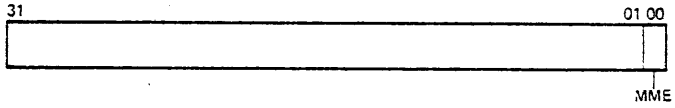
IO RESET PROCESSOR REGISTER

TK-3267

HEX NAME ID#

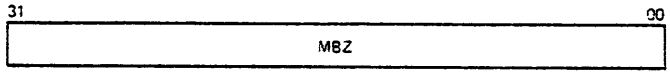
38 MME — MEMORY MANAGEMENT ENABLE

WRITE 1 ALSO CAUSES MICROCODE TO INVALIDATE TB.



39 TBIA — TRANSLATION BUFFER INVALIDATE ALL

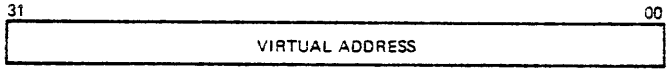
RESERVED OPERAND FAULT IF READ



WRITE ONLY

3A TBIS — TRANSLATION BUFFER INVALIDATE SINGLE

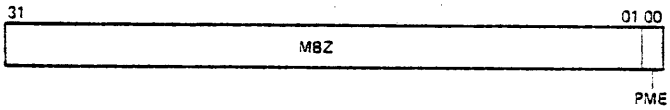
RESERVED OPERAND FAULT IF READ



WRITE ONLY

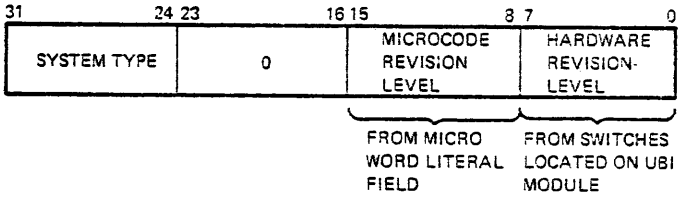
3D PMR — PERFORMANCE MONITOR REGISTER

RESERVED OPERAND FAULT IF >1



3E SID — SYSTEM IDENTIFICATION (READ ONLY)

RESERVED OPERAND FAULT IF WRITE



00 UNDEFINED	FROM MICRO WORD LITERAL FIELD	BACKPLANE JUMPERS
01 11/780		
10 11/750		
11 NEBULA		

TK-2099

Figure 5-41

VAX-11/750 LEVEL II

Microcode

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

OBJECTIVES

Utilizing the Comet microcode listing, correctly trace a microroutine for a specific machine function, listing only the microaddresses.

Provided with a Comet microcode macro expansion and a Comet microcode listing, write each of the field values for that microinstruction.

Provided with a schematic diagram, trace the origin and destination of a specific signal within the microsequencer logic.

Given a series of true/false questions, correctly indicate as true or false statements regarding the comet microsequencer operation.

Provided with a laboratory exercise procedure, the student will

a) load Microdiagnostics
b) operate the remote diagnosis module
c) trace microroutines
d) set up and trace selected signals

SAMPLE TEST ITEM

Referring to the CUI module schematic, drawing number 1 of 14, locate the signal called "RCS Ø1 CS AD 13 L". Trace the origin of this signal and all of the destinations in the space below.

LAB EXERCISE

a) load microdiagnostics
b) operate the RDM Module
c) trace microroutines
d) set up and trace selected signals

RESOURCES

1. COMET CPU Microcode listing

2. DPM module schematic

3. LSI Chip Schematic

   SAC
   MSQ
   PHB
   IRB

4. Wall charts

5. COMDEC, Microword decoding program

OUTLINE

I. INTRODUCTION TO MICROCODE

A. Why Microcode?
   1. Concepts
   2. Advantages

B. Comet CPU Microinstruction
   1. 80 bits
   2. Vertical functionality
   3. Microcode/hardware relationship
   4. Fields and functionality
   5. How is microcode created?

C. Summary

II. MICROCODE LISTINGS

A. Microprogrammers Code
B. Microcode Listing/Macrocode Listing Similarities
C. File Structure
D. Reading the Listing
   1. Assembler directives
   2. Addressing constraints
   3. Machine definitions
   4. Macro expansions
   5. Next address field
   6. CREF

E. Summary

III. MICROSEQUENCER AND CONTROL STORE SUBSYSTEM

A. Purpose
B. Cycle Time
C. LSI Chips
   1. SAC
   2. MSQ
   3. PHB
   4. IRD

E. CCS Interface
F. Block Diagram Analysis
G. CCS Module Block Diagram Analysis
H. Schematic Analysis
   1. Major addressing modes
   2. LSI chip functionality
   3. Timing

IV. CLASSROOM EXERCISE
LOCATE AND TRACE INIT MICROROUTINE

V. CPU CONSOLE MICROCODE

A. Console Emulator
B. Console Interface
C. Microroutines - Character by Character Parse
D. Console Functions
E. Console Functions Flow Diagram Analysis

VI. LABORATORY EXERCISE 6 - TBS

VII. LESSON SUMMARY

I. INTRODUCTION TO MICROCODE

A. Why Microcode?

1. Basic computer designs

   a. direct hardware decode of macro instruction requires elaborate timing and hardware design

   b. microprogrammed machine architecture allows general purpose design to be customized with ROM microcode.

2. Microcoding has created a demand for an individual that understands hardware and software to write microroutines

3. Microcode can repair design problems without changing hardware.

4. Microcode updates require changing only control store ROMS.

B. The advantages or microprogramming are clearly apparent to other hardware designs

C. COMET CPU Microinstruction

1. 8Ø bits wide
2. Vertical functionality
3. Microcode/hardware relationship

   a. DPM module - green
   b. UBI module - blue
   c. MIC module - yellow
   d. WCS module - red

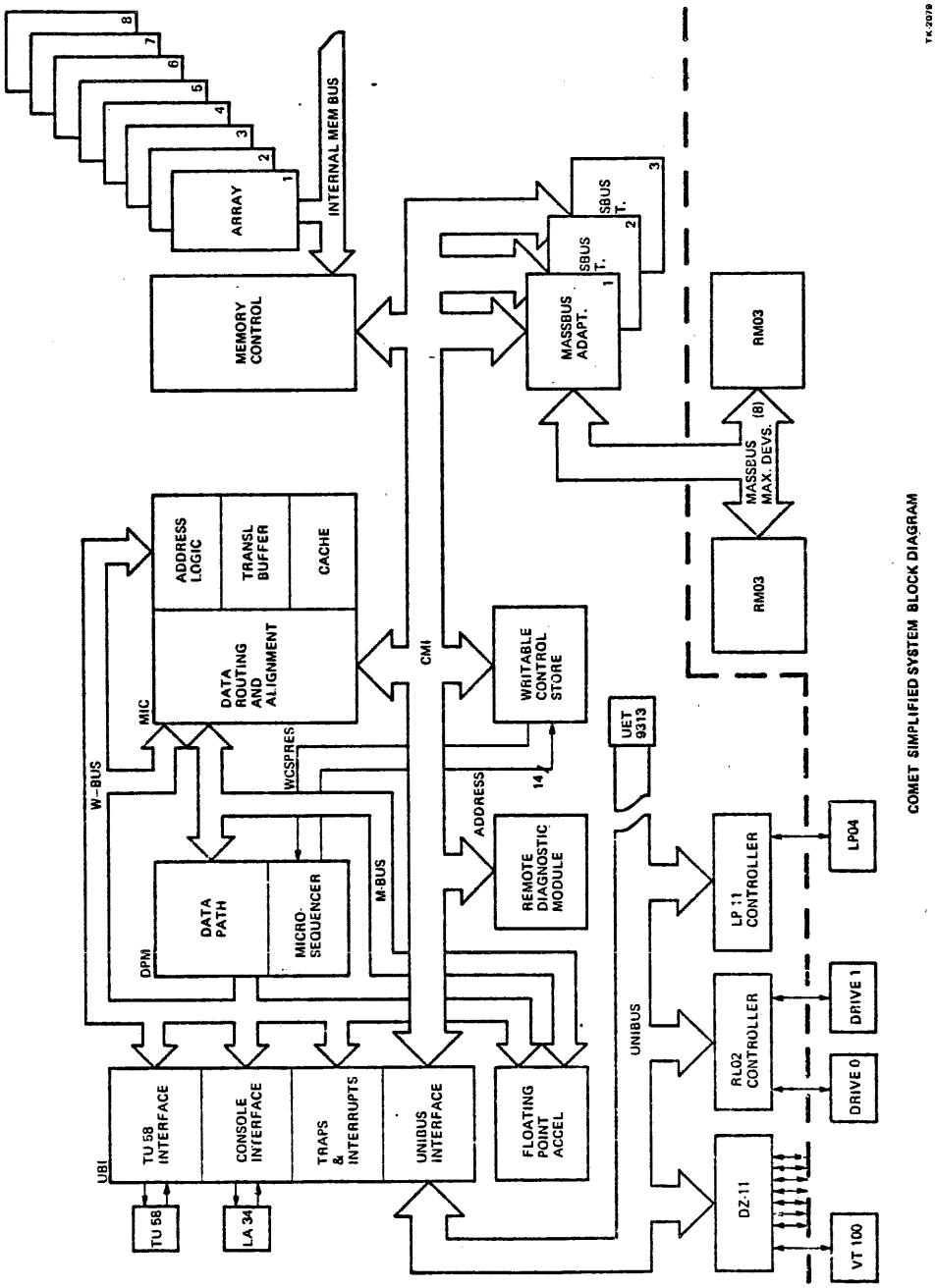

TK 2079
COMET SIMPLIFIED SYSTEM BLOCK DIAGRAM
INTERNAL MEM BUS
ARRAY
MEMORY CONTROL
SBUS T.
MASSBUS ADAPT.
RM03
MASSBUS MAX. DEVS. (8)
ADDRESS LOGIC
TRANSL BUFFER
CACHE
DATA ROUTING AND ALIGNMENT
MIC
CMI
WRITABLE CONTROL STORE
UET 9313
W-BUS
WCSPRES
ADDRESS
REMOTE DIAGNOSTIC MODULE
DATA PATH
MICRO-SEQUENCER
DPM
M-BUS
UNIBUS
LP 11 CONTROLLER
LP04
RL02 CONTROLLER
DRIVE 1
DRIVE 0
DZ-11
VT 100
UBI
TU 58 INTERFACE
CONSOLE INTERFACE
TRAPS & INTERRUPTS
UNIBUS INTERFACE
FLOATING POINT ACCEL
TU 58
LA 34

Figure 6-1

Figure 6-2

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)          14:40:3 9-Mar-1979        Page 30
; DEFIN .MIC [130,2112] DEFIN.MIC

                                ;969    .TOC    "DEFIN.MIC"
                                ;970    .TOC    "REVISION 53.3"
                                ;971    ;       P. R. GUILBAULT
                                ;972
;973    .NOBIN
;974    .RTOL
;975    .HEXADECIMAL
;976    .SOURCE/33
;977    .SET/NATIVE=1
;978    .NOCREF                 ;SET UP FOR CREF ONLY WHEN FULL ASSEMBLY
;979
;980    .TOC    "       Revision History"
;981
;982    ; 53    CORRECT WIDTH OF CCODE(Compatibility) ROM
;983    ;       ADD NEW MDR_0 CONFLICT WITH MSRC PER BINDER 05-MAR-79
;984    ;       ADD NEW MSRC/VA CONFLICT WITH BUS PER BINDER 15-FEB-79
;985    ; 52    CHANGE IRD1 ROM DEFINITION PER SMITH 02-FEB-79
;986    ;       ADD MICRO ORDERS FOR OPSPEC FIELDS IN IRD ROMS
;987    ;       CHANGE IRDX ROM DEFINITION PER SMITH 02-FEB-79
;988    ;       DELETE 'BUT/MBUS19TO18' PER LI 8-FEB-79
;989    ;       CHANGE CMODE ROM PER SMITH 02-FEB-79
;990    ; 51    UPDATE V023 & V025 AND DISABLE MSRC/TB PER BINDER 25-JAN-79
;991    ; 50    ADD VALIDITY CHECKS FOR IRD ROM'S
;992    ; 49    CHANGE 'MSRC/MTEMP11' TO 'MSRC/ERRCOD' TO REFLECT ITS PROPER USE
;993    ;       CHANGE VALIDITY CHECKS TO ALLOW 'CLRTB.VA_WB' WITH 'PRB.RD.PTE' WHEN NOT 'BUT/UVCTR'
;994    ;       CHANGE NOTATION FOR MULTIPLE VALIDITY CHECKS
;995    ; 48    DELETE 'WCTRL/VA_VAS_WB' PER BINDER 15-NOV-78
;996    ;       DELETE 'WXTRL/RDM' & 'WCTRL/RDM_WB' PER KRAUS 16-NOV-78
;997    ;       ADD 'WCTRL/FPA.ENABLE_WB5' PER KRAUS 12-OCT-78
;998    ;       CORRECT DESCRIPTION OF BRATST IN CC TABLE
;999    ;       ADD INFORMATION ABOUT CCBR, SRKSTA, AND SPASTA BUTS
;1000   ;       ADD VALIDITY CHECKS TO DEFIN AND DELETE VALID.MIC
;1001   ;           REVISION HISTORY FROM VALID :
;1002   ;                01      GET RID OF DUMMY CHECKS AND REMOVE COMENT STATUS OF CASE CHECKS
;1003   ;                        GET RID OF V003 BECAUSE IT IS IDENTICAL TO V001
;1004   ;                00      INITIAL RELEASE
;1005   ;       ADD 'WCTRL/MDR_0' AS CONFLICT WITH 'MSRC/MDR' PER BINDER 21-DEC-78
;1006   ;       ADD NEW CONFLICT FOR SOURCE PC OR PCBACK AND READ OR WRITE
;1007   ;       ADD NOTE TO BUT'S ON IR THAT ARE DIFFERENT IN COMPATIBILITY MODE
;1008   ;       CHANGE 'BUS vs MSRC' VALIDITY PER BINDER 29-DEC-78
;1009   ;       CORRECT COMMENTS FOR 'ROT/RL.RM.P', 'ROT/RL.RM.PS', 'ROT/ASL.R.P', & 'ROT/ASL.M.P'
;1010   ;       CORRECT 'CCPSL/MDR_OSR.CCBR_BRATST' & 'WCTRL/MDR_IR' MICRO ORDER ASSIGNMENTS PER SMITH 12-JAN-79
;1011   ;       ADD VALIDITY CHECKS FOR MULTLPLY AND DIVIDE SPECIAL FUNCTIONS
;1012   ; 47    CHANGE 'WCTRL/STEPC' TO 'WCTRL/CM.TP.FPD.F5.STEPC'
;1013   ;       CHANGE 'WCTRL/FLAGS' TO 'WCTRL/CM.TP.FPD.FLAGS'
;1014   ;       RENAME IRD ROM FIELDS TO BE MORE CONSISTANT WITH IRD ROM MACROS
;1015   ;       INCOPORATE CHANGES PER LI 14-DEC-78
;1016   ;        1.)    CHANGE 'WX_S.Q_D' TO 'WX_S.Q_0'
;1017   ;        2.)    CHANGE 'WX_D_S.Q_D' TO 'WX_D_S.Q_0'
;1018   ;        3.)    DELETE 'MUXDZ' FIELD
;1019   ;        4.)    DELETE 'DQ4' FIELD
;1020   ;        5.)    ADD NEW SPECIAL FUNCTIONS
;1021   ; 46    CHANGE ALL V003 TO V001
;1022   ;       CORRECT VALIDITY CHECK ON 'CCPSL/MDR_OSR.CCBR_BRATST'
;1023   ; 45    CORRECT COMPATABILITY MODE IRD ROM DEFINITION
```

ENTIRE LISTING NAME

SUBFILE NAME

LINE NUMBER

DO NOT LIST BINARY OUTPUT

NUMBERS ARE FROM RIGHT TO LEFT FOR BINARY

RADIX IS HEX

VAX NATIVE INSTRUCTION MICROCODE

DO NOT CROSS REFERENCE FOLLOWING CODE

INSERT TEXT IN TABLE OF CONTENTS

MICRO2 ASSEMBLER DIRECTIVES 1

Figure 6-3

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979
; DEFIN .MIC [130,2112]          Machine Definition              : IRD1 ROM

;1961   .TOC "  Machine Definition              : IRD1 ROM"
;1962   .ICODE
;1963   .WIDTH/32
;1964
;1965
;1966
;1967   ; +-+-+-+-------------+-+-------------+-+-------------+-+-------------+
;1968   ; |V|V|I|             |I|             |F|             |F|             |
;1969   ; |I|F|F|  IRD1.FPA   |O|     IRD1    |F|   FPD.FPA   |O|     FPD     |
;1970   ; |R|P|O|             |P|             |O|             |P|             |
;1971   ; |D|D|P|             | |             |P|             | |             |
;1972   ; |1| | |             | |             | |             | |             |
;1973   ; +-+-+-+-------------+-+-------------+-+-------------+-+-------------+
;1974   ; |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|
;1975   ; |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|
;1976   ; +-+-+-+-------------+-+-------------+-+-------------+-+-------------+
;1977
;1978   FPD      /=<6:0>
;1979   FPD.FPA /=<14:8>
;1980   IRD1     /=<22:16>
;1981   IRD1.FPA/=<30:24>
;1982
;1983   FOP /=<07:07>
;1984           NOP=0
;1985           LOD=1
;1986   FFOP/=<15:15>
;1987           NOP=0
;1988           LOD=1
;1989   IOP /=<23:23>
;1990           NOP=0
;1991           LOD=1
;1992   IFOP/=<31:31>
;1993           NOP=0
;1994           LOD=1
;1995
;1996   VFPD /=<32:32>,          .VALIDITY=<V060>
;1997   VIRD1/=<33:33>,          .VALIDITY=<V061>
```

SWITCH FROM DEFAULT ROM (U) AND DEFINE IRD ROM AS 32 BITS WIDE.

MICRO2 ASSEMBLER DIRECTIVES 2

Figure 6-4

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                    Page 54
; DEFIN .MIC [130,2112]          Machine Definition                  : IRDX ROM

;1998   .TOC "  Machine Definition                  : IRDX ROM"
;1999   .OCODE <------------------------------------------------------SWITCH FROM DEFAULT ROM (U) TO OSR ROM AND DEFINE BITS AS 96 BIT WORD.
;2000   .WIDTH/96
;2001
;2002
;2003   ;      +---+-------------------+-------------------+---+-------------------+-------------------+
;2004   ;      | 0 |                   |                   | 0 |                   |                   |
;2005   ;      | O |                   |                   | F |                   |                   |
;2006   ;      | P |   CNT0.REG        |   CNT0.MEM        | O |   CNT0.FPA.REG    |   CNT0.FPA.MEM    |
;2007   ;      |   |                   |                   | P |                   |                   |
;2008   ;      +---+-------------------+-------------------+---+-------------------+-------------------+
;2009   ;      |4 4|4 4 4 4 4 4 3 3 3 3|3 3 3 3 3 2 2 2 2 2|2 2|2 2 1 1 1 1 1 1 1 1|1 0 0 0 0 0 0 0 0 0|
;2010   ;      |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|
;2011   ;      +---+-------------------+-------------------+---+-------------------+-------------------+
;2012   ;
;2013   ;
;2014   ; +-+-+---+-------------------+-------------------+---+-------------------+-------------------+
;2015   ; |V|V| 1 |                   |                   | 1 |                   |                   |
;2016   ; |C|C| O |                   |                   | F |                   |                   |
;2017   ; |N|N| P |   CNT1.REG        |   CNT1.MEM        | O |   CNT1.FPA.REG    |   CNT1.FPA.MEM    |
;2018   ; |T|T|   |                   |                   | P |                   |                   |
;2019   ; |1|0|   |                   |                   |   |                   |                   |
;2020   ; +-+-+---+-------------------+-------------------+---+-------------------+-------------------+
;2021   ; |9|9|9 9|9 9 9 9 8 8 8 8 8 8|8 8 8 7 7 7 7 7 7 7|7 7|6 6 6 6 6 6 6 6 6 5|5 5 5 5 5 5 5 5 4 4|
;2022   ; |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|
;2023   ; +-+-+---+-------------------+-------------------+---+-------------------+-------------------+
;2024
;2025   CNT0.FPA.MEM/=<10:0>
;2026   CNT0.FPA.REG/=<21:11>,   .VALIDITY=<V062>
;2027   0FOP/=<23:22>
;2028           NOP=0
;2029           LOD=3
;2030
;2031   CNT0.MEM/=<34:24>
;2032   CNT0.REG/=<45:35>,       .VALIDITY=<V063>
;2033   0OP /=<47:46>
;2034           NOP=0
;2035           LOD=3
;2036
;2037   CNT1.FPA.MEM/=<58:48>
;2038   CNT1.FPA.REG/=<69:59>,   .VALIDITY=<V064>
;2039   1FOP/=<71:70>
;2040           NOP=0
;2041           LOD=3
;2042
;2043   CNT1.MEM/=<82:72>
;2044   CNT1.REG/=<93:83>,       .VALIDITY=<V065>
;2045   1OP /=<95:94>
;2046           NOP=0
;2047           LOD=3
;2048
;2049   VCNT0/=<96:96>,          .VALIDITY=<V066>
;2050   VCNT1/=<97:97>,          .VALIDITY=<V067>
```

MICRO2 ASSEMBLER DIRECTIVES 3

Figure 6-5

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979
; INIT  .MIC [130,2112]          Initialize Microcode for the Console and Power up

                                ;3595  .TOC "  Initialize Microcode for the Console and Power up"
                                ;3596  0:
                                ;3597  IN.INIT:
                                ;3598          ;-------------------------------;
                                ;3599          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
                                ;3600          CLEAR FLAG2,                    ;PROCESS INIT CLEAR FLAG2
U 0000, 7100,7DF0,7FFF,8450,083E ;3601          NEXT/IN.PSL.LONLIT              ;GOTO REG FLOW
                                ;3602
                                ;3603  .REGION/<INIT.R1L>,<INIT.R1H>/<INIT.R2L>,<INIT.R2H>/<INIT.R3L>,<INIT.R3H>
                                ;3604
                                ;3605  IN.PC_0:
                                ;3606          ;-------------------------------;
                                ;3607          PC_R[ZERO],                     ;PC GETS 0
                                ;3608          CLEAR FLAG1,                    ;FOR CHARLIE'S CLEAR TB SUBR
U 0818, 0080,56E4,08D8,4850,0001 ;3609          RETURN [1]                      ;RETURN+1
                                ;3610
                                ;3611  IN.VA_0:
                                ;3612          ;-------------------------------;
                                ;3613          VA_R[ZERO],                     ;VA GETS 0
U 0825, 4800,58E4,08D8,4A50,0001 ;3614          RETURN [1]                      ;RETURN+1
                                ;3615
                                ;3616  IN.CN.INIT:
                                ;3617          ;-------------------------------;
                                ;3618          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
U 0839, 3500,7DF0,7FFF,8450,083E ;3619          SET FLAG2                       ;PROCESS INIT CLEAR FLAG2
                                ;3620
                                ;3621  IN.PSL.LONLIT:
                                ;3622          ;-------------------------------;
U 083E, 0000,58F4,03D4,0050,0820 ;3623          PSL_R[LONLIT].CLEAR FLAG0       ;PSL GETS LONLIT CLEAR FLAG0
                                ;3624
                                ;3625  =0
                                ;3626          ;0------------------------------;
                                ;3627          PUSH,                           ;JSR
                                ;3628          STEPC_2.                        ;
                                ;3629          CRAR_ZLITO[00].                 ;CRAR GETS 2
U 0820, 5A00,C370,0340,2450,481B ;3630          NEXT/TM\PO_0                    ;NOW IF WE CONWRITE
                                ;3631                                          ;WE WILL WRITE TO RXCS
                                ;3632
                                ;3633          ;1------------------------------;
                                ;3634          CONREGS_D_M[SISR]_R[ZERO],      ;RXCS GETS 0
U 0821, 89EF,58E6,03D8,2C50,08A0 ;3635          DEC STEPC                       ;SISR GETS 0
                                ;3636
                                ;3637
```

(0) LOCATE IN MACRO FILE

REGION DIRECTIVE

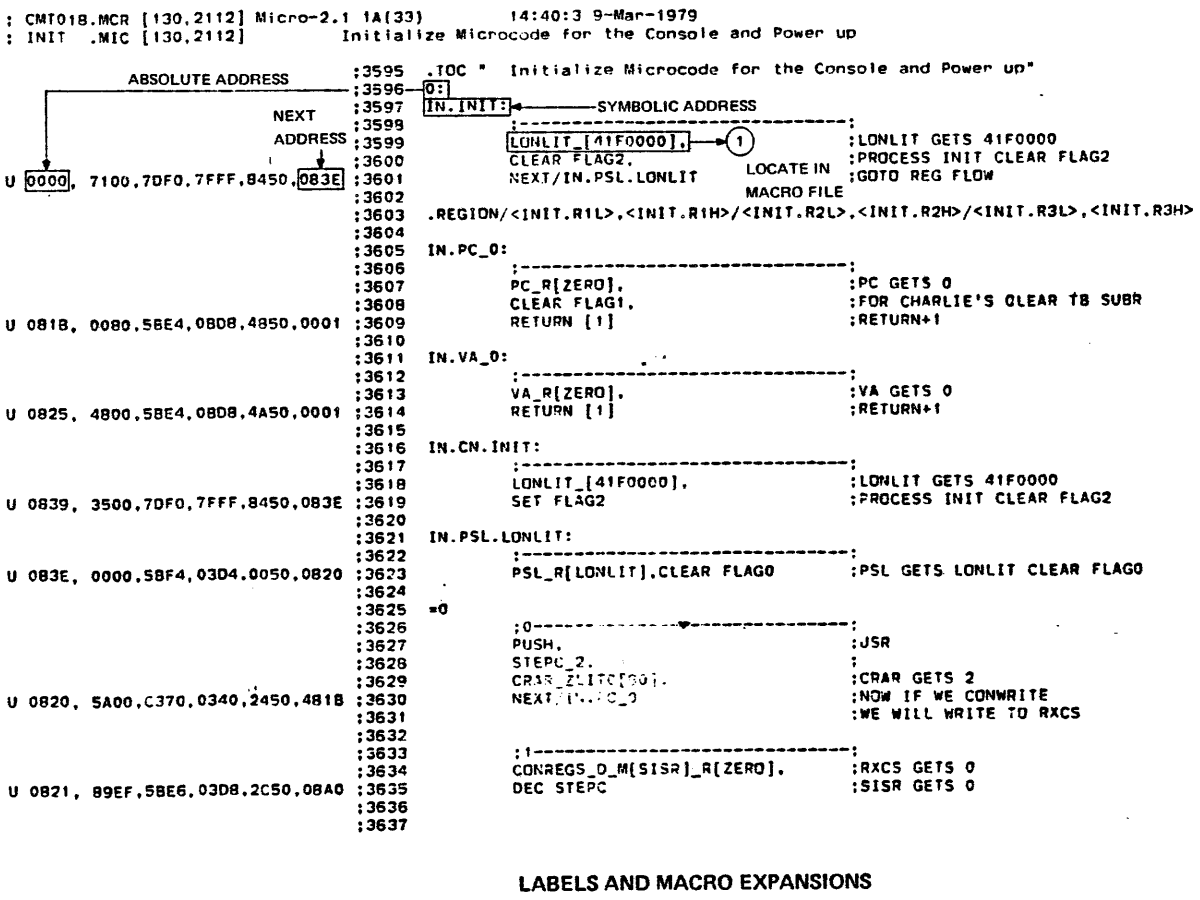
Figure 6-6

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979
; MACRO .MIC [130,2112]          Control Store Region Expressions

;2311    .TOC     "       Control Store Region Expressions"
;2312
;2313    ;Initialize
;2314              .SET/INIT.R1L=0800
;2315              .SET/INIT.R1H=0FCF
;2316        (0)-> .SET/INIT.R2L=0400
;2317              .SET/INIT.R2H=07FF
;2318              .SET/INIT.R3L=0000
;2319              .SET/INIT.R3H=03FF
;2320
;2321    ;Console
;2322              .SET/CONSOL.R1L=0800
;2323              .SET/CONSOL.R1H=0FCF
;2324              .SET/CONSOL.R2L=0400
;2325              .SET/CONSOL.R2H=07FF
;2326              .SET/CONSOL.R3L=0000
;2327              .SET/CONSOL.R3H=03FF
;2328
;2329    ;Integer, Logical, and Address
;2330              .SET/INTLOG.R1L=0800
;2331              .SET/INTLOG.R1H=0FCF
;2332              .SET/INTLOG.R2L=0400
;2333              .SET/INTLOG.R2H=07FF
;2334              .SET/INTLOG.R3L=0000
;2335              .SET/INTLOG.R3H=03FF
;2336
;2337    ;Floating Point and CRC
;2338              .SET/FLOAT.R1L=0800
;2339              .SET/FLOAT.R1H=0FCF
;2340              .SET/FLOAT.R2L=0400
;2341              .SET/FLOAT.R2H=07FF
;2342              .SET/FLOAT.R3L=0000
;2343              .SET/FLOAT.R3H=03FF
;2344
;2345    ;Variable Length Bit Field
;2346              .SET/VIELD.R1L=0800
;2347              .SET/VIELD.R1H=0FCF
;2348              .SET/VIELD.R2L=0400
;2349              .SET/VIELD.R2H=07FF
;2350              .SET/VIELD.R3L=0000
;2351              .SET/VIELD.R3H=03FF
;2352
;2353    ;Control Instructions
;2354              .SET/CONTRL.R1L=0800
;2355              .SET/CONTRL.R1H=0FCF
;2356              .SET/CONTRL.R2L=0400
;2357              .SET/CONTRL.R2H=07FF
;2358              .SET/CONTRL.R3L=0000
;2359              .SET/CONTRL.R3H=03FF
```

REGION DIRECTIVE MACROS

Figure 6-7



LABELS AND MACRO EXPANSIONS

Figure 6-8

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                    Page 105
; CONSOL.MIC [130,2112]          Console                          : CONSOLE EXAMINE AND DEPOSIT

                                  ;4192  ;*****************************************************************************
                                  ;4193  ;                          FINISHING OFF E OR D
                                  ;4194  ;*****************************************************************************
                                  ;4195
                                  ;4196  =00 CONSTRAINT BLOCK = 4 WORDS
                                  ;4197  CN.E.FLAG0:
                                  ;4198          ;00-----------------------------;
                                  ;4199          FLAG0?,                         ;EXAMINE????
                                  ;4200          R[TEMP12]_RNUM,                 ;FOR POSSIBLE BUMPING RNUM LATER
U 0930, 0856,0364,4330,0450,08C4  ;4201          NEXT/CN.E.FLAG0.CLR             ;
                                  ;4202
                                  ;4203  CN.E.IS.IT.A.SP:
                                  ;4204          ;01-----------------------------;
                                  ;4205          PUSH,                           ;GET NEXT CHARACTER
                                  ;4206          M[TEMP2]_ZLIT0[0],              ;TEMP2 GETS 0
U 0931, 1862,C370,0300,0450,4A9C  ;4207          NEXT/CN.GET.NEXT.CHAR           ;RET+1
                                  ;4208
                                  ;4209          ;10-----------------------------;
U 0932, 4800,0364,4300,0450,08A4  ;4210          FLAG0?                          ;EXAMINE????
                                  ;4211
U0933  NOT USED                   ;4212
                                  ;4213  =  END PREVIOUS BLOCK
                                  ;4214  =0 <------------------ CONSTRAINT BLOCK = 2 WORDS
                                  ;4215          ;0------------------------------;
                                  ;4216          WB_M[TEMP6]-ZLIT0[2A],          ;IS IT AN *
                                  ;4217          WX.EQ.0?,                       ;NO  --*****************DEPOSIT********
U 08A4, 5806,C100,A315,0450,0860  ;4218          NEXT/CN.NOT.ASTERICK            ;
                                  ;4219
                                  ;4220          ;1------------------------------;
                                  ;4221          WB_M[TEMP6]-ZLIT0[2A],          ;IS IT AN *
                                  ;4222          WX.EQ.0?,                       ;NO  --*****************EXAMINE********
U 08A5, 5806,C100,A315,0450,09A8  ;4223          NEXT/CN.E.NOT.ASTERICK          ;
                                  ;4224
                                  ;4225  =000 <------------------ CONSTRAINT BLOCK = 8 WORDS
                                  ;4226  CN.NOT.ASTERICK:
                                  ;4227          ;000----------------------------;
                                  ;4228          WB_M[TEMP6]-ZLIT0[50],          ;NOT *-IS IT A P????
                                  ;4229          WX.NE.0?,                       ;
U 0860, 1806,C100,A728,0450,0800  ;4230          NEXT/CN.D.SL.P                  ;
                                  ;4231
                                  ;4232          ;001----------------------------;
                                  ;4233          PUSH,                           ;PROCESS NUMBER(S)
                                  ;4234          M[TEMP8]_ZLIT0[0D],             ;1 NUMBER TO PROCESS
                                  ;4235          SET FLAG1,                      ;FOR PROCNO
U 0861, 14E8,C370,0306,8450,4800  ;4236          NEXT/CN.PROCNO                  ;RET+4 SO RETURN TO 101(CN.D.ASTERICK)
                                  ;4237
                                  ;4238  =101 <------------------ LOCATION 5 OF 8 WORD BLOCK
                                  ;4239          ;101----------------------------;
                                  ;4240          R[TEMP1]_M[VA],                 ;RESTORE TEMP1
U 0865, C85B,5924,0304,0450,09AB  ;4241          NEXT/CN.D.ASTERICK              ;
```

ADDRESSING CONSTRAINTS

Figure 6-9

```
;1166    .TOC "  Machine Definition                : ALPCTL"
;1167
;1168    ALPCTL/=<57:48>,.DEFAULT=364              ;ALP SPECIAL FUNCTIONS                                          ;ALU OPERATION FOR
;1169            NOP=364                                    ;ALUOD/OR,MUX/Z.S,DQ1/NOP                               ;SETTING OF ALU FLAGS
;1170            WX_D_Q.Q_D=2D7                             ;WMUX & D <- Q OLD      Q <- D OLD                       D+Q+CI.BCD
;1171            WX_D_Q.Q_M=0D7                             ;WMUX & D <- Q OLD      Q <- MBUS                        M+Q+CI.BCD
;1172            WX_D_R.Q_D=257                             ;WMUX & D <- RBUS       Q <- D OLD                       D+R+CI.BCD
;1173            WX_D_R.Q_M=057                             ;WMUX & D <- RBUS       Q <- MBUS                        M+R+CI.BCD
;1174            WX_D_R.Q_XM=157                            ;WMUX & D <- RBUS       Q <- S/Z MBUS                   XM+R+CI.BCD
;1175            WX_D_S.Q_0=357                             ;WMUX & D <- SUP ROT    Q <- 0                           0+S+ 0.BCD
;1176            WX_D_S.Q_R=3D7                             ;WMUX & D <- SUP ROT    Q <- RBUS                        R+S+ 0.BCD
;1177            WX_D_S.Q_XM=1D7                            ;WMUX & D <- SUP ROT    Q <- S/Z MBUS                   XM+S+ 0.BCD
;1178            WX_Q.Q_D=2C7                               ;WMUX     <- Q OLD      Q <- D                           D-Q-CI.BCD
;1179            WX_Q.Q_M=0C7                               ;WMUX     <- Q OLD      Q <- MBUS                        M-Q-CI.BCD
;1180            WX_R.Q_D=247                               ;WMUX     <- RBUS       Q <- D                           D-R-CI.BCD
;1181            WX_R.Q_M=047                               ;WMUX     <- RBUS       Q <- MBUS                        M-R-CI.BCD
;1182            WX_R.Q_XM=147                              ;WMUX     <- RBUS       Q <- S/Z MBUS                   XM-R-CI.BCD
;1183            WX_S.Q_0=347                               ;WMUX     <- SUP ROT    Q <- 0                           0-S- 0.BCD
;1184            WX_S.Q_R=3C7                               ;WMUX     <- SUP ROT    Q <- RBUS                        R-S- 0.BCD
;1185            WX_S.Q_XM=1C7                              ;WMUX     <- SUP ROT    Q <- S/Z MBUS                   XM-S- 0.BCD
;1186
;1187            WX_D_Q_S=373                               ;WMUX & D & Q <- SUP ROT
;1188            WX_D_S=372                                 ;WMUX & D     <- SUP ROT
;1189            WX_Q_S=371                                 ;WMUX & Q     <- SUP ROT
;1190            WX_S=370                                   ;WMUX         <- SUP ROT
;1191            WX_D_Q_.NOT.S=363                          ;WMUX & D & Q <- .NOT.(SUP ROT)
;1192            WX_D_.NOT.S=362                            ;WMUX & D     <- .NOT.(SUP ROT)
;1193            WX_Q_.NOT.S=361                            ;WMUX & Q     <- .NOT.(SUP ROT)
;1194            WX_.NOT.S=360                              ;WMUX         <- .NOT.(SUP ROT)
;1195
;1196            WX_D_DSL.SQL=24E                           ;WMXU & D <- D SHF LEFT     Q <- SHF LEFT       (D-R-CI).SL
;1197            WX_D_DSL.SQR=24F                           ;WMXU & D <- D SHF LEFT     Q <- SHF RIGHT      (D-R-CI).SL
;1198            WX_D_DSR.SQL=24A                           ;WMXU & D <- D SHF RIGHT    Q <- SHF LEFT       (D-R-CI).SR
;1199            WX_D_DSR.SQR=24B                           ;WMXU & D <- D SHF RIGHT    Q <- SHF RIGHT      (D-R-CI).SR
;1200
;1201            WB_LOOPF=378                               ;WB<31:30> <- 0'LOOP FLAG
;1202            WB_LOOPF.Q_0=379                           ;WB<31:30> <- 0'LOOP FLAG   Q     <- 0
;1203            WB_LOOPF.D_0=37A                           ;WB<31:30> <- 0'LOOP FLAG   D     <- 0
;1204            WB_LOOPF.Q_D_0=37B                         ;WB<31:30> <- 0'LOOP FLAG   Q & D <- 0
;1205            WB_ALUF=37C                                ;WB<31:30> <- ALUSO'ALKC
;1206            WB_ALUF.Q_S=37D                            ;WB<31:30> <- ALUSO'ALKC    Q     <- S
;1207            WB_ALUF.D_S=37E                            ;WB<31:30> <- ALUSO'ALKC    D     <- S
;1208            WB_ALUF.Q_D_S=37F                          ;WB<31:30> <- ALUSO'ALKC    Q &  D <- S
;1209
;1210            MULFAST+=279,           .VALIDITY=<V50-54.21>   ;MULTIPLY +RBUS BY Q (2 ITERATIONS PER CYCLE)
;1211            MULSLOW+=27B,           .VALIDITY=<050.051>     ;MULTIPLY +RBUS BY Q (1 ITERATION  PER CYCLE)
;1212            MULFAST-=269,           .VALIDITY=<V50-54.21>   ;MULTIPLY -RBUS BY Q (2 ITERATIONS PER CYCLE)
;1213            MULSLOW-=26B,           .VALIDITY=<050.051>     ;MULTIPLY -RBUS BY Q (1 ITERATION  PER CYCLE)
;1214            DIVFAST+=26C,           .VALIDITY=<V50-54.21>   ;DIVIDE Q BY +RBUS (2 ITERATIONS PER CYCLE)
;1215            DIVSLOW+=26E,           .VALIDITY=<050.051>     ;DIVIDE Q BY +RBUS (1 ITERATION  PER CYCLE)
;1216            DIVFAST-=27C,           .VALIDITY=<V50-54.21>   ;DIVIDE Q BY -RBUS (2 ITERATIONS PER CYCLE)
;1217            DIVSLOW-=27E,           .VALIDITY=<050.051>     ;DIVIDE Q BY -RBUS (1 ITERATION  PER CYCLE)
;1218            REM=26A,                .VALIDITY=<V050>        ;UNSHIFT REMAINDER (RBUS MUST BE 0)
;1219            DIVDA=27F,              .VALIDITY=<V050>        ;DIVIDE DOUBLE ADD
;1220            DIVDS=26F,              .VALIDITY=<V050>        ;DIVIDE DOUBLE SUB
```

MACHINE FIELD DEFINITION

Figure 6-10

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                Page 65
; MACRO .MIC [130,2112]          Basic Macros

;2446   .TOC "  Basic Macros"
;2447
;2448   CCOP1                              "CC/CCOP1.CCBR_SIGND"
;2449   CCOP2                              "CC/CCOP2.CCBR_SIGND"
;2450   CLEAR ADD1(FLAG0)                  "MISC/CLR.FLAG0"
;2451   CLEAR ADD2(FLAG1)                  "MISC/CLR.FLAG1"
;2452   CLEAR BOOT(FLAG MMNOINT)           "MISC/CLR.MMNOINT"
;2453   CLEAR FLAG0                        "MISC/CLR.FLAG0"
;2454   CLEAR FLAG1                        "MISC/CLR.FLAG1"
;2455   CLEAR FLAG2                        "MISC/CLR.FLAG2"
;2456   CLEAR FLAG3                        "MISC/CLR.FLAG3"
;2457   CLEAR FPD                          "MISC/CLR.FPD"
;2458   CLEAR MOPZERO(FLAG1)               "MISC/CLR.FLAG1"
;2459   CLEAR MUL1(FLAG2)                  "MISC/CLR.FLAG2"
;2460   CLEAR MUL2(FLAG3)                  "MISC/CLR.FLAG3"
;2461   CLEAR OPZERO(FLAG3)                "MISC/CLR.FLAG3"
;2462   CLEAR OVER(FLAG2)                  "MISC/CLR.FLAG2"
;2463   CLEAR READ(FLAG1)                  "MISC/CLR.FLAG1"
;2464   CLEAR REGINT(FLAG1)                "MISC/CLR.FLAG1"
;2465   CLEAR SAMESIGN(FLAG4)              "MISC/CLR.MMNOINT"
;2466   CLEAR STACK FLAG                   "MISC/CLR.STACKFLG"
;2467   CLEAR SUB(FLAG1)                   "MISC/CLR.FLAG1"
;2468   CLEAR TP                           "MISC/CLR.TP"
;2469   CLEAR UNDER(FLAG3)                 "MISC/CLR.FLAG3"
;2470   CLEAR WRITE(FLAG1)                 "MISC/CLR.FLAG1"
;2471   CLOBBER MTEMP0                     "MSRC/TEMP0,SPW/MLONG"
;2472   CLOBBER MTEMP0 DEF                 "SPW/MLONG"
;2473
;2474   DEC STEPC                          "MISC/DEC.SC"
;2475   DISABLE INT                        "MISC/SET.MMNOINT"
;2476   DIVDA SOR IN R[]                   "ALPCTL/DIVDA,RSRC/@1,ROT/0"
;2477   DIVDS SOR IN R[]                   "ALPCTL/DIVDS,RSRC/@1,ROT/0"
;2478   DIVFAST+ SOR IN R[]                "ALPCTL/DIVFAST+,RSRC/@1,ROT/0"
;2479   DIVFAST- SOR IN R[]                "ALPCTL/DIVFAST-,RSRC/@1,ROT/0"
;2480
;2481   FLUSH XB                           "WCTRL/PC_WB,WB_M[PC]"
;2482
;2483   IO RESET                           "BUS/IOINIT"
;2484   IRD1                               "BUT/IRD1"
;2485   IRDX []                            "BUT/IRDX,NEXT/@1"
;2486   ISIZE[]                            "ISTRM/ISIZE_DSIZE,DTYPE/@1"
;2487
;2488   MULFAST+ CAND IN R[]               "ALPCTL/MULFAST+,RSRC/@1,ROT/0"
;2489   MULFAST- CAND IN R[]               "ALPCTL/MULFAST-,RSRC/@1,ROT/0"
;2490
;2491   NOP                                "ALPCTL/NOP"
;2492
;2493   PUSH                               "JSR/PUSH"
;2494   PUSH RBS+                          "MSRC/PSHADD"
;2495   PUSH RBS-                          "MSRC/PSHSUB"
;2496
;2497   PROCESS INIT                       "BUS/PRINIT"
;2498
;2499   RETURN []                          "SUT/RETURN,NEXT/@1"
;2500   RETURN AND INHIBIT DESTINATIONS    "SUT/RET.DINH"
```

BASIC MACROS

Figure 6-11

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                Page 67
; MACRO .MIC [130,2112]          Bus Function Macros

;2527   .TOC "  Bus Function Macros"
;2528
;2529   READ                                  "BUS/READ"
;2530   READ.LONG                             "BUS/READ.LNG"
;2531   READ.LONG.MOD                         "BUS/READ.LNG.MOD"
;2532   READ.MOD                              "BUS/READ.MOD"
;2533   READ.MOD.LOCK                         "BUS/READ.MOD.LCK"
;2534   READ.NOTRAP                           "BUS/READ.NT"
;2535   READ.PHY                              "BUS/READ.PHY"
;2536   READ.SECOND                           "BUS/READ.SEC"
;2537
;2538   WRITE                                 "BUS/WRITE,WCTRL/WDR_WB"
;2539   WRITE -M[]                            "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/ZERO,ALU/B-A-CI,ALUCI/ZERO,MUX/M.R1"
;2540   WRITE -Q                              "BUS/WRITE,WCTRL/WDR_WB,MUX/R.Q,RSRC/ZERO,ALU/A-B-CI,ALUCI/ZERO"
;2541   WRITE D+R[]+ALKC                      "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,MUX/D.R1,ALU/A+B+CI,ALUCI/ALKC"
;2542   WRITE M[]                             "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,ALU/OR,MUX/M.S,ROT/ZERO"
;2543   WRITE M[]+PSLC                        "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/ZERO,MUX/M.R1,ALU/A+B+CI,ALUCI/PSLC"
;2544   WRITE M[]+Q                           "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A+B+CI"
;2545   WRITE M[]+Q+PSLC                      "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A+B+CI,ALUCI/PSLC"
;2546   WRITE M[]-PSLC                        "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/ZERO,MUX/M.R1,ALU/A-B-CI,ALUCI/PSLC"
;2547   WRITE M[]-Q                           "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A-B-CI"
;2548   WRITE M[]-Q-PSLC                      "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/A-B-CI,ALUCI/PSLC"
;2549   WRITE M[].AND.ZLIT0[]                 "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/ZLIT0,MUX/M.S,ALU/AND"
;2550   WRITE M[].ANDNOT.Q                    "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/ANDNOT"
;2551   WRITE M[].ANDNOT.R[]                  "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/@2,ALU/ANDNOT,MUX/M.R1"
;2552   WRITE M[].ANDNOT.ZLIT8[]              "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/ZLIT8,MUX/M.S,ALU/ANDNOT"
;2553   WRITE M[].OR.Q                        "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/OR"
;2554   WRITE M[].OR.R[]                      "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,RSRC/@2,ALU/OR,MUX/M.R1"
;2555   WRITE M[].XOR.Q                       "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,MUX/M.Q1,ALU/XOR"
;2556   WRITE M[].XZ                          "BUS/WRITE,WCTRL/WDR_WB,MSRC/@1,ALPCTL/WX_S,ROT/XZ,MM"
;2557   WRITE NOTREG                          "BUS/WRITE,NOREG,WCTRL/WDR_WB"
;2558   WRITE Q                               "BUS/WRITE,WCTRL/WDR_WB,RSRC/ZERO,MUX/R.Q,ALU,OR"
;2559   WRITE Q.NOT                           "BUS/WRITE,WCTRL/WDR_WB,RSRC/ZERO,MUX/R.Q,ALU,A-B-CI,ALUCI/ONE"
;2560   WRITE R[]                             "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,ALU/OR,MUX/R.S,ROT/ZERO"
;2561   WRITE R[]+CONX(4)                     "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,ALU/A+B+CI,MUX/R.S,ROT/CONX.SIZ,DTYPE/LONG"
;2562   WRITE R[]-D-ALKC                      "BUS/WRITE,WCTRL/WDR_WB,RSRC/@1,MUX/D.R1,ALU/B-A-CI,ALUCI/ALKC"
;2563   WRITE R[]-M[]                         "BUS/WRITE,WCTRL/WDR_WB,MSRC/@2,RSRC/@1,ALU/B-A-CI,MUX/M.R1"
;2564   WRITE R[]-M[]-1                       "BUS/WRITE,WCTRL/WDR_WB,MSRC/@2,RSRC/@1,ALU/B-A-CI,MUX/M.R1,ALUCI/ONE"
;2565   WRITE XB PC_PC+1                      "BUS/WRITE,WCTRL/WDR_WB,MSRC/XB.PC_PC+1,ROT/ZERO,ALU/OR,MUX/M.S,
;2566                                                 ISTRM/ISIZE_DSIZE,DTYPE/BYTE"
;2567   WRITE XB PC_PC+4                      "BUS/WRITE,WCTRL/WDR_WB,MSRC/XB.PC_PC+1,ROT/ZERO,ALU/OR,MUX/M.S,
;2568                                                 ISTRM/ISIZE_DSIZE,DTYPE/LONG"
;2569   WRITE ZLIT0[]                         "BUS/WRITE,WCTRL/WDR_WB,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2570   WRITE.LONG                            "BUS/WRITE.LNG,WCTRL/WDR_WB.UR"
;2571   WRITE.LONG D                          "BUS/WRITE.LNG,WCTRL/WDR_WB.UR,RSRC/ZERO,MUX/D.R1,ALU/OR"
;2572   WRITE.LONG M[].ANDNOT.Q               "BUS/WRITE.LNG,WCTRL/WDR_WB.UR,MSRC/@1,MUX/M.Q1,ALU/ANDNOT"
;2573   WRITE.LONG M[].OR.Q                   "BUS/WRITE.LNG,WCTRL/WDR_WB.UR,MSRC/@1,MUX/M.Q1,ALU/OR"
;2574   WRITE.LONG.NOTRAP                     "BUS/WRITE.NT.LNG"
;2575   WRITE.NOTRAP                          "BUS/WRITE.NT"
;2576   WRITE.PHY                             "BUS/WRITE.PHY"
;2577   WRITE.PHY M[]                         "BUS/WRITE.PHY,WCTRL/WDR_WB,MSRC/@1,ALU/OR,MUX/M.S,ROT/ZERO"
;2578   WRITE.PHY R[]                         "BUS/WRITE.PHY,WCTRL/WDR_WB,RSRC/@1,ALU/OR,MUX/R.S,ROT/ZERO"
;2579   WRITE.SECOND                          "BUS/WRITE.SEC"
;2580   WRITE.SECOND.UL                       "BUS/WRITE.UL.SEC"
;2581   WRITE.UL M[]                          "BUS/WRITE.UL,WCTRL/WDR_WB,MSRC/@1,ALU/OR,MUX/M.S,ROT/ZERO"
```

BUS FUNCTION MACROS

Figure 6-12

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                    Page 69
; MACRO .MIC [130,2112]          Register Transfer Macros

;2586   .TOC "  Register Transfer Macros"
;2587
;2588   ALUS_BCD SIGN.ZERO(M[])          "CCMISC/ALUS_DSDZ.CCBR_ALUS,MSRC/@1,RSRC/ZERO,ALU/OR,MUX/M.R1"
;2589   ALUS_SIGND                      "CCMISC/ALUS_SIGND.CCBR_ALUS"
;2590   ALUS_UNSGN                      "CCMISC/ALUS_UNSGN.CCBR_ALUS"
;2591   ASTLVL_M[].RL.24                "WCTRL/ASTLVL_WB,ALPCTL/WX_S,MSRC/@1,ROT/RR.MM.SIZ,DTYPE/BYTE"
;2592   ASTLVL_R[]_M[]                  "WCTRL/ASTLVL_WB,SPW/MLONG,RSRC/@1,ALU/OR,MUX/M.S,ROT/ZERO,MSRC/@2"
;2593   ASTLVL_[]                       "WCTRL/ASTLVL_WB,LITRL/@1,LIT/LITRL,ROT/ZLIT24,ALPCTL/WX_S"
;2594
;2595   BUS GRANT M[]_IPL               "BUS/GRANT,WCTRL/GRANT,SPW/MLONG,MSRC/@1"
;2596
;2597   CC_M[]                          "CCPSL/CC_WB.CCBR_ALUS,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZERO"
;2598   CC_M[].NOTAND.R[]               "CCPSL/CC_WB.CCBR_ALUS,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/NOTAND"
;2599   CC_M[].OR.R[]                   "CCPSL/CC_WB.CCBR_ALUS,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/OR"
;2600   CC_M[]_MB.ANDNOT.CONX(1)        "CCPSL/CC_WB.CCBR_ALUS,ALU/ANDNOT,MUX/M.S,SPW/MLONG,MSRC/@1,ROT/CONX.SIZ,DTYPE/BYTE"
;2601   CC_M[]_MB.ANDNOT.CONX(4)        "CCPSL/CC_WB.CCBR_ALUS,ALU/ANDNOT,MUX/M.S,SPW/MLONG,MSRC/@1,ROT/CONX.SIZ,DTYPE/LONG"
;2602   CC_M[]_MB.OR.CONX(1)            "CCPSL/CC_WB.CCBR_ALUS,ALU/OR,MUX/M.S,SPW/MLONG,MSRC/@1,ROT/CONX.SIZ,DTYPE/BYTE"
;2603   CC_M[]_ZLIT0[]                  "CCPSL/CC_WB.CCBR_ALUS,SPW/MLONG,MSRC/@1,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@2"
;2604   CC_ZLIT0[]                      "CCPSL/CC_WB.CCBR_ALUS,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2605   CC_[]                           "CCPSL/CC_WB.CCBR_ALUS,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2606
;2607   CONREGS_D_M[]_R[]               "WCTRL/CONWRITE,MSRC/@1,SPW/MLONG,ALU/OR,MUX/R.S,ROT/ZERO,RSRC/@2,DQ1/D_WX"
;2608   CONREGS_M[]                     "WCTRL/CONWRITE,WB_M[@1]"
;2609   CONREGS_M[].OR.ZLIT16[]         "WCTRL/CONWRITE,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZLIT16,LIT/LITRL,LITRL/@2"
;2610   CONREGS_M[].RR.16               "WCTRL/CONWRITE,ROT/RR.MM.SIZ,DTYPE/WORD,MSRC/@1,ALPCTL/WX_S"
;2611   CONREGS_R[]                     "WCTRL/CONWRITE,ALU/OR,MUX/R.S,ROT/ZERO,RSRC/@1"
;2612   CONREGS_ZLIT16[]                "WCTRL/CONWRITE,ALPCTL/WX_S,ROT/ZLIT16,LIT/LITRL,LITRL/@1"
;2613
;2614   CRAR_ZLIT0[]                    "WCTRL/LOADCRAR,ALPCTL/WX_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2615   CRAR_ZLIT16[]                   "WCTRL/LOADCRAR,LITRL/@1,LIT/LITRL,ROT/ZLIT16,ALPCTL/WX_S"
;2616
;2617   D(OD)_ZLIT0[]                   "DQ1/D_WX,ROT/ZLIT0,LIT/LITRL,LITRL/@1,ALUOD/OR.OD,MUX/Z.S"
;2618   D_(M[].RR.P).AND.R[]            "DQ1/D_WX,MSRC/@1,RSRC/@2,ROT/RR.MM.P,ALU/AND,MUX/R.S"
;2619   D_(R[] M[]).RL.P                "ALPCTL/WX_D_S,MSRC/@2,RSRC/@1,ROT/RL.MM.P"
;2620   D_D+R[]                         "DQ1/D_WX,RSRC/@1,MUX/D.R1,ALU/A+B+CI"
;2621   D_D+ZLIT0[]                     "DQ1/D_WX,MUX/D.S,ALU/A+B+CI,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2622   D_D-R[]                         "DQ1/D_WX,RSRC/@1,MUX/D.R1,ALU/A-B-CI"
;2623   D_D-ZLIT0[]                     "DQ1/D_WX,LIT/LITRL,LITRL/@1,ROT/ZLIT0,MUX/D.S,ALU/A-B-CI"
;2624   D_D.AND.ZLIT0[]                 "DQ1/D_WX,ALU/AND,MUX/D.S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2625   D_D.AND.ZLIT28[]                "DQ1/D_WX,ALU/AND,MUX/D.S,ROT/ZLIT28,LIT/LITRL,LITRL/@1"
;2626   D_D.XOR.ZLIT12[]                "DQ1/D_WX,MUX/D.S,ALU,XOR,ROT/ZLIT12,LIT/LITRL,LITRL/@1"
;2627   D_M[]                           "DQ1/D_WX,MSRC/@1,RSRC/ZERO,ALU/OR,MUX/M.R1"
;2628   D_M[]+R[]                       "DQ1/D_WX,MSRC/@1,RSRC/@2,ALU/A+B+CI,MUX/M.R1"
;2629   D_M[]+ZLIT0[]                   "DQ1/D_WX,ALU/A+B+CI,MUX/M.S,MSRC/@1,ROT/ZLIT0,LIT/LITRL,LITRL/@2"
;2630   D_M[]-R[]                       "DQ1/D_WX,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/A-B-CI"
;2631   D_M[].AND.R[]                   "DQ1/D_WX,ALU/AND,MUX/M.R1,MSRC/@1,RSRC/@2"
;2632   D_M[].RR.16                     "ALPCTL/WX_D_S,MSRC/@1,ROT/RR.MM.SIZ,DTYPE/WORD"
;2633   D_M[].RR.16 Q_R[]               "ALPCTL/WX_D_S,Q_R,MSRC/@1,ROT/RR.MM.SIZ,DTYPE/WORD,RSRC/@2"
;2634   D_Q Q_D                         "ALPCTL/WX_D_Q,Q_D"
;2635   D_Q_M[]                         "DQ1/Q_D_WX,MSRC/@1,ROT/ZERO,MUX/M.S,ALU/OR"
;2636   D_R[]                           "DQ1/D_WX,ALU/OR,MUX/R.S,RSRC/@1,ROT/ZERO"
;2637   D_R[]-D-ALKC                    "DQ1/D_WX,RSRC/@1,MUX/D.R1,ALU/B-A-CI,ALUCI/ALKC"
;2638   D_R[]-M[]                       "DQ1/D_WX,RSRC/@1,MSRC/@2,MUX/M.R1,ALU/B-A-CI"
;2639   D_SEXT(M[])                     "DQ1/D_WX,MSRC/@1,RSRC/ZERO,MUX/XM.R,ALUXM/SIGN,ALU/OR"
;2640   D_SEXT(M[]) PL<4-0>_31 PL<5>_1  "DQ1/D_WX,MSRC/@1,ROT/DLIT0.PL_LIT,LIT/LITRL,LITRL/1FF,MUX/XM.S,ALU/AND"
```

REGISTER TRANSFER MACRO'S

Figure 6-13

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                Page 84
; MACRO .MIC [130,2112]          Branching Macros

;3367   .TOC "  Branching Macros"
;3368
;3369   (M[TEMP3]-SL)BYTE RANGE CHECK?  "BUT/SRKSTA,MSRC/@1,MUX/M.S,ALU/A-B-CI,ROT/SL"
;3370   (PL+SL).GT.32?                  "BUT/SRKSTA,ROTSRK/VIELD.000"
;3371
;3372   ABSVAL M[]<7-0>?                "BUT/SPKSTA,ROT/MINUS1,MSRC/@1,MUX/M.S,ALU/AND"
;3373   ADD1(FLAG0)?                    "BUT/FLAG0"
;3374   ADD2(FLAG1) ADD1(FLAG0)?        "BUT/FLAG1TOO"
;3375   ALLOW INT?                      "BUT/CCBR1.INT-TS"
;3376   ALUS?                           "BUT/CCBR,CC/NOP.CCBR_ALUS"
;3377   APT LOAD?                       "CC/NOP.CCBR_ALUS,BUT/CCBR"
;3378
;3379   BCD SIGN M[]?                   "BUT/SRKSTA,ROT/BCDSWP,MSRC/@1"
;3380   BCD SIGN.ZERO?                  "BUT/CCBR,CC/NOP.CCBR_ALUS"
;3381   BOOT(FLAG MMNOINT)?             "BUT/MM.NOINT"
;3382   BRA ON ADD?                     "BUT/BRA.ON.ADD"
;3383
;3384   CCOP1 SIGND?                    "BUT/CCBR,CC/CCOP1.CCBR_SIGND"
;3385   CCOP2 SIGND CMP .NOT.IR0?       "BUT/CCBR1.CCBR0.IR0,CC/CCOP2.CCBR_SIGND"
;3386   CHECK INTERRUPTS?               "BUT/CCBR1.INT-TS,DTYPE/LONG"
;3387   CMP SIGNS?                      "BUT/CCBR,CCMISC/NOP.CCBR_CSIGNS"
;3388   COUNT OR INT TIMER?             "BUT/CCBR1.INT-TS"
;3389
;3390   DBZ STEPC?                      "BUT/DBZ.SC"
;3391   DSIZE?                          "BUT/DSIZE"
;3392
;3393   EXPONENT RANGE?                 "BUT/SRKSTA"
;3394
;3395   FLAG0?                          "BUT/FLAG0"
;3396   FLAG1 FLAG2.XOR.FLAG3?          "BUT/F1.XOR23"
;3397   FLAG1?                          "BUT/FLAG1"
;3398   FLAG2?                          "BUT/FLAG2"
;3399   FLAG3?                          "BUT/FLAG3"
;3400   FLAG<1-0>?                      "BUT/FLAG1TOO"
;3401   FPD?                            "BUT/FPD"
;3402   FPS3?                           "BUT/FPS3"
;3403   FRO.FLTZ?                       "BUT/FRO.FLTZ"
;3404
;3405   HALT?                           "BUT/FPS1"
;3406
;3407   INTPEND OR TIMER?               "BUT/INT-TIMSERV"
;3408   IP.TS?                          "BUT/CCBR1.INT-TS,CCMISC/NOP.CCBR_BRATST"
;3409   IR<2>?                          "BUT/IR2"
;3410   IR<5>?                          "BUT/IR5"
;3411   IR<2-0>?                        "BUT/IR.2TOO"
;3412
;3413   LOD INC BRA?                    "BUT/LOD.INC.BRA"
;3414
;3415   MDR_GPR.R RNUM.EQ.7?            "BUT/SPASTA,WCTRL/MDR_WB,RSRC/GPR.R,ROT/ZERO,MUX/R.S,ALU/OR"
;3416   MDR_ZEXT(OSR) BRATST?           "BUT/CCBR,CCPSL/MDR_OSR.CCBR_BRATST"
;3417
;3418   MICRO VECTOR?                   "BUT/UVCTR,CLKX/XTND"
;3419
;3420   MM IPEND OR TIMER?              "BUT/MM.ALLOW.INT"
;3421   MM.ALLOW.INT?                   "BUT/MM.ALLOW.INT"
```

BRANCHIN G MACROS

Figure 6-14

```
; CMI018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                    Page 90
; INIT  .MIC [130,2112]          Initialize Microcode for the Console and Power up

                                  ;3595  .TOC "  Initialize Microcode for the Console and Power up"
ABSOLUTE ADDRESS                  ;3596  0:
                                  ;3597  IN.INIT:  <------ SYMBOLIC ADDRESS
                                  ;3598          ;-------------------------------;
                                  ;3599          LONLIT_[41F0000],  --> (1)      ;LONLIT GETS 41F0000
                                  ;3600          CLEAR FLAG2,                    ;PROCESS INIT CLEAR FLAG2
                                  ;3601          NEXT/IN.PSL.LONLIT   LOCATE IN  ;GOTO REG FLOW
                                  ;3602                               MACRO FILE
                                  ;3603  .REGION/<INIT.R1L>,<INIT.R1H>/<INIT.R2L>,<INIT.R2H>/<INIT.R3L>,<INIT.R3H>
                                  ;3604
                                  ;3605  IN.PC_0:
                                  ;3606          ;-------------------------------;
                                  ;3607          PC_R[ZERO],                     ;PC GETS 0
                                  ;3608          CLEAR FLAG1,                    ;FOR CHARLIE'S CLEAR TB SUBR
U 0818, 0080,5BE4,0BD8,4850,0001  ;3609          RETURN [1]                      ;RETURN+1
                                  ;3610
                                  ;3611  IN.VA_0:
                                  ;3612          ;-------------------------------;
                                  ;3613          VA_R[ZERO],                     ;VA GETS 0
U 0825, 4800,5BE4,0BD8,4A50,0001  ;3614          RETURN [1]                      ;RETURN+1
                                  ;3615
                                  ;3616  IN.CN.INIT:
                                  ;3617          ;-------------------------------;
                                  ;3618          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
U 0839, 3500,7DF0,7FFF,8450,083E  ;3619          SET FLAG2                       ;PROCESS INIT CLEAR FLAG2
                                  ;3620
                                  ;3621  IN.PSL.LONLIT:
                                  ;3622          ;-------------------------------;
U 083E, 0000,5BF4,03D4,0050,0820  ;3623          PSL_R[LONLIT],CLEAR FLAG0       ;PSL GETS LONLIT CLEAR FLAG0
                                  ;3624
                                  ;3625  *0
                                  ;3626          ;0------------------------------;
                                  ;3627          PUSH,                           ;JSR
                                  ;3628          STEPC_2,                        ;
                                  ;3629          CRAR [illegible]                ;CRAR GETS 2
U 0820, 5A00,C370,0340,2450,4818  ;3630          NEXT [illegible]                ;NOW IF WE CONWRITE
                                  ;3631                                          ;WE WILL WRITE TO RXCS
                                  ;3632
                                  ;3633          ;1------------------------------;
                                  ;3634          CONREGS_D_M[SISR]_R[ZERO],      ;RXCS GETS 0
U 0821, 89EF,5BE6,03D8,2C50,08A0  ;3635          DEC STEPC                       ;SISR GETS 0
                                  ;3636
                                  ;3637
```

LABELS AND MACRO EXPANSIONS

Figure 6-15

```
;2641   D_SEXT(XB) PC_PC+1          "DQ1/D_WX,MSRC/XB.PC_PC+1,RSRC/ZERO,MUX/XM.R,ALUXM/SIGN,ALU/OR,DTYPE/BYTE,ISTRM/ISIZE_DSIZE"
;2642   D_SEXT(XB) PC_PC+2          "DQ1/D_WX,MSRC/XB.PC_PC+1,RSRC/ZERO,MUX/XM.R,ALUXM/SIGN,ALU/OR,DTYPE/WORD,ISTRM/ISIZE_DSIZE"
;2643   D_ZEXT(M[])                "DQ1/D_WX,MSRC/@1,RSRC/ZERO,MUX/XM.R,ALUXM/ZERO,ALU/OR"
;2644   D_ZLIT0[]                  "ALPCTL/WX_D_S,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2645   D_ZLIT0[]-D                "DQ1/D_WX,MUX/D.S,ALU/B-A-CI,ROT/ZLIT0,LIT/LITRL,LITRL/@1"
;2646   D_ZLIT12[]                 "ALPCTL/WX_D_S,ROT/ZLIT12,LIT/LITRL,LITRL/@1"
;2647   D_ZLIT24[]                 "ALPCTL/WX_D_S,ROT/ZLIT24,LIT/LITRL,LITRL/@1"
;2648
;2649   FLAGS_D_R[]                "WCTRL/FLAGS_WB,RSRC/@1,ROT/ZERO,ALU/OR,MUX/R.S,DQ1/D_WX"
;2650   FLAGS_M[].AND.ZLIT0[]      "WCTRL/FLAGS_WB,MSRC/@1,ALU/AND,MUX/M.S,ROT/ZLIT0,LIT/LITRL,LITRL/@2"
;2651   FLAGS_R[]                  "WCTRL/FLAGS_WB,RSRC/@1,ROT/ZERO,ALU/OR,MUX/R.S"
;2652
;2653   INIR_M[]_Q                 "WCTRL/INIR_WB,MSRC/@1,SPW/MLONG,ALU/OR,MUX/R.Q,RSRC/ZERO"
;2654   IPL_M[].RL.16              "WCTRL/IPL_WB,ALPCTL/WX_S,MSRC/@1,ROT/RR.MM.SIZ,DTYPE/WORD"
;2655
;2656   LONLIT_[] →(1)             "LIT/LONLIT,LONLIT/<.NOT[<LONLIT/@1>]>" →(2) LOCATE IN DEFINE FILE
;2657
;2658   MDR_(M[] R[]).RR.P         "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ROT/RR.MR.P,ALPCTL/WX_S"
;2659   MDR_-1                     "WCTRL/MDR_WB,ROT/MINUS1,ALPCTL/WX_S"
;2660   MDR_-M[]                   "WCTRL/MDR_WB,MSRC/@1,ALU/B-A-CI,ALUCI/ZERO,RSRC/ZERO,MUX/M.R1"
;2661   MDR_0                      "WCTRL/MDR_0"
;2662   MDR_M[]                    "WCTRL/MDR_WB,MSRC/@1,RSRC/ZERO,MUX/M.R1,ALU/OR"
;2663   MDR_M[]+ALKC               "WCTRL/MDR_WB,MSRC/@1,ALU/A+B+CI,ALUCI/ALKC,RSRC/ZERO,MUX/M.R1"
;2664   MDR_M[]+R[]+ALKC           "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,MUX/M.R1,ALU/A+B+CI,ALUCI/ALKC"
;2665   MDR_M[]+ZLIT24[]           "WCTRL/MDR_WB,MSRC/@1,ALU/A+B+CI,MUX/M.S,ROT/ZLIT24,LIT/LITRL,LITRL/@2"
;2666   MDR_M[].AND.OLIT8[]        "WCTRL/MDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/OLIT8,MUX/M.S,ALU/AND"
;2667   MDR_M[].AND.ZLIT0[]        "WCTRL/MDR_WB,MSRC/@1,LIT/LITRL,LITRL/@2,ROT/ZLIT0,MUX/M.S,ALU/AND"
;2668   MDR_M[].ASR.P              "WCTRL/MDR_WB,MSRC/@1,ROT/ASR.M.P,ALPCTL/WX_S"
;2669   MDR_M[].FPLIT              "WCTRL/MDR_WB,MSRC/@1,ROT/FPLIT,ALPCTL/WX_S"
;2670   MDR_M[].OR.(R[].RR.24)     "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ROT/RR.RR.SIZ,DTYPE/LONG,MUX/M.S,ALU/OR"
;2671   MDR_M[].OR.ZLIT24[]        "WCTRL/MDR_WB,ALU/OR,MUX/M.S,MSRC/@1,ROT/ZLIT24,LIT/LITRL,LITRL/@2"
;2672   MDR_M[].RL.24              "WCTRL/MDR_WB,MSRC/@1,ROT/RR.MM.SIZ,DTYPE/BYTE,ALPCTL/WX_S"
;2673   MDR_M[].RL.8               "WCTRL/MDR_WB,MSRC/@1,DTYPE/LONG,ROT/RR.MM.SIZ,ALPCTL/WX_S"
;2674   MDR_M[].RL.9               "WCTRL/MDR_WB,ALPCTL/WX_S,ROT/RL.MM.PTE,MSRC/@1"
;2675   MDR_M[].RR.16              "WCTRL/MDR_WB,MSRC/@1,ROT/RR.MM.SIZ,DTYPE/WORD,ALPCTL/WX_S"
;2676   MDR_M[].XOR.R[]            "WCTRL/MDR_WB,MSRC/@1,RSRC/@2,ALU/XOR,MUX/M.R1"
;2677   MDR_M[].XOR.ZLIT12[]       "WCTRL/MDR_WB,MSRC/@1,ROT/ZLIT12,LIT/LITRL,LITRL/@2,MUX/M.S,ALU/XOR"
;2678   MDR_M[]_R[].RR.16          "WCTRL/MDR_WB,MSRC/@1,SPW/MLONG,RSRC/@2,ROT/RR.RR.SIZ,DTYPE/WORD,ALPCTL/WX_S"
;2679   MDR_M[]_ZLIT0[]            "WCTRL/MDR_WB,MSRC/@1,SPW/MLONG,LIT/LITRL,LITRL/@2,ROT/ZLIT0,ALPCTL/WX_S"
;2680   MDR_Q                      "WCTRL/MDR_WB,RSRC/ZERO,MUX/R.Q,ALU/OR"
;2681   MDR_Q_M[]                  "WCTRL/MDR_WB,DQ1/Q_WX,MSRC/@1,ROT/ZERO,MUX/M.S,ALU/OR"
;2682   MDR_R[]                    "WCTRL/MDR_WB,RSRC/@1,ROT/ZERO,MUX/R.S,ALU/OR"
;2683   MDR_R[]-M[]-ALKC           "WCTRL/MDR_WB,MSRC/@2,RSRC/@1,ALU/B-A-CI,ALUCI/ALKC,MUX/M.R1"
;2684   MDR_R[].RR.24              "WCTRL/MDR_WB,RSRC/@1,ROT/RR.RR.SIZ,DTYPE/LONG,ALPCTL/WX_S"
;2685   MDR_R[]_M[]                "WCTRL/MDR_WB,RSRC/@1,SPW/RLONG,MSRC/@2,ALU/OR,MUX/M.S,ROT/ZERO"
;2686   MDR_R[]_RB-CONX.SIZ        "WCTRL/MDR_WB,RSRC/@1,ROT/CONX.SIZ,MUX/R.S,ALU/A-B-CI,SPW/RLONG,DTYPE/IDEP"
;2687   MDR_SEXT(M[])              "WCTRL/MDR_WB,MSRC/@1,RSRC/ZERO,MUX/XM.R,ALUXM/SIGN,ALU/OR"
;2688   MDR_SEXT(XB)+R[] PC_PC+I   "WCTRL/MDR_WB,RSRC/@1,MSRC/XB.PC_PC+I,MUX/XM.R,ALUXM/SIGN,ALU/A+B+CI"
;2689   MDR_XB PC_PC+2             "WCTRL/MDR_WB,MSRC/XB.PC_PC+I,RSRC/ZERO,MUX/M.R1,ALU/OR,ISTRM/ISIZE_DSIZE,DTYPE/WORD"
;2690   MDR_XB PC_PC+4             "WCTRL/MDR_WB,MSRC/XB.PC_PC+I,RSRC/ZERO,MUX/M.R1,ALU/OR,ISTRM/ISIZE_DSIZE,DTYPE/LONG"
;2691   MDR_XB PC_PC+I             "WCTRL/MDR_WB,MSRC/XB.PC_PC+I,RSRC/ZERO,MUX/M.R1,ALU/OR,ISTRM/ISIZE_DSIZE,DTYPE/IDEP"
;2692   MDR_ZEXT(M[])              "WCTRL/MDR_WB,MSRC/@1,RSRC/ZERO,MUX/XM.R,ALUXM/ZERO,ALU/OR"
;2693   MDR_ZEXT(OSR)              "CCPSL/MDR_OSR.CCBR_BRATST"
;2694   MDR_ZLIT0[]                "WCTRL/MDR_WB,LIT/LITRL,LITRL/@1,ROT/ZLIT0,ALPCTL/WX_S"
;2695   MDR_ZLIT16[]               "WCTRL/MDR_WB,LIT/LITRL,LITRL/@1,ROT/ZLIT16,ALPCTL/WX_S"
```

MACRO EXPANSIONS 2

Figure 6-16

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                    Page 42
; DEFIN .MIC [130,2112]          Machine Definition                : ISTRM, JSR, LIT, LITRL, LONLIT, MISC

;1504    .TOC "  Machine Definition                : ISTRM, JSR, LIT, LITRL, LONLIT, MISC"
;1505
;1506    ISTRM/=<33:33>,.DEFAULT=0
;1507            NOP=0              ;ISIZE IS DETERMINED BY HARDWARE
;1508            ISIZE_DSIZE=1      ;ISIZE IS DETERMINED BY DSIZE
;1509
;1510    JSR/=<14:14>,.DEFAULT=0 ;SUBROUTINE CONTROL
;1511            NOP=0              ;NO OPERATION
;1512            PUSH=1             ;PUSH CURRENT ADDRESS ON MICRO STACK
;1513
;1514    LIT/=<77:76>,.DEFAULT=0 ;DEFINE UWORD FIELD INTERPRETATIONS
;1515            NORMAL=0           ;FIELDS ARE NORMAL
;1516            LITRL=1            ;SHORT LITERAL FIELD ENABLED
;1517            FPAWAIT=2          ;WAIT FOR FPA TO COMPLETE PROCESSING
;1518            LONLIT=3           ;LONG LITERAL FIELD ENABLED
;1519
;1520    LITRL/=<39:31>             ;SHORT LITERAL
;1521
;1522    LONLIT/=<62:31>            ;LONG LITERAL
;1523
;1524    MISC/=<75:71>,.DEFAULT=10        ;DEFINE MISC FUNCTIONS
;1525            NOP=10
;1526
;1527            CLR.FLAG0=0        ;CLEAR FLAG 0
;1528            CLR.FLAG1=1        ;CLEAR FLAG 1
;1529            CLR.FLAG2=2        ;CLEAR FLAG 2
;1530            CLR.FLAG3=3        ;CLEAR FLAG 3
;1531            CLR.MMNOINT=4      ;CLEAR FLAG 4
;1532            CLR.STACKFLG=5     ;CLEAR FLAG 5
;1533
;1534            SET.FLAG0=8        ;SET FLAG 0
;1535            SET.FLAG1=9        ;SET FLAG 1
;1536            SET.FLAG2=0A       ;SET FLAG 2
;1537            SET.FLAG3=0B       ;SET FLAG 3
;1538            SET.MMNOINT=0C     ;SET FLAG 4
;1539            SET.STACKFLG=0D    ;SET FLAG 5
;1540
;1541            RSBC=18            ;RETURN AND SUPPRESS BUS CYCLE
;1542            RNUM_2REG=11       ;RNUM <- COMP MODE SECOND REG
;1543            CLR.TP=12          ;PSL<TP> <- 0
;1544            CLR.FPD=1C         ;PSL<FPD> <- 0
;1545            SET.FPD=1D         ;PSL<FPD> <- 1
;1546            FORCE.TB=1E        ;FORCE TB PARITY ERROR
;1547            FORCE.CACHE=1F     ;FORCE CACHE PARITY ERROR
;1548
;1549            DEC.SC=13          ;STEP CNT <- STEP CNT - 1
;1550            SC_2=14            ;STEP CNT <- 2
;1551            SC_6=15            ;STEP CNT <- 6
;1552            SC_14=16           ;STEP CNT <- 14
;1553            SC_30=17           ;STEP CNT <- 30
```

MACRO EXPANSIONS 3

Figure 6-17

```
; CMI018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                    Page 90
; INIT  .MIC [130,2112]          Initialize Microcode for the Console and Power up

                                ;3595  .TOC "  Initialize Microcode for the Console and Power up"
                                ;3596  0:
                                ;3597  IN.INIT:
                                ;3598          ;--------------------------------;
                                ;3599          LONLIT_[41F0000],          LOCATE IN  ;LONLIT GETS 41F0000
                                ;3600          CLEAR FLAG2,          (3)  MACRO      ;PROCESS INIT CLEAR FLAG2
U 0000, 7100,7DF0,7FFF,8450,083E;3601          NEXT/IN.PSL.LONLIT         FILE       ;GOTO REG FLOW
                                ;3602
                                ;3603  .REGION/<INIT.R1L>,<INIT.R1H>/<INIT.R2L>,<INIT.R2H>/<INIT.R3L>,<INIT.R3H>
                                ;3604
                                ;3605  IN.PC_0:
                                ;3606          ;--------------------------------;
                                ;3607          PC_R[ZERO],                      ;PC GETS 0
                                ;3608          CLEAR FLAG1,                     ;FOR CHARLIE'S CLEAR TB SUBR
U 0818, 0080,5BE4,08D8,4850,0001;3609          RETURN [1]                       ;RETURN+1
                                ;3610
                                ;3611  IN.VA_0:
                                ;3612          ;--------------------------------;
                                ;3613          VA_R[ZERO],                      ;VA GETS 0
U 0825, 4800,5BE4,08D8,4A50,0001;3614          RETURN [1]                       ;RETURN+1
                                ;3615
                                ;3616  IN.CN.INIT:
                                ;3617          ;--------------------------------;
                                ;3618          LONLIT_[41F0000],                ;LONLIT GETS 41F0000
U 0839, 3500,7DF0,7FFF,8450,083E;3619          SET FLAG2                        ;PROCESS INIT CLEAR FLAG2
                                ;3620
                                ;3621  IN.PSL.LONLIT:
                                ;3622          ;--------------------------------;
U 083E, 0000,5BF4,03D4,0050,0820;3623          PSL_R[LONLIT].CLEAR FLAG0        ;PSL GETS LONLIT CLEAR FLAG0
                                ;3624
                                ;3625  =0
                                ;3626          ;0------ --------------------------;
                                ;3627          PUSH,                            ;JSR
                                ;3628          STEPC_2,                         ;
                                ;3629          CRAR_ZLIT0[00],                  ;CRAR GETS 2
U 0820, 5A00,C370,0340,2450,481B;3630          NEXT/N=.PC_0                     ;NOW IF WE CONWRITE
                                ;3631                                           ;WE WILL WRITE TO RXCS
                                ;3632
                                ;3633          ;1--------------------------------;
                                ;3634          CONREGS_D_M[SISR]_R[ZERO],       ;RXCS GETS 0
U 0821, 89EF,5BE6,03D8,2C50,08A0;3635          DEC STEPC                        ;SISR GETS 0
                                ;3636
                                ;3637
```

MACRO EXPANSIONS 4

Figure 6-18

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                Page 65
; MACRO .MIC [130,2112]          Basic Macros

;2446   .TOC "  Basic Macros"
;2447
;2448   CCOP1                                  "CC/CCOP1.CCBR_SIGND"
;2449   CCOP2                                  "CC/CCOP2.CCBR_SIGND"
;2450   CLEAR ADD1(FLAG0)                      "MISC/CLR.FLAG0"
;2451   CLEAR ADD2(FLAG1)                      "MISC/CLR.FLAG1"
;2452   CLEAR BOOT(FLAG MMNOINT)               "MISC/CLR.MMNOINT"
;2453   CLEAR FLAG0                            "MISC/CLR.FLAG0"
;2454   CLEAR FLAG1                            "MISC/CLR.FLAG1"
;2455   CLEAR FLAG2  <-(3)                     "MISC/CLR.FLAG2"  ->(4) LOCATE IN DEFINE FILE
;2456   CLEAR FLAG3                            "MISC/CLR.FLAG3"
;2457   CLEAR FPD                              "MISC/CLR.FPD"
;2458   CLEAR MOPZERO(FLAG1)                   "MISC/CLR.FLAG1"
;2459   CLEAR MUL1(FLAG2)                      "MISC/CLR.FLAG2"
;2460   CLEAR MUL2(FLAG3)                      "MISC/CLR.FLAG3"
;2461   CLEAR OPZERO(FLAG3)                    "MISC/CLR.FLAG3"
;2462   CLEAR OVER(FLAG2)                      "MISC/CLR.FLAG2"
;2463   CLEAR READ(FLAG1)                      "MISC/CLR.FLAG1"
;2464   CLEAR REGINT(FLAG1)                    "MISC/CLR.FLAG1"
;2465   CLEAR SAMESIGN(FLAG4)                  "MISC/CLR.MMNOINT"
;2466   CLEAR STACK FLAG                       "MISC/CLR.STACKFLG"
;2467   CLEAR SUB(FLAG1)                       "MISC/CLR.FLAG1"
;2468   CLEAR TP                               "MISC/CLR.TP"
;2469   CLEAR UNDER(FLAG3)                     "MISC/CLR.FLAG3"
;2470   CLEAR WRITE(FLAG1)                     "MISC/CLR.FLAG1"
;2471   CLOBBER MTEMP0                         "MSRC/TEMP0,SPW/MLONG"
;2472   CLOBBER MTEMP0 DEF                     "SPW/MLONG"
;2473
;2474   DEC STEPC                              "MISC/DEC.SC"
;2475   DISABLE INT                            "MISC/SET.MMNOINT"
;2476   DIVDA SOR IN R[]                       "ALPCTL/DIVDA,RSRC/@1,ROT/0"
;2477   DIVDS SOR IN R[]                       "ALPCTL/DIVDS,RSRC/@1,ROT/0"
;2478   DIVFAST+ SOR IN R[]                    "ALPCTL/DIVFAST+,RSRC/@1,ROT/0"
;2479   DIVFAST- SOR IN R[]                    "ALPCTL/DIVFAST-,RSRC/@1,ROT/0"
;2480
;2481   FLUSH X8                               "WCTRL/PC_WB,WB_M[PC]"
;2482
;2483   IO RESET                               "BUS/IOINIT"
;2484   IRD1                                   "BUT/IRD1"
;2485   IRDX []                                "BUT/IRDX,NEXT/@1"
;2486   ISIZE[]                                "ISTRM/ISIZE_DSIZE,DTYPE/@1"
;2487
;2488   MULFAST+ CAND IN R[]                   "ALPCTL/MULFAST+,RSRC/@1,ROT/0"
;2489   MULFAST- CAND IN R[]                   "ALPCTL/MULFAST-,RSRC/@1,ROT/0"
;2490
;2491   NOP                                    "ALPCTL/NOP"
;2492
;2493   PUSH                                   "JSR/PUSH"
;2494   PUSH R8S+                              "MSRC/PSHADD"
;2495   PUSH R8S-                              "MSRC/PSHSUB"
;2496
;2497   PROCESS INIT                           "BUS/PRINIT"
;2498
;2499   RETURN []                              "BUT/RETURN,NEXT/@1"
;2500   RETURN AND INHIBIT DESTINATIONS        "BUT/RET.DINH"
```

MACRO EXPANSIONS 5

Figure 6-19

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979
; DEFIN .MIC [130,2112]          Machine Definition              : ISTRM, JSR, LIT, LITRL, LONLIT, MISC

;1504    .TOC "  Machine Definition              : ISTRM, JSR, LIT, LITRL, LONLIT, MISC"
;1505
;1506    ISTRM/=<33:33>,,DEFAULT=0
;1507            NOP=0               ;ISIZE IS DETERMINED BY HARDWARE
;1508            ISIZE_DSIZE=1       ;ISIZE IS DETERMINED BY DSIZE
;1509
;1510    JSR/=<14:14>,,DEFAULT=0 ;SUBROUTINE CONTROL
;1511            NOP=0               ;NO OPERATION
;1512            PUSH=1              ;PUSH CURRENT ADDRESS ON MICRO STACK
;1513
;1514    LIT/=<77:76>,,DEFAULT=0 ;DEFINE UWORD FIELD INTERPRETATIONS
;1515            NORMAL=0            ;FIELDS ARE NORMAL
;1516            LITRL=1             ;SHORT LITERAL FIELD ENABLED
;1517            FPAWAIT=2           ;WAIT FOR FPA TO COMPLETE PROCESSING
;1518            LONLIT=3            ;LONG LITERAL FIELD ENABLED
;1519
;1520    LITRL/=<39:31>              ;SHORT LITERAL
;1521
;1522    LONLIT/=<62:31>             ;LONG LITERAL
;1523
;1524    MISC/=<75:71>,,DEFAULT=10       ;DEFINE MISC FUNCTIONS
;1525            NOP=10
;1526
;1527            CLR.FLAG0=0         ;CLEAR FLAG 0
;1528            CLR.FLAG1=1         ;CLEAR FLAG 1
;1529    (4) --> CLR.FLAG2=2         ;CLEAR FLAG 2
;1530            CLR.FLAG3=3         ;CLEAR FLAG 3
;1531            CLR.MMNOINT=4       ;CLEAR FLAG 4
;1532            CLR.STACKFLG=5      ;CLEAR FLAG 5
;1533
;1534            SET.FLAG0=8         ;SET FLAG 0
;1535            SET.FLAG1=9         ;SET FLAG 1
;1536            SET.FLAG2=0A        ;SET FLAG 2
;1537            SET.FLAG3=0B        ;SET FLAG 3
;1538            SET.MMNOINT=0C      ;SET FLAG 4
;1539            SET.STACKFLG=0D     ;SET FLAG 5
;1540
;1541            RSBC=1B             ;RETURN AND SUPPRESS BUS CYCLE
;1542            RNUM_2REG=11        ;RNUM <- COMP MODE SECOND REG
;1543            CLR.TP=12           ;PSL<TP> <- 0
;1544            CLR.FPD=1C          ;PSL<FPD> <- 0
;1545            SET.FPD=1D          ;PSL<FPD> <- 1
;1546            FORCE.TB=1E         ;FORCE TB PARITY ERROR
;1547            FORCE.CACHE=1F      ;FORCE CACHE PARITY ERROR
;1548
;1549            DEC.SC=13           ;STEP CNT <- STEP CNT - 1
;1550            SC_2=14             ;STEP CNT <- 2
;1551            SC_6=15             ;STEP CNT <- 6
;1552            SC_14=16            ;STEP CNT <- 14
;1553            SC_30=17            ;STEP CNT <- 30
```

MACRO EXPANSIONS 6

Figure 6-20

```
; CM1018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                  Page 90
; INIT  .MIC [130,2112]          Initialize Microcode for the Console and Power up

                                ;3595  .IOC "  Initialize Microcode for the Console and Power up"
                                ;3596  0:
                                ;3597  IN.INIT:
                                ;3598          ;-------------------------------;
                                ;3599          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
                                ;3600          CLEAR_FLAG2,                    ;PROCESS INIT CLEAR FLAG2
U 0000, 7100,7DF0,7FFF,8450,083E;3601          NEXT/IN.PSL.LONLIT              ;GOTO REG FLOW
                                ;3602                                          LOCATE IN FIELDNAME & DEFINE VALUES CREF
        LOCATE IN UPC CREF      ;3603  .REGION/<INIT.R1L>,<INIT.R1H>/<INIT.R2L>,<INIT.R2H>/<INIT.R3L>,<INIT.R3H>
                                ;3604
                                ;3605  IN.PC_0:
                                ;3606          ;-------------------------------;
                                ;3607          PC_R[ZERO],                     ;PC GETS 0
                                ;3608          CLEAR_FLAG1,                    ;FOR CHARLIE'S CLEAR TB SUBR
U 081B, 0080,5BE4,0BD8,4850,0001;3609          RETURN [1]                      ;RETURN+1
                                ;3610
                                ;3611  IN.VA_0:
                                ;3612          ;-------------------------------;
                                ;3613          VA_R[ZERO],                     ;VA GETS 0
U 0825, 4800,5BE4,0BD8,4A50,0001;3614          RETURN [1]                      ;RETURN+1
                                ;3615
                                ;3616  IN.CN.INIT:
                                ;3617          ;-------------------------------;
                                ;3618          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
U 0839, 3500,7DF0,7FFF,8450,083E;3619          SET FLAG2                       ;PROCESS INIT CLEAR FLAG2
                                ;3620
                                ;3621  IN.PSL.LONLIT:
                                ;3622          ;-------------------------------;
U 083E, 0000,5BF4,03D4,0050,0820;3623          PSL_R[LONLIT],CLEAR FLAG0       ;PSL GETS LONLIT CLEAR FLAG0
                                ;3624
                                ;3625  =0
                                ;3626          ;0------ ------------------------;
                                ;3627          PUSH,                           ;JSR
                                ;3628          STEPC_2,                        ;
                                ;3629          CRAR_ZLIT0[60],                 ;CRAR GETS 2
U 0820, 5A00,C370,0340,2450,481B;3630          NEXT/NO.PC_0                    ;NOW IF WE CONWRITE
                                ;3631                                          ;WE WILL WRITE TO RXCS
                                ;3632
                                ;3633          ;1------ ------------------------;
                                ;3634          CONREGS_D_M[SISR]_R[ZERO],      ;RXCS GETS 0
U 0821, 89EF,5BE6,03D8,2C50,08A0;3635          DEC STEPC                       ;SISR GETS 0
                                ;3636
                                ;3637
```

NEXT ADDRESS FIELD

Figure 6-21

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                    Page 875
;                          Cross Reference Listing -       Field Names and Defined Values

IL.ROTLMEM                6091 #  7818    7818
IL.ROTLREG                6104 #  7818    7818    7818    7818
IL.SBWCMEM                6320 #  7825    7825    7825    7825    7825    7825
IL.SBWCREG                6315 #  7824    7824    7824    7824
IL.SOR+.END+              6974 #  6983    6998
IL.SOR+.END-              6980 #
IL.SOR-.END+              6985 #
IL.SOR-.END-              6991 #
IL.SUB2.B.W.L.MEM         6273 #  7832    7832    7832    7832    7832    7832    7846    7846    7846    7846    7846
                          7846    7860    7860    7860    7860    7860    7860
IL.SUB2.B.W.L.REG         6268 #  7831    7831    7831    7831    7845    7845    7845    7845    7859    7859    7859
                          7859
IL.SUB3.B.W.L.MEM         6280 #  7839    7839    7853    7853    7866    7866
IL.SUB3.B.W.L.REG         6274 #  7839    7839    7839    7839    7853    7853    7853    7853    7866    7866    7866
                          7866
IL.TST.B.W.L              5877 #  7872    7872    7872    7872    7872    7872    7879    7879    7879    7879    7879
                          7879    7886    7886    7886    7886    7886    7886
IL.XOR2.B.W.L.MEM         6354 #  7894    7894    7894    7894    7894    7894    7908    7908    7908    7908    7908
                          7908    7921    7921    7921    7921    7921    7921
IL.XOR2.B.W.L.REG         6349 #  7893    7893    7893    7893    7907    7907    7907    7907    7920    7920    7920
                          7920
IL.XOR3.B.W.L.MEM         6361 #  7901    7901    7915    7915    7928    7928
IL.XOR3.B.W.L.REG         6355 #  7901    7901    7901    7901    7915    7915    7915    7915    7928    7928    7928
                          7928
IN.CLR.CACHE              3664 #  3698
IN.CN.INIT                3616 #  4834    4892    5035
IN.DEC.D                  3667    3695 #
IN.INIT                   3597 #
IN.PC_0                   3605 #  3630    3672
IN.PSL.LONLIT             3601    3621 #  <------------------------ INDICATES LOCATION OF LABEL
IN.VA_0                   3611 #  3661
LS.LDPCTX                 25797 # 26103   26103   26103   26103
LS.LDPCTX.GPRS.0          25821 # 25833
LS.LDPCTX.GPRS.1          25816   25824   25829 #
LS.LDPCTX.POBR            25828   25835 #
LS.LDPCTX.PSL             25897   25904 # 25928
LS.LDPCTX.PUSH            25901   25931 #
LS.LDPCTX.SUB1            25907   25934   25940 #
LS.MODE.CHECK             25801   25968   26082 #
LS.SVPCTX                 25964 # 26133   26133   26133   26133
LS.SVPCTX.GPRS.2          25976 # 25988
LS.SVPCTX.GPRS.4          25979   25984 #
LS.SVPCTX.IPL             26019   26025 #
LS.SVPCTX.PC              25982   25989 #
LS.SVPCTX.SPS             26023   26044 #
MM.BUT.XB.TBMISS          28659 #
MM.DEC.VA                 28373   28380 # 28401   28411
MM.FLUSH.XB               28611   28642   28649 #
MM.GET.ACV                28725 # 28748   28807   28825   28846
MM.GET.BUT.FLUSH          29006 # 29063
MM.GET.BUT.PTE            28662   28982 #
MM.GET.BUT.PTE05          28989 # 29028
MM.GET.BUT.PTE10          28993   29004   29014 # 29021
MM.GET.BUT.PTE20          29039   29044   29046 #
```

MICROINSTRUCTION CROSS REFERENCE

Figure 6-22

```
; CMT018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979
;                       Location / Line Number Index

U 0708   20852#  20858#  20952#  20958#  21079#  21084#  21095#  21099#
U 0710   21156#  21161#  21167#  21171#  21290#  21294#  21313#  21316#
U 0718   21347#  21350#  21411#  21416#  21447#  21450#  21456#  21461#
U 0720   21570#  21574#  21616#  21620#  21684#  21688#  21730#  21733#
U 0728   21797#  21800#  21888#  21892#  22221#  22225#  22244#  22247#
U 0730   22262#  22265#  22269#  22272#  22281#  22286#  22296#  22299#
U 0738   22337#  22340#  22344#  22348#  22352#  22355#  22382#  22385#
U 0740   22390#  22394#  22422#  22427#  22433#  22437#  22530#  22533#
U 0748   22548#  22552#  22559#  22563#  22607#  22613#  22620#  22624#
U 0750   22915#  22919#  23178#  23181#  23259#  23262#  23267#  23270#
U 0758   23298#  23302#  23566#  23570#  23605#  23609#  23615#  23618#
U 0760   23630#  23635#  23674#  23678#  23746#  23750#  23968#  23971#
U 0768   24312#  24315#  24320#  24323#  24352#  24356#  24614#  24618#
U 0770   24630#  24633#  24662#  24666#  24675#  24679#  24687#  24691#
U 0778   24700#  24704#  24781#  24787#  24883#  24887#  24892#  24896#
U 0780   25237#  25243#  25462#  25465#  25482#  25485#  25602#  25606#
U 0788   25645#  25649#  25682#  25686#  25759#  25763#  25816#  25819#
U 0790   25824#  25828#  25852#  25855#  25878#  25883#  25897#  25901#
U 0798   25934#  25938#  25979#  25982#  26031#  26034#  26349#  26355#
U 07A0   26362#  26366#  26390#  26394#  26543#  26547#  26552#  26556#
U 07A8   26704#  26708#  26757#  26761#  26915#  26919#  26986#  26990#
U 07B0   27008#  27011#  27034#  27038#  27105#  27109#  27167#  27170#
U 07B8   27180#  27183#  27436#  27440#  27515#  27518#  27603#  27607#
U 07C0   27717#  27722#  27833#  27836#  27842#  27845#  28080#  28084#
U 07C8   28157#  28160#  28193#  28197#  28449#  28454#  28462#  28466#
U 07D0   29248#  29252#  29312#  29316#  29368#  29372#  29912#  29916#
U 07D8   29920#  29923#  30005#  30009#  30203#  30206#  30225#  30229#
U 07E0   30264#  30268#  30274#  30277#  30623#  30627#  30844#  30849#
U 07E8   30892#  30895#  30961#  30964#  30990#  30994#  31104#  31108#
U 07F0   31216#  31219#  31225#  31229#  31237#  31240#  31350#  31353#
U 07F8   31360#  31363#  31371#  31374#  31483#  31486#  31493#  31496#
U 0800   16967#  16972#  16977#  16981#  16986#  16990#  16995#  16999#
U 0808   17003#  17007#  17012#  17016#  17021#  17025#  17030#  17034#
U 0810   17038#  17042#  17047#  17051#  17056#  17060#  17065#  17069#
U 0818   17073#  17077#  17082#  3609    17868#  17872#  17877#  17881#
U 0820   3630#   3635#   17087#  12632#  17092#  3614    17097#  12636#
U 0828   17102#  3687#   17107#  24343#  17112#  3632#   17116#  24347#
U 0830   17121#  6528#   17126#  6532#   17131#  15106#  17136#  15109#
U 0838   17141#  3619    17146#  23056#  4016#   4019#   3623    23060#  <-- INDICATES LOCATION OF MICROINSTRUCTION
U 0840   3683    3661#   3667#   3672#   3677#   3698    7093#   7098#       AND THAT THE LOCATION IS NOT CONSTRAINED
U 0848   28111#  28115#  28119#  28123#  4070#   4075#   26605#  23108#
U 0850   4161#   4166#   4171#   28212#  4177#   4131#   4185#   28216#
U 0858   3981    28127#  5317#   5322#   4141#   4145#   26609#  23112#
U 0860   4230#   4236#   9439#   9444#   3985    4241#   15814#  15818#
U 0868   26459#  26463#  26467#  26471#  4152#   4156#   3988    23116#
U 0870   3994    4267#   10045#  10051#  4275#   4279#   15905#  15909#
U 0878   23222#  23227#  23232#  23238#  23241#  23245#  23249#  23253#
U 0880   5537#   5096#   5542#   5102#   5108#   5113#   16857#  16863#
U 0888   5119#   6568#   4191    6571#   17974#  17977#  17981#  17986#
U 0890   5663#   5668#   5673#   5677#   5681#   5685#   5691#   5696#
U 0898   5700#   6894#   4321    6898#   17992#  17997#  18002#  18007#
U 08A0   3643#   3650#   3653#   4341    4218#   4223#   9361#   9365#
U 08A8   9373#   9381#   12313#  12317#  24294#  24298#  24302#  24306#
U 08B0   7014#   7018#   5334#   23381#  7022#   4377    5339#   23384#
```

MICROINSTRUCTION CROSS REFERENCE 2

Figure 6-23

```
; CMI018.MCR [130,2112] Micro-2.1 1A(33)       14:40:3 9-Mar-1979                                    Page 90
; INIT  .MIC [130,2112]        Initialize Microcode for the Console and Power up

                              ;3595  .TOC "  Initialize Microcode for the Console and Power up"
                              ;3596  0:
                              ;3597  IN.INIT:
                              ;3598          ;-------------------------------;
                              ;3599          LONLIT_[41F0000].               ;LONLIT GETS 41F0000.
                              ;3600          CLEAR FLAG2,                    ;PROCESS INIT CLEAR FLAG2
U 0000, 7100,7DF0,7FFF,8450,083E ;3601       NEXT/IN.PSL.LONLIT              ;GOTO REG FLOW
                              ;3602
                              ;3603  .REGION/<INIT.R1L>,<INIT.R1H>/<INIT.R2L>,<INIT.R2H>/<INIT.R3L>,<INIT.R3H>
                              ;3604
                              ;3605  IN.PC_0:
                              ;3606          ;-------------------------------;
                              ;3607          PC_R[ZERO],                     ;PC GETS 0
                              ;3608          CLEAR FLAG1,                    ;FOR CHARLIE'S CLEAR TB SUBR
U 0818, 0080,58E4,0808,4850,0001 ;3609       RETURN [1]                      ;RETURN+1
                              ;3610
                              ;3611  IN.VA_0:
                              ;3612          ;-------------------------------;
                              ;3613          VA_R[ZERO],                     ;VA GETS 0
U 0825, 4800,58E4,0808,4A50,0001 ;3614       RETURN [1]                      ;RETURN+1
                              ;3615
                              ;3616  IN.CN.INIT:
                              ;3617          ;-------------------------------;
                              ;3618          LONLIT_[41F0000],               ;LONLIT GETS 41F0000
U 0839, 3500,7DF0,7FFF,8450,083E ;3619       SET FLAG2                       ;PROCESS INIT CLEAR FLAG2
                              ;3620
          FROM FIELD NAME CREF ---> ;3621    IN.PSL.LONLIT:
                              ;3622          ;-------------------------------;
U 083E, 0000,58F4,0304,0050,0820 ;3623       PSL_R[LONLIT].CLEAR FLAG0       ;PSL GETS LONLIT CLEAR FLAG0
                              ;3624
                FROM UPC CREF ---> ;3625     =0
                              ;3626          ;0------------------------------;
                              ;3627          PUSH,                           ;JSR
                              ;3628          STEPC_2,                        ;
                              ;3629          CRAR_ZLIT[00],                  ;CRAR GETS 2
U 0820, 5A00,C370,0340,2450,4818 ;3630       NEXT/NW.PC_0                    ;NOW IF WE CONWRITE
                              ;3631                                          ;WE WILL WRITE TO RXCS
                              ;3632
                              ;3633          ;1------------------------------;
                              ;3634          CONREGS_0_M[SISR]_R[ZERO],      ;RXCS GETS 0
U 0821, 89EF,5BE6,0308,2C50,08A0 ;3635       DEC STEPC                       ;SISR GETS 0
                              ;3636
                              ;3637
```

MICROINSTRUCTION CROSS REFERENCE 3

Figure 6-24

```
; CMI018.MCR [130,2112] Micro-2.1 1A(33)        14:40:3 9-Mar-1979                                    Page 90
; INIT  .MIC [130,2112]          Initialize Microcode for the Console and Power up

                                ;3595  .TOC "  Initialize Microcode for the Console and Power up"
                                ;3596  0:
                                ;3597  IN.INIT:
                                ;3598          ;--------------------------------;
                                ;3599          LONLIT_[41F0000],                ;LONLIT GETS 41F0000
                                ;3600          CLEAR FLAG2,                     ;PROCESS INIT CLEAR FLAG2
U 0000, 7100,7DF0,7FFF,8450,083E;3601          NEXT/IN.PSL.LONLIT               ;GOTO REG FLOW
                                ;3602
                                ;3603  .REGION/<INIT.R1L>,<INIT.R1H>/<INIT.R2L>,<INIT.R2H>/<INIT.R3L>,<INIT.R3H>
                                ;3604
                                ;3605  IN.PC_0:
                                ;3606          ;--------------------------------;
                                ;3607          PC_R[ZERO],                      ;PC GETS 0
                                ;3608          CLEAR FLAG1,                     ;FOR CHARLIE'S CLEAR TB SUBR
U 0818, 0080,58E4,0BD8,4850,0001;3609          RETURN [1]                       ;RETURN+1
                                ;3610
                                ;3611  IN.VA_0:
                                ;3612          ;--------------------------------;
                                ;3613          VA_R[ZERO],                      ;VA GETS 0
U 0825, 4800,58E4,0BD8,4A50,0001;3614          RETURN [1]                       ;RETURN+1
                                ;3615
                                ;3616  IN.CN.INIT:
                                ;3617          ;--------------------------------;
                                ;3618          LONLIT_[41F0000],                ;LONLIT GETS 41F0000
U 0839, 3500,7DF0,7FFF,8450,083E;3619          SET FLAG2                        ;PROCESS INIT CLEAR FLAG2
                                ;3620
                                ;3621  IN.PSL.LONLIT:
                                ;3622          ;--------------------------------;
U 083E, 0000,58F4,03D4,0050,0820;3623          PSL_R[LONLIT],CLEAR FLAG0        ;PSL GETS LONLIT CLEAR FLAG0
                                ;3624
                                ;3625  =0
                                ;3626          ;0-------------------------------;
                                ;3627          PUSH,                            ;JSR
                                ;3628          STEPC_2,                         ;
                                ;3629          CRAR_[00],                       ;CRAR GETS 2
U 0820, 5A00,C370,0340,2450,481B;3630          NEXT/N_PC_0                      ;NOW IF WE CONWRITE
                                ;3631                                           ;WE WILL WRITE TO RXCS
                                ;3632
                                ;3633          ;1-------------------------------;
                                ;3634          CONREGS_D_M[SISR]_R[ZERO],       ;RXCS GETS 0
U 0821, 89EF,58E6,03D8,2C50,08A0;3635          DEC STEPC                        ;SISR GETS 0
                                ;3636
                                ;3637
```

MICROINSTRUCTION CROSS REFERENCE 4

Figure 6-25

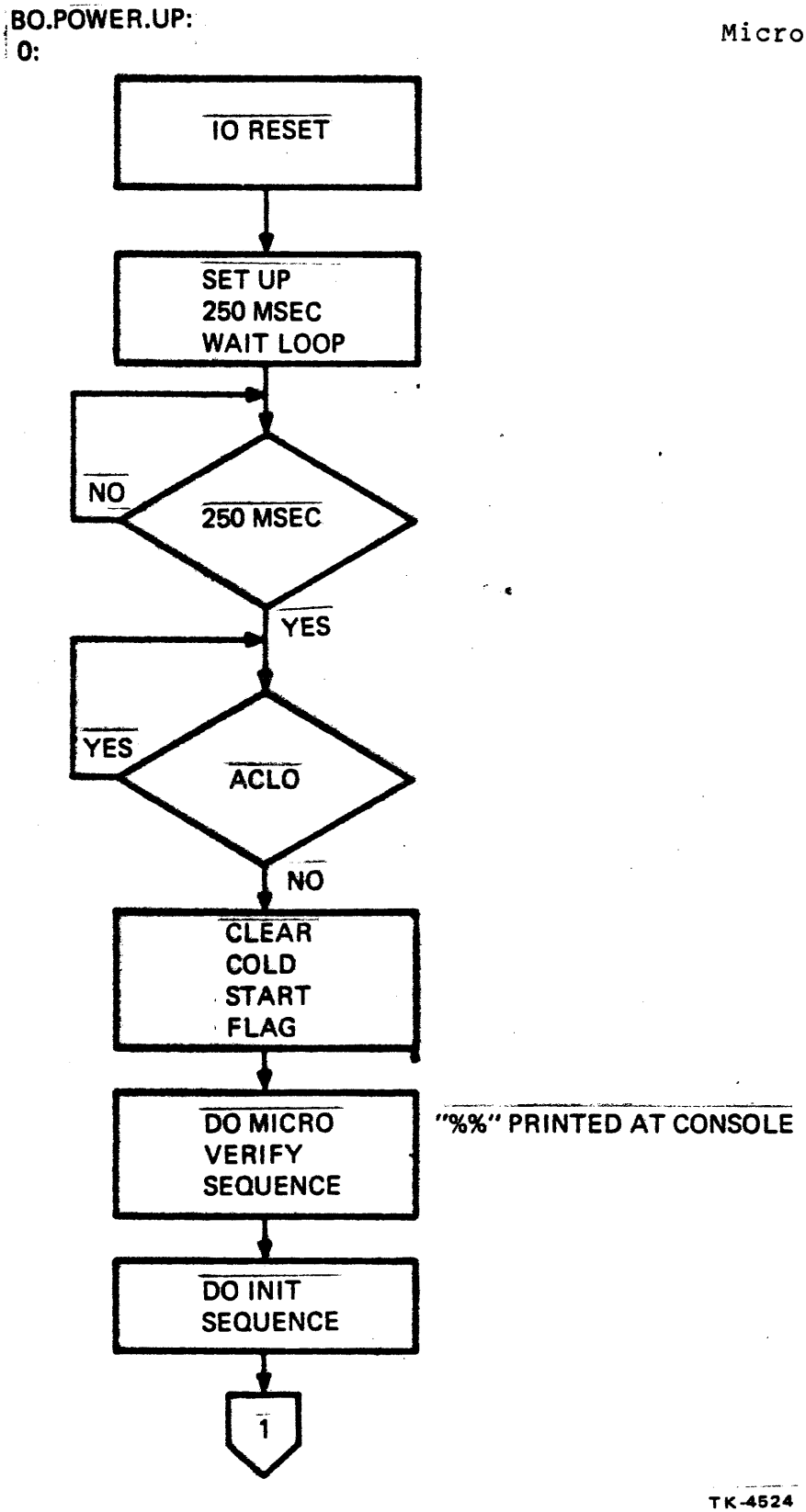

BO.POWER.UP:
0:
IO RESET
SET UP
250 MSEC
WAIT LOOP
NO
250 MSEC
YES
YES
ACLO
NO
CLEAR
COLD
START
FLAG
DO MICRO
VERIFY
SEQUENCE
"%%" PRINTED AT CONSOLE
DO INIT
SEQUENCE
1
TK-4524

Figure 6-26

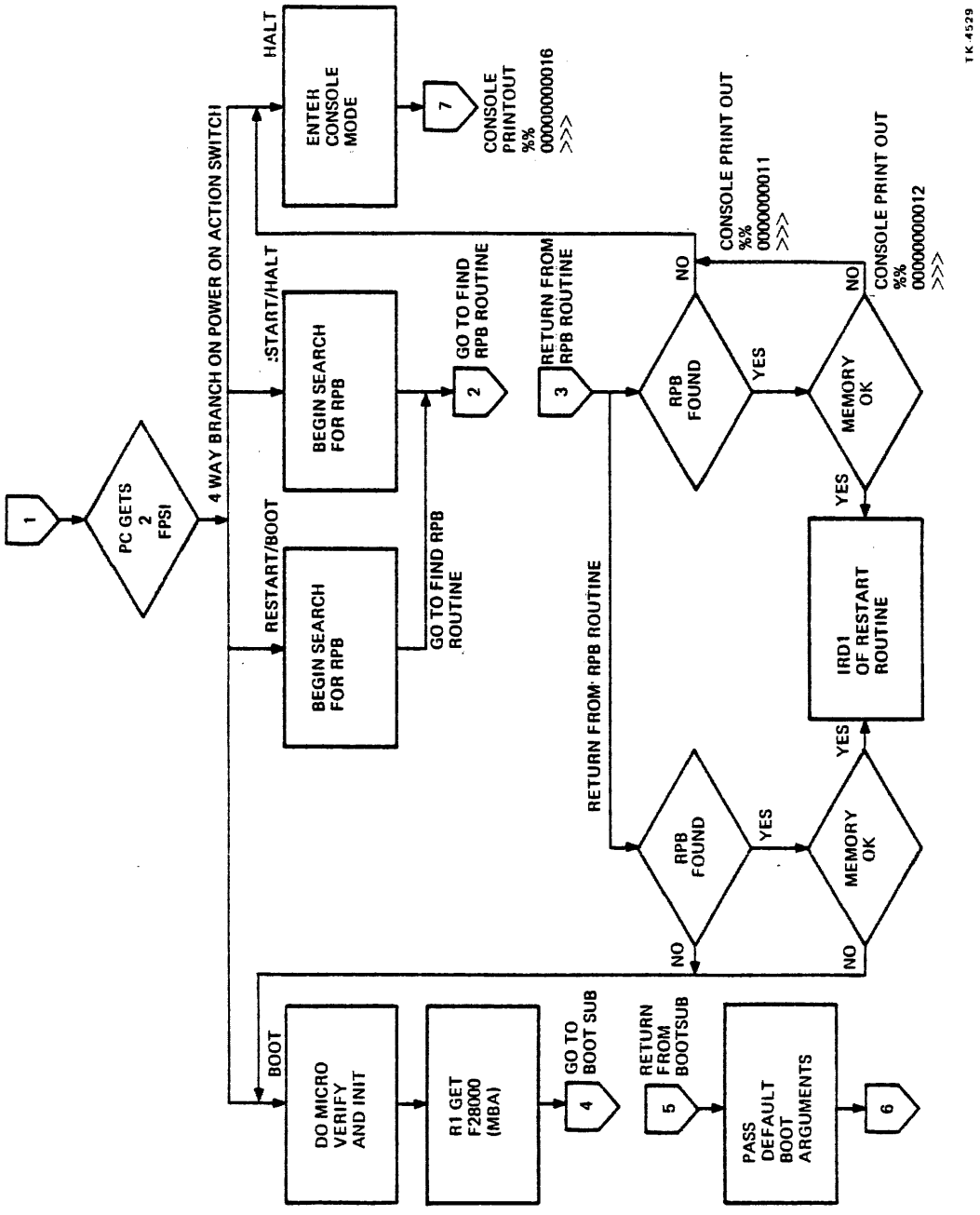

1
PC GETS 2 FPSI
4 WAY BRANCH ON POWER ON ACTION SWITCH
HALT
ENTER CONSOLE MODE
7
CONSOLE PRINTOUT %% 0000000016 >>>
:START/HALT
BEGIN SEARCH FOR RPB
GO TO FIND RPB ROUTINE
2
RESTART/BOOT
BEGIN SEARCH FOR RPB
GO TO FIND RPB ROUTINE
3
RETURN FROM RPB ROUTINE
RPB FOUND
NO
YES
CONSOLE PRINT OUT %% 0000000011 >>>
MEMORY OK
NO
YES
CONSOLE PRINT OUT %% 0000000012 >>>
IRD1 OF RESTART ROUTINE
RETURN FROM RPB ROUTINE
RPB FOUND
NO
YES
MEMORY OK
NO
YES
BOOT
DO MICRO VERIFY AND INIT
R1 GET F28000 (MBA)
4
GO TO BOOT SUB
5
RETURN FROM BOOTSUB
PASS DEFAULT BOOT ARGUMENTS
6
TK 4529

Figure 6-27

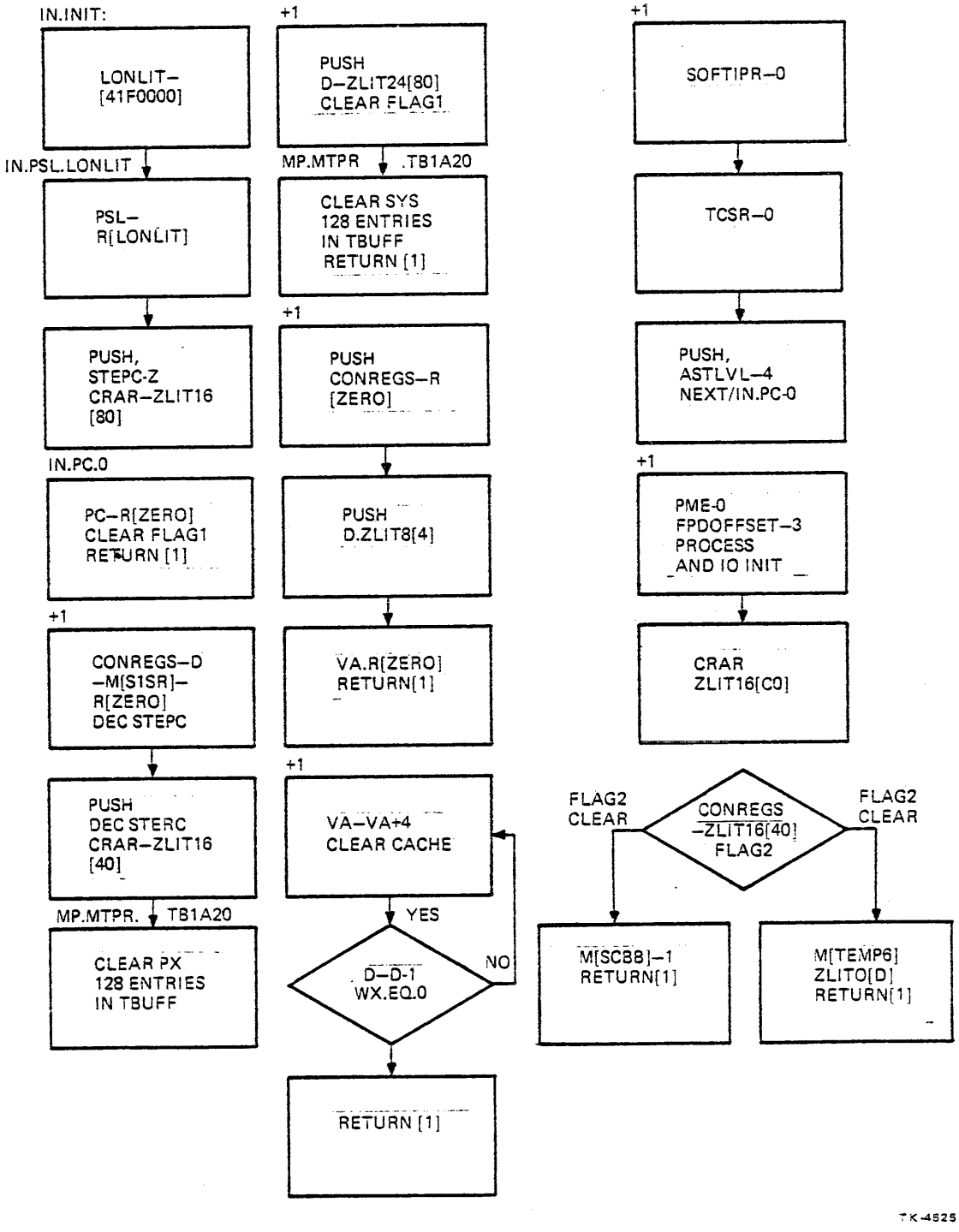

IN.INIT:
LONLIT–
[41F0000]
IN.PSL.LONLIT
PSL–
R[LONLIT]
PUSH,
STEPC-Z
CRAR–ZLIT16
[80]
IN.PC.0
PC–R[ZERO]
CLEAR FLAG1
RETURN [1]
+1
CONREGS–D
–M[S1SR]–
R[ZERO]
DEC STEPC
PUSH
DEC STERC
CRAR–ZLIT16
[40]
MP.MTPR. TB1A20
CLEAR PX
128 ENTRIES
IN TBUFF
+1
PUSH
D–ZLIT24[80]
CLEAR FLAG1
MP.MTPR .TB1A20
CLEAR SYS
128 ENTRIES
IN TBUFF
RETURN [1]
+1
PUSH
CONREGS–R
[ZERO]
PUSH
D.ZLIT8[4]
VA.R[ZERO]
RETURN[1]
+1
VA–VA+4
CLEAR CACHE
YES
D–D-1
WX.EQ.0
NO
RETURN [1]
+1
SOFTIPR–0
TCSR–0
PUSH,
ASTLVL–4
NEXT/IN.PC-0
+1
PME-0
FPDOFFSET–3
PROCESS
AND IO INIT
CRAR
ZLIT16[C0]
FLAG2
CLEAR
CONREGS
–ZLIT16[40]
FLAG2
FLAG2
CLEAR
M[SCBB]–1
RETURN[1]
M[TEMP6]
ZLITO[D]
RETURN[1]
TK-4525

Figure 6-28

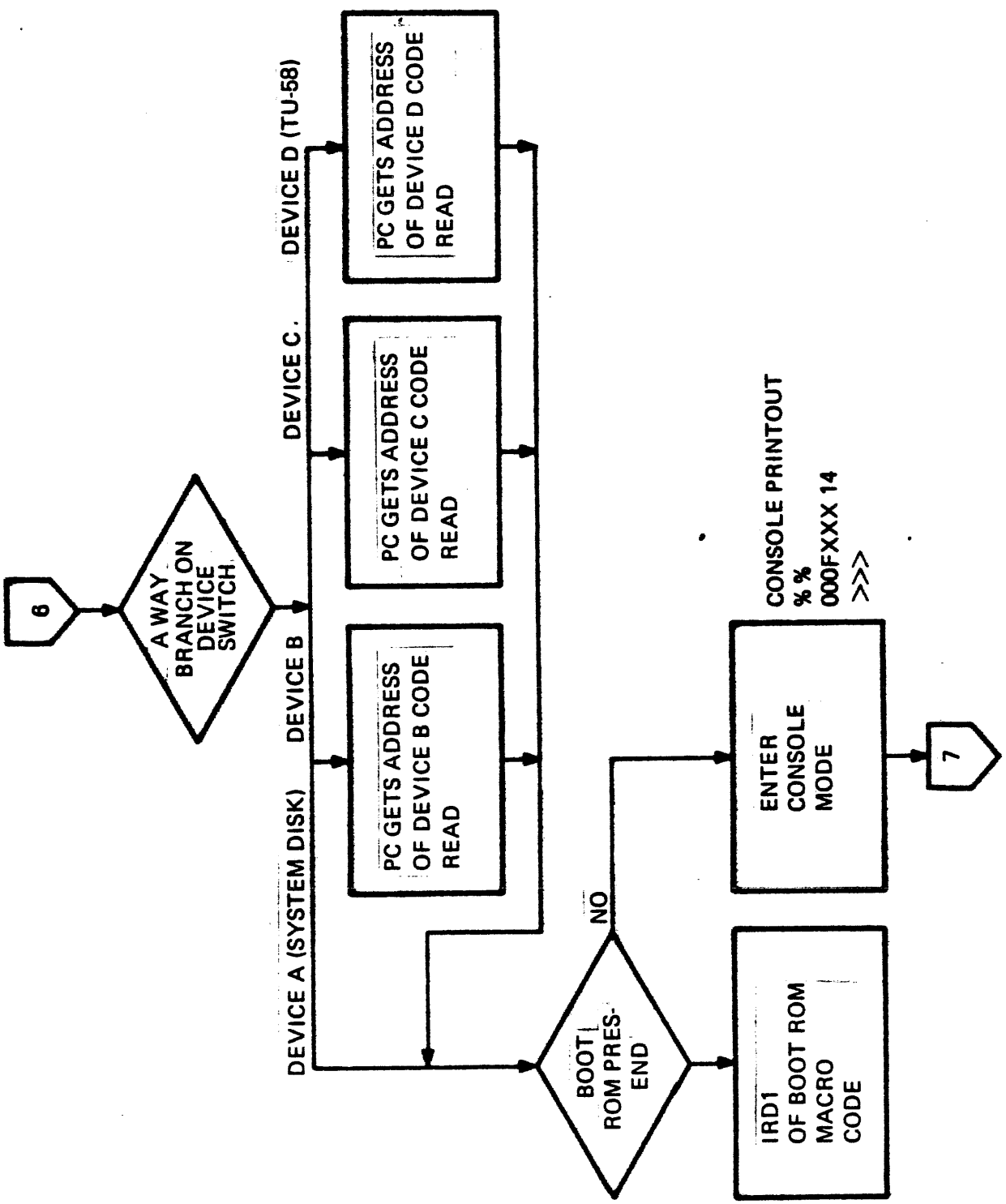

6
A WAY BRANCH ON DEVICE SWITCH
DEVICE A (SYSTEM DISK)
DEVICE B
DEVICE C.
DEVICE D (TU-58)
PC GETS ADDRESS OF DEVICE B CODE READ
PC GETS ADDRESS OF DEVICE C CODE READ
PC GETS ADDRESS OF DEVICE D CODE READ
BOOT ROM PRES-END
NO
IRD1 OF BOOT ROM MACRO CODE
ENTER CONSOLE MODE
CONSOLE PRINTOUT
% %
000FXXX 14
>>
7
TK-4527

Figure 6-29

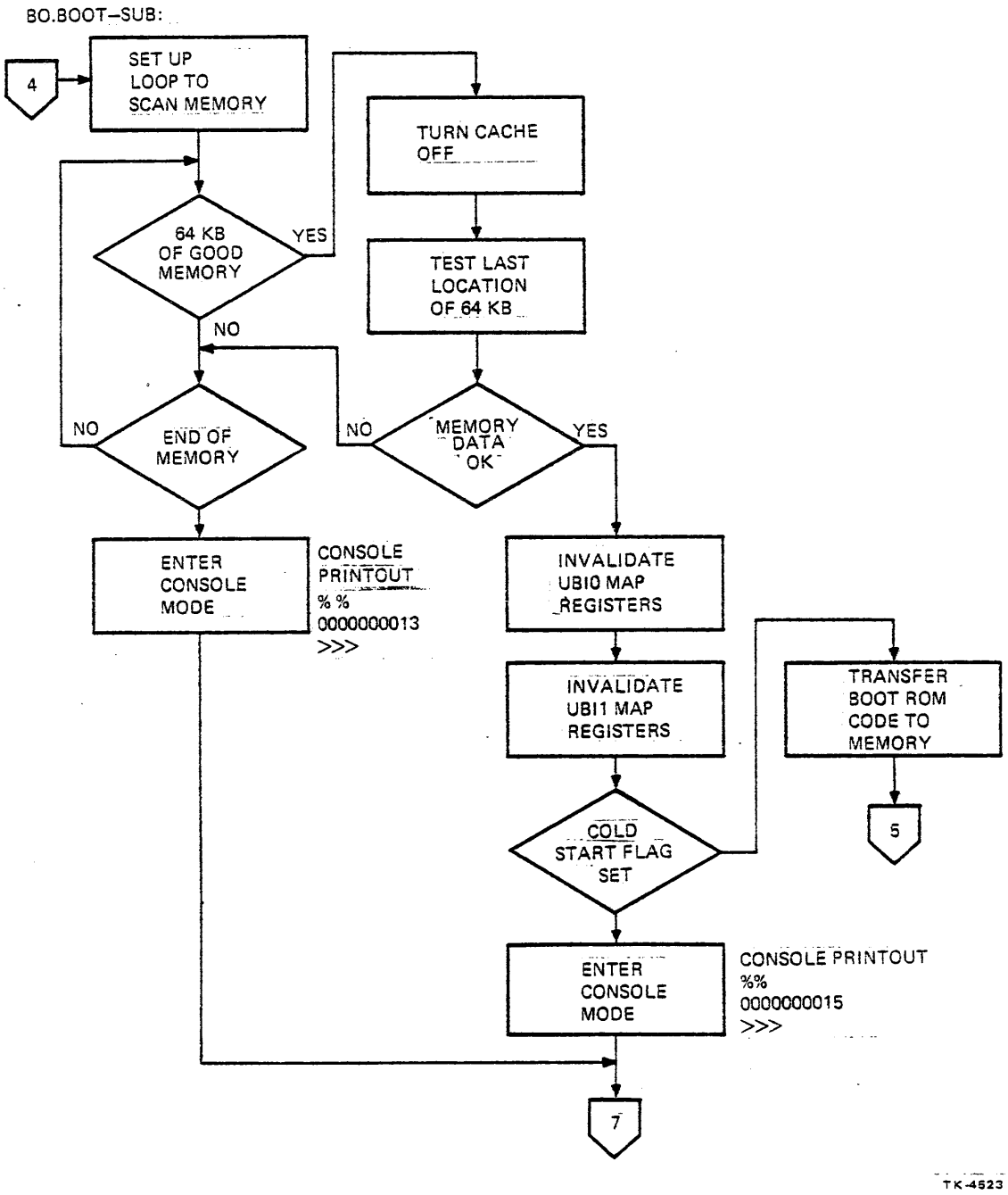

BO.BOOT–SUB:
4
SET UP LOOP TO SCAN MEMORY
TURN CACHE OFF
64 KB OF GOOD MEMORY
YES
NO
TEST LAST LOCATION OF 64 KB
NO
END OF MEMORY
NO
MEMORY DATA OK
YES
ENTER CONSOLE MODE
CONSOLE PRINTOUT
% %
0000000013
>>>
INVALIDATE UBI0 MAP REGISTERS
INVALIDATE UBI1 MAP REGISTERS
TRANSFER BOOT ROM CODE TO MEMORY
5
COLD START FLAG SET
ENTER CONSOLE MODE
CONSOLE PRINTOUT
%%
0000000015
>>>
7
TK-4523

Figure 6-30

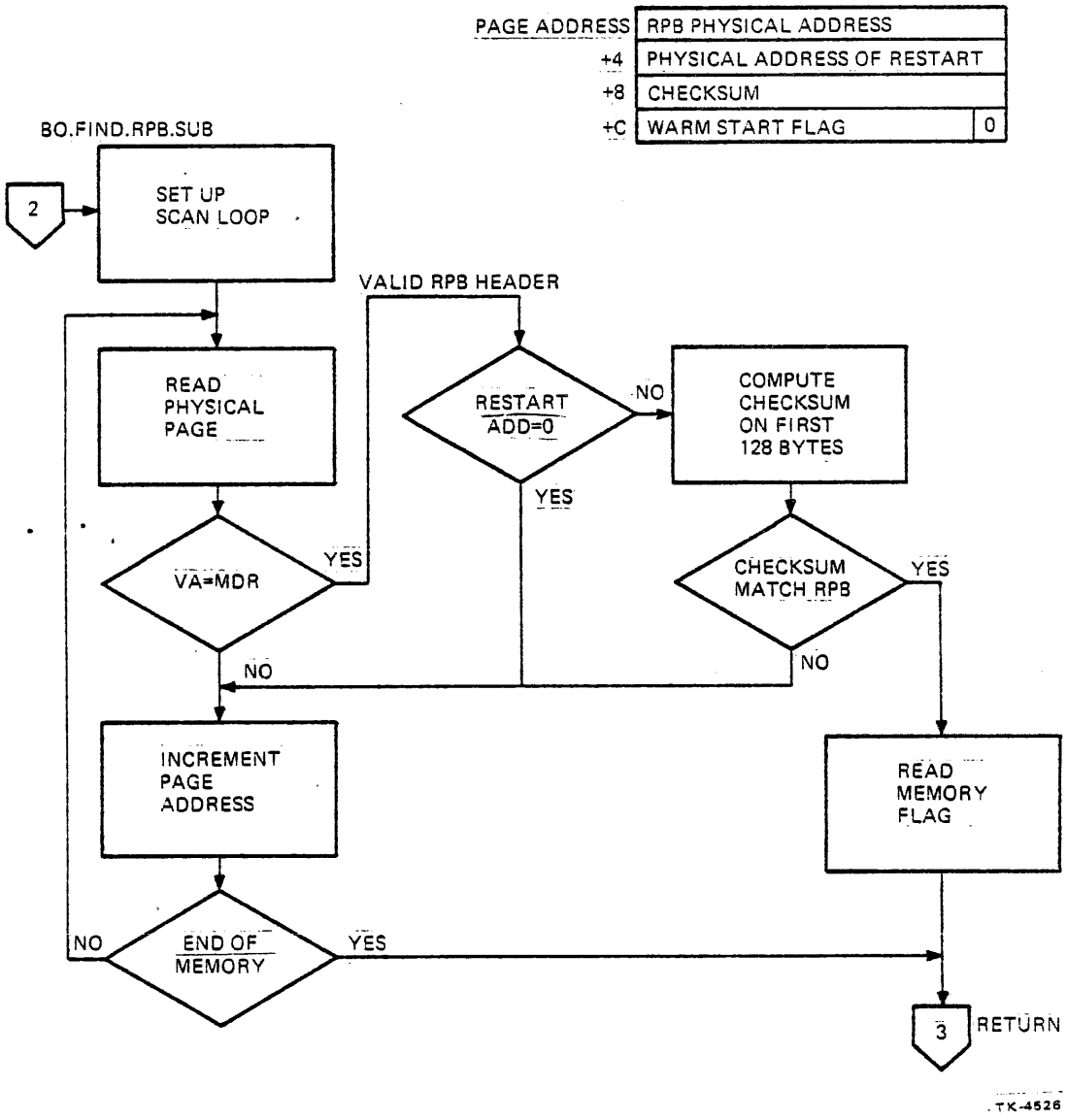

PAGE ADDRESS
RPB PHYSICAL ADDRESS
+4
PHYSICAL ADDRESS OF RESTART
+8
CHECKSUM
+C
WARM START FLAG
0
BO.FIND.RPB.SUB
2
SET UP SCAN LOOP
VALID RPB HEADER
READ PHYSICAL PAGE
RESTART ADD=0
NO
COMPUTE CHECKSUM ON FIRST 128 BYTES
YES
VA=MDR
YES
CHECKSUM MATCH RPB
YES
NO
NO
INCREMENT PAGE ADDRESS
READ MEMORY FLAG
NO
END OF MEMORY
YES
3
RETURN
TK-4526

Figure 6-31

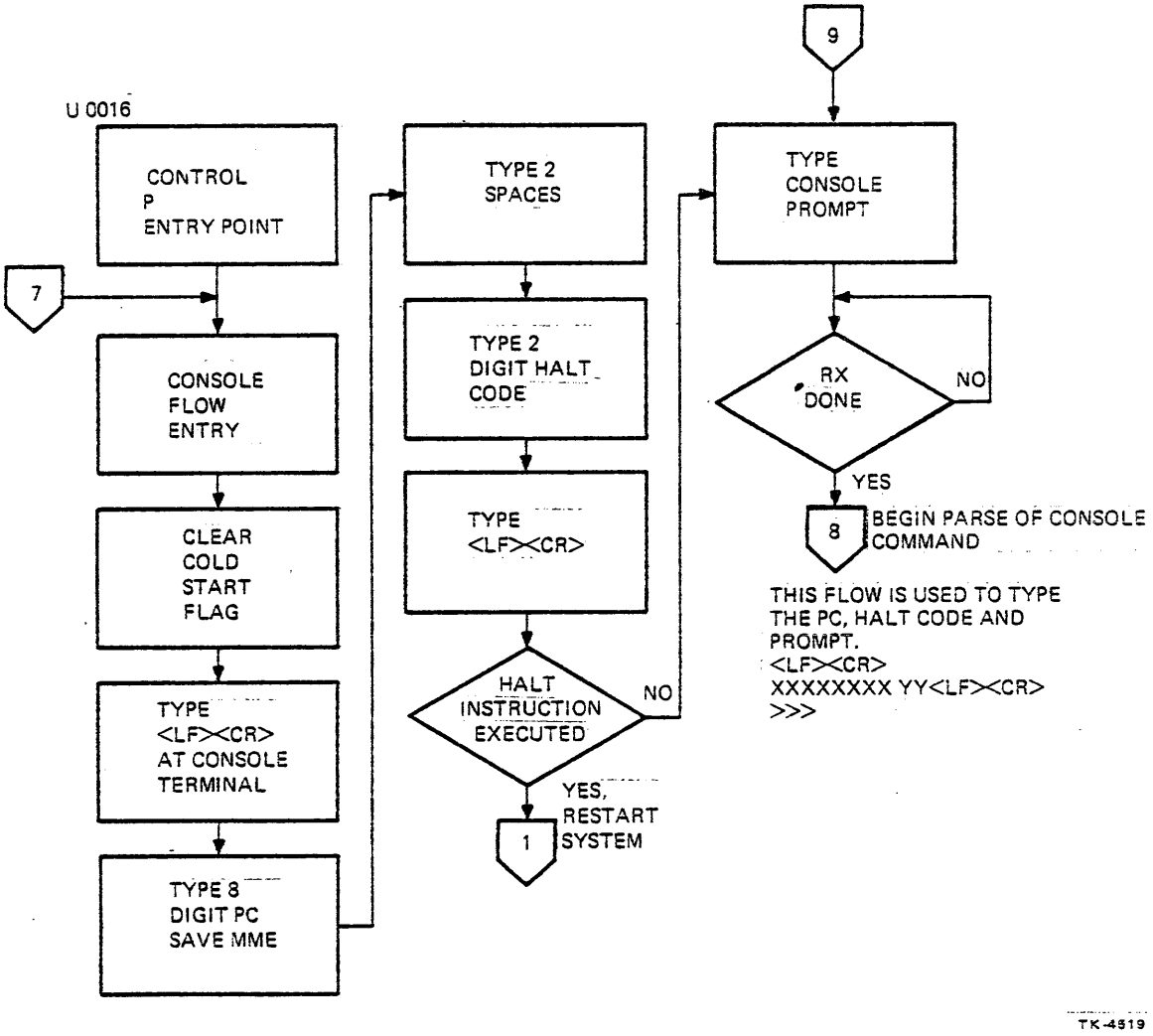

U 0016
CONTROL P ENTRY POINT
7
CONSOLE FLOW ENTRY
CLEAR COLD START FLAG
TYPE <LF><CR> AT CONSOLE TERMINAL
TYPE 8 DIGIT PC SAVE MME
TYPE 2 SPACES
TYPE 2 DIGIT HALT CODE
TYPE <LF><CR>
HALT INSTRUCTION EXECUTED
NO
YES, RESTART SYSTEM
1
9
TYPE CONSOLE PROMPT
RX DONE
NO
YES
8
BEGIN PARSE OF CONSOLE COMMAND
THIS FLOW IS USED TO TYPE THE PC, HALT CODE AND PROMPT.
<LF><CR>
XXXXXXXX YY<LF><CR>
>>>
TK-4519

Figure 6-32

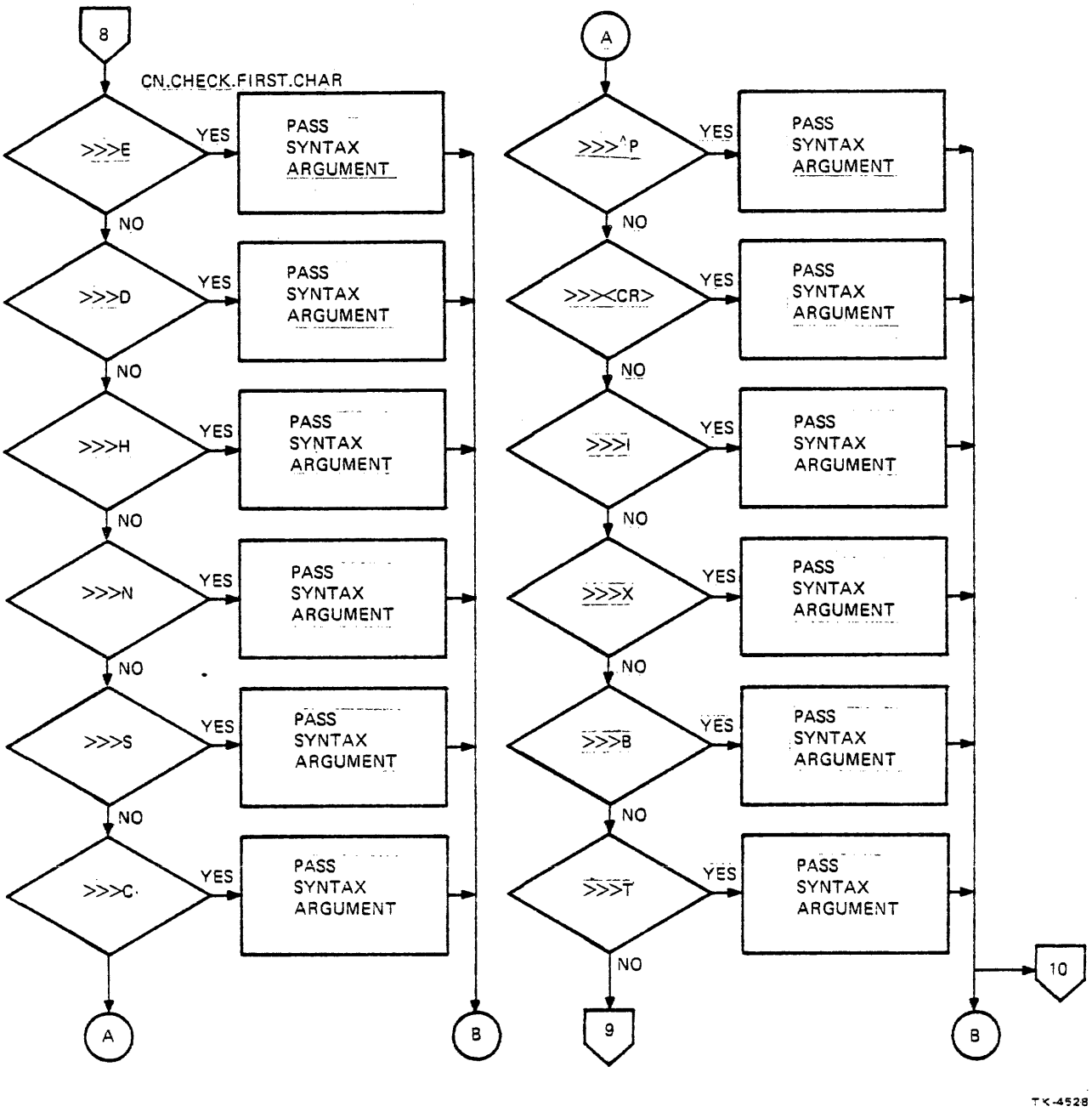

8
CN.CHECK.FIRST.CHAR
>>>E
YES
PASS SYNTAX ARGUMENT
NO
>>>D
YES
PASS SYNTAX ARGUMENT
NO
>>>H
YES
PASS SYNTAX ARGUMENT
NO
>>>N
YES
PASS SYNTAX ARGUMENT
NO
>>>S
YES
PASS SYNTAX ARGUMENT
NO
>>>C
YES
PASS SYNTAX ARGUMENT
A
B
A
>>>^P
YES
PASS SYNTAX ARGUMENT
NO
>>><CR>
YES
PASS SYNTAX ARGUMENT
NO
>>>I
YES
PASS SYNTAX ARGUMENT
NO
>>>X
YES
PASS SYNTAX ARGUMENT
NO
>>>B
YES
PASS SYNTAX ARGUMENT
NO
>>>T
YES
PASS SYNTAX ARGUMENT
NO
9
10
B
TK-4528

Figure 6-33

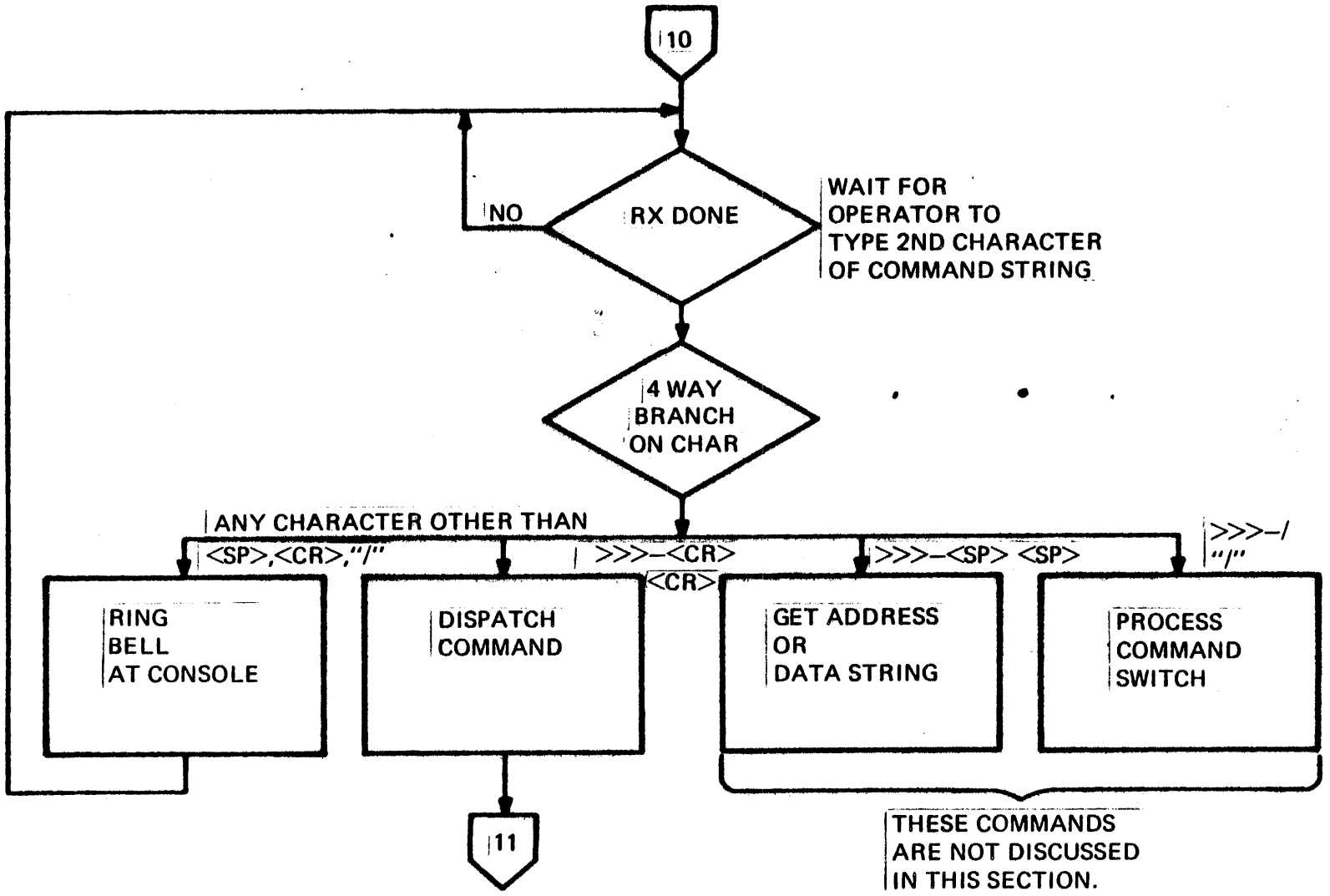

10
NO
RX DONE
WAIT FOR OPERATOR TO TYPE 2ND CHARACTER OF COMMAND STRING
4 WAY BRANCH ON CHAR
ANY CHARACTER OTHER THAN <SP>,<CR>,"/"
>>>–<CR>
<CR>
>>>–<SP> <SP>
>>>–/ "/"
RING BELL AT CONSOLE
DISPATCH COMMAND
GET ADDRESS OR DATA STRING
PROCESS COMMAND SWITCH
11
THESE COMMANDS ARE NOT DISCUSSED IN THIS SECTION.
TK-4522

Figure 6-34

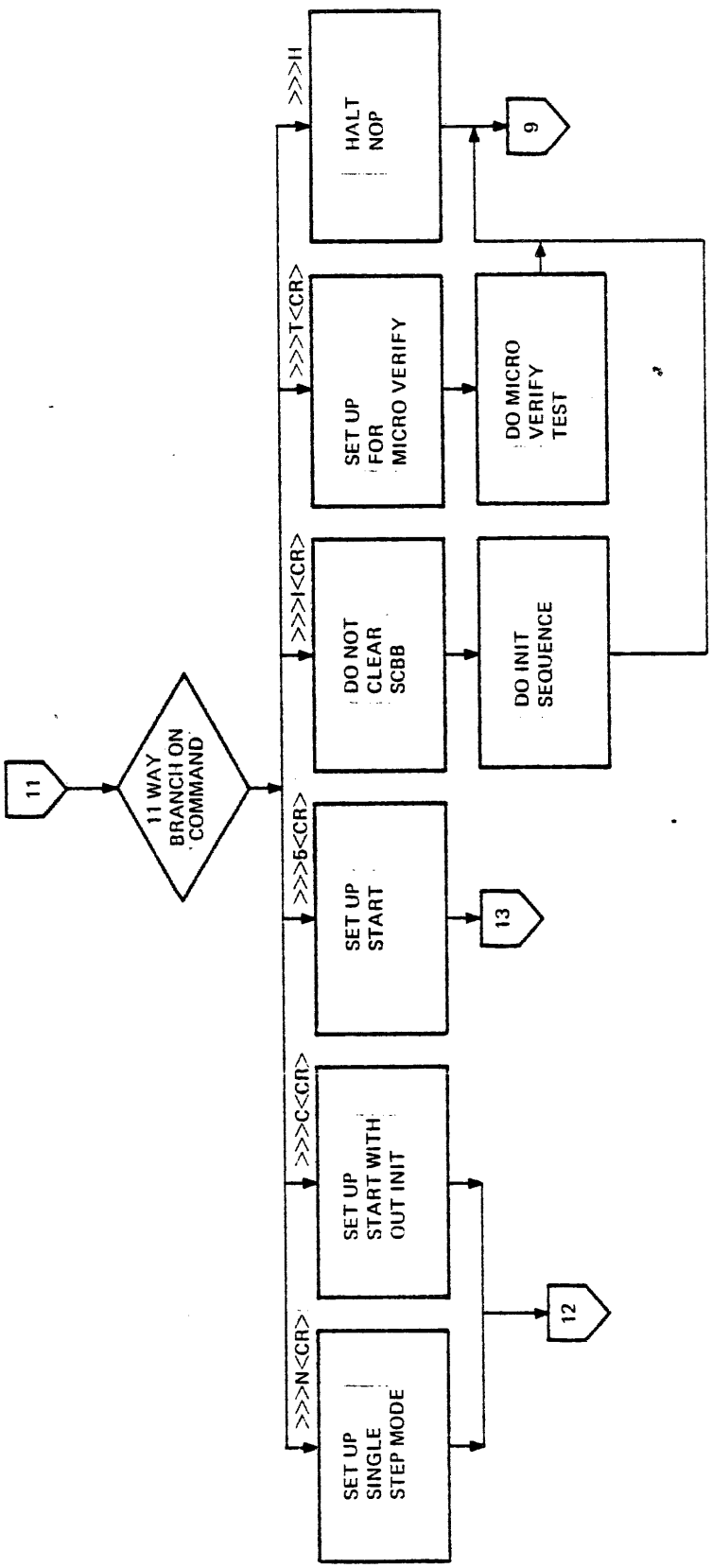

TK 4520
11
11 WAY BRANCH ON COMMAND
>>>H
HALT NOP
9
>>>T<CR>
SET UP FOR MICRO VERIFY
DO MICRO VERIFY TEST
>>>I<CR>
DO NOT CLEAR SCBB
DO INIT SEQUENCE
>>>6<CR>
SET UP START
13
>>>C<CR>
SET UP START WITH OUT INIT
>>>N<CR>
SET UP SINGLE STEP MODE
12

Figure 6-35

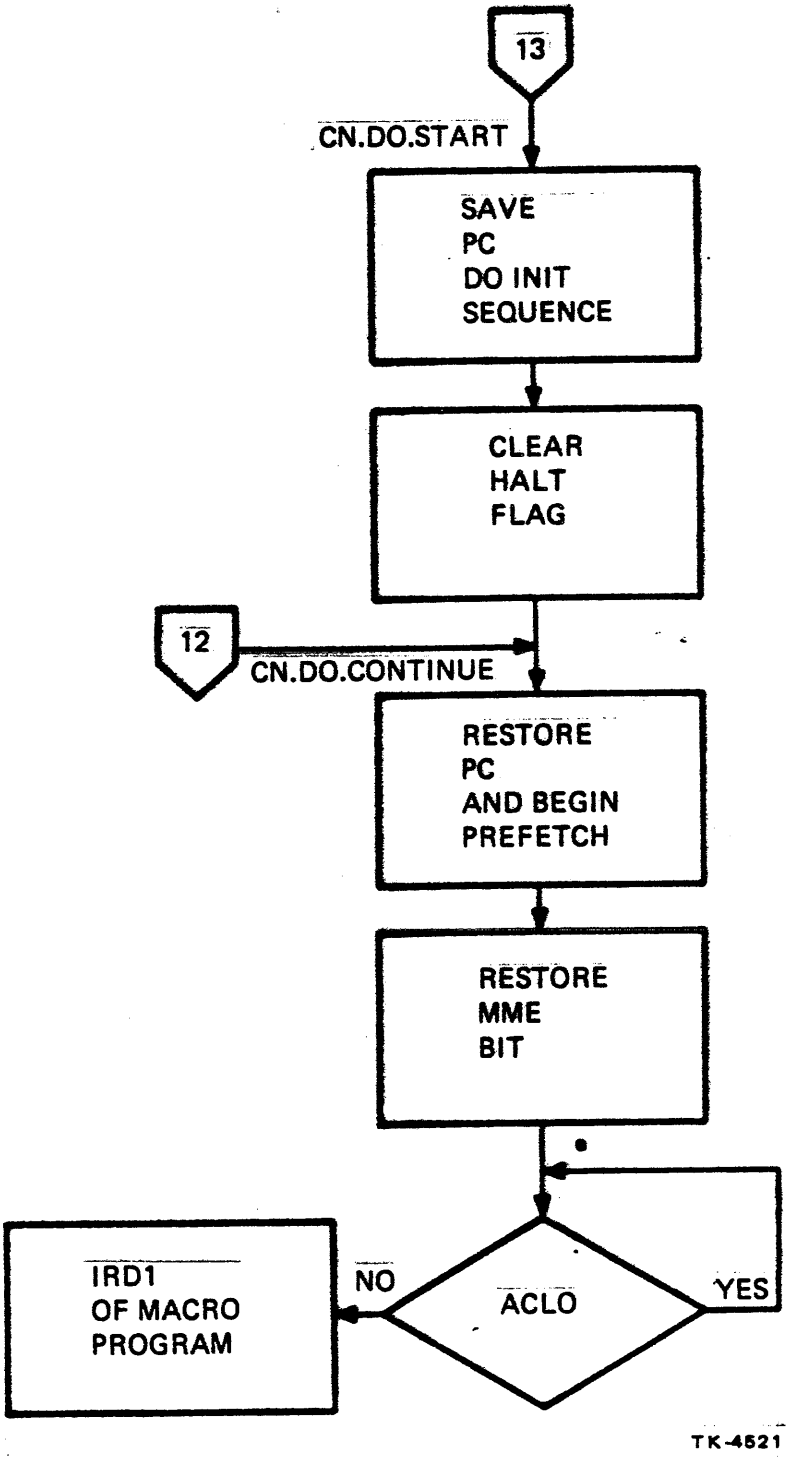
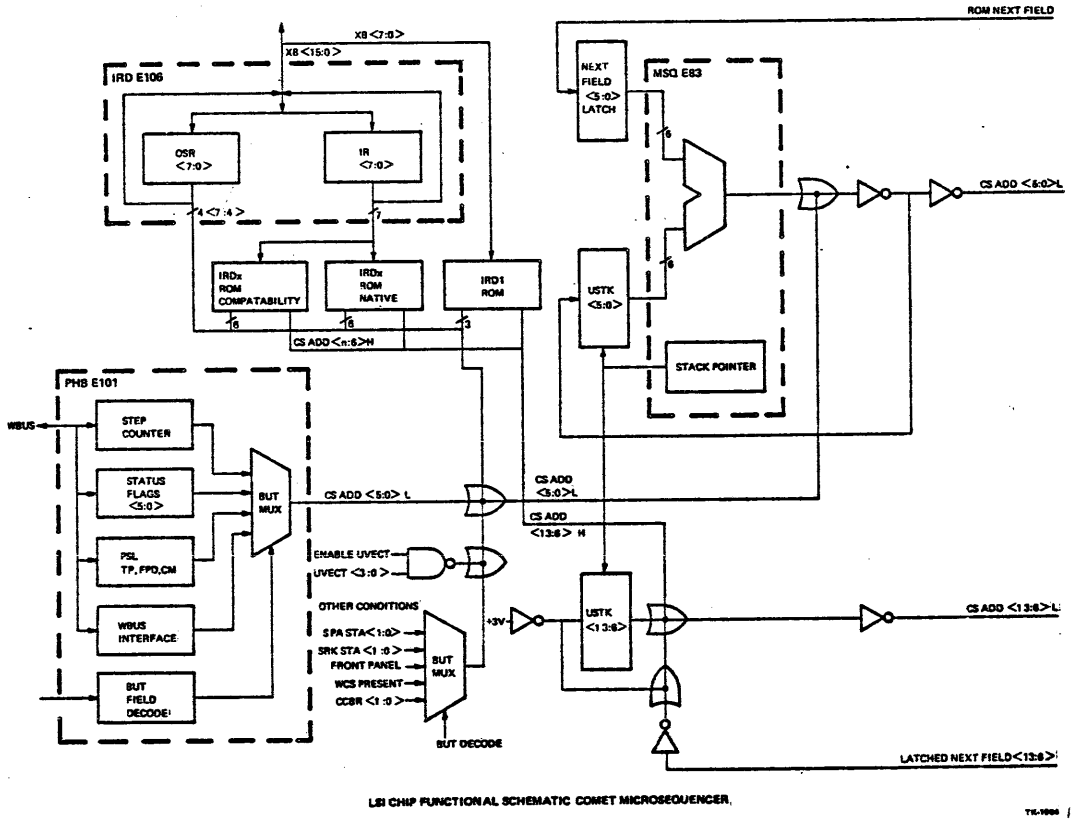

13
CN.DO.START
SAVE
PC
DO INIT
SEQUENCE
CLEAR
HALT
FLAG
12
CN.DO.CONTINUE
RESTORE
PC
AND BEGIN
PREFETCH
RESTORE
MME
BIT
IRD1
OF MACRO
PROGRAM
NO
ACLO
YES
TK-4521

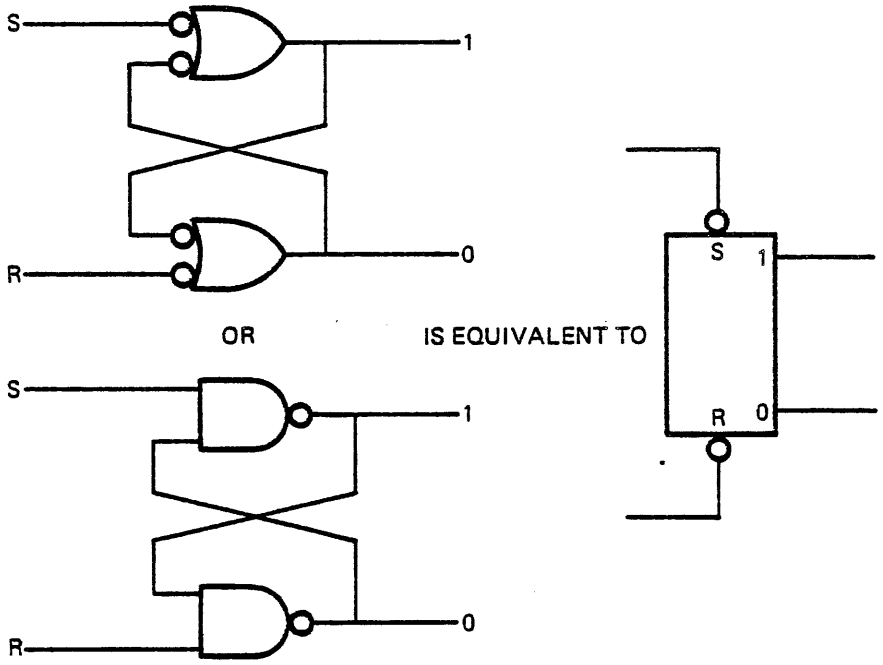
Figure 6-36

Figure 6-37



LSI CHIP FUNCTIONAL SCHEMATIC COMET MICROSEQUENCER.

Figure 6-38



LOGIC CIRCUIT EQUIVALENCES

TK-2097

Figure 6-39

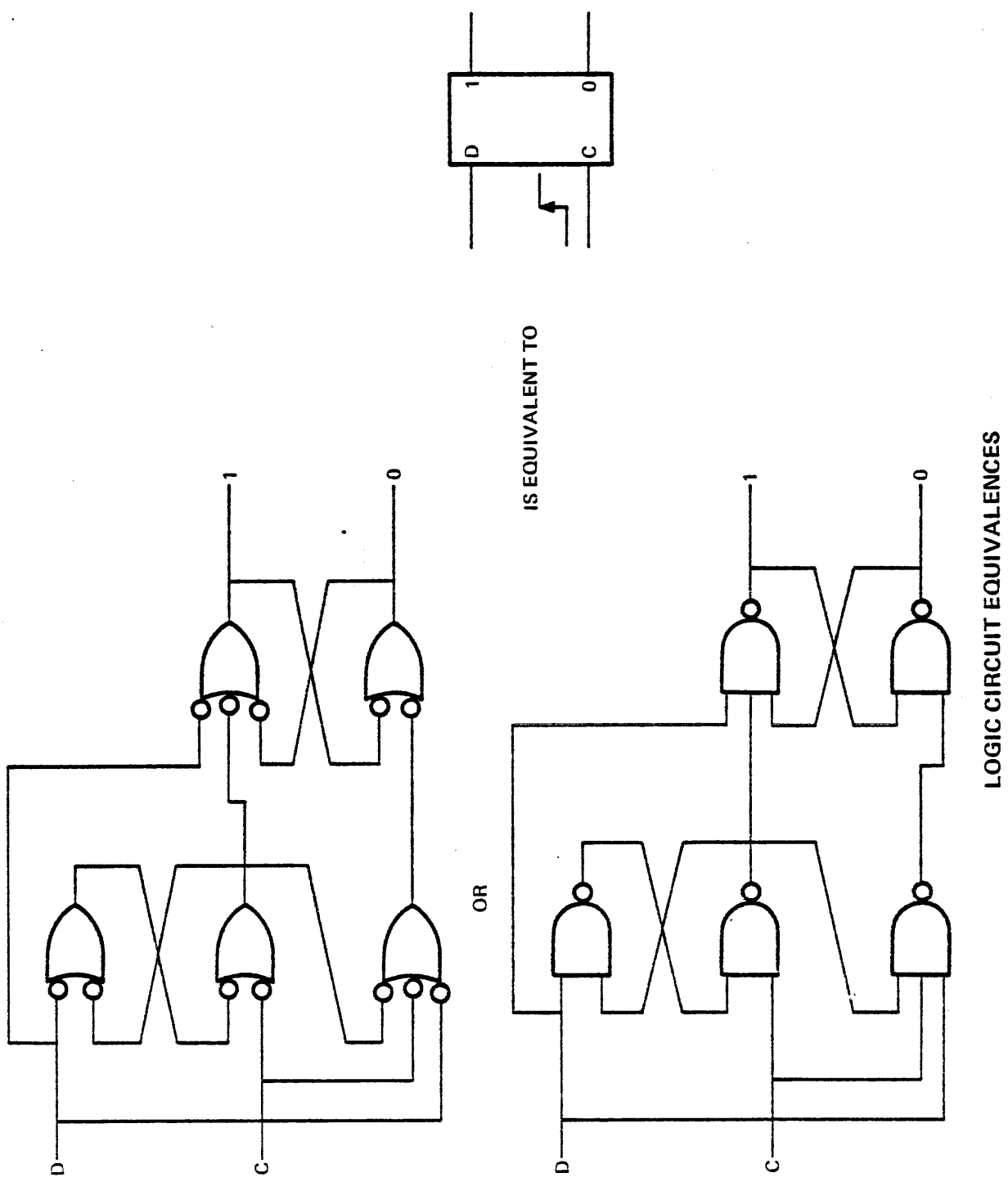
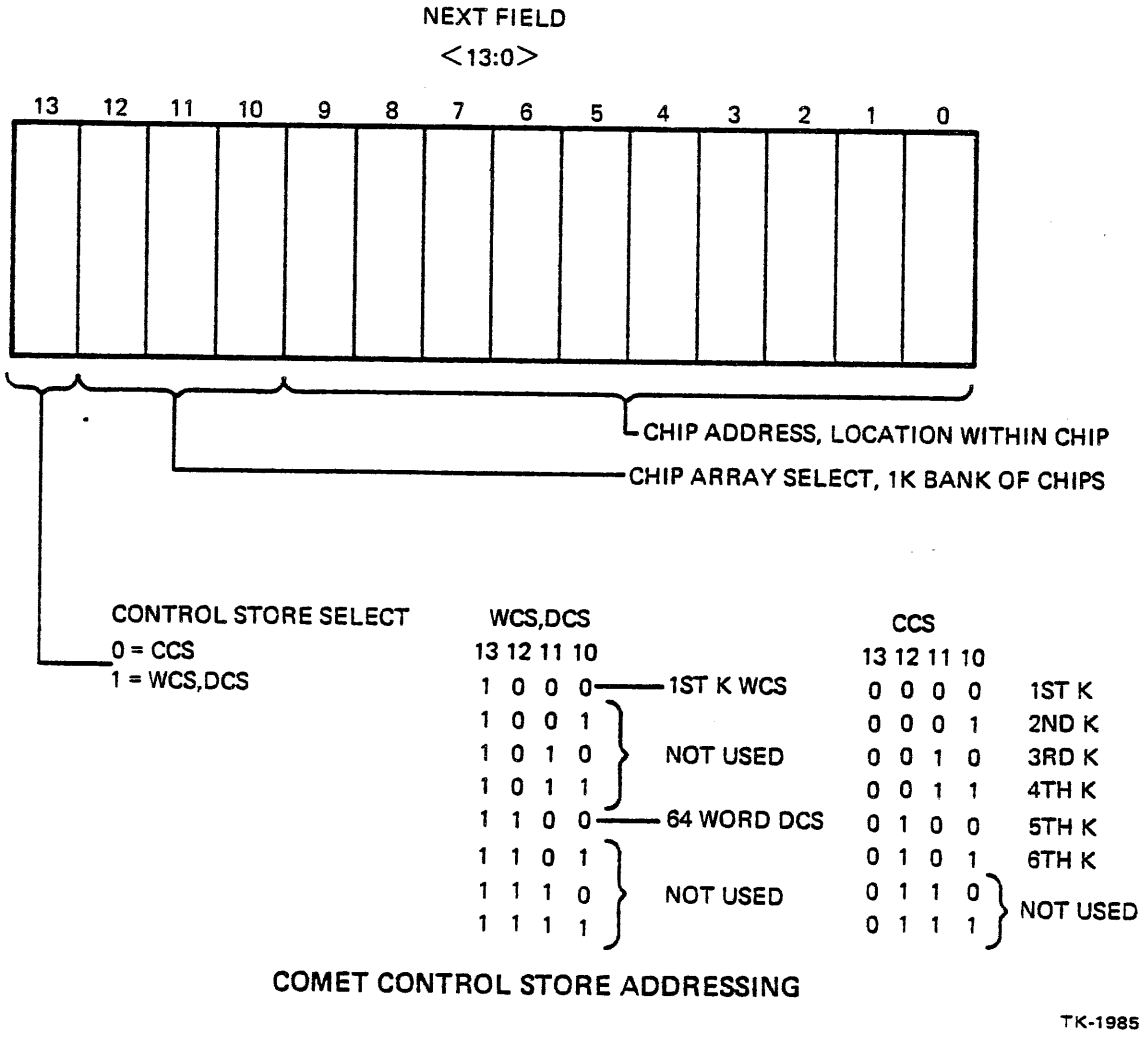

TK-2098
LOGIC CIRCUIT EQUIVALENCES
1
0
D
C
IS EQUIVALENT TO
OR
1
0
D
C

Figure 6-40



COMET CONTROL STORE ADDRESSING

TK-1985

Figure 6-41

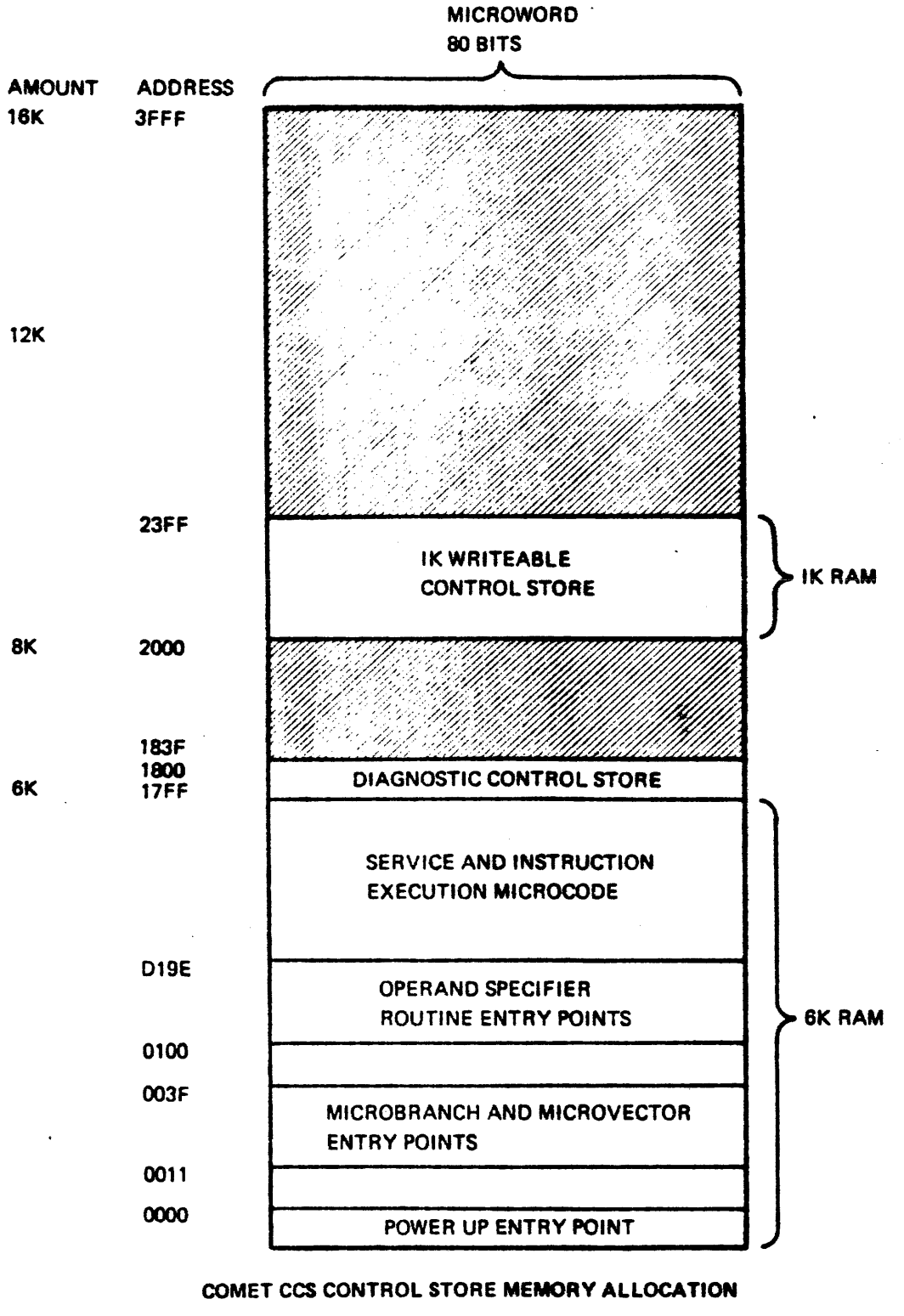

MICROWORD
80 BITS
AMOUNT
ADDRESS
16K
3FFF
12K
23FF
IK WRITEABLE
CONTROL STORE
IK RAM
8K
2000
183F
1800
6K
17FF
DIAGNOSTIC CONTROL STORE
SERVICE AND INSTRUCTION
EXECUTION MICROCODE
D19E
OPERAND SPECIFIER
ROUTINE ENTRY POINTS
6K RAM
0100
003F
MICROBRANCH AND MICROVECTOR
ENTRY POINTS
0011
0000
POWER UP ENTRY POINT
COMET CCS CONTROL STORE MEMORY ALLOCATION
TK-1983

Figure 6-42

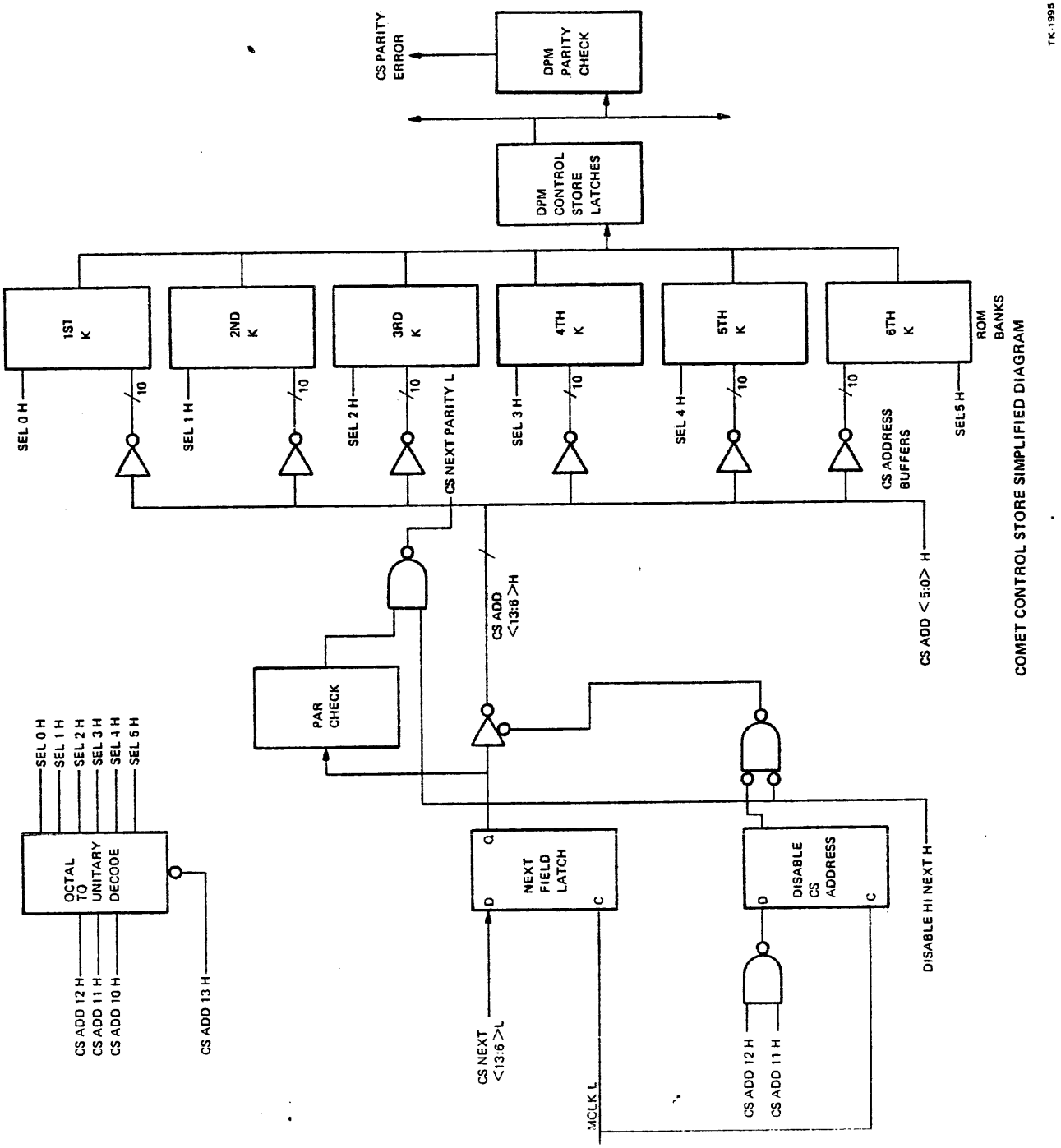

CS PARITY ERROR
DPM PARITY CHECK
DPM CONTROL STORE LATCHES
1ST K
2ND K
3RD K
4TH K
5TH K
6TH K
ROM BANKS
SEL 0 H
SEL 1 H
SEL 2 H
SEL 3 H
SEL 4 H
SEL5 H
10
CS NEXT PARITY L
CS ADDRESS BUFFERS
CS ADD <5:0> H
CS ADD <13:6>H
PAR CHECK
OCTAL TO UNITARY DECODE
CS ADD 12 H
CS ADD 11 H
CS ADD 10 H
CS ADD 13 H
NEXT FIELD LATCH
DISABLE CS ADDRESS
DISABLE HI NEXT H
CS NEXT <13:6>L
MCLK L
TK-1995
COMET CONTROL STORE SIMPLIFIED DIAGRAM

Figure 6-43

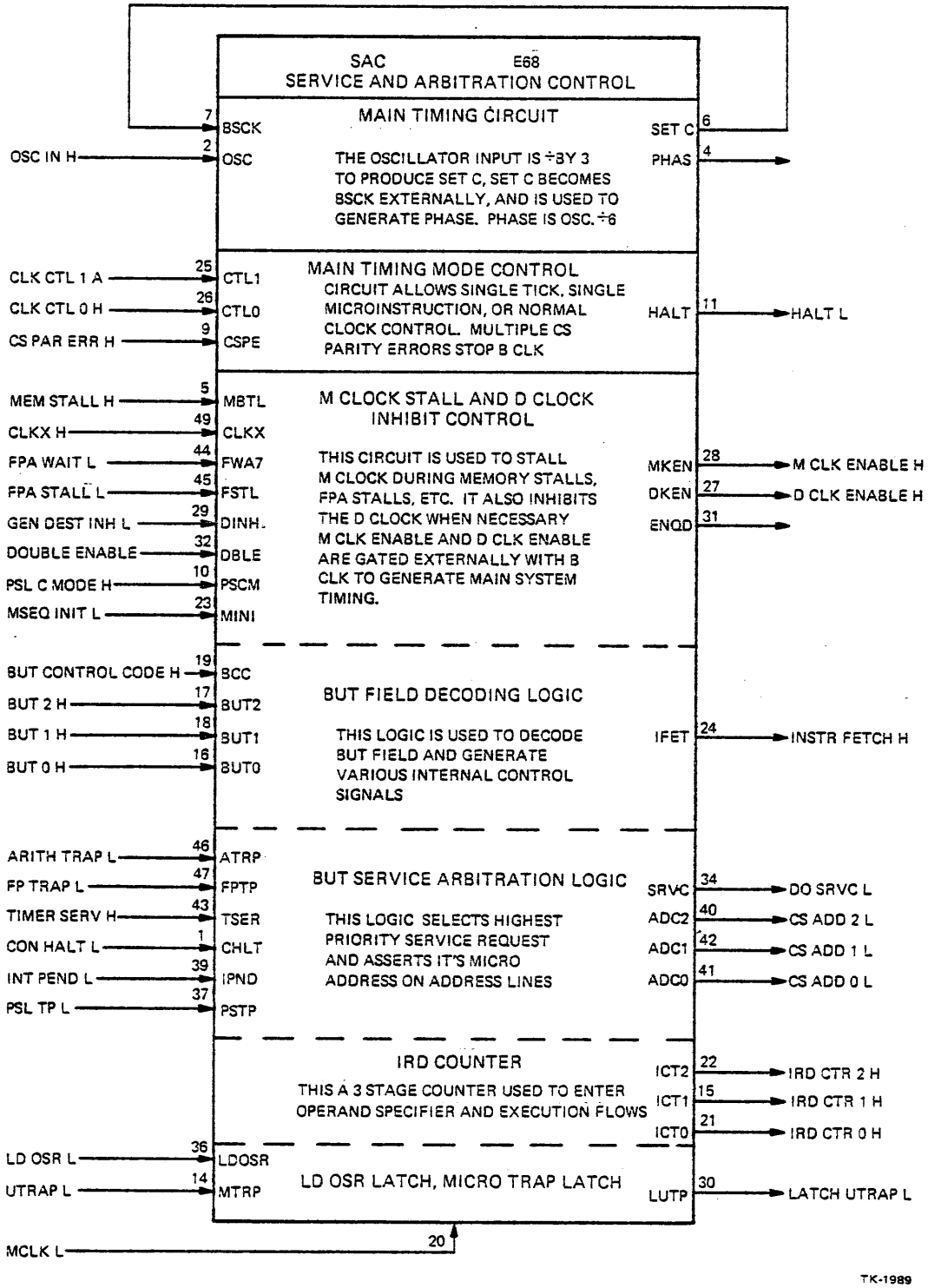

SAC E68
SERVICE AND ARBITRATION CONTROL
MAIN TIMING CIRCUIT
7 BSCK
SET C 6
OSC IN H 2 OSC
THE OSCILLATOR INPUT IS ÷BY 3 TO PRODUCE SET C, SET C BECOMES BSCK EXTERNALLY, AND IS USED TO GENERATE PHASE. PHASE IS OSC. ÷6
PHAS 4
MAIN TIMING MODE CONTROL
CLK CTL 1 A 25 CTL1
CLK CTL 0 H 26 CTL0
CS PAR ERR H 9 CSPE
CIRCUIT ALLOWS SINGLE TICK, SINGLE MICROINSTRUCTION, OR NORMAL CLOCK CONTROL. MULTIPLE CS PARITY ERRORS STOP B CLK
HALT 11 HALT L
M CLOCK STALL AND D CLOCK INHIBIT CONTROL
MEM STALL H 5 MBTL
CLKX H 49 CLKX
FPA WAIT L 44 FWA7
FPA STALL L 45 FSTL
GEN DEST INH L 29 DINH
DOUBLE ENABLE 32 DBLE
PSL C MODE H 10 PSCM
MSEQ INIT L 23 MINI
THIS CIRCUIT IS USED TO STALL M CLOCK DURING MEMORY STALLS, FPA STALLS, ETC. IT ALSO INHIBITS THE D CLOCK WHEN NECESSARY M CLK ENABLE AND D CLK ENABLE ARE GATED EXTERNALLY WITH B CLK TO GENERATE MAIN SYSTEM TIMING.
MKEN 28 M CLK ENABLE H
DKEN 27 D CLK ENABLE H
ENQD 31
BUT FIELD DECODING LOGIC
BUT CONTROL CODE H 19 BCC
BUT 2 H 17 BUT2
BUT 1 H 18 BUT1
BUT 0 H 16 BUT0
THIS LOGIC IS USED TO DECODE BUT FIELD AND GENERATE VARIOUS INTERNAL CONTROL SIGNALS
IFET 24 INSTR FETCH H
BUT SERVICE ARBITRATION LOGIC
ARITH TRAP L 46 ATRP
FP TRAP L 47 FPTP
TIMER SERV H 43 TSER
CON HALT L 1 CHLT
INT PEND L 39 IPND
PSL TP L 37 PSTP
THIS LOGIC SELECTS HIGHEST PRIORITY SERVICE REQUEST AND ASSERTS IT'S MICRO ADDRESS ON ADDRESS LINES
SRVC 34 DO SRVC L
ADC2 40 CS ADD 2 L
ADC1 42 CS ADD 1 L
ADC0 41 CS ADD 0 L
IRD COUNTER
THIS A 3 STAGE COUNTER USED TO ENTER OPERAND SPECIFIER AND EXECUTION FLOWS
ICT2 22 IRD CTR 2 H
ICT1 15 IRD CTR 1 H
ICT0 21 IRD CTR 0 H
LD OSR L 36 LDOSR
UTRAP L 14 MTRP
LD OSR LATCH, MICRO TRAP LATCH
LUTP 30 LATCH UTRAP L
MCLK L 20
TK-1989

Figure 6-44

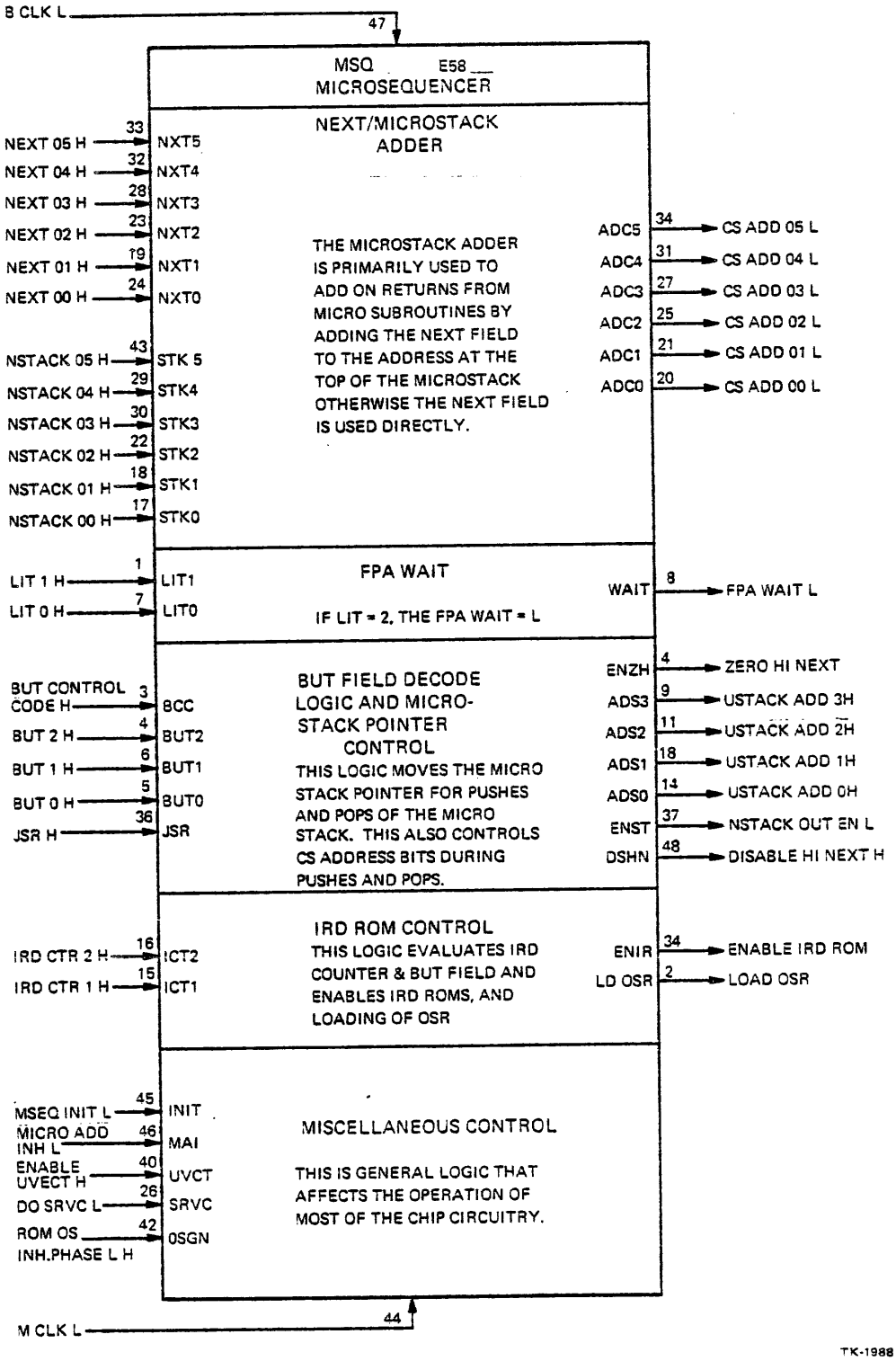

B CLK L
47
MSQ E58
MICROSEQUENCER
NEXT/MICROSTACK ADDER
NEXT 05 H
NEXT 04 H
NEXT 03 H
NEXT 02 H
NEXT 01 H
NEXT 00 H
NXT5
NXT4
NXT3
NXT2
NXT1
NXT0
NSTACK 05 H
NSTACK 04 H
NSTACK 03 H
NSTACK 02 H
NSTACK 01 H
NSTACK 00 H
STK 5
STK4
STK3
STK2
STK1
STK0
THE MICROSTACK ADDER IS PRIMARILY USED TO ADD ON RETURNS FROM MICRO SUBROUTINES BY ADDING THE NEXT FIELD TO THE ADDRESS AT THE TOP OF THE MICROSTACK OTHERWISE THE NEXT FIELD IS USED DIRECTLY.
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
CS ADD 05 L
CS ADD 04 L
CS ADD 03 L
CS ADD 02 L
CS ADD 01 L
CS ADD 00 L
FPA WAIT
LIT 1 H
LIT 0 H
LIT1
LIT0
IF LIT = 2, THE FPA WAIT = L
WAIT
FPA WAIT L
BUT FIELD DECODE LOGIC AND MICRO-STACK POINTER CONTROL
BUT CONTROL CODE H
BUT 2 H
BUT 1 H
BUT 0 H
JSR H
BCC
BUT2
BUT1
BUT0
JSR
THIS LOGIC MOVES THE MICRO STACK POINTER FOR PUSHES AND POPS OF THE MICRO STACK. THIS ALSO CONTROLS CS ADDRESS BITS DURING PUSHES AND POPS.
ENZH
ADS3
ADS2
ADS1
ADS0
ENST
DSHN
ZERO HI NEXT
USTACK ADD 3H
USTACK ADD 2H
USTACK ADD 1H
USTACK ADD 0H
NSTACK OUT EN L
DISABLE HI NEXT H
IRD ROM CONTROL
IRD CTR 2 H
IRD CTR 1 H
ICT2
ICT1
THIS LOGIC EVALUATES IRD COUNTER & BUT FIELD AND ENABLES IRD ROMS, AND LOADING OF OSR
ENIR
LD OSR
ENABLE IRD ROM
LOAD OSR
MISCELLANEOUS CONTROL
MSEQ INIT L
MICRO ADD INH L
ENABLE UVECT H
DO SRVC L
ROM OS INH.PHASE L H
INIT
MAI
UVCT
SRVC
OSGN
THIS IS GENERAL LOGIC THAT AFFECTS THE OPERATION OF MOST OF THE CHIP CIRCUITRY.
44
M CLK L
TK-1988

Figure 6-45

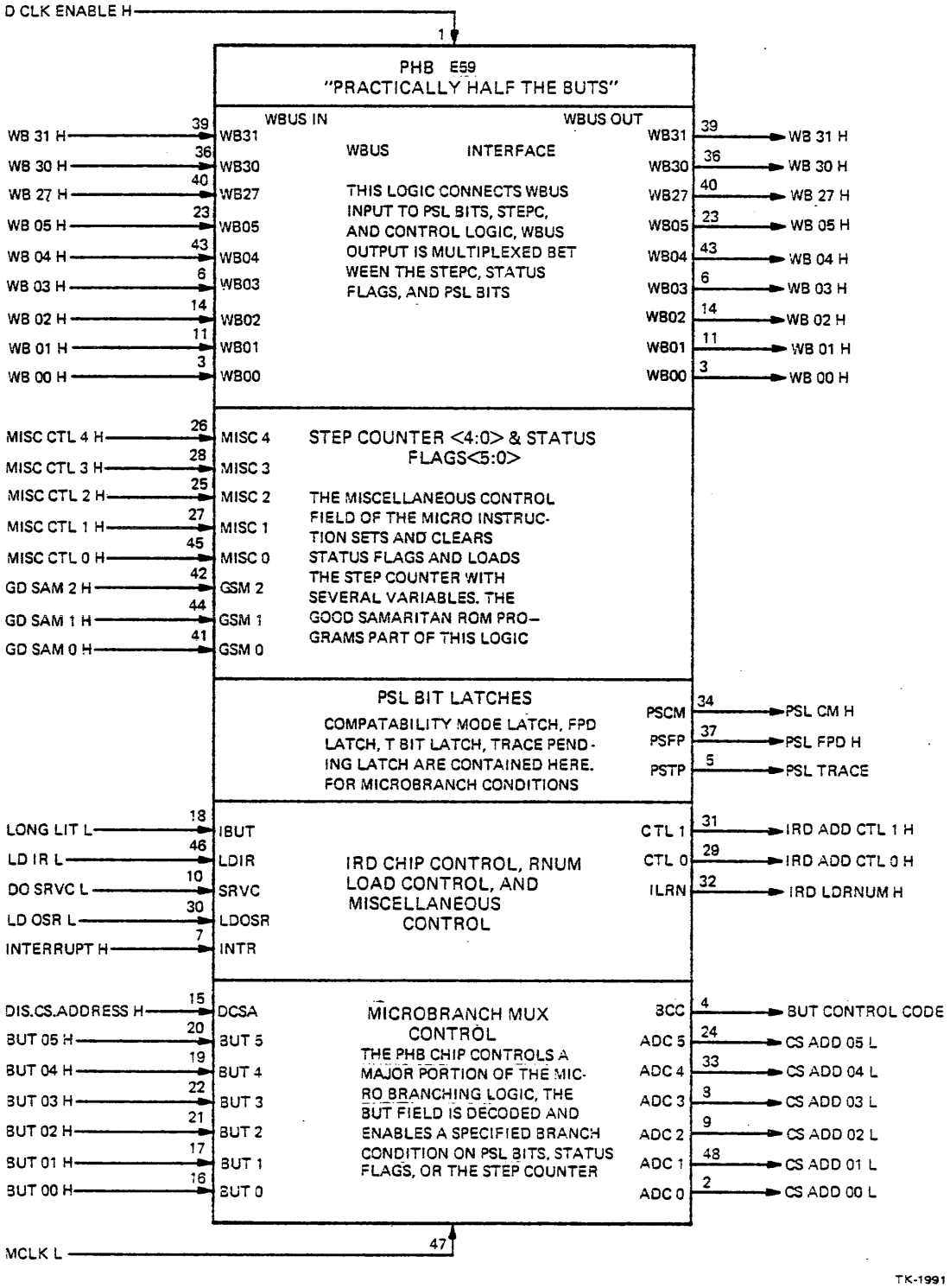

D CLK ENABLE H
1
PHB E59
"PRACTICALLY HALF THE BUTS"
WBUS IN
WBUS OUT
WB 31 H
WB 30 H
WB 27 H
WB 05 H
WB 04 H
WB 03 H
WB 02 H
WB 01 H
WB 00 H
WB31
WB30
WB27
WB05
WB04
WB03
WB02
WB01
WB00
WBUS INTERFACE
THIS LOGIC CONNECTS WBUS INPUT TO PSL BITS, STEPC, AND CONTROL LOGIC, WBUS OUTPUT IS MULTIPLEXED BETWEEN THE STEPC, STATUS FLAGS, AND PSL BITS
MISC CTL 4 H
MISC CTL 3 H
MISC CTL 2 H
MISC CTL 1 H
MISC CTL 0 H
GD SAM 2 H
GD SAM 1 H
GD SAM 0 H
MISC 4
MISC 3
MISC 2
MISC 1
MISC 0
GSM 2
GSM 1
GSM 0
STEP COUNTER <4:0> & STATUS FLAGS<5:0>
THE MISCELLANEOUS CONTROL FIELD OF THE MICRO INSTRUCTION SETS AND CLEARS STATUS FLAGS AND LOADS THE STEP COUNTER WITH SEVERAL VARIABLES. THE GOOD SAMARITAN ROM PROGRAMS PART OF THIS LOGIC
PSL BIT LATCHES
COMPATABILITY MODE LATCH, FPD LATCH, T BIT LATCH, TRACE PENDING LATCH ARE CONTAINED HERE. FOR MICROBRANCH CONDITIONS
PSCM
PSFP
PSTP
PSL CM H
PSL FPD H
PSL TRACE
LONG LIT L
LD IR L
DO SRVC L
LD OSR L
INTERRUPT H
IBUT
LDIR
SRVC
LDOSR
INTR
IRD CHIP CONTROL, RNUM LOAD CONTROL, AND MISCELLANEOUS CONTROL
CTL 1
CTL 0
ILRN
IRD ADD CTL 1 H
IRD ADD CTL 0 H
IRD LDRNUM H
DIS.CS.ADDRESS H
BUT 05 H
BUT 04 H
BUT 03 H
BUT 02 H
BUT 01 H
BUT 00 H
DCSA
BUT 5
BUT 4
BUT 3
BUT 2
BUT 1
BUT 0
MICROBRANCH MUX CONTROL
THE PHB CHIP CONTROLS A MAJOR PORTION OF THE MICRO BRANCHING LOGIC, THE BUT FIELD IS DECODED AND ENABLES A SPECIFIED BRANCH CONDITION ON PSL BITS, STATUS FLAGS, OR THE STEP COUNTER
BCC
ADC 5
ADC 4
ADC 3
ADC 2
ADC 1
ADC 0
BUT CONTROL CODE
CS ADD 05 L
CS ADD 04 L
CS ADD 03 L
CS ADD 02 L
CS ADD 01 L
CS ADD 00 L
MCLK L
47
TK-1991

Figure 6-46

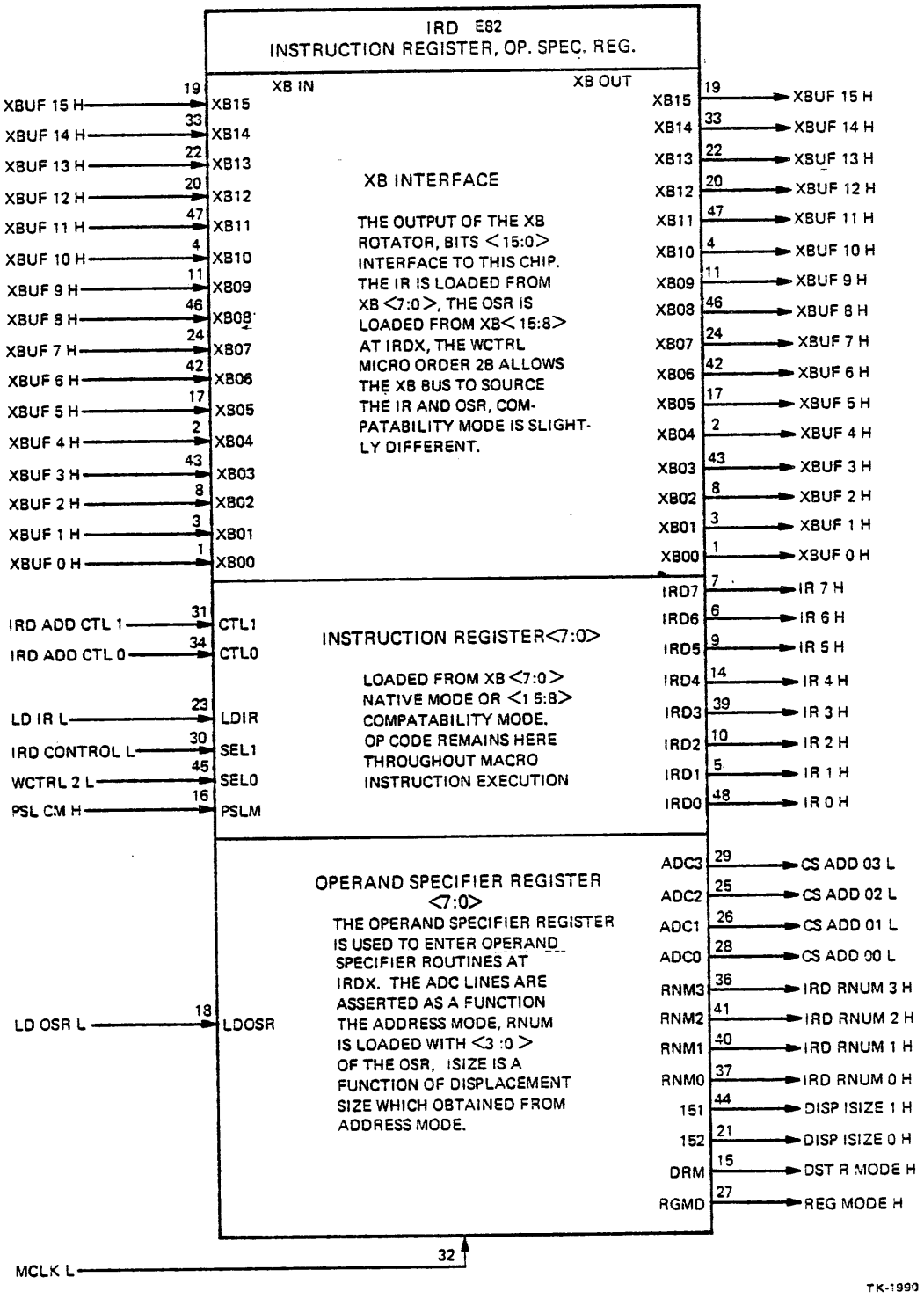

IRD E82
INSTRUCTION REGISTER, OP. SPEC. REG.
XB IN
XB OUT
XB INTERFACE
THE OUTPUT OF THE XB ROTATOR, BITS <15:0> INTERFACE TO THIS CHIP. THE IR IS LOADED FROM XB <7:0>, THE OSR IS LOADED FROM XB< 15:8> AT IRDX, THE WCTRL MICRO ORDER 28 ALLOWS THE XB BUS TO SOURCE THE IR AND OSR, COM-PATABILITY MODE IS SLIGHT-LY DIFFERENT.
XBUF 15 H
XBUF 14 H
XBUF 13 H
XBUF 12 H
XBUF 11 H
XBUF 10 H
XBUF 9 H
XBUF 8 H
XBUF 7 H
XBUF 6 H
XBUF 5 H
XBUF 4 H
XBUF 3 H
XBUF 2 H
XBUF 1 H
XBUF 0 H
XB15
XB14
XB13
XB12
XB11
XB10
XB09
XB08
XB07
XB06
XB05
XB04
XB03
XB02
XB01
XB00
INSTRUCTION REGISTER<7:0>
LOADED FROM XB <7:0> NATIVE MODE OR <15:8> COMPATABILITY MODE. OP CODE REMAINS HERE THROUGHOUT MACRO INSTRUCTION EXECUTION
IRD ADD CTL 1
IRD ADD CTL 0
LD IR L
IRD CONTROL L
WCTRL 2 L
PSL CM H
CTL1
CTL0
LDIR
SEL1
SEL0
PSLM
IRD7
IRD6
IRD5
IRD4
IRD3
IRD2
IRD1
IRD0
IR 7 H
IR 6 H
IR 5 H
IR 4 H
IR 3 H
IR 2 H
IR 1 H
IR 0 H
OPERAND SPECIFIER REGISTER <7:0>
THE OPERAND SPECIFIER REGISTER IS USED TO ENTER OPERAND SPECIFIER ROUTINES AT IRDX. THE ADC LINES ARE ASSERTED AS A FUNCTION THE ADDRESS MODE, RNUM IS LOADED WITH <3 :0 > OF THE OSR, ISIZE IS A FUNCTION OF DISPLACEMENT SIZE WHICH OBTAINED FROM ADDRESS MODE.
LD OSR L
LDOSR
ADC3
ADC2
ADC1
ADC0
RNM3
RNM2
RNM1
RNM0
151
152
DRM
RGMD
CS ADD 03 L
CS ADD 02 L
CS ADD 01 L
CS ADD 00 L
IRD RNUM 3 H
IRD RNUM 2 H
IRD RNUM 1 H
IRD RNUM 0 H
DISP ISIZE 1 H
DISP ISIZE 0 H
DST R MODE H
REG MODE H
MCLK L
TK-1990

Figure 6-47

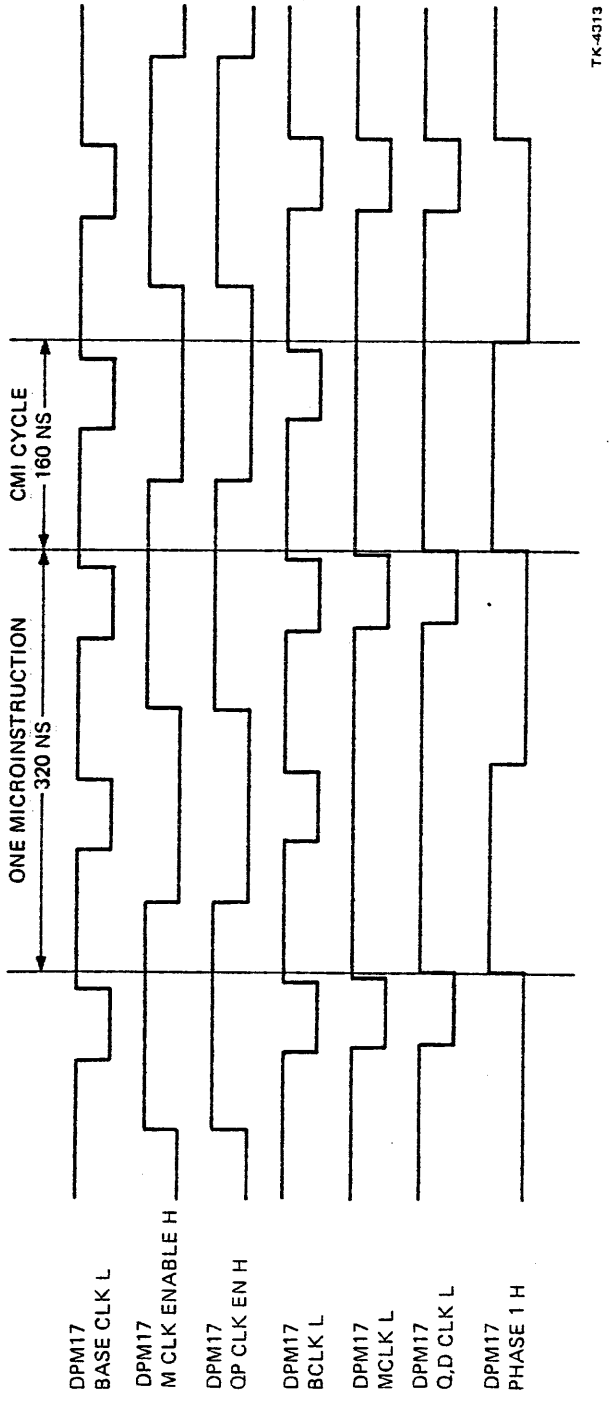

TK-4313
CMI CYCLE
160 NS
ONE MICROINSTRUCTION
320 NS
DPM17
BASE CLK L
DPM17
M CLK ENABLE H
DPM17
QP CLK EN H
DPM17
BCLK L
DPM17
MCLK L
DPM17
Q,D CLK L
DPM17
PHASE 1 H

Figure 6-48

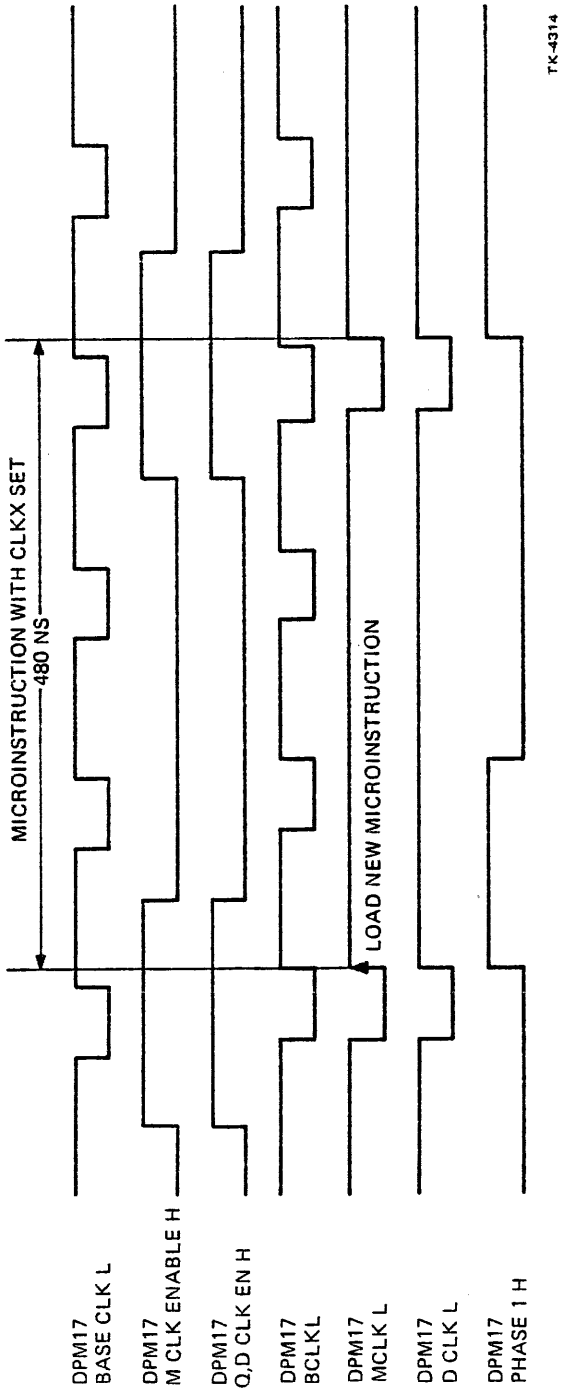

MICROINSTRUCTION WITH CLKX SET
480 NS
LOAD NEW MICROINSTRUCTION
DPM17
BASE CLK L
DPM17
M CLK ENABLE H
DPM17
Q,D CLK EN H
DPM17
BCLKL
DPM17
MCLK L
DPM17
D CLK L
DPM17
PHASE 1 H
TK-4314

Figure 6-49

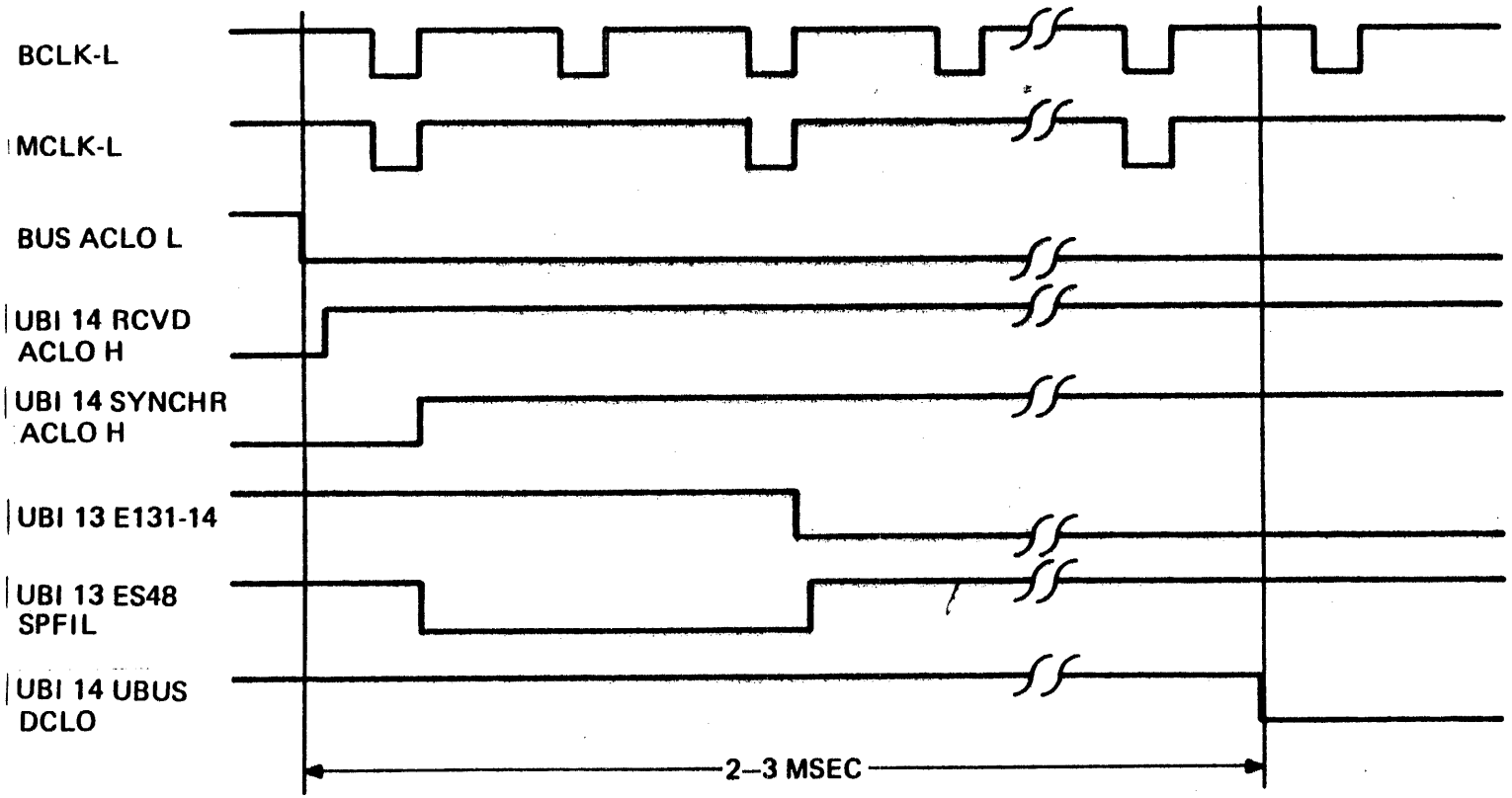

BCLK-L
MCLK-L
BUS ACLO L
UBI 14 RCVD ACLO H
UBI 14 SYNCHR ACLO H
UBI 13 E131-14
UBI 13 ES48 SPFIL
UBI 14 UBUS DCLO
2–3 MSEC
TK-4316

Figure 6–50

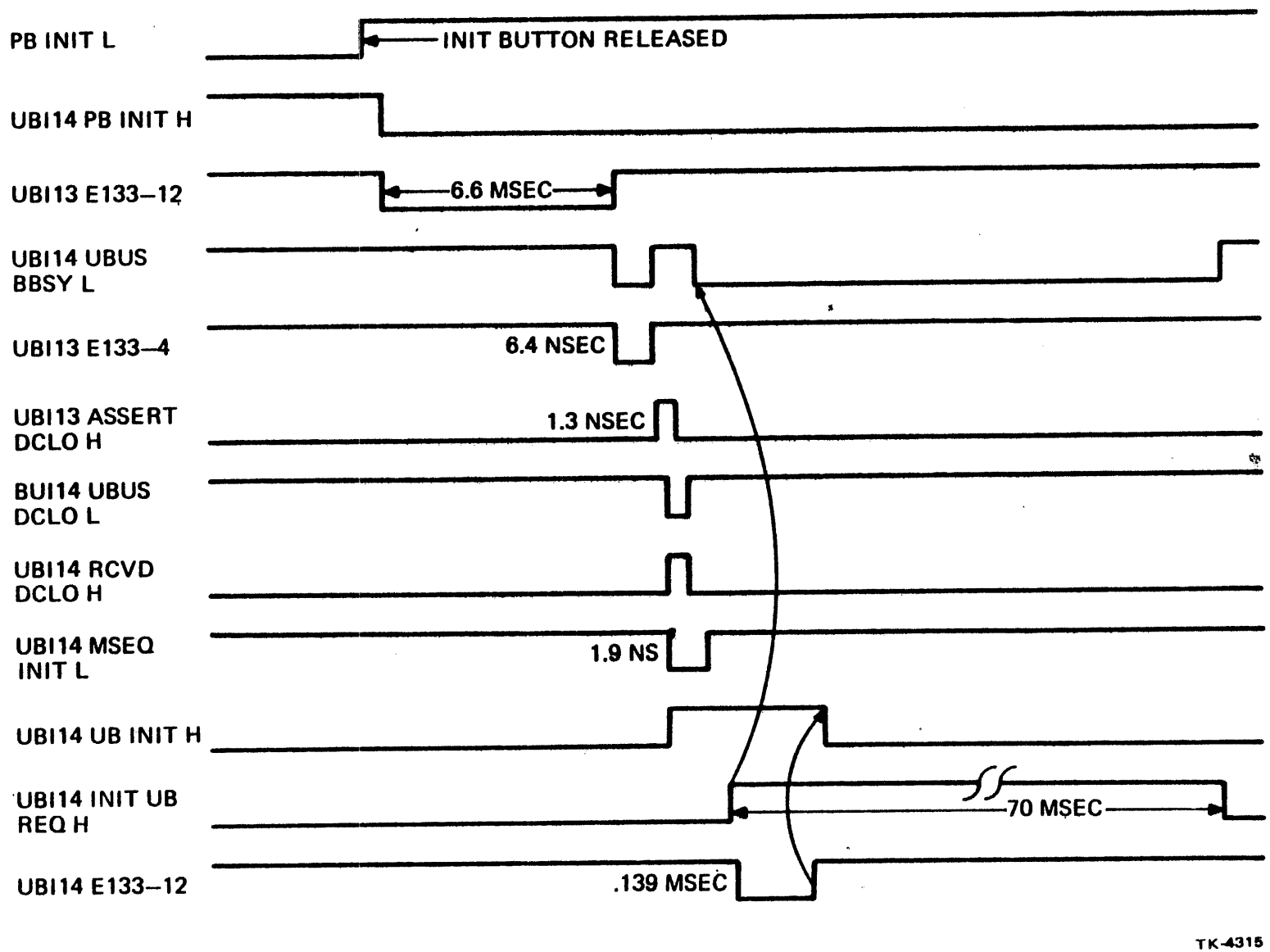

PB INIT L
INIT BUTTON RELEASED
UBI14 PB INIT H
UBI13 E133–12
6.6 MSEC
UBI14 UBUS BBSY L
UBI13 E133–4
6.4 NSEC
UBI13 ASSERT DCLO H
1.3 NSEC
BUI14 UBUS DCLO L
UBI14 RCVD DCLO H
UBI14 MSEQ INIT L
1.9 NS
UBI14 UB INIT H
UBI14 INIT UB REQ H
70 MSEC
UBI14 E133–12
.139 MSEC
TK-4315

Figure 6-51

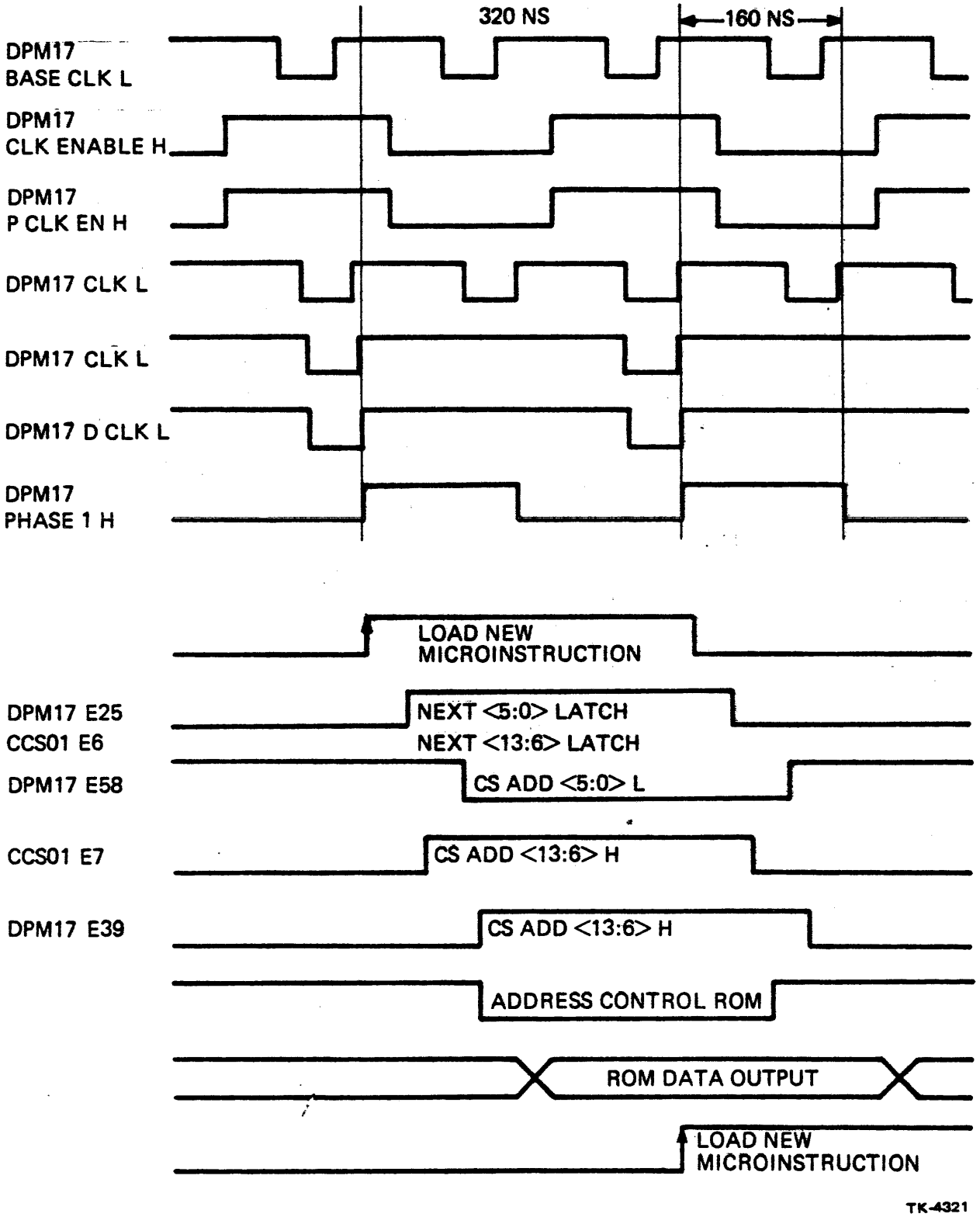

320 NS
160 NS
DPM17
BASE CLK L
DPM17
CLK ENABLE H
DPM17
P CLK EN H
DPM17 CLK L
DPM17 CLK L
DPM17 D CLK L
DPM17
PHASE 1 H
LOAD NEW
MICROINSTRUCTION
DPM17 E25
NEXT <5:0> LATCH
CCS01 E6
NEXT <13:6> LATCH
DPM17 E58
CS ADD <5:0> L
CCS01 E7
CS ADD <13:6> H
DPM17 E39
CS ADD <13:6> H
ADDRESS CONTROL ROM
ROM DATA OUTPUT
LOAD NEW
MICROINSTRUCTION
TK-4321

Figure 6-52

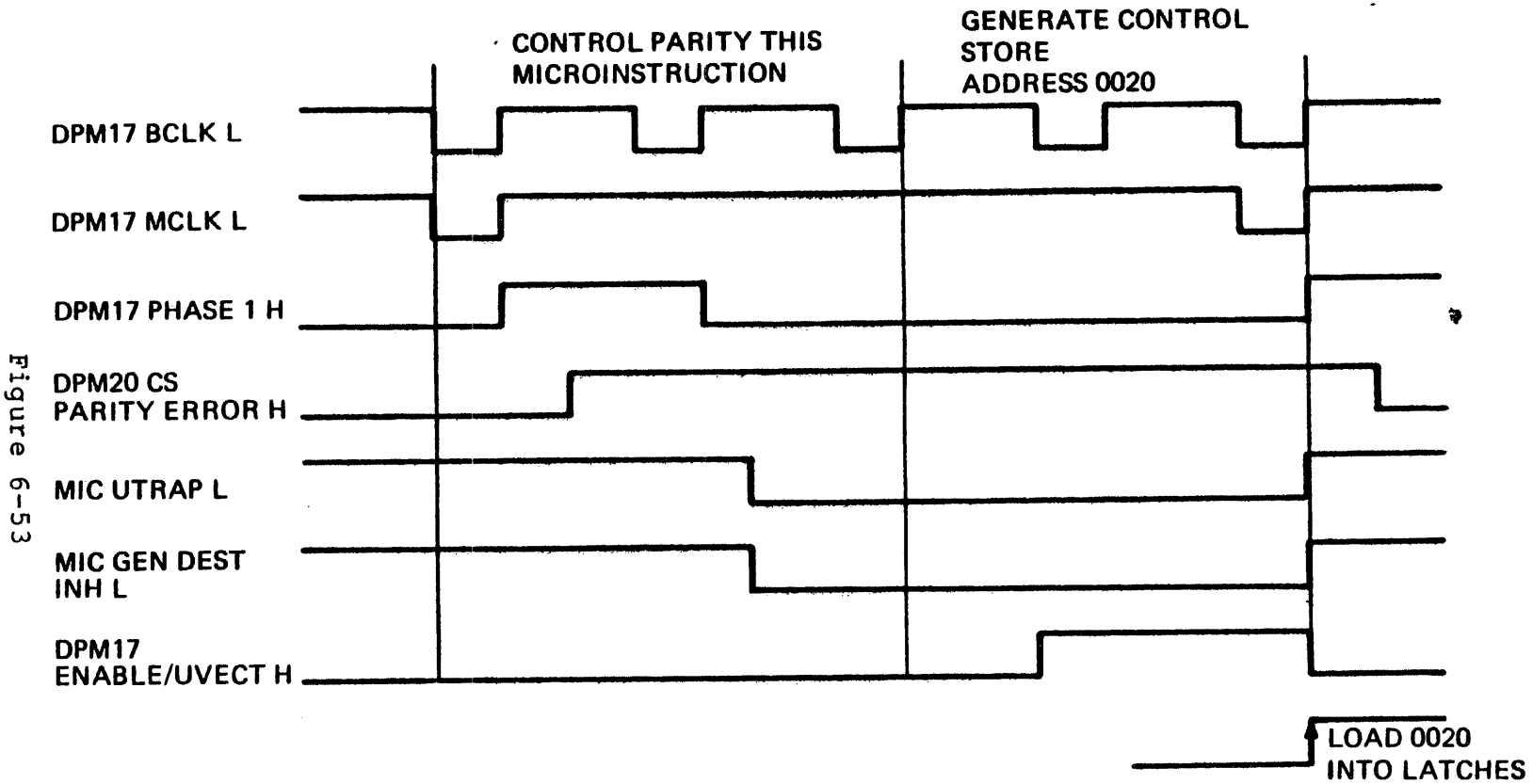

CONTROL PARITY THIS MICROINSTRUCTION
GENERATE CONTROL STORE ADDRESS 0020
DPM17 BCLK L
DPM17 MCLK L
DPM17 PHASE 1 H
DPM20 CS PARITY ERROR H
MIC UTRAP L
MIC GEN DEST INH L
DPM17 ENABLE/UVECT H
LOAD 0020 INTO LATCHES
TK-4322

Figure 6-53

VAX-11/750 Level II

Data Path

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The COMET data path is 32 bits wide. The main components are five different types of LSI gate array chips and two arrays of scratch pad registers. Some special features include a rotator capable of multiple bit shifting and variable length bit field extraction; an arithmetic and logic processor (ALP) capable of BCD calculation and hardware controlled multiply and divide; and two sets of scratch pad registers for microcode temporaries.

Figure 2 depicts the block diagram of the data path.

Two tri-state buses are used to interface with other non-data path logic. The two buses are the MBUS and the WBUS.

The MBUS is used to receive data from the scratch pad and the memory interface logic. Data coming from these sources are latched during the first half of the cycle and must be turned off during the last half of the cycle.

The WBUS is used mainly to send the data path results to the various destinations of the CPU. Examples of these destinations are the WDR (Write Data Register), PC, VA, and the scratch pads, etc.

In addition, two other internal data buses are present. These are the RBUS and the SBUS. The RBUS interfaces the ALP with the scratch pad array containing most of the VAX architectural registers. The SBUS sends the rotator output to the ALP.

In general, data from the two arrays of scratch pad can be operated on directly by the rotator and the ALU, with the results written back to the same location in the same cycle.

OBJECTIVES

Identify the various data path entities by answering multiple choice questions. The entities will include:

a) system clock and timing
b) control store
c) register
d) buses
e) super rotator
f) arithmetic and logical section

Utilizing the DPM print set, trace the signal path for a preselected signal in the Comet data path.

Given a faulty processor, isolate the defective data path gate arrays by running the applicable diagnostics.

SAMPLE TEST ITEM

The process of extracting and zero extending an operand is the function of the _____.

a) arithmetic and logical section
b) super rotator
c) control store
d) general registers

RESOURCES

Data Path Specifications
Microcode Listing

OUTLINE

VII. A. 3 Sections of Data Path

1. Scratch pad
2. Rotator
3. Arithmetic & logical

B. Data Path Registers

C. Data Path Buses

D. Scratch Pad Logic

1. RAM R.
2. RAM M.
3. RSRC
4. MSRC
5. RNUM
6. RBS
7. SPA Status
8. Scratch Pad Address Control

E. Long Literal Register

F. Super Rotator Logic

1. The inputs
2. The outputs
3. The SRM chips
4. The SRK chip
5. The ALP chips

G. Multiple Bit Shifting

H. Variable Bit Field Extraction

I. Generate Constants

J. Various Bit Shuffling

K. Rotation of Data

L. ALP Logic Section

1. Inputs & outputs
2. Misc signals & associated circuits

OUTLINE (Continued)

M. ALP Function

   1. Arithmetic
   2. Others

N. ALP Control Function Chart
O. Microcode Example
P. Print Familiarization
Q. Summary

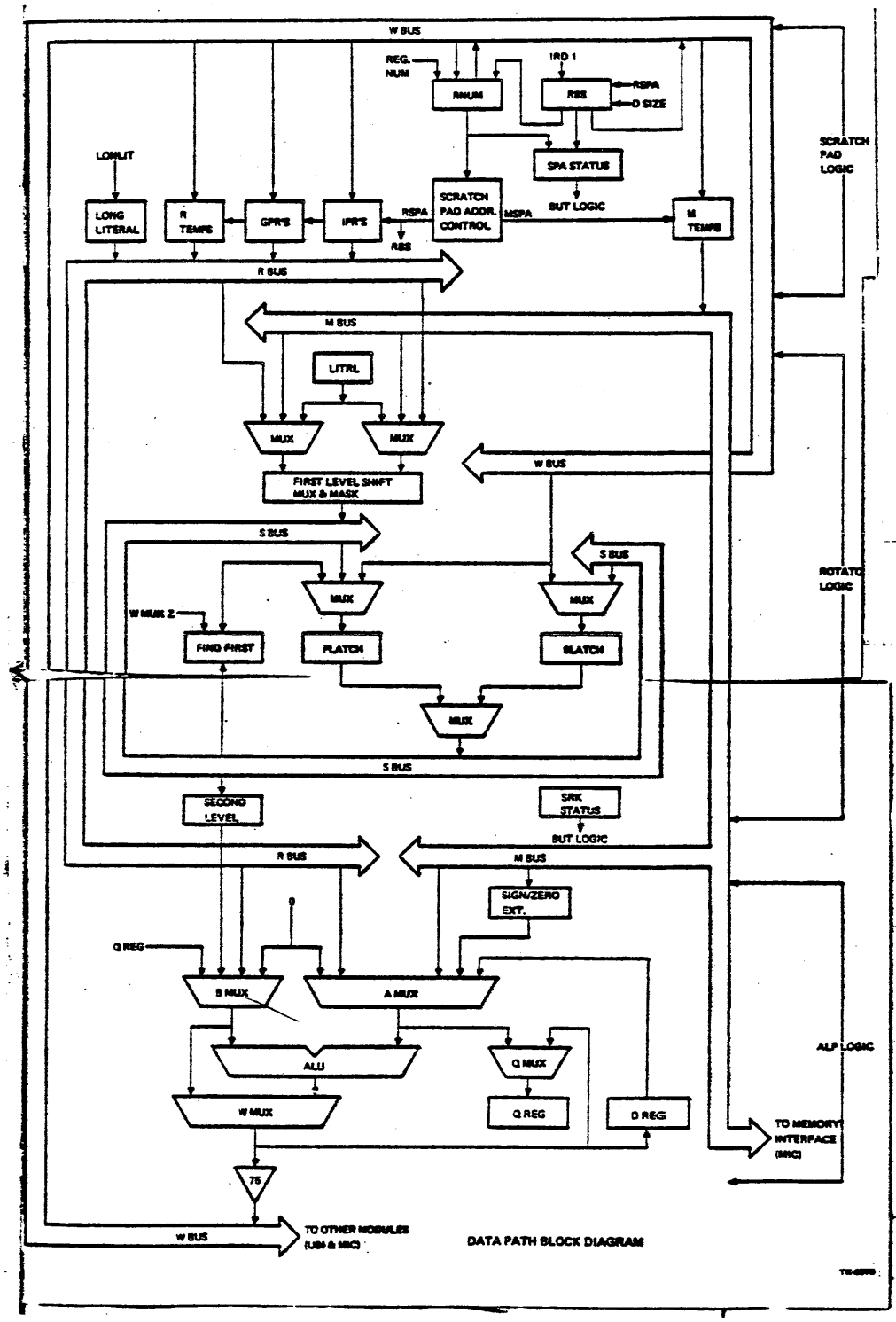

W BUS
REG. NUM
IRD 1
RNUM
RSS
RSPA
D SIZE
SPA STATUS
SCRATCH PAD LOGIC
LONLIT
LONG LITERAL
R TEMPS
GPR'S
IPR'S
RSPA
RSS
SCRATCH PAD ADDR. CONTROL
MSPA
BUT LOGIC
M TEMPS
R BUS
M BUS
LITRL
MUX
MUX
W BUS
FIRST LEVEL SHIFT MUX & MASK
S BUS
S BUS
ROTATE LOGIC
MUX
MUX
W MUX Z
FIND FIRST
PLATCH
SLATCH
MUX
S BUS
SECOND LEVEL
SRK STATUS
BUT LOGIC
R BUS
M BUS
SIGN/ZERO EXT.
Q REG
B MUX
A MUX
ALU
Q MUX
ALP LOGIC
W MUX
Q REG
D REG
TO MEMORY INTERFACE (MIC)
75
W BUS
TO OTHER MODULES (UBI & MIC)
DATA PATH BLOCK DIAGRAM

Figure 7-1

REGISTER	WIDTH	CLOCK
SCRATCH PAD*	32 BITS	D
Q REGISTER	32 BITS	QD
D REGISTER	32 BITS	QD
LONLIT	32 BITS	D
RNUM	4 BITS	D/M
RBSP**	3 BITS	M
PLATCH	6 BITS	D
SLATCH	6 BITS	D

* SCRATCH PAD REGS. ARE DIVIDED INTO TWO SECTIONS. 16M & 48R.

** RBSP POINTS TO A 6 WORD BY 7 BIT STACK CALLED RBS

DATA PATH REGISTERS

TK-3054

Figure 7-2

Scratch Pad Section

The scratch pad section of the data path consists of the Scratch pad register and the scratch pad address logic (SPA chip). Figure 3 illustrates the associated logic.

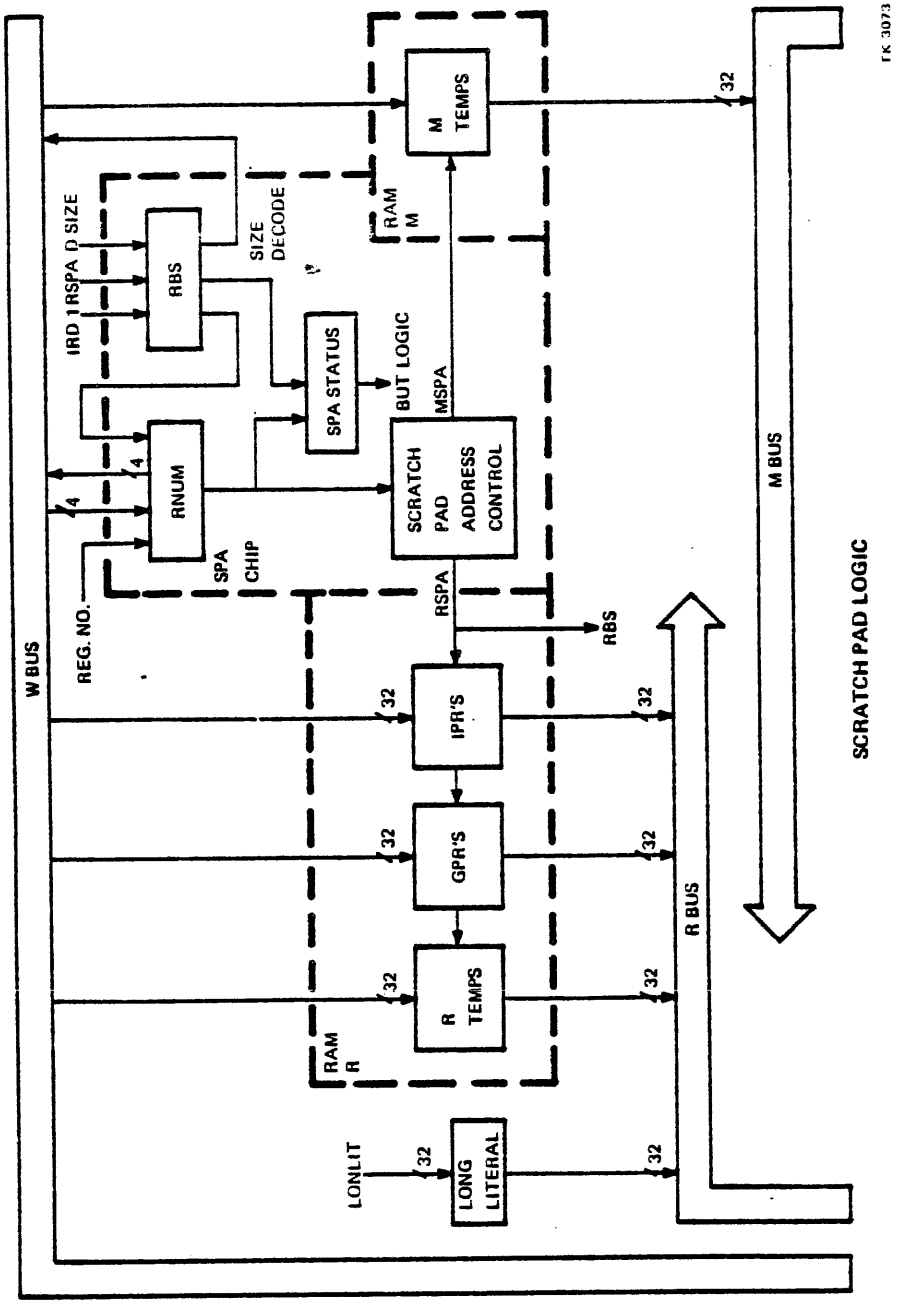

W BUS
REG. NO.
SPA CHIP
RNUM
4
4
IRD 1RSPA 0 SIZE
RBS
SIZE DECODE
SPA STATUS
BUT LOGIC
MSPA
SCRATCH PAD ADDRESS CONTROL
RSPA
RBS
RAM M
M TEMPS
32
M BUS
SCRATCH PAD LOGIC
IPR'S
GPR'S
R TEMPS
RAM R
R BUS
LONLIT
LONG LITERAL

Figure 7-3

REG NO	RSRC	ASSIGNMENT/PURPOSE
0	00	DUAL PORT TEMP 0
1	01	DUAL PORT TEMP 1
2	02	DUAL PORT TEMP 2
3	03	DUAL PORT TEMP 3
4	04	DUAL PORT TEMP 4
5	05	DUAL PORT TEMP 5
6	06	DUAL PORT TEMP 6
7	07	DUAL PORT TEMP 7
8	08	R S-PAD TEMP 8
9	09	R S-PAD TEMP 9
10	0A	R S-PAD TEMP 10
11	0B	R S-PAD TEMP 11
12	0C	R S-PAD TEMP 12
13	0D	R S-PAD TEMP 13
14	0E	MEMORY MANAGEMENT TEMP 5
15	0F	MEMORY MANAGEMENT TEMP 1

R TEMP

Figure 7-4

REG NO	RSRC	ASSIGNMENT/PURPOSE
0	10	GPR 0
1	11	GPR 1
2	12	GPR 2
3	13	GPR 3
4	14	GPR 4
5	15	GPR 5
6	16	GPR 6
7	17	GPR 7
8	18	GPR 8
9	19	GPR 9
10	1A	GPR 10
11	1B	GPR 11
12	1C	GPR 12
13	1D	GPR 13
14	1E	STACK POINTER
15	1F	MICRO CODE TEMPORARY

GPR

TK-3068

Figure 7-5

REG NO	RSRC	ASSIGNMENT/PURPOSE
0	20	KERNEL STACK POINTER
1	21	EXECUTIVE STACK POINTER
2	22	SUPERVISOR STACK POINTER
3	23	USER STACK POINTER
4	24	INTERRUPT STACK POINTER
5	25	PROCESS CONTROL BLOCK BASE
6	26	MEMORY MANAGEMENT TEMP 2
7	27	MEMORY MANAGEMENT TEMP 3
8	28	P0 BASE REGISTER
9	29	P0 LENGTH REGISTER
10	2A	P1 BASE REGISTER
11	2B	P1 LENGTH REGISTER
12	2C	SYSTEM BASE REGISTER
13	2D	SYSTEM LENGTH REGISTER
14	2E	NEXT INTERVAL REGISTER
15	2F	MEMORY MANAGEMENT TEMP 4

IPR

Figure 7-6

REG NO	MSRC	ASSIGNMENT/PURPOSE
0	00	DUAL PORT TEMP 0
1	01	DUAL PORT TEMP 1
2	02	DUAL PORT TEMP 2
3	03	DUAL PORT TEMP 3
4	04	DUAL PORT TEMP 4
5	05	DUAL PORT TEMP 5
6	06	DUAL PORT TEMP 6
7	07	DUAL PORT TEMP 7
8	08	M S-PAD TEMP 8
9	09	M S-PAD TEMP 9
10	0A	M S-PAD TEMP 10
11	0B	ERROR CODE, MEM FAULTS & ARITH TRAP
12	0C	FPD PACK ROUTINE OFFSET
13	0D	MEMORY MANAGEMENT TEMP 0
14	0E	SYSTEM CONTROL BLOCK BASE
15	0F	SOFTWARE INT SUMMARY REGISTER

M TEMP

TK-3069

Figure 7-7

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.TMP2	-
27	MM.TEMP3	-
28	P0BR	-
29	P0LR	-
2A	P1BR	-
2B	P1LR	-
2C	SBR	-
2D	SLR	-
2E	SPNICR.SPICR	-
2F	MM.TEMP4	-

TK-3060

Figure 7-8

RSRC ASSIGNMENTS (CONT)

RSRC <5:0> HEX	RAM-R REGISTER	OPERATION
30	TEMP.R	-
31	DST.R	-
32	IPR.R	-
33	CRP.R	-
34	(TEMP0)	-
35	(TEMP7)	LONLIT
36	(TEMP0)	ZERO
37	(TEMP0)	ZERO.CLRRBSP
38	TEMP.ROR1	-
39	DST.POR1	-
3A	IPR.ROR1	-
3B	GPR.ROR1	-
3C	TEMP.R+1	-
3D	DST.R+1	-
3E	IPR.R+1	-
3F	GPR.R+1	-

TK-3061

Figure 7-8 (Continued)

MSRC ASSIGNMENTS

MSRC<4:0> HEX	RAM-M REGISTER	OPERATION	DESCRIPTION
00-0A	TEM P0-TEMP10	-	MICROCODE TEMPORARIES
08	ERRCOD	-	ERROR CODE
0C	F PDOFFSET	-	FPD PACK ROUTINE OFFSET
0D	MM.TEMP0	-	MEMORY MANAGEMENT TEMP
0E	SCBB	-	SYSTEM CONTROL BLOCK BASE
0F	SISR	-	SOFTWARE INT SUMMARY
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 <-- PNUM
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

TK-3079

Figure 7-9

LOAD SIGNAL ASSERTED	MSRC	OPERATION SPECIFIED	RNUM CONTENTS
YES	XXXXX	MICROSEQUENCER	REG. FIELD SPECIFIED BY MICROCODE
NO	11101	RNUM ← WBUS	WBUS<3:0>
NO	11100	POP RBS	REG FIELD OF RBS

RNUM LOADING CHART

TK-3050

Figure 7-10

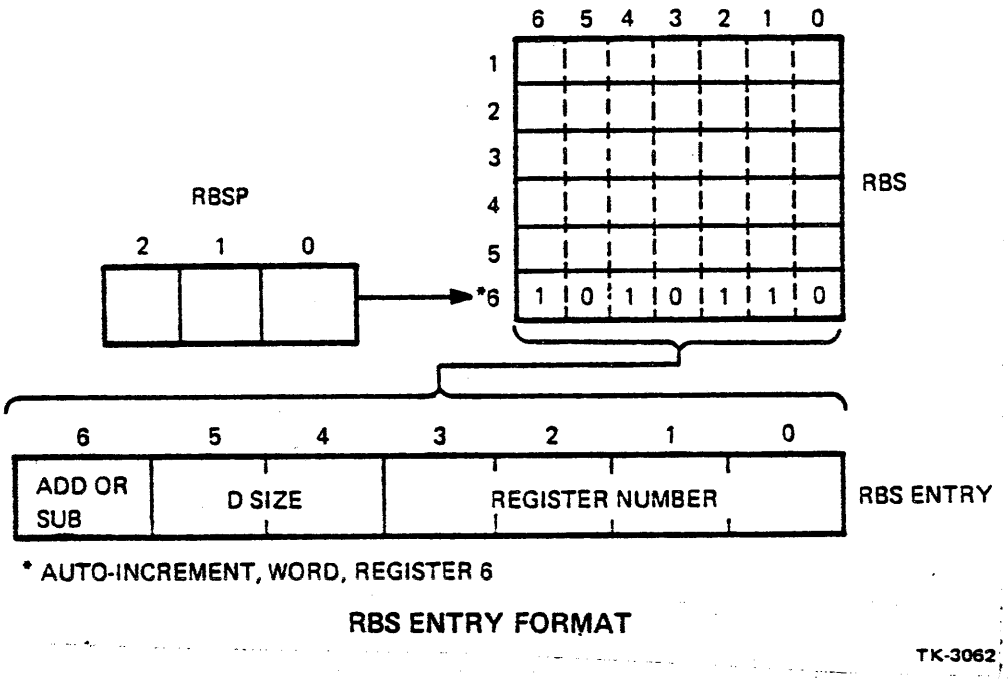

6 5 4 3 2 1 0
1
2
3
4
5
*6
1 0 1 0 1 1 0
RBS
RBSP
2 1 0
6 5 4 3 2 1 0
ADD OR SUB
D SIZE
REGISTER NUMBER
RBS ENTRY
* AUTO-INCREMENT, WORD, REGISTER 6
RBS ENTRY FORMAT
TK-3062

Figure 7-11

D SIZE	DATA TYPE	ENCODED VALUE ON SBUS
00	BYTE	0001
01	WORD	0010
10	LONG	0100
11	QUAD	1000

ADD OR SUB	OPERATION
0	SUBTRACT (DEC)
1	ADD (INC)

REGISTER NO.	0 THRU F

RBS ENTRY FIELDS

TK-3063

Figure 7-12

IF RSRC OR RNUM SPECIFIES GPR

SPASTA 1:0	RNUM REG CONTENT	GPR USE
0 1	1 1 1 0	VAX MODE SP
1 0	0 1 1 1	COMP. MODE PC
1 1	0 1 1 0	COMP. MODE SP
0 0	ALL OTHER VAL.	X

IF MSRC SPECIFIES RNUM ← WBUS & RSRC IS NOT GPR

SPASTA 1:0	WBUS 3:0	IPR USE
1 1	0 - 4	PROCESSOR CONTROL SP'S
1 0	5-7, E, F	RESERVED
0 0	8 - D	ALL OTHERS
0 1	X	X

IF MSRC SPECIFIES A POP RBS & RSRC IS NOT GPR

SPASTA 1:0	RBS BIT 6	INDICATED MODE
0 0	0	AUTO-INC
1 0	1	AUTO-DEC

SPA STATUS

TK-3051

Figure 7-13

IF MSRC SPECIFIES WBUS ← RBSP & RSRC IS NOT GPR

SPASTA 1:0	R B S P	RBS CONDITION
0 1	0	EMPTY
0 0	ALL OTHER VAL.	NOT EMPTY

IF RSRC OR RNUM IS NOT GPR, STATUS IS DEFINED FOR THE FOLLOWING ONLY.

MSRC 68:64	OPERATION SPECIFIED
1 1 1 0 0	POP RBS
1 1 1 0 1	RNUM ← WBUS
1 1 1 1 0	WBUS ← RBSP

SPA STATUS

TK-3064

Figure 7-14

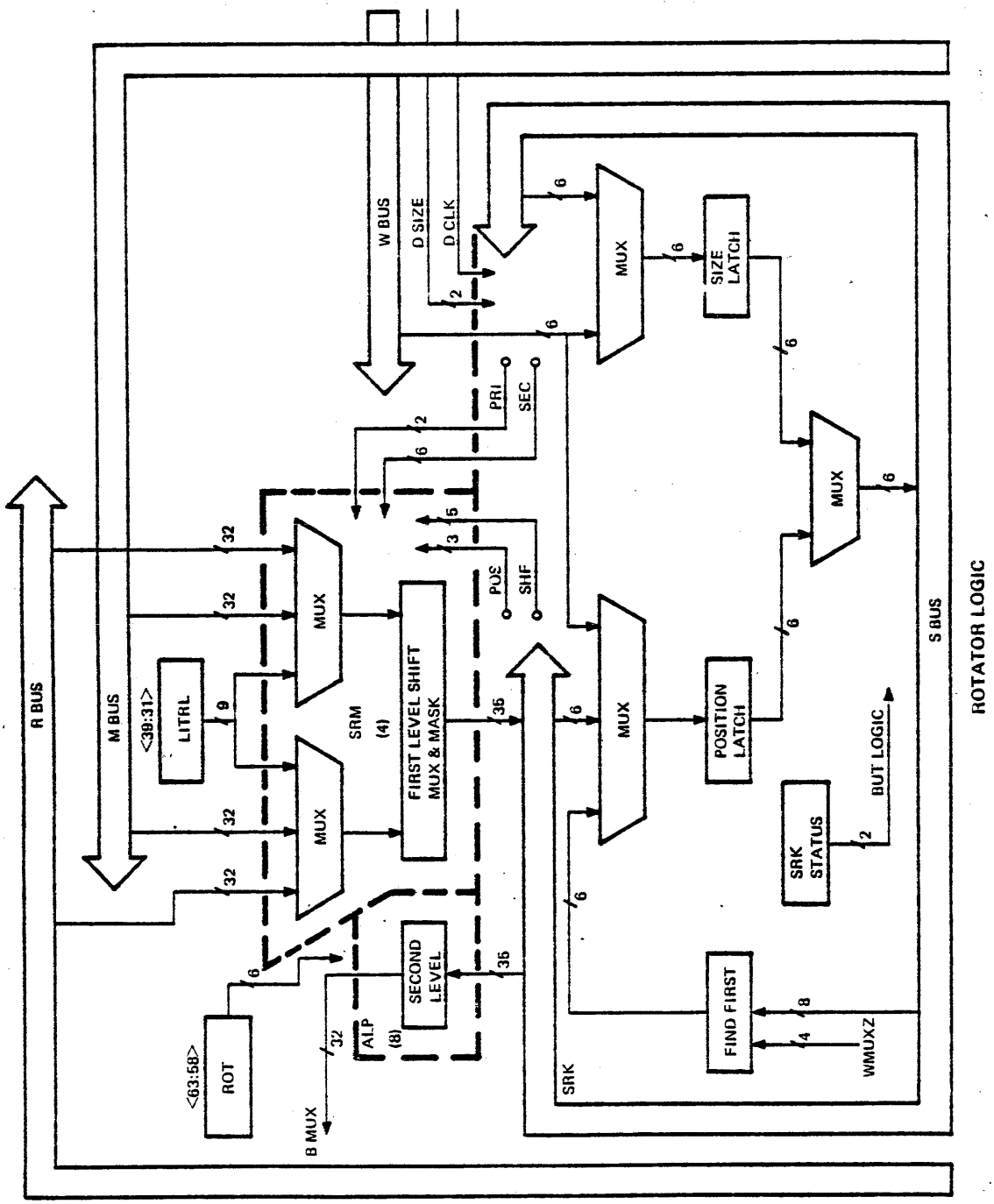

R BUS
M BUS
W BUS
D SIZE
D CLK
S BUS
ROTATOR LOGIC
MUX
SIZE LATCH
PRI
SEC
PUS
SHF
LITRL
<30:31>
SRM
(4)
FIRST LEVEL SHIFT MUX & MASK
SECOND LEVEL
ALP
(8)
ROT
<63:58>
B MUX
POSITION LATCH
BUT LOGIC
SRK STATUS
FIND FIRST
WMUXZ
SRK

Figure 7-15

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, these are 9-bit input data directly from the following micro-fields: 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 will be 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.

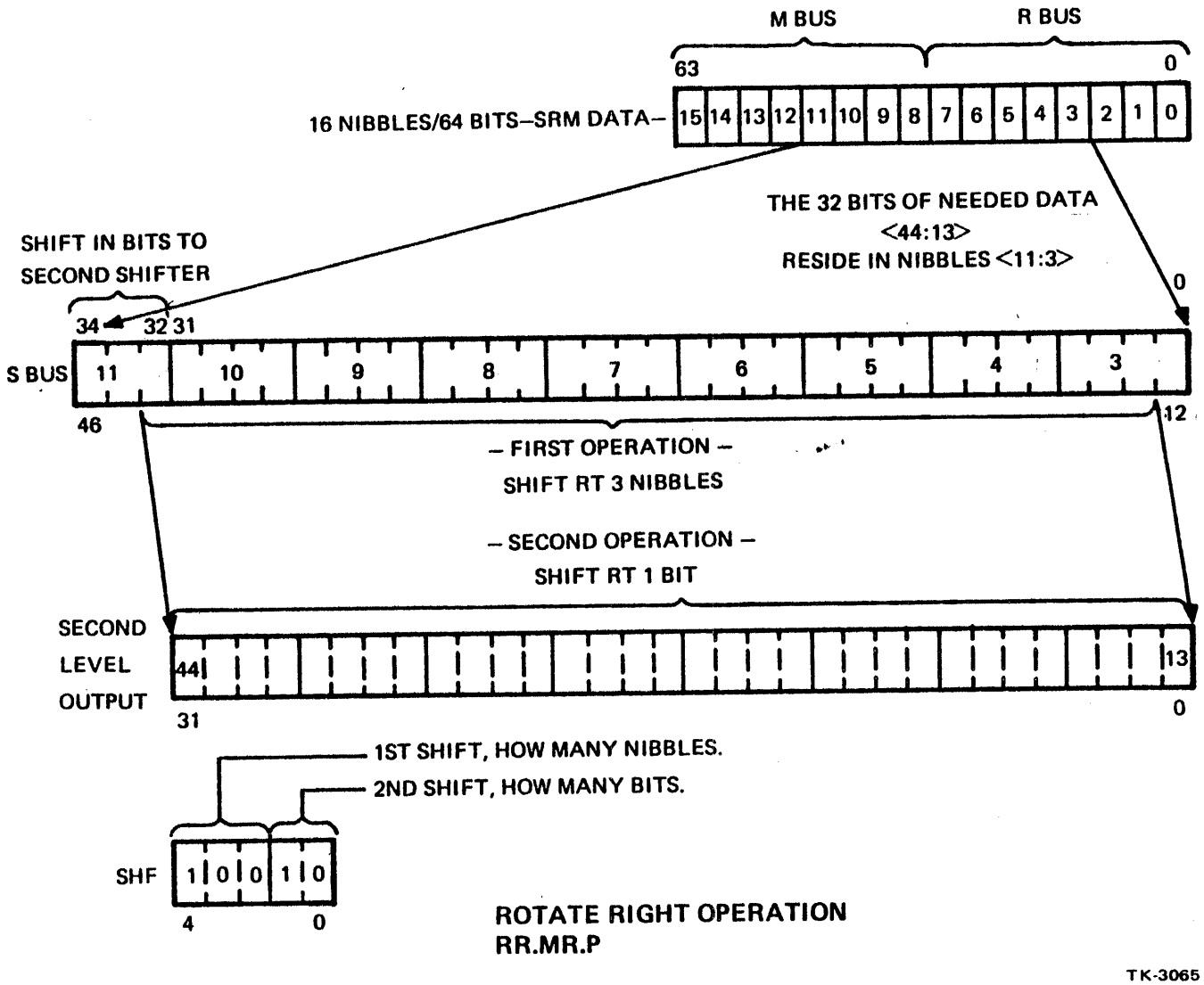
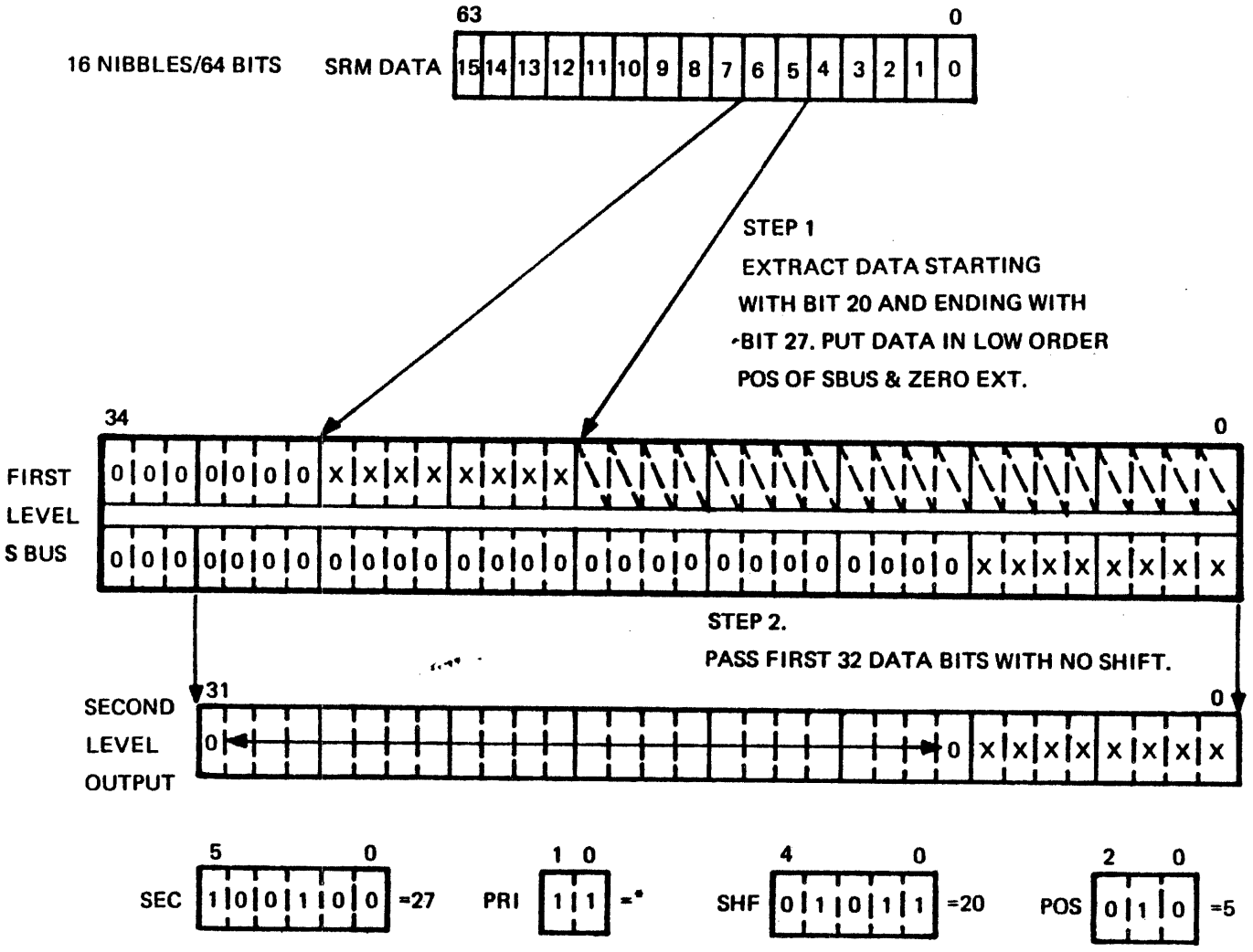

M BUS
R BUS
63
0
16 NIBBLES/64 BITS–SRM DATA–
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
THE 32 BITS OF NEEDED DATA
<44:13>
RESIDE IN NIBBLES <11:3>
SHIFT IN BITS TO
SECOND SHIFTER
34
32 31
0
S BUS
11
10
9
8
7
6
5
4
3
46
12
– FIRST OPERATION –
SHIFT RT 3 NIBBLES
– SECOND OPERATION –
SHIFT RT 1 BIT
SECOND
LEVEL
OUTPUT
44
13
31
0
1ST SHIFT, HOW MANY NIBBLES.
2ND SHIFT, HOW MANY BITS.
SHF
1 0 0 1 0
4
0
ROTATE RIGHT OPERATION
RR.MR.P
TK-3065

Figure 7-16



ROTATOR EXTRACT AND ZERO OPERATION
XZ.MR

TK-3066

Figure 7-17

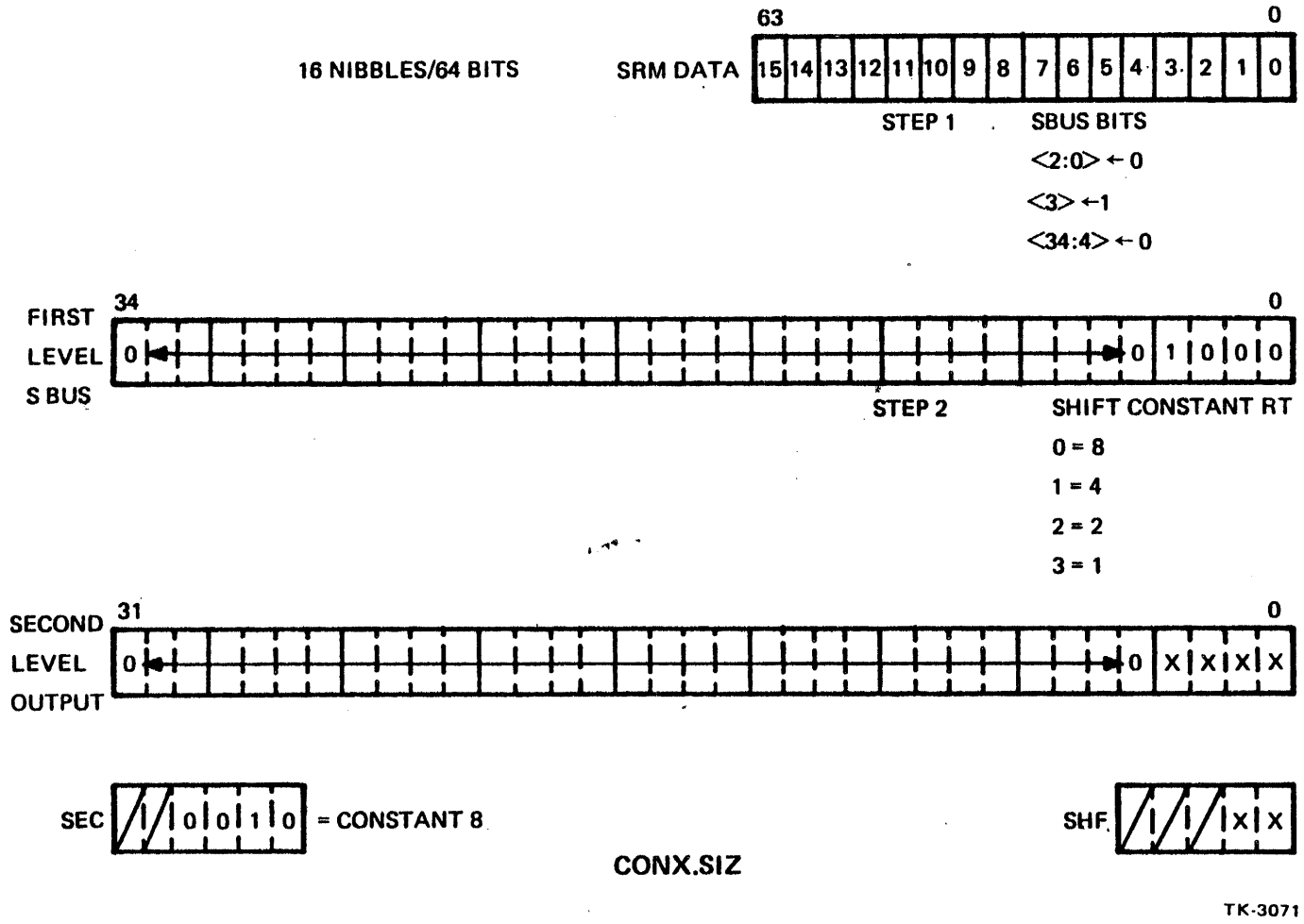

16 NIBBLES/64 BITS
SRM DATA
63
0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
STEP 1
SBUS BITS
<2:0> ← 0
<3> ← 1
<34:4> ← 0
FIRST LEVEL S BUS
34
0
0 1 0 0 0
STEP 2
SHIFT CONSTANT RT
0 = 8
1 = 4
2 = 2
3 = 1
SECOND LEVEL OUTPUT
31
0
0 X X X X
SEC
1 0 0 1 0
= CONSTANT 8
SHF
1 1 X X
CONX.SIZ
TK-3071

Figure 7-18

THIS FUNCTION IS USED TO CONVERT A 4 DIGIT BCD STRING ON THE MBUS TO A 4 DIGIT NUMERIC (ASCII) STRING. TO USE THIS FUNCTION PROPERLY, A CONSTANT XXX3 XX33 (HEX) MUST BE SET UP ON THE RBUS.

ALTERNATE 4 BIT CHUNKS FROM THE MBUS AND THE RBUS ARE SHUFFLED ONTO THE SBUS AS FOLLOWS:

SBUS<03:00> <-- MBUS<07:04>
SBUS<07:04> <-- RBUS<03:00>
SBUS<11:08> <-- MBUS<03:00>
SBUS<15:12> <-- RBUS<07:04>
SBUS<19:16> <-- MBUS<15:12>
SBUS<23:20> <-- RBUS<03:00>
SBUS<27:24> <-- MBUS<11:08>
SBUS<31:28> <-- RBUS<19:16>
SBUS<34:32> <-- 0

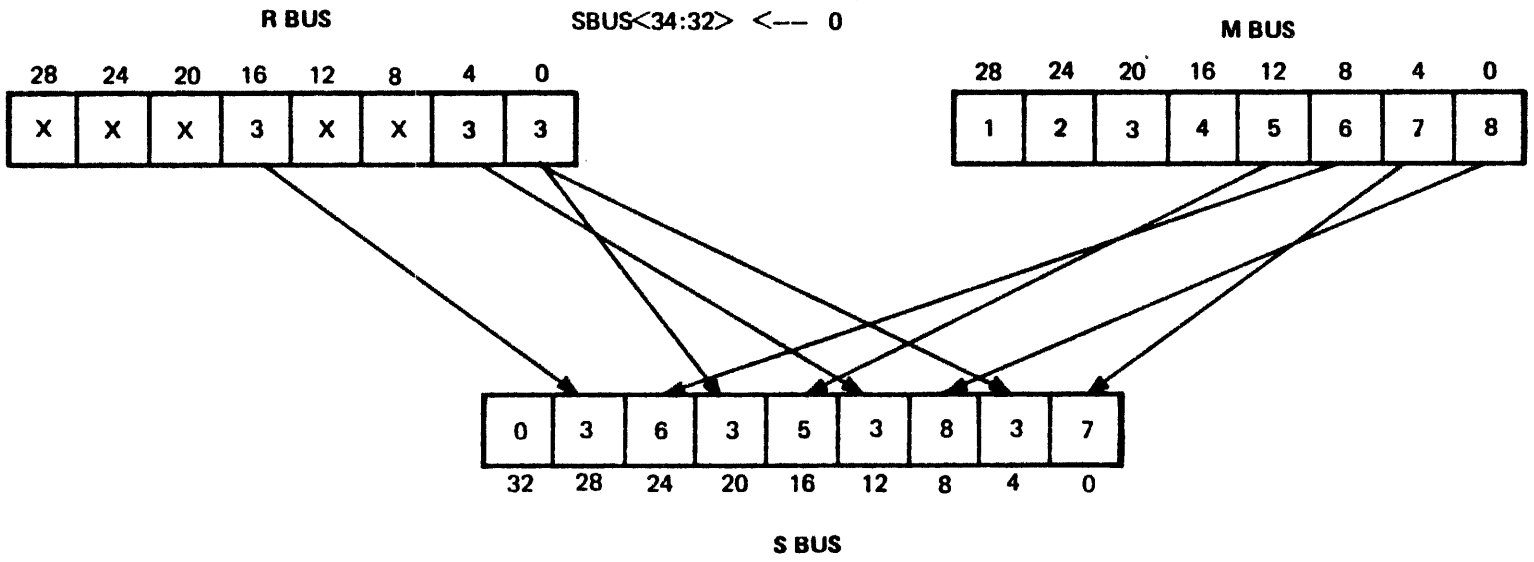
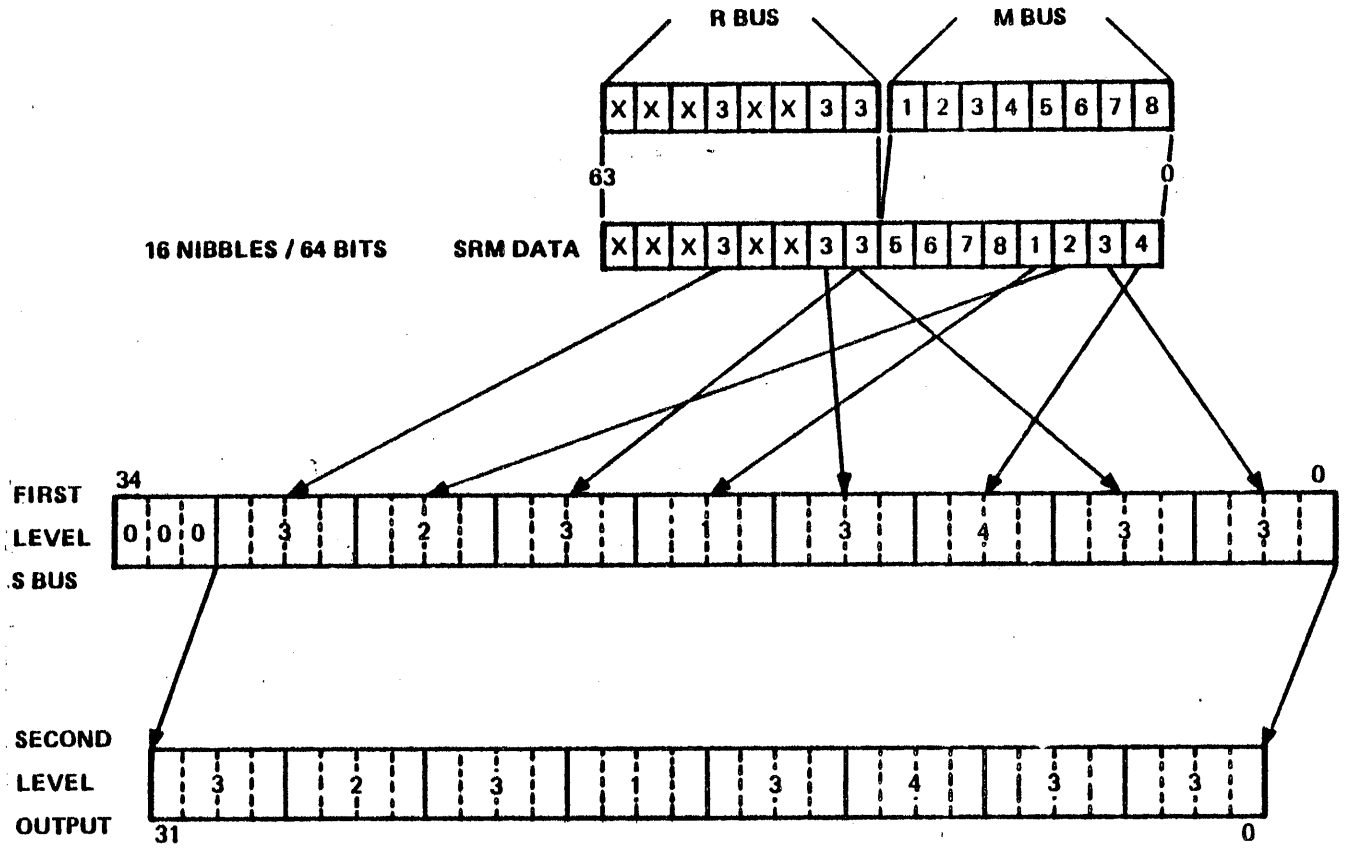


Figure 7-19

CVTPN CONVERSION OF BCD TO ASCII (SHUFFLE)
BCD.SWP

TK-3053



D SIZE <1:0>	BYTES ROTATED
00	1
01	2
10	3
11	0

RR.MM SIZ ROTATE RT M BUS & M BUS BY 2 BYTES FOLLOWED BY CVTPN

TK-3057

Figure 7-20

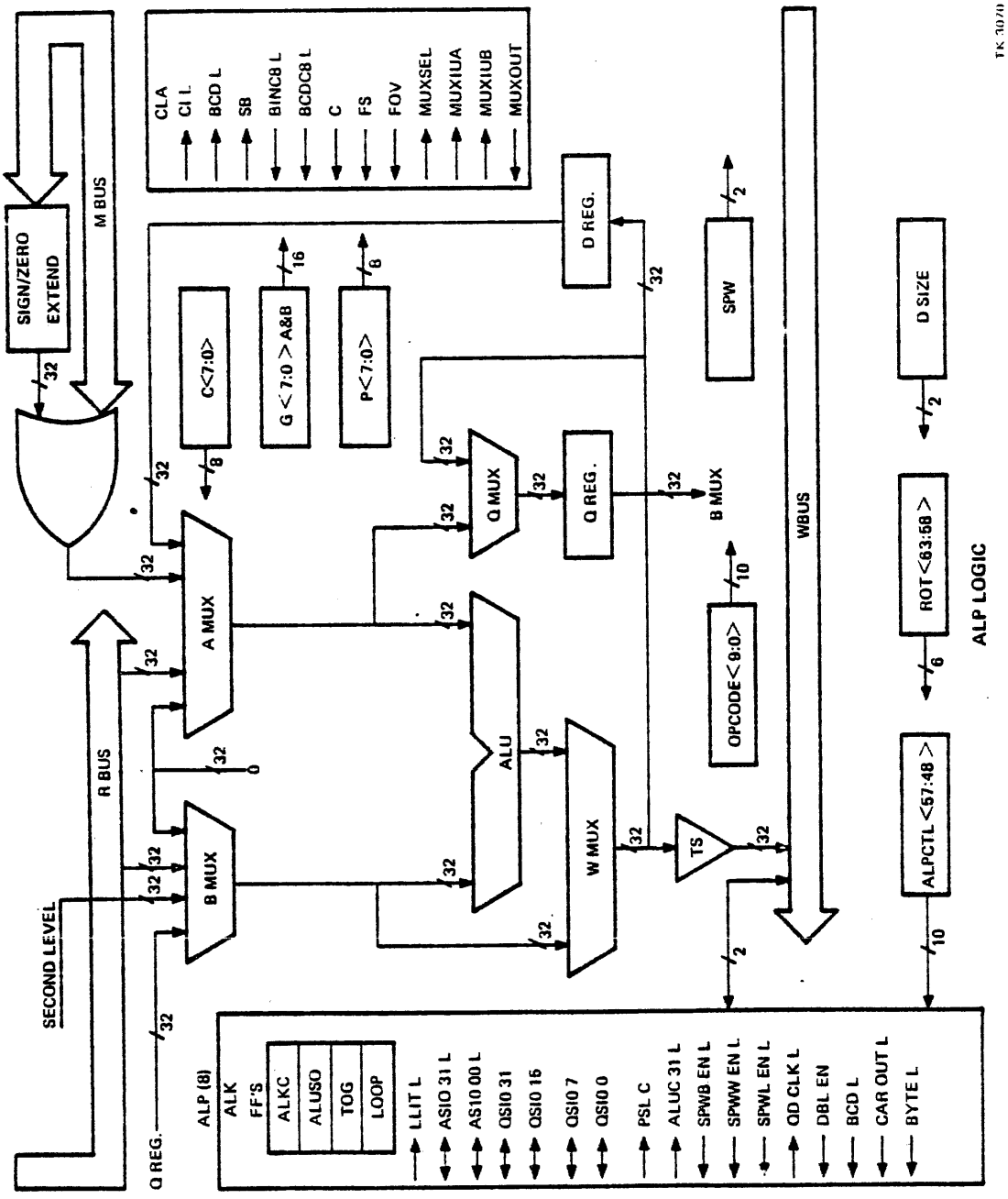

CLA
CI L
BCD L
SB
BINC8 L
BCDC8 L
C
FS
FOV
MUXSEL
MUXIUA
MUXIUB
MUXOUT
SIGN/ZERO EXTEND
M BUS
C<7:0>
G<7:0>A&B
P<7:0>
D REG.
SPW
D SIZE
Q MUX
Q REG.
B MUX
A MUX
ALU
W MUX
TS
WBUS
OPCODE<9:0>
ROT<63:58>
ALPCTL<57:48>
ALP LOGIC
R BUS
SECOND LEVEL
B MUX
Q REG.
ALP (8)
ALK
FF'S
ALKC
ALUSO
TOG
LOOP
LLIT L
ASIO 31 L
ASIO 00 L
OSIO 31
OSIO 16
OSIO 7
OSIO 0
PSL C
ALUC 31 L
SPWB EN L
SPWW EN L
SPWL EN L
QD CLK L
DBL EN
BCD L
CAR OUT L
BYTE L

Figure 7-21

ALP 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, AMUX and BMUX
2. ALU
3. Output mux, WMUX
4. Q Register
5. D Register
6. WBUS control
7. Status logic

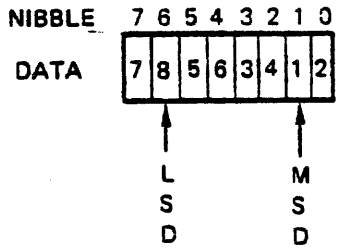
A BCD STRING 1 2, 3 4 5, 6 7 8 WOULD BE STORED IN MEMORY AS FOLLOWS:

12 ——ADDRESS X
34 ——ADDRESS X + 1
56 ——ADDRESS X + 2
78 ——ADDRESS X + 3

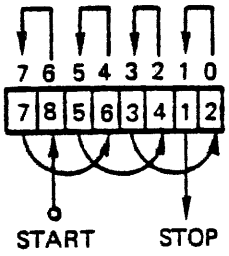
WHEN READ OUT AS A LONGWORD

7	8	5	6	3	4	1	2

APPEARS AT THE DATA PATH



IN ORDER TO PERFORM AN ARITHMETIC FUNCTION ON TWO SUCH STRINGS (ADD), THE CARRY FROM NIBBLE 6 WOULD HAVE TO BE PROPAGATED TO NIBBLE 7, AND NIBBLE 7 PROPAGATED TO NIBBLE 4, AND SO ON.....



CLA

TK-3052

Figure 7-22

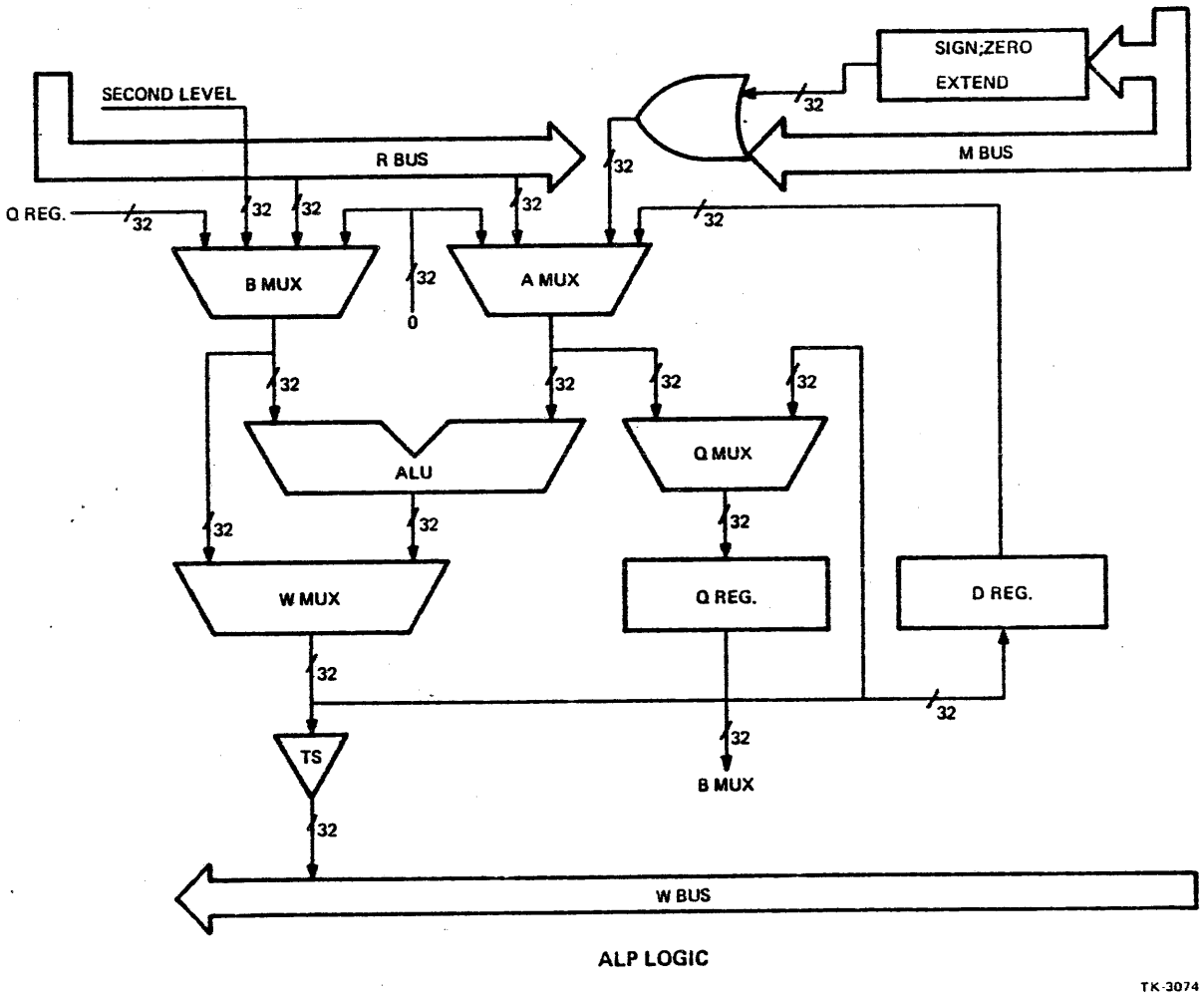
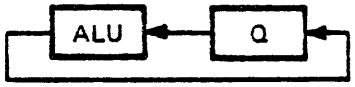
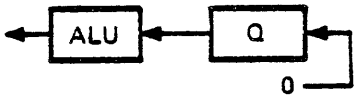
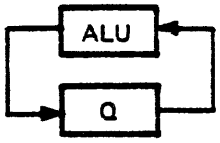
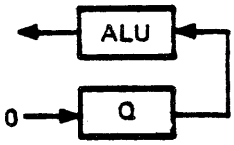
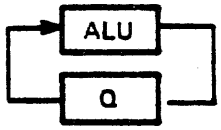
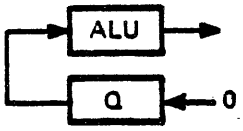
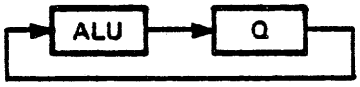
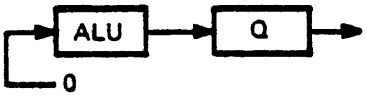
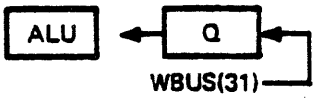
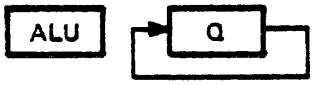
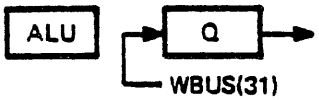
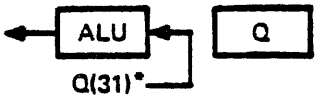
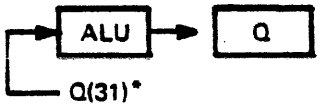

SECOND LEVEL
R BUS
SIGN;ZERO
EXTEND
M BUS
Q REG.
32
B MUX
A MUX
0
ALU
Q MUX
W MUX
Q REG.
D REG.
TS
B MUX
W BUS
ALP LOGIC
TK-3074

Figure 7-23

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.

ALU/Q SHIFT & ROTATE

TK-3059

Figure 7-24

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 WERE DOING BINARY ARITHMETIC, BUT THE P AND G SIGNALS ARE CALCULATED BASED ON 10. EXTRA LOGIC ARE USED TO ADJUST THE 4 BIT ALU OUTPUT TO A TRUE BCD RESULT.

THE FIVE LOGICAL OPERATIONS ARE:

A.AND.B

A.OR.B

A.ANDNOT.B

B.ANDNOT.A

A.XOR.B

ALU ARITHMETIC FUNCTIONS

TK-3055

Figure 7-25

ALU FUNCTIONS

THE ALU CAN PERFORM 16 LOGICAL AND ARITHMETIC OPERATIONS WHICH IS SPECIFIED BY ALPCTL <5:2>.

IN GENERAL, THE 16 ALU OPERATIONS ARE CLASSIFIED INTO THREE GROUPS: BINARY ARITHMETIC, BCD ARITHMETIC AND LOGICAL.

ALPCTL <5:2>	ALU OPERATION	GROUP
0000	A–B–CI	BINARY ARITH
0001	A–B–CI, BCD	BCD ARITH
0010	(A–B–CI).SR	BINARY ARITH
0011	(A–B–CI).SL	BINARY ARITH
0100	A+B+CI	BINARY ARITH
0101	A+B+CI, BCD	BCD ARITH
0110	(A+B+CI).SR	BINARY ARITH
0111	(A+B+CI).SL	BINARY ARITH
1000	A.AND.B	LOGICAL
1001	A.OR.B	LOGICAL
1010	(A.AND.B).SR	LOGICAL
1011	(A.AND.B).SL	LOGICAL
1100	B–A–CI	BINARY ARITH
1101	A.XOR.B	LOGICAL
1110	A.AND.(.NOT.B)	LOGICAL
1111	(.NOT.A) .AND B	LOGICAL

NOTATIONS

A = A MUX

B = B MUX

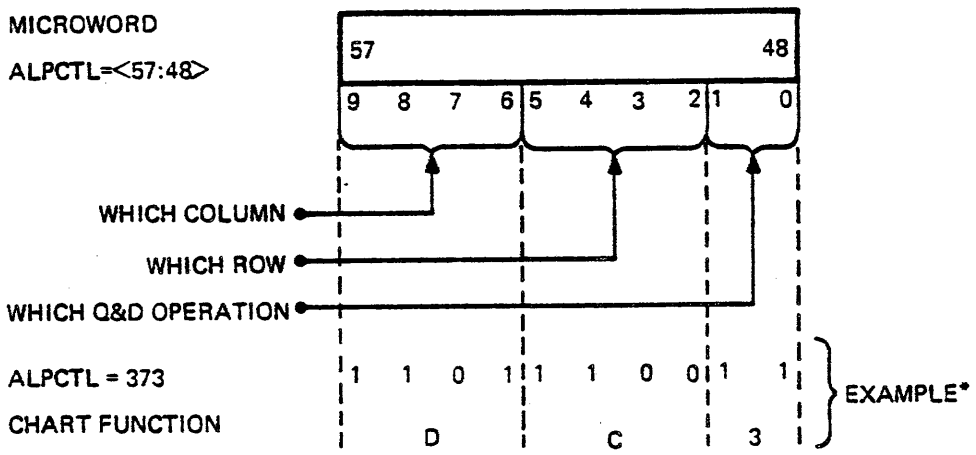
CI = CARRY INPUT

SR = SHIFT RIGHT

SL = SHIFT LEFT

TK-3058

Figure 7-26



COLUMN "D" – AMUX GETS ZERO
BMUX GETS SUPER ROTATOR

ROW "C" BMUX MINUS AMUX MINUS CARRY IN

(SR–0–0)

OPERATION "3" WMUX GETS SUPER ROTATOR
QREG & DREG GETS WMUX

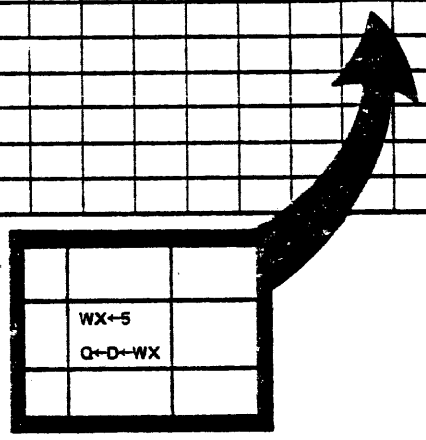
*PASS THE SBUS THROUGH THE ALU

READING THE ALPCTL FUNCTION CHART

TK-3056

Figure 7-27

MICRO ORDER → / AMUX,BMUX / ALU		A	B	C	D	E	F
		D,Q1	D,Q2	D,S	0,S	R,Q	R,S
0	A–B–CI						
1	A–B–CI,BCD						
2	(A–B–CI).SR						
3	(A–B–CI).SL						
4	A+B+CI						
5	A+B+CI,BCD						
6	(A+B+CI).SR						
7	(A+B+CI).SL						
8	A.AND.B						
9	A.OR.B						
A	(A.AND.B).SR						
B	(A.AND.B).SL						
C	B–A–CI						
D	A.XOR.B						
E	A.AND.(.NOT.B)						
F	(.NOT.A).AND.B						



ALPCTL FUNCTION CHART COLUMNS A-F

TK-3067

Figure 7-28

ALPCTL

This is a 10-bit field used by the data path to control the ALP logic. The 10-bit field specifies 1024 functions. Most of them can be grouped together based on (1) the ALU operation, (2) the inputs to the ALU, and (3) the Q and D registers control. Such grouping of the ALP functions is depicted in Figure 28, the ALPCTL FUNCTION CHART.

In the ALP FUNCTION CHART, there are 16 major columns. Each column is identified by ALPCTL<9:6>, which in general specified the inputs to the ALU. There are also 16 major rows. Each row is identified by ALPCTL<5:2>, which in general specifies the ALU operation. At the intersection of a major column and a major row, there are four blocks which are further identified by ALPCTL<1:0>. Each block specifies an operation on the Q and D registers with the given ALU operation and the ALU inputs.

Functions that cannot be readily specified by the above scheme are called ALP special functions. All these functions are marked off with a shaded corner in the ALPCTL FUNCTION CHART.

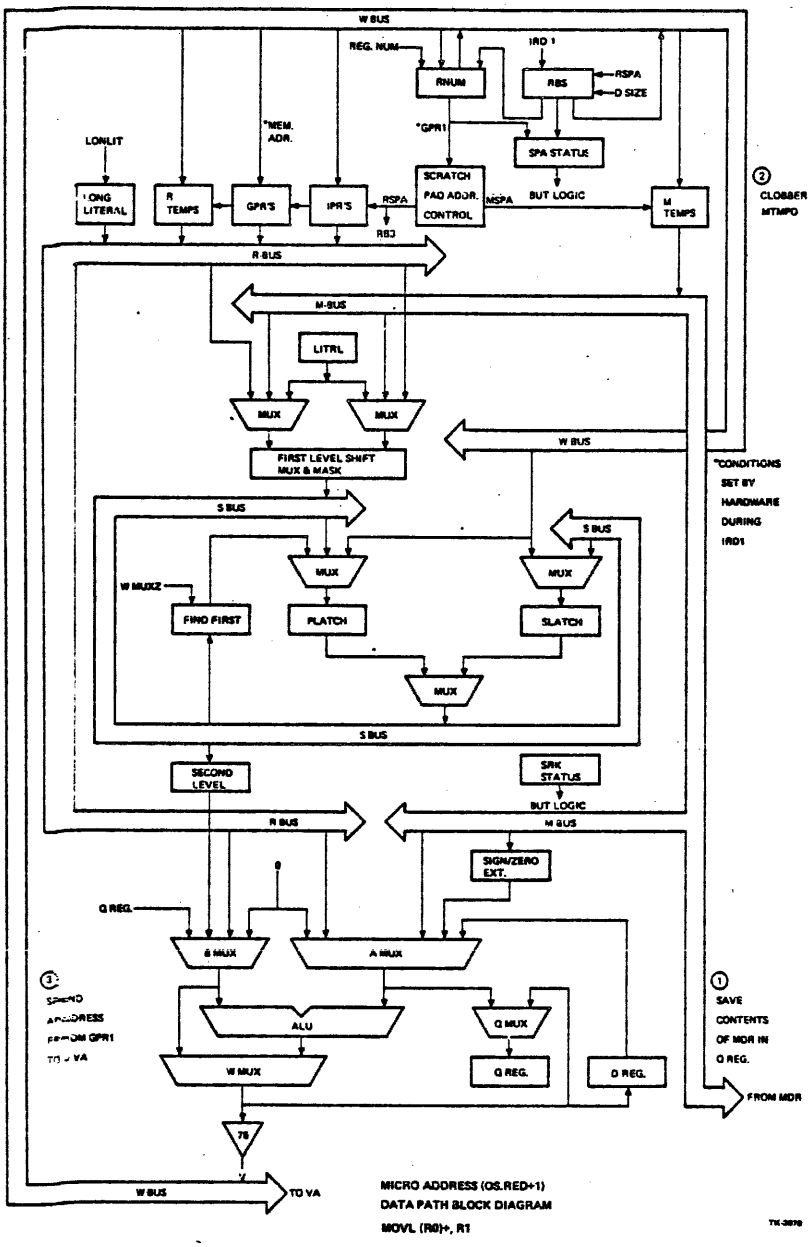

W BUS
REG. NUM
IRD 1
RNUM
RBS
RSPA
D SIZE
*MEM. ADR.
*GPR1
SPA STATUS
LONLIT
LONG LITERAL
R TEMPS
GPR'S
IPR'S
RSPA
RBD
SCRATCH PAD ADDR. CONTROL
MSPA
BUT LOGIC
M TEMPS
CLOBBER MTMPO
R BUS
M BUS
LITRL
MUX
MUX
W BUS
FIRST LEVEL SHIFT MUX & MASK
*CONDITIONS SET BY HARDWARE DURING IRD1
S BUS
S BUS
MUX
MUX
W MUX2
FIND FIRST
PLATCH
SLATCH
MUX
S BUS
SECOND LEVEL
SRK STATUS
BUT LOGIC
R BUS
M BUS
SIGN/ZERO EXT.
Q REG.
B MUX
A MUX
SPEND ADDRESS FROM GPR1 TO VA
ALU
Q MUX
W MUX
Q REG.
D REG.
SAVE CONTENTS OF MDR IN Q REG.
FROM MDR
76
W BUS
TO VA
MICRO ADDRESS (OS.RED+1)
DATA PATH BLOCK DIAGRAM
MOVL (R0)+, R1

Figure 7-29

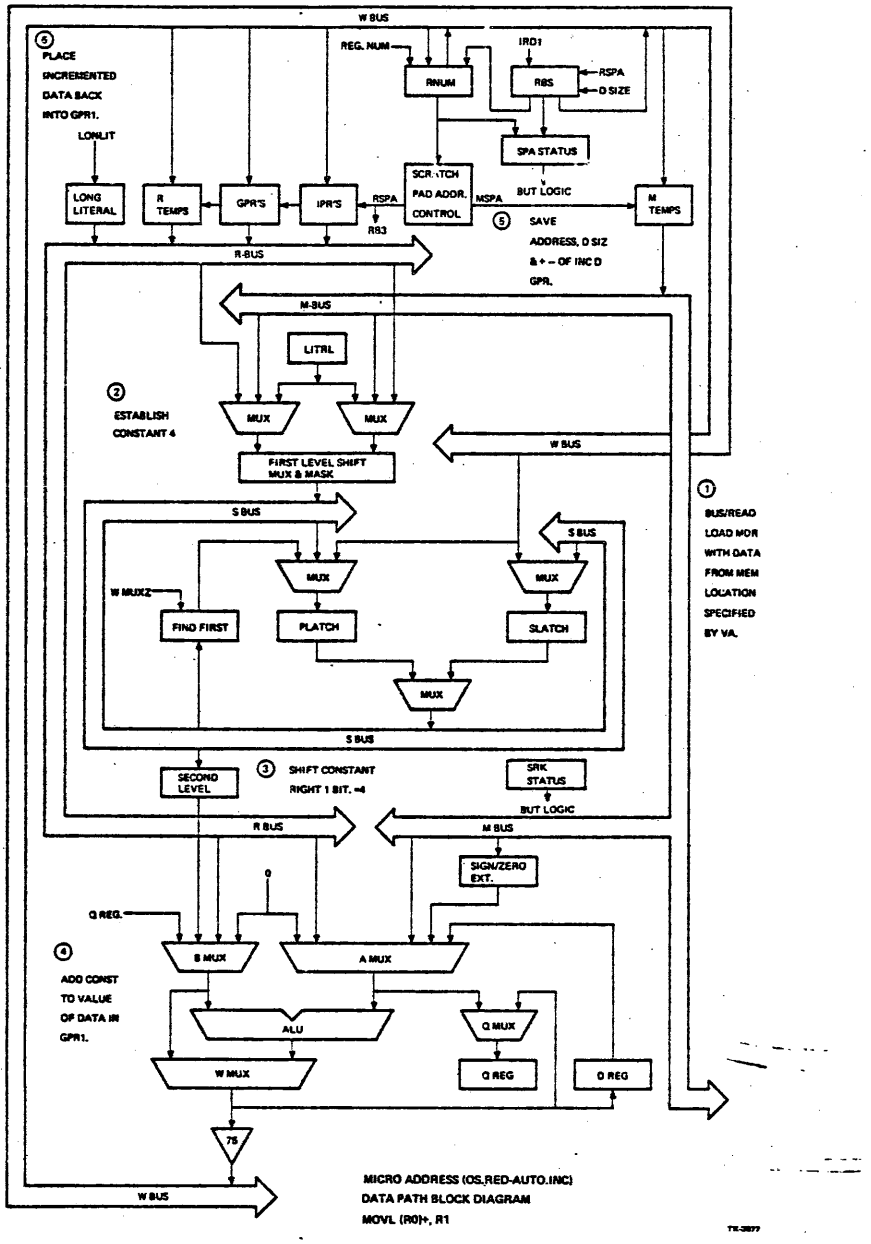

W BUS
REG. NUM
IRD1
RNUM
RBS
RSPA
D SIZE
SPA STATUS
SCRATCH PAD ADDR. CONTROL
MSPA
BUT LOGIC
LONG LITERAL
R TEMPS
GPR'S
IPR'S
RSPA
RB3
M TEMPS
R-BUS
M-BUS
LITRL
MUX
FIRST LEVEL SHIFT MUX & MASK
S BUS
W MUXZ
FIND FIRST
PLATCH
SLATCH
SECOND LEVEL
SHIFT CONSTANT RIGHT 1 BIT. =4
SRK STATUS
R BUS
M BUS
SIGN/ZERO EXT.
Q REG.
B MUX
A MUX
ALU
Q MUX
W MUX
Q REG
D REG
75
1 BUS/READ LOAD MDR WITH DATA FROM MEM LOCATION SPECIFIED BY VA.
2 ESTABLISH CONSTANT 4
3 SHIFT CONSTANT RIGHT 1 BIT. =4
4 ADD CONST TO VALUE OF DATA IN GPR1.
5 SAVE ADDRESS, D SIZ & +/- OF INC D GPR.
6 PLACE INCREMENTED DATA BACK INTO GPR1.
LONLIT
MICRO ADDRESS (OS,RED-AUTO.INC)
DATA PATH BLOCK DIAGRAM
MOVL (R0)+, R1

Figure 7-30

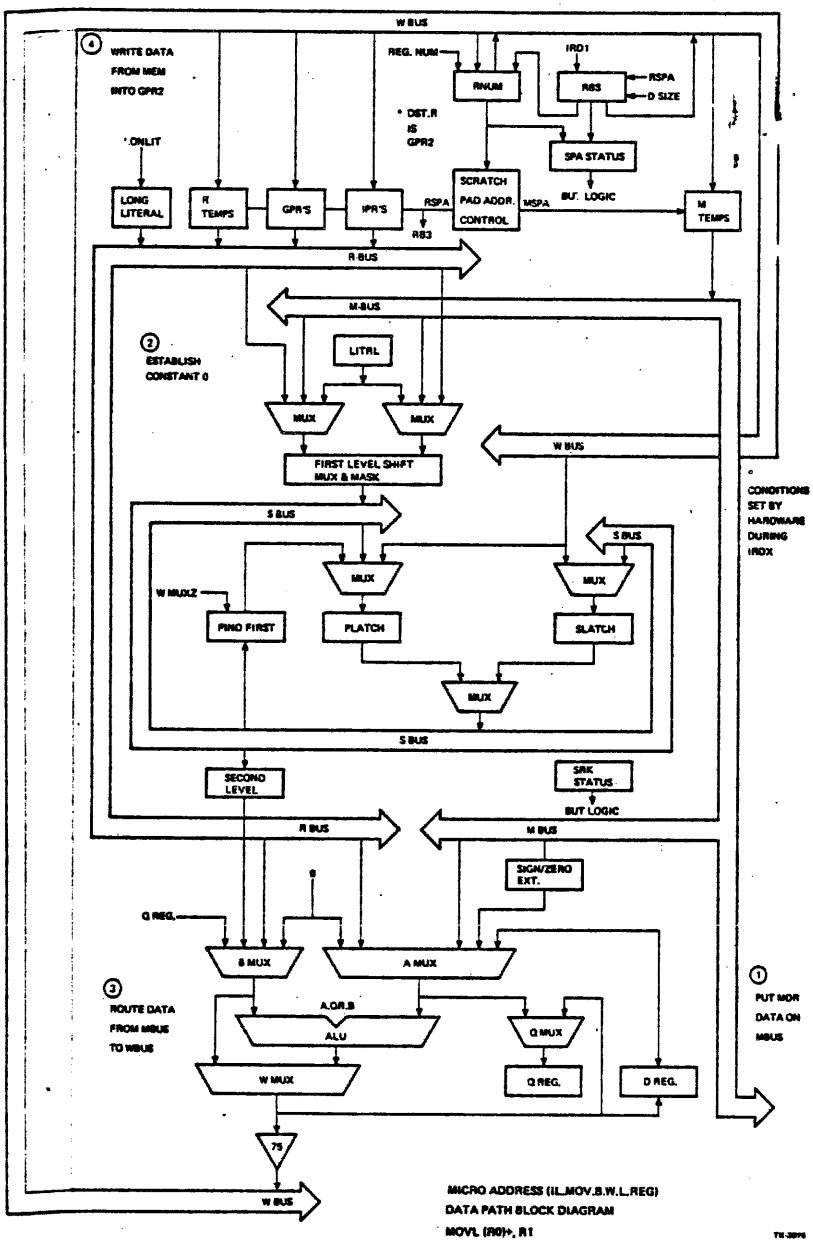

W BUS
WRITE DATA FROM MEM INTO GPR2
REG. NUM
IRD1
RNUM
RBS
RSPA
D SIZE
* DST.R IS GPR2
* ONLIT
SPA STATUS
LONG LITERAL
R TEMPS
GPR'S
IPR'S
RSPA
RB3
SCRATCH PAD ADDR. CONTROL
MSPA
BUF. LOGIC
M TEMPS
R BUS
M BUS
ESTABLISH CONSTANT 0
LITRL
MUX
MUX
W BUS
FIRST LEVEL SHIFT MUX & MASK
CONDITIONS SET BY HARDWARE DURING IRDX
S BUS
S BUS
MUX
MUX
W MUXLZ
PINO FIRST
PLATCH
SLATCH
MUX
S BUS
SECOND LEVEL
SRK STATUS
BUF LOGIC
R BUS
M BUS
SIGN/ZERO EXT.
Q REG.
B MUX
A MUX
ROUTE DATA FROM MBUS TO MBUS
A,OR,B
ALU
Q MUX
W MUX
Q REG.
D REG.
PUT MDR DATA ON MBUS
75
W BUS
MICRO ADDRESS (IL,MOV,B,W,L,REG)
DATA PATH BLOCK DIAGRAM
MOVL (R0)+, R1

Figure 7-31

CHAPTER 7 INTERVAL TIMER AND TIME OF YEAR CLOCK

7.1 INTRODUCTION TO INTERVAL TIMER

The Interval Timer is an integral part of the Comet CPU hardware and it is used primarily to schedule events and control the amount of time a particular task can operate. The operation of the Comet Interval Timer from the software level is consistent with other VAX family 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 microsecond intervals which makes the operation consistent with other VAX family Timers. The maximum interval then could be expressed as (((2**32)-1)*.000001)/60 which works out to be around 71 hours or approximately 3 days. It does require external dedicated scratchpads to maintain the interval count, so the TOK gate array was placed on the DPM module. The Interval Timer is accessible to the VAX-11 macro code through Internal Processor Registers (IPRs). These IPRs can be accessed with MTPR and MFPR macro instructions, and also from the console terminal. An explantion of the Internal Processor Registers will follow in subsequent paragraphs. The Interval Timer operation is basically straightforward. 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 and when the Timer overflows at the end of the desired interval, a macro level interrupt request is booked with the CPU. If the IPL level of the Timer Interrupt Request (IPL 18) is greater than the current PSL IPL, the timer service macro routine is entered via SCBB+C0. This would terminate the current task, if something else of higher priority had not done so.

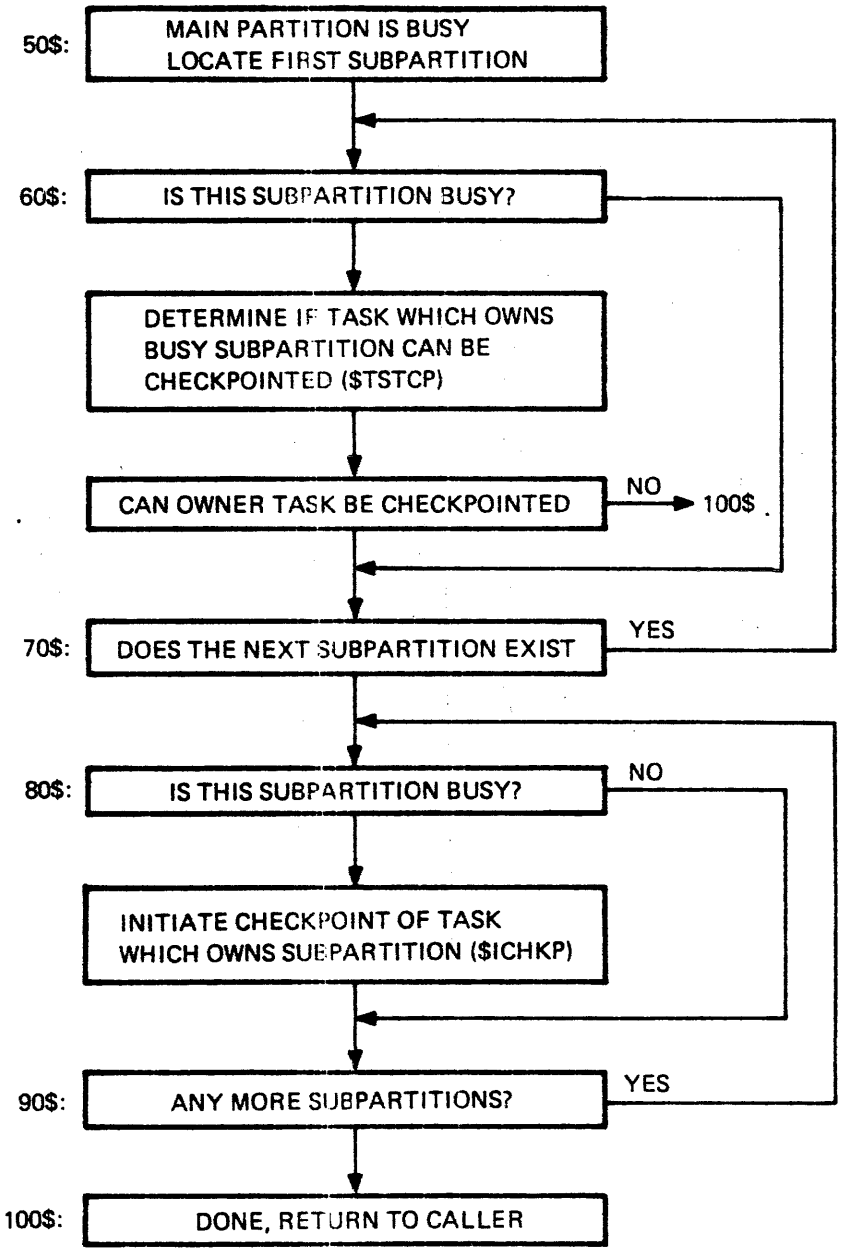
7.2 DETAILED DESCRIPTION OF THE TIMER CIRCUITRY

For the following discussion you will need the module schematic diagram of the DPM and CCS module. Refer to the CCS module schematic page CCS14 and locate E5. E5 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 the 7490 IC which is a decade divider. The output of E4 is a symmetrical 1 MHz signal that provides the increment interval of 1 microsecond. The signal TOK OSC OUT H is wired from slot 5 (CCS module) to slot 2 (DPM module). Refer to the DPM module schematic page DPM13. The TOK gate array is shown in the lower left corner. 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 will clear any interrupt requests left in the gate array and set the logic to a known state. BCLK 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 macro instructions and the interval timer service microroutines. There is a full 32 bit bi-directional 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 micro-routine 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 that the interrupt request can be arbitrated among the other requests. This concludes the detailed circuit description of the Timer circuitry. Timer functionality is verified with the Hardcore instruction test EVKAA. a failure of the timer can be isolated to one of three components, the oscillator, the decade divider, or the TOK gate array.

7.3 INTERVAL TIMER FIRMWARE REQUIREMENTS

The implementation of the Interval Timer in the Comet CPU is not at first obvious. Figure 7-32 shows the VAX 11 Interval Timer IPRs as they appear to the software. There are 3 registers associated with the Interval Timer. IPR 19 is the Next Interval Count Register (NICR) and this register is loaded with the 2's complement of the desired interval. The number loaded into this register is the two's complement of the desired interval in seconds divided by 1 microsecond. The IPR 1A is the Interval Count Register (ICR) and 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. This register controls the operation of the Interval Timer. The function of the bits in the ICCS is explained below.


50$:
MAIN PARTITION IS BUSY
LOCATE FIRST SUBPARTITION
60$:
IS THIS SUBPARTITION BUSY?
DETERMINE IF TASK WHICH OWNS
BUSY SUBPARTITION CAN BE
CHECKPOINTED ($TSTCP)
CAN OWNER TASK BE CHECKPOINTED
NO
100$
70$:
DOES THE NEXT SUBPARTITION EXIST
YES
80$:
IS THIS SUBPARTITION BUSY?
NO
INITIATE CHECKPOINT OF TASK
WHICH OWNS SUBPARTITION ($ICHKP)
90$:
ANY MORE SUBPARTITIONS?
YES
100$:
DONE, RETURN TO CALLER

TK-1724

ICCS BIT	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, This bit must be set by the VAX 11 macro code to enable Timer interrupt requests at IPL 18.
<5>	SC	This is a write only bit that the macro programmer can use to step the interval clock 1 count at a time. Each write to the ICCS with bit <5>=1 will step the interval timer 1 count.
<4>	TR	Transfer moves the NICR contents to the ICR.
<Ø>	RUN	This bit starts the interval counter incrementing until it overflows. This bit would be set after the transferring the NICR to the ICR.

Figure 7-33 shows how the hardware is implemented. The TOK gate array does not contain all the circuitry, as stated earlier, to make the timer function. The first register in Figure 7-33 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 as either bits <15:Ø> of the NICR or as <15:Ø> of the ICR depending on which is desired. The high half of both the NICR and ICR are maintained in an RTEMP scratchpad that is 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 micro-vector to control store address ØØ14. At location ØØ14 is the micro-service routine that will update the scratchpad portion of the ICR. The RTEMP scratchpad that contains the

high half of the ICR is a single 32 bit location that is called R[SPNICR.SPICR]. The scratch pad 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:Ø> of R[SPNICR.SPICR]. Figure 7-33 shows how this is laid out. The timer service microcode has to access the scratchpad by rotating the contents properly. As you can see the NICR IPR is scratchpad memory in bits <31:16> and <15:Ø> actually live in the TOK gate array. The same is true about the ICR. The ICCS shown in the bottom register interfaces to the TOK gate array bits <31:16>. The MTPR and MFPR instructions have to rotate the write and read data to the ICCS 16 bits to the left. The bits described previously in the ICCS register are visible to the WBUS rotated left 16 bit positions. The following TOK control bits are explained below.

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 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 <2Ø>) which is set by the macro program to initiate the transfer of the NICR data to the ICR.

ICCS BIT	FUNCTION
SR (WBUS <19>)	SR means service request, SR is set by the TOK gate array to request service from the timer service micro routine 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 a transfer to the ICR is done and it prevents the ICR from being auto loaded after interrupt.

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:Ø> and the second is if TR is set indicating that the previous macro instruction was an MTPR that set the TRANSFER bit (WBUS <2Ø>). At the next BUT SERVICE the TIMER SERVICE request, if honored, will invoke the TIMER SERVICE microroutine that begins at control store address ØØ14. 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 IRD1 on the VAX-11 macro instruction preempted by the TIMER SERVICE request.

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 will drive the MICRO VECTOR address lines <2:Ø> with the highest priority request encoded into a micro address. 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 ØØ3B. The microcode would transfer control of the macro program to the Timer Service Routine that is pointed to by contents of SCBB+CØ. This routine must clear the IR bit of the ICCS before using the timer again or an Interval Timer ERROR will occur.

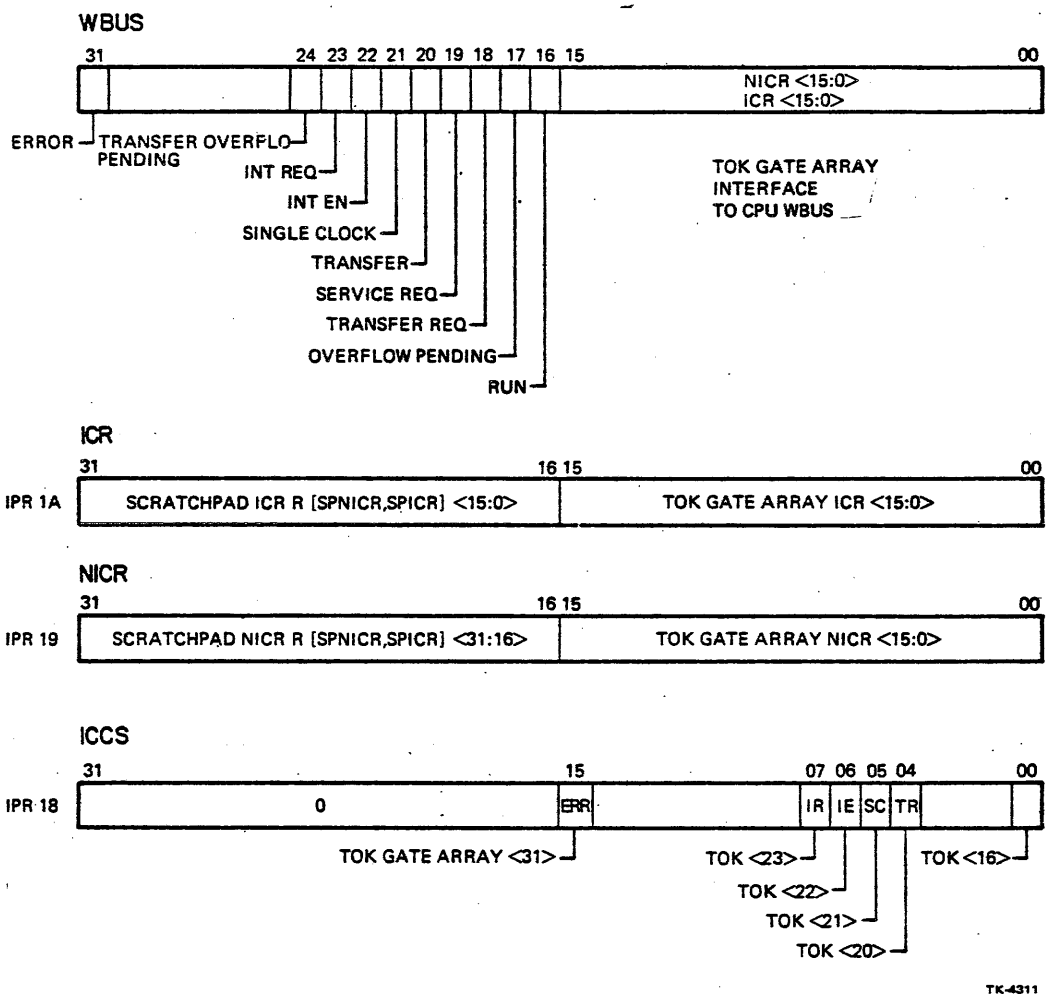
7.5 TIMER MACRO CODING EXAMPLE

Figure 7-34 is an example macro program that activates the interval timer. This is a stand alone program and could not operate under VMS as is. All this routine does is set up the interval timer with a 1Ø second interval. The timer is started and the CPU just waits for the interrupt that occurs 1Ø seconds later when the counter overflows. When the counter overflows, the interrupt service routine is entered via SCB+CØ where it just halts the CPU. If C is typed at the console the program will reload the timer and wait for another 1Ø seconds until the counter overflows. That all this program is capable of, but it does show how to load the timer, start it, and handle the interrupt at Vector SCBB+CØ. Let's analyze it.

Lines 4,5, and 6 are assembler directives that build the SCB in the low two pages of memory (Ø to 3FC). The value associated with label INTERVAL is the test interval in microseconds. 1ØØØØØØØ microseconds is the same as ten seconds. The label ST_TIM has the value 51 hex associated with it and this will be used to set bit <6> Interrupt Enable, bit <4> Transfer NICR to ICR, and bit <Ø> 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 2Ø longwords\for the stack space. Line 23 is the beginning of the main program to get things going. The first instruction sets up the stack pointer. The next instruction points the SCBB to address Ø in memory. At line 25 the interval value defined at line 8 is negated (2's complement) and put in RØ. The address of the service routine (TIM_SERV) is moved into the SCB so that timer interrupt will vector 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 just waits for the interrupt.

When the interval timer overflows, the interrupt request at IPL 18 is generated and if honored, the macro code resumes at the label called TIM_SERV. The interrupt service routine must clear bit <7> in the ICCS or when the REI is executed, the IPL 18 interrupt request is immediately generated again. The HALT instruction is there to print 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. This means that the timer can be reloaded from the NICR continuously. The primary intent of the program is to show the mechanism by which the timer establishes intervals of execution time for programs in a time shared environment. This concludes the discussion of the interval timer operation.


WBUS
31
24 23 22 21 20 19 18 17 16 15
00
NICR <15:0>
ICR <15:0>
ERROR
TRANSFER OVERFLO PENDING
INT REQ
INT EN
SINGLE CLOCK
TRANSFER
SERVICE REQ
TRANSFER REQ
OVERFLOW PENDING
RUN
TOK GATE ARRAY INTERFACE TO CPU WBUS
ICR
31
16 15
00
IPR 1A
SCRATCHPAD ICR R [SPNICR,SPICR] <15:0>
TOK GATE ARRAY ICR <15:0>
NICR
31
16 15
00
IPR 19
SCRATCHPAD NICR R [SPNICR,SPICR] <31:16>
TOK GATE ARRAY NICR <15:0>
ICCS
31
15
07 06 05 04
00
IPR 18
0
ERR
IR
IE
SC
TR
TOK GATE ARRAY <31>
TOK <23>
TOK <22>
TOK <21>
TOK <20>
TOK <16>
TK-4311

```
TEST            TIMER                                   27-AUG-1980 16:39:20  VAX-11 Macro V02.45        Page   1
                                                        19-AUG-1980 12:44:02  _DLA0:[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
```

VAX-11/750 LEVEL II

Remote Diagnostics

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The RDM was placed at this section of the course to allow the student to understand its use in the overall system. That use is: the RDM is a TOOL to be used by the technician to maintain the equipment. The RDM is NOT needed by the customer to operate the system. Due to this fact the maintenance philosophy is: THE RDM IS A FIELD REPLACABLE UNIT AND WILL NOT BE REPAIRED. This makes sense if you say to yourself; "Self I can't take up the customer's system time fixing my repair equipment."

What you should obtain, from this lesson and following lessons, is the confidence in the use of the RDM and the many functions available to you when using the RDM. These functions will be mentioned in this lesson but not all will be reinforced with labs directly after the lesson. The different uses will be spread out to allow you the chance to apply these uses in actual troubleshooting situations as they arise in the course.

There are two levels of maintenance involved with the use of the RDM.

1. By the branch or support person on site to fix a problem by running Micro Diagnostics (TO RUN MICROs YOU HAVE TO HAVE AN RDM) or by taking an instruction through a complete cycle one micro instruction at a time. (AGAIN YOU NEED THE RDM TO SINGLE STEP MICROINSTRUCTIONS.)

2. The same functions may be used remotely by the DDC in Colorado to help the branch or support person perform fault isolation or preventive maintenance.

THESE ARE NOT ALL THE FUNCTIONS OF THE RDM. They do show that it is needed to perform onsite maintenance as well as remote diagnosis.

OBJECTIVES

At the completion of this lesson the student will be able to distinguish between the two basic modes of operation available using the RDM.

The student will be able to match the blocks in a blank RDM block diagrams to its function.

The student will be able to match the RDM commands to their related function in running micro-diagnostics while troubleshooting the machine.

SAMPLE TEST ITEM

Match the function in the right column to the command in the left column by placing the proper letters in the space beside the command.

EXAMINE	A. Enter RDM mode
DEPOSIT	B. Place data into a location
RET	C. Read data from a location
^P	D. Return to previous mode

RESOURCES

11/750 RDM Maintenance Card
11/750 Option Technical Manual and Microdiagnostic User's Guide

OUTLINE

VI. Remote Diagnostic Module

- A. Reasons for RDM
- B. Physical Characteristics
  1. Location
  2. Power
  3. Front Panel Indicators
- C. Operational Overview
- D. Block Diagram
- E. Addressing
- F. Pseudo Instructions

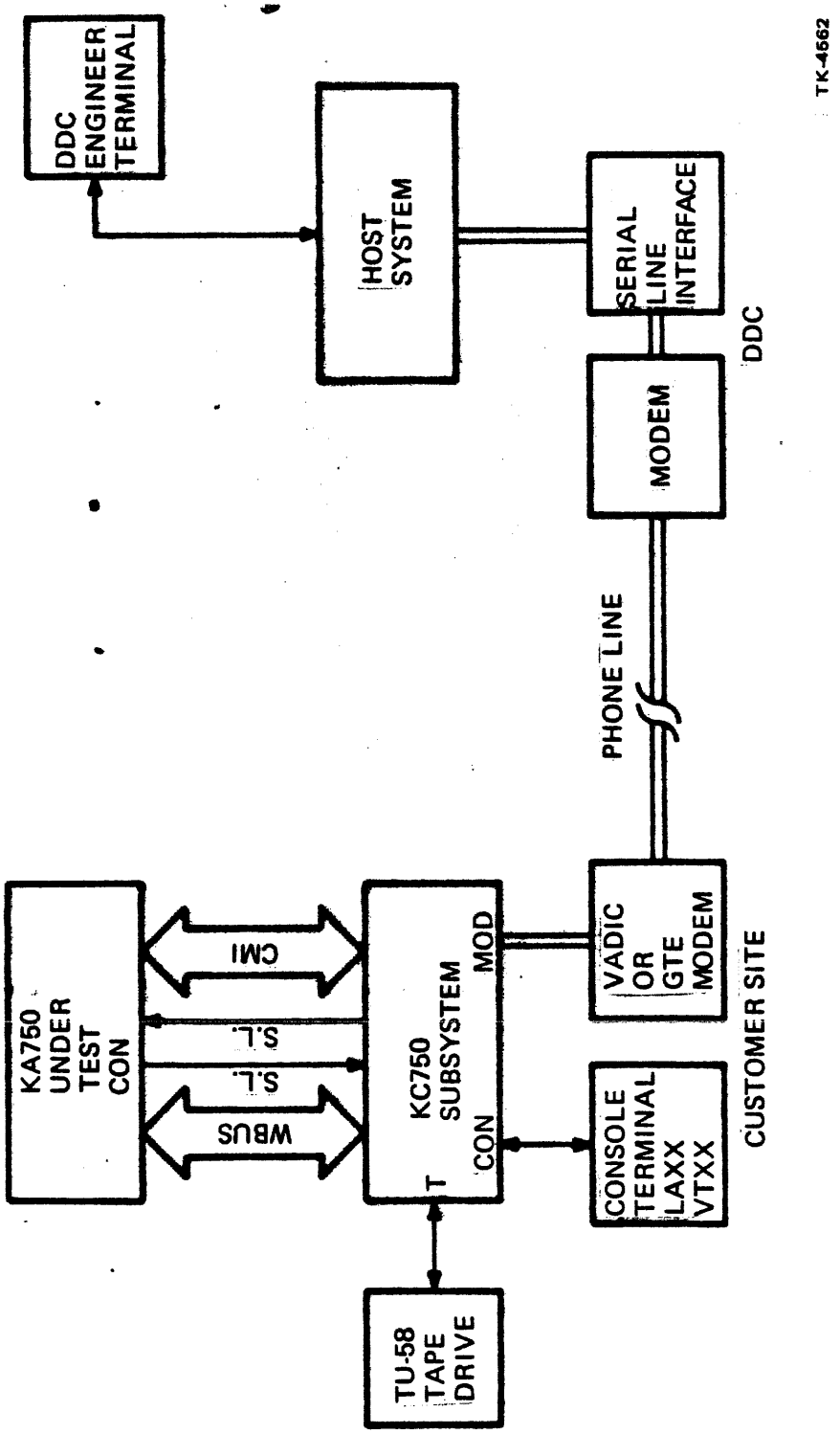


Figure 6-1 RDM Functional Location

Physical Characteristics

1. Located in slot 6 of extended hex backplane as noted in introduction.

2. Power

   a. MAX +5V @ 12.ØA +12V @ 12ØMA -15V @ 85MA
      TYP +5V @ 9.3A +12V @ 6ØMA -15V @ 3ØMA

3. Control Panel Interconnections

   Connections to the control panel will be incorporated in the basic cabinet and its wire harness. The processor will have full use of all processor specific controls and indicators whether the RDM is installed or not. When the RDM is installed in its slot the RD specific functions will also be operational. The RD functions implemented on the control panel are as follows:

   a. INDICATORS

      1. REMOTE - This green light is lit by the RDM software whenever it detects that the control panel key switch is in one of the two remote positions.

      2. CARRIER - This amber light is lit by the RD software whenever it detects that the remote port carrier is present. It is an indicator to the customer that the DDC has established connection.

      3. TEST - This green light is lit by the DDC software to indicate that tests are in progress.

      4. FAULT - This red light is lit by the RDM software if it detects a fault in its own logic. No tests should be attempted when the fault indicator is lit.

   b. SWITCH SETTINGS

      The processer's keyswitch has 5 positions, two of which allow remote connection. The REMOTE and REMOTE SECURE positions allow the DDC to connect to the processor. The REMOTE position allows the control console to enter console mode at any time while the REMOTE SECURE prevents the console from being used in other than program mode.

1. There are also switch settings that relate to the remote BAUD rate but those will be covered in the installation section in the final week.

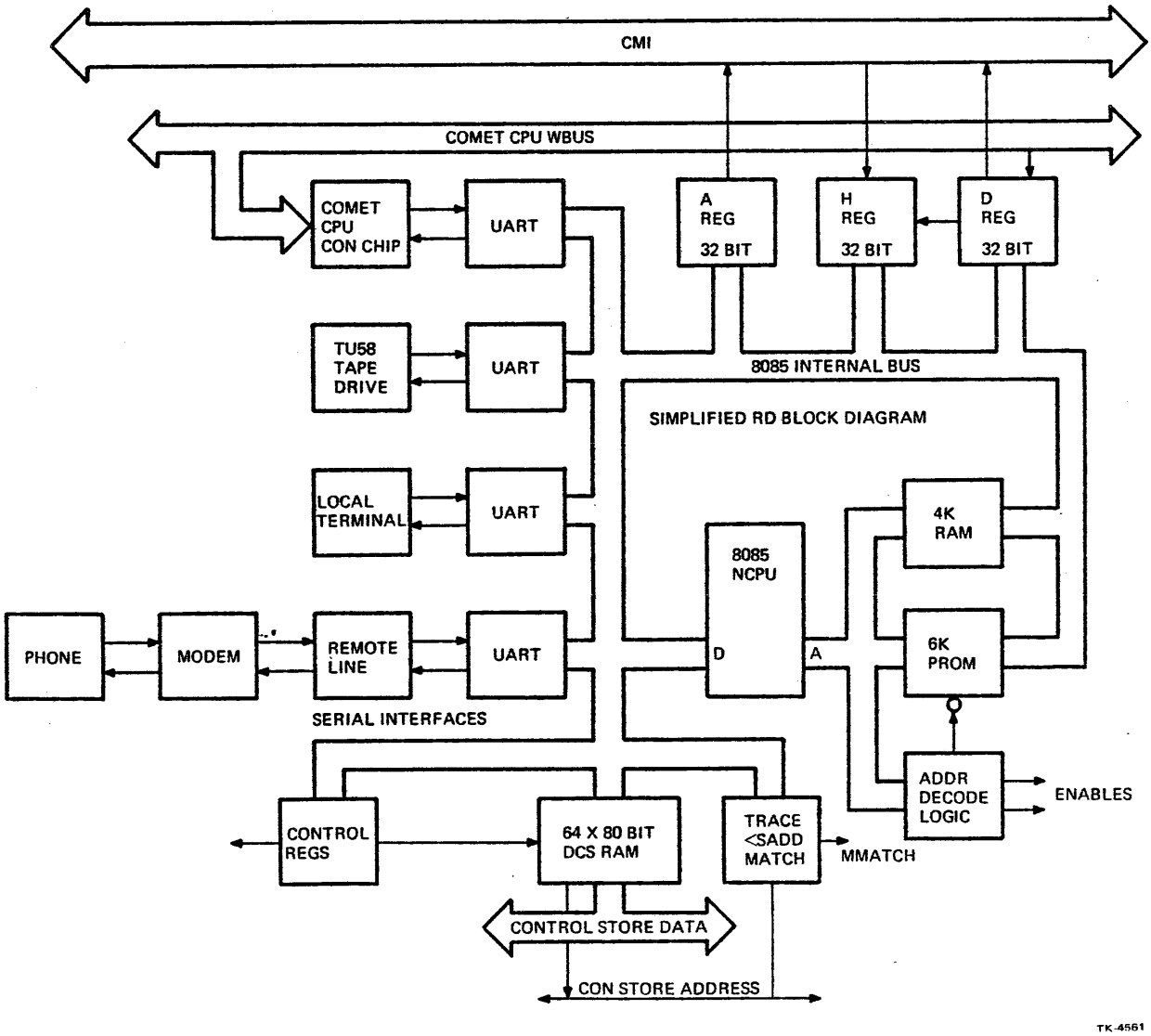


Figure 6-2 Simplified RDM Block Diagram

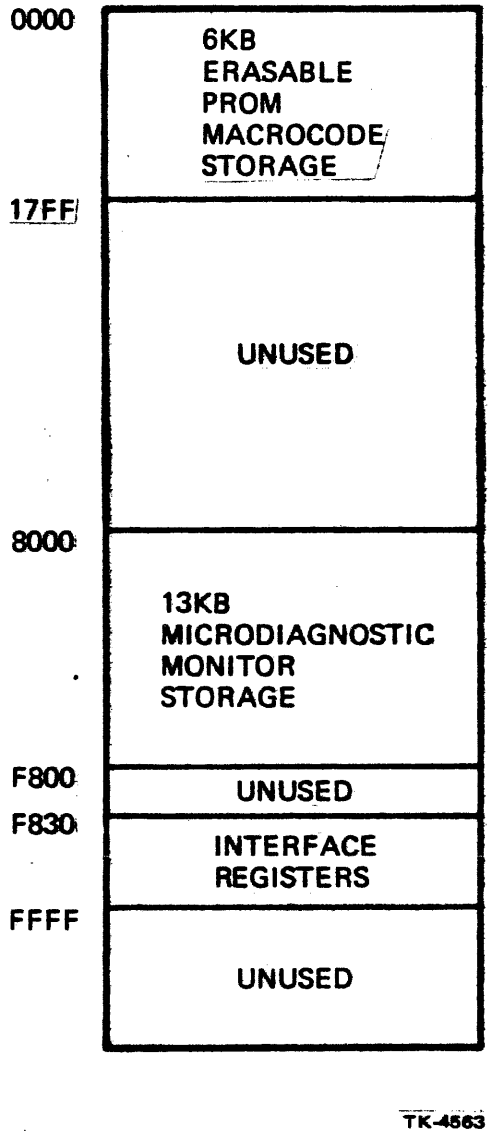


Figure 6-3 8085 I/O Addressing

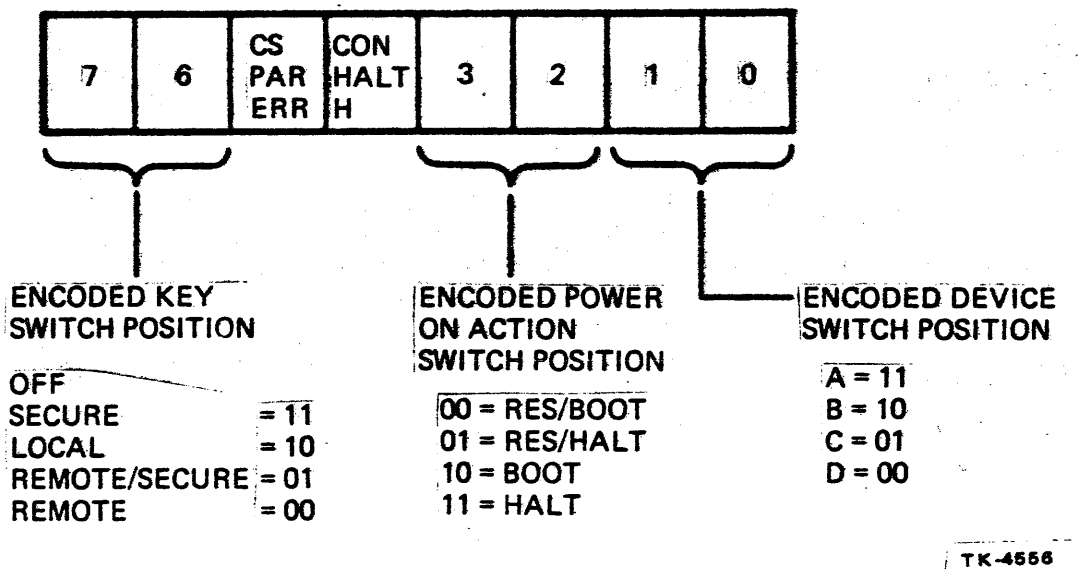


Figure 6-4 Front Panel Status Register F820

Remote D	Red	-	This indicates that the keyswitch is in either of the remote positions.
RD Test	Green	-	This indicator is illuminated if a remote diagnostic session in protocol mode is in progress. The transparent mode turns the lamp off.
Carrier	Amber	-	Carrier indicator is on if the modem is receiving the carrier from the telephone line.
RD Fault	Red	-	Indicates a hardware fault on RDM module if constantly on. Normal sequence is to illuminate for 10 seconds at power up and then go out. If lamp is always one, replace RDM.

Figure 6-5 RDM Operator Control Panel Indicators

VAX-11/750 LEVEL II

CMI

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The CMI is a tristate synchronized inforamtion path for data exchanges between the central processing cluster (CP Cluster), memory, and adapters of the Comet system.

SYNOPSIS

The CPU Memory Interconnect module includes lecture on the architecture and types of data transfers used.

OBJECTIVES

Provided with a multiple choice test, correctly answer questions regarding CMI architecture and types of transfers.

SAMPLE TEST ITEM

Which of the following best describe the CMI?

a) 55 lines of equal length forming a 2 layer belt
b) 8Ø lines of various lengths formed by etch
c) 55 lines etched into the backplane, all equal length
d) 45 etched lines of various length

RESOURCES

Comet Specification

MODULE OUTLINE

VIII. CPU MEMORY INTERCONNECT (CMI)

A. CMI Structure

1. CMI protocol
2. CMI status bits
3. CMI control signals
4. Data/Address
5. Clock

B. Address/Data Transfers

1. Address format

a. function code
b. mask field

C. Summary

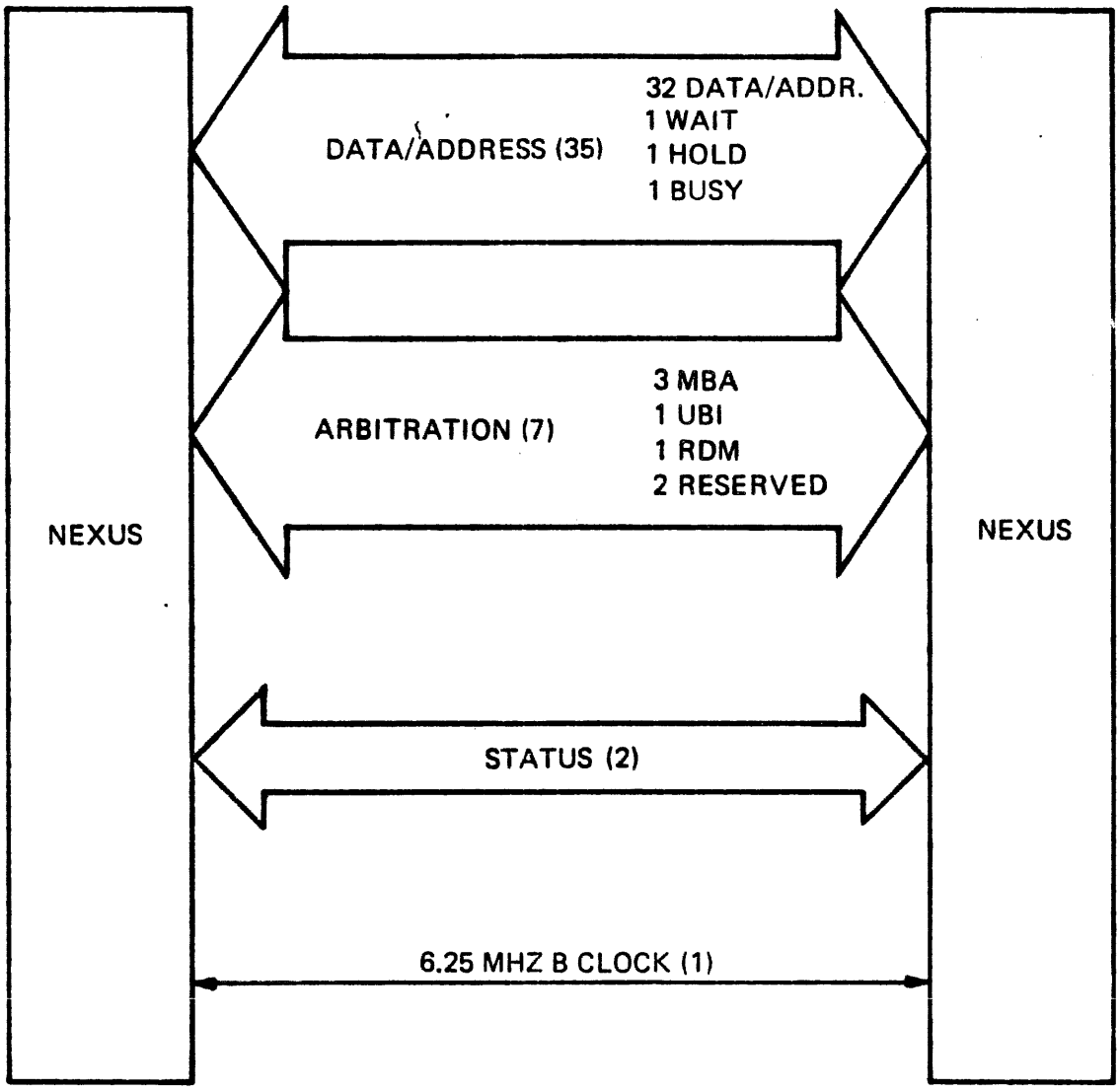


Figure 8-1 CMI Structure

ITEM	PRIORITY
RDM	7
RESERVED	6
RESERVED	5
CUI	4
OPTION	3
OPTION	2
OPTION	1
CPU	NONE

CMI PROTOCOL

TK-2063

Figure 8-2 CMI Protocol

ST1	ST0	INTERPRETATION
0	0	NON EXISTENT MEMORY
0	1	NON CORRECTABLE ERROR
1	0	CORRECTED DATA
1	1	NO ERRORS

THE CMI STATUS BITS

TK-2062

Figure 8-3 CMI Status Bits

DATA/ADDRESS BUS BUSY

THE DATA/ADDRESS BUS BUSY (DBBZ) SIGNAL INDICATES THE AVAILABILITY OF THE DATA/ADDRESS BUS. THE ABSENCE OF DBBZ INDICATES TO ALL NEXUS THAT THE DATA/ADDRESS BUS WILL BE FREE AT THE BEGINNING OF THE NEXT CMI CYCLE.

DBBZ

DBBZ IS ASSERTED BY THE BUS MASTER FOR ONE CMI CYCLE WHEN AN ADDRESS FORMAT IS PLACED ON THE DATA/ADDRESS BUS. DURING A READ, THE SLAVE ALSO ASSERTS DBBZ IN THE FOLLOWING CYCLE AND CONTINUES TO ASSERT IT UNTIL THE READ DATA IS READY FOR TRANSMISSION. DURING A WRITE, DBBZ IS ASSERTED BY THE SLAVE WHILE IT PREPARES TO ACCEPT THE WRITE DATA. FOR THIS CASE, DBBZ IS NOT ASSERTED IF THE NEXUS IS IMMEDIATELY READY.

HOLD

THE HOLD SIGNAL CAN BE ASSERTED BY ANY NEXUS TO PREVENT OTHER NEXUS FROM GAINING CONTROL OF THE DATA/ADDRESS BUS. THE HOLD SIGNAL IS PRIMARILY PROVIDED TO ALLOW BUS WATCHING CACHES TO CONTROL THE RATE AT WHICH WRITE TRANSACTIONS OCCUR ON THE CMI. WHILE HOLD IS ASSERTED, ALL BUS REQUESTS ARE IGNORED.

WAIT

THE WAIT SIGNAL IS ASSERTED BY A NEXUS WHEN IT INITIATES AN INTERRUPT TRANSACTION. THE ASSERTION OF WAIT IS AN INDICATION TO THE CPU THAT AN INTERRUPT TRANSACTION IS IN PROGRESS ON THE UNIBUS AND THAT A WRITE VECTOR OPERATION MAY BE PENDING. WAIT IS REMOVED AT THE BEGINNING OF THE CMI CYCLE FOLLOWING THE COMPLETION OF THE INTERRUPT TRANSACTION. THE REMOVAL OF WAIT ALLOWS THE CPU TO CONTINUE NORMAL OPERATION.

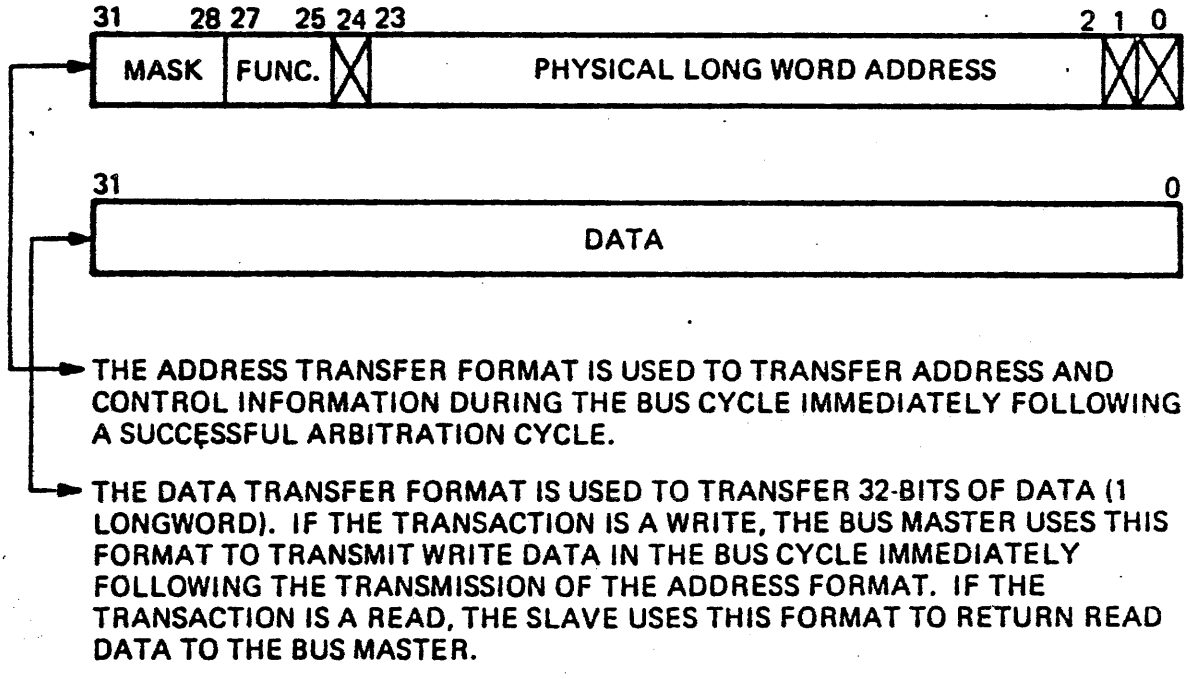
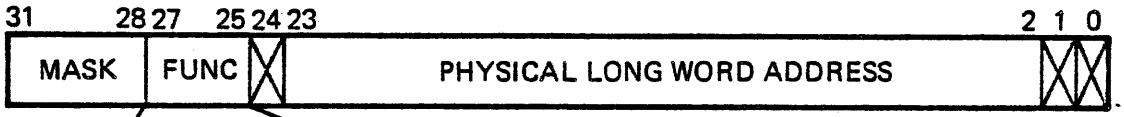


Figure 8-4 CMI Data/Address Formats



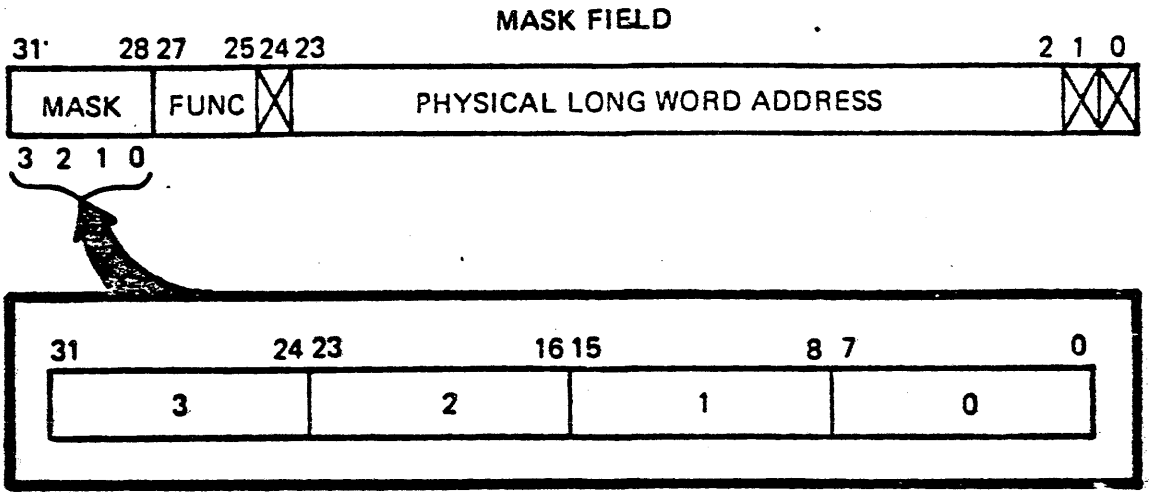
FUNCTION CODES

DATA/ADDRESS 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 UNLO CK
1	1	0	WRITE VECTOR
1	1	1	(UNDEFINED)

CMI FUNCTION CODES

TK-2073

Figure 8-5 CMI Function Codes



EACH BIT IN THE MASK FIELD CORRESPONDS TO A PARTICULAR BYTE IN SUBSEQUENTLY TRANSFERRED DATA FORMAT. THESE BITS SPECIFY WHICH OF THE CORRESPONDING DATA BYTES ARE TO BE READ OR WRITTEN. THE BYTE IS SELECTED WHEN THE MASK BIT IS SET.

NEXUS WHICH ARE CAPABLE OF TRANSFERRING LONGWORDS IGNORE THE MASK ON READS AND ALWAYS RETURN ALL FOUR BYTES (E.G., MEMORY).

TK-2041

Figure 8-6 Mask Field

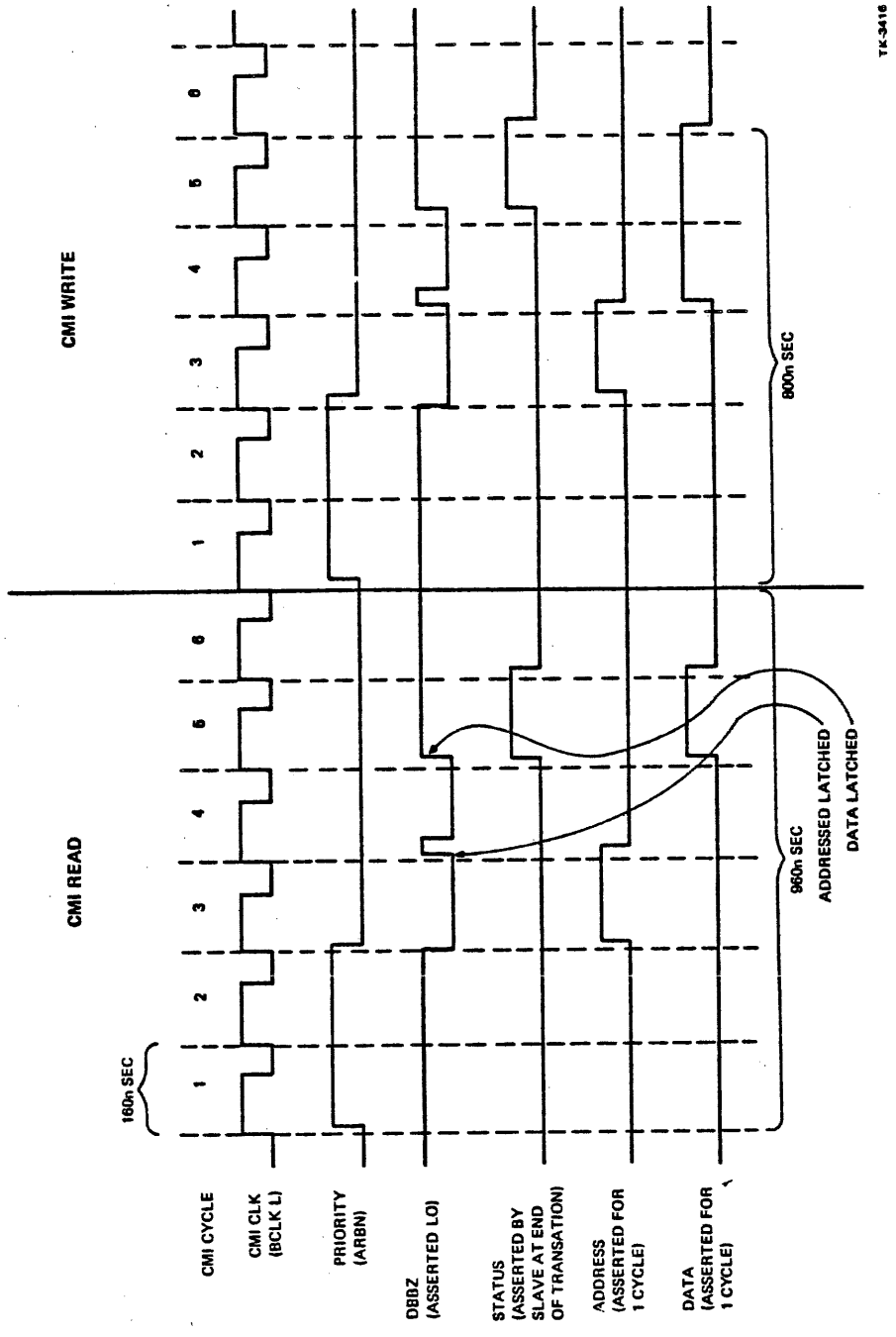


Figure 8-7 CMI Read/Write Timing

VAX-11/750 LEVEL II

Address Translation

Student Guide

Course Produced by Educational Services Department
of
Digital Equipment Corporation

OBJECTIVES

1. To utilize provided worksheets in order to perform address translations from a system virtual address to a physical address and from a processor virtual address to a physical address.

2. To utilize the console terminal and be able to determine what type of error occurred during an address translation by using the stack.

3. To correctly indicate on a series of true/false questions, statements regarding the 11/750 address translation procedures.

4. To run and interpret MIC Module Microdiagnostics that relate to address translation.

SAMPLE TEST ITEM

True or False

___ 1. Bit 31 in the system virtual address denotes which page table is accessed.

LAB EXERCISE

a. Load and run microdiagnostics

b. Run RDM to step through Translation Buffer Double T.B. miss.

RESOURCES

1. 11/750 Micro Listings

2. 11/750 Microdiagnostics and Listings for MIC Module

3. MIC Module Schematics

4. Student Guide

INTRODUCTION

When we talk about address translation we actually mean to take a virtual address (system or processor) and translate it to a physical address.

To understand the concept you will have to understand virtual address space in relation to physical address space and how the system uses the translation to control access to certain areas in the machine.

You will learn the actual machine translation from a virtual address to a physical address and the controlling factors in performing this translation.

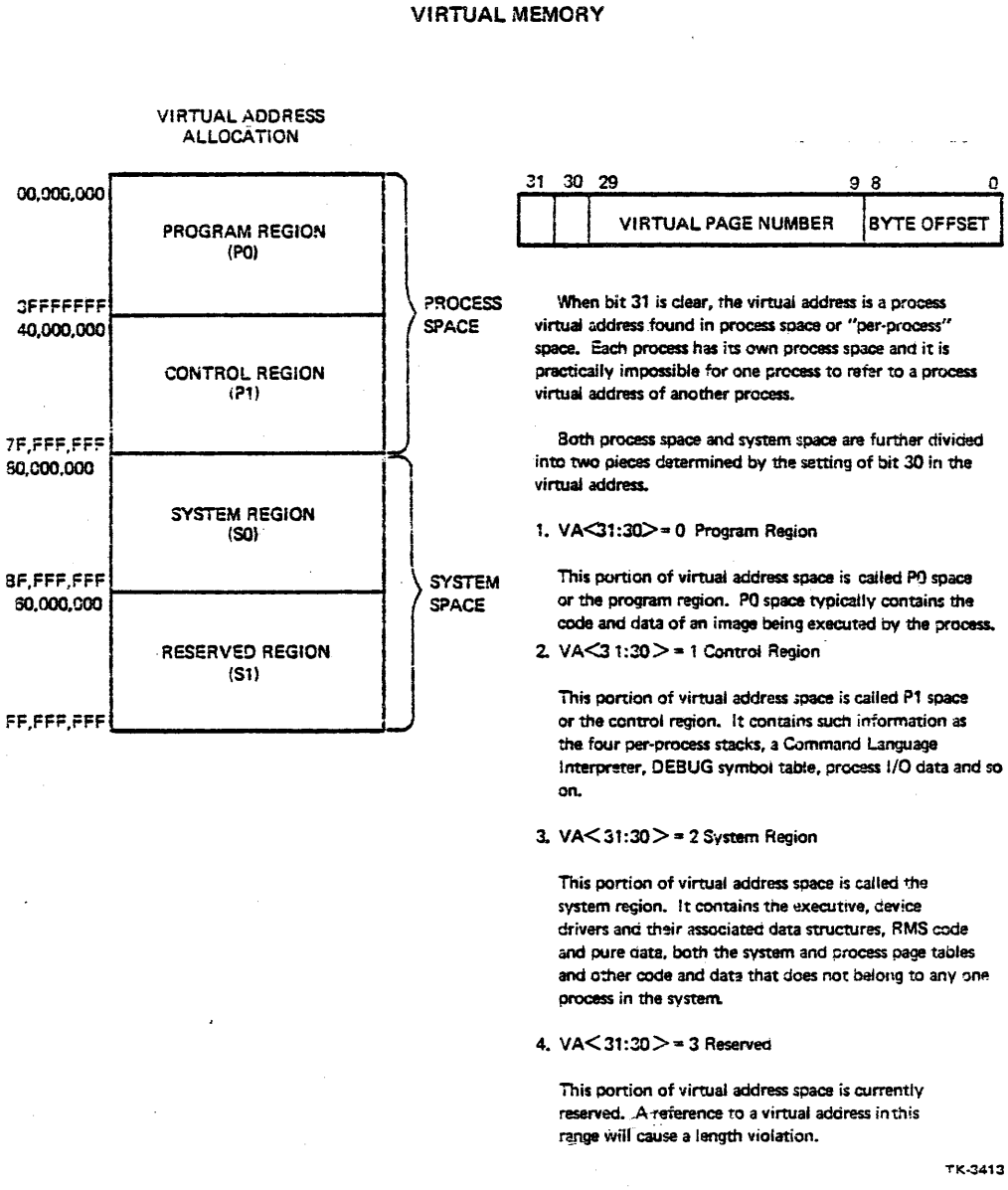


Figure 9-1 Virtual Memory

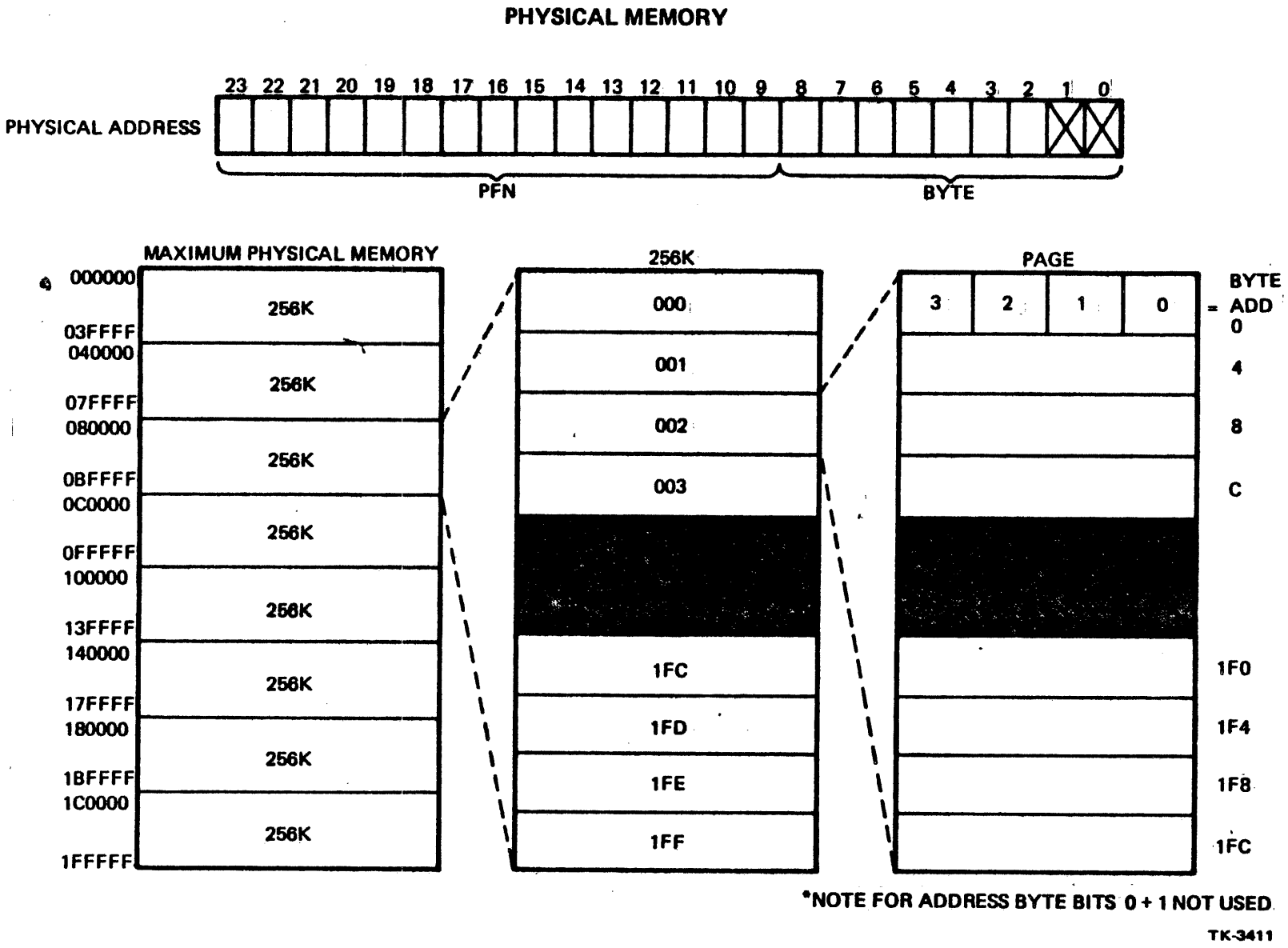


Figure 9-2 Physical Memory

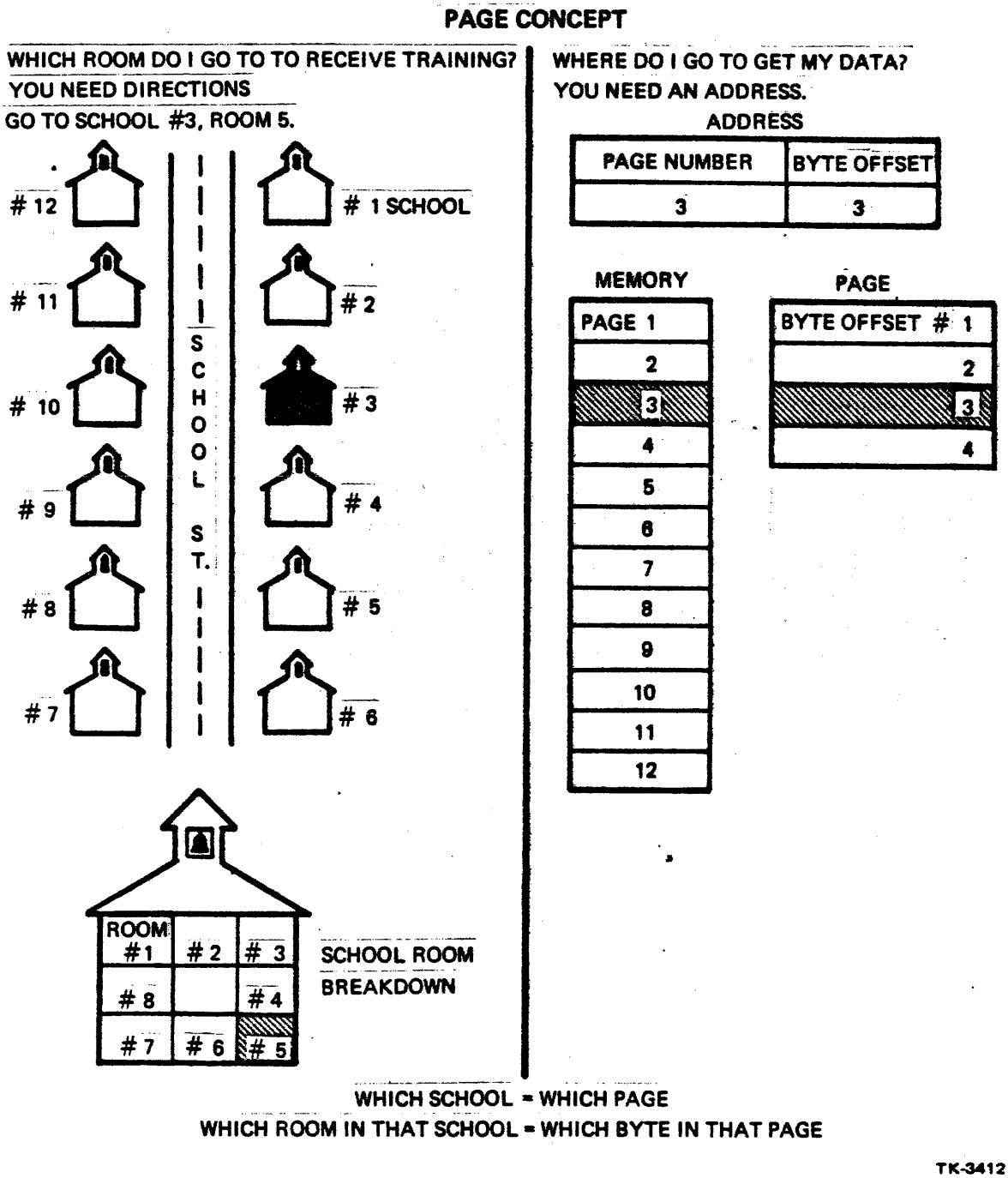


Figure 9-3 Page Concept

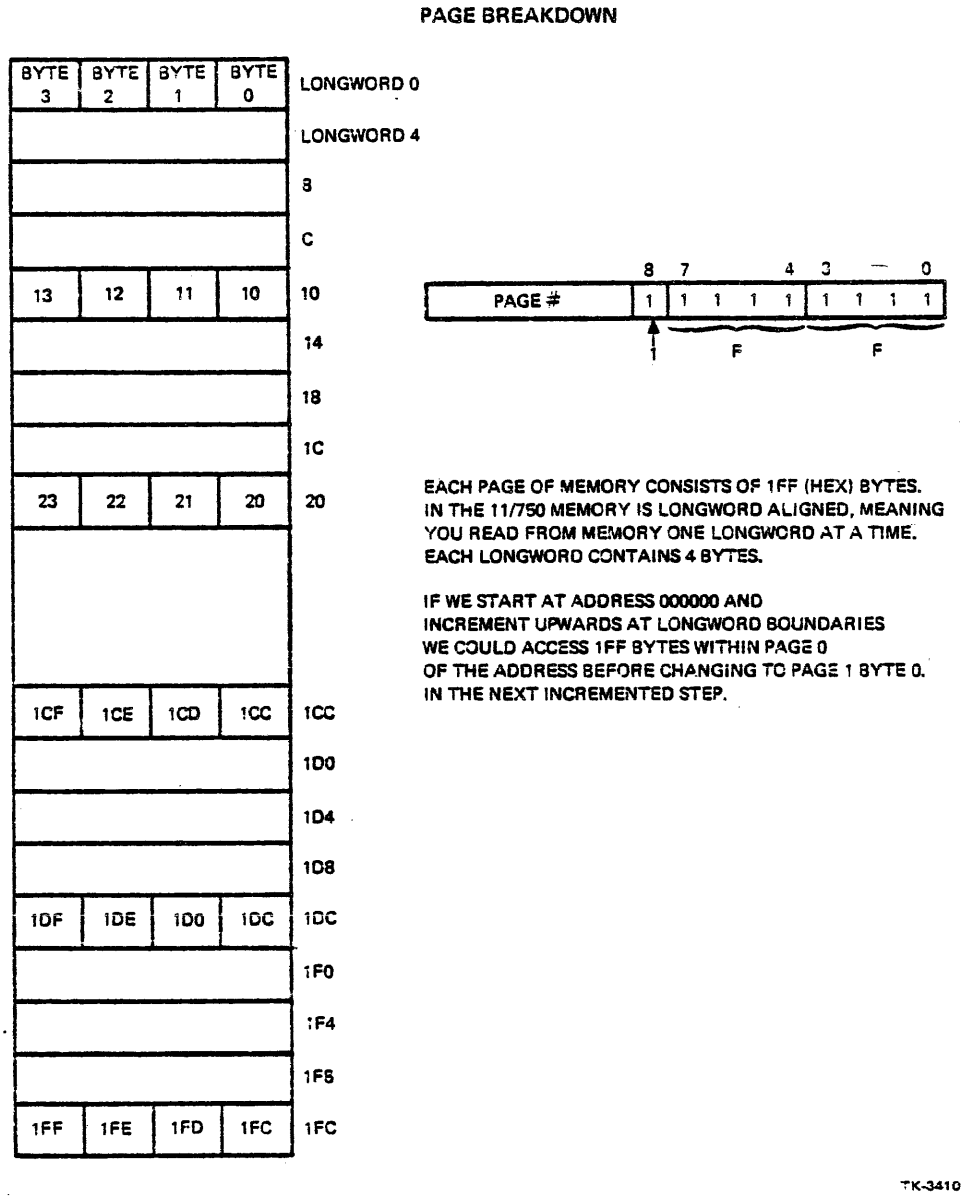
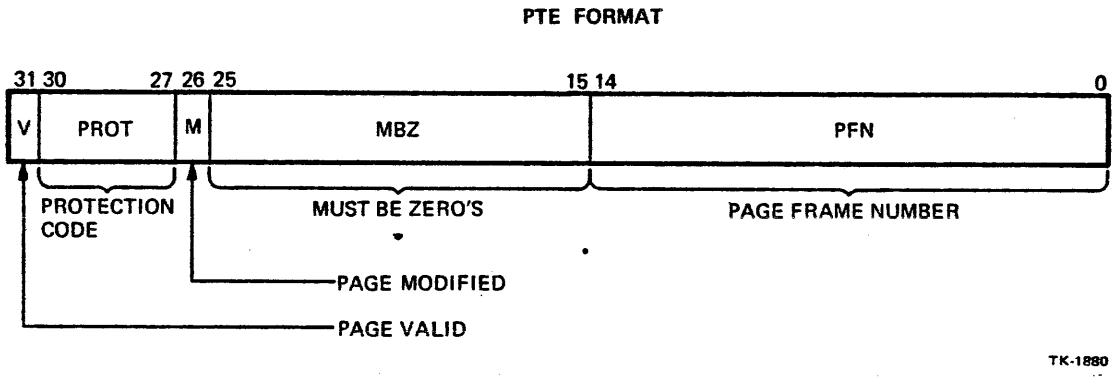


Figure 9-4 Page Breakdown



Page Tables and Mapping Registers

Figure 9-5 Page Table Entries

The system page table is built at initialization time and is located in contiguous pages of physical memory. Process page tables are set up at process creation and altered at image activation, image exit and in response to various system services. Process page tables are located in virtually contiguous pages of system space. That is, process page tables need not be physically contiguous. This design feature prevents a potentially serious fragmentation problem in physical memory.

System Page Table (SPT)	Describes the physical location and status of all pages in the system region of virtual address space.	
System Base Register (SBR)	Points to the starting physical location of the System Page Table.	IPR # ØC
System Length Register (SLR)	Specifies the number of entries in the System Page Table	IPR # ØD
Ø Page Table (PØPT)	Describes the physical location and status of all pages in the program region of virtual address space.	
PØ Base Register (PØBR)	Contains the system virtual address of the PØ page table.	IPR # Ø8
PØ Length Register (PØLR)	Specifies the number of entries in the PØ page table.	IPR # Ø9
1 Page Table (P1PT)	Describes the physical location and status of all pages in the control region of virtual address space.	
P1 Base Register (P1BR)	Contains the system virtual address of the P1 page table.	IPR # ØA
P1 Length Register (P1LR)	Specifies the number of entries in the non-existent portion of the P1 page table.	IPR # ØB

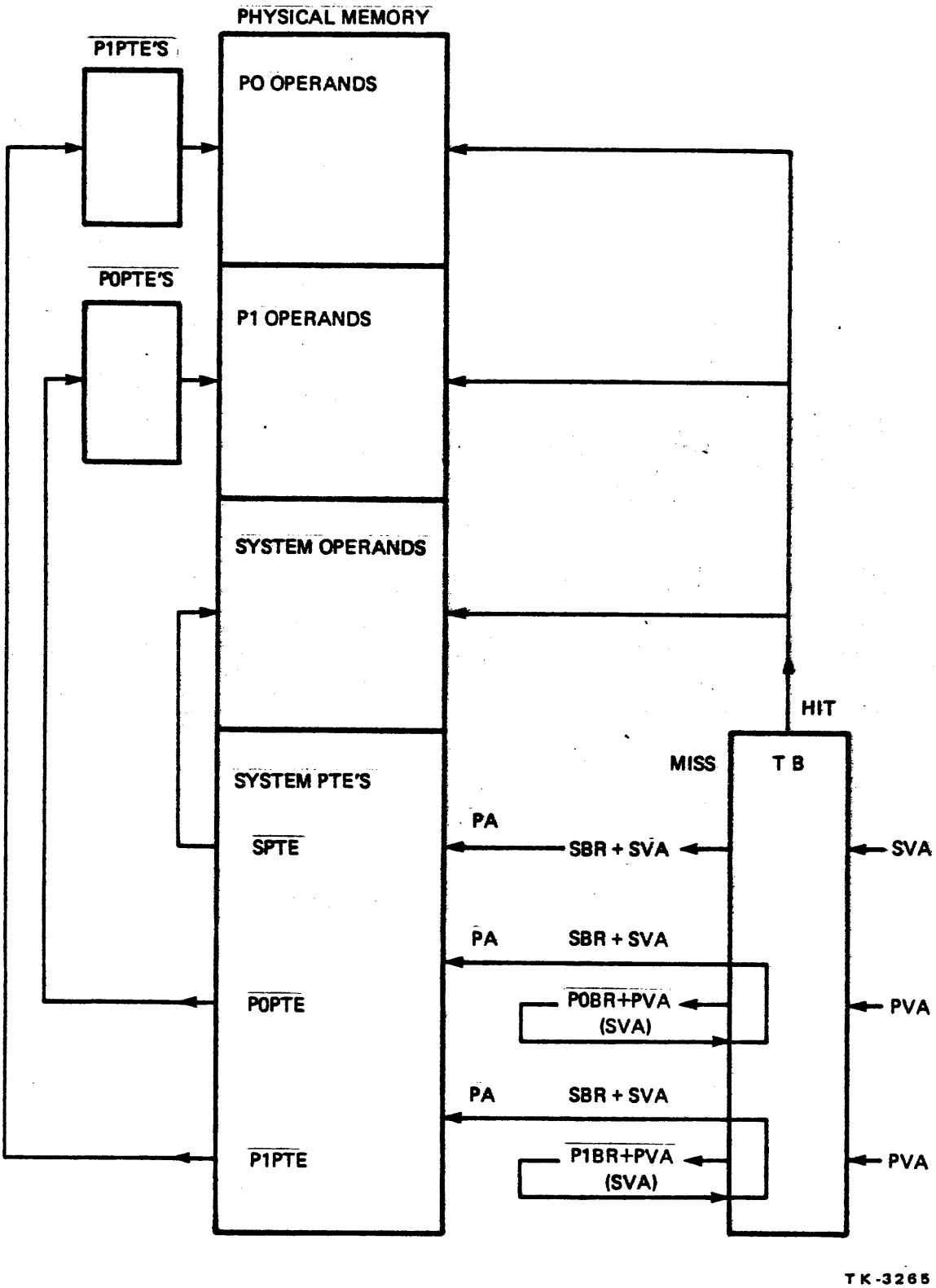


Figure 9-6 Address Translation Block Concept

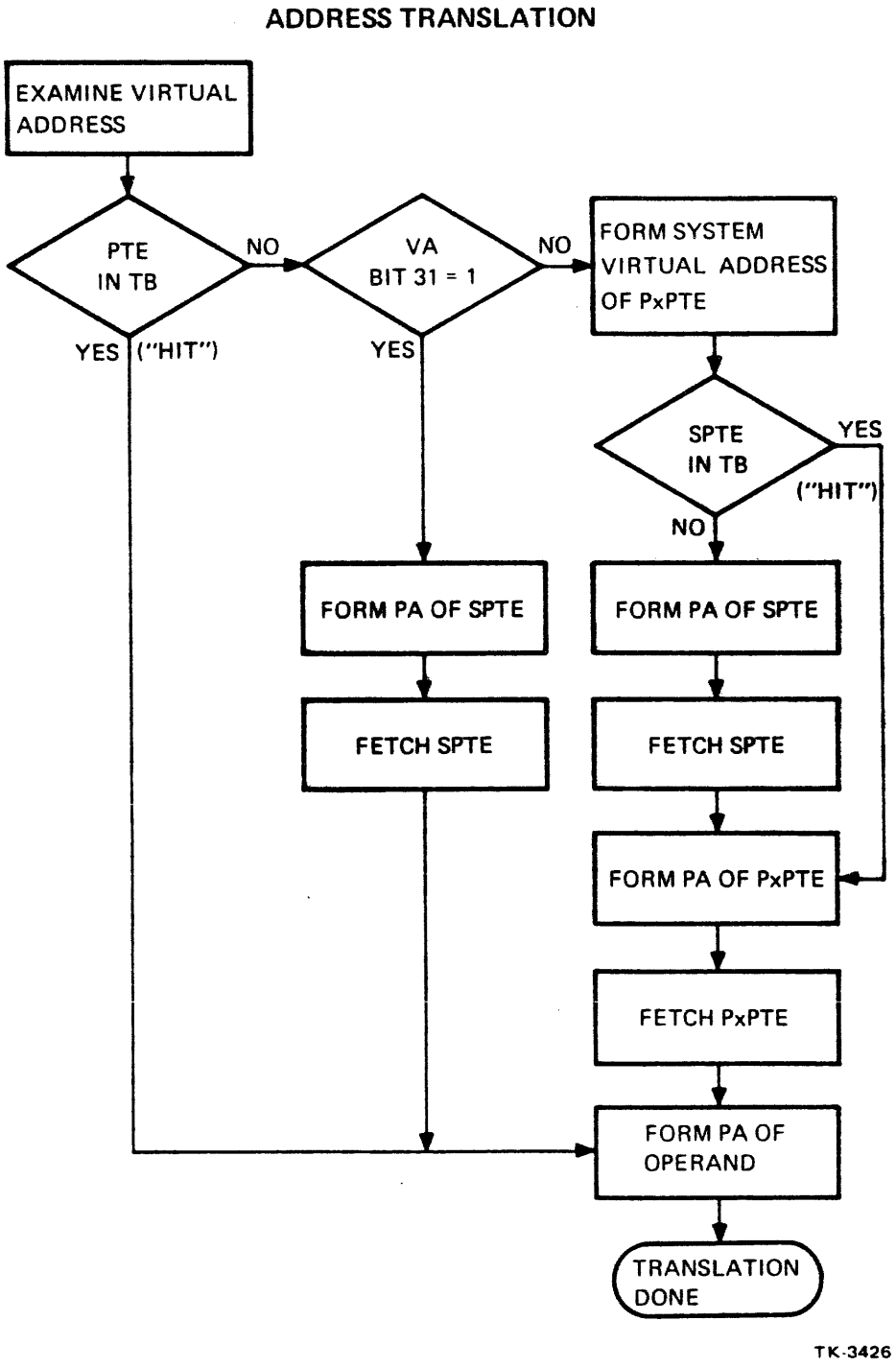
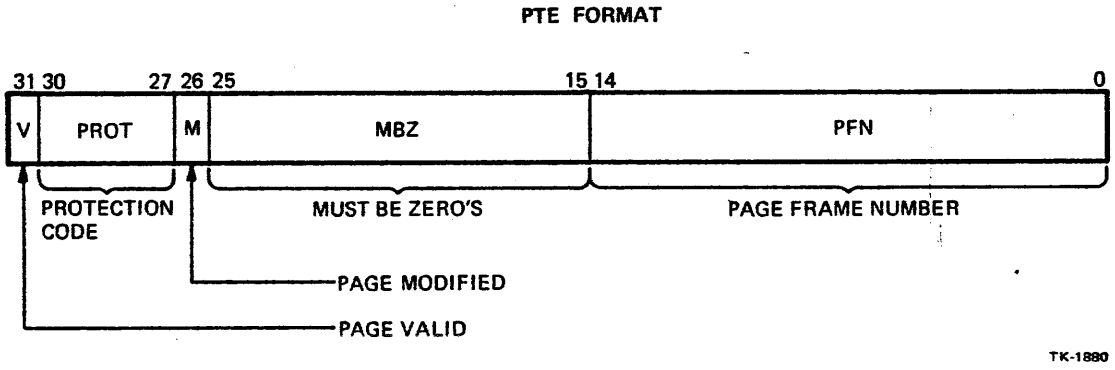


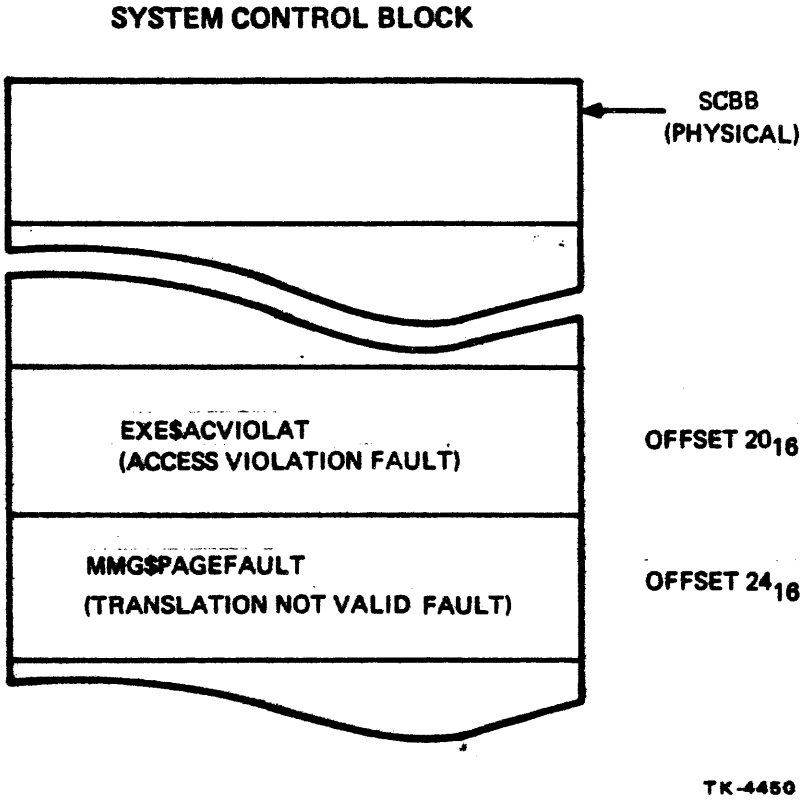
Figure 9-7 Address Translation



Page Tables and Mapping Registers

Figure 9-8 Page Table Entries

The system page table is built at initialization time and is located in contiguous pages of physical memory. Process page tables are set up at process creation and altered at image activation, image exit and in response to various system services. Process page tables are lcoated in virtually contiguous pages of system space. That is, process page tables need not be physically contiguous. This design feature prevents a potentially serious fragmentation problem in physical memory.



Memory Management Exceptions

Figure 9-9

During address translation, two differrent kinds of exception can occur: access violation and translation-not-valid exception. Both forms of exception are faults. That is, the processor backs up the faulting instruction so that it can be restarted when (or if) the exception is resolved. Two adjacent longwords in the system control block are set up at initialization time to point to the routines which will service these exceptions. Both exceptions are handled on the Kernel stack.

Management Management Exceptions

Access Violation

An access violation can occur in two different forms. A protection code violation occurs when the intended access request (read, modify, write) is not allowed for the current access mode. Recall that the protection code is found in bits <3Ø:27> of the appropriate page table entry.

A length violation occurs when the virtual page number of the address to be translated is greater than or equal to the contents of the appropriate length register. (Because P1 space grows toward smaller addresses, th length violation fault occurs when VA<29:9> is strictly less than the contents of P1LR.

When an access violation occurs, the faulting PSL and PC are pushed onto the kernel stack, followed by the virtual address which caused the access violation. Finally, a longword fully describing the access violation is pushed onto the stack. Note that bit <Ø> of the reason mask distinguishes length violations from protection code violations.

PROTECTION CODE IN PTE	CURRENT ACCESS MODE			
	KERNEL	EXECUTIVE	SUPERVISOR	USER
0000	--	--	--	--
0001	UNPREDICTABLE		UNPREDICTABLE	
0010	RW	--	--	--
0011	R	--	--	--
0100	RW	RW	RW	RW
0101	RW	RW	--	--
0110	RW	R	--	--
0111	R	R	--	--
1000	RW	RW	RW	--
1001	RW	RW	R	--
1010	RW	R	R	--
1011	R	R	R	--
1100	RW	RW	RW	R
1101	RW	RW	R	R
1110	RW	R	R	R
1111	R	R	R	R

-- (NO ACCESS)
R (READ-ONLY ACCESS)
RW (READ AND WRITE ACCESS)

TK-3567

Figure 9-10 Use of Protection Codes for Access Control

Memory Management Exceptions

Access Violations

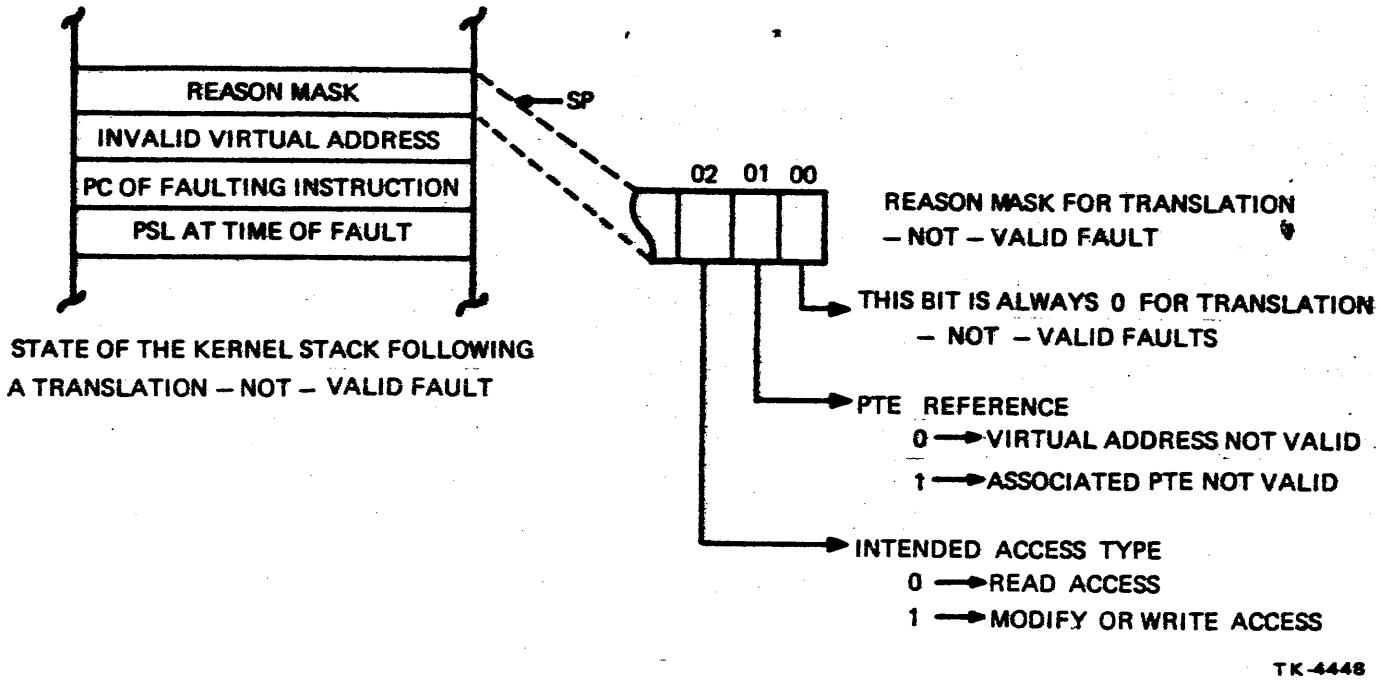


Figure 9-11 State of Kernel Stack Following Access Violation Fault

Memory Management Exception

Translation-Not-Valid Fault

A Translation-Not-Valid fault occurs when the Valid Bit (VA<31>) is clear. The faulting PSL and PC, followed by the invalid virtual address and reason mask, are pushed onto the kernel stack. Control is passed to an executive routine called the pager, which will use the information in the invalid PTE to locate the page and add it to the working set of the requesting process. (The information contained in an invalid PTE and the actions taken by the pager will be discussed in the next module.)

Since process page tables are mapped (by SPT entries), address translation for process virtual addresses can incur page faults both in translating the system virtual address of the process page table entry and in translating the process virtual address itself. These two different cases are distinguished by bit <1> of the reason mask.

Memory Management Exceptions

Translation Not Valid Fault

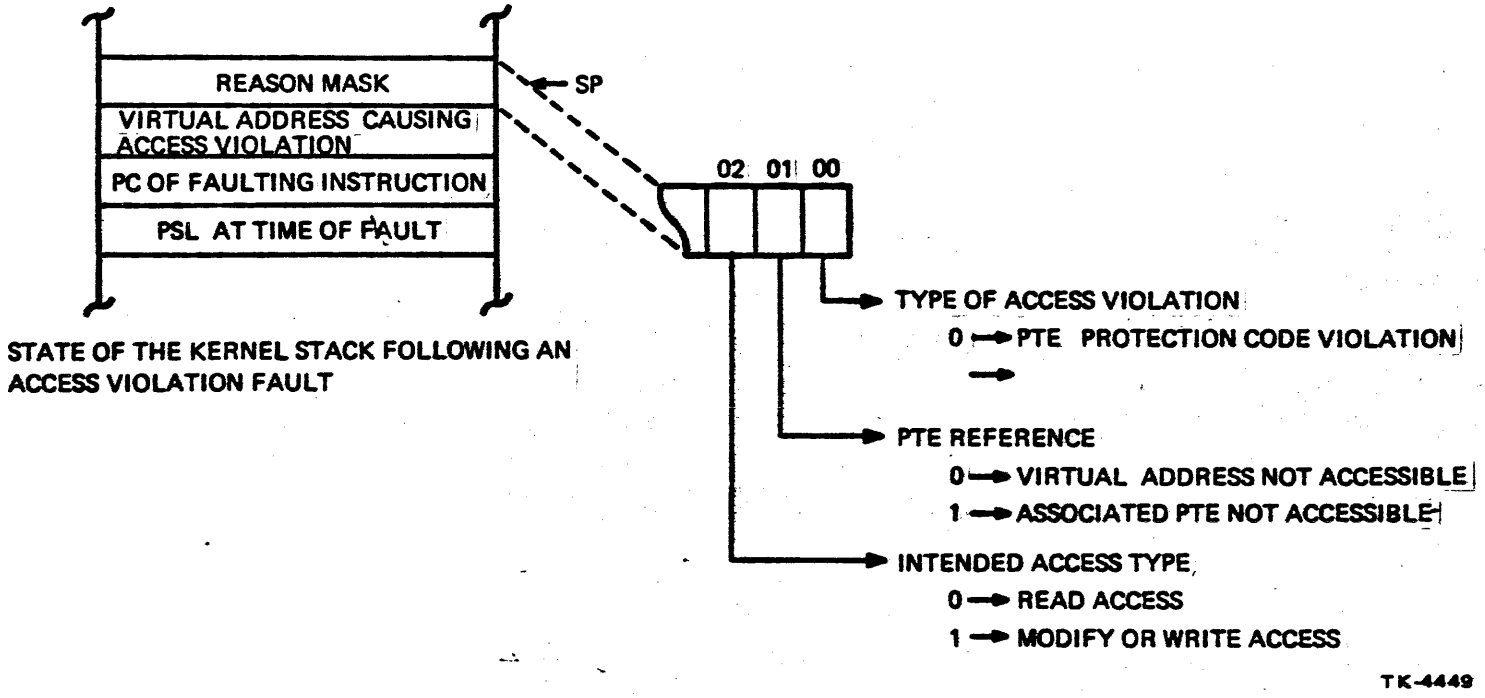


Figure 9-12 State of Kernel Stack Following a Translation Not Valid Fault

NOTE

The address translation mechanism checks the protection code before it checks the valid bit. Thus, if a given address translation could cause both an access violation and a page fault, the access violation will be taken. This design avoids the overhead of faulting into a process working set a page which it is not allowed to access. A further discussion is found on page 108 of the VAX-11/780 Hardware Handbook.

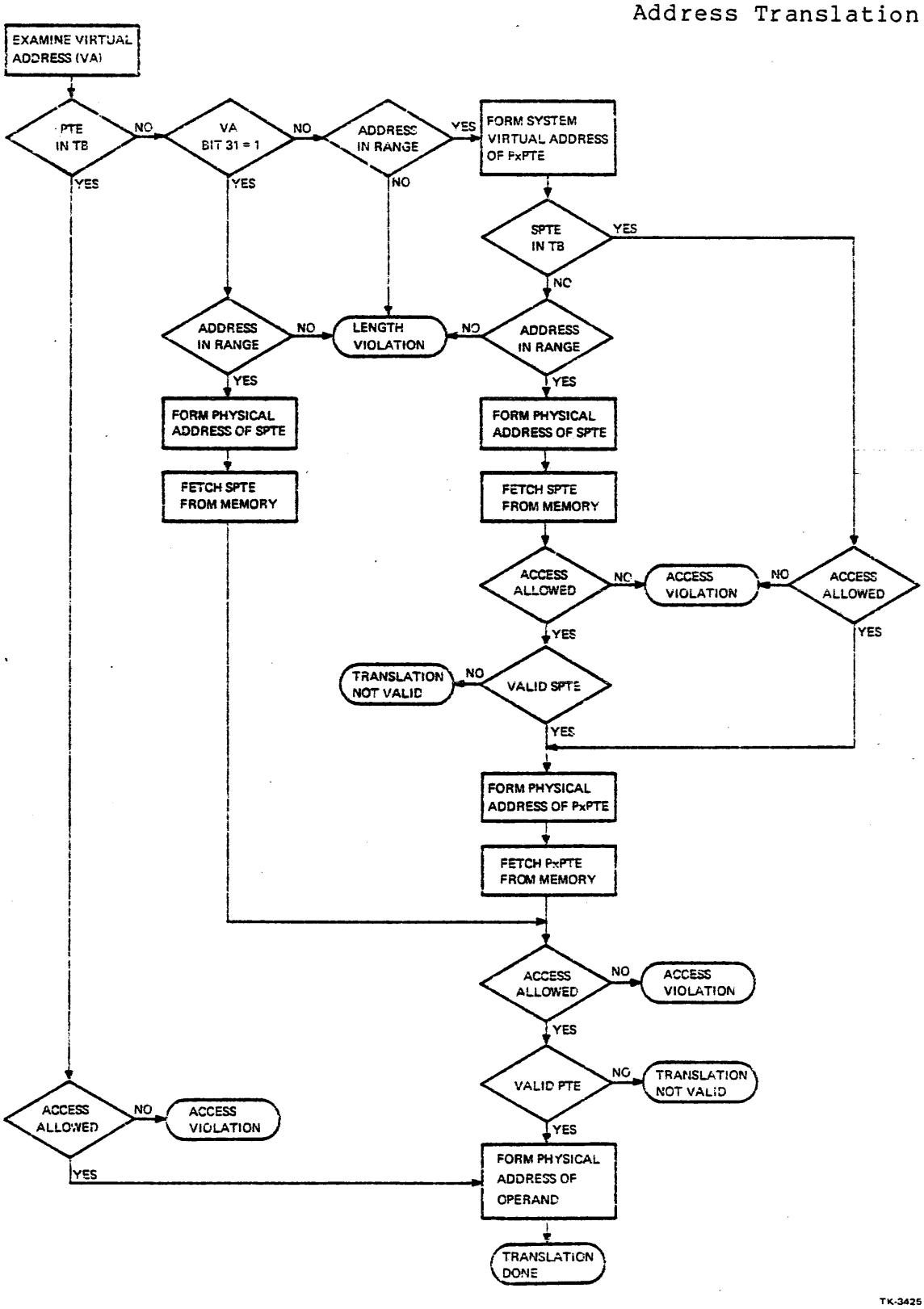


Figure 9-13 Address Translation Faults

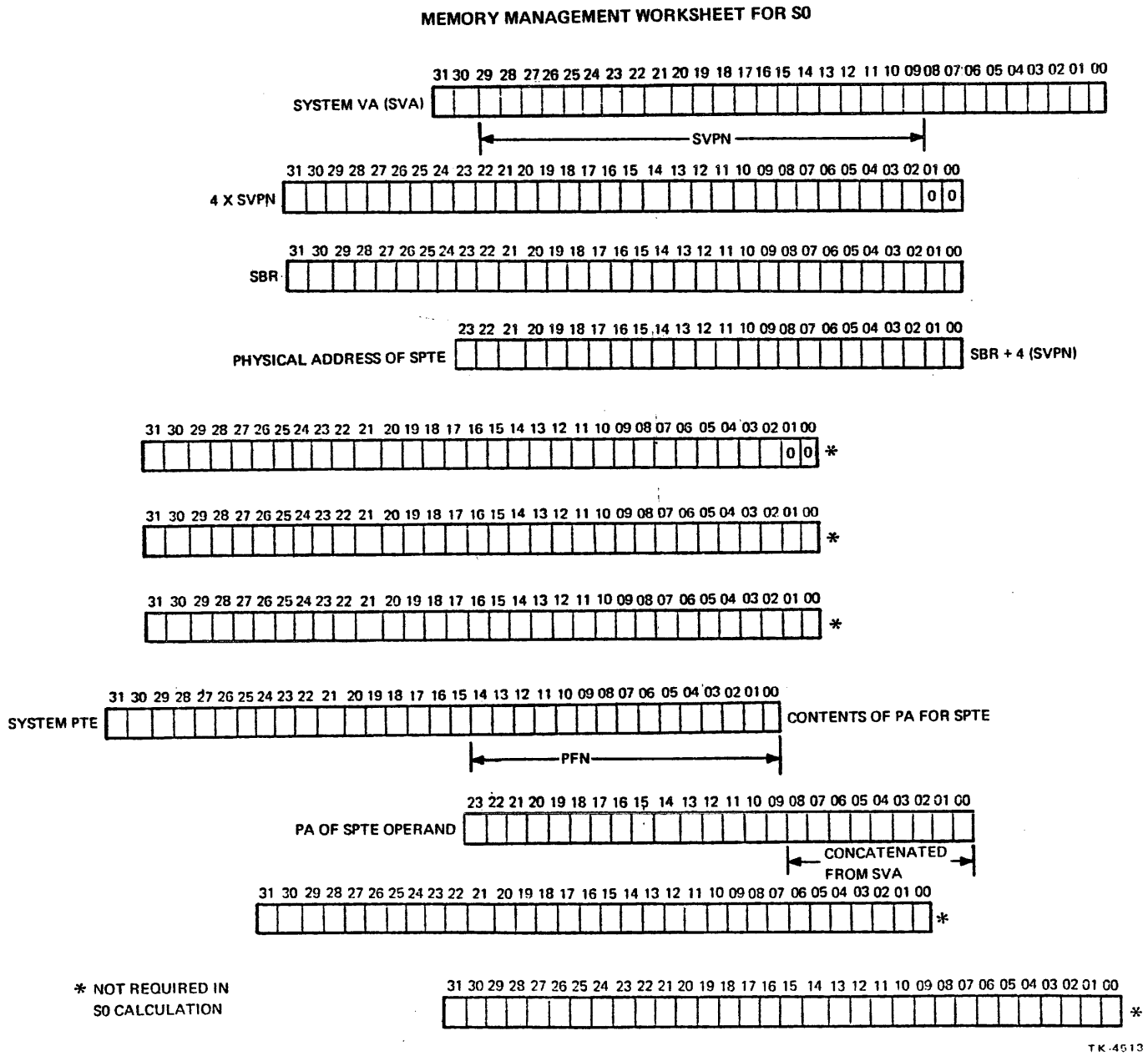


Figure 9-14 Memory Management Worksheet for SØ

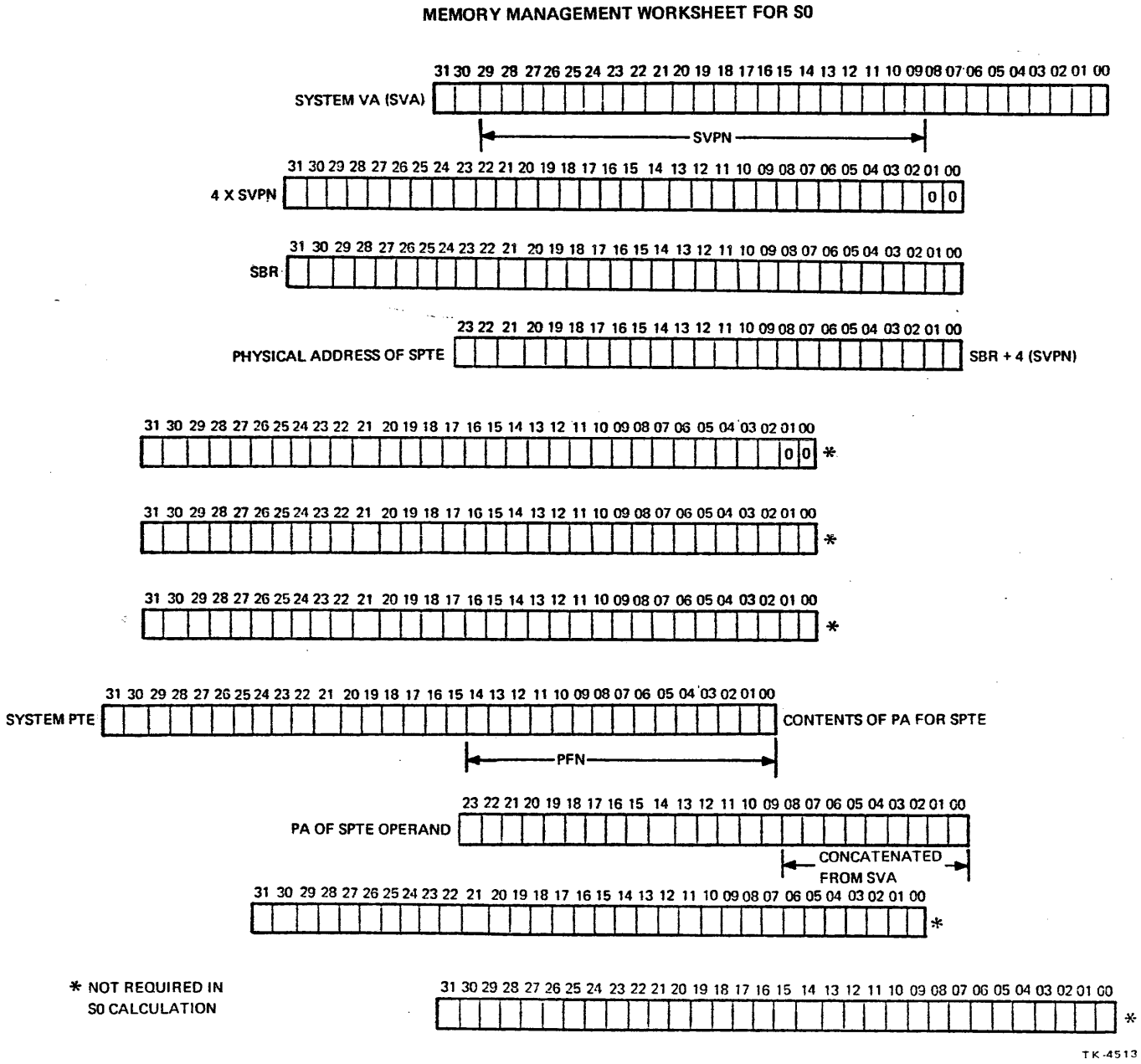


Figure 9-15 Memory Management Worksheet for SØ

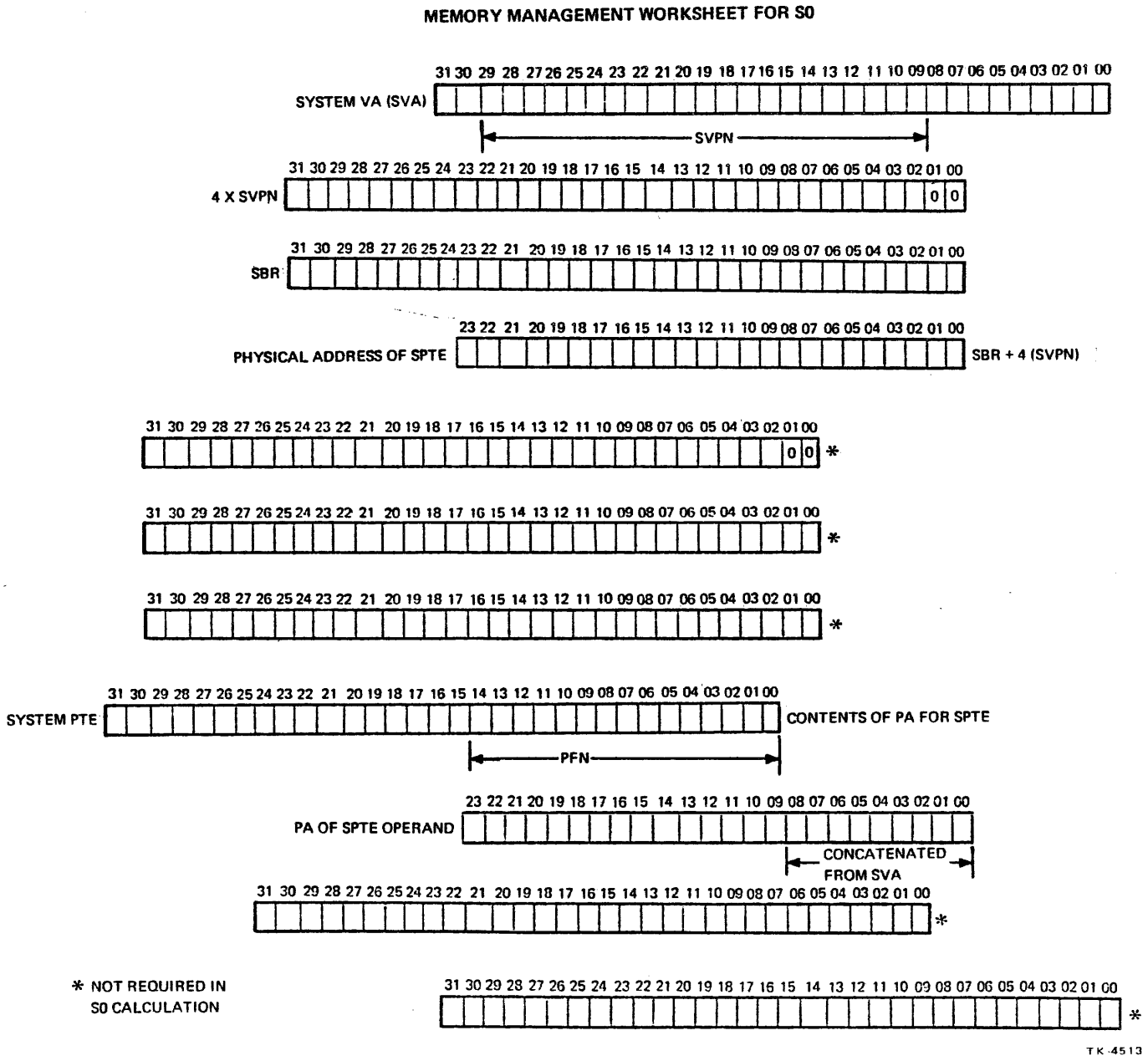


Figure 9-16 Memory Management Worksheet for SØ

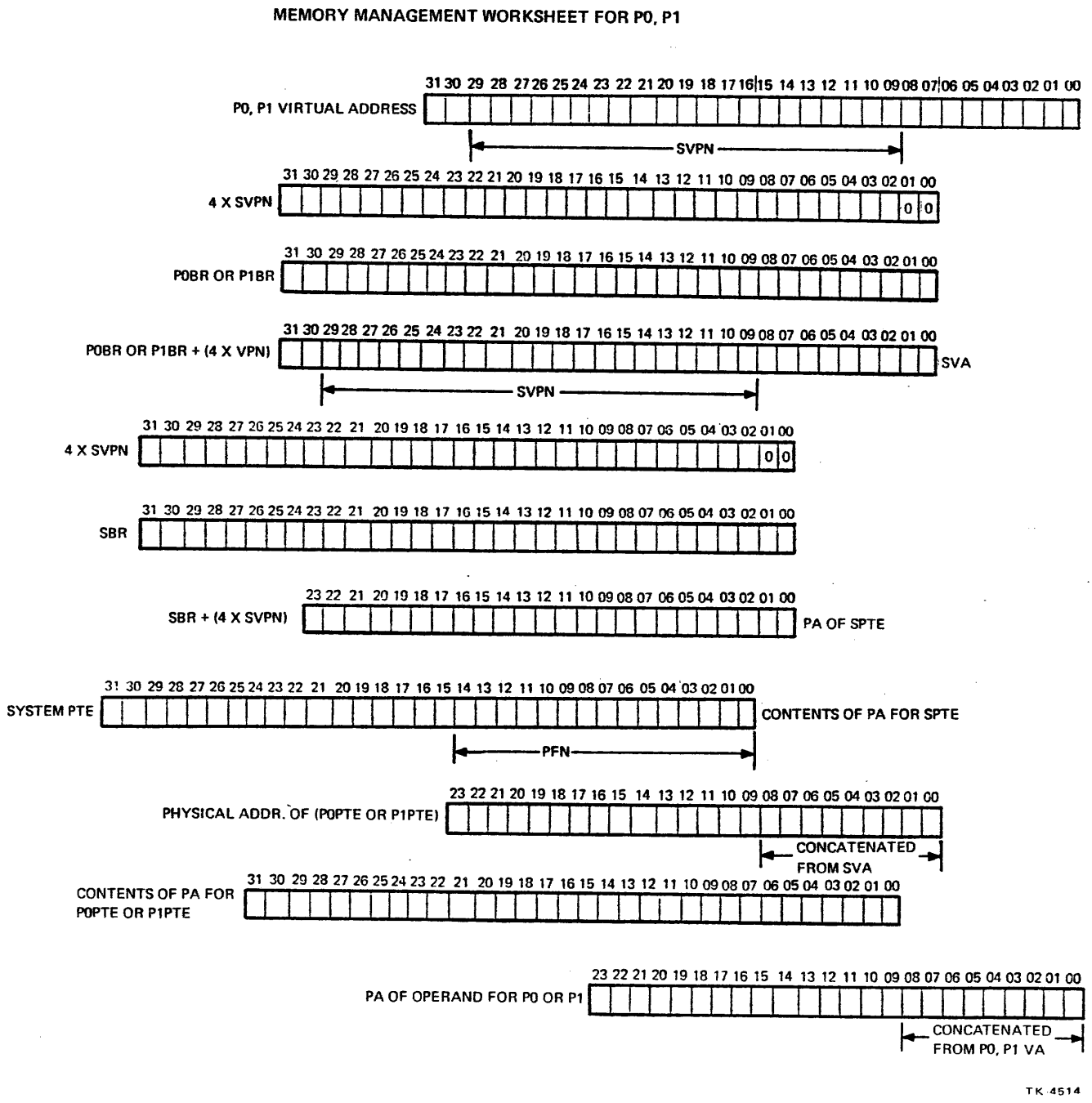


Figure 9-17 Memory Management Worksheet for P0, P1

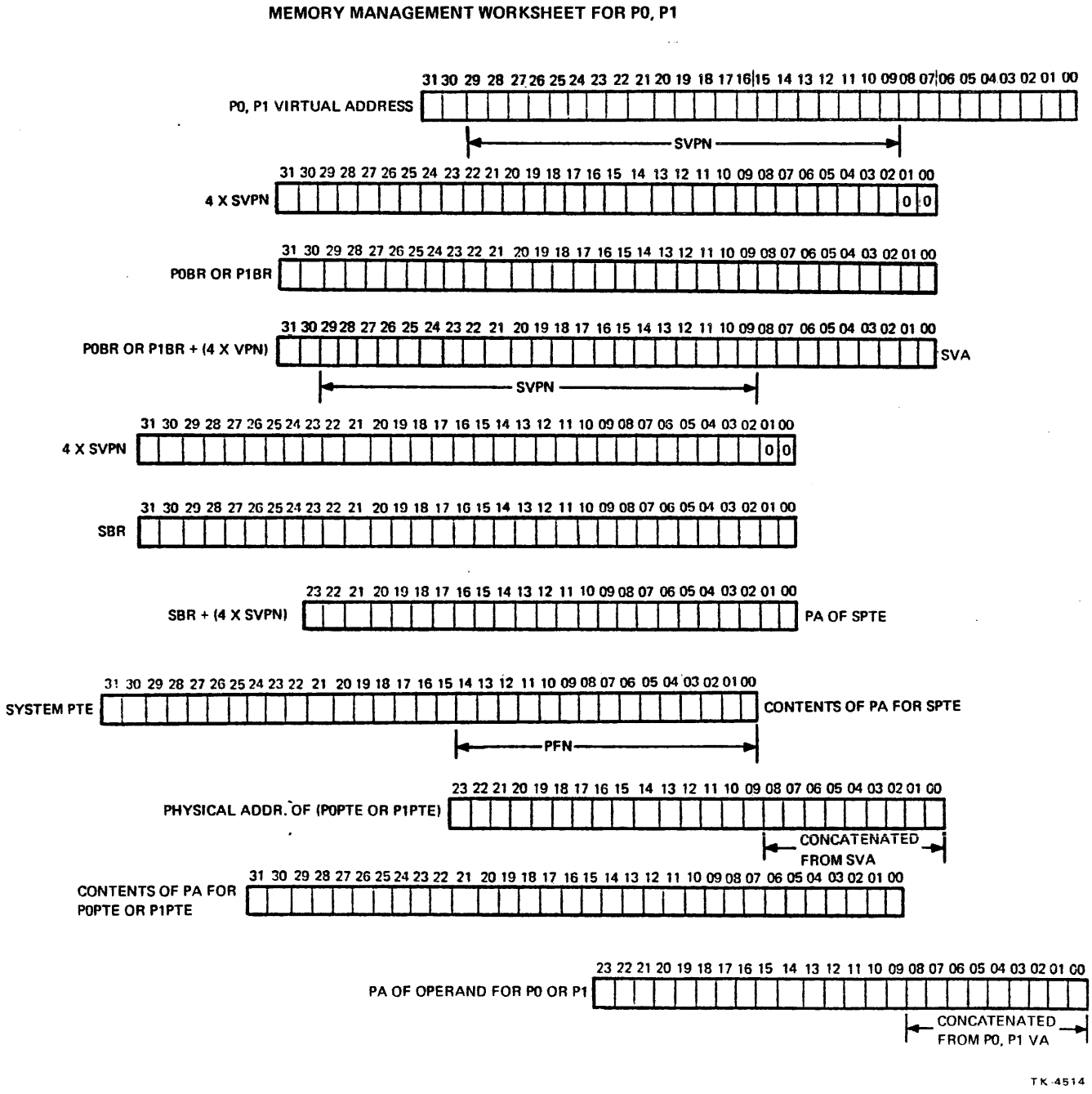


Figure 9-18 Memory Management Worksheet for P0, P1

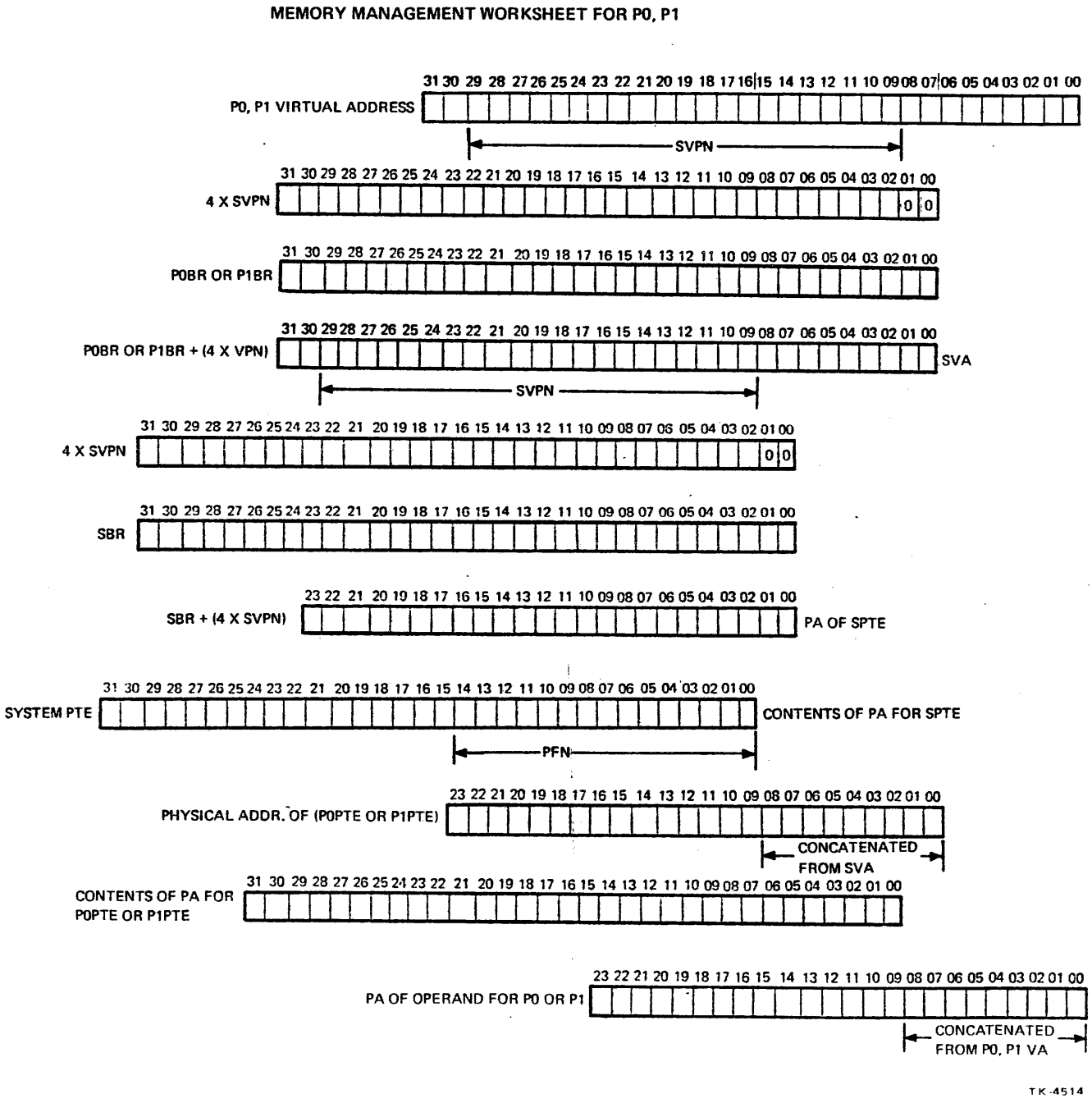


Figure 9-19 Memory Management Worksheet for P0, P1

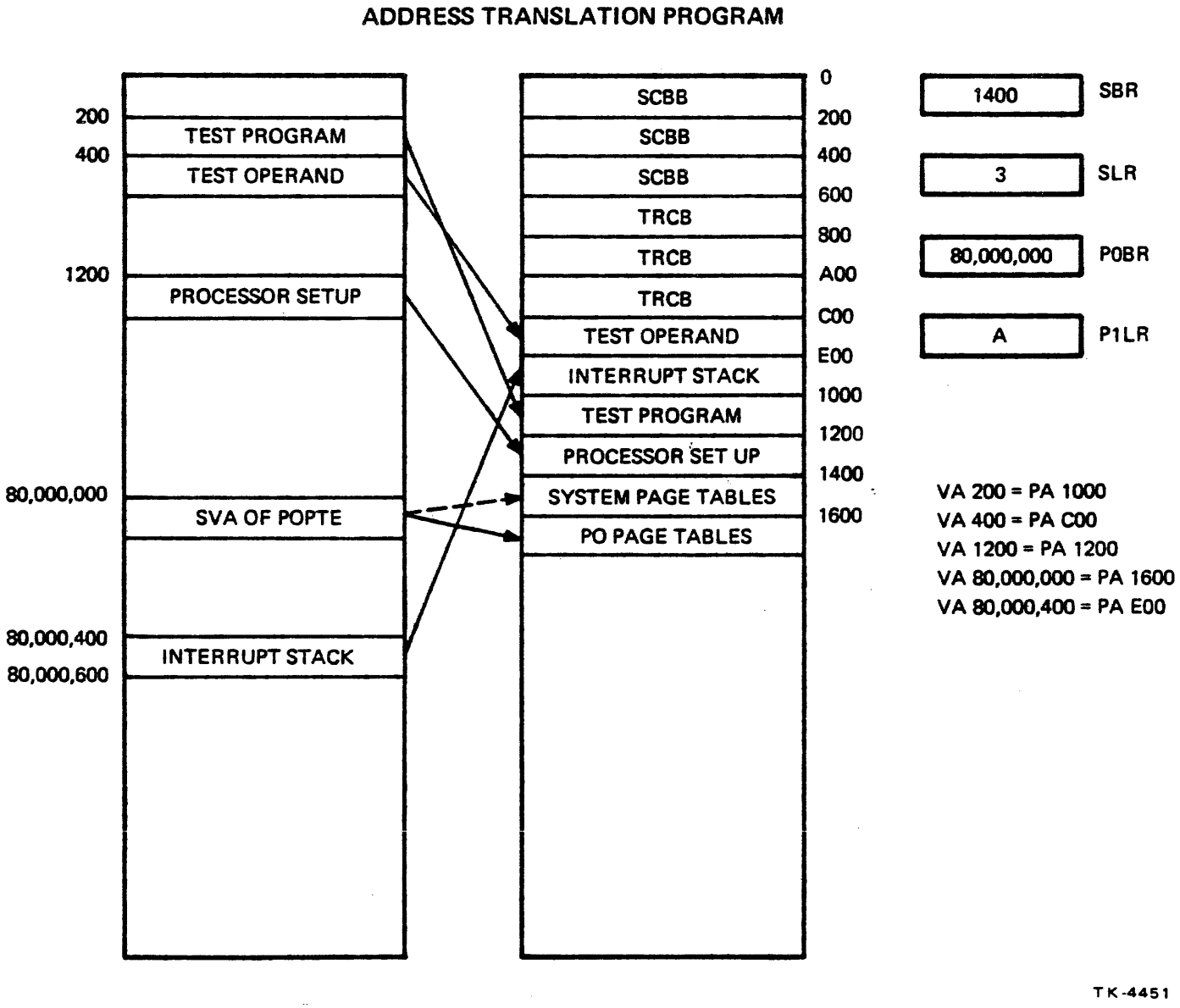


Figure 9-20 Address Translation Program

VAX-11/750 LEVEL II

System Introduction

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

VAX 11/750 LEVEL II

Memory Address Logic

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

Memory Address Logic

INTRODUCTION

The memory address logic is located on the MIC module and consists of four individual address chips that are identical.

The purpose of this logic is to provide virtual address information to the translating buffer and physical address information to the data routing and alignment section.

The function of the logic is to manipulate memory address for:

a. Normal PC
b. Branch offsets
c. Prefetching
d. Microcoded memory references
e. Snapshooting the CMI

MODULE X: MEMORY ADDRESS LOGIC

SYNOPSIS

This module presents technical information concerning the memory address logic of the 11/750 CPU at both the block diagram and chip level to include the following:

a. virtual address registers
b. program counter
c. program counter backup
d. program counter increment

and a fault isolation laboratory exercise.

OBJECTIVES

Identify the major memory address logic components by correctly labeling them on a blank block diagram.

Utilizing the MIC module schematics, trace the signal path for a preselected signal in the memory address logic.

Given a diagnostic printout of a failing 11/750 CPU, isolate the defective memory address logic gate array.

SAMPLE TEST ITEM

In the memory address logic, the signal LATCH MAL is used to:

a. load PC
b. load MA
c. select W Bus as input to address logic
d. none of the above

LAB EXERCISE

a. load microdiagnostics
b. run microdiagnostics
c. interpret error printout
d. isolate malfunction to module
e. isolate malfunction to chip
f. perform appropriate repairs

RESOURCES

11/750 Specifications
11/750 Microcode Listing
11/750 Schematic Drawings

OUTLINE

A. Characteristics

   1. Location
   2. Implementation

B. Purpose

C. Functions

   1. Prefetching
   2. Normal PC
   3. Branch offsets
   4. Snapshooting CML
   5. Microcoded memory references

D. Detailed Description

   1. Eight bit slice
   2. Major sections

E. Add Chip Inputs/Outputs

F. Signal Flow

Address Logic Purpose/Functions

Purpose: provide address information to:

A. Translation Buffer
B. Memory Data Register

Function: manipulate memory address for:

A. Normal PC
B. Branch Offset
C. Prefetching
D. Microcoded Memory References
E. Snapshooting CMI

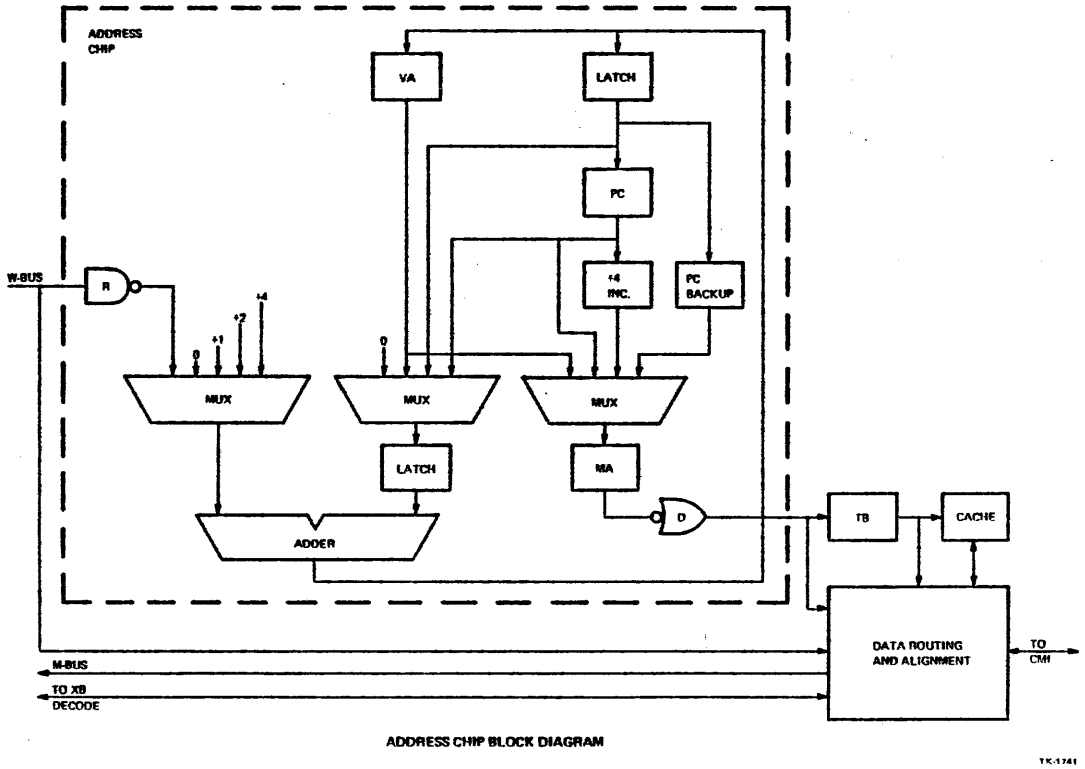


Figure 10-1 Simplified Chip Block Diagram

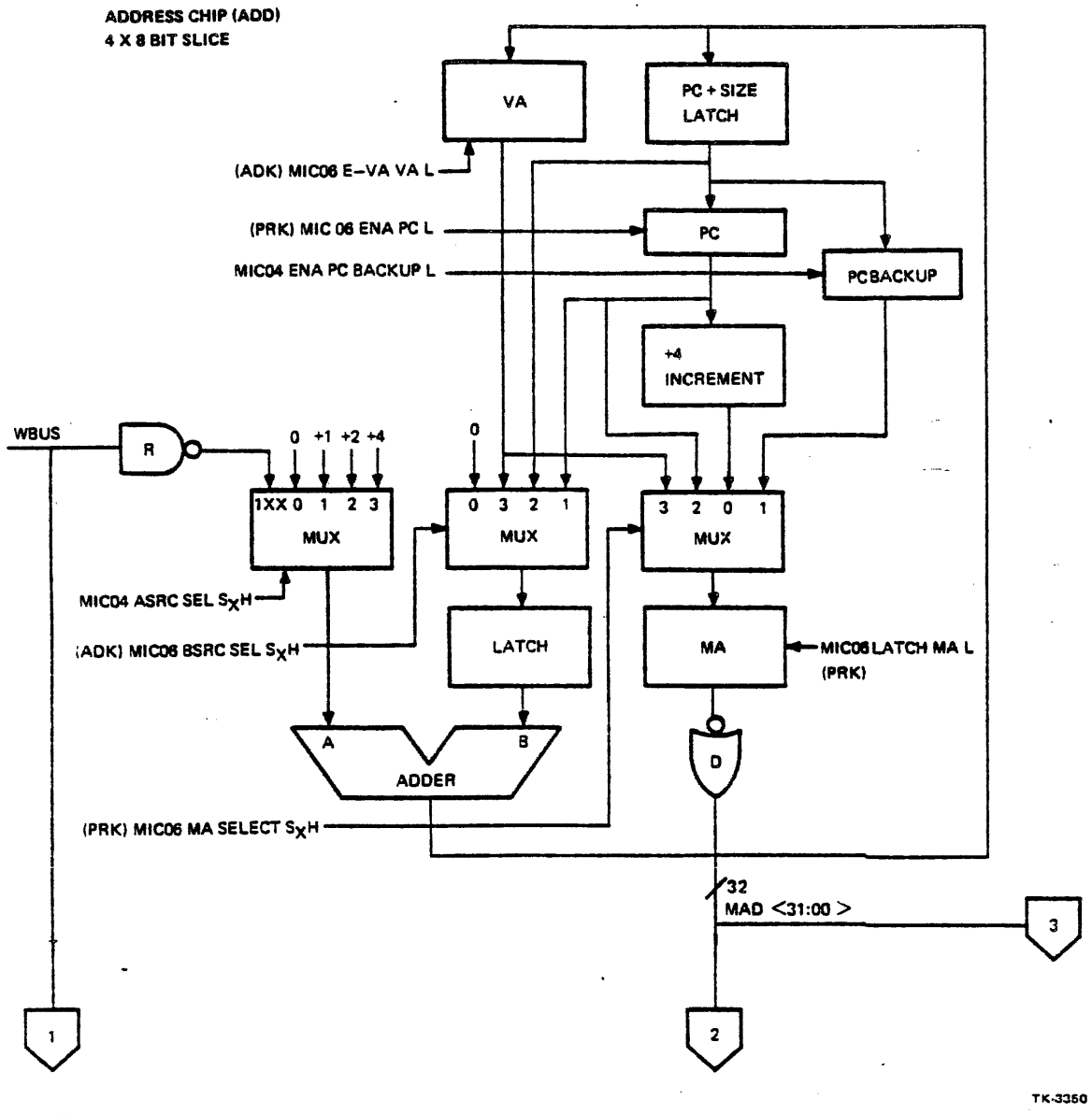


Figure 10-2 Address Chip 4 X 8 Size and Signals

Table 13-1 ASRC, B SRL and MA Select Lines

ASRC <S2:SØ> H TRUE

Controls A MUX Input to ADDER, From MIC 4

S2	S1	SØ	A Input
L	L	L	Ø
L	L	H	+1
L	H	L	+2
L	H	H	+4
H	X	X	WBUS

B SRC <S1:SØ> H TRUE

Control B MUX Input, From ADK Chip

S1	SØ	B Input
L	L	Ø
L	H	PC
H	L	VA SAVE (PC+SIZE)
H	H	VA

MA SELECT <S1:SØ> H TRUE

Controls MUX Input to MA LATCH, From PRK Chip

S1	SØ	MA Input
L	L	INCREMENTER
L	H	PC BACKUP
H	L	PC
H	H	VA

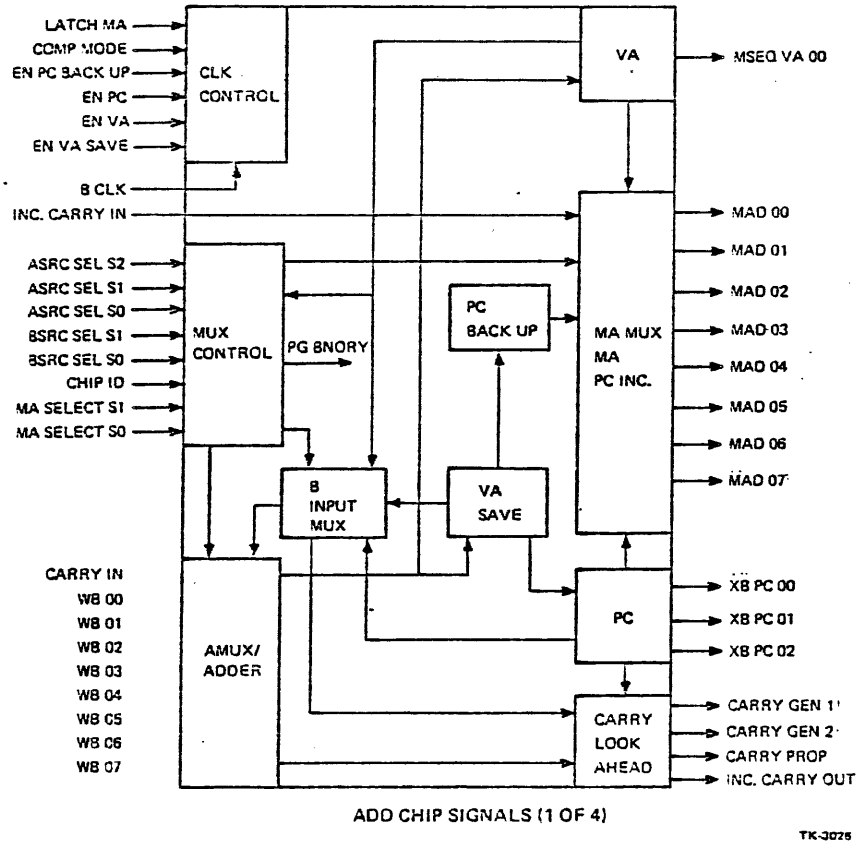


Figure 10-3 Address Chip Signals In/Out

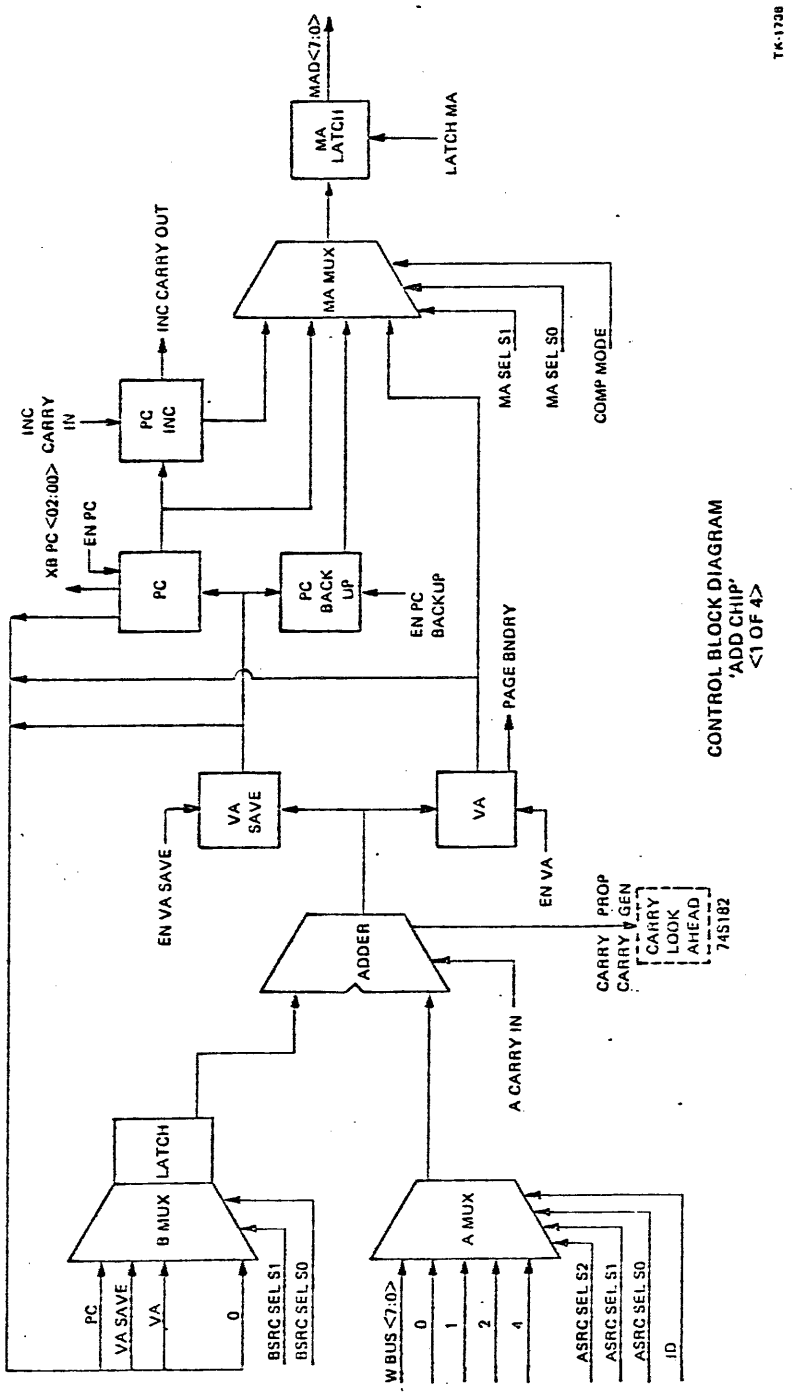


Figure 10-4 Address Chip Control Block Diagram

As you look at the Memory Interconnect Module [MIC] you must first look at all of its basic funtions as written in 2.6 of the manual and attempt to put them in perspective. To do this one of the ideas you must maintain is that the MIC module only functions in two basic fashions.

1. performs micro coded orders
2. monitors non micro coded functions
   a. prefetch needed I-STREAM data
   b. monitor micro trap conditions

Let's look at these two functions in an overview separately.

1. Performs Micro Coded Functions

Micro coded functions are functions that the MIC performs under direct control of the different micro fields W control, bus function and M source that are sourced from the control store module for a specific micro address. Some examples of these are:

a. read or write to memory [or an I/O device]
b. source data from the MDR to the M bus
c. probe translation buffer for access violations
d. write to status and control registers on MIC

Since these functions are coming from the micro word and use some of the same circuitry you must realize that not all functions can be performed at once. The MIC decodes one micro instruction at a time for its needed fields. [W control, M source, and bus function]. You cannot tell the MIC to read from memory in the same micro instruction that tells it to load the virtual address with data from the WBUS. For example: it takes at least four micro words to perform the following macro instruction. MOVL (R1),(R2).

2. Non Micro Coded Functions.

Some of the non micro coded functions relate directly to the micro coded functions, such as; while the MIC is reading from memory data in address 1000 it is possible that memory management is enabled. This would have been enabled by writing to the MME status and control register (a micro coded function). Once it is enabled the MIC will monitor for translation buffer hits, misses and access violations. This will be performed independently of the microed function of read. This monitoring function is due to the access control violation chip [ACV] and the micro trap chip [UTR]. These chips are constantly monitoring for hits, misses and

access violations when MM is enabled. If any improper conditions are found during a micro coded function a micro trap is performed to place the proper micro address on the micro address lines that places the machine in the proper routine to handle the improper condition.

That is only one example of monitoring error conditions. The ACV chip is also constantly monitoring the control store parity condition in the machine for parity errors.

Another non micro coded function is to fetch I-STREAM data for use by the processor independent of the micro code. To do this the MIC must first load the execution buffers with the needed data to start with. This is done by loading the "PC" with an address thus causing the condition of "flushing the execution buffer". Flushing causes the MIC to take the address in the "PC" and reading from memory two longwords and storing them into the execution buffers [XBØ,XB1]. The I-STREAM data is then constantly monitored by the prefetch control chip [PRK],UTR using the updated "PC" to see if one of the XBs are empty and need to be refilled with another instruction from memory. If this occurs the MIC will perform a non micro coded function called prefetch. This causes the MIC to generate a non micro coded read to memory (or cache) to keep the XBs full for use by the processor. This prefetch function cannot be performed at the same time as a microcoded read as they use the same data path on the MIC.

This brings up another non micro coded function of the MIC. That being, what happens if the instruction being executed asks for data from memory (via the virtual address register VA)? This data is not I-STREAM data but data needed by the operand to complete the instruction. This data is not stored in the XBs and may not be cached so it might have to be fetched from memory. Now this takes more time than the micro word takes to execute, so it is possible that the next micro word from the control store says get the data I just asked for and it may not be available yet. If this happens at this time the MIC would generate a stall condition to the DPM module stopping the micro word from performing the function until the data that was asked for was fetched and stored for its use.

> **NOTE: The MIC didn't automatically generate the stall condition when it went to memory to fetch the needed data.**

Only when the micro word needed the data and it wasn't available did the MIC generate the stall condition. This

stall condition, access violation checks, cache and translation buffer hits and misses are monitored for both prefetches and micro coded memory references.

VAX-11/750 LEVEL II

Translation Buffer

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The location of the translation buffer is on the MIC module. The purpose of the translation buffer is to store PTEs for address translation and access rights.

Its function is to provide the 15 most significant address bits which correspond to the 23 most significant virtual address bits.

MODULE XI: TRANSLATION BUFFER

SYNOPSIS

This module is designed as a block diagram and schemetic level analysis of the following:

a. characteristics
b. inputs/outputs
c. microroutines
d. PTE formats
e. tag and index formats
f. parity checking

Also included is a fault isolation laboratory exercise.

OBJECTIVES

Identify the major translation buffer logic components by correctly labeling each on a blank block diagram.

With a malfunction inserted on the MIC module of the Comet CPU, utilize all available documentation, diagnostics and test equipment to isolate the malfunction to the chip level.

Given statements concerning the translation buffer, correctly select the major components described from a list of components, writing the answer in the space provided.

SAMPLE TEST ITEM

The translation buffer is a/an _____.

a. instruction decoder
b. two way set associative memory
c. physical address generator
d. none of the above

LAB EXERCISE

a. load microdiagnostics
b. run microdiagnostics
c. interpret error printout
d. isolate malfunction to module
e. isolate malfunction to chip
f. perform appropriate repairs

RESOURCES

Comet Specifications
Comet Print Set
Comet Microcode Listing

OUTLINE

A. Purpose

B. Characteristics

   1. Type
   2. Location
   3. Interfaces with

C. Function

   1.
   2.

D. Translation Buffer Organization

   1. Tag
   2. Data Stores
   3. Physical Dimensions

E. Simplified Block Diagram

   1. 2-way set associative cache
   2. VA bit grouping
   3. Operation
   4. Parity
   5. Control

OUTLINE (continued)

F. Address Translation

G. Detailed Block Diagram Description

   1. Read cycle
   2. Write cycle

H. Microroutines

   1. Invalidate
   2. TB Miss (Read Cycle)

I. Registers

Translation Buffer

Purpose: To store Page Table Entries (PTEs) for address translation and access rights.

Function:
1. Provide 15 most significant physical address bits corresponding to the 23 most significant address bits of a virtual address to the Physical Address Bus (PA).
2. Provide 4 access conrol bits and modify bit to Access Control Violation (ACV) chip and utrap (UTR) chip for control functions.

Characteristics: 2-way set associative cache mode up 2K discrete RAMs on Mic module. Cache similar to 11/70 in that divided into Tag and Data store with each containing 518 locations.

The translation buffer (TB) is transparent to the microcode. That is, when a regular read or write function is being performed the microcode does not determine whether TB is enabled or disabled. This is done by the Memory Management Enable bit. Bit 0 in S/C Register # 0 in ADK chip (IPR Register # 38 bit 0) by enabling this bit you do two things:

1. Change output of AMUX select from ADK chip to say S1 Low/S0 Low instead of S1 High/S0 Low. This allows only lower byte of MAD lines to Physical Address Lines (PAD) to be concatenated with output of TB data store for 24-bit physical address when DBUS select is cache to DBUS.

2. Allows outputs from ADK for enabling errors to be checked from TB and allows passing of data from TB data to PAD.

To look at the TB and see how it works, we must realize some basic facts.

1. There will always be an address inputted to the TAG store with or without MME. Whatever is input will cause an output that will be checked for bits and Parity without enabled control signals. Whatever signal (hit or error) will not be used unless MME is set and TB Parity Enable generated from ADK chip.

2. The TB data store is always receiving the MAD lines also and will always output whatever it has that relates to the address bit not allowed out to the PAD unless "TB output En" generated by ADK chip when MME. Data Parity is detected but not used by utrap chip unless "TB Parity Enable" generated by ADK.

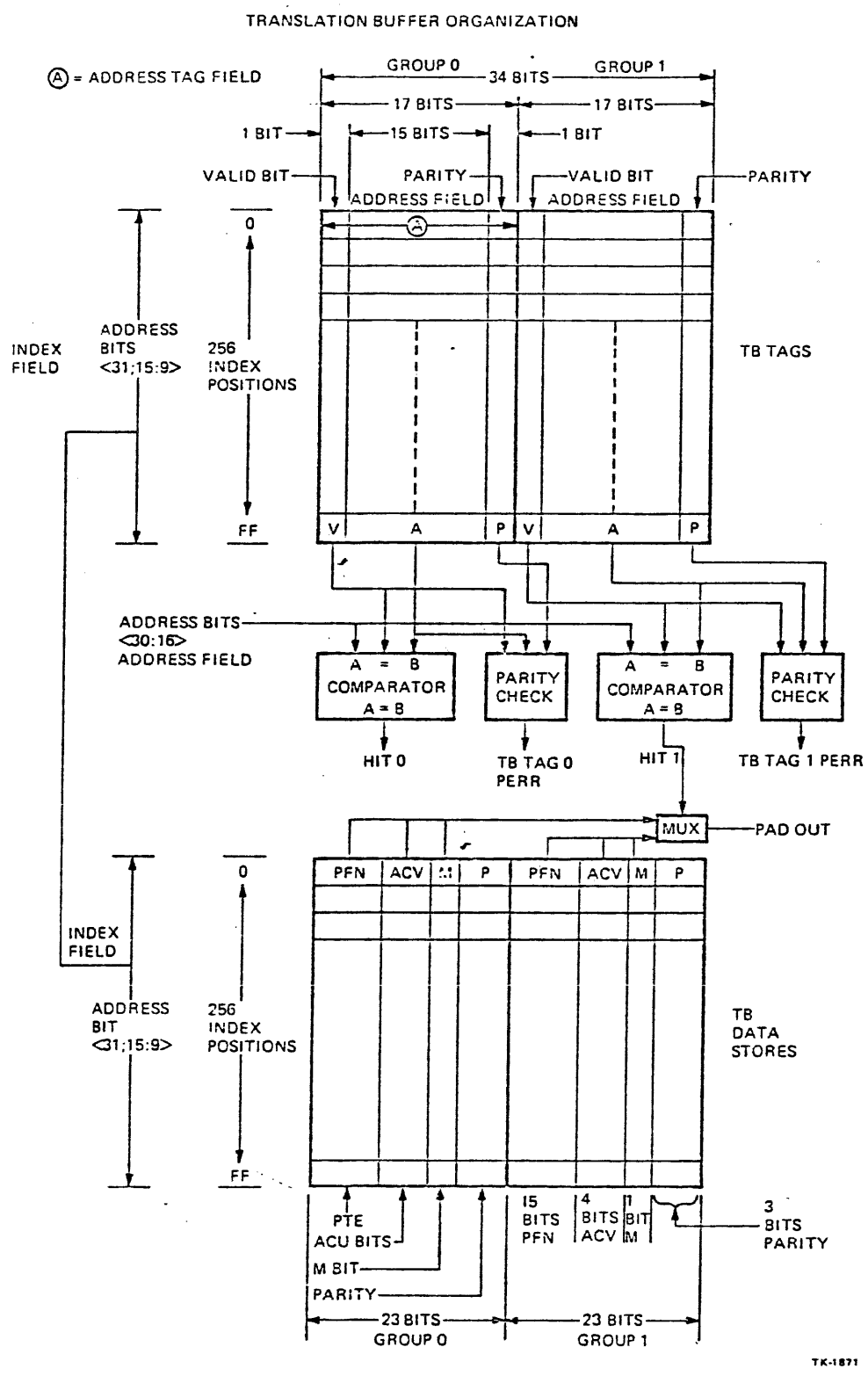


Figure 11-1 Translation Buffer Organization

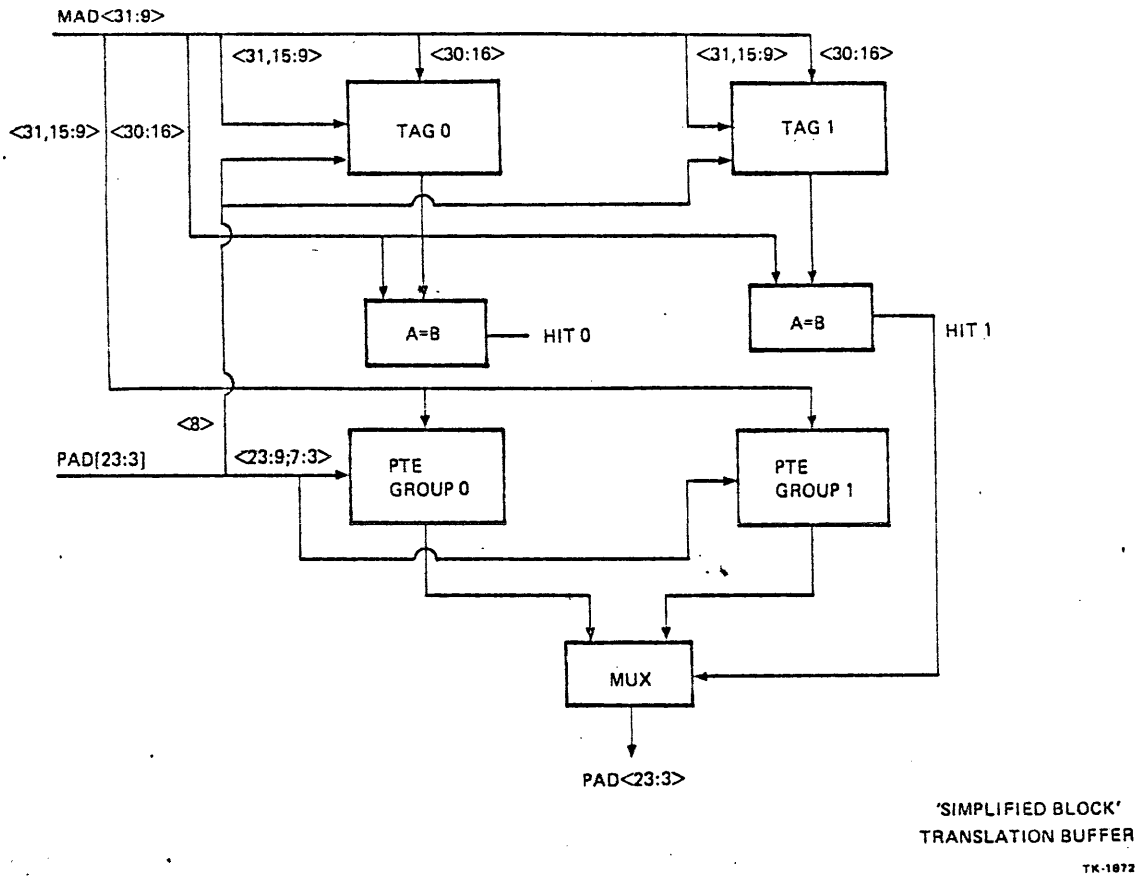


Figure 11-2 Translation Buffer Simplified Block

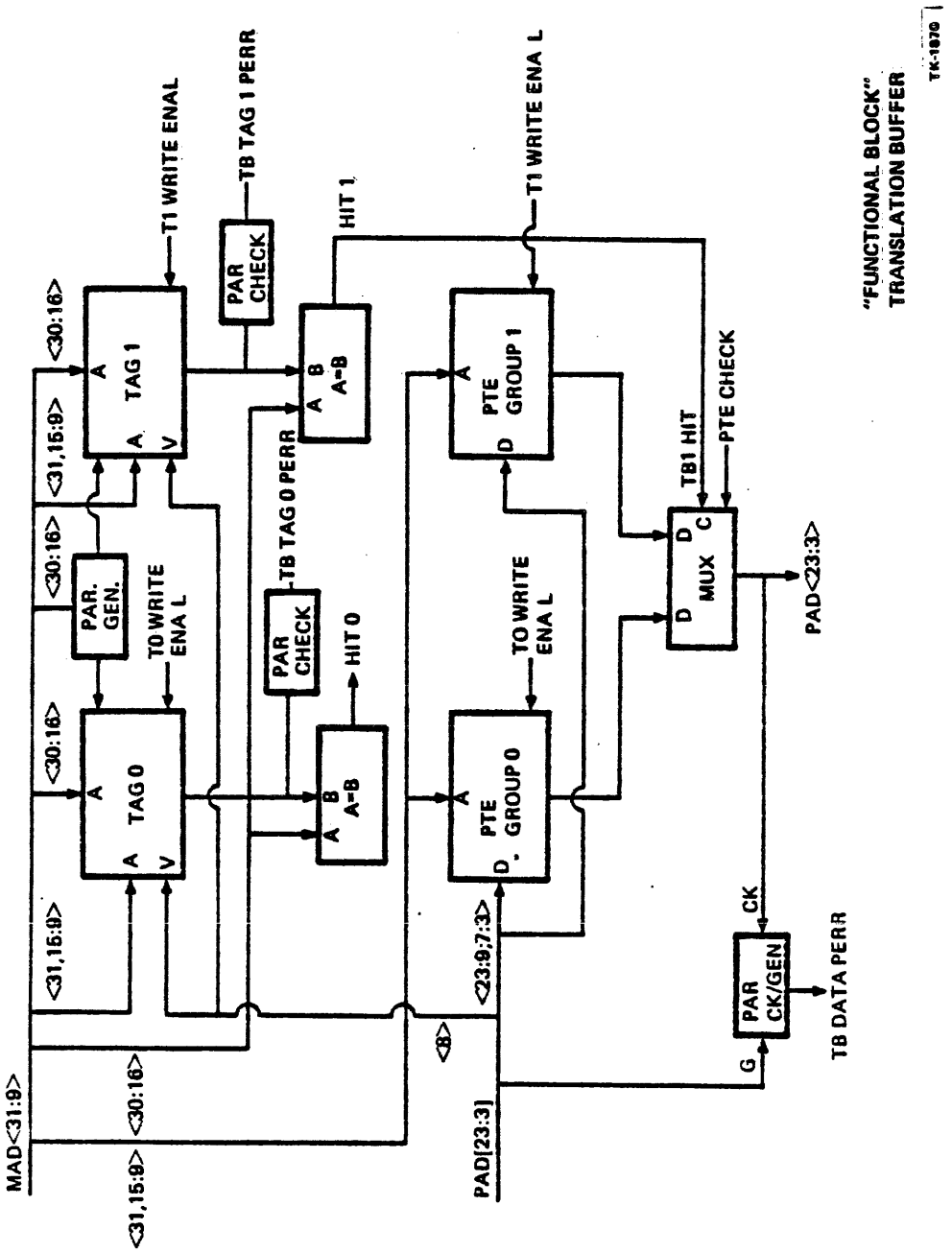


Figure 11-3 Translation Buffer Functional Block

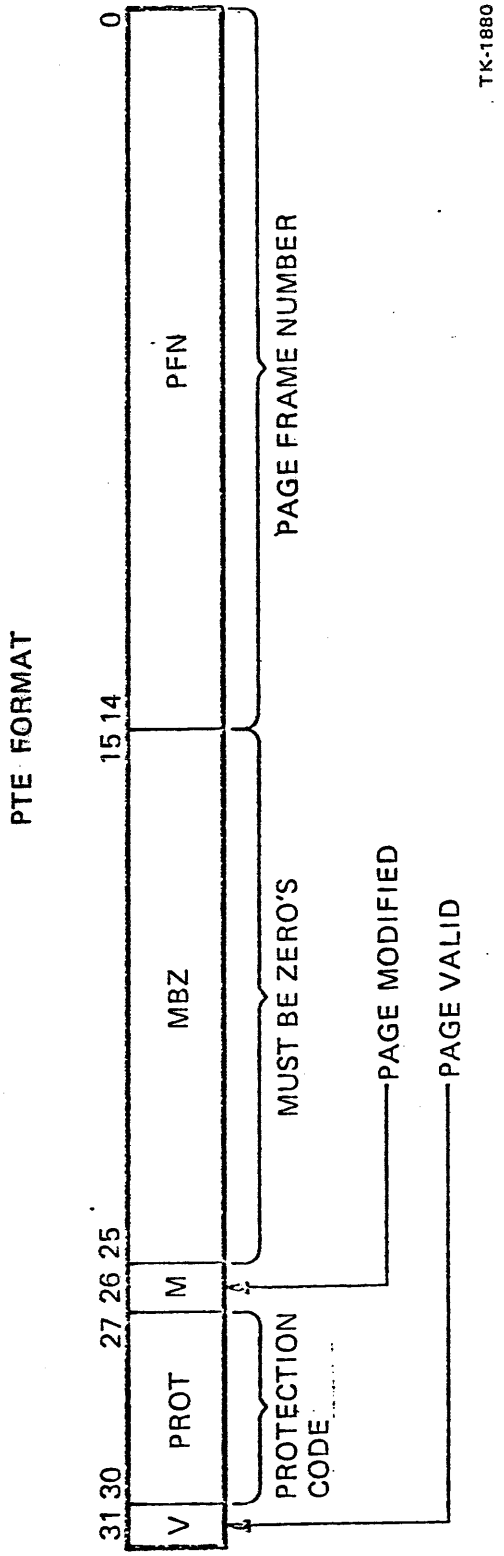


Figure 11-4 PTE Format

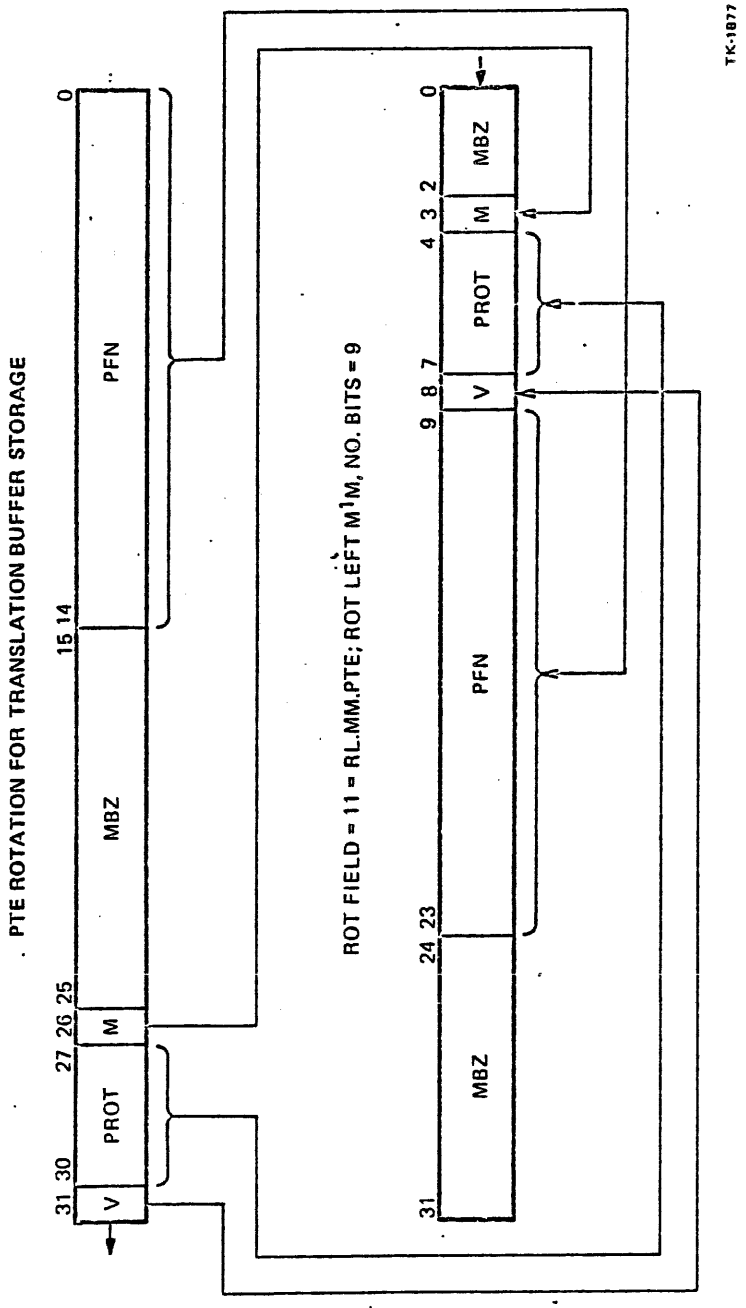


Figure 11-5 PTE Rotation for Translation Buffer Storage

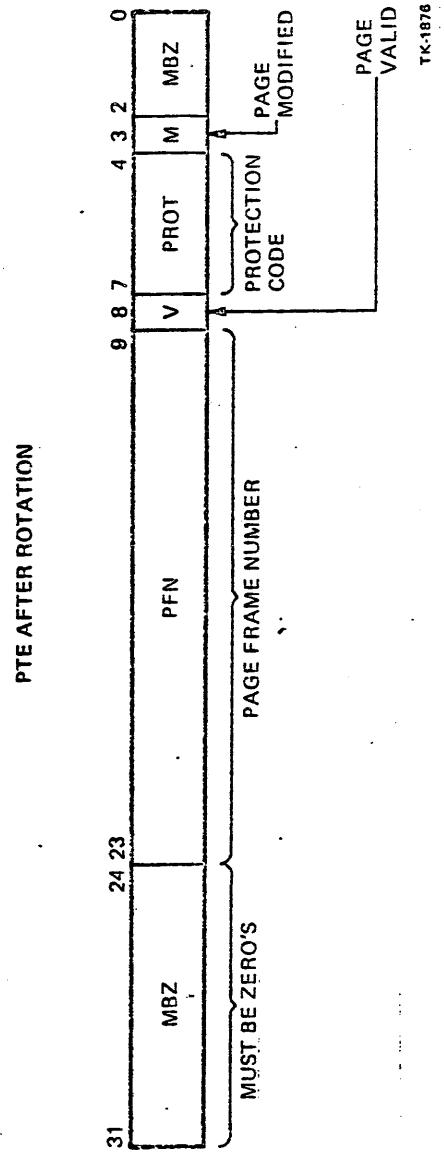


Figure 11-6 PTE After Rotation

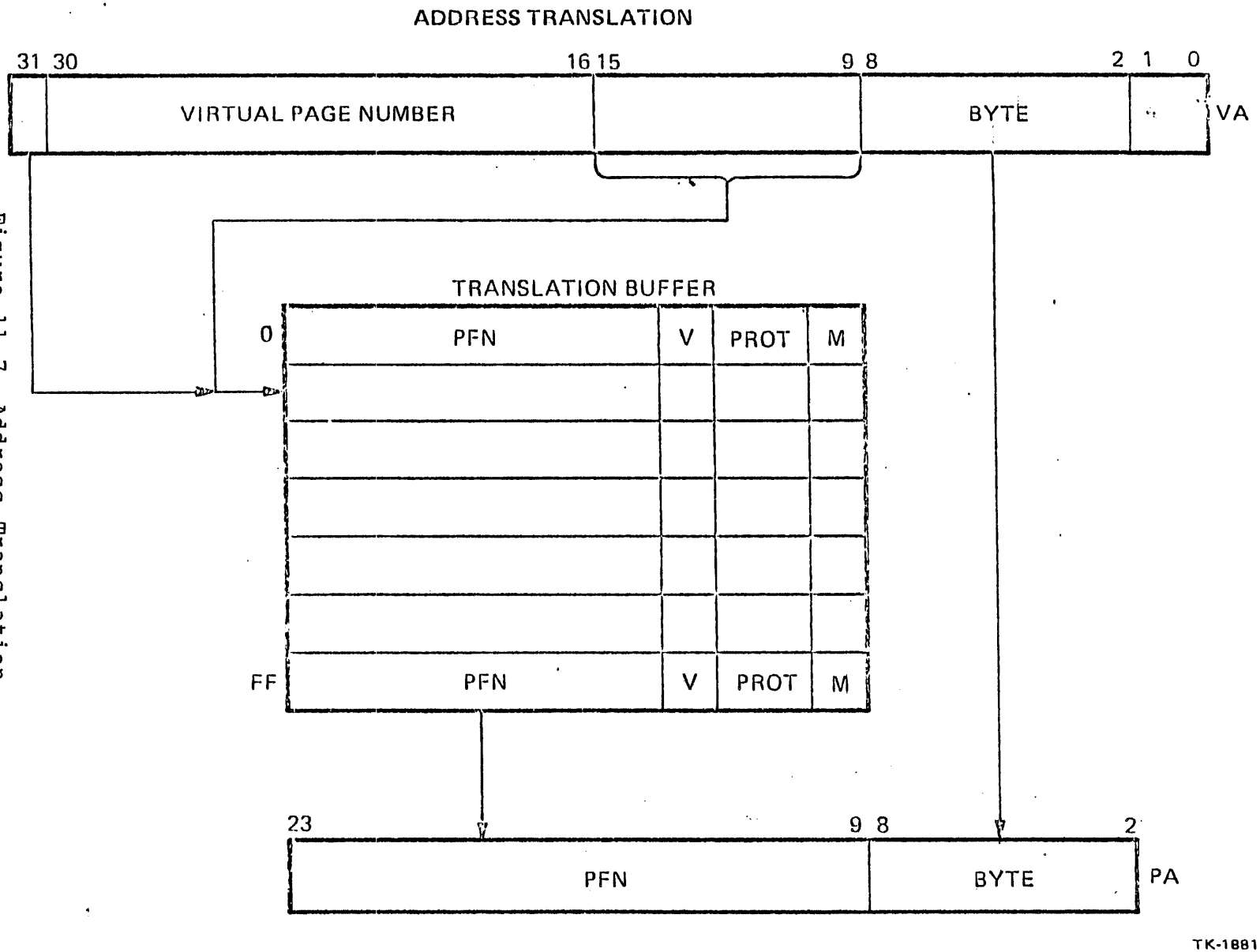


Figure 11-7 Address Translation Buffer

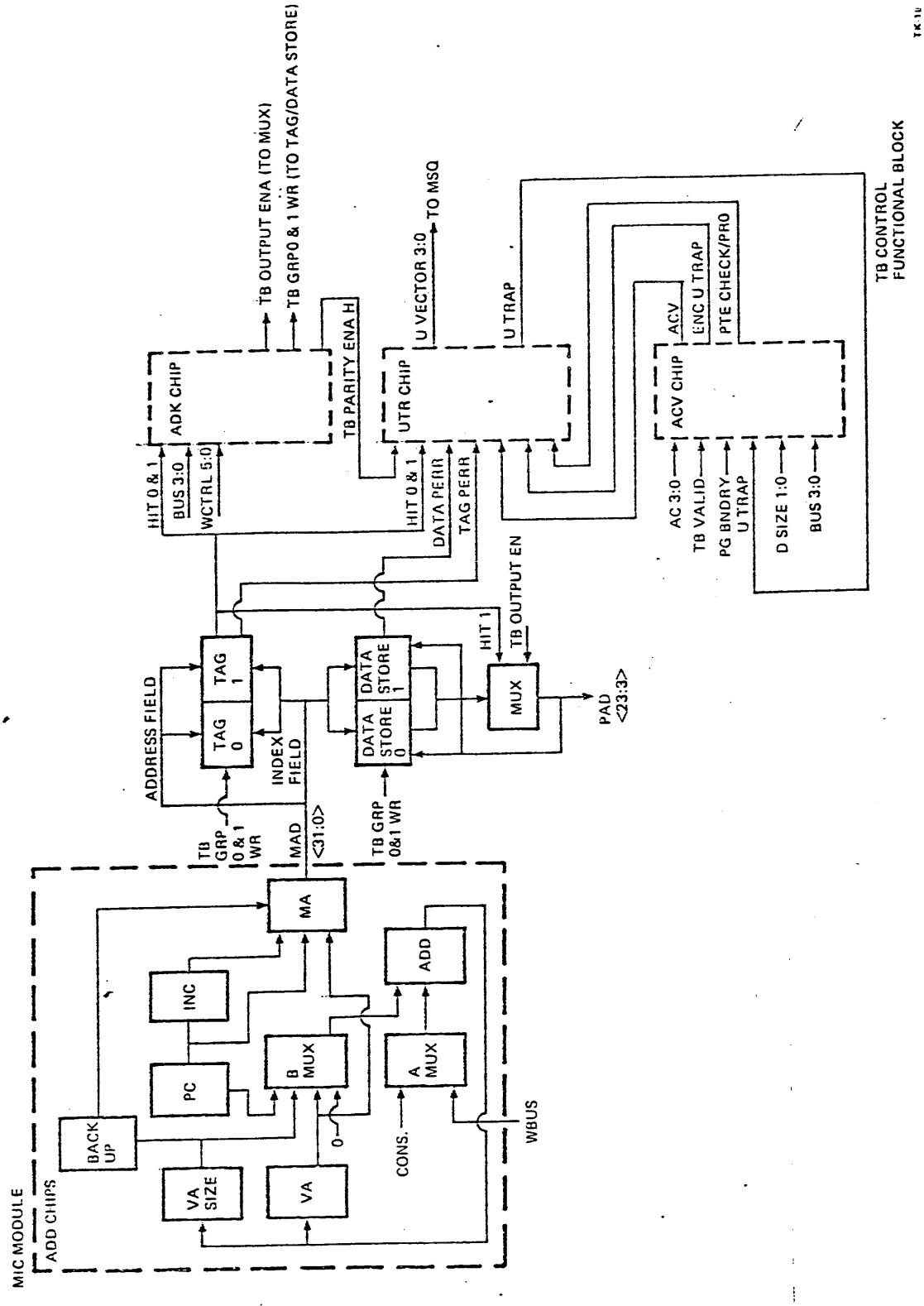
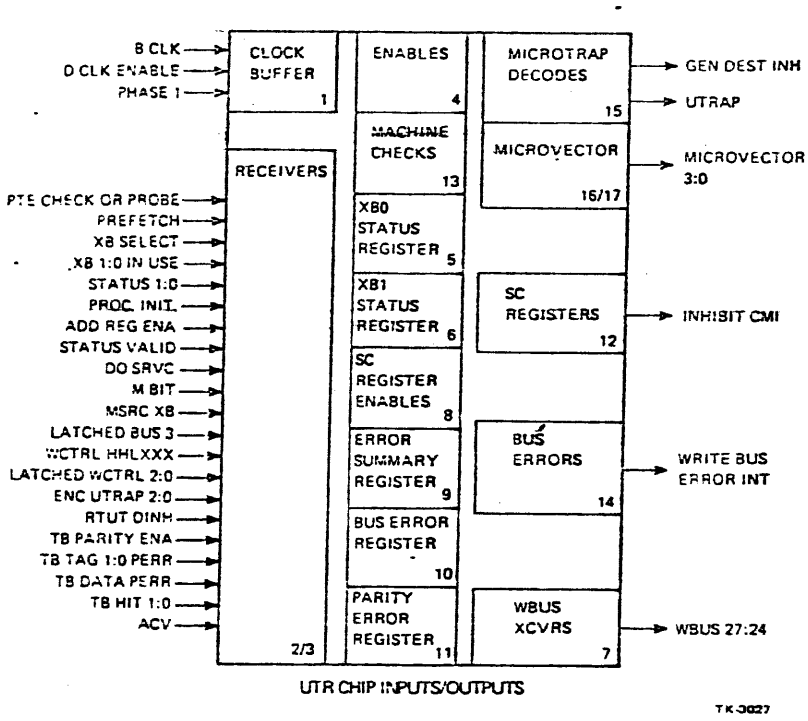


Figure 11-8 Translation Buffer Control Functional Block



***Note:** Some signals are sent to UTR chip even if memory management is not enabled, but are not checked unless "TB Parity Enable H" is sent from ADK.

Figure 11-9 UTR Chip Inputs/Outputs

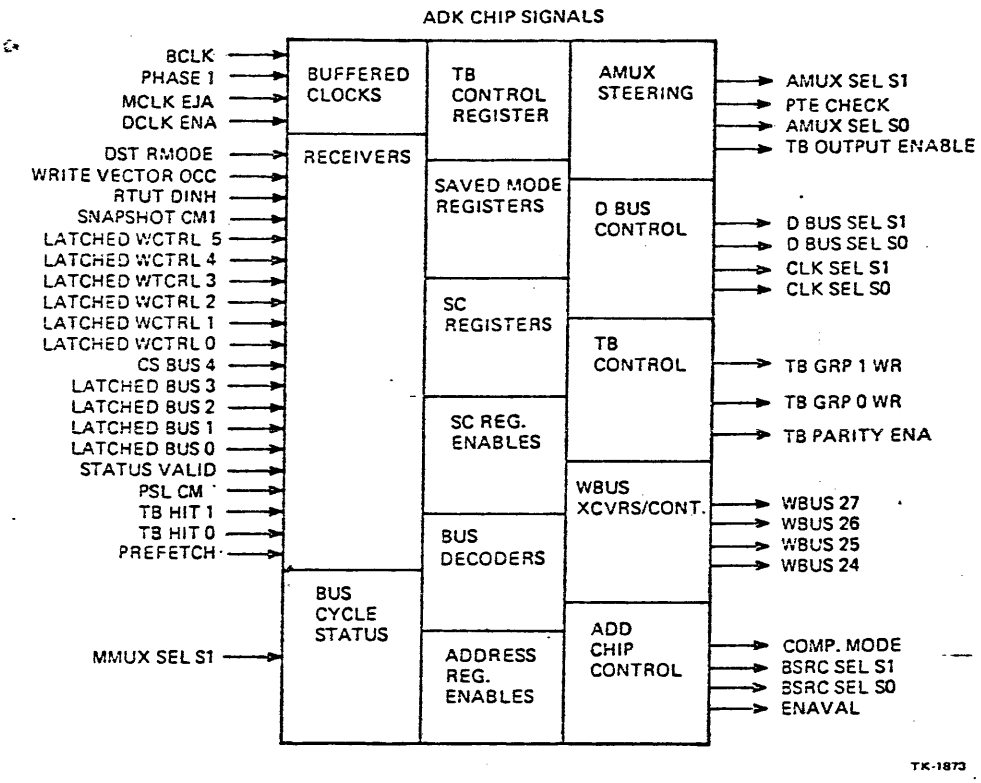


Figure 11-1Ø ADK Chip Signals Input/Output

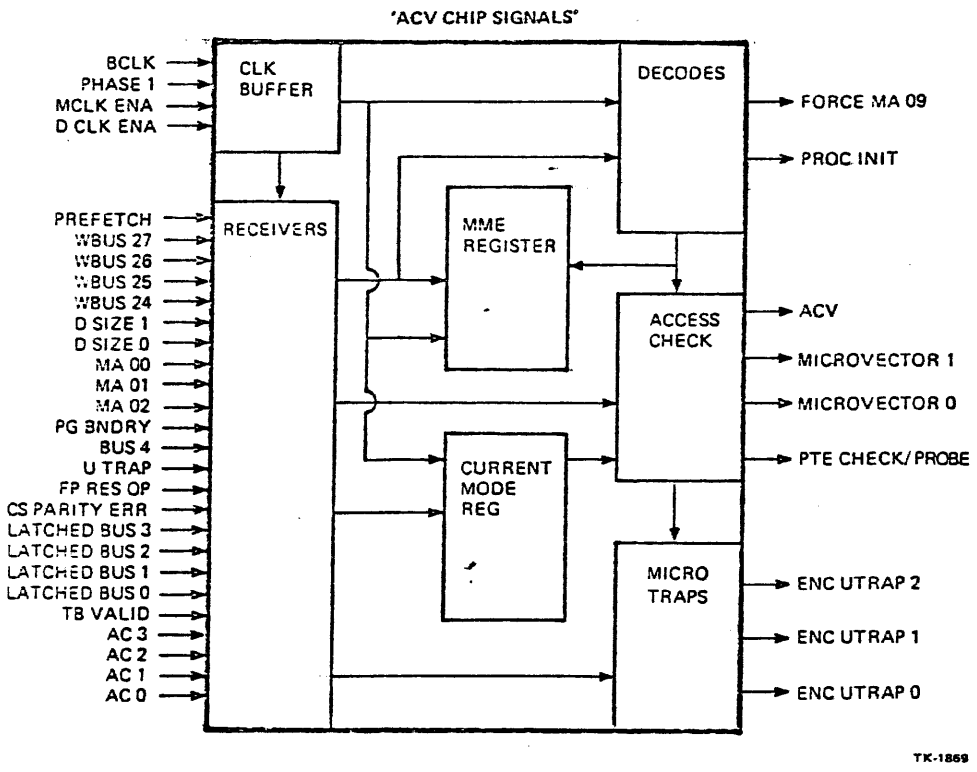


Figure 11-11 ACV Chip Signals Input/Output

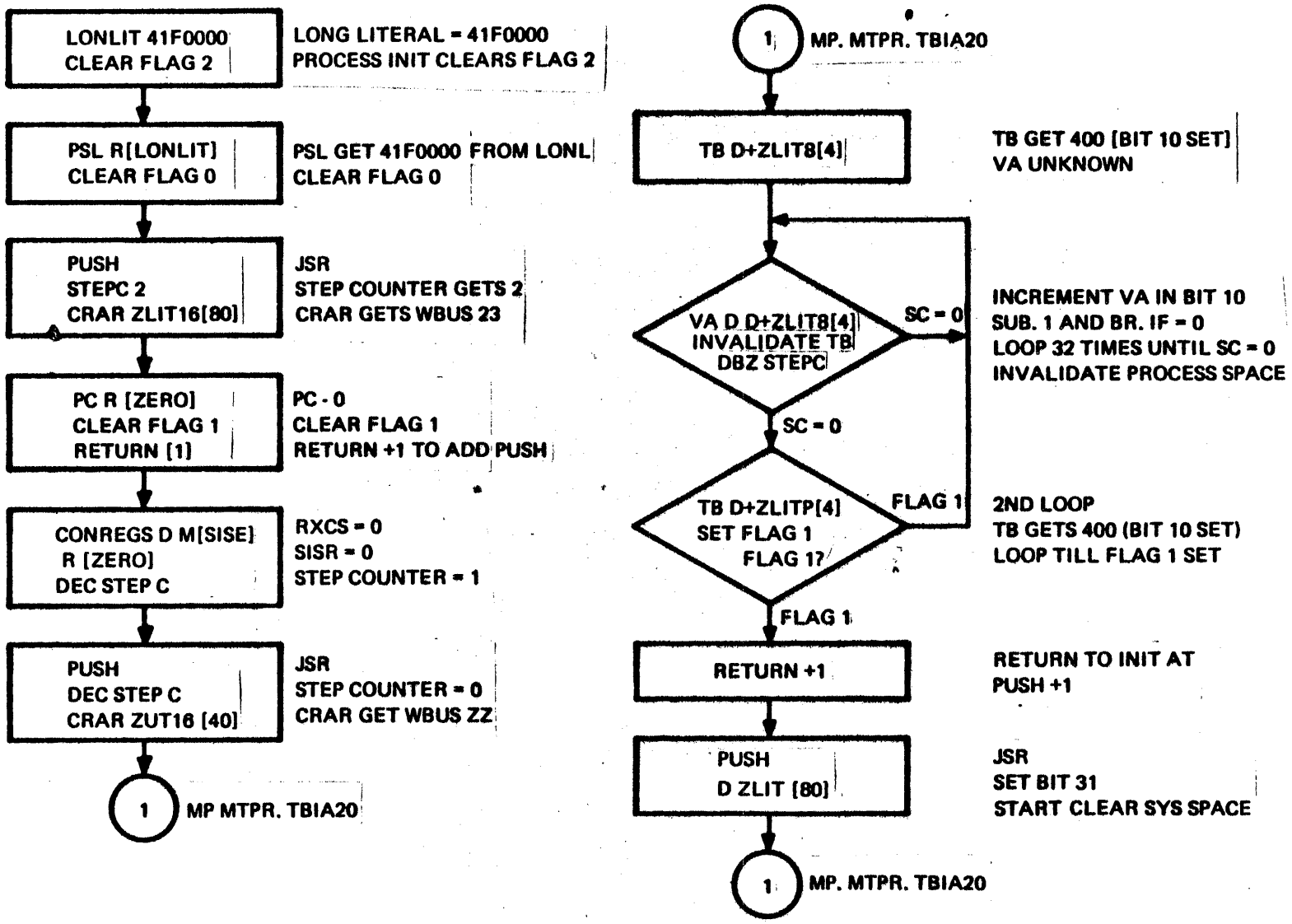


Figure 11-12 Input Routine <CLEAR TB>

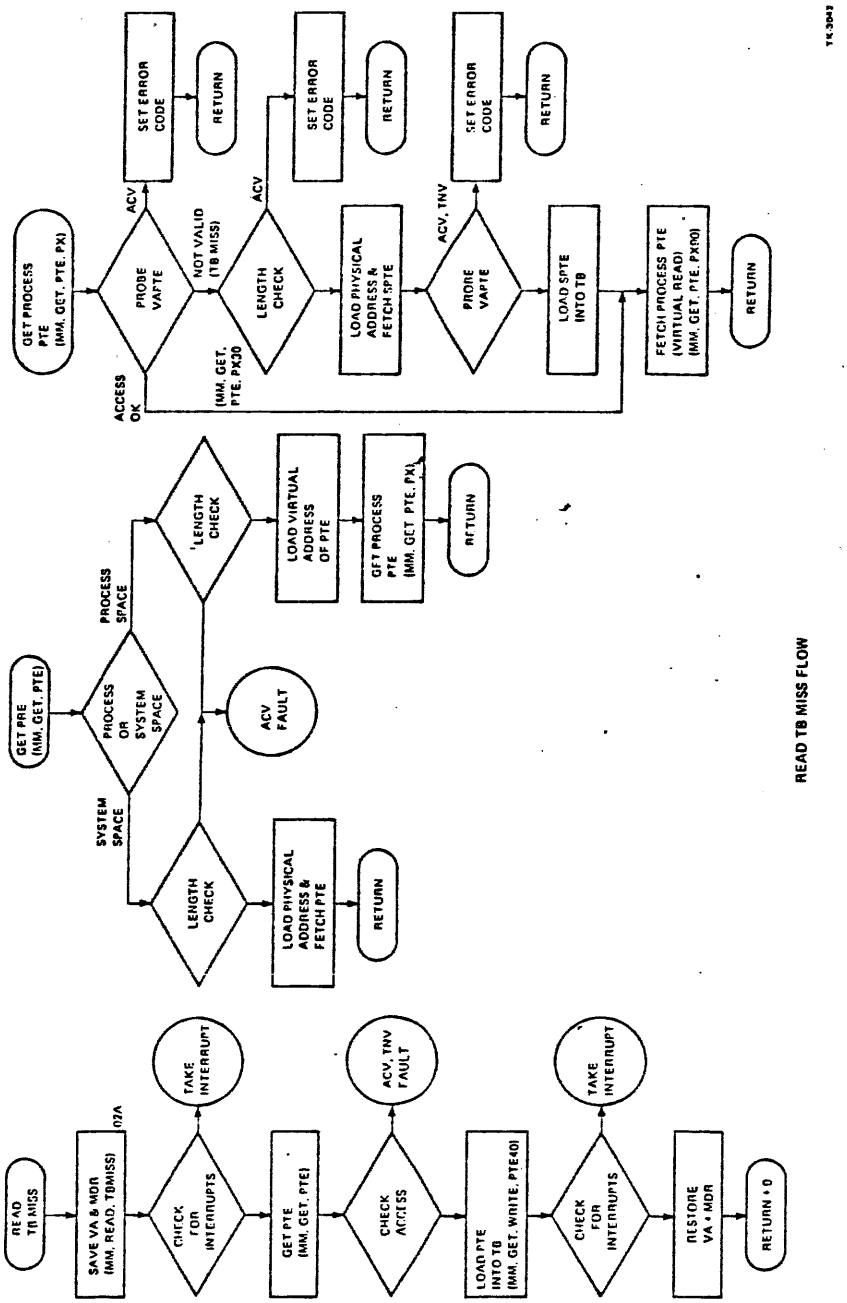


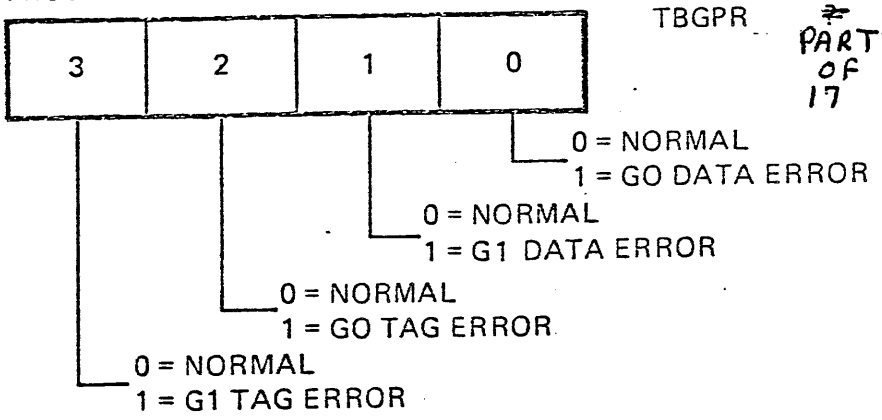
Figure 11-13 Read TB Miss Flow

TRANSLATION BUFFER REGISTERS

PROCESSOR REGISTER <3:0>

NAME	PR#	MEMSCAR#
TBGPR	PART OF 17	D

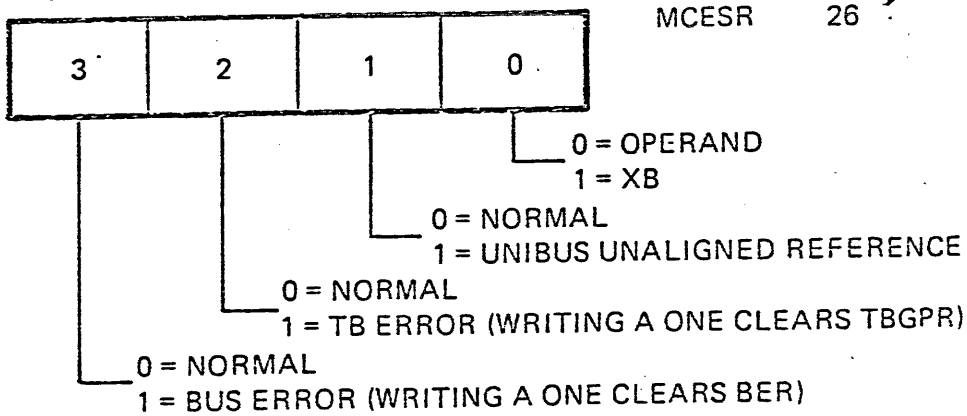
'UTR CHIP'



PROCESSOR REGISTER <3:0>

NAME	PR#	MEMSCAR#
MCESR	26	8

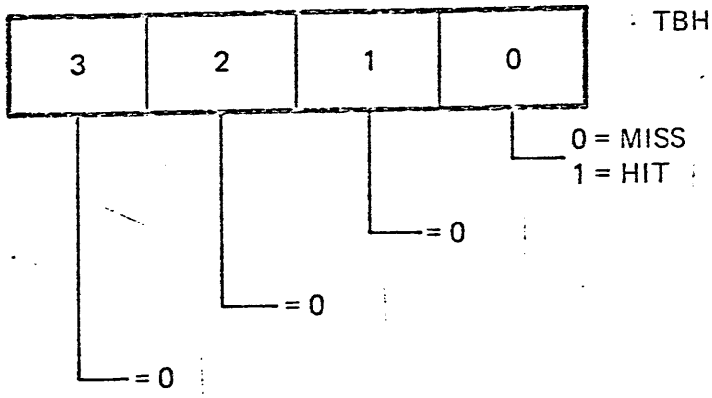
'UTR CHIP'



PROCESSOR REGISTER <3:0>

NAME	PR#	MEMSCAR#
TBHR	7	C

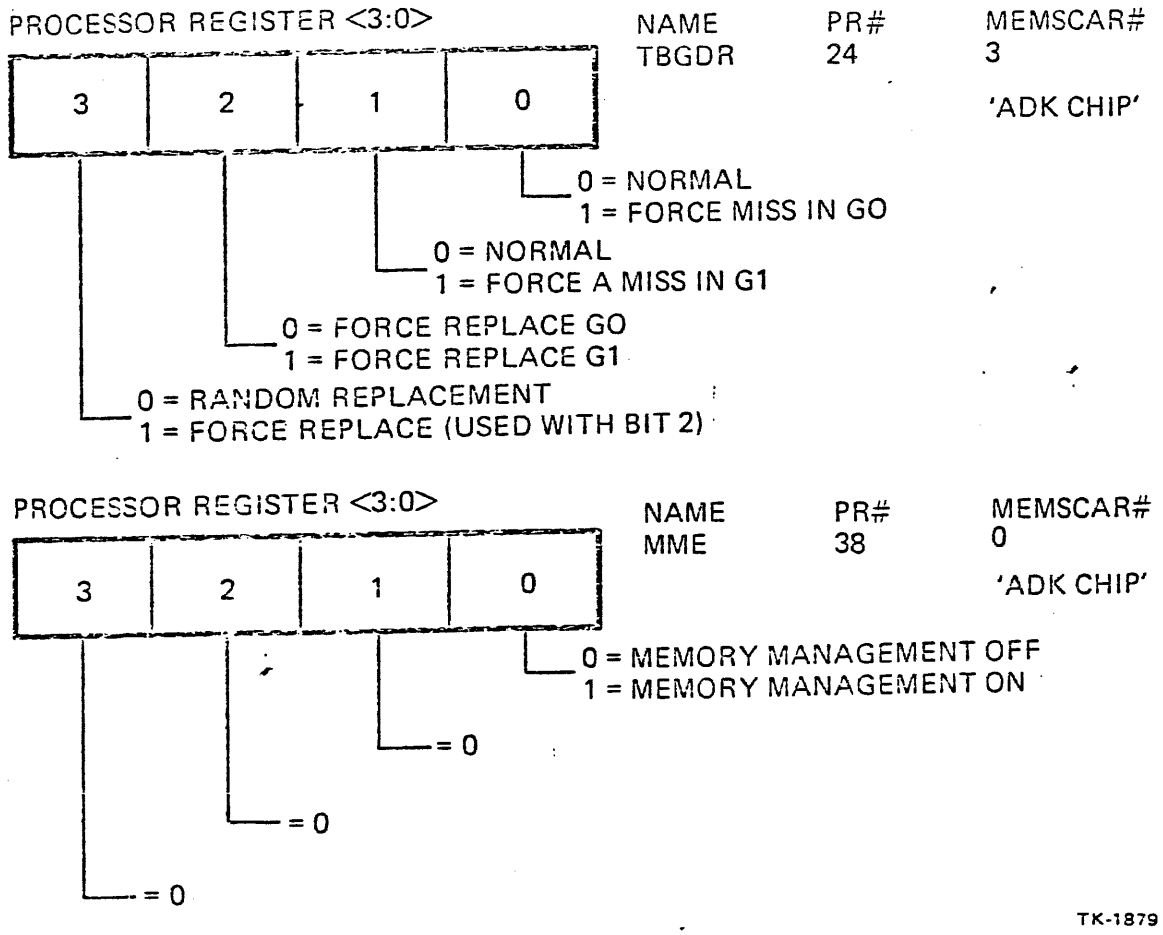
'UTR CHIP'



TK-1878

Translation Buffer S/C Registers

TRANSLATION BUFFER REGISTERS CON'T



Note: S/C Registers are loaded by microcode when a Processor Register is loaded.

Translation S/C Registers

VAX 11/750 LEVEL II

Cache

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The cache is located on the MIC module. Its purpose is to increase system operation speed by decreasing memory cycle time. The function used is storing data in a 1K direct mapped data cache with 1K X 14 TAG store and 1K X 36 data store.

MODULE XII: CACHE

SYNOPSIS

The cache module is designed as a block diagram and schematic level analysis of the following:

a. characteristics
b. inputs/outputs
c. tag/index formats
d. control registers
e. parity checking

Also included is a fault isolation laboratory exercise.

OBJECTIVES

Given a faulty Comet CPU, isolate the defective cache chip, utilizing all available documentation, diagnostics and test equipment.

Given statements concerning cache and several possible definitions for each, select the one correct definition.

Given a blank block diagram of cache, correctly identify each block by labeling it.

Utilizing the MIC print set, trace the signal path for a preselected signal from origin to destination.

SAMPLE TEST ITEM

The signal EN CACHE L on page 11 of 17 in the MIC print set, is utilized to:

a. control output data
b. control input data
c. turn cache on
d. enable parity checker

LAB EXERCISE

a. load microdiagnostics
b. run microdiagnostics
c. interpret error printout
d. isolate malfunction to module
e. isolate malfunction to chip
f. perform appropriate repairs

RESOURCES

Comet Specifications
Comet Print Set
Comet Microcode Listing

OUTLINE

A. Function

B. Purpose

C. Characteristics

D. Simplified Block Diagram Physical Description

1. Tag Store
2. Data Store
3. Pad <23:Ø2>
4. Parity Generators
5. Parity Checkers
6. Comparator

E. Cache Organization

1. Tag
2. Comparator
3. Tag Parity Checker
4. Data Store
5. Data Parity Checker
6. Pad

OUTLINE (continued)

F. Simplified Block Diagram (Cycle Description)

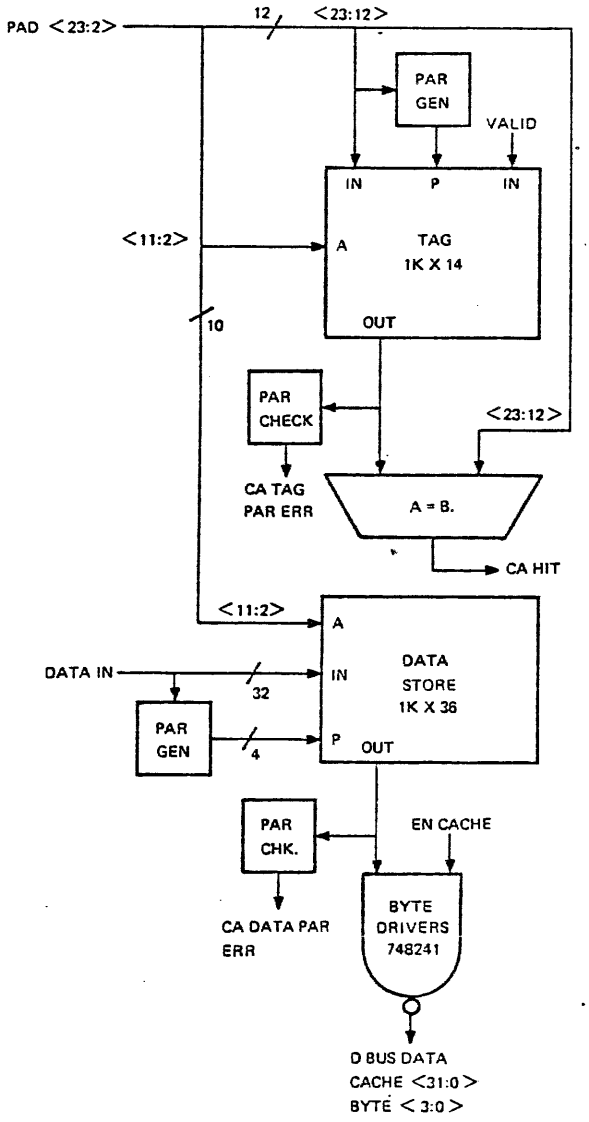
1. Read Hit
2. Read Miss
3. Write (Four Byte)
4. Write Hit (1-2 Bytes)
5. Write Miss

G. Detailed Block Diagram

1. Chip Overview
2. CAK Chip
3. CMK Chip
4. UTR Chip

H. Clear Cache Init Routine

I. Registers



CACHE SIMPLIFIED BLOCK DIAGRAM

Figure 12-1

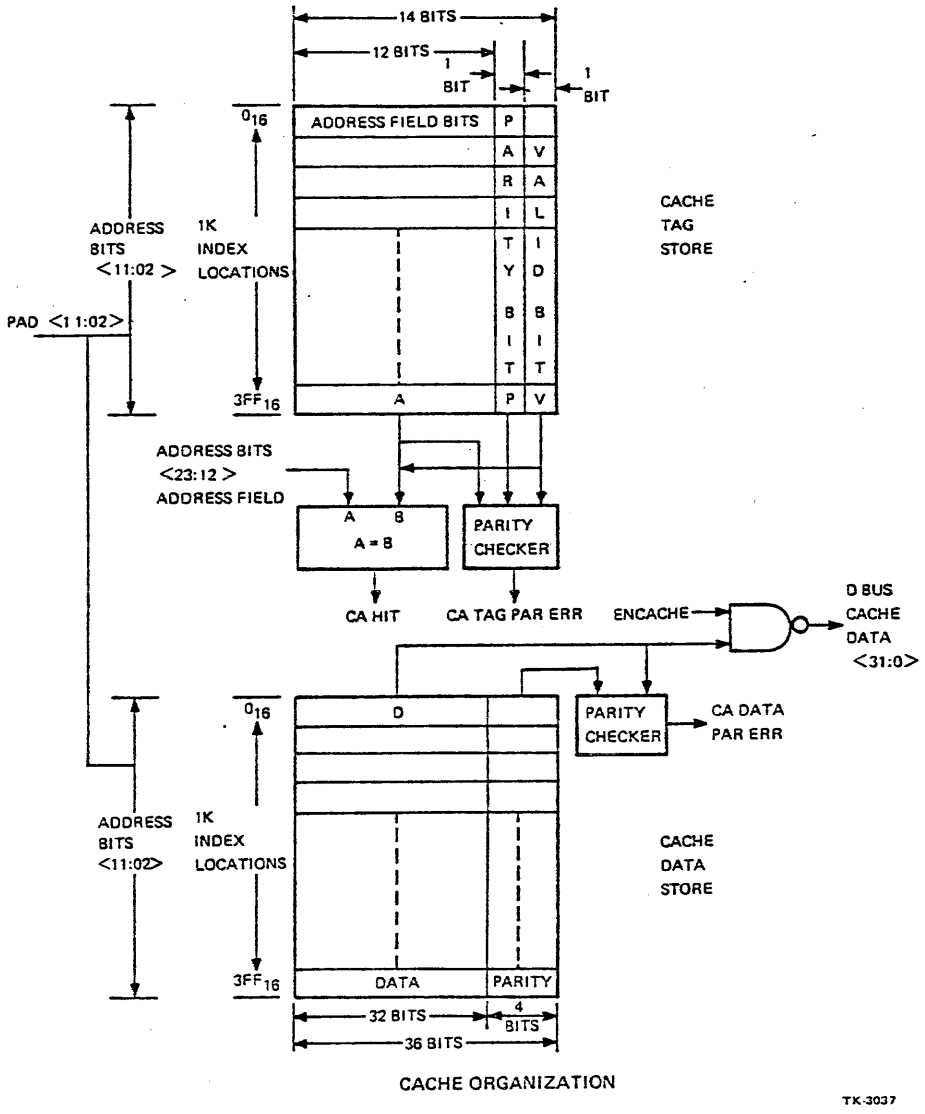
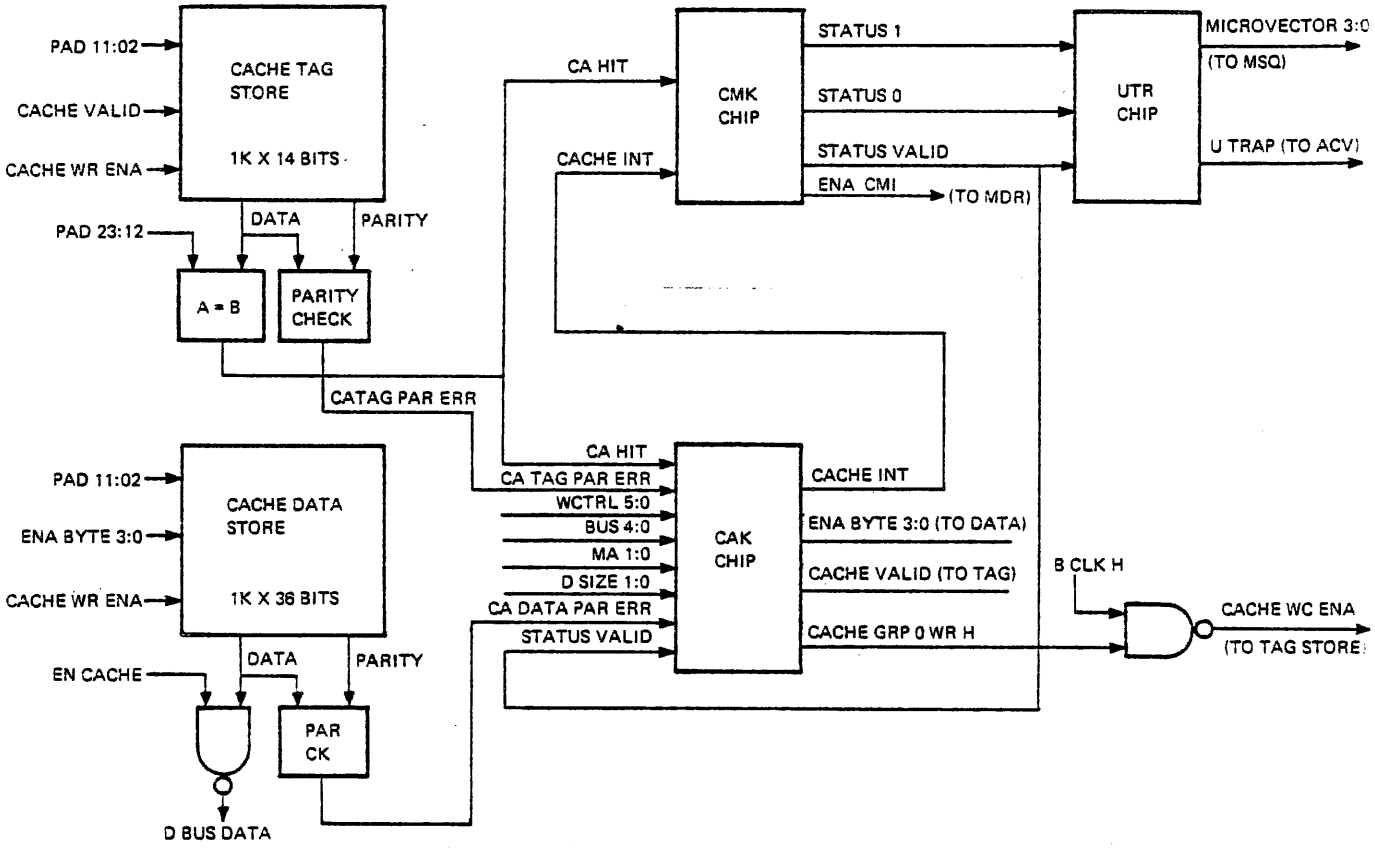

14 BITS
12 BITS
1 BIT
1 BIT
0_{16}
ADDRESS FIELD BITS
P
A V
R A
I L
T I
Y D
B B
I I
T T
ADDRESS BITS <11:02>
1K INDEX LOCATIONS
PAD <11:02>
$3FF_{16}$
A
P V
CACHE TAG STORE
ADDRESS BITS <23:12> ADDRESS FIELD
A
B
A = B
PARITY CHECKER
CA HIT
CA TAG PAR ERR
ENCACHE
D BUS CACHE DATA <31:0>
0_{16}
D
PARITY CHECKER
CA DATA PAR ERR
ADDRESS BITS <11:02>
1K INDEX LOCATIONS
CACHE DATA STORE
$3FF_{16}$
DATA
PARITY
32 BITS
4 BITS
36 BITS
CACHE ORGANIZATION
TK-3037

Figure 12-2



CACHE DETAILED BLOCK DIAGRAM

TK-3039

Figure 12-3

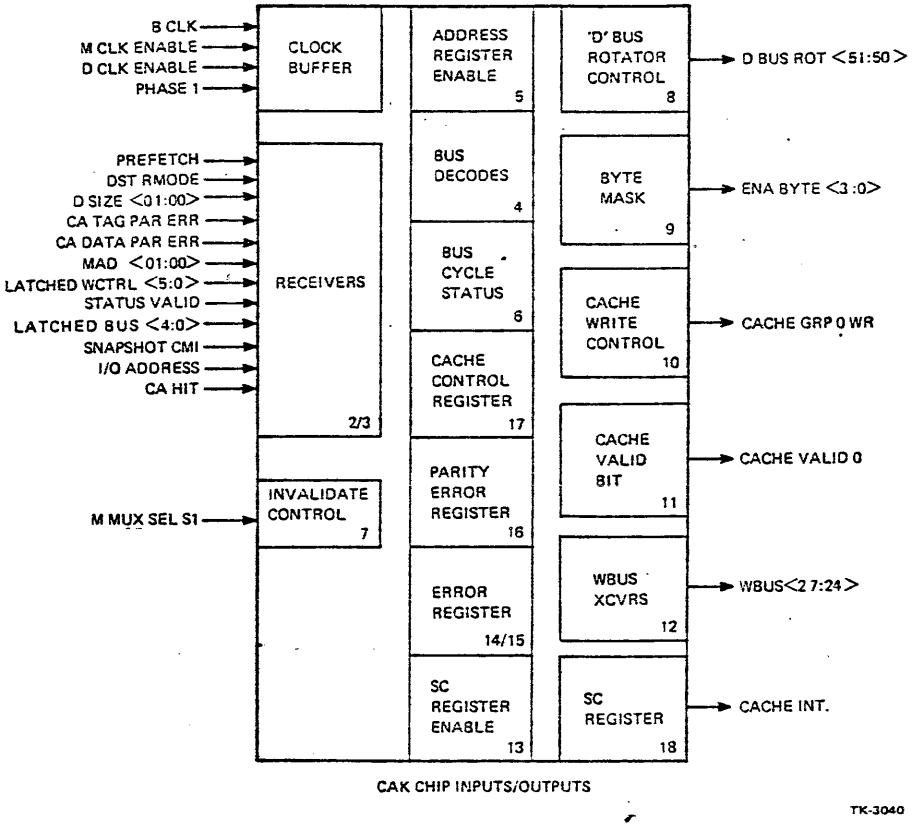

B CLK
M CLK ENABLE
D CLK ENABLE
PHASE 1
CLOCK BUFFER
PREFETCH
DST RMODE
D SIZE <01:00>
CA TAG PAR ERR
CA DATA PAR ERR
MAD <01:00>
LATCHED WCTRL <5:0>
STATUS VALID
LATCHED BUS <4:0>
SNAPSHOT CMI
I/O ADDRESS
CA HIT
RECEIVERS 2/3
M MUX SEL S1
INVALIDATE CONTROL 7
ADDRESS REGISTER ENABLE 5
BUS DECODES 4
BUS CYCLE STATUS 6
CACHE CONTROL REGISTER 17
PARITY ERROR REGISTER 16
ERROR REGISTER 14/15
SC REGISTER ENABLE 13
'D' BUS ROTATOR CONTROL 8
BYTE MASK 9
CACHE WRITE CONTROL 10
CACHE VALID BIT 11
WBUS XCVRS 12
SC REGISTER 18
D BUS ROT <51:50>
ENA BYTE <3:0>
CACHE GRP 0 WR
CACHE VALID 0
WBUS<27:24>
CACHE INT.
CAK CHIP INPUTS/OUTPUTS
TK-3040

Figure 12-4

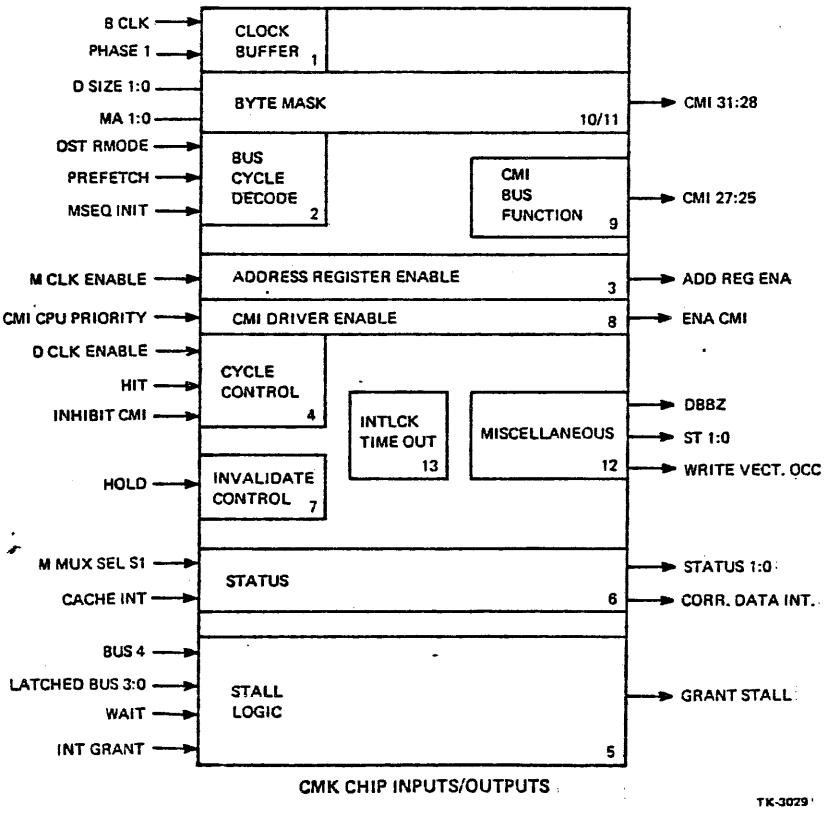

B CLK
PHASE 1
CLOCK BUFFER 1
D SIZE 1:0
MA 1:0
BYTE MASK 10/11
CMI 31:28
DST RMODE
PREFETCH
MSEQ INIT
BUS CYCLE DECODE 2
CMI BUS FUNCTION 9
CMI 27:25
M CLK ENABLE
ADDRESS REGISTER ENABLE 3
ADD REG ENA
CMI CPU PRIORITY
CMI DRIVER ENABLE 8
ENA CMI
D CLK ENABLE
HIT
INHIBIT CMI
CYCLE CONTROL 4
INTLCK TIME OUT 13
MISCELLANEOUS 12
DBBZ
ST 1:0
WRITE VECT. OCC
HOLD
INVALIDATE CONTROL 7
M MUX SEL S1
CACHE INT
STATUS 6
STATUS 1:0
CORR. DATA INT.
BUS 4
LATCHED BUS 3:0
WAIT
INT GRANT
STALL LOGIC 5
GRANT STALL
CMK CHIP INPUTS/OUTPUTS
TK-3029

Figure 12-5

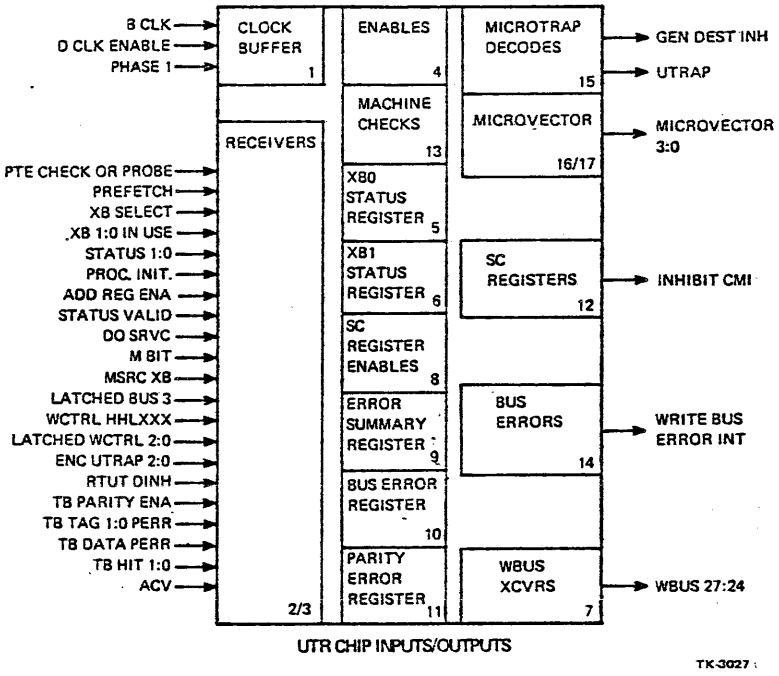

B CLK
D CLK ENABLE
PHASE 1
CLOCK BUFFER 1
RECEIVERS 2/3
PTE CHECK OR PROBE
PREFETCH
XB SELECT
XB 1:0 IN USE
STATUS 1:0
PROC. INIT.
ADD REG ENA
STATUS VALID
DO SRVC
M BIT
MSRC XB
LATCHED BUS 3
WCTRL HHLXXX
LATCHED WCTRL 2:0
ENC UTRAP 2:0
RTUT DINH
TB PARITY ENA
TB TAG 1:0 PERR
TB DATA PERR
TB HIT 1:0
ACV
ENABLES 4
MACHINE CHECKS 13
X80 STATUS REGISTER 5
XB1 STATUS REGISTER 6
SC REGISTER ENABLES 8
ERROR SUMMARY REGISTER 9
BUS ERROR REGISTER 10
PARITY ERROR REGISTER 11
MICROTRAP DECODES 15
MICROVECTOR 16/17
SC REGISTERS 12
BUS ERRORS 14
WBUS XCVRS 7
GEN DEST INH
UTRAP
MICROVECTOR 3:0
INHIBIT CMI
WRITE BUS ERROR INT
WBUS 27:24
UTR CHIP INPUTS/OUTPUTS
TK-3027

Figure 12-6

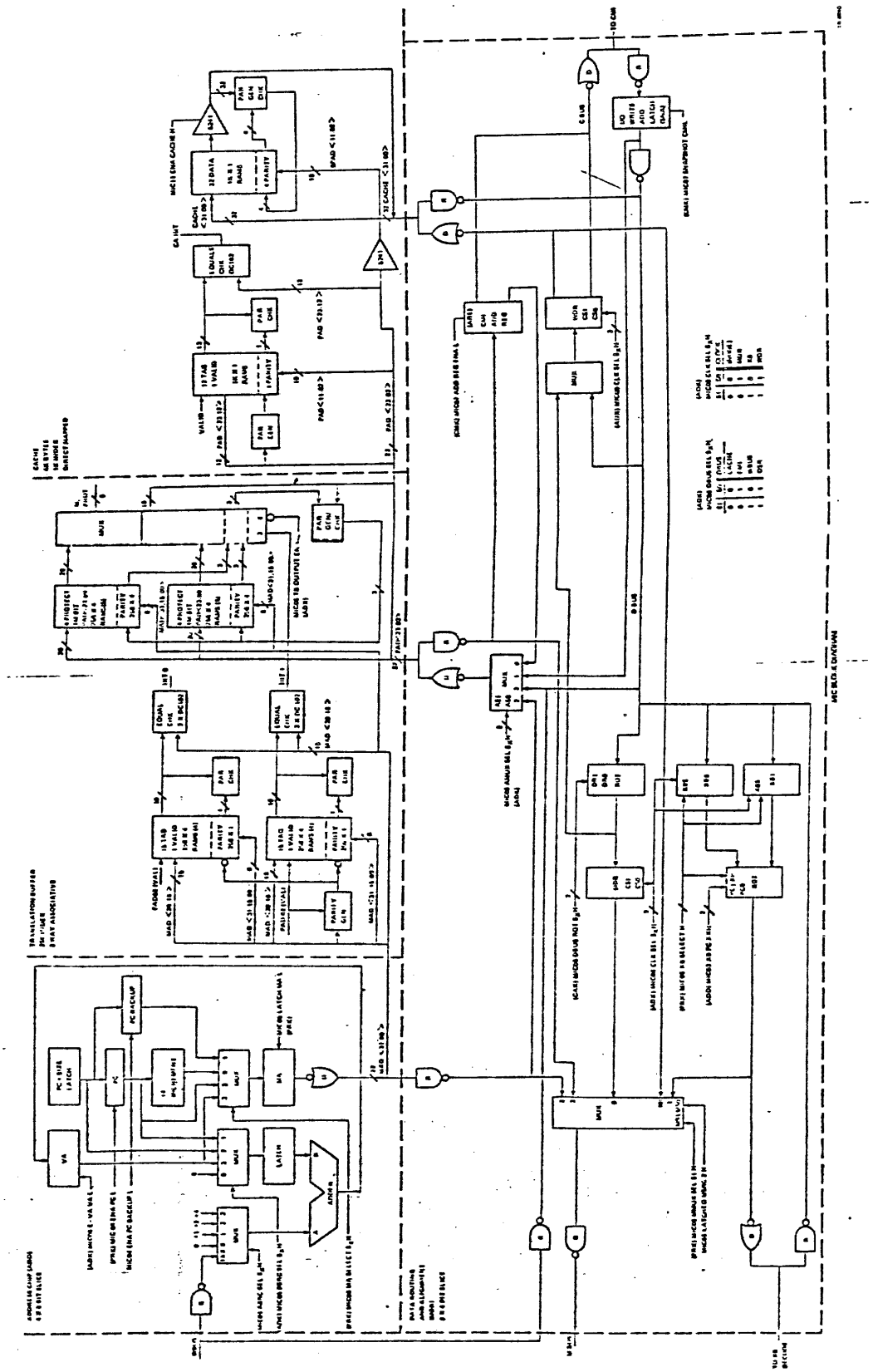


Figure 12-7

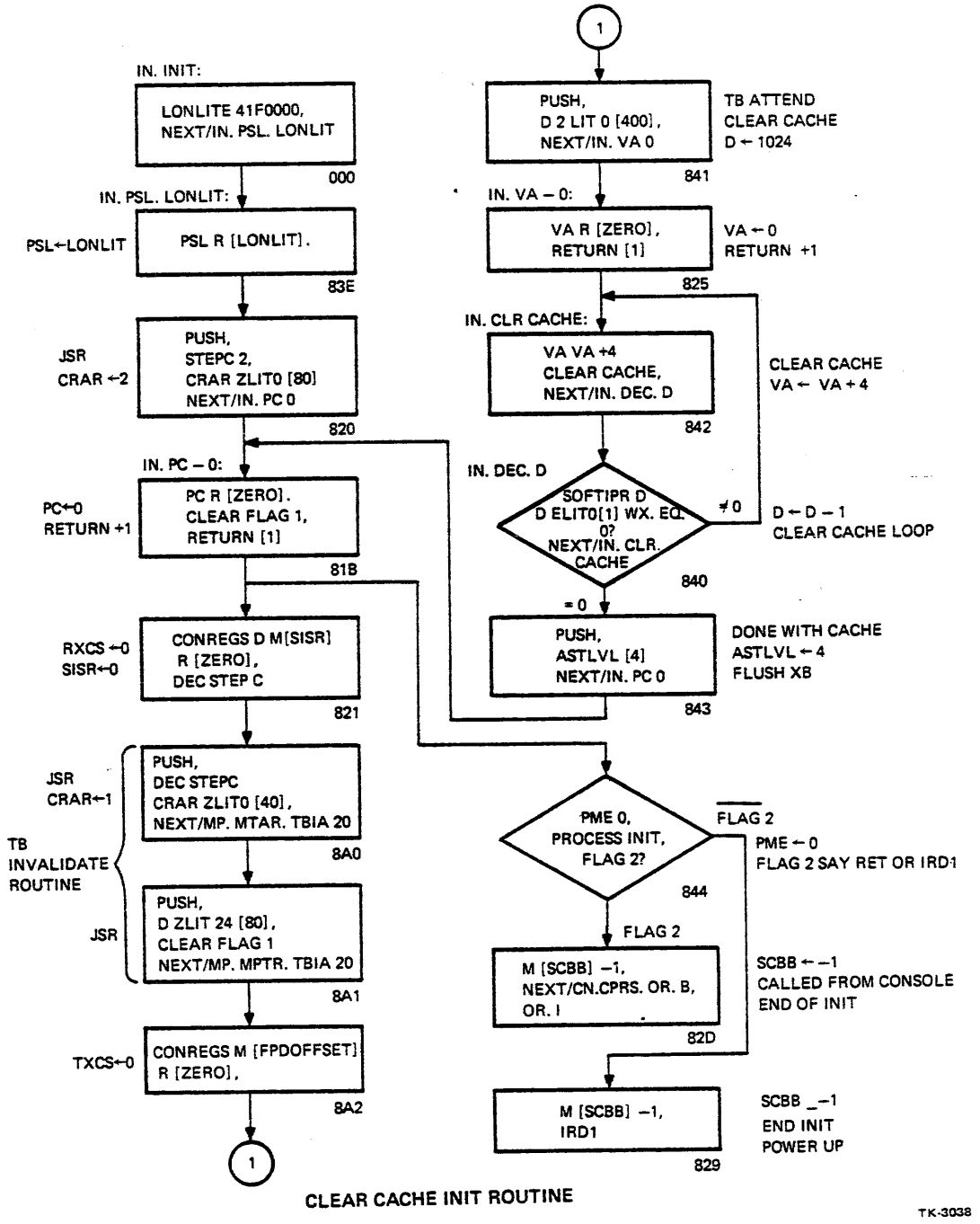
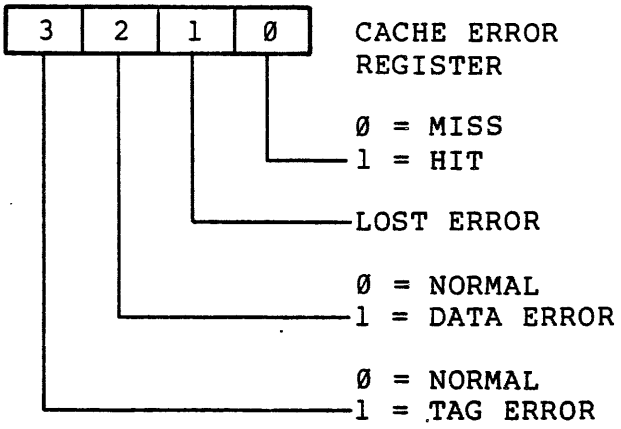
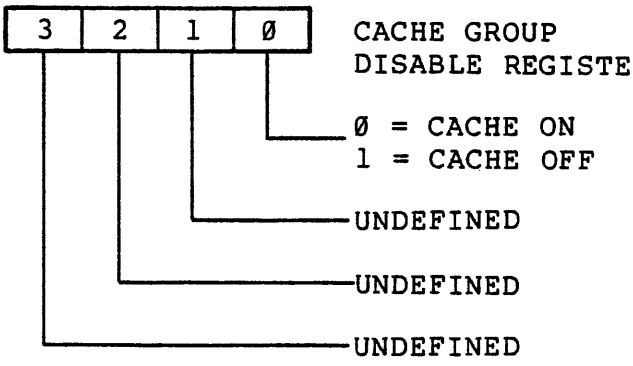

IN. INIT:
LONLITE 41F0000,
NEXT/IN. PSL. LONLIT
000
IN. PSL. LONLIT:
PSL←LONLIT
PSL R [LONLIT].
83E
JSR
CRAR ←2
PUSH,
STEPC 2,
CRAR ZLIT0 [80]
NEXT/IN. PC 0
820
IN. PC – 0:
PC←0
RETURN +1
PC R [ZERO].
CLEAR FLAG 1,
RETURN [1]
81B
RXCS ←0
SISR←0
CONREGS D M[SISR]
R [ZERO],
DEC STEP C
821
JSR
CRAR←1
PUSH,
DEC STEPC
CRAR ZLIT0 [40],
NEXT/MP. MTAR. TBIA 20
8A0
TB
INVALIDATE
ROUTINE
JSR
PUSH,
D ZLIT 24 [80],
CLEAR FLAG 1
NEXT/MP. MPTR. TBIA 20
8A1
TXCS←0
CONREGS M [FPDOFFSET]
R [ZERO],
8A2
1
1
PUSH,
D 2 LIT 0 [400],
NEXT/IN. VA 0
TB ATTEND
CLEAR CACHE
D ← 1024
841
IN. VA – 0:
VA R [ZERO],
RETURN [1]
VA ← 0
RETURN +1
825
IN. CLR CACHE:
VA VA +4
CLEAR CACHE,
NEXT/IN. DEC. D
842
CLEAR CACHE
VA ← VA + 4
IN. DEC. D
SOFTIPR D
D ELIT0[1] WX. EQ.
0?
NEXT/IN. CLR.
CACHE
≠0
D ← D – 1
CLEAR CACHE LOOP
840
=0
PUSH,
ASTLVL [4]
NEXT/IN. PC 0
DONE WITH CACHE
ASTLVL ← 4
FLUSH XB
843
PME 0,
PROCESS INIT,
FLAG 2?
FLAG 2
PME ← 0
FLAG 2 SAY RET OR IRD1
844
FLAG 2
M [SCBB] –1,
NEXT/CN.CPRS. OR. B,
OR. I
SCBB ← –1
CALLED FROM CONSOLE
END OF INIT
82D
M [SCBB] –1,
IRD1
SCBB _–1
END INIT
POWER UP
829
CLEAR CACHE INIT ROUTINE
TK-3038

Figure 12-8

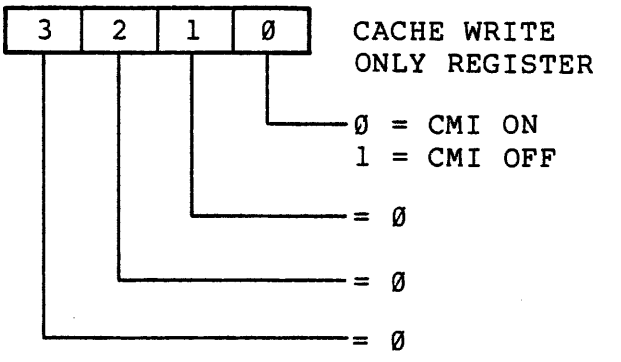
NAME	PR #	MEMSCAR #
	27	4



NAME	PR #	MEMSCAR #
	25	6



NAME	PR #	MEMSCAR #
	PART OF 17	E



CACHE REGISTERS

Cache S/C Registers

VAX-11/750 LEVEL II

Data Routing and Alignment

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The data routing and alignment (DR+A) is located on the MIC module and made up primarily of MDR chips.

Its purpose is to take or give data to/from the data path portion of the CPU.

The function of the DR+A is to interface with the following:

I. A. Input buses
   1. W bus
   2. MA bus

   B. Output bus - M bus

   C. Bidirectional buses
   1. PA bus
   2. Cache bus
   3. CMI bus
   4. XB decode bus

   D. Internal buses
   1. C bus
   2. D bus

II. Perform data transfers on read and write cycles.

III. Data alignment for use by the data path module.

MODULE XIII: DATA ROUTING AND ALIGNMENT

SYNOPSIS

The data routing and alignment module is designed as a block and schematic diagram analysis of the following:

a. data routing
b. cache/CMI interface
c. WDR
d. MDR
e. W bus interface
f. XB interface
g. CMI address register
h. I/O address latch
i. microroutines

OBJECTIVES

Identify the data routing and alignment elements by answering multiple choice questions. The elements include:

a. data routing
b. cache/CMI interface
c. CPU bus interfaces

Utilizing the MIC module schematic diagram, trace the origin and destination of a specified signal.

With a malfunction inserted in the 11/750 CPU on the MIC module, utilize all available documentation and test equipment to isolate malfunction to chip level.

SAMPLE TEST ITEM

Select the statement that is NOT true about the data routing and alignment logic.

a. Memory data received from the CMI is stored in the MDR.
b. The CMI address latch holds the virtual address of the operand or instruction referenced.
c. The XB is an 8 byte instruction cache.
d. Immediate operands go to the data paths via the M bus.

LAB EXERCISE

a. load 11/750 microdiagnostics
b. run microdiagnostics
c. interpret error print out
d. isolate malfunction to a module
e. isolate malfunction to a chip
f. perform appropriate repairs

RESOURCES

Comet Specification
Comet Print Set
Comet Microcode Listing

OUTLINE

XIII. Data Routing and Alignment

- A. General Characteristics
- B. Purpose
- C. Functions
- D. Simplified Block Diagram
- E. Memory Cycles
  - 1. Basic
  - 2. With Cache
- F. Virtual Memory Addressing
- G. Control Block Diagram Signals
- H. Data Rotation
- I. Chip Descriptions
  - 1. MDR
  - 2. PRK

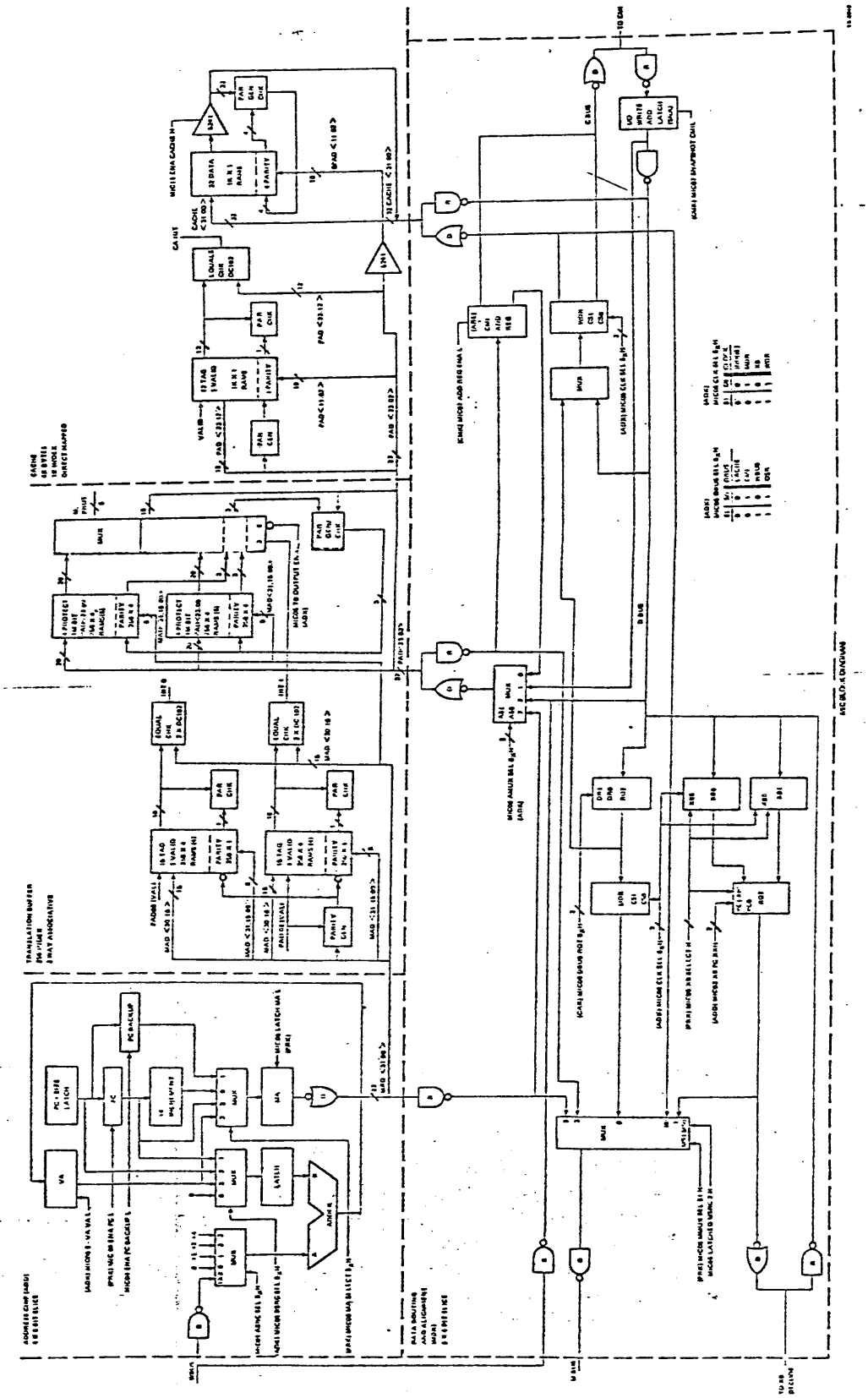


Figure 13-1 MIC Block Diagram

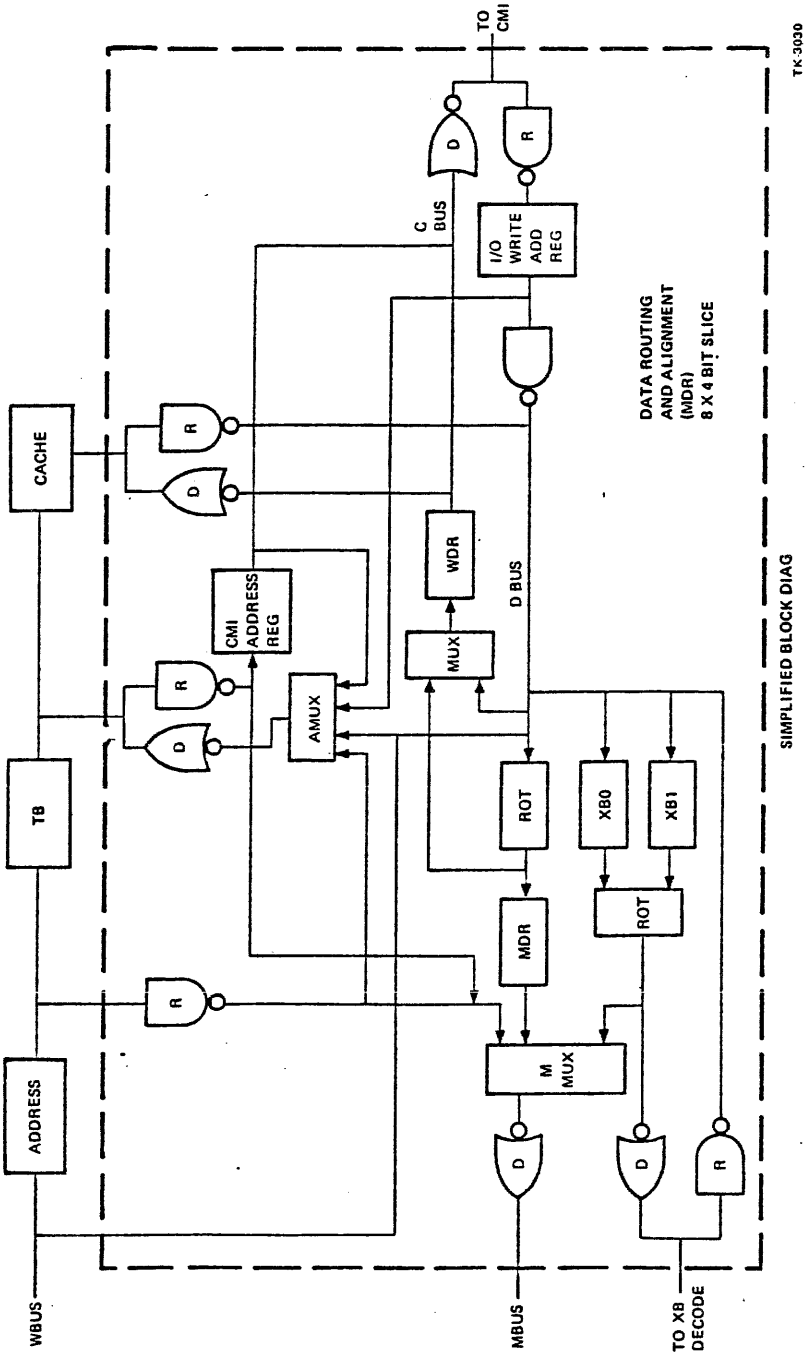


Figure 13-2 Simplified DR + A BLock

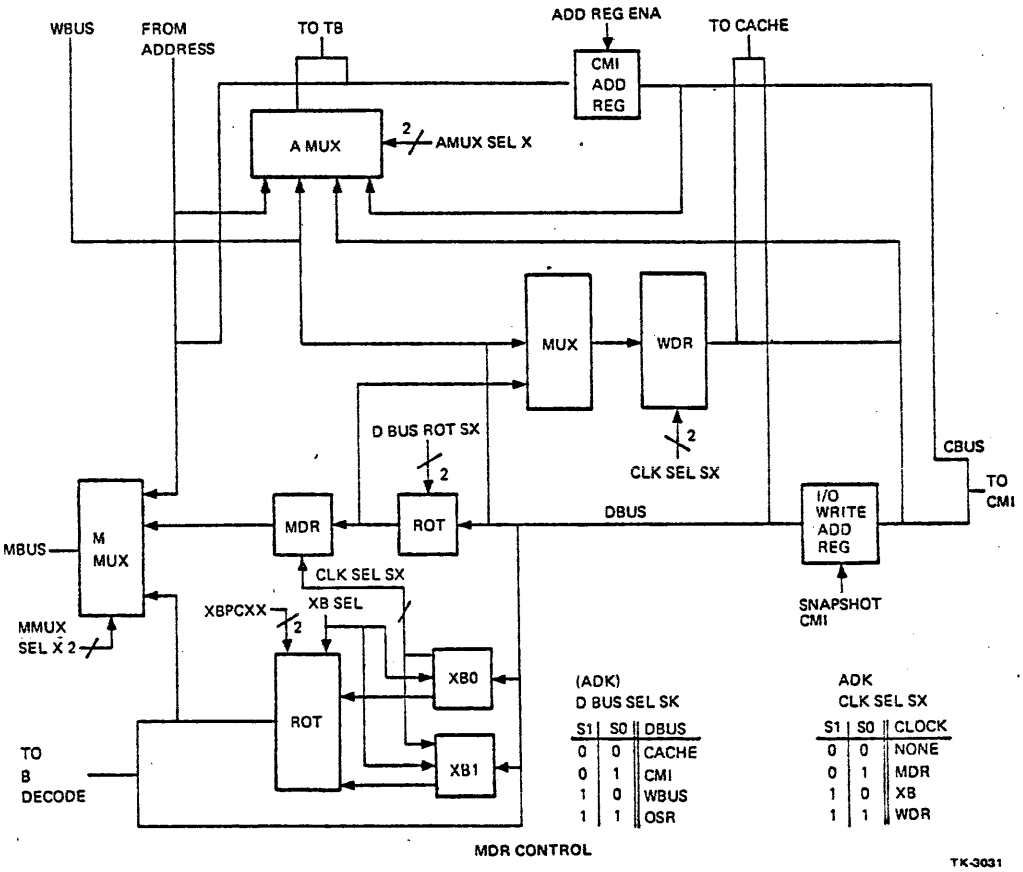


Figure 13-3 DR + A Control Block

A MUX SEL <S1:SØ> H TRUE

Controls MUX Input to A MUX, From ADK Chip, To PA BUS

S1	SØ	Input to PA BUS
L	L	CMI ADD REG
L	H	CMI DATA
H	L	MA BUS
H	H	D BUS

* Note: During VA transfer with MME, A MUX SEL = CMI address and D BUS SEL = Cache only lower byte of address allowed to pass through AMUX.

M MUX SEL <S1:SØ> H TRUE

M MUX SEL control what inputs are selected to the M BUS. S1 is actually named M MUX SEL 1 from PRK. 5Ø is actually latched MSRC 2 from MIC 5.

S1	SØ	Data to M BUS
L	L	MDR
L	H	XB DATA
H	L	MA BUS
H	H	PA BUS

DBUS ROT S<1:Ø> H

DBUS ROT S<1:Ø> H cause data from the DBUS to appear on the inputs to the MDR and WDR byte rotator as shown in the following chart:

DBUS ROT S1 H	DBUS ROT SØ H	ROT OUT (BYTES)			
LOW	LOW	3	2	1	Ø
LOW	HIGH	Ø	3	2	1
HIGH	LOW	1	Ø	3	2
HIGH	HIGH	2	1	Ø	3

CLK SEL S<1:Ø> H

CLK SEL S<1:Ø> H determine which DBUS destinations will be clocked on the low to high transition of B CLK L according to the following chart:

CLK SEL S1 H	CLK SEL SØ H	ENABLE
LOW	LOW	NOTHING
LOW	HIGH	MDR
HIGH	LOW	XB
HIGH	HIGH	WDR

In addition to the conditions listed in the chart, portions of the MDR must be enabled for data returning from a READ, SECOND REFERENCE. A READ< SECOND REFERENCE is decoded in the MDR chip when:

DBUS ROT S<1:Ø> H not equal to zero and
CLK SEL S<1:Ø> H equal zero and
WDR not being sourced onto the MBUS

Then, clocks are enabled for bytes of the MDR as shown in the following chart:

DBUS ROT S1 H	DBUS ROT SØ H	BYTES ENABLED			
LOW	HIGH	3	x	x	x
HIGH	LOW	3	2	x	x
HIGH	HIGH	3	2	1	x

Any time the MDR or any portion of the MDR is enabled or XB is enabled, and CMI DATA is selected on the DBUS, the WDR is also enabled. In this case, the WDR MUX is steered to DBUS data enabled. In this case, the WDR MUX is steered to DBUS data instead of the DBUS ROTATOR output.

Execution Buffer

XB DATA SELECTION

PC <Ø1:ØØ> H
PC <Ø1:ØØ> H and XB SELECT H determine which bytes of each XB will appear on the XB DECODE BUS, and on the MBUS if the MBUS MULTIPLEXER is steered to XB DATA, according to the following chart:

					XB DEC BYTE 1	XB DEC BYTE Ø
XB SEL H	PC Ø1 H	PC ØØ H	MBUS BYTE 3	MBUS BYTE 2	MBUS BYTE 1	MBUS BYTE Ø
LOW	LOW	LOW	XB1 BYTE 3	XB1 BYTE 2	XB1 BYTE 1	XB1 BYTE Ø
LOW	LOW	HIGH	XBØ BYTE Ø	XB1 BYTE 3	XB1 BYTE 2	XB1 BYTE 1
LOW	HIGH	LOW	XBØ BYTE 1	XBØ BYTE Ø	XB1 BYTE 3	XB1 BYTE 2
LOW	HIGH	HIGH	XBØ BYTE 2	XBØ BYTE 1	XBØ BYTE Ø	XB1 BYTE 3
HIGH	LOW	LOW	XBØ BYTE 3	XBØ BYTE 2	XBØ BYTE 1	XBØ BYTE Ø
HIGH	LOW	HIGH	XB1 BYTE Ø	XBØ BYTE 3	XBØ BYTE 2	XBØ BYTE 1
HIGH	HIGH	LOW	XB1 BYTE 1	XB1 BYTE Ø	XBØ BYTE 3	XBØ BYTE 2
HIGH	HIGH	HIGH	XB1 BYTE 2	XB1 BYTE 1	XB1 BYTE Ø	XBØ BYTE 3

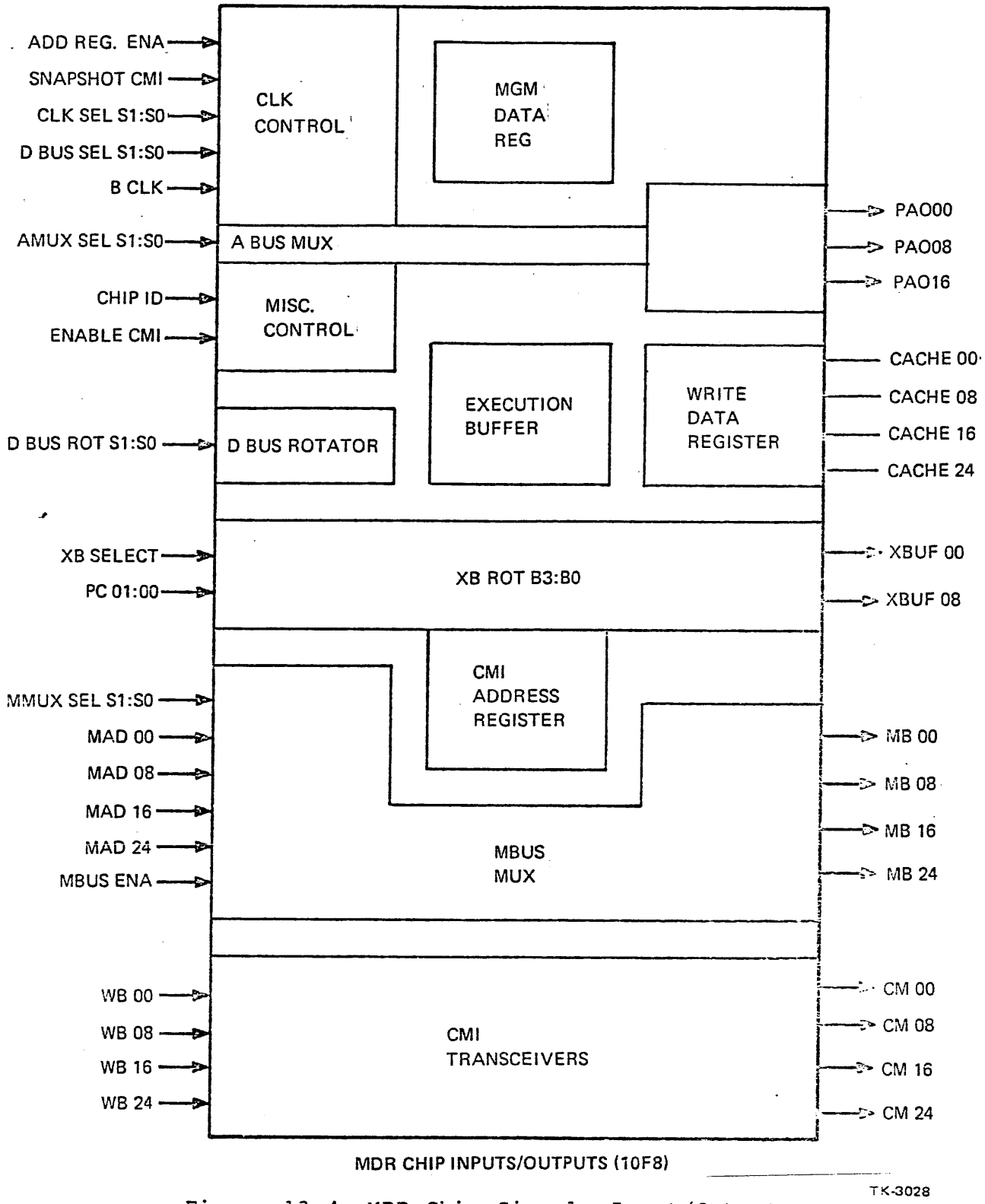


Figure 13-4 MDR Chip Signals Input/Output

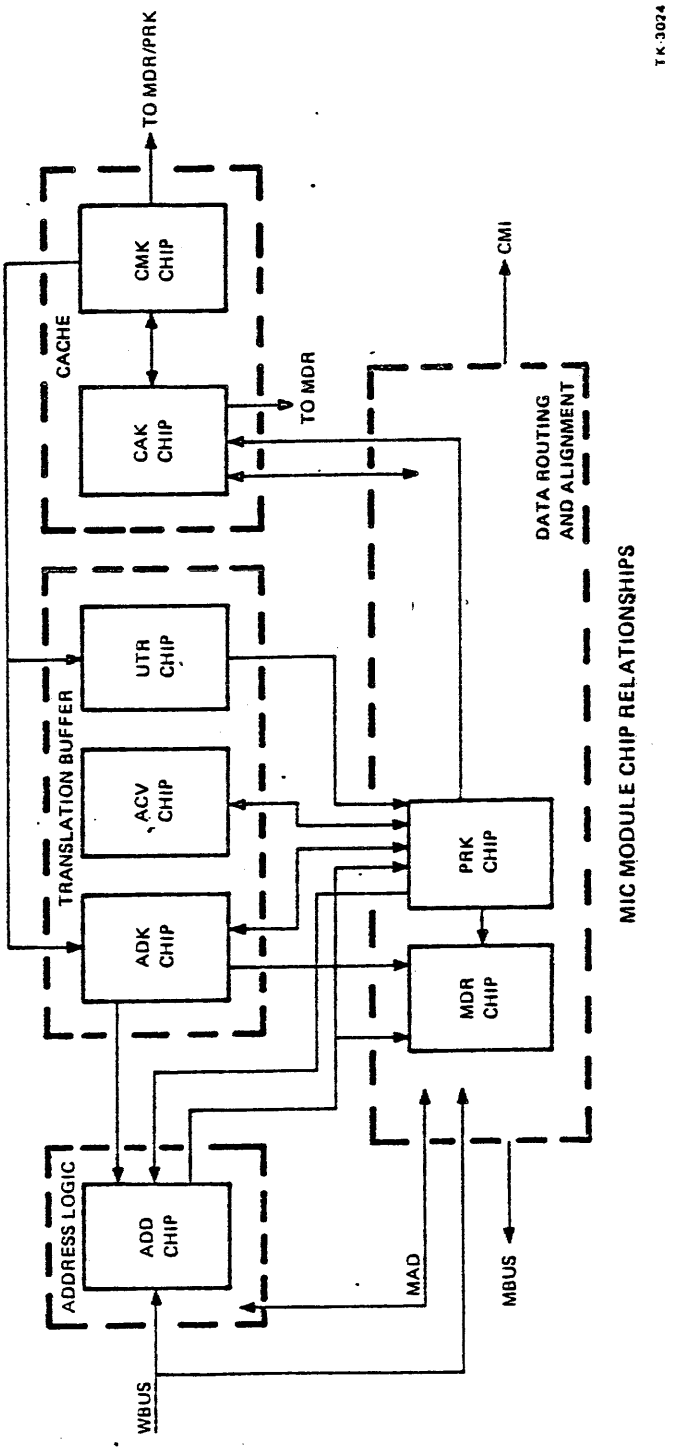


Figure 13-5 MIC Module Chip Relationships

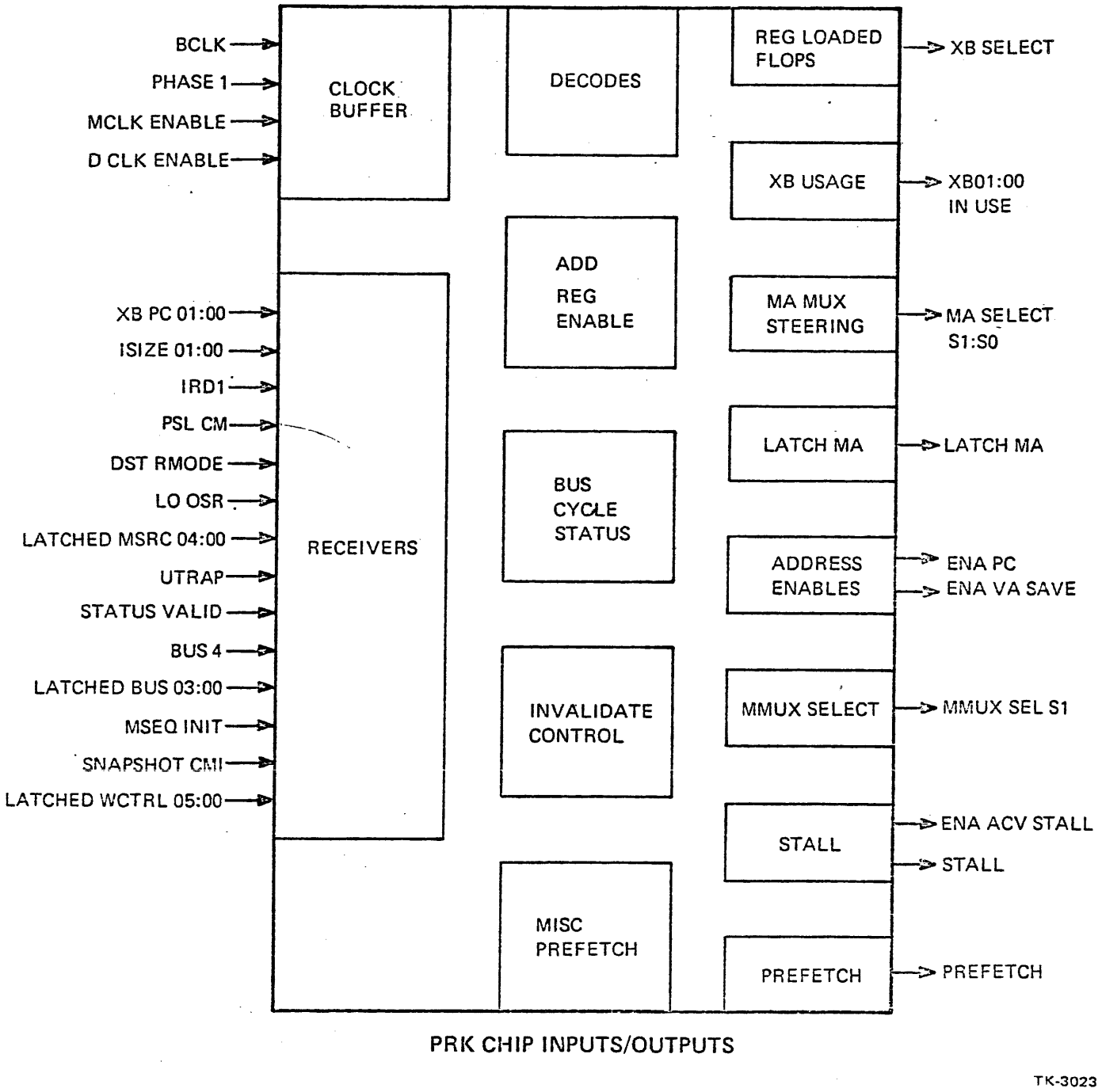


Figure 13-6 PRK Chip Signals Input/Output

VAX 11/750 LEVEL II

Execution Buffer

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The Execution Buffer is located on the MIC module just as the DR+A is located in the MDR chips. Its purpose is to effectively eliminate the time spent by the CPU waiting for the next instruction or waiting for two memory cycles when bytes that are needed cross longword boundaries.

Its function is to act as a two longword FIFO buffer that provides two bytes addressed by PC and PC+1 to the XB decode bus.

MODULE 14: EXECUTION BUFFER

SYNOPSIS

The execution buffer module is designed as a block diagram and schematic level analysis of the following:

a. instruction register
b. operand specifier register
c. instruction decoder

Execution flows and operand specifier routines will also be covered in detail. A fault isolation laboratory exercise will be utilized to check student comprehension.

OBJECTIVES

Given a list of register bits and a list of functions, correctly match the register bits to its function.

Identify the major execution buffer logic components by correctly labeling each on a blank block diagram.

With a malfunction inserted on the MIC module of the Comet CPU, utilize all available documentation, diagnostics and test equipment, to isolate the malfunction to the chip level.

Identify as true or false statements regarding the execution buffer segment of the 11/750 CPU, writing the correct answer in the space provided.

SAMPLE TEST ITEM

Indicate in the space provided whether the statement is true or false.

a. the instruction register is 32 bits. _____
b. the instruction decode is performed by roms.

c. the operand specifier is always byte zero (Ø). _____

LAB EXERCISE

a. load 11/750 microdiagnostics
b. run microdiagnostics
c. interpret error printout
d. isolate malfunction to a module
e. further isolate malfunction to chip
f. perform appropriate repairs.

RESOURCES

11/750 Specification
11/750 Print Set
11/750 Microcode Listing

OUTLINE

XIV. Execution Buffer

A. Purpose
B. Function
C. General Description
D. Simplified Block Diagram
E. Prefetch Cycle General Description
F. Prefetch Detailed Description
G. XB Selection And Control

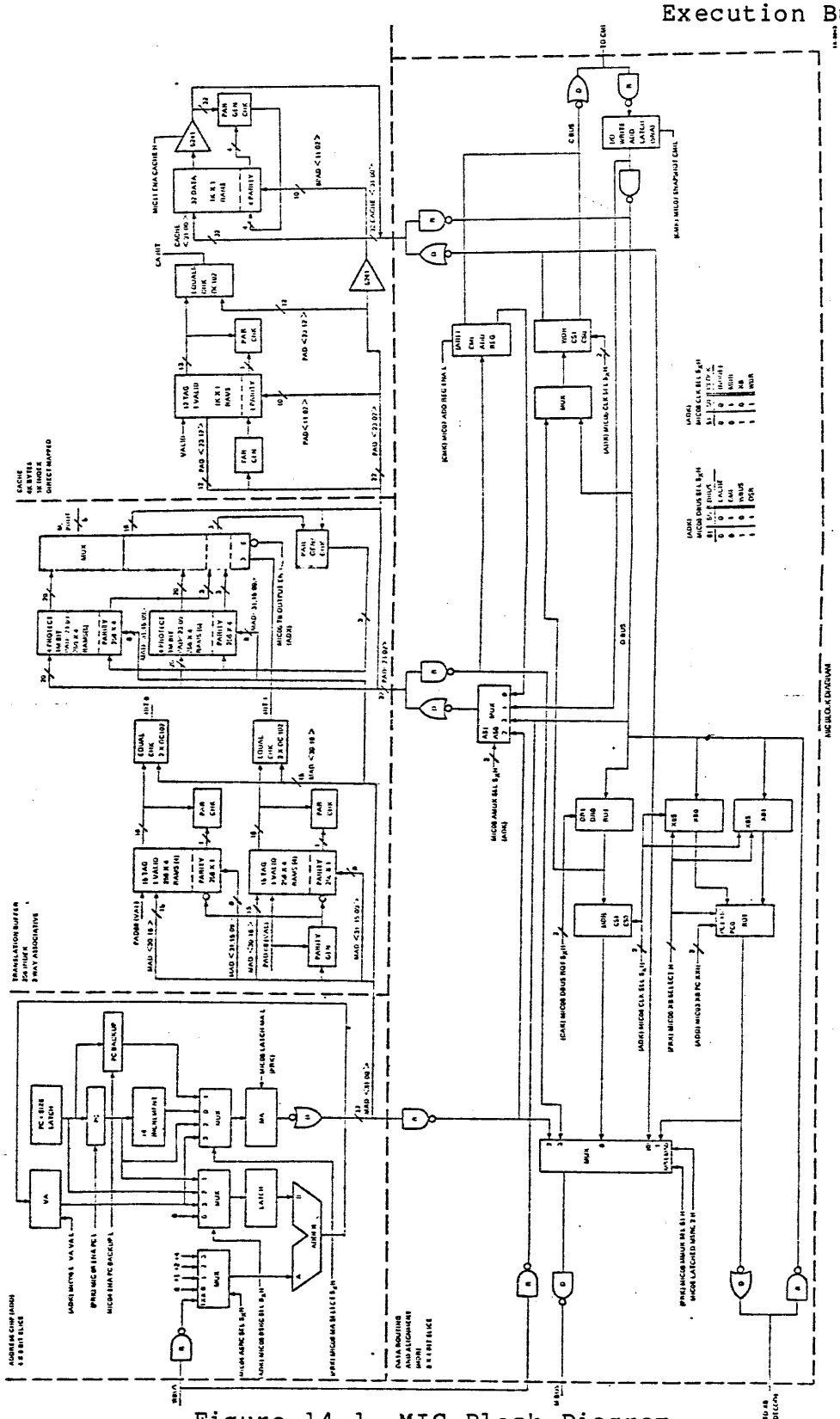


Figure 14-1 MIC Block Diagram

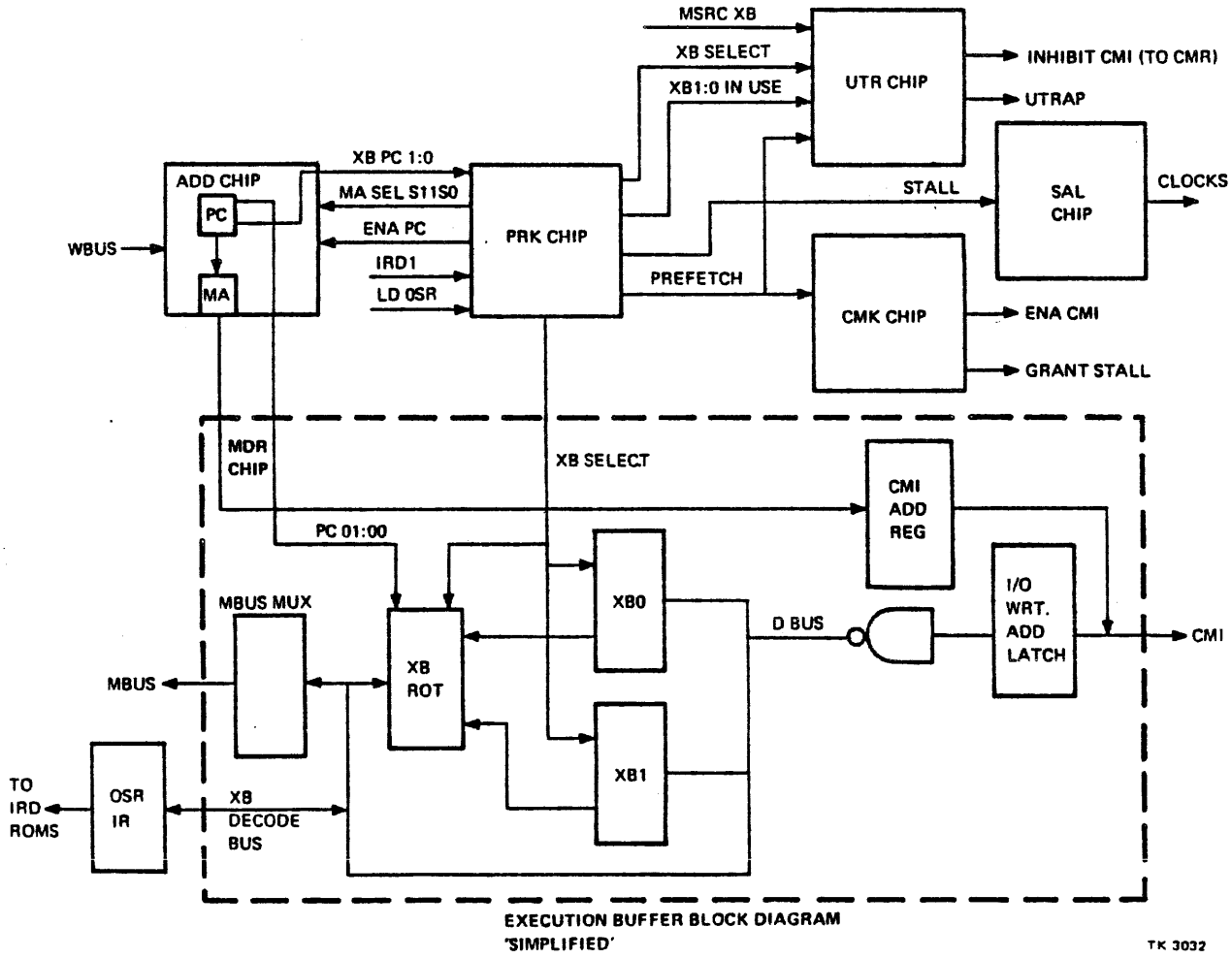


Figure 14-2 Execution Buffer Simplified Block

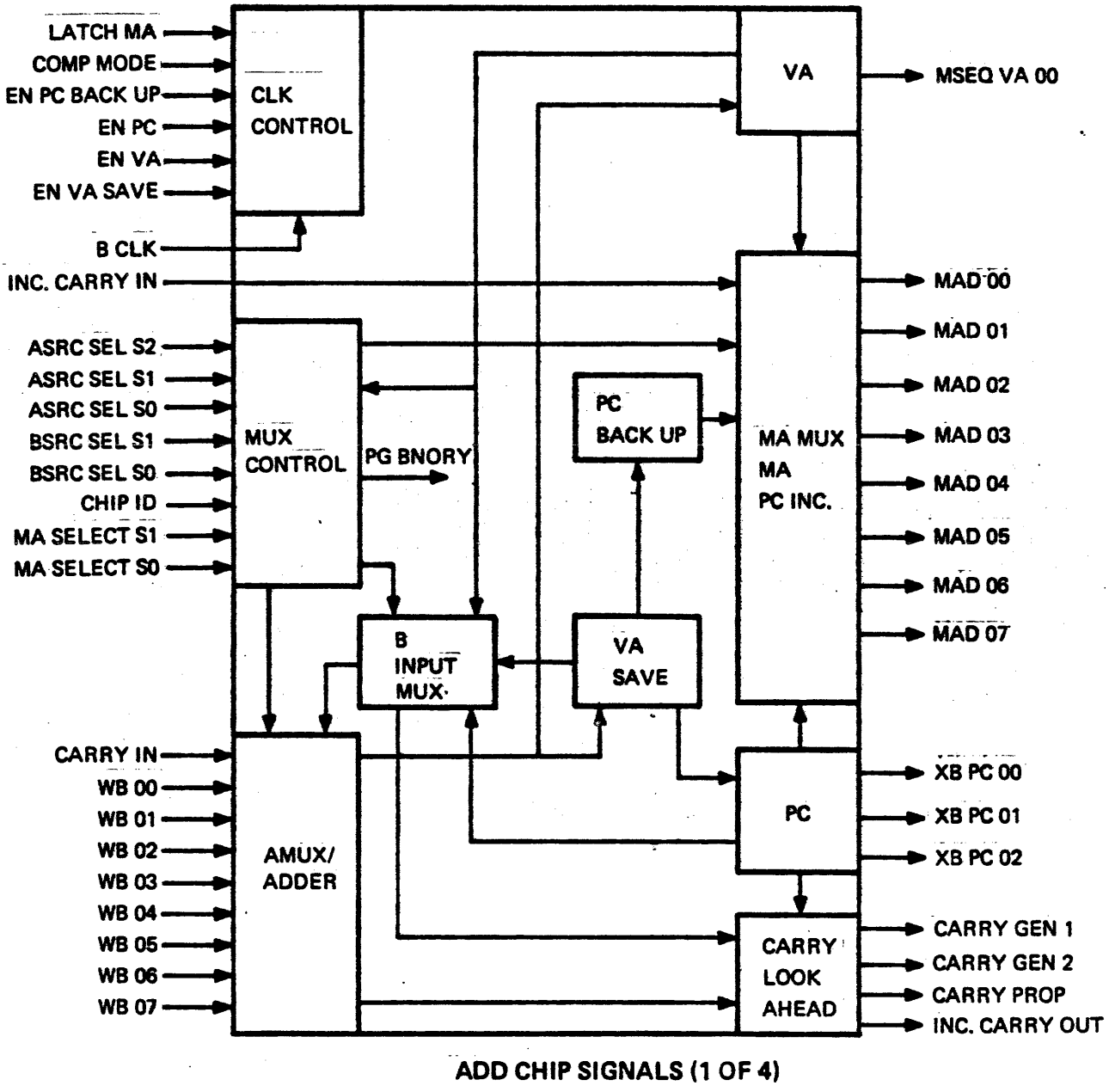


Figure 14-3 Add Chip Signals Input/Output

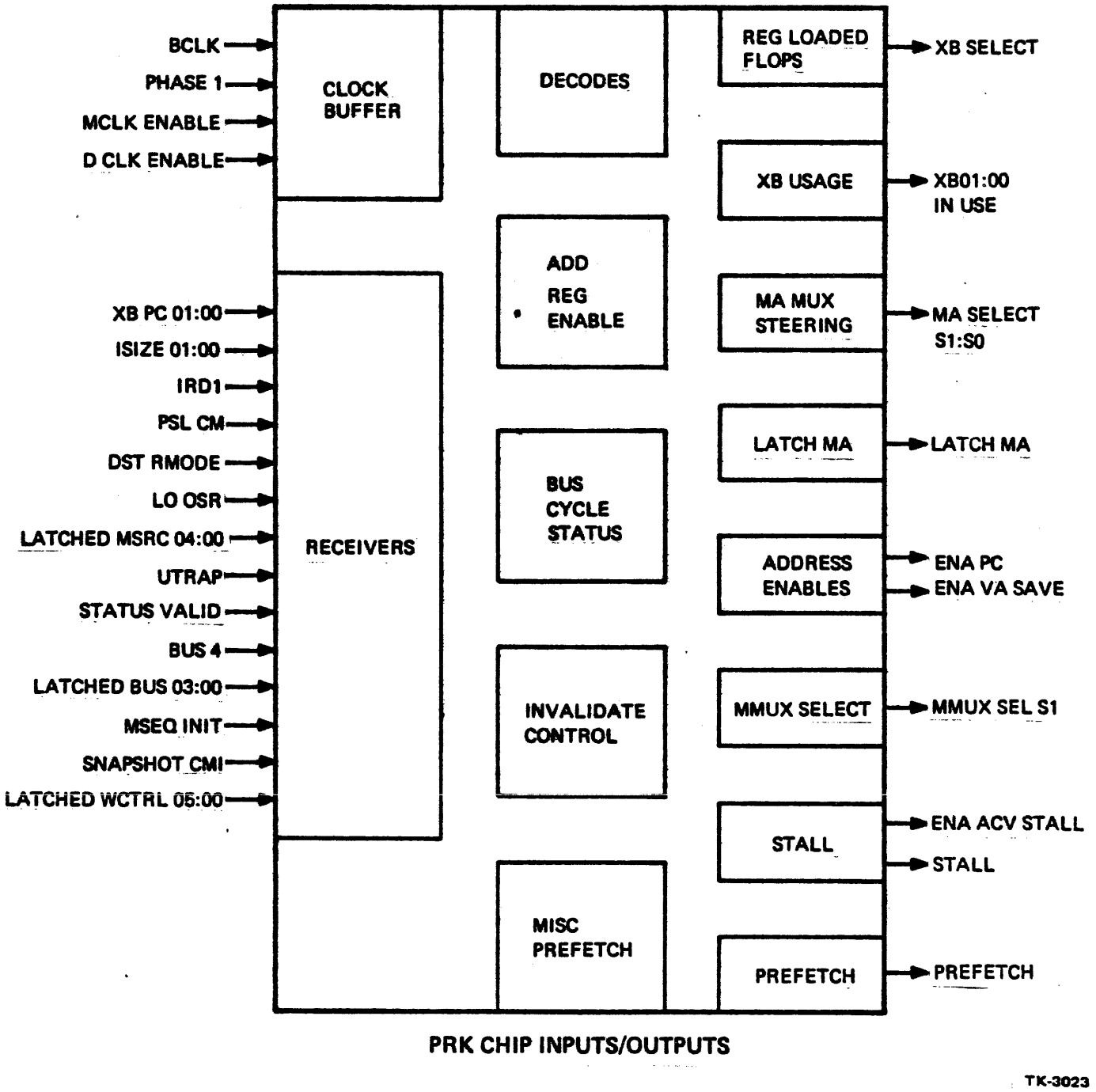


Figure 14-4 PRK Chip Signals Inputs/Outputs

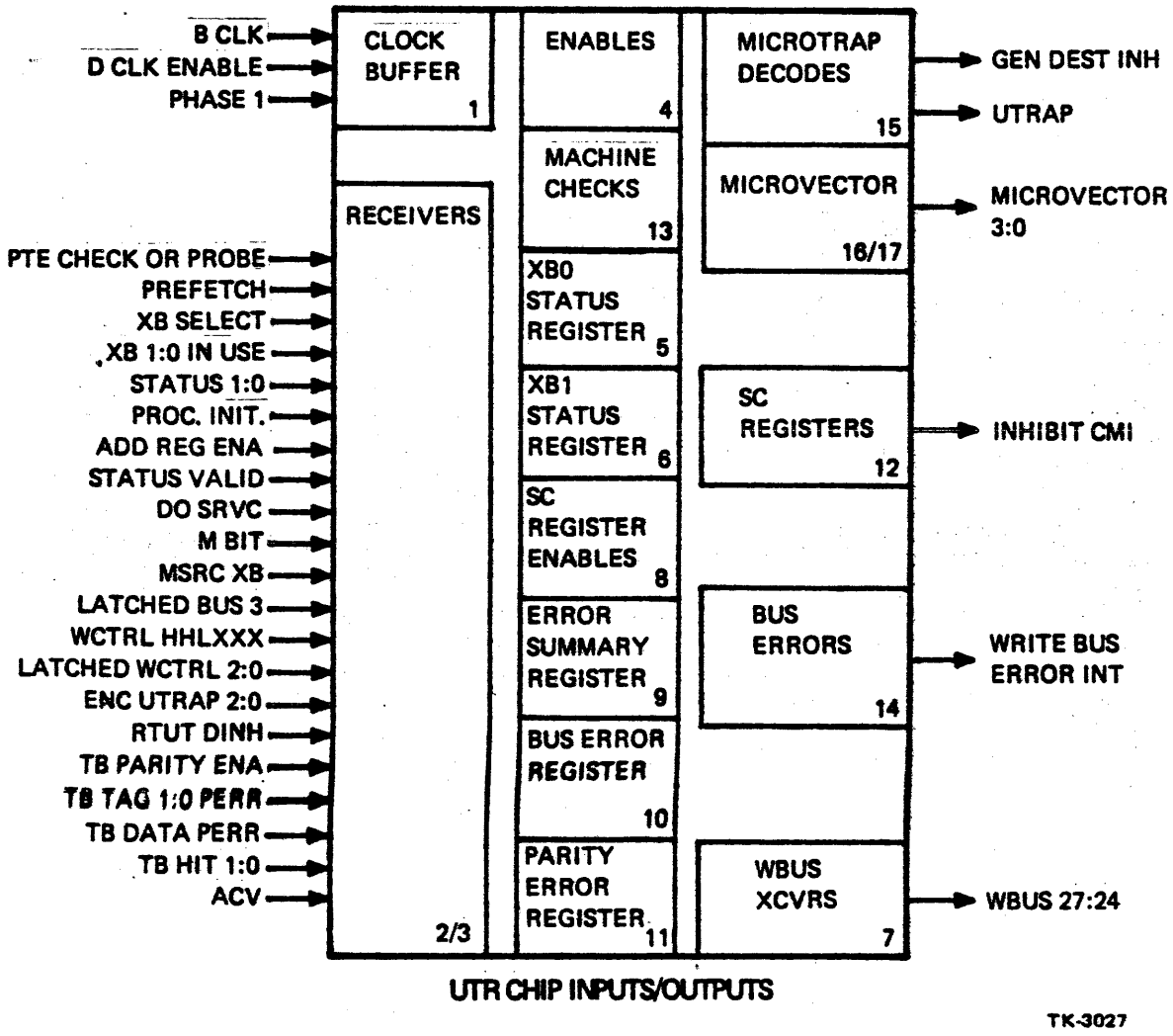


Figure 14-5 UTR Chip Inputs/Outputs

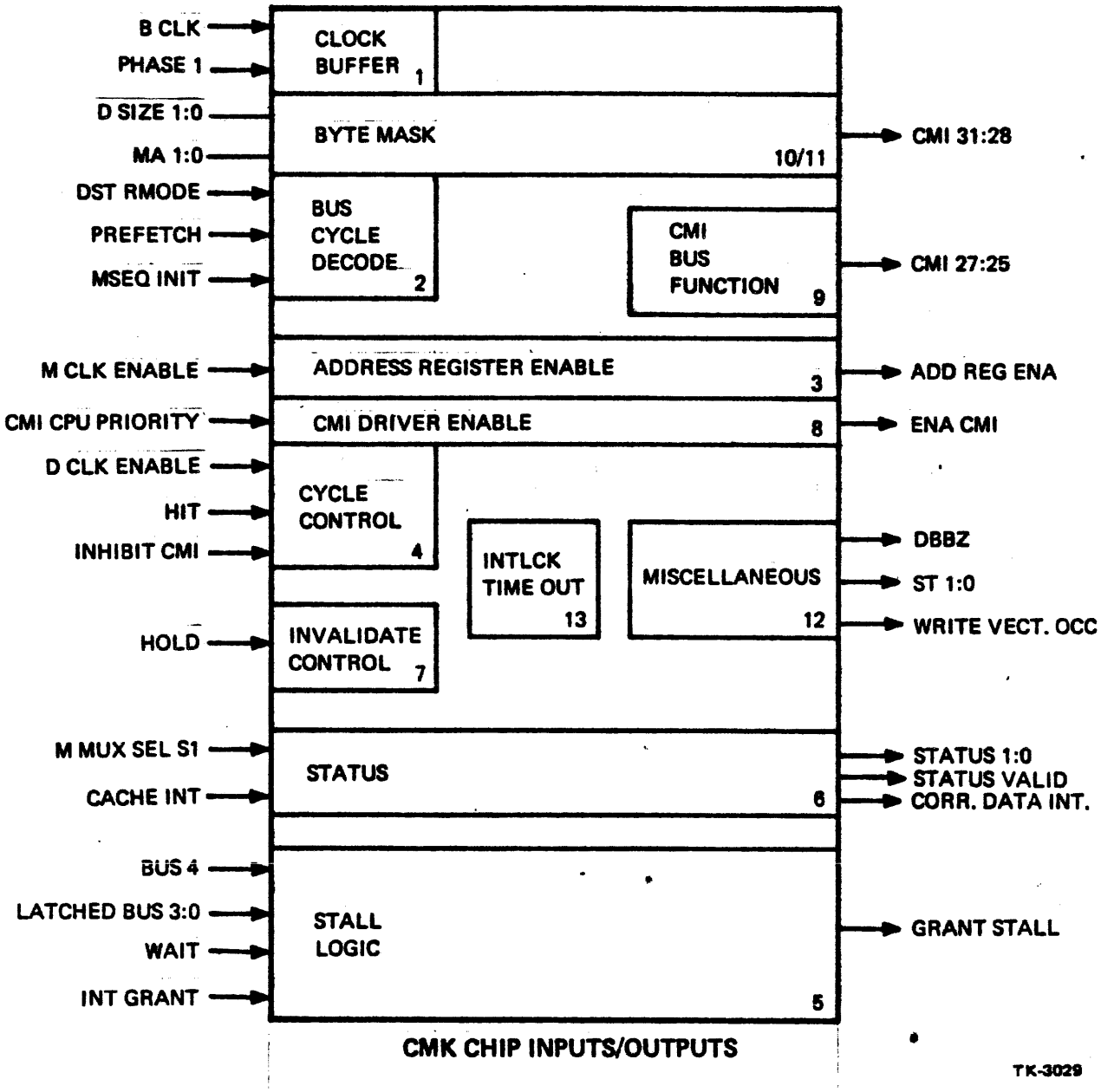


Figure 14-6 CMK Chip Inputs/Outputs

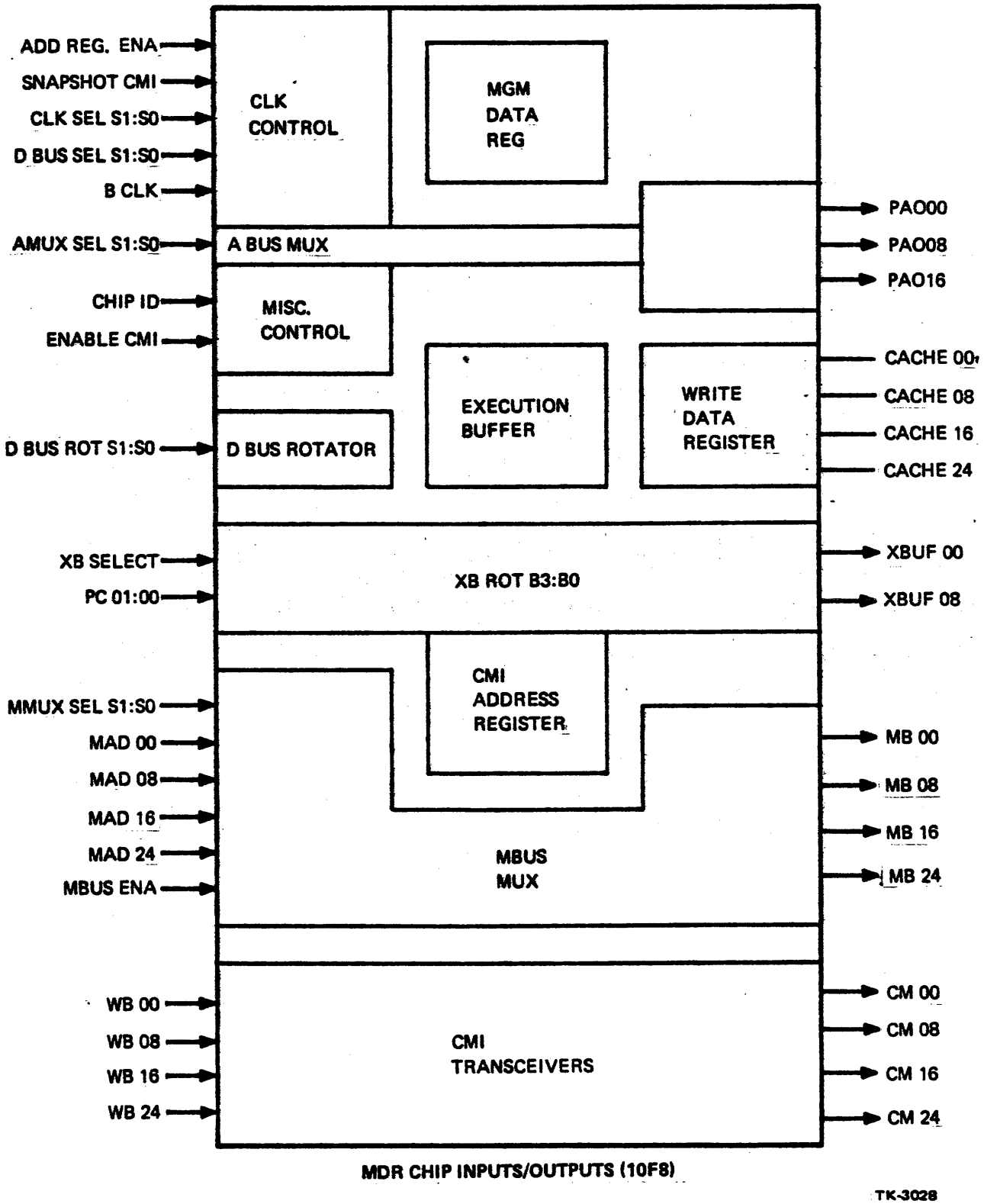


Figure 14-7 MDR Chip Signals Input/Output

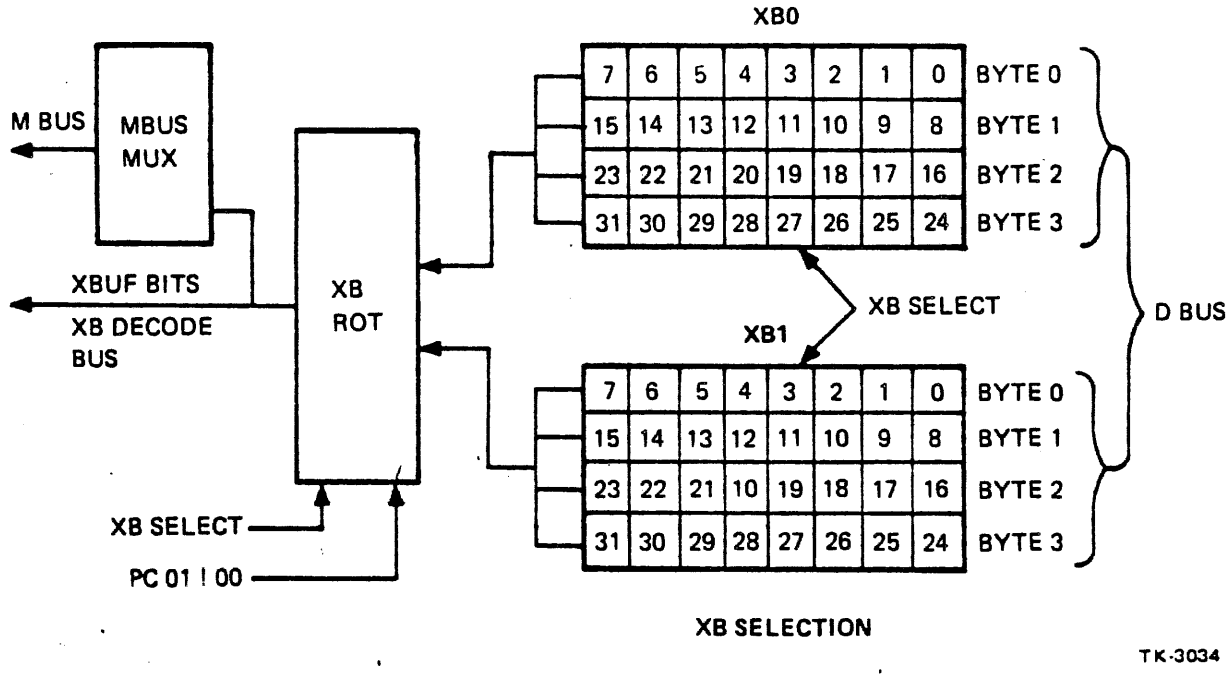


Figure 14-8 XB Selection

XB DATA SELECTION

PC <Ø1:ØØ> H
PC <Ø1:ØØ> H and XB SELECT H determine which bytes of each XB will appear on the XB DECODE BUS, and on the MBUS if the MBUS multiplexer is steered to XB DATA, according to the following chart:

					XB DEC BYTE 1	XB DEC BYTE Ø
XB SEL H	**PC Ø1 H**	**PC ØØ H**	**MBUS BYTE 3**	**MBUS BYTE 2**	**MBUS BYTE 1**	**MBUS BYTE Ø**
LOW	LOW	LOW	XB1 BYTE 3	XB1 BYTE 2	XB1 BYTE 1	XB1 BYTE Ø
LOW	LOW	HIGH	XBØ BYTE Ø	XB1 BYTE 3	XB1 BYTE 2	XB1 BYTE 1
LOW	HIGH	LOW	XBØ BYTE 1	XBØ BYTE Ø	XB1 BYTE 3	XB1 BYTE 2
LOW	HIGH	HIGH	XBØ BYTE 2	XBØ BYTE 1	XBØ BYTE Ø	XB1 BYTE 3
HIGH	LOW	LOW	XBØ BYTE 3	XBØ BYTE 2	XBØ BYTE 1	XBØ BYTE Ø
HIGH	LOW	HIGH	XB1 BYTE Ø	XBØ BYTE 3	XBØ BYTE 2	XBØ BYTE 1
HIGH	HIGH	LOW	XB1 BYTE 1	XB1 BYTE Ø	XBØ BYTE 3	XBØ BYTE 2
HIGH	HIGH	HIGH	XB1 BYTE 2	XB1 BYTE 1	XB1 BYTE Ø	XBØ BYTE 3

IRD1	LDOSR	#BYTES
FALSE	FALSE	0
FALSE	TRUE	1
TRUE	FLASE	1
TRUE	TRUE	2

NUMBER OF BYTES OF I STREAM MSRC# XB

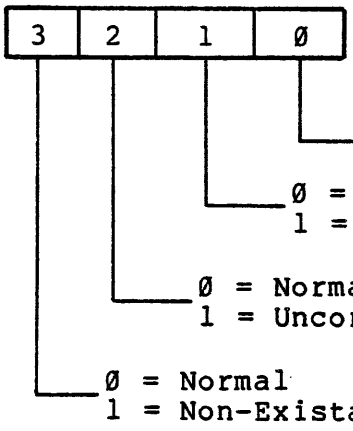
TK-3033

Figure 14-9 Number of Bytes of I-Stream MSRC # XB

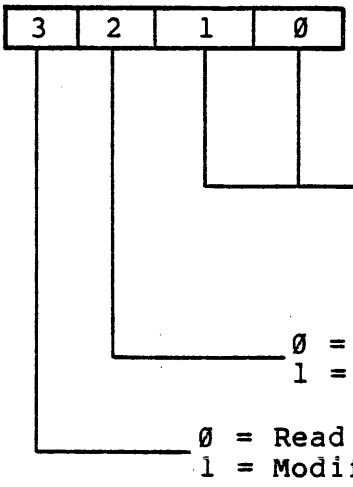
TWO LONGWORD BUFFERS IN MDR CHIPS

X	X	X	X	X	X	X	X	PC JUST LOADED WITH 1. BOTH BUFFERS EMPTY. TWO BYTES OF I-STREAM REQUIRED.
				PC=1				
					LONGWORD ADD=Ø			
X	X	X	X	3	2	1	Ø	LONGWORD AT ADDRESS Ø FETCHED. TWO BYTES OF I-STREAM USED. PC INCREMENTS TO 3. TWO BYTES OF I-STREAM REQUIRED.
		PC=3						
	LONGWORD ADD=4				LONGWORD ADD=Ø			
7	6	5	4	3	2	1	Ø	LONGWORD AT ADDRESS 4 FETCHED. TWO BYTES OF I-STREAM USED. PC INCREMENTS TO 5. FOUR BYTES OF I-STREAM REQUIRED.
PC=5								
					LONGWORD ADD=4			
X	X	X	X	7	6	5	4	FIRST BUFFER NOW EMPTY. LONGWORD AT ADDRESS 8 MUST BE FETCHED TO SATISFY REQUIREMENT.
				PC=5				
	LONGWORD ADD=8				LONGWORD ADD=4			
8	A	9	8	7	6	5	4	LONGWORD AT ADDRESS 8 FETCHED. FOUR BYTES OF I-STREAM USED. PC INCREMENTS TO 9.
PC=9								
					LONGWORD ADD=8			
X	X	X	X	B	A	9	8	ONE BUFFER BECOMES EMPTY. THREE BYTES REMAIN IN OTHER BUFFER. PREFETCHER WILL FILL EMPTY BUFFER FROM ADDRESS C (HEX).
				PC=9				

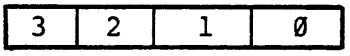
NAME	PR #	MEMSCAR #



NAME	PR #	MEMSCAR #



NAME	PR #	MEMSCAR #



Write Vector Occurred Register — Part of 17 — 2

1 = Vector in MDR

Bit <Ø> is READ/WRITE and is initially Ø. In addition, it is cleared by an INTERRUPT GRANT bus function and set by a WRITE VECTOR transaction on the CMI or by a READ LOCK TIMEOUT.

Miscellaneous Registers

Memory Interface Connect Control Logic General Description

Six 48 pin gate array chips and associated logic chips working in conjunction with each other monitor WCNTRL, bus function and MSRC fields from the microword to control address and data gating, aligning and mutliplexing to and from the data path module and CMI bus. During this gating, aligning and multiplexing, the chips will also monitor the condition of the operation being performed during the execution of this microinstruction that may cause a microtrap condition to occur. If this happens the control logic will generate the proper microaddress to respond to the microtrap condition generated.

The basic functions the control logic must perform are related to the six control chips that are located on this module. They are:

1. Address control chip (ADK) - Drives multiplexing and gating of address from ADD chip and helps control transfer through MDR chip. Also used for controlling translation buffer group disables or group displacement.

2. Prefetch control chip (PRK) - Controls prefetching of I-Stream data from memory to be brought to execution buffers XBØ or XB1 independently of microcode. PRK also used in conjunction with ADK to generate or load proper address from/to ADD chips.

3. Cache control chip (CAK) - In conjunction with ADK chip controls disabling and enabling of cache. Controls driving and receiving of data to and from MDR and cache. Monitors CMK chips snapshot CMI output for invalidation of cache on CMI writes.

4. Access control violation (ACV) - Monitors and generates ACV signal for access violations and translation not valid during TB usage or PTE checks and probes on WBUS. Monitors and generates codes to microtrap chip in privileged sequence for the following microtrap conditions.

    1. Control store parity error
    2. FPA received operand
    3. Unaligned unibus data
    4. Write crossing page boundary
    5. Write unlock crossing page boundary
    6. Unaligned data, write unlock
    7. Unaligned data

5. Microtrap chip (UTR) - Monitors microtrap conditions during microinstruction and generates encoded microvector bits <3:Ø>. These bits are used in conjunction with bits <5:4> from microsequencer chip on DPM to generate 6-bit microaddress. This address points to the proper microroutine to handle the microtrap condition decoded by microtrap chip. Also monitors status from CMK chip to generate write bus error interrupt to interrupt chip on UBJ module.

6. CPU memory interconnect control chip (CMK) - Monitors and transmits control signals to and from CMI bus. These signals are DBBZ and Hold. Transmits and monitors byte mask and function bits <31:25> onto/from CMI bus. Monitors status lines of CMI. Generates corrected data interrupt to UBI module. Generates grant stall in response to INT Grant from UBI module to stall microcode during an interrupt from UBI module.

As we continue on to the discussion of the MIC you must bear in mind that it monitors actions being performed by the same microword. That is, it will be monitoring the same WCNTL lines, bus function lines and MSRL lines so they may work together to perform the function specified by the microword. All WCNTL lines and bus function lines from CSS module are latched on low to high transition of M CLK and feed to needed locations on MIC module. M CLK Enable H, B CLK L, D CLK Enable H and Phase 1 H will also be used internal to all control chips for transmitting and receiving purposes along with loading or reading internal registers (Ex: MEMSCAR).

Following are the selected bus functions and their definitions:

BUS FUNCTIONS

A 5-bit microfield is required to specify which of the following bus functions is to be performed during each microstep: (BUS CONTROL CODES IN PARENTHESES ARE IN HEX)

(Ø7) NO FUNCTION
(1Ø) READ

Replace the contents of the MDR with the contents of the memory location specified by the virtual address presently in the VA and DSIZE.

(14) READ WITH MODIFY INTENT

Checked for WRITE access. Otherwise, same as READ unless the resulting physical address is in UNIBUS space, in which 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.

(Ø2) READ, NO MICROTRAP

Same as READ, but suppress ACV 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 microseconds of the time it was initiated, a READ LOCK TIMEOUT will occur. The READ LOCK operation will be aborted, a NONEXISTENT MEMORY machine check will occur, and the WRITE VECTOR OCCURRED bit will be set in the appropriate status/control register.

(ØØ) 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.

(Ø6) 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 which was not previously fetched should be clocked into the MDR.

(ØB) WRITE UNLOCK, SECOND REFERENCE

See WRITE UNLOCK and WRITE, SECOND REFERENCE

Note

There exists in the memory interface control logic a "CMI Write Size Latch" which is used in performing certain write bus functions.

There are actually three categories of write bus functions:

1. Those which load the "Write Size Latch"
2. Those which use the latched size
3. Those which always write all four bytes regardless of DSIZE

Category 1 includes:

WRITE
WRITE IF NOT RMODE
WRITE UNLOCK
(WRITE LONGWORD)

Note

WRITE LONGWORD causes the "Write Size Latch" to be loaded with DSIZE, but always writes all four bytes.

Category 2 includes:

WRITE, SECOND REFERENCE
WRITE UNLOCK, SECOND REFERENCE
WRITE, NO MICROTRAP

Category 3 includes:

WRITE PHYSICAL ADDRESS
WRITE LONGWORD, NO MICROTRAP
WRITE LONGWORD

The "Write Size Latch" is loaded with DSIZE during any microstep which specifies a Category 1 write bus function, regardless of any destination inhibits or microtraps which might occur during that microstep.

(Ø4) READ LOCK TIMEOUT TEST

(Special function for testing timeout counter in MDR chips.)

(18) WRITE

Replace the contents of the memory location specified by the virtual address presently in the VA and DSIZE with the contents of the WDR.

(1A) WRITE IF NOT REGISTER MODE

Same as WRITE unless RMODE (REGISTER MODE) from the microsequencer is asserted, in which case do nothing.

(19) WRITE LONGWORD

Same as WRITE, except the two least significant bits of the address are ignored. (For FIELD instructions.)

(ØC) WRITE, NO MICROTRAP

Same as WRITE, but suppress ACV, unaligned data, and page boundary crossing microtraps.

(ØE) WRITE, NO MICROTRAP, LONG

Same as WRITE, NO MICROTRAP, except that a longword is written ignoring the latched write size. Used for writing the Mbit during mapping subroutines.

(1B) WRITE UNLOCK

Same as WRITE. In addition, releases the interlock set by a READ LOCK operation.

(Ø8) 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.

(ØA) 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 which was not previously stored should be written into the specified memory location.

(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.

(Ø9) REI CHECK

Check for:

Saved PSL <current mode> GEQU ASTLVL and Saved PSL <IS> = Ø

and check the saved PSL (on the WBUS) against the PSL for any of the following conditions. Indicate the results of the checks on the microvector lines:

1. Saved PSL <current mode> LSSU PSL <current mode>
2. Saved PSL <IS> EQLU 1 and PSL <IS> EQLU Ø
3. Saved PSL <IS> EQLU 1 and saved PSL <current mode> NEQU Ø
4. Saved PSL <IS> EQLU 1 and saved PSL <IPL> EQLU Ø
5. Saved PSL <IPL> GRTU Ø and saved PSL <current mode> NEQU Ø
6. Saved PSL <previous mode> LSSU saved PSL <current mode>
7. Saved PSL <IPL> GTRU PSL <IPL>

(Ø3) I/O INITIALIZE

Generate UNIBUS INIT.

(Ø1) PROCESSOR INITIALIZE

Generate a reset signal which initializes status/control registers.

(ØF) INTERRUPT 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 will stall the processor 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 will cause an interrupt vector to be written into the MDR. If so, a status register bit will be set.

(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 microorders:

```
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 (PHYS. ADD.)
```

On PROBE the microvector lines are:

```
MICROVECTOR <3> = (PBOK .AND. V .AND. AC) .OR. PA
MICROVECTOR <2> = M .AND. ((V .AND. AC) .OR. PA)
MICROVECTOR <1> = V .OR. PA
MICROVECTOR <Ø> = (AC .AND. V) .OR. PA
```

On PTE CHECK the microvector lines are:

```
MICROVECTOR <3> = Ø
MICROVECTOR <2> = M .AND. V .AND. AC
MICROVECTOR <1> = V .AND. AC
MICROVECTOR <Ø> = 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.

The last group of microorders from the microcode that the MIC module needs for performing its functions are the MSRC group bits <68:64> from the microcode. The following are the MSRC codes required for the MIC module. The MSRC assignments in parenthesis are in Hex.

(12) MBUS <- MDR
(13) MBUS <- WDR
(17) MBUS <- XB
(18) MBUS <- MA
(19) MBUS <- PC SAVE
(1A) MBUS <- PC
(1B) MBUS <- VA
(1F) MBUS <- TB Data (Address in VA)

Note

Bits <31:24> will always read as ones.

There are 1Ø registers internal to 3 control chips that are designated as status and control registers (S/C). The microcode reads from and write to these registers by loading a 4-bit S/C register (not included in the 1Ø S/Cs) called S/C address register using WCNTL lines and WBUS. These registers will be referred to in the microcode as MEM S/C REG numbers, yet may be included in a different numbered Internal processor register (IPR) discussed previously. The following registers, location, S/C register numbers and IPR6 numbers were discussed previously.

Another important group of controls to be used by the MIC module are the WCNTRL lines from your microword (bits <3Ø:25>). these lines control the source and destination of data and address.

the following wbus control codes are required for the memory interface: (wctrl assignments in parentheses are hex)

(2Ø) 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 <- Ø
(28) TB DATA <- (WBUS)
(29) TB VALID BIT <- Ø
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 <- Ø
VA <- (WBUS)
(Invalidate both groups at the index position addressed by VA. The address in the VA register will be interpreted as a physical address.)
(2E) WDR <- (WBUS)
(2F) MDR <- IR, ZERO EXTENDED
(3Ø) STATUS/CONTROL REGISTER <- WBUS<27:24>
(31) PREVIOUS MODE REGISTER <- WBUS<23:22>
(32) WBUS<27:24> <- STATUS/CONTROL REGISTER
(33) BUS GRANT
WBUS<2Ø:16> <- IPL OF CURRENT UNIBUS GRANTEE
(34) STATUS/CONTROL ADDRESS REGISTER <- WBUS<27:24>
(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<2Ø:16>
(3D) IPL REGISTER <- WBUS<2Ø:16>
(3E) RESERVED
(3F) WBUS<2Ø:16> <- IPL OF LAST UNIBUS GRANTEE

There are two ways to read from memory

1. Read bus function (microword dependent)
2. Prefetch (microcode independent)

You may write to memory only under a write bus function (microdependent).

The following functions read, write may create what is called a "bus cycle" decode from the microword. A definition for "bus cycle" would be the starting of and ending of retrieving data from a location or depositing data to a location. On a read from memory at address 1000 a bus cycle may include going out on the CMI to memory or retrieving the data needed from cache which would not need a CMI function to memory. This fact in itself shows that a definite period of time cannot be assigned to the term "bus cycle". You can, by use of signal names, give the term "bus cycle" a relative time period.

A bus cycle starts at "Address Register Enable" and ends with "Status Valid". This means a bus cycle starts when the address needed for read or write is enabled into CMI ADDRESS REG on MDR. It ends upon the CMK chip receiving status from CMI or data from CA received with no errors.

The term "bus cycle" will be used only when referring to microdependent bus functions that are decoded to need a "bus cycle". The same function of reading from memory of cache may be accomplished by prefetch. That is, Add REG ENA and status valid signals are used to perform the read start and end bracketing. Since prefetch is microcode independent and it uses the same path as a bus function read or write, they cannot be done at the same time. Therefore, "bus cycle" and prefetch use the same basic path for address and data, but at different times.

Using the instruction MOVL (R1) (R2) as an example for control logic in the MIC module includes a lot of functionality for the module.

1. You must read from memory at address in GPR#1 and store it.

2. You must take the data you read and write it to address stored in GPR#2.

Independent of this instruction are the functions of cache enabled or disabled translation buffer enabled or disabled and the fact that you might prefetch I-Stream data from memory to the execution buffer.

Along with these actions, a write to memory may occur on the CMI causing the MIC module to snapshot the CMI. This being, taking address that was written on CMI by someone other than CPU and checking data cache for a hit and invalidating if a hit.

You may also generate a microtrap condition during the reading or writing to memory that may cause you to leave the microsubroutine that handles the MOVL (R1) (R2) instruction, to make sure the instruction is completed properly (Ex: TB) miss may perform address translation and return to finish instruction. It is also possible the instruction may not be able to be completed even with microtrap intervention and you would have to notify the operating program. Ex: Access control violation needs macrointervention.

These are the functions and different conditions we will cover here. To perform this we will take the microinstruction through each microword needed to perform the macro MOVL (R1) (R2) using the needed fields in the MIC module.

Bus function
WCNTL
MSRC

Following the four microwords needed to perform the MOVL instruction we'll take it through all four and explain what needs to be done in the MIC module for each one.

MOVL (R1) (R2) Take data longword from address in R1 and move it to address in R2.

First Microword

```
OS.RED 108 address
FPA_Q_M[MDR] VA_R[GPR.R]     (RN) register deferred mode
CLOBBER MTEMPØ REF           operand address is GPR
NEXT/OS.READ.EXIT            (RNUM) put garbage in
                             MTEMPØ.
```

What the MIC module must do in the first microinstruction during the first half or phase 1 H is

1. Source the contents of MDR to the MBUS to be stored in Q register. The contents of MDR for this particular instruction are meaningless but the microinstruction is used by more macroinstructions than MOVL long and considerations must be taken.

2. During last half or phase 2 it loads the address from GPR#1 via the WBUS to the VA Register in ADD chip.

Second Microword

```
OS.READ.EXIT 1ØE
READ.SIZE [IDEP]            Read memory at VA size
IRDx[1]
```

During this micro the MIC module must

1. Take address from VA register and retrieve the data from either main memory or cache and store into the MDR (Reg). (We now have data from address R1.)

2. Make available next byte from execution buffer being used to DPM to decode operand specifier (R2) and DPM generates next microaddress to go to and update PC.

Third Microword

```
OS.WRT2 158
Q_M[MDR] VA_R[GPR.R],       (RN) Register deferred mode
CLOBBER MTEMPØ DEF.         GPR (RNUM) is operand
IRDx[1]                     address. Put garbage in
                            MTEMPØ.
```

MIC Function

1. Source MDR to MBUS to be stored in Q by DPM.

2. Load address from GPR#2 via WBUS to VA Reg. in ADD chip.

Fourth Microword

```
IL.MOV.B.W.L.MEM (OR MOVA)
R[DST.R].SIZ - Q Q - D, WRITE NOT REG
SIZE[IDEP], CCOP2, IRD1
```

MIC Module

1. Takes address from VA register and data from WBUS and writes to memory (and cache if a hit) depending upon instruction size using write not Reg.

2. Make available next two bytes from execution buffer for decoding of next instruction and update PC.

Now that we have the overall picture of all four microwords let us take a look at each one and see how the MIC module performs them.

First Microword - OS.RED

```
FPA_Q_M[MDR] VA_R[GPR.R]
CLOBBER M TEMP Ø DEF.
NEXT/OS.READ.EXIT
```

Decoded MSRC Field = 12 or MBUS <- MDR

Decoded WCNTL = 25 or VA <- WBUS

Decoded Bus Function = 7 or No Op.

As we said previously, the two functions performed by the MIC module for this microword are:

1. Source contents of MDR to QMBUS
2. Load address from GPR#1 to VA Register in ADD chip

These are done at different halves of the microword, since it is possible to take data fromMDR and have DPM work on it and send it back to MIC. This does not happen in this instance but it is possible.

Phase 1

1. MDR (Reg) sourced to MBUS on Phase 1 H due to MUX Sel 1 H being L (L=Ø) and MSRC 2 from CSS being L (L=Ø). "MUX Sel 1 H" = 1 comes from PRK chip due to the fact that no other function needing the MBUS MUX was decoded and MSRC line 2 was latch as Low (false).

2. VA Reg is loaded on Phase 1 L from WBUS in ADD chip due to:

   A. ASRC Sel line 2 is H coming from miscellaneous control saying WCNTL line 1 latched to Ø (Low) and Phase 1 L causing WBUS through AMUX input to adder (ASRC lines Ø,1 can be anything).

B. ADK chip decoded WCNTL as WBUS -> VA and selecting B SRC Sel lines SØ, S1 to S1/Ø SØ/Ø (Ø = Low) selecting Ø input to BMUX of adder. WBUS + Ø = WBUS.

C. ADK also sending "ENA VA L" to latch output of adder to VA because of decoded WCNTL.

Second Microword

As we see, the second microword must read from the address specified in VA. The read may be performed from cache (if enabled and data is available) or from main memory. To show this we have what happens during a cache hit and a cache miss (both with TB off).

The second function of updating the PC due to DPM taking next operand specifier will be covered under the prefetch area.

Basic Read Cache Hit, TB Off

Microword at address 1ØE:

OS.RED - READ EXIT: READ SIZE [IDEP] IRDx[1]	Read memory from address in VA. SIZE = DSIZE Ex:VA = 1ØØØ Size = Longword

Bus Func = Read, Meaning - Memory read from VA and DSIZE from define file and placed into MDR

WCNTL = From Define File - No Op

MSRC = From Define File - Default M Temp

As you can see there is no mention of cache or translation buffer within the microword. You will not see any with the exceptions of invalidating TB or cache, making checks of TB for access or validity but never in conjunction with normal operations on a read or write.

Following this microword through the MIC block diagram we will find how the address is sent and data is received.

Five of the six control chips (not microtrap chips) monitor the bus function and checked to see if any other functions that may be using the needed paths are being done i.e., prefetch, snapshot CMI for invalidating cache, for example. If the paths that are needed are free a Bus Cycle is started within the MIC module.

We must take the address from the VA register (1ØØØ) through the memory address latch on the ADD chip to the memory address lines. (PRK chip decodes read using address in VA) PRK sets MA Sel lines to S1/1 SØ/1 with a 1 = a high true allowing VA through MA latch.

The address is now on the memory address lines and must get onto the physical address bus to be sent to cache. The ADK chip also decoded read and ORed the fact that memory management had been disabled to get the AMUX Sel line SØ, S1 to S1/1 SØ/Ø (1 H line). This allowed bits <Ø2:23> of the VA to be fed onto the physical address lines (PADs).

Note

Bits <Ø1:ØØ> not used as we read only longwords.

The address (bits <Ø2:23>) are now sent to cache and are checked for a hit, validity and errors. At the same time this is being done the same address from PAD is being latched into the CMI address register when "ADD REG ENA L" comes from CMK chip. (This says start bus cycle.) The latching into the CMI address register is done in case of a cache miss and we need to go to main memory. In this case it will not be used.

If the cache has a hit the signal cache hit is sent to the CAK chip and the CMK chip.

When sent to CAK it is ORed with the fact there are no errors (the signals Tag or Data Parity errors are false) and the fact that cache is enabled (from S/Ø Reg #6). This will

Hit No Errors Cache Enable	Enable needed byte outputs

allow the CAK chip to use the monitored DSIZE and VA lines to generate the proper byte enables. These byte enables go to the cache data store and allow the data selected by PAD to be outputted from the RAMs. In case of longword at 1ØØØ Enable byte Ø and 1, 2, and 3 would be used.

The cache hit signal also goes to CMK chip along with cache INT from CAK. (Cache INT meaning Tag or Data Parity errors are not valid if not a hit). These signals will be used to stop a CMI cycle to memory by the CMK chip not sending ENA CMI to MDR chip and generate status valid for stopping bus cycle in MIC.

The data outputted from the cache must be sourced onto the D Bus in the MDR chip and stored into the Memory Data Register (MDR Reg). To do this the ADK chip, when it decodes read function and no forced CMI, will steer the cache data to the DBUS by using DBUS SØ, S1 (S1 ANDed with fact no write vector occurred) to give us S1/Ø SØ/1 (1 = High) selecting the MDR (Reg) to CLK in data from DBUS output of rotator. The D Bus rotator used inputs from CAK chip which said no rotation or DBUS Sel SØ, S1 = S1/Ø SØ/Ø (1 = High).

The microword has now performed the function of read memory at address 1ØØØ and stores data in MDR. What you just read was the functionality and signals needed to accomplish this bus cycle. The timing is shown on attached sheet.

Basic Read Cache Miss, TB Off

Microword at Address 1ØE:

OS.READ - READ EXIT:	Read Memory from Address
READ SIZE [IDEP]	In VA. SIZE = DSIZE
IRDx[1]	EX:VA = 1ØØØ Size=Longword

Bus Func = Read, Meaning - Memory read from VA and DSIZE from define file and placed into MDR.

WCNTL = From define file - No Op

MSRC = From define file - Default microtrap

What must be done in this example is to read data from address in VA and store it in MDR. This is the same as a cache hit in overall functionality but the data comes from main memory and not cache. Also, we must write data from 1ØØØ back to cache when received from memory.

Following the microword through the MIC block diagram, we will find how the address is sent to main memory and data received and stored into MDR and cache.

Five of the control chips (not microtrap) monitored the bus function and checked to see if any other functions, that may be using the needed paths are being performed, that is, prefetch, cache invalidate due to snapshot. If all paths that are needed are available we will continue and start a bus cycle within the MIC module.

We must take the address from the VA Register (1ØØØ) through the memory address latch on the ADD chip to the memory address lines (MAD). To do this the PRK chip has decoded the read function using VA and sets MA SEL SØ, S1 to S1/1 SØ/1 (1 = High) allowing VA through MA Latch to MAD.

With the address now on MAD we must get the address to the PAD via the PA MUX. The ADK chip also decoded the read function and memory management disabled to set the AMUX SEL lines SØ, S1 to S1/1 SØ/Ø (1 = High). This allows bits <23:Ø2> to pass to the PRD lines.

Note

Bits <Ø1:ØØ> not used as we used only longwords.

Address bits <23:Ø2> are being fed to cache and written to the CMI address register. The cache is checked and no hit is recorded. The address is also being latched into CMI address register when "Add Reg Ena L" comes from CMK chip. This says start MIC module bus cycle.

THe fact that no hit signal was received from cache causes the CAK chip to not send ENA bytes to cache which would enable cache to output data.

The fact that no hit signal goes to the CMK chip allows the CMK to arbitrate for CMI bus and send "ENA CMI L" to the MDR chip when bus won. This signal will be used to pass the address in CMI Address Reg (1ØØØ) to be sent onto the CMI bus <23:Ø2>. At the same time the CMK chip is going to send out the remaining functions needed to read from the address (1ØØØ) in main memory, i.e., function of read on lines <27:25> ØØØ and byte mask on lines <31:28> 1111. Also sets DBBZ on CMI. The CMK diagnostic DBBZ and memory asserts DBBZ and sends needed data from 1ØØØ. When memory has data and status on lines (CMI lines) the memory diagnostics DBBZ. This tells CMK chip to monitor "status line <Ø1:ØØ>" for possible error. Assume no error. The data on the CMI lines are received by MDR chip.

At this point the ADK chip has passed the time to where cache -> DBUS was needed and asserts the DBUS Sel lines SØ, S1 to S1/Ø SØ/1 (1 = High) which selects data from CMI onto the DBUS. The ADK also knows that the data that is on the DBUS must be latched into MDR (Reg). So it

generates CLK Sel SØ, S1 to S1/1 SØ/1 (1 = High) to perform this. Also because the MDR was selected and CMI data was selected to the DBUS the WMUX steers the data from the DBUS into the WDR (used to store data to be written to cache).

The basic function of retrieving data from 1ØØØ is complete, but we must store that data into cache. We still have the address in the CMI register and the data is in the WDR. Let's do it!

The CMK chip monitored the status lines <1:Ø> and found status valid and outputs that signal to the control chips. When the CAK chip sees the data is correct by receiving status valid, it says cache GR Ø WR EN to allow cache to be written. To have the cache written to we need to get the address through the PA MUX and data from WNR. Data is easy because as long as D Bus Sel does not have cache selected to D Bus the drivers are driving the data from WDR to cache. Because "Add Reg L" is now High from CMK and no other function is needed in the ADK chip, the ADK outputs AMUX Sel to select CMI address register and the drivers pass the CMI address (1ØØØ) to cache to be used with cache GRØ and data to write into cache the data retrieved from memory. Now all the functions of a read with cache miss are complete and another microword may be worked on. The basic timing is on attached sheet.

Third Microword

One way or the other we now have the data, from the address that was stored in R1, latched into the MDR (Reg). The third microword now must do two things:

1. Store data that is to be written into a Q Reg.

2. Load address from GPR#2 to VA Reg. in ADD chip.

OS.WRT2 158

Q_M[MDR] VA_R[CDR.R] CLOBBER MTEMPØ DEF IRDX [1]	(RN) Register mode differed. GPR (RNUM) is operand address. Put garbage in M Temp Ø.

Decoded MSRC Field = 12 or MBUS <- MDR

Decoded WCNTL = 25 or VA <- WBUS

Decoded Bus Function = 7 or No Op

As you can see, this function is very similar to the first microword. It uses the same signals and paths to perform the function. All we are doing is setting up to perform a write from address in VA instead of a read.

With Microword OEE

IL.MOV.B.W.L.MEM (also used for MOVA) R[DST.R].Size_Q Q_D, Write not Reg, Size [IDep], COOP2, IRD1	Write data stored in Q to address in VA. (Ex: VA = 2000)
Bus function decode = 1A or	Write to memory (and cache) if register mode not decoded. If register mode is decoded MIC will not do anything, the DPM mode takes data from Q register to R2. In case Reg mode not decoded start "bus cycle" for write using address in VA register.
WCNTL = 2E or	Data from WBUS -> WDR rotated for longword alignment.
MSRC = 0 or	M Temp 0 -> MBUS (no function for this macro).

We will say code is enabled and a hit is recorded. That means the data to be written will be written to cache and main memory.

Following the microword through the MIC block diagram we find we have to worry about transferring data and an address at the same time.

Five of the six control chips (not microtrap) monitor the bus function which will be used to open paths for the address and help start the CMI write function. All chips monitor the WCNTL lines and then in effect will open paths for the data to be transferred.

For ease of understanding we will take the address from VA to CMI address latch and then bring the data from WBUS to WDR even though it is happening at the same time.

We take the address from VA (address originally in R2) through MA MUX of ADD chip by PRK chip decoding a write and setting the MA Sel lines S1, SØ to S1/1 SØ/1 (1 = High) allowing the contents of VA to pass through to MAD bits.

We must now get address from MAD to the physical address (PAD) lines to be sent to cache and CMI address register. Only bits <23:Ø2> go to the PAD. To do this the ADK chip also decoded the write and ORed the fact memory management was disabled to set the AMUX Sel lines S1, SØ to S1/1 SØ/Ø (1 = High). This allows bits <23:Ø2> to be passed to the PAD to be sent to cache and check for hit and be received at the CMI address register. When the CMK chip decoded bus function and determined that no other function was needed, the same path bit (CMK) asserted "ADD REG ENA L" to start "bus cycle" and latch address into the CMI address register.

To get the data to the WDR register we must take it from WBUS through the DBUS rotator through WDR MUX.

The ADK chip which selected MAD -> PAD also selected the WBUS to DBUS by outputting DBUS Sel S1, SØ to S1/1 SØ/Ø (1 = High). While this is being performed the CAK chip selected the DBUS rotator to pass the bytes through as they were 3, 2, 1, Ø by writing DBUS ROT S1, SØ to S1/Ø SØ/Ø (1 = High).

Internally to the MDR chip because CMI data to DBUS was not selected, DBUS ROT was steered through the WDR MUX. When the ADK selected MAD -> PAD and WBUS -> DBUS it also was setting CLK Sel Line S1, SØ to S1/1 SØ/1 (1 = High) to latch data from WDR MUX into WDR on first L -> H transition of B CLK.

The point at which we are now, data in WDR and address in CMI takes one microcycle (two B CLKs). We must now write to memory and to cache. What would happen when the next microword wanted to use some portion of the MIC that we still must use? Well, since we set "Add Reg Ena L" and have not received "status valid" we are still in a MIC module "bus cycle" and if the control chips noted the next microword wanted the paths needed it would stall the processor. Stalling the microcode will be covered later. We will assume the MIC is not needed so a stall will not occur.

With data and address where we want it we will send the address and function/byte mask onto the CMI at the same time we write data to cache because of the bit we received.

We will discuss the CMI function first. When the CMK chip first decoded a write to memory it knew it would have to generate a "CMI Bus Cycle" (the CMI bus cycle may be a part of MIC module bus cycle) so it would monitor the signal input "CMI CPU PRI L". When this occurs and data address is set the CMI would assert DBBZ to the CMI along with write function bits <27:25> and byte mask bits <31:28> to the proper values at the L -> H transition of B CLK. At the same L -> H transition of B CLK and ENI CMI from CMK the MDR drives the output of CMI address register to CMI. All those signals stay asserted until the next L -> H transition of B CLK. At this time CMK deasserts DBBZ and the WDR (data to be written) is asserted on CMI. The memory controller will assert DBBZ if not able to write at this time and deassert DBBZ and send status when it is able. It is also possible that the controller was able to write the data immediately upon receiving it and not assert DBBZ but assert status. Whichever happens the chopping of DBBZ causes the CMK chip to stop sending "ENA CMI L" to MDR chips and monitor the status line on CMI. With status valid received we end the MIC module "bus cycle" and allow the microcode to continue if it was stalled.

While the above was happening we were also writing the data to cache at the address specified from CMI address register. We did this by using the same signals that we need on a cache read miss, but they were generated by the decode of a write bus function and a cache bit.

We have read about reads and writes with cache off and on, yet we have not mentioned the translation buffer (TB). What is it and what does it do? It is used to store translated addresses for the use of the system. Page frame numbers of virtual addresses are used to search for page table entries stored in TB tag on 1. These PTEs if found may use the virtual address to source a physical address onto the physical address (PAD) lines to be used in the same way a physical address was used to reads or writes previously mentioned.

The main difference between virtual address and the use of microcoded read physical, write physical or using VA and PC with memory management off is where the address goes when leaving the MA Latch.

The major difference comes from the ADK chip and MDR chip. Normally with memory management off the address flows (bits <31:00>) through the MA Latch and all bits except <01:00> are passed through the AMUX on MDR to physical address MUX.

READ or WRITE with Memory Management Enabled

Translation buffer (TB) is enabled when memory management bit set in physical/virtual address register = S/C Reg Ø in ADK chip. Microcode sets bit in S/C Reg Ø due to VAX instruction MTPR #38 the #1.

With this bit set and bus function not read or write physical the TB is enabled to output:

1. Address onto PAD bits <23:Ø9>

2. Data and tag points errors and access control violations

3. Hit or miss

If mm was not enabled or read or write physical was decoded and mm was enabled you would still address the TB and check for hit or miss but:

1. Address would not be allowed to be outputted to PAD

2. Tag Parity errors and access violations would be sent to ACV or microtrap chip but not used. Data parity not sent

3. Hit or miss would still be sent to microtrap chip but not used because TB parity error not available. (TB Parity ENA generated by mm enabled and no read or write physical from ADK)

By looking at Figure ? let us take an example of one of the microwords we have used previously. Read with cache hit, this time TB enabled

```
OS.RED.EXIT 1ØE          Read memory at VA size IDep
Read.Size [IDEP]
IRDx[1]

Bus Function  = Read     Memory read from VA and DSIZE
                         and placed into MDR

WCNTL = No Op

MSRC = Default M Temp Ø
```

What must be done is to take address from VA register. Use this address to retrieve data and place data in MDR.

Five of the six control chips (not microtrap) monitor the bus function and check to see if any other functions that may be using the needed paths are being performed, i.e., prefetch, snapshot CMI. If no other functions are being performed the control chips are free to start a MIC "bus cycle".

We must first take the address from VA register in ADD chip through MA MUX to MAD. This is done by PRK having decoded read bus function and setting MA Sel lines S1, SØ S1/1 SØ/1 (1 = High) allowing VA through MA Latch to MAD lines.

Now we have to go into parallel actions.

1. Pass only bits <Ø8:Ø2> through AMux to PAD and

2. Check for TB to perform its function and put bits <23:Ø9> to PAD.

We will take the AMUX function first. With ADK chip decoding a Read function MM enabled the selection of AMUX Sel S1 and SØ = S1/Ø SØ/Ø (Ø = Low). This would normally be seen as selecting the CMII address register to PAD bus.

But when used in conjunction with DBUS Sel lines only the MA lines <8:ØØ> are enabled and driven to PAD.

To do this the ADK chip has also selected DBUS Sel SØ S1 to S1/Ø SØ/Ø say cache to DBUS. When this happens along with AMUX Sel S1, SØ being S1/Ø SØ/Ø, the AMUX passes whatever is on MAD lines <Ø8:Ø2> to PAD.

This is a special function used in MDR for this case. At this time of the microword DBUS Sel would be selecting cache to DBUS normally. CMI ADD Reg would only be used through the AMUX on a cache replacement if a miss or write to TB, which would both involve no cache -> DBUS selection.

Now for the TB function that was happening at the same time. As stated previously the address lines <31 and 15:09> always feed the TB and generate a hit or no hit signal. When MM is enabled (S/C Reg #0 in ADK) and a hit is recorded in the TB tag store the signals TB "OUTPUT ENA L" and "TB PARITY ENA H" are generated.

1. "TB OUTPUT ENA L" is used to send to TB data store to output the physical PFN onto the PAD lines. Also used in TB on (MIC 16) to enable TB parity out.

2. TB parity enable is sent to microtrap chip to be ANDed with hit (or lack of) so the microtrap chip will at this time monitor to TB miss, data and tag, M bit, parity errors and ACV from ACV chip.

If any of these conditions exist we leave the microinstruction via a microtrap and proceed to the proper routine specified by microvector lines <5:0>. Lines <3:0> coming from microtrap from microtrap chip and lines <5:4> from microsequencer chip.

If none of these conditions exist you continue the instruction the exact same way as the read, cache hit TB off was discussed.

VAX-11/750 LEVEL II

Unibus/Unibus Interface

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

Unibus/Unibus Interface

INTRODUCTION

The COMET Unibus Interface (CUI) serves three purposes. It allows the processor to access registers on the Unibus, it allows devices on the Unibus to perform DMA transfers to COMET memory, and it allows Unibus devices to interrupt the processor.

There are several characteristics of the VAX architecture and the COMET memory system that require more than a "straight-through" connection from the Unibus to the CMI. Addresses that are contiguous in the virtual address space may be discontiguous in the physical address space on 512 byte boundaries. Since all Unibus NPR devices broadcast sequential addresses, a means must be provided to break these up into disjoint 512 byte blocks.

The VAX architecture imposes no restrictions on the alignment of data in memory. Unibus word transfer NPR devices, however, only transfer word date on even addresses. The CUI provides a mechanism that allows the tranfser to be effectively shifted by one byte to accommodate requests for I/O buffers on odd byte addresses.

The CMI has 24 address bits, the Unibus has 18. A method is provided that allows a Unibus device access to all of the CMI address space.

Finally, the CMI is four bytes wide, the Unibus is two bytes wide. Utilization of both CMI and Unibus can be improved if each two sequential Unibus transfers are compressed into a single CMI transfer.

SYNOPSIS

This module contains technical information concerning the characteristics and functions of the Unbius and the Unibus interface.

OBJECTIVES

Given a True/False test, correctly determine if the Unibus/UBI characteristics listed are true or false.

Given statements concerning the Unbius and Unibus interface functions, and several possible definitions for each, select the one correct definition.

SAMPLE TEST ITEM

Identify the following statements as true or false.

a) The Unibus contains 56 lines.
b) The Unibus interfaces to the W Bus.
c) The Unibus buffered data paths hold 8 bytes.
d) The Unibus interface performs Unibus to CMI address mapping.

RESOURCES

Comet Specification
Peripherals Handbook

MODULE OUTLINE

XV. Unibus/Unibus Interface

A. Unibus structure

1. 41 data transfer lines
2. 12 priority arbitration lines
3. Initialization lines

B. Unibus interface characteristics

C. Unibus memory & I/O space allocation

1. CMI address space assigned to the UBI
2. Control and status registers
3. Diagnostic status registers

D. Unibus to CMI address registers

1. Minimum & Maximum addresses
2. Mapping example, unibus to CMI conversion
3. CMI address transfer

E. Unibus interface data paths

1. CPU write
2. CPU read
3. NPR DATI, DDP
4. NPR DATO, BDP to buffer
5. NPR DATO, DDP
6. NPR DATO, BDP write from buffer
7. Purge
8. Read the map
9. Write the map

F. Data positioning

1. Reads from memory, data buffering
2. Writes to memory, data buffering
3. Write to memory, byte offset

G. Unibus microcode

1. 28 bit word format
2. Field definitions
3. Micro code breakdown
4. Read code at address OFF
5. Read code at address OOF
6. Read code at address OOO

H. Micro code flow charts

1. First fork
2. DDP DATI
3. BDP DATO

I. Print familiarization

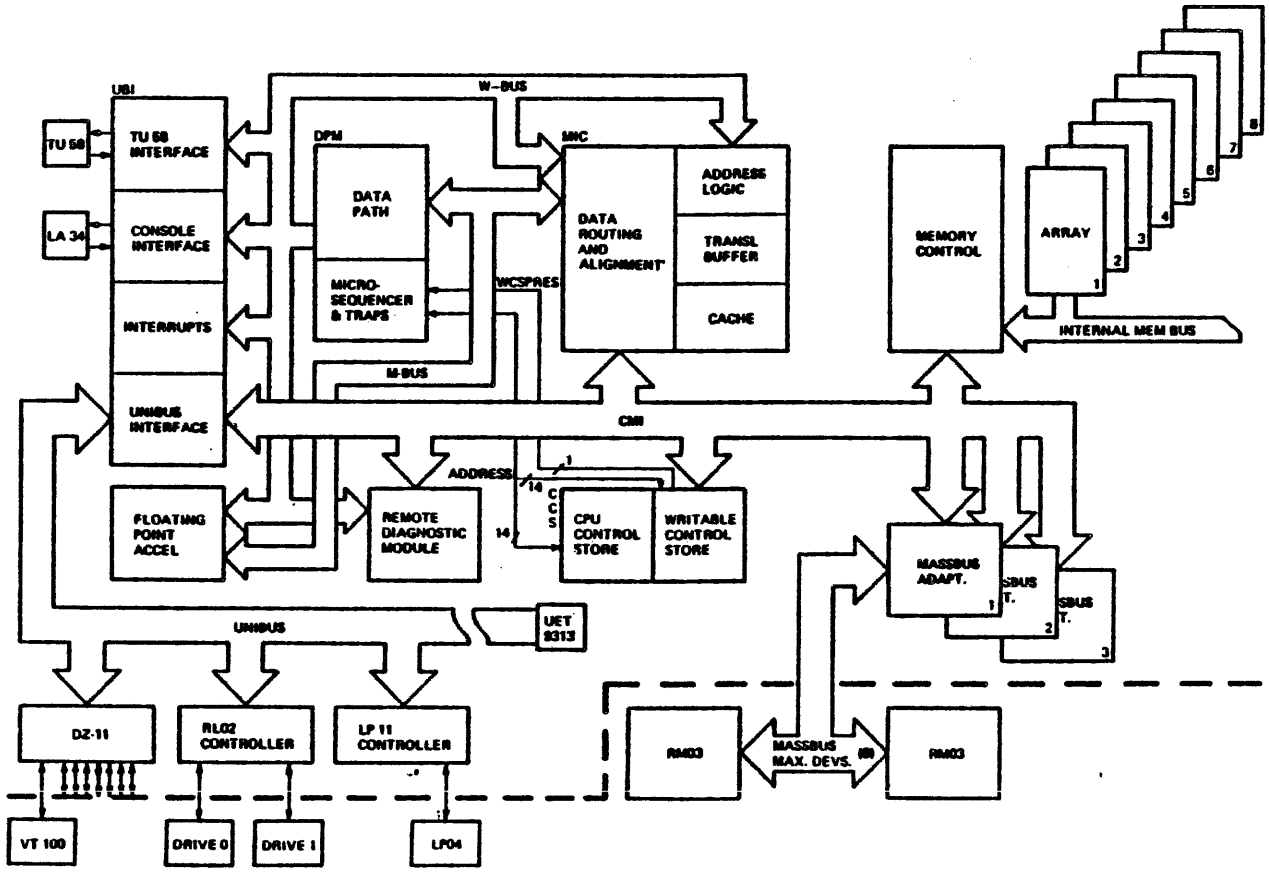
1. Increment logic
2. Unibus address mux
3. Unibus data path chips
4. CUI map
5. UCN Chip
6. CUI control ROMS
7. CUI map decode

J. Summary

UNIBUS SUMMARY

In the Comet system, the Unibus connects PDP-11 devices to the Comet Unibus Interface (CUI) of the CP Cluster. The 56 lines of the asynchronous Unibus can be divided into three functional groups: priority arbitration, data transfer, and initialization signals. The 12 lines of the priority arbitration group comprise those signals required for selection of the next bus master while the current bus master is still in control of the bus.

The 41 bidirectional lines of the data transfer group are used during data transfers to or from a slave device. The initialization group consists of the initialization and power fail signals. table 10-1 describes the bus signals within each group.



COMET SIMPLIFIED SYSTEM BLOCK DIAGRAM

Figure 15-1.

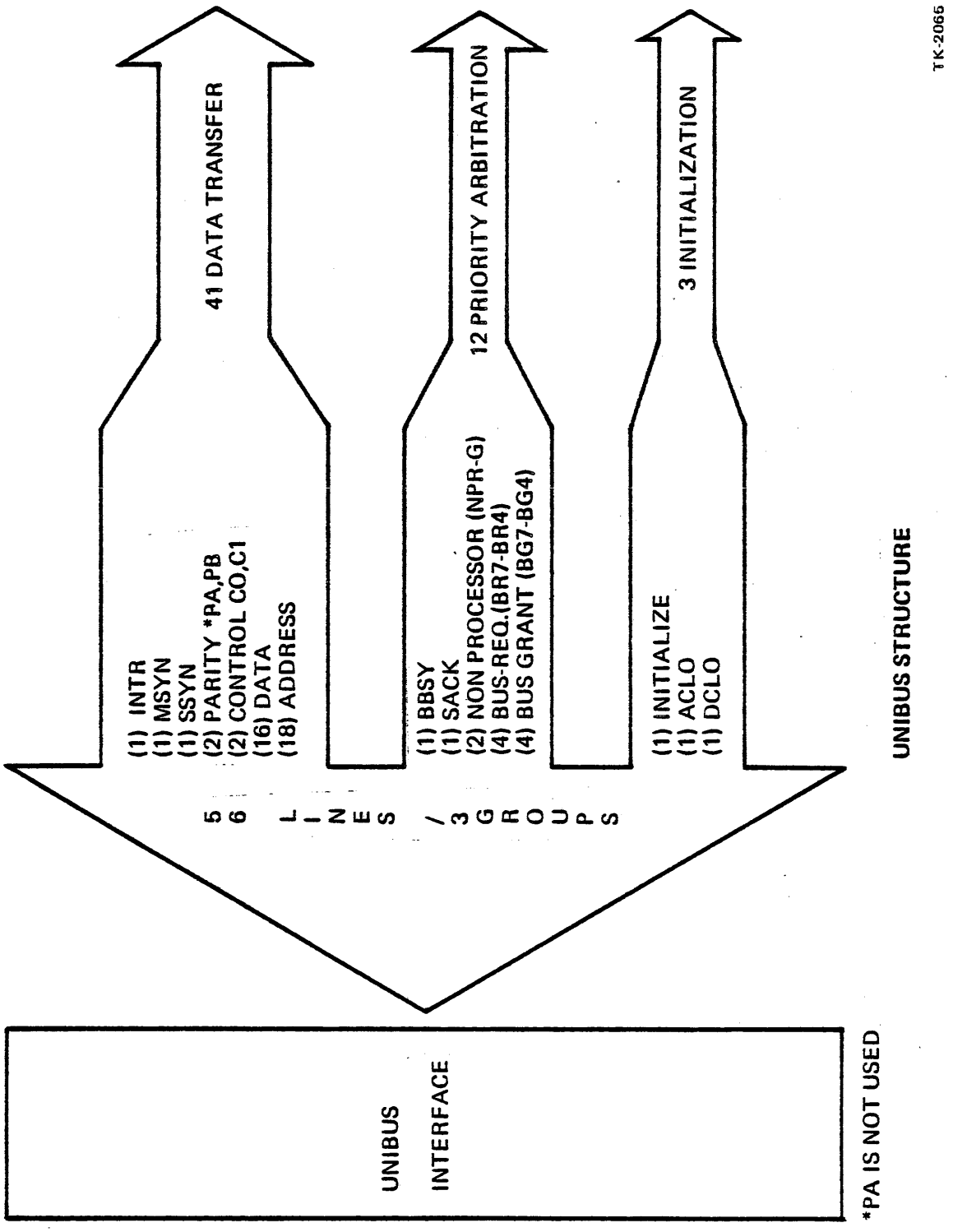

TK-2066
41 DATA TRANSFER
12 PRIORITY ARBITRATION
3 INITIALIZATION
(1) INTR
(1) MSYN
(1) SSYN
(2) PARITY *PA,PB
(2) CONTROL CO,C1
(16) DATA
(18) ADDRESS
(1) BBSY
(1) SACK
(2) NON PROCESSOR (NPR-G)
(4) BUS-REQ.(BR7-BR4)
(4) BUS GRANT (BG7-BG4)
(1) INITIALIZE
(1) ACLO
(1) DCLO
56 LINES / 3 GROUPS
UNIBUS STRUCTURE
UNIBUS
INTERFACE
*PA IS NOT USED

Figure 15-2.

DATA FLOW CONTROL SIGNALS C1 & C0

C1	C0	TRANSFER OPERATION
0	0	DATA IN (DATI): A DATA WORD OR BYTE TRANSFER INTO THE MASTER FROM THE SLAVE.
0	1	DATA IN PAUSE (DATIP): SIMILAR TO DATI EXCEPT IT IS ALWAYS FOLLOWED BY A DATOB.
1	0	DATA OUT (DATO): A DATA WORD IS TRANFERRED FROM MASTER TO SLAVE.
1	1	DATA OUT BYTE (DATOB): SAME AS DATO EXCEPT A BYTE IS TRANFERRED.

TK-2047

Figure 15-3.

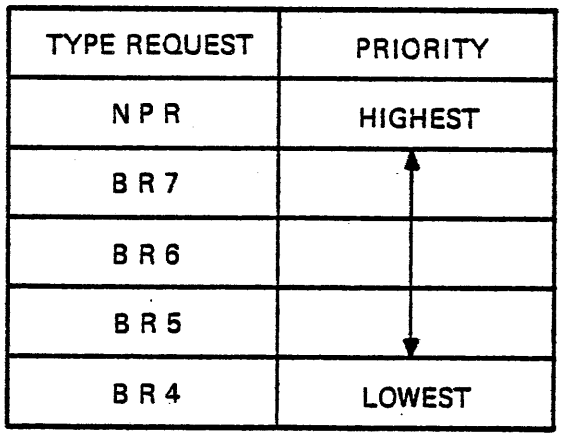
TRANSFER REQUESTS

There are two types of requests for control of the Unibus: Non-Processor Request (NPR) and Bus Requests (BR). The NPR is used when a device requests a direct memory or device access transfer (i.e., a transfer not requiring processor intervention). Normally, NPR transfers are used between a mass storage device (e.g., disk) and memory. A device issues an NPR by asserting the NPR line; the processor (CUI in the comet system) honors the request by asserting the Non-Processor Grant (NPG) line.

The BR is used when a device interrupts the processor to request service. This type of request is used to notify the processor of an error condition or required transfer. A device issues a BR by asserting its assigned Br line (BR7-BR4); the processor (CUI in Comet) honors the request be asserting the corresponding Bus Grant (BG) line (BG7-BG4).

Request Priority

The device structure priority structure is organized as follows:



* IF TWO DEVICES ISSUE SIMULTANEOUS REQUESTS, BUS GRANT WILL GO TO THE HIGHEST PRIORITY REQUEST.

BUS REQUEST PRIORITY

TK-2058

Figure 15-4

The priority arbitration logic is structured such that if two devices on different BR levels issue simultaneous requests, the bus is granted to the device with the highest priority. The lowest priority device must keep its requested asserted in order to gain control of the bus when the highest priority device is finished (providing no other higher priority device issues a BR).

Since there are only five priority levels, more than one device may be assigned to a specific request level. If more than one device makes a request at the same level, the device closest (electrically) to the processor has the highest priority.

Priority Arbitration Sequence

Priority arbitration involves the signal sequence which selects the next bus master. The operation does not actually transfer bus control but only selects the next bus master.

The device requiring service asserts its BR (or NPR) line. In practice, the Comet Unibus Interface May be receiving several simultaneous BR signals. These signals enter the Unibus arbitration logic of the CUI. If enabled by software, the CUI then conducts a dialogue with the CPU and asserts the corresponding BG (or NPR) line. The grant is propagated through each device on the asserted BG line. The first device on the line having BR asserted acknowledges the grant by asserting SACK, blocks the grant from following devices, and clears its BR. The Unibus Adaptor responds to SACK by clearing BG. If SACK is not asserted by the requesting device, BG is Ored with the other grant lines and returned as a SACK signal to clear BG.

The device will keep SACK asserted until the current bus master relinquishes the bus control by clearing its BBSY. SACK asserted prevents other devices from gaining bus control. Once the current bus master has relinquished the bus and negated BBSY, the requesting device asserts BBSY and negates SACK, becoming the new bus master. Priority arbitration can be performed at the same time as the data transaction of the servicing of an interrupt. While one device is using the bus, the arbitration logic is free to monitor other requests and issue an appropriate grant.

THE UNIBUS INTERFACE

- ALLOWS THE PROCESSOR TO ACCESS REGISTERS ON THE UNIBUS

  THE CMI HAS 24 ADDRESS BITS WHILE THE UNIBUS ONLY HAS 18. A METHOD IS USED THAT ALLOWS ANY UNIBUS DEVICE ACCESS TO ALL OF THE CMI ADDRESS SPACE.

- ALLOWS DEVICES ON THE UNIBUS TO PERFORM DMA TRANSFERS TO MAIN MEMORY.

  THE CMI IS 4 BYTES WIDE WHILE THE UNIBUS IS ONLY 2 BYTES. TO MORE EFFICIENTLY UTILIZE THE CMI, THE DATA IS MOVED ONTO THE CMI AFTER EACH PAIR OF UNIBUS TRANFERS.

  THE UNIBUS MOVES DATA ON EVEN ADDRESS BOUNDRIES, TO BE ABLE TO ACCESS I/O BUFFERS ON ODD BYTE ADDRESSES, A MECHANISM IS IN USE THAT SHIFTS THE ADDRESS BY ONE BYTE.

- ALLOWS UNIBUS DEVICES TO INTERRUPT THE PROCESSOR.

TK-2050

Figure 15-5

Table 15-1. Unibus Signal Description

Signal	Description
Data Transfer Group	
Address Lines (A <17:00>)	These lines are used by the master device to select the slave (actually a unique memory or device register address). A<17:01> specifies a unique 16-bit word; A00 specifies a byte within the word.
Data Lines (D<15:00>)	These lines transfer information between master and slave.
Control (C1, C0)	These siganls are coded by the master device to control the slave in one of the four possble data transfer operations specified below. Note that the transfer direction is always designated with respect to the master device. *See Figure 10-3

Table 15-1. Unibus Signal Description (Cont.)

Signal	Description		
Control (C1, CØ)	C1	CØ	TRANSER OPERATION
	Ø	Ø	Data In (DATI): a data word or byte transferred into the master from the slave.
	Ø	1	Data In Pause (DATIP): similar to DATI except that it is always followed by a DATO/B to the same location.
	1	Ø	Data Out (DATO): a data word is transferred out of the master to the slave.
	1	1	Data Out Byte (DATOB): identical to DATO except a byte is transferred instead of a full word.

Table 15-1 Unibus Signal Description (cont.)

Signal	Description
Parity A-B (PA, PB)	These signals transfer Unibus parity information. PA is currently unused and high. PB, when true, indicates a device parity error.
Master Synchronization (MSYN)	MSYN is asserted by the master to indicate to the slave that valid address and control information (and data on a DATOB) is present on the bus.
Slave Synchronization (SSYN)	SSYN is asserted by the slave. On a DATO it indicates that the slave has latched the write data. On a DATI/P it indicates that the slave has asserted read data on the Unibus.
Interrupt (INTR)	This signal is asserted by an interrupting device, after it becomes bus master, to inform the processor that an interrupt is to be performed, and that the interrupt vector is present on the D lines. INTR is negated upon receipt of the assertion of SSYN by the processor at the end of the transaction. INTR may be asserted only by a device which became MASTER by receiving a BG signal.

Table 15-1 Unibus Signal Description (cont.)

Signal	Description
Priority Arbitration Group	
Bus Request (BR7-BR4)	These signals are used by peripheral devices on the Unibus and nexus on the CMI to request control of the bus for an interrupt operation.
Bus Grant (BG7-BG4)	These signals form processor's response to a bus request. Only one of the four will be asserted at any time.
Nonprocessor Request (NPR)	This is a bus request from a device for a transfer not requiring CPU intervention (i.e., direct memory access).
Nonprocessor Grant (NPG)	This is the bus grant in response to an NPR.
Select Acknowledge (SACK)	SACK is asserted by a bus-requesting device after having received a grant. Bus control passes to this device when the current bus master completes its operation.
Bus Busy (BBSY)	BBSY indicates that the data lines of the bus are in use.

Table 15-1 Unibus Signal Description (cont.)

Initialization Group	
Initialize (INIT)	This signal is asserted by the terminator board (UET) when DC LO is asserted on the Unibus. INIT stays asserted for 10 ms following the negation of DC LO.
AC Line Low (AC LO)	This signal initiates the power fail trap sequence, and may also be issued in peripheral devices to terminate operations in preparation for power loss.
DC Line Low (DC LO)	This signal is available from each system power supply and remains clear as long as all dc voltages are within the specified limits. If an out-of-voltage condition occurs, DC LO is asserted.

ADDRESS SPACE

Any processor access with a physical address in the range of FC0000 through FFFFFF will map directly on to the Unibus in the range 0 through 777777 (octal). Any device on the CMI other than the processor that does a CMI transaction in that address range will be ignored by the CUI. See Figure 10-7.

CUI ADDRESSES

The CUI is assigned a block of 8KB of CMI address space for the map, CSR's and DSR's. See Figure 10-8.

CONTROL AND STATUS REGISTERS

Proper operation of transfers through the BDP's requires some intervention on the part of system software. The use of BDP's is not totally transparent. When a device has finished a series of transfers to CMI memory (DATO(B)'s), it is possible that some data will remain in the buffer if the transfer did not end on an even longword boundary. It is necessary for the software to initiate action that causes this data to be written to memory. When a device has finished a transaction that involves DATI(P)'s, data is left in the buffer with a corresponding Unibus address in the address register. Should the contents of the map be changed at the location corresponding to the address in the address register, there will no longer be the correct association between address and data in the buffer. It is therefore necessary to clear the buffer following this class of transactions. See Figure 10-9.

BDP SDR (Diagnostic Status Register)

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.

BITS <31:28>	BF3:BF0
BIT <27>	CD

See Figure 15-10.

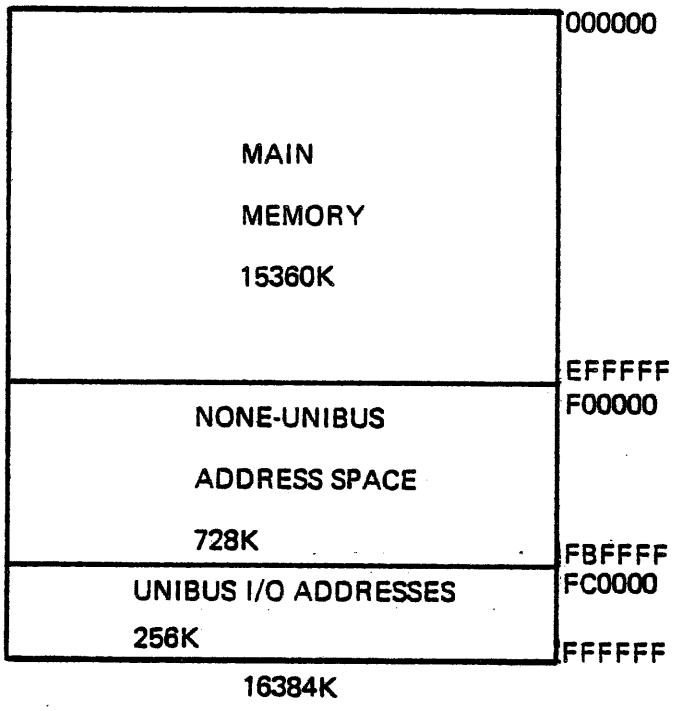

000000
MAIN
MEMORY
15360K
EFFFFF
F00000
NONE-UNIBUS
ADDRESS SPACE
728K
FBFFFF
FC0000
UNIBUS I/O ADDRESSES
256K
FFFFFF
16384K
ADDRESS SPACE ALLOCATION
TK-2071

Figure 15-7

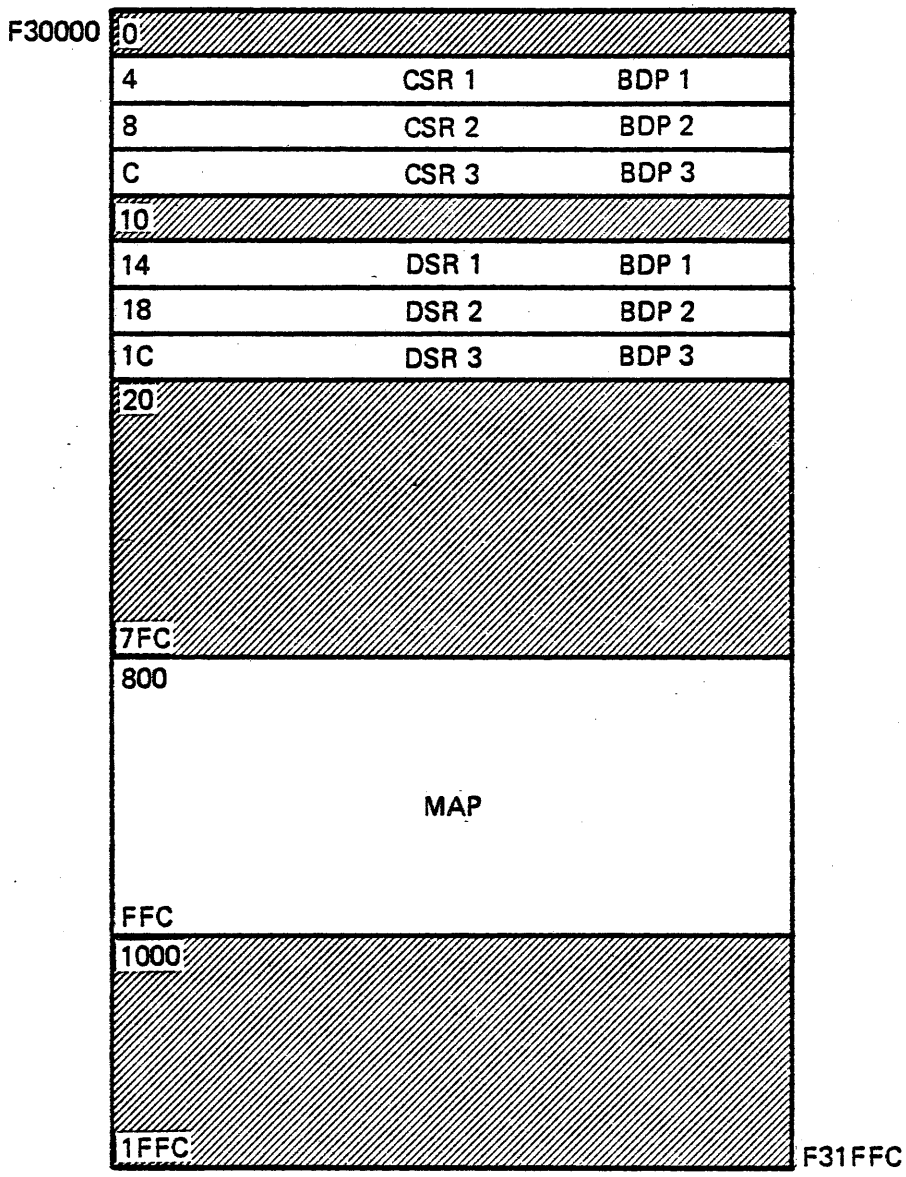

F30000
0
4
CSR 1
BDP 1
8
CSR 2
BDP 2
C
CSR 3
BDP 3
10
14
DSR 1
BDP 1
18
DSR 2
BDP 2
1C
DSR 3
BDP 3
20
7FC
800
MAP
FFC
1000
1FFC
F31FFC
8K BYTE OF CMI ADDRESS SPACE
ASSIGNED TO THE UNIBUS INTERFACE
TK-2068

Figure 15-8

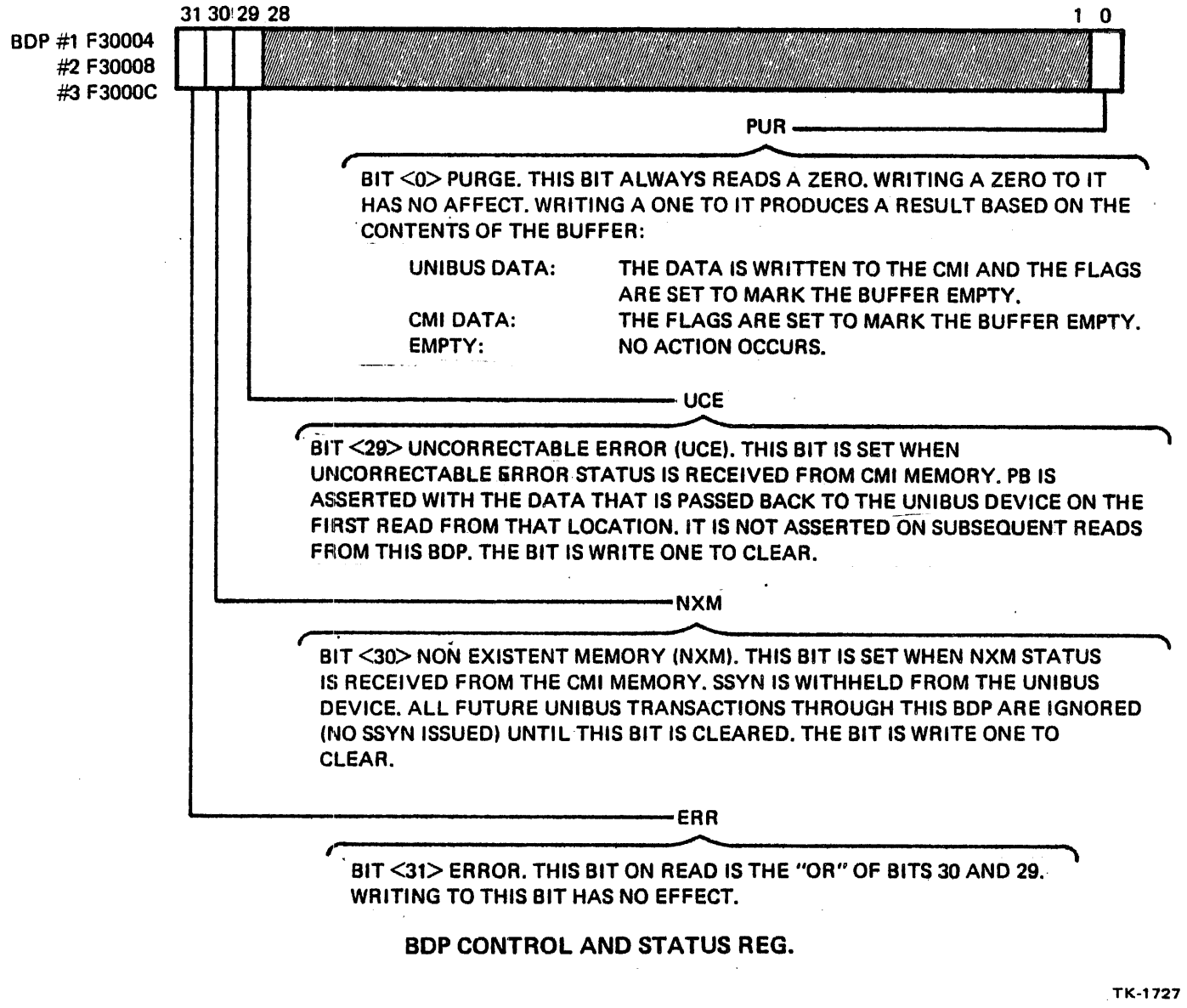
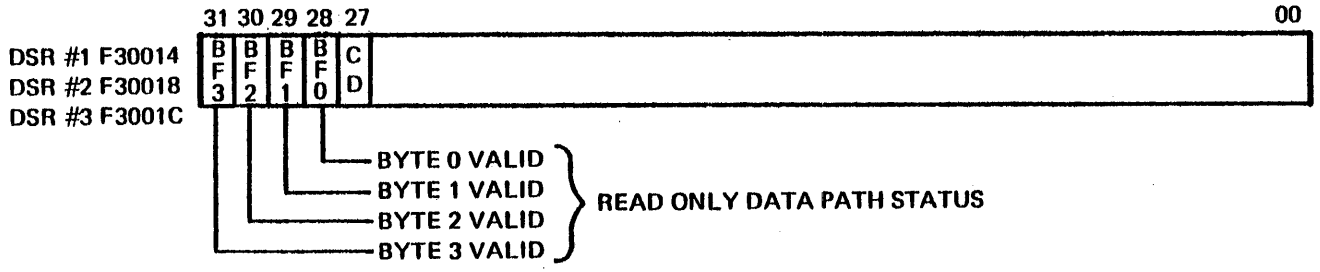

31 30 29 28
1 0
BDP #1 F30004
#2 F30008
#3 F3000C
PUR
BIT <0> PURGE. THIS BIT ALWAYS READS A ZERO. WRITING A ZERO TO IT HAS NO AFFECT. WRITING A ONE TO IT PRODUCES A RESULT BASED ON THE CONTENTS OF THE BUFFER:
UNIBUS DATA: THE DATA IS WRITTEN TO THE CMI AND THE FLAGS ARE SET TO MARK THE BUFFER EMPTY.
CMI DATA: THE FLAGS ARE SET TO MARK THE BUFFER EMPTY.
EMPTY: NO ACTION OCCURS.
UCE
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.
NXM
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.
ERR
BIT <31> ERROR. THIS BIT ON READ IS THE "OR" OF BITS 30 AND 29. WRITING TO THIS BIT HAS NO EFFECT.
BDP CONTROL AND STATUS REG.
TK-1727

Figure 15-9



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.

CUI DIAGNOSTIC STATUS REGISTER

TK-1726

Figure 15-10

UNIBUS MAP

Unibus address bits <17:9> are used to enter into a 512 location by 19 bit wide memory. The data coming out of that memory is used to determine the details of how the transaction is to be handled. The map data field is divided into four sections. See Figure 10-11.

MAP SECTIONS

PFN

The PFN is a 15 bit field. On Unibus initiated transactions that cause a CMI read or write cycle to occur, the map PFN is concatenated with Unibus address <8:2> to form a 22 bit longword address on the CMI.

DATA PATH NUMBER

This two bit field is used to select one of the four data paths. Data paths 1, 2, 3 are called buffered data paths. Data path 0 is called the direct data path.

OFFSET

When the offset bit is a 1, it causes the transaction to behave as if the Unibus address supplied by the DMA device was incremented by 1. This allows devices that only produce even byte addresses to access buffers on odd byte boundaries. A transaction that causes a word to cross page boundaries because the offset bit is set must have data path number, offset, and valid identical in both map entries. Any differences will yield unpredictable results.

VALID BIT

When the valid bit is a 1, the CUI processes the transaction. When it is 0, the CUI ignores the receipt of MSYN. The valid bit must be set to 0 for map entries that correspond to sections of Unibus address space in which there are slaves that are expected to respond to transactions that originate on the Unibus. Transactions that start on the CMI and cause a Unibus transaction to occur are always ignored by the CUI and can never wrap back through the CUI onto the CMI.

MAP ACCESS FROM THE CMI

The map is accessible for both reading and writing from the CMI. Each entry uses up a longword address or the CMI. The format of the map data fields as they appear on the CMI is shown on Figure 10-11.

The logic that causes the map to be written ignores the byte mask bits on the CMI. It assumes that any write to the map addresses is a longword write. Any write other than a longword will cause the contents of the map at the written location to become unpredictable.

A Unibus initiated transaction that causes a CMI transaction in the map address space on the CMI will properly address the map and cause appropriate action. Note, however, that writing the map requires a longword write and the only way that a Unibus device can cause a longword write is to do two sequential transfers within a longword to a buffered data path. It is also a requirement that the buffered data path not receive any other transfer in between the two that made up the longword.

The CMI byte mask controls the setting of the A1, A0, and (for writes) the C0 lines of the Unibus. For reads the setting is: See Figure 10-18.

Byte Mask	A1	A0
1111	0	0
1110	0	0
1100	1	0
1000	1	0

The processor will never produce any other values with reads to the Unibus.

For writes:

Byte Mask	A1	A0	C0
0001	0	0	1
0010	0	1	1
0100	1	0	1
1000	1	1	1
0011	0	0	0
1100	1	0	0

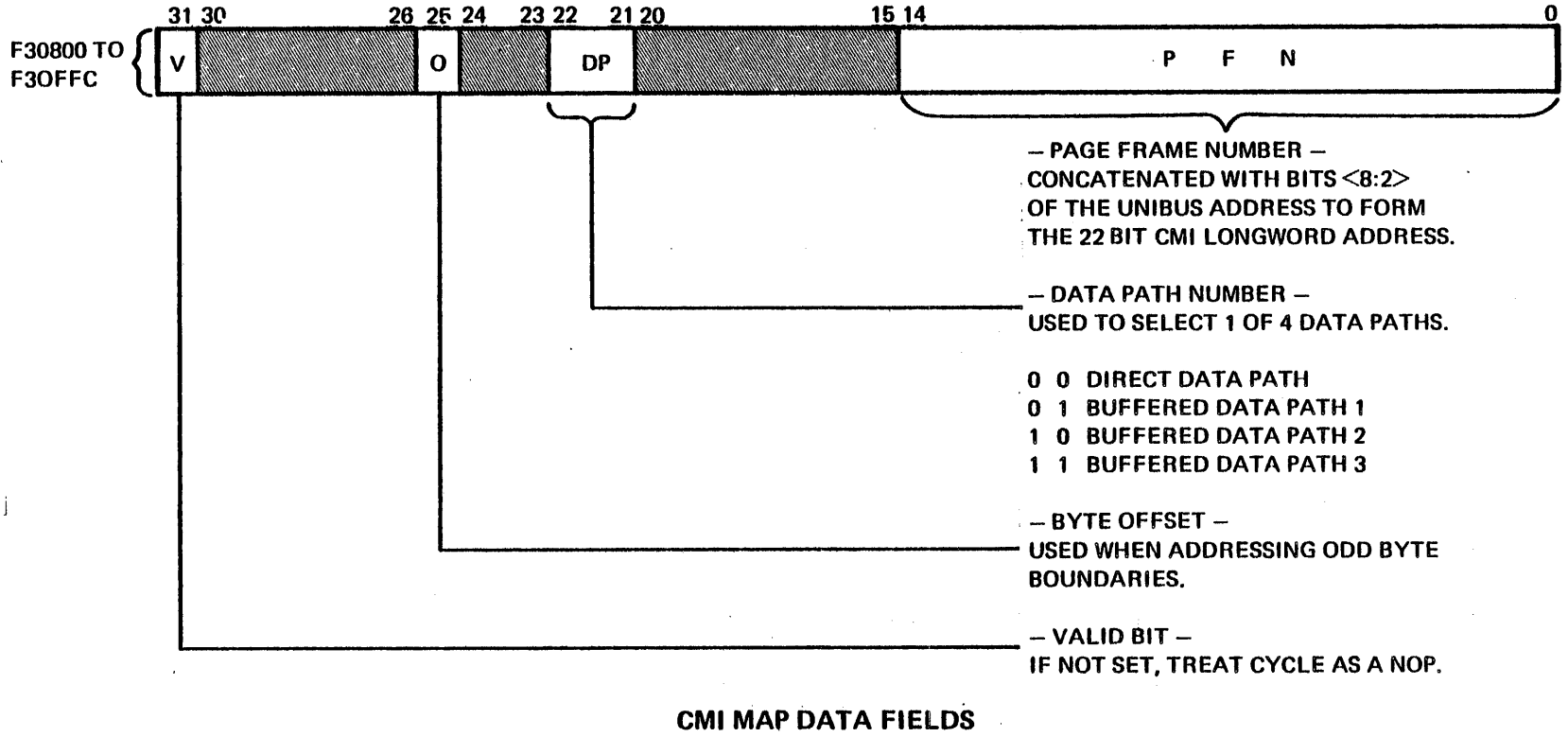
The different types of CMI operations are mapped into Unibus operations as follows:

CMI	UNIBUS	NOTE
Read	DATI	
Read with modify intent	DATIP	(1)
Read lock	DATIP	(1)
Write	DATO(B)	(2)
Write unlock	DATO(B)	(2)
Write vector	No Response	(3)

1) When a CMI operation that causes a DATIP occurs, BBSY is asserted and held until the end of the next operation that does not cause a DATIP to occur.

2) The choice of DATO or DATOB is made based on the byte mask.

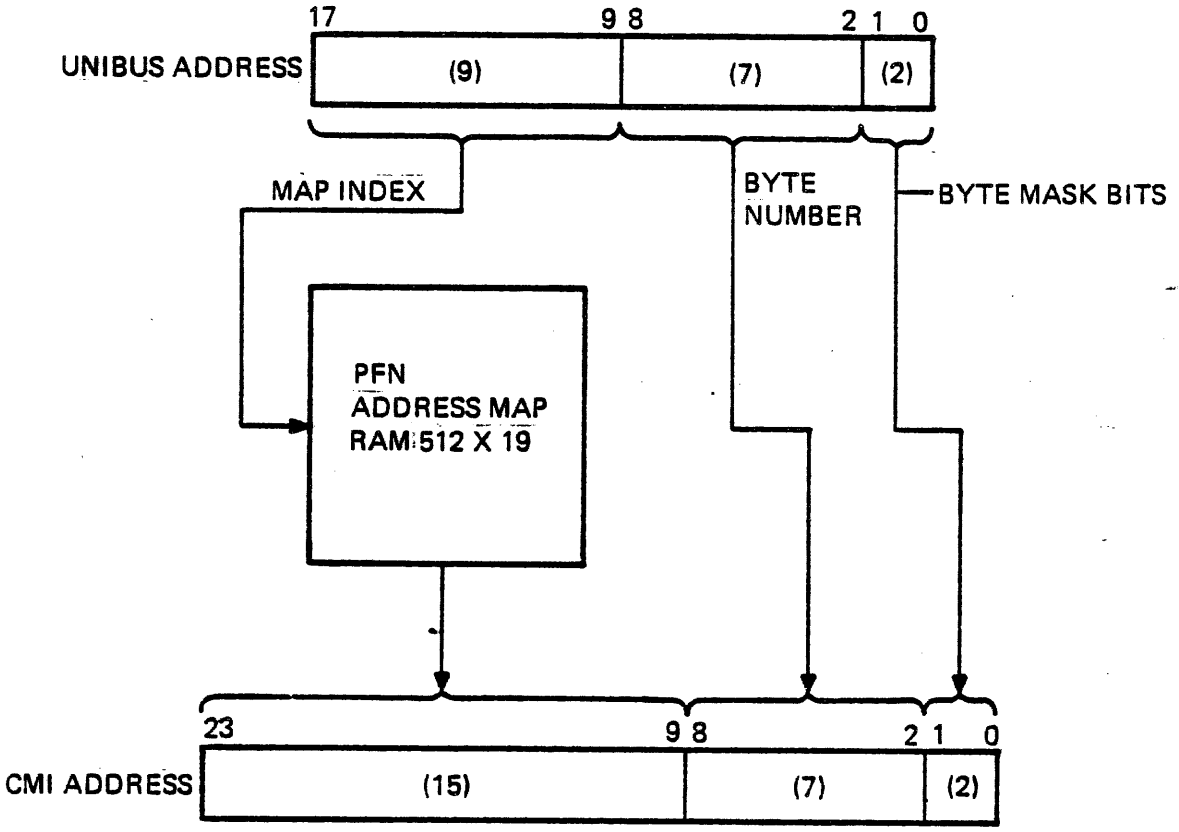
3) The processor will never actually issue a write vector.



CMI MAP DATA FIELDS

TK-1739

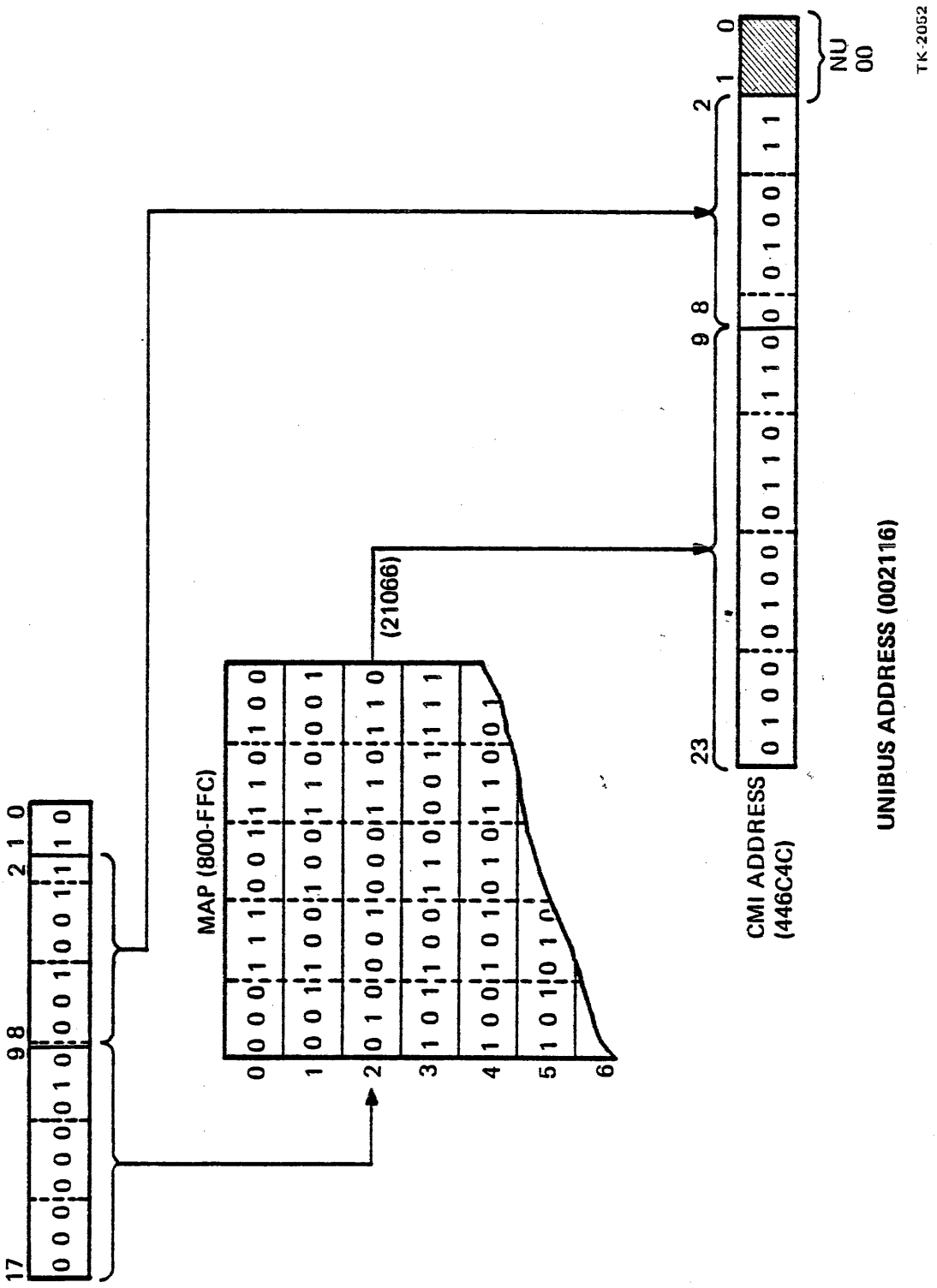
Figure 15-11



UNIBUS TO CMI ADDRESS TRANSLATION

TK-2066

Figure 15-12


TK-2052
NU
00
2 1 0
9 8
23
0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 1

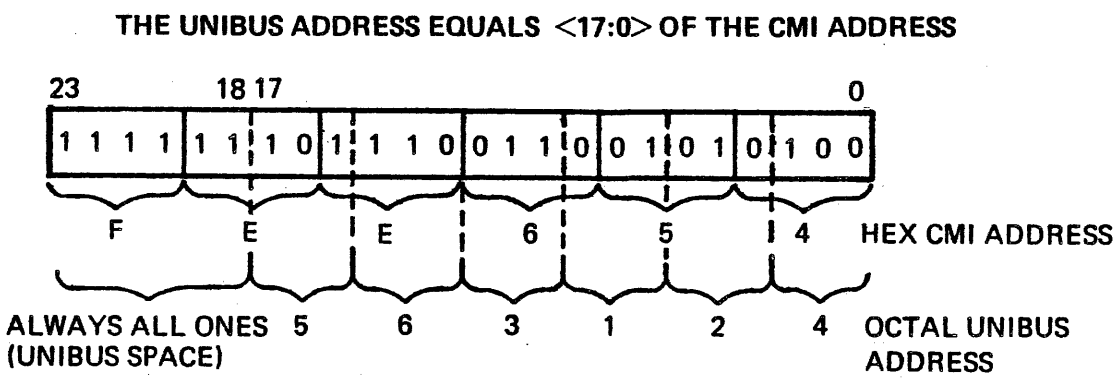
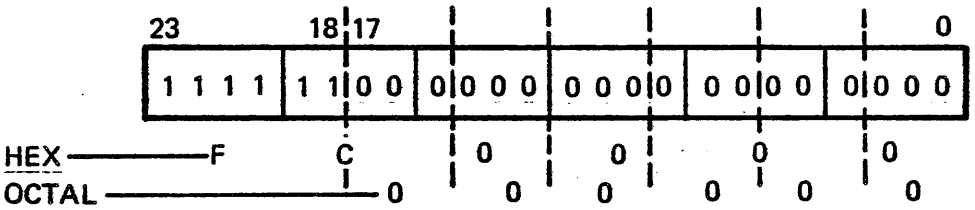
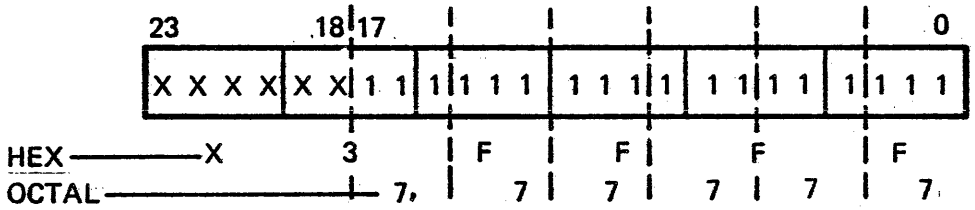

THE UNIBUS ADDRESS EQUALS <17:0> OF THE CMI ADDRESS
23
18 17
0
1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 1 0 1 0 0
F
E
E
6
5
4
HEX CMI ADDRESS
ALWAYS ALL ONES
(UNIBUS SPACE)
5
6
3
1
2
4
OCTAL UNIBUS
ADDRESS
TK-2048

Figure 15-14

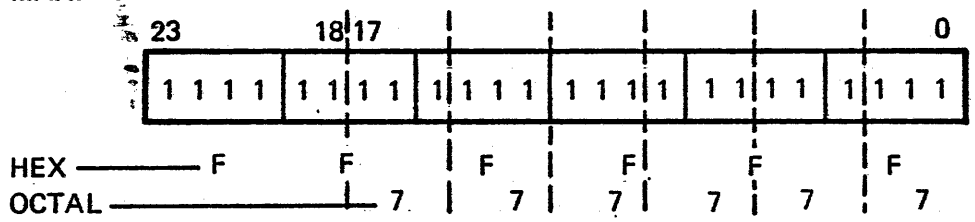
MINIMUM ADDRESS REQUIREMENTS FOR DIRECT TRANSLATION FROM CMI ADDRESSES TO UNIBUS ADDRESSES.



MAXIMUM UNIBUS ADDRESS AT UBI



MAXIMUM UNIBUS ADDRESS ON CMI



TK-2090

Figure 15-15

HARDWARE COMPONENTS

DATA BUFFERS

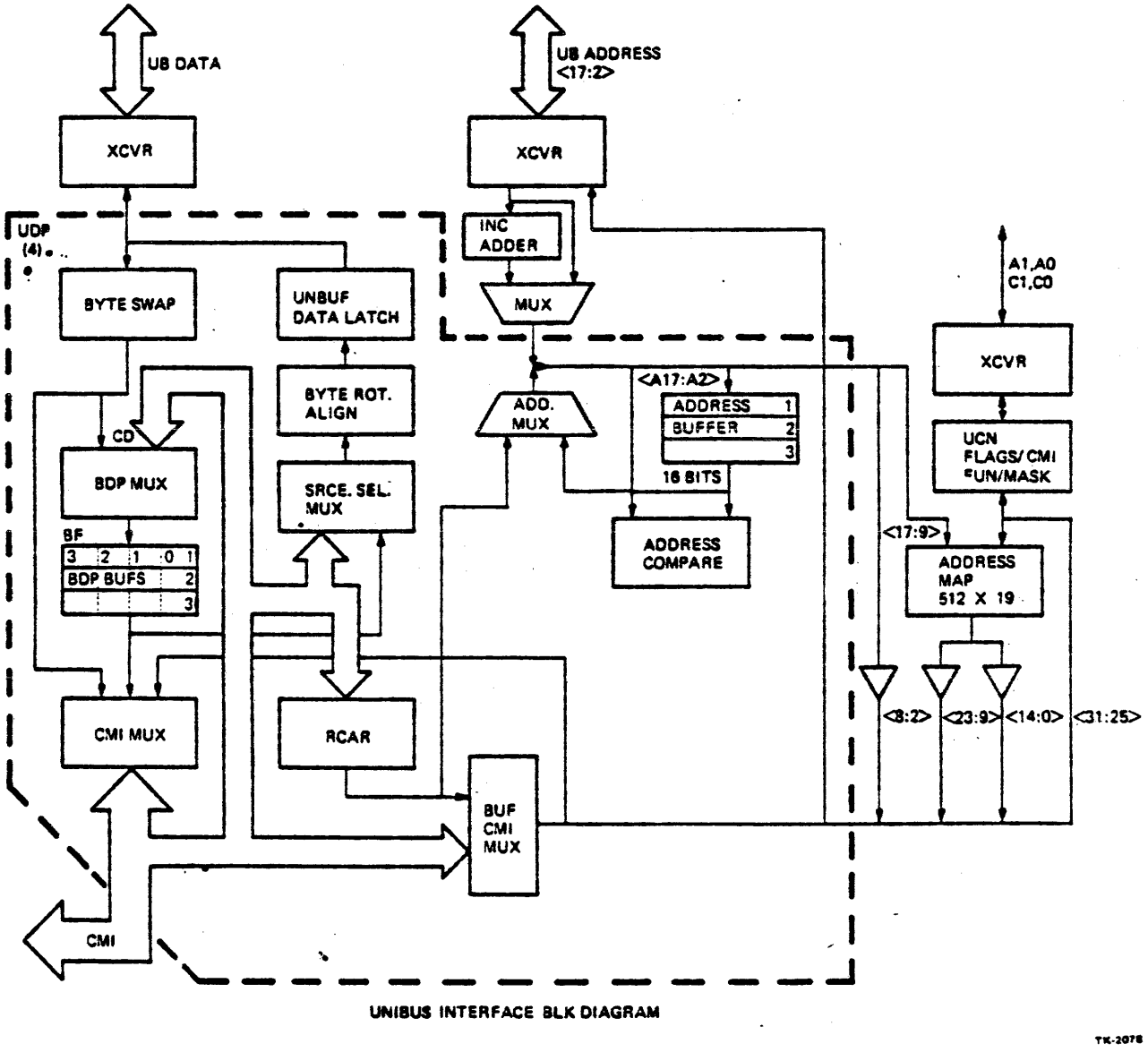
Each BDP consists of a data storage buffer of 4 bytes. This storage buffer can be loaded from the Unibus or the CMI, and its contents can be output to either the Unibus or the CMI. Data can be loaded into the buffer one or two bytes at a time from the Unibus, but is always loaded 4 bytes at a time from the CMI.

ADDRESS REGISTER

Each BDP has a sixteen bit address register that can be loaded from either Unibus addresses <17:2> or <17:2> + 1 (if offset is on). There is circuitry that compares the stored address with the address on the Unibus to see if there is a match. The address held in the register is the Unibus longword (Unibus A17:A2) corresponding to the data in the data buffer.

FLAGS

There are five flags that keep track of the data in the data buffer, named CD and BF3 through BFØ. If CD=1, then the buffer has four bytes of data from the CMI and BF3 though BFØ are always Ø. If DC=Ø, then, then BF3 through BFØ indicate which bytes in the data buffer have valid Unibus data. If they are all Ø, then the buffer is considered empty.



UNIBUS INTERFACE BLK DIAGRAM

TK-2078

Figure 15-16

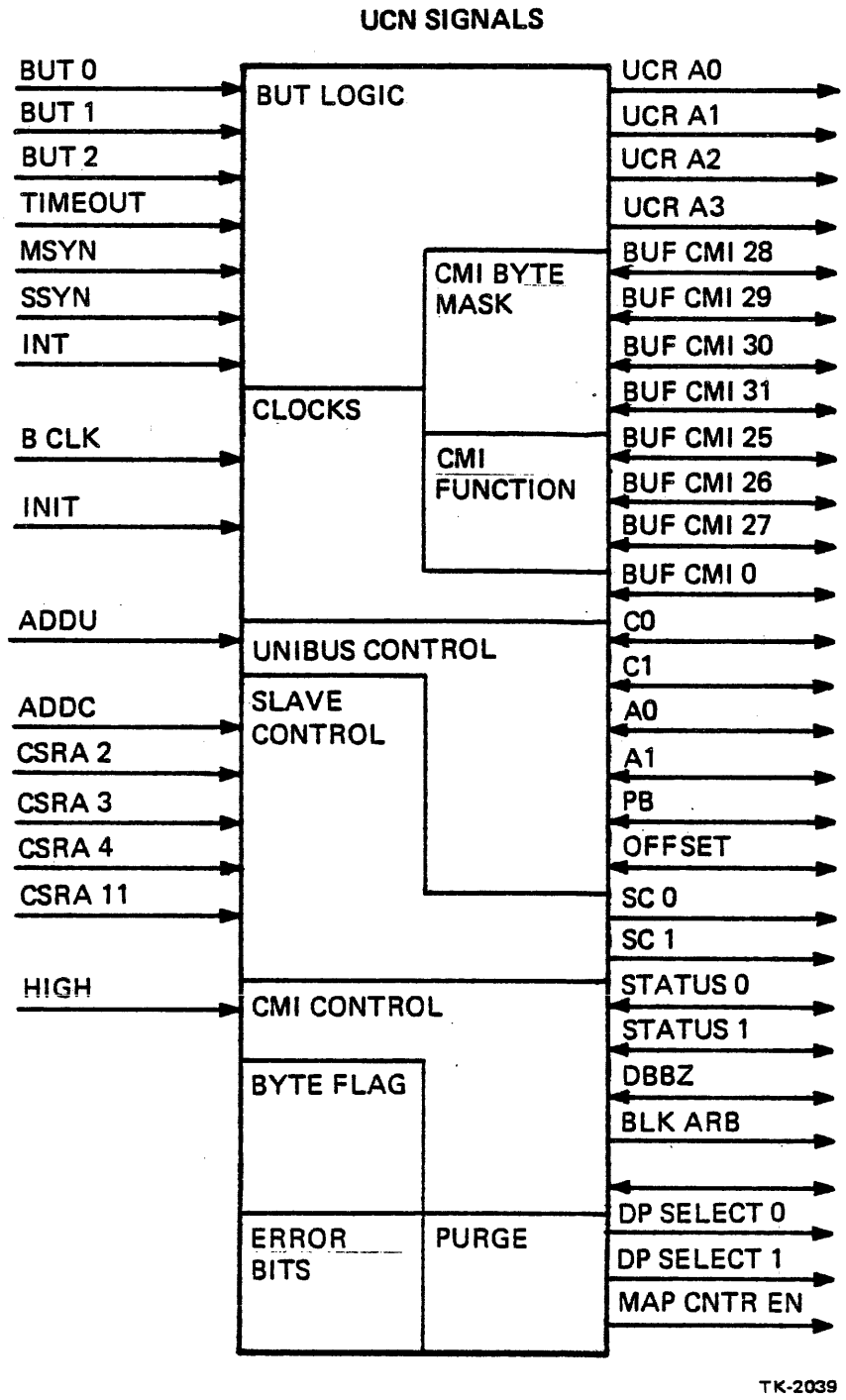
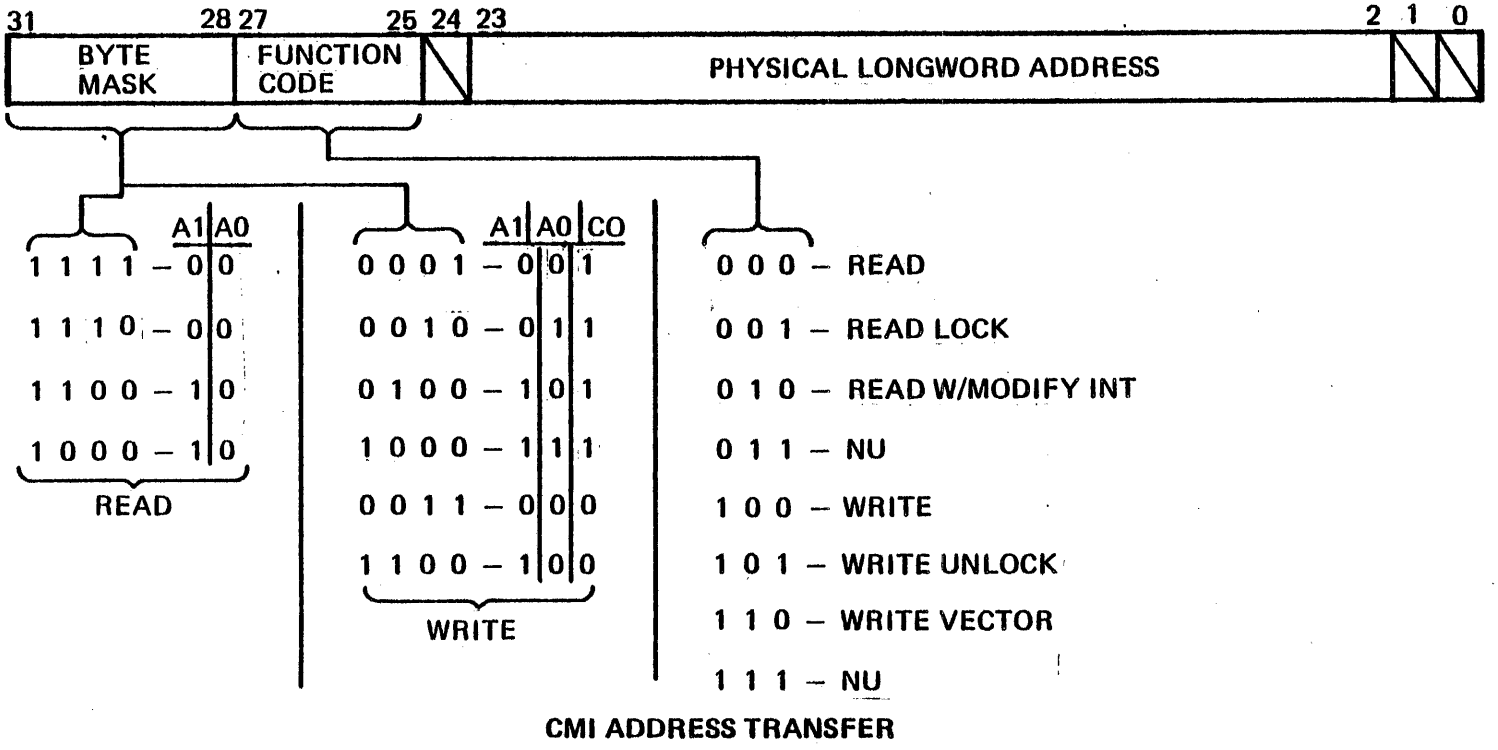

UCN SIGNALS
BUT 0
BUT 1
BUT 2
TIMEOUT
MSYN
SSYN
INT
B CLK
INIT
ADDU
ADDC
CSRA 2
CSRA 3
CSRA 4
CSRA 11
HIGH
BUT LOGIC
CLOCKS
CMI BYTE MASK
CMI FUNCTION
UNIBUS CONTROL
SLAVE CONTROL
CMI CONTROL
BYTE FLAG
ERROR BITS
PURGE
UCR A0
UCR A1
UCR A2
UCR A3
BUF CMI 28
BUF CMI 29
BUF CMI 30
BUF CMI 31
BUF CMI 25
BUF CMI 26
BUF CMI 27
BUF CMI 0
C0
C1
A0
A1
PB
OFFSET
SC 0
SC 1
STATUS 0
STATUS 1
DBBZ
BLK ARB
DP SELECT 0
DP SELECT 1
MAP CNTR EN
TK-2039

Figure 15-17



CMI ADDRESS TRANSFER

TK-2057

Figure 15-18

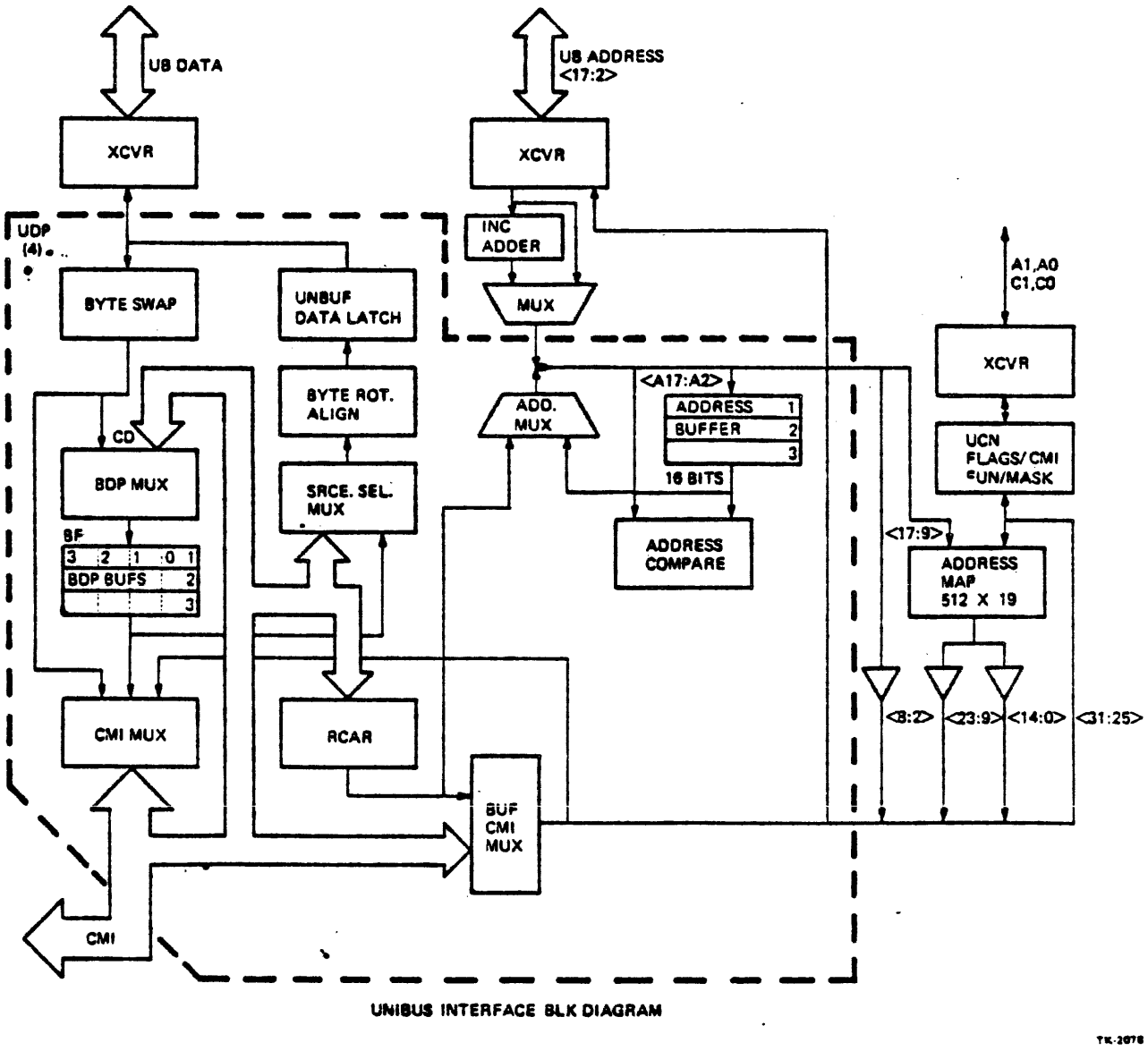

UB DATA
XCVR
UDP
(4)
BYTE SWAP
UNBUF
DATA LATCH
BYTE ROT.
ALIGN
SRCE. SEL.
MUX
CD
BDP MUX
BF
3 2 1 0 1
BDP BUFS 2
3
CMI MUX
RCAR
CMI
BUF
CMI
MUX
UB ADDRESS
<17:2>
XCVR
INC
ADDER
MUX
ADD.
MUX
<A17:A2>
ADDRESS 1
BUFFER 2
3
16 BITS
ADDRESS
COMPARE
A1,A0
C1,C0
XCVR
UCN
FLAGS/CMI
ERUN/MASK
<17:9>
ADDRESS
MAP
512 X 19
<8:2>
<23:9>
<14:0>
<31:25>
UNIBUS INTERFACE BLK DIAGRAM
TK-2078

Figure 15-19

TYPES OF TRANSACTIONS

The table below indicates what type of CMI function is initiated as a result of a Unibus cycle.

UNIBUS C1, CØ	CMI FUNCTION
DATI	Read
DATIP	Read Lock
DATO(B)	Write or write unlock

If a DATO(B) follows a DATIP, then a write lock will go out on the CMI, otherwise an ordinary write will occur.

STATUS

CMI STATUS	CUI RESPONSE TO UNIBUS
No error, or corrected data	SSYN issued
NXM	SSYN withheld
Uncorrectable Error	PB asserted with SSYN

OFFSET

If the offset bit in the map is set, then the transaction will be treated as if the Unibus address were incremented by 1. Note that if this is a DATI(P) or a DATO and if A1=1, two CMI cycles will occur since the two bytes of Unibs data fall across a longword boundary.

BUFFERED DATA PATHS

When the data path section of the map has a value of 1, 2, or 3, then a buffered data path has been selected. Each of the three BDPs consists of four bytes of data storage, 16 bits of address storage, five flag bits, and logic to make the BDP operate. The general intent of the BDP is that when the Unibus transactions are occurring with sequential addresses (either ascending or descending), only one CMI transfer is needed for every two Unibus transfers.

DIRECT DATA PATH

When the data path bits in the map specify Ø, the transaction is said to use the direct data path (DDP). This means that SSYN is not issued by the CUI until the CMI transaction corresponding to the UNIBUS has been completed.

DATI(P) WITH BYTE OFFSET

When byte offset is asserted out of the map, the behavior depends on whether or not it causes this transaction to wrap around across a longword boundary. If it doesn't (Unibus A1=Ø) then the data is shifted one byte to the left. If it wraps (A1=1) the CUI effectively acts as if two sequential transfers occur, the first at the given Unibus address, the second at the address incremented. The two CMI reads are pieced together to form the Unibus data word and SSYN is issued. The data buffer and address register hold the information from the second read at the end of the transaction.

DATO(B) BEHAVIOR

The CUI behavior on Unibus DATO(B)'s is primarily dependent on the contents of the buffer.

Buffer has Unibus data, no address match.

The data in the buffer is written out on the CMI, and the flags are set to mark the buffer empty.

Buffer is empty or has CMI data.

The Unibus data is put in the data buffer, the Unibus address is put in the address register, the flags are set to indicate the appropriate Unibus data and SSYN is issued.

Buffer has Unibus data, address match.

If the data on the Unibus combined with the data in the buffer forms a full four byte longword, then a CMI write is performed, the buffer is marked as empty, and SSYN is issued. If a full longword is not formed, then the Unibus data is put in the buffer and the flags are set. SSYN is asserted.

DATO WITH BYTE OFFSET

If this wraps across a longword boundary, it is treated as two one byte writes. If it does not cross a boundary, it is handled the same as DATO.

DATOB WITH BYTE OFFSET

If A1 A0=11, this is handled the same as with the address effectively incremented by 1. If A1 A0 = 11, then it is treated as if it were a DATOB in the next longword with A1 A0 = 00, except that address match is forced to no match.

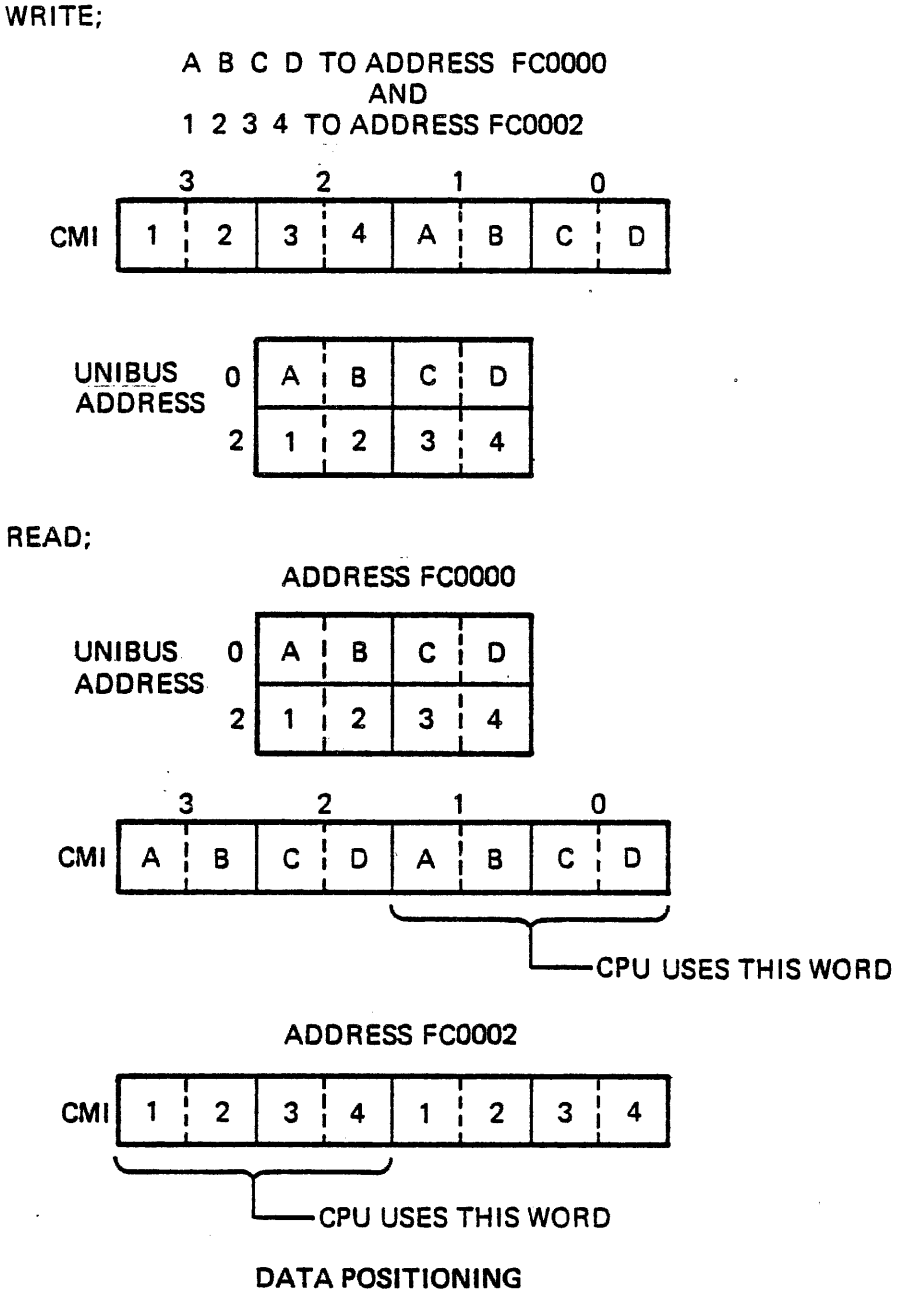

WRITE;
A B C D TO ADDRESS FC0000
AND
1 2 3 4 TO ADDRESS FC0002
3 2 1 0
CMI 1 2 3 4 A B C D
UNIBUS ADDRESS
0 A B C D
2 1 2 3 4
READ;
ADDRESS FC0000
UNIBUS ADDRESS
0 A B C D
2 1 2 3 4
3 2 1 0
CMI A B C D A B C D
CPU USES THIS WORD
ADDRESS FC0002
CMI 1 2 3 4 1 2 3 4
CPU USES THIS WORD
DATA POSITIONING
TK-2053

Figure 15-20

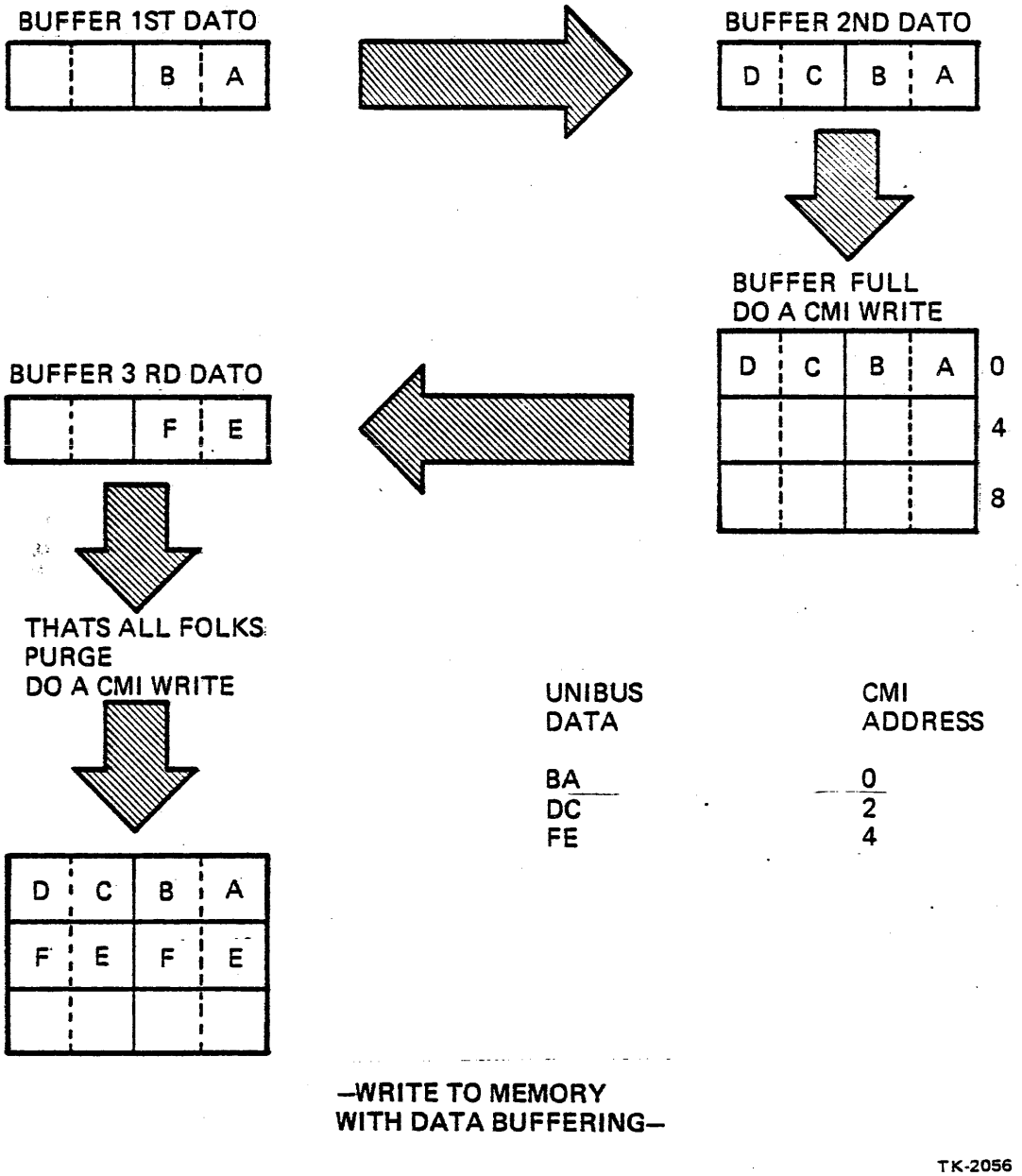
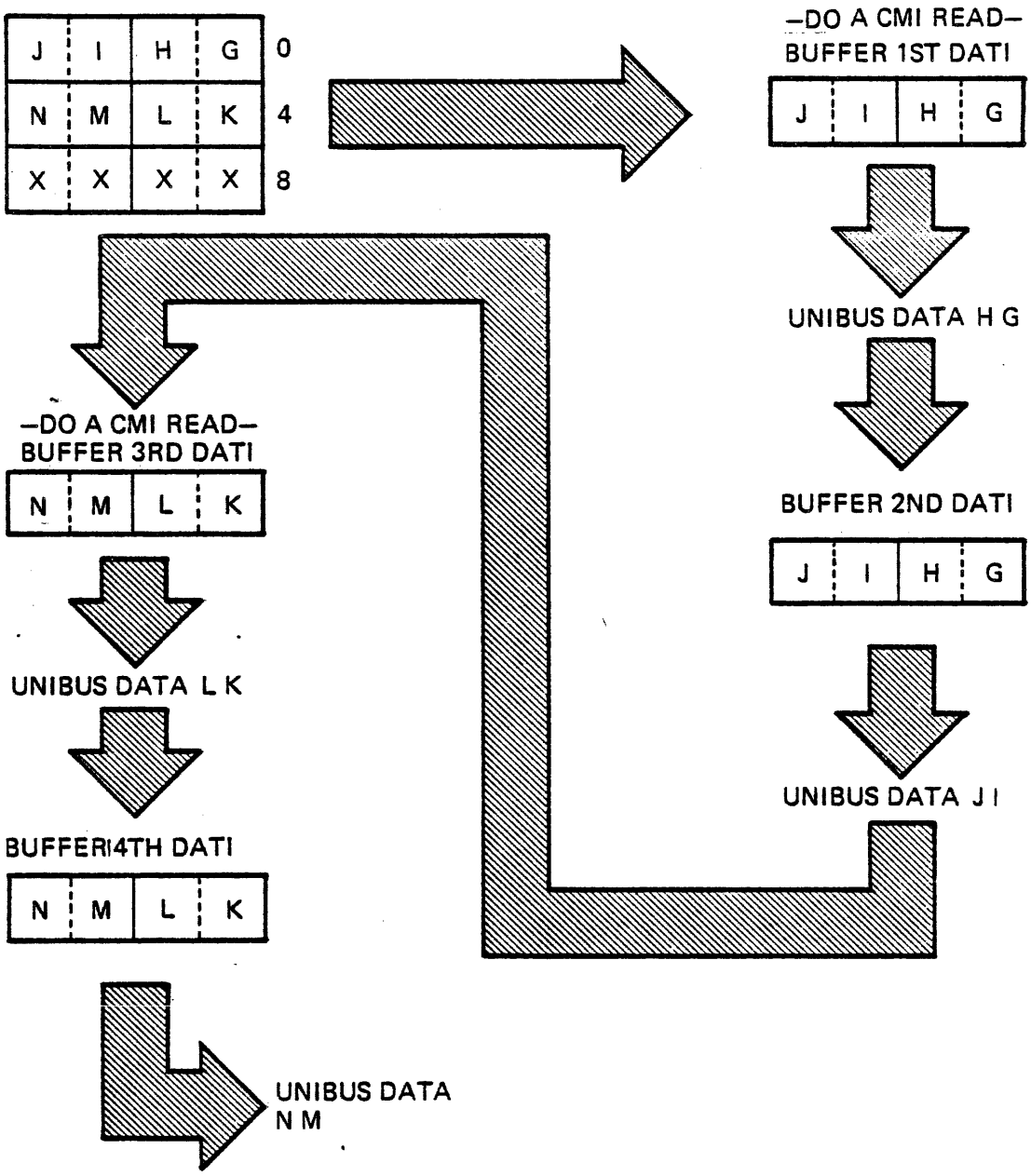

BUFFER 1ST DATO
B
A
BUFFER 2ND DATO
D
C
B
A
BUFFER FULL
DO A CMI WRITE
D
C
B
A
0
4
8
BUFFER 3 RD DATO
F
E
THATS ALL FOLKS
PURGE
DO A CMI WRITE
D
C
B
A
F
E
F
E
UNIBUS
DATA
CMI
ADDRESS
BA
DC
FE
0
2
4
—WRITE TO MEMORY
WITH DATA BUFFERING—
TK-2056

Figure 15-21



DEVICE REQUESTS 4 WORDS FROM CMI MEM LOCATIONS 0,2,4,6.

–READS FROM MEMORY
WITH DATA BUFFERING–

TK-2054

Figure 15-22

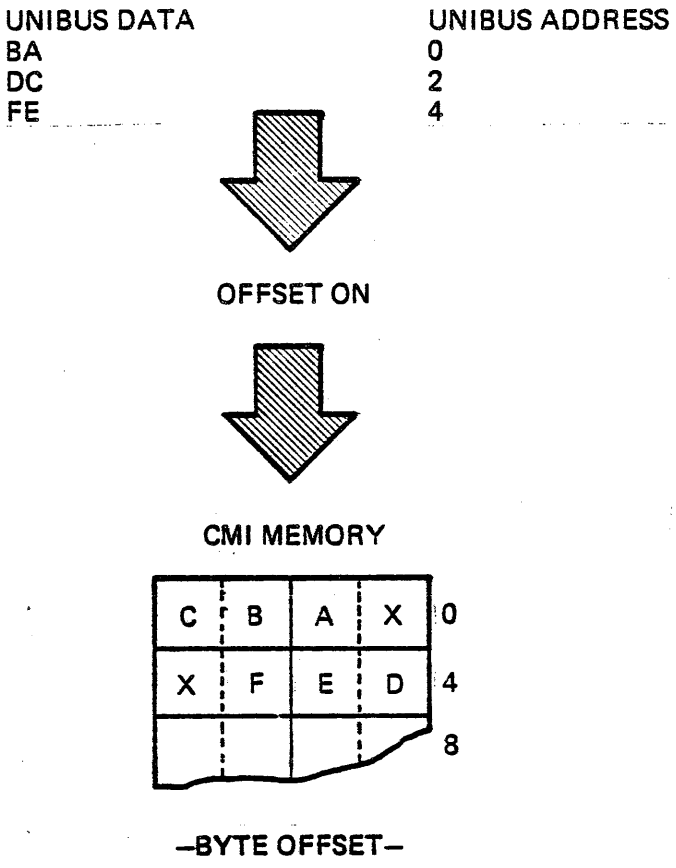

UNIBUS DATA
BA
DC
FE
UNIBUS ADDRESS
0
2
4
OFFSET ON
CMI MEMORY
C
B
A
X
0
X
F
E
D
4
8
–BYTE OFFSET–
TK-2055

Figure 15-23

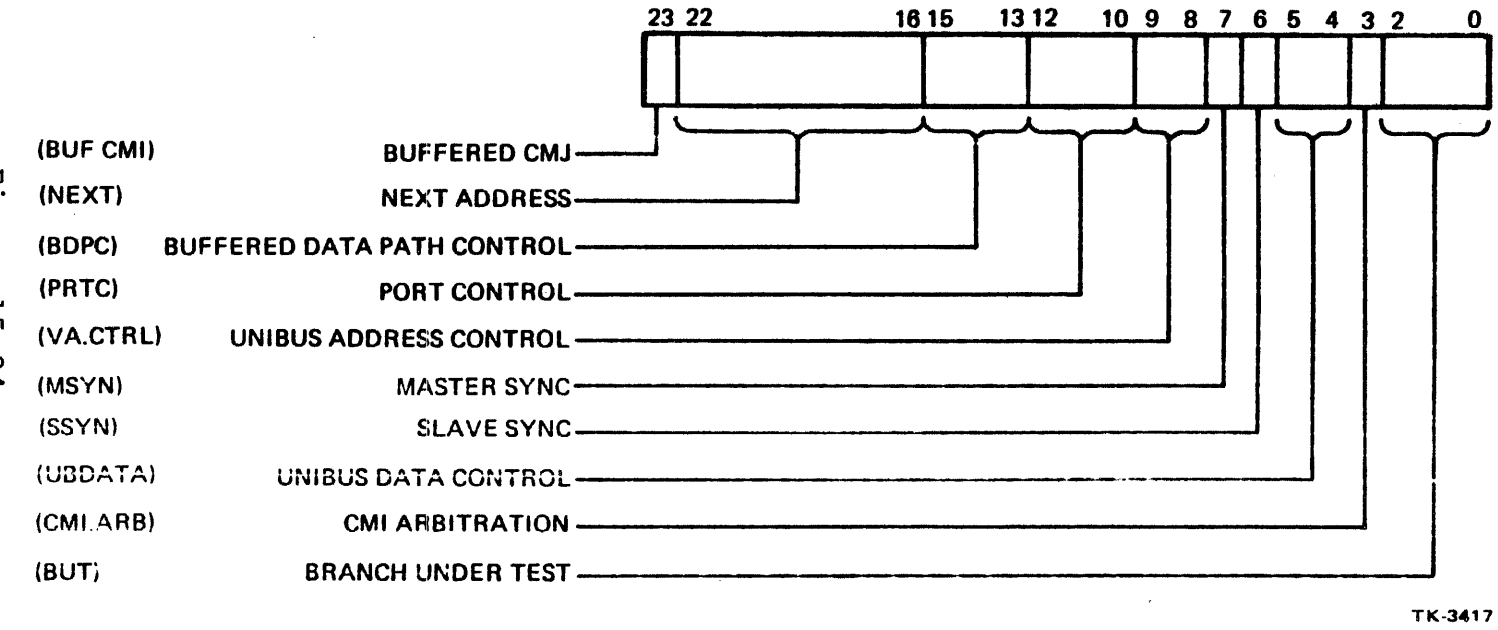

CUI MICROWORD
23 22
16 15
13 12
10 9 8 7 6 5 4 3 2 0
(BUF CMI)
BUFFERED CMJ
(NEXT)
NEXT ADDRESS
(BDPC)
BUFFERED DATA PATH CONTROL
(PRTC)
PORT CONTROL
(VA.CTRL)
UNIBUS ADDRESS CONTROL
(MSYN)
MASTER SYNC
(SSYN)
SLAVE SYNC
(UBDATA)
UNIBUS DATA CONTROL
(CMI.ARB)
CMI ARBITRATION
(BUT)
BRANCH UNDER TEST
TK-3417

Figure 15-24

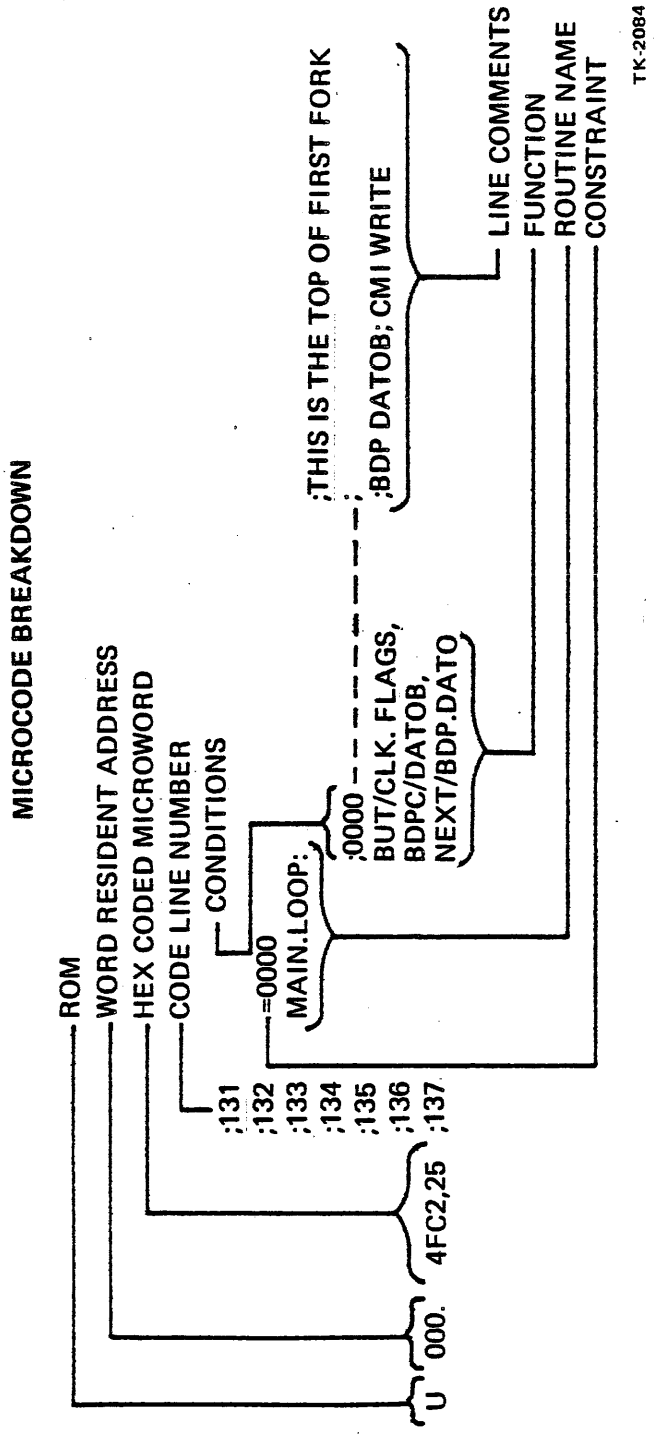

MICROCODE BREAKDOWN
ROM
WORD RESIDENT ADDRESS
HEX CODED MICROWORD
CODE LINE NUMBER
CONDITIONS
;THIS IS THE TOP OF FIRST FORK
;BDP DATOB; CMI WRITE
LINE COMMENTS
FUNCTION
ROUTINE NAME
CONSTRAINT
U
000.
4FC2,25
;131
;132
;133
;134
;135
;136
;137
=0000
MAIN.LOOP:
;0000
BUT/CLK. FLAGS,
BDPC/DATOB,
NEXT/BDP.DATO
TK-2084

Figure 15-25

CUI MICROWORD

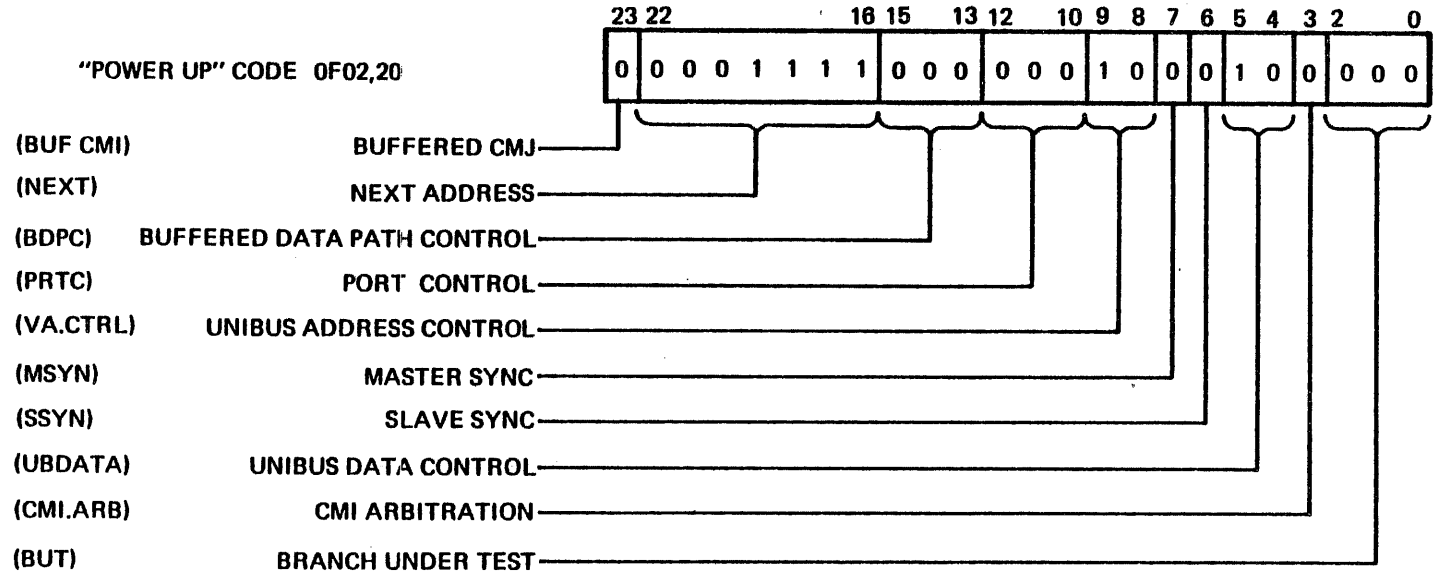


Figure 15-26

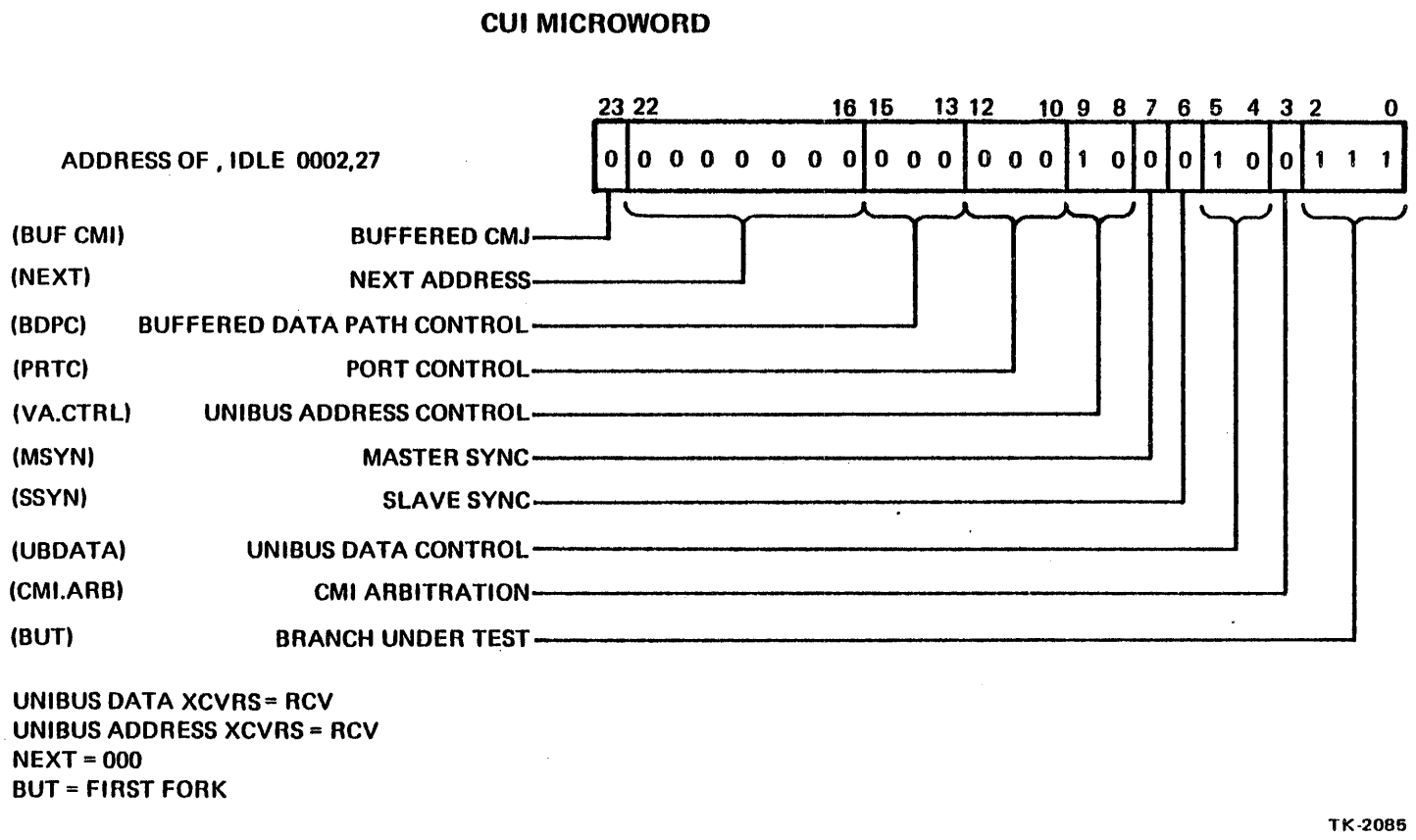

CUI MICROWORD
23 22 16 15 13 12 10 9 8 7 6 5 4 3 2 0
ADDRESS OF , IDLE 0002,27
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1
(BUF CMI) BUFFERED CMJ
(NEXT) NEXT ADDRESS
(BDPC) BUFFERED DATA PATH CONTROL
(PRTC) PORT CONTROL
(VA.CTRL) UNIBUS ADDRESS CONTROL
(MSYN) MASTER SYNC
(SSYN) SLAVE SYNC
(UBDATA) UNIBUS DATA CONTROL
(CMI.ARB) CMI ARBITRATION
(BUT) BRANCH UNDER TEST
UNIBUS DATA XCVRS = RCV
UNIBUS ADDRESS XCVRS = RCV
NEXT = 000
BUT = FIRST FORK
TK-2085

Figure 15-27

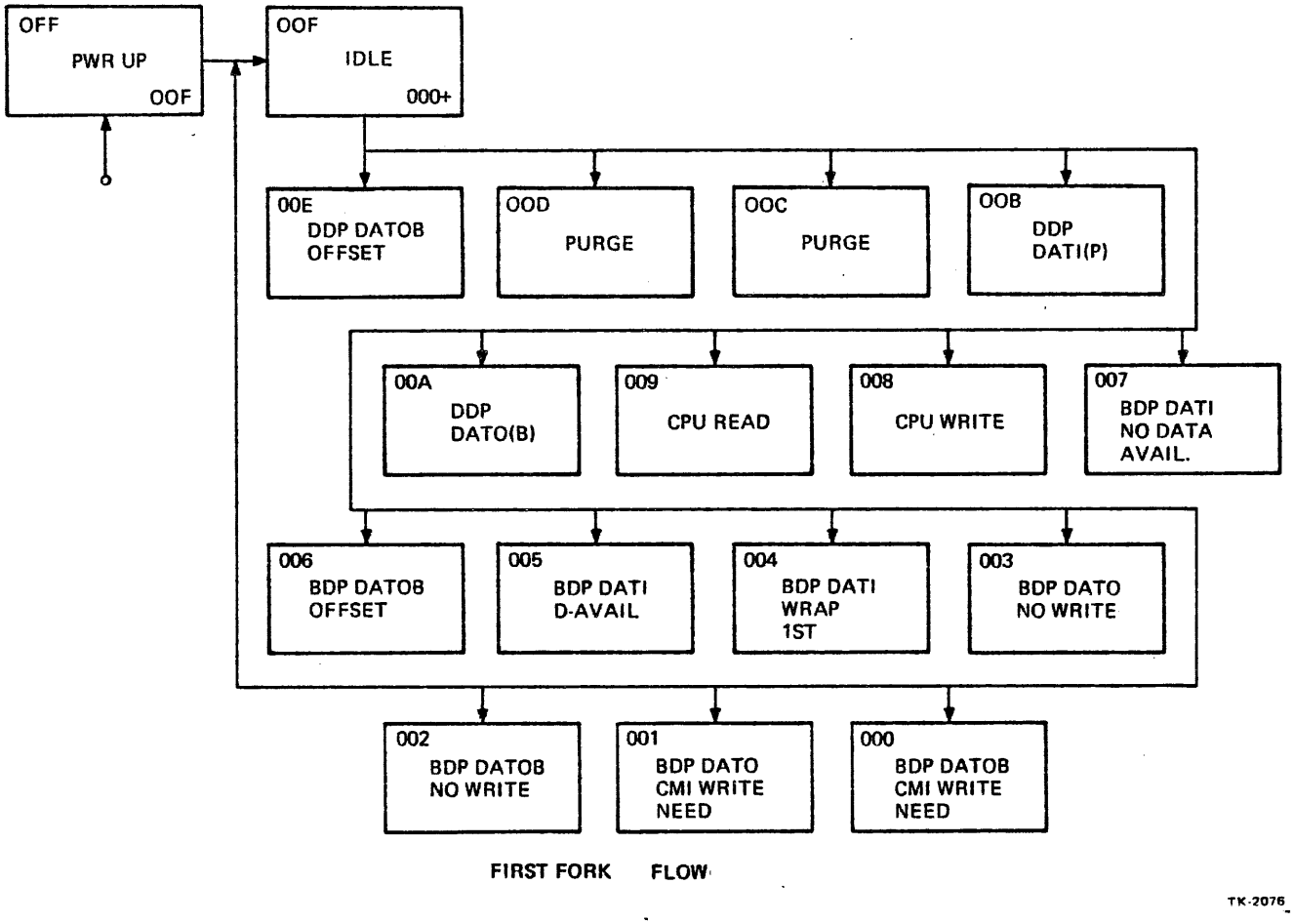

OFF
PWR UP
OOF
OOF
IDLE
000+
00E
DDP DATOB
OFFSET
OOD
PURGE
OOC
PURGE
OOB
DDP
DATI(P)
00A
DDP
DATO(B)
009
CPU READ
008
CPU WRITE
007
BDP DATI
NO DATA
AVAIL.
006
BDP DATOB
OFFSET
005
BDP DATI
D-AVAIL
004
BDP DATI
WRAP
1ST
003
BDP DATO
NO WRITE
002
BDP DATOB
NO WRITE
001
BDP DATO
CMI WRITE
NEED
000
BDP DATOB
CMI WRITE
NEED
FIRST FORK FLOW
TK-2076

Figure 15-28

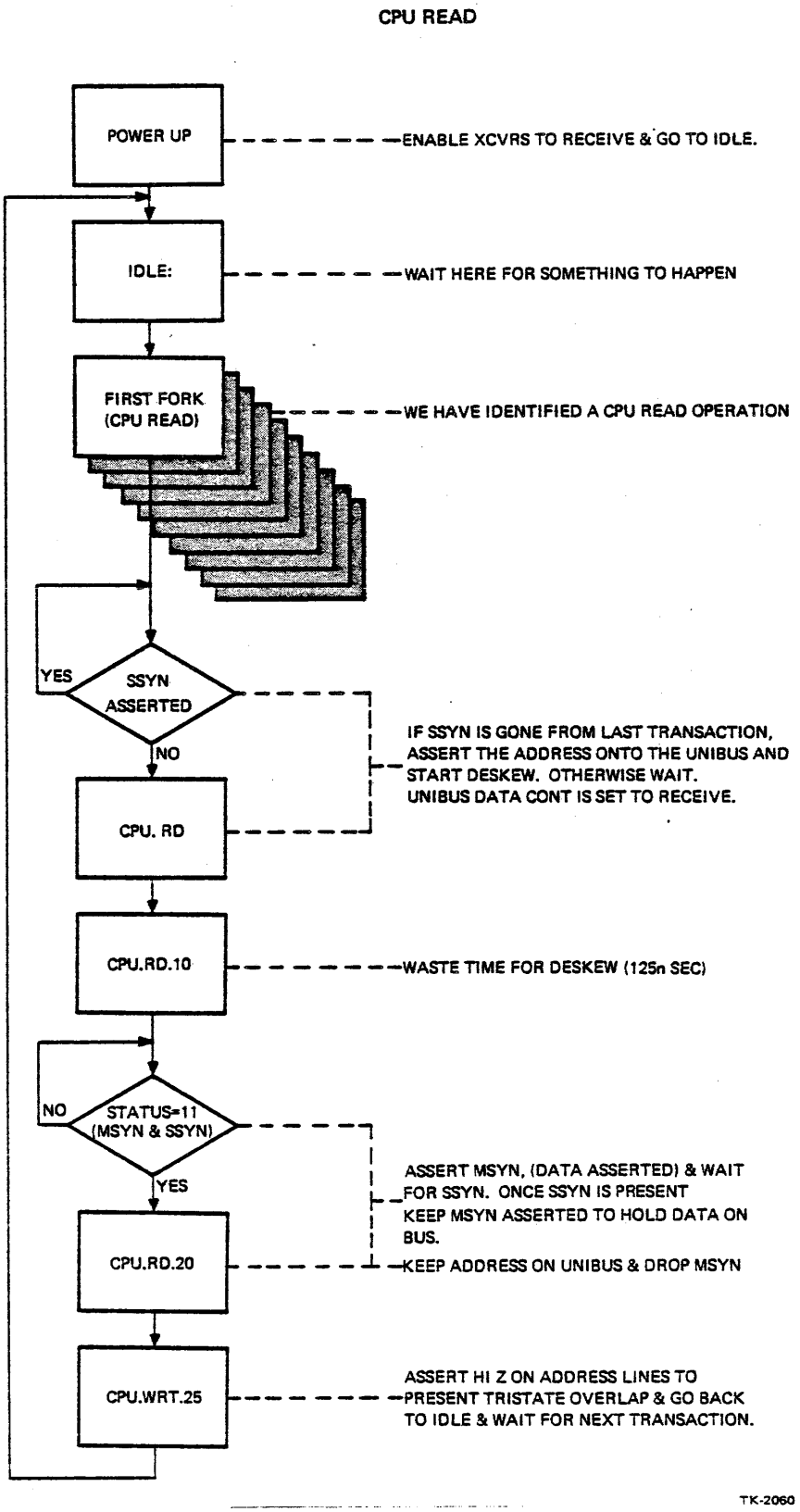

CPU READ
POWER UP
ENABLE XCVRS TO RECEIVE & GO TO IDLE.
IDLE:
WAIT HERE FOR SOMETHING TO HAPPEN
FIRST FORK (CPU READ)
WE HAVE IDENTIFIED A CPU READ OPERATION
YES
SSYN ASSERTED
NO
IF SSYN IS GONE FROM LAST TRANSACTION, ASSERT THE ADDRESS ONTO THE UNIBUS AND START DESKEW. OTHERWISE WAIT. UNIBUS DATA CONT IS SET TO RECEIVE.
CPU. RD
CPU.RD.10
WASTE TIME FOR DESKEW (125n SEC)
NO
STATUS=11 (MSYN & SSYN)
YES
ASSERT MSYN, (DATA ASSERTED) & WAIT FOR SSYN. ONCE SSYN IS PRESENT KEEP MSYN ASSERTED TO HOLD DATA ON BUS.
CPU.RD.20
KEEP ADDRESS ON UNIBUS & DROP MSYN
CPU.WRT.25
ASSERT HI Z ON ADDRESS LINES TO PRESENT TRISTATE OVERLAP & GO BACK TO IDLE & WAIT FOR NEXT TRANSACTION.
TK-2060

Figure 15-29

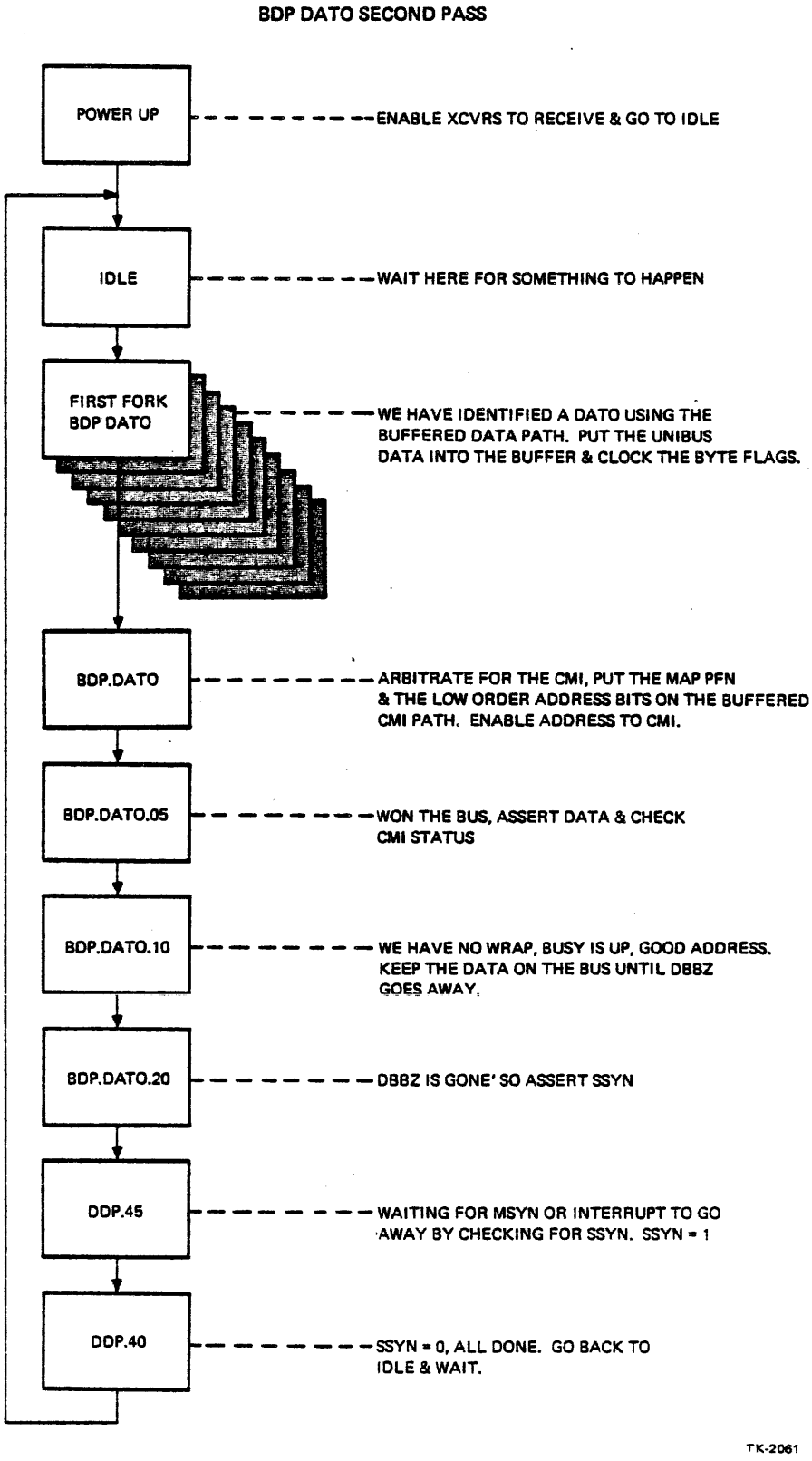

BDP DATO SECOND PASS
POWER UP
ENABLE XCVRS TO RECEIVE & GO TO IDLE
IDLE
WAIT HERE FOR SOMETHING TO HAPPEN
FIRST FORK
BDP DATO
WE HAVE IDENTIFIED A DATO USING THE BUFFERED DATA PATH. PUT THE UNIBUS DATA INTO THE BUFFER & CLOCK THE BYTE FLAGS.
BDP.DATO
ARBITRATE FOR THE CMI, PUT THE MAP PFN & THE LOW ORDER ADDRESS BITS ON THE BUFFERED CMI PATH. ENABLE ADDRESS TO CMI.
BDP.DATO.05
WON THE BUS, ASSERT DATA & CHECK CMI STATUS
BDP.DATO.10
WE HAVE NO WRAP, BUSY IS UP, GOOD ADDRESS. KEEP THE DATA ON THE BUS UNTIL D8BZ GOES AWAY.
BDP.DATO.20
D8BZ IS GONE' SO ASSERT SSYN
DDP.45
WAITING FOR MSYN OR INTERRUPT TO GO AWAY BY CHECKING FOR SSYN. SSYN = 1
DDP.40
SSYN = 0, ALL DONE. GO BACK TO IDLE & WAIT.
TK-2061

Figure 15-30

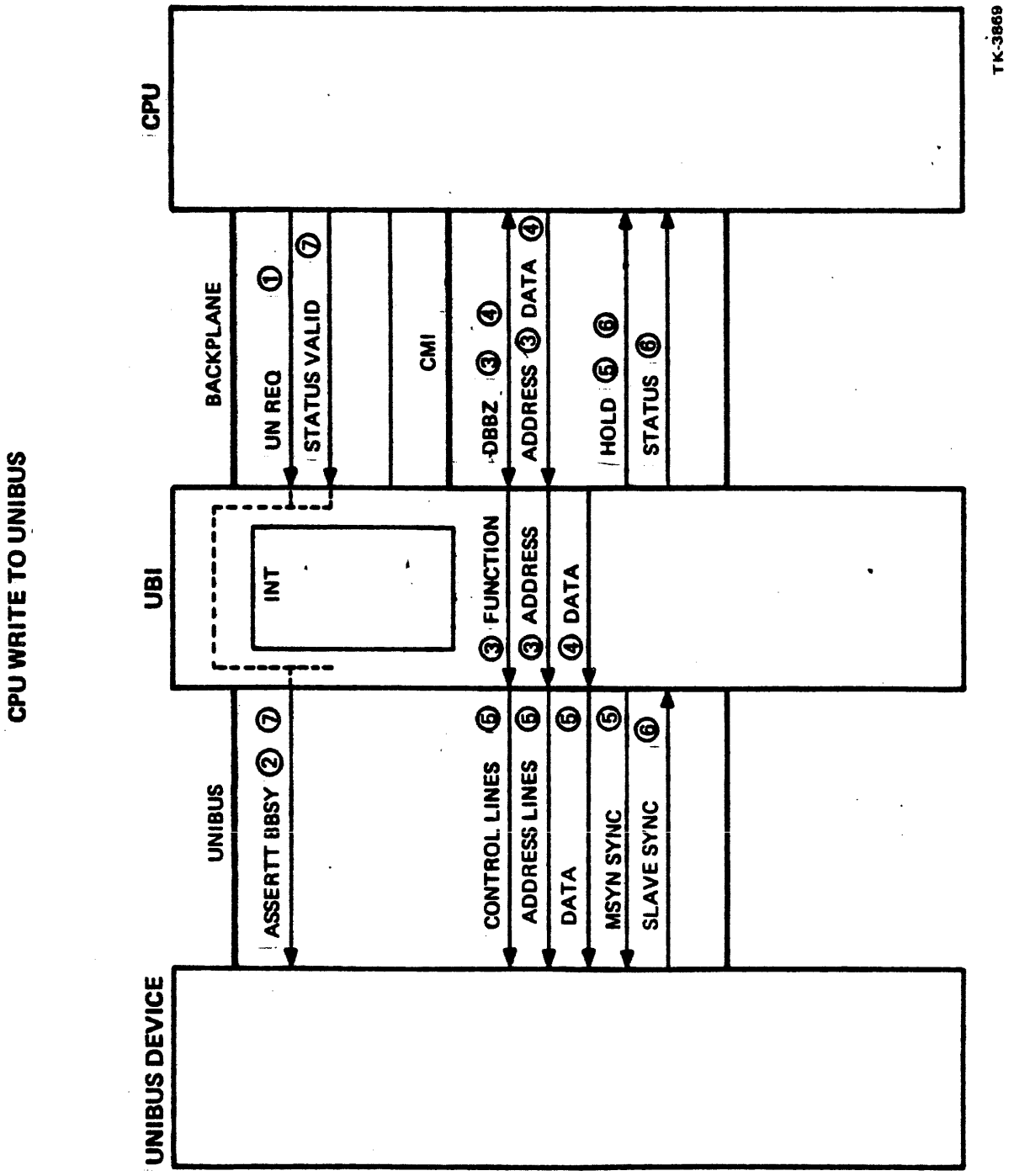


Figure 15-31 CPU Write to Unibus

Signal Explanation for CPU Write to UNIBUS

1. The MIC Module decodes the address to be sent to CMI. If address is a Unibus address, the MIC sends "un REQ" via the backplane to the UBI module. (This says arbitrate for the Unibus you have an address on theh way).

2. At this point, I'm saying the UBI arbitrated for the bus and won, and is asserting bus busy.

3. The CPU via the MIC module asserts DBBZ on the CMI along with the address and bus function of WRITE. When UBI receives it, enables go out to drivers to place address on Unibus. CØ and C1 generated from Function. NOT USED UNTIL SYNC IS SENT.

4. CPU drops DBBZ and UBI puts DBBZ on the line. CPU also sends data to be written on CMI. DATA is passed to Unibuss drivers that are enabled. NOT USED UNTIL MASTER SYNC SENT.

5. UBI sends Master SYNC and device should receive Address, data and control signals. UBI sets hold on CMI.

6. When slave sync is returned from the device, the UBI drops Hold and returns status to CPU via CMI.

7. UBI drops BBSY on Unibus when it receives "Status Valid" from MIC module.

DISK NPR XFER TO MEMORY, BDP EMPTY.

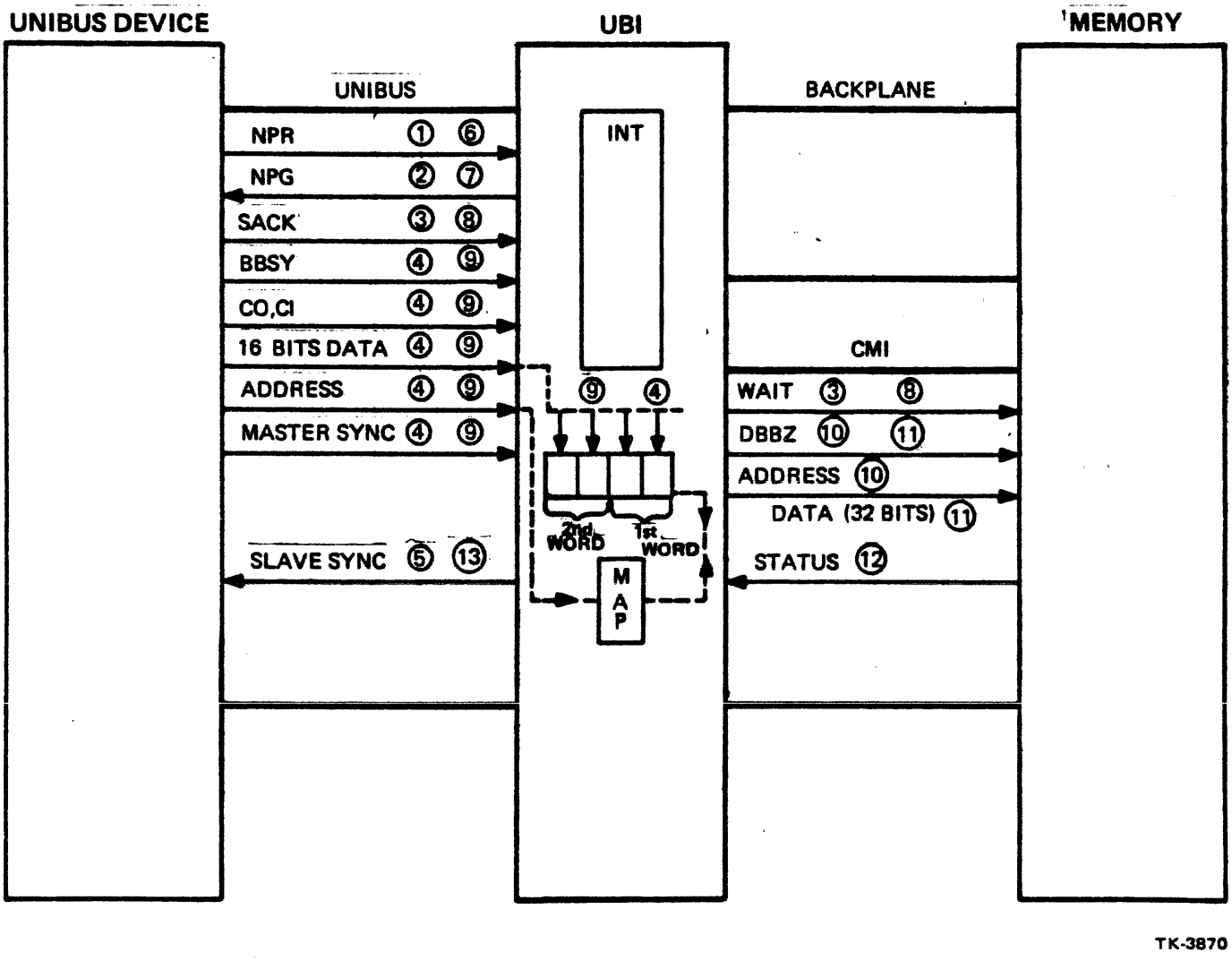


Figure 15-32 Disk NPR XFER to Memory BDP Empty

Signal Explanation for Normal NPR Disk Transfer to Memory

Ground rules for this explanation.

A. 16 Data lines on Unibus, 32 on CMI
B. BUffer in CMI is empty
C. Starting at even address

1. DIsk sends NPR on Unibus.

2. UBI arbitrates and sends back NPG.

3. DIsk receives NPG and sends SACK. UBI asserts wait when SACK received (not used here).

4. When bus available disk asserts BBSY, CØ, C1, ADDRESS, data and master sync. When UBI receives this it stores the 16 bits of data in a buffered data path because it is empty. (It holds 32 bits or 4 bytes). SACK is dropped to UBI so wait is dropped on CMI.

5. Slave sync is sent to device from UBI.

6. Since Unibus cycle complete another NPR is sent.

7. NPG again sent in response by UBI.

8. SACK sent from disk and UBI again asserts wait.

9. When Unibus available disk again asserts BBSY, CØ, C1, ADDRESS, Data and Master Sync. 16 bits of data now stored in remaining section of buffer, and address sent to map to get proper location in memory to send data to.

1Ø. UBI (when finished arbitrating for CMI) asserts DBBZ and Address on CMI.

11. UBI drops DBBZ and Memory Controller puts it on CMI. UBI asserts data (32 bits from BDP).

12. Status sent by controller to UBI, UBI clocks status when DBBZ deasserted.

13. SLAVE SYNC sent to Device.

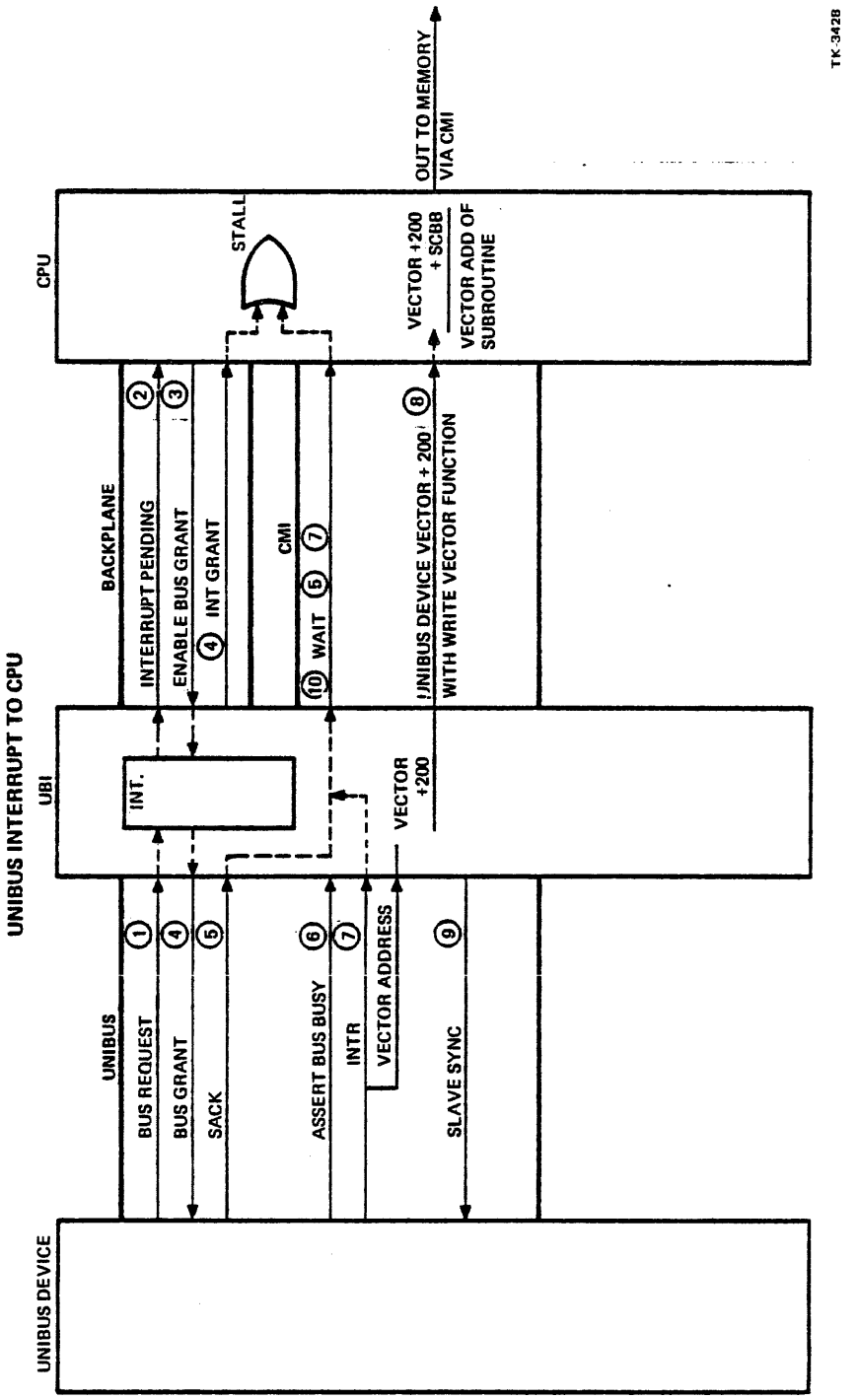


Figure 15-33 Unibus Interrupt to CPU

Signal Explanation for Unibus Interrupt to CPU

1. BR generated by unibus device. UBI Synchronizes BR to M clk to get SBR signal. SBR sent to Interrupt chip.

   The Interrupt chip checks SBR level (4, 5, 6, or 7) which will give you the corresponding IPL level of IPL 14, 15, 16, or 17. This level is compared to present IPL level and if SBR has a higher IPL then two things take place.

   A. INT pending signal sent to DPM module and MIC module

   B. The Interrupt chip on UBI also generates u vector lines Ø, 1 and 2 to the state needed to identify the type of interrupt pending.

2. Interrupt Pending sent by UBI to CPU modules DPM and MIC is used to generate remaining vector lines 3, 4, 5 to give you the proper micro vector address that starts the microroutine to handle the incoming interrupt. Caution; INT pending is sent to SAC chip on DPM while macro code is running but will not be interpreted until IRDI of next instruction (macro). when this time arrives THE SAC chip generates DO service signal and Enable u vector to go to the MSQ chip and generate bits 3, 4, and 5 to pull down the porper bits for the address of the microvector. When these signals are ored with bits Ø, 1, 2 from UBI module you have microaddress of the routine to handle the interrupt that is pending from Unibus.

3. The microroutine starts at microaddress and the first function is to send, via the WCNTL F.LA, A "33" which says Enable sending Bus Grant to the Int. chip on UBI.

4. The Int. chip (and associated logic) then sends the appropriate bus grant on the Unibus AND also hands INT grant back to the MIC module at the CMK chip. The CMK generates "GRANT STALL" to stall the microcode before the next microaddress. The CPU MICROCODE REMAINS STALLED UNTIL THE VECTOR HAS BEEN WRITTEN TO THE MIC Module.

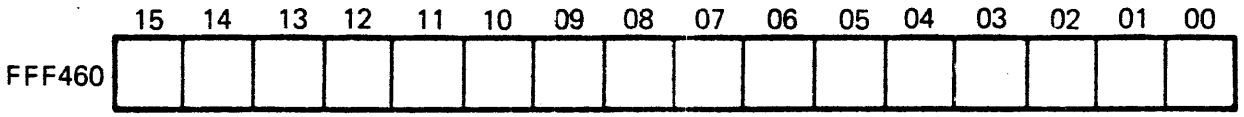
5. SACK is returned by the unibus device who issued the BR in response to the BG. SACK will at this time in the UBI module assert the "WAIT" line on the CMI. This will go to the MIC Module and replace int grant to hold the CPU stalled. It will also drop BG in the UBI.

6,7 When Unibus device that sent BR SEES Bus Busy on the Unibus dropped by previous device, he will assert bus busy, INTR and vector address (on DATA lines) all to be sent to the UBI module. At this time, INTR will replace SACK to keep the wait line pulled.

8. The UBI module sends Unibus Device Vector plus 200 (by pulling address line 9 low) on the CMI with a write vector function. This causes #9.

9. Slave sync is asserted on unibus because INTR and write vector sent. When device receives Slave Sync, INTR is dropped to UBI.

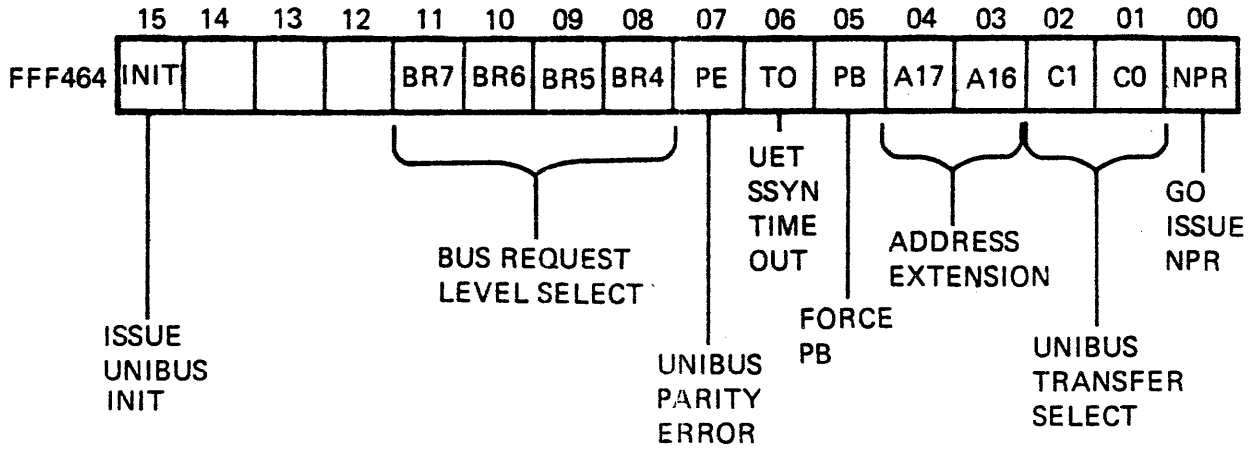
10. UBI no longer has INTR so the wait line is dropped causing stall to be dropped and the microcode goes to previously defined microaddress to handle interrupt. The loss of wait line to CMK chip on Mic is not the only way to install the machine. If the CMK monitors a write vector function on the CMI and sets the bit in write vector written register that will also install machine by dropping Grant Stall.



UNIBUS ADDRESS REGISTER

UNIBUS DATA REGISTER

UET CONTROL STATUS REGISTER



TK-5609

VAX 11/750 LEVEL II

Interrupts and Exceptions

Student Workbook

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The interrupt circuit is a composite of both TTL and gate array logic, with the center of attention focused on the INT chip housed on the UBI module.

The INT chip uses signals from other chips on the UBI board and signals originating on both the MIC module and the DPM module with control coming from the control store module via WCTRL.

The various signals are used to produce UNIBUS grants, U Vectors to the control store, and by way of the W Bus, route IPL data to and from the INT circuits.

MODULE XVI: INTERRUPTS AND EXCEPTIONS

SYNOPSIS

This module consists of theory on interrupts and exception handling utilizing block diagrams and microroutines.

OBJECTIVES

Utilizing the block diagram and the micro listing, trace the operation of the stack and associated circuitry while servicing each of the following:

a. traps
b. interrupts
c. exceptions

SAMPLE TEST ITEM

Which of the following is a typical example of an Exception?

a. A power failure.
b. The attempt to execute a privileged instruction.
c. A parity error.
d. An error detected on the Unibus.

RESOURCES

Processor Specifications

OUTLINE

XVI. Interrupts and Exceptions

- A. Interrupt
- B. Exception
  1. Execution of Privilged or Reserved Instructions
  2. Trace Traps
  3. Compatability Mode Faults
  4. Breakpoint Instruction Execution
  5. Arithmetic Traps
- C. 3 Types of Exceptions
  1. Traps
  2. Fault
  3. Abort
- D. Interrupt Priority Level
- E. Vector
- F. System Control Block Base Register
- G. Interrupt Block Diagram
  1. INT Chip
  2. INT Chip Inputs
  3. INT Chip Outputs
- H. Interrupt Registers
  1. SPFIR
  2. WEIR

3. CPIR

4. CDIR

5. HSIPR

6. IPL

7. IS

8. CURMODE

9. ASTLVL

I. Operations Performed

1. Save and return values of parts of the PSL and AST level via the W Bus.

2. Receiving and storing the value of the HSIPR which is used in interrupt arbitration.

3. Placing various data onto the Micro Vector lines.

4. Perform REI check calculations.

5. Arbitration of all interrupt requests, encoding the highest priority pending interrupt and generation of the interrupt pending signal.

6. Unibus arbitration within the group of BR devices and issuing of BGs.

7. Unibus ACLO/DCLO, initiate functions are handed by the INT Chips associated TTL circuits.

8. The generation of Micro sequencer INIT is also handled by the associated TTL circuitry.

9. Unibus arbitration among the CMI, NPR devices and the BR devices.

10. Informing the CMI that it may talk to the Unibus.

11. Request Unibus number 2 via NPR.

J. Interrupt and Exception Microcode for a Unibus INT

1. I and E Mic, IE.UNIBUS.INT:

2. Address OF1A

3. Address OF1B

4. Address OF1C

5. Address OF1D

6. Address OF1E

7. Address OF1F

8. Address OFØØ

K. Summary

INTERRUPT AND EXCEPTIONS

- INTERRUPT - An event other than an exception, branch, jump, case, or call instruction that changes the normal flow of instruction execution. Interrupts are generally external to the process executing when the interrupt occurs.

- EXCEPTION - An event detected by 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. There are 3 types of exceptions.

  - TRAP - An exception conditions 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.

  - FAULT - A condition that occurs in the middle of an instruction that leaves the registers and memory in a consistant state which allows the instruction to restart and for correct results once the fault has been cleared or eliminated.

  - ABORT - An exception that occurs in the middle of an instruction and leaves the registers and memory in an indeterminate state which may prohibit an instruction restart.

Figure 16-1

VECTORS 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
SCBB+8	KERNEL STACK INVALID	1F	E
SCBB+C	POWER FAIL	1E	E
SCBB+10	RESERVED OPCODE	1F	E
SCBB+14	CUSTOMER OPCODE XFC	1F	E
SCBB+18	RESEVED OPERAND	1F	E
SCBB+1C	RESERVED ADDRESS MODE	1F	E
SCBB+20	ACCESS VIOLATION	1F	E
SCBB+24	TRANSLATION INVALID	1F	E
SCBB+28	TRACE TRAP	1F	E
SCBB+2C	BREAKPOINT OPCODE	1F	E
SCBB+30	COMPATABILITY MODE	1F	E
SCBB+34	ARITHMETIC TRAP	1F	E
SCBB+40	CHMK	1F	E
SCBB+44	CHME	1F	E
SCBB+48	CHMS	1F	E
SCBB+4C	CHMU	1F	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	TU-58 RECEIVE	17	I
SCBB+F4	TU-58 TRANSMIT	17	I
SCBB+F8	CONSOLE RECEIVE	14	I
SCBB+FC	CONSOLE TRANSMIT	14	I
SCBB+160	MASSBUS ADAPTOR 0	15	I
SCBB+164	MASSBUS ADAPTOR 1	15	I
SCBB+168	MASSBUS ADAPTOR 2	15	I
SCBB+200	UNIBUS (SCBB+200+UNIBUS VECTOR)	14-17	I

TK-3273

Figure 16-2 Vector and System Control Block Format

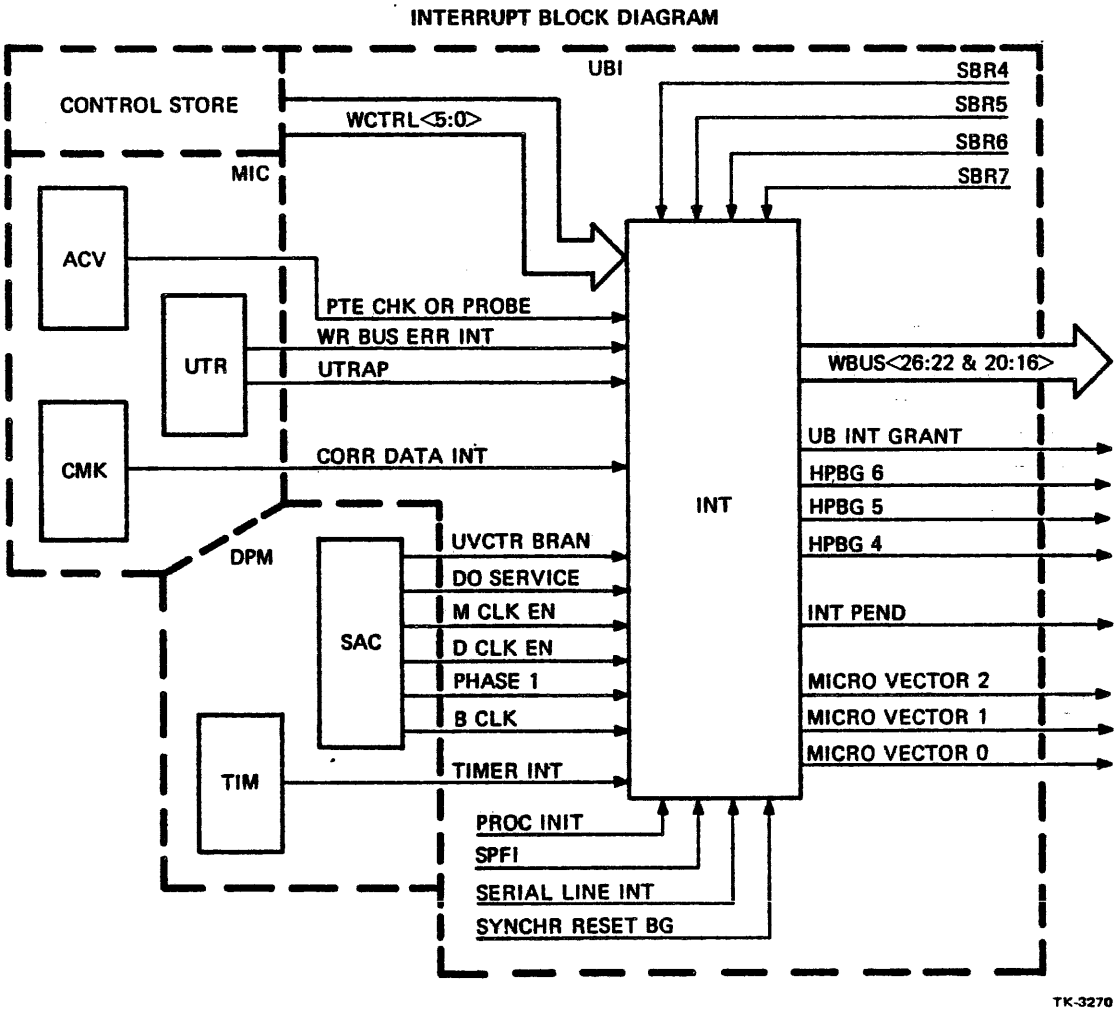


Figure 16-3 Interrupt Block Diagram

MICRO VECTOR VALUE CHART

IPL	NAME	U VECTOR
00	NO INTERRUPT REQUEST PRESENT	000
01-0F	(HSIPR) HIGHEST SOFTWARE INTERRUPT PENDING REQ.	000
14	(SLINE INT) SERIAL LINE INTERRUPT	001
14-17	(SBRn) SYNCHRONOUS BUS REQUEST (4-7)	010
18	(TIMER INT) INTERVAL TIMER INTERRUPT	011
1A	(CDIR) CORRECTED DATA INTERRUPT REQUEST	100
1B	(CPIR) CACHE PARITY ERROR INTERRUPT REQ.	101
1D	(WEIR) WRITE BUS ERROR INTERRUPT REQUEST	110
1E	(SPFIR) SYNCHRONOUS POWER FAIL INTERRUPT REQUEST	111

BRANCH ON MICROTEST MICRO TARGETS

38=SOFT INTERRUPT REQUEST
39=CONSOLE INTERRUPT REQUEST
3A=UNIBUS INTERRUPT REQUEST
3B=INTERVAL TIMER INTERRUPT
3C=CORRECTED MEMORY DATA
3D=CACHE PARITY ERROR
3E=WRITE BUS ERROR
3F=POWER FAIL INTERRUPT

TK-3272

Figure 16-4 Microvector Value Chart

INT REGISTERS

NAME	NUMBER OF BITS	IPL	COMMENTS
SPFIR	1	1E	LATCH FOR SPFI
WEIR	1	1D	LATCH FOR WEI
CPIR	1	1B	LATCH FOR CPI
CDIR	1	1A	LATCH FOR CDI
HSIPR	4	01-0F	SOFTWARE INTERRUPTS
IPL	5	00-1F	INTERRUPT PRIORITY LEVEL
IS	1	–	INTERRUPT STACK FLAG
CURMODE	2	–	CURRENT MODE
ASTLVL	3	SOFT	ASYNCHRONOUS SYSTEM TRAP LEVEL
LUBIPR	2	14-17	LAST GRANTED UNIBUS IPR
PRVMODE	2	–	PREVIOUS MODE

TK-3271

Figure 16-5 Interrupt Registers

Interrupts and exceptions in some ways are alike in what they accomplish within the machine. They also have their differences. To understand the differences and how they both function you must be able to answer two questions.

1. Why do interrupts and exceptions occur?

2. How does the machine get to the proper MICRO address to handle interrupts and exceptions when they occur?

We'll attempt to answer these questions one at a time to give you an idea how the machine handles interrupts and exceptions.

1. Why do interrupts and exceptions occur?

When the machine is running normally it is executing one MACRO instruction at a time. After completion of the MACRO the machine goes to the execution buffer on the MIC to fetch and execute another MACRO instruction. This is the normal flow of operation. There are some MACRO instructions that during their execution go to different areas of the program and not the next MACRO instruction in the execution buffer; such as jump, branch etc. These examples are still a normal type flow to the machine. When a normal flow needs to be changed either an interrupt of an exception occurs. IN ALL CASES A MICRO ADDRESS IS GENERATED THAT POINTS TO A ROUTINE TO HANDLE THE INTERRUPT OR EXCEPTION.

INTERRUPT - An event other than an exception, branch, jump, case or call instruction that changes the normal flow of instruction execution. Interrupts are generally external to the process executing when the interrupt occurs. EXAMPLE: Interrupt from the console terminal.

EXCEPTION - An event detected by 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. There are three types of exceptions.

A. 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 be executed. EXAMPLE: Arithmetic Trap

B. Fault - A condition that occurs in the middle of an instruction that leaves the registers and memory in a

constant state which allows the instruction to restart and for correct results once the fault has been cleared or eliminated. EXAMPLE: Translation Buffer Miss

C. Abort - An exception that occurs in the middle of an instruction and leaves the registers and memory in an indeterminate state which may prohibit an instruction restart. EXAMPLE: Kernel stack invalid.

These are the reasons that interrupts and exceptions occur. The second and maybe the most important question is:

2. How does the machine get to the proper MICRO address to handle the interrupts and exceptions when they occur?

To figure this out let us redefine some terms previously explained. Look in the micro-code listings in the chart file for the FIXED CONTROL STORE ADDRESS CHART. You should note that these addresses are broken down into three sections; 1X, 2X and 3X. These sections relate to the MICRO addresses that all interrupts and exceptions go to when handling the events other than the normal flows within the machine. If there are three types of exceptions along with interrupts why aren't there four sections? The answer lies in terminology.

An interrupt is an interrupt and all of them start at MICRO addresses beginning with 3(X).

Exceptions are where the ambiguous statements begin.

A TRAP is an exception, explained previously, that when generated will always go to MICRO addresses beginning at 1(X).

This leaves two types of exceptions that relate to one group FIXED CONTROL STORE ADDRESSES. The exceptions fault and abort, explained previously, are both classified as MICRO TRAPS in the FIXED CONTROL STORE ADDRESS CHART. Using these redefined terms for faults and aborts they will now both be discussed as MICRO TRAPS from now on.

Looking at the redefined terms what follows is a brief overview of what happens at the time an interrupt, trap or micro trap occurs. Knowledge of the Micro Sequencer Chip (MSQ) from the manual or class is assumed.

After the execution of each macro-instruction, a test must be made to see if there are any traps or pending interrupts to be serviced. This test is called BUT SERVICE. It is done by the hardware one cycle after the completion of the macro-instruction to allow the Condition Codes to become stable for checking overflow. This is one cycle after the first IR Decode branch of the next micro-instruction. There is no micro-order which invokes this test.

If a trap condition or interrupt is pending, then the micro-vector associated with the highest priority event is asserted on the Control Store address lines overriding the address mode specified by the current micro-instruction. The address of the instruction during which the test was made is pushed on the micro-stack. When the BUT SERVICE test is true, all action in the micro-cycle is inhibited by the hardware. This includes starting bus cycles, updating the PC, IR, or OSR and writing destinations.

If there is a micro-trap condition in the same micro-cycle in which the BUT SERVICE test is true, then BUT SERVICE has higher priority than the micro-trap condition. The reason for this is that the micro-trap was caused while attempting to execute the next macro-instruct. The exception to that is a Control Store Parity Error which overrides the BUT SERVICE test.

During the execution of long macro-instructions interrupts can be detected by micro-orders in the BUT field. If an interrupt has occurred then a micro-branch to the appropriate service routine is specified.

A micro-trap is a mechanism for handling conditions which prevent a micro-instruction from completions successfully. The micro-sequencer does a utrap operation at the end of the micro-cycle by forcing a JSR to a routine which corrects the problem. After the condition has been corrected, the routine returns to the micro-instruction and re-executes it. This transaction is transparent to the micro-programmer.

With this in mind let us find out how the machine actually gets to the physical MICRO-ADDRESS to handle the proper Interrupt, Trap or Micro-Trap.

All Interrupts are generated from the Interrupt chip located on the UBI module and, depending on which Interrupt is generated goes to MICRO-ADDRESS locations 38 through 3F. How are these addresses generated? The address comes from three location in the CPU, broken down by bits 5-Ø.

Bits Ø, 1 and 2 come from the INTERRUPT CHIP on the UBI Module.

Bit 3 comes from the UTRAP CHIP on the M/C Module.

Bits 4 and 5 are generated from the MSQ CHIP on the DATA PATH Module.

As the machine is running macro code a request for an interrupt is sent to the INTERRUPT CHIP. As an example we will use a Console Interrupt, although ALL interrupts are handled the same way. The Interrupt Chips will check the IPL of the request to the IPL now in its IPL Register. (In the Int. Chip.) If the requested IPL is higher then the signal INT PENDING is sent to the SAC CHIP on the Data Path Module. Nothing will happen until IRD1 of the MACRO Instruction that is now running in the machine. What happens when IRD1 is decoded by the SAC from the BUT FIELD is going to cause action by the INT. Chip, MSQ Chip, and U Trap Chip simultaneously, but on paper can only be explained one chip at a time.

When IRD1 is decoded and INT Pending is asserted the SAC chip generates DO Service L and ENABLE UVECTOR H. These signals will cause the three previously mentioned chips to output the needed MICRO ADDRESS 39 for CONSOLE INTERRUPT.

INT. CHIP - When request for console interrupt n DO service L was generated by SAC the Interrupt Chip, on the trailing edge of MCLK, allow the bits to be driven onto the uvector lines. These bits would equal ØØ1 for Console Interrupt.

MSQ CHIP - When DO Service L and ENABLE UVECT H are generated, along with the fact MICRO ADDS INH L is H, and MSEQ INIT L is H, the MSQ chip outputs a 3Ø onto uvector lines. This always occurs on Interrupt. (See Chart.) Use for Traps and micro traps also.

Micro ADDR Inhibl	Enable uvector H	Do Serv L	Internal MSQ Initial	CS Lines 5-Ø
H	H	H	H	2Ø HEX
H	L	L	H	1Ø HEX
H	H	L	H	3Ø HEX

UTRAP CHIP - The signal DO Service L also goes to the TRI-STATE driver to uvector Line 3 to be shut off and float

to a High which is a 1. This line is sent to the DPM (just below the MSQ chip in prints) and is allowed to be ored with the 3Ø from MSQ chip due to the fact uvector is enabled.

What we end up with is the Micro address 39 to be sent to the control store module. That answers the question of how the micro-address was generated for an Interrupt.

How is the Micro address generated for a Trap Condition?

TRAP - As you can see by the "FIXED CONTROL ADDRESS" chart all traps are 1X. This will be a simpler action because there are only Two Chips involved. SAC for bits Ø, 1 and 2. MSQ for bits 3, 4, and 5.

If you look at the 5 possible traps and look at the SAC chip you will see that all 5 traps feed directly to the CHIP.

11 = Arith Trap
12 = FPA Trap
14 = Timer Services
15 = T-Bit Trap (PSL TB)
16 = Console ^P Trap (Console Halt)

SAC - During IRD1 But Service is performed and a Trap Condition exists for an Example we will use ARIH Trap L. The SAC chip at this time only outputs DO Service L. NOT ENABLE uvector. The SAC chip also outputs to the uvector lines ØØ1 for bits 2, 1, Ø. These lines will determine which type of Trap exists.

MSQ - Do Serv L is received from SAC and referring to previous chart you find the #1Ø being outputted to uvector lines. #1Ø or with 1 gives you the #11 to the control store for the proper address to handle the Arith Trap.

That should answer how the proper Trap address gets to the Control Store.

How does a Micro-Trap address get sent to the Control Store? We know that a micro trap is a mechanism for handling conditions which prevent a micro-instruction from completions successfully. To do this the proper address needs to be sent to the control to the routine to handle the condition.

MICRO-TRAP - To find how the address is generated you must go to the UTrap Chip on the MIC mode and the list of Micro Traps in the "Fixed Control Address" Chart. We will not be concerned at this point which micro-trap has priority over

the others as that is covered in the MIC section. We are concerned with how the address is generated. For an Example we will take a TB miss UTrap, 2A. You notice all micro-Traps are 2X. These chips are involved Utrap chip, SAL and MSQ.

UTRAP CHIP - The utrap chip is constantly monitoring the events during each micro instruction and if a Translation Buffer Miss occurs during read MICRO-instruction it cannot be completed. You trap chip see the TB Parity Era H signal and NO TB Hit from Ø or 1 the signal Utrap is generated and the uvector lines 3-Ø are set to 1Ø1Ø (but not driven until trailing edge of MCLK).

SAC - The Sac chip receives Utrap L signal and generates ENABLE UVECTOR H, but NOT do service.

MSQ - The MSQ receives the Enable uvector H signal and referring to previous chart you find the MSQ chip outputs 2Ø to be ored with the uvector lines that are just below the MSQ in prints that come from the UTrap Chip which are allowed to pass with the Enable uvector H signal from SAC chip. The address 2A is sent to the control store for the routine to handle a TB miss.

This should explain how the address of a micro-trap is sent to the control store module.

VAX-11/750 LEVEL II

Console Interface

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The console interface chip is an asynchronous serial line interface between the COMET console terminal and the CPU. It contains logic to do limited character recognition of the received characters for entering in Console Mode. It asserts signals to request both micro and macro level interrupts. Addressing of internal registers is indirect through Console Register Address Register (CRAR).

SYNPOSIS

This module is designed to be a block diagram and chip level presentation on the following:

a) Transmit parallel to serial converter
b) Receive serial to parallel converter
c) Microroutine

OBJECTIVES

Given a block diagram of the Console Interface, correctly identify each block by labeling it.

Provided with a list of Console Interface functions and signals, correctly match the signal to its function.

SAMPLE TEST ITEM

Utilizing the CUI Print Set, match the below listed Console Interface signals to their functions:

a)	INSTR FETCH H	Disables console interface
b)	SERIAL LINE INT.	Asserted when data is available for output
c)	CON SO	When asserted, equals data out
d)	M CLK L	Latch data in

RESOURCES

Comet Specifications
Comet Print Set

MODULE OUTLINE

XVII. CONSOLE INTERFACE

A. CON Chip

1. WCTRL
2. W-BUS
3. SERIAL IN/OUT

B. Transmitter Half of CON Chip

C. Receiver Half of CON Chip

D. Print Famliarization

E. Summary

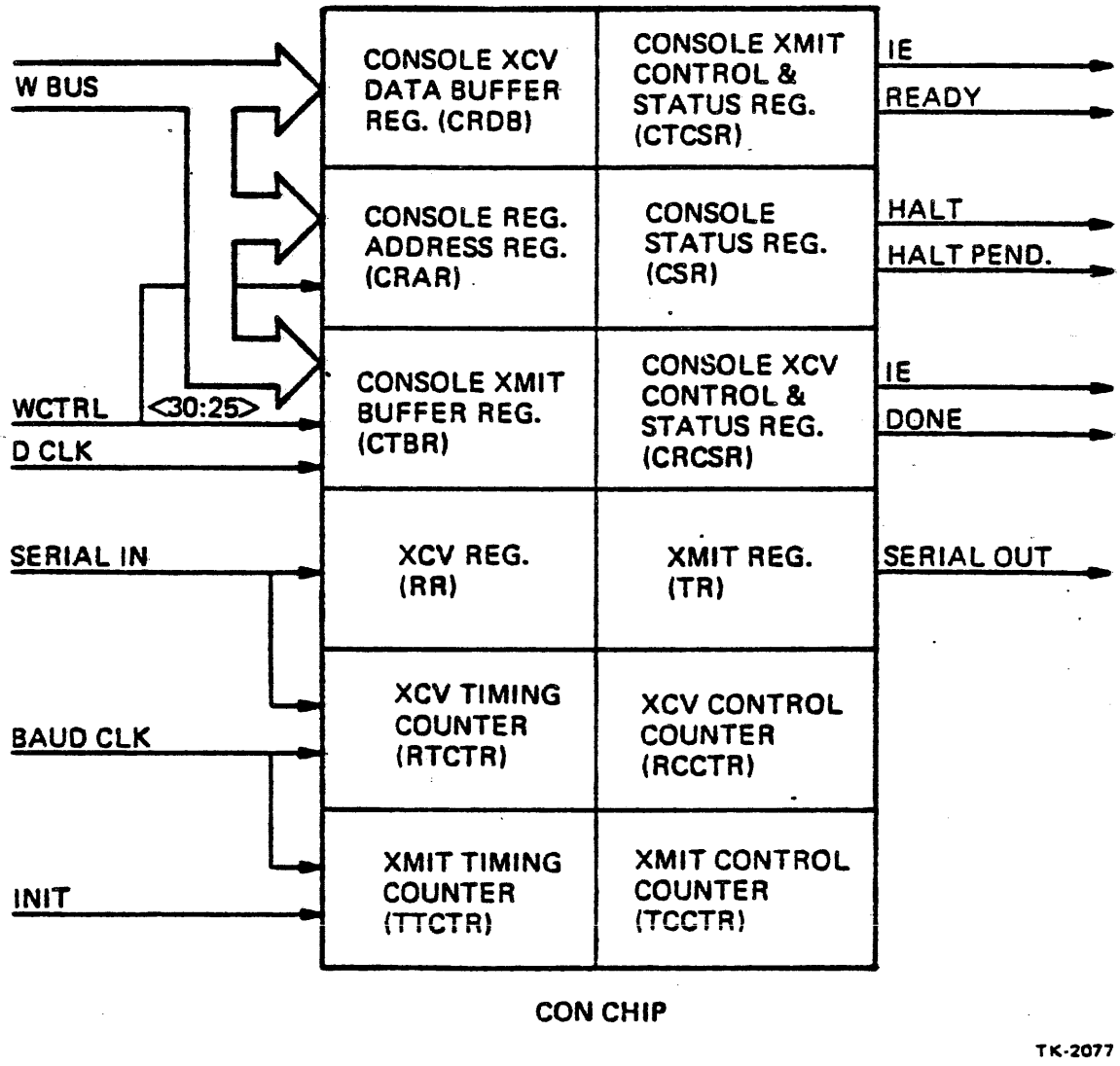


Figure 17-1 CON Chip Input/Output Signals

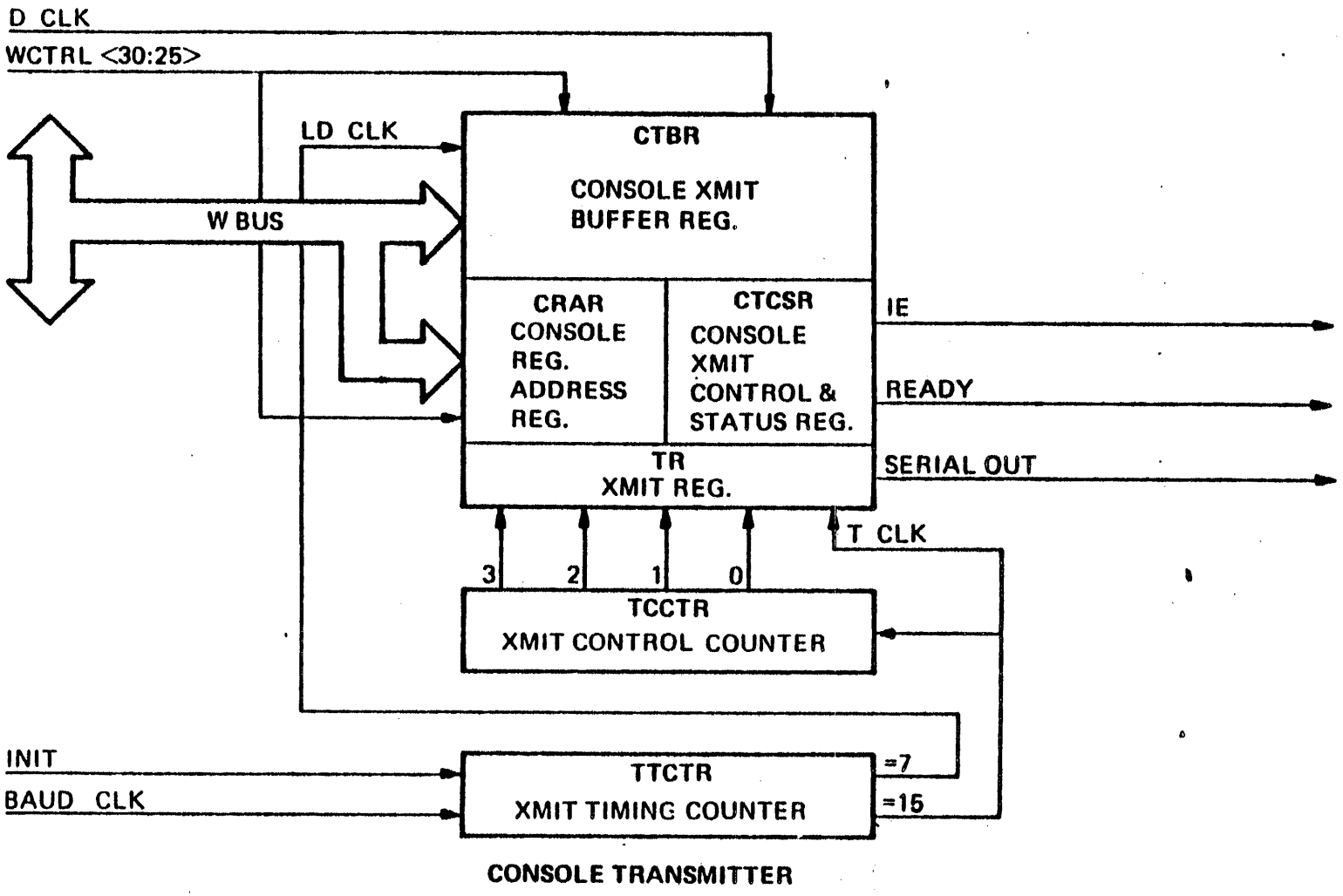


Figure 17-2 Console Transmitter

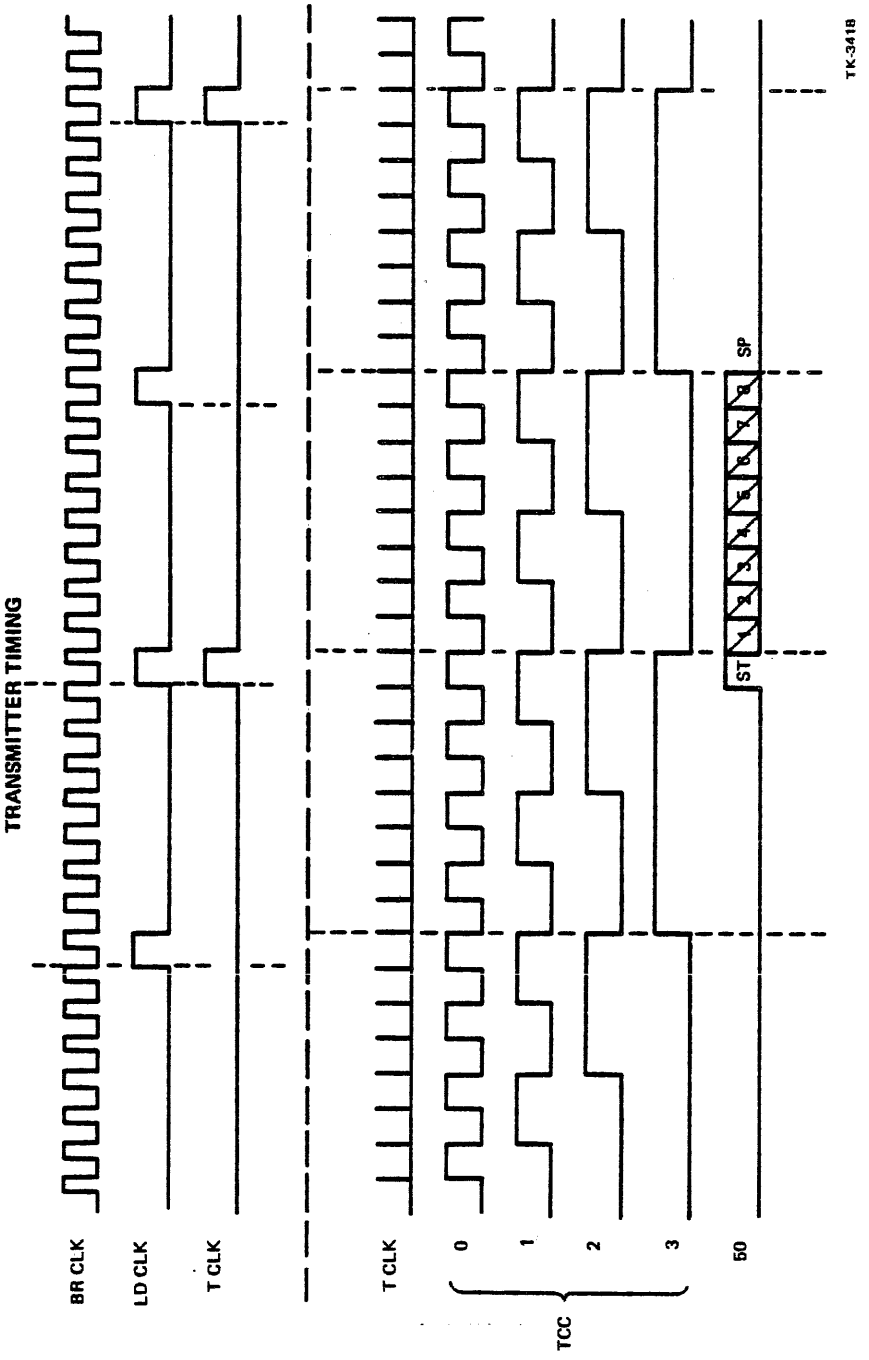


Figure 17-3 Transmitter Timing

CRAR
CONSOLE REG. ADDRESS REG.
ADDRESS FORMAT

REGISTER	ADDRESS
CTDB	00
CRDB	00
CTCSR	01
CRCSR	10
CSR	11

TK-2042

Figure 17-4 Console Register Address Format

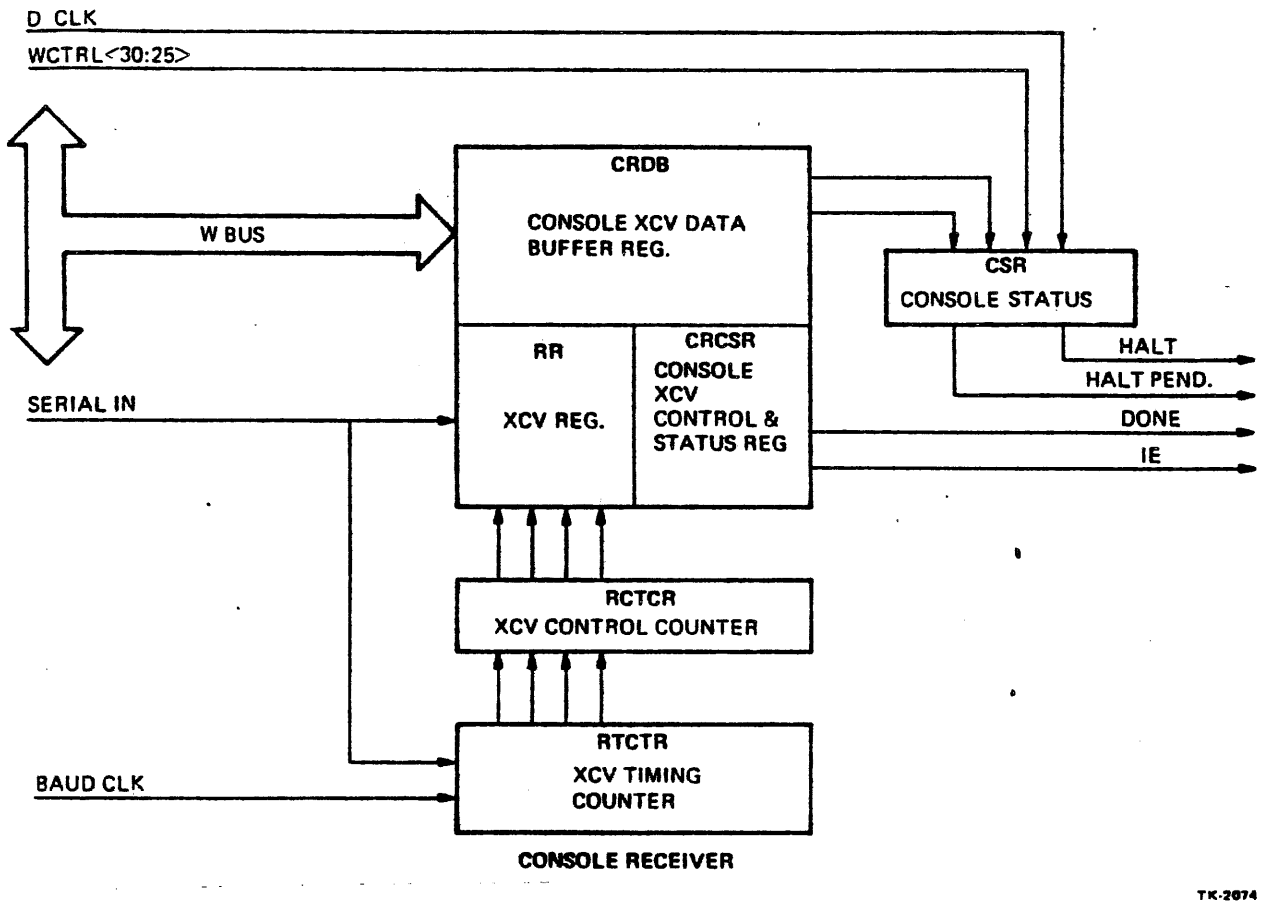


Figure 17-5 Console Receiver

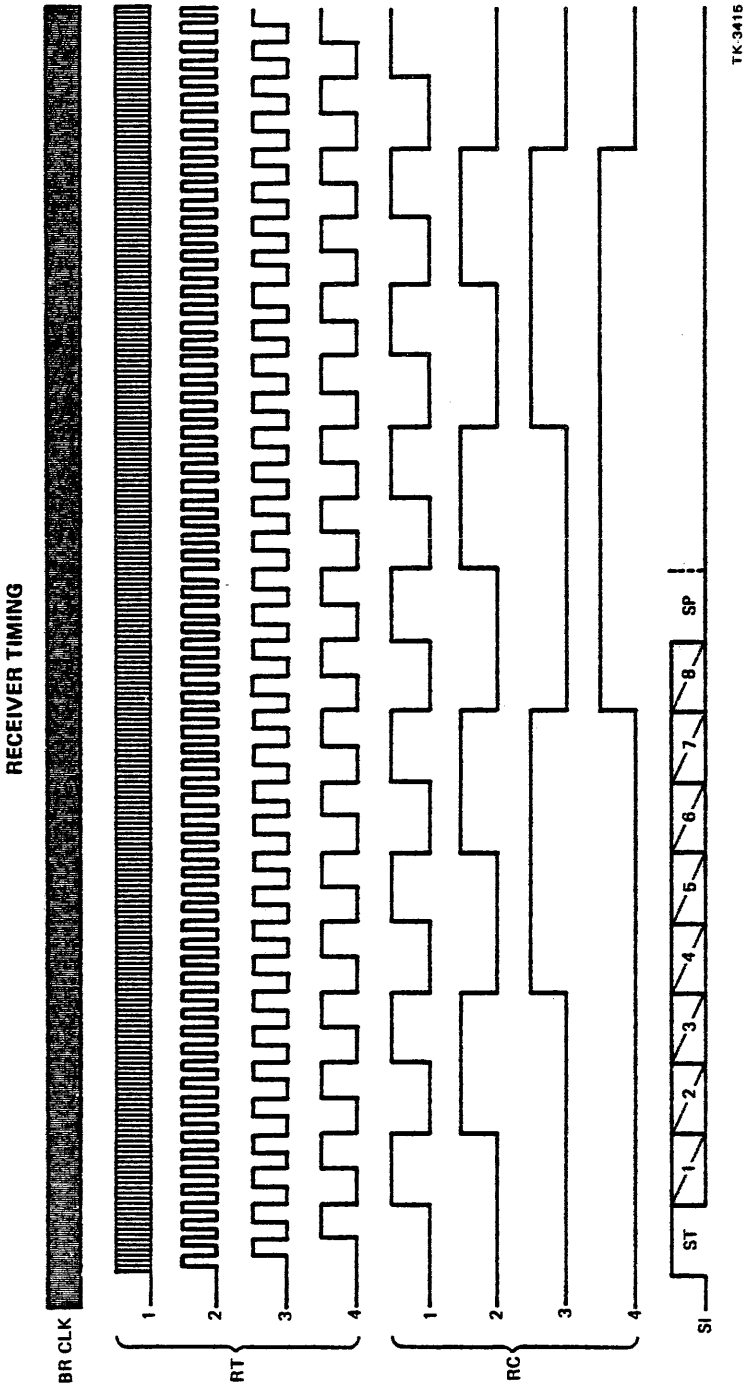


Figure 17-6 Receiver Timing

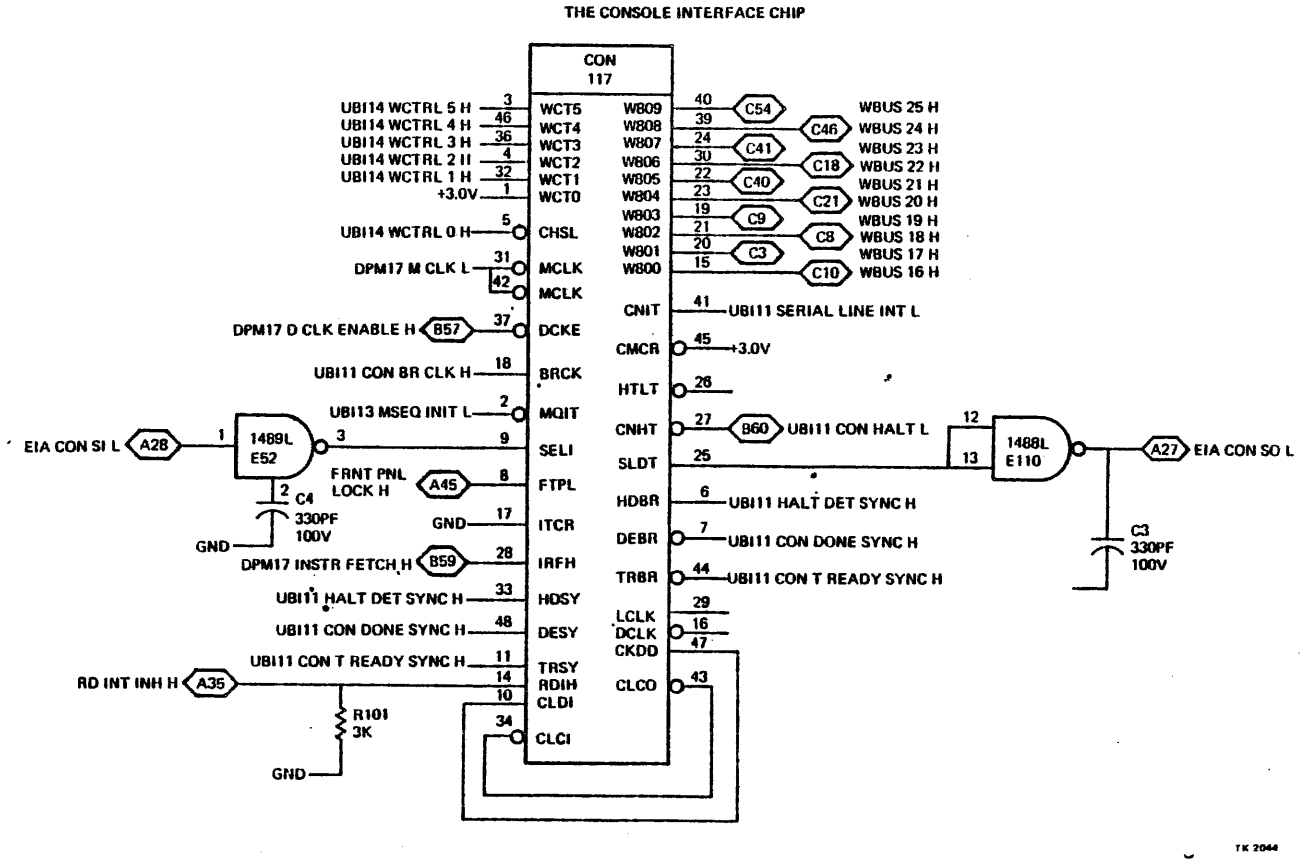


Figure 17-7 Console Interface Chip

VAX-11/750 LEVEL II

TU58 Interface

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The TU58 interface chip is an asynchronous serial line interface between the TU58 tape unit and the CPU. With very few exceptions, it is identical to the Console Interface.

The TU58 mag tape unit is a low cost mass memory device with random access, block formatted, pocket size cartridge media.

SYNPOSIS

This module is designed to be a block diagram and chip level presentation on the following:

a) Transmit parallel to serial converter
b) Receive serial to parallel converter
c) Microroutines

OBJECTIVES

Given a block diagram of the TU58 Interface, correctly identify each block by labeling it.

Provided with a list of TU58 Interface functions and signals, correctly match the signal to its function.

SAMPLE TEST ITEM

Utilizing the UBI print set, match the TU58 Interface signals listed below to their function.

a).	TU SO L	Data out
b).	M CLK L	Latch data in
c).	TU SI L	Data in
d).	MSEQ INIT L	Initialize chip and clear registers

RESOURCES

Comet Specifications
Comet Print Set

MODULE OUTLINE

XVIII. TU58/INTERFACE

- A. CON Chip
- B. TU58
  1. Specifications
  2. Component Identification, Mechanical
  3. Component Identification, Electrical
  4. Block Diagram
  5. Registers
  6. Register Addresses
  7. Data Format
- C. Summary

SELECTED TU58 SPECIFICATIONS

CARTRIDGE CAPACITY	262,144 BYTES 512 BLOCKS OF 512 BYTES EA.
TRACK FORMAT	140. FEET PER CART. .150 IN WIDE 2 TRACKS, EACH CONTAINING 1024 NUMBERED RECORDS 4 RECORDS=1-512 BYTE BLOCK
BIT DENSITY	800 BPI
READ/WRITE TAPE SPEED	30 IPS
SEARCH TAPE SPEED	60 IPS
AVERAGE ACCESS TIME	9.3 SEC.
MAXIMUM ACCESS TIME	28 SEC.
DATA TRANSFER RATE	41.7 US/DATA BIT 24KBPS
INTERFACE BUFFERING	150 TO 38.4K BAUD, JUMPER SELECTED
DRIVES PER CONTROLLER	1 OR 2 ONLY ONE CAN OPERATE AT A TIME.

TK-2049

Figure 18-1 Selected TU58 Specifications

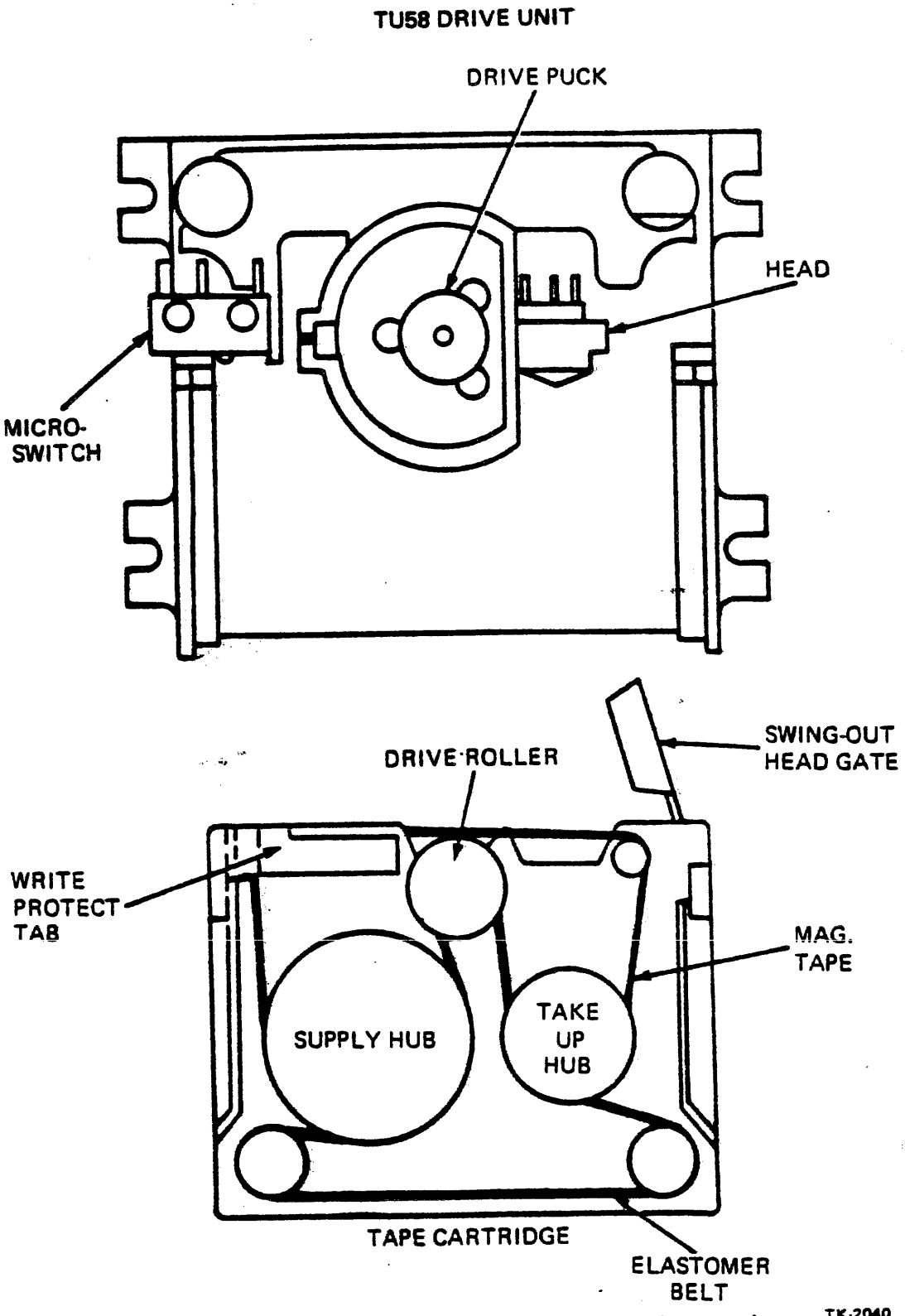
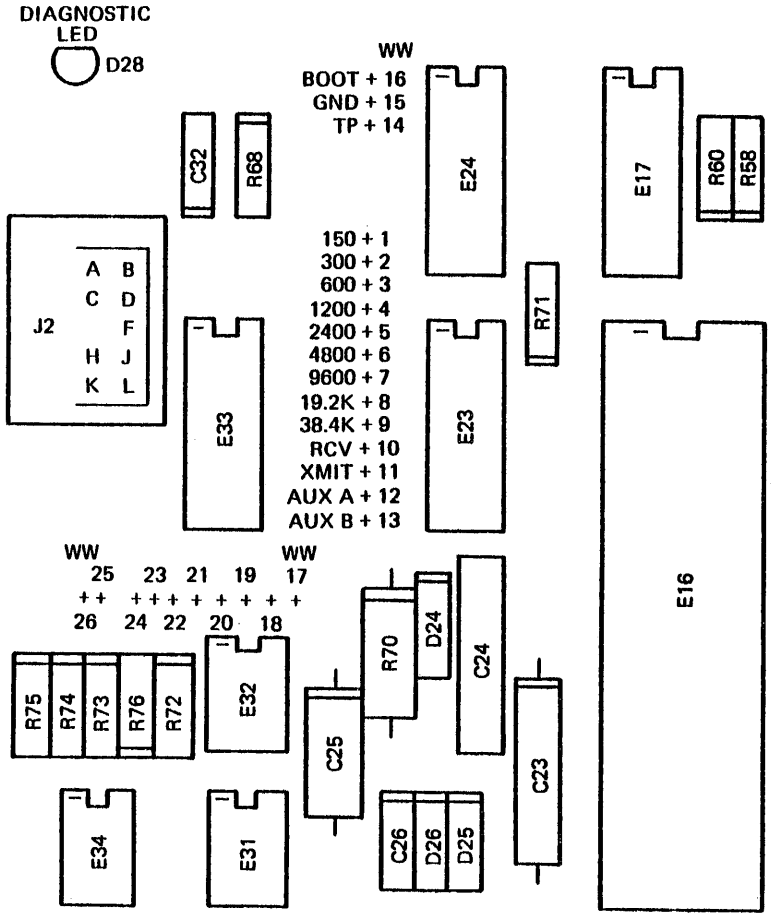


Figure 18-2 Tape Cartridge and Drive Unit



WW1	150	BAUD
WW2	300	BAUD
WW3	600	BAUD
WW4	1200	BAUD
WW5	2400	BAUD
WW6	4800	BAUD
WW7	9600	BAUD
WW8	19200	BAUD
WW9	38400	BAUD
WW10	UART RECEIVE CLOCK	
WW11	UART TRANSMIT CLOCK	
WW12	AUXILIARY A	
WW13	AUXILIARY B	
WW14	FACTORY TEST POINT	
WW15	GROUND	
WW16	BOOT	
WW17	RS-423 DRIVER	
WW18	RS-423 COMMON (GROUND)	
WW19	TRANSMIT LINE +	
WW20	TRANSMIT LINE –	
WW21	RS-422 DRIVER +	
WW22	RS-422 DRIVER –	
WW23	RECEIVER SERIES RESISTOR	
WW24	(JUMP FOR RS-422)	

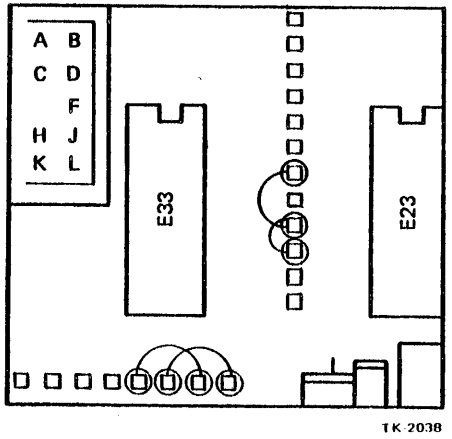


Figure 18-3 Interface Selection Jumper Locations

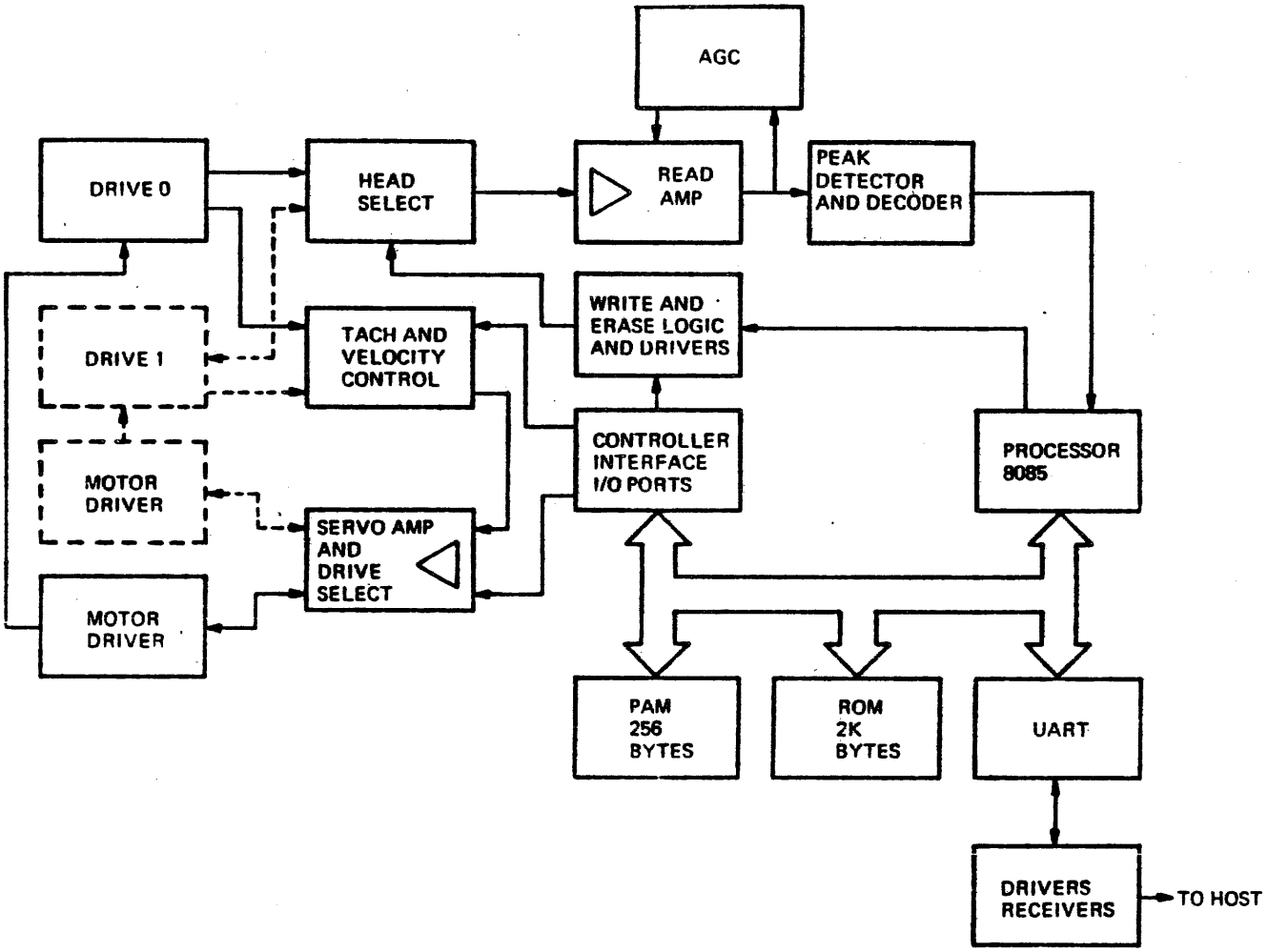


Figure 18-4 TU58 Block Diagram

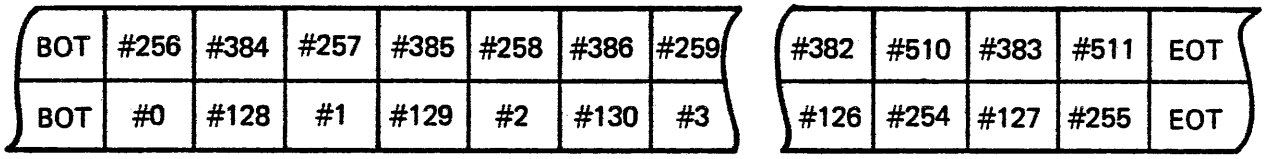


Figure 18-5 Block Locations on Tape

VAX-11/750 LEVEL II

Condition Codes

Student Guide

Course produced by Educational Services Department
of
Digital Equipment Corporation

CONDITION CODE LOGIC DESCRIPTION

The Condition Code logic in the Comet CPU is designed to set or clear the PSL N, Z, V, and C bits according to the architectural definition of each VAX 11 macro instruction and the result of the data path operation. The condition code logic also determines whether or not conditional VAX 11 branch instructions are satisfied so that the microcode can microbranch properly. A third function of CCC is to initiate all arithmetic traps. Most of the logic circuitry to perform these three functions is implemented within a gate array called CCC. CCC is 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:Ø>. The PSW lives 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 <2Ø: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. We will limit this discussion to the PSW which is 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 what is called the CC CONTROL (E14) ROM on the DPM module. This rom is not defined in the microcode listing so figure 18-1 is included in this discussion of what the rom content is for the various CC and WCTRL field functions. We should look at the CC and WCTRL fields and understand what fields are relevant to the CCC gate array. The vertical functionality of the microword is explained in a previous lesson. 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	ØØØ111
NOP.CCBR_CSIGN	Ø1	ØØØ11Ø
WB_ATCR.CCBR_SIGND	ØØ	ØØØ111
ALUS_DSDC.CCBR_ALUS	ØØ	ØØØ11Ø
ALUS_SIGND.CCBR_ALUS	11	ØØØ11Ø
ALUS_UNSGN.CCBR_ALUS	1Ø	ØØØ11Ø
SETV.CCBR_SIGND	Ø1	ØØØ111

Notice that 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 below 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 really variations on the WCTRL microorders that are not defined as WCTRL functions. You'll notice that 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 defined in the CCPSL or CCMISC micro-order of the microword. 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...

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 particulary useful for microbranching on the result of ALU operations or WBUS data. The CCBR bits can 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 VAX11 macro conditional branch instructions. Basically 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 macro instruction would assert CCBR <0> if the PSL Z bit was clear during the execution. There is a

most useful chart in the microcode listing called BUT/CCBR. Locate this chart in the CHARTS.MIC file of the microcode listing. This chart defines the CCBR bits <1:Ø> for each of the CCMISC, CCPSL and CC micro-orders. The CCBR bits <1:Ø> are generated in the CCC gate array under control the redefined CC and WCTRL fields.

The CC field of the microword also can effect the CCBR bits <1:Ø> as shown in the chart. The CC field also has the 2 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=Ø
        NOP.CCBR_SIGND=Ø,
        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 effect 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:Ø> 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 macro instruction set to set the condition codes. The CCOP2 micro-order is used to set the condition codes for the remainder of the macro instruction set. Locate the VAX NATIVE MODE CONDITION CODE CHART in the charts microcode file CHARTS.MIC. This chart indicates which CC micro-order must be specified for a particular macro instruction in the far right column. The 4 columns across the page describe how each PSL condition code bit is affected when the CCOP1 or the CCOP2 micro-order is specified.

To understand how this works we will trace the microcode executed for a VAX 11 macro instruction and see how the condition codes are set. Before we can do this we should review 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 VAX 11 macroinstruction being executed. The VAX 11 macro instruction that we will trace is.

```
ADDL2  RØ, R1     ; Where RØ is 7FFFFFFF and R1
                  ; is equal to ØØØØØØØ1
```

Remember how to read the IRD ROMS? We will get to that momentarily. First of all, what type of instruction is this? Well this happens to be an INTEGER add so we should find the microcode for this VAX 11 macro instruction in the INTLOG.MIC file of the microcode listing. What we are

looking for is the D Size rom macros which 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 CØ. The D Size ROM macro should appear as below...

```
ØDØ:   SIZE [LONG] [LONG] [ Ø] [ Ø] [ Ø] [ Ø]    ;ADDL2
```

The way one reads this macro is starting from the left column, the number ØDØ 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. The reason there is six locations is because the VAX 11 macro instructions can have up to six operand specifiers that must program the size of the data path during each execution phase. In the ADDL2 macro instruction, there are only 2 operands, so the D size ROM must be blasted with data size for 1st and 2nd operand specifier evaluations. The SIZE of the data path for each operand specifier evalution is contained within the []. As you can see 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 Ø in unused locations. The ADDL2 instruction contains the size [LONG] in the first and second operand specifier evaluations. Refer to the DEFIN.MIC file for the D size rom definition and you'll find that the data size definitions are as follows...

```
IF DSIZE = [BYTE] THEN DSIZE <1:Ø> = Ø
IF DSIZE = [WORD] THEN DSIZE <1:Ø> = 1
IF DSIZE = [LONG] THEN DSIZE <1:Ø> = 2
IF DSIZE = [QUAD] THEN DSIZE <1:Ø> = 3
```

So the D Size rom would be blasted with a 2 bit binary size code for every execution phase of the macro instruction. It is important to remember that the D Size rom output is used only if the DTYPE field of the microword specifies IDEP (data size is instruction dependent). The D Size bits <1:Ø> go to the CCC gate array so that the PSL condition codes are properly set according to the data size of the VAX 11 macroinstruction. Now that we know how to read the D size ROM, let's trace the ADDL2 macro instruction through the microcode. We have to refer to the IRD1 and IRDx rom macros located at the end of the INTLOG.MIC file. The IRD1 and IRDx rom macros appear as below...

```
.ICODE :        OPS                          OPS
    ØCØ: FPD    [NOP][IE.OPCOD.DEC]          [NOP][IE.OPCOD.DEC]
         IRD1   [LOD][OS.RED]                [LOD][OS.RED]
```

This is the IRD1 rom macro definition for ADDL2. The IRD1 ROM is addressed by the opcode of the instruction to be executed and the FPD and the signal FPA PRESENT.This means the macro instruction opcode provides the base target

address in the rom of which there are 4 locations. That is what this macro is used for. It 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 IRD1 of an ADDL2 instruction because it is not interruptable. If it is set, then the machine will vector to location SCBB+1Ø and execute the RESERVED to DEC opcode instruction fault service routine. FPA PRESENT is a signal that is used to change the flow if an FPA is present or not. You'll note the IRD1 rom macro has 2 targets across the page. 1 with FPA and 1 without FPA. The OPS bit is used to load the OSR at IRD1 and IRDx. The IRD1 rom macro could be changed to show how the rom is addressed as follows.

```
ØDØ:    FPD     NOT FPA        FPA
     NOT FPD     NOT FPA        FPA
```

What this shows us is at base IRD1 rom address ØDØ, the 4 locations that are blasted are all the possible combinations of FPD and FPA PRESENT. The contents of the [] is the label of a microroutine that is entered for each of the 4 possible combinations. In the case that we are using, an ADDL2 does not use the FPA, FPD should be clear, and both the source and destination operands are in registers. For this discussion we will assume 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. We will see this in a little bit. Lets look at the IRDx rom macro, This is similiar to the IRD1 macro except that the IRD COUNTER output addresses these ROMS.

```
.OCODE          OPS       REG                        MEM
    ØCØ: CNTØ [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. This means there are eight possible targets at IRDx (CNTØ has 4 combinations and so does CNT 1). CNTØ 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 CNTØ 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 really meaningless. If the destination was 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 RØ, R1, the microcode goes to the following two rom addresses.

```
IRD1                        [OS.RED]
IRDX                        [IL.ADD2.B.W.L.REG]
```

With this knowledge we can trace the microinstructions. They are reproduced below from the OSR.MIC and INTLOG.MIC files respectively...

```
1ØØ:
OS.RED:
      ;ØØØØ------------------      ;Rn        REGISTER MODE
      FPA_Q_M[MDR] MDR_R[GPR.R],    ; PLACE OP (GPR(RNUM))
                                     IN MDR
      CLOBBER MTEMPØ DEF, IRDX [1]  ; SAVE MDR IN Q
```

This moves the source operand from RØ 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:                          ; 8Ø AØ CØ
       ;-----------------------------------;
       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 <RØ> is added to the destination GPR <R1>, which is selected by RNUM, and that 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 ØØØØØØØ1 is 8ØØØØØØØ. This is an integer overflow and 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:Ø>=Ø	ALU<31>V	ALU<31>CO
1	Ø	1	Ø

2.9.2 VAX 11 BRANCH INSTRUCTION IMPLEMENTATION

The CCC gate array also is used to decide if a VAX 11 macro

branch instruction is satisfied, so that one of two things can happen. If the branch condition is NOT satisfied, the hardware must bump the PC to the next sequential instruction and do the IRD1. 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. We are going to trace a VAX 11 macro branch instruction called BNEQ. This macro instruction branches if the PSLZ bit is clear. We will show both paths that are followed. The BNEQ instruction is located in the CONTRL.MIC file. First lets look at the IRD1 rom macro for a BNEQ in the back of the CONTRL.MIC file. The Macro appears below.

```
.ICODE          OPS             REG                 OPS    FPA REG

     Ø12: FPD [NOP][IE.OPCOD.DEC]                   [NOP][IE.OPCOD.DEC]
          IRD1[LOD][CO.BRCND]                       [LOD][CO.BRCND]

.OCODE
     Ø12: CNTØ[NOP][IE.BAD.IRD]                     [NOP][IE.BAD.IRD]
          CNT1[NOP][IE.BAD.IRD]                     [NOP][IE.BAD.IRD]
```

The IRD1 macro specifies that the address of the BNEQ microcode is CO.BRCND which is the target address for all the conditional branch instructions. Notice that this instruction will NOT do an IRDx and that the address for a fault is [IE.BAD.IRD] which initiates a Machine Check Exception. The microcode sequence for the BNEQ is shown below.

```
=1ØØØ
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. Locate the this macro in the MACRO.MIC file and verify that this is true. This micro instruction has two possible destinations. If the PSL Z bit is set, the microcode will read the microinstruction at CO.BRCND-DECIDE. If the PSL Z bit is clear, the microcode will execute 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

```
=Ø
CO.NOP:
```

```
CO.BRCND-DECIDE:
;----------------------------; NO BRANCH IF CONTROL COMES
IRD1                         ; HERE,GO DO NEXT INSTRUCTION
```

Simply do IRD1 and execute the next sequential instruction. If the PSL Z bit is clear, the CCBR bits <1:Ø> are equal Ø1, according to the to the CCPSL micro-order at location CO.BRCND. The following microinstructions are executed

```
CO.BRCND-BRANCH:
;1---------------------------; BRANCH IF CONTROL COMES
PC PC+SEXT(M[MDR]),          ; HERE, CALCULATE NEW PC
SIZE [IDEP], NEXT/CO.NOP     ; WASTE CYCLE TO LET PC CATCH
                             ; UP
```

The PC gets the sign extended MDR if the branch condition is satisfied and the data size of the value written into the PC is determined by the D Size ROM. 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 IRD1, the micromachine is stalled until the XB is filled. The next microinstruction is at CO.NOP which IRD1 and it is shown above.

We have seen two of the major functions of the CCC gate array and how the microcode implements setting the condition codes and branching macro instructions. The CCC gate array also generates 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 up by a VAX 11 macro routine. The functions of these bits briefly is described below.

PSW <7>	Decimal Overflow TRAP enable.
PSW <6>	Floating Underflow TRAP enable.
PSW <5>	Integer Overflow TRAP enable.

If an arithmetic operation is performed that causes one of the TRAP conditions, the CCC gate array will assert 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 should be entered. The type of arithmetic trap is logged the Arithmetic Trap Code Register (ATCR) which is also contained in the CCC gate array. The Arithmetic trap results in aborting the next macro instruction and performing the trap service from SCBB+3Ø. The trap microcode pushes the PSL, PC of the NEXT instruction, and the ATCR on the stack.

This completes the discussion of the microcode implementation.

2.9.3 HARDWARE IMPLEMENTATION OF CONDITION CODE LOGIC

The actual hardware that implements the condition code logic is a small percentage of the total. The condition code logic is on the DPM module and we'll refer to the print set. Refer to DPM2Ø. At the beginning of this discussion we stated that the CCC gate array is controlled by 4 bit field called CC CTRL <3:Ø>. This field comes from the output of the rom E14 on DPM2Ø. The address input to this ROM is the CC and WCTRL fields of the microword that is latched on DPM2Ø and DPM12. The output is called CC CTRL <3:Ø> H and these 4 signals go to the CCC gate array shown on DPM1Ø. Figure 18-1 shows how the CC CTRL lines and the GOOD SAMARITAN ROM are programmed for various combinations of the WCTRL and CC fields. The reason the signal LIT Ø H is present is because if the LIT field is 1 or 3, the CC field is not interpreted and becomes part of the short or long literal. On DPM1Ø again let's look at the CCC gate array inputs. The lines CC CTRL <3:Ø> are the control input to the gate array. The VAX 11 or compatibilty macro instruction opcode is latched in E12 and is the input to combinational logic that sets the PSL condition codes according to the architectural defintions and data path results. The D Size bits <1:Ø> 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:Ø>, the same is true about the source of the C bit and V bit. The C and V bits would be selected as a function of data size also. 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 interface to the WBUS is a bi-directional interface that essentially 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 mux on DPM16 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:Ø> go to the BUT mux on DPM15 and 16 for microbranching on their state.

The functionality of the CCC gate array is tested with the microdiagnostics and indirectly with the macro diagnostics. Figure 18-1 is included here to show the programming of the CC CTRL rom and the GOOD SAMARITAN rom which are not blasted by the microprogrammers.

This concludes the discussion of the Condition Code Logic.

GOOD SAMARITAN ENCODING

GOOD SAMARITAN INPUTS				GOOD SAMARITAN OUTPUTS
WCTRL Function	WCTRL	CC	LIT Ø H	CC CTRL <3:Ø> G.S. <7:4>
WRITE PSL	ØØ	X	X	9
WRITE PSW	Ø1	X	X	B
READ PSL	Ø4	X	X	3
WRITE CC	Ø5	X	X	A
CC MISC 1	Ø6	Ø	Ø	5
CC MISC 1	Ø6	1	Ø	8
CC MISC 1	Ø6	2	Ø	7
CC MISC 1	Ø6	3	Ø	6
CC MISC 2	Ø7	Ø	Ø	2
CC MISC 2	Ø7	1	Ø	F
CC MISC 2	Ø7	2	Ø	Ø
CC MISC 2	Ø7	3	Ø	1
CC MISC 2	Ø7	X	1	Ø
Any other WCRTL Function		Ø	Ø	Ø
Any other WCTRL Function		1	Ø	C
Any other WCTRL Function		2	Ø	E
Any other WCTRL Function		X	1	Ø

Figure 18-1

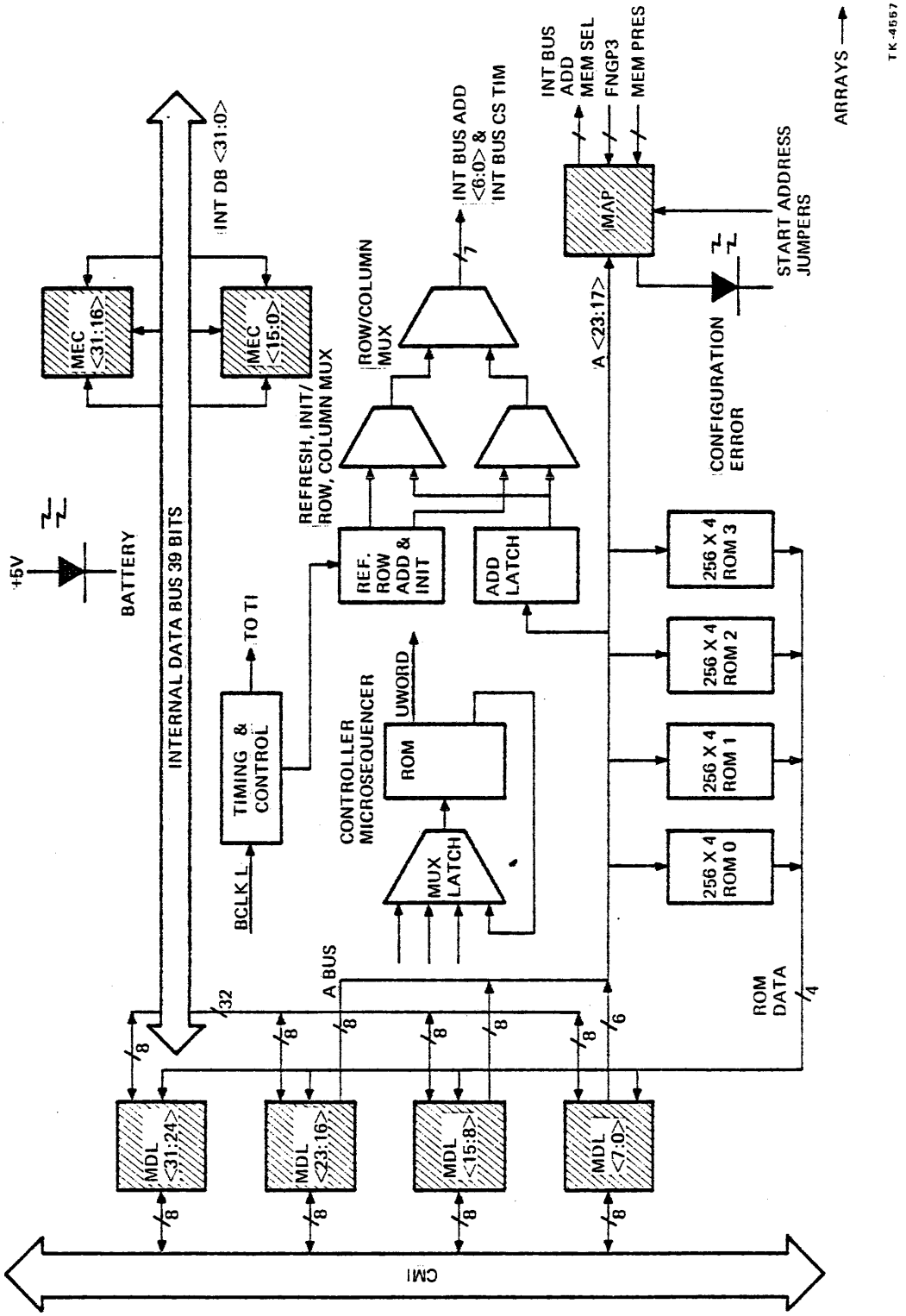
This concludes the discussion of the Condition Code logic.

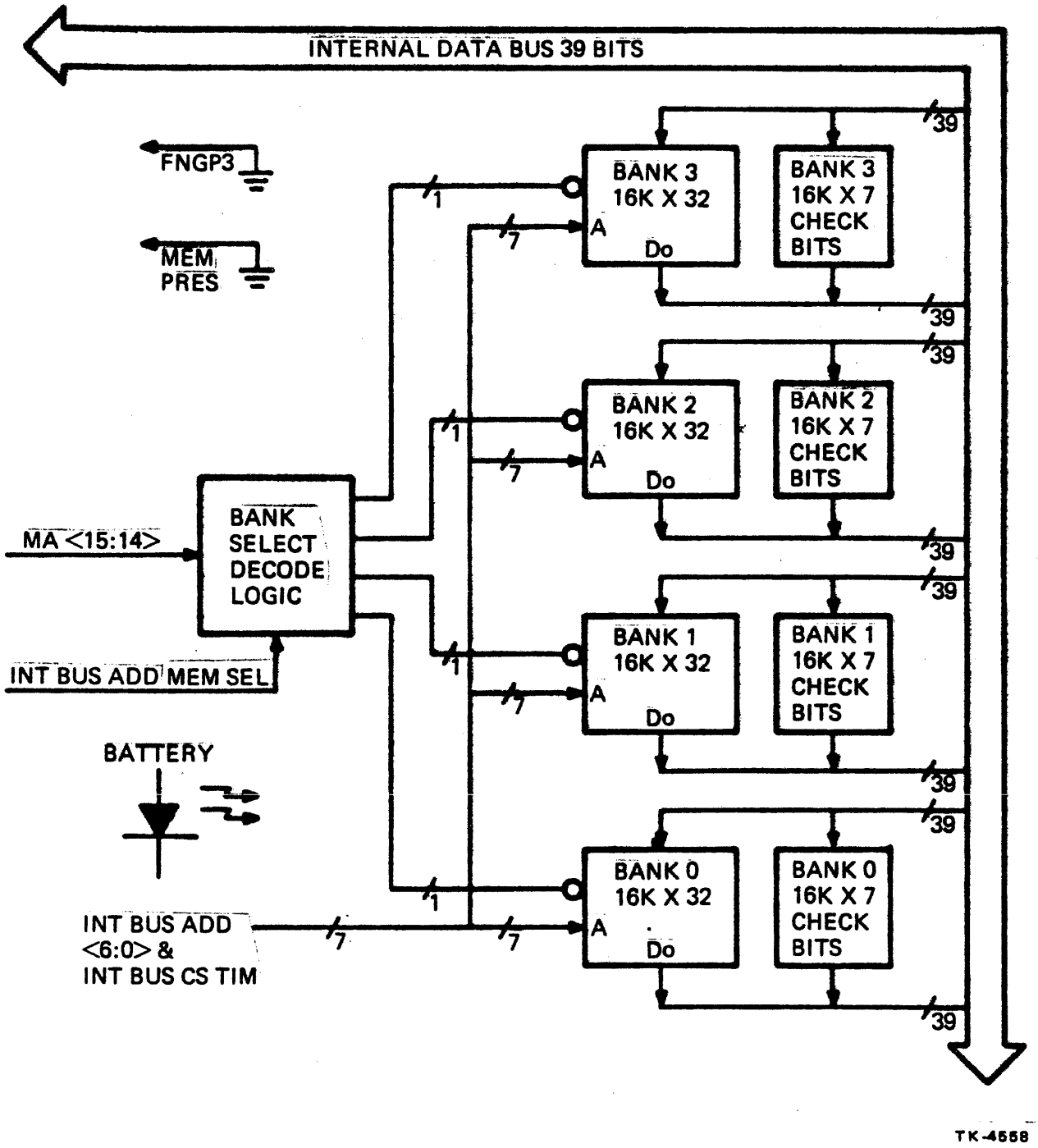
VAX-11/750 LEVEL II

Memory Subsystem

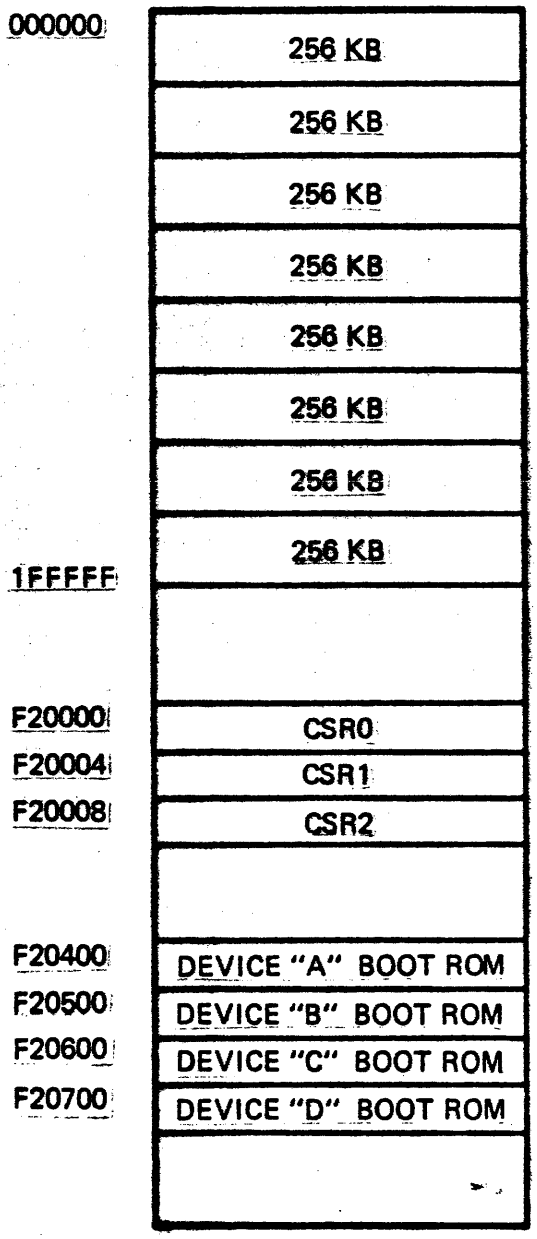
Student Workbook

Course produced by Educational Services Department
of
Digital Equipment Corporation

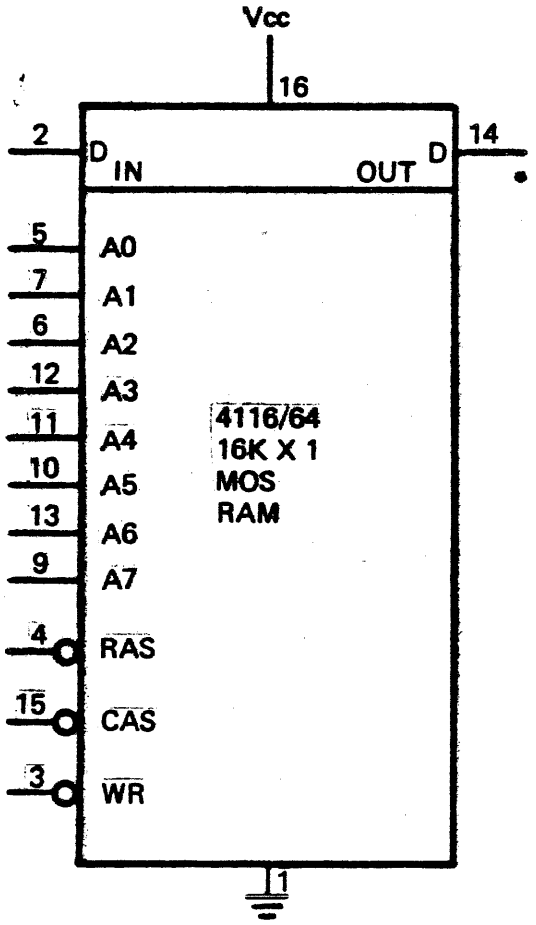

INT DB <31:0>
INTERNAL DATA BUS 39 BITS
MEC <31:16>
MEC <15:0>
+5V
BATTERY
TIMING & CONTROL
TO TI
BCLK L
REFRESH, INIT/ ROW, COLUMN MUX
REF. ROW ADD & INIT
ADD LATCH
ROW/COLUMN MUX
INT BUS ADD <6:0> & INT BUS CS TIM
INT BUS ADD MEM SEL FNGP3 MEM PRES
MAP
A <23:17>
START ADDRESS JUMPERS
CONFIGURATION ERROR
ARRAYS
TK-4567
CONTROLLER MICROSEQUENCER
ROM
UWORD
MUX LATCH
256 X 4 ROM 3
256 X 4 ROM 2
256 X 4 ROM 1
256 X 4 ROM 0
ROM DATA
A BUS
32
8
6
4
MDL <31:24>
MDL <23:16>
MDL <15:8>
MDL <7:0>
CMI

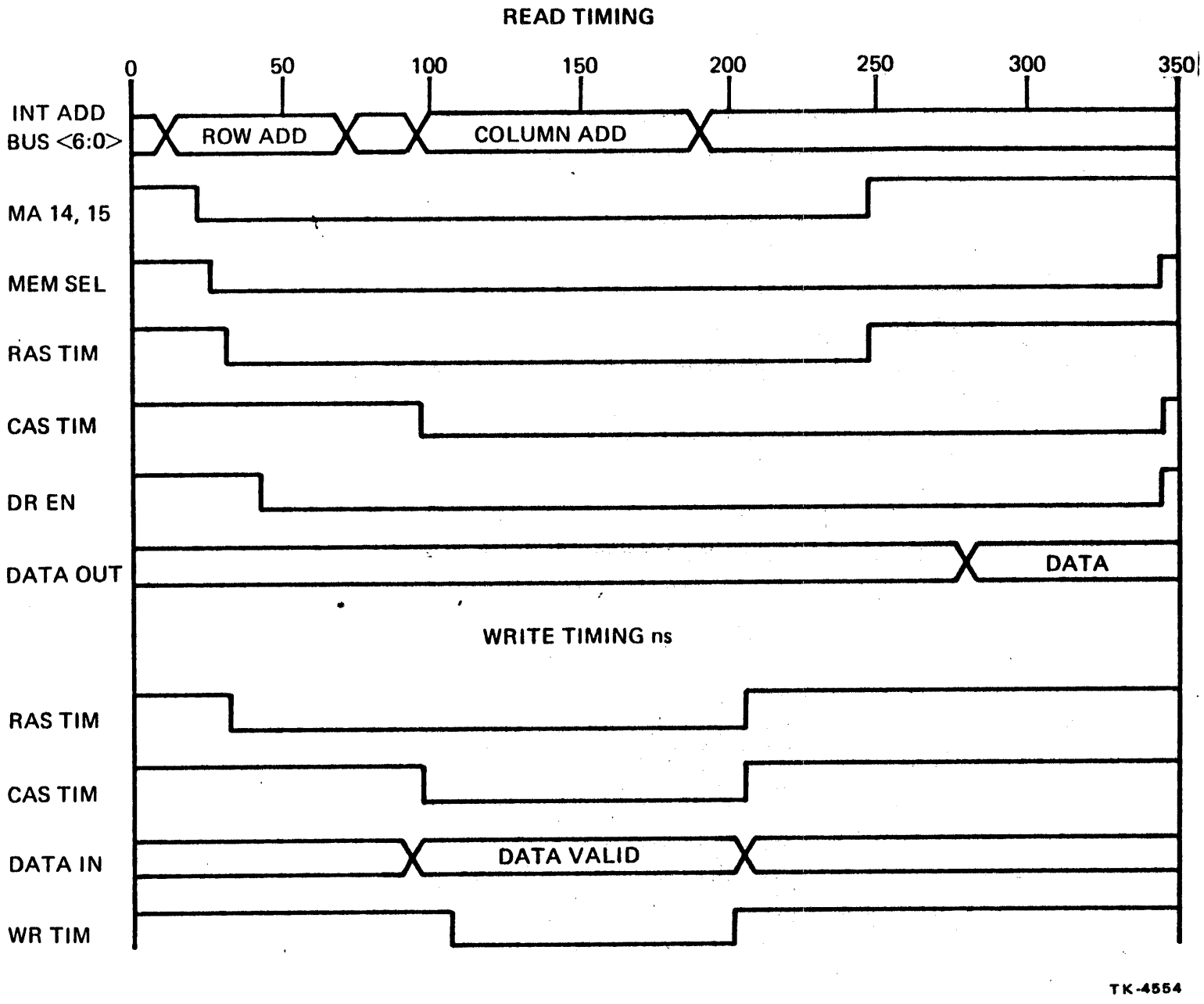

INTERNAL DATA BUS 39 BITS
FNGP3
MEM PRES
MA <15:14>
BANK SELECT DECODE LOGIC
INT BUS ADD MEM SEL
BATTERY
INT BUS ADD <6:0> & INT BUS CS TIM
BANK 3 16K X 32
BANK 3 16K X 7 CHECK BITS
BANK 2 16K X 32
BANK 2 16K X 7 CHECK BITS
BANK 1 16K X 32
BANK 1 16K X 7 CHECK BITS
BANK 0 16K X 32
BANK 0 16K X 7 CHECK BITS
A
Do
1
7
39
TK-4658

PHYSICAL MEMORY MAP



TK-4560

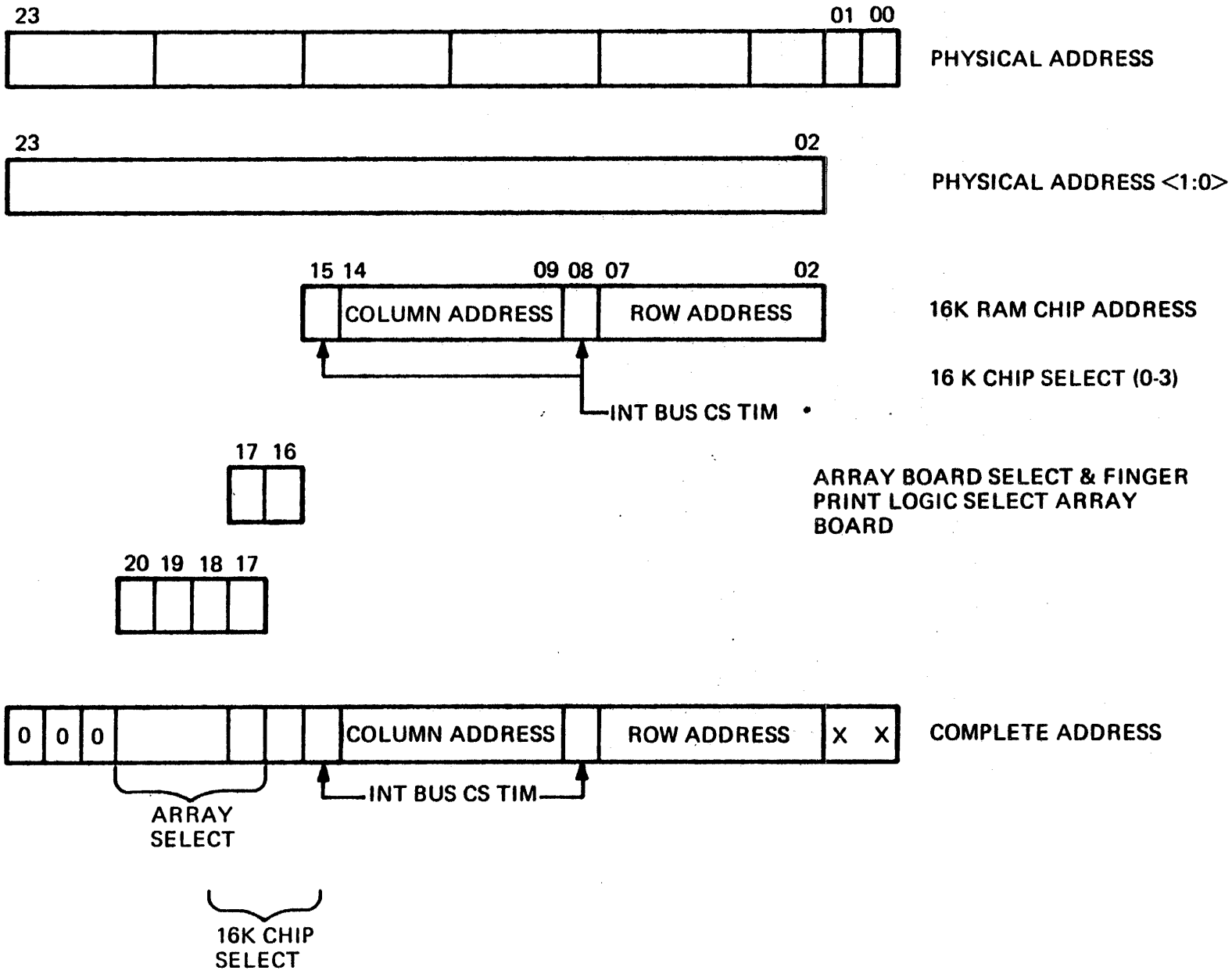

Vcc
16
2
D IN
OUT D
14
5
A0
7
A1
6
A2
12
A3
11
A4
10
A5
13
A6
9
A7
4
RAS
15
CAS
3
WR
1
4116/64
16K X 1
MOS
RAM
TK-4559


READ TIMING
0
50
100
150
200
250
300
350
INT ADD BUS <6:0>
ROW ADD
COLUMN ADD
MA 14, 15
MEM SEL
RAS TIM
CAS TIM
DR EN
DATA OUT
DATA
WRITE TIMING ns
RAS TIM
CAS TIM
DATA IN
DATA VALID
WR TIM
TK-4554

Comet Memory Microprogram Functionality

	Function	Purpose	Entry Address
1.	IDLE	NOP	0000
2.	CSR 1 WR	WRITE TO CSR 1 (F20004)	0001
3.	CSR 0 WR	WRITE TO CSR 0 (F20000)	0010
4.	CSR RD	READ CSR 0, 1, 2	0011
5.	ROM RD	ASSEMBLE BOOT ROM LONGWORD	0100
6.	UNUSED		0101
7.	INIT	WRITE ALL ZEROS	0110
8.	UNUSED		0111
9.	MEM 4 BYTE WR	LONGWORD WRITE, MASK=1111 (ECC)	1000
10.	MEM 4 BYTE WR	LONGWORD WRITE, MASK=1111 (ECC OFF)	1001
11.	REFRESH	REFRESH DYNAMIC RAMS	1010
12.	BYTE WRITE	ANY WRITE OTHER THAN LONG	1011
13.	MEM READ DIAG PG	DIAGNOSTIC PAGE MODE	1100
14.	MEM READ DIAG PG	DIAGNOSTIC PAGE MODE	1101
15.	MEMORY READ	READS A LOCATION	1110
16.	MEMORY READ (ECC)	READS A LOCATION (ECC OFF)	1111

Figure 20-6


23
01 00
PHYSICAL ADDRESS
23
02
PHYSICAL ADDRESS <1:0>
15 14
09 08 07
02
COLUMN ADDRESS
ROW ADDRESS
16K RAM CHIP ADDRESS
16 K CHIP SELECT (0-3)
INT BUS CS TIM
17 16
ARRAY BOARD SELECT & FINGER PRINT LOGIC SELECT ARRAY BOARD
20 19 18 17
0 0 0
COLUMN ADDRESS
ROW ADDRESS
X X
COMPLETE ADDRESS
INT BUS CS TIM
ARRAY SELECT
16K CHIP SELECT

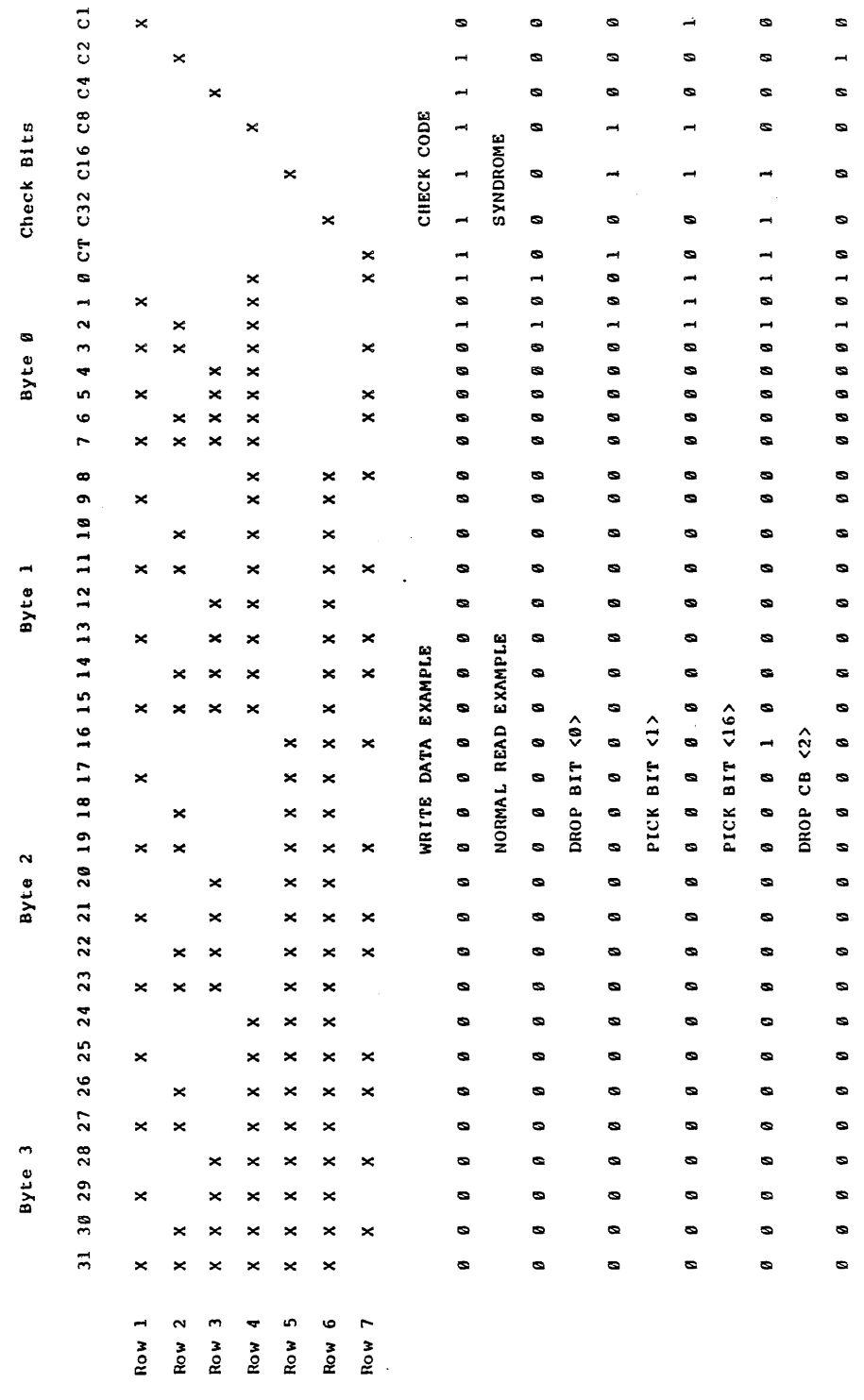
TK-4555

Check Bit Code Generation Algorithm
(Refer to Check Bit Generation Chart)

Desired Parity

EVEN	C1 = 1,	IF THERE IS AN ODD NUMBER OF ONES IN ROW 1
EVEN	C2 = 1,	IF THERE IS AN ODD NUMBER OF ONES IN ROW 2
ODD	C4 = 1,	IF THERE IS AN EVEN NUMBER OF ONES IN ROW 3
ODD	C8 = 1,	IF THERE IS AN EVEN NUMBER OF ONES IN ROW 4
ODD	C16 = 1,	IF THERE IS AN EVEN NUMBER OF ONES IN ROW 5
ODD	C32 = 1,	IF THERE IS AN EVEN NUMBER OF ONES IN ROW 6
EVEN	CT = 1,	IF THERE IS AN ODD NUMBER OF ONES IN ROW 7

Figure 20-8



	Byte 3							Byte 2							Byte 1							Byte 0							Check Bits						
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CT	C32	C16	C8	C4	C2	C1
Row 1	X		X		X		X		X		X		X		X		X		X		X		X		X		X		X						X
Row 2	X	X			X	X			X	X			X	X			X	X			X	X			X	X			X	X					X		
Row 3	X	X	X	X					X	X	X	X					X	X	X	X					X	X	X	X								X	
Row 4	X	X	X	X	X	X	X	X									X	X	X	X	X	X	X	X	X	X	X	X	X	X					X		
Row 5	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X																		X			
Row 6	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X									X				
Row 7		X		X		X	X			X	X		X			X		X	X		X			X		X	X		X	X						
	WRITE DATA EXAMPLE																																CHECK CODE						
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	0
	NORMAL READ EXAMPLE																																SYNDROME						
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
	DROP BIT <0>																																						
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0
	PICK BIT <1>																																						
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1
	PICK BIT <16>																																						
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
	DROP CB <2>																																						
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0

Figure 20-9 Memory Check Bit and Syndrome Generation Chart

Memory System Single Bit Error Syndrome Chart

Bit Position In Error	CSR 0 <6:0> Syndrome	Hex Syndrome
0	1011000	58
1	0011001	19
2	0011010	1A
3	1011011	5B
4	0011100	1C
5	1011101	5D
6	1011110	5E
7	0011111	1F
8	1101000	68
9	0101001	29
10	0101010	2A
11	1101011	6B
12	0101100	2C
13	1101101	6D
14	1101110	6E
15	0101111	2F
16	1110000	70
17	0110001	31
18	0110010	32
19	1110011	73
20	0110100	34
21	1110101	75
22	1110110	76
23	0110111	77
24	0111000	38
25	1111001	79
26	1111010	7A
27	0111011	3B
28	1111100	7C
29	0111101	3D
30	0111110	3E
31	1111111	7F

Figure 20-10

MEMORY SYSTEM SINGLE BIT ERROR SYNDROME CHART

24	0111000	38
25	1111001	79
26	1111010	7A
27	0111011	3B
28	1111100	7C
29	0111101	3D
30	0111110	3E
31	1111111	7F

Figure 20-11

VAX-11/750 LEVEL II

Writable Control Store

Student Workbook

Course produced by Educational Services Department
of
Digital Equipment Corporation

INTRODUCTION

The Writable Control Store (WCS) is an optional extension of the control store that permits the customer to design his own microroutines or macro instructions that use WCS microcode. The WCS interfaces to the CMI for data input and output.

The WCS RAM data is connected to the control store bus lines and is enabled when control store address bit 13 is asserted. The WCS module is a daughter board connected to the motherboard module via 4 signal connectors, the mother board is the 6K x8Ø main control store module located within the CPU.

OBJECTIVES

From a list of statements concerning the WCS, identify each statement as true or false.

Provided with the laboratory procedure, write a program to transfer 1Ø Comet CPU microinstructions from main memory to WCS.

SAMPLE TEST ITEM

Identify the following statements as true or false.

a) In order to store one microinstruction in WCS, 8 CMI bus cycles are required. _____.

b) The WCS address register has the processor register address of 2C hex. _____.

c) Microinstruction execution speed is slower when executed from WCS because of interfacing with CMI. _____.

d) The base address of the WCS module in the CMI memory space is FEØØØØ. _____.

RESOURCES

WRITEABLE CONTROL STORE
SCHEMATIC DIAGRAM

OUTLINE

XXI. WRITEABLE CONTROL STORE (WCS)

A. WCS Specifications and Characteristics
B. CMI Interface
C. Memory Address Allocation
D. Control Store Interface
E. Programming Examples
F. Laboratory Exercise 16
    1. Write a routine to transfer 1Ø Comet CPU microinstructions from memory to WCS
    2. WCS Fault Isolation
G. Summary

WCS Module Functionality

The WCS module is a daughter board that attaches to the main control store module in the CPU. Loading the WCS is accomplished by writing the desired WCS data on the CMI with the destination of the WCS. Data contained within the WCS may be read onto the CMI and transferred to memory for comparison with input data if so desired.

Loading one 80 bit microinstruction into WCS requires 4 CMI write cycles because the WCS rams are loaded sequentially 20 bits at a time. This means that bits 21 to 31 of the data are ignored. Bit 20 has a special function that will be discussed later.

Refer to Figure 22-1, the WCS block diagram. At the left of the drawing are CMI interface drivers to receive and transmit to and from the CMI. The received address and DBBZ are used to activate the WCS Control logic when a transfer of data to or from the WCS occurs. Basically the chip enables and write enables are generated on CMI writes to WCS so that CMI data is sequentially loaded into the rams from left to right.

The WCS rams can be addressed from two sources; either from the microsequencer (CSA<9:0>) or the CMI address latch. Naturally when loading or reading WCS, the address latch is gated through the two to one mux and becomes the ram address. The WCS ram data output has two destinations, the DPM module control store latches and an interface to the transmit side of the CMI. Some of the technical specifications that one should be familiar with are:

1. WCS ram read access time is to 70 - 90ns.

2. Module timing is derived from CPU B clock.

3. No parity bit generation or checking

To understand the functionality of the WCS CMI interface let's design an example program to load one 80 bit microinstruction into WCS address 2000, refer to figure 22-2 which is the comet physical memory organization and locate hex address F00000 on the left of the drawing, this is the I/O address for WCS. It extends from F00000 to F03FFC because 4 longwords are required to load one WCS location. Our example program will require 4 longwords of data to

build one microinstruction. The next question is, where is WCS address 2000 and how do we load WCS address 2000 from the CMI? Simple, refer to Figure 22-3, the control store memory allocation. Note that WCS address 2000 is the first location of WCS from the control store side. From the CMI side, locations F00000, F00004, F00008, and F0000C correspond to WCS address 2000. Remember 4 longword writes are required to load one WCS location. Examine the following macro code listing, Figure 22-3. Note that the microinstruction itself is irrelevant, but the code that loads it is what we want to study. This subroutine is rather useless but could be doctored up so that parameters can be passed to it. Each time the instruction on line 1700 is executed, a CMI write to WCS occurs. The address bits <1:0> are irrelevant on the CMI because of longword alignment. Bits <3:2> of the CMI address lines are used to sequentially load the WCS rams. Refer to the following table.

LOAD WCS RAM BITS	IF	CMI ADDRESS BITS 3	2
<19:0>	=	0	0
<39:20>	=	0	1
<58:40>	=	1	0
<79:60>	=	1	1

Since the program is using autoincrement addressing mode this will automatically sequence through and load 20 bits at a time into the WCS Rams, it is important to insure that the macro program set the pointer to WCS to double quad word boundaries, that is, initially the pointer must be as follows...

CMI ADDRESS BITS

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	1	1	1	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	0	0

This insures that the correct longword is loaded into the proper location in WCS. The 2nd part of the program reads the data written into WCS and compares it with the data in the table to verify that the WCS was properly loaded.

Reading and writing WCS both drive the chip enable decoder shown in the block diagram Fig. 22-1. Writes to the WCS enable the upper WCS data drivers, while reads enable the lower data drivers RAMS to the CMI. Reading the RAMS is possible provided the CPU microcode is not running out of WCS. This will be the rule rather than the exception. The microword parity bits <79:78> must be contained in the data transferred to WCS because there is no parity generator within the WCS module.

Execution of the microinstructions from WCS is similar to that of any microinstruction in the main control store. The WCS is enabled any time the next address is 2000 to 23FF Hex, refer to Fig. 22-5, Comet Control Store addressing. Bit 13, if set, indicates a WCS address. Since the WCS is only 1K, bits <12:10> are irrelevant. Bits <9:0> directly address the RAMS.

At the beginning of the discussion we mentioned bit 20 of the WCS data transmitted to WCS had special meaning. This bit sets a flip-flop called WCS PRESENT whose output goes to the microsequencer. The flop is used for a microbranch condition called WCS PRESENT. It will be used in the situation where if the macro vector bits <1:0> = 2 and there is no WCS present a console halt occurs.

To summarize, the basic function of the WCS is to provide the customer with capability of creating his own micorcode programs to enhance performance.

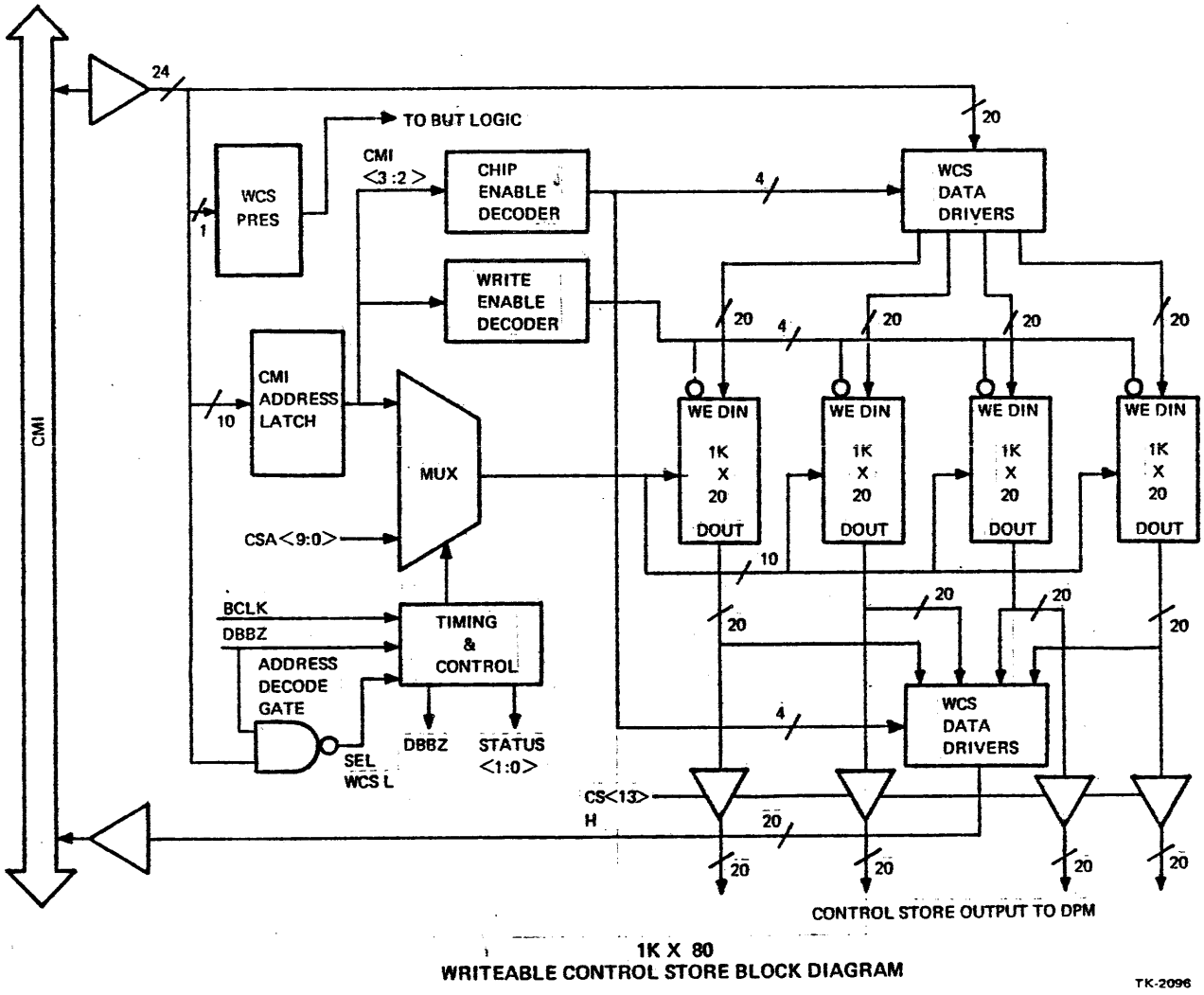
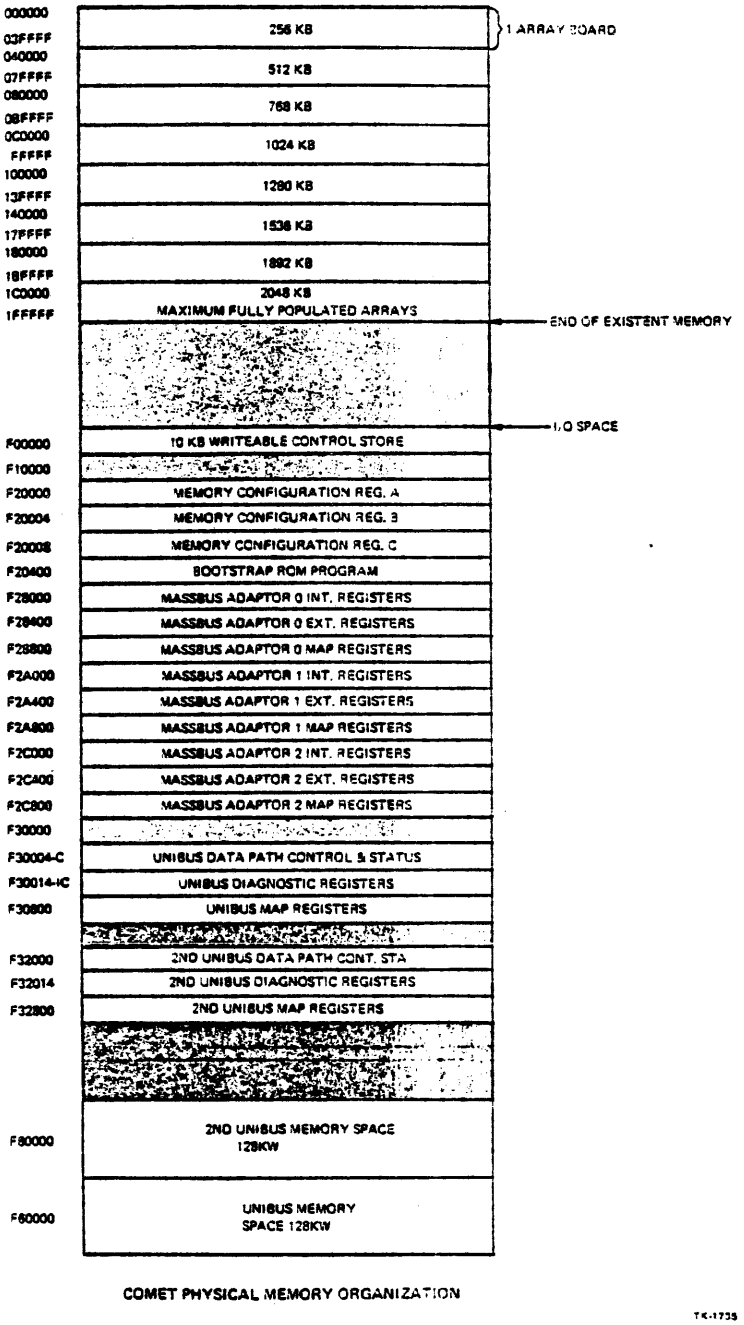


Figure 21-1. 1K X 8Ø Writable Control Store Block Diagram



COMET PHYSICAL MEMORY ORGANIZATION

TK-1735

Figure 21-2. Comet Physical Memory Organization

```
LOAD          AND VERIFY WCS                                  20-JUN-1979 07:37:59    VAX-11 MACRO V02.30                PAGE   1
                                                              20-JUN-1979 07:37:49    DMA0:[BEN]COMWCS.MAR;11                (1)

                                  0000   100                  .TITLE LOAD AND VERIFY WCS
                             00000000   200                  .PSECT ALIGN LONG
                                  0000   300
                     00012345     0000   400  TABLE::        .LONG ^X00012345        ;THIS TABLE REPRESENTS 1 80 BIT
                     00023456     0004   500                  .LONG ^X00023456        ;MICROINSTRUCTION
                     00034567     0008   600                  .LONG ^X00034567
                     00045678     000C   700                  .LONG ^X00045678        ;THIS LONGWORD MUST INCLUDE THE
                                  0010   800                                          ;PROPER PARITY IN BITS <19:18>
                                  0010   900                                          ;ACCORDING TO MICROCODE SPEC
                                  0010  1000
                     50     D4    0010  1100  INIT:           CLRL R0                 ;THIS SETS UP LOOP COUNT ETC.
51   00F00000 8F      D0    0012  1200                  MOVL #^XF00000, R1      ;F00000 IS THE BASE CMI ADDRESS
52          E4 AF     DE    0019  1300                  MOVAL TABLE, R2         ;OF WCS LOCATION 0.
                      05    001D  1400                  RSB
                            001E  1500
            EF AF     16    001E  1600  START::         JSB INIT
81          82        D0    0021  1700  1$::            MOVL (R2)+,(R1)+        ;WRITE DATA FROM TABLE TO WCS
01          03        F1    0024  1800                  ACBL #3, #1, R0, 1$     ;LOOP 4 TIMES TO WRITE 4 LONGWORDS
FFF7        50              0027
                            002A  1900
            E3 AF     16    002A  2000                  JSB INIT
82          81        D1    002D  2100  2$::            CMPL (R1)+,(R2)+        ;READ WCS AN COMPARE TO TABLE DATA
            0A        12    0030  2200                  BNEQ 3$                 ;IF ERROR, CLEAR R0 AND RETURN
01          03        F1    0032  2300                  ACBL #3, #1, R0, 2$
FFF5        50              0035
50          01        D0    0038  2400                  MOVL #1, R0             ;NO ERROR, DO A NORMAL RETURN
                      05    003B  2500                  RSB
            50        D4    003C  2600  3$::            CLRL R0
                      05    003E  2700                  RSB
                            003F  2800                  .END START
```

Figure 21-3.

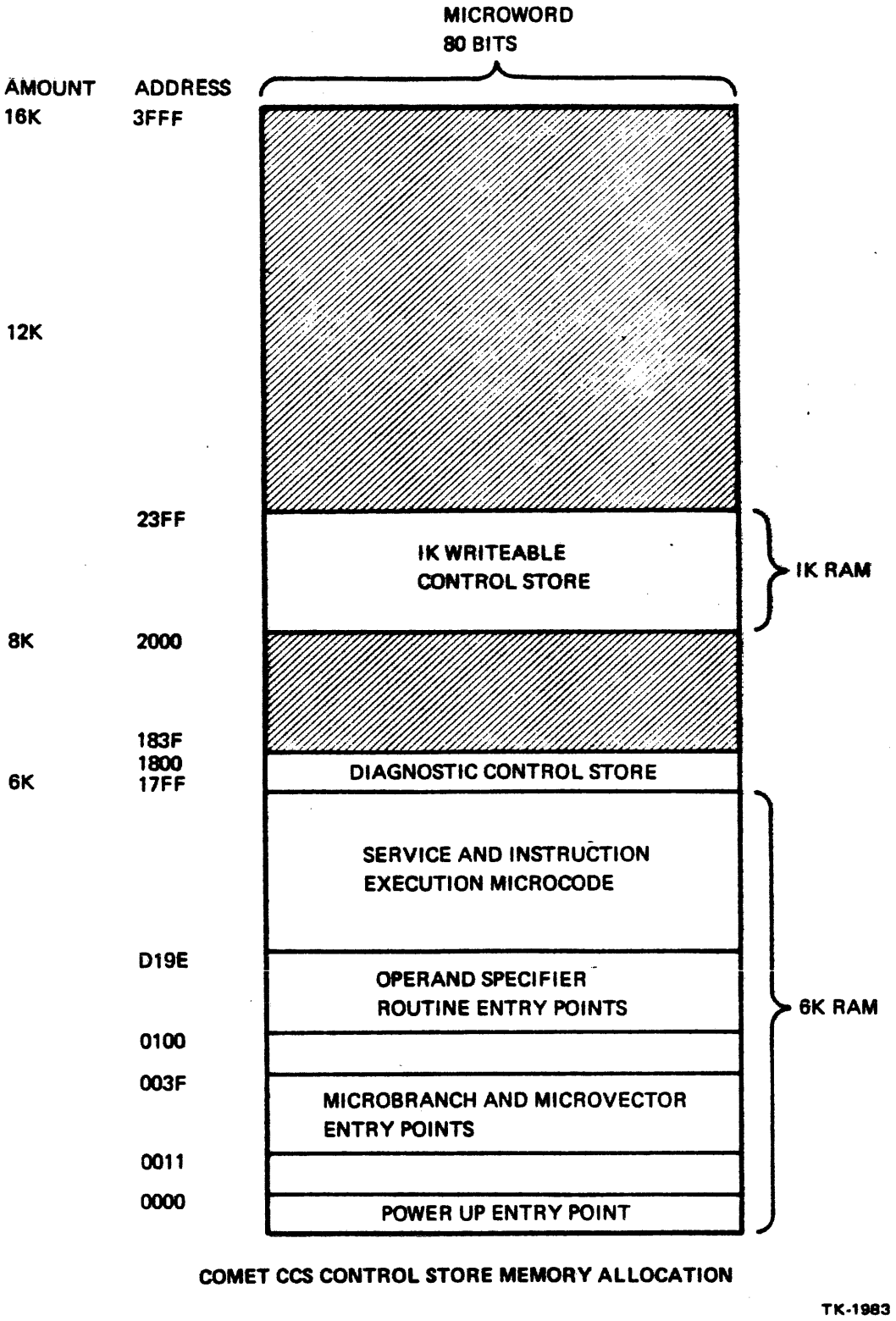


Figure 21-4. Comet CCS Control Store Memory Allocation

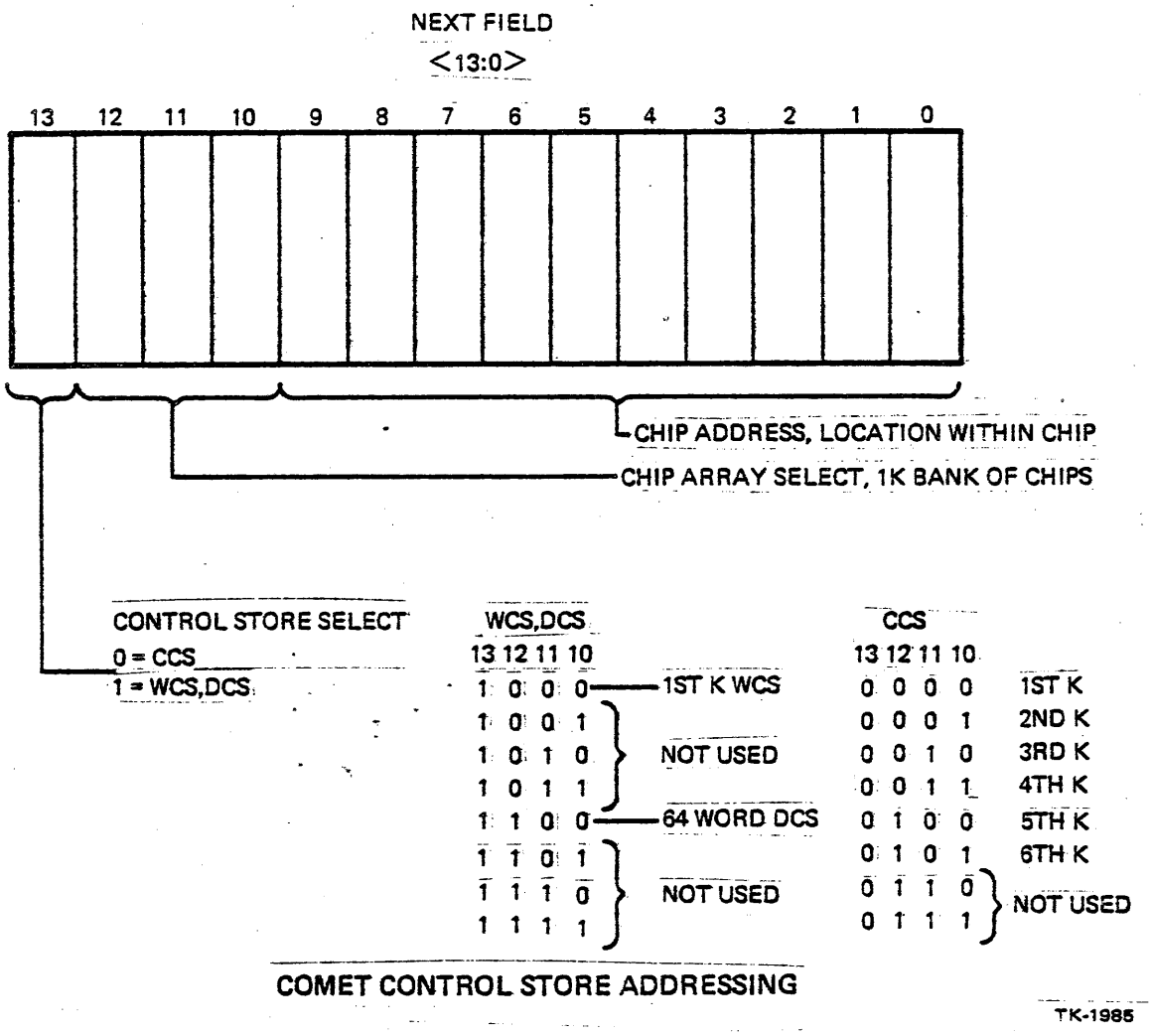
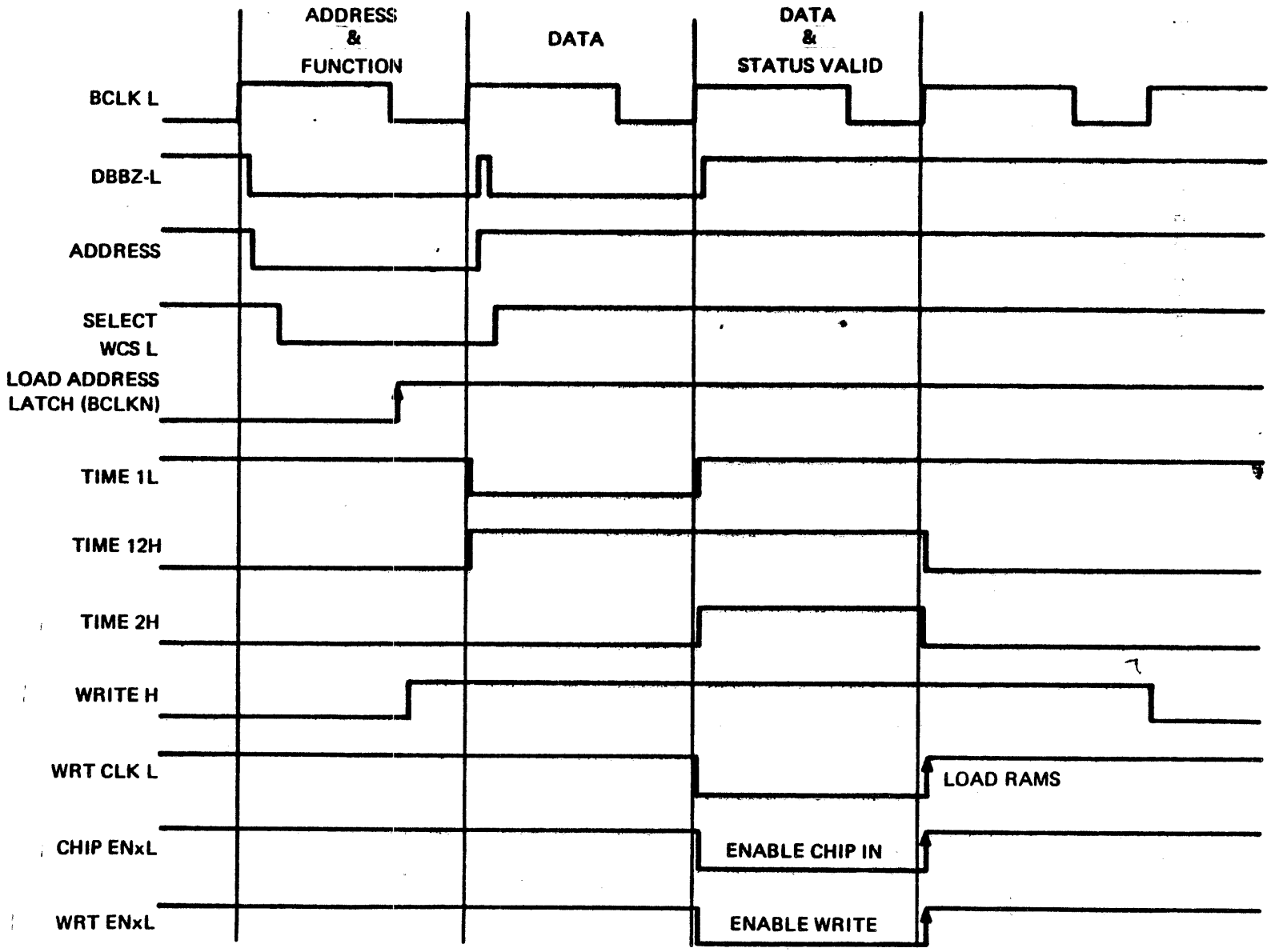
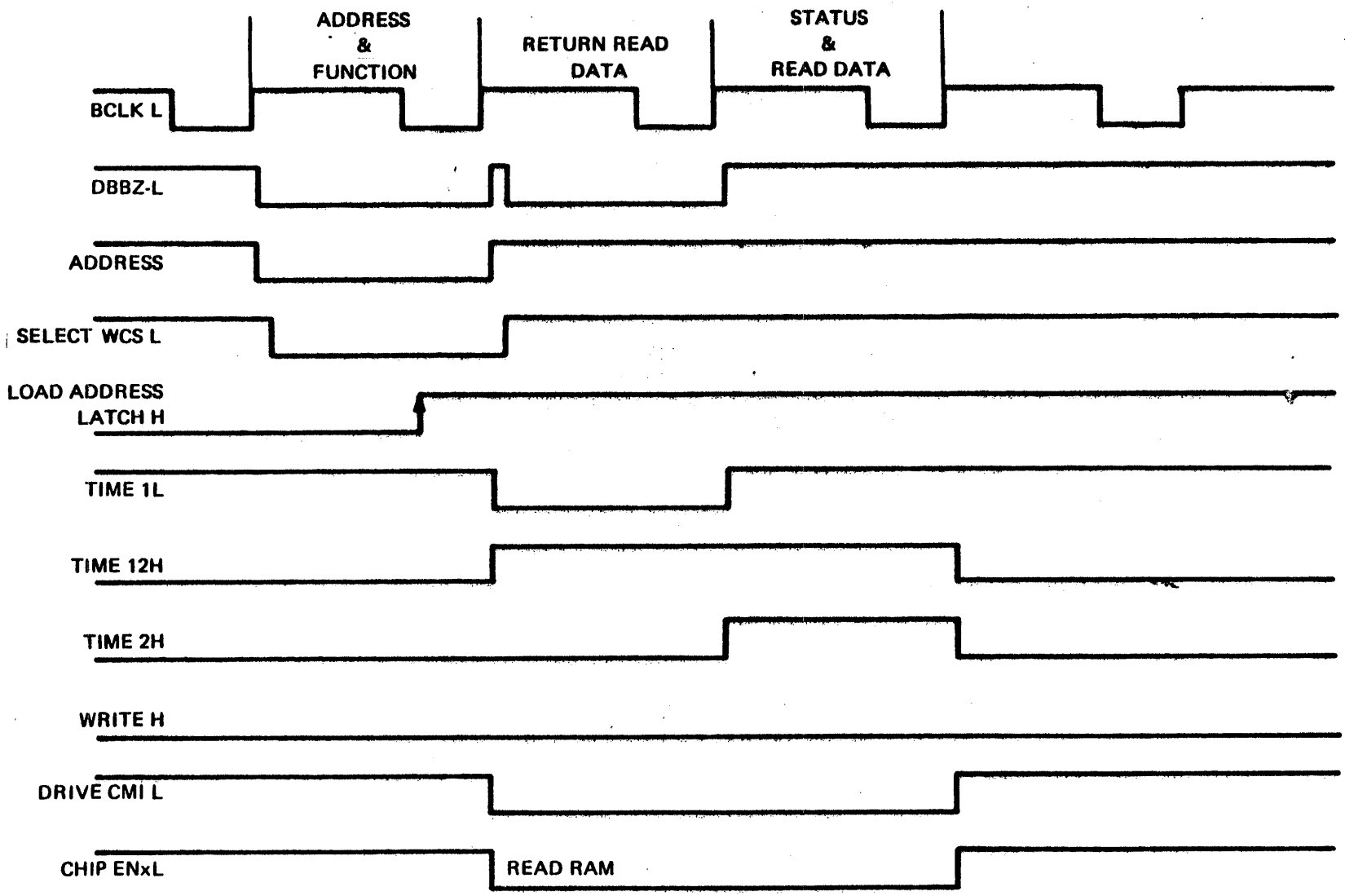


Figure 21-5. Comet Control Store Addressing


ADDRESS & FUNCTION
DATA
DATA & STATUS VALID
BCLK L
DBBZ-L
ADDRESS
SELECT WCS L
LOAD ADDRESS LATCH (BCLKN)
TIME 1L
TIME 12H
TIME 2H
WRITE H
WRT CLK L
CHIP ENxL
WRT ENxL
LOAD RAMS
ENABLE CHIP IN
ENABLE WRITE


ADDRESS
&
FUNCTION
RETURN READ
DATA
STATUS
&
READ DATA
BCLK L
DBBZ-L
ADDRESS
SELECT WCS L
LOAD ADDRESS
LATCH H
TIME 1L
TIME 12H
TIME 2H
WRITE H
DRIVE CMI L
CHIP ENxL
READ RAM

VAX-11/750 Level II

Power Systems

Student Guide

Course Produced By Educational Services Department
of
Digital Equipment Corporation

Power System

INTRODUCTION

The VAX-11/750 power system lesson consists of a block diagram of power distribution including memory battery backup and the power cells for the time of year clock. After a classroom lecture, a lab will be utilized to troubleshoot power system failures.

OBJECTIVES

Given a system that won't power up, isolate the malfunction and replace the faulty field replaceable unit.

SAMPLE TEST ITEM

The time of year receives its power from

a. +5 V supply
b. +12 V supply
c. 1.5 V dry cells
d. Battery backup

RESOURCES

1. VAX-11/750 System Maintenance Guide

2. VAX-11/750 Power System Technical Description

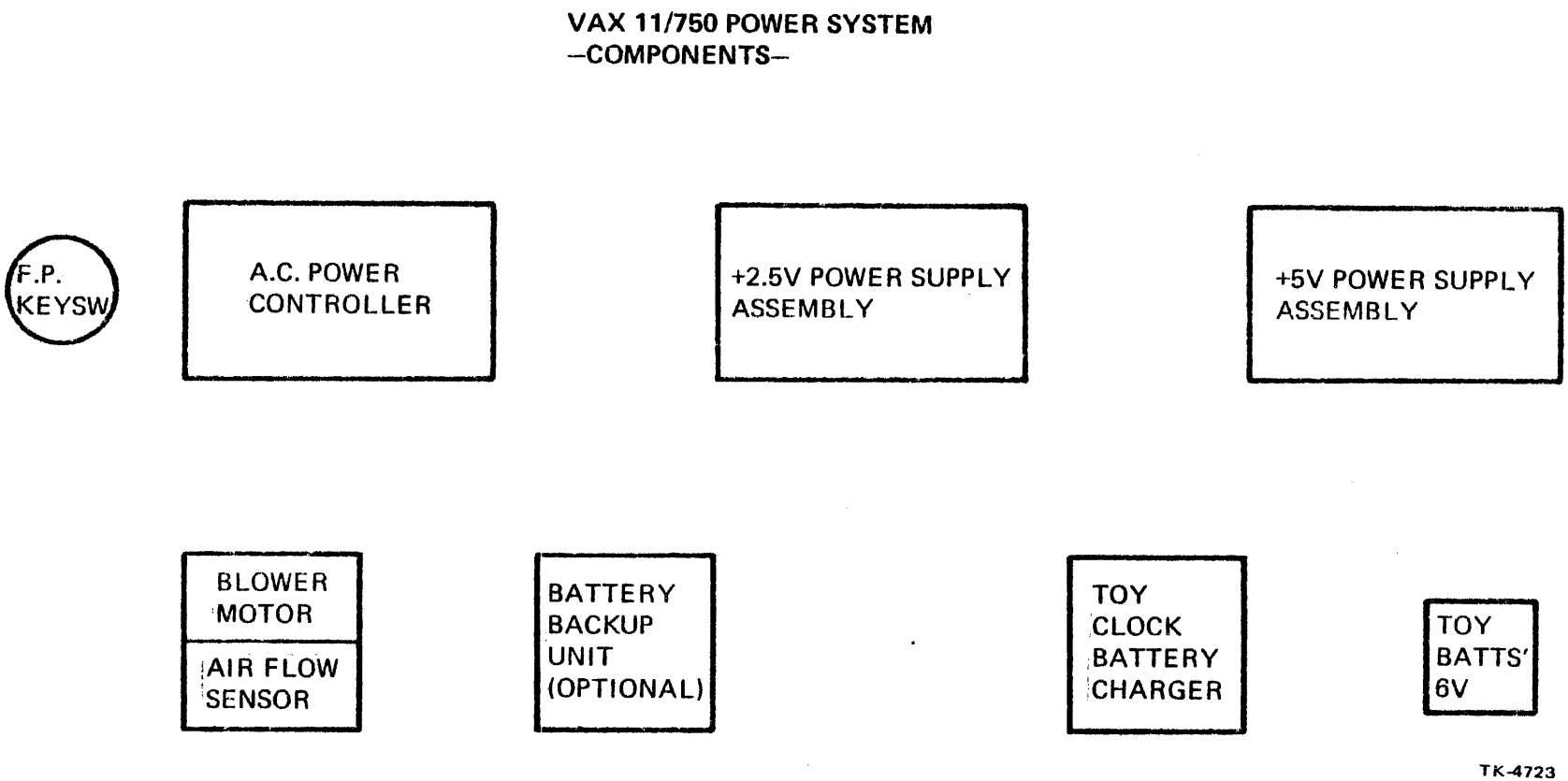


Figure 24-1 Power Components

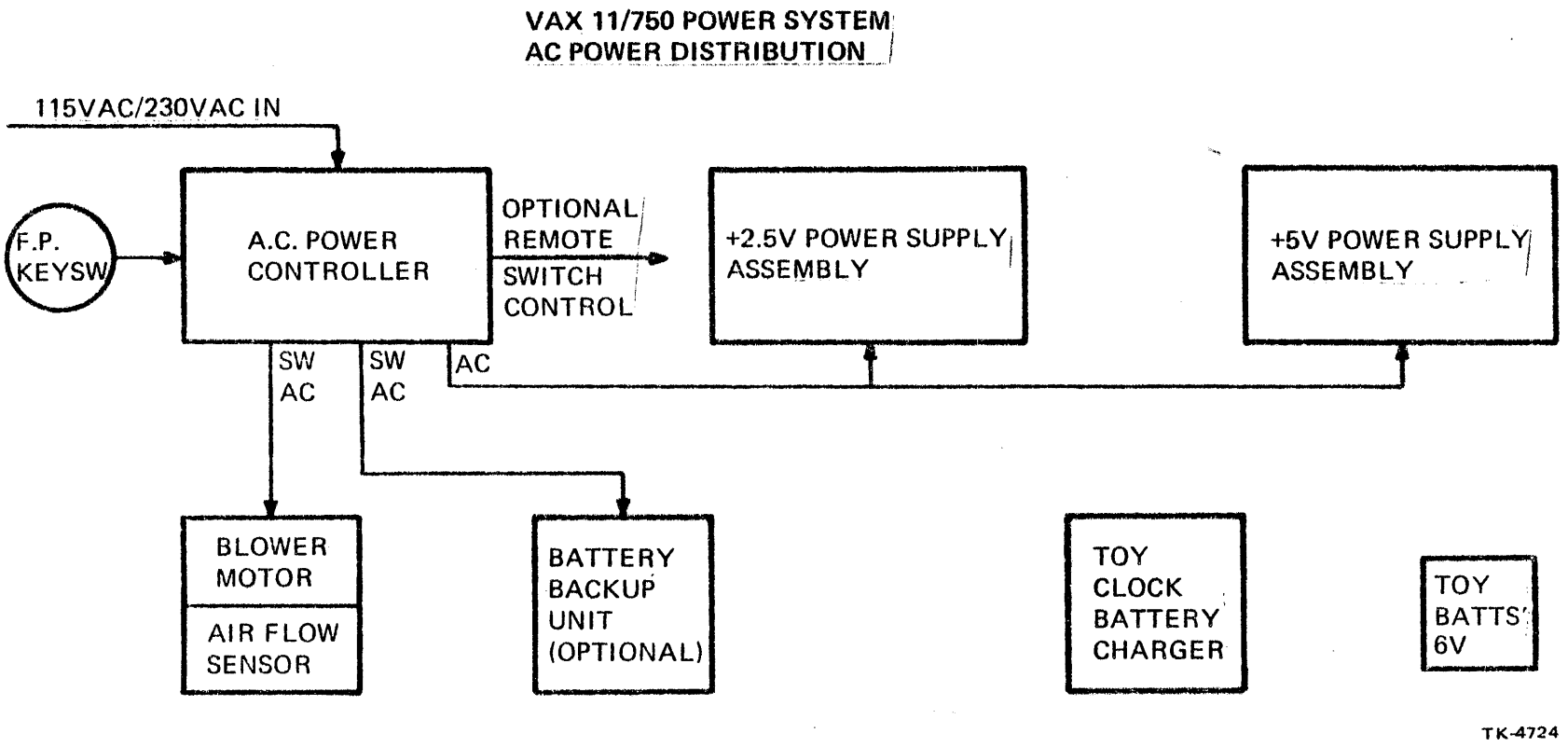


Figure 24-2 AC Distribution

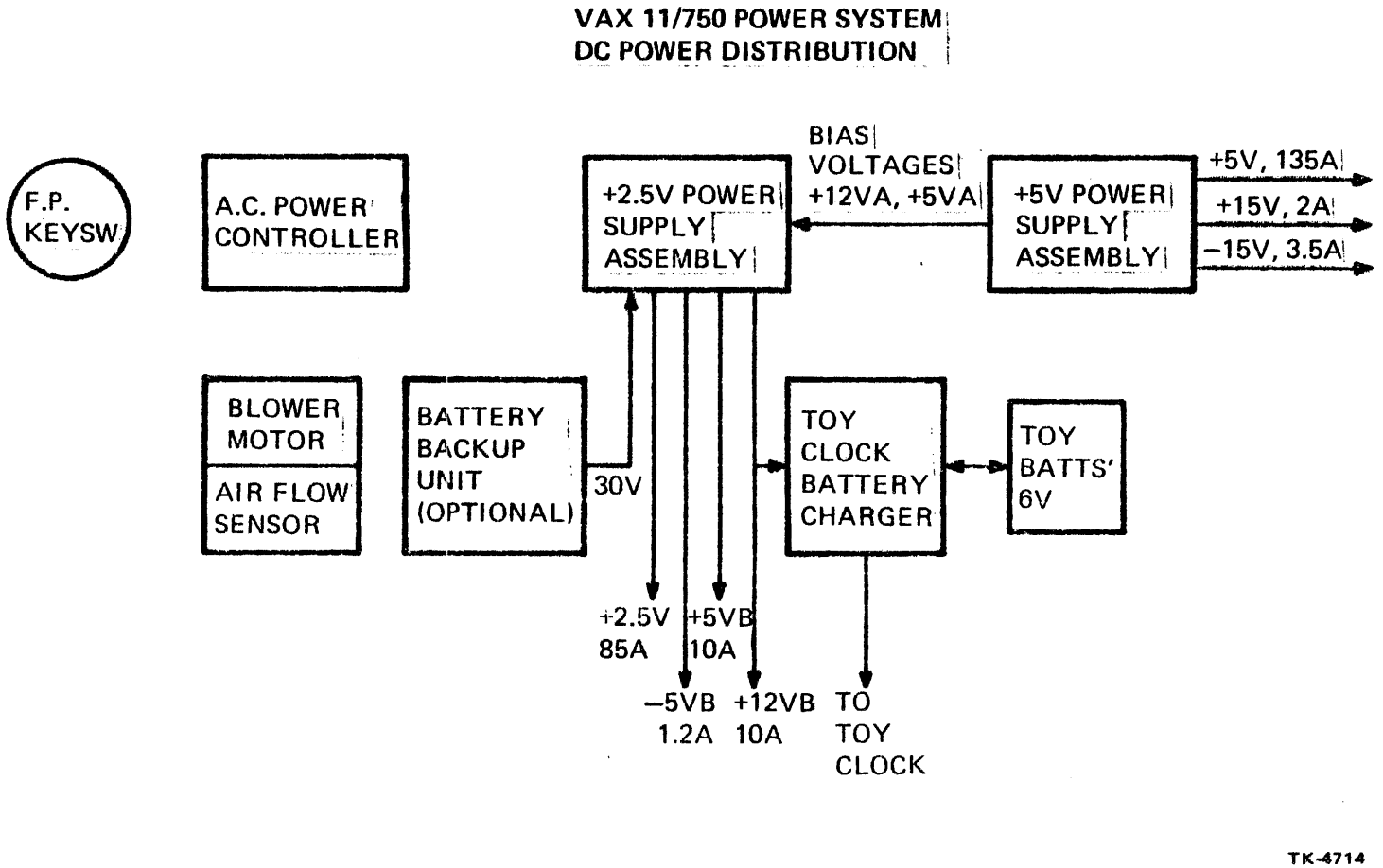


Figure 24-3 DC Distribution

CONTROLLER INDICATORS

Overvoltage (red)	- Indicates that there is an overvoltage condition in either the +2.5V or +5V power supply. The correct voltage of the failing box will not be present. Also, the appropriate fail indicator will be on.
Overcurrent (red)	- Indicates that there is an overcurrent condition in either the +2.5V or +5V power supply. The failing box will nott have an output. Also, the appropriate indicator will be on.
DC OK (green)	- Indicates the power system is in correct functioning order. If any other status indicator is on, this indicator is off.
+5 Fail (red)	- Indicates the +5V power supply is malfunctioning. The 5V box will not have a correct output.
+2.5 Fail (red)	- Indicates the +2.5V ppower supply is malfunctioning. The 2.5V box will not have a corret output.
Plug in reg fail (red)	- Indicates that either the ±5 volt, +12 volt, or ±14 volt regulator is malfunctioning.
Overtemp indicator (clear)	- Indicates an overtemperature condition in either the +2.5 volt or +5 volt power supply.
AC Power Indicator (amber)	- Indicates that AC is applied to the controller. It is on and remains on as long as the AC power cord is plugged in and AC is present.

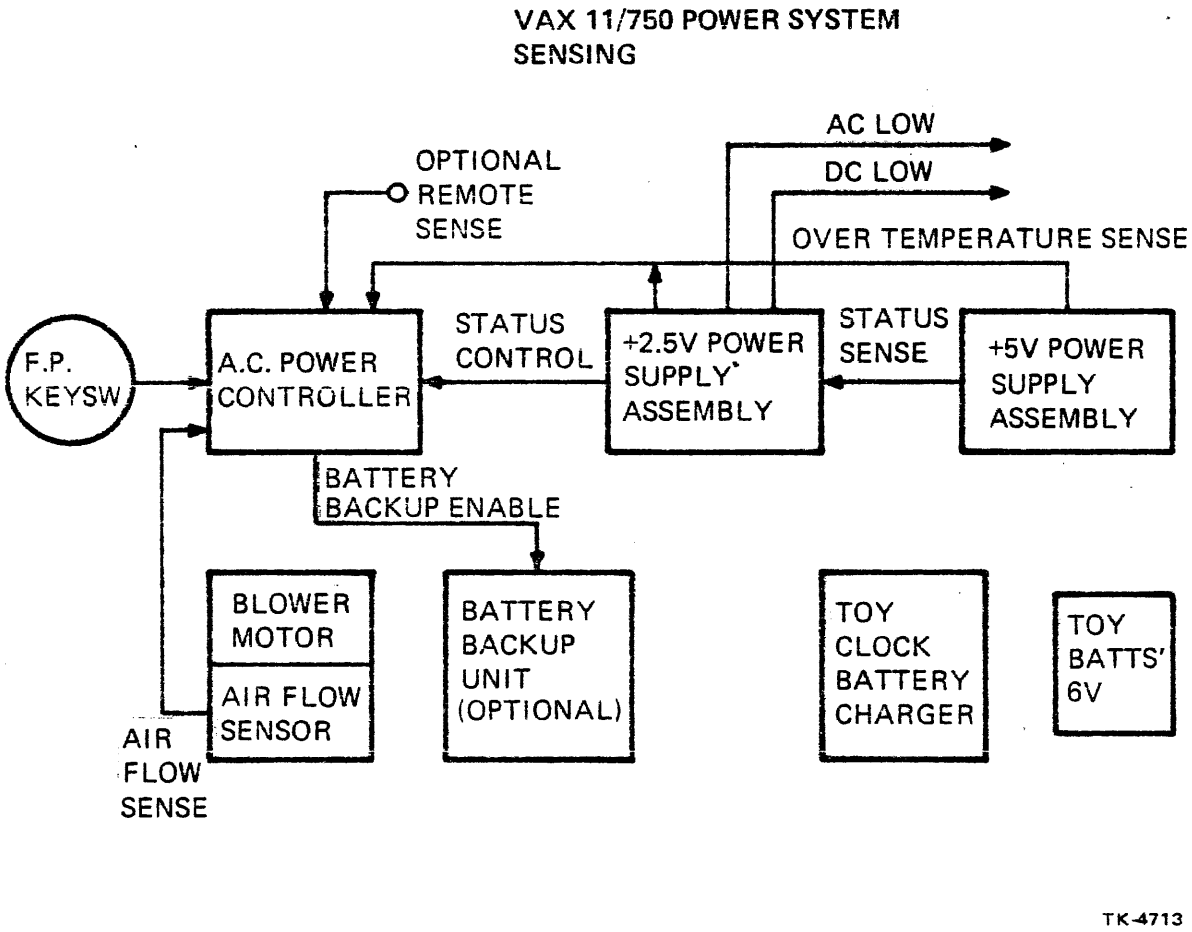


Figure 24-4 Power Sensing

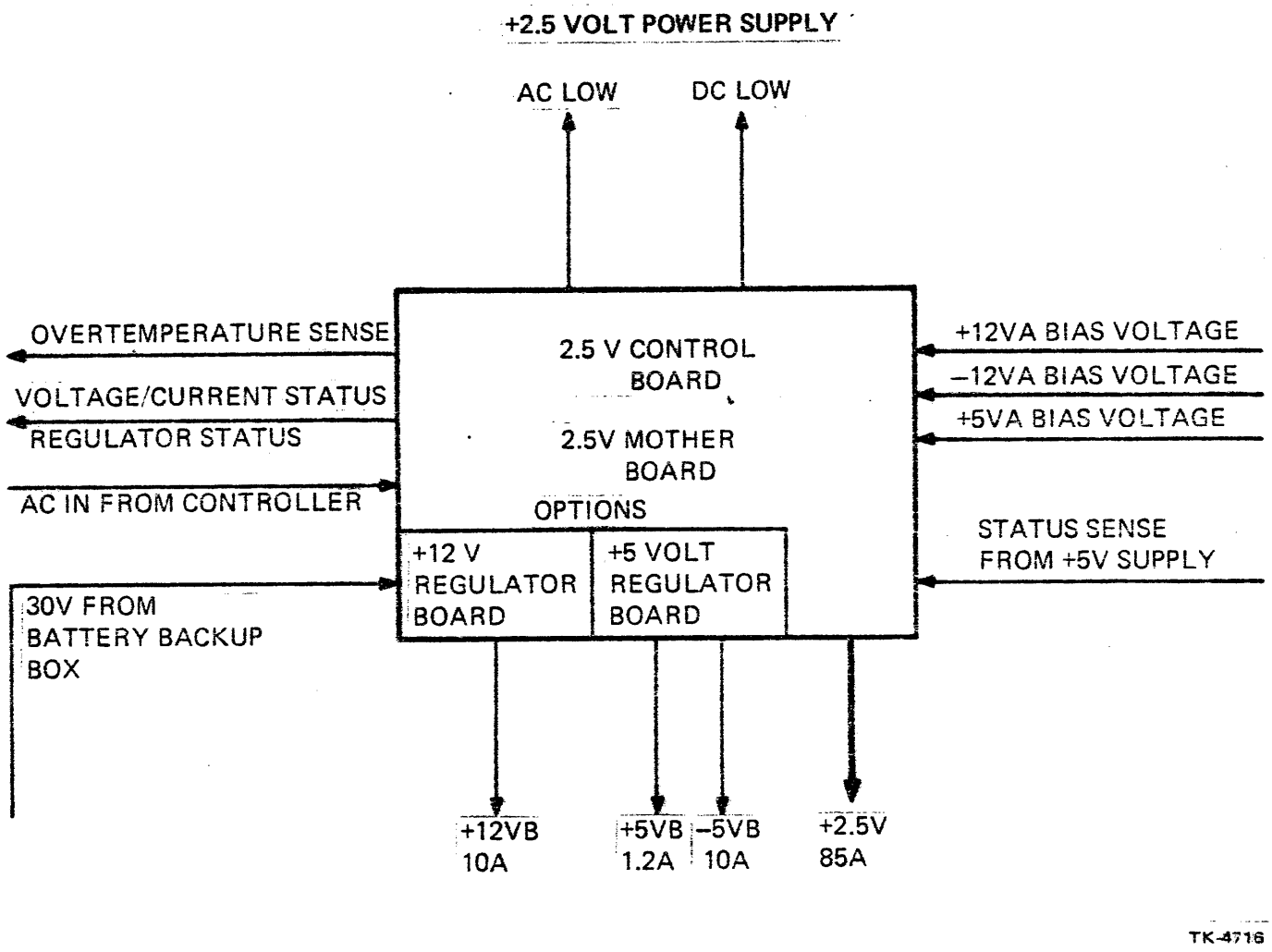


Figure 24-5 +2.5 Volt Supply

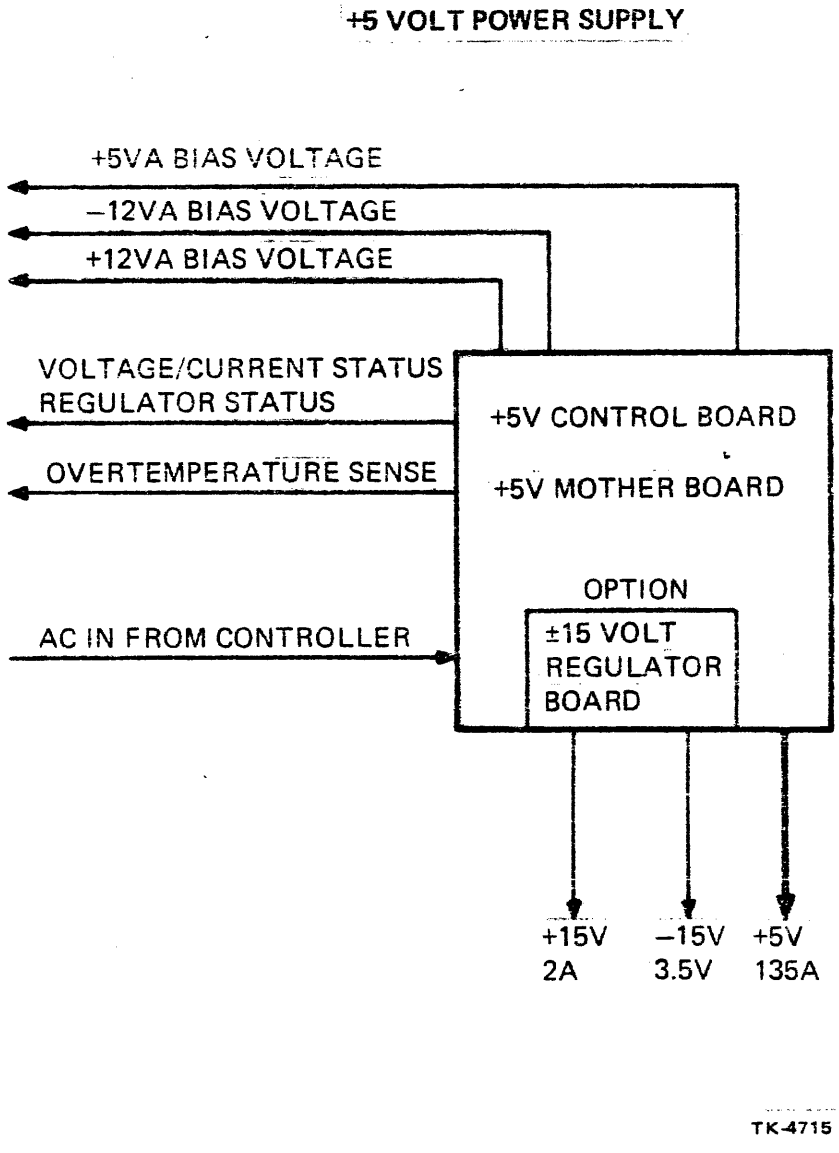


Figure 24-6 +5 Volt Supply

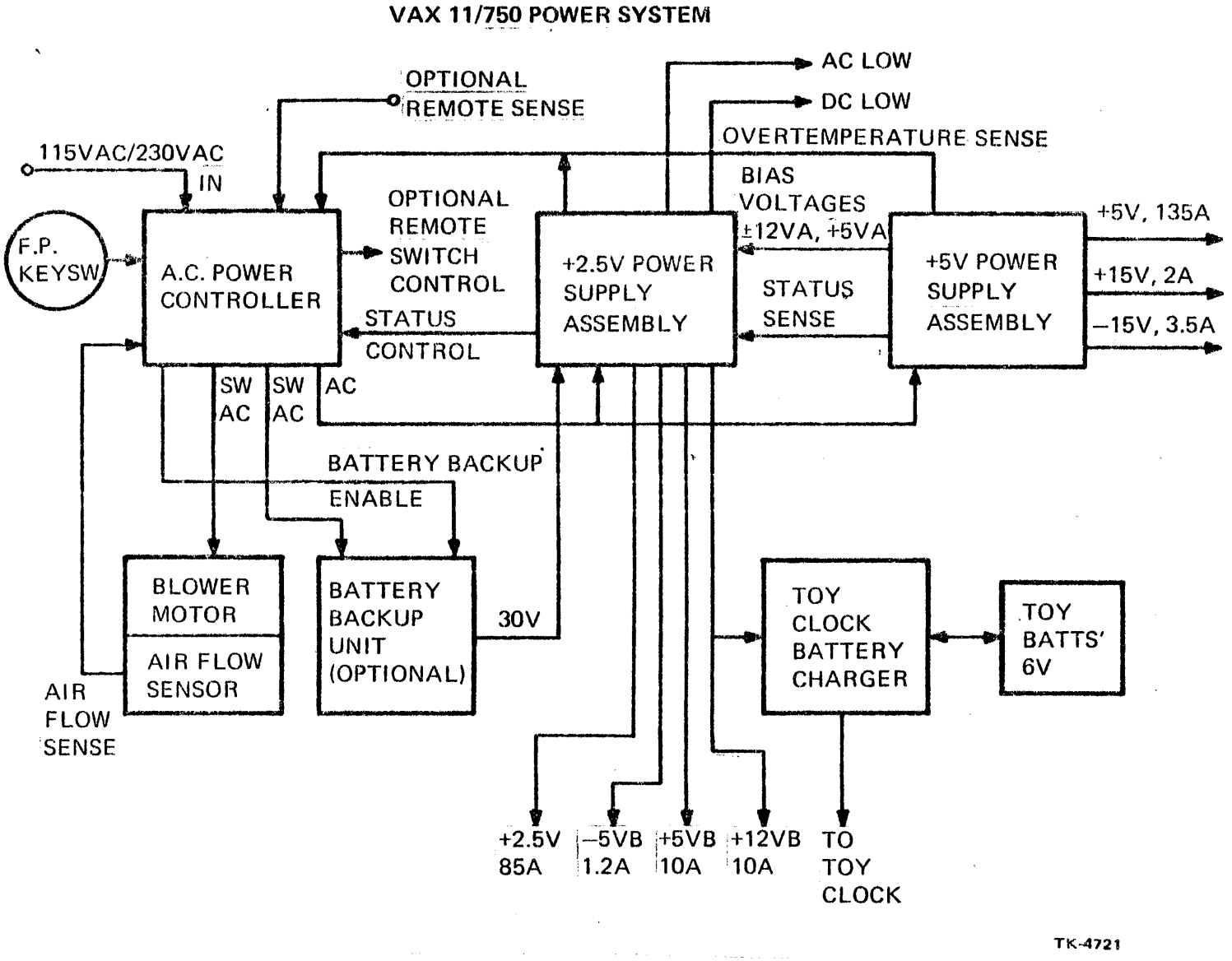


Figure 24-7 Total System

VAX-11/750 LEVEL II

Appendices

Course produced by Educational Services Department
of
Digital Equipment Corporation

Appendix A
Vectors and System Control Block

System Control Block Format

Vector	Description	IPL	I/E
SCBB+Ø	Not used	-	-
SCBB+4	Machine Check CS Parity Bad Ird Memory Error Cache Parity	1F	E
SCBB+8	Kernel Stack Invalid	1F	E
SCBB+C	Power Fail	1E	I
SCBB+1Ø	Reserved Opcode	1F	E
SCBB+14	Customer Opcode XFC	1F	E
SCBB+18	Reserved Operand	1F	E
SCBB+1C	Reserved Address Mode	1F	E
SCBB+2Ø	Access Violation	1F	E
SCBB+24	Translation Invalid	1F	E
SCBB+28	Trace Trap	1F	E
SCBB+2C	Breakpoint Opcode	1F	E
SCBB+3Ø	Compatability Mode	1F	E
SCBB+34	Arithmetic Trap	1F	E
SCBB+4Ø	CHMK	1F	E
SCBB+44	CHME	1F	E
SCBB+48	CHMS	1F	E
SCBB+4C	CHMU	1F	E
SCBB+54	Corrected Read Data	1A	I
SCBB+6Ø	Write Bus Error	1D	I
SCBB+84	Soft Interrupt	1	I
SCBB+88	Soft Interrupt	2	I
SCBB+8C	Soft Interrupt	3	I
SCBB+9Ø	Soft Interrupt	4	I
SCBB+94	Soft Interrupt	5	I
SCBB+98	Soft Interrupt	6	I
SCBB+9C	Soft Interrupt	7	I
SCBB+AØ	Soft Interrupt	8	I
SCBB+A4	Soft Interrupt	9	I
SCBB+A8	Soft Interrupt	A	I
SCBB+AC	Soft Interrupt	B	I
SCBB+BØ	Soft Interrupt	C	I
SCBB+B4	Soft Interrupt	D	I
SCBB+B8	Soft Interrupt	E	I
SCBB+BC	Soft Interrupt	F	I

SCBB+CØ	Interval Timer	18	I
SCBB+FØ	TU-58 Receive	17	I
SCBB+F4	TU-58 Transmit	17	I
SCBB+F8	Console Receive	14	I
SCBB+FC	Console Transmit	14	I
SCBB+16Ø	Massbus Adaptor Ø	15	I
SCBB+164	Massbus Adaptor 1	15	I
SCBB+168	Massbus Adaptor 2	15	I
SCBB+2ØØ	Unibus (SCBB+2ØØ+Unibus Vector)	14-17	I

Appendix B
Vector Operation

If Vector bits <1:Ø> are as follows...

Vector bits <1:Ø>=Ø	Use Kernel Stack unless IS bit = 1
Vector bits <1:Ø>=1	Use Interrupt Stack
Vector bits <1:Ø>=2	Trap to WCS location 2ØØ1, if WCS is not present or disabled, trap to location ØØØ1 in CS. Remove backplane jumper from slot 5 B44 to B48 if WCS is installed.
Vector bits <1:Ø>=3	Halt at Vector (PC points to interrupted instruction or faulted instruction

Appendix C
Machine Check and Write Bus Error

Logout Area and Error Codes...

Machine Check Exception Stack Logout Table

(SP)	LENGTH PARAMETER	00000028	SUMMARY PARAMETER
(SP)+4	SUMMARY PARAMETER	0000000X	
(SP)+8	VA	XXXXXXXX	1 = CS Parity Error
(SP)+C	PC	XXXXXXXX	2 = Memory Error
(SP)+10	MDR	XXXXXXXX	3 = Cache Parity
(SP)+14	SAVED MODE REG	0000000X	4 = Write Bus Error
(SP)+18	RLTO	0000000X	5 = Corrected Data
(SP)+1C	TBGPR	0000000X	7 = Bad IRD
(SP)+20	CAER	0000000X	
(SP)+24	BER	0000000X	
(SP)+28	MCESR	0000000X	
(SP)+2C	PC	XXXXXXXX	
(SP)+30	PSL	XXXXXXXX	

ACV OR TNV
Translation Not Valid or Access Violation exception stack logout (Exception Service from Vectors 24 and 20 Respectively)

(SP)	Error Code (See Below)
(SP)+4	Virtual Address Referenced
(SP)+8	Program Counter
(SP)+12	Processor Status Longword

Error Codes:
- 0= Read Access Violation or XB Access Violation or PTE Fetched not valid for read
- 1= Accessing System 1 Space (S1) or length Violation
- 2= No Access to Process Page Table (from SPTE)
- 3= Process PTE VA not in System Virtual Space
- 4-7= Same as 0 to 3 but for write access rather than read access

COMPATABILTY MODE TRAP
Compatabilty Mode Stack Logout (Exception Service from Vector 30)

(SP)	Error Codes (See Table Below)
(SP)+4	Program Counter
(SP)+8	Processor Status Longword

Error Codes: Ø= PDP 11 Reserved Operand
1= Breakpoint Opcode Executed
2= I/Ø Trap
3= Emulator Trap
4= Trap
5= Reserved Instruction (HALT)
6= Odd Address Referenced

ARITHMETIC TRAP
Arithmetic Trap Stack Logout (Exception Service from Vector 34)

(SP) Error Code
(SP)+4 Program Counter
(SP)+8 Processor Status Longword

Error Codes: Ø= Undefined
1= Integer Overflow
2= Integer divide by zero
3= Floating Overflow
4= Floating/Decimal divide by zero
5= Floating Underflow
6= Decimal Overflow
7= Subscript Out of Range

Appendix D
Console Commands

The Comet system has console functionality similiar to the VAX 11/780. These commands are illustrated below with examples.

Commands

The console prompt is the same as the VAX 11/780 ">>>" and appears at the beginning of every line.

"^P"	Enter Comet Console mode
"^D" RDM>	Enter RDM console mode
>>>E	Examine Command
>>>D	Deposit Command
>>>I<CR>	Init Command, Invalidates TB, Cache, and does Processor Init and Unibus Init
>>>T<CR>	Test Command, Runs Micro Verify Microroutine explained below
>>>S 1000<CR> >>>S<CR>	Start command, The command may have an address argument following or carriage return. If a carriage return is typed, the address in the PC is used. The start command does an init sequence before going to IRD1 of the macroinstruction pointed to by the PC.
>>>C<CR>	The Continue command is the same as the start command and starts macrocode execution at the address in the PC.
>>>N<CR>	This command is used to single step the macroinstructions after the PC is loaded.
>>>B<CR>	The Boot command in this example will boot the device selected by the front panel DEVICE switch

```
>>>X             Apt Load and Dump Command
```

Command Switches for Examine and Deposit Console Commands

Size switches	/B	sets the data size to byte
	/W	word
	/L	long
Function	/G	GPR
	/I	IPR
	/P	Physical Memory
	/V	Virtual Memory
	<SP>P	PSL

Command Switches for Boot Console Command

```
>>>B/X DDCU<CR>  Boot Device selected by DDCU typed
                 at console and inhibit Micro Verify
                 Test.

>>>B/n DDCU<CR>  Boot Device selected by DDCU typed
                 at console and pass a four digit
                 number as software control flags to
                 VMB.EXE in R5

>>>B DDCU<CR>    Boot device specified by operator
```

Examples

```
>>>D/G/L F 1000<CR>        ;Put 1000H in PC
>>>D/P 1000 005251D0<CR>  ;Put code in address 1000H
>>>E/I  25<CR>             ;Examine cache disable Reg
>>>I<CR>                   ;Do Init sequence
>>>B/10/X DMA0<CR>         ;Boot Diagnostic Supervisor
                          ;without Micro Verify from
                          ;DMA0
```

CONSOLE COMMAND ERROR CODES

If an illegal console command is attempted or command is aborted because of a microtrap or some other condition a two digit error code is typed out and the console waits for new input. For example...

```
>>>E P<CR> !Examine PSL
>>>E<CR>   !Implies Examine Next Location, this is illegal.
?11        !Question Mark and error code is typed by console
>>>        !At this point ready for new command
```

Error Codes

20= Deposit or Examine of Memory Failed (Access Violation, Translation not valid, Bus Error, TB Parity Error, or Control Store Parity Error)
11= Illegal access of an IPR
30= Apt Loading Checksum error
33= Attempt to Boot from unknown Device type (DM,DL,DT,DR)
34= Boot Device Controller not "A","B","C", or "D"

MICRO VERIFY

The following table indicates the microtest sequence during micro verify. If a test failure occurs, the PC is replaced with an error code and the failure letter is typed. Micro Verify then merges to console front end flows.

Normal Test sequence as appears at console after power up with FPS1 set to HALT

```
%%
00000000 16
>>>
```

Test in Progress	Test Name	Fail Character
1.	R-Bus, W-Bus, D Reg Tests	@
2.	M-Bus, Q-Reg Test	C
3.	Scratchpad Test	E
4.	Scratchpad Explicit Address Test Mtemps	F
5.	Scratchpad Explicit Address Test Rtemps	I
6.	Scratchpad Explicit Address test IPRs	J
7.	Scratchpad Explicit Address test GPRs	L
8.	Dual Port Address Test	L
9.	XB,IR, and OSR tests	O
10.	XB,PC, and PC+Isize test	Q
11.	D-Size Tests	R
12.	D-Size Tests	T
13.	Cache Parity Checker Test	X
14.	TB Parity Checker Test	[
15.	Control Store Parity Checker Test	]
16.	Cache Test	^

Test Failure sequence would appear at console as follows

%F This indicates a failure of the Mtemp Scratchpad address test.

ØØØØØXXX FF PC contains loop count or point at which test
>>> failed and "FF" indicates micro verify failure.

CONSOLE HALT Error Codes that are typed upon execution of the following conditions:

Control P while in console mode	Code=ØØ
Execute TEST console command	Code=Ø1
Control P Halt or single macroinstruction mode>>>N<CR>	Code=Ø2
Interrupt Stack Not Valid	Code=Ø4
Halt Instruction Executed	Code=Ø6
Vector Bits <1:Ø>=3, Halt at Vector	Code=Ø7
Vector Bits <1:Ø>=2, WCS disabled or not present	Code=Ø8
Change Mode Instruction executed on Interrupt Stack	Code=ØA
Change mode instruction executed and vector <1:Ø>not=Ø	Code=ØB
Double Bus Write error halt	Code=ØF
Power up and can't find RPB, FPS1 at RESTART/HALT	Code=11
Power up and warm start flag false FPS1 at RESTART/HALT	Code=12
Power up and can't find good 64K of memory	Code=13
Power up and booting, but bad or no Boot ROM	Code=14
Power up and cold start flag set during boot subroutine	Code=15
Power up halt FPS1 at HALT position	Code=16
Micro verify test failure	Code=FF

The format for entering console mode is that the PC is typed and a two digit error code is immediately following. For Example...

```
ØØØ1ØØØ4  Ø6
>>>
```

The preceding example indicates that a halt instruction was executed at location 1ØØØ3.

Appendix E
RDM Console Command Summary

Control Key Functions

Control D	Enter RDM console mode.
Control P	Enter Comet Console Mode
Control U	Abort current Command Line
Control O	Inhibit Printing of text
Control R	Retype current command line
Control C	Cancel current function (Repeat console command)
Control S	Disable CPU output to active Terminal
Control Q	Continue Output to Terminal after Control S

RDM Console Commands

RDM>TE — Load and Run Microdiagnostics

RDM>TE/C — Load Micromonitor and go to Micromonitor parser

```
MIC>
```

RDM>TE FILENAME.EXT — Load different monitor program and transfer control to it. (WCS Debugger etc.)

RDM>LOA FILENAME.EXT <PHYS ADDRESS> — Load RT11 file from TU-58 into CMI memory at <PHYS ADDRESS>. If no address is specified, default is Ø

RDM>TA — Enable Talk mode between local and Remote Terminal (Used during RD session)

RDM>E — Examine Command, the following are valid Examine command switches

E/B <ADDRESS>	Data size is byte
E/W <ADDRESS>	Data size is word
E/L <ADDRESS>	Data size is long

RDM>D — Deposit Command, the following are valid Deposit command switches

D/B <ADDRESS> <DATA>	Data size is byte
D/W <ADDRESS> <DATA>	Data size is word
D/L <ADDRESS> <DATA>	Data size is long

RDM>SE 2ØØ1 Set micromatch address at 2ØØ1 and generate scope sync

RDM>TR Trace until micromatch, dumps DCS RAM for 64 Rom States prior to micromatch, most recent microaddress is printed first.

RDM>CL Clear stop on Micromatch

RDM>STE Single Microinstruction cycle

RDM>STE/T Single Tick Clock

RDM>STO Stop CPU Clock

RDM>CON Restart the CPU clock

RDM>PAR <CS ADDRESS>
Perform a Parity Scan of the control store beginning at the location specified. There is bad parity written into location 17FD so that is where the parity scan stop.

RDM>UA <CS ADDRESS>
Reads the control store microinstruction at the <CS ADDRESS and latches the microinstruction. Clock is stopped.

RDM>UA/C <CS ADDRESS>
Similiar to above except microinstruction is not latched.

RDM>INI Do a processor Init (same as Front Panel Init)

RDM>SH Displays CPU control store address of current microinstruction, and next field of the next microinstruction. (Clock must be stopped.)

RDM>SH/V Displays the version and date of the RDM 8Ø85 rom macrocode.

RDM>REP Repeat the last console command continuously

RDM>R E/B Ø Repeat the current console command continuosly

RDM>RET	Return to program I/O mode
RDM>RET/D	Return to program I/O mode but leave microbreak set.

RDM CONSOLE ERROR CODES

Tape function errors

TAP:Ø1	UART - Device timeout
TAP:Ø2	UART - Error from UART
TAP:Ø3	UART - Data Set Ready dropped
TAP:Ø4	UART - Receive Overflow
TAP:Ø5	Tape checksum error received
TAP:Ø6	Tape count byte exceeded maximum
TAP:Ø7	Tape no end packet, invalid operation
TAP:Ø8	Tape invalid packet received
TAP:Ø9	Tape file not found
TAP:12	Tape Directory Error
TAP:13	Tape flag received, not command or data
TAP:14	Tape Read Length Error, not all records fit
TAP:C9	Tape Bad Record number
TAP:DØ	Tape Bad Operation Code
TAP:DF	Tape Motor stopped
TAP:EØ	Tape Block not found
TAP:EF	Tape Data check error
TAP:F5	Tape write protocol error
TAP:F7	Tape cartridge not present
TAP:F8	Tape Bad Unit number
TAP:EE	Tape End of medium
TAP:FF	Tape diagnostic failure

Terminal Error Codes

TRM:ØA	Terminal Control C received
TRM:ØB	Terminal Command input buffer overloaded
TRM:ØC	Terminal Control D received
TRM:ØD	Terminal Command Input larger than buffer
TRM:ØE	Terminal Remote Line CRC error occured

CMI Error Codes

CMI:ØØ	Nonexistent memory
CMI:Ø1	Corrected Read Data
CMI:Ø2	Read Data Substitute

General Errors

SYNTAX ERROR	Error in entering console commands
INVALID COMMAND RDM	does not know the command just entered
RDM:1Ø	Operation already in progress
RDM:11	Invalid operation code contained in Macro

Appendix F
Power Up and Boot Error Reports

FPS1 set to either RESTART/BOOT or BOOT

%%%% xxxxxxxx 13 >>>	This indicates that a good 64KB section of memory was not found and return to console mode
%%%% xxxxxxxx 14 >>>	This indicates a failure or nonexistence of the boot ROM
xxxxxxxx Ø6	If a halt instruction is executed after typing a console
>>>	Boot command, this indicates a failure of the read of logical block Ø from the selected boot device. The PC should be equal to the base address of the first good 64KB of memory plus FX16 for TU58 or FX2Ø for RKØ6. This failure occurs in the Boot ROM routine.

VMB PRIMARY BOOT FAILURES

BOOT is the program name for VMB.EXE
The "F" indicates a fatal error and the type of error is reported.

%BOOT-F-Unknown processor	This indicates that CPU is not a Comet or 11/78Ø, check SID register for proper jumpering in the CPU type field on the Backplane.
%BOOT-F-Unexpected Exception	This indicates that one of the following exceptions occurred. 1. Access Violation 2. Breakpoint Opcode 3. Reserved Operand 4. TBit Trap 5. Page Fault (TNV)

%BOOT-F-Unexpected Machine Check	This indicates some sort of machine Check occurred. Check all adaptors using console examine and deposit commands. Probabably a timeout.
%BOOT-F-Nonexistent Drive	Self explanatory, Check DEFBOO.CMD on 11/780 and insure system disk is drive being booted.
%BOOT-F-Unable to locate BOOT file	VMB can't find [SYSEXE]SYSBOOT.EXE or if bit 4 in R5 is set, VMB can't find [SYSMAINT]DIAGBOOT.EXE
%BOOT-F-Bootfile not contiguous	Indicates that [SYSEXE]SYSBOOT.EXE or [SYSMAINT]DIAGBOOT.EXE is not contiguous on system disk. Recopy or rebuild
%BOOT-F-I/O error reading boot file	Indicates problem reading boot file from disk by $QIO service.