EK-KDJ1B-UG-001

KDJ11-B CPU Module

User's Guide

digital™

EK-KDJ1B-UG-001

KDJ11-B CPU Module

User's Guide

Prepared by Educational Services
of Digital Equipment Corporation

Preliminary Edition, January 1986
1st Edition, November 1986

© Digital Equipment Corporation 1986.
All Rights Reserved.
Printed in U.S.A.

The material in this manual is for informational purposes and is subject to change without notice. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

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

The manuscript for this book was created on a VAX-11/780 system and, via a translation program, was automatically typeset by Digital's DECset Integrated Publishing System. The book was produced by Educational Services Development and Publishing in Marlboro, MA.

The following are trademarks of Digital Equipment Corporation:

digital™
DEC
DECmate
DECnet
DECsystem-10
DECSYSTEM-20
DECUS
DECwriter
DIBOL
EduSystem
IAS
MASSBUS
MicroPower/Pascal
MINC-11
OMNIBUS
OS/8
PDP
PDT
P/OS
Professional
Q-Bus
Q22-Bus
Rainbow
RSTS
RSX
RT
RT-11
TOPS-10
TOPS-20
UNIBUS
VAX
VAXstation
VAXstation II
VMS
VT
Work Processor

CONTENTS

PREFACE

CHAPTER 1 ARCHITECTURE

1.1 DESCRIPTION 1-1
1.2 DCJ11-A FEATURES 1-2
1.2.1 Stack Limit Protection 1-3
1.2.2 Kernel Protection 1-3
1.2.3 General Registers 1-4
1.2.4 Stack Pointer 1-4
1.2.5 Program Counter 1-4
1.2.6 Processor Status Word (17 777 776) 1-4
1.2.7 CPU Error Register (17 777 766) 1-6
1.2.8 Program Interrupt Request Register (17 777 772) 1-7
1.3 INTERRUPTS 1-8
1.3.1 Sunset Loops 1-8
1.3.2 Red Stack Aborts 1-10
1.3.3 Addressing Errors 1-10
1.3.4 Bus Timeout Errors 1-10
1.3.5 Interrupt Vector Timeouts 1-10
1.3.6 No SACK Timeouts 1-10
1.4 MEMORY MANAGEMENT 1-10
1.4.1 Memory Mapping 1-11
1.4.1.1 16-Bit Mapping 1-11
1.4.1.2 18-Bit Mapping 1-12
1.4.1.3 22-Bit Mapping 1-12
1.4.2 Compatibility 1-13
1.4.3 Virtual Addressing 1-13
1.4.4 Interrupts Under Memory Management 1-14
1.4.5 Construction of a Physical Address 1-14
1.4.6 Memory Management Registers 1-16
1.4.6.1 Page Address Registers 1-18
1.4.6.2 Page Descriptor Register 1-18
1.4.7 Fault Recovery Registers 1-20
1.4.7.1 Memory Management Register 0 (17 777 572) 1-20
1.4.7.2 Memory Management Register 1 (17 777 574) 1-20
1.4.7.3 Memory Management Register 2 (17 777 576) 1-20
1.4.7.4 Memory Management Register 3 (17 772 516) 1-22
1.4.7.5 Instruction Back-Up/Restart Recovery 1-23
1.4.7.6 Clearing Status Registers Following Abort 1-23
1.4.7.7 Multiple Faults 1-23

1.4.8 Typical Usage Examples ... 1-23
1.4.8.1 Typical Memory Page ... 1-24
1.4.8.2 Nonconsecutive Memory Pages ... 1-25
1.4.8.3 Stack Memory Pages ... 1-26
1.4.9 Transparency ... 1-26
1.5 CACHE MEMORY ... 1-27
1.5.1 Parity ... 1-29
1.5.1.1 Parity Errors ... 1-29
1.5.1.2 Multiple Cache Parity Errors ... 1-29
1.5.2 Memory System Registers ... 1-30
1.5.2.1 Cache Control Register (17 777 746) ... 1-30
1.5.2.2 Hit/Miss Register (17 777 752) ... 1-32
1.5.2.3 Memory System Error Register (17 777 744) ... 1-32
1.6 PRIVATE MEMORY INTERCONNECT ... 1-34
1.6.1 PMI Protocol ... 1-34
1.6.1.1 Bus Device NPR ... 1-34
1.6.1.2 Bus Device Interrupt ... 1-34
1.6.2 PMI Data Transfers ... 1-34
1.6.2.1 Data In/Data In Pause ... 1-34
1.6.2.2 Block Data In ... 1-35
1.6.2.3 Data Out/Data Out Byte ... 1-35
1.7 TERMINAL INTERFACE ... 1-35
1.7.1 Receiver Control/Status Register (17 777 560) ... 1-36
1.7.2 Receiver Buffer Register (17 777 562) ... 1-37
1.7.3 Transmitter Control/Status Register (17 777 564) ... 1-38
1.7.4 Transmitter Buffer Register (17 777 566) ... 1-39
1.8 BOOT AND DIAGNOSTIC FACILITY ... 1-39
1.8.1 Control/Status Register (17 777 520) ... 1-40
1.8.2 Page Control Register (17 777 522) ... 1-42
1.8.3 Configuration and Display Register (17 777 524) ... 1-43
1.8.3.1 Boot and Diagnostic Configuration Register ... 1-43
1.8.3.2 Boot and Diagnostic Display Register ... 1-43
1.8.4 Maintenance Register (17 777 750) ... 1-43
1.9 LINE TIME CLOCK ... 1-45
1.9.1 Line Time Clock Register (17 777 546) ... 1-45

CHAPTER 2 CONFIGURATION

2.1 INTRODUCTION ... 2-1
2.2 MODULE CONFIGURATION ... 2-1
2.2.1 Jumper Wires ... 2-1
2.2.1.1 W10 Jumper ... 2-1
2.2.1.2 W20 Jumper ... 2-1
2.2.1.3 W40 Jumper ... 2-3
2.2.2 Switchpack ... 2-3
2.2.2.1 Baud Rate Selection ... 2-4
2.2.2.2 Dialog Mode ... 2-5
2.2.2.3 Device Bootstrap Programs ... 2-5
2.2.2.4 Console Enable ... 2-6
2.2.3 Diagnostic LEDs ... 2-6
2.3 EEPROM CONFIGURATION PARAMETERS ... 2-7
2.3.1 Enable Halt-on-Break ... 2-9
2.3.2 Disable User Friendly Format ... 2-9
2.3.3 ANSI Video Terminal ... 2-9

2.3.4 Power-Up Modes 2-9
2.3.4.1 Dialog Mode 2-9
2.3.4.2 Automatic Mode 2-10
2.3.4.3 ODT Mode 2-10
2.3.4.4 Mode 24 2-10
2.3.5 Restart 2-10
2.3.6 Ignore Battery 2-10
2.3.7 PMG Count 2-10
2.3.8 Disable Clock CSR 2-10
2.3.9 Force Clock Interrupts 2-11
2.3.10 Clock Select 2-11
2.3.11 Enable ECC Test 2-11
2.3.12 Disable Long Memory Test 2-11
2.3.13 Disable ROM 2-11
2.3.14 Enable Trap-on-Halt 2-12
2.3.15 Allow Alternate Boot Block 2-12
2.3.16 Disable Setup Mode 2-12
2.3.17 Disable All Testing 2-12
2.3.18 Enable Unibus Memory Test 2-12
2.3.19 Disable UBA ROM 2-12
2.3.20 Enable UBA Cache 2-12
2.3.21 Enable 18-Bit Mode 2-12
2.4 SYSTEM INSTALLATION 2-13
2.4.1 LSI-11 Based Systems 2-13
2.4.2 Restricted LSI-11 Systems 2-15
2.4.3 Unibus Based Systems 2-16
2.5 MODULE CONTACT FINGER IDENTIFICATION 2-18
2.6 MODULE INSTALLATION PROCEDURE 2-21
2.7 SPECIFICATIONS 2-21

CHAPTER 3 CONSOLE ON-LINE DEBUGGING TECHNIQUE (ODT)

3.1 INTRODUCTION 3-1
3.1.1 Terminal Interface 3-1
3.2 ODT OPERATION OF THE CONSOLE SERIAL LINE INTERFACE 3-2
3.2.1 Console ODT Input Sequence 3-2
3.2.2 Console ODT Output Sequence 3-2
3.3 CONSOLE ODT ENTRY CONDITIONS 3-2
3.4 CONSOLE ODT COMMAND SET 3-3
3.4.1 / (ASCII 057) - Slash 3-4
3.4.2 <CR> (ASCII 15) - Carriage Return 3-4
3.4.3 <LF> (ASCII 12) - Line Feed 3-5
3.4.4 $ (ASCII 044) or R (ASCII 122) - Internal Register Designator 3-5
3.4.5 S (ASCII 123) - Processor Status Word Designator 3-5
3.4.6 G (ASCII 107) - Go 3-6
3.4.7 P (ASCII 120) - Proceed 3-6
3.4.8 <CTRL> <SHIFT> S (ASCII 23) - Binary Dump 3-6
3.5 KDJ11-B ADDRESS SPECIFICATION 3-7
3.5.1 Processor I/O Addresses 3-7
3.5.2 Stack Pointer Selection 3-7
3.5.3 Entering Octal Digits 3-8
3.5.4 ODT Timeout 3-8
3.5.5 General Registers 3-8

CHAPTER 4 BOOT ROMS AND DIAGNOSTICS

4.1 INTRODUCTION 4-1
4.2 POWER-UP OR RESTART 4-1
4.2.1 Dialog (Mode 0) 4-1
4.2.2 Automatic (Mode 1) 4-2
4.2.3 ODT (Mode 2) 4-2
4.2.4 24/26 (Mode 3) 4-2
4.3 FORCED DIALOG MODE 4-2
4.4 HELP COMMAND 4-2
4.5 BOOT COMMAND 4-3
4.6 LIST COMMAND 4-4
4.7 SETUP MODE 4-5
4.7.1 Exit (1) 4-5
4.7.2 Select Configuration Parameters (2) 4-6
4.7.3 Select Bootstrap Translations (3) 4-8
4.7.4 Select Automatic Boot Sequence (4) 4-8
4.7.5 Select Console Message (5) 4-10
4.7.6 Define Switchpack Boot Selections (6) 4-11
4.7.7 List Available Bootstrap Programs (7) 4-12
4.7.8 Setup Table Initialization (8) 4-12
4.7.9 Save the Setup Table in the EEPROM (9) 4-12
4.7.10 Load EEPROM Data Into the Setup Table (10) 4-12
4.7.11 Delete a Boot Program From the EEPROM (11) 4-12
4.7.12 Load an EEPROM Boot Program Into Memory (12) 4-12
4.7.13 Edit or Create a Boot Program in the EEPROM (13) 4-12
4.7.14 Save a Boot Program in the EEPROM (14) 4-13
4.7.15 Enter ROM ODT (15) 4-14
4.8 MAP COMMAND 4-15
4.9 TEST COMMAND 4-16
4.10 DIAGNOSTIC TESTS 4-16
4.10.1 CPU or Halt Switch (Test 77) 4-18
4.10.2 CPU and MMU (Test 76) 4-18
4.10.3 Turn on MMU, Run CPU and MMU (Test 75) 4-18
4.10.4 Turn on PMI, Check UBA Reboot Bit (Test 74) 4-18
4.10.5 Power-Up to Mode 2: ODT (Test 73) 4-18
4.10.6 Power-Up to Mode 3: 24 (Test 72) 4-19
4.10.7 EEPROM Checksum (Test 71) 4-19
4.10.8 CPU ROM Checksum and PCR (Test 70) 4-19
4.10.9 Miscellaneous CPU and EIS (Test 67) 4-19
4.10.10 Console SLU Test 1 (Test 66) 4-19
4.10.11 Console SLU Test 2 (Test 65) 4-19
4.10.12 Console SLU Test 3 (Test 64) 4-19
4.10.13 MMU Aborts (Test 63) 4-20
4.10.14 Cache Memory (Test 62) 4-20
4.10.15 Line Clock (Test 61) 4-20
4.10.16 Floating-Point Instruction (Test 60) 4-20
4.10.17 Reserved (Test 57) 4-20
4.10.18 Exit Standalone Mode (Test 56) 4-20
4.10.19 UBA Register Response (Test 55) 4-20
4.10.20 Memory Sizing Routine (Test 54) 4-21
4.10.21 Memory Location 0 (Test 53) 4-21
4.10.22 Memory Locations 0 to 4K Words (Test 52) 4-21
4.10.23 Cache Operation With Memory (Test 51) 4-21
4.10.24 Complete Memory Data/Byte Exercise (Test 50) 4-21

4.10.25 Memory Parity/ECC (Test 47) 4-21
4.10.26 Memory Address Shorts (Test 46) 4-22
4.10.27 UBA Boot ROM (Test 45) 4-22
4.10.28 UBA Map Registers Data Path (Test 44) 4-22
4.10.29 UBA Unmapped Diagnostic Data (Test 43) 4-22
4.10.30 UBA Mapped Diagnostic Data (Test 42) 4-22
4.10.31 UBA Floating Address/Data (Test 41) 4-22
4.10.32 UBA Address Overflow (Test 40) 4-23
4.10.33 UBA Cache Data (Test 37) 4-23
4.10.34 UBA Cache LRU (Test 36) 4-23
4.10.35 UBA Cache Tag Store (Test 35) 4-23
4.10.36 UBA Cache Parity Error (Test 34) 4-23
4.10.37 Unibus Memory Data/Byte Exercise (Test 33) 4-23
4.10.38 Unibus Memory Parity (Test 32) 4-23
4.10.39 Unibus Memory Address Shorts (Test 31) 4-23
4.10.40 Exit (Test 30) 4-23
4.11 DIAGNOSTIC TEST ERROR MESSAGES 4-24
4.11.1 Test Number 4-24
4.11.2 Address of the Error 4-24
4.11.3 Register Set 1 4-24
4.11.4 Optional User Commands 4-24
4.11.4.1 Rerun Test 4-24
4.11.4.2 Loop on Test 4-24
4.11.4.3 Map Memory and I/O Page 4-25
4.11.4.4 Advance to the Next Test 4-25
4.11.5 Typical Displays 4-25
4.12 ROM CODE BOOT PROGRAMS 4-26
4.12.1 Error Messages for Bootstrap Programs 4-28
4.12.2 LSI Bus Selected Error Messages 4-28
4.13 MESSAGE DISPLAY CONSTRAINTS 4-29

CHAPTER 5 FUNCTIONAL THEORY

5.1 INTRODUCTION 5-1
5.2 DCJ11-A MICROPROCESSOR 5-2
5.2.1 Initialization 5-2
5.2.2 Output Signals 5-2
5.2.2.1 Address Input/Output (MAIO <3:0> H) 5-2
5.2.2.2 Bank Select (MBS1 H, MBS0 H) 5-4
5.2.2.3 Address Latch Enable (MALE L) 5-4
5.2.2.4 Stretch Control (MSCTL L) 5-4
5.2.2.5 Strobe (MSTRB L) 5-4
5.2.2.6 Buffer Control (MBUFCTL L) 5-4
5.2.2.7 Predecode Strobe (MPRDC L) 5-5
5.2.2.8 I/O Map Enable (JMAP L) 5-5
5.2.2.9 Clock (MCLK H) 5-5
5.2.3 Input Signals 5-5
5.2.3.1 MMISS L 5-5
5.2.3.2 Data Valid (MDV L) 5-5
5.2.3.3 Continue (MCONT L) 5-5
5.2.3.4 DMA Request (MDMR L) 5-5
5.2.3.5 MIRQ <7:4> H 5-5
5.2.3.6 MHALT H 5-5
5.2.3.7 MEVNT H 5-5

5.2.3.8 MPWR FAIL L ... 5-5
5.2.3.9 MPARITY L ... 5-5
5.2.3.10 MABORT L ... 5-6
5.2.3.11 FPA FPE L ... 5-6
5.2.4 MDAL <21:01> ... 5-6
5.2.5 DCJ11-A Transactions ... 5-6
5.2.5.1 NOP ... 5-6
5.2.5.2 Bus Read ... 5-7
5.2.5.3 Bus Write ... 5-8
5.2.5.4 General Purpose Read ... 5-8
5.2.5.5 General Purpose Write ... 5-9
5.2.5.6 IACK ... 5-10
5.3 BUS ARBITRATOR ... 5-10
5.3.1 PMI Cycle Request ... 5-12
5.4 DATA PATH CONTROLLER ... 5-12
5.4.1 Cycle Encoder ... 5-12
5.4.2 Oscillator ... 5-15
5.4.3 Next Address MUX ... 5-17
5.4.3.1 Default ... 5-17
5.4.3.2 External Read/Write ... 5-17
5.4.3.3 LSI/Unibus ... 5-18
5.4.3.4 Interrupt Vector ... 5-18
5.4.3.5 DC350/394 Accesses ... 5-18
5.4.3.6 Byte Allocation ... 5-18
5.4.3.7 DMA Monitor ... 5-18
5.4.3.8 Standalone Mode ... 5-18
5.4.4 Control Store ... 5-18
5.5 CACHE MEMORY AND DMA STORE ... 5-22
5.5.1 Cache Memory ... 5-24
5.5.2 Cache Tag Store ... 5-24
5.5.3 Cache Data Parity Logic ... 5-25
5.5.4 Valid Tag Bit ... 5-26
5.5.5 DMA Tag Store ... 5-26
5.5.6 Cache Control ... 5-26
5.6 DC350/394 GATE ARRAY ... 5-28
5.6.1 A-Multiplexer ... 5-30
5.6.2 Cache Data Path ... 5-30
5.6.3 Parity Interrupt and Abort ... 5-31
5.6.4 Address Decode ... 5-32
5.6.5 Cycle Decoder ... 5-33
5.6.6 Miscellaneous ... 5-33
5.7 DC351 GATE ARRAY ... 5-34
5.7.1 DMA Tag Data Path ... 5-35
5.7.2 Clock Start Logic ... 5-35
5.7.3 Flush Counter ... 5-35
5.7.4 Main Memory Parity Error ... 5-35
5.8 TIMEOUT ... 5-36
5.8.1 DMA Requests ... 5-36
5.8.2 NXM or Interrupt Requests ... 5-36
5.9 BUS DISTRIBUTION ... 5-37
5.9.1 Internal Bus Control ... 5-37
5.9.2 LSI-11 Bus Control ... 5-38
5.9.3 PMI Bus Control ... 5-38
5.10 CONSOLE SERIAL LINE UNIT ... 5-38
5.10.1 Halt-on-Break ... 5-41

5.10.2 Console Interrupt Arbitration 5-41
5.11 CONFIGURATION AND DISPLAY 5-42
5.12 BOOT AND DIAGNOSTIC ROMS 5-42
5.13 CONFIGURATION EEPROM 5-43
5.14 FLOATING-POINT ACCELERATOR 5-44
5.14.1 FPA Operation 5-44

CHAPTER 6 EXTENDED LSI-11 BUS

6.1 INTRODUCTION 6-1
6.2 BUS SIGNAL NOMENCLATURE 6-2
6.3 DATA TRANSFER BUS CYCLES 6-3
6.3.1 Bus Cycle Protocol 6-4
6.3.1.1 Device Addressing 6-4
6.3.1.2 DATI 6-5
6.3.1.3 DATO(B) 6-7
6.3.1.4 DATIO(B) 6-10
6.4 DIRECT MEMORY ACCESS 6-12
6.5 INTERRUPTS 6-14
6.5.1 Device Priority 6-14
6.5.2 Interrupt Protocol 6-15
6.5.3 4-Level Interrupt Configurations 6-18
6.6 CONTROL FUNCTIONS 6-19
6.6.1 Memory Refresh 6-19
6.6.2 Halt 6-19
6.6.3 Initialization 6-19
6.6.4 Power Status 6-19
6.6.4.1 BDCOK H 6-19
6.6.4.2 BPOK H 6-19
6.6.4.3 Power-Up 6-20
6.6.4.4 Power-Down 6-20
6.6.5 BEVNT L 6-21
6.7 BUS ELECTRICAL CHARACTERISTICS 6-21
6.7.1 Signal Level Specification 6-21
6.7.2 AC Bus Load Definition 6-21
6.7.3 DC Bus Load Definition 6-21
6.7.4 120 Ω LSI-11 Bus 6-21
6.7.5 Bus Drivers 6-22
6.7.6 Bus Receivers 6-22
6.7.7 KDJ11-B Bus Termination 6-23
6.7.7.1 Bus Interconnection Wiring 6-23
6.7.7.2 Backplane Wiring 6-23
6.7.7.3 Intrabackplane Bus Wiring 6-24
6.7.7.4 Power and Ground 6-24
6.7.7.5 Maintenance and Spare Pins 6-24
6.8 SYSTEM CONFIGURATIONS 6-24
6.8.1 Rules for Configuring Single-Backplane Systems 6-25
6.8.2 Rules for Configuring Multiple-Backplane Systems 6-26
6.8.3 Power Supply Loading 6-27

CHAPTER 7 PRIVATE MEMORY INTERCONNECT BUS

7.1 DESCRIPTION 7-1
7.2 PMI INTERFACE 7-1
7.2.1 PMI Bus Master Signals 7-1
7.2.2 PMI Slave Signals 7-1
7.2.3 PMI Unibus Adapter Signals 7-1
7.2.4 LSI Bus Signals 7-1
7.3 PMI OPERATION IN AN LSI-11 SYSTEM 7-6
7.4 PMI OPERATION IN A UNIBUS SYSTEM 7-6
7.4.1 Bus Device NPR or DMA 7-6
7.4.2 PMI Bus Device Interrupt 7-8
7.5 PMI DATA TRANSFERS 7-9
7.5.1 PMI Data In/Data In Pause 7-9
7.5.2 PMI Block Data In 7-11
7.5.3 PMI Data Out/Data Out Byte 7-13
7.6 PMI INTERRUPT PROTOCOL 7-15
7.7 PMI POWER-UP/POWER-DOWN 7-15

CHAPTER 8 ADDRESSING MODES

8.1 INTRODUCTION 8-1
8.2 ADDRESSING MODES 8-1
8.2.1 Single-Operand Addressing 8-3
8.2.2 Double-Operand Addressing 8-3
8.2.3 Direct Addressing 8-4
8.2.3.1 Register Mode 8-6
8.2.3.2 Autoincrement Mode [OPR (Rn)+] 8-8
8.2.3.3 Autodecrement Mode [OPR −(Rn)] 8-9
8.2.3.4 Index Mode [OPR X(Rn)] 8-11
8.2.4 Deferred (Indirect) Addressing 8-12
8.2.5 Use of the PC as a General Purpose Register 8-16
8.2.5.1 Immediate Mode [OPR #n,DD] 8-16
8.2.5.2 Absolute Mode [OPR @#A] 8-17
8.2.5.3 Relative Addressing Mode [OPR A or OPR X(PC)] 8-18
8.2.5.4 Relative-Deferred Addressing Mode [OPR @A or OPR @X(PC)] 8-19
8.2.6 Use of the General Purpose Registers as a Stack Pointer 8-19

CHAPTER 9 BASE INSTRUCTION SET

9.1 INSTRUCTION SET 9-1
9.2 INSTRUCTION FORMATS 9-4
9.3 BYTE INSTRUCTIONS 9-7
9.4 LIST OF INSTRUCTIONS 9-8
9.5 SINGLE-OPERAND INSTRUCTIONS 9-12
9.5.1 General 9-12
9.5.2 Shifts and Rotates 9-17
9.5.3 Multiple-Precision 9-22
9.5.4 PSW Operators 9-24
9.6 DOUBLE-OPERAND INSTRUCTIONS 9-26
9.6.1 General 9-26
9.6.2 Logical 9-32

9.7 PROGRAM CONTROL INSTRUCTIONS ... 9-34
9.7.1 Branches ... 9-34
9.7.2 Signed Conditional Branches ... 9-39
9.7.3 Unsigned Conditional Branches ... 9-41
9.7.4 Jump and Subroutine Instructions ... 9-43
9.7.5 Traps ... 9-47
9.7.6 Miscellaneous Program Controls ... 9-51
9.7.7 Reserved Instruction Traps ... 9-54
9.7.8 Trace Trap ... 9-54
9.8 MISCELLANEOUS INSTRUCTIONS ... 9-55
9.9 CONDITION CODE OPERATORS ... 9-59

CHAPTER 10 FLOATING-POINT ARITHMETIC

10.1 INTRODUCTION ... 10-1
10.2 FLOATING-POINT DATA FORMATS ... 10-1
10.2.1 Nonvanishing Floating-Point Numbers ... 10-1
10.2.2 Floating-Point Zero ... 10-1
10.2.3 Undefined Variables ... 10-2
10.2.4 Floating-Point Data ... 10-2
10.3 FLOATING-POINT STATUS REGISTER (FPS) ... 10-2
10.4 FLOATING EXCEPTION CODE AND ADDRESS REGISTERS ... 10-6
10.5 FLOATING-POINT INSTRUCTION ADDRESSING ... 10-7
10.6 ACCURACY ... 10-8
10.7 FLOATING-POINT INSTRUCTIONS ... 10-9

CHAPTER 11 PROGRAMMING TECHNIQUES

11.1 INTRODUCTION ... 11-1
11.2 POSITION-INDEPENDENT CODE ... 11-1
11.2.1 Use of Addressing Modes in the Construction of Position-Independent Code ... 11-1
11.2.2 Comparison of Position-Dependent and Position-Independent Code ... 11-3
11.3 STACKS ... 11-5
11.3.1 Pushing onto a Stack ... 11-5
11.3.2 Popping from a Stack ... 11-6
11.3.3 Deleting Items from a Stack ... 11-6
11.3.4 Stack Uses ... 11-7
11.3.5 Stack Use Examples ... 11-8
11.3.6 Subroutine Linkage ... 11-9
11.3.6.1 Return from a Subroutine ... 11-10
11.3.6.2 Subroutine Advantages ... 11-10
11.3.7 Interrupts ... 11-10
11.3.7.1 Interrupt Service Routines ... 11-10
11.3.7.2 Nesting ... 11-11
11.3.8 Reentrancy ... 11-12
11.3.8.1 Reentrant Code ... 11-12
11.3.8.2 Writing Reentrant Code ... 11-13
11.3.9 Coroutines ... 11-13
11.3.9.1 Coroutine Calls ... 11-13
11.3.9.2 Coroutines Versus Subroutines ... 11-14
11.3.9.3 Using Coroutines ... 11-15
11.3.10 Recursion ... 11-17
11.3.11 Processor Traps ... 11-19

11.3.11.1 Trap Instructions 11-20
11.3.11.2 Use of Macro Calls 11-20
11.3.12 Conversion Routines 11-21
11.4 PROGRAMMING THE PROCESSOR STATUS WORD 11-25
11.5 PROGRAMMING PERIPHERALS 11-25
11.6 PDP-11 PROGRAMMING EXAMPLES 11-26
11.7 LOOPING TECHNIQUES 11-32

APPENDIX A ROM CODE DIFFERENCES

A.1 GENERAL A-1
A.2 V6.0 AND V7.0 DIFFERENCES A-1
A.2.1 Boot Support for Tape MSCP Devices (TK50/TU81) A-1
A.2.2 Disk MSCP Automatic Boot Routine A-1
A.2.3 Dialog Mode Boot Command for Disk MSCP Boot A-2
A.2.4 Disk MSCP Boot (DU) A-2
A.2.5 8-Unit Restriction for MSCP Automatic Boot A-2
A.2.6 Irregular Monitoring of Keyboard During Automatic Boot Sequence A-2
A.2.7 Addition of Single-Letter Mnemonic in Automatic Boot List A-2
A.2.8 Setup Mode Disable A-3
A.2.9 Disable All Testing Parameter A-3
A.2.10 Edit/Create Command A-3
A.2.11 Initialize Command for the PMG Counter A-4
A.2.12 PMG Parameter Warning A-4
A.2.13 Setup Command 4 Printout A-4
A.2.14 MU (TK50/TU81) Device A-4
A.2.15 Setup Command 5 ..... A-5
A.2.16 Memory Initialization at Power-Up A-5
A.2.17 Power-Up Option Set to 3 with Battery Backed Up Memory A-6
A.2.18 Halt-on-Break A-6
A.2.19 Local Language Support A-6
A.2.20 Addition of Map Command Feature A-6
A.2.21 EEPROM Load Error Before Dialog Mode A-6
A.2.22 Test Command Decimal Numbers A-6
A.2.23 Test Command Execution of a Single Test A-7
A.2.24 Test Errors in Tests 72 to 75 ..... A-7
A.2.25 Bypass Errors for Test Failures A-7
A.2.26 Test 76 and 75 Error Messages A-7
A.2.27 Starting Automatic Boot Sequence Message A-7
A.2.28 Device Name and Number After Message A-8
A.2.29 Incorrect Message for Invalid Unit Number A-8
A.2.30 Dialog Mode Header Message A-8
A.2.31 Map Command Message A-8
A.2.32 List Device Descriptions A-8
A.2.33 Loss of the First Line in a List Header A-8
A.2.34 <CTRL> R and <CTRL> U Echo A-8
A.2.35 Power-Up or Restart Mode Set to 3 (LSI Bus Only) A-10
A.2.36 Automatic Boot Misleading Error Message (LSI Bus Only) A-10
A.2.37 APT Halt-on-Break Detect (LSI Bus Only) A-10
A.2.38 B Mnemonic for ROM Boots (Unibus Only) A-10
A.2.39 Error in List Command When Unknown ROM is Found (Unibus Only) A-10
A.2.40 Power-Up or Restart Mode Set to 3 (Unibus Only) A-10
A.2.41 Saving KMCR Bits <5:0> in the EEPROM (Unibus Only) A-11

A.3 V7.0 AND V8.0 DIFFERENCES A-11
A.3.1 M9312 MultiROM Bootstrap Support (PDP-11/84 Only) A-11
A.3.2 Small Memory Automatic Boot Problem for RQDX3 ..... A-11
A.3.3 RA**nn** Disk Spinup Time Delay for Automatic Boot A-11
A.3.4 Addition of RESET Instruction at Beginning of Code A-12
A.3.5 Addition of New Setup Command 5 ..... A-12
A.3.6 Memory Test Coverage Problem A-12
A.3.7 List Command Device Descriptions A-12
A.3.8 Manufacturing Test Loop Problem A-12

APPENDIX B SETUP PARAMETER WORKSHEETS

B.1 PURPOSE B-1
B.2 FUNCTION B-1

APPENDIX C MNEMONICS

FIGURES

1-1 Programming Model 1-2
1-2 Processor Status Word Register 1-4
1-3 CPU Error Register 1-6
1-4 Program Interrupt Request Register 1-7
1-5 16-Bit Mapping 1-11
1-6 18-Bit Mapping 1-12
1-7 22-Bit Mapping 1-12
1-8 Virtual Address Mapping into Physical Address 1-13
1-9 Interpretation of a Virtual Address 1-14
1-10 Displacement Field of a Virtual Address 1-14
1-11 Construction of a Physical Address 1-15
1-12 Active Page Register 1-16
1-13 Page Address Register 1-18
1-14 Page Descriptor Register 1-19
1-15 Memory Management Register 0 (MMR0) 1-20
1-16 Memory Management Register 1 (MMR1) 1-21
1-17 Memory Management Register 3 (MMR3) 1-22
1-18 Typical Memory Page 1-24
1-19 Nonconsecutive Memory Pages 1-25
1-20 Typical Stack Memory Page 1-26
1-21 Cache Physical Address 1-27
1-22 Cache Data Format 1-27
1-23 Cache Control Register (CCR) 1-30
1-24 Hit/Miss Register (HMR) 1-32
1-25 Memory System Error Register (MSER) 1-32
1-26 Receiver Control/Status Register 1-36
1-27 Receiver Buffer Register 1-37
1-28 Transmitter Control/Status Register 1-38
1-29 Transmitter Buffer Register 1-39

1-30 Controller Status Register 1-40
1-31 Configuration and Display Register 1-43
1-32 Maintenance Register 1-43
1-33 Line Time Clock Register 1-45
2-1 KDJ11-B Module Layout 2-2
2-2 Pin Assignments for Connectors J2 and J3 2-3
2-3 KDJ11-B Module Contacts 2-18
4-1 Help Commands 4-2
4-2 Booting an RL01/RL02 4-3
4-3 Available Boot Programs 4-5
4-4 Typical Translation Table 4-9
4-5 Automatic Boot Sequence Example 4-9
4-6 Select Console Message Example 4-10
4-7 Switchpack Boot Selection 4-11
4-8 Edit/Create an EEPROM Boot 4-13
4-9 Typical Map Mode Display 4-15
4-10 Continuous Testing Display 4-16
4-11 Loop Test Display 4-16
4-12 Typical Diagnostic Error Display 4-25
4-13 Typical Memory Test Error Display 4-26
4-14 Typical Unexpected Trap Error Display 4-26
4-15 General Bootstrap Error Messages 4-28
4-16 User Friendly Error Message 4-28
5-1 KDJ11-B Functional Block Diagram 5-1
5-2 DCJ11-A Microprocessor Logic 5-3
5-3 NOP Transaction 5-6
5-4 Stretched NOP Transaction 5-6
5-5 Bus Read Transaction 5-7
5-6 Stretched Bus Read Transaction 5-7
5-7 Bus Write Transaction 5-8
5-8 General Purpose Read Transaction 5-9
5-9 General Purpose Write Transaction 5-9
5-10 Interrupt Acknowledge Transaction 5-10
5-11 Bus Arbitrator 5-11
5-12 PMI Cycle Request 5-12
5-13 Data Path Controller 5-13
5-14 Cycle Encoder 5-13
5-15 Oscillator Outputs 5-15
5-16 Oscillator Control 5-16
5-17 Next Address Multiplexer 5-17
5-18 Control Store 5-19
5-19 Internal Bus Control Signals 5-22
5-20 Cache Memory System 5-23
5-21 Cache Physical Address 5-23
5-22 Cache Data Format 5-23
5-23 Cache Memory 5-24
5-24 Cache Tag Store 5-24
5-25 Cache Data Parity Logic 5-25
5-26 Valid Tag Bit 5-27
5-27 DMA Tag Store 5-27
5-28 Cache Control Signals 5-27
5-29 DC350/394 Gate Array 5-28
5-30 DC351 Gate Array 5-34
5-31 NXM/Interrupt Timeout Logic 5-36
5-32 Internal Bus Control 5-37

5-33 LSI-11 Bus Control Signals 5-38
5-34 PMI Bus Control Signals 5-39
5-35 Console Serial Line Logic 5-39
5-36 Console Interrupt Arbitration 5-41
5-37 Configuration and Display Circuits 5-42
5-38 Boot and Diagnostic ROM Logic 5-43
5-39 Configuration EEPROM Logic 5-43
5-40 Floating-Point Accelerator 5-44
6-1 DATI Bus Cycle 6-5
6-2 DATI Bus Cycle Timing 6-6
6-3 DATO or DATO(B) Bus Cycle 6-8
6-4 DATO or DATO(B) Bus Cycle Timing 6-9
6-5 DATIO or DATIO(B) Bus Cycle 6-10
6-6 DATIO or DATIO(B) Bus Cycle Timing 6-11
6-7 DMA Request/Grant Sequence 6-12
6-8 DMA Request/Grant Bus Cycle Timing 6-13
6-9 Interrupt Request/Acknowledge Sequence 6-15
6-10 Interrupt Protocol Timing 6-16
6-11 Position-Independent Configuration 6-18
6-12 Position-Dependent Configuration 6-18
6-13 Power-Up/Power-Down Timing 6-20
6-14 Bus Line Termination 6-23
6-15 Single-Backplane Configuration 6-25
6-16 Multiple-Backplane Configuration 6-26
8-1 Single-Operand Addressing 8-3
8-2 Double-Operand Addressing 8-3
8-3 Mode 0, Register 8-4
8-4 Mode 2, Autoincrement 8-5
8-5 Mode 4, Autodecrement 8-5
8-6 Mode 6, Index 8-5
8-7 INC R3 Increment 8-6
8-8 ADD R2,R4 Add 8-7
8-9 COMB R4 Complement Byte 8-7
8-10 CLR (R5)+ Clear 8-8
8-11 CLRB (R5)+ Clear Byte 8-8
8-12 ADD (R2)+,R4 Add 8-9
8-13 INC –(R0) Increment 8-9
8-14 INCB –(R0) Increment Byte 8-10
8-15 ADD –(R3),R0 Add 8-10
8-16 CLR 200(R4) Clear 8-11
8-17 COMB 200(R1) Complement Byte 8-11
8-18 ADD 30(R2),20(R5) Add 8-12
8-19 Mode 1, Register-Deferred 8-12
8-20 Mode 3, Autoincrement-Deferred 8-13
8-21 Mode 5, Autodecrement-Deferred 8-13
8-22 Mode 7, Index-Deferred 8-13
8-23 CLR @R5 Clear 8-14
8-24 INC @(R2)+ Increment 8-14
8-25 COM @–(R0) Complement 8-15
8-26 ADD @1000(R2),R1 Add 8-15
8-27 ADD #10,R0 Add 8-17
8-28 CLR @#1100 Clear 8-17
8-29 ADD @#2000 Add 8-18
8-30 INC A Increment 8-18
8-31 CLR @A Clear 8-19

9-1 Single-Operand Group 9-4
9-2 Double-Operand Group 1 9-4
9-3 Double-Operand Group 2 9-4
9-4 Program Control Group Branch 9-4
9-5 Program Control Group JMP 9-5
9-6 Program Control Group JSR 9-5
9-7 Program Control Group RTS 9-5
9-8 Program Control Group Traps 9-5
9-9 Program Control Group Subtract 9-5
9-10 Mark 9-6
9-11 Call to Supervisor Mode 9-6
9-12 Set Priority Level 9-6
9-13 Operate Group 9-6
9-14 Condition Group 9-6
9-15 Move To and From Previous Instruction/Data Space Group 9-7
9-16 Byte Instructions 9-7
10-1 Single-Precision Format 10-2
10-2 Double-Precision Format 10-3
10-3 2's Complement Format 10-3
10-4 Floating-Point Status Register 10-3
10-5 Floating-Point Addressing Modes 10-9
11-1 Word and Byte Stacks 11-5
11-2 Push and Pop Operations 11-6
11-3 Byte Stack Used as a Character Buffer 11-9
11-4 JSR Stack Condition Example 11-9
11-5 Nested Interrupt Service Routines and Subroutines 11-11
11-6 Reentrant Routines 11-12
11-7 Sharing Control of a Routine 11-13
11-8 Coroutine Example 11-14
11-9 Coroutines Versus Subroutines 11-15
11-10 Coroutine Path 11-16
11-11 Coroutine Interaction 11-16
11-12 Recursive Routine Flow 11-17
A-1 Program for Continuous Loop A-3
A-2 PMG Count Value Warning Message A-4
A-3 Single-Letter Description for Command 4 ..... A-5
A-4 Automatic Boot Sequence Message A-7
A-5 V6.0 Incorrect Message A-8
A-6 V7.0 Correct Error Message A-9
A-7 V6.0 List Header Error A-9
A-8 V7.0 Correct List Header A-9

TABLES

1-1 General Purpose Registers ... 1-3
1-2 Stack Pointer (PSW <15:14> or <13:12>) ... 1-4
1-3 Processor Status Word Bit Description ... 1-5
1-4 CPU Error Register Bit Description ... 1-6
1-5 Program Interrupt Request Bit Description ... 1-7
1-6 KDJ11-B Interrupts ... 1-8
1-7 KDJ11-B Compatibility ... 1-13
1-8 Memory Management Register Addresses ... 1-17
1-9 Page Descriptor Register Bit Description ... 1-19
1-10 MMR0 Bit Description ... 1-21
1-11 MMR3 Bit Description ... 1-22
1-12 Cache Response Matrix ... 1-28
1-13 Cache Parity Errors ... 1-29
1-14 Cache Control Register Description ... 1-31
1-15 Memory System Error Register Description ... 1-33
1-16 Baud Rate Selection ... 1-35
1-17 RCSR Bit Description ... 1-36
1-18 RBUF Bit Description ... 1-37
1-19 XCSR Bit Description ... 1-38
1-20 XBUF Bit Description ... 1-39
1-21 Control/Status Register Bit Description ... 1-40
1-22 Maintenance Register Bit Description ... 1-44
1-23 Line Time Clock Register Bit Description ... 1-45
2-1 Jumper Wire Functions ... 2-2
2-2 J2 and J3 Connectors ... 2-4
2-3 Baud Rate Selections ... 2-4
2-4 Bootstrap Program Selection ... 2-5
2-5 Diagnostic and System Status LED Display ... 2-6
2-6 Configuration Parameters ... 2-8
2-7 LSI-11 Compatible Options ... 2-13
2-8 Unibus Compatible Options ... 2-16
2-9 KDJ11-B Module and LSI-11 Bus Signals ... 2-19
2-10 Module PMI Signal Assignments ... 2-20
3-1 Console ODT Commands ... 3-3
4-1 Setup Mode Commands ... 4-6
4-2 Configuration Parameters ... 4-6
4-3 Switchpack Selections ... 4-11
4-4 ROM ODT Commands ... 4-14
4-5 Diagnostic LED Displays ... 4-17
4-6 Bootstrap Error LED Displays ... 4-27
5-1 MAIO Coding ... 5-3
5-2 Bank Select Address Codes ... 5-4
5-3 General Purpose Read Codes ... 5-8
5-4 General Purpose Write Codes ... 5-9
5-5 Control Signals ... 5-11
5-6 Cycle Encoder Status ... 5-14
5-7 Transactions Selected by LCYCCD Outputs ... 5-14
5-8 Oscillator Control Signals ... 5-16
5-9 Selection of NA <1:0> Status ... 5-17
5-10 Control Store Outputs ... 5-20
5-11 Cache Parity ... 5-26

5-12 LTC Interrupts 5-29
5-13 AMUX Selections 5-30
5-14 Parity Interrupt and Abort Logic 5-31
5.15 CCR Register Selections 5-31
5-16 Address Decoding 5-32
5-17 DEVCD Outputs 5-32
5-18 Cycle Decoding 5-33
5-19 Register Selection 5-40
5-20 Baud Rate Selections 5-40
6-1 Summary of Signal Line Functions 6-1
6-2 Data Transfer Bus Cycles 6-3
6-3 Data Transfer Bus Signals 6-4
6-4 Position-Independent, Multilevel Device Requirements 6-17
7-1 PMI Bus Master Signals 7-2
7-2 PMI Slave Signals 7-3
7-3 PMI Unibus Adapter Signals 7-4
7-4 LSI Bus Signals 7-5
8-1 Sample KDJ11-B Instructions 8-4
9-1 Instruction Set 9-1
10-1 FPS Register Bit Description 10-4
A-1 ROM Part Numbers A-1
A-2 Setup Command 4 Automatic Boot Lists A-5
A-3 ROM Code Test Selections A-6
A-4 New List Command Device Descriptions A-13

PREFACE

This user's guide contains descriptions of the KDJ11-B CPU module architecture, configuration, system requirements and programming. The module architecture is described in Chapter 1 and is supported by the description of functional theory in Chapter 5. The configuration requirements are described in Chapter 2 and are selected from the preprogrammed ROMs described in Chapter 4. These ROMs also contain the bootstrap programs and diagnostic testing for the module. Additional testing can be accomplished by using the ODT techniques found in Chapter 3.

The system requirements for the extended LSI-11 bus (Q22-Bus) are covered in Chapter 6 and the private memory interconnect bus operation is described in Chapter 7.

The base instruction set is described in Chapter 9 and the floating-point instruction set in Chapter 10. The addressing modes are covered in Chapter 8 and some of the programming techniques for the instruction sets are given in Chapter 11.

Appendix A details the changes incorporated in V7.0 and V8.0 of the ROM code. Appendix B provides worksheets to record the configuration being used for a system. Appendix C lists the mnemonics frequently used in this guide.

CHAPTER 1
ARCHITECTURE

1.1 DESCRIPTION

The KDJ11-B is a quad-height processor module for LSI-11 bus systems. It is designed for use in high speed, real-time applications and for multiuser, multitasking environments. The module can also function as a CPU in PDP-11 Unibus systems, when it is used in conjunction with the KTJ11-B Unibus adapter module.

The module interfaces to the standard 22-bit LSI bus and has the additional control signals necessary for communication via the Private Memory Interconnect (PMI). The PMI protocol uses the C/D interconnect bus and allows high speed data transfers across the bus, including double word reads. The LSI bus can address up to 4 Mbytes of main memory. Block mode Direct Memory Access (DMA) transfers – allowed on the extended bus, are supported by the KDJ11-B module. The MSV11-J memory module and the KTJ11-B Unibus adapter module are compatible with the PMI protocol.

The KDJ11-B module executes the complete PDP-11/70 base instruction set, including the Extended Instruction Set (EIS) and the MTPS, MFPS, MFPT, CSM, TSTSET, and WRTLCK instructions. It also supports the FP11 floating-point instruction set that is compatible with FP11-A, -C, -E, and -F floating-point processors. Full 22-bit memory management is provided for both instruction references and data references in three protection modes: kernel, supervisor, and user.

The three protection modes provide the ability to implement layered software protection. Memory management separately manages the three modes, allowing each one to access different sections of main memory. Furthermore, each section can have different access protection rights. Each mode uses a separate system stack pointer that offers an additional degree of isolation. The protection modes are organized so that a higher protection mode can always enter a lower protection mode, while a lower protection mode can never accidentally enter a higher protection mode. Kernel mode has full privileges and can execute all instructions. Supervisor mode and user mode, the two lower privileged modes, cannot execute certain instructions.

The module uses a DCJ11-A microprocessor chip as a central processor having memory management and floating-point processing capability. It also has an 8-Kbyte cache memory, a line time clock, a console serial line unit, and a boot facility with diagnostics.

The 8-Kbyte write-through direct map cache has a dual tag store that allows concurrent operations of the CPU and DMA. The cache is transparent to user programs and acts as a high speed buffer between the processor and main memory. The data stored in the cache represents the most active portion of the main memory in use. The processor accesses main memory only when data is not available in the cache.

The full-duplex console serial line unit provides an interface for the console terminal. The unit is a DC319 Digital Link Asynchronous Receiver/Transmitter (DLART) that partially supports the RS-423, and fully supports the RS-232-C EIA standards.

The KDJ11-B module supports console emulation (micro-ODT). This allows users to interrogate and write main memory and CPU registers as if a console switch panel and display lights were available.

The boot and diagnostic facility features two sockets for Read-Only Memory (ROM) chips that contain the boot and diagnostic programs. It also has a third socket for an Electrically Erasable Programmable ROM (EEPROM) chip that contains configuration data and space for the user's loadable boot code. The operation of the boot and diagnostic facility is described in detail in Chapter 4.

The KDJ11-BB and -BF modules provide sockets for the installation of the optional FPJ11 Floating-Point Accelerator (FPA) chip. This is a coprocessor that significantly improves the execution speed of floating-point instructions. The KDJ11-BC (M8190-00) version of the module cannot use the FPJ11 optional FPA chip.

Self-diagnostic display LEDs are provided on the KDJ11-B module. They indicate the status of the module and system when the module is powered up. The LEDs aid in troubleshooting module failures.

The user-visible registers are shown in Figure 1-1 and are classified as general purpose, system control, memory system, floating-point and Memory Management Registers (MMRs).

1.2 DCJ11-A FEATURES

The DCJ11-A microprocessor operates in three modes: kernel, supervisor, and user. A program operating in the kernel mode has complete control of the system and incorporates protection mechanisms against any external interferences. Programs operating in the supervisor and user modes can be inhibited from executing certain instructions and can be denied direct access to the system peripherals. This feature is used to provide complete executive protection in a multiprogram environment.

There are 16 general purpose registers, as listed in Table 1-1, but only 8 are visible to the user at any given time. The general purpose registers provide a Stack Pointer (SP) for each of the three operating modes and a Program Counter (PC). The remaining 12 registers are divided into two groups of general purpose registers, R0–R5 and R0'–R5'. All of these registers can be used as accumulators, deferred addresses, index references, autoincrement, autodecrement, and stack pointers.

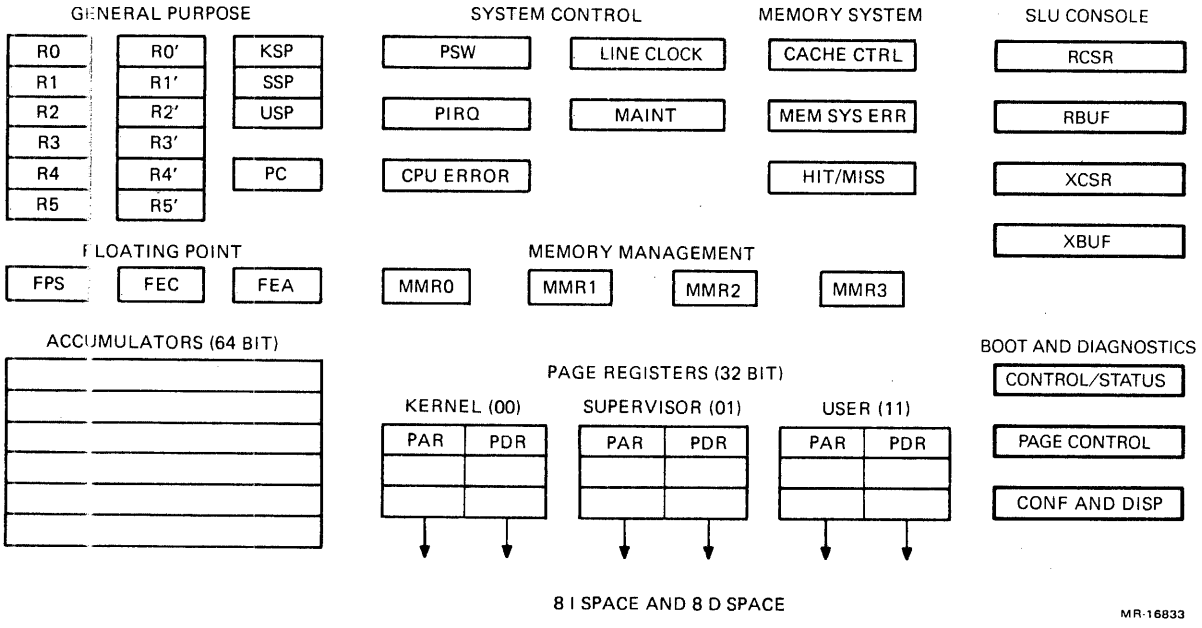


Figure 1-1 Programming Model

Table 1-1 General Purpose Registers

Register Number	Designation		
0	R0	R0'	-
1	R1	R1'	-
2	R2	R2'	-
3	R3	R3'	-
4	R4	R4'	-
5	R5	R5'	-
6	KSP	SSP	USP
7	PC	-	-

The system control registers are the Processor Status Word (PSW), the Program Interrupt ReQuest (PIRQ), and the CPU error register.

1.2.1 Stack Limit Protection

The DCJ11 monitors the kernel stack references against the fixed limit of 400. A yellow stack trap occurs at the end of the current instruction when the address of the stack reference is less than 400. A yellow stack trap can only occur in the kernel mode during a stack reference. This is defined as a mode 4 or 5 reference through R6, a JSR trap, or an interrupt stack push.

The microprocessor also checks for kernel stack aborts during interrupts, traps, and abort sequences. When a kernel stack push causes an abort during one of these conditions, a red stack trap occurs. This type of stack trap sets bit 2 in the CPU error register and loads virtual address 4 into the kernel stack pointer (R6). A trap through location 4 in the kernel space now occurs and the old PC and PSW are saved in locations 0 and 2, respectively, of the kernel space.

1.2.2 Kernel Protection

The following mechanisms are used to protect the kernel operating system against external interference.

- In the kernel mode, the HALT, RESET, and SPL instructions are executed as specified. In the supervisor or user modes, the HALT instruction causes a trap through location 4, but the RESET and SPL instructions are treated as NOPs.

- In the kernel mode, the RTI and RTT instructions can freely change bits <15:11> and <7:5> of the PSW register. In the supervisor or user modes, these instructions can only change bits <15:11> of the PSW register.

- In the kernel mode, the MTPS instruction can change bits <7:5> of the PSW register. In the supervisor or user modes, the MTPS instruction cannot change bits <7:5> of the PSW register.

- All the trap and interrupt vector addresses are classified as kernel space addresses, no matter what memory management mode the system is using or the contents of the PSW at the time the interrupt or trap occurs.

- The kernel stack references are checked for stack overflow, but the supervisor and user stack references are not checked.

1.2.3 General Registers

There are two groups of six registers designated R0–R5 and R0'–R5'. The group currently in use is selected by bit 11 in the PSW. When bit 11 is set (1), the R0'–R5' group is selected, and when bit 11 is cleared (0), the R0–R5 group is selected.

1.2.4 Stack Pointer

Register six (R6) is designated as the system stack pointer. There are three stack pointers available, one for each corresponding protection mode. However, only one is visible to the user at a given time. Processor status bits 14 and 15 select the active stack pointer used for all instructions except MFPI, MFPD, MTPI, and MTPD. When these instructions select R6 as the destination register, bits 12 and 13 of the PSW select the active stack pointer. In both cases, the 2-bit selection codes described in Table 1-2 are used to select the active register.

1.2.5 Program Counter

The PC contains the 16-bit address of the next instruction stream word to be accessed. It is designated as R7 and controls the sequencing of instructions. The PC is directly addressable by single- and double-operand instructions and is a general purpose register, although it is normally not used as an accumulator.

1.2.6 Processor Status Word (17 777 776)

The PSW provides the current and previous operational modes, the general purpose register group being used, the current priority level, the condition code status, and the trace trap bit used for program debugging. The PSW is initialized at power-up and is cleared with a console start. The PSW register is defined in Figure 1-2 and is described in Table 1-3.

Table 1-2 Stack Pointer (PSW <15:14> or <13:12>)

Code	Selected R6
00	Kernel Stack Pointer (KSP)
01	Supervisor Stack Pointer (SSP)
10	Illegal – User stack pointer selected
11	User Stack Pointer (USP)

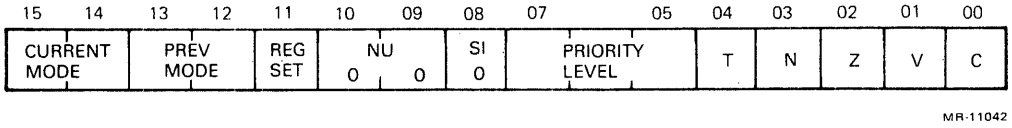


Figure 1-2 Processor Status Word Register

Table 1-3 Processor Status Word Bit Description

Bit(s)	Name	Status	Function
<15:14>	Current mode	R/W	Indicates the current operating mode and is coded as follows. **Bits 15 14 Mode** 0 0 Kernel 0 1 Supervisor 1 0 Illegal 1 1 User
<13:12>	Previous mode	R/W	Indicates the previous operating mode and is coded the same as bits <15:14>.
11	Register set	R/W	Selects the group of general purpose registers being used. When the bit is set, the R0'–R5' group is selected and when cleared, the R0–R5 group is selected.
<10:9>	Not used	R	Read as zeros.
8	Suspended information	R/W	Reserved.
<7:5>	Priority	R/W	Indicates the current priority level of the processor and is coded as follows. **Bits 7 6 5 Priority Level** 1 1 1 7 1 0 0 6 1 0 1 5 1 0 0 4 0 1 1 3 0 1 0 2 0 0 1 1 0 0 0 0
4	Trap*	R/W	The trap bit is inactive when it is cleared. When set, the processor traps to location 14 at the end of the current instruction. It is useful for debugging programs and setting breakpoints.
3	Negative	R/W	Condition code N is set when the previous operation result was negative.
2	Zero	R/W	Condition code Z is set when the previous operation result was zero.
1	Overflow	R/W	Condition code V is set when the previous operation resulted in an arithmetic overflow.
0	Carry	R/W	Condition code C is set when the previous operation caused a carry out.

* The T-bit cannot be set by explicitly writing to the PSW. It can only be changed by the RTI/RTT instructions.

1.2.7 CPU Error Register (17 777 766)

The CPU error register identifies the source of any trap or abort condition that caused a trap through location 4. Six separate error conditions are identified in Figure 1-3 and are described in Table 1-4. The register is cleared by any write reference, by power-up, or by console start. It is not changed by the RESET instruction.

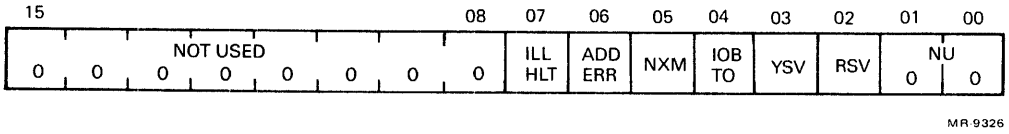


Figure 1-3 CPU Error Register

Table 1-4 CPU Error Register Bit Description

Bit(s)	Name	Status	Function
<15:8>	Not used	Read	Read as zeros.
7	Illegal HALT	Read	Set when execution of a HALT instruction is attempted in user or supervisor mode.
6	Address error	Read	Set when word access to an odd byte address or an instruction fetch from an internal register is attempted.
5	Nonexistent memory	Read	Set when a reference to main memory times out.
4	I/O bus timeout	Read	Set when a reference to the I/O page times out.
3	Yellow stack violation	Read	Set on a yellow stack overflow trap. (Kernel mode stack reference less than 400 octal).
2	Red stack violation	Read	Set on a red stack trap – a kernel stack push abort during an interrupt, abort, or trap sequence.
<1:0>	Not used	Read	Read as zeros.

1.2.8 Program Interrupt Request Register (17 777 772)

The PIRQ register implements a software interrupt facility. A request is initiated by setting one of the bits <15:9>, which corresponds to a program interrupt request for priority levels 7 through 1. Bits <7:5> and <3:1> are set by hardware to the encoded value of the highest pending request set. When the interrupt is acknowledged, the processor vectors to address 240 for a service routine. It is the responsibility of the service routine to clear the interrupt request. The PIRQ register is defined in Figure 1-4 and is described in Table 1-5. The PIRQ register is cleared at power-up, by a console start, or by the RESET instruction.

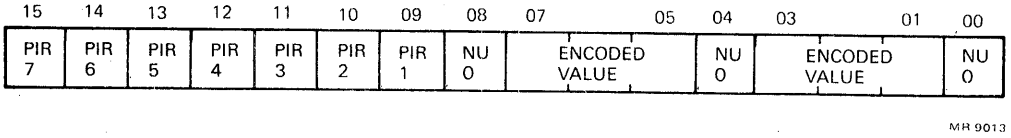


Figure 1-4 Program Interrupt Request Register

Table 1-5 Program Interrupt Request Bit Description

Bit(s)	Name	Status	Function
15	Level 7	R/W	Requests an interrupt priority of level 7.
14	Level 6	R/W	Requests an interrupt priority of level 6.
13	Level 5	R/W	Requests an interrupt priority of level 5.
12	Level 4	R/W	Requests an interrupt priority of level 4.
11	Level 3	R/W	Requests an interrupt priority of level 3.
10	Level 2	R/W	Requests an interrupt priority of level 2.
9	Level 1	R/W	Requests an interrupt priority of level 1.
<7:5>	Encoded value	R/W	Bits <7:5> represent the encoded value of the highest priority level set in bits <15:9>.
<3:1>	Encoded value	R/W	Bits <3:1> represent the encoded value of the highest priority level set in bits <15:9>. It is the same value as bits <7:5>.
0	Not Used		Read as zero.

1.3 INTERRUPTS

The KDJ11-B module uses a variety of trap, hardware, and software interrupts. Their order of priority is given in Table 1-6. Four interrupt request lines allow external hardware to interrupt the processor on four interrupt levels using an externally supplied vector. Seven levels of software interrupt requests are supported through use of the PIRQ register. A variety of internally vectored traps are provided to flag error conditions, and certain instructions result in a trap condition.

Interrupts and traps are requests that cause the KDJ11-B to temporarily suspend the execution of the current program and service the device or condition that caused the interrupt or trap. The KDJ11-B has eight levels of interrupt priority and the current priority level is defined by bits <7:5> of the processor status register. Therefore, only interrupts with a higher priority than the current priority can interrupt the current program. The only exception to this is the nonmaskable interrupt or trap that occurs independent of the processor priority. These nonmaskable interrupts have their own priority structure (Table 1-6).

1.3.1 Sunset Loops

A sunset loop is an infinite loop caused by illegally mapped vectors. The following sunset loops can be exited by asserting the BHALT input.

Interrupts	Cause
Parity error	Bad parity in the parity vector
Trace trap	Trace vector has T-bit set
All PIRQs	PIRQ vector priority level does not block that level
Aborts	Any abort that occurs during a service routine such as reading the vector or pushing onto the stack. These include nonexistent memory, I/O timeouts, MMU aborts, parity aborts, and odd address aborts.

Sunset loops that cannot be exited are caused by external inputs that are not being reset or cleared. These can be MPWRF L, MFPE L, MIRQ <3:0> H, and MEVNT L.

Table 1-6 KDJ11-B Interrupts

Interrupt	Internal/ External	Vector Address	Priority Level
Red stack trap (CPU error register, bit 2)	Internal	4	NM*
Address error (CPU error register, bit 6)	Internal	4	NM
Memory management violation (MMR0, bits <13:15>)	Internal	250	NM
Timeout/nonexistent memory (CPU error register, bits <4:5>)	Internal	4	NM
Parity error (PARITY, ABORT)	External	114	NM
Trace (T-bit) trap (PSW, bit 4)	Internal	14	NM
Yellow stack trap (CPU error register, bit 3)	Internal	4	NM

Table 1-6 KDJ11-B Interrupts (Cont)

Interrupt	Internal/ External	Vector Address	Priority Level
Power fail (PWRF)	External	24	NM
FP exception (FPE)	External	244	NM
PIR 7 (PIRQ, bit 15)	Internal	240	7
IRQ 7	External	User defined	7
PIR 6 (PIRQ, bit 14)	Internal	240	7
BEVNT (LTC)	External	100	6
IRQ 6	External	User defined	6
PIR 5 (PIRQ, bit 13)	Internal	240	5
IRQ 5	External	User defined	5
PIR 4 (PIRQ, bit 12)	Internal	240	4
IRQ 4	External	User defined	4
PIR 3 (PIRQ, bit 11)	Internal	240	3
PIR 2 (PIRQ, bit 10)	Internal	240	2
PIR 1 (PIRQ, bit 9)	Internal	240	1
Halt line (HALT)†	External	None	
FP instruction exception		244	
TRAP (trap instruction)		34	
EMT (emulator trap instruction)		30	
IOT (I/O trap instruction)		20	
BPT (breakpoint trap instruction)		14	
CSM (call to supervisor mode instruction)		10	
HALT instruction		4	
WAIT (wait-for-interrupt instruction)		Does not trap, but frees the bus when waiting for external interrupt.	

* NM = Nonmaskable

† The halt line usually has the lowest priority. However, it has highest priority during vector reads. This allows the user to break out of potential infinite loops called sunset loops. A sunset loop could occur if a vector has not been properly mapped during memory management operations.

1.3.2 Red Stack Aborts

A red stack abort happens when an abort occurs while pushing the PC and PSW onto the kernel stack while in the process of servicing an interrupt, an abort or a trap routine. This type of abort sets bit 2 of the CPU error register, loads the kernel stack pointer (R6) with virtual address 4, and then traps through location 4 in the kernel space. The old PC and PSW are saved in locations 0 and 2 of the kernel space.

The service routine to clear bit 2 of the CPU error register reads the vector at virtual address 4 in the kernel space. An emergency stack is then set up in the new mode at virtual address 4 and executes a trap through virtual address 4. This insures that the old PC and PSW are saved in kernel space locations 0 and 2.

1.3.3 Addressing Errors

An addressing error occurs when an odd address is used with a word reference (odd address error), or an instruction stream fetch attempts to access an internal processor register. The internal processor registers are the PDRs, PARs, CPU error, PSW, PIRQ, MMR0–MMR3, Hit/Miss, and CCR. When an addressing error happens, it sets bit 6 of the CPU error register and traps through virtual address 4 of the kernel data space.

1.3.4 Bus Timeout Errors

A bus timeout error occurs if the BRPLY L bus signal is not asserted within 10 μseconds after the KDJ11-B asserts the BDIN L or BDOUT L signals. The I/O page timeout error sets bit 4 of the CPU error register if the address references the I/O page. The nonexistent memory timeout error sets bit 5 of the CPU error register for all other address errors. As a result of the error condition, the KDJ11-B traps through virtual address 4 of the kernel space. In a Unibus system, the KDJ11-B does not time out, but relies on the Unibus adapter module to assert the PMI timeout signal.

1.3.5 Interrupt Vector Timeouts

An interrupt vector timeout occurs if the BRPLY L bus signal is not asserted within 10 μseconds after the KDJ11-B acknowledges an interrupt by asserting the BIAK L bus signal. The timeout is ignored by the KDJ11-B and it continues as if the interrupt request did not occur. In a Unibus system, the KDJ11-B does not time out, but relies on the Unibus adapter module to assert the PMI timeout signal.

1.3.6 No SACK Timeouts

The no SACK timeout occurs when the BSACK L bus signal is not asserted within 10 μseconds after the KDJ11-B grants a DMA request by asserting BDMG L. The timeout is ignored by the KDJ11-B and it continues as if the DMA request did not occur.

1.4 MEMORY MANAGEMENT

KDJ11-B memory management provides the hardware for complete memory management and protection. It is designed to be a memory management facility for accessing all of physical memory and for multiuser, multiprogramming systems where memory protection and relocation facilities are necessary.

In multiprogramming environments, several user programs are resident in memory at any given time. The tasks of the supervisory program include the following.

- Control the execution of the various user programs
- Manage the allocation of memory and peripheral device resources
- Safeguard the integrity of the system as a whole by control of each user program

In a multiprogramming system, memory management provides the means for assigning memory pages to a user program and for preventing that user from making any unauthorized access to pages outside his assigned area. Thus, a user can effectively be prevented from accidental or willful destruction of any other user program or the system executive program.

The following are the basic characteristics of KDJ11-B memory management.

- 16 user mode memory pages
- 16 supervisor mode memory pages
- 16 kernel mode memory pages
- 8 pages in each mode for instructions
- 8 pages in each mode for data
- Page lengths from 64 to 8192 bytes
- Each page provided with full protection and relocation
- Transparent operation
- 3 modes of memory access control
- Memory access to 4 Mbytes

1.4.1 Memory Mapping

The processor can perform 16-, 18-, or 22-bit address mapping. The I/O page, which is the uppermost 4K words of memory, always uses the physical address locations 17 760 000 to 17 777 777.

1.4.1.1 16-Bit Mapping – There is a direct mapping relocation from virtual to physical addresses. The lowest 28K virtual addresses are the same corresponding physical addresses. The I/O page physical addresses are located in the upper 4K block as shown in Figure 1-5.

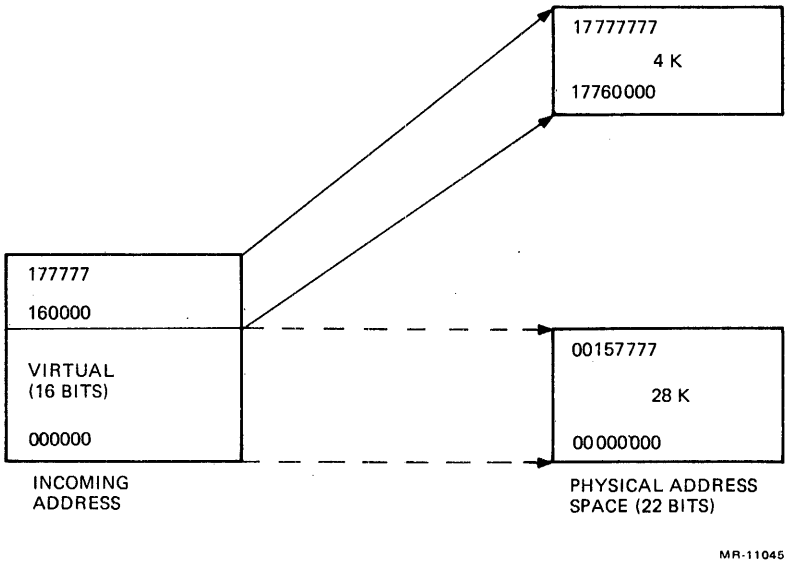


Figure 1-5 16-Bit Mapping

1.4.1.2 18-Bit Mapping – Each of the three modes: kernel, supervisor, and user, are allocated 32K addresses that are mapped into 128K words of physical address space. The lowest 124K words of physical memory, or the I/O page, can be referenced as shown in Figure 1-6.

1.4.1.3 22-Bit Mapping – This mode uses the full 22-bit address to access all of the physical memory. The upper 4K block is still the I/O page as shown in Figure 1-7.

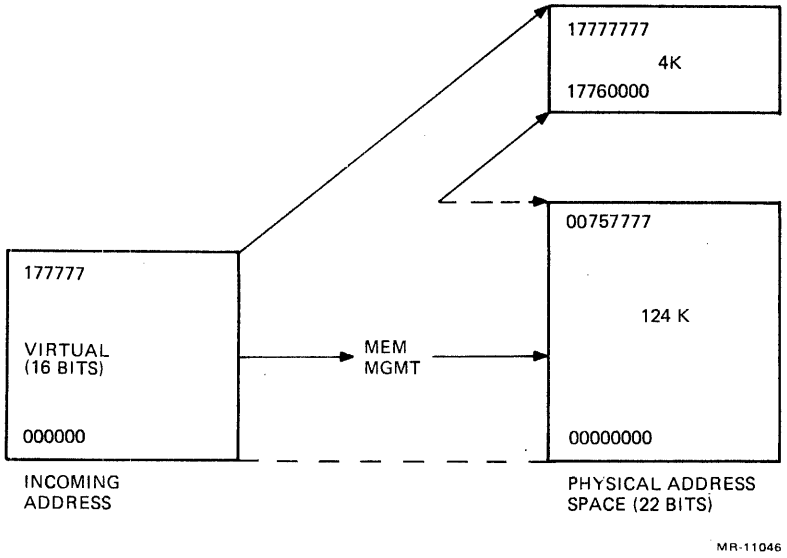


Figure 1-6 18-Bit Mapping

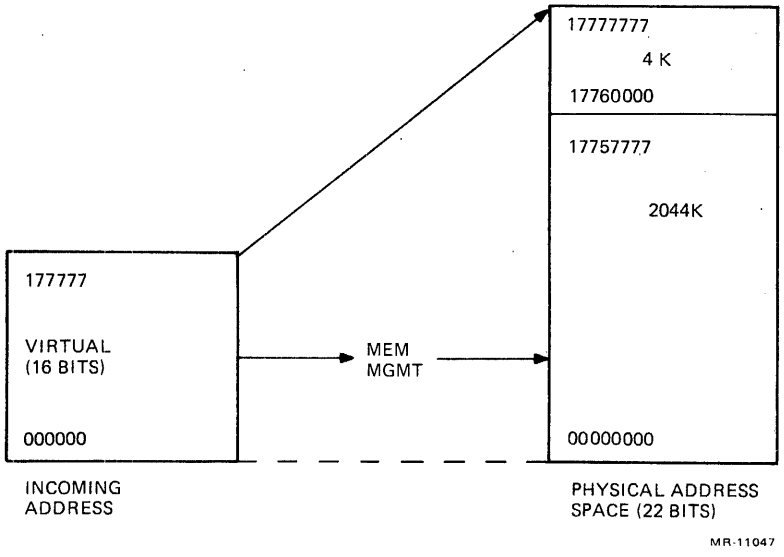


Figure 1-7 22-Bit Mapping

1.4.2 Compatibility

The operation of 16-, 18-, and 22-bit mapping can be used to provide compatibility among other PDP-11 computers. This means that software written and developed for any PDP-11 computer can be run on the KDJ11-B without modification. Refer to Table 1-7.

1.4.3 Virtual Addressing

When memory management is operating, the normal 16-bit address is no longer interpreted as a direct physical address, but as a virtual address containing information to be used in constructing a new 22-bit physical address. The information contained in the virtual address is combined with relocation information contained in the page address register to yield a 22-bit physical address, as shown in Figure 1-8. Using memory management, memory can be dynamically allocated in pages, each composed of from 1 to 128 integral blocks of 64 bytes.

The starting physical address for each page is an integral multiple of 64 bytes, and each page has a maximum size of 8192 bytes. Pages may be located anywhere within the physical address space. The determination of which set of 16 page registers is used to form a physical address is made by the current mode of operation (i.e., kernel, supervisor, or user mode) and by whether the reference is for instructions or data.

Table 1-7 KDJ11-B Compatibility

Mapping	Memory Management	System
16-bit	Off	PDP-11/05, 11/10, 11/15, 11/20, 11/03
18-bit	On	PDP-11/35, 11/40, 11/45, 11/50, 11/23
22-bit	On	PDP-11/70, 11/44, 11/24, 11/23 PLUS

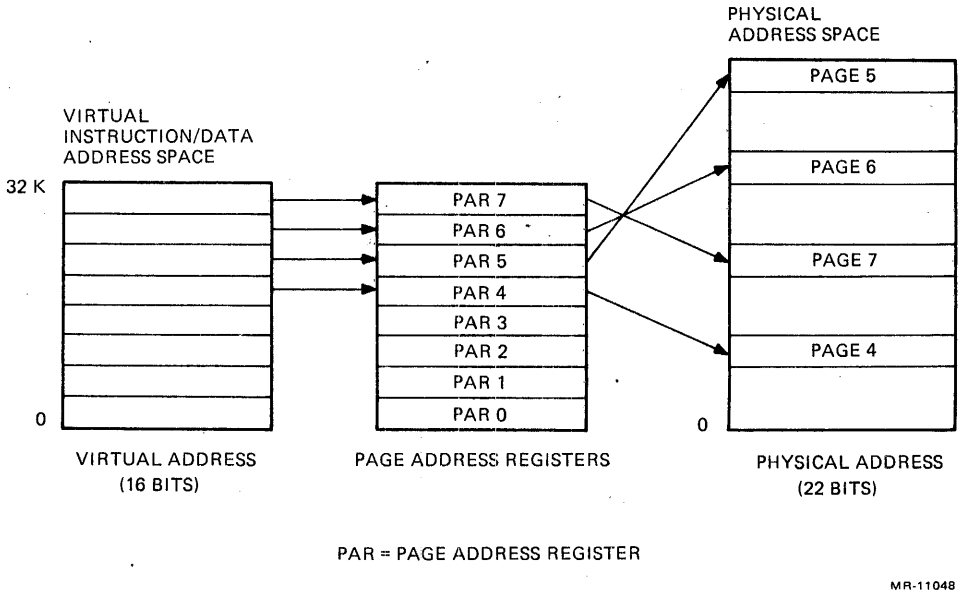


Figure 1-8 Virtual Address Mapping into Physical Address

1.4.4 Interrupts Under Memory Management

Memory management relocates all addresses. When it is enabled, all traps, aborts, and interrupt vectors are mapped using the kernel mode data space mapping registers. Therefore, when a vectored transfer occurs, the new PC and PSW are obtained from two consecutive words physically located at the trap vector, and are mapped using kernel mode data space registers.

The stack used for the "push" of the current PC and PSW is specified by bits <15:14> of the new PSW. The PSW mode bits also determine the new mapping register set. This allows the kernel mode program to have complete control over servicing all traps, aborts or interrupts. The kernel program may assign the service of some of these conditions to a supervisor or user mode program by simply setting the mode bits of the new PSW in the vector to return control to the appropriate mode.

1.4.5 Construction of a Physical Address

All addresses with memory relocation enabled either reference information in instruction (I) space or data (D) space. I space is used for all instruction fetches, index words, absolute addresses, and immediate operands; D space is used for all other references. I space and D space each have eight Page Address Registers (PARs) in each mode of CPU operation (kernel, supervisor, and user). MMR3 can disable D space and map all references (instructions and data) through I space, or can enable D space and map all references through both I and D space.

The basic information needed for the construction of a physical address comes from the virtual address, which is illustrated in Figure 1-9, and the appropriate PAR set.

The Virtual Address (VA) consists of the following.

- The Active Page Field (APF). This 3-bit field determines which of the eight page address registers from the set PAR0–PAR7 is used to form the physical address.
- The displacement field. This 13-bit field contains an address relative to the beginning of a page. The longest page length is 8 Kbytes as determined by the 13 bits. The displacement field is further subdivided into two fields as shown in Figure 1-10.

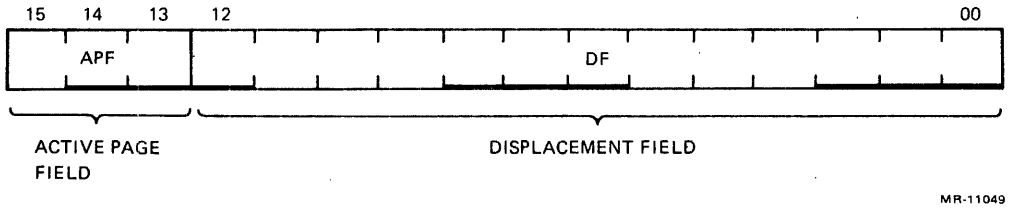


Figure 1-9 Interpretation of a Virtual Address

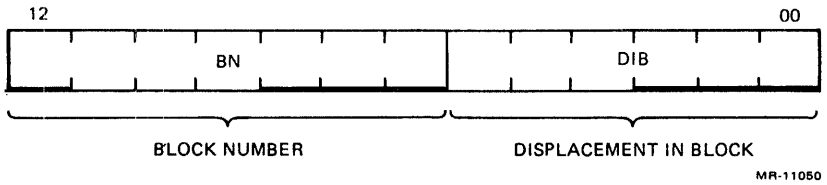


Figure 1-10 Displacement Field of a Virtual Address

The displacement field consists of the following.

- The block number. This 7-bit field is interpreted as the block number within the current page.
- The displacement in block. This 6-bit field contains the displacement within the block referred to by the block number.

The remainder of the information needed to construct the physical address comes from the contents of the PAR referenced by the Page Address Field (PAF). This 16-bit register specifies the starting address of the memory page. The PAF is actually a block number in the physical memory. For instance, PAF = 3 indicates a starting address of 96 (3 × 32 words in physical memory).

The construction of the Physical Address (PA) is illustrated in Figure 1-11. The logical sequence involved in constructing a PA is as follows.

1. Select a set of PARs. This depends on the space being referenced and the protection mode being used.
2. The APF of the VA selects one of eight page address registers (PAR0–PAR7) from the appropriate set.
3. The PAF of the selected PAR contains the starting address of the currently active page as a block number in physical memory.
4. The block number from the VA is added to the PAF to yield the number of the block in physical memory. These are bits <21:6> of the PA being constructed.
5. The displacement in block from the displacement field of the VA is joined to the physical block number to yield a true 22-bit PA.

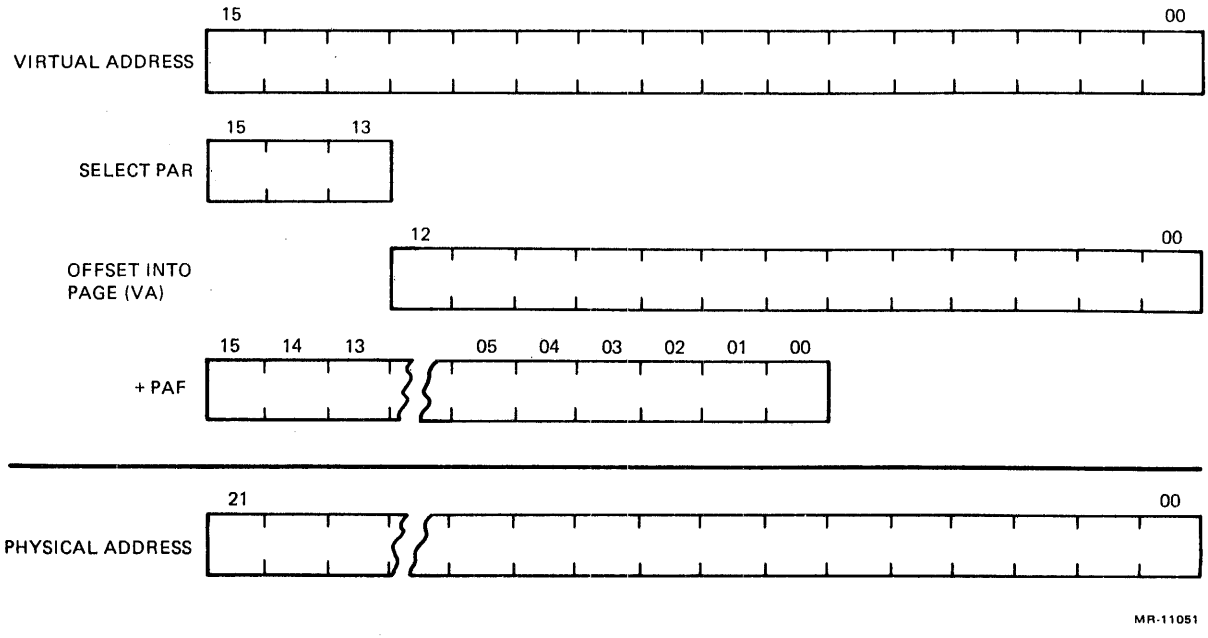


Figure 1-11 Construction of a Physical Address

1.4.6 Memory Management Registers

Memory management implements 3 sets of 32 16-bit registers as shown in Figure 1-12. One set of registers is used in kernel mode, another in supervisor mode, and the third in user mode. The protection mode in use determines which set is to be used. Each set is subdivided into two groups of 16 registers. One group is used for references to instruction (I) space, and one to data (D) space. The I space group is used for all instruction fetches, index words, absolute addresses, and immediate operands. The D space group is used for all other references, providing it has not been disabled by MMR3. Each group is further subdivided into two parts of eight registers. One part is the PAR whose function was described previously. The other part is the Page Descriptor Register (PDR). PARs and PDRs are always selected in pairs by the top three bits of the virtual address. A PAR/PDR pair contains all the information needed to describe and locate a currently active memory page.

The MMRs are located in the uppermost 8 Kbytes of physical address space, which is designated as the I/O page. The addresses allocated to the MMRs are listed in Table 1-8.

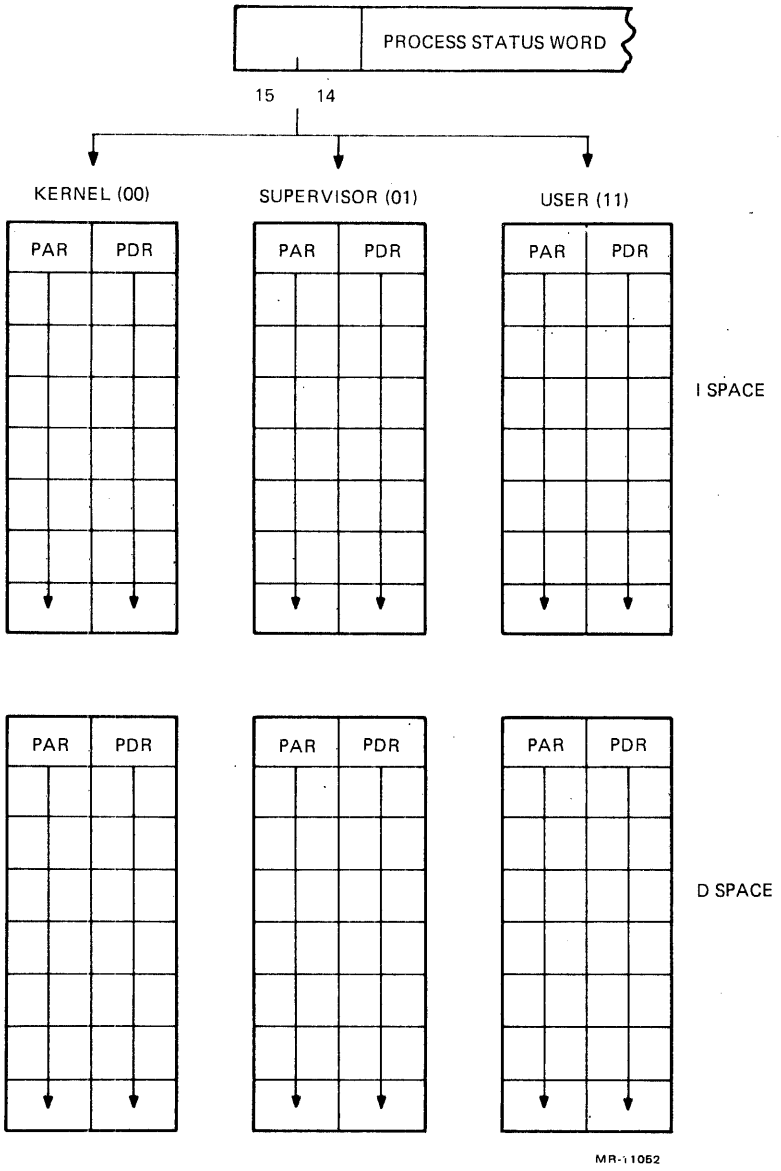


Figure 1-12 Active Page Register

Table 1-8 Memory Management Register Addresses

Register	Address
Memory management register 0 (MMR0)	17 777 572
Memory management register 1 (MMR1)	17 777 574
Memory management register 2 (MMR2)	17 777 576
Memory management register 3 (MMR3)	17 772 516
User I space descriptor register (UISDR0)	17 777 600
.	.
.	.
.	.
User I space descriptor register (UISDR7)	17 777 616
User D space descriptor register (UDSDR0)	17 777 620
.	.
.	.
.	.
User D space descriptor register (UDSDR7)	17 777 636
User I space address register (UISAR0)	17 777 640
.	.
.	.
.	.
User I space address register (UISAR7)	17 777 656
User D space address register (UDSAR0)	17 777 660
.	.
.	.
.	.
User D space address register (UDSAR7)	17 777 676
Supervisor I space descriptor register (SISDR0)	17 772 200
.	.
.	.
.	.
Supervisor I space descriptor register (SISDR7)	17 772 216
Supervisor D space descriptor register (SDSDR0)	17 772 220
.	.
.	.
.	.
Supervisor D space descriptor register (SDSDR7)	17 772 236
Supervisor I space address register (SISAR0)	17 772 240
.	.
.	.
.	.
Supervisor I space address register (SISAR7)	17 772 256

Table 1-8 Memory Management Register Addresses (Cont)

Register	Address
Supervisor D space address register (SDSAR0)	17 772 260
.	.
.	.
.	.
Supervisor D space address register (SDSAR7)	17 772 276
Kernel I space descriptor register (KISDR0)	17 772 300
.	.
.	.
.	.
Kernel I space descriptor register (KISDR7)	17 772 316
Kernel D space descriptor register (KDSDR0)	17 772 320
.	.
.	.
.	.
Kernel D space descriptor register (KDSDR7)	17 772 336
Kernel I space address register (KISAR0)	17 772 340
.	.
.	.
.	.
Kernel I space address register (KISAR7)	17 772 356
Kernel D space address register (KDSAR0)	17 772 360
.	.
.	.
.	.
Kernel D space address register (KDSAR7)	17 772 376

1.4.6.1 Page Address Registers – The PAR contains the PAF, a 16-bit field that specifies the starting address of the page as a block number in physical memory.

The PAR (Figure 1-13) contains the PAF that may be alternatively thought of as a relocation register containing a relocation constant, or as a base register containing a base address. These registers are not changed by either console starts or by the RESET instruction. They are undefined at power-up.

1.4.6.2 Page Descriptor Register – The PDR contains information relative to page expansion, page length, and access control. The register is shown in Figure 1-14 and is described in Table 1-9.

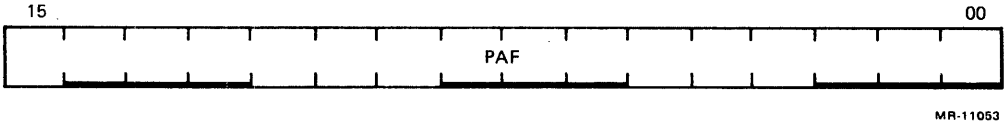


Figure 1-13 Page Address Register

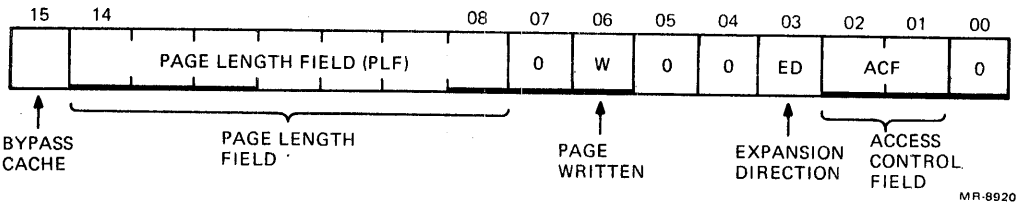


Figure 1-14 Page Descriptor Register

Table 1-9 Page Descriptor Register Bit Description

Bit(s)	Name	Status	Function
15	Bypass cache	R/W	This bit implements a conditional cache bypass mechanism. If the PDR accessed during a relocation operation has this bit set, the reference goes directly to main memory. Read or write hits result in invalidation of the accessed cache location.
<14:8>	Page length field	R/W	This field specifies the block number that defines the page boundary. The block number of the virtual address is compared against the page length field to detect length errors. An error occurs when expanding upwards, if the block number is greater than the page length field, and when expanding downwards, if the block number is less than the page field.
7	Not used	RO	Read as zero.
6	Page written	RO	The written into (W) bit indicates whether the page has been written into since it was loaded in memory. When this bit is set, it indicates a modified page. The W-bit is automatically cleared when the PAR of that page is written.
<5:4>	Not used	RO	Read as zeros.
3	Expansion direction	R/W	This bit specifies the direction in which the page expands. If it = 0, the page expands upward from block number 0 to include blocks with higher addresses; if it = 1, the page expands downward from block number 127 to include blocks with lower addresses.
<2:1>	Access control field	R/W	This field contains the access code for this particular page. The access code specifies the manner in which a page may be accessed and whether or not a given access should result in an abort of the current operation. Implemented codes are as follows. 00 Nonresident – abort all accesses 01 Read only – abort on write 10 Not used – abort all accesses 11 Read/write access
0	Not used	RO	Read as zero.

1.4.7 Fault Recovery Registers

Aborts generated by the memory management hardware are vectored through kernel virtual location 250. MMRs 0, 1, 2, and 3 are used to determine why the abort occurred and to allow for program restarting.

NOTE

An abort to a location which is itself an invalid address causes another abort. Thus, the kernel program must ensure that kernel virtual address 250 is mapped into a valid address. Otherwise, a loop requiring console intervention occurs.

1.4.7.1 Memory Management Register 0 (17 777 572) – MMR0 provides control and records memory management unit status. The register contains abort and status flags as shown in Figure 1-15 and described in Table 1-10.

1.4.7.2 Memory Management Register 1 (17 777 574) – MMR1 records any autoincrement or autodecrement of a general purpose register, including explicit references through the PC. The increment or decrement amount by which the register was modified is stored in 2's complement notation. The lower byte is used for all source operand instructions and the destination operand may be stored in either byte, depending on the mode and instruction type. The register is cleared at the beginning of each instruction fetch. The register is defined in Figure 1-16.

1.4.7.3 Memory Management Register 2 (17 777 576) – MMR2 is loaded with the program counter of the current instruction and is frozen when any abort condition is posted in MMR0.

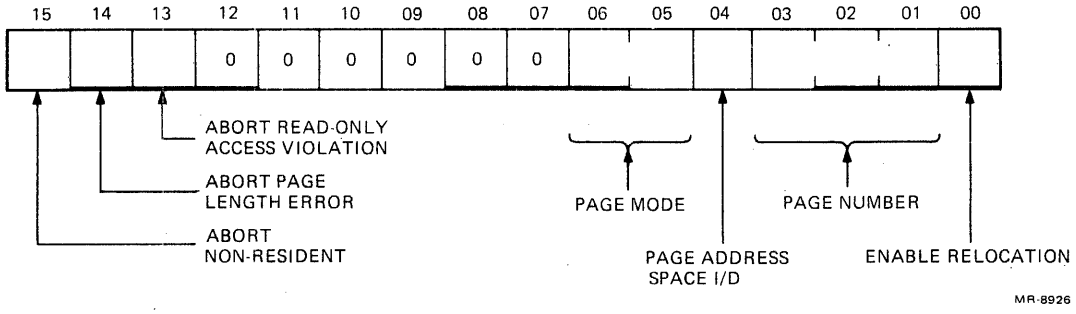


Figure 1-15 Memory Management Register 0 (MMR0)

Table 1-10 MMR0 Bit Description

Bit(s)	Name	Status	Function
15*	Nonresident abort	R/W	Bit 15 is set by attempting to access a page with an access control field key equal to 0 or 2. It is also set by attempting to use memory relocation with a processor mode (PSW <15:14>) of 2.
14*	Page length abort	R/W	Bit 14 is set by attempting to access a location in a page with a block number (virtual address bits <12:6>) that is outside the area authorized by the page length field of the PDR for that page.
13*	Read only abort	R/W	Bit 13 is set by attempting to write in a read-only page. Read-only pages have access keys of 1.
<12:7>	Not used	RO	Read as zeros.
<6:5>	Processor mode	RO	Bits <6:5> indicate the processor (kernel, supervisor, user, illegal) associated with the page causing the abort (kernel = 00, supervisor = 01, user = 11, illegal = 10). If the illegal mode is specified, an abort is generated and bit 15 is set.
4	Page space	RO	Bit 4 indicates the address space (I or D) associated with the page causing the abort (0 = I space, 1 = D space).
<3:1>	Page number	RO	Bits <3:1> contain the page number of the page causing the abort.
0	Enable relocation	R/W	Bit 0 enables relocation. When it is set to 1, all addresses are relocated. When it is set to 0, memory management is inoperative and addresses are not relocated.

* Bits <15:13> can be set by an explicit write. However, such an action does not cause an abort. Whether set explicitly or by an abort, setting any bit in bits <15:13> causes memory management to freeze the contents of MMR0 <6:1>, MMR1, and MMR2. The status registers remain frozen until MMR0 <15:13> is cleared by an explicit write.

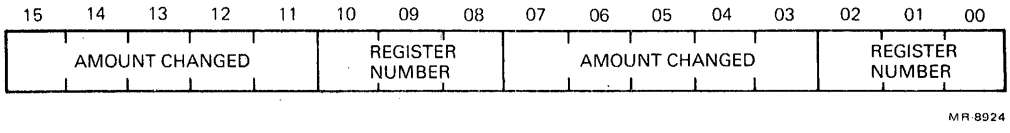


Figure 1-16 Memory Management Register 1 (MMR1)

1.4.7.4 Memory Management Register 3 (17 772 516) – MMR3 enables the data space for the kernel, supervisor, and user operating modes. It also selects either 18- or 22-bit mapping and enables the request for the supervisor macroinstruction (CSM). MMR3 is cleared during power-up, by a console start, or by a RESET instruction. The register is shown in Figure 1-17 and is defined in Table 1-11.

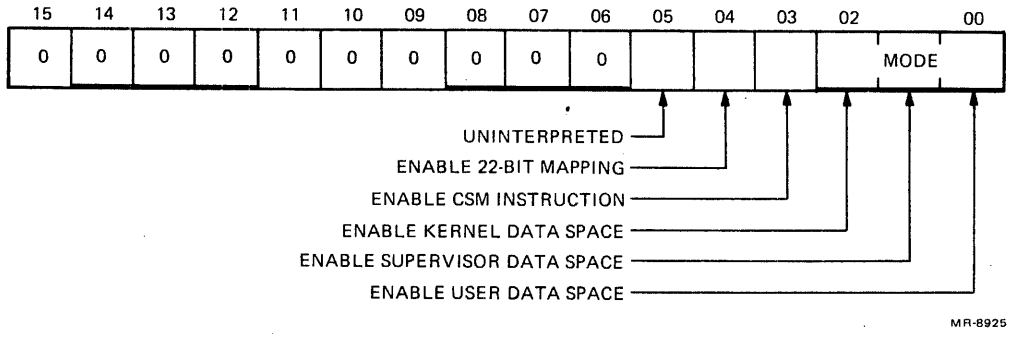


Figure 1-17 Memory Management Register 3 (MMR3)

Table 1-11 MMR3 Bit Description

Bit(s)	Name	Status	Function
<15:6>	Not used	RO	Read as zeros.
5	Uninterpreted	R/W	This bit can be set or cleared under program control, but it is not interpreted by the KDJ11-B.
4	Enable 22-bit mapping	R/W	This bit enables 22-bit memory addressing (the default is 18-bit addressing).
3	Enable CSM instruction	R/W	This bit enables recognition of the Call Supervisor Mode (CSM) instruction.
2	Kernel data space	R/W	This bit enables the data space mapping for the kernel operating mode.
1	Supervisor data space	R/W	This bit enables the data space mapping for the supervisor operating mode.
0	User data space	R/W	This bit enables the data space mapping for the user operating mode.

1.4.7.5 Instruction Back-Up/Restart Recovery – The process of "backing up" and restarting a partially completed instruction involves the following.

1. Performing the appropriate memory management tasks to alleviate the cause of the abort (e.g., loading a missing page).

2. Restoring the general purpose registers indicated in MMR1 to their contents at the start of the instruction, by subtracting the "modify value" specified in MMR1.

3. Restoring the PC to the "abort-time" PC by loading R7 with the contents of MMR2, which contains the value of the virtual PC at the time the "abort-generating" instruction was fetched.

Note that this back-up/restart procedure assumes that the general purpose register used in the program segment will not be used by the abort recovery routine. This is automatically the case if the recovery program uses a different general purpose register set.

1.4.7.6 Clearing Status Registers Following Abort – At the end of a fault service routine, bits <15:13> of MMR0 must be cleared (set to 0) to resume error checking. On the next memory reference following the clearing of these bits, the various registers resume monitoring the status of the addressing operations. MMR2 is then loaded with the next instruction address, MMR1 stores the register change information, and MMR0 logs the memory management status information.

1.4.7.7 Multiple Faults – Once an abort occurs, any subsequent errors occurring while the memory management registers are still frozen does not change MMR0, MMR1, or MMR2. The information saved in MMR0–MMR2 always refers to the first abort that it detected.

1.4.8 Typical Usage Examples

The memory management unit provides a general purpose memory management tool. It can be used in a manner as simple or as complex as desired. It can be anything from a simple memory expansion device to a complete memory management facility.

The variety of meaningful ways to use the facilities offered by the memory management unit means that both single-user and multiprogramming systems have complete freedom to make whatever memory management decisions best suit their individual needs. Although a knowledge of what most types of computer systems seek to achieve may indicate that certain methods of using the memory management unit are more common than others, there is no limit to the ways to use these facilities.

In typical applications, the control over the actual memory page assignments and their protection resides in a supervisory program that operates in kernel mode. This program sets access keys in such a way as to protect itself from willful or accidental destruction by other supervisor or user mode programs. The facilities are also provided in such a way that the kernel mode program can dynamically assign memory pages of varying sizes in response to system needs.

1.4.8.1 Typical Memory Page – When the memory management unit is enabled, the kernel, supervisor, and user mode programs each have eight active pages described by the appropriate PARs and PDRs for data and eight pages for instructions. Each segment is made up of from 1 to 128 blocks and is pointed to by the PAF of the corresponding PAR as illustrated in Figure 1-18.

The memory segment illustrated in Figure 1-18 has the following attributes.

- Page length: 40 blocks
- Virtual address range: 140000–144777
- Physical address range: 312000–316777
- Nothing has been modified (i.e., written) in the page
- Read-only protection
- Upward expansion

These attributes were determined according to the following scheme.

1. Page address register (PAR6) and page descriptor register (PDR6) were selected by the APF of the VA. (Bits <15:13> of the VA = 68.)

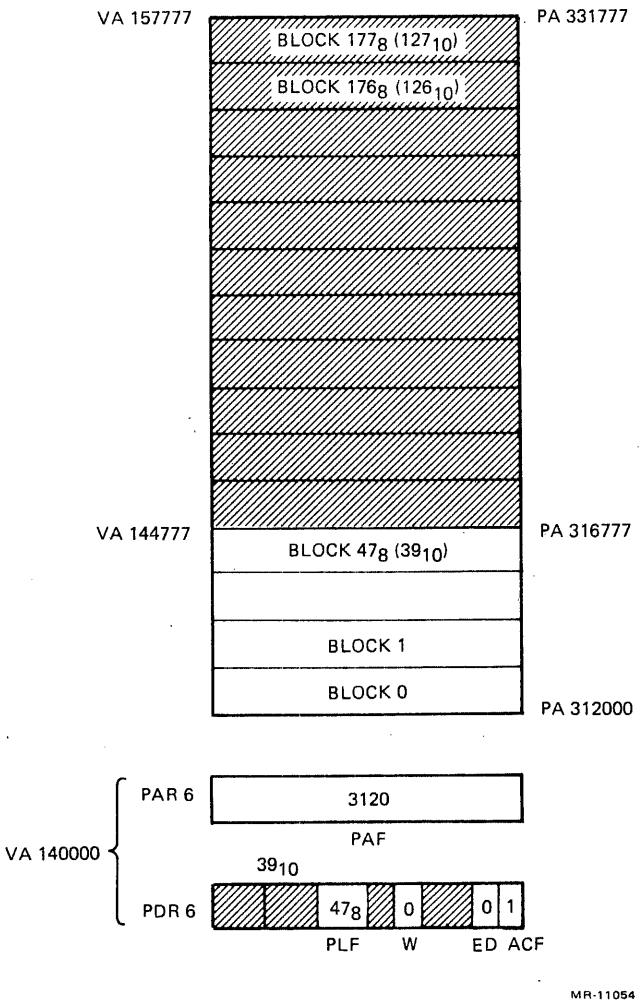


Figure 1-18 Typical Memory Page

2. The initial address of the page was determined from the PAF of PAR6 (312000 = 31208 blocks × 408 (3210) words per block × 2 bytes per word).

NOTE

The PAR that contains the PAF constitutes what is often referred to as a base register containing a base address or a relocation register containing a relocation constant.

3. The page length (478 + 1 = 4010 blocks) was determined from the Page Length Field (PLF) contained in PDR6. Any attempt to reference beyond the 4010 blocks in this page causes a page length error, which results in an abort, vectored through kernel virtual address 250.

4. The PAs were constructed according to the scheme illustrated in Figure 1-11.

5. The W-bit indicates that no locations in this page have been modified (i.e., written). If an attempt is made to modify any location in this particular page, an access control violation abort occurs. If the page is involved in a disk swapping or memory overlay scheme, the W-bit is used to determine whether the page has been modified and therefore requires saving before overlay.

6. The page is read-only protected (i.e., no locations in the page may be modified). The mode of protection was specified by the access control field of PDR6.

7. The expansion direction is upward (ED bit set to 0). If more blocks are required in this segment, they must be added by assigning blocks with higher relative addresses.

The attributes that describe the page shown in Figure 1-18 are determined under software control. The parameters describing the page are loaded into the appropriate PAR and PDR under program control. In a normal application, the page, which itself contains these registers, is assigned to the control of a kernel mode program.

1.4.8.2 Nonconsecutive Memory Pages – Higher virtual addresses do not necessarily map to higher physical addresses. It is possible to set up the PAFs of the PARs so that higher virtual address blocks may be located in lower physical address blocks as illustrated in Figure 1-19.

Although a single memory page must consist of a block of contiguous locations, consecutive virtual memory pages do not have to be located in consecutive physical address locations. The assignment of memory pages is not limited to consecutive nonoverlapping physical address locations.

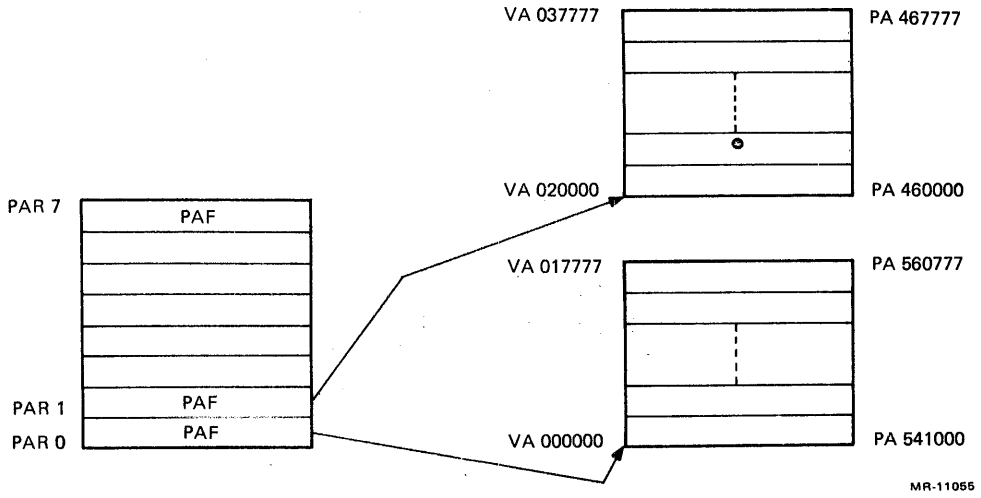


Figure 1-19 Nonconsecutive Memory Pages

1.4.8.3 Stack Memory Pages – When constructing programs, it is often desirable to isolate all program variables from pure code (i.e., program instructions) by placing them on a register indexed stack. These variables can then be "pushed" or "popped" from the stack area as needed. Since stacks expand by adding locations with lower addresses, when a memory page containing "stacked" variables needs more room, it must "expand down" by adding blocks with lower relative addresses to the current page. This mode of expansion is specified by setting the expansion direction bit of the appropriate PDR to a 1. Figure 1-20 illustrates a typical stack memory page and has the following parameters.

PAR6: PAF = 3120
PDR6: PLF = 1758 or 12510 (12810–3)
ED = 1
W = 0 or 1
ACF = **nnn** (to be determined by programmer as necessary)

In this case the stack begins 128 blocks above the relative origin of the memory page and extends downward for a length of three blocks. A page length error abort is generated by the hardware whenever an attempt is made to reference any location below the assigned area (i.e., when the block number from the VA is less than the PLF of the appropriate PDR).

NOTE
The W-bit is set by hardware.

1.4.9 Transparency

In a multiprogramming application, it is possible for memory pages to be allocated so that a program appears to have a complete 64 Kbyte memory configuration. Using relocation, a kernel mode supervisory-type program can perform all memory management tasks entirely transparent to a supervisor or user mode program. In effect, a system can use its resources to provide maximum throughput and response to a number of users, each of whom seems to have a powerful system all to himself.

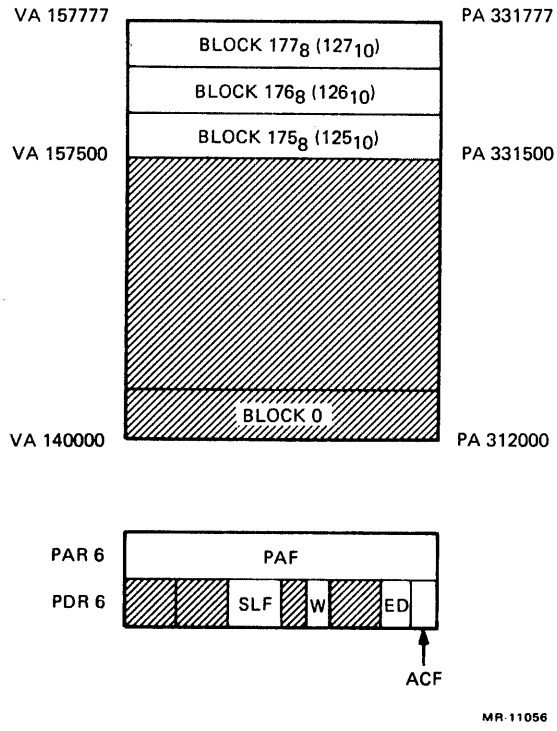


Figure 1-20 Typical Stack Memory Page

1.5 CACHE MEMORY

The statistics from executing programs clearly indicate that at any given moment, a program spends most of its time within a relatively small section of code. The KDJ11-B cache memory exploits this phenomenon by using a small amount of high speed memory to store the most recently accessed memory locations. Cached code executes much faster than noncached code because of the large difference between the access times of the cache memory and the LSI-11 bus main memory.

Figure 1-21 illustrates how the KDJ11-B cache is constructed. It is a direct map (set size one; block size one), 8-Kbyte cache. Each physical address is logically subdivided into a 9-bit label, 12-bit index, and 1-bit byte select field.

The index field is used to select one of 4096 separate cache entries. Each cache entry contains a 9-bit tag field (TAG), tag parity bit (P), tag valid bit (V), two bytes of cache data (B0 and B1) and two corresponding byte parity bits (P0 and P1). (See Figure 1-22.)

A physical address is considered cached when the tag field of the cache entry specified by the index field equals the label field, the valid bit is set, and no parity errors are seen. When a cache read hit occurs (i.e., the address is cached during a read operation), B1 and B0 are used as the source of the data. When a cache read miss occurs (i.e., the address is not cached), main memory is accessed to obtain the data.

A physical address is stored in the cache whenever the cache is allocated. To allocate the cache, the tag field of a cache entry specified by the index field is set equal to the label field, the V-bit is set, B1 and B0 are loaded with the fresh data, and the parity bits are correctly calculated. This guarantees that the next access to this address will report a cache hit. It should be noted that allocating the cache typically destroys a previously allocated valid cache entry. The cache is allocated whenever a read miss or word write miss occurs.

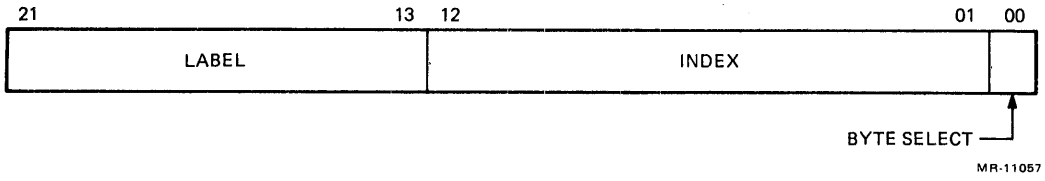


Figure 1-21 Cache Physical Address

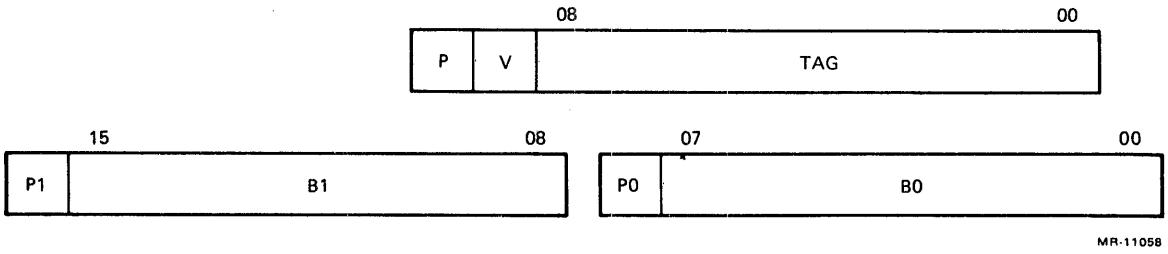


Figure 1-22 Cache Data Format

Write cycles are separated into word write and byte write operations. Main memory is always updated during writes. A cache hit causes the proper byte(s) to be written in both the cache and in main memory. This is called writing through the cache. A cache miss during a word write allocates the cache, but because two bytes are allocated together, a byte write only updates main memory. The cache response matrix is summarized in Table 1-12.

The I/O page (top 8 Kbytes) is never cached and therefore always reports misses. This is because the I/O page contains dynamic status registers, which, when read, must always convey the latest information.

When the system is powered up, the cache must be cleared and correct parity written into each entry. This is called flushing the cache.

A potential stale data problem can occur when a DMA device writes to a cached location. Therefore, a DMA TAG store, which is an identical copy of the cache TAG store, is maintained and monitors each DMA transaction. When DMA writes references to a cache stored location, the processor is interrupted and the overwritten cache entry is invalidated.

Table 1-12 Cache Response Matrix

Operation	DMA Hit	DMA Miss	CPU Hit	CPU Miss
Read	Read memory, no cache change	Read memory, no cache change	Read cached data, allocate cache	Read memory
Write word	Invalidate cache, update memory	Update memory, no cache change	Write through cache to memory	Write memory, allocate cache
Write byte	Invalidate cache, update memory	Update memory, no cache change	Write through cache to memory	Write memory, no cache change
Read bypass	N/A	N/A	Read memory, invalidate cache	Read memory, no cache change
Write bypass	N/A	N/A	Write memory, invalidate cache	Write memory, no cache change
Read force miss	N/A	N/A	Read memory, no cache change	Read memory, no cache change
Write force miss	N/A	N/A	Write memory, no cache change	Write memory, no cache change

For both diagnostic and availability reasons, it is important to be able to turn off the cache via software. The cache is disabled by setting either of the force cache miss bits (2 and 3) in the cache control register. When disabled, all references are forced to miss the cache. That is, main memory is always accessed, cache parity errors are ignored, and no cache allocation is performed. The cache is essentially removed from the system. This is different from bypassing the cache. Bypass references access the main memory, check cache parity, and invalidate the cache entry if previously allocated. Read references that bypass the cache check for parity errors and invalidate any address hits.

1.5.1 Parity

The KDJ11-B module has a main memory parity error detection mechanism that is in the DC351 gate array. The BDAL 16 and 17 data lines are sampled by the negation of either TDIN H or RDSTRB H when the 16 bits of data are read into the module. These parity bits are used to generate the MEM PERR H output that initiates an abort via the DC350/394 gate array. BDAL bit 16 is the parity error signal and BDAL bit 17 is the parity abort error signal. When both are asserted (1), an abort occurs through the vector at virtual address 114 in kernel space.

The cache memory also has a parity error detection mechanism. A parity error in the cache is not considered fatal because the main memory system has a backup copy of the data. The cache uses even parity for the even data bytes stored in the cache memory and odd parity for the odd data bytes stored in the cache memory. It also uses even parity for the tag field stored in the cache memory.

1.5.1.1 Parity Errors – A parity error is indicated when a single bit error occurs. Parity errors can occur in either the main memory or the cache memory. A main memory parity error is always fatal since the data stored in this memory is wrong and it cannot be restored. This type of parity error always causes an abort through virtual address 114 in the kernel space. Cache parity errors are not considered to be fatal since the data in the cache memory can be updated with the correct data from the main memory. When they occur, the KDJ11-B module aborts, interrupts, or continues without an abort or interrupt. The action is determined by the state of bits 7 and 0 in the cache control register as defined in Table 1-13.

1.5.1.2 Multiple Cache Parity Errors – If a cache parity error occurs before the error status from a previous cache parity error is cleared from the memory system error register, then no abort or interrupt occurs. The main memory is accessed again to retrieve the correct data and the corrupted cache entry data is updated with the correct data. This prevents a cache hardware failure from generating an infinite series of interrupt or abort service loops.

Table 1-13 Cache Parity Errors

CCR <7>	CCR <0>	Action
0	0	Update cache, interrupt through 114
0	1	Update cache only
1	X*	Update cache, abort through 114 used only for diagnostics

* X = Either 1 or 0.

1.5.2 Memory System Registers

The memory system registers consist of the Cache Control Register (CCR), the Memory System Error Register (MSER), and the Hit/Miss Register (HMR). These registers are used by modules to control the memory system and report any errors that occur.

1.5.2.1 Cache Control Register (17 777 746) – The CCR controls the operation of the cache memory. The cache bypass, abort, and force miss functions can be controlled by software via this register. The CCR is shown in Figure 1-23 and is described in Table 1-14. The register is cleared by either a power-up or a console start. It is unaffected by the RESET instruction.

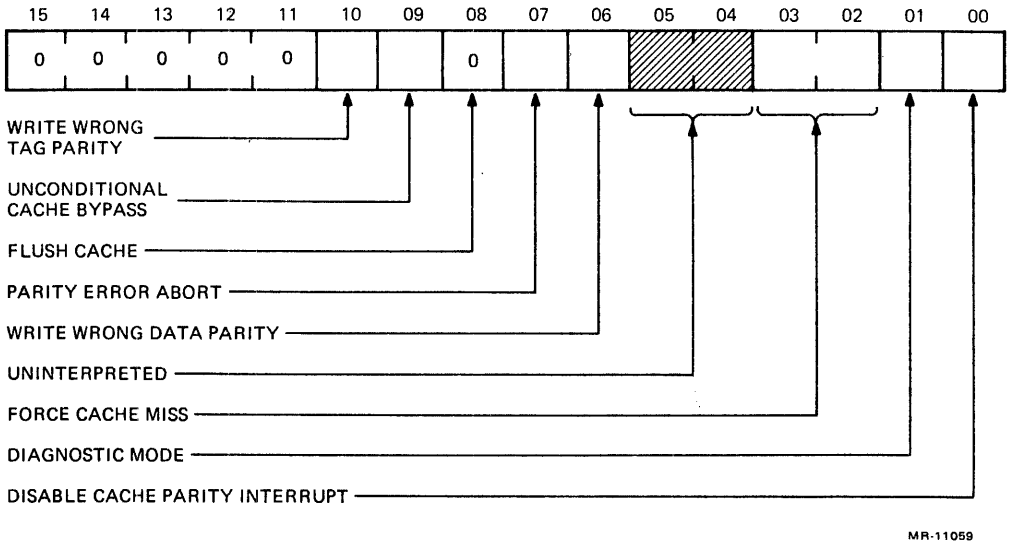


Figure 1-23 Cache Control Register (CCR)

Table 1-14 Cache Control Register Description

Bit(s)	Name	Status	Function
<15:11>	Not used	R	Read as zeros.
10	Write wrong tag parity	R/W	When set (1), this bit causes the cache tags to be written with wrong parity on all update cycles. This causes a cache tag parity error to occur on the next access to that location.
9	Bypass cache	R/W	When set (1), this bit forces all CPU memory references to go directly to main memory. Read hits result in invalidation of accessed locations in the cache.
8	Flush cache*	W	When set (1), this bit causes the entire contents of the cache to be declared invalid. Writing a 0 into this bit has no effect.
7	Enable parity error abort	R/W	This bit is used with bit 0 to define the action taken as a result of a parity error. This bit is reserved for diagnostic purposes.
6	Write wrong data parity	R/W	When set (1), this bit causes high and low parity bytes to be written with wrong parity on all update cycles. This causes a cache parity error to occur on the next access to that location.
<5:4>	Uninterpreted	R/W	These bits can be set or cleared under program control, but are not interpreted by the KDJ11-B.
<3:2>	Force miss	R/W	When either bit is set, all CPU memory references go directly to main memory. The cache tag and data stores are not changed. The parity is not checked. When set (1), these bits remove the cache memory from the system.
1	Diagnostic mode	R/W	When set (1), all nonbypass and nonforced miss word writes allocate the cache, irrespective of NonExistent Memory (NXM) errors. In addition, NXM writes do not trap.
0	Disable cache parity interrupt	R/W	Bits <7:0> specify the action to take following a cache parity error. If both bits are cleared (0) and a parity error occurs, an interrupt through vector 114 is generated. If bit 7 is cleared and bit 0 is set, a cache parity error neither aborts the reference nor generates an interrupt. In any case, all cache parity errors force a memory reference, and update the cache with the fresh data.

* It takes approximately 1 millisecond to flush the cache. During this time DMA and interrupt requests are not serviced and no data processing occurs.

1.5.2.2 Hit/Miss Register (17 777 752) – The HMR records the status of the most recent cache accesses. The HMR is a shift register that records a hit as a 1 and a miss as a 0 for the most recent memory reads. A hit represents data located in the cache memory and a miss means the data is located in the main memory. Bit 0 represents the most recent memory access and is shifted to the left on successive memory access. The HMR is a read-only register and is shown in Figure 1-24.

1.5.2.3 Memory System Error Register (17 777 744) – The MSER is a read-only register. The register monitors parity error aborts and records the type of parity error. The MSER is shown in Figure 1-25 and is described in Table 1-15. The MSER is cleared by any write reference, during power-up, and by a console start. It is unaffected by the RESET instruction.

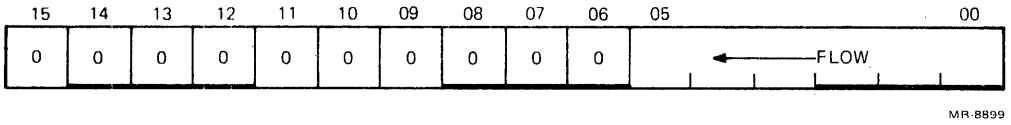


Figure 1-24 Hit/Miss Register (HMR)

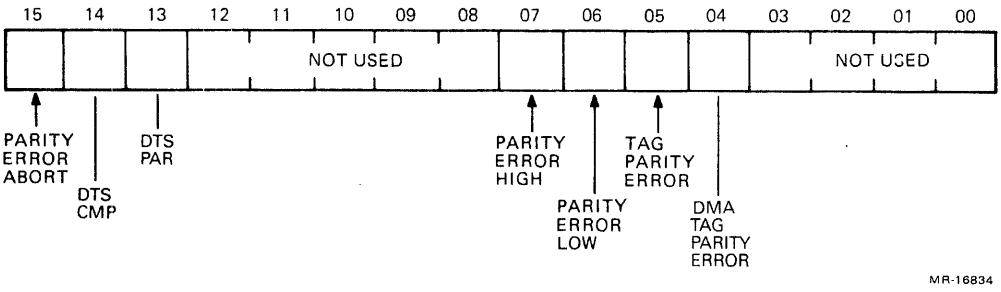


Figure 1-25 Memory System Error Register (MSER)

Table 1-15 Memory System Error Register Description

Bit(s)	Name	Status	Function
15	CPU abort	R	This bit is set (1) when a cache or main memory parity error aborts on instruction. Cache parity errors cause an abort only when bit 7 of the CCR is set. Main memory parity errors always cause an abort.
14	DTS CMP	R	The DTS CMP bit is set by the output of the DMA tag store comparator when a cache miss occurred for the previous non-I/O page reference while in the standalone mode (BCSR bit 8 set). When BCSR bit 8 is clear, DTS CMP is also clear.
13	DTS PAR	R	The DTS PAR bit is set by the output of the DMA tag store parity check logic when an error occurred for the previous non-I/O page reference while in the standalone mode (BCSR bit 8 set). When BCSR bit 8 is clear, DTS PAR is clear.
<12:8>	Not used		Read as zeros.
7*	Cache high byte parity error	R	The cache high byte parity error bit is set when a parity error is detected in the high byte data during a CPU cache read. This bit is also set by a low byte parity error and by the set conditions of MSER bits 5 or 4, when bit 7 of the CCR is cleared.
6*	Cache low byte parity error	R	The cache low byte parity error bit is set when a parity error is detected in the low byte data during a CPU cache read. This bit is also set by a high byte parity error and by the set conditions of MSER bits 5 or 4, when bit 7 of the CCR is cleared.
5*	Cache CPU tag parity error	R	The cache CPU tag parity error bit is set when a parity error is detected in the CPU tag field during a CPU cache read. MSER bits 6 or 5 are also set by a high or low byte parity error, when bit 7 of the CCR is cleared.
4*	Cache DMA tag parity error	R	The cache DMA tag parity error bit is set when a parity error is detected in the DMA tag field during a DMA write operation.
<3:0>	Not used		Read as zeros.

* If a force miss condition is set by CCR bits 3 or 2, or if the CPU tag valid bit is cleared, then cache parity errors and cache DMA parity errors are ignored by MSER bits <7:4>.

1.6 PRIVATE MEMORY INTERCONNECT

The PMI is a unique Q22-Bus protocol that provides a high performance data path between the KDJ11-B module and the MSV11-J memory modules. These modules comprise the private memory and interface with the KDJ11-B via a backplane structure that uses the Q22-Bus as the A/B slots and an interconnecting interface as the C/D slots. This backplane structure allows data and address information to be multiplexed and transmitted via the Q22-Bus BDAL <21:0> data/address lines, while the PMI protocol nonmultiplexed control lines use the C/D interconnecting interface. The PMI protocol functions in a Unibus system by using the KTJ11-B Unibus adapter module designed to interface with a PMI system. The PMI interface consists of 14 control signals used on the C/D interface, 6 Q22-Bus control signals, the bank 7 select signal (BBS7) and the 22 data/address lines (BDAL <22:0>). A complete description of the PMI operation is provided in Chapter 7.

1.6.1 PMI Protocol

In a Q22-Bus system, the KDJ11-B CPU is the default Q22-Bus master and PMI master. Any device on the Q22-Bus that has the capability to be a bus master can take control of the bus and execute normal data transfers over the Q22-Bus. However, it does not become the PMI master.

In a Unibus system, the KDJ11-B CPU is the default PMI master and the KTJ11-B Unibus adapter is the default Unibus master. When the CPU addresses the Unibus memory or I/O page as the PMI master, the Unibus adapter responds as a slave to the CPU and controls the Unibus side of the transaction as the Unibus master.

The Unibus adapter can become the PMI master when the CPU issues a DMA grant or performs an interrupt transaction. The DMA or interrupt grant is accepted by the Unibus adapter and it becomes the PMI master and the Unibus slave. It also passes the DMA or interrupt grant onto a Unibus device, which then becomes the Unibus master. In Unibus systems, the bus master and PMI master can be requested by a NonProcessor Request (NPR) or interrupt request from a Unibus device, or a DMA or interrupt request.

1.6.1.1 Bus Device NPR – Any Unibus device that is capable of being a bus master can issue an NPR request and become the bus master to control data transfers. During these data transfers, the Unibus adapter is the PMI master and responds as a slave if the device accesses the PMI memory, the PMI I/O page or the Unibus adapter I/O page.

1.6.1.2 Bus Device Interrupt – Any Unibus device that is capable of being a bus master can issue a BR7 through 4 request and become the bus master to control data or interrupt vector transfers. In both cases, the Unibus adapter is the PMI master and responds as a slave if the device performs an interrupt vector transaction or accesses the PMI memory, the PMI I/O page or the Unibus adapter I/O page.

1.6.2 PMI Data Transfers

There are three general categories for the PMI data transfer cycles. These are the Data In/Data In Pause (DATI/DATIP), the Block Data In (DATBI), and the Data Out/Data Out Byte (DATO/DATOB) cycles. They are described in the following paragraphs.

On the Q22-Bus, the bus master can perform a read-modify-write cycle that transmits an address, reads a data word or byte and then writes the data word or byte. The PMI read-modify-write is performed by a DATIP cycle that is followed by a DATO/DATOB cycle. The PMI bus master has the responsibility to control the bus for the duration of both cycles.

1.6.2.1 Data In/Data In Pause – The DATI and DATIP cycles are used to read one or two words when the PMI bus master accesses the PMI memory. When the PMI bus master accesses the I/O page or the Unibus memory it can only read one word. The PMI bus master detects an I/O page reference by the assertion of TBS7 and a Unibus memory reference by the assertion of RPUBMEM.

The PMI DATIP is identical to the DATI cycle except that TPBYT is asserted with TADDR to indicate that the cycle immediately following the current cycle is going to be a DATO cycle to the same address.

1.6.2.2 Block Data In – The DATBI cycle is used to read up to 16 words of data when the PMI bus master accesses the PMI memory. The PMI bus master cannot use the DATBI cycle when accessing the I/O page or the Unibus memory. The PMI bus master detects an I/O page reference by the assertion of TBS7, and a Unibus memory reference by the assertion of RPUBMEM.

The PMI bus master can only start DATBI transfers on even word boundaries. This means that address bits <1:0> must be equal to zeros. The PMI bus master cannot use the DATBI cycle to transfer across 16-word address boundaries. This means that the PMI bus master must terminate DATBI data transfers when it reaches a memory location where the address bits <4:1> are all equal to ones.

1.6.2.3 Data Out/Data Out Byte – The DATO and DATOB cycles are used by the PMI bus master to transfer a single word or byte to a PMI slave.

1.7 TERMINAL INTERFACE

The KDJ11-B provides a DLART serial line interface on the module. The console is connected to the module directly or via an interfacing panel. The transmit and receive baud rates are always identical and are determined by the status of bits <2:0> of the Boot and diagnostic facility Configuration Register (BCR). These bits can be selected directly by the configuration switches (8, 7 and 6), or remotely via an external connector that is mounted on the module. The bit settings to select the baud rate are described in Table 1-16.

The DLART uses four registers designated as the Receiver Control/Status Register (RCSR), Receiver BUFfer (RBUF), Transmitter Control/Status Register (XCSR), and the Transmitter BUFfer (XBUF). These registers are described below.

Table 1-16 Baud Rate Selection*

Switches (Bits)			
6 (2)	**7 (1)**	**8 (0)**	**Baud Rate**
1	1	1	300
1	1	0	600
1	0	1	1,200
1	0	0	2,400
0	1	1	4,800
0	1	0	9,600
0	0	1	19,200
0	0	0	38,400

* 1 = switch off; 0 = switch on.

1.7.1 Receiver Control/Status Register (17 777 560)

The RCSR is used to receive console On-line Debugging Technique (ODT) commands and input characters. The console ODT does not execute output bus cycles to this address; the RCSR responds only to input bus cycles. The system software may affect certain bits, such as interrupt enable (bit 6), but the console ODT ignores this. The RCSR is shown in Figure 1-26 and is described in Table 1-17.

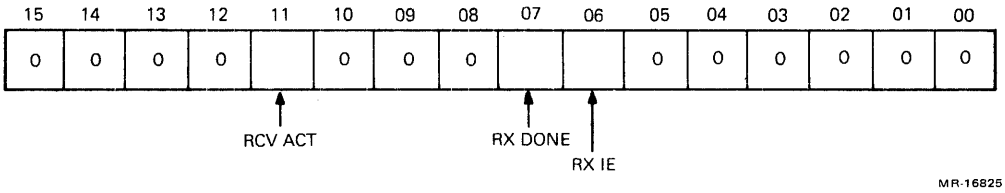


Figure 1-26 Receiver Control/Status Register

Table 1-17 RCSR Bit Description

Bit(s)	Name	Status	Function
<15:12>	Not used	RO	Read as zeros.
11	RCV ACT	RO	The RCV ACT bit is set by the start bit of the serial input data and is cleared by the stop bit at the end of the serial input data. The RX DONE bit is set by the next bit time after RCV ACT is cleared.
<10:8>	Not used	RO	Read as zeros.
7	RX DONE	RO	The RX DONE bit is set when a character is received and is ready to be read from the RBUF register. The bit is cleared by reading the RBUF register and by power-up.
6	RX IE	RW	The RX IE bit is set when RXIRQ is enabled and a program interrupt is requested while RX DONE is set with this bit. The bit is cleared by BUS INIT and by power-up.
<5:0>	Not used	RO	Read as zeros.

1.7.2 Receiver Buffer Register (17 777 562)

The RBUF is used to receive console ODT commands and input data. The console ODT does not execute output to this address; the RBUF responds only to input bus cycles. The system software operates similarly, but the diagnostics may cause output cycles and may not operate properly. The RBUF is shown in Figure 1-27 and is described in Table 1-18.

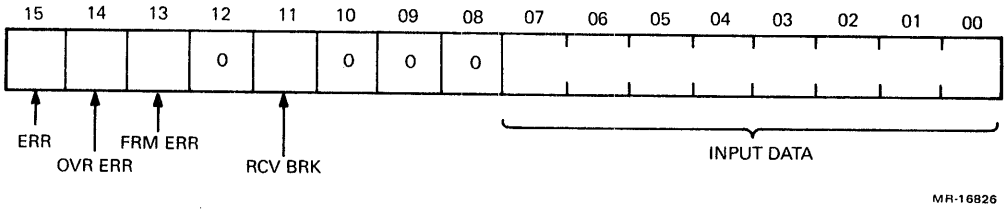


Figure 1-27 Receiver Buffer Register

Table 1-18 RBUF Bit Description

Bit(s)	Name	Status	Function
15	ERR	RO	The ERR bit is set when the OVR ERR bit or the FRM ERR bit is set. This bit does not generate a program interrupt and is clear when both of these bits are clear.
14	OVR ERR	RO	The OVR ERR bit is set when a previous character was received but was not read before it was overwritten by the current character.
13	FRM ERR	RO	The FRM ERR bit is set when the current character has no stop bit. This bit is used to detect breaks.
12	Not used	RO	Read as zero.
11	RCV BRK	RO	The RCV BRK bit is set when the end of the serial data input remains in the space condition for all 11 bits. The bit remains set until the serial data input returns to the mark condition.
<10:8>	Not used	RO	Read as zeros.
<7:0>	Input data	RO	These eight bits are an ASCII character read as input when RCSR bit 7 is set.

1.7.3 Transmitter Control/Status Register (17 777 564)

The XCSR is used to transmit data for the console ODT. The console ODT does not execute input bus cycles to this address; the XCSR responds only to output bus cycles. The system software may cause output cycles that affect certain bits, such as interrupt enable (bit 6), but the console ODT ignores this. The XCSR is shown in Figure 1-28 and described by Table 1-19.

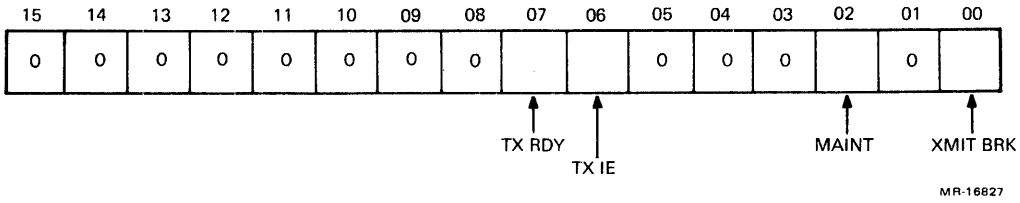


Figure 1-28 Transmitter Control/Status Register

Table 1-19 XCSR Bit Description

Bit(s)	Name	Status	Function
<15:8>	Not used	RO	Read as zeros.
7	TX RDY	RO	The TX RDY bit is set when the XBUF is cleared and can receive another character. The bit is cleared when the XBUF is full. It is also set by power-up and by BUS INIT.
6	TX IE	RW	The TX IE bit is set when TXIRQ is enabled and a program interrupt is requested while TX RDY is set with this bit. The bit is cleared by BUS INIT and by power-up.
<5:3>	Not used	RO	Read as zeros.
2	MAINT	RW	The maintenance bit is set during a self-test that disconnects the external serial input and connects it to the internal serial output. This bit is cleared by BUS INIT and by power-up.
1	Not used	RO	Read as zero.
0	XMIT BRK	RW	The XMIT BRK bit is set when the output serial data is forced into the space condition. The bit is cleared by BUS INIT and by power-up.

1.7.4 Transmitter Buffer Register (17 777 566)

The XBUF is used to transmit data for the console ODT. The console ODT does not execute input bus cycles to this address; the XBUF responds only to output bus cycles. The system software operates similarly, but the diagnostics may cause input cycles and may not operate properly. The XBUF is shown in Figure 1-29 and is described in Table 1-20.

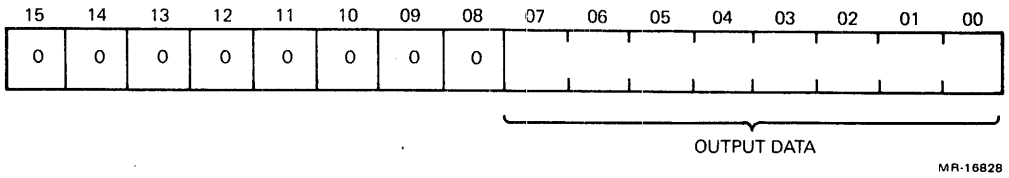


Figure 1-29 Transmitter Buffer Register

Table 1-20 XBUF Bit Description

Bit(s)	Name	Status	Function
<15:8>	Not used	NA	Read as zeros.
<7:0>	Output data	WO	These eight bits are an ASCII character transmitted as output when XCSR bit 7 is set.

1.8 BOOT AND DIAGNOSTIC FACILITY

The boot and diagnostic facility consists of two sockets that accommodate 8K, or 16K words of 16-bit ROM and another socket that can accommodate 2, 4, or 8 Kbytes of either 8-bit ROM or 8-bit EEPROM memory. The Control/Status Register (CSR), Page Control Register (PCR), maintenance register, Configuration and Display Register (CDR), and the boot and diagnostic facility are described in detail in Chapter 4.

The KDJ11-B module populates the first two sockets with 16K words of 16-bit ROM. This ROM contains the standalone diagnostics, the configuration routines, the memory diagnostics, and the boot programs for the various standard devices. This ROM can be replaced by the user for special purpose applications, but the user should include the configuration routines and the standalone diagnostics in the new ROM.

The remaining socket is normally populated with 2 Kbytes of 8-bit EEPROM containing the module configuration data and space for optional user provided boot programs. The configuration data uses 105 bytes of space. The remaining space is left available for user programs. This EEPROM can be replaced with 4 or 8 Kbytes of EEPROM and when the application does not require the erasable feature, the user can provide an 8 Kbyte ROM.

1.8.1 Control/Status Register (17 777 520)

The CSR allows the ROMs to test battery backup and reboot status. It can set the parameters for the Processor Mastership Grant (PMG) counter and the Line Time Clock (LTC). It also enables the console halt-on-break feature, and controls the entry and exit to the standalone mode. The CSR is shown in Figure 1-30 and is described in Table 1-21.

The CSR allows the programs to selectively disable the ROM response to addresses 17 765 000 through 17 765 776 and/or 17 773 000 through 17 773 776 and to control the read/write access to the EEPROM. Any program that accesses the I/O page can use the CSR to alter the PMG and the LTC parameters and also to control access to the ROM and EEPROM memory.

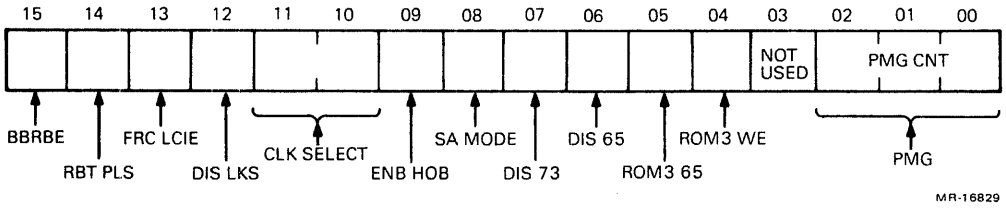


Figure 1-30 Controller Status Register

Table 1-21 Control/Status Register Bit Description

Bit(s)	Name	Function
15	Battery backup reboot enable	When this bit is set (1), it indicates that the battery backup voltage failed to maintain the memory system during the last power failure. When this bit is clear (0), it indicates that the battery backup voltage maintained the system memory during the last power failure or that there is no battery backup for the system.
14	Reboot pulse	This read-only bit is set (1) when the DCOK input is pulsed while the POK input remains asserted. This condition can only occur in Q22-Bus systems and indicates that a system reboot was requested by the control panel switch or by a special Q22-Bus device. This bit is cleared (0) by the negation of the POK input. A similar bit for Unibus systems is used in the KTJ11-B CSR.
13	Force line clock interrupt enable	This bit is set (1) to enable the clock selected by bits <11:10> to unconditionally request interrupts. When this bit is clear (0), the selected clock can only be enabled by setting bit 6 of the LTC register under program control to request interrupts. This bit is cleared by the negation of the DCOK input.
12	Line clock status register disable	This bit is set (1) to disable the LTC register. When cleared (0), the LTC is enabled and responds to address 11 777 546. This bit is cleared by the negation of the DCOK input.

Table 1-21 Control/Status Register Bit Description (Cont)

Bit(s)	Name	Function
<11:10>	Clock select	These read/write bits select the clock used by the LTC as an interrupt request. These bits are cleared by the negation of the DCOK input. The clock is selected as follows. **Bits 11 10 Clock Selection** 0 0 External BEVNT line 0 1 On-board 50 Hz* 1 0 On-board 60 Hz* 1 1 On-board 800 Hz
9	Enable Halt-on-Break	This read/write bit is set (1) to enable the console Serial Line Unit (SLU) halt-on-break feature. When clear (0), this feature is disabled. This bit is cleared by the negation of the DCOK input.
8	Standalone mode	This read/write bit is set (1) to enable the standalone mode by which the KDJ11-B operates using only the cache memory. The external memory and peripherals are disabled. When this bit is clear (0), the standalone mode is disabled and the system is operational. This bit is reset by the negation of the DCOK input.
7	Disable 17 773 000	When this read/write bit is set (1), the ROM memory addresses 17 773 000 through 17 773 776 are disabled. An external ROM responds to these addresses. When this bit is clear (0), the on-board ROMs respond to these addresses using the high byte of the PCR as the most significant address bits. This bit is cleared by the negation of the DCOK input.
6	Disable	When this read/write bit is set (1), the ROM memory addresses 17 765 000 through 17 765 776 are disabled. An external ROM responds to these addresses. When this bit is reset (0), the boot and diagnostic ROM memory is enabled. This allows the ROM memory selected by bit 5 of the CSR to use the low byte of the PCR as the most significant address bits. This bit is cleared by the negation of the DCOK input.
5	ROM socket 3 at 17 765 000	When this read/write bit is set (1), the 8-bit ROM in socket 3 responds to addresses 17 765 000 through 17 765 776, provided that bit 6 of the CSR is clear. When this bit is clear (0), the 16-bit ROM is selected to respond to these addresses. In either case, the low byte of the PCR provides the most significant bits of the address. This bit is cleared by the negation of the DCOK input.
4	ROM socket 3 write enable	When this read/write bit is set (1), and CSR bit 5 is set while CSR bit 6 is clear, the program can write to ROM socket 3, which normally contains the EEPROM. This bit is cleared by the power-up and initialize routines.

Table 1-21 Control/Status Register Bit Description (Cont)

Bit(s)	Name	Function
3	Not used	Read as zero.
<2:0>	Processor mastership grant count	These read/write bits are coded to select the length of time for the PMG counter to overflow. If any bit is set (1), the PMG counter begins counting whenever the KDJ11-B accesses the I/O page or external memory. When the counter overflows, the KDJ11-B has bus mastership during the next DMA arbitration cycle and it suppresses all DMA requests. When bits <2:0> are clear (0), the PMG counter is disabled, and the KDJ11-B is blocked from bus mastership as long as DMA requests are pending. These bits are cleared by the negation of DCOK.

PMG Count Bits

2	1	0	Count Time
0	0	0	Disabled†
0	0	1	0.4 μseconds
0	1	0	0.8 μseconds
0	1	1	1.6 μseconds
1	0	0	3.2 μseconds
1	0	1	6.4 μseconds
1	1	0	12.8 μseconds
1	1	1	25.6 μseconds

* Recommended for clock.
† The PMG count of zero (disabled) is not recommended for most systems and is reserved for special applications.

1.8.2 Page Control Register (17 777 522)

The PCR is a read/write register that is word and byte addressable. Only bits <14:9> and <6:1> can be used and the remaining bits are always read as zero. The high byte provides the most significant bits of the 16-bit ROM address (sockets 1 and 2) when accessed by bus addresses 17 773 000 through 17 773 776. The low byte provides the most significant bits of the 8-bit EEPROM or 16-bit ROM (socket 3) when accessed by bus addresses 17 765 000 through 17 765 776. The CSR register bits <7:4> control access to the ROM and EEPROM sockets and memory.

The PCR bits <14:9> are used as address bits <14:9> for addresses within 17 773 000 through 17 773 776 and the actual address bits <8:0> are used to form a 15-bit address for the ROM memory in sockets 1 and 2. The PCR bits <6:1> are used as address bits <14:9> for addresses within 17 765 000 through 17 765 766 and the actual address bits <8:0> are used to form a 15-bit address for the ROM memory in socket 3. When the 8-bit EEPROM is used in socket 3, the PCR bits <5:1> are used as address bits <13:9> for addresses within 17 765 000 to 17 765 776 and bits <8:0> of the actual address are used to form a 14-bit address. This register is cleared by the negation of the DCOK input.

1.8.3 Configuration and Display Register (17 777 524)

The CDR (Figure 1-31) consists of two independent registers: the read-only boot and diagnostic configuration register and the write-only boot and diagnostic display register. These registers are accessed by the same address and are described in the following paragraphs.

1.8.3.1 Boot and Diagnostic Configuration Register – This read-only register reflects the status of the configuration switchpack (switches 8 to 1) located on the module. The status of switches 5 to 8 can be remotely controlled via the J2 connector of the module and a cable connecting it to an external control panel. The status of switches 1 to 4 can be remotely controlled via the J3 connector.

1.8.3.2 Boot and Diagnostic Display Register – The display register allows the boot and diagnostic programs to control and update the eight LEDs mounted on the module. These LEDs can also be used to enable a display on an external control panel via the J2 connector. The register uses bits <7:0> to drive the LEDs. A 0 written into one these bits causes the respective LED to be illuminated. Writing a 1 into one of these bits turns off the respective LED. All bits <7:0> are cleared by the negation of DCOK and then all the LEDs are illuminated.

1.8.4 Maintenance Register (17 777 750)

The maintenance register is a 16-bit word that is read by the DCJ11-A during the power-up sequence, where the CPU executes a general purpose read. The word contains the power-up code, the halt/trap option bit, and the FPA (if this option is installed). This data is used by the DCJ11-A microprocessor; the remaining bits provide information on the module and system parameters for use by the operating system and the diagnostics. The read-only maintenance register is shown in Figure 1-32 and is described in Table 1-22.

The code for the power-up mode is hardwired as mode 2 to use the standard bootstrap operation. This sets the PSW to 340 and begins program execution at 173 000. This address starts the boot and diagnostic code that runs the standalone diagnostics before initiating the user specified, power-up option stored in the EEPROM as part of the configuration data. Since the bootstrap address is always the same, bits <15:9> of this register are not used to reference a bootstrap program and are now available to define some of the system parameters.

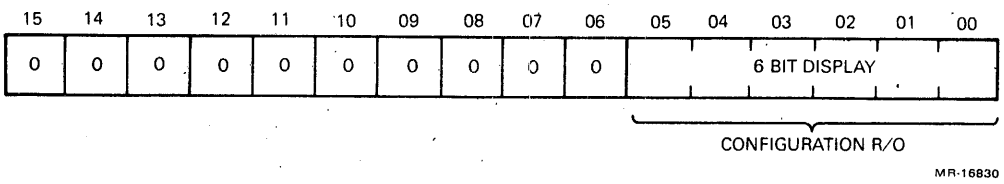


Figure 1-31 Configuration and Display Register

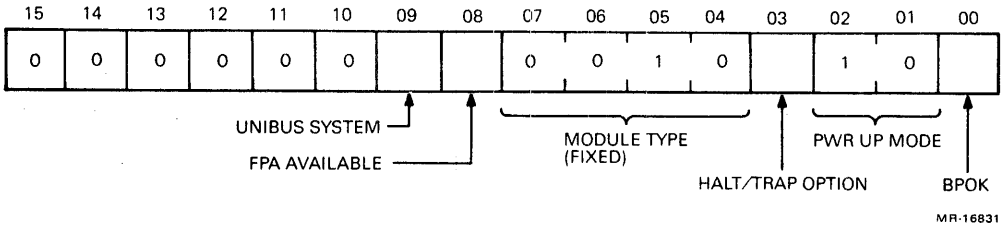


Figure 1-32 Maintenance Register

Table 1-22 Maintenance Register Bit Description

Bit(s)	Name	Function
<15:11>	Not used	Read as zeros.
10	Reserved	Reserved for future use.
9	Unibus system	This read-only bit monitors the status of the external Unibus adapter line. When set (1), it indicates that the system includes a Unibus adapter module. When reset (0), it indicates that it is a Q22-Bus system.
8	FPA available	This read-only bit is set (1) only if an FPA chip is installed on the module.
<7:4>	Module ID	The 0010 code identifies this module as a KDJ11-B microprocessor.
3	Halt/Trap option	This read/write bit determines how the HALT instruction is used in the kernel mode. When set (1), the trap option is selected causing the CPU to trap to location 4. When reset (0), the halt option is selected and the CPU halts and enters the console ODT mode. The trap option is not intended for normal use and is reserved for controller applications. This bit is cleared by the negation of the DCOK input and is set when the boot and diagnostic code selects the trap option by setting a bit in the configuration ROM.
<2:1>	Power-up code	These two bits are hardwired to always read as 2. At power-up, these bits cause the microprocessor to set the PSW to 340 and start executing the program at address 173 000. This is the starting address for the boot and diagnostic ROM program. These programs execute diagnostics to check the functions of the module and to implement the user's selected power-up routine as specified by the configuration data.
0	BPOK H	This bit is set (1), when the Q22-Bus signal BPOK H is asserted, indicating that the ac power is okay.

1.9 LINE TIME CLOCK

The LTC provides the system with a timing reference of fixed intervals that are determined by the Q22-Bus BEVNT line or by one of the on-board clock frequency signals, as programmed by bits <11:10> of the CSR. The BEVNT line cycles at the ac line frequency and produces intervals of 16.7 milliseconds for a 60 Hz input or 20.0 milliseconds for a 50 Hz input. The three on-board clock frequencies are 50 Hz, 60Hz, and 800 Hz. When enabled, these clocks are used to generate a program interrupt request with a priority level of BR6 and an interrupt vector address of 100. The on-board clock is recommended.

1.9.1 Line Time Clock Register (17 777 546)

The LTC register allows line clock interrupts to be enabled or disabled under program control. These line clock interrupts can be unconditionally enabled by setting bit 13 of the CSR register. Program recognition of the LTC register can be disabled by setting bit 12 of the CSR register. The normal configuration used for the KDJ11-B has both of these bits cleared. They are set by the boot and diagnostic ROM programs as selected by the configuration data. The LTC register is shown in Figure 1-33 and is described by Table 1-23.

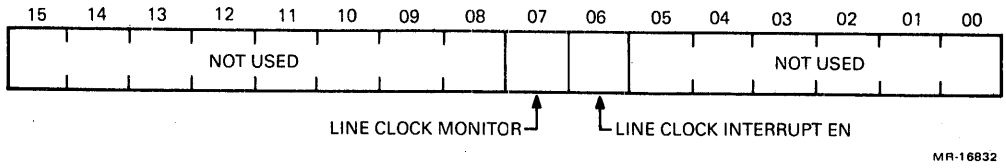


Figure 1-33 Line Time Clock Register

Table 1-23 Line Time Clock Register Bit Description

Bit(s)	Name	Function
<15:8>	Not used	Read as zeros.
7	LCM	The LCM bit is set (1) by the leading edge of the external BEVNT L signal, the bus INIT signal, or one of the three on-board clocks. The bit is cleared automatically by processor interrupt acknowledges. It is also cleared by writing a 0 into it.
6	LCIE	When the LCIE read/write bit is set (1), it allows the set condition of bit 7 to initiate a program interrupt request. When this bit is cleared (0), the line clock interrupts are disabled. This bit is forced when bit 13 of the Boot and diagnostic Control/Status Register (BCSR) is set. This bit is cleared by power-up and by the bus INIT signal.
<5:0>	Not used	Read as zeros.

CHAPTER 2
CONFIGURATION

2.1 INTRODUCTION

This chapter discusses the configuration requirements and other factors to consider when configuring the KDJ11-B module and installing it into an LSI-11 system. The module must be installed in a backplane that has the extended LSI-11 bus in the A/B rows and the interconnecting bus in the C/D rows. A 22-bit LSI bus utilizes the full capability of the module and the interconnecting bus is required because of the PMI feature of the module.

The H9278-A backplane is designed to accommodate the module in LSI-11 based systems. The H9277-A backplane is designed to accommodate the module and adapt it to a Unibus based system. The UniBus Adapter module (UBA) provides the interface requirements necessary to interface Unibus modules with the LSI-11 bus. In addition, the MSV11-J memory modules are designed to function with the PMI capability of the module.

The user must consider the following items to determine the configuration requirements for the module. If the module is installed in a prepackaged system, the user should be aware of the system components and their intended use.

1. Select the features controlled by the jumpers and switches located on the module.
2. Define the type of system and the mass storage devices being supported.
3. Select the desired configuration parameters available in the EEPROM.
4. Determine the bootstrap programs necessary to support the system.
5. Know the system differences if an existing system is being upgraded.

2.2 MODULE CONFIGURATION

The KDJ11-B module has 3 jumper wires and 8 switches mounted in a switchpack on the module as shown in Figure 2-1. Also, there are six red LEDs to monitor the diagnostic testing and a green LED to monitor the 5 Vdc applied to the module. The six red LEDs and switches 5 through 8 are wired to the J2 connector. Switches 1 through 4 are wired to the J3 connector. This allows the user to remotely monitor and control their status.

2.2.1 Jumper Wires

The 3 jumper wires are designated as W10, W20 and W40. The W10 jumper uses TP10 and TP11 to connect the on-board oscillator to the SLU. The W20 jumper is reserved and uses TP20, TP21 and TP22. The W40 jumper uses TP40, TP41 and TP42 to select the size of the EEPROM. The jumper is a push-on connector that provides a connection between two of the test points. The jumper configurations and uses are defined below and are summarized in Table 2-1.

2.2.1.1 W10 Jumper – The W10 jumper is used by manufacturing personnel to disable the on-board oscillator and connect an external oscillator during final test. This jumper is not optional to the user and is always used to connect TP10 to TP11.

2.2.1.2 W20 Jumper – The W20 jumper is always connected to TP20 and TP21.

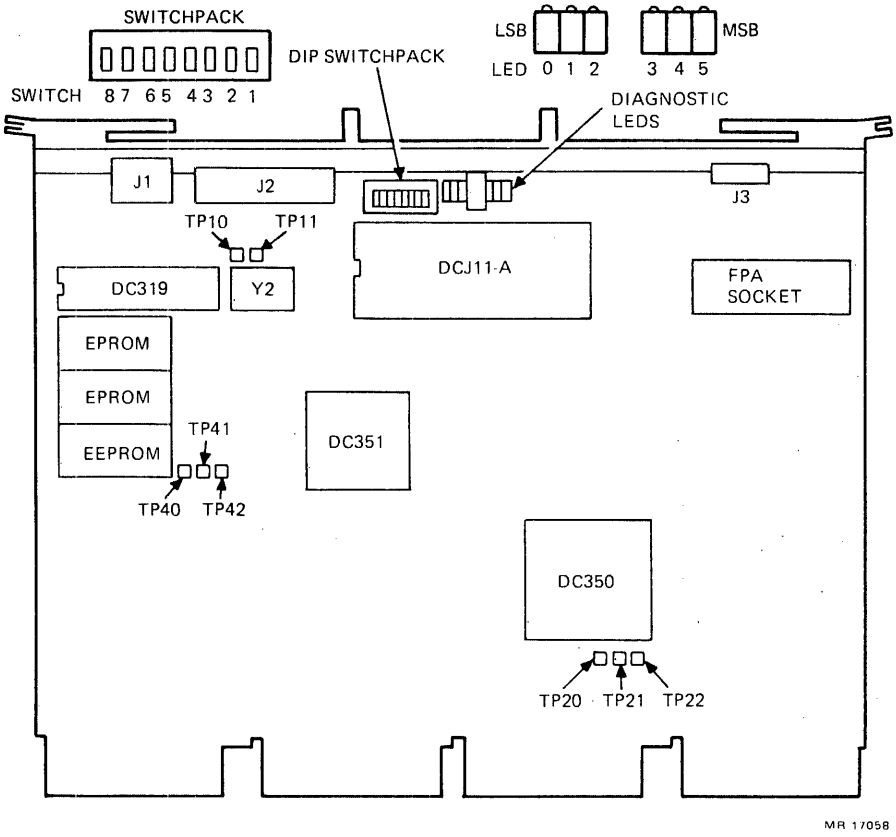


Figure 2-1 KDJ11-B Module Layout

Table 2-1 Jumper Wire Functions

Jumper	Function
W10 installed* (TP10 and TP11)	The on-board 614.4 kHz oscillator drives the SLU.
W10 removed (TP10 and TP11)	This condition is reserved for factory testing.
W20 installed* (TP20 and TP21)	The module is a single processor.
W20 installed (TP21 and TP22)	This condition is reserved.
W40 installed* (TP40 and TP41)	The standard 2K EEPROM is used.
W40 installed (TP41 and TP42)	The user selects an 8K EEPROM.

* Standard factory configuration.

2.2.1.3 W40 Jumper – The W40 jumper is used to provide an additional address bit when the user wishes to use an 8K EEPROM in place of the standard 2K EEPROM. When the jumper connects TP40 and TP41, the standard 2K EEPROM is installed in the socket, and when the jumper connects TP41 to TP42, the user can install either a 4K or 8K EEPROM in the socket. Note that when a 2K EEPROM is used, it is offset in the socket by two pins.

2.2.2 Switchpack

The switchpack contains 8 individual switches that are used to select the baud rate of the console SLU, a device bootstrap program, the dialog mode and the operating mode at power-up. The switches are wired to the read-only configuration register, where bit 7 corresponds to switch 1 and bit 0 corresponds to switch 8. A switch placed in the on position is read as a 0 and a switch placed in the off position is read as a 1.

The switches are wired to connectors J2 and J3 to allow the user to remotely monitor the status of the switches. They can also be remotely controlled or selected via these connectors. When using this mode, all the switches must be placed in the off position. Then the user can remotely turn a switch on by grounding the corresponding pin on the connector cabling. The pin identifications for the J2 and J3 connectors are shown in Figure 2-2 and are listed in Table 2-2. The functions controlled by the switches are described in the paragraphs that follow.

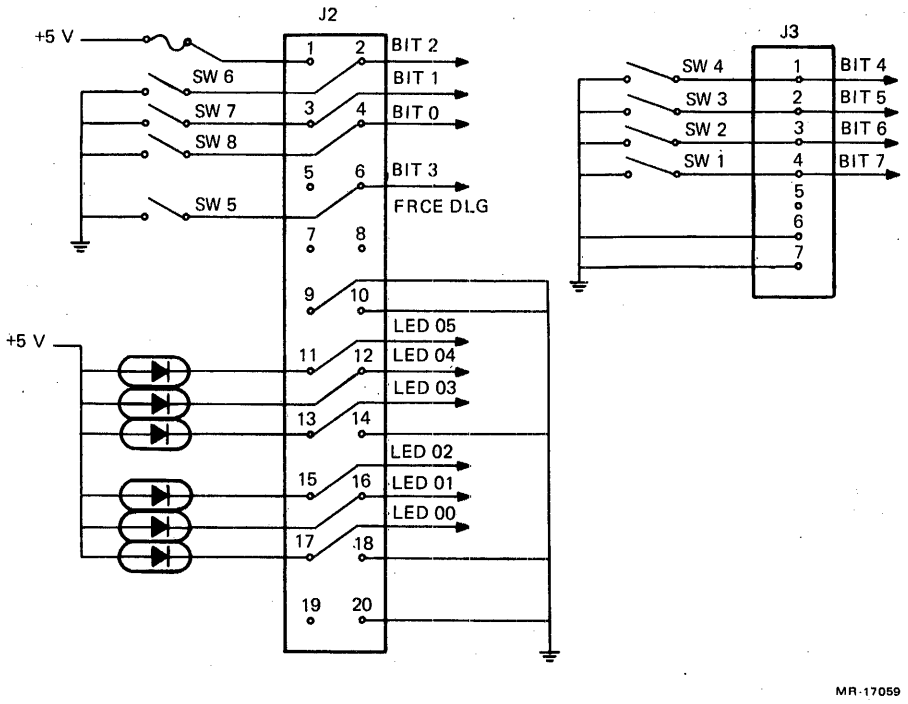


Figure 2-2 Pin Assignments for Connectors J2 and J3

Table 2-2 J2 and J3 Connectors

J2 Connector		J3 Connector	
Pin	**Signal**	**Pin**	**Signal**
1	+5.0 V fused	1	Switch 4
2	Switch 8	2	Switch 3
3	Switch 7	3	Switch 2
4	Switch 6	4	Switch 1
5	NC	5	NC
6	Switch 5	6	GND
7	NC	7	GND
8	NC		
9	GND		
10	GND		
11	LED 05 MSB		
12	LED 04		
13	LED 03		
14	GND		
15	LED 02		
16	LED 01		
17	LED 00 LSB		
18	GND		
19	NC		
20	GND		

2.2.2.1 Baud Rate Selection – The baud rate for the SLU is selected by switches 6, 7 and 8. The baud rate selections and the switch conditions are listed in Table 2-3.

Table 2-3 Baud Rate Selections

Switches			Register Bits			
6	**7**	**8**	**2**	**1**	**0**	**Baud Rate**
On	On	On	0	0	0	38,400
On	On	Off	0	0	1	19,200
On	Off	On	0	1	0	9,600
On	Off	Off	0	1	1	4,800
Off	On	On	1	0	0	2,400
Off	On	Off	1	0	1	1,200
Off	Off	On	1	1	0	600
Off	Off	Off	1	1	1	300

2.2.2.2 Dialog Mode – Setting switch 5 in the off position sets BCR register bit 3 and unconditionally forces the ROM code to enter the dialog mode when the diagnostic tests are completed. The switch in the off position also disables the automatic boot sequence and the bootstrap options associated with switches 2, 3, and 4. The dialog mode allows the user to establish a dialog with the system via the console terminal by using the six dialog commands that are described in Chapter 4. When switch 5 is set in the on position, register bit 3 is reset and the user can select any one of the power-up routines.

2.2.2.3 Device Bootstrap Programs – Up to six programs used to boot the mass storage devices in the system can be predetermined and stored in the EEPROM. The user determines the types of devices that need bootstrap programs and is able to select these programs by using setup command 4. This procedure is described in detail in Chapter 4. The following example shows the selections for typical LSI-11 and Unibus systems.

Boot Selection	LSI-11	System Device	Unibus	System Device
Boot 1	A	MSCP sniffer	A	MSCP sniffer
Boot 2	DL0	RL01/RL02	DL0	RL01/RL02
Boot 3	MS0	TSV05/TK25	MS0	TS11/TU80
Boot 4	E	End auto boot	E	End auto boot
Boot 5	Blank	Not used	Blank	Not used
Boot 6	Blank	Not used	Blank	Not used

When enabled by setup command 6, switches 2, 3 and 4 are used to select a single bootstrap program for a device in the system. When switches 2 through 4 are set from 1 to 6 as shown in Table 2-4, the ROM code enters the auto boot mode and attempts to boot only the device selected by the switches. If the boot is unsuccessful, the ROM code displays the normal error message and enters the dialog mode.

If all three switches are on and the console is enabled by switch 1 off, the module immediately powers up in the ODT mode. If the console is disabled by switch 1 on, the module loops on the standalone mode tests. If all three switches are off and the console is enabled by switch 1 off, the module powers up in dialog, automatic, ODT or 24 mode, as determined by the configuration parameter selected in the EEPROM. If all three switches are off and the console is disabled by switch 1 on, the ROM code automatically tries to boot the devices in the sequence determined by setup command 4.

NOTE

When the dialog mode is selected by switch 5, the bootstrap program options are disabled and the system enters the dialog mode.

Table 2-4 Bootstrap Program Selection

Switches			Register Bits			
2	3	4	6	5	4	EEPROM Boot
On	On	On	0	0	0	Determines power-up status
On	On	Off	0	0	1	Selects boot #1
On	Off	On	0	1	0	Selects boot #2
On	Off	Off	0	1	1	Selects boot #3
Off	On	On	1	0	0	Selects boot #4
Off	On	Off	1	0	1	Selects boot #5
Off	Off	On	1	1	0	Selects boot #6
Off	Off	Off	1	1	1	Determines boot status

2.2.2.4 Console Enable – The system console is enabled by setting switch 1 to the off position. This also sets a 1 in register bit 7. If the switch is in the on position, the system console is disabled and bit 7 is reset to 0. If the system console is disabled, the output line to the console is suppressed. Any input from the console results in an error message to inform the user that the console is disabled.

2.2.3 Diagnostic LEDs

The six red LEDs are mounted on the module edge in order to be visible to the user. They are also wired to connector J2 so they can be remotely monitored. The LEDs are coded as two octal numbers to represent the diagnostics or the system status listed in Table 2-5. A complete description of the diagnostics is in Chapter 4.

Table 2-5 Diagnostic and System Status LED Display

LED Code	Diagnostic
77	CPU or halt switch
76	CPU and MMU
75	Turn on MMU, run CPU and MMU
74	Turn on PMI, check UBA reboot bit
73	Power-up to mode 2: ODT
72	Power-up to mode 3: 24
71	EEPROM checksum
70	CPU ROM checksum and PCR
67	Miscellaneous CPU and EIS
66	Console SLU test 1
65	Console SLU test 2
64	Console SLU test 3
63	MMU aborts
62	Cache memory
61	Line clock
60	Floating-point instruction
57	Reserved
56	Exit standalone mode
55	UBA register response
54	Memory sizing routine
53	Memory location 0
52	Memory locations 0 to 4K words
51	Cache operation with memory
50	Complete memory data/byte exercise
47	Memory parity/ECC
46	Memory address shorts
45	UBA boot ROM
44	UBA map registers data path
43	UBA unmapped diagnostic data
42	UBA mapped diagnostic data
41	UBA floating address/data
40	UBA address overflow
37	UBA cache data
36	UBA cache LRU
35	UBA cache tag store
34	UBA cache parity error
33	Complete Unibus memory data/byte exercise
32	Unibus memory parity
31	Unibus memory address shorts
30	Exit

Table 2-5 Diagnostic and System Status LED Display (Cont)

LED Code	System Status
27	Not used
26	Not used
25	Not used
24	DECNET boot (DLV11-E/F, DUV11) waiting for host reply
23	XON not received after XOFF, type <CTRL> Q to correct
22	Xmit ready bit does not set
21	Drive error
20	Controller error
17	Invalid boot device selection (i.e., AA)
16	Invalid unit number selection
15	Nonexistent drive
14	Nonexistent controller
13	No tape
12	No disk
11	Invalid boot block
10	Drive not ready
07	No bootable device found in automatic boot mode
06	Console disabled by switch 1 on, and no force dialog or APT break received; ROM code has entered ODT for APT
05	Not used
04	Dialog mode
03	UBA ROM boot in progress
02	EEPROM boot in progress
01	CPU ROM boot in progress
00	Start secondary boot with display blanked

2.3 EEPROM CONFIGURATION PARAMETERS

The general configuration parameters are stored in the EEPROM and can be modified or changed by the user to meet the requirements necessary for the intended use of the module. The user can determine the parameters by entering the dialog mode and selecting the setup command. The dialog mode is entered by setting switch 5 to the off position and powering up the system, or by typing a <CTRL> C while running the diagnostics. The setup mode is entered by typing S followed by <RETURN>. The system then prints a list of all the setup commands with a short description of each. The second command provides a list of all the parameters available and the current status of these parameters. The second command is selected by typing 2 and pressing <RETURN>. The system then prints a list of the parameters as shown in Table 2-6. The last column provides the current status. The status described represents the default status.

When the setup mode command 2 is executed and the list of parameters with the current status is printed out, the first parameter is repeated for the user's approval or change. To change the parameter, the user types in the new value and presses the Return key. Note that the first two parameters apply only to LSI-11 systems and the last four parameters apply only to Unibus systems. Therefore, the parameters printed out are dependent on the type of system, but they are printed sequentially from A to B to C, etc.

The user can select the next parameter by pressing <RETURN> repeatedly until the parameter to be changed is reached. Another way to reach the desired parameter is to type the item reference letter (Table 2-6) and press the Return key. The user can proceed to the next parameter by pressing <Return>, <Line Feed> or < . >, and can return to the preceding parameter by typing < ^ > or < - >. The setup mode command 2 is exited by typing <CTRL> Z.

Table 2-6 Configuration Parameters

Item	Parameter	Selections	Status
A	Enable halt-on-break	(0) = No (1) = Yes	= 1
B	Disable user friendly format	(0) = No (1) = Yes	= 1
C	ANSI video terminal	(0) = No (1) = Yes	= 1
D	Power-up	(0) = Dialog (1) = Automatic (2) = ODT (3) = 24	= 1
E	Restart	Same as power-up	= 1
F	Ignore battery	(0) = No (1) = Yes	= 0
G	PMG count	Select from 0–7	= 7
H	Disable clock CSR	(0) = No (1) = Yes	= 0
I	Force clock interrupts	(0) = No (1) = Yes	= 0
J	Clock frequency	(0) = Power supply (1) = 50 Hz (2) = 60 Hz (3) = 800 Hz	= 0
K	Enable EEC test	(0) = No (1) = Yes	= 1
L	Disable long memory test	(0) = No (1) = Yes	= 0
M	Disable ROM	(0) = No (1) = Disable 165 (2) = Disable 173 (3) = Disable both	= 0
N	Enable trap-on-halt	(0) = No (1) = Yes	= 0
O	Allow alternate boot block	(0) = No (1) = Yes	= 0
P	Disable setup mode	(0) = No (1) = Yes	= 0
Q	Disable all testing	(0) = No (1) = Yes	= 0
R	Enable Unibus memory test	(0) = No (1) = Yes	= 1
S	Disable UBA ROM	(0) = No (1) = Yes	= 0
T	Enable UBA cache	(0) = No (1) = Yes	= 1

Table 2-6 Configuration Parameters (Cont)

Item	Parameter	Selections		Status
U	Enable 18-bit mode	(0) = No	(1) = Yes	= 0
Type <CTRL> Z to exit or press <RETURN> to proceed.				
A	Enable halt-on-break	(0) = No	(1) = Yes	= 1

2.3.1 Enable Halt-on-Break

When this parameter is set to 1, it enables the processor to halt if the console SLU detects a break condition. When it is reset to 0, the processor ignores console break conditions. This parameter is enabled only when the first break or valid character is received after the system is powered up or restarted. This parameter only applies to LSI-11 systems. The Unibus systems enable the halt-on-break parameter by setting the front panel key switch to the enable position.

2.3.2 Disable User Friendly Format

The type of messages sent to the console during power-up is determined by this parameter. Both conditions of this parameter provide user friendly messages, but the messages are more friendly when the parameter is set to 0. The user friendly mode is normally used when the automatic boot mode is selected. The standard format is always selected for Unibus systems. This parameter is only used for LSI-11 systems.

2.3.3 ANSI Video Terminal

When this is set to 1, it indicates that the console terminal is an ANSI video terminal. If it is reset to 0, the console terminal must be a hard-copy terminal or a non-ANSI video terminal. When an ANSI video terminal is selected, use of the Delete key erases the previous character on the screen. This is accomplished by the ROM code sending a backspace, a space and then another backspace to the console terminal. If a hard-copy terminal is selected, the Delete key is interpreted by the ROM code as a slash character and the deleted character is identified by the slash character. When the system is powered up and an ANSI video terminal is selected, the ROM code clears the video screen and positions the cursor at line 9 and column 1. This parameter is only used by the ROM code and not by the operating system.

NOTE

A VT52 terminal is not an ANSI video terminal and if a VT52 is selected, the parameter must be set to 0 in order to prevent the clear screen command from locking up the VT52.

2.3.4 Power-Up Modes

There are four power-up mode selections available to the user. When the system is started, the ROM code checks a status bit to determine if the system is powering up or being restarted. The ROM code then checks the status of the selected parameter and uses the mode selected for that parameter. The same modes are used by both the power-up parameter and the restart parameter.

2.3.4.1 Dialog Mode – After the diagnostics are completed, the ROM code enters the dialog mode. This mode is selected by keying in a 0.

2.3.4.2 Automatic Mode – At the completion of the diagnostics, the ROM code enters the automatic boot routine and tries to boot the predetermined device or devices. The devices are previously selected and loaded into the EEPROM. The user can select up to six individual devices to be automatically booted. The system attempts to sequentially boot the devices on the list until a device is successfully booted or the end of the list is reached. The factory setting or default list consists of A, DL0 and MS0. The A device is a special mnemonic letter that tries to boot a Mass Storage Control Protocol (MSCP) device in the range of 0 to 7. This mode is selected by keying in a 1.

2.3.4.3 ODT Mode – In this mode, a limited set of diagnostics is run and the ROM code executes a halt instruction and passes control to the DCJ11-A micro-ODT code. The user can continue the diagnostic testing and enter the dialog mode by typing P, as long as none of the register data was changed. This mode is normally used only for debugging and is selected by keying in a 2.

2.3.4.4 Mode 24 – After a limited set of diagnostics is run, the ROM code loads the contents of location 26 into the PSW and then transfers control to the address referenced by the contents of location 24. This mode is used when the memory uses battery backup or when nonvolatile memory is present and it is necessary to recover from a power fail condition. This mode is selected by keying in a 3.

2.3.5 Restart

The selections for the restart mode are identical to those used in the power-up mode. However, these selections are independent of each other and the selection for the restart mode can be different than that selected for the power-up mode.

2.3.6 Ignore Battery

This parameter is used in conjunction with mode 24 during power-up or restart. When the user selects 0, the battery OK signal must be present in order to execute mode 24. When a 1 is selected, mode 24 is executed regardless of the battery status. If this parameter is reset to 0, and the battery OK signal is not present for power-up mode 24, the restart mode then determines the action taken. If the restart mode is also mode 24, the system defaults to the dialog mode.

2.3.7 PMG Count

This parameter has a range of 0 to 7, and it determines the value of the PMG counter in the BCSR. When a 0 is selected, the counter is disabled. The counter enables the KDJ11-B to suppress DMA requests and make the processor the busmaster during the next DMA arbitration cycle after the counter overflows. The different values of the PMG counter are listed below. This parameter is normally set to 7, and it is recommended that the 0 value not be used because it may cause erratic operation.

Value	Overflow Time
0	Disabled
1	0.4 μseconds
2	0.8 μseconds
3	1.6 μseconds
4	3.2 μseconds
5	6.4 μseconds
6	12.8 μseconds
7	25.6 μseconds

2.3.8 Disable Clock CSR

The LTC status register DIS LKS function is controlled by the status of this parameter. The BCSR register uses bit 12 for this feature and when the parameter is set to 1, the feature is disabled. When reset to 0, the bit is cleared and the LTC register responds with address 17 777 546. This parameter is normally set to 0.

2.3.9 Force Clock Interrupts

FRC LCIE bit 13 of the BCSR register is controlled by this parameter. When set to 1, the clock unconditionally requests interrupts provided that the priority of the processor is 5 or less. When reset to 0, the clock can request interrupts only when the clock CSR is enabled, bit 6 of the LTC register is set to 1, and the priority of the processor is 5 or less. This parameter is normally reset to 0. If this parameter is set to 1, it is recommended that the user disable the clock CSR parameter.

2.3.10 Clock Select

This parameter selects the frequency used to drive the clock interrupts. This function is also controlled by bits <11:10> of the BCSR register. The 4 selections are described in the following chart.

Select	Clock Frequency
0	External BEVNT LSI-11 bus input
1	Internal 50 Hz clock
2	Internal 60 Hz clock
3	Internal 800 Hz clock

2.3.11 Enable ECC Test

The ECC memory test is enabled when this parameter is set to 1. This test uses bit 4 of the memory CSR to determine if the type of memory is ECC or parity. The test is automatically aborted if parity memory exists. The test tries to read and write into bit 4 and if it is successful, the ROM code assumes that it is an ECC memory. When the parameter is reset to 0, the ECC test is bypassed. Normally the ECC test is always enabled – even when the system only has parity memory. The ECC test is never used for Unibus memory.

2.3.12 Disable Long Memory Test

This test checks the memory addresses for shorts in reading data from memory. When the parameter is set to 1, the tests are limited to only the first 256 Kbytes of memory, and if it is reset to 0, the test is executed on all available memory. This parameter is normally reset to 0 to test all the memory for the system.

NOTE

If the long memory test is disabled and parity type memory exists above the 256 Kbyte limit, then parity errors are very likely to occur in the memory above 256 Kbytes.

2.3.13 Disable ROM

The user is allowed to selectively disable all or part of the boot ROM code. The ROM code uses two pages of 256 words in the I/O page. One page responds to the 17 773 000 through 17 773 777 addresses and the other page responds to the 17 765 000 through 17 765 777 addresses. These pages are automatically enabled during power-up and restart. One or both of these pages can be disabled by the ROM code. The user can select a choice as described below, but normally none of the pages are disabled.

Value	Disabled ROM Pages
0	None*
1	Page 17 765 00†
2	Page 17 773 00†
3	Both pages

* Default.
† Recommended.

2.3.14 Enable Trap-on-Halt

When this parameter is set to 1, the processor traps to location 4 when a halt instruction is executed in the kernel mode. If the parameter is reset to 0, then the processor enters the DCJ11-A micro-ODT mode when a halt instruction is executed in the kernel mode. Normally this parameter is reset to 0.

2.3.15 Allow Alternate Boot Block

During the boot process, the boot block of the device is loaded into memory and the ROM code checks location 0 to determine if the device is bootable. If set to 0 and the data is incorrect, the ROM code types out an error message indicating that the device is not bootable. When this parameter is set to 1, the ROM code checks location 0 for any value other than 0. If it is reset to 0, the ROM code checks location 0 for a value within the range 240 to 277 and then checks location 2 for a value within the range 400 to 777. This parameter is normally reset to 0 for standard bootstraps, but may be set to 1 to allow the proper booting requirements for some users' operating systems.

2.3.16 Disable Setup Mode

This parameter is used to enable the user to enter the setup mode from the dialog mode when set to 0, and it disables this feature when set to 1. The setup mode command is not available. This parameter is set to 1. Setup mode is unconditionally enabled, regardless of how this parameter is set, if the force dialog mode is enabled. This parameter is used to prevent unauthorized entry into the setup mode and assumes that the forced dialog switch or switch 5 on the module is on to prevent entry into forced dialog mode. This parameter is only available for the V7.0 version of the ROM code.

2.3.17 Disable All Testing

This parameter is set (1) to prevent the diagnostic testing by the ROM code, provided that the forced dialog mode is not selected. The ROM code does not change any memory locations unless they are changed by the selected boot program. This parameter was installed because there are times when the user needs an almost immediate response at power-up or needs the memory to remain unaltered. This parameter should only by used when necessary.

NOTE

If the testing is disabled and memory parity is being used, then memory parity errors are possible after power-up.

2.3.18 Enable Unibus Memory Test

This parameter is set (1) specifically to test any Unibus memory in the system and is disabled when reset to 0. This feature is only used in Unibus systems.

2.3.19 Disable UBA ROM

This parameter is used to control the ROMs located on the UBA by copying its status into bit 3 of the UBA Diagnostic Control/Status Register (DCSR) after a normal boot. When it is set to 1, the ROMs on the UBA are disabled and ROMs located on other Unibus modules are enabled. If it is reset to 0, the UBA ROMs are enabled. This parameter is normally reset to 0, and only applies to Unibus systems. It is ignored when the user tries to boot the UBA or the M9312 boot ROMs.

2.3.20 Enable UBA Cache

The cache located on the UBA is enabled and tested by the ROM code when this parameter is set to 1. If a failure occurs during the testing of the cache, then the cache is disabled. When the parameter is reset to 0, the UBA cache is always disabled. This parameter is normally set to 1 and is only used for Unibus systems.

2.3.21 Enable 18-Bit Mode

This parameter is used to select 18- or 22-bit addressing modes and its status is copied into bit 5 of the UBA KTJ11 Memory Configuration Register (KMCR). When set to 1, the memory uses 18-bit addressing and when reset to 0, the memory uses 22-bit addressing. This parameter is normally reset to 0, and is only used for Unibus systems.

2.4 SYSTEM INSTALLATION

The KDJ11-B module can be used in any system that incorporates a backplane with the extended LSI-11 bus in rows A and B, and the interconnecting bus in rows C and D. The Micro PDP-11/73 system is a prepackaged LSI-11 based system that uses the H9278-A backplane. The PDP-11/84 system is a prepackaged Unibus system that uses the H9277-A backplane. Both of these systems use the MSV11-J memory modules as a private memory and utilize the PMI feature of the KDJ11-B module.

2.4.1 LSI-11 Based Systems

An LSI-11 based system can be custom designed using the KDJ11-B module and compatible LSI-11 components. These systems can incorporate the PMI feature by using the MSV11-J memory modules. A list of compatible LSI-11 options is provided in Table 2-7.

NOTE

It is recommended that the ac and dc loading for the final configuration be checked for conformance with the LSI-11 bus loading rules. It is also recommended to check for overloading on the +5 V and +12 V power supplies.

Table 2-7 LSI-11 Compatible Options

Name	Option	Identification
Backplanes		
H9276	4 × 9	LSI-11/CD
H9278-A	4 × 8	3 LSI-11/CD slots and 5 LSI-11/LSI-11 slots
Memory		
MCV11-D	M8631	CMOS nonvolatile memory
MSV11-L	M8059	MOS memory
MSV11-J	M8637	MOS memory
MSV11-P	M8067	MOS memory
MSV11-Q	M7551	MOS memory
MRV11-D	M8578	PROM/ROM module
Options		
AAV11-C	A6008	D/A converter
ADV11-C	A8000	A/D converter
AXV11-C	A0026	D/A and A/D combination converter
DEQNA-K	M7504	Adapter, Q22-Bus to Ethernet
DLV11	M7940	Asynchronous serial line interface
DLV11-E	M8017	Asynchronous serial line interface
DLV11-F	M8028	Asynchronous serial line interface
DLV11-J	M8043	Four asynchronous serial line interfaces (CS Rev. E or later, ECO M8043–MR002 installed)
DZQ11-M	M3106	4-line asynchronous multiplexer
DLVJ1-M	M8043	4-line asynchronous multiplexer
DHV11-M	M3104	8-line asynchronous multiplexer

Table 2-7 LSI-11 Compatible Options (Cont)

Name	Option	Identification
DMV11-AC	M8053-MA	Synchronous communications interface
DMV11-AF	M8064-MA	Synchronous communications interface
DPV11	M8020	Programmable synchronous EIA line
DRV11	M7941	Parallel interface
DRV11-J	M8049	Parallel interface
DUV11	M7951	Programmable synchronous EIA line
DZV11	M7957	4-line asynchronous EIA multiplexer
IBV11-A	M7954	IEEE instrument bus interface
KLESI-Q	M7740	Adapter, Q22-Bus to LESI
KPV11-A	M8016	Power-fail and LTC generator (KPV11-B and -C are not compatible)
KWV11-C	M4002	Programmable real-time clock
LAV11	M7949	LA180 line printer interface
LPV11	M8027	LA180/LP05 printer interface
RLV12	M8061	RL01/2 controller
RQDX1	M8639	MSCP controller for RX50 floppy disk and RD51 Winchester drive
KDA50-QA	M7164/ M7165	Adapter, Q22-Bus to SDI
RDQX1-E	M7512	RD/RX extender
RXV21	M8029	RX02 controller
TQK25	M7605	TK25 controller
TQK50	M7503	TK50 controller
TSV05	M7196	Magnetic tape interface
RQDX2	M8639-YB	Controller, MicroPDP-11 systems
RQDX3-M	M7555	Controller, MicroPDP-11 and MicroVAX II systems
Bus Cable Cards		
M9404		Cable connector
M9405		Cable connector

When building a custom LSI-11 system, the placement of the KDJ11-B module is dependent upon the use of the MSV11-J module and the following rules.

1. When two MSV11-J modules are used in the system, they are inserted into the first two slots, and the KDJ11-B module is inserted into the third slot of the backplane.

2. When one MSV11-J module is used in the system, it is inserted into the first slot, and the KDJ11-B module is inserted into the second slot of the backplane.

3. If no MSV11-J modules are used, the KDJ11-B module is inserted into the first slot of the backplane.

Any additional LSI-11 options are inserted below the KDJ11-B module and must conform to the following rules.

1. All options must be inserted below the KDJ11-B module.

2. No dual options may be inserted in the C and D rows used as the interconnecting bus.

3. Dual options can only be inserted in slots designated as A and B rows.

4. Any open A rows, or C and D rows of a slot must be filled with an M9047 grant module if any modules follow the grant chain.

5. The terminating resistors on the backplane should be removed when using an extended backplane system.

2.4.2 Restricted LSI-11 Systems

There are many LSI-11 options that are not compatible because they were designed primarily for 16- and 18-bit systems or for a particular application. The LSI-11 options not compatible are backplanes, memories, or I/O devices that are not capable of 22-bit addressing. They may generate or decode erroneous addresses if used in systems that implement 22-bit addressing. Memory and memory addressing devices that implement only 16- or 18-bit addressing may be used in a 22-bit backplane, but the size of the system memory must be restricted to the address range of these devices (64 Kbytes for systems with a 16-bit device, 256 Kbytes for systems with an 18-bit device). Consider the following when adding restricted LSI-11 options to the system.

1. The option must not use pins BC1, BD1, BE1 or BF1 except as the required BDAL 18–21 connections. Some early LSI-11 options were allowed to use these pins as test points or for user provided interconnections.

2. If the option is a DMA device, it must support the full 22-bit addressing requirement.

3. If the option responds to non-I/O page addresses, it must also decode the BDAL 18–21 address lines as part of the address.

4. The power requirements for each option must be considered to avoid overloading the power supply.

5. The switching and electrical parameters of the option must conform to the LSI-11 specification of DEC STD 160.

6. The speed differences between the KDJ11-B and other Q22-Bus processors, operating systems, and diagnostics may cause problems.

NOTE

DMA devices having 18 bits can potentially work in a 22-bit system by buffering I/O in the 18-bit address space.

2.4.3 Unibus Based Systems

A Unibus based system can be custom designed by using the KDJ11-B CPU module, the MSV11-J memory module, the KTJ11-B UBA and compatible Unibus components. This type of system must be installed in the H9277-A backplane and incorporates the PMI feature. A list of compatible Unibus modules is provided in Table 2-8.

The following requirements must be considered when adding a Unibus option to the system.

1. The timing and electrical parameters of the option must conform to the Unibus specification of DEC STD 158.

2. The ac and dc loading of the Unibus must be within the allowable specifications.

3. The power requirements must not exceed the ratings of the power supply and hardware. The speed differences between Unibus processors, diagnostics, and operating systems may cause problems.

Table 2-8 Unibus Compatible Options

Name	Identification
Communications	
DHU11-AP	16-line DMA multiplexer
DL11-XP	Single channel modem
DN11-XP	Auto dial unit
DUP11-AP	Synchronous interface
DMP11-XP	Synchronous multidrop interface
DMR11-XP	DDCMP interface
DEUNA-AA	Ethernet interface
KMC11-MP	I/O processor
KMS11-BX	X.25 packet switch interface
DZS11-EA	Statistical multiplexer
Disk and Tape	
RC25-XA	
RA60-CX	
RA81-AX/CX	
RA80-AX/CX	
TU80-AX	
TU81-AX	
TK50	

Table 2-8 Unibus Compatible Options (Cont)

Name	Identification
Options	
DH11-XP	16-channel DMA multiplexer
DZ11-XP	16-channel multiplexer
DMC11-AL, AR, DA	DDCMP channel interface
DR11-B	DMA parallel interface
DR11-K	Digital I/O bit interrupt
IEC11-AB, BA	IEEE interface
KG11-A	CRC arithmetic unit
KMC11-B	I/O processor
DV11-AP, 1P, 2P, 3P	Synchronous multiplexer
KW11-A	Dual programmable clock
PCL11-B	Interprocessor bus
RX211-BX	
RH11	RP05, 6 and RM02 disks
RK611	RK07 disk
RL211-AK	
TS11-AA	Tape unit
TJE16	
TJU77	
AA11-KT	Real-time I/O
AD11-KT	
AM11-K	
AR11-KT	
RA80	
TU58-DX	
DB11-MP	Unibus repeater
BA11-K, L	Expansion boxes
BA23-CC/CD	FCC expander cabinet
H9642-XX	Option cabinet

2.5 MODULE CONTACT FINGER IDENTIFICATION

The LSI-11 type modules, including the KDJ11-B, all use the same contact (pin) identification system. The contacts used on a quad-height module are identified in Figure 2-3. The LSI-11 bus signals are assigned to rows A and B, each with 18 contacts on the component side and the solder side. The KDJ11-B bus signals are identified along with the LSI-11 bus signals in Table 2-9 and the pins are identified as follows.

AE2	Module side, identifier side (solder side) Pin identifier (Pin E) Row identifier (Row A)

The positioning notch between the two rows of pins mates with a protrusion on the connector block for correct module positioning. A complete description of the backplane and bus operation is provided in Chapter 6.

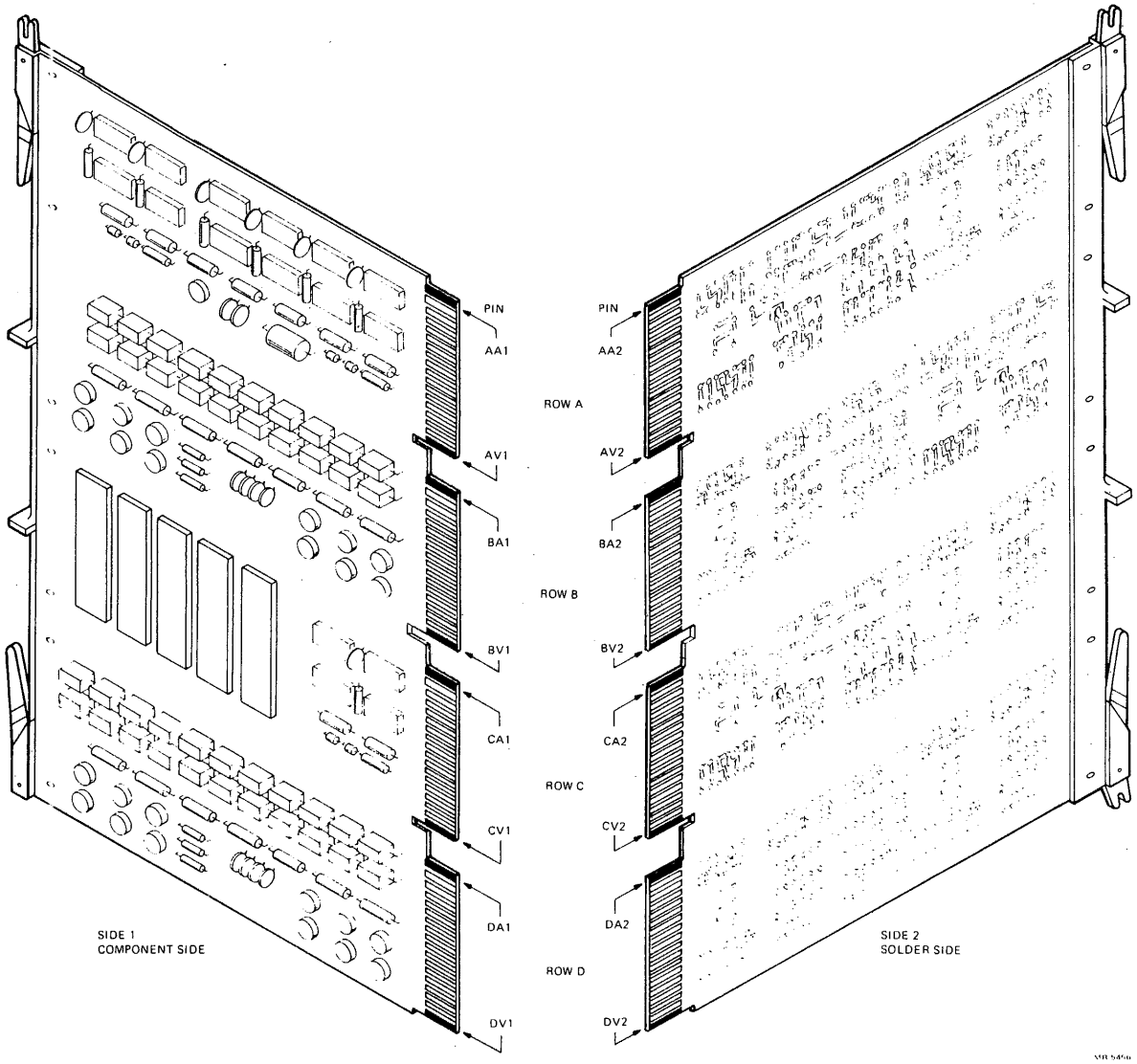


Figure 2-3 KDJ11-B Module Contacts

Table 2-9 KDJ11-B Module and LSI-11 Bus Signals

Component Side			Solder Side		
Pin	**LSI-11 Bus**	**KDJ11-B**	**Pin**	**LSI-11 Bus**	**KDJ11-B**
AA1	BIRQ 5 L	BIRQ 5 L	AA2	+5 V	+5 V
AB1	BIRQ 6 L	BIRQ 6 L	AB2	−12 V	Not used
AC1	BDAL 16 L	BDAL 16 L	AC2	GND	GND
AD1	BDAL 17 L	BDAL 17 L	AD2	+12 V	Not used
AE1	SSPARE 1	Not used	AE2	BDOUT L	BDOUT L
AF1	SSPARE 2	SRUN L*	AF2	BRPLY L	BRPLY L
AH1	SSPARE 3	Not used	AH2	BDIN L	BDIN L
AJ1	GND	GND	AJ2	BSYNC L	BSYNC L
AK1	MSPARE A	Not used	AK2	BWTBT L	BWTBT L
AL1	MSPARE A	Not used	AL2	BIRQ 4 L	BIRQ 4 L
AM1	GND	GND	AM2	BIAKI L	Not used
AN1	BDMR L	BDMR L	AN2	BIALO L	BIAK L
AP1	BHALT L	BHALT L	AP2	BBS 7 L	BBS 7 L
AR1	BREF L	Not used	AR2	BDMGI L	Not used
AS1	+12 V	Not used	AS2	BDMGO L	BDMGO L
AT1	GND	GND	AT2	BINIT L	BINIT L
AU1	PSPARE 1	Not used	AU2	BDAL 0 L	BDAL 0 L
AV1	+5 V	+5 V	AV2	BDAL 1 L	BDAL 1 L
BA1	BDCOK H	BDCOK H	BA2	+5 V	+5 V
BB1	BPOK H	BPOK H	BB2	−12 V	Not used
BC1	SSPARE 4	BDAL 18 L	BC2	GND	GND
BD1	SSPARE 5	BDAL 19 L	BD2	+12 V	Not used
BE1	SSPARE 6	BDAL 20 L	BE2	BDAL 2 L	BDAL 2 L
BF1	SSPARE 7	BDAL 21 L	BF2	BDAL 3 L	BDAL 3 L
BH1	SSPARE 8	Not used	BH2	BDAL 4 L	BDAL 4 L
BJ1	GND	GND	BJ2	BDAL 5 L	BDAL 5 L
BK1	MSPARE B	Not used	BK2	BDAL 6 L	BDAL 6 L
BL1	MSPARE B	Not used	BL2	BDAL 7 L	BDAL 7 L
BM1	GND	GND	BM2	BDAL 8 L	BDAL 8 L
BN1	BSACK L	BSACK L	BN2	BDAL 9 L	BDAL 9 L
BP1	BIRQ 7 L	BIRQ 7 L	BP2	BDAL 10 L	BDAL 10 L
BR1	BEVNT L	BEVNT L	BR2	BDAL 11 L	BDAL 11 L
BS1	PSPARE 4	Not used	BS2	BDAL 12 L	BDAL 12 L
BT1	GND	GND	BT2	BDAL 13 L	BDAL 13 L
BU1	PSPARE 2	Not used	BU2	BDAL 14 L	BDAL 14 L
BV1	+5 V	+5 V	BV2	BDAL 15 L	BDAL 15 L

* The SRUN L signal is primarily used to drive a panel run light indicator. It is used for BA11-N and later systems. It indicates that the processor is executing instructions.

The KDJ11-B module also uses rows C and D in the backplane for the PMI feature. The C and D rows provide an interconnection between modules placed in adjacent slots. The signals assigned to the C and D rows are identical for the KDJ11-B CPU module, the MSV11-J memory module and the KTJ11-B UBA. The module signals are identified in Table 2-10.

Table 2-10 Module PMI Signal Assignments

Component Side		Solder Side	
Pin	**KDJ11-B**	**Pin**	**KDJ11-B**
CA1	Not used	CA2	+5 V
CB1	PSSEL L	CB2	Not used
CC1	SRUN L	CC2	GND
CD1	PUBMEM L	CD2	Not used
CE1	PBCYC L	CE2	Not used
CF1	PUBSYS L	CF2	Not used
CH1	PHBPAR L	CH2	Not used
CJ1	PSBFUL L	CJ2	Not used
CK1	PLBPAR L	CK2	Not used
CL1	Not used	CL2	Not used
CM1	PRDSTB	CM2	Not used
CN1	Not used	CN2	Not used
CP1	PBLKM L	CP2	Not used
CR1	PBSY L	CR2	Not used
CS1	Not used	CS2	Not used
CT1	GND	CT2	Not used
CU1	Not used	CU2	Not used
CV1	PUBTMO L	CV2	Not used
DA1	Not used	DA2	+5 V
DB1	PWTSTB L	DB2	Not used
DC1	PBYT L	DC2	GND
DD1	PMAPE L	DD2	Not used
DE1	Not used	DE2	Not used
DF1	Not used	DF2	Not used
DH1	Not used	DH2	Not used
DJ1	Not used	DJ2	Not used
DK1	Not used	DK2	Not used
DL1	Not used	DL2	Not used
DM1	Not used	DM2	Not used
DN1	CNSL LOCK L	DN2	Not used
DP1	Not used	DP2	Not used
DR1	Not used	DR2	Not used
DS1	Not used	DS2	Not used
DT1	GND	DT2	Not used
DU1	Not used	DU2	Not used
DV1	Not used	DV2	Not used

2.6 MODULE INSTALLATION PROCEDURE

Certain guidelines should be followed when installing or replacing the KDJ11-B module or any LSI-11 option used in the system. They are as follows.

1. Verify dc power before inserting the module in a backplane.
2. Ensure that no dc power is applied to the backplane when removing or inserting the module.
3. Verify the configuration of the module jumpers.
4. Insert the KDJ11-B module into the backplane with the component side facing up.
5. Ensure that either the module or the selected system components provide the power-up protocol.
6. Use a single switch to apply all power to the system.

2.7 SPECIFICATIONS

Identification	M8190
Size	Quad
Dimensions	26.5 cm × 22.8 cm (10.5 in × 8.9 in)
Power Consumption	+5 V ± 5% at 5.5 A (maximum) +12 V ± 5% at 0.1 A (maximum)
AC Bus Loads	1 unit load
DC Bus Loads	1 unit load
Environmental:	
Storage	−40°C to +65°C (−40°F to 150°F), 10% to 90% relative humidity, noncondensing
Operating	For ambient temperatures above +55°C, sufficient air flow must be provided to limit the module temperature to less than +65°C. For inlet temperatures below +55°C, air flow must be provided to limit temperature rise across the module to +10°C. Derate maximum temperature by 1°C (1.8°F) for each 305 m (1000 ft) above 2440 m (8000 ft).

CHAPTER 3
CONSOLE ON-LINE DEBUGGING TECHNIQUE (ODT)

3.1 INTRODUCTION

The console octal debugging technique, normally called the console ODT, allows the KDJ11-B to respond to commands and information entered via a console terminal connected to the module. The console interface uses addresses 17 777 560 through 17 777 566 to communicate with the DCJ11 microprocessor. The addresses of the console terminal are generated in the microcode and cannot be changed. Communication between the microprocessor and the user is a stream of ASCII characters interpreted as console commands. These commands are a subset of the commands used in the ODT-11 software for microcomputers.

This feature is called the microcode on-line debugging technique, or micro-ODT. The KDJ11-B micro-ODT accepts 22-bit addresses, allowing it to access 4088 Kbytes of memory, plus the 8 Kbyte I/O page. Micro-ODT provides a more sophisticated range of debugging techniques, including access to memory locations by virtual address.

3.1.1 Terminal Interface

The KDJ11-B provides a DLART serial line interface on the module. The console is connected to the module directly or via an interfacing panel. This allows the console to communicate with the KDJ11-B. The DLART uses four registers designated as the RCSR, RBUF, XCSR and XBUF. These registers are described in Chapter 1.

Console ODT uses bit 7 of the RCSR and the XCSR registers and the low bytes of the RBUF and XBUF registers. The other bits used by these registers are ignored with the following exceptions.

The XCSR maintenance bit 2 and the XMIT break bit 0 must be cleared in order for the console ODT to function. The interrupt enable bits 6 of the XCSR and RCSR registers have no effect during console ODT.

CAUTION

The user must not write any data into the XCSR that will set bits 2 or 0 to a 1. That is, no data that ends with 1, 3, 4, 5, 6 or 7 may be entered into this register. If this is done, the user has to cycle the power-up sequence or hit the restart switch.

3.2 ODT OPERATION OF THE CONSOLE SERIAL LINE INTERFACE

The processor microcode operates the serial line interface in half-duplex mode by using program I/O techniques rather than interrupts. This means that when the ODT microcode is busy printing characters using the output side of the interface, the microcode is not monitoring the input side for incoming characters. Any characters coming in while the ODT microcode is printing are lost. Overrun errors detected by the DLART are ignored because the microcode does not check any error bits in the serial line interface registers.

Therefore, the user should not "type ahead" to ODT because those characters will not be recognized. If another processor is at the end of the serial line, it must obey half-duplex operation. In other words, no input characters should be sent until the processor output is finished.

3.2.1 Console ODT Input Sequence

The input sequence for ODT is as follows.

1. Test RCSR bit 7 (DONE flag) at 17 777 560 using a DATI bus cycle. If it is a 0, continue testing.

2. If RCSR bit 7 is a 1, read the low byte of RBUF at 17 777 562 using a DATI bus cycle.

3.2.2 Console ODT Output Sequence

The output sequence of ODT is as follows.

1. Test bit 7 (DONE flag) of the XCSR at 17 777 564 using a DATI bus cycle. If it is a 0, continue testing.

2. If XCSR bit 7 is a 1, write to the XBUF at 17 777 566 using a DATO bus cycle. The desired character is in the low byte.

3.3 CONSOLE ODT ENTRY CONDITIONS

The ODT console mode can be entered in the following ways.

- Execution of a HALT instruction in kernel mode, provided the trap option is not selected in the maintenance register (address 17 777 750, bit 3). The trap option is reset by the negation of DCOK.

- Assertion of the BHALT signal on the bus. Note that the signal must be asserted long enough to be seen at the end of a macroinstruction by the service state in the processor. BHALT is asserted if the halt-on-break feature is enabled by setting BCSR bit 9 to a 1, and then the SLU console receives a break character.

- At power-up when the power-up option is selected or at power-up and restart if the halt switch is depressed.

ODT causes the following conditions upon entry.

1. Performs a DATI from RBUF (input data buffer at 17 777 562) and then ignores the character present in the buffer. This operation prevents the ODT from interpreting erroneous characters or user program characters as a command.

2. Prints a carriage return (<CR>) and line feed (<LF>) on the console terminal.

3. Prints the contents of the PC (program counter R7) in six digits.

4. Prints a <CR> and <LF>.

5. Prints the prompt character @.

6. Enters a wait loop for the console terminal input. The DONE flag (bit 7) in the RCSR at 17 777 560 is constantly being tested for a 1 via a DATI by the processor. If bit 7 is a 0, the processor keeps testing.

3.4 CONSOLE ODT COMMAND SET

The ODT command set is listed in Table 3-1 and is described in the following paragraphs. The commands are a subset of ODT-11 and use the same command characters. ODT has 10 internal states and each state recognizes certain characters as valid input and responds with a question mark (?) to all others.

The parity bit (bit 7) on all input characters is ignored (i.e., not stripped) by console ODT, and if the input character is echoed, the state of the parity bit is copied to the output buffer (XBUF). Output characters internally generated by ODT (e.g., <CR>) have the parity bit equal to 0. All commands are echoed except for <LF>.

In order to describe the use of a command, other commands are mentioned before they have been defined. For the novice user, these paragraphs should be scanned first for familiarization and then reread for detail. The word "location," as used in the following paragraphs, refers to a bus address, processor register, or PSW.

The descriptions of the ODT commands include examples of the printouts that the processor outputs to the console terminal in response to the commands entered by the user. In the examples that follow, the processor output portions are boldface.

Table 3-1 Console ODT Commands

Command	Symbol	Function
Slash	/	Prints the contents of a specified location.
Carriage return	<CR>	Closes an open location.
Line feed	<LF>	Closes an open location and then opens the next contiguous location.
Internal register designator	$ or R	Opens a specific processor register.
PSW designator	S	Opens the PSW; must follow a $ or R command.
Go	G	Starts execution of a program.
Proceed	P	Resumes execution of a program.
Binary dump	<CTRL> <SHIFT> S	Manufacturing use only.

3.4.1 / (ASCII 057) – Slash

This command is used to open a bus address, processor register, or PSW and is normally preceded by othe characters that specify a location. In response to /, ODT prints the contents of the location (six characters and then a space (ASCII 40). After printing is complete, ODT waits for either new data for that locatio or a valid close command. The space character is issued so that the contents of the location and possibl new contents entered by the user are legible on the terminal.

Example: @00001000/**012525 <SPACE>**

Where: @ = ODT prompt character.

00001000	=	octal location in the Q22-Bus address space desire by the user (leading 0s are not required).
/	=	command to open and print contents of location.
012525	=	contents of octal location 1000.
<SPACE>	=	space character generated by ODT.

The / command can be used without a location specifier to verify the data just entered into a previousl opened location. The / produces this result only if it is entered immediately after a prompt character tha follows a location previously closed by a <CR>. A / issued immediately after the processor enters OD mode causes ? <CR> <LF> to be printed because a location has not yet been opened.

Example: @1000/**012525 <SPACE>** 1234 <CR> <CR> <LF>
@/001234 <SPACE>

Where:

first line	=	new data of 1234 entered into location 1000 an location closed with <CR>.
second line	=	a / entered without a location specifier and th previous location opened to reveal the new content correctly entered into memory.

3.4.2 <CR> (ASCII 15) – Carriage Return

This command is used to close an open location. If the contents of a location are to be changed, the use must precede the <CR> with the new data. If no change is desired, <CR> closes the location withou altering its contents.

Example: @R1/**004321 <SPACE>** <CR> <CR> <LF>
@

Processor register R1 was opened and no change was desired, so the user issued <CR>. In response to th <CR>, ODT printed <CR> <LF> @.

Example: @R1/**004321 <SPACE>** 1234 <CR> <CR> <LF>
@

In this case, the user desired to change R1. The new data, 1234, was entered before the <CR>. OD deposited the new data into the open location and then printed <CR> <LF> @. ODT echoes the <CR> entered by the user before it prints <CR> <LF> @.

3.4.3 <LF> (ASCII 12) – Line Feed

This command is used to close an open location and then open the next contiguous location. Bus addresses and processor registers are incremented by two and one, respectively. If the PSW is open when an <LF> is issued, it is then closed, <CR> <LF> @ is printed, and no new location is opened. If the open location contents is to be changed, the new data must precede the <LF>. If no data is entered, the location is closed without being altered.

Example:

@R2/**123456 <SPACE>** <LF> <CR> <LF>
@R3/**054321 <SPACE>**

In this case, the user entered <LF> with no data preceding it. In response, ODT closed R2 and then opened R3. When a user has the last register, R7, open, and issues <LF>, ODT "rolls over" to the first register, R0. ODT opens location 0 if the last location in the I/O page (17 777 776) is open and the user issues an <LF>.

Unlike other commands, console ODT does not echo the <LF>. Instead, it prints <CR>, then <LF>, so that terminal printers operate properly. In order to make this easier to decode, console ODT does not echo ASCII characters in the range 0 to 17 (octal), but responds with ? <CR> <LF> @.

3.4.4 $ (ASCII 044) or R (ASCII 122) – Internal Register Designator

Either character, $ or R, when followed by a register number (0 to 7) or PSW designator (S), opens the processor register specified. The $ character is recognized to be compatible with ODT-11. The R character was introduced as a one-key-stroke representation of its function. Lower case r (ASCII 162) is treated the same as R.

Examples:

@$0 /**000123 <SPACE>**

@R7/**000123 <SPACE>** <LF>
@R0/**054321 <SPACE>**

If more than one character (digit or S) follows the R or $, ODT uses the last character as the register designator. An exception: if the last three digits are 077 or 477, ODT opens the PSW rather than R7.

3.4.5 S (ASCII 123) – Processor Status Word Designator

This designator is for opening the PSW and must be used after the user has entered an R or $ register designator. Lower case s (ASCII 163) is treated the same as S.

Example:

@RS/**100377 <SPACE>** 0 <CR> <CR> <LF>
@/**000010 <SPACE>**

Note that the trace bit (bit 4) of the PSW cannot be modified by the user. This is to prevent the PDP-11 program debugging utilities (e.g., ODT-11) that use the T-bit for single-stepping from being accidentally harmed by the user. If the user issues an <LF> while the PSW is open, the word is closed and ODT prints <CR> <LF> @. No new location is opened in this case.

3.4.6 G (ASCII 107) – Go

This command is used to start program execution at a location entered immediately before the G. This function is equivalent to the Load Address and Start switch sequence on other PDP-11 consoles.

Example: @200 G <NULL> <NULL>

The ODT sequence for a G, after echoing the command character, is as follows.

1. Print two nulls (ASCII 0) so the bus initialize that follows does not flush the G character from the double buffered UART chip in the serial line interface.

2. Load R7 (PC) with the entered data. If no data is entered, 0 is used. (In the above example, R7 equals 200 and that is where program execution begins.)

3. The Floating-Point Status register (FPS) and the PSW are cleared to 0.

4. The LSI-11 bus is initialized by the processor asserting BINIT L for 12.6 μseconds, negating BINIT L, and then waiting for 110 μseconds.

5. The service state is entered by the processor. Anything to be serviced is processed. If the BHALT L bus signal is asserted, the processor reenters the console ODT state. This feature is used to initialize a system without starting a program (R7 is altered). If the user wants to single-step a program, he/she issues a G and then successive P commands, all with the BHALT L bus signal asserted.

3.4.7 P (ASCII 120) – Proceed

This command is used to resume execution of a program and corresponds to the Continue switch on other PDP-11 consoles. No machine state visible to the programmer is altered using this command.

Example: @P

Program execution resumes at the place pointed to by R7. After the P is echoed, the ODT state is left and the processor immediately enters the state to fetch the next instruction. If a halt request is asserted, it is recognized at the end of the instruction (during the service state) and the processor then enters the ODT state. Upon entry, the contents of the PC (R7) is printed. In this fashion, a user can single-step through a program and get a PC "trace" displayed on the terminal.

3.4.8 <CTRL> <SHIFT> S (ASCII 23) – Binary Dump

This command is used for test purposes by manufacturing and is not a normal user command. The command is normally received from another computer and not the system console. It is recommended that this command not be issued from the terminal because the console ODT echoes back the ASCII 23 code, and this may cause the keyboard to lock up, preventing data from being displayed on the screen. There is no reason to issue this command from a terminal because it then dumps the binary data. The terminal is intended to receive ASCII data. This command is intended to more efficiently display a portion of the memory, as compared to using the / and <LF> commands.

This command can accidentally be entered on many terminals by typing <CTRL> S, <CTRL> s, <CTRL> 3, or in many cases by pressing <NO SCROLL>, since all these conditions normally generate the ASCII 23 code. If the user accidentally enters this command, it is recommended that the user reset the terminal and type an "a" at least three times in order to ensure that the console ODT is ready to accept commands again. The command protocol is as follows.

1. After a prompt character, ODT receives a <CTRL> <SHIFT> S command and echoes it.

2. The host system at the other end of the serial line must then send two 8-bit bytes, which ODT interprets as a starting address. These two bytes are not echoed. The first byte specifies starting address <15:08>, and the second byte specifies starting address <07:00>. Bus address bits <21:16> are always forced to 0; the dump command is restricted to the first 32K words of address space. The starting address may be even or odd.

3. After the second address byte is received, ODT outputs 10 bytes to the serial line, starting at the address previously specified. When the output is finished, ODT prints <CR> <LF> @.

3.5 KDJ11-B ADDRESS SPECIFICATION

The KDJ11-B micro-ODT accepts 22-bit addresses, allowing it to access 4088 Kbytes of memory, plus the 8 Kbyte I/O page. All I/O page addresses must be entered by users with the full 22 bits specified. For example, to open the RCSR of the SLU (DLART), the user must enter 17 777 560, not 17 7 560 or 777 560.

3.5.1 Processor I/O Addresses

Certain processor and memory management registers have I/O addresses assigned to them for programming purposes. If referenced in ODT, the PSW responds to its bus address, 17 777 776. Processor registers R0 through R7 do not respond (i.e., timeout occurs) to bus addresses 17 777 700 through 17 777 707 if referenced in ODT.

The MMRs and PAR/PDR pairs can be accessed from ODT by entering their bus address.

Example: @17777572/**000001 <SPACE>**

In this case, MMR0 is opened to show the memory management enable bit set.

The FP11 accumulators cannot be accessed from ODT. Only FP11 instructions can access these registers.

3.5.2 Stack Pointer Selection

Accessing kernel, supervisor and user stack pointer registers is accomplished in the following way. Whenever R6 is referenced in ODT, it accesses the SP specified by the PSW current mode bits (PSW <15:14>). This is done for convenience. If a program operating in kernel mode (PSW <15:14> = 00) is halted and R6 is open, the KSP is accessed.

Similarly, if a program is operating in user mode (PSW <15:14> = 11), the R6 command accesses the USP. If a different SP is desired, PSW <15:14> must be set by the user to the appropriate value, and then the R6 command can be used. If an operating program has been halted, the original value of PSW <15:14> must be restored in order to continue execution.

Example: PS = 140000
@R6/**123456 <SPACE>**

The USP has been opened.

@RS/140000 <SPACE> 0 <CR> <CR> <LF>

@R6/123456 <SPACE> <CR> <CR> <LF>

@RS/000000 <SPACE> 140000 <CR> <CR> <LF>

@P

In this case, the KSP was desired. The PSW was opened and PSW <15:14> was set to 00 (kernel mode). Then R6 was examined and closed. The original value of PSW <15:14> was restored, and then the program was continued using the P command.

3.5.3 Entering Octal Digits

When the user is specifying an address, console ODT uses the last eight digits if more than eight have been entered. When the user is specifying data, console ODT uses the last six octal digits if more than six were entered. The user need not enter leading 0s for either address or data; console ODT forces 0s as the default. If an odd address is entered, console ODT responds to the error by printing ? <CR> <LF> or @.

3.5.4 ODT Timeout

If the user specifies a nonexistent address or causes a parity error, ODT responds to the bus timeout by printing ? <CR> <LF> or @.

3.5.5 General Registers

Whenever R0 through R5 are referenced in console ODT, they access the general register set currently specified by PSW bit 11. If a program is operating in general register set 0 (PSW bit 11 set to 0), the program is halted. A general register is opened and register set 0 is accessed. Similarly, if a program is operating in register set 1, references to R0 through R5 access register set 1.

If a specific register set is desired, PSW bit 11 must be set by the user to the appropriate value, and then the R0 through R5 commands can be used. If an operating program has been halted, the original value of PSW bit 11 must be restored in order to continue execution.

Example: PSW = 000000

@R4/052525<SPACE> <CR> <CR> <LF>

R4 in register set 0 has been opened.

@RS/000000<SPACE> 4000 <CR> <CR> <LF>

@R4/177777<SPACE> <CR> <CR> <LF>

@RS/004000<SPACE> 0 <CR> <CR> <LF>

@P

In this case, R4 in register set 1 was desired. The PSW was opened and PSW bit 11 was set to 1 (selecting register set 1). Then R4 was examined and closed. The original value of PSW bit 11 was restored and the program was continued by using the P command.

CHAPTER 4
BOOT ROMS AND DIAGNOSTICS

4.1 INTRODUCTION

The boot and diagnostic ROMs consist of two Erasable Programmable ROMs (EPROMs) and one EEPROM that provides either 2 Kbytes or 8 Kbytes of memory. The two EPROMs contain the basic boot and diagnostic code used for the diagnostics, the standard boot programs, the EEPROM setup programs and the general support routines. The EEPROM stores all the variable parameters, such as the hardware configuration, boot device selections and any special user bootstrap programs.

During power-up or restart, the KDJ11-B passes control to the ROM code. This code establishes the configuration of the module, runs the diagnostics to test the KDJ11-B, tests all available memory and tests the UBA in Unibus systems. After all the diagnostics are complete, the ROM code determines if a previously selected device is ready to be booted or to enter the dialog mode, which will allow the user to input the device to be booted via the console terminal.

The user is able to select and edit the contents of the EEPROM by using the commands established by the dialog mode. In this mode, the Help, Boot, List, Setup, Map, and Test commands allow the user to custom select the features of the ROM code to meet the boot and diagnostic requirements of any LSI bus or Unibus based system that can support a variety of devices. The commands used in the dialog mode and the diagnostics are described in detail in this chapter.

NOTE

There are three versions of the ROM code. Version V6.0 was used with some of the earlier modules. Versions V7.0 and V8.0 are currently being used. The differences between these code revisions are described in Appendix A.

4.2 POWER-UP OR RESTART

The ROM code checks a status bit to determine if the system is powering up or being restarted. The system checks the status of the reboot pulse bit in the BCSR for LSI bus operation and the same bit in the KMCR for Unibus operation. When this bit is set, the system enters the restart routine and if it is clear, the system powers up. There are four selections for the power-up or restart routines available to the user that can be designated by the configuration parameters. The ROM code checks the status of the selected mode and enters the mode selected for either the power-up or the restart routine. The selections for the restart mode are identical to those used in the power-up mode. However, these selections are independent of each other and the selection for the restart mode can be different than that selected for the power-up mode.

4.2.1 Dialog (Mode 0)

The ROM code executes the diagnostics as determined by the EEPROM and then automatically enters the dialog mode. The user is now able to boot a device, select the setup mode or run more diagnostics.

4.2.2 Automatic (Mode 1)

At the completion of the diagnostics, the ROM code enters the automatic boot routine and then tries to boot the predetermined device or devices. The devices are previously selected and loaded into the EEPROM. The user can select up to six individual devices to be automatically booted. The system attempts to sequentially boot the devices on the list until a device is successfully booted or the end of the list is reached. The factory setting or default list consists of A, DL0 and MS0. The A device tries to boot an MSCP device in the range of 0 to 7.

4.2.3 ODT (Mode 2)

In this mode, a limited set of diagnostics is run and the ROM code executes a halt instruction and passes control to the DCJ11-A micro-ODT code as described in Chapter 3. The user can continue the diagnostic testing and enter the dialog mode by typing P and pressing the Return key, provided none of the register data was changed. This mode is normally used for debugging.

4.2.4 24/26 (Mode 3)

After a limited set of diagnostics is run, the ROM code loads the contents of location 26 into the PSW and then transfers control to the address referenced by the contents of location 24. This mode is used when the memory uses battery backup or when nonvolatile memory is present and it is necessary to recover from a power fail condition.

4.3 FORCED DIALOG MODE

The forced dialog mode allows the user to enter the dialog mode when the module is powered up in a mode other than the dialog mode. The user can select the forced dialog mode when the module is powered up in the ODT, 24/26 or the automatic mode by using switch 5 (BCR bit 3) of the module switchpack. When this switch is off, BCR bit 3 is set (1), and the module automatically enters the dialog mode when the diagnostics are complete. When this switch is on, BCR bit 3 is cleared, and the module does not enter dialog mode.

However, if the switch is on and the module is powered up in the automatic mode while the diagnostics are enabled, the user may force entry into the dialog mode by typing <CTRL> C or <CTRL> P. The <CTRL> C or <CTRL> P commands are keyed after the "Testing in progress - Please wait" message is displayed, but before the "1 2 3 4 5 6 7 8 9" message is completely displayed. The <CTRL> C and <CTRL> P entries are not echoed and after the diagnostics are complete, the dialog mode is entered. <CTRL> C is the preferred command. Recognition of the <CTRL> P command provides compatibility with the PDP-11 ROM code.

4.4 HELP COMMAND

A complete list and brief descriptions of all the help commands are shown in Figure 4-1. A command is executed by typing H and pressing the Return key or by keying ? only. The system returns to the dialog mode after the list is displayed.

```
Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key: H

Command     Description

Help        Type this message
Boot        Load and start a program from a device
List        List boot programs

Setup       Enter Setup mode
Map         Map memory and I/O page
Test        Continuous self test - Type CTRL C to exit
```

MR-17235

Figure 4-1 Help Commands

4.5 BOOT COMMAND

This command allows the user to boot a device. The command is executed by typing B, pressing the space bar, then typing the device name followed by the unit number of the device. The command uses arguments for the device name and the unit number to assist the user. When the name of the device is not used, the program prompts the user for it, and if the unit number is not used, the program assumes that it is zero. The unit numbers can range from 0 to 255, depending on the device and the boot program. Figure 4-2 is an example of how to boot an RL01 or RL02 device with the name DL and the unit number 2.

The boot command also uses three qualifiers to further define the situation when a nonstandard condition is used in booting a device. The qualifiers use a slash with a letter, as follows.

/A Identifies that the CSR address is nonstandard and requests the user for the actual address.

/O Identifies that the unit number given is an octal number rather than a decimal number for unit numbers above 7.

/U In a Unibus system, if the boot exists in the base ROM and also in the UBA, the base ROM boot is overridden and the boot from the UBA module or the M9312 module is used.

The format used for a qualifier is to type the device name, the unit number followed by a / and the type of qualifier. If more than one qualifier is being used, only one / is used as shown in the examples.

When the ROM code has a device name, it searches for the first boot program with the same device name. The ROM code sequentially searches in the areas listed below while attempting to match the device name. If the /U qualifier is used, the first two areas are skipped because the boot program is located on the UBA or M9312 module.

1st area	Searches the EEPROM
2nd area	Searches the KDJ11-B ROM code
3rd area	Searches the UBA module (Unibus only)
4th area	Searches the M9312 module (Unibus only)

```
Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key: B DL2

Trying DL2

Starting system

RT-11FB (S) V05.01

.SET TT QUIET

.R DATIME
Date? [dd-mmm-yy]?
```

MR-17236

Figure 4-2 Booting an RL01/RL02

The following examples of the boot mode commands are provided to show the user how to interpret the variations. If the user types a colon after the unit number, it is ignored as shown in the last example.

B DL	Boot DL0
B DL1	Boot DL1
B DU8	Boot DU unit 8
B DU10/O	Boot DU unit 8
B DU/A Address = 17 760 400	Boot DU0 with nonstandard CSR address of 17 760 400
B DU3/U	Boot DU3 using boot from UBA or M9312 module, instead of KDJ11-B module
B DU11/UO	Boot DU unit 9 using UBA or M9312 boot, instead of KDJ11-B boot
B DU11/U/O	Invalid format; causes an invalid entry error message
B DU3	Boot DU unit 3

The single letter B implements a method of supporting non-Digital boot devices on the LSI bus or Unibus. If either one of the following cases does not meet all of the conditions, then the ROM code prints out an invalid device message.

In an LSI bus system, the letter B causes the ROM code to disable the KDJ11-B ROM and check location 17 773 000 for the existence of a ROM on the bus. If one is located, and location 17 773 000 of the ROM is not 0 (halt), then the ROM code passes control to location 173 000, memory management is turned off and R0 is set to the unit number of the bus device.

In a Unibus system, the letter B causes the ROM code to transfer control to the address stored in location 17 773 024 of a ROM located on the Unibus, provided there is a stored address and it is 165 000 or greater.

4.6 LIST COMMAND

A list of the bootstrap programs available to the user is displayed by using the list command in the dialog mode. The command is executed by typing L and pressing the Return key. This command provides a listing of the available bootstrap programs and returns to the dialog mode after the list is completed. The user can change the contents of the EEPROM by using the setup commands and thereby changing the list of available programs.

The list contains all the bootstrap programs available in the ROM code and also those stored in the EEPROM as shown in Figure 4-3. The information in the list describes the device name, the range of unit numbers, the type of device and where the program is stored. The device name is normally a two-letter mnemonic, but in some cases it may only be a single letter. This name must always use the letters from A to Z. Any lower case letters are automatically converted to upper case letters by the ROM code. The unit number range is a list of valid unit numbers that can be used with a particular boot program. The range varies from a single device designated as 0, up to as many as 255 multiple devices, depending on the type of device. The type of device that is associated with a name is usually the actual name of the device. For example, a device named as a DL would actually be an RL01 or RL02.

```
Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key: L

Device  Unit
name    Numbers Source      Device Type

DU      0-255   CPU ROM     MSCP (RA80/81/60, RD51/52, RX50, RC25, ....)
DL      0-3     CPU ROM     RL01/RL02
DX      0-1     CPU ROM     RX01
DY      0-1     CPU ROM     RX02
DD      0-1     CPU ROM     TU57
DK      0-7     CPU ROM     RK05
MS      0-3     CPU ROM     TK25, TS05
XH      0-1     CPU ROM     DECNET DEQNA
NU      0-15    CPU ROM     DECNET DUV11
NE      0-15    CPU ROM     DECNET DLV11-E
NF      0-15    CPU ROM     DECNET DLV11-F
```

MR-17237

Figure 4-3 Available Boot Programs

In a Unibus system, the list also contains any bootstrap programs that are stored on the UBA and the M9312 module (if it is present in the system). These programs are identified in the source listing as either UBA ROM or M9312.

4.7 SETUP MODE

The setup mode is entered by the user keying S and pressing the Return key. This mode allows the user to list and change most of the configuration parameters and the bootstrap programs. These commands are used to select the configuration parameters and the bootstrap programs required for the system devices.

The ROM code loads the first 105 bytes of the EEPROM into memory, starting at location 2000. This area of memory is referred to as the setup table and it contains all the configuration parameters described in Chapter 2. The EEPROM also contains the resident boot programs and various types of information used by the system. The first 105 bytes of information are needed by the ROM code to determine the configuration of the module, the boot program for the device, and the test selections and modes. Setup mode allows the user to change the setup table and the bootstrap programs.

Setup mode has fifteen commands that allow the user to select, edit, create and change the contents of the EEPROM. Whenever the setup mode is entered, the commands are listed and described as shown in Table 4-1. To execute a command, the user keys the command number (1 to 15) and presses the Return key. A description of these commands is included in this section. Command 1 is the exit command that returns the user to the dialog mode. The user can also exit the setup mode by typing <CTRL> C. To exit a command and remain in the setup mode, the user types <CTRL> Z. When in the process of editing or changing the contents of the EEPROM, always terminate the process by keying the Return character. If the process is terminated by a <CTRL> C or <CTRL> Z, the changes are ignored and lost.

4.7.1 Exit (1)

The exit command (1) allows the user to exit the setup mode and return to the dialog mode. This can also be accomplished by typing <CTRL> C.

Table 4-1 Setup Mode Commands

Command	Description
1	Exit
2	Select configuration parameters
3	Select bootstrap translations
4	Select automatic boot sequence
5	Select console message
6	Define switchpack boot selections
7	List available boot programs
8	Setup table initialization
9	Save the setup table in the EEPROM
10	Load EEPROM data into the setup table
11	Delete a boot program from the EEPROM
12	Load an EEPROM boot program into memory
13	Edit or create boot program in the EEPROM
14	Save a boot program in the EEPROM
15	Enter ROM ODT

4.7.2 Select Configuration Parameters (2)

The configuration parameters controlled by this command are listed in Table 4-2 and are printed out when command 2 is entered. The user is then able to change the configuration parameters to meet the desired system requirements. Chapter 2 defines the parameters and explains how they are changed or selected.

Table 4-2 Configuration Parameters

Item	Parameter	Selections	Status
A	Enable halt-on-break	(0) = No (1) = Yes	= 1
B	Disable user friendly format	(0) = No (1) = Yes	= 1
C	ANSI video terminal	(0) = No (1) = Yes	= 1
D	Power-up	(0) = Dialog (1) = Automatic (2) = ODT (3) = 24	= 1
E	Restart	Same as power-up	= 1
F	Ignore battery	(0) = No (1) = Yes	= 0

Table 4-2 Configuration Parameters (Cont)

Item	Parameter	Selections	Status
G	PMG count	Select from 0–7	= 7
H	Disable clock CSR	(0) = No (1) = Yes	= 0
I	Force clock interrupts	(0) = No (1) = Yes	= 0
J	Clock frequency	(0) = Power supply (1) = 50 Hz (2) = 60 Hz (3) = 800 Hz	= 0
K	Enable EEC test	(0) = No (1) = Yes	= 1
L	Disable long memory test	(0) = No (1) = Yes	= 0
M	Disable ROM	(0) = No (1) = Disable 165 (2) = Disable 173 (3) = Disable both	= 0
N	Enable trap-on-halt	(0) = No (1) = Yes	= 0
O	Allow alternate boot block	(0) = No (1) = Yes	= 0
P	Disable setup mode	(0) = No (1) = Yes	= 0
Q	Disable all testing	(0) = No (1) = Yes	= 0
R	Enable Unibus memory test	(0) = No (1) = Yes	= 1
S	Disable UBA ROM	(0) = No (1) = Yes	= 0
T	Enable UBA cache	(0) = No (1) = Yes	= 1
U	Enable 18-bit mode	(0) = No (1) = Yes	= 0
Type <CTRL> Z to exit or press the Return key to proceed.			
A	Enable halt-on-break	(0) = No (1) = Yes	= 1

4.7.3 Select Bootstrap Translations (3)

The bootstrap translation table lists the devices in a system that use a nonstandard CSR address. The table is needed to allow multiple MSCP devices with different controllers to be booted. When a bootstrap program is entered, the device unit number is stored in R0 and the device name (mnemonic) is stored in R2. The translation table is referenced by the bootstrap program trying to find a match for the device name and unit number. If a match is found, the CSR address is defined in the table, and if no match is found, the bootstrap program defaults to the standard CSR address. The translation table is printed out when command 3 is executed and then the user can change the table based on the particular system requirements.

An example of a translation table is shown in Figure 4-4. In this system, the user has an RD52 and an RX50 using an RQDX1 controller at the standard address of 172 150. The system also has an RC25 with a KLESI controller. Since the RQDX1 and KLESI controllers both use the same standard CSR address, one of them must respond to a different address. In this example, the KLESI controller is set to respond to CSR address 17 760 500. The RC25 has a unit number plug set for units 4 and 5, requiring that there be two entries into the translation table. The RD52 is unit 0 and the RX50 is units 1 and 2.

The user can select an entry by pressing the Return key. If there are no changes required for that entry, pressing the Return key again selects the next entry. When a new entry is required, the user keys the device name, the unit number and the nonstandard CSR address.

4.7.4 Select Automatic Boot Sequence (4)

The automatic boot sequence is created by selecting up to six devices and listing the order in which they are to be booted. The list is printed out when command 4 is executed. The user can change the list to meet the particular system requirements. If there is no existing list, the code prompts the user for a device name. The user responds by typing in the single- or double-letter mnemonic associated with the selected device. The code then prompts for the unit number of the device and the user responds by typing the unit number. This prompting continues until there are six devices entered or until the letter E is entered after the last device. Each entry in the list is defined by a device name and a unit number. If the same device is used more than once with a different unit number, each unit number requires a separate entry. A, B and E are special characters and are interpreted by the ROM code as follows.

A The ROM code searches for up to eight MSCP devices, unit numbers 0–7, at the standard CSR address and then determines if they have fixed or removable media. First, it attempts to boot the removable media devices one at a time, and then to boot the fixed media devices one at a time. Any MSCP devices that use nonstandard CSR addresses are not included in this sequence and must be individually selected by a separate entry.

B The ROM code checks for a ROM located off the KDJ11 module and responds to address 17 773 000. If the ROM exists and the first location is not zero, the ROM code aborts its internal code and jumps to location 17 773 000 in the external ROM.

E When the user lists five or fewer devices in the automatic boot sequence, the sixth or next entry must use the letter E. The ROM code interprets an E as the end of the automatic boot sequence and does not try to boot any other devices.

An example of the automatic boot sequence is shown in Figure 4-5. The prompt sequence used to add a DY device with the unit number 0 is also shown. Note how the list is terminated by the letter E.

```
TT1     blank
Device name   = DU <CR>
Unit number   = 4 0 <CR>
CSR address   = 17760500 <CR>
TT1     DU4 =address 17760500

TT2     blank
Device name   = DU <CR>
Unit number   = 5 <CR>
CSR address   = 17760500 <CR>
TT2     DU5 =address 17760500

TT3     blank
Device name   = <CTRL> Z
```

MR-17238

Figure 4-4 Typical Translation Table

```
KDJ11-B Setup mode
Press the RETURN key for Help
Type a command then press the RETURN key: 4

List/change the Automatic boot selections in the Setup table

Boot 1 = A          MSCP Automatic boot
Boot 2 = DL0
Boot 3 = MS0
Boot 4 = E          Exit Automatic boot
Boot 5 = blank
Boot 6 = blank

Type CTRL Z to exit or press the RETURN key for No change

Boot 1 = A          MSCP Automatic boot
Device name         =

Boot 2 = DL0        =
Device name

Boot 3 = MS0        =
Device name

Boot 4 = DY
Device name         =
Unit number         = 0

Boot 5 = E          = Exit Automatic boot
Device name

Boot 6 = blank      =
Device name

KDJ11-B Setup mode
Press the RETURN key for Help
Type a command then press the RETURN key:
```

MR-17239

Figure 4-5 Automatic Boot Sequence Example

4.7.5 Select Console Message (5)

The user is allowed to select a console message to be sent to the terminal at the start of the ROM code and at any time a <CTRL> Q is received by the console that is not a normal response to a previous <CTRL> Q (i.e., not an XON or XOFF). This message is normally used for systems having terminals that do not power up with the current language characters selected. There are two messages – one for English and another for the resident foreign language (if there is one). Only one message is sent and that is the one that matches the current selected language. Each message is allowed to contain up to ten bytes and these are listed when command 5 is executed. A typical listing is shown in Figure 4-6. The user must enter the message in octal code. Therefore an A is entered as 101 and an ESCape (ESC) character is entered as 033. The default code for both messages is all ten bytes set to 000, and the first byte of 000 terminates the message. Any time the current language is changed, the appropriate message is also changed.

NOTE
This feature is seldom used.

```
Type a command then press the RETURN key: 5

List/change the terminal Setup message in the Setup table

     Non
     ENGLISH        ENGLISH
     0 = 000        0 = 000
     1 = 000        1 = 000
     2 = 000        2 = 000
     3 = 000        3 = 000
     4 = 000        4 = 000
     5 = 000        5 = 000
     6 = 000        6 = 000
     7 = 000        7 = 000
     8 = 000        8 = 000
     9 = 000        9 = 000

Type CTRL Z to exit or press the RETURN key for No change

     Non ENGLISH                ENGLISH
     0 = 000 New = 031          0 = 000 New =
     1 = 000 New = 042          1 = 000 New =
     2 = 000 New =              2 = 000 New =
     3 = 000 New =              3 = 000 New =
     4 = 000 New =              4 = 000 New =
     5 = 000 New =              5 = 000 New =
     6 = 000 New =              6 = 000 New =
     7 = 000 New =              7 = 000 New =
     8 = 000 New =              8 = 000 New =
     9 = 000 New =              9 = 000 New =
```

MR-17240

Figure 4-6 Select Console Message Example

4.7.6 Define Switchpack Boot Selections (6)

The user can select the bootstrap program for a device by setting switches 2, 3 and 4 of the switchpack located on the module. These switches can be set on the module or remotely controlled via the J3 connector. The selected bootstrap programs are designated as SB1 through SB6, and any bootable device can be assigned to these designations. When a program is selected via SB1 through SB6, the ROM code attempts to boot only the device selected. When command 6 is executed, the switch selections are enabled as shown in Table 4-3. The typical types of devices assigned are shown in Figure 4-7. The special configuration is used for the automatic boot sequence.

Table 4-3 Switchpack Selections

SW2	SW3	SW4	Selected Device
On	On	On	Special configuration
On	On	Off	SB1
On	Off	On	SB2
On	Off	Off	SB3
Off	On	On	SB4
Off	On	Off	SB5
Off	Off	On	SB6
Off	Off	Off	EEPROM selects ODT mode or loops the diagnostic tests at power-up.

```
KDJ11-B Setup mode
Press the RETURN key for Help
Type a command then press the RETURN key: 6

List/change the switch boot selections in the Setup table

Switches 2,3,4    on     on     off    = DX0
Switches 2,3,4    on     off    on     = DL2
Switches 2,3,4    on     off    off    = DL0
Switches 2,3,4    off    on     on     = E0
Switches 2,3,4    off    off    off    = blank
Switches 2,3,4    off    on     on     = blank

Type CTRL Z to exit or press the RETURN key for No change

Switches 2,3,4    on     on     off    = DX0
Device name       =

Switches 2,3,4    on     off    on     = DL2
Device name       =
```

MR-17241

Figure 4-7 Switchpack Boot Selection

4.7.7 List Available Bootstrap Programs (7)

A list of the bootstrap programs available to the user is displayed when command 7 is executed. This list contains all the bootstrap programs available in the ROM code and also those stored in the EEPROM. The information in the list describes the device name, the range of unit numbers, the type of device and where the program is stored. The device name is normally a two-letter mnemonic, but in some cases it may only be a single letter. This name must always use the letters from A to Z. Any lower case letters are automatically converted to upper case letters by the ROM code. The unit number range is a list of valid unit numbers that can be used with a particular boot program. The range varies from a single device designated as 0, up to as many as 255 multiple devices, depending on the type of device. The type of device that is associated with a name is usually the actual name of the device. For example, a device named as a DL would actually be an RL01 or RL02.

4.7.8 Setup Table Initialization (8)

The setup table is initialized to the default values when command 8 is executed. This command does not change the contents of the table stored in the EEPROM, but the contents do change if the save command is also executed.

4.7.9 Save the Setup Table in the EEPROM (9)

The current contents of the setup table in memory is stored in the EEPROM when command 9 is executed. This is the only command that can actually write anything into the first 105 bytes of the EEPROM.

4.7.10 Load EEPROM Data Into the Setup Table (10)

The setup table data that is stored in the EEPROM is read into the current setup in memory when command 10 is executed.

4.7.11 Delete a Boot Program From the EEPROM (11)

The user is prompted for the device name of the bootstrap program to be deleted when command 11 is executed. After the device name is keyed, the ROM code searches for that boot program in the EEPROM and deletes it when it is found. All the boot programs that follow the deleted program are then automatically moved up into the space made available by the deleted boot program.

4.7.12 Load an EEPROM Boot Program Into Memory (12)

The user is prompted for the device name of the bootstrap program to be loaded into memory when command 12 is executed. After the device name is keyed, the ROM code searches for that boot program and loads it into the memory when it is found. This command is used to allow access to the EEPROM boot program so that it can be reviewed and edited.

4.7.13 Edit or Create a Boot Program in the EEPROM (13)

A boot program listing stored in the EEPROM can be edited or changed by updating its parameters. A list of parameters and the current status are displayed, along with the amount of space available for boot programs in the EEPROM (Figure 4-8), when command 13 is executed. The user can change the device name and description, the beginning and ending address of the program in memory, and the start address of the program. When the changes are complete, the ROM code enters ROM ODT, which is a code version of the micro-ODT.

The beginning address is the first location used by the program in memory and the ending address is the address of the last byte of code in memory. If in doubt, use the address + 2 of the last byte of code. Using larger numbers wastes valuable space in the EEPROM. The start address is the address to which the ROM code transfers control.

```
KDJ11-B Setup mode
Press the RETURN key for Help
Type a command then press the RETURN key: 13

Edit/create an EEPROM boot

Type CTRL Z to exit or press the RETURN key for No change

1410 Bytes free in the EEPROM

Device name              = AA          New = EA
Beginning address        = 000600      New = 10000
Last byte address        = 000615      New = 10177
Start address            = 000600      New = 10000
Highest Unit number      = 3           New = 255
Device Description       = EA BOOT     New = RM02/03

ROM ODT> 010000/000000  012705
ROM ODT> 010002/000000  101
ROM ODT> 010004/000000  012706
ROM ODT> 010006/000000  1000
```

MR-17242

Figure 4-8 Edit/Create an EEPROM Boot

The highest unit number defines the range of valid unit numbers for a particular device. If this value is set at 3, then the range for that device is 0 to 3. The maximum range for any device is 0 to 255. An invalid unit number error occurs at boot time, if a unit number that is not within the defined range is used.

The description of a device is optional, but use of a device description is recommended. The description is normally the physical name located on the outside of the device, such as RX02 or RK05. The description is limited to 11 characters or spaces.

4.7.14 Save a Boot Program in the EEPROM (14)

The user can take a boot program that is stored in memory and write it into the EEPROM by executing command 14. This is the only command that can actually write a program into the EEPROM. The other commands are used to edit programs that reside in memory. The device name of a boot program that is being written or stored in the EEPROM must not match the name of an existing program already stored in the EEPROM. If this condition occurs, the user must delete the existing program or change the name of the new program being stored. When there are two or more programs with the same name in the EEPROM, only the first program is used to boot a device.

4.7.15 Enter ROM ODT (15)

The user enters the ROM ODT mode when command 15 is executed. The ROM code opens up the addresses defined by the beginning address of the program. The ROM ODT is not the same as the console ODT described in Chapter 3. The only addresses that can be accessed in ROM ODT are the memory addresses from 0 to 28K words (0–157 776). Any attempt to address the I/O page, the internal registers or any other address is not allowed. The commands used by the ROM ODT and their functions are listed in Table 4-4.

Table 4-4 ROM ODT Commands

Command	Symbol	Function
Slash	/	Displays the contents of a specified location or if no address is specified, it displays the contents of the last location opened. If the open location is an odd number, then the contents display is a byte. If the open location is an even number, then the contents display is a word. Leading zeros are assumed and only the last six octal digits are interpreted.
Return	<CR>	Closes an open location.
Line feed	<LF>	Closes an open location and opens the next location. The location is incremented by 2 if a word was read and by 1 if a byte was read.
Period	.	Alternate character for <LF>. It is useful for a VT2**XX** terminal and convenient when using a keypad.
Up arrow	↑	Closes an open location and opens the previous location. The location is decremented by 2 if a word was read and by 1 if a byte was read.
Minus	–	Alternate character for ↑. It is useful for a VT2**XX** terminal and convenient when using a keypad.
Delete	DELETE	Deletes the previous character typed.

4.8 MAP COMMAND

The entire memory system is identified and all valid addresses in the I/O page are displayed as shown in Figure 4-9 . The command is executed by typing M and pressing the Return key. The system returns to the dialog mode after the memory is mapped and the valid I/O page addresses are displayed.

The entire memory from location 0 to 17 756 000 is mapped in 1,024-byte increments, but not every location is identified due to the amount of time required. The routine attempts to identify the size by the start and end address of each memory, the CSR address for each memory (if applicable), the CSR type (ECC or parity), and the general type of bus (PMI or Q22-Bus). The map command does not work if two memories share some common addresses or have CSRs with the same address. After the memory is mapped, all the addresses in the I/O page that respond to an inquiry are displayed. The address range of the I/O page is from 17 760 000 to 17 777 776. All the addresses displayed that are on the KDJ11-B module and those that are on the KTJ11-B module are briefly described. Addresses that are assigned to the external bus have no descriptions unless a memory CSR is present.

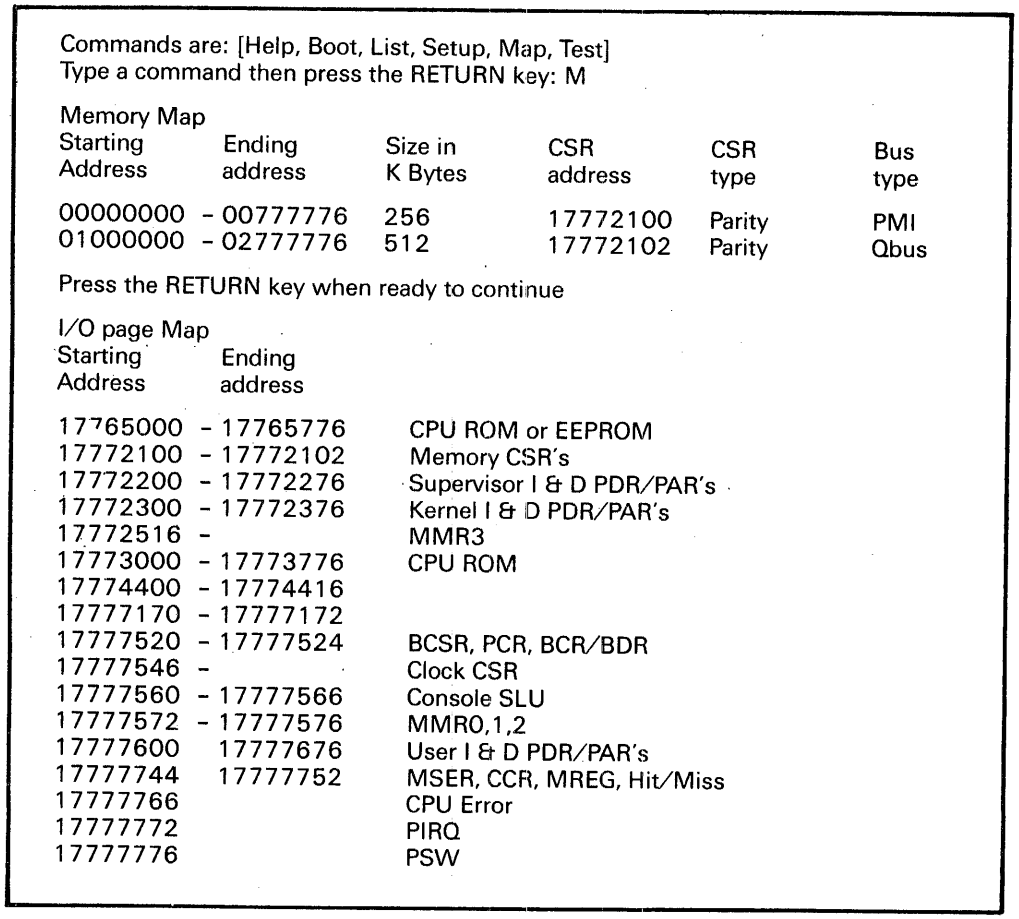


Figure 4-9 Typical Map Mode Display

4.9 TEST COMMAND

This command runs the diagnostic tests in a continuous loop until the user exits the loop by typing <CTRL> C. The command is executed by typing T and pressing the Return key. If the user wishes to loop on an individual test, the command is executed by keying T followed by the test number, and then pressing the Return key. The loop continues until an error is detected or the user stops it by typing <CTRL> C. If the test number selected is not a loopable test, then the general test loop is run for all the tests.

The ROM code starts the loop at test 70, runs all the applicable tests to the end at test 30 and continues the loop until the user exits by typing <CTRL> C. The system then displays the total number of loops completed and if any errors are detected, the total number of errors is also displayed (Figure 4-10). If the user loops on a single test, the display is as shown in Figure 4-11.

NOTE

The <CTRL> C commands are not echoed by the ROM code on the console terminal.

4.10 DIAGNOSTIC TESTS

The diagnostic tests contained in the ROM code are executed as part of the power-up sequence or when restarting the KDJ11-B module. Control is passed to the ROM code and the diagnostics check the module and the available memory before passing control on to the boot program for a previously selected device. These diagnostics are capable of testing both LSI-11 systems and Unibus systems that use the KDJ11-B module as the CPU.

The diagnostic tests are numbered in octal from 77 to 30 and the diagnostic error messages are numbered in octal from 27 to 00. The LEDs display the test number being executed and if a test fails, the LEDs blink the failing test number. A complete list of all the tests and their LED codes is given in Table 4-5. If the console device is working, an error message is displayed and the user is advised to take some appropriate action.

```
Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key: T

Continuous self test - Type CTRL C to exit

Total Passes = 4
Total Errors = 0
```

MR-17244

Figure 4-10 Continuous Testing Display

```
Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key: T 60

Looping on test 60 - Type CTRL C to exit

Total Passes = 1876
Total Errors = 0
```

MR-17245

Figure 4-11 Loop Test Display

Table 4-5 Diagnostic LED Displays

LED Code	Diagnostic
77	CPU or halt switch
76	CPU and MMU
75	Turn on MMU, run CPU and MMU
74	Turn on PMI, check UBA reboot bit
73	Power-up to mode 2: ODT
72	Power-up to mode 3: 24
71	EEPROM checksum
70	CPU ROM checksum and PCR
67	Miscellaneous CPU and EIS
66	Console SLU test 1
65	Console SLU test 2
64	Console SLU test 3
63	MMU aborts
62	Cache memory
61	Line clock
60	Floating-point instruction
57	Reserved
56	Exit standalone mode
55	UBA register response
54	Memory sizing routine
53	Memory location 0
52	Memory locations 0 to 4K words
51	Cache operation with memory
50	Complete memory data/byte exercise
47	Memory parity/ECC
46	Memory address shorts
45	UBA boot ROM
44	UBA map registers data path
43	UBA unmapped diagnostic data
42	UBA mapped diagnostic data
41	UBA floating address/data
40	UBA address overflow
37	UBA cache data
36	UBA cache LRU
35	UBA cache tag store
34	UBA cache parity error
33	Complete Unibus memory data/byte exercise
32	Unibus memory parity
31	Unibus memory address shorts
30	Exit

4.10.1 CPU or Halt Switch (Test 77)

The LEDs are set to 77 when the module is powered up or restarted. The LEDs continue to display 77 if the DCJ11 microprocessor is not able to execute an instruction out of the ROM, or if the halt switch is on to force a halt condition.

4.10.2 CPU and MMU (Test 76)

The LEDs are set to 76, indicating that the first instruction was executed. The CPU enters the standalone mode, sets the PSW to priority 7, turns off the MMU, clears the PCR and sets up the SP. The CPU jumps to the high page of the ROM, address 173 **XXX**, if not already there.

The switchpack on the module is read. If switch 1 is off and switches 2 and 3 are on, the CPU halts immediately. If the switches are not set to produce a halt, the test continues. The CPU executes some simple tests such as reading and writing the general register, branch instructions and a JSR instruction. All 48 PARs are tested using a rotating ones pattern and a unique address pattern. PDR read/write bits <14:8> and <3:1> are checked using a rotating ones pattern and a unique address pattern. At the end, all the PDRs are set to 077 406 and KISDR0 is set to 077 402.

The second page of the ROM is selected in the low byte of the PCR. The CPU jumps to the second page of ROM at address 165 **XXX**. In all, the following functions are performed.

1. All three SPs are tested.
2. MMR2 logs the address if the fetch instruction is verified.
3. MMR3 bits <5:0> are decremented from 77 to 0.
4. The CPU jumps to the high page at 173 **XXX**.
5. The CPU enables the trap area in the low page at 165 **XXX**.
6. The CPU turns on the MMU and enables 22-bit mapping.
7. The CPU verifies that the trap area is available with MMU on.

4.10.3 Turn on MMU, Run CPU and MMU (Test 75)

This test verifies the ability to write data with MMU using all three modes and both spaces. It sets the W-bit properly in a PDR and initializes all the free PARs and PDRs. For the remaining tests, the MMU stays on, but only the kernel I space is used. All free PARs and PDRs are then used as flag storage, input keyboard storage, the stack, etc.

The IOT, trap, BPT, and EMT instructions are tested and verify that the central subroutine call program and the change page program function properly. The ROM now executes virtually all the tests from the high page addresses (17 773 **XXX**), and uses the low page addresses (17 765 **XXX**) to handle all traps and other instructions to virtual addresses 0 through 276. Jumping to another page and calling up a routine from another page are accomplished by executing the EMT, trap, and BPT instructions, which also transfer control to the lower page.

The routines transmit a null character to the system console and if the console address times out, that time out is ignored for the moment. The next page of the ROM is read.

4.10.4 Turn on PMI, Check UBA Reboot Bit (Test 74)

If power-up mode 3 (24/26) is selected, the ignore battery status bit in the EEPROM is checked. When the ignore battery parameter is selected, power-up mode 3 is unconditionally executed. When the ignore battery mode parameter is not selected and the current condition is power-up, the restart mode is used to determine the current mode. If the restart mode is also mode 3, the default is to the dialog mode. If the current condition is restart, then the default is to the dialog mode.

4.10.5 Power-Up to Mode 2: ODT (Test 73)

The selected mode is ODT and this is indicated by the LED display. If the user proceeds from the ODT without changing any registers, the ROM code continues to run selected tests and enters the dialog mode when completed.

4.10.6 Power-Up to Mode 3: 24 (Test 72)

The contents of location 24 is saved in memory and a rotating ones test is executed at location 24. The original data is restored to location 24 if the test passes, and control is transferred to the address stored in that location. The contents of location 26 is used to set the PSW. The display is blanked before the control is transferred.

4.10.7 EEPROM Checksum (Test 71)

This test sets bit 5 of the BCSR to select the 8-bit EEPROM while the program is running out of the high byte PCR area (17 773 000–17 773 776). The 8-bit EEPROM is enabled into address area 17 765 000–17 765 776. This is verified by reading location 165 314 in the area and checking that the low byte is read back as 252. The test verifies that the contents of location 165 022 is zero. If either of these locations is incorrect, the first 105 bytes of the EEPROM are automatically initialized to the factory defaults. This allows a module in production to automatically initialize itself. When this is complete, an 8-bit checksum is read and accumulated for the first 105 bytes of the EEPROM and the result is verified as zero. If an error occurs, the error is printed out in the selected language.

4.10.8 CPU ROM Checksum and PCR (Test 70)

Using the high byte PCR area (17 773 000–17 773 776) as the program area, the ROM is checksummed by using the low byte of the PCR to select the 64 pages of ROM. The low byte of the PCR writes the pages of ROM into address area 17 765 000–17 765 776. The PCR is validated each time it is loaded by checking the next to the last location in each page and verifying that each byte contains the selected page number. The only exceptions to this are pages 70 through 76(8), which do not contain page numbers, but store ASCII text.

Using the low byte PCR area (17 765 000–17 765 776) as the program area, the 64 pages of ROM are written into the high byte PCR area at 17 773 000–17 773 776. The page number is verified in each byte of the next to the last word in each page.

4.10.9 Miscellaneous CPU and EIS (Test 67)

The JSR, RTI, RTS and the ASHC instructions are tested.

4.10.10 Console SLU Test 1 (Test 66)

All four of the console registers are tested for responses.

4.10.11 Console SLU Test 2 (Test 65)

The console DLART is set in the maintenance mode and a delay is allowed to settle any incoming characters. The receive buffer is cleared, and two characters, 0 and 377, are transmitted and verified.

NOTE

When the DLART is in the internal loopback mode, the transmit output is still connected to the EIA output driver. In order to prevent "garbage" characters from being printed on the console, the SLU test transmits a null character and deletes characters that are nonprinting.

4.10.12 Console SLU Test 3 (Test 64)

The transmitter and receiver are checked so that they can cause interrupts at the correct priority level. The transmit logic is used to set receiver error bits and verifies that the error bits can be cleared.

If the terminal type in the EEPROM is ANSI video and the dialog mode is not forced, the screen is then cleared and the cursor is positioned at line 8, column 1. The 'Testing in progress – please wait' message is then sent to the screen.

4.10.13 MMU Aborts (Test 63)
The protection bits in the PDRs are tested to cause aborts when the conditions are violated. This test verifies that aborts occur through virtual address 250 and that changes in the general purpose registers affected by the abort are properly recorded in MMR1.

4.10.14 Cache Memory (Test 62)
The cache is not tested to see if it is flushed after power-up. The upper address bits of the CCR are loaded into the cache tag fields at power-up. Since this address is in the I/O page, any access to it causes a miss regardless of the valid bit. The following tests are run and any trap to location 114 is a parity error on the CPU tag store.

1. Check the CCR read/write bit
2. Check the MSER timeout
3. Check the cache data path word
4. Check the hit/miss register timeout
5. Check the cache data pattern
6. Check the 4K word cache
7. Check the DMA tag store comparator
8. Check the CPU and DMA tag store timeouts
9. Check the cache flush
10. Check the hit/miss register
11. Check the cache tag store floating 1s and 0s
12. Check the CPU and DMA tag store shorts and data
13. Check the unconditional bypass on reads and writes
14. Check data store parity errors
15. Check CPU/DMA tag store parity errors

4.10.15 Line Clock (Test 61)
If the EEPROM indicates that the LTC register is enabled, bits 6 and 7 are checked so that they can be set and cleared. The clock interrupt to location 100 is checked and the correct BR level is verified. When an interrupt occurs and the clock is in sync, the test verifies that LCM bit 7 is cleared by the interrupt.

4.10.16 Floating-Point Instruction (Test 60)
Some of the floating-point instructions are executed.

4.10.17 Reserved (Test 57)
This test space is reserved for future expansion.

4.10.18 Exit Standalone Mode (Test 56)
In the 22-bit mode, the exit standalone mode is checked by using the guaranteed timeout address of 17 760 000 to verify that the timeout logic works without hanging up the CPU.

4.10.19 UBA Register Response (Test 55)
The DCSR, KMCR and DDR registers in the UBA are addressed to verify that they respond properly via the Unibus. With the UBA in the diagnostic mode, the test verifies that the address, data and control lines available via the DDR register are turned off.

4.10.20 Memory Sizing Routine (Test 54)

The system is first tested for any memory available in a Unibus system. If there is a Unibus memory, the KMCR register is set to the correct value to properly map the memory. The memory is sized in 4K word increments and the testing is based on 4K word increments.

The LSI based memory is sized in 1K word increments from 0 to 2 megawords. This is done on a word basis every 1K words. The routine starts at location 0 and sizes the memory consecutively upward. The routine reads the contents of each location and writes the data back into that location. This is a nondestructive sizing routine that proceeds upward until a timeout occurs or address 17 776 000 is reached.

When the first 128K word boundary is reached (while operating in the 22-bit mode), the routine verifies that the first two locations, 0 and 2, can be uniquely addressed in the second 128K word boundary. If these locations cannot be uniquely addressed, memory is defined as 18-bit mode and memory size is set at 124K words.

If there is any nonconsecutive memory found, the ROM code displays a message indicating that gaps are present, but no errors occur because of the gaps. The routine displays the applicable memory size messages when the routine is completed, unless a user friendly mode is selected to suppress these messages.

4.10.21 Memory Location 0 (Test 53)

Memory location 0 is checked for a response without timing out.

4.10.22 Memory Locations 0 to 4K Words (Test 52)

The first 4K words of memory are completely verified before the main memory tests are loaded and executed. A test of rotating ones and zeros is performed on physical address 0, and the ability to write separate bytes into the word locations is verified. A short/data address test on address bits 0 to 10 is executed for physical addresses from 0 to 17 776.

4.10.23 Cache Operation With Memory (Test 51)

This test verifies that the cache is allocated during a read to memory and that the cache is bypassed when the cache bypass bit is set.

4.10.24 Complete Memory Data/Byte Exercise (Test 50)

The first two word locations for each 4K word block are sequentially tested as follows.

1. The physical addresses are tested using a rotating ones pattern and then a rotating zeros pattern.
2. The locations are tested for byte operation by using the 252 and 125 data patterns.

NOTE

The Unibus memory is not tested unless it is enabled in the EEPROM. If it is enabled, then the testing is delayed until the UBA tests are completed and the UBA exits the diagnostic mode.

4.10.25 Memory Parity/ECC (Test 47)

The first two locations of each 4K word block are tested to determine which CSR controls the current address range. If there is a CSR, its ability to log the address during an error is verified. The ability to abort due to an error is also verified.

When ECC testing is enabled by the EEPROM and if bit 4 of the CSR is a read/write bit, then the ability of the ECC logic to correct single bit errors for all 16 bits of a floating ones and zeros test is verified. When this test is enabled, it is only run on 32K word boundaries.

4.10.26 Memory Address Shorts (Test 46)

This test can be looped on with the dialog mode test command. The test is relocated to the first 4K words of memory. This allows the test to execute out of the cache to improve its speed. The MMU is set up so that any memory address being tested has the cache bypass bit set in its PDR to prevent the cache from responding to that memory location. During the test, PDR1 is set up to prevent any writes to the first 4K words of memory, thus protecting the test in the cache memory. The following steps are performed.

1. From physical address 0 to the top of memory, test in 4K blocks. From first to last address in block, test by 1 word. Write location with 125 252 pattern.

2. From physical address 0 to the top of memory, test in 4K blocks. From first to last address in block, test by 1 word. Read 125 252 and write location with 025 252 pattern.

3. From physical address 0 to the top of memory, test in 4K blocks. From first to last address in block, test by 1 word. Read location for 025 252 pattern.

4. At the end of the test, verify that none of the memory CSRs has bit 15 set to a one, indicating that an error occurred.

4.10.27 UBA Boot ROM (Test 45)

All 256 words of the UBA ROM are verified by responding to the addresses from 17 773 000 to 17 773 776. The UBA ROM is then disabled and a check is made for the presence of a ROM module on the Unibus. A flag is set if a ROM module responds to all these locations. This flag is used later to allow the ROM code to search for a boot program on an M9312 module on the Unibus.

4.10.28 UBA Map Registers Data Path (Test 44)

All 32 Unibus map register pairs are tested with a rotating ones and zeros pattern and a unique address pattern.

4.10.29 UBA Unmapped Diagnostic Data (Test 43)

With mapping disabled, a floating ones and zeros test is executed through a floating address pattern, using diagnostic DATI and DATO cycles for 124K words of memory.

NOTE

Tests 42 to 34 are not executed if the 22-bit mode is disabled in the KMCR by the EEPROM, or if the Unibus memory is limited to 124K words.

4.10.30 UBA Mapped Diagnostic Data (Test 42)

This test verifies that each mapping register can be indirectly selected and can relocate a physical address from the CPU. It also checks to ensure that if Unibus memory is present, the applicable mapping register is disabled.

4.10.31 UBA Floating Address/Data (Test 41)

With mapping enabled, a floating ones and zeros test is executed through a floating address pattern using diagnostic DATI and DATO cycles for up to 2044K words of memory. This test floats a 1 and 0 across both inputs of the UBA address summing logic.

4.10.32 UBA Address Overflow (Test 40)
This test verifies that a carry can be rippled across the adder. PA is set to 2, the map register is set to 17 777 776 and the resulting address is 0.

NOTE
Tests 37 to 34 are executed only if the cache enable bit is set in the KMCR.

4.10.33 UBA Cache Data (Test 37)
The 32 cache locations are tested with a floating ones and zeros pattern, and the cache valid bits and the hit logic are checked.

4.10.34 UBA Cache LRU (Test 36)
All 24 valid combinations of the Least Recently Used (LRU) logic in the cache are checked.

4.10.35 UBA Cache Tag Store (Test 35)
A floating ones and zeros pattern test is executed through the cache tag store.

4.10.36 UBA Cache Parity Error (Test 34)
This test verifies that if an ECC error occurs during a read from memory, the applicable set is invalidated.

4.10.37 Unibus Memory Data/Byte Exercise (Test 33)
The first two word locations for each 4K word block are sequentially tested as follows.

1. The physical addresses are tested using a rotating ones pattern and then a rotating zeros pattern.
2. The locations are tested for byte operation by using the 252 and 125 data patterns.

4.10.38 Unibus Memory Parity (Test 32)
The first two locations of each 4K word block are tested to determine which CSR controls the current address range. If there is a CSR, then its ability to log the address during an error is verified. The ability to abort due to an error is also verified.

4.10.39 Unibus Memory Address Shorts (Test 31)
Every location in memory is tested for shorts in its address. The test is relocated to the first 4K words of memory. This allows the test to execute out of the cache memory to improve its speed. The MMU is set up so that any memory address being tested has the cache bypass bit set in its PDR to prevent the cache from responding to that memory location. During the test, PDR1 is set up to prevent any writes to the first 4K words of memory, thus protecting the test in cache memory. The following steps are performed.

1. From physical address 0 to the top of memory, test in 4K blocks. From first to last address in block, test by 1 word. Write location with 125 252 pattern.

2. From physical address 0 to the top of memory, test in 4K blocks. From first to last address in block, test by 1 word. Read 125 252 and write location with 025 252 pattern.

3. From physical address 0 to the top of memory, test in 4K blocks. From first to last address in block, test by 1 word. Read location for 025 252 pattern.

4. At the end of the test, verify that none of the memory CSRs has bit 15 set to a one, indicating that an error occurred.

4.10.40 Exit (Test 30)
This is the test exit routine.

4.11 DIAGNOSTIC TEST ERROR MESSAGES

A diagnostic test error message is displayed when an error is detected during the execution of a diagnostic test. When an error occurs, the ROM code displays the following data for the user and then waits for the user's response.

1. The number of the test that failed.
2. The test description.
3. A reference to the troubleshooting documentation.
4. The address of the error.
5. The contents of register set 0 (R0–R6) and kernel PAR3.
6. The failing address, good and bad data for some memory tests.
7. Up to four command options for the user.

4.11.1 Test Number

The error number is the number of the test that the ROM code is executing when failure occurs. The only exception is when an unexpected trap occurs. In this case, the error is the test number plus 100. An unexpected trap occurring during test 62 displays an error number of 162. Unexpected traps are always considered fatal errors.

4.11.2 Address of the Error

The address of the error is broken down to the actual PC, the page number in the ROM and the reference address in the program listing. In the case of an unexpected trap, the error address is the address following the unexpected error trap.

4.11.3 Register Set 1

The tests do not use register set 1 because this set is mainly used by the ROM code support routines.

4.11.4 Optional User Commands

There are up to four optional commands available to the user. They are displayed in the chart that follows. To execute one of the commands, the user keys in the command number and presses the Return key.

Command	Description
1	Rerun test
2	Loop on test
3	Map memory and I/O page
4	Advance to the next test

4.11.4.1 Rerun Test – If the test passes, the ROM code continues the testing routine. If all the other tests successfully pass, the ROM code displays the total number of errors and enters the dialog mode, regardless of the EEPROM mode selection.

4.11.4.2 Loop on Test – This command causes the ROM code to continuously loop on the failed test. The loops are generally very large and are not intended to be used as scope loops. The test continues to run even if an error occurs or the end of the test is reached. In either case, the test is started again and loops until the user types <CTRL> C at the console. At this time, the display contains the total number of errors and the total number of successful passes. Both the error counter and the pass counter have a maximum value of 65,535. If either counter reaches its maximum value, it locks up at that value and does not overflow.

4.11.4.3 Map Memory and I/O Page – This command is normally used when a memory error occurs. The map command may point to where the memory is not configured properly. In a multimemory system where one of the memories fails, this command can identify the failing memory if it has a CSR. This command is not available for tests 76 through 56 because the bus is not turned on.

4.11.4.4 Advance to the Next Test – The user can bypass the failing test by using this command to continue the testing. This command is only allowed for errors that are generally considered nonfatal. If the error is fatal and the user wants to bypass the error, the command is executed by typing <CTRL> 0, typing 4 and pressing the Return key.

NOTE

It is important to warn the user that bypassing any error, fatal or nonfatal, is an assumed risk.

4.11.5 Typical Displays

Three typical displays of diagnostic errors are shown in the following figures. A general type of error is shown in Figure 4-12. A memory test error in which the failing address, good data and bad data are displayed in addition to the standard error information, is shown in Figure 4-13. An unexpected trap error is shown in Figure 4-14.

```
Testing in progress - Please wait
Memory Size is 256 K Bytes
9 Step memory test
  Step 1 2 3 4 5 6 7 8

Error 46
Memory CSR Error

See troubleshooting documentation

Error PC = 173436     PCR page = 15     Program listing address = 015436

R0 = 060000     R1 = 052525     R2 = 172100     R3 = 172344
R4 = 100000     R5 = 040000     R6 = 172300     Par3 = 010000

Command     Description

    1       Rerun test
    2       Loop on test
    3       Map memory and I/O page
    4       Advance to the next test

Type a command then press the RETURN key:
```

MR-17246

Figure 4-12 Typical Diagnostic Error Display

```
Testing in progress - Please wait
Memory Size is 256 K Bytes
9 Step memory test
  Step 1  2  3

Error 46
Memory Error

Error PC = 173256     PCR page = 15     Program listing address = 015256

R0 = 060000     R1 = 125252     R2 = 000002     R3 = 052525
R4 = 000100     R5 = 040000     R6 = 172300     Par3 = 001000

Expected data     = 125252
Bad data          = 000002
Address           = 00100000

Command     Description

    1       Rerun test
    2       Loop on test
    3       Map memory and I/O page

Type a command then press the RETURN key:
```

MR-17247

Figure 4-13 Typical Memory Test Error Display

```
Testing in progress - Please wait

Error 162
Unexpected trap to location 250 MMU

See troubleshooting documentation

Updated PC = 173436     PCR page = 15     Program listing address = 015436

R0 = 101365     R1 = 076410     R2 = 177746     R3 = 177744
R4 = 101367     R5 = 000250     R6 = 172276     Par3 = 052400

Command     Description

    1       Rerun test
    2       Loop on test

Type a command then press the RETURN key:
```

MR-17248

Figure 4-14 Typical Unexpected Trap Error Display

4.12 ROM CODE BOOT PROGRAMS

The boot and diagnostic ROMs provide the following primary bootstrap programs for Unibus devices and LSI bus devices.

LSI Bus Bootstrap Programs		Unibus Bootstrap Programs	
Mnemonic	**Device**	**Mnemonic**	**Device**
DU	MSCP	DU	MSCP
DK	RKO5	DK	RKO5
DL	RLO1/02	DL	RLO2
DX	RXO1	DX	RXO1
DY	RXO2	DY	RXO2
DD	TU58	DD	TU58
XH	DEQNA	NU	DUV11
NE	DLV11-E	NF	DLV11-F
MS	TSO5/TK25	MU	TK50

The primary bootstrap program normally reads 256 words from the device into memory, starting at location 0. When the secondary bootstrap program is loaded without any errors, the ROM code transfers control to location 0 with the MMU turned off. The contents of R0 is the unit number of the device, and the contents of R1 is the base address of the device CSR. Sometimes the contents of R1 is the base address plus an offset. After secondary bootstrap program loading is complete, the display is blanked. Then the ROM code displays the 'Starting system' message before transferring control to the secondary bootstrap program.

During the execution of the bootstrap program, the ROM code attempts to detect any errors and take appropriate action. A list of errors often detected by the ROM code is given in Table 4-6, along with the associated LED displays.

Table 4-6 Bootstrap Error LED Displays

LED Code	Module Function
27	Not used
26	Not used
25	Not used
24	DECNET boot (DLV11-E/F, DUV11) waiting for host reply
23	XON not received after XOFF, type <CTRL> Q to correct
22	Xmit ready bit does not set
21	Drive error
20	Controller error
17	Invalid boot device selection (i.e., AA)
16	Invalid unit number selection
15	Nonexistent drive
14	Nonexistent controller
13	No tape
12	No disk
11	Invalid boot block
10	Drive not ready
07	No bootable device found in automatic boot mode
06	Console disabled by switch 1 on, and no force dialog or APT break received; ROM code has entered ODT for APT
05	Not used
04	Dialog mode
03	UBA ROM boot in progress
02	EEPROM boot in progress
01	CPU ROM boot in progress
00	Start secondary boot with display blanked

4.12.1 Error Messages for Bootstrap Programs

When an error is detected, the ROM code displays an error message on the console device. These error messages are applicable to all CPU ROM resident boot programs and any EEPROM boot programs that were written to pass these error messages back to the CPU ROM. Typical messages are shown in Figure 4-15. Messages 14, 16 and 17 apply only to UBA or M9312 ROM boots.

4.12.2 LSI Bus Selected Error Messages

The LSI bus systems can select user friendly mode with the automatic boot mode and receive error messages that assist the user in determining what caused the error. These messages may request the user to insert a disk, tell the user that the drive is nonexistent or explain where to seek help. A sample of this type of message is shown in Figure 4-16.

```
Trying DU1
Message 14
Non existent controller, address 17772152

Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key:
```

MR-17249

Figure 4-15 General Bootstrap Error Messages

```
Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key: B DL1

Trying DL1
Message 12
No disk present, or drive is not loaded

Command     Description

    1       Reboot
    2       Go to Dialog mode

Type a command then press the RETURN key: 2
```

MR-17250

Figure 4-16 User Friendly Error Message

4.13 MESSAGE DISPLAY CONSTRAINTS

As a result of self-testing diagnostics being designed into the newer terminals, it may take up to 5 seconds after power-up before a terminal is ready. This condition imposes constraints on the output of software messages. The terminal must indicate that it is ready to accept data by transmitting an XON character after the diagnostics are complete. Any data sent to the terminal before it is ready is ignored. Sometimes the terminal may already be powered up and no XON character is transmitted. Older terminals may not support the XON protocol.

The KDJ11-B ROM code executes various tests upon power-up. A console test is executed using the DLART maintenance feature and when it is complete (within a half second), the first message is sent to the console. The ROM code assumes that the console is ready to receive messages. If the ROM code receives an XON while it is testing, it retypes or sends only the most important message to the console. If, during an error, the ROM code receives an XON, the error message is retyped or sent. The user can get the current message retyped or sent by pressing the Return key. The XON feature is ignored once the dialog mode is entered.

CHAPTER 5
FUNCTIONAL THEORY

5.1 INTRODUCTION

The KDJ11-B is a quad-height microprocessor module that is designed for use in the LSI-11 based systems and can be adapted to operate in a Unibus based system. The module is a multilayered printed circuit board that uses both the LSI-11 bus (A and B rows) and the interconnecting bus (C and D rows) of the LSI-11 backplane. Figure 5-1 shows the variety of interconnecting data paths between the primary and secondary functional blocks of the module. These functional blocks include the following.

Primary	Secondary
DCJ11 microprocessor	Console SLU
Cache memory	Boot and diagnostic ROMs
Bus arbitrator	Configuration EEPROM
Data path controller	Configuration switches and LEDs
DC350/394 gate array	FPA option
DC351 gate array	
Bus distribution	

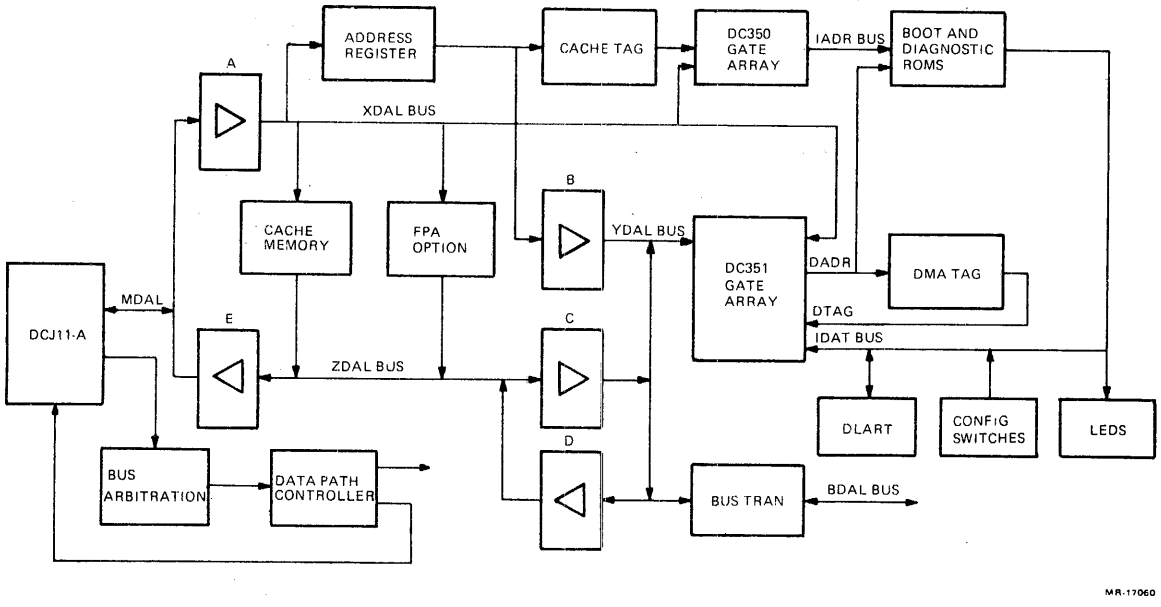


Figure 5-1 KDJ11-B Functional Block Diagram

The primary control system is the DCJ11-A with cache memory, the bus arbitrator and the data path controller. The DCJ11-A is a microprocessor chip that provides the memory management function and executes the associated PDP-11 instruction set. All the KDJ11-B operations and data transfers are initiated by the DCJ11-A. The cache memory is an on-board 8-Kbyte direct map cache memory. The DC350/394 gate array contains the cache data path logic to support the cache memory. The cache memory is transparent to all programs and is designed with high speed Random-Access Memory (RAM). The memory locations currently being accessed by the DCJ11-A are automatically stored in the cache memory from the PMI. The next time these locations are accessed, the data is retrieved from the cache memory, eliminating the time-consuming LSI-11 bus transaction. Full parity protection is provided for the cache memory. Many of the parity calculations are done by the DC350/394 cache data path logic.

The DCJ11-A microprocessor normally operates with the cache memory to provide high speed execution of the current program. While this is occurring, the data path controller monitors the DMA writes into the private memory from the main memory system. The DMA tag store checks each DMA write address to ensure that the data contained in the cache memory is not being updated. If the DMA address is cached, then the DCJ11-A is interrupted and the cache memory is updated with the new data. The DC351 gate array contains the DMA data path logic used to check the DMA address with the DMA tag store.

The bus arbitrator monitors the DCJ11-A operation and when a transaction requires it to access external or internal data that is not cached, the bus arbitrator passes control over to the data path controller while the DCJ11-A waits for the data. The data path controller recognizes the type of transaction being executed and provides the control signals for the data to be routed from its source to the DCJ11-A or to a destination selected by the DCJ11-A. The data is routed by the data path controller via the MDAL, XDAL, YDAL, and the ZDAL busses while the IADR, DADR, DTAG, CTAG and IDAT busses are controlled by the DC350/394 and DC351 gate arrays. In this way, all the internal and external addresses can be accessed. This includes all the registers described in Chapter 1.

The secondary functional blocks allow the user to connect the system console and configure the module for the user's requirements. The boot programs contained in the ROMs allow an automatic boot procedure to be executed during power-up. The diagnostics evaluate the performance of the module at this time. Any errors detected by the diagnostics are reported to the user by the system console and are displayed in the LED codes on the module. This data, as well as the baud rate selection and the dialog mode, can be remotely monitored and controlled through the connectors on the module. The module also has a socket that accepts the optional FPA chip.

5.2 DCJ11-A MICROPROCESSOR

The DCJ11-A is a microprocessor contained on a 60-pin VLSI chip. It provides many of the system timing signals and performs all the arithmetic and logic functions. The I/O pins are shown in Figure 5-2. The signals and bus transactions are described in the following paragraphs.

5.2.1 Initialization

The CDCOK H input is driven by the RDCOK H input, which is asserted by the LSI bus input BDCOK H. The BDCOK H bus signal is asserted when stable dc power is applied to the bus.

5.2.2 Output Signals

The DCJ11-A output signals control the various module functions and are described in Paragraphs 5.2.2.1 through 5.2.2.9.

5.2.2.1 Address Input/Output (MAIO <3:0> H) – These four signals classify the current transaction as a bus read, bus word write, bus byte write, general purpose read, general purpose write, interrupt acknowledge, or NOP, as shown in Table 5-1. These signals are buffered as the JAIO <3:0> inputs to the FPA socket. The JAIO <3:0> outputs are also latched as the LAIO <3:0> inputs to the DC350/394 gate array and the cycle encoder logic.

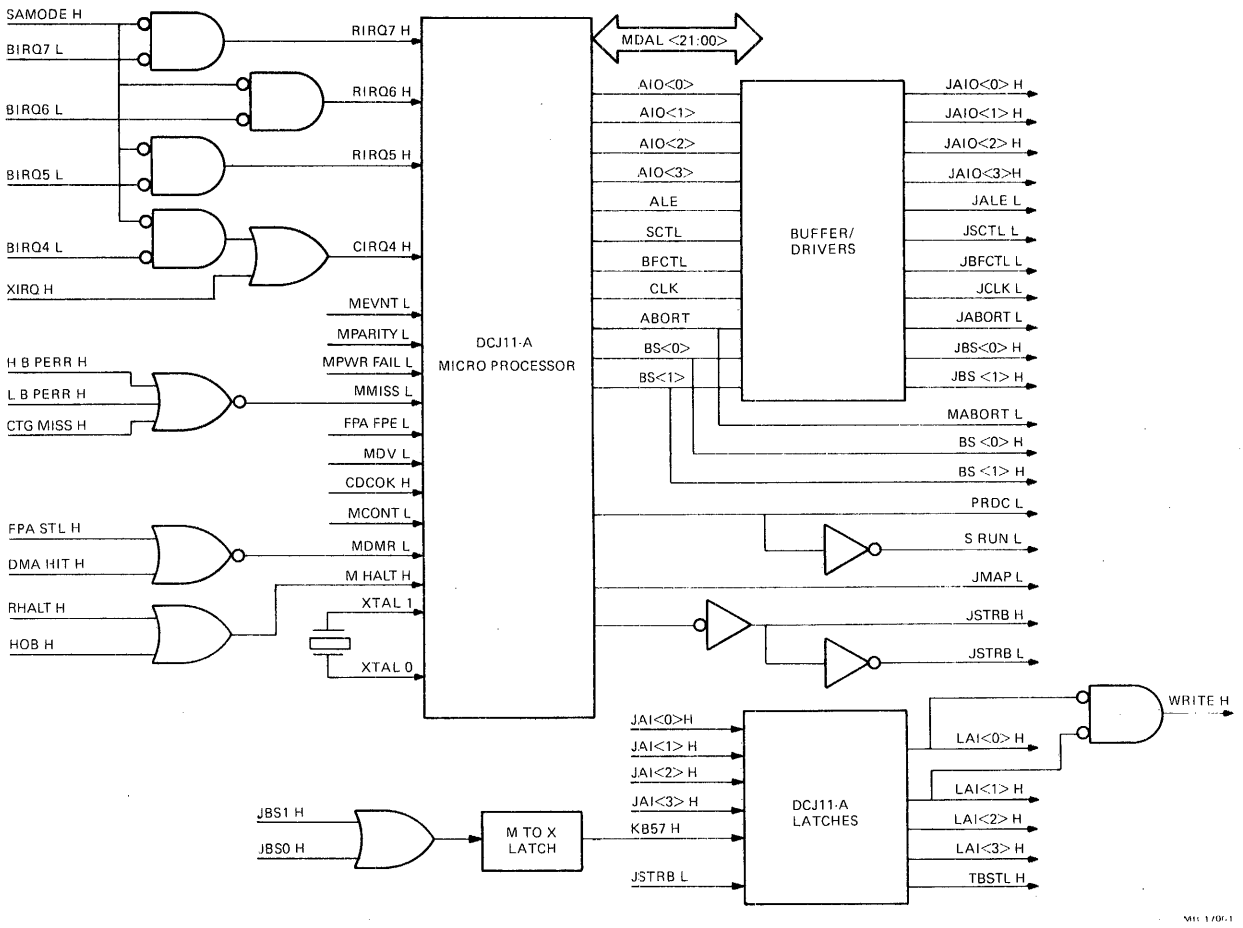


Figure 5-2 DCJ11-A Microprocessor Logic

Table 5-1 MAIO Coding

MAIO Signals				
3	**2**	**1**	**0**	**Transaction Type**
1	1	1	1	Non-I/O (NOP)*
1	1	1	0	General purpose read†
1	1	0	1	Interrupt acknowledge (read vector)‡
1	1	0	0	Instruction stream request read
1	0	1	1	Read-Modify-Write, no bus lock§
1	0	1	0	Read-Modify-Write, bus lock§
1	0	0	1	Data stream read
1	0	0	0	Instruction stream demand read
0	1	1	–	General purpose byte write†
0	1	0	–	General purpose word write†
0	0	1	–	Bus byte write§
0	0	0	–	Bus word write§

* An NOP transaction is an internal operation that does not require a bus transfer.
† A general purpose transaction is used to access interface devices that are not directly addressable by the DAL bus.
‡ Interrupt acknowledge (IACK) transactions are in response to the DCJ11-A granting an interrupt request.
§ A bus transaction uses the DAL bus to access memory, I/O devices or explicit addressable registers.

5.2.2.2 Bank Select (MBS1 H, MBS0 H) – These signals are time multiplexed during the transaction. During the first portion of a bus transaction, they are used to define the type of address on the MDAL bus. The MBS1 H and MBS0 H signals are inputs to the DC350/394 gate array and are buffered as the JBS1 H and JBS0 H outputs. These are ORed together and the output becomes the I/O page signal TBS7 via the M to X bus latches. The addresses identified by the MBS0 H and MBS1 H signals are defined in Table 5-2.

The memory types are all addresses below 17 600 000. The system board register types are bus addressable registers in the range of 17 777 740 to 17 777 751. The internal register types are addressable registers that reside within the DCJ11-A. The external I/O types are addresses greater than 17 577 777 that are neither internal registers nor system registers.

During the second half of the transaction, the MBS1 H signal indicates the cache bypass status and the MBS0 H signal indicates the cache force miss status as described below.

MBS1 H asserted – Cache bypass
MBS1 H negated – No cache bypass

MBS0 H asserted – Cache force miss
MBS0 H negated – No cache force miss

5.2.2.3 Address Latch Enable (MALE L) – The MALE L output is buffered and driven as the JALE L output. It is asserted at the start of a transaction and latches the physical address from the MDAL bus to the XDAL bus. It also latches the address, the MAIO <3:0> code and the MBS1 H, MBS0 H code in the FPA and the DC350/394 gate array.

5.2.2.4 Stretch Control (MSCTL L) – The MSCTL L output is buffered and driven as the JSCTL L output to the FPA socket and the DC350/394 gate array. It is asserted for the stretched portion of a transaction and negated when the DCJ11-A receives the MCONT L input. The JSCTL L is used by the bus arbitration controller to enable the control store functions. It also activates the ABORT L I/O signal.

5.2.2.5 Strobe (MSTRB L) – The MSTRB L output is buffered and driven as the JSTRB H and the JSTRB L outputs. The signal is asserted at the end of the second DCJ11-A clock period and is negated at the end of the transaction. The STRB H output goes to the FPA socket and it latches the MCONT L input from the control store. The STRB L output latches the XDAL address register, the DCJ11-A outputs, the XDAL to the YDAL, and is used by the DC350/394 and DC351 gate arrays.

5.2.2.6 Buffer Control (MBUFCTL L) – The MBUFCTL L is buffered and driven as the JBUFCTL L output. This output is negated to enable the MDAL bus output data on the XDAL bus and is asserted to enable the ZDAL bus input data on the MDAL bus.

Table 5-2 Bank Select Address Codes

MBS1	MBS0	Address Type
0	0	Memory
0	1	System board register
1	0	External I/O
1	1	Internal register

5.2.2.7 Predecode Strobe (MPRDC L) – This signal is asserted for the first two DCJ11-A clock periods of any transaction that decodes a PDP-11 instruction. It goes to the FPA socket. The MPRDC L output is also inverted and drives the SRUN L output of the module.

5.2.2.8 I/O Map Enable (JMAP L) – This signal is asserted when the DCJ11-A receives the DMA request input MDMR L to stretch the next transaction. The JMAP L output goes to the DC350/394 gate array.

5.2.2.9 Clock (MCLK H) – The MCLK H output is buffered and driven as the JCLK H output. This signal is used by many of the functions as a timing reference.

5.2.3 Input Signals

The DCJ11-A receives status and control information from a variety of input signals. These signals and their associated functions are described below.

5.2.3.1 MMISS L – The MMISS L input reports the cache memory hit and miss status during bus read and write transactions. This input is enabled by either the HB PERR H or the LB PERR H output from the cache data parity logic or the CTG MISS H output from the DC350/394 gate array.

5.2.3.2 Data Valid (MDV L) – The MDV L input is generated by the control store and is used to latch in read data from the MDAL bus.

5.2.3.3 Continue (MCONT L) – The MCONT L input is generated by the control store to indicate that the current stretched transaction can end.

5.2.3.4 DMA Request (MDMR L) – The MDMR L input is used to stall the DCJ11-A by stretching the next transaction. It is asserted by the FPA STL H signal from the FPA socket or by the DMA HIT H output from the DC351 gate array. The input is sampled at the beginning of the current transaction, and, when present, it asserts the JMAP L output and stretches the next transaction until the MCONT L input is received.

5.2.3.5 MIRQ <7:4> H – These inputs are coded priority levels from the external devices that drive the LSI-11 bus signals BIRQ <7:4> L. The BIRQ <7:4> L inputs are inhibited by the SAMODE H input that is asserted when the module is in the standalone mode. The XIRQ H input is asserted by the on-board DLART and is a level 4 interrupt. The MIRQ <7:4> H inputs are interrupt requests to the DCJ11-A and are coded to determine a priority level. The acknowledgment of these inputs is dependent on the current priority level of the PSW.

5.2.3.6 MHALT H – The MHALT H input is driven by the LSI-11 bus signal BHALT L or the HOB H input from the on-board DLART logic. This input is the lowest interrupt priority for an internal or external device.

5.2.3.7 MEVNT H – The MEVNT H input is driven by the DC350/394 gate array and causes a trap to location 100. This input is the line clock interrupt request and is asserted by the line clock logic in the DC350/394 gate array. The line clock interrupt vector is stored at location 100.

5.2.3.8 MPWR FAIL L – This input is from the DC350/394 gate array and is asserted by the negation of the LSI-11 bus signal BPOK H. It is used to generate the nonmaskable power fail interrupt. The MPWR FAIL L input is negated when it is acknowledged by a general purpose write to address 140 or when the LSI-11 bus signal BDCOK H is negated.

5.2.3.9 MPARITY L – The MPARITY L input is asserted by the DC350/394 gate array when a parity error is detected. This input is a nonmaskable interrupt to the DCJ11-A.

5.2.3.10 MABORT L – The MABORT L signal is an I/O line that can be driven by the DCJ11-A or the DC350/394 gate array. The MABORT L signal is buffered and driven as the JABORT L output to the FPA socket, the cycle encoder logic and the next address MUltipleXer (MUX) logic. The signal is used in conjunction with the MPARITY L input to determine when the DCJ11-A aborts the current transaction.

5.2.3.11 FPA FPE L – The FPA FPE L input is driven by the FPA socket when a floating-point exception occurs. This input is a nonmaskable interrupt request.

5.2.4 MDAL <21:01>

The MDAL <21:01> bus is a time-multiplexed data/address bus. The basic bus consists of DAL bits <15:0> and is bidirectional. DAL bits <21:16> are outputs only and are used as the extended bus. The data being transmitted or received is dependent on the type of transaction being performed by the DCJ11-A.

5.2.5 DCJ11-A Transactions

The DCJ11-A controls the type of transaction being executed and indicates this to the module circuits by coding the MAIO <3:0> signals. The standard transactions only read and write to the cache memory. Any other transaction requires a stretched cycle in which MSCTL L is asserted. The bus arbitration controller monitors the MSCTL L output and when it is asserted, the data path controller is enabled. This controls the data flow until it is ready for the DCJ11-A and then it asserts the MCONT L input. There are six basic transactions performed and these are described in Paragraphs 5.2.5.1 through 5.2.5.6.

5.2.5.1 NOP – This transaction performs a DCJ11-A internal operation and does not require the use of the MDAL bus. The normal transaction is shown in Figure 5-3. The stretched transaction (Figure 5-4) occurs when MDMR L is asserted early in the transaction and remains stretched until the MCONT L input is asserted to end the transaction.

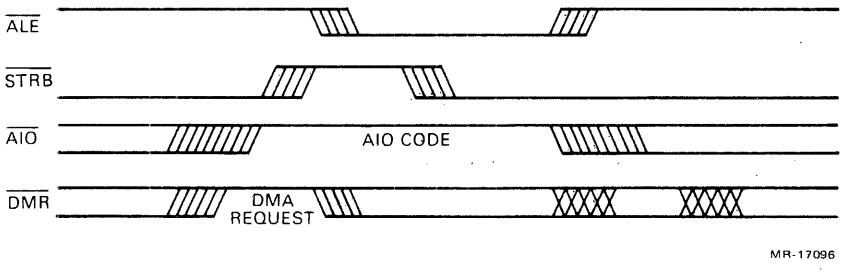


Figure 5-3 NOP Transaction

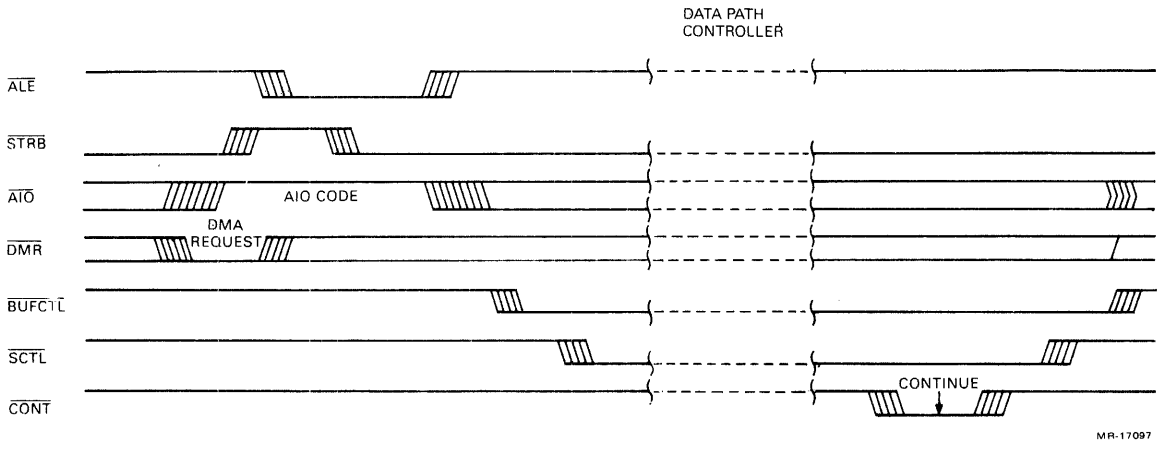


Figure 5-4 Stretched NOP Transaction

5.2.5.2 Bus Read – The bus read transaction uses the MDAL bus to read data from cache memory, main memory, I/O devices or the addressable module registers. These transactions occur during instruction stream reads, data stream reads and the read portion of read-modify-writes. The transaction reads complete words and if only a byte is required, the DCJ11-A ignores the excess byte. A bus read transaction (Figure 5-5) occurs when the physical address scores a hit in the cache memory. The DCJ11-A aborts the transaction if any memory management or address errors assert the MABORT L signal. When this happens, all current information is ignored and the transaction is immediately aborted.

The noncache or stretched bus read transaction (Figure 5-6) is used when the data must be accessed via the LSI-11 bus. This occurs when any of the following conditions exist.

1. Either MBS1 H or MBS0 H is set to 1 indicating an I/O address
2. Cache bypass is indicated
3. Cache force miss is indicated
4. MDMR L is asserted
5. Cache MMISS L is reported

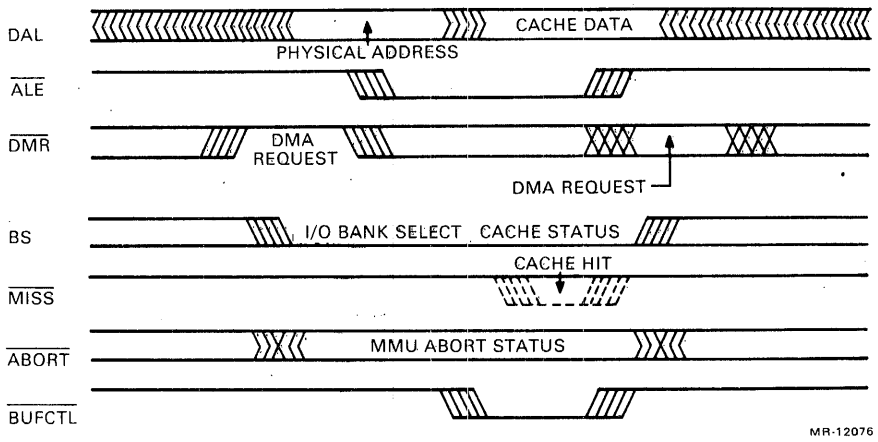


Figure 5-5 Bus Read Transaction

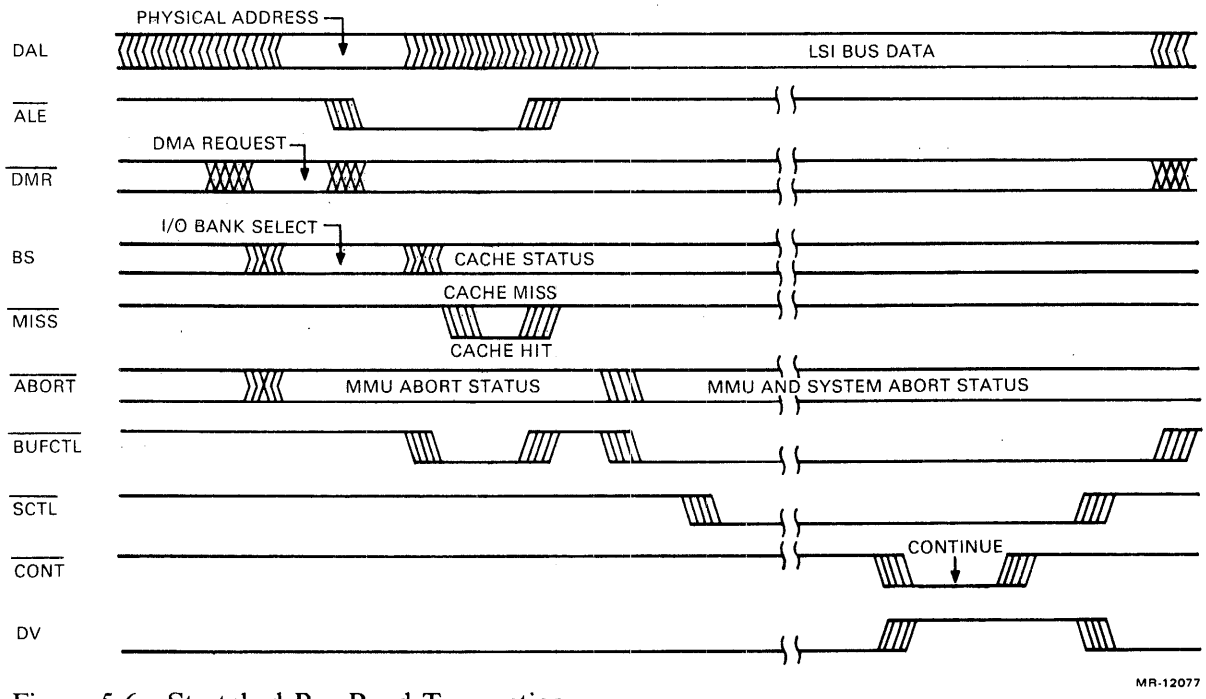


Figure 5-6 Stretched Bus Read Transaction

The MBUFCTL L and MSCTL L outputs are asserted during the stretched portion of the read transaction. The data is read by the DCJ11-A when data valid (MDV L) is asserted. When the transaction is stretched only because the MDMR L input was asserted, MDV L is not asserted because it will overwrite the valid data received from the cache. The transaction remains stretched until the MCONT L input is asserted to end the transaction.

5.2.5.3 Bus Write – The bus write transaction writes data to memory, I/O devices, or other addressable registers via the DAL bus. The transaction can write either bytes or words as determined by the MAIO code. The DCJ11-A reports any memory management or address errors by enabling the MABORT L signal. This causes the transaction to be terminated immediately and all data should be ignored.

The write transaction, as shown in Figure 5-7, and all bus write transactions are stretched. The MSCTL L signal is asserted and the write data is on the bus during the stretched portion of the transaction. For byte writes, an even address selects the low byte and an odd address selects the high byte. The data for the remaining byte is not used.

5.2.5.4 General Purpose Read – The general purpose read transaction accesses non-user-addressable module hardware. The MDAL address used for general purpose reads is in the form of 17 777 **XXX**, where the **XXX** bits represent the general purpose read code described in Table 5-3. The codes use MDAL bits <7:0> to access the hardware.

All general purpose read transactions (Figure 5-8) are stretched. The DCJ11-A reads the data when MDV L is asserted. The transaction is stretched until MCONT L is asserted to end the transaction.

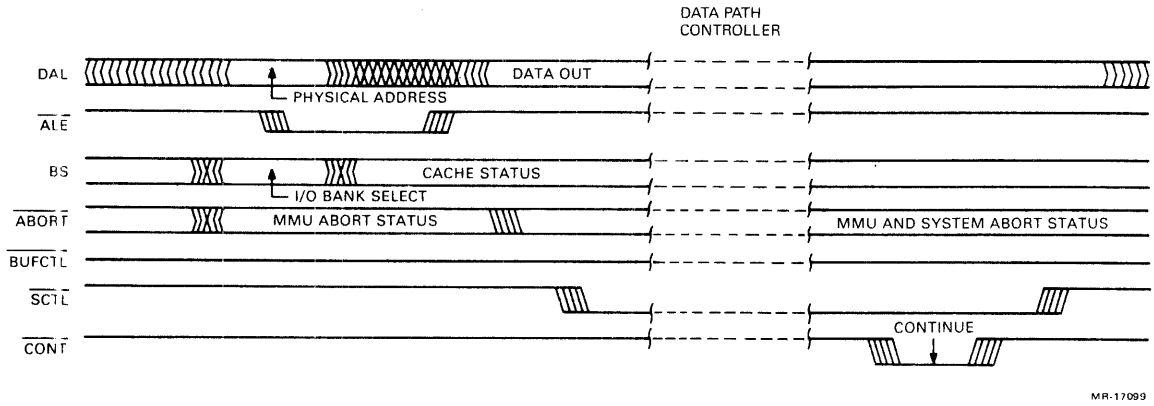


Figure 5-7 Bus Write Transaction

Table 5-3 General Purpose Read Codes

Code	Function
000	Reads the maintenance register during power-up and determines the options selected by the user
001	Reserved
003	Reserved

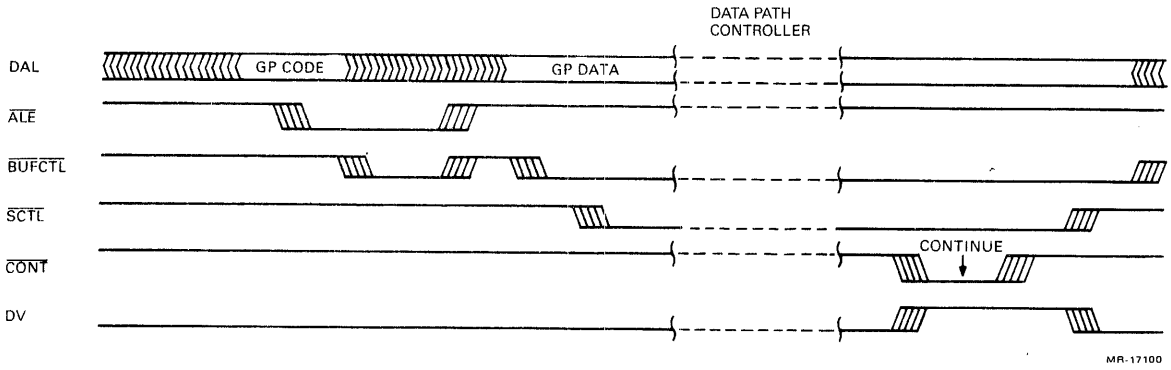


Figure 5-8 General Purpose Read Transaction

5.2.5.5 General Purpose Write – The general purpose write transaction accesses non-user-addressable module hardware. The MDAL address used for general purpose writes is in the form 17 777 **XXX**, where the **XXX** bits represent the general purpose write code described in Table 5-4. The codes use MDAL bits <7:0> to access the hardware.

All general purpose write transactions (Figure 5-9) are stretched. The DCJ11-A writes the data when MSCTL L is asserted during the stretched portion of the transaction. The transaction is stretched until MCONT L is asserted to end the transaction.

Table 5-4 General Purpose Write Codes

Code	Function
003	Reserved
014	Asserts bus reset signal
034	Indicates exit from console ODT mode
040	Reserved for future use
100	Acknowledges EVNT interrupt
114	Negates bus reset signal
140	Acknowledges power fail
220	Microdiagnostic test 1 passed
224	Microdiagnostic test 2 passed
230	Microdiagnostic test 3 passed
234	Indicates entrance into console ODT mode

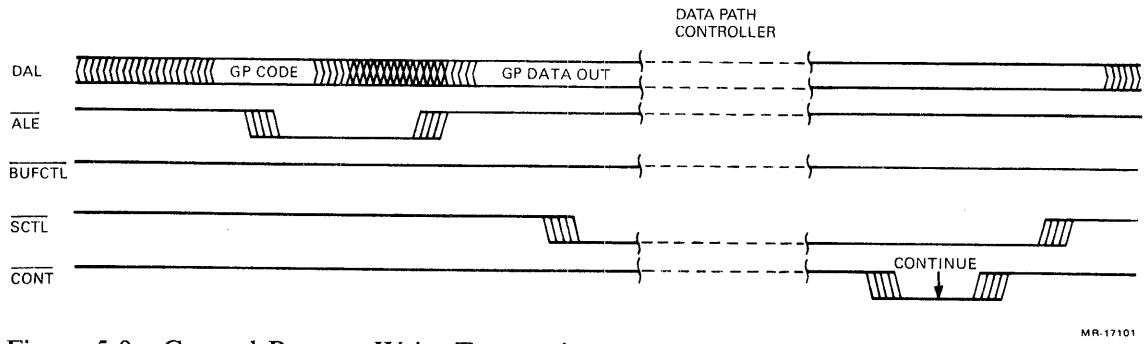


Figure 5-9 General Purpose Write Transaction

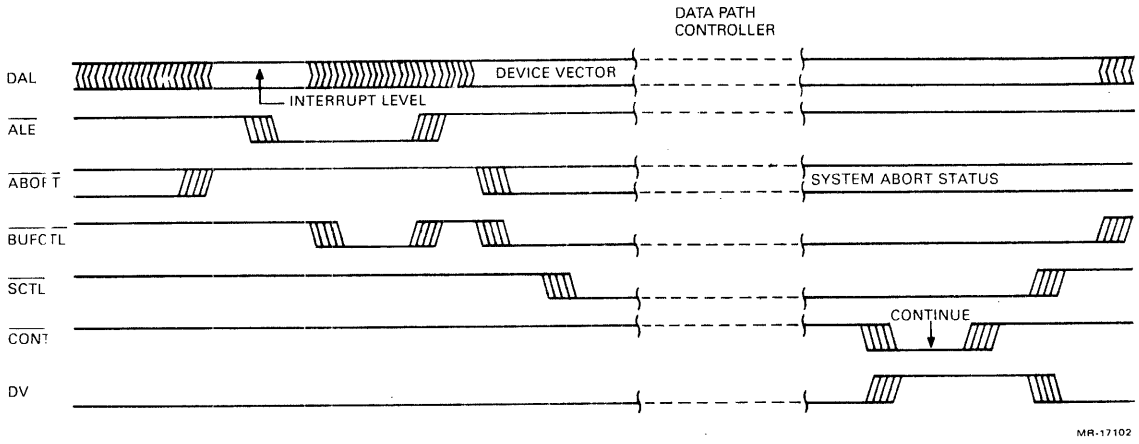


Figure 5-10 Interrupt Acknowledge Transaction

5.2.5.6 IACK – The read interrupt vector transaction acknowledges an interrupt request received on one of the MIRQ <7:4> H inputs by reading a device interrupt vector. All interrupt vector transactions (Figure 5-10) are stretched. The device interrupt vector is latched by the DCJ11-A when the MDV L input is asserted.

5.3 BUS ARBITRATOR

The bus arbitrator controls the bus operations that occur between the DCJ11-A, the data path controller and the DMA requests. It uses the JCLK H input to synchronize its operation with the DCJ11-A. With the use of some external logic, it provides the five primary control signals to the data path controller, and the DMA grant output to the LSI-11 bus, as shown in Figure 5-11. All 16 basic transactions performed by the KDJ11-B module use the DCJ11-A during the first portion of the cycle, the data path controller during the mid-range of the cycle and the DCJ11-A at the end of the cycle. The DCJ11-A SCTL L input and the data path controller input JC1:SEQ BSY H are the two primary control signals used by the bus arbitrator. The SCTL L signal is asserted as an indication to give control of the bus to the data path controller. The JC1:SEQ BSY H signal is asserted to indicate that the data path controller is using the bus and is negated to allow the bus arbitrator to return control of the bus to the DCJ11-A.

The LAIO <3:0> inputs to the DC350/394 gate array are decoded to determine the current transaction and the JSCTL L input is monitored to indicate the cycle is stretched. The PMG counter bits <2:0> of the BCSR in the DC350/394 gate array are decoded and the counter is enabled whenever an I/O page location or external memory is referenced by the DCJ11-A. All DMA requests are suppressed when the counter overflows, and the DCJ11-A has bus mastership during the next DMA arbitration cycle. If the counter is disabled, the DCJ11-A is blocked from bus mastership as long as DMA requests are pending. The inputs to the bus arbitration logic represent the current status of the module activities and are used to control the five signals used by the data path controller.

The internal transactions are defined by the assertion of the AUX CYC L output and the external transactions are defined by the assertion of the EXT CYC (1) H output, as shown in Table 5-5. These are the primary signals used to start the oscillator in the data path controller. The DMG H output is asserted as the DMA grant signal to the external bus. The CPU MSTR L is used by the control store logic in the data path controller.

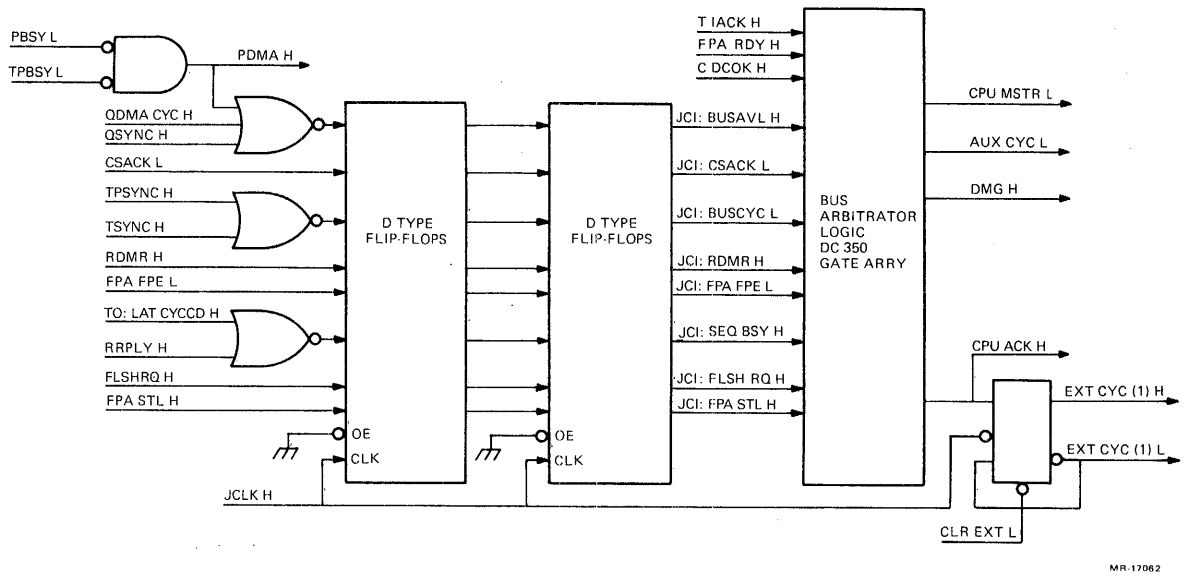


Figure 5-11 Bus Arbitrator

Table 5-5 Control Signals

Code	Transaction	Bus Arbitrator Output
0	General purpose read from FPA	AUX CYC L
1	General purpose write to FPA	AUX CYC L
2	IDAT data read	AUX CYC L
3	IDAT data write	AUX CYC L
4	DC350/394 data/General purpose read	AUX CYC L
5	DC350/394 data/General purpose write	AUX CYC L
6	EXT bus data read	EXT CYC (1) H
7	EXT bus data write	EXT CYC (1) H
8	EXT bus L byte write	EXT CYC (1) H
9	EXT bus H byte write	EXT CYC (1) H
10	EXT bus FPA write	EXT CYC (1) H
11	Q-Bus DATIO (DATOB)	EXT CYC (1) H
12	Interrupt vector read	AUX CYC L
13	Standalone mode cache write	AUX CYC L
14	Non-I/O or ABORT recovery	AUX CYC L
15	Monitor DMA write cycles	PDMA/QDMA CYC H

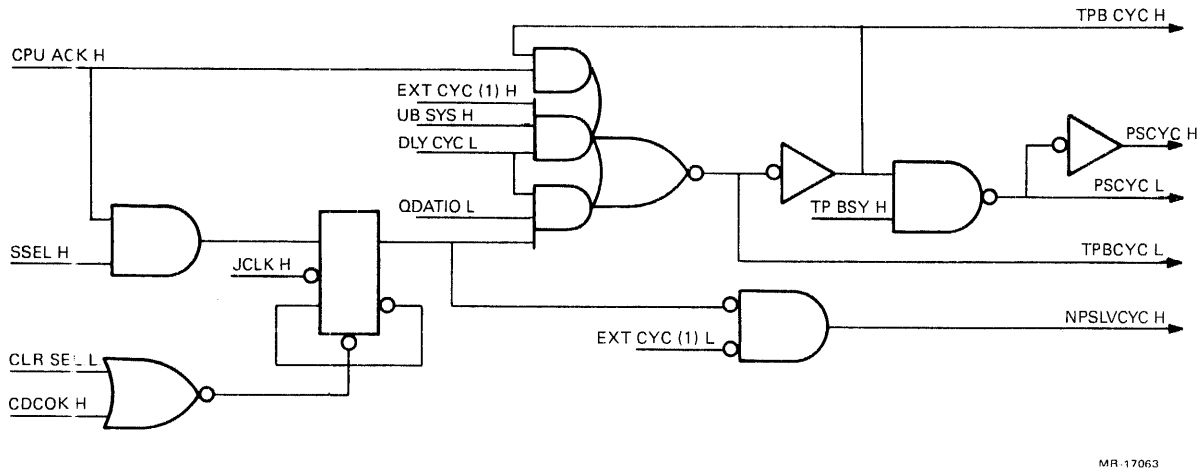


Figure 5-12 PMI Cycle Request

5.3.1 PMI Cycle Request

In addition to providing the oscillator signals to the data path controller, the CPU ACK H and CPU MSTR L outputs are used to generate the NPSLVCYC H and the TPBCYC L control signals to the next address MUX of the data path controller, as shown in Figure 5-12.

The JK input is set by the assertion of CPU ACK H and SSEL H, and is clocked by the JCLK input. This sets the output low, and with the EXT CYC (1) L set, the NPSLVCYC H output is asserted.

The TPBCYC L output is set by one of the following conditions.

1. The assertion of CPU ACT H, provided TPBCYC L is already asserted.
2. The assertion of EXT CYC (1) H and UBSYS H, provided the DLY CYC L input is negated.
3. The JK output set low, provided DLY CYC L and Q DATIO L are asserted.

5.4 DATA PATH CONTROLLER

The data path controller provides the control signals necessary to execute the 16 basic transactions. The bus arbitrator makes the data path controller the bus master whenever the DCJ11-A enters the stretched cycle mode or DMA write cycles.

The data path controller consists of the cycle encoder, the oscillator, the next address MUX and the control store, as shown in Figure 5-13. The cycle encoder and the next address MUX control the addressing of the control store. The oscillator enables the control store outputs and provides the time base for the sequencing of the control store outputs.

5.4.1 Cycle Encoder

The cycle encoder (Figure 5-14) is a programmable logic array that encodes the 12 input signals into the 4 LCYCCD outputs, which select one of the 16 basic transactions. The 12 inputs are encoded and the output latch is opened by the assertion of T0:LATCYCCD H. The LCYCCD outputs are latched 20 ns later by the assertion of LATCYCCD H input. The LCYCCD outputs enabled by the input status are shown in Table 5-6, and the transactions selected by these outputs are shown in Table 5-7.

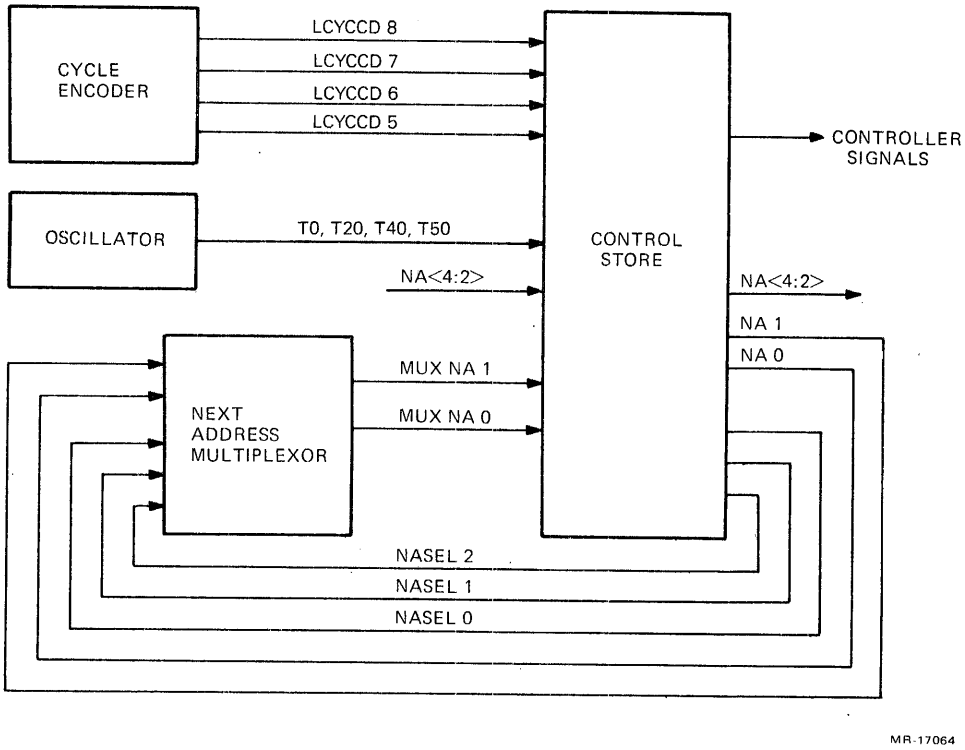


Figure 5-13 Data Path Controller

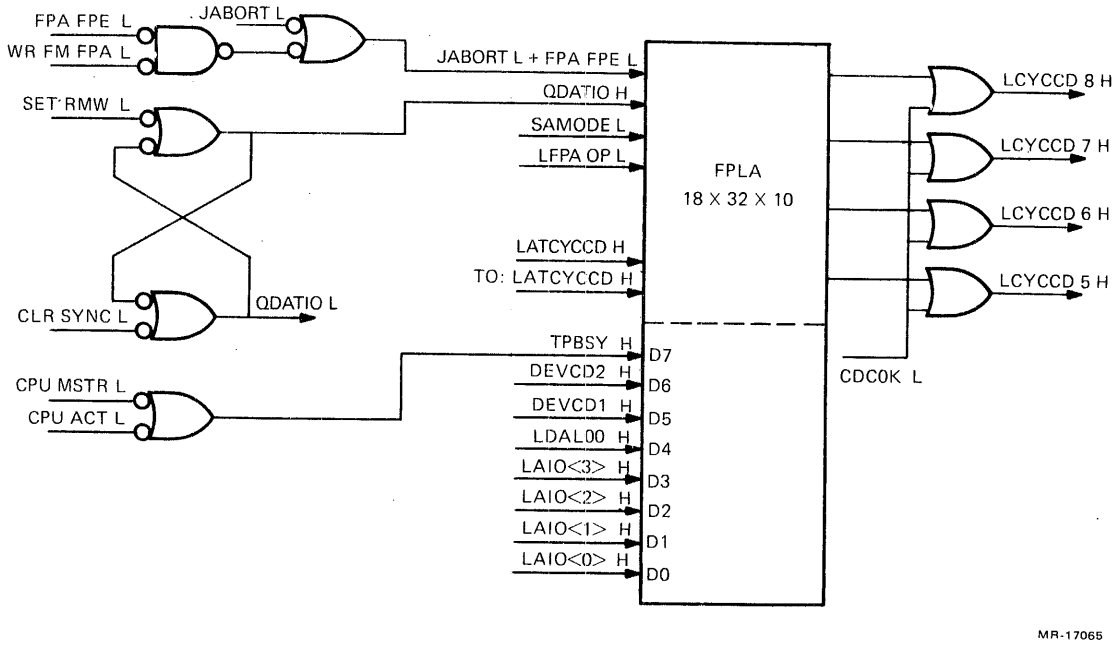


Figure 5-14 Cycle Encoder

Table 5-6 Cycle Encoder Status

LAIO <3:0>				DEVCD		LDAL				JABORT and			LCYCCD Outputs			
3	2	1	0	2	1	0	LFPA OP	QDATIO (B)	Standalone Mode	FPA	FPE	TPBSY	8	7	6	5
1	1	1	0	X	X	1	X	X	X	0		1	0	0	0	0
0	1	X	X	X	X	1	X	X	X	0		1	0	0	0	1
1	0	X	X	0	X	X	X	X	X	0		1	0	0	1	0
1	1	0	0	0	X	X	X	X	X	0		1	0	0	1	0
0	0	X	X	0	X	X	X	X	X	0		1	0	0	1	1
1	0	X	X	1	1	X	X	X	X	0		1	0	1	0	0
1	1	0	0	1	1	X	X	X	X	0		1	0	1	0	0
1	1	1	0	X	X	0	X	X	X	0		1	0	1	0	0
0	0	X	X	1	1	X	X	X	X	0		1	0	1	0	1
0	1	X	X	X	X	0	X	X	X	0		1	0	1	0	1
1	0	X	X	1	0	X	X	X	0	0		1	0	1	1	0
1	1	0	0	1	0	X	X	X	0	0		1	0	1	1	0
0	0	0	X	1	0	X	1	0	0	0		1	0	1	1	1
0	0	1	X	1	0	0	1	0	0	0		1	1	0	0	0
0	0	1	X	1	0	1	1	0	0	0		1	1	0	0	1
0	0	X	X	1	0	X	0	0	0	0		1	1	0	1	0
0	0	X	X	1	0	X	X	1	0	0		1	1	0	1	1
1	1	0	1	X	X	X	X	X	X	0		1	1	1	0	0
0	0	X	X	1	0	X	X	X	1	0		1	1	1	0	1
1	1	1	1	X	X	X	X	X	X	X		1	1	1	1	0
X	X	X	X	X	X	X	X	X	X	1		1	1	1	1	0
X	X	X	X	X	X	X	X	X	0	X		0	1	1	1	1

Table 5-7 Transactions Selected by LCYCCD Outputs

Code	Transaction	LCYCCD Outputs 8	7	6	5
0	General purpose read from FPA	0	0	0	0
1	General purpose write to FPA	0	0	0	1
2	IDAT data read	0	0	1	0
3	IDAT data write	0	0	1	1
4	DC350/394 data/General purpose read	0	1	0	0
5	DC350/394 data/General purpose write	0	1	0	1
6	EXT bus data read	0	1	1	0
7	EXT bus data write	0	1	1	1
8	EXT bus L byte write	1	0	0	0
9	EXT bus H byte write	1	0	0	1
10	EXT bus FPA write	1	0	1	0
11	Q-Bus DATIO (DATOB)	1	0	1	1
12	Interrupt vector read	1	1	0	0
13	Standalone mode cache write	1	1	0	1
14	Non-I/O or ABORT recovery	1	1	1	0
15	Monitor DMA write cycles	1	1	1	1

5.4.2 Oscillator

The oscillator logic is used to start and stop an oscillator within an 85 ns period and its output drives a multitap delay line to provide the T0 H, T20 H, T40 H, T50 H, T70 H and T85 H outputs, as shown in Figure 5-15. The T40 H output controls the duty cycle of the 85 ns period by turning the oscillator off after 40 ns. The delayed outputs are used to sequence some of the control store signals during a single period of the oscillator.

The logic is controlled by a variety of signals to provide the start, stop, and restart conditions for the oscillator, as shown in Figure 5-16. The control signals for the 16 basic transactions are categorized as primary start signals and secondary restart signals in Table 5-8. The primary start signals are generated by the bus arbitrator as the AUX CYC L and EXT CYC (1) H inputs. The secondary signals are the restart conditions when the KDJ11-B module must wait for a response from other bus devices. These are the reply signal (RRPLY H) or the no reply/timeout signal (CRPLY H). The LAT RDSTRB H input is a restart signal from the PMI bus when the read strobe is latched. The PDMA CYC H or the QDMA CYC H inputs are the primary start signals for monitoring DMA write cycles. The JSTALL L input is enabled when a DMA hit occurs and is the restart signal. The assertion of JSTALL L requires that the addressed location in the cache memory be invalidated.

The oscillator is controlled by a variety of AND gates that are ORed together as part of the control logic. During power-up, the CDCOK H input presets a flip-flop to start the oscillator and the T40 H output is used to clear the flip-flop and establish the duty cycle by asserting the T40 L input. The control store asserts the DL CONT H input and allows the oscillator to continue while the module is initialized. The control store negates the DL CONT H input and when the T20 H, T40 H and T50 H inputs become negated, the DL BUSY L signal is negated, allowing the oscillator to be dormant, as shown in Figure 5-15.

The assertion of AUX CYC L clocks a flip-flop to start the oscillator. The assertion of EXT CYC (1) H input also starts the oscillator, provided the DLYCYC L input is negated. The oscillator is turned off by the control store negation of DL CONT H and DL BUSY L to enable the restart conditions. The LAT RDSTRB H input from the PMI bus can also restart the oscillator, provided the DL BUSY L input is negated. The control store asserts the WT4 RPLY H and the WT4 NRPLY H inputs, and these enable the RRPLY H or the CRPLY H inputs to preset a flip-flop and restart the oscillator.

The PDMA H input is clocked into a flip-flop by RPBCYC H to enable the PDMA CYC H input to start the oscillator. The QDMA CYC H input is able to start the oscillator, provided that SET STL L is negated. The SET STL L input is asserted by the control store to enable JSTALL L to preset the flip-flop and restart the oscillator.

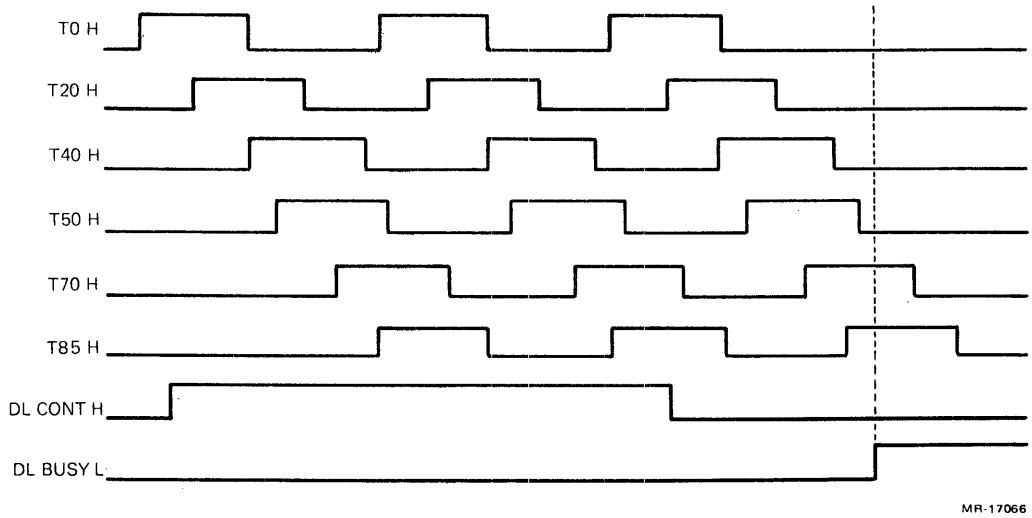


Figure 5-15 Oscillator Outputs

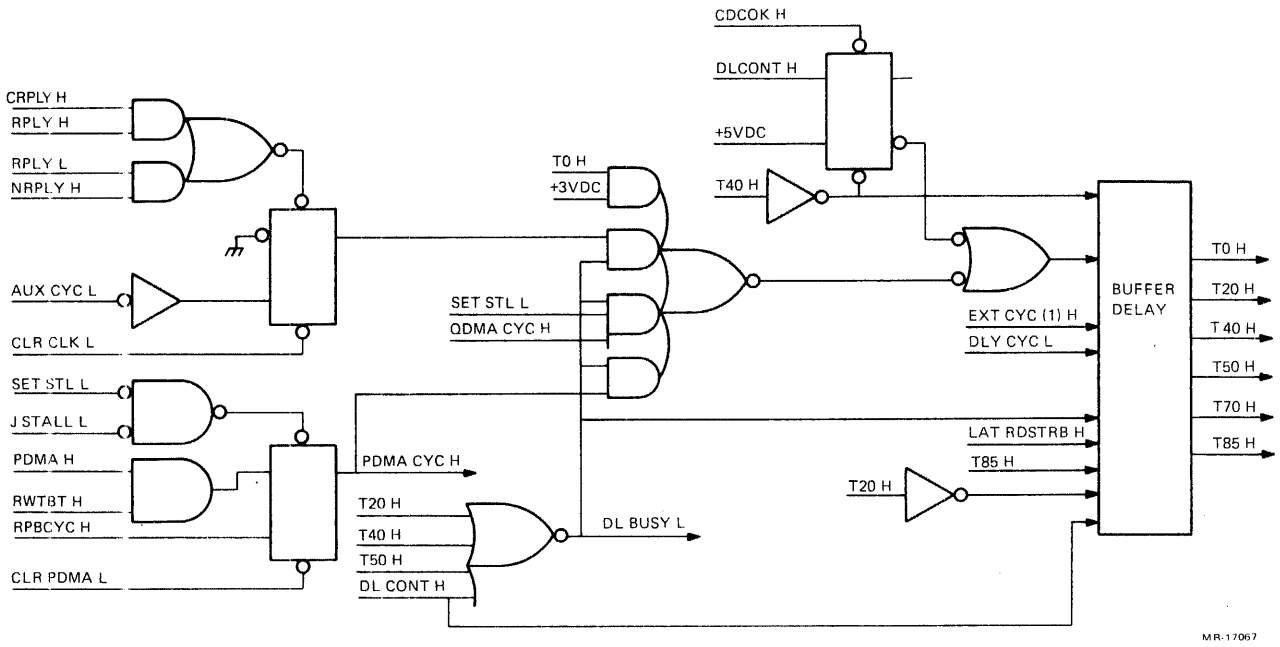


Figure 5-6 Oscillator Control

Table 5-8 Oscillator Control Signals

Code	Transaction	Primary	Secondary
0	General purpose read from FPA	AUX CYC L	None
1	General purpose write to FPA	AUX CYC L	None
2	IDAT data read	AUX CYC L	None
3	IDAT data write	AUX CYC L	None
4	DC350/394 data/General purpose read	AUX CYC L	None
5	DC350/394 data/General purpose write	AUX CYC L	None
6	EXT bus data read	EXT CYC (1) H	CRPLY H, RRPLY H, LAT RDSTRB H
7	EXT bus data write	EXT CYC (1) H	CRPLY H, RRPLY H
8	EXT bus L byte write	EXT CYC (1) H	CRPLY H, RRPLY H
9	EXT bus H byte write	EXT CYC (1) H	CRPLY H, RRPLY H
10	EXT bus FPA write	EXT CYC (1) H	CRPLY H, RRPLY H
11	Q-Bus DATIO (DATOB)	EXT CYC (1) H	CRPLY H, RRPLY H
12	Interrupt vector read	AUX CYC L	CRPLY H, RRPLY H
13	Standalone mode cache write	AUX CYC L	None
14	Non-I/O or ABORT recovery	AUX CYC L	None
15	Monitor DMA write cycles	PDMA CYC H QDMA CYC H	JSTALL L

5.4.3 Next Address MUX

The next address MUX is an 8-bit 2:1 multiplexer that allows the control store to branch within the selected page to a routine that is determined by the conditions sampled at the multiplexer. The next address select signals (NA SEL <2:0>) and the next address bits (NA <1:0>) are driven to the control store as inputs to the multiplexer (Figure 5-17). The NA SEL <2:0> inputs select one of eight MUX input conditions given in Table 5-9.

5.4.3.1 Default – The default condition selects the NA <1:0> signals as they were copied from the control store.

5.4.3.2 External Read/Write – During external data reads and writes, the MUX NA 0 output determines if the bus cycle is a PMI transfer by the status of TPBCYC. The MUX NA 1 output determines if the cache memory should be allocated by the status of either the J CACHE DIS or UBMEM inputs.

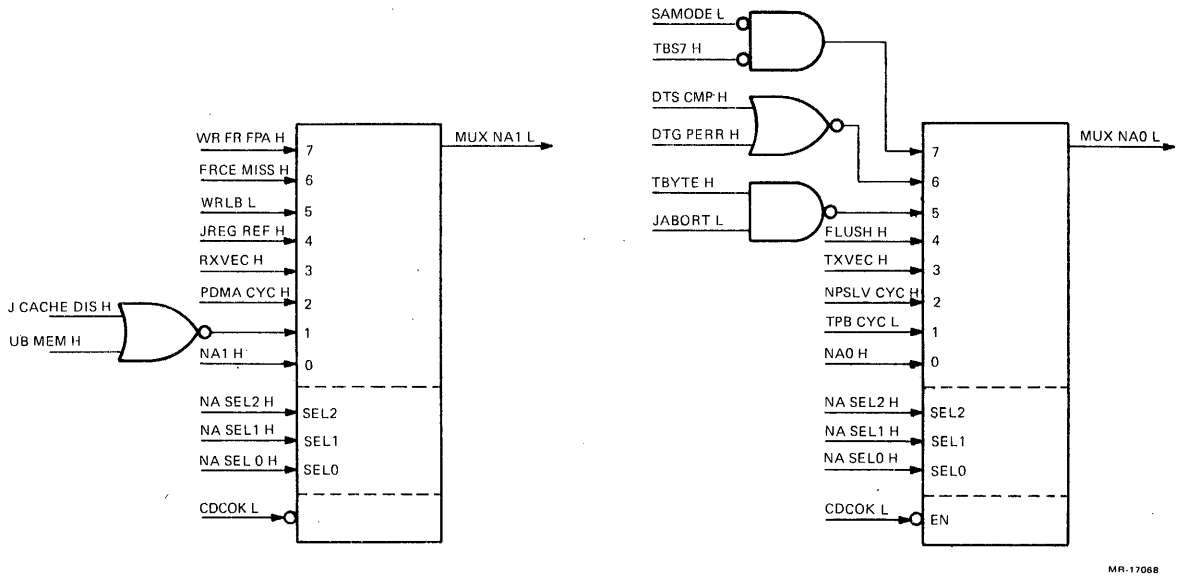


Figure 5-17 Next Address Multiplexer

Table 5-9 Selection of NA <1:0> Status

	NA SEL Input Bits			
Condition Selected	**2**	**1**	**0**	**MUX Inputs**
Default	0	0	0	0
External Read/Write	0	0	1	1
LSI/Unibus	0	1	0	2
Interrupt vector	0	1	1	3
DC350/394 accesses	1	0	0	4
Byte allocation	1	0	1	5
DMA monitor	1	1	0	6
Standalone mode	1	1	1	7

5.4.3.3 LSI/Unibus – During external data reads and writes, the MUX NA 0 output determines if the cycle should be an LSI bus or Unibus transfer by the status of NPSLV CYC. The MUX NA 1 output determines if the cache should be invalidated during an interrupt vector read from the Unibus by the status of PDMA CYC. The controller is stalled while waiting for the interrupt vector or RRPLY. If the Unibus interrupt master performs a DMA transfer, the cache is invalidated.

5.4.3.4 Interrupt Vector – The MUX NA 0 output is determined by the status of TXVEC and the MUX NA 1 output is determined by the status of RXVEC. These inputs have their respective vectors read from the DC350/394 gate array if asserted; otherwise, the vectors are read from the external bus.

5.4.3.5 DC350/394 Accesses – The MUX NA 0 output is determined by the status of FLUSH and if asserted, the cache is flushed. The MUX NA 1 output is determined by the status of JREG REF and if asserted, an internal register is read and the DCJ11-A is not stalled.

5.4.3.6 Byte Allocation – The MUX NA 0 output is determined by the status of TBYTE, provided JABORT L is not asserted. The MUX NA 1 output is determined by the status of WRLB. During external bus writes and standalone mode cache reads, these bits determine which byte may be accessed. These bits are not used for external bus data, H byte and L byte write transactions. During Q-Bus DATIO transactions, the MUX NA 0 output defines the cycle as a DATI transaction for a Read-Modify-Write cycle and inhibits the negation of SYNC until the transaction is complete.

5.4.3.7 DMA Monitor – The MUX NA 0 output is determined by the status of DTS CMP and DTG PERR. The MUX NA 1 output is determined by the status of FRCE MISS. When monitoring DMA cycles, these outputs determine if the cache should be invalidated because of a DMA hit or the assertion of FRCE MISS H.

5.4.3.8 Standalone Mode – The MUX NA 0 output is determined by the standalone mode input, provided the TBS7 H input is negated. The MUX NA 1 output is determined by the status of WR FM FPA. During DC350/394 transactions, the MUX NA 0 output specifies that the DMA tag store parity bit and the tag store be validated if the cycle was a standalone mode read cache miss. During standalone mode cache accesses, the MUX NA 1 output determines if the write data is obtained from the DCJ11-A or the FPA.

5.4.4 Control Store

The control store is a 512 × 48 PROM that contains 16 pages. Each page has 32 entries that are 48 bits wide, as shown in Figure 5-18. The 16 pages represent the 16 basic transactions selected by the cycle encoder logic and those outputs are address bits <8:5>. Each entry in a page is sequenced to execute the transaction by using the NA <4:2> outputs as address bits <4:2>. Address bits NA <1:0> are selected by the NA SEL <2:0> outputs that select an output from the next address MUX. This allows the routine to branch within a selected page. At the end of each page, the control store resets the system back to an initialized state before exiting the page. The signals provided by the control store are listed in Table 5-10. Some of them are modified by the time delay signals from the oscillator.

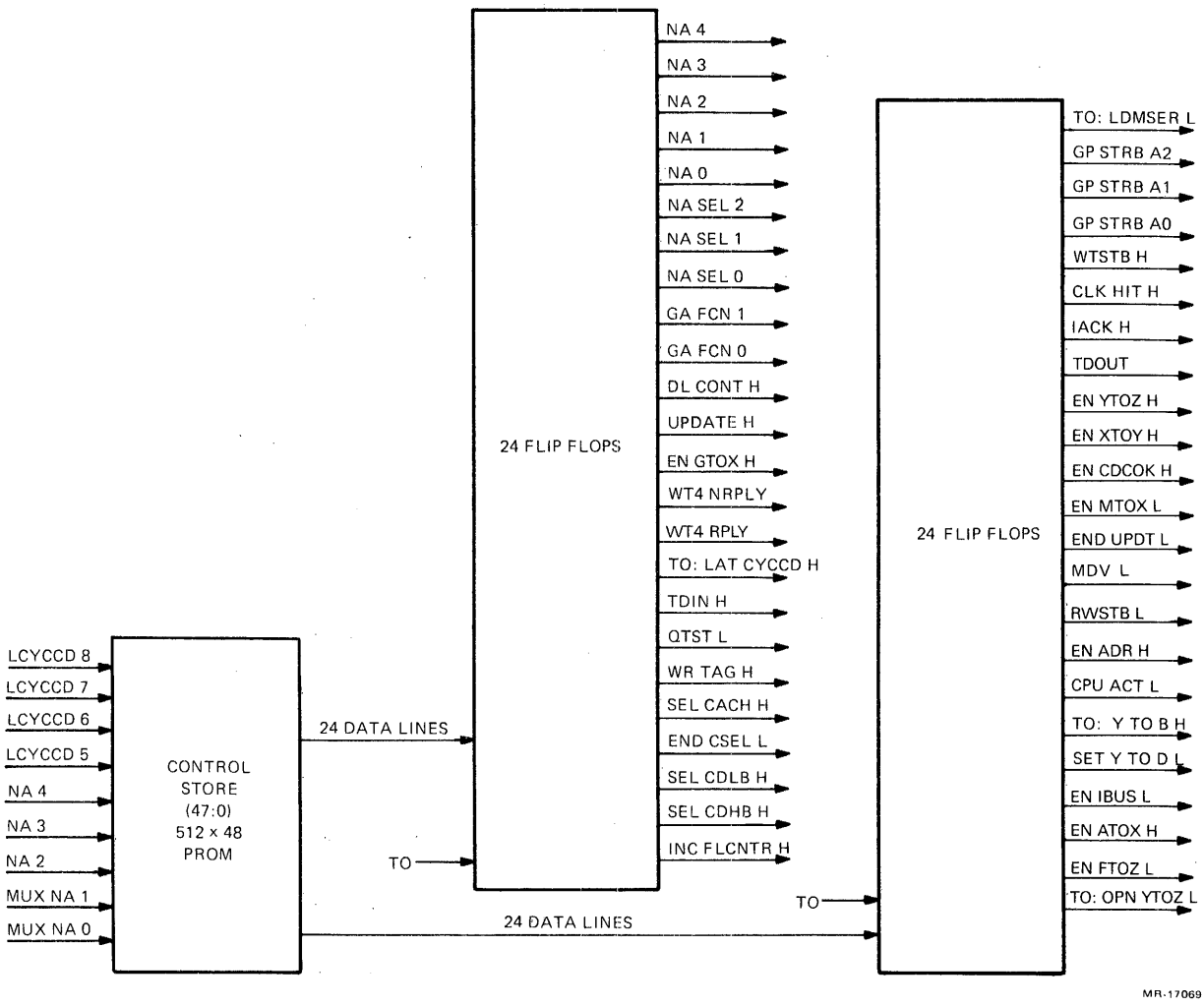


Figure 5-18 Control Store

Table 5-10 Control Store Outputs

Signal	Function
NA <4:2>	Control store address bits <4:2>.
NA <1:0>	Control store default next address bits <1:0> to MUX.
NA SEL <2:0>	Next address MUX selection code.
GA FCN <1:0>	DC350/394 gate array encodes the following functions. 0 Selects the address register 1 Selects the console RX vector 2 Selects the console TX vector 3 Selects another DC350/394 register
DL CONT H	Clocked at T20, enables the oscillator to continue running.
UPDATE H	Allows updating of the tag, parity and valid bits in the DMA and cache tag store.
EN GTOX H	Enables the DC350/394 gate array to drive the XDAL bus.
WT4 NRPLY H	Used to restart the oscillator when there is no RPLY.
WT4 RPLY	Used to restart the oscillator when there is RPLY.
T0:LAT CYCCD H	Opens the latches for the CYCCD selections at T0 and latches the data at T20.
TDIN H	Clocked as T50:DIN H, LSI-11 bus DATI strobe.
QTST L	Enables the QSYNC and opens the XTOY bus on FPA/PMI write dumps.
WR TAG H	Enables cache writes.
SEL CACH H	Selects both bytes of the cache.
END CSEL L	Clocked as T20:END CSEL L to disable cache byte selects at T20.
SEL CDLB H	Clocked as T50:SEL CDLB H to select cache low byte at T50.
SEL CDHB H	Clocked as T50:SEL CDHB H to select cache high byte at T50.
INC FLCNTR H	Increments the flush counter
T0:LD MSER L	Clocked as T50:LD MSER L to load the error status into the MSER at T50.

Table 5-10 Control Store Outputs (Cont)

Signal	Function
GP STRB <A2:A0>	General purpose strobes encoded to select output signals as follows.
CLR CLK L	0 Resets the clock flip-flop
SET RMW L	1 Sets the RMW flag for LSI-11 bus DATIO cycles
CLR LSYNC L	2 Resets SYNC, DHIT flip-flop and clears YTOD latch
CLR YTOD L	3 Opens DMA address latch in the DC351 gate array
CLR SEL L	4 Resets the external cycle requestflip-flop
SET STL L	5 Enables JSTALL to restart the oscillator
CLR PDMA L	6 Clears the PDMA flip-flop
NC	7 Not used
WTSTB H	Write data strobe for PMI.
CLK HIT H	Samples the DMA tag comparator.
IACK H	Interrupt acknowledge.
TDOUT H	LSI-11 bus DATO strobe.
EN YTOZ H	Enables the YDAL bus to the ZDAL bus.
EN XTOY H	Enables the XDAL bus to the YDAL bus.
EN CDCOK H	Enables the CDCOK H signal.
END UPDT L	Clocked as T50:END UPDT L to disable various bus control signals.
EN MTOX L	Enables the MDAL bus to the XDAL bus.
MDV L	Enables the DCJ11-A data strobe.
RWSTB L	Read/Write strobe for IADR bus or DC350/394 gate array.
EN ADR H	Enables BDAL addressing and is byte control for LSI-11 DATIO cycles.
CPU ACT L	Enables TPBCYC L signal during external bus cycles.
T0:YTOB H	Clocked as T50:YTOB L to enable the YDAL bus to the BDAL bus.
SET YTOD L	Latches the DMA address buffer.
EN IBUS L	Enables the IDAT bus to and from the YDAL bus.
EN ATOX H	Clocked as T20:EN ATOX H to enable the address register to the XDAL bus.
T0:EN FTOZ L	Clocked as T50:EN FTOZ L to enable the FPA to the ZDAL bus.
T0:OPN YTOZ L	Clocked at T40 as OPN YTOZ (1) H to open the ZDAL buffer.

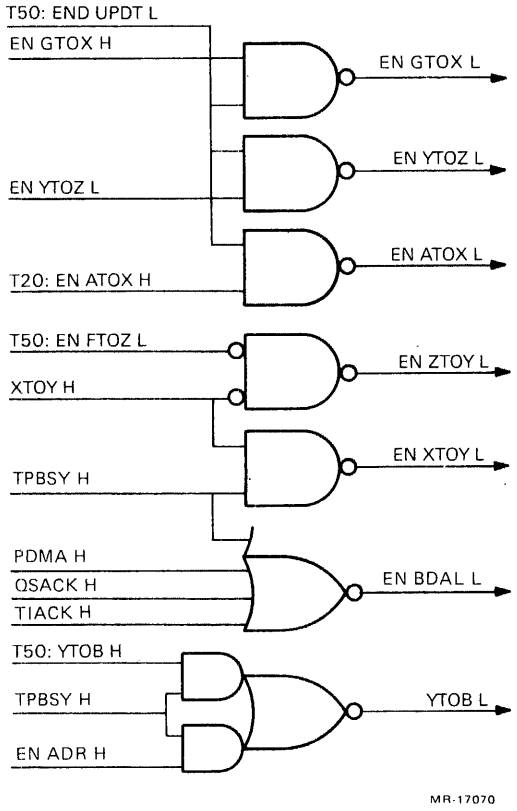


Figure 5-19 Internal Bus Control Signals

Additional internal bus control signals are generated from the control store output signals as shown in Figure 5-19. The EN GTOX L, EN YTOX L and EN ATOX L signals are asserted by their respective inputs; EN GTOX H, EN YTOX H and EN ATOX H are asserted when the T50:END UPDT L input is negated. The EN ZTOY L output is enabled by asserting T50:EN FTOZ L while the XTOY H input is negated. The EN XTOY L output is asserted by having both the XTOY H and TPBSY H inputs asserted.

The LSI-11 address and data bus transceivers are enabled to the YDAL bus by the assertion of EN BDAL L, and the YDAL bus drives the BDAL bus when YTOB L is asserted. Whenever TPBSY H, PDMA H, QSACK H or TIACK H is asserted, the EN BDAL L output is asserted. The YTOB L output is asserted when either T50:YTOB H or EN ADR H inputs are asserted while TPBSY H is also asserted.

5.5 CACHE MEMORY AND DMA STORE

The cache memory consists of a 4K RAM for data storage, the cache tag store, the data parity generating logic, the valid tag bit logic and the cache control logic, as shown in Figure 5-20. The cache memory is used to temporarily store data received from the system memory that the DCJ11-A is currently using. This allows the DCJ11-A to quickly access on-board data without performing external bus transactions.

The physical address is divided into three sections as shown in Figure 5-21. The byte select bit is used to access either high or low bytes of data. The index bits are used as the address of the cache memory. The label bits are stored as the tag store for valid cache entries. Each cache entry is organized as shown in Figure 5-22. The high and low data bytes are stored as data. The label bits with a tag valid bit (V) and the tag parity bit (P) as even parity are stored as tag store data. The low byte parity (P0) is stored as even parity and the high byte parity (P1) is stored as odd parity in the tag store. The byte parity is predicted by the cache data parity logic. The DMA store is an identical copy of the cache tag store and it is used to monitor the main memory DMA updates while the cache tag store monitors the DCJ11-A requirements.

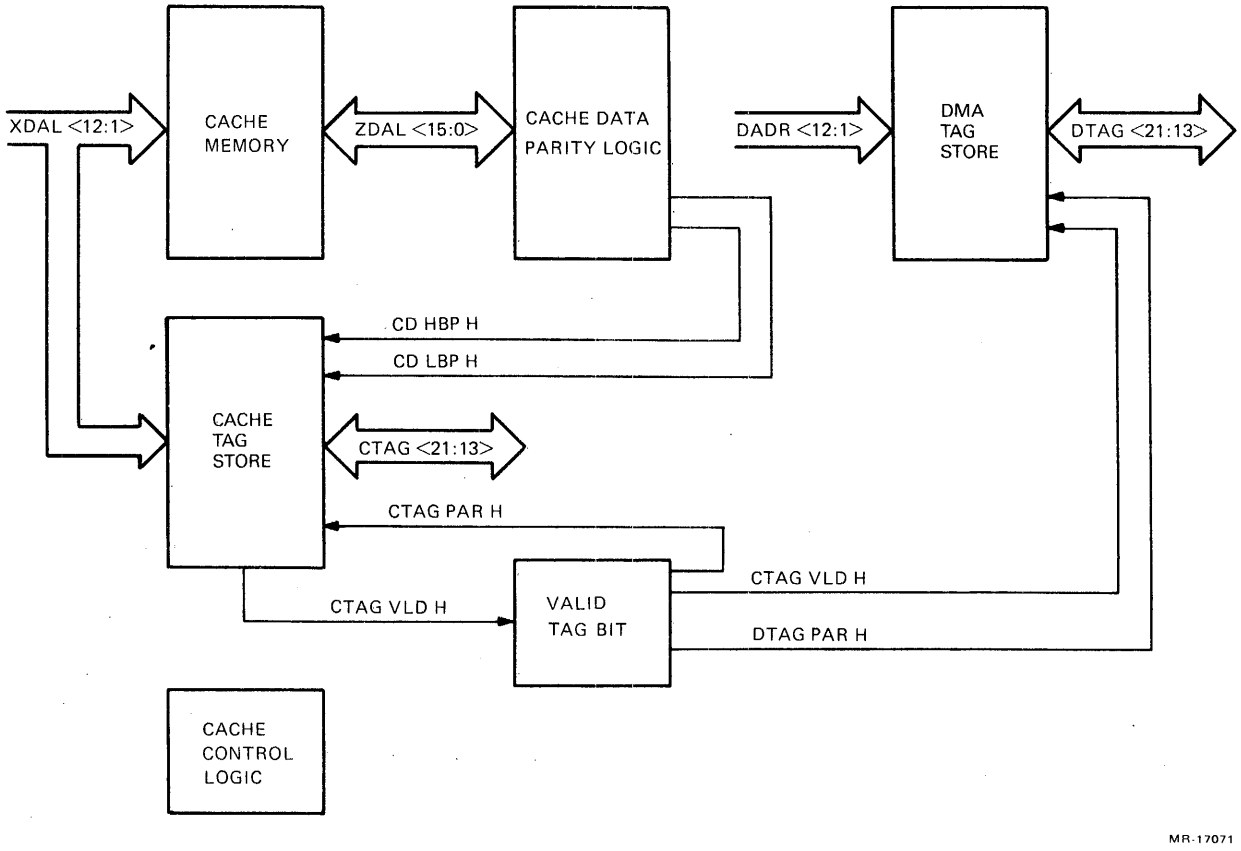


Figure 5-20 Cache Memory System

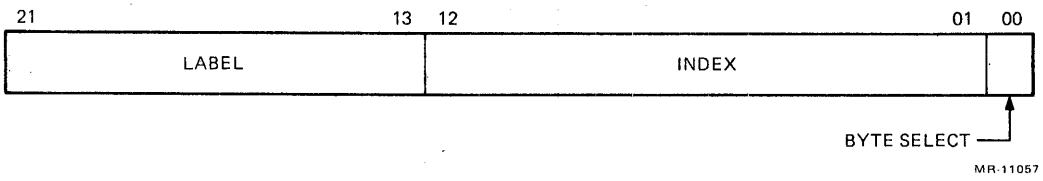


Figure 5-21 Cache Physical Address

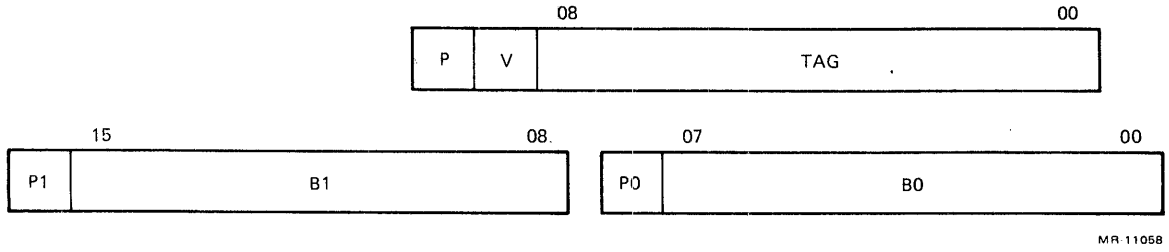


Figure 5-22 Cache Data Format

5.5.1 Cache Memory

The cache data RAM (Figure 5-23) is 8 Kbytes of read/write memory addressed by the index field, that is, XDAL bus bits <12:1>. These bits always access the data stored in an address location, but the data is not validated until the label field of the address is verified as the tag store by the DC350/394 gate array.

The read/write operations are controlled by the cache control signals CSEL CDHB L, CSEL CDLB L and WR CACH L. The low byte of cache data is read when the CSEL CDLB L input is asserted and is written when both the CSEL CDLB L and WR CACH L inputs are asserted. The high byte of cache data is read when the CSEL CDHB L input is asserted and is written when both the CSEL CDHB L and WR CACH L inputs are asserted. The data is routed via the ZDAL bus to the DCJ11-A.

5.5.2 Cache Tag Store

The tag RAM (Figure 5-24) is a 4K × 16 read/write memory that stores 13 bits of data and three bits that are not used. The data consists of the 9-bit label field (address bits <21:13>), the high and low byte data parity bits (CDHBP H and CDLBP H), the tag parity bit (CTAGPAR H), and the tag valid bit (CTAG VLD H). The data is received from the cache data path in the DC350/394 gate array. The read/write operations are controlled by the cache control signals CSEL CDHB L, CSEL CDLB L and WR CACH L. The low byte of the cache tag store data is read when the CSEL CDLB L input is asserted and is written when both the CSEL CDLB L and WR CACH L inputs are asserted. The high byte of cache tag store data is read when the CSEL CDHB L input is asserted and is written when both the CSEL CDHB L and WR CACH L inputs are asserted. The high and low byte data parity bits (CDHBP H and CDLBP H) are used by the cache data parity logic. The cache tag data, the tag parity bit (CTAG PAR H) and the tag valid bit (CTAG VLD H) are used by the DC350/394 gate array.

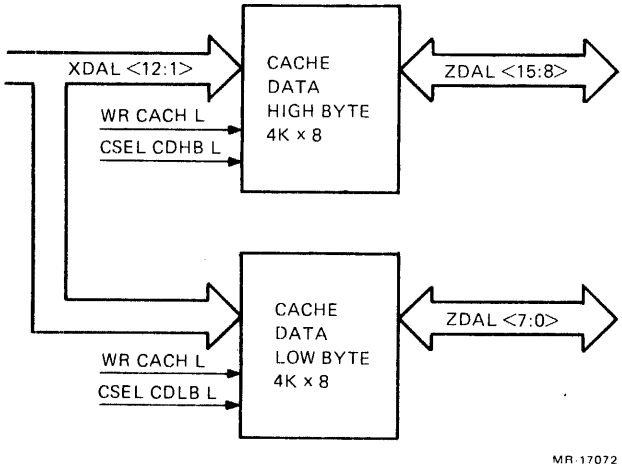


Figure 5-23 Cache Memory

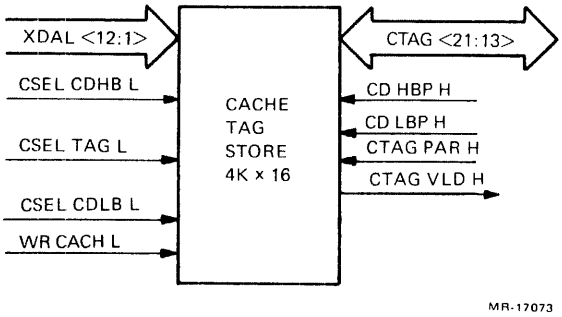


Figure 5-24 Cache Tag Store

5.5.3 Cache Data Parity Logic

The cache data parity logic provides an even parity bit (CDLBP H) for the low byte of data and an odd parity bit (CDHBP H) for the high byte of data. It also checks these parity bits when the data is accessed from the cache memory and reports any errors as the HB PERR H and LB PERR H outputs to the DCJ11-A and the DC350/394 gate array.

The high byte parity bit (PHBPAR L) and low byte parity bit (PLBPAR L) signals are received from memory when the cache memory is being updated, as shown in Figure 5-25. These inputs are buffered and latched by the assertion of the COPN YTOZ H input. The latched outputs provide a correct parity bit (HBPAR H, LBPAR H) and a wrong parity bit (HBPAR L, LBPAR L) as inputs to the 4-bit multiplexer.

The data from the cache memory is read via the ZDAL bus to the 9-bit parity generator. The respective parity bit is also an input to the parity generator, provided the UPDATE L signal is negated. An odd parity bit and an even parity bit are generated on the 9-bit input by the parity generator and both of these bits are inputs to the 4-bit multiplexer.

The inputs to the multiplexer represent the original parity bit, a predicted parity bit and two wrong parity bits. The EN PRD H input selects the predicted parity when asserted and the correct parity bit when negated. The asserted WR WRONG PAR H input selects one of the two wrong parity bits, depending on the status of EN PRD H. The selected output determines the status of the CDLBP H or CDHBP H outputs, when they are enabled (Table 5-11), by asserting the UPDATE L output.

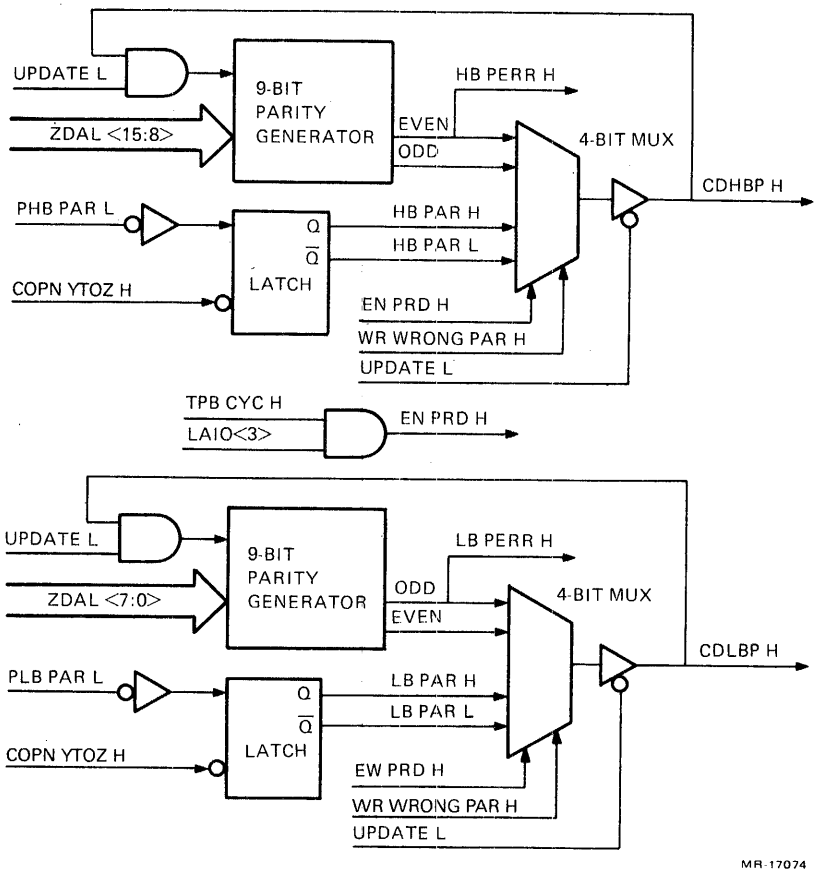


Figure 5-25 Cache Data Parity Logic

Table 5-11 Cache Parity

EN PRD H	WR WRONG PAR H	CDHBP H	CDLBP H
0	0	Even parity	Odd parity
1	0	HB PAR H	LB PAR H
0	1	Odd parity	Even parity
1	1	HB PAR L	LB PAR L

5.5.4 Valid Tag Bit

The valid tag bits for the DMA tag store and the cache tag store are generated by the same logic, as shown in Figure 5-26. The DTAG VLD H and CTAG VLD H are driven by the tag valid bit logic and are enabled by the assertion of the UPDATE L signal. The DTAG PAR H output is also copied from the CTAG PAR H signal when UPDATE L is asserted. The valid bit output is good, provided the following four conditions are valid.

1. The JBS1 H signal is negated to indicate that the address is either for memory or system board register.

2. The address is not a nonexistent memory (NXM L negate), or an address invalidated by DMFL INV L negated.

3. The COPN YTOZ H signal is negated, or parity address bit 17 or 16 is negated.

4. The COPN YTOZ L signal is asserted. The predict signal EN PRD H and the DATI signal T50:DIN H are negated.

5.5.5 DMA Tag Store

The DMA tag store (Figure 5-27) is a 4K × 12 read/write memory that stores 11 bits of data and one bit that is not used. The stored data is the same data stored in the cache tag store except for the two cache memory parity bits. The DC351 gate array controls the operation of the DMA tag store. It is addressed by bits <12:1> of the DADR bus and the data is read via DTAG bus. The DTAG VLD H and DTAG PAR H outputs are used by the DC351 gate array to validate the DMA tag store data. This allows the DMA tag store to operate independently from the cache memory and monitor DMA transfers while the DCJ11-A is using the cache memory.

5.5.6 Cache Control

The cache control signals are used to read and write the cache memory, the cache tag store, the DMA tag store and enable the cache parity logic. The input signals used by the cache control logic (Figure 5-28) are from the control store. The CSEL CDHB L, CSEL CDLB L and CSEL TAG L signals are all asserted by asserting the SEL CACH H input. The high byte select signal CSEL CDHB L and the CSEL TAG L signal are asserted when the T50:SEL CDHB H input is asserted while the T20:END CSEL L input is negated The low byte select signal CSEL CDLB L and the CSEL TAG L signal are asserted when the T50:SEL CDLB H input is asserted while the T20:END CSEL L input is negated.

The WR CACH L signal is enabled by asserting WR TAG H while T50:END UPDT L is negated. UPDATE L is enabled by asserting UPDATE H while T50:END UPDT L is negated.

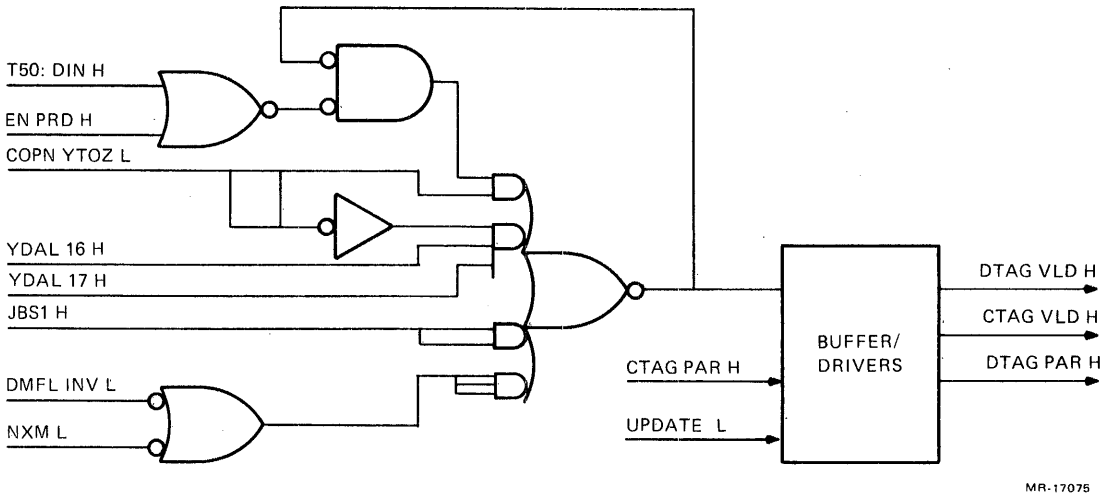


Figure 5-26 Valid Tag Bit

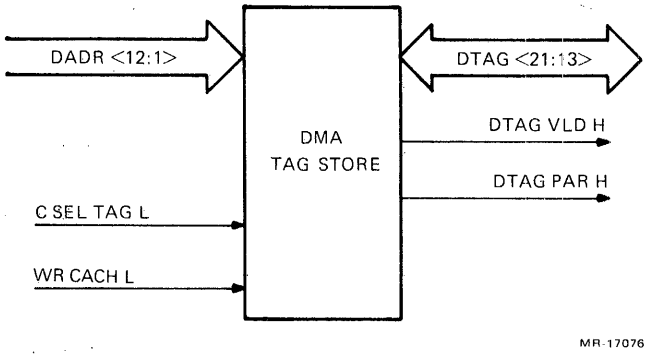


Figure 5-27 DMA Tag Store

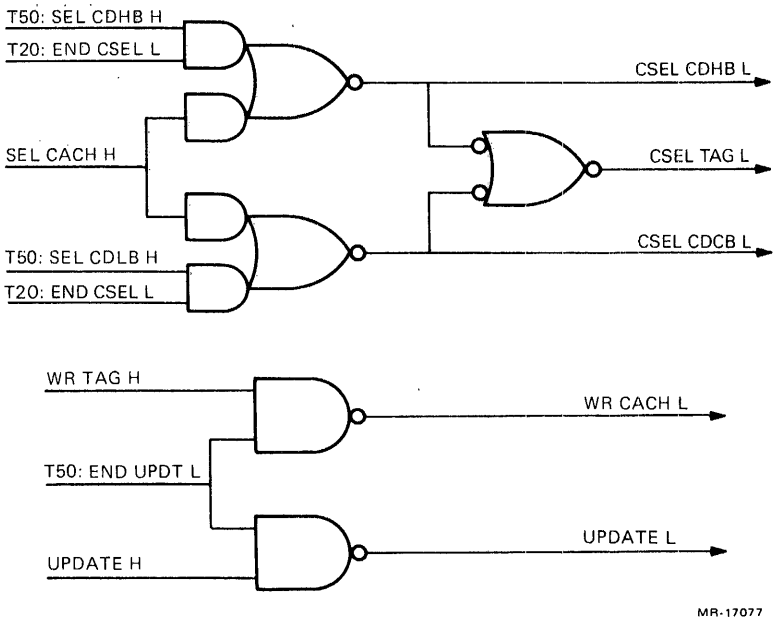


Figure 5-28 Cache Control Signals

5.6 DC350/394 GATE ARRAY

The DC350/394 gate array (Figure 5-29) contains many of the functions of the KDJ11-B module. It controls the cache data path, the address decoding for the on-board registers, the parity interrupts and aborts, the KDJ11-B bus requests and the bus arbitrator logic, which is described in Paragraph 5.3.

A copy of the cache control register is stored in the gate array. In addition, the DC350/394 gate array contains the following on-board registers and the console vector generator to access an 8:1 multiplexer (AMUX) that drives the XDAL and CTAG busses.

On-Board Registers

Address
Boot and diagnostic Control/Status (BCSR)
Memory System Error (MSER)
Line Time Clock status (LTC)
Maintenance (MTRG)
Page Control (PCR)

Console Vector Generator

Receive vector 60
Transmit vector 64

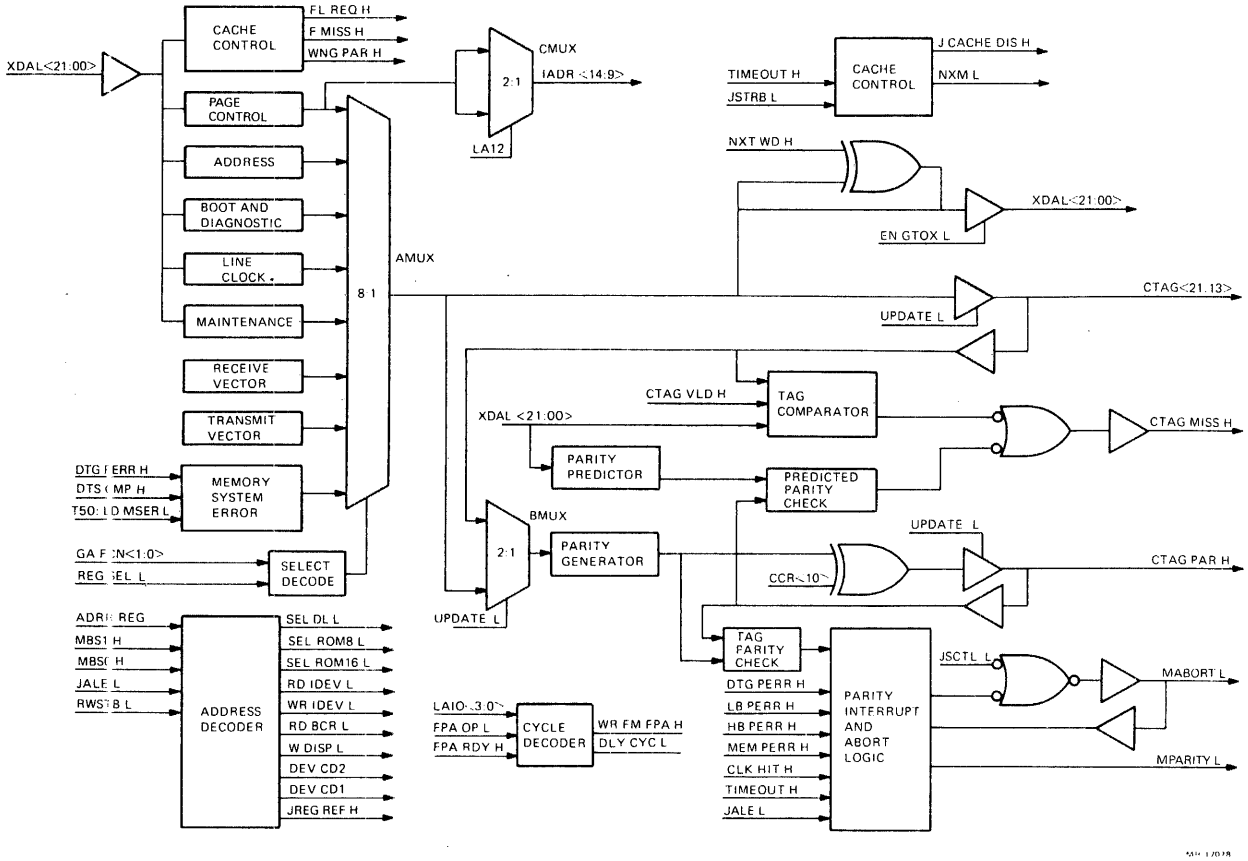


Figure 5-29 DC350/394 Gate Array

The address register latches the 22-bit physical address from the XDAL bus when the JSTRB L input is asserted at the start of every cycle. Bits <22:13> of this register are used as the tag store data and bits <12:0> are used for decoding the addresses of the explicitly addressable registers and addressing the cache memory. Bit 12 is used to select a 2:1 multiplexer (CMUX) input from the PCR that drives bits <14:9> of the IADR bus.

The BCSR is used to control the boot and diagnostic ROMs. It asserts the standalone mode output (SAMODE L) when bit 8 is set and the ENB HOB H output when bit 9 is set. The battery backup failure input (BBRBE H) is used to set bit 15 when the RPOK H input is asserted.

The MSER monitors the status of parity errors in the memory system. The contents of the register is updated from the parity/abort logic when the T50:LD MSER L input is asserted. The DTS CMP H input is used to set bit 14 and the DTG PERR H input is used to set bit 13 when the CLK HIT H input is asserted.

The LTC status register, enabled by setting bit 6, allows the clock logic to generate clock interrupts by asserting the MEVENT L output. The interrupts are determined by clock select bits 10 and 11 of the BCSR register, as shown in Table 5-12 . The logic uses REVNT H, KDJ800 H and KDJ60 H inputs as the timing base of the interrupts.

The MTRG maintains the status of the system and is read by the DCJ11-A during the power-up routine. The PSLV H, UBSYS H, FPA OP L and RPOK H inputs are used by bits <10:8> and 0 to indicate the system status.

The PCR uses bits <14:9> or <6:1> to drive the IADR bus via the CMUX, depending on the status of bit 12 in the address register. Boot ROM address 17 773 000 uses bits <14:9> as an address and boot ROM address 17 765 000 uses bits <6:1> as an address.

The CCR is a copy of the cache control register in the DCJ11-A and is used to control the parity and abort functions. The FLSHRQ H output is asserted when bit 8 is set and is the flush cache request to the DC351 gate array. The FRCE MISS H output is asserted when bit 3 or bit 2 is set and all DCJ11-A reads are reported as misses. The WR WRONG PAR H output is asserted when bit 6 is set and the wrong data parity is written for both bytes on DCJ11-A read misses and write hits.

The console vector generator provides the receive and transmit vectors when these interrupts are acknowledged by the DCJ11-A.

Table 5-12 LTC Interrupts

CLK SEL 1	CLK SEL 0	Interrupt Source
0	0	REVNT H
0	1	KDJ800/16 or 50 Hz
1	0	KDJ60 or 60 Hz
1	1	KDJ800 or 800 Hz

Table 5-13 AMUX Selections

GAFCN1	GAFCN0	BCSR	MSER	LTC	MTRG	PCR	Selected Data
0	0	X	X	X	X	X	Address register
0	1	X	X	X	X	X	Receive vector 60
1	0	X	X	X	X	X	Transmit vector 64
1	1	1	0	0	0	0	Boot and configuration
1	1	0	1	0	0	0	Memory system error
1	1	0	0	1	0	0	Line time clock status
1	1	0	0	0	1	0	Maintenance
1	1	0	0	0	0	0	Page control

5.6.1 A-Multiplexer

The A-Multiplexer (AMUX) is a 16-bit, 8:1 multiplexer that interfaces to the XDAL bus. It is controlled by the GAFCN1 and GAFCN0 inputs, along with the address decode logic outputs for the BCSR, MSER, LTC, MTRG, and PCR register selections. The AMUX is used to select a register or vector (Table 5-13) and drive its contents onto the XDAL bus.

5.6.2 Cache Data Path

The Cache Data Path (CDP) uses the address register, the AMUX, the 2:1 B multiplexer (BMUX), a parity generator, a parity predictor, a tag comparator and the cache control logic. When the cache is being written, XDAL bits <12:1> are the index bits used via the XDAL bus to address the cache memory location where the data is stored. The EN GTOX L input is asserted to enable the gate array or the AMUX output to drive the XDAL bus. The NXT WD H input is asserted to increment the current address and enable the second word from the system to be loaded into the next cache location. XDAL bits <22:13> are the label data driven via the CTAG bus when the UPDATE L input is asserted and stored at the same address in the tag store. An even parity bit is generated for the tag data and can be modified when the write wrong tag parity bit (bit 10) in the CCR is set. The tag parity bit is stored with the label data when the UPDATE L input is asserted.

An address from the DCJ11-A is received and the CDP must validate it as a hit or miss for the cache memory. The address is latched in the address register and index bits <12:1> are driven onto the XDAL bus via the AMUX to address the cache memory. The cache store data is read via CTAG bus bits <21:13>, along with the CTAG PAR H and the CTAG VLD H bits. Address bits <21:13> are compared with tag data bits <12:13> by the tag comparator and an error is reported to the DCJ11-A by the assertion of the CTG MISS H output. A parity bit is predicted for address bits <21:13> and another parity bit is produced for tag data bits <21:13> by the parity generator. Both of these parity bits are checked with the CTAG PAR H input by the assertion of the CTAG VLD H input. An error in the predicted parity enables the CTG MISS H output, and an error in the tag parity check is reported to the parity interrupt and abort logic by the negation of the JALE L input.

The cache control logic asserts the NXM L output when the TIMEOUT H input is asserted, except for word write cycles with bit 1 set in the CCR. The output is cleared at the end of the cycle by the assertion of the JSTRB L input. The cache control also asserts the JCACHE DIS H output for any I/O page access, force cache miss, read or write miss with bypass set for no parity errors or nonbypass miss with no tag parity errors.

5.6.3 Parity Interrupt and Abort

The parity interrupt and abort logic drives the MPARITY L and MABORT L outputs to the DCJ11-A. The MABORT L output is enabled only when the JSCTL L input is asserted; otherwise, it is used as an input from the DCJ11-A. Asserting the MABORT L output alone results in an abort to location 4. Asserting the MPARITY L output alone results in an interrupt to location 114. When both outputs are asserted, a parity abort to location 114 is also generated.

The logic uses the DTG PERR H, HB PERR H, LB PERR H, MEM PERR H, CLK HIT H and TIMEOUT H inputs. It also uses the internally generated tag parity error signal. These inputs represent parity errors from the cache memory, the main memory, the cache tag and the DMA tag. The TIMEOUT H input is asserted whenever nonexistent memory is addressed. These inputs are used with bits 7 and 0 of the CCR and the read request (RD REQ) signal from the cycle decode logic. The operations of the parity interrupt and abort logic are described in Table 5-14. The CPE status represents any cache parity error.

The parity interrupts and aborts can only occur if bits <7:4> of the MSER register were cleared after any previous cache parity error. These bits represent cache parity error status. This allows the system to respond to a nonfatal cache parity error without being interrupted for additional parity errors. When bit 4 of the MSER register is set, bits 7 and 0 of the CCR can enable the outputs as shown in Table 5-15.

The MPARITY L output is asserted for main memory or cache parity errors provided CCR bit 0 is clear and CCR bit 7 is set. The MABORT L output is asserted when a cache parity error occurs during a request read, for any main memory parity error, for a nonexistent address, and whenever a cache parity error occurs when CCR bit 7 is set.

Table 5-14 Parity Interrupt and Abort Logic

REQ RD	CPE	CCR7	CCR0	NXM	MEMPERR	MPARITY L	MABORT L
X	0	X	X	0	0	Negated	Negated
X	X	X	X	1	X	Negated	Asserted
X	X	X	X	0	1	Asserted	Asserted
1	1	X	X	0	0	X	Asserted
0	1	0	0	0	0	Asserted	Negated
0	1	0	1	0	0	Negated	Negated
0	1	1	X	0	0	Asserted	Asserted

Table 5-15 CCR Register Selections

CCR7	CCR0	MPARITY L	MABORT L
0	0	Asserted	Negated
0	1	Negated	Negated
1	X	Asserted	Negated

5.6.4 Address Decode

The on-board register addresses are decoded as shown in Table 5-16. The decoder uses address register bits <12:1> and the bank select inputs MBS1 H and MBS0 H. The BCSR bits 12 and <7:5> are monitored since they can enable or disable the LTC register and the boot addresses.

The address decoder enables the SELDL L, SEL ROM8 L or SEL ROM16 L outputs when addressed, and the selected output is asserted when the JALE L input is asserted. The on-board device read and write signals, RDIDEV L and WRIDEV L, are enabled by the read and write cycles and are asserted when the RWSTB L input is asserted.

The CDR is read onto the IDAT bus when addressed and the RDBCR L output is asserted during a read cycle. During a write cycle, the WRDISP L output is enabled and the data is written in the LED display register when the RWSTB L input is asserted.

Registers that are external to the DC350/394 gate array are encoded by the DEVCD2 H and DEVCD1 H outputs to the cycle encoder, as shown in Table 5-17. The JREG REF H output is asserted when a DCJ11-A internal register is selected.

The address decode logic also enables the select decode logic to send the RWSTB L input to the internal registers on a write cycle. The RWSTB L input also goes to the abort logic for nonexistent module addresses, standalone mode bus read misses, bypass and force misses, and I/O page references not on the module.

Table 5-16 Address Decoding

Address Bits <12:1>	MBS1	MBS0	Decoded Address
X XXX XXX XX0 00	0	1	Nonexistent register
X XXX XXX XX0 01	0	1	Nonexistent register
X XXX XXX XX0 10	0	1	MSER
X XXX XXX XX0 11	0	1	CCR
X XXX XXX XX1 00	0	1	MTRG
1 111 101 010 00	1	0	BCSR
1 111 101 010 01	1	0	PCR
1 111 101 010 10	1	0	CDR
1 111 101 100 11	1	0	LTC register
1 111 101 110 XX	1	0	Console SLU
1 011 XXX XXX XX	1	0	Boot address 17 773 000
0 101 XXX XXX XX	1	0	Boot address 17 765 000

Table 5-17 DEVCD Outputs

DEVCD2	DEVCD1	Selection
0	X	Internal device decoded
1	0	No selection
1	1	Register within the gate array
1	1	DCJ11-A internal register
1	1	NXM abort during standalone mode

Table 5-18 Cycle Decoding

LAIO 3	2	1	0	Cycle Selected
1	1	1	1	Non-I/O
1	1	0	0	Request read
1	0	X	X	Demand read
0	1	X	X	General purpose write
1	1	1	0	General purpose read
0	0	0	X	Word write
0	0	1	X	Byte write

5.6.5 Cycle Decoder

The cycle decoder decodes the LAIO <3:0> H inputs to control the various functions within the DC350/394 gate array, as shown in Table 5-18. The read/write cycles go to the address decoder for register selection and the request read and non-I/O cycles go to the parity/abort logic. A request read forces an abort on cache parity errors and a non-I/O inhibits aborts on stretched non-I/O cycles. The TBYTE L output is asserted for all read-modify-writes and external bus write byte cycles.

When any write cycle is decoded and the FPA OP L input is asserted, the FPA write cycle output WRFM FPA L is asserted. The FPA DLY H output is asserted when the FPA data is not ready, causing the FPA RDY H input to be negated while the WRFM FPA L output is asserted. The FPA DLY H or the RSBFUL H signal is used as the DLYCYC L output to the delay oscillator and PMI cycle request logic.

5.6.6 Miscellaneous

The TINIT H output is asserted when the RPOK H input is negated or for any general purpose write to location 14, and is negated on any general purpose write to location 214.

The MPWR FAIL L output, which is the power fail interrupt to the DCJ11-A, is asserted by the negation of the RPOK H input. It is negated by a general purpose write to location 140 or when RPOK H is asserted.

The Unibus map enable (PMAPE H) is asserted when the JMAP L input is asserted from the DCJ11-A. The JMAP L input is sampled when the JALE L input is asserted, provided the MABORT L output is not asserted. The PMAPE H output is negated when the RPOK H input is negated.

The JSTALL L output is used to stall the clock oscillator. The output is asserted, provided the JMAP L input is asserted when sampled by the negation of the JALE L input. It is also asserted whenever the JSCTL L input is asserted and whenever the LAIO H bit 3 input is negated for a write cycle.

5.7 DC351 GATE ARRAY

The DC351 gate array controls the DMA tag data path, the clock start logic, the flush counter, and the main memory parity error logic, as shown in Figure 5-30. It uses the 9-bit bidirectional DTAG bus, the 22-bit YDAL bus (16-bits are bidirectional and 6-bits are input only), and the 12-bit output-only DADR bus and XDAL bus. The data from the flush counter and DMA tag data path uses a 12-bit 2:1 multiplexer to drive the outputs on the DADR bus and XDAL bus. The YDAL bus and the IDAT bus are enabled and bidirectional when the EN IBUS L input is asserted. The YDAL bus drives the IDAT bus when the WRITE H input is asserted, and the IDAT bus drives the YDAL bus when the WRITE H input is negated.

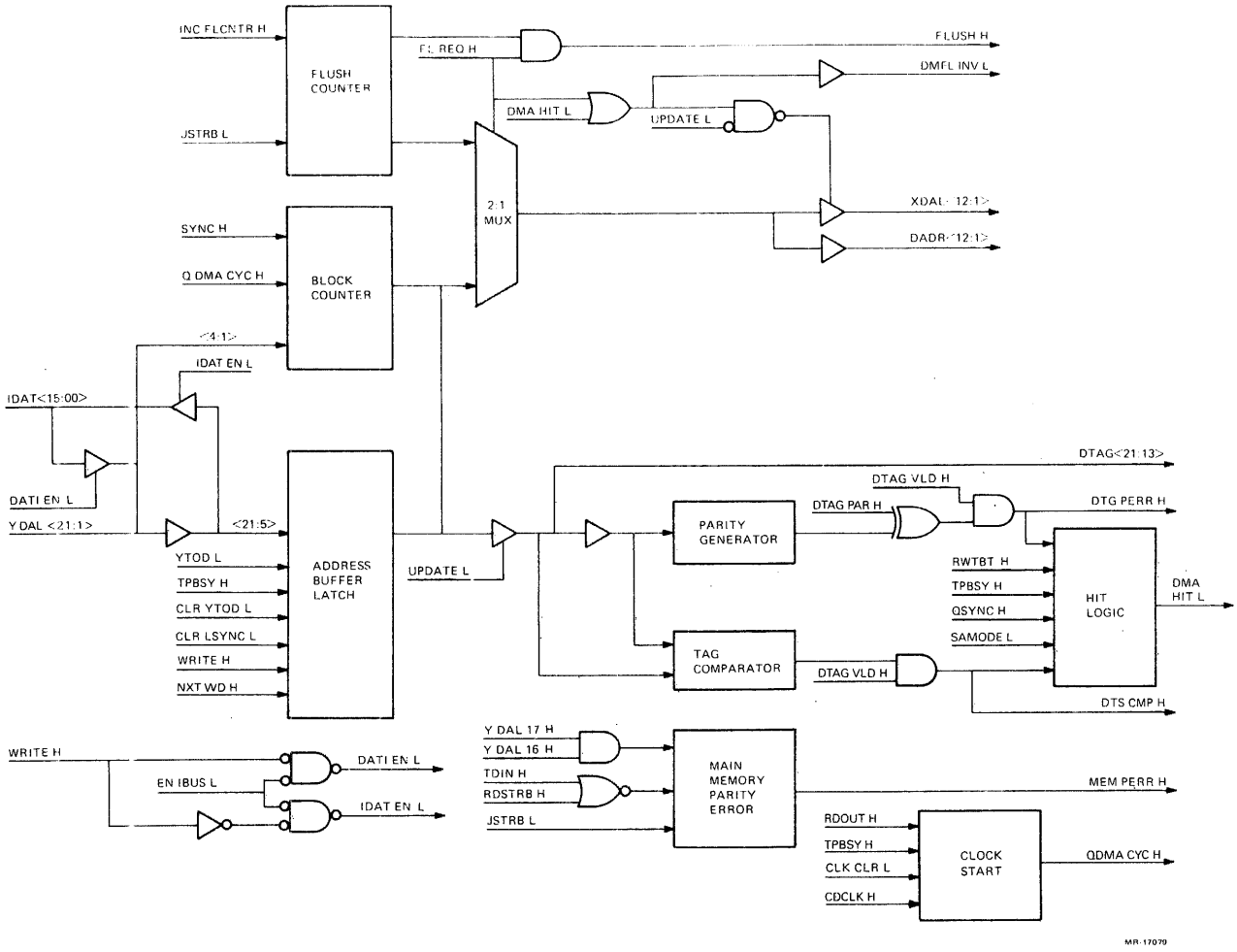


Figure 5-30 DC351 Gate Array

5.7.1 DMA Tag Data Path

The DMA tag data path writes YDAL bus bits <21:13> as the DMA tag data. This data is driven on the DTAG bus to the DMA tag store by the assertion of the UPDATE L input. YDAL bus bits <12:1> are used to address the DMA tag store and are driven on the DADR bus.

The DMA tag data path consists of address buffers and latches, a tag comparator, a parity generator and the hit logic. Any address on the YDAL bus can be compared with the DMA tag store to determine if that address is being used by the cache memory. The latching logic for the YDAL buffers is preset by asserting the SET YTOD L input, and with the TPBSY H input asserted, the address is latched into the buffers by the assertion of the SYNC H input. The latches are reset by asserting the CLR YTOD L input or the CLR LSYNC L input. The NXT WD H input is asserted to increment the address in the buffer enabling the second word address.

During DMA cycles, every address on the YDAL bus is compared with the DMA tag store. The DADR bus addresses the DMA tag store and the data is compared to bits <21:13> of the YDAL bus. A parity bit is also generated for these bits and is compared with the DTAG PAR H input – the stored DMA parity bit. The results of these checks are enabled by the assertion of the DTAG VLD H input. The DTS CMP H output is asserted when a valid comparison of the DMA tag data is made, and the DTG PERR H output is asserted when a parity error is detected. The DMA HIT L output is asserted when there is a valid comparison of data and parity, and the CLK HIT H input is asserted. The DMA HIT L output is also asserted when the RWTBT H and the TPBSY inputs are asserted and sampled by the assertion of the QSYNC H input. The DMA HIT L output is inhibited when the SAMODE L input is asserted during standalone operations. When a hit occurs, the current data is invalidated and the new data is written into the cache memory.

5.7.2 Clock Start Logic

The clock start logic enables the QDMA CYC H output to trigger the delay oscillator during DMA write cycles on the LSI-11 bus. The output is asserted by the assertion of the RDOUT H input, provided the TPBSY H input is negated. The QDMA CYC H output is negated by the assertion of the CLR CLK L input. The clock start logic is initialized by the assertion of the CDCOK H input.

5.7.3 Flush Counter

The contents of the cache memory is flushed or cleared during power-up and whenever bit 8 of the CCR is set. This requires that each address location in the cache tag and DMA tag store is addressed and cleared or invalidated. The DC350/394 gate array asserts the FL REQ H input to initiate the flush sequence. This input allows the 12-bit output from the flush counter to drive DADR <12:1> and XDAL <12:1> bus outputs via the 2:1 multiplexer. It also asserts the FLUSH H output to the next address MUX logic, and the DMFL INV L output is asserted to invalidate the tag valid bit for the tag stores. The flush counter is cleared to zero by the assertion of the JSTRB L input. The DMA tag store is addressed by the DADR bus and the cache tag store is addressed by the XDAL bus.

The flush counter is incremented by the assertion of the INC FLCNTR H input from the control store and the next location in the tag stores is addressed. This cycle continues until all the locations are addressed and cleared. Then the flush counter overflows and negates the FLUSH H output to the next address MUX logic to end the cycle.

5.7.4 Main Memory Parity Error

The parity bits from the main memory are received via YDAL bus bits 17 and 16. When both of these bits are asserted high, the MEM PERR H output is asserted by the negation of the TDIN H input or the RDSTRB H input. The MEM PERR H output indicates that there was a parity error in the read cycle from the main memory. The output is cleared by the assertion of the JSTRB L input.

5.8 TIMEOUT

The KDJ11-B module has two timeout logic circuits – one for the DMA requests and the other for nonexistent memory or interrupt acknowledge. Both of these timeout circuits (Figure 5-31) use the same principle, but are controlled by different signals.

5.8.1 DMA Requests

The DMA request timeout logic uses a monostable multivibrator that is continuously clocked by the TOUT CLK H signal, which is driven by the JCLK H input. The multivibrator is set by the assertion of either RRPLY H, UBSYS H, or the combination of DMG H and TRPLY H. This sets the output low, the CSACK L output is negated and the logic waits for the system to acknowledge the reply by asserting the CSACK L input. The capacitive network allows the multivibrator to run for 10 μseconds before it automatically resets the output and asserts the CSACK L output. However, the CSACK L output can be asserted anytime the QSACK H input is received as the acknowledgment for the reply request.

5.8.2 NXM or Interrupt Requests

The NXM or interrupt request timeout logic uses the same type of logic that the DMA request timeout uses. The multivibrator is set by the assertion of UBSYS H or the negation of WT4RPLY H and waits for the system to acknowledge by asserting the RRPLY H input. The RRPLY H input asserts the CRPLY H output, or the multivibrator times out and asserts the CRPLY H and TIMEOUT H outputs. The assertion of the Unibus timeout signal (PUBTMO H) is allowed to override the logic and assert the TIMEOUT H and CRPLY H outputs.

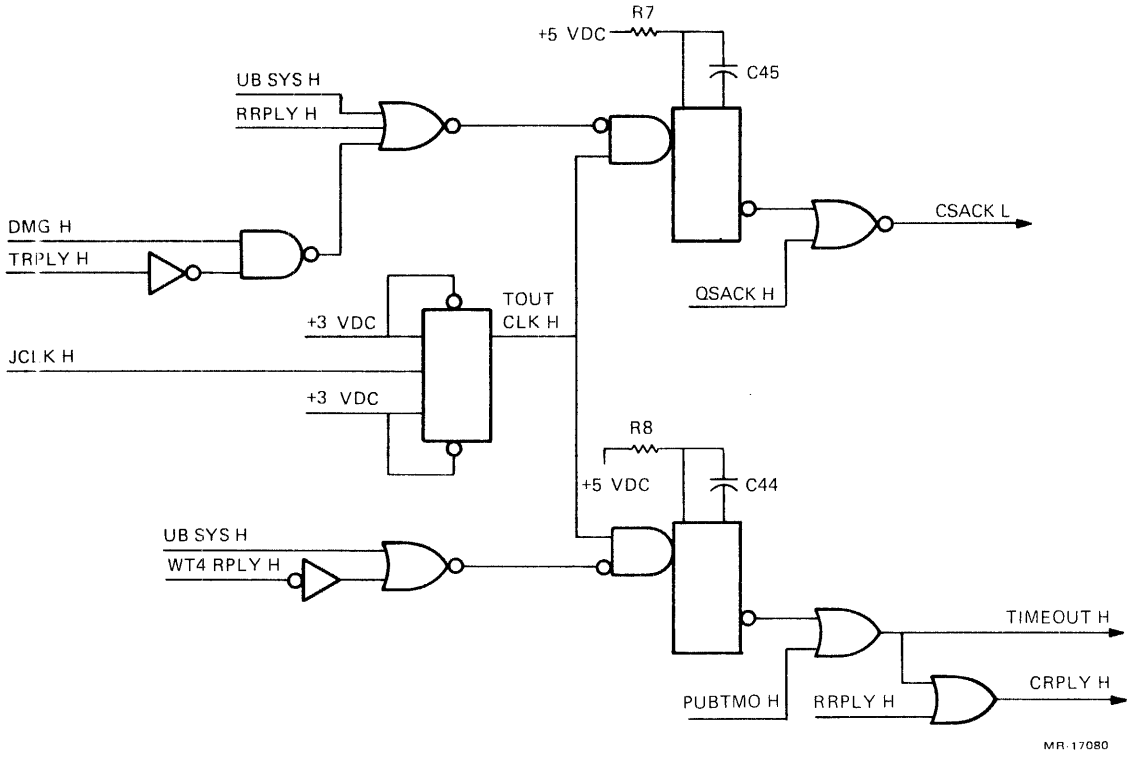


Figure 5-31 NXM/Interrupt Timeout Logic

5.9 BUS DISTRIBUTION

The bus distribution consists of the internal bus control logic, the LSI-11 bus control logic and the PMI bus control logic. This distribution network allows the addressing of any internal register that may be located in the DLART, the DC350/394 gate array, the boot and diagnostic ROMs, or the configuration and display switches. It also allows access to the cache memory and tag stores, the FPA, and the DC351 gate array. The LSI-11 bus can be accessed for standard LSI-11 bus transactions and the private memory, located on the LSI-11 bus, can be accessed for high speed DMA transactions by the PMI control logic.

5.9.1 Internal Bus Control

The internal bus control network routes the addresses and data to and from the various module components via the MDAL, XDAL, YDAL, ZDAL and BDAL busses, as shown in Figure 5-32. The MDAL bus interfaces with the ZDAL input bus and the XDAL output bus to enable the DCJ11-A microprocessor to read and write data within the system. The XDAL bus has an address register to store the current address and is primarily driven by the DCJ11-A to access the cache memory and the YDAL bus. It also provides a bidirectional data path to the DC350/394 and DC351 gate arrays. The ZDAL bus routes data from the cache memory, the FPA and external data from the YDAL bus to the DCJ11-A. It also drives the YDAL bus with cache memory and FPA data. The YDAL bus is a bidirectional bus that interfaces with the BDAL bus and the ZDAL bus. It is also driven by the XDAL bus with the current address from the DCJ11-A, and accesses the internal IADR bus and IDAT bus via the DC351 gate array.

Access to and from the individual busses is controlled by a group of latching drivers that are primarily controlled by the DCJ11-A and the data path controller. A latch is opened and the data is latched by the input signal to the Latch Enable (LE) input, and the data is driven onto the latched bus by the input signal to the Output Enable (OE) input. For example, the COPN YTOZ H input latches the data from the YDAL bus when asserted, and the EN YTOZ L input drives the data onto the ZDAL bus when asserted.

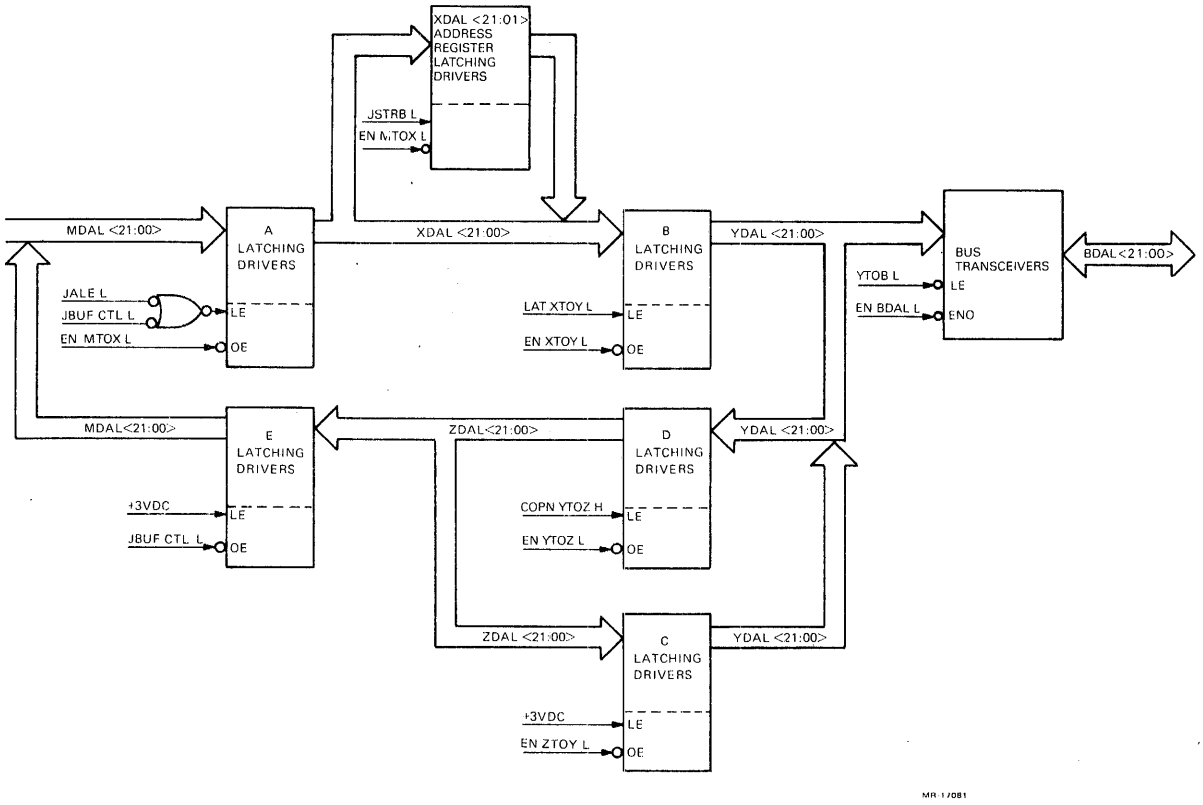


Figure 5-32 Internal Bus Control

5.9.2 LSI-11 Bus Control

The LSI-11 bus control signals establish the data communications path between the KDJ11-B module and the rest of the devices in the system, as shown in Figure 5-33. The addressing and data lines are driven by the bus transceivers that interface with the YDAL bus (Figure 5-32). The LSI-11 bus control signals are used by the handshaking protocol necessary to execute the bus transactions. The operation of the LSI-11 bus is described in greater detail in Chapter 6.

5.9.3 PMI Bus Control

The PMI bus control signals provide the interface between the KDJ11-B module and the PMI bus, as shown in Figure 5-34. The operation of the PMI bus is detailed in Chapter 7.

5.10 CONSOLE SERIAL LINE UNIT

The console SLU is a DC319 DLART that provides the KDJ11-B module with a serial line I/O for the system console terminal, as shown in Figure 5-35. The full-duplex unit interfaces via an RS-423 EIA connector and is also RS-232C compatible. The SLU has four internal registers designated as RCSR, RBUF, XCSR and XBUF. These registers transmit and receive serial line data via the console unit and format the data for internal parallel communications using the IDAT bus. The contents of these registers are described in Chapter 1.

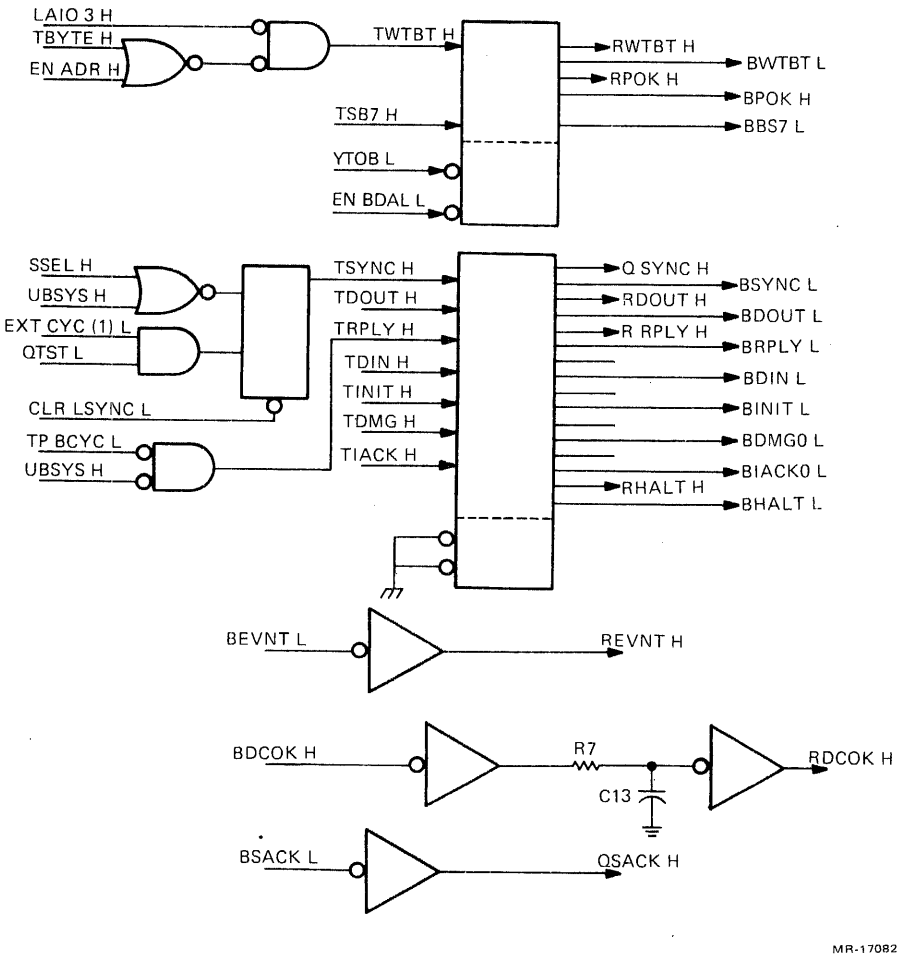


Figure 5-33 LSI-11 Bus Control Signals

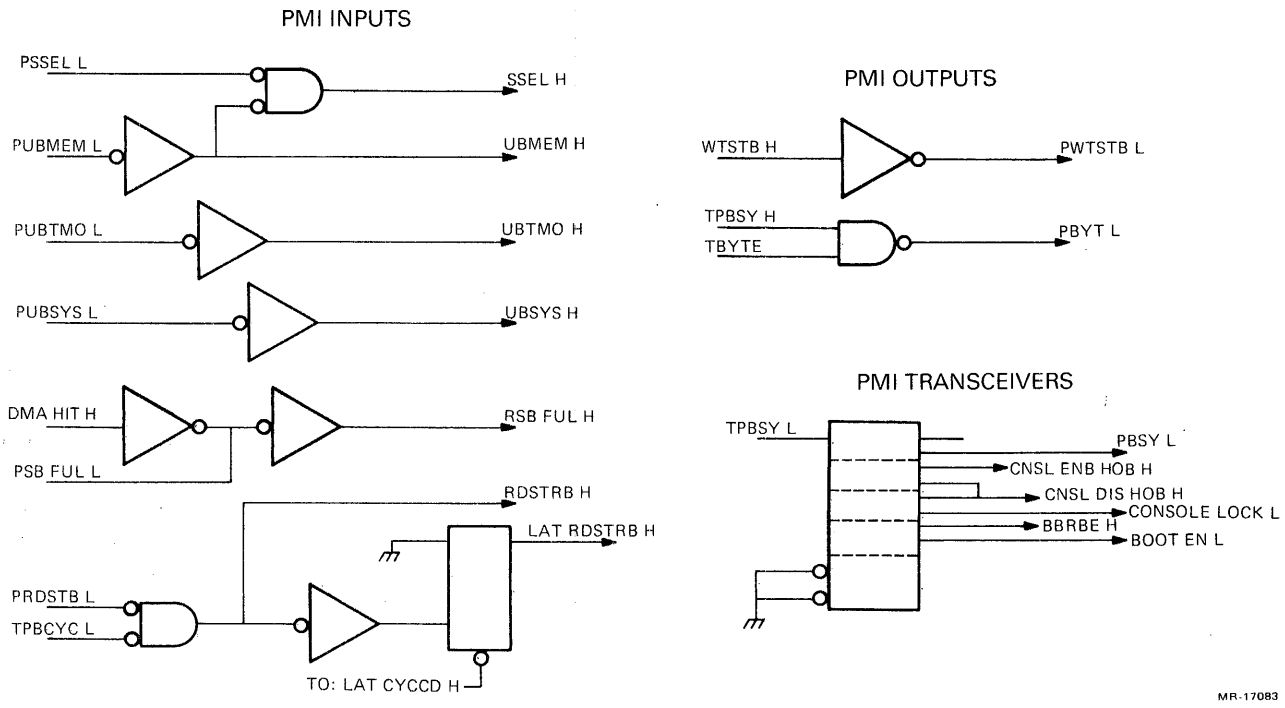


Figure 5-34 PMI Bus Control Signals

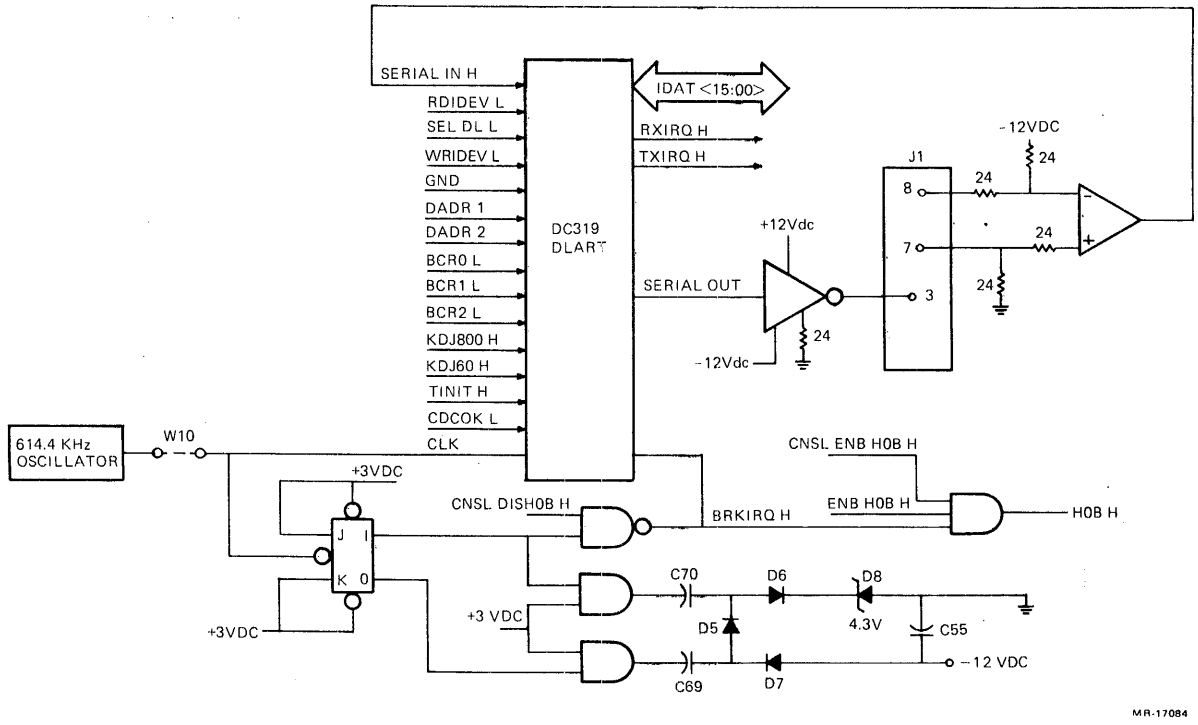


Figure 5-35 Console Serial Line Logic

The DLART is enabled by asserting the SELDL input and is initialized by asserting the TINIT input. The transmit interrupt request (TXIRQ) output is asserted when the internal XMIT RDY and XMIT IE bits of the XCSR are set. The receive interrupt request (RXIRQ) output is asserted when the internal RCV DONE and RCV IE bits of the RCSR are set. These signals go to the console interrupt arbitration logic and interrupt the DCJ11-A by enabling the CIRQ4 input. The transmitter interrupt vector is at location 064 and the receiver interrupt vector is at location 060.

There are two real-time clock interrupts – the KDJ800 input at 800 Hz and the KDJ60 input at 60 Hz. These inputs are controlled by bits 10 and 11 of the BCSR register.

The contents of the internal registers are read onto the IDAT bus when the RDIDEV input is asserted, and the data on the IDAT bus is written into the internal registers when the WRIDEV input is asserted. The internal registers are addressed (Table 5-19) by the DADR1 and DADR2 inputs, which represent bits 1 and 2 of the address on the DADR bus. Since there are no byte cycles, address bit 0 is grounded by pin 21 of the DLART.

The DLART transmits and receives data using a common baud rate that is determined by the status of the BCR0, BCR1 and BCR2 inputs. The input conditions to select a baud rate are shown in Table 5-20. The status of these inputs is selected by configuration switches 6, 7 and 8. Switch 8 selects the BCR0 input, switch 7 selects the BCR1 input, and switch 6 selects the BCR2 input. The baud rate frequencies are based on the clock input frequency of 614.4 kHz.

The serial input data is received via the SERIAL IN input and the serial output data is transmitted via the SERIAL OUT output. These signals are routed to and from J1 by the external drivers. The +12 Vdc reference is from the +15 Vdc input on the backplane and the −12 Vdc reference is from the circuit shown in Figure 5-35.

Table 5-19 Register Selection

Address	DADR2	DADR1	Register
17 777 560	0	0	Receiver status (RCSR)
17 777 562	0	1	Receiver Data Buffer (RBUF)
17 777 564	1	0	Transmitter Status (XCSR)
17 777 566	1	1	Transmitter Data Buffer (XBUF)

Table 5-20 Baud Rate Selections*

BCR2	BCR1	BCR0	Baud Rate
0	0	0	300
0	0	1	600
0	1	0	1,200
0	1	1	2,400
1	0	0	4,800
1	0	1	9,600
1	1	0	19,200
1	1	1	38,400

* 1 indicates the input is asserted low; 0 indicates the input is asserted high.

5.10.1 Halt-on-Break

The break detected interrupt request output (BRK IRQ H) is asserted when the RCV BRK bit of the RBUF register is set. This bit is set when the console transmits a break condition to the DLART. The halt-on-break function is enabled when bit 9 of the BCSR is set by allowing the ENB HOB H output from the DC350/394 gate array to be asserted while the CNSL ENB HOB H input from the PMI interface is also asserted. When these three inputs are asserted, the HOB H input to the DCJ11-A is asserted to enable the halt condition. When the CNSL DIS HOB signal from the PMI interface is asserted, the BRK IRQ H output is negated.

5.10.2 Console Interrupt Arbitration

The console interrupt arbitration logic (Figure 5-36) determines the sequence of the RXIRQ and TXIRQ interrupt requests. If either request is asserted, the XIRQ output is asserted to the DCJ11-A as a level 4 interrupt request on the CIRQ4 input, and the DCJ11-A initiates an Interrupt ACKnowledge transaction (IACK). The RXIRQ and TXIRQ inputs are latched by the flip-flops when the TDIN input is asserted and provide the RXVEC and TXVEC outputs to the next address MUX. When the IACK input is asserted, it is NANDed with RXVEC (if it is present) to negate the RXIRQ request. The TXVEC is inhibited from being reset while the RXVEC is asserted. If TXVEC is asserted and RXVEC is negated, the asserted IACK input negates the TXIRQ request. The asserted IACK input enables the TIACK output, provided that neither the RXVEC nor the TXVEC outputs are asserted.

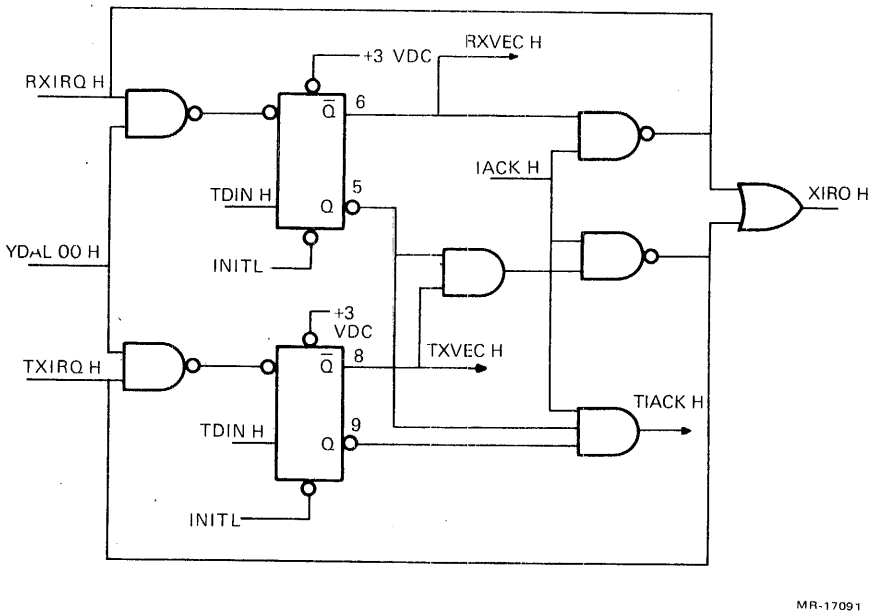


Figure 5-36 Console Interrupt Arbitration

5.11 CONFIGURATION AND DISPLAY

The configuration and display circuits consist of a switchpack having eight switches, six red LEDs and one green LED. The green LED monitors the +5 Vdc module supply voltage. The switches and red LEDs can be remotely operated via the J2 and J3 connectors. The circuits are shown in Figure 5-37.

The switches select the EEPROM bootstrap programs, enable the dialog mode, select the console baud rate and control the system console operation. The switch functions are described in Chapter 2. The switchpack data is driven onto the IDAT bus by the buffer/drivers when the RDBCR input is asserted and it goes to the CDR as bits <7:0>.

The red LEDs are encoded so that LEDs 0 through 2 and 3 through 5 are a binary representation of a two-digit octal number display for the diagnostic tests and error messages described in Chapter 4. The IDAT bus is used to drive the LED display using bits <15:8> of the CDR. The flip-flops are cleared by the assertion of RDCOK, and the IDAT bus data is latched to drive the display by the assertion of the WR DISP input.

5.12 BOOT AND DIAGNOSTIC ROMS

The boot and diagnostic ROMs contain the ROM code to support the boot and diagnostic programs discussed in Chapter 4. The ROMs are addressed by using IADR bus bits <14:9> from the DC350/394 and DADR bus bits <8:1> from the DC351. The IADR address bits are selected by the DC350/394 from the PCR and depend on the address to be either 17 773 000 or 17 765 000. The ROMs are enabled by the assertion of the SEL ROM16 input. The address decoder in the DC350/394 asserts this input when a location in the ROM is addressed. The data is driven onto the IDAT bus by the DC350/394 assertion of the RDIDEV input. The boot and diagnostic ROM logic is shown in Figure 5-38.

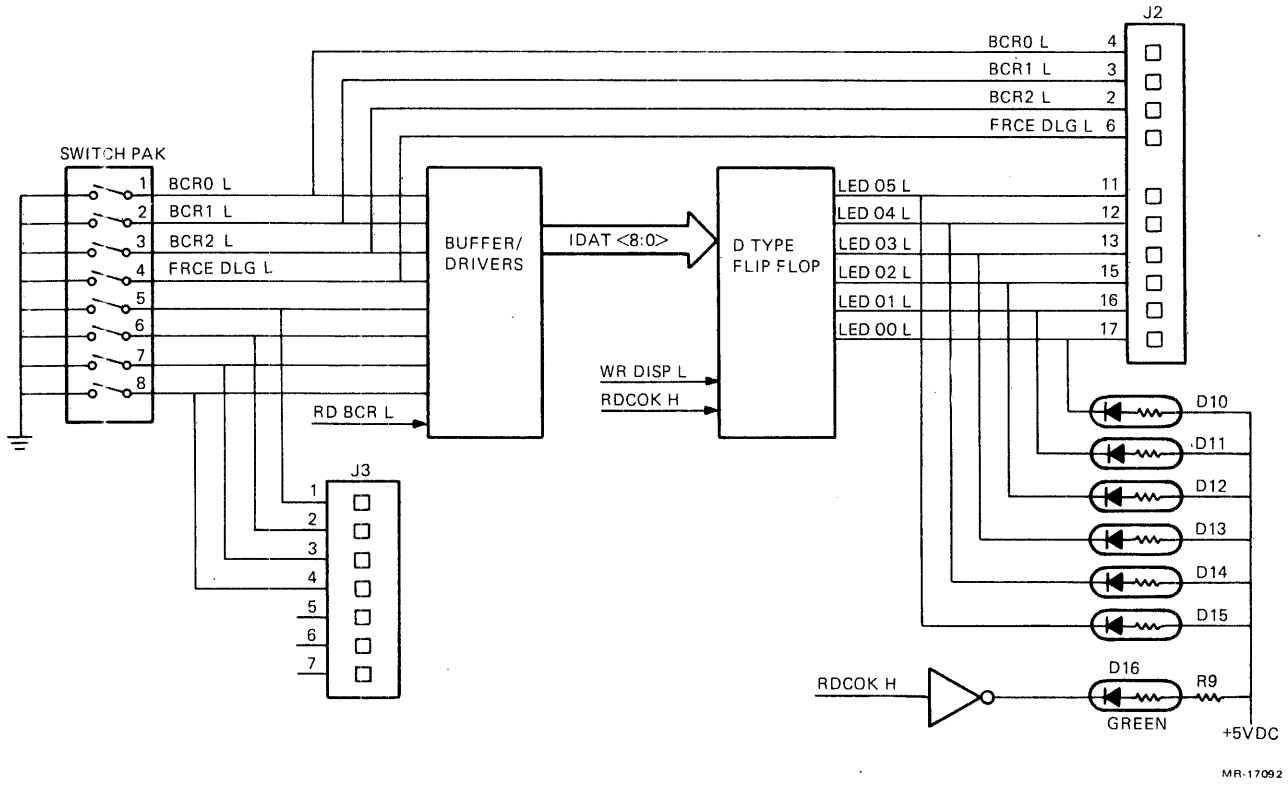


Figure 5-37 Configuration and Display Circuits

5.13 CONFIGURATION EEPROM

The configuration EEPROM provided with the module is a 2K × 8 EEPROM that is offset in a 28-pin socket. In this mode of operation, the W40 jumper connects the TP40 and TP41 pins to provide the WR IDEV input to the EEPROM. The user can optionally use a 4K or 8K EEPROM and use the W40 jumper to connect the TP41 and TP42 pins. This connects IADR bus bit 12 to the socket. IADR bus bit 13 and the WR IDEV input are connected to the pins not used by the 2K EEPROM. This condition provides the two additional address bits required for the expanded EEPROM.

The EEPROM is addressed by using IADR bus bits <13:9> from the DC350/394 and DADR bus bits <8:1> from the DC351. The IADR address bits are selected by the DC350/394 from the PCR and depend on the address to be either 17 773 000 or 17 765 000. The EEPROM is enabled by the assertion of the SEL ROM8 input. The address decoder in the DC350/394 asserts this input when a location in the EEPROM is addressed. The data is driven onto the IDAT bus by the DC350/394 assertion of the RDIDEV input. Data is written into the EEPROM when the DC350/394 asserts the WR IDEV input. The configuration EEPROM logic is shown in Figure 5-39.

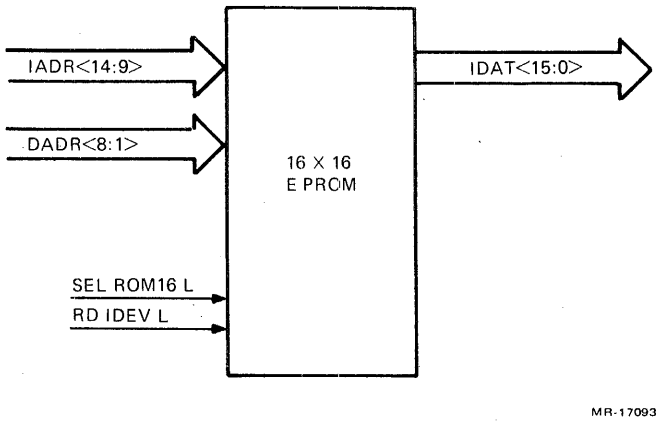


Figure 5-38 Boot and Diagnostic ROM Logic

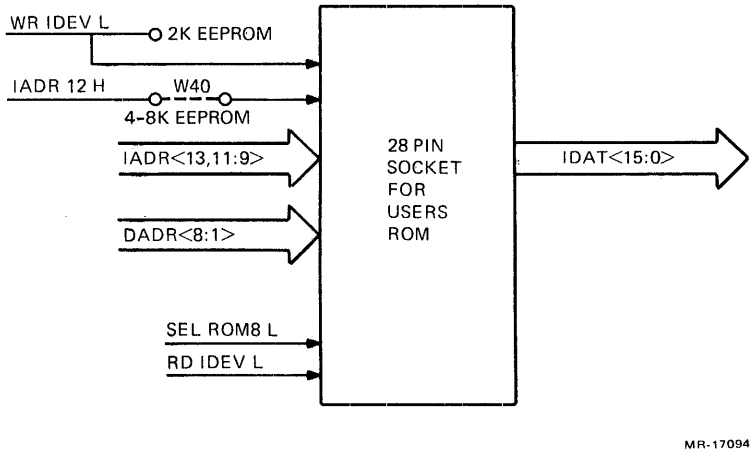


Figure 5-39 Configuration EEPROM Logic

5.14 FLOATING-POINT ACCELERATOR

The FPA is an optional 40-pin chip that can be mounted on the module. The FPA is a floating-point coprocessor that improves system performance (3 to 5 times greater speed) in floating-point applications. The FPA chip informs the DCJ11-A microprocessor of its presence on the module by asserting the FPA OP output during the power-up routine. The microprocessor then offloads all the floating-point functions to the FPA chip. The operation of the FPA is transparent to the system, except for the increased speed. The FPA socket and control signals are shown in Figure 5-40.

5.14.1 FPA Operation

When the DCJ11-A starts to decode an instruction, it asserts the MPRDC input and that is sampled by the FPA when the JSTRB input is asserted. The FPA monitors the JAIO <3:0> inputs that are encoded to represent the current microprocessor I/O cycle (see Table 5-1). The XDAL <1:0> inputs are the two least significant bits of the current address and are decoded by the FPA to determine the type of cycle (Table 5-4). The JAIO <3:0> and XDAL <1:0> input data is latched by the assertion of the JALE H input. The FPA loads the instruction stream data into a buffer and the DCJ11-A executes the bus cycles necessary to obtain the operand data. The instructions and data are transmitted via the ZDAL bus. When the data is valid on the ZDAL bus, the MDV input is asserted and the data is latched by the FPA. The FPA proceeds to execute the floating-point instructions stored in its buffer.

When the results are ready in the FPA, it asserts the FPA RDY output and moves the data into an output buffer. The control store acknowledges this condition by asserting the T50:EN FTOZ input that allows the output buffer to move the data onto the ZDAL bus. The FPA asserts the FPA STL output to stall the DCJ11-A until the data is gated onto the ZDAL bus. If a floating-point error or exception occurs during the processing, the FPA asserts the FPA FPE output to the DCJ11-A and cancels the output cycle. This condition is acknowledged by the control store assertion of MCONT to the DCJ11-A and a general purpose read cycle is executed to clear the exception.

The FPA chip is initialized and the FPS is cleared by asserting the CDCOK input. The initialization condition is cleared by the negation of the CDCOK input and the negation of the JSCTL input. The JABORT input is monitored by the FPA while the JSCTL input is asserted during a stretched cycle. If the JABORT input is asserted while JSCTL is asserted, the FPA does not complete the current I/O cycle. This FPA functions on KDJ11-BB or KDJ11-BF modules.

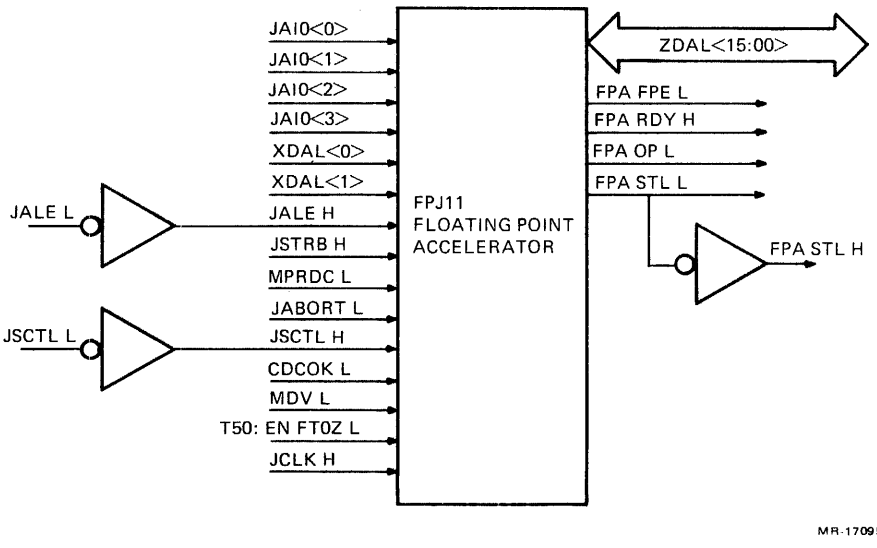


Figure 5-40 Floating-Point Accelerator

CHAPTER 6
EXTENDED LSI-11 BUS

6.1 INTRODUCTION

The processor, memory and I/O devices communicate via signal lines that constitute the extended LSI-11 bus. The extended LSI-11 bus contains 4 extra address lines (BDAL <21:18>) in addition to the 38 original LSI-11 bus lines. The four additional address lines extend the 256-Kbyte physical address space of the LSI-11 bus to 4 Mbytes. Addresses, 8-bit bytes or 16-bit data words, bus synchronization, and control signals are sent along these 42 lines. Addresses may be 16-, 18-, or 22-bits wide, depending on the addressing capability of the processor installed in the system. The 16-bit data and the first 16 address bits are time-multiplexed over the same 16 data/address lines. Two additional address bits (<17:16>) and the memory parity bits are also time-multiplexed over 2 signal lines. The signal lines are functionally divided as listed in Table 6-1. Refer to Chapter 2 for a list of the extended LSI-11 bus signals.

The LSI-11 bus lines are treated as transmission lines that are terminated in their characteristic impedance (Z0) at both the near and far ends of the bus. The near end of the bus is defined as the first bus interface slot in the backplane; the far end is the last bus interface slot.

Table 6-1 Summary of Signal Line Functions

Quantity	Function	Bus Signal Mnemonic
16	Data/address lines	BDAL <15:0>
2	Memory parity/address lines	BDAL <17:16>
4	Address lines	BDAL <21:18>
6	Address and data transfer control lines	BSYNC, BDIN, BDOUT, BWTBT, BBS7, BRPLY
3	DMA control lines	BDMR, BDMG, BSACK
5	Interrupt control lines	BIRQ4, BIRQ5, BIRQ6, BIRQ7, BIAK
6	System control lines	BPOK, BDCOK, BINIT, BHALT, BREF, BEVNT

Most LSI-11 bus signals are bidirectional and use a terminating resistor network connected between +5 V and ground to provide a negated (high) signal level. Devices may be connected to any point along the bus to receive signals from the near or far end of the bus via high-impedance bus receivers, or to transmit signals to the near or far end through gated open-collector bus drivers. A bus driver asserts a signal by causing the line to go from a high level (approximately 3.4 V) to a low level (approximately 0.5 V). The electrically bidirectional lines sometimes carry signals that are functionally unidirectional. These functionally unidirectional lines carry signals that are required to travel in only one direction. For example, when a device asserts a bus request signal (BIRQ), the signal always travels from the requesting device to the processor and never in the reverse direction.

The interrupt acknowledge (BIAK) and DMA grant (BDMG) signals are physically unidirectional signals that are wired to each LSI-11 bus slot in a daisy-chain scheme. These signals are generated by the processor in response to interrupt and DMA requests and are transmitted to the bus via output signal pins. Each of the output signals (BIAKO or BDMGO) is received on a device input pin (BIAKI or BDMGI) and is conditionally retransmitted via a device output pin (BIAKO or BDMGO). These signals are received from higher-priority devices and are retransmitted to lower-priority devices on the bus.

Bus master/slave relationship communication between devices on the bus is asynchronous. A master/slave relationship exists throughout each bus transaction. At any time, there is one device that has control of the bus. This controlling device is termed the bus master. The master device controls the bus when communicating with another device on the bus, termed the slave. The bus master (typically the KDJ11-B processor or a DMA device) initiates a bus transaction. The slave device responds by acknowledging the transaction in progress and by receiving data from, or transmitting data to, the bus master. The extended LSI-11 bus control signals transmitted or received by the bus master or bus slave device must complete the sequence according to the protocol established for transferring address and data information. The processor controls bus arbitration (i.e., it "decides" which device is to be bus master at any given time).

A typical example of a master/slave relationship is the processor, as master, fetching an instruction from memory, which is always a slave. Another example is a disk drive, as master, transferring data to memory, again, as the slave. Any device except the processor can be master or slave depending on the circumstances. Communication on the extended LSI-11 bus is interlocked; for each control signal issued by the master device, there must be a response from the slave in order to complete the transfer. It is the master/slave signal protocol that makes the extended LSI-11 bus asynchronous. The asynchronous operation allows both fast and slow devices to use the bus and eliminates the need for synchronizing clock pulses between the bus master and slave device.

Since bus cycle completion by the bus master requires response from the slave device, each bus master must include a timeout error circuit that aborts the bus cycle if the slave device does not respond to the bus transaction within 10 μs. The KDJ11-B has a bus timer that restarts the clock when no device responds to BDIN L or BDOUT L within 10 μs. An immediate trap to location 48 occurs. The slowest peripheral or memory device must respond in less than 10 μs to prevent a bus timeout error.

6.2 BUS SIGNAL NOMENCLATURE

Throughout the following protocol specifications, bus signals are referred to in several different ways.

1. In general discussions where timing, polarity, and physical location are unimportant, the base signal name without any prefixes or suffixes is used. For example:

   SYNC, WTBT, BS7, DAL <21:0> or the DAL lines

2. Most signals on the backplane etch are asserted low and are referred to with a prefix character B, and a suffix (space) L. For example:

   BSYNC L, BWTBT L, BBS7 L, BDAL <21:0> L

   BPOK H and BDCOK H are asserted high.

3. Receivers and drivers are considered to be part of the bus. Signal inputs to drivers are referred to with a prefix character T, for transmit. For example:

   TSYNC, TWTBT, TBS7, TDAL <21:0>

4. Signal outputs of receivers are referred to with the prefix character R, for receive. For example:

   RSYNC, RWTBT, RBS7, RDAL <21:0>

Whenever timing is important, the designations in items 3 and 4 above are used to reference timing to a receiver output or driver input. For example, after receipt of the negation of RDIN, the slave negates its TRPLY (0 ns minimum, 8000 ns maximum). It must maintain data valid on its TDAL lines until 0 ns (minimum) after the negation of RDIN, and must negate its TDAL lines 100 ns (maximum) after the negation of its TRPLY.

6.3 DATA TRANSFER BUS CYCLES

Data is transferred between a bus master and slave device to accomplish various functions. The data transfer bus cycles and their functions are described in Table 6-2.

These bus cycles, executed by bus master devices, transfer 16-bit words or 8-bit bytes to or from slave devices. The data to be written in the destination byte during byte output operations is valid on the appropriate BDAL lines. For example, BDAL <15:8> contains the high byte, and BDAL <7:0> contains the low byte. Table 6-3 describes the bus signals used in a data transfer operation.

Data transfer bus cycles can be reduced to three basic types: DATI, DATO(B) and DATIO(B). These transactions occur between the bus master and one slave device selected during the addressing portion of the bus cycle.

Table 6-2 Data Transfer Bus Cycles

Bus Cycle Mnemonic	Description	Function (with respect to the bus master)
DATI	Data word input	Read
DATO	Data word output	Write
DATOB	Data byte output	Write byte
DATIO	Data word input/output	Read-modify-write
DATIOB	Data word input/byte output	Read-modify-write byte

Table 6-3 Data Transfer Bus Signals

Mnemonic	Description	Function
BDAL <21:0> L	22 data/address lines	BDAL <21:18> L are used for 22-bit extended addressing; BDAL <17:16> L are used for 18-bit extended addressing, memory parity error, and memory parity error enable functions; BDAL <15:0> L are used for 16-bit addressing, word and byte transfers.
BSYNC L	Synchronize	Strobe signals
BDIN L	Data input strobe	
BDOUT L	Data output strobe	
BRPLY L	Reply	
BWTBT L	Write/byte control	Control signals
BBS7 L	Bank 7 select	

6.3.1 Bus Cycle Protocol

Before initiating a bus cycle, the previous bus transaction must be complete (BSYNC L negated) and the device must become bus master. The bus cycle is divided into two parts – an addressing portion, and a data transfer portion. During the addressing portion, the bus master outputs the address for the desired slave device (memory location or device register). The selected slave device responds by latching the address bits and holding this condition for the duration of the bus cycle (until BSYNC L becomes negated). During the data transfer portion of the bus cycle, the operations performed vary slightly, depending on the type of data transfer desired.

6.3.1.1 Device Addressing – The device addressing portion of a data transfer bus cycle comprises an address setup/deskew time and an address hold/deskew time. During the address setup/deskew time, the bus master does the following.

1. It asserts TDAL <21:0> with the desired slave device address bits.
2. It asserts TBS7 if a device in the I/O page is being addressed.
3. It asserts TWTBT if the cycle is a DATO(B) bus cycle.
4. It asserts TSYNC 150 ns (minimum) after gating TDAL, TBS7, and TWTBT onto the bus.

During this time the address, RBS7, and RWTBT signals are asserted at the slave bus receiver for at least 75 ns before RSYNC becomes active. Devices in the I/O page ignore the 9 high-order address bits RDAL <21:13> and, instead, decode RBS7 along with the 13 low-order address bits. An active RWTBT signal indicates that a DATO(B) operation follows, while an inactive RWTBT indicates a DATI or DATIO(B) operation.

The address hold/deskew time begins after RSYNC is asserted. The slave device uses the active RSYNC to clock RDAL address bits, RBS7 and RWTBT, into its internal logic. RDAL <21:0>, RBS7, and RWTBT remain active for 25 ns (minimum) after RSYNC becomes active. RSYNC remains active for the duration of the bus cycle.

Memory and peripheral devices are addressed similarly, except for the way they respond to RBS7. Addressed peripheral devices must not decode address bits on RDAL <17:13>. Addressed peripheral devices may respond to a bus cycle only when RBS7 is asserted during the addressing portion of the cycle. When asserted, RBS7 indicates that the device address resides in the I/O page (the upper 8-Kbyte address space). Memory devices generally do not respond to addresses in the I/O page. However, some system applications may permit memory to reside in the I/O page for use as DMA buffers, ROM bootstraps, diagnostics, etc.

6.3.1.2 DATI – The DATI bus cycle is a read operation that inputs data from the slave device to the bus master. The operations performed by the bus master and slave device during a DATI are shown in Figure 6-1. The DATI bus cycle timing is shown in Figure 6-2. Data consists of 16-bit word transfers over the bus. During the data transfer portion of the DATI bus cycle, the bus master asserts TDIN 100 ns (minimum) after it asserts TSYNC. The slave device responds to RDIN active by asserting:

1. TRPLY after receiving RDIN, and 125 ns (maximum) before TDAL bus driver data bits are valid,

2. TDAL <17:0> L with the addressed data and error information.

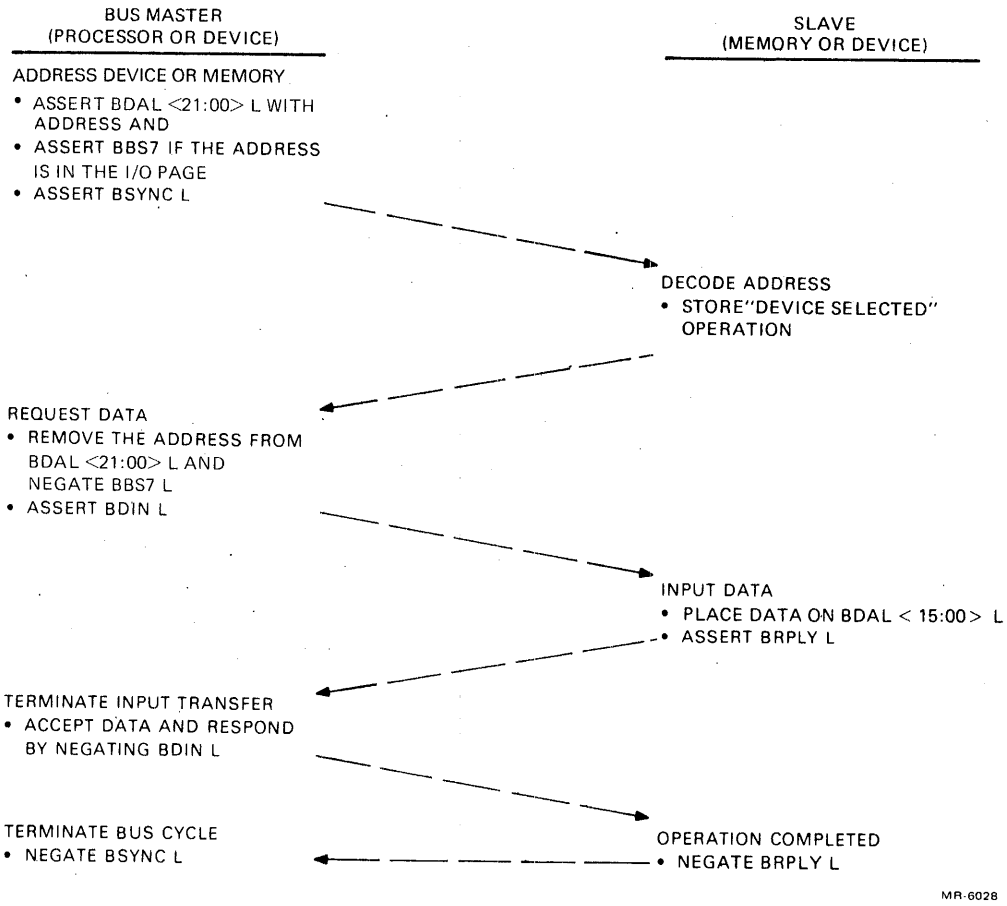
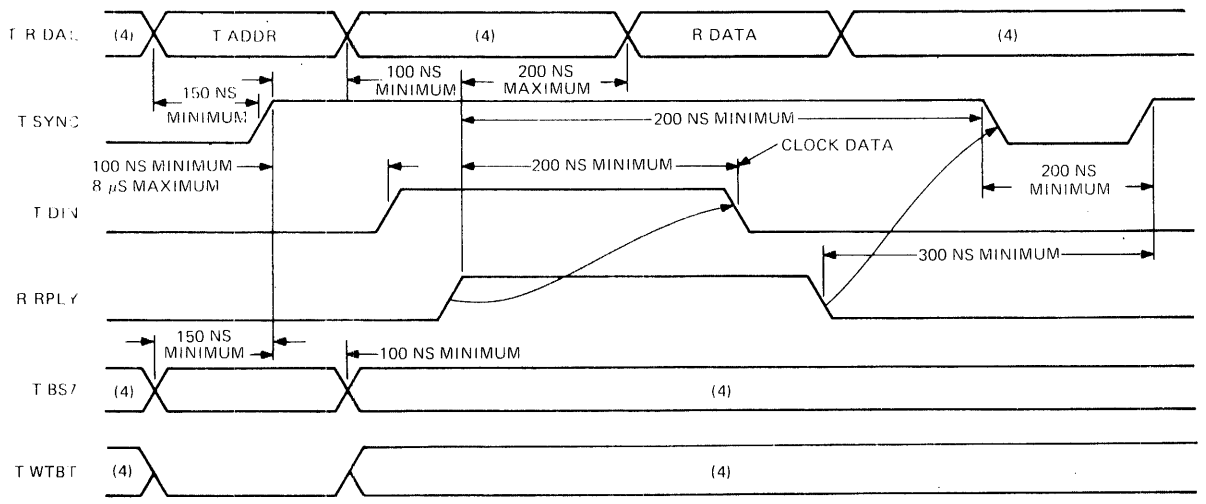
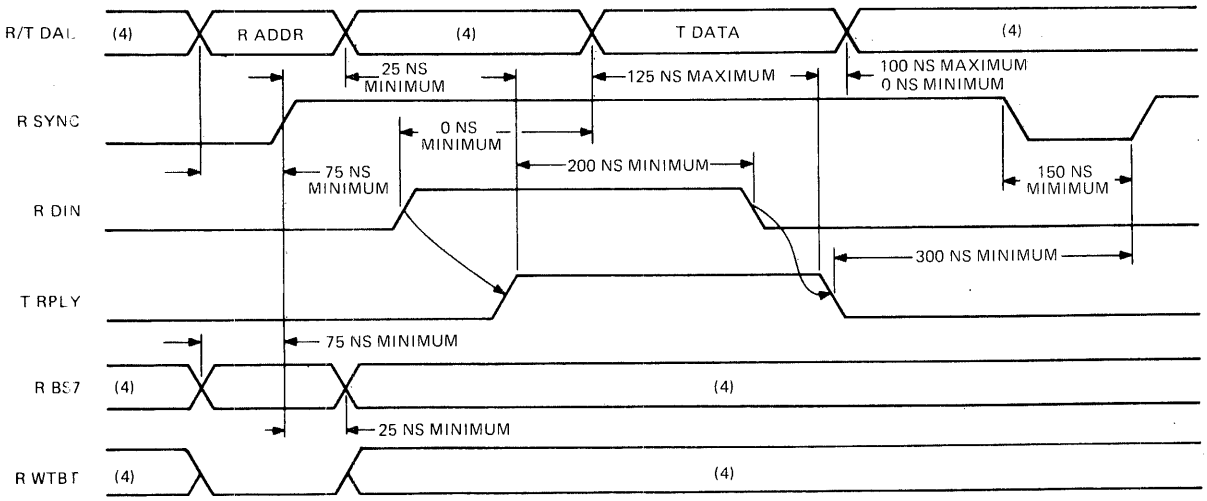


Figure 6-1 DATI Bus Cycle



TIMING AT MASTER DEVICE



TIMING AT SLAVE DEVICE

NOTES:

1. TIMING SHOWN AT MASTER AND SLAVE DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR 6037

Figure 6-2 DATI Bus Cycle Timing

When the bus master receives RRPLY, it does the following.

1. It waits at least 200 ns deskew time and then accepts input data at RDAL <15:0> bus receivers. RDAL <17:16> are monitored for a possible parity error indication.

2. It negates TDIN 150 ns (minimum) after RRPLY becomes active.

The slave device responds to RDIN negation by negating TRPLY and removing read data from TDAL bus drivers. TRPLY must be negated 100 ns (maximum) prior to removal of read data. The bus master responds to the negated RRPLY by negating TSYNC.

Conditions for the next TSYNC assertion are as follows.

1. TSYNC must remain negated for 200 ns (minimum).

2. TSYNC must not become asserted within 300 ns of the previous RRPLY negation.

6.3.1.3 DATO(B) – DATO(B) is a write operation. Data is transferred in 16-bit words (DATO) or 8-bit bytes (DATOB) from the bus master to the slave device. The data transfer output can occur after the addressing portion of a bus cycle when TWTBT has been asserted by the bus master, or immediately following an input transfer part of a DATIO(B) bus cycle. The operations performed by the bus master and slave device during a DATO(B) bus cycle are shown in Figure 6-3. The DATO(B) bus cycle timing is shown in Figure 6-4.

The data transfer portion of a DATO(B) bus cycle comprises a data setup/deskew time and a data hold/deskew time. During the data setup/deskew time, the bus master outputs the data on TDAL <15:0> 100 ns (minimum) after TSYNC is asserted. If it is a word transfer, the bus master negates TWTBT while gating data onto the bus. If the transfer is a byte transfer, the bus master asserts TWTBT while gating data onto the bus. During a byte transfer, the condition of BDAL 00 L during the address cycle selects the high or low byte. If asserted, the high byte (BDAL <15:8> L) is selected. Otherwise, the low byte (BDAL <7:0> L) is selected. An asserted BDAL 16 L at data transfer time forces a parity error to be written into memory (if the memory is parity memory). BDAL 17 L is not used for write operations. The bus master asserts TDOUT L 100 ns (minimum) after the TDAL and TWTBT bus driver inputs are stable. The slave device responds to RDOUT by accepting the input data and asserting TRPLY (8 μs maximum to avoid bus timeout). This completes the data setup/deskew time.

During the data hold/deskew time the bus master negates TDOUT 150 ns (minimum) after the assertion of RRPLY. TDAL <21:0> bus drivers remain stable for at least 100 ns after TDOUT negation. The bus master then negates TDAL inputs. The slave device senses RDOUT negation and negates TRPLY. The bus master responds by negating TSYNC. The processor, however, does not negate TSYNC for at least 175 ns after negating TDOUT. This completes the DATO(B) bus cycle. Before the next cycle, TSYNC must remain unasserted for at least 200 ns. Also, TSYNC may not be asserted until 300 ns (minimum) after RRPLY is negated.

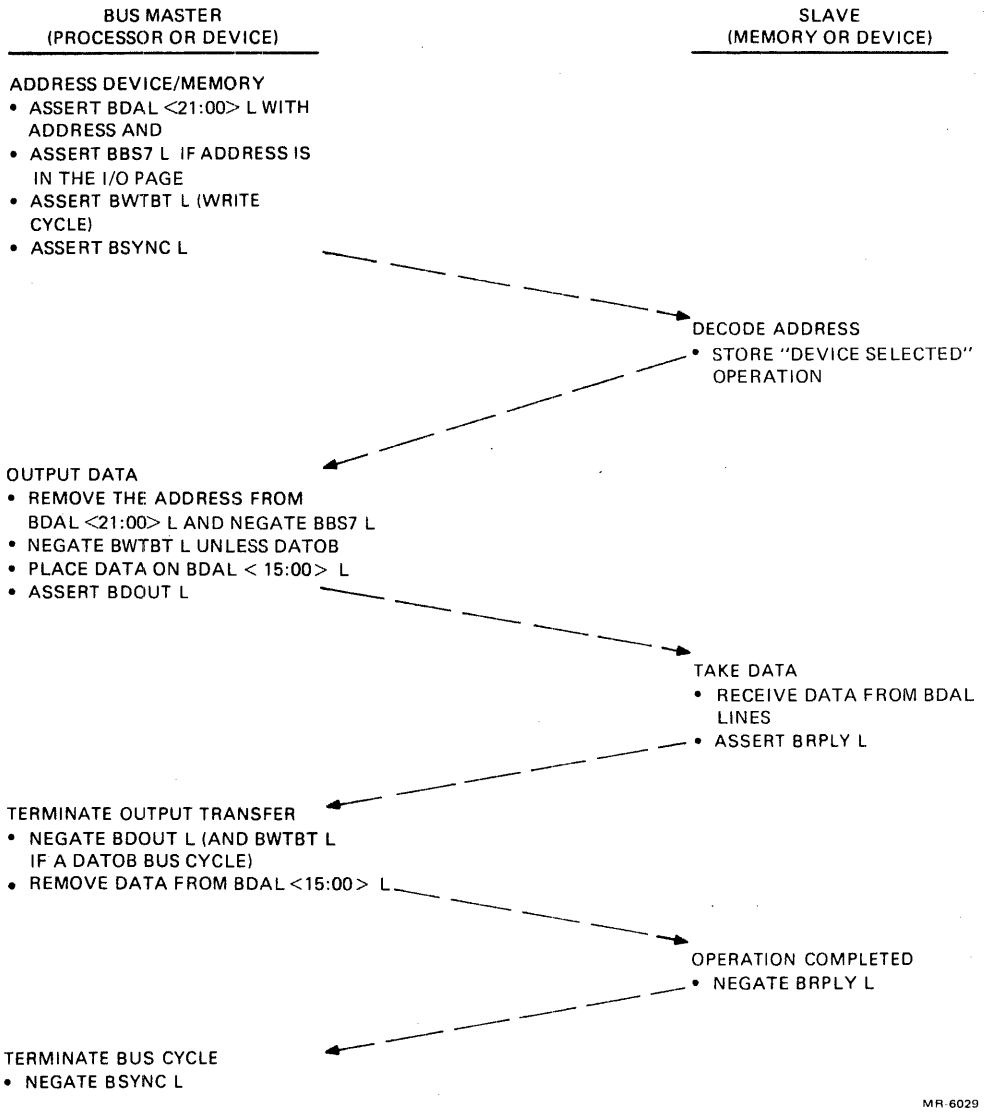
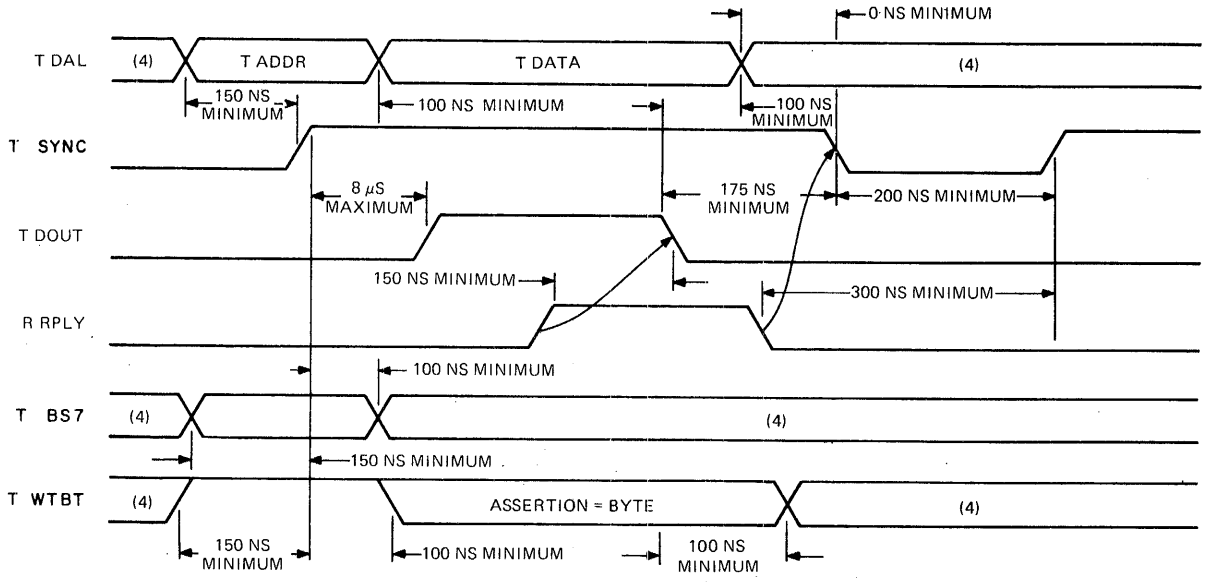
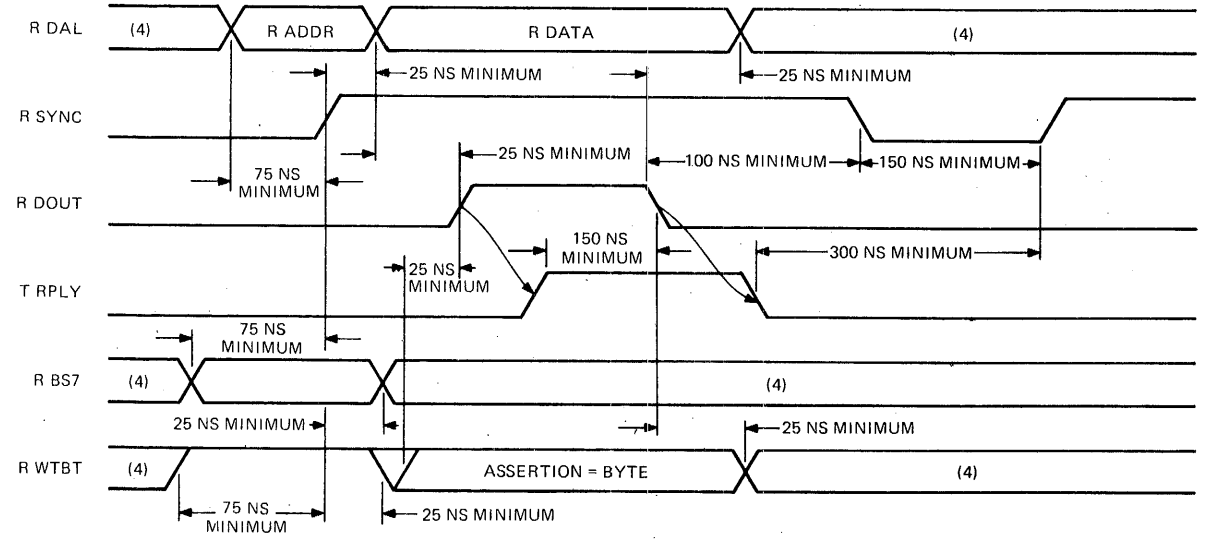


Figure 6-3 DATO or DATO(B) Bus Cycle



TIMING AT MASTER DEVICE



TIMING AT SLAVE DEVICE

NOTES:

1. TIMING SHOWN AT MASTER AND SLAVE DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS.
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR-1179

Figure 6-4 DATO or DATO(B) Bus Cycle Timing

6.3.1.4 DATIO(B) – The protocol for a DATIO(B) bus cycle is identical to the addressing and data transfer portions of the DATI and DATO(B) bus cycles. After addressing the device, a DATI cycle is performed as explained in Paragraph 6.3.1.2, except TSYNC is not negated. TSYNC remains active for an output word or byte transfer [DATO(B)]. The bus master maintains at least 200 ns between RRPLY negation during the DATI cycle and TDOUT assertion. The cycle is terminated when the bus master negates TSYNC, which follows the same protocol as described for DATO(B). The operations performed by the bus master and slave device during a DATIO or DATIO(B) bus cycle are shown in Figure 6-5. The DATIO and DATIO(B) bus cycle timing is shown in Figure 6-6.

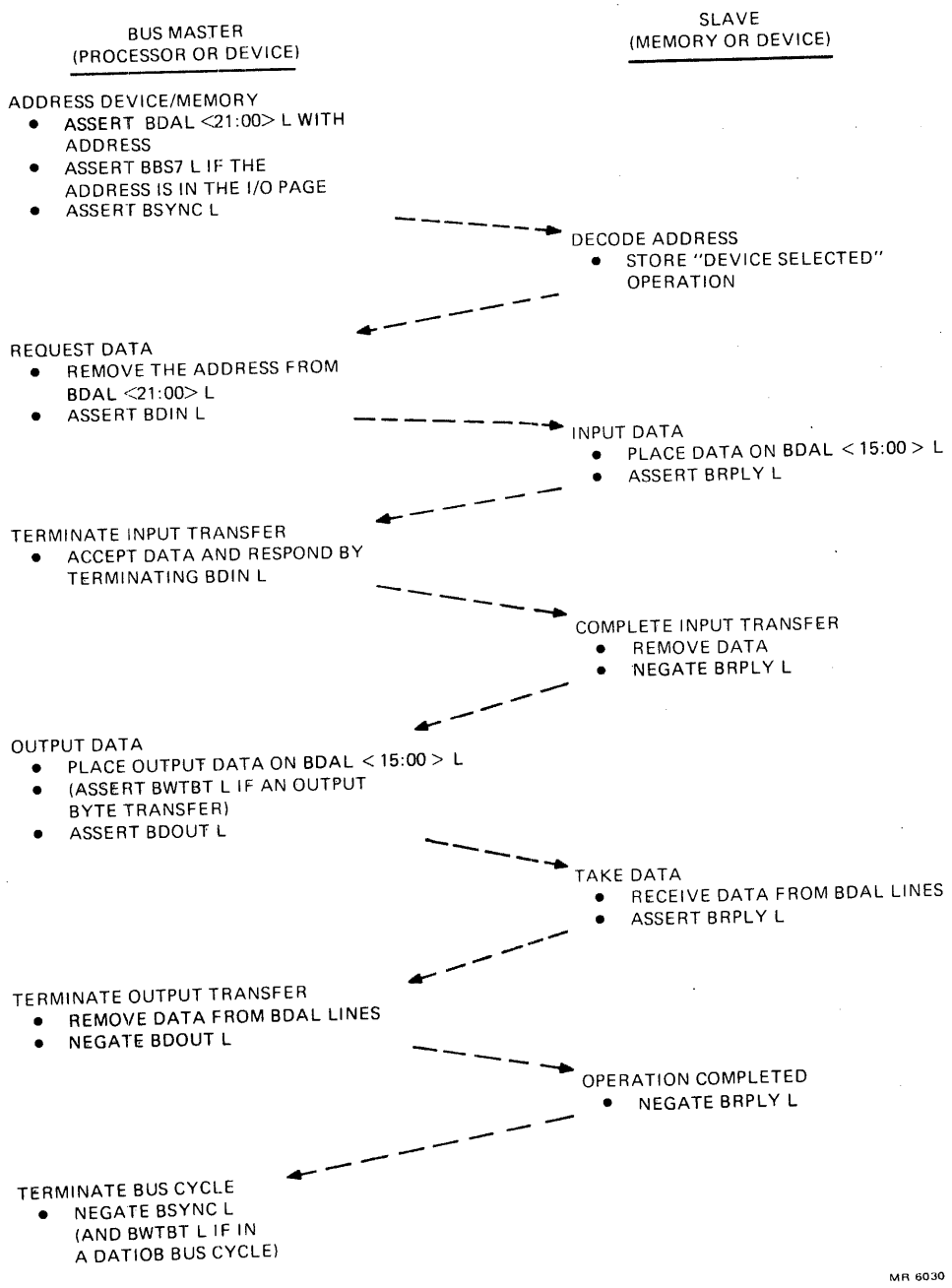
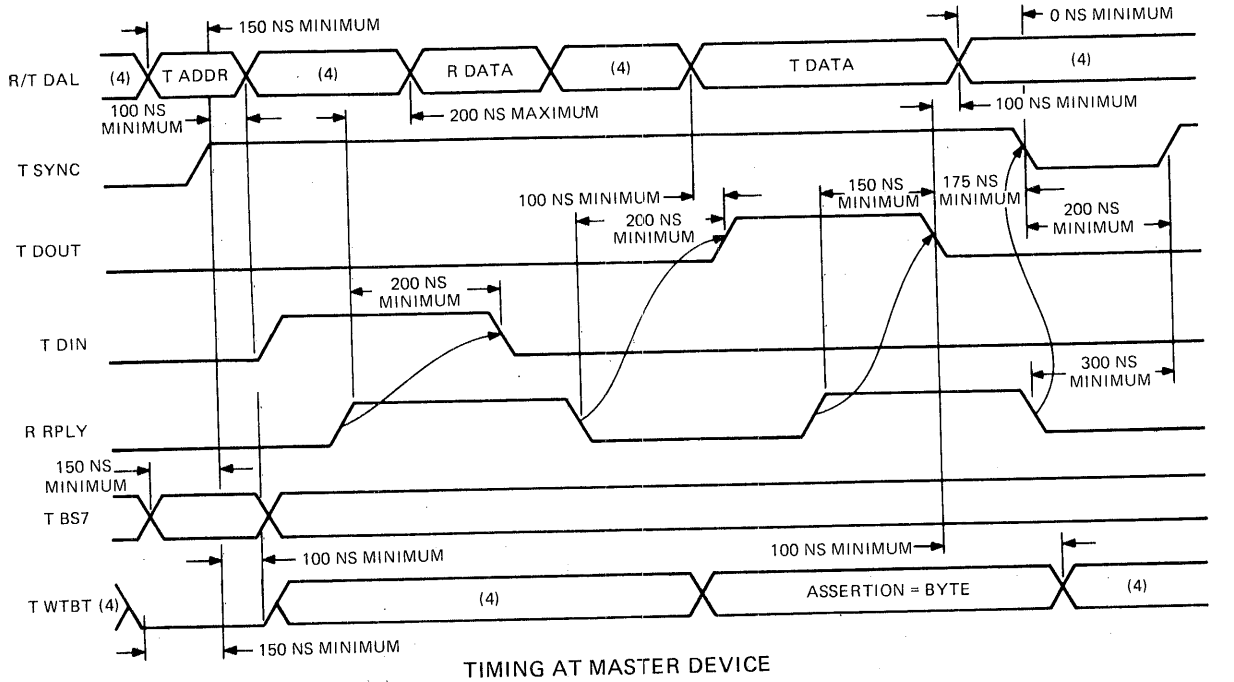
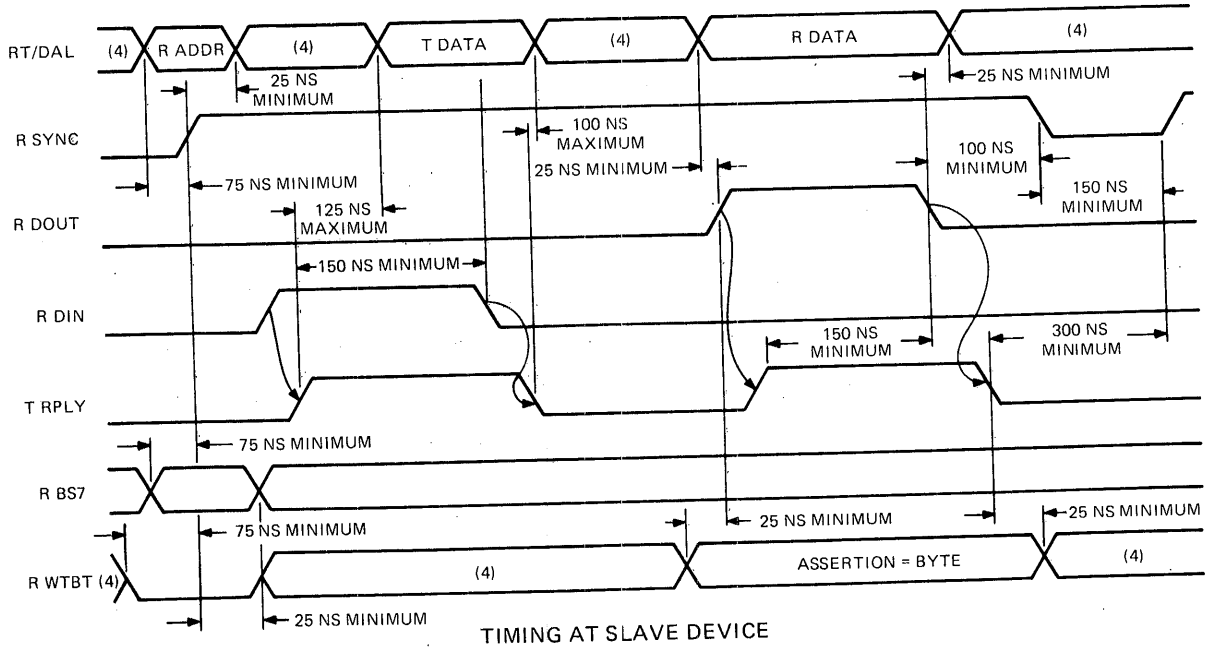


Figure 6-5 DATIO or DATIO(B) Bus Cycle



TIMING AT MASTER DEVICE



TIMING AT SLAVE DEVICE

NOTES:

1. TIMING SHOWN AT REQUESTING DEVICE BUS DRIVER INPUTS AND BUS RECEIVER OUTPUTS
2. SIGNAL NAME PREFIXES ARE DEFINED BELOW:
   T = BUS DRIVER INPUT
   R = BUS RECEIVER OUTPUT
3. BUS DRIVER OUTPUT AND BUS RECEIVER INPUT SIGNAL NAMES INCLUDE A "B" PREFIX.
4. DON'T CARE CONDITION.

MR 6036

Figure 6-6 DATIO or DATIO(B) Bus Cycle Timing

6.4 DIRECT MEMORY ACCESS

DMA capability allows direct data transfers between I/O devices and memory. This is useful when using mass storage devices (e.g., disk drives) that move large blocks of data to and from memory. A DMA device only needs to know the starting address in memory, the starting address in mass storage, the length of the transfer, and whether the operation is read or write. When this information is available, the DMA device can transfer data directly to or from memory. Since most DMA devices must perform data transfers in rapid succession or lose data, DMA requests are assigned the highest priority level.

DMA is accomplished after the processor (normally bus master) has passed bus mastership to the highest-priority DMA device that is requesting the bus. The processor arbitrates all requests and grants the bus to the DMA device located closest (electrically) to the processor. A DMA device remains bus master until it relinquishes its mastership. The following control signals are used during bus arbitration.

Signal	Name
BDMGI L	DMA grant input
BDMGO L	DMA grant output
BDMR L	DMA request line
BSACK L	Bus grant acknowledge

A DMA transaction is divided into three phases: the bus mastership acquisition phase, the data transfer phase, and the bus mastership relinquish phase. The operations performed by the processor and bus master during the DMA request/grant sequence are shown in Figure 6-7. The DMA request/grant bus cycle timing is shown in Figure 6-8.

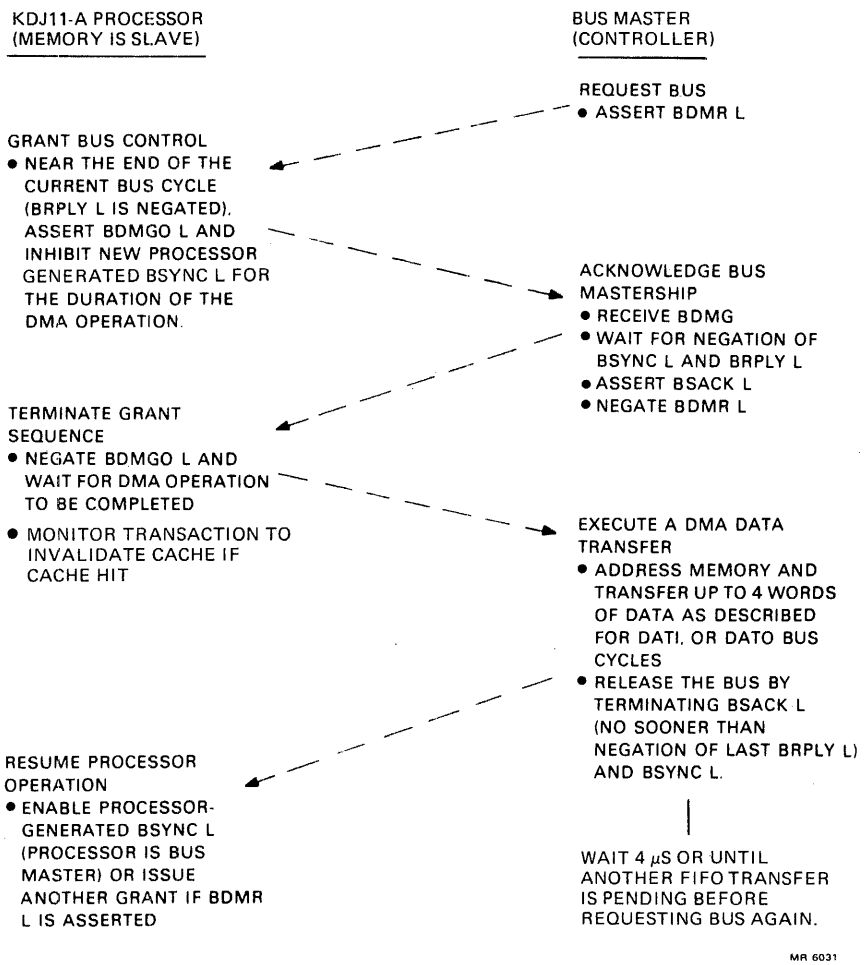


Figure 6-7 DMA Request/Grant Sequence

During the bus mastership acquisition phase, a DMA device requests the bus by asserting TDMR. The processor arbitrates the request and initiates the transfer of bus mastership by asserting TDMG. The maximum time between BDMR L assertion by the DMA device and BDMGO L assertion by the processor is DMA latency. This time is processor-dependent. The KDJ11-B asserts TDMG 1.4 μs (maximum) after the assertion of RDMR.

BDMGO L/BDMGI L is one of two signals that are daisy-chained through each module in the backplane. The signal is driven out of the processor on the BDMGO L pin, enters each module on the BDMGI L pin and exits on the BDMGO L pin. This signal passes through the modules in descending order of priority until it is stopped by the requesting device. The requesting device blocks the output of BDMGO L and asserts TSACK. If no device responds to the DMA grant, the processor clears the grant and rearbitrates the bus.

NOTE

The KDJ11-B uses a no SACK timer that clears BDMGO L if BSACK L is not received from the DMA device within 10 μs.

During the data transfer phase, the DMA device continues asserting BSACK L. If multiple data transfers are performed during this phase, consideration must be given to the use of the bus for other system functions, such as memory refresh (if required). The actual data transfer is performed in the same manner as the data transfer portion of DATI, DATO(B) and DATIO(B) bus cycles.

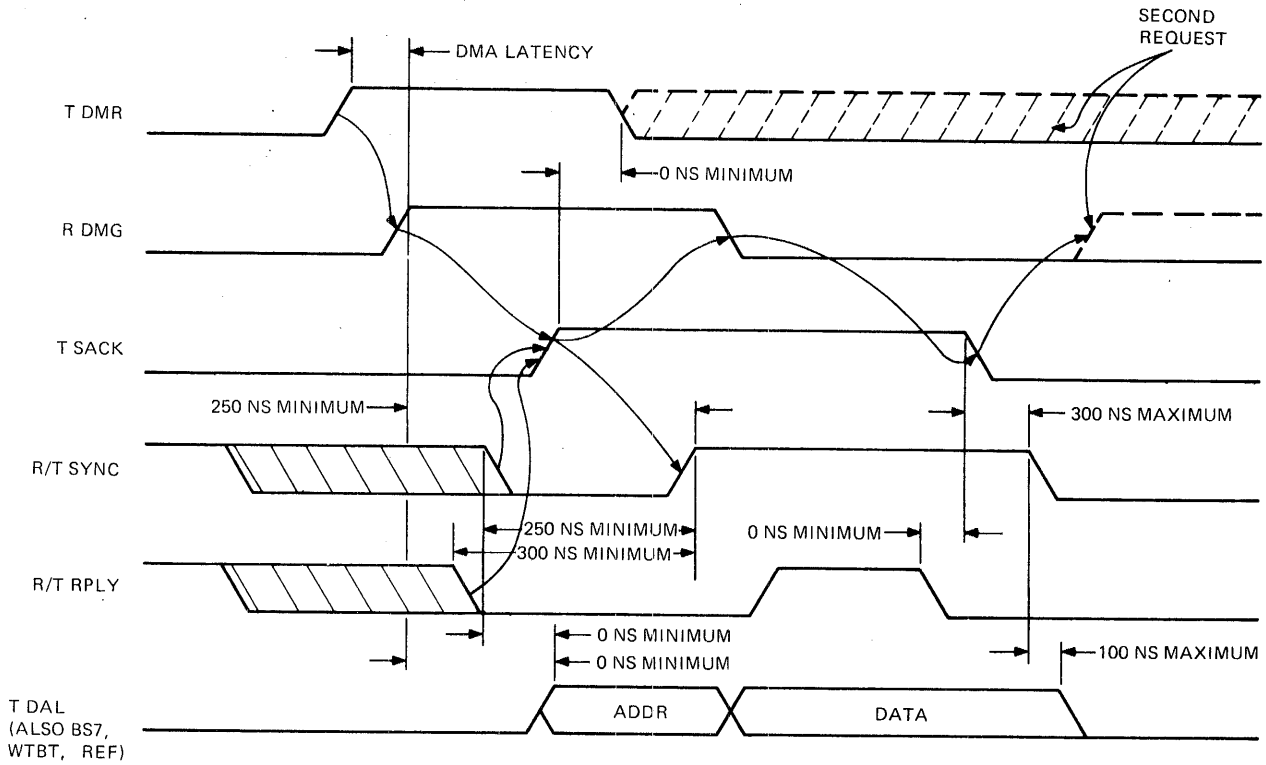


Figure 6-8 DMA Request/Grant Bus Cycle Timing

The DMA device can assert TSYNC L for a data transfer 0 ns (minimum) after it receives RDMGI L, 250 ns (minimum) after RSYNC is negated, and 300 ns (minimum) after RRPLY is negated.

During the bus mastership relinquish phase, the DMA device relinquishes the bus by negating TSACK. This occurs after the last data transfer cycle (RRPLY negated) is completed (or aborted). TSACK may be negated up to 300 ns (maximum) before negating TSYNC.

6.5 INTERRUPTS

The interrupt capability of the LSI-11 bus allows any I/O device to temporarily suspend (interrupt) current program execution and divert processor operation for service of the requesting device. The processor inputs a vector from the device to start the service routine (handler). As with a device register address, the hardware fixes the device vector at locations within a designated range of addresses between 000 and 7778. The vector indicates the first of a pair of addresses. The content of the first address is read by the processor; it is the starting address of the interrupt handler. The content of the second address is a new PSW. PSW bits <7:5> can be programmed to a priority level from 0 to 78. Only interrupts on a level higher than the number in the PSW priority level field are serviced by the processor. If the interrupt priority level of the new PSW is higher than that of the original PSW, the new PSW raises the interrupt priority level and thus prevents lower-level interrupts from breaking into the current interrupt service routine. Control is returned to the interrupted program when the interrupt service routine is complete.

The original (interrupted) program address (PC) and its associated PSW are stored on a stack. The original PC and PSW are restored by a return from interrupt instruction (RTI or RTT) at the end of the service routine. The use of the stack and the LSI-11 bus interrupt scheme can allow interrupts to occur within interrupts (nested interrupts) if the requesting interrupt has a higher priority level than the interrupt currently being serviced.

Interrupts can be caused by LSI-11 bus options and can also originate in the processor. Interrupts originating in the processor are called traps and are caused by programming errors, hardware errors, special instructions, and maintenance features. The following are the LSI-11 bus signals used in interrupt transactions.

Signal	Name
IRQ4 L	Interrupt request priority level 4
BIRQ5 L	Interrupt request priority level 5
BIRQ6 L	Interrupt request priority level 6
BIRQ7 L	Interrupt request priority level 7
BIAKI L	Interrupt acknowledge input
BIAKO L	Interrupt acknowledge output
BDAL <15:0> L	Data/address lines
BDIN L	Data input strobe
BRPLY L	Reply

6.5.1 Device Priority

The LSI-11 bus supports the following two methods of determining device priority.

- Distributed arbitration – Priority levels are implemented on the hardware. When devices of equal priority level request an interrupt, priority is given to the device electrically closest to the processor.

- Position-defined arbitration – Priority is determined solely by electrical position on the bus. The device closest to the processor has the highest priority, while the device at the far end of the bus has the lowest priority.

The KDJ11-B uses both methods – distributed arbitration, with four levels of priority, and position-defined arbitration within each level. Interrupts on these priority levels are enabled/disabled by bits in the processor status word (PSW <7:5>). Single-level interrupt (position-defined) devices that interrupt on BIRQ4 can also be used in KDJ11-B systems, but must be placed in a bus slot following the last bus slot in which a position-independent device is installed.

6.5.2 Interrupt Protocol

Interrupt protocol has three phases: the interrupt request phase, the interrupt acknowledge and priority arbitration phase, and the interrupt vector transfer phase. The operations performed by the processor and interrupting device are shown in Figure 6-9. Interrupt protocol timing is shown in Figure 6-10.

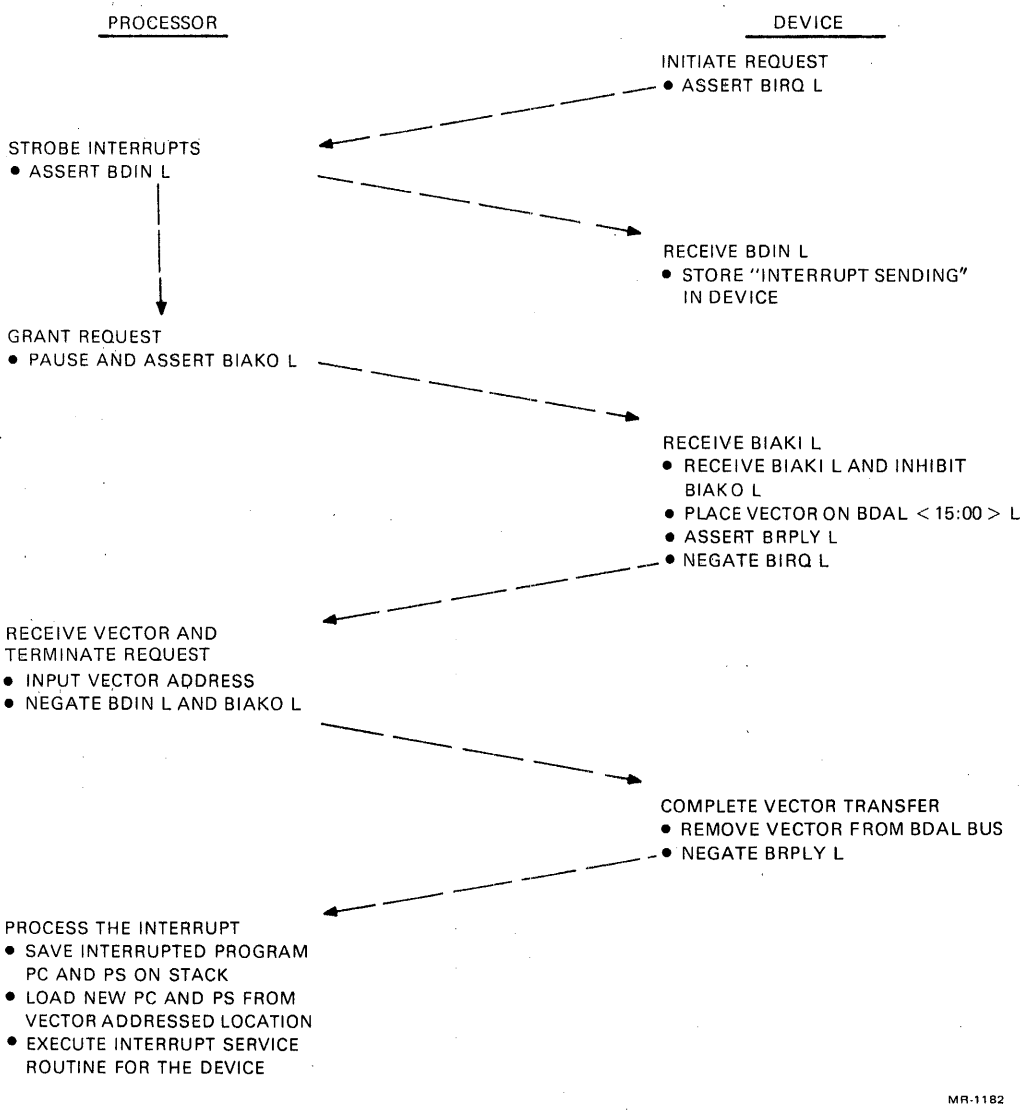


Figure 6-9 Interrupt Request/Acknowledge Sequence

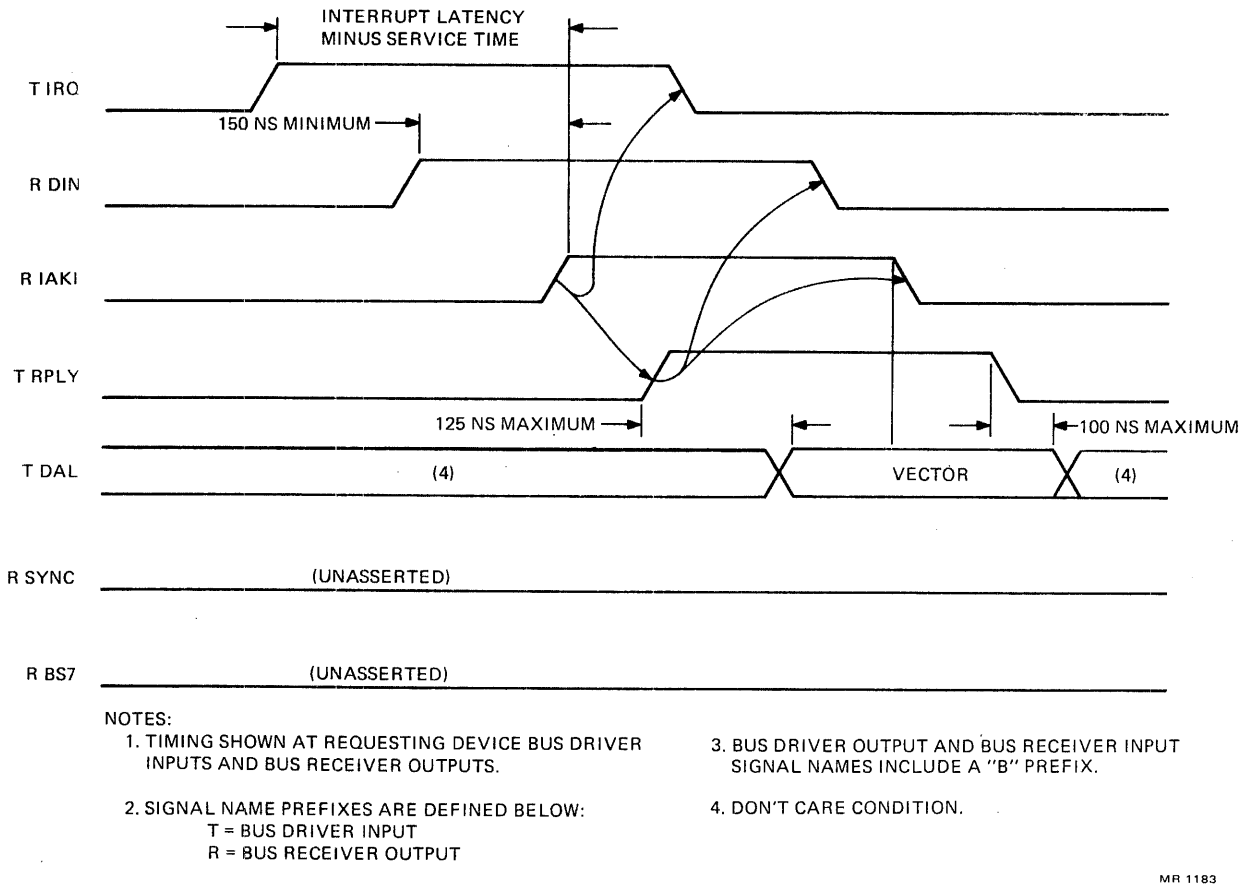


Figure 6-10 Interrupt Protocol Timing

The interrupt request phase begins when a device meets its specific conditions for interrupt requests (e.g., when the device is ready, done, or when an error has occurred). The interrupt enable bit in a device status register must be set. The device then initiates the interrupt by asserting the interrupt request line(s). BIRQ4 L is the lowest hardware priority level and is asserted for all interrupt requests for compatibility with previous LSI-11 processors. The level at which a device is configured must also be asserted. (A special case exists for level 7 devices that must also assert level 6.) The interrupt request line remains asserted until the request is acknowledged.

Interrupt Level	**Lines Asserted by Device**
4	BIRQ4 L
5	BIRQ4 L, BIRQ5 L
6	BIRQ4 L, BIRQ6 L
7	BIRQ4 L, BIRQ6 L, BIRQ7 L

During the interrupt acknowledge and priority arbitration phase, the KDJ11-B acknowledges interrupts under the following conditions.

1. The device interrupt priority is higher than the current priority level stored in PSW <7:5>.
2. The processor has completed instruction execution and no additional bus cycles are pending.

The processor acknowledges the interrupt request by asserting TDIN and, 225 ns (minimum) later, by asserting TIAKO. The device electrically closest to the processor receives the acknowledge on its RIAKI bus receiver.

On the leading edge of RDIN, each bus option capable of requesting interrupts decides whether to accept or to pass on the RIAKI signal. A device that does not support position-independent, multilevel interrupts accepts RIAKI if it is requesting an interrupt when RDIN asserts. A device that does support position-independent, multilevel interrupts accepts RIAKI if it is requesting an interrupt and if there is no higher-priority request pending when RDIN asserts. This decision must be clocked into a flip-flop, which settles within 150 ns of TDIN.

Devices that support position-independent, multilevel interrupts assert from one to three interrupt request lines when requesting an interrupt. Table 6-4 presents the Interrupt ReQuest (IRQ) lines a device at each level must assert in order to request an interrupt, and lists the lines it must monitor to determine whether a higher-priority device is requesting an interrupt.

During the interrupt vector transfer phase, the responding interrupt device receives RIAKI and then asserts TRPLY. The vector address must be stable at TDAL <8:2> 125 ns (maximum) after TRPLY is asserted. The processor receives the assertion of RRPLY and, 200 ns (minimum) later, it inputs the vector address and negates both TDIN and TIAKI. The interrupting device negates TRPLY after the negation of RIAKI, and removes the vector address from TDAL <8:2> 100 ns (maximum) after TRPLY negates. Since vector addresses are constrained between 000 and 7748, none of the remaining TDAL lines are used.

Table 6-4 Position-Independent, Multilevel Device Requirements

Interrupt Level	IRQ Lines Asserted	IRQ Lines Monitored
4	TIRQ4	RIRQ5, RIRQ6
5	TIRQ4, TIRQ5	RIRQ6
6	TIRQ4, TIRQ6	RIRQ7
7	TIRQ4, TIRQ6, TIRQ7	

6.5.3 4-Level Interrupt Configurations

Users having high-speed peripherals and desiring better software performance can use the 4-level interrupt scheme. Both position-independent and position-dependent configurations can be used with the 4-level interrupt scheme.

The position-independent configuration is shown in Figure 6-11. This configuration allows peripheral devices that use the 4-level interrupt scheme to be placed in the backplane in any order. These devices must send out interrupt requests and monitor higher-level request lines, as described in Paragraph 6.5.2. The level 4 request is always asserted by a requesting device, regardless of priority, to allow compatibility if an LSI-11 or LSI-11/2 processor is in the same system. If two or more devices of equally high priority request an interrupt, the device physically closest to the processor wins arbitration. Devices that use the single-level interrupt scheme must be modified or be placed at the end of the bus for arbitration to function properly.

The position-dependent configuration is shown in Figure 6-12. This configuration is simpler to implement, but has the following constraint: Peripheral devices must be ordered so that the highest-priority device is located closest to the processor, with the remaining devices placed in the backplane in decreasing order of priority.

With this configuration each device must only assert its own level and level 4 (for compatibility with an LSI-11 or LSI-11/2). Monitoring higher-level request lines is unnecessary. Arbitration is achieved through the physical positioning of each device on the bus. Single-level interrupt devices on level 4 must be positioned last on the bus.

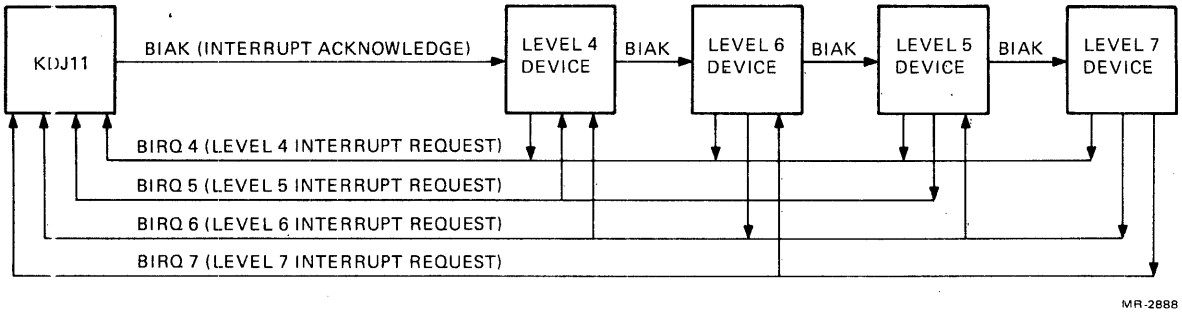


Figure 6-11 Position-Independent Configuration

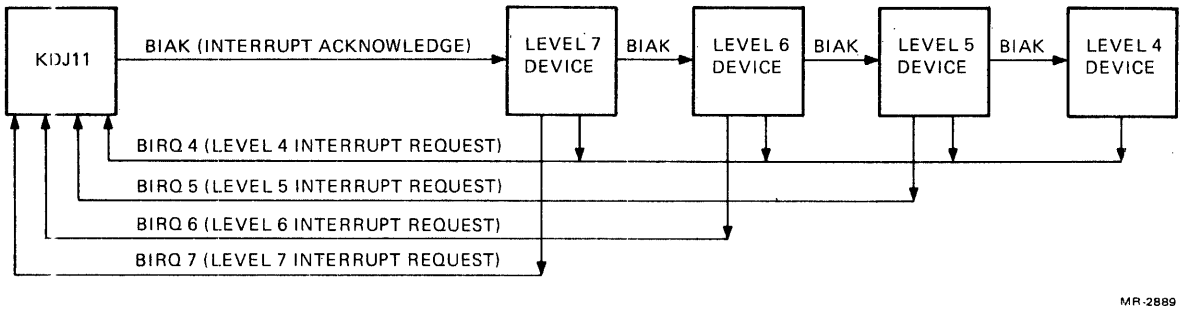


Figure 6-12 Position-Dependent Configuration

6.6 CONTROL FUNCTIONS

The following LSI-11 bus signals provide system control functions.

Signal	Name
BREF L	Memory refresh
BHALT L	Processor halt
BINIT L	Initialize
BPOK H	Power OK
BDCOK H	DC power OK
BEVNT L	External event interrupt request

6.6.1 Memory Refresh

If BREF is asserted during the address portion of a bus data transfer cycle, it causes all dynamic MOS memories to be addressed simultaneously. The sequence of addresses required for refreshing the memories is determined by the specific requirements of each memory. The complete memory refresh cycle consists of a series of refresh bus transactions. (A new address is used for each transaction.) The entire cycle must be completed within 2 ms. Multiple data transfers by DMA devices must be avoided since they could delay memory refresh cycles. The KDJ11-B does not perform memory refresh.

6.6.2 Halt

Assertion of BHALT L stops program execution and forces the processor unconditionally into console ODT mode. The processor does not assert the BHALT L bus line when it comes to a programmed halt.

6.6.3 Initialization

Devices along the bus are initialized when BINIT L is asserted. The processor asserts the BINIT L signal under the following conditions.

1. During a power-down sequence
2. During a power-up sequence
3. During the execution of a RESET instruction
4. After detection of a G character in ODT mode (if the processor features an ODT mode and a G command within it), and before execution of the code starting at the address that preceded the G command

6.6.4 Power Status

Power status protocol is controlled by two signals, BDCOK H and BPOK H. These signals are driven by an external device (usually the power supply) and are defined as follows.

6.6.4.1 BDCOK H – The assertion of this line indicates that dc power has been stable for at least 3 ms. Once asserted this line remains asserted until the power fails.

6.6.4.2 BPOK H – The assertion of this line indicates that there is at least an 8 ms reserve of dc power and that BDCOK H has been asserted for at least 70 ms. Once BPOK H has been asserted, it must remain asserted for at least 3 ms.

The negation of this line indicates that power is failing and that only 4 ms of dc power reserve remains. The negation of this line during processor operation initiates a power-fail trap sequence.

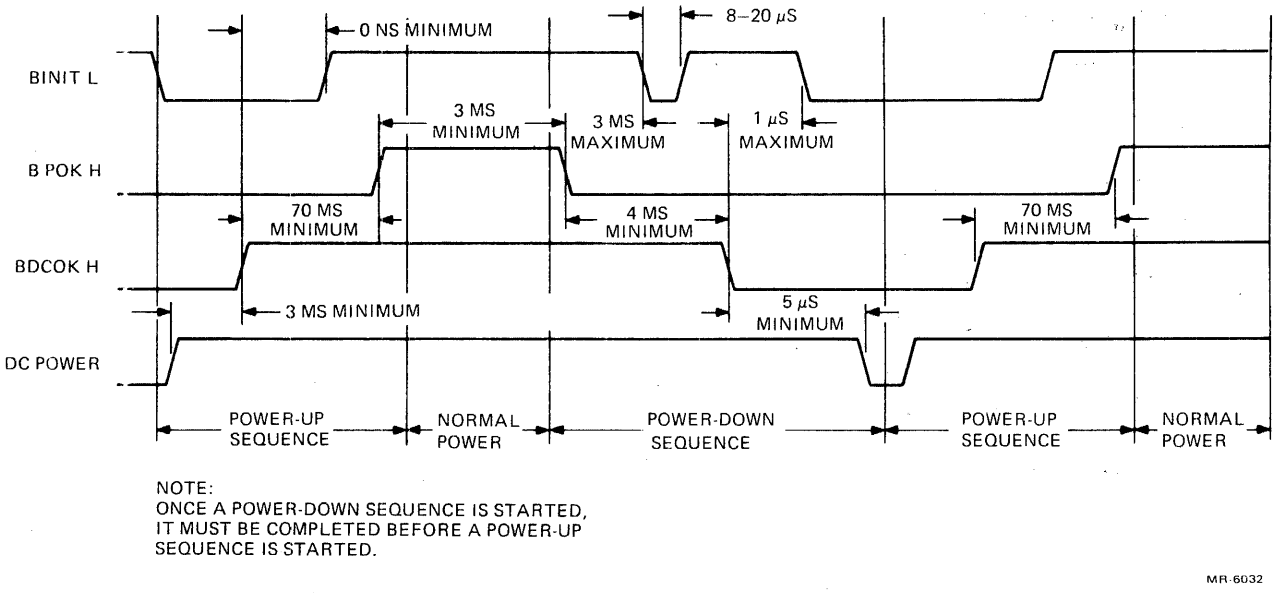


Figure 6-13 Power-Up/Power-Down Timing

6.6.4.3 Power-Up – The timing diagram for the power-up/power-down sequence is shown in Figure 6-13. The following events occur during a power-up sequence.

1. Logic associated with the power supply negates BDCOK H during power-up and asserts BDCOK H 3 ms (minimum) after dc power is restored to voltages within specification.

2. The processor asserts BINIT L after receiving nominal power and negates BINIT L 0 ns (minimum) after the assertion of BDCOK H.

3. Logic associated with the power supply negates BPOK H during power-up and asserts BPOK H 70 ms (minimum) after the assertion of BDCOK H. If power does not remain stable for 70 ms, BDCOK H is negated. Therefore, devices must suspend critical actions until BPOK H is asserted.

4. BPOK H must remain asserted for a minumum of 3 ms. BDCOK H must remain asserted 4 ms (minimum) after the negation of BPOK H.

6.6.4.4 Power-Down – The following events occur during a power-down sequence.

1. If the ac voltage to a power supply drops below 75% of the nominal voltage for one full line cycle (15 to 24 ms), BPOK H is negated by the power supply. Once BPOK H is negated, the entire power-down sequence must be completed.

   A device that requested bus mastership before the power failure that has not become bus master must maintain the request until BINIT L is asserted or the request is acknowledged (in which case regular bus protocol is followed).

2. Processor software must execute a RESET instruction 3 ms (minimum) after the negation of BPOK H. This asserts BINIT L for 8 to 20 μs. Processor software executes a HALT instruction immediately following the RESET instruction.

3. BDCOK H must be negated a minimum of 4 ms after the negation of BPOK H. This 4 ms allows mass storage and similar devices to protect themselves against erasures and erroneous writes during a power failure.

4. The processor asserts BINIT L 1 μs (minimum) after the negation of BDCOK H.

5. The dc power must remain stable for a minimum of 5 μs after the negation of BDCOK H.

6. BDCOK H must remain negated for a minimum of 3 ms.

6.6.5 BEVNT L

The BEVNT L signal is an external line clock interrupt request to the processor. When BEVNT L is asserted, the processor internally assigns location 1008 as the vector address for the BEVNT service routine. Because the vector is internally assigned, the processor does not execute the protocol for reading in the interrupt vector address (as is the case for other external interrupt requests).

6.7 BUS ELECTRICAL CHARACTERISTICS

This paragraph contains information about the electrical characteristics of the LSI-11 bus.

6.7.1 Signal Level Specification

Input Logic Levels

TTL logical low:	0.8 Vdc (maximum)
TTL logical high:	2.0 Vdc (minimum)

Output Logic Levels

TTL logical low:	0.4 Vdc (maximum)
TTL logical high:	2.4 Vdc (minimum)

6.7.2 AC Bus Load Definition

The amount of capacitance a module presents to a bus signal line is the ac bus load. This capacitance is measured between each module signal line and ground, and is expressed in ac unit loads, where each unit load is defined as 9.35 pF.

6.7.3 DC Bus Load Definition

The amount of leakage current a module presents to a bus signal line is the dc bus load. A dc unit load is defined as 105 μA flowing into a module device when the signal line is in the unasserted (high) state.

6.7.4 120 Ω LSI-11 Bus

The electrical conductors interconnecting the bus device slots are treated as transmission lines. A uniform transmission line, terminated in its characteristic impedance, propagates an electrical signal without reflections. Insofar as bus drivers, receivers, and wiring connected to the bus have finite resistance and nonzero reactance, the transmission line impedance becomes nonuniform, and thus introduces distortions into pulses propagated along it. Passive components of the LSI-11 bus (such as wiring, cabling, and etched signal conductors) are designed to have a nominal characteristic impedance of 120 Ω.

The maximum length of the interconnecting cable in multiple-backplane systems (excluding wiring within the backplane) is limited to 4.88 m (16 ft).

NOTE

The KDJ11-B processor (as well as all standard Digital-supplied LSI-11 interfaces) connects to the bus via special drivers and receivers described in Paragraphs 6.7.5 and 6.7.6.

The KDJ11-B processor provides resistive (250 Ω) pull-up on all bussed lines to 3.4 Vdc for this wired-OR interconnecting scheme.

6.7.5 Bus Drivers

Devices driving the 120 Ω LSI-11 bus must have open collector outputs and meet the specifications that follow.

DC Specifications*

- Vcc may vary from 4.75 V to 5.25 V.
- Output low voltage when sinking 70 mA of current: 0.7 V (maximum).
- Output high leakage current when connected to 3.8 Vdc: 25 μA (even if no power is applied to them, except for BDCOK H and BPOK H).

AC Specifications

- Bus driver output pin capacitance load: Not to exceed 10 pF.
- Propagation delay: Not to exceed 35 ns.
- Driver skew (difference in propagation time between slowest and fastest bus driver): Not to exceed 25 ns.
- Rise/fall times: Transition time from 10% to 90% for positive transition, and from 90% to 10% for negative transition, must be no faster then 5 ns.

6.7.6 Bus Receivers

Devices that receive signals from the 120 Ω LSI-11 bus must meet the following requirements.

DC Specifications†

- Vcc may vary from 4.75 V to 5.25 V.
- Input low voltage: 1.3 V (maximum).
- Input high voltage: 1.7 V (minimum).
- Maximum input leakage current when connected to 3.8 Vdc: 80 μA with Vcc between 0.0 V and 5.25 V.

AC Specifications

- Bus receiver input pin capacitance load: Not to exceed 10 pF.
- Propagation delay: Not to exceed 35 ns.
- Receiver skew (difference in propagation time between slowest and fastest receiver): Not to exceed 25 ns.

---

* These conditions must be met at worst-case supply voltage, temperature, and input signal levels.

† These conditions must be met at worst-case supply voltage, temperature, and output signal conditions.

6.7.7 KDJ11-B Bus Termination

The 120 Ω LSI-11 bus should be terminated at each end by an appropriate resistive termination. A pair of resistors in series from +5.0 V to ground is used to establish a voltage for each bidirectional line when that line is not being driven (negated). The parallel impedance of this pair of resistors is 250 Ω. The terminating resistors are shown in Figure 6-14. The KDJ11-B contains terminating resistor networks in 18-pin single-in-line packages to provide the 120 Ω (terminations for the data/address, synchronization, and control lines) at the processor end of the bus.

Some system configurations do not require terminating resistors at the far end of the bus. If the system configuration does require such termination, it is typically provided by an M9404-YA cable connector module.

6.7.7.1 Bus Interconnection Wiring – The bus interface for the module connectors is provided by one, two, or three backplanes, depending on the system configuration. Since each backplane may contain up to 9 slots, a system may have a maximum of 27 module interfaces to the bus.

6.7.7.2 Backplane Wiring – The wiring that interconnects all device interface slots on the LSI-11 bus must meet the following specifications.

1. The conductors must be arranged so that each line exhibits a characteristic impedance of 120 Ω (measured with respect to the bus common return).

2. Crosstalk from a pulse-driven line to an undriven line to which a constant 5 V is applied must be less than 5% of the 5 V. Note that worst-case crosstalk is manifested by simultaneously driving all but one signal line and measuring the effect on the undriven line.

3. The dc resistance of a bus segment signal path, as measured between the near-end terminator and far-end terminator modules (including all intervening connectors, cables, backplane wiring, connector-module etch, etc.), must not exceed 2 Ω.

4. The dc resistance of a bus segment common return path, as measured between the near-end terminator and far-end terminator modules (including all intervening connectors, cables, backplane wiring, connector-module etch, etc.), must not exceed an equivalent of 2 Ω per signal path. Thus, the composite signal return path dc resistance must not exceed 2 Ω divided by 40 bus lines, or 50 MΩ. Note that although this common return path is nominally at ground potential, the conductance must be part of the bus wiring; the specified low-impedance return path must be provided by the bus wiring as distinguished from common system or power ground path.

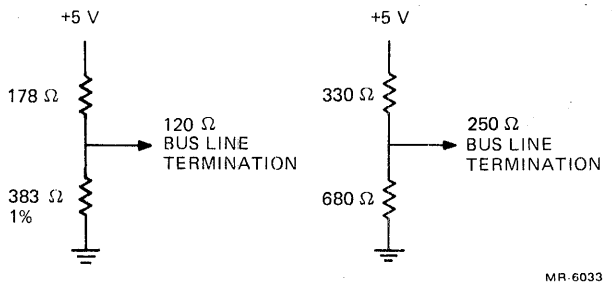


Figure 6-14 Bus Line Termination

6.7.7.3 Intrabackplane Bus Wiring – The wiring that interconnects the bus connector slots within one contiguous backplane is part of the overall bus transmission line. Due to implementation constraints, the nominal characteristic impedance of 120 Ω may not be achievable. Distributed wiring capacitance in excess of the amount required to achieve the nominal 120 Ω impedance may not exceed 60 pF per signal line per backplane.

6.7.7.4 Power and Ground – Each bus interface slot has connector pins assigned for the following dc voltages.

Voltage	Number of Pins
+5 Vdc	Three pins, 4.5 A (maximum) per bus device slot
+12 Vdc	Two pins, 3.0 A (maximum) per bus device slot
Ground	Eight pins, shared by power return and signal return

The maximum allowable current per pin is 1.5 A. The +5 Vdc must be regulated to +5% and the maximum ripple should not exceed 100 mV peak-to-peak. The +12 Vdc must be regulated to +3% and the maximum ripple should not exceed 200 mV peak-to-peak.

NOTE
Power is not bussed between backplanes on any interconnecting LSI-11 bus cables.

6.7.7.5 Maintenance and Spare Pins – There are four M SPARE pins per bus device slot assigned to maintenance (AK1, AL1, BK1, BL1). The maintenance pins on the basic LSI-11 system are not bussed from module to module. Instead, at each bus device slot, the maintenance pins are shorted together as pairs. These pins must be shorted together for some modules to operate. This allows a module to use these pins during initial testing as two separate points. This feature is used by Digital for manufacturing tests only. Spare pins are allocated on the backplane as follows.

S SPARES – Four pins: AE1, AH1, BH1, AF1 (with the exception of AF1 in slot 1), are reserved for the particular use of a module or set of modules. They may be used as test points or for intermodule connection. Appropriate wires must be added for intermodule communication since these pins are not connected in any way. The processor uses AF1 in slot 1 as an output pin for the SRUN signal. S SPARE lines cannot be used as bus connections.

P SPARES – Two pins: AU1 and BU1, are similar to the S SPARE pins except that they are located in a manner that causes dc voltages to appear on them if a module is inserted backwards. Use of these pins is not recommended.

6.8 SYSTEM CONFIGURATIONS

LSI-11 bus systems can be divided into two types. The first type comprises those systems that use only one backplane, the second type comprising those systems that use multiple backplanes. Two sets of configuration rules are necessary to accommodate the different electrical characteristics of the two types of systems.

Three characteristics of each component in an LSI-11 bus system must be known before configuring any system.

- Power consumption – The total amount of current drawn from the +5 Vdc and +12 Vdc power supplies by all modules in the system.

- AC bus loading – The amount of capacitance a module presents to a bus signal line. AC loading is expressed in ac unit loads, where one ac unit load equals 9.35 pF of capacitance.

- DC bus loading – The amount of dc leakage current a module presents to a bus signal when the line is high (undriven). DC loading is expressed in terms of dc unit loads, where one dc unit load equals 105 μA (nominal).

Power consumption, ac loading, and dc loading specifications for each module are included in the *Microcomputer Interfaces Handbook*.

NOTE

The ac and dc loads and the power consumption of the processor module, terminator module, and backplane must be included in determining the total bus loading of a backplane.

6.8.1 Rules for Configuring Single-Backplane Systems

The following rules apply only to single-backplane systems. Any extension of the bus off the backplane is considered a multiple-backplane system and must be configured accordingly. A single-backplane configuration diagram is shown in Figure 6-15.

1. The bus can accommodate modules that have up to 35 ac loads (total) before the termination is required. The processor has on-board termination for one end of the bus. If more than 20 ac loads are included, the other end of the bus must be terminated.

2. A 120 Ω terminated bus can accommodate modules comprising up to 45 ac loads (total).

3. The bus can accommodate modules up to 20 dc loads (total).

4. The bus signal lines on the backplane can be up to 35.6 cm (14 in) long.

5. It is recommended that the far end of the bus be terminated with 240 Ω.

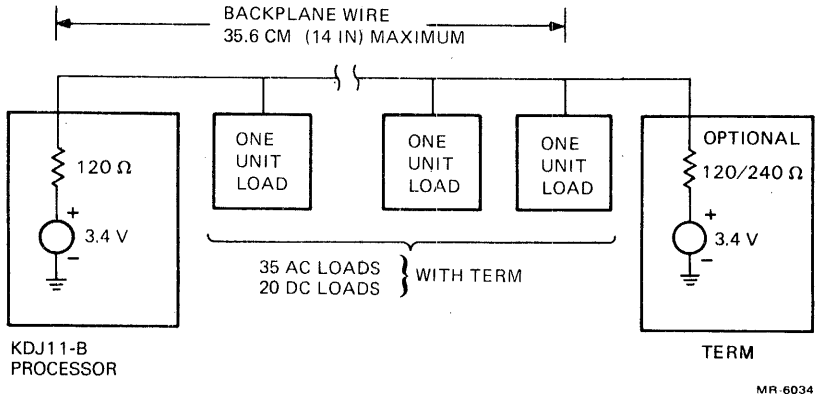


Figure 6-15 Single-Backplane Configuration

6.8.2 Rules for Configuring Multiple-Backplane Systems

Multiple-backplane systems can contain a maximum of three backplanes. A configuration diagram for a multiple-backplane system is shown in Figure 6-16.

1. The signal lines on each backplane can be up to 25.4 cm (10 in) long.

2. Each backplane can accommodate modules that have up to 20 ac loads (total). Unused ac loads from one backplane may not be added to another backplane if the second backplane loading will then exceed 20 ac loads. Loading backplanes equally is recommended.

3. The dc loading of all modules in all backplanes cannot exceed 30 loads (total).

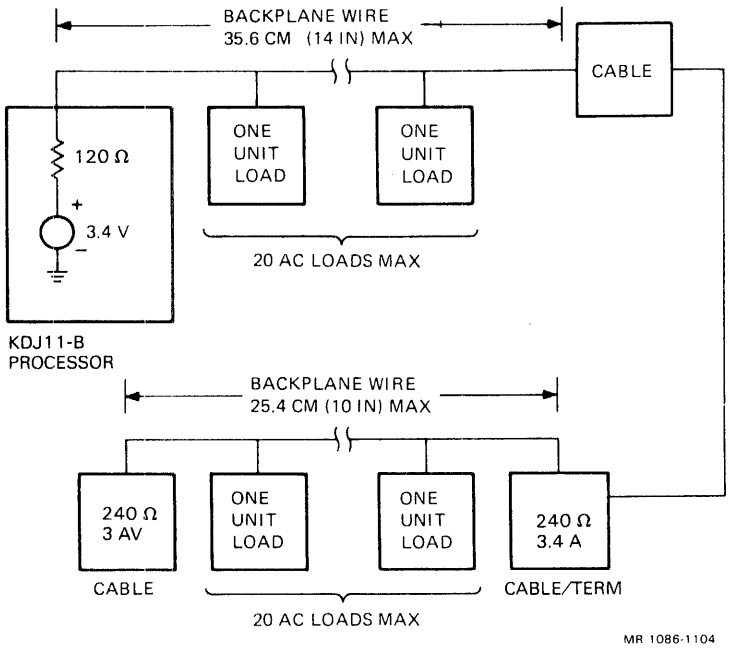


Figure 6-16 Multiple-Backplane Configuration

4. The first backplane must have an impedance of 120 Ω (obtained via the processor module). The second backplane is terminated by 240 Ω resistor networks contained on the backplane.

5. The cables connecting the backplanes must observe the following conditions.

   a. The cable(s) connecting the two backplanes must be 61 cm (2 ft) or greater in length.

   b. The length of the cables must not exceed 4.88 m (16 ft).

   c. The cables used must have a characteristic impedance of 120 Ω.

6.8.3 Power Supply Loading

Total power requirements for each backplane can be determined by obtaining the total power requirements for each module in the backplane. Obtain separate totals for +5 V and +12 V power. Power requirements for each module are specified in the *Microcomputer Interfaces Handbook*.

Do not attempt to distribute power via the LSI-11 bus cables in multiple-backplane systems. Provide separate, appropriate power wiring from each power supply to each backplane. Each power supply should be capable of asserting BPOK H and BDCOK H signals according to bus protocol. This is required if automatic power-fail/restart programs are implemented or if specific peripherals require an orderly power-down halt sequence. The proper use of the BPOK H and BDCOK H signals is strongly recommended.

CHAPTER 7
PRIVATE MEMORY INTERCONNECT BUS

7.1 DESCRIPTION

The PMI bus provides a high performance communications path between the KDJ11-B CPU module, the MSV11-J memory modules and the KTJ11-B UBA. The PMI bus consists of 14 signals that support the PMI protocol and the additional LSI bus signals that are shared with the LSI bus protocol. The address and data information is multiplexed using the same LSI bus data/address lines. The PMI protocol is designed for LSI systems and unique LSI-controlled Unibus systems that use the UBA.

7.2 PMI INTERFACE

The PMI interface signals are defined as the PMI bus master signals, the PMI slave signals and the PMI Unibus adapter signals. These interface signals are assigned to the C and D rows of the backplane and are defined as the interconnect bus. The PMI interface signals on the C/D bus are normally assigned two pins to provide an interconnection between the slots. The KDJ11-B module is only assigned one pin and therefore its position in the backplane is critical. The LSI bus signals that are used with the PMI protocol use the A and B rows of the backplane defined as the LSI bus.

7.2.1 PMI Bus Master Signals

The PMI bus master controls the PMI bus cycles by using the nonmultiplexed control signals described in Table 7-1. These signals are asserted low and negated high.

7.2.2 PMI Slave Signals

The PMI slave responds to the bus master by the nonmultiplexed signals listed in Table 7-2. These signals are asserted low and negated high by any device that is capable of being a slave.

7.2.3 PMI Unibus Adapter Signals

The UBA is used exclusively for Unibus systems. The PMI incorporates a special group of signals to establish communications between the KDJ11-B and the UBA. These signals are nonmultiplexed as described in Table 7-3 and are not used in any LSI based system.

7.2.4 LSI Bus Signals

The PMI protocol uses some of the standard LSI bus signals in conjunction with the PMI high speed control signals. These LSI bus signals may not be used exactly as they are used in an LSI bus operation. The LSI bus signals used with the PMI are listed with their PMI functions in Table 7-4.

Table 7-1 PMI Bus Master Signals

Pin	Mnemonic	Function
DC1	PBYT L	**PMI Byte** PBYT L is asserted or negated in conjunction with the BWTBT L LSI bus signal to select the type of bus cycle as follows.

BWTBT L	PBYT L	Bus Cycle
H	H	DATI or DATBI
H	L	DATIP
L	H	DATO
L	L	DATOB

CE1	PBCYC L	**PMI Bus Cycle** The PMI bus master starts a PMI cycle by asserting PBCYC L and ends a PMI cycle by negating PBCYC L.
CP1	PBLKM L	**PMI Block Mode** To read more than two words, the PMI bus master uses PBLKM L and PBCYC L to control the timing of the DATBI cycle. Both PBLKM L and PBCYC L are asserted at the start of the DATBI cycle, and after reading two words PBLKM L is negated. If there are more than two words that remain to be read, PBLKM L is asserted and negated every time two words are read (except for the last two words, where it remains negated). After reading the last two words, PBCYC is also negated.
DB1	PWTSTB L	**PMI Write Strobe** After the bus master gates the data onto the bus, PWTSTB is asserted to latch the data into the write buffer of the PMI slave.

Table 7-2 PMI Slave Signals

Pin	Mnemonic	Function
CB1	PSSEL L	**PMI Slave Selected** Whenever a slave is addressed by the BDAL bus lines, it responds by asserting PSSEL L. The UBA does not assert this signal.
CH1	PHBPAR L	**PMI High Byte Data Parity** This signal is generated by the selected PMI memory module during DATI and DATBI cycles. It provides an odd parity bit for the high data byte transmitted on BDAL <15:8>.
CK1	PLBPAR L	**PMI Low Byte Data Parity** This signal is generated by the selected PMI memory module during DATI and DATBI cycles. It provides an even parity bit for the low data byte transmitted on BDAL <7:0>.
CM1	PRDSTB L	**PMI Read Strobe** This signal is asserted and negated by the selected PMI memory module to control data transfers during DATI and DATBI cycles. The bus master uses the negating edge of PRDSTB L to latch the first data word. The second data word is latched at a specified time after PRDSTB L is negated.
CJ1	PSBFUL L	**PMI Slave Buffer Full** The selected PMI slave asserts PSBFUL L during DATO and DATBO cycles to indicate that its write buffer is full and, consequently, it cannot respond to another cycle request. The bus master may output another address while PSBFUL L is asserted, but it must not assert PBCYC L until PSBFUL L is negated.

Table 7-3 PMI Unibus Adapter Signals

Pin	Mnemonic	Function
DD1	PMAPE L	**PMI Unibus Map Enable** The KDJ11-B asserts this signal when bit 5 of MMR3 is set. The signal is negated when bit 5 is cleared or reset. The UBA enables the Unibus map when PMAPE L is asserted and disables the Unibus map when PMAPE L is negated. The memory modules do not use this signal.
CF1	PUBSYS L	**PMI Unibus System** In a Unibus system, PUBSYS L is asserted by the UBA to direct the KDJ11-B to follow PMI protocol for all data transfers, whether PSSEL L is asserted or not. LSI-11 bus protocol is disabled for all PMI devices when PUBSYS L is asserted. In an LSI-11 system, PUBSYS L is always negated. If PSSEL L is negated, the KDJ11-B follows LSI-11 protocol and the PMI memory then responds to the LSI-11 protocol by the LSI DMA devices.
CD1	PUBMEM L	**PMI Unibus Memory** The UBA asserts PUBMEM L to indicate that Unibus memory space is being addressed. The signal is latched when PBCYC L is asserted. When a PMI slave is addressed, it asserts PSSEL L, but it must not respond to the PMI control signals if PUBMEM L is asserted. The KDJ11-B ignores the PSSEL L signal if PUBMEM L is asserted.
CV1	PUBTMO L	**PMI Unibus Timeout** The UBA asserts PUBTMO L in response to any of the following conditions. • When an NXM timeout occurs and the KDJ11-B addresses the Unibus • When a sack timeout occurs during an interrupt cycle • When a Unibus interrupting device was granted bus mastership, but fails to execute an interrupt transaction
CR1	PBSY L	**PMI Busy** This signal is asserted by the PMI bus master (KDJ11-B or UBA) when it gains control of the PMI bus. The PMI bus master negates this signal when it relinquishes PMI mastership. The KDJ11-B is the bus master at power-up and when the bus is idle.

Table 7-4 LSI Bus Signals

Pin	Mnemonic	Function
AK2	BWTBT L	**Write Byte (PMI Write Indication)** In a PMI system, BWTBT L is used in conjunction with PBYT L to define the data transfer cycle. BWTBT L and PBYT L are asserted for this purpose when the bus master gates the address onto the BDAL lines.

BWTBT L	PBYT L	Bus Cycle
H	H	DATI or DATBI
H	L	DATIP
L	H	DATO
L	L	DATOB

AF2	BRPLY L	**Reply** During PMI cycles, BRPLY L is asserted by the KDJ11-B and the PMI slave to prevent the next bus master from gaining control of the bus too soon. In a Unibus system, BRPLY L is asserted by the UBA as a slave response during the PMI DATOB cycle and interrupt vector DATI cycle.

NOTE

The PMI memory slave modules in a Unibus system must have BRPLY L disabled at all times.

AH2	BDIN L	**Data Input** The BDIN L signal is only used in PMI Unibus systems during interrupt grant cycles. The KDJ11-B asserts BDIN L after it gates the interrupt priority, BDAL bits <3:0>, onto the bus. The UBA then latches the interrupt priority data using the leading edge of BDIN L.
AM2	BIAKI L	**Interrupt Acknowledge In**
AN2	BIAKO L	**Interrupt Acknowledge Out** These signals are only used in PMI Unibus systems during the interrupt grant cycles. The KDJ11-B asserts the BIAKI L signal and the UBA acknowledges it by asserting one of the Unibus bus grant signals.
BB1	BPOK H	**Power OK** This signal is only used in PMI Unibus systems for the Unibus power-up/power-down protocol. This signal is asserted and negated by the UBA in response to the Unibus AC LO signal. The assertion of AC LO may be prolonged by the Unibus devices or the PMI memory during power-up.

7.3 PMI OPERATION IN AN LSI-11 SYSTEM

The KDJ11-B is the default bus master in an LSI-11 system. Any bus device that has the appropriate circuits can become the bus master and control data transfers via the LSI-11 bus. The KDJ11-B relinquishes control of the bus by acknowledging a DMA request from a DMA device which then becomes bus master. During the time that a DMA device is bus master, there is no PMI master. The standard LSI-11 bus operations are described in Chapter 6.

If the KDJ11-B receives a DMA request while performing a PMI cycle or while gating an address onto the bus, it must also perform the following relationships.

1. If the KDJ11-B has gated an address onto the bus for a PMI cycle or an LSI bus cycle and wants to abort the cycle, it removes the address and control signals from the bus and asserts the BDMG L signal.

2. In a PMI data transfer cycle, the KDJ11-B asserts the BDMG L signal after it asserts the BRPLY L signal.

3. In a PMI DATIP cycle, the KDJ11-B negates the BRPLY L signal before the PMI slave removes the data from the bus.

4. In a PMI DATOB cycle, the KDJ11-B negates the BRPLY L signal before it removes the data from the bus.

5. In a PMI DATOB cycle, the PMI slave negates the BRPLY L signal before it is ready to receive the BSYNC L signal from a DMA device.

The KDJ11-B can regain bus mastership only after BSYNC L and BSACK L have been negated by the DMA device.

7.4 PMI OPERATION IN A UNIBUS SYSTEM

In a Unibus system the KDJ11-B CPU is the default PMI master and the KTJ11-B UBA is the default Unibus master. When the CPU as the PMI master addresses the Unibus memory or I/O page, the UBA responds as a PMI slave while simultaneously controlling the Unibus side of the transaction as the bus master.

The UBA can become the PMI master when the CPU issues a DMA grant or performs an interrupt transaction. The DMA or interrupt grant is accepted by the UBA and passes the DMA or interrupt grant onto a Unibus device, which would then become the Unibus master.

In Unibus systems, the bus master and PMI master can be requested by an NPR or interrupt request from a bus device, or a DMA or interrupt request from the UBA.

7.4.1 Bus Device NPR or DMA

Any Unibus device that is capable of being a Unibus master can issue an NPR or DMA request to become bus master and control data transfers. When a Unibus device becomes the bus master through an NPR or DMA request, it can perform Unibus DATI, DATIP, DATO and DATOB cycles. The UBA responds as a Unibus slave when accessing PMI memory, the PMI I/O page or a UBA I/O page location on behalf of a Unibus master. During the same cycle, the UBA also acts as the PMI bus master to control the PMI portion of the data transfer for accesses to PMI memory or the PMI I/O page.

The KDJ11-B and the UBA use the following protocol to arbitrate an NPR.

1. The UBA asserts the DMA request (DMR) after receipt of a Unibus NPR or when it is ready to transfer data to or from memory.

2. The KDJ11-B bus arbitrator asserts the DMA grant (DMGO) after receiving the DMR input and after the negation of BSACK by the UBA.

> **NOTE**
> **The KDJ11-B does not always give DMA requests unconditional priority. The KDJ11-B can be programmed to retain top priority for a predetermined amount of time while waiting to perform a memory transfer or honor an interrupt request.**

3. The UBA enters the DMA cycle if it is the highest requesting priority or it asserts the nonprocessor grant (NPG) to the Unibus after receiving the DMG from the KDJ11-B.

4. Since the UBA does not have the required priority it cannot be the next bus master. Instead, it negates bus busy (BBSY) after the assertion of DMR and clears the Unibus.

5. The device with the highest priority asserts select acknowledge (SACK) to the UBA and negates the NPR after the UBA asserts NPG.

6. This device is now master of the Unibus and asserts BBSY and SACK when the previous bus master relinquishes the bus by negating BBSY. The new bus master may then initiate data transfer cycles.

7. The UBA asserts BSACK to the KDJ11-B after receiving Unibus SACK or because of a timeout occurring 10 μs after it asserts NPG. If Unibus SACK is not received within 10 μs after the assertion of NPG, the UBA automatically asserts BSACK.

8. The UBA asserts transmitted PMI busy (PBSY) after it is negated by the PMI bus master. The UBA is now the PMI bus master and can initiate PMI data transfer cycles.

9. The KDJ11-B bus arbitrator negates DMGO after BSACK is asserted. Since the UBA provides the timeout function, the KDJ11-B maintains DMGO until it receives BSACK.

10. The UBA negates NPG after the KDJ11-B negates DMGO.

11. The device that is the current bus master negates SACK after it asserts BBSY and receives the negation of NPG.

12. The UBA negates BSACK after the Unibus SACK is negated and after BBSY is asserted. The KDJ11-B bus arbitrator continues arbitration for 75 ns after BSACK is negated.

13. The bus master negates BBSY after it has cleared the bus.

14. If the KDJ11-B is the next PMI bus master, the UBA or the current bus master clears the bus, the PMI control data and negates PBSY to relinquish control of the PMI bus.

7.4.2 PMI Bus Device Interrupt

Any Unibus device that is capable of being a bus master can issue a BR7 through 4 request and become the bus master to control data or interrupt vector transfers. In both cases, the UBA is the PMI master and responds as a slave if the device performs an interrupt vector transaction or accesses the PMI memory, the PMI I/O page or the UBA I/O page. When a Unibus device becomes the bus master through an interrupt request, it can perform the same Unibus data transfers described for the NPR.

The KDJ11-B and the UBA use the following protocol to arbitrate an interrupt request.

1. In response to a Unibus device, the UBA asserts an interrupt request on BIRQ <7:4>.

2. The KDJ11-B bus arbitrator responds as follows.

   a. Asserts interrupt level on BDAL <3:0>.

   b. Asserts BDIN 150 ns after gating BDAL <3:0>.

   c. Asserts the interrupt acknowledge grant (BIAKO) 250 ns after asserting BDIN.

3. The UBA latches BDAL <3:0> when BDIN is asserted and asserts the Unibus interrupt level BG <7:4> after BIAKO is asserted.

4. Since the UBA does not have the highest priority, it negates BBSY after it asserts BG <7:4> and clears the Unibus.

5. The Unibus device with the highest priority asserts select acknowledge (SACK) after it receives BG <7:4> and negates its interrupt request.

6. The UBA asserts BSACK to the KDJ11-B after the device asserts SACK.

NOTE

The UBA asserts PUBTMO to indicate a timeout if SACK is not received within 10 μs after the assertion of BG <7:4>. The KDJ11-B cancels the interrupt cycle and becomes the PMI bus master by receiving PUBTMO.

7. The UBA asserts PBSY after it asserts BSACK and after the previous PMI bus master negates PBSY. The UBA now has control of the PMI and may initiate PMI data transfer or interrupt cycles after PBSY is asserted.

8. The UBA negates BG <7:4> after BSACK is asserted and negates BDGMO if BSACK is not asserted within the 10 μs timeout period.

9. The new Unibus master asserts BBSY after it asserts SACK and the previous bus master negates BBSY.

10. The bus master negates SACK after the negation of BG <7:4> and after the assertion of BBSY.

11. The UBA negates BSACK to the KDJ11-B after the negation of SACK and the assertion of PBSY.

12. The KDJ11-B resumes NPR arbitration for 75 ns after the negation of BSACK, but does not resume BIRQ arbitration until the interrupt request is aborted by the assertion of PUBTMO or the completion of the interrupt operation.

13. If a Unibus device responds to BG <7:4> with one or more DMA transfers, the UBA responds as it would to a device that received bus mastership by an NPR request. The assertion of BDIN and BIAKO by the KDJ11-B has no effect on the PMI protocol.

14. If the Unibus master relinquishes control without sending the interrupt vector, the UBA asserts PUBTMO, indicating a timeout to the KDJ11-B, and the interrupt cycle is aborted.

15. The Unibus master negates BBSY after it clears the Unibus.

7.5 PMI DATA TRANSFERS

There are three general categories of PMI data transfer cycles. They are the DATI/DATIP, DATBI, and DATO/DATOB cycles. They are briefly described below.

On the Q22-Bus, the bus master can perform a read-modify-write (DATIO or DATIOB) cycle that transmits an address, reads a data word or byte, and then writes the data word or byte to the same address. The PMI read-modify-write is performed by a DATIP cycle followed by a DATO or DATOB cycle. The PMI bus master has the responsibility of controlling the bus for the duration of both cycles.

7.5.1 PMI Data In/Data In Pause

The DATI and the DATIP cycles are used to read one or two words when the PMI bus master accesses the PMI memory. When the PMI bus master accesses the I/O page or the Unibus memory, it can read only one word. The PMI bus master detects an I/O page reference by the assertion of TBS7 and a Unibus memory reference by the assertion of PUBMEM.

The PMI DATIP cycle is identical to the DATI cycle except that TPBYT is asserted with TADDR to indicate that the cycle immediately following the current cycle will be a DATO cycle to the same address. The protocol used by the DATI and DATIP cycles is as follows.

1. When the PMI master assumes control of the bus, the BDAL <21:0> lines are addressed, BBS7 is asserted, and PBYT is asserted for DATIP cycles.

2. Each PMI slave asserts PSSEL within 45 ns after it receives the asserted BDAL <21:0> and BBS7 signals, if necessary.

3. The UBA asserts PUBMEM within 100 ns ns after it receives the asserted BDAL <21:0> and BBS7 signals, if necessary.

4. The PMI master receives PSSEL within 130 ns after gating the asserted BDAL <21:0> and BBS7 signals.

5. The PMI master receives PUBMEM within 120 ns after gating the asserted BDAL <21:0> and BBS7 signals.

6. If PSSEL is asserted and PUBMEM is negated, the PMI master proceeds as follows.

   a. The PMI master asserts PBCYC within 130 ns after gating the BDAL <21:0>, BBS7, and PBYT signals and only after PSBFUL is negated.

   b. The PMI master continues to assert the BDAL <21:0>, BBS7, and PBYT signals for a minimum of 40 ns and a maximum of 100 ns after asserting PBCYC.

   c. The UBA latches PUBMEM when PBCYC is asserted.

   d. The PMI slave receives stable BDAL <21:0>, BBS7, and PBYT signals for 65 ns (minimum) before PBCYC is asserted and for 30 ns after PBCYC is asserted.

   e. The PMI slave receives a valid PUBMEM from 10 ns (minimum) before the assertion of PBCYC and until 10 ns before PBCYC is negated.

7. If PSSEL is negated and the KDJ11-B is the PMI master, then PMI cycles are performed with the UBA responding as a slave, and follow the routine listed above.

8. If PSSEL is negated and PUBMEM is asserted, the UBA is the PMI master and it aborts the PMI cycle and does not respond as a Unibus slave.

9. The assertion of BRPLY by the PMI slave is optional in LSI systems. Its protocol is as follows.

   a. The PMI slave asserts BRPLY after PBCYC is asserted.

   b. The PMI slave negates BRPLY within 100 ns after the negation of PRDSTB.

NOTE
In Unibus systems with PMI memory as a slave, BRPLY must be disabled at all times.

10. The PMI slave gates the data onto the bus within 125 ns after the assertion of PBCYC.

11. The PMI slave gates PHBPAR and PLBPAR parity bits after the assertion of PBCYC. These parity bits are generated only for the memory locations being cached on the KDJ11-B from the main memory.

12. The PMI slave asserts PRDSTB after the assertion of PBCYC.

13. The PMI slave negates PRDSTB within 150 ns after the assertion of PBCYC. It is negated within 75 ns after the first data word is gated on the bus and 55 ns after the PHBPAR and PLBPAR bits are gated for the first word.

14. The PMI slave maintains the data word, PHBPAR and PLBPAR for 30 ns after negating PRDSTB.

15. The PMI master receives the first data word from 10 ns before PRDSTB is negated and until 20 ns after PRDSTB is negated.

16. The PMI master receives PHBPAR and PLBPAR from 35 ns before PRDSTB is negated and until 10 ns after PRDSTB is negated.

17. If the PMI master is executing a single word read, it negates PBCYC after PRDSTB is negated and latches the data before PRDSTB is negated. The following process is used only with double word reads.

    a. The PMI slave gates the second word data onto the bus after PRDSTB is negated.

    b. The PMI slave gates the second word PHBPAR and PLBPAR bits onto the bus within 100 ns after PRDSTB is negated.

    c. The PMI master receives the second data word within 145 ns after PRDSTB is negated.

    d. The PMI master receives the second word PHBPAR and PLBPAR bits within 120 ns after PRDSTB is negated.

    e. If the PMI master is reading two words, it negates PBCYC after latching the second word.

    f. The PMI slave removes the second word data from the bus within 50 ns after PBCYC is negated.

7.5.2 PMI Block Data In

The DATBI cycle is used to read up to 16 words of data when the PMI bus master accesses the PMI memory. The PMI bus master cannot use the DATBI cycle when accessing the I/O page or the Unibus memory. The PMI bus master detects an I/O page reference by the assertion of TBS7, and a Unibus memory reference by the assertion of PUBMEM.

The PMI bus master can only start DATBI transfers on even word boundaries. This means that address bits <1:0> must be equal to zeros. The PMI bus master cannot use the DATBI cycle to transfer across 16 word address boundaries. This means that the PMI bus master must terminate DATBI data transfers when it reaches a memory location where the address bits <4:1> are all equal to ones. The protocol used by the DATBI cycle is as follows.

1. When the PMI master assumes control of the bus, the BDAL <21:0> lines are addressed and BBS7 is asserted.

2. Each PMI slave asserts PSSEL within 45 ns after it receives the asserted BDAL <21:0> and BBS7 signals, if necessary.

3. The UBA asserts PUBMEM within 100 ns ns after it receives the asserted BDAL <21:0> and BBS7 signals, if necessary.

4. The PMI master receives PSSEL within 130 ns after gating the asserted BDAL <21:0> and BBS7 signals.

5. The PMI master receives PUBMEM within 120 ns after gating the asserted BDAL <21:0> and BBS7 signals.

6. If PSSEL is asserted and PUBMEM is negated, the PMI master proceeds as follows.

   a. The PMI master asserts PBCYC within 130 ns after gating the BDAL <21:0>, BBS7, BWTBT and PBYT signals, and after PSBFUL is negated.

   b. The PMI master continues to assert the BDAL <21:0>, BBS7, BWTBT and PBYT signals for a minimum of 40 ns and a maximum of 100 ns after it asserts PBCYC.

   c. The UBA latches PUBMEM when PBCYC is asserted.

   d. The PMI slave receives stable BDAL <21:0>, BBS7, BWTBT and PBYT signals for 65 ns (minimum) before PBCYC is asserted and for 30 ns after PBCYC is asserted.

   e. The PMI slave receives a valid PUBMEM from 10 ns (minimum) before the assertion of PBCYC and until 10 ns before PBCYC is negated.

7. If PSSEL is negated and the KDJ11-B is the PMI master, the PMI cycles are performed with the UBA responding as a slave, and follow the routine listed above.

8. If PSSEL is negated and PUBMEM is asserted, the UBA is the PMI master and it aborts the PMI cycle and does not respond as a Unibus slave.

9. The PMI master asserts PBLKM within 50 ns after PBCYC is asserted.

10. The assertion of BRPLY by the PMI slave is optional in LSI systems. Its protocol is as follows.

    a. The PMI slave asserts BRPLY after PBCYC is asserted.

    b. The PMI slave negates BRPLY within 100 ns after the negation of PRDSTB.

NOTE
In Unibus systems with PMI memory as a slave, BRPLY must be disabled at all times.

11. The PMI slave gates the data onto the bus within 125 ns after the assertion of PBCYC.

12. The PMI slave gates PHBPAR and PLBPAR parity bits after the assertion of PBCYC. These parity bits are generated only for the memory locations being cached on the KDJ11-B from the main memory.

13. The PMI slave asserts PRDSTB after the assertion of PBCYC.

14. The PMI slave negates PRDSTB within 150 ns after the assertion of PBCYC. It is negated within 75 ns after the first data word is gated on the bus and 55 ns after the PHBPAR and PLBPAR bits are gated for the first word.

15. The PMI slave maintains the data word, PHBPAR and PLBPAR for 30 ns after negating PRDSTB.

16. The PMI master receives the first data word from 10 ns before PRDSTB is negated and until 20 ns after PRDSTB is negated.

17. The PMI master receives PHBPAR and PLBPAR from 35 ns before PRDSTB is negated and until 10 ns after PRDSTB is negated.

18. The PMI slave gates the second word data onto the bus within 80 ns after PRDSTB is negated.

19. The PMI slave gates the second word PHBPAR and PLBPAR bits onto the bus within 100 ns after PRDSTB is negated.

20. The PMI master receives the second data word within 145 ns after PRDSTB is negated.

21. The PMI master receives the second word PHBPAR and PLBPAR bits within 120 ns after PRDSTB is negated.

22. If four or more data words are to be transmitted, the sequence proceeds as follows.

    a. The bus master negates PBLKM within 240 ns after the negation of PRDSTB and after latching the second word data.

    b. The PMI slave removes the second word data when PBLKM is negated.

    c. The PMI slave asserts PRDSTB after the negation of PBLKM.

    d. The PMI master asserts PBLKM 40 to 70 ns after negating it.

    e. Return to step 13 above.

    If two more data words are to be transmitted, the sequence proceeds as follows.

    a. The bus master negates PBLKM within 240 ns after the negation of PRDSTB and after latching the second word data.

    b. The PMI slave removes the second word data when PBLKM is negated.

    c. The PMI slave asserts PRDSTB after the negation of PBLKM.

    d. Return to step 13 above.

    If the last data word is to be transmitted, the sequence proceeds as follows.

    a. The bus master negates PBCYC after latching the last word data.

    b. The PMI slave removes the last word data from bus within 50 ns after PBCYC is negated.

7.5.3 PMI Data Out/Data Out Byte

The DATO and DATOB cycles are used by the PMI bus master to transfer a single word or byte to a PMI slave. The protocol used by the DATO and DATOB cycles is as follows.

1. When the PMI master assumes control of the bus, the BDAL <21:0> lines are addressed, and BBS7 and BWTBT are asserted for DATO cycles. In addition, PBYT is asserted for DATOB cycles.

2. Each PMI slave asserts PSSEL within 45 ns after it receives the asserted BDAL <21:0> and BBS7 signals, if necessary.

3. The UBA asserts PUBMEM within 100 ns ns after it receives the asserted BDAL <21:0> and BBS7 signals, if necessary.

4. The PMI master receives PSSEL within 130 ns after gating the asserted BDAL <21:0> and BBS7 signals.

5. The PMI master receives PUBMEM within 120 ns after gating the asserted BDAL <21:0> and BBS7 signals.

6. If PSSEL is asserted and PUBMEM is negated, the PMI master proceeds as follows.

   a. The PMI master asserts PBCYC within 130 ns after gating the BDAL <21:0>, BBS7, BWTBT and PBYT signals, and after PSBFUL is negated.

   b. The PMI master continues to assert the BDAL <21:0>, BBS7, BWTBT and PBYT signals for a minimum of 40 ns and a maximum of 100 ns after it asserts PBCYC.

   c. The UBA latches PUBMEM when PBCYC is asserted.

   d. The PMI slave receives stable BDAL <21:0>, BBS7, BWTBT and PBYT signals for 65 ns (minimum) before PBCYC is asserted and for 30 ns after PBCYC is asserted.

   e. The PMI slave receives a valid PUBMEM from 10 ns (minimum) before the assertion of PBCYC and until 10 ns before PBCYC is negated.

7. If PSSEL is negated and the KDJ11-B is the PMI master, the PMI cycles are performed with the UBA responding as a slave, and follow the routine listed above.

8. If PSSEL is negated and PUBMEM is asserted, the UBA is the PMI master and it aborts the PMI cycle and does not respond as a Unibus slave.

9. The PMI slave asserts BRPLY within 50 ns after the assertion of PBCYC (LSI bus systems only).

NOTE
In Unibus systems with PMI memory as a slave, BRPLY must be disabled at all times.

10. The PMI master gates the data onto the bus within 80 ns after the assertion of PBCYC.

11. The PMI master asserts PWTSTB within 75 ns after the data is placed on the bus.

12. The PMI maintains the data on the bus for 30 ns after it asserts PWTSTB.

13. The PMI slave receives the data from within 10 ns before the assertion of PWTSTB and until 20 ns after the assertion of PWTSTB.

14. The PMI slave asserts PSBFUL within 50 ns after the assertion of PWTSTB.

15. The PMI master negates PWTSTB 40 ns after asserting it.

16. The PMI master negates PBCYC after negating PWTSTB.

17. The PMI slave negates BRPLY within 300 ns (LSI systems) and cannot perform another PMI or LSI bus cycle during this period.

7.6 PMI INTERRUPT PROTOCOL

The PMI interrupt protocol consists of the interrupt request, granting the interrupt and fetching the interrupt vector to service the interrupt. The LSI requirements for an interrupt are defined in Chapter 6. The Unibus requirements for the request and grant are described in Paragraph 7.4.2. The transfer of the interrupt vector from the requesting Unibus device to the KDJ11-B requires a combination of the Unibus and LSI bus protocols as follows.

1. Once the requesting device is the bus master, it places the interrupt vector on the Unibus after it asserts BBSY.

2. The requesting device asserts INTR after the vector data is on the Unibus.

3. The UBA is the PMI bus master and asserts BRPLY after it receives INTR on the Unibus.

4. The UBA receives the interrupt vector and places it on the BDAL data lines within 75 ns after BRPLY is asserted.

5. The UBA latches the interrupt vector within 75 ns after INTR is asserted and then asserts SSYN on the Unibus.

6. The requesting device is the bus master and it removes the vector after it receives SSYN. It also negates INTR at this time.

7. The requesting device negates BBSY after negating INTR to relinquish bus mastership.

8. The KDJ11-B latches the vector data within 200 ns after the UBA-asserted BRPLY.

9. The KDJ11-B negates BDIN and BIAKO after it latches the vector data.

10. The UBA negates BRPLY after BIAKO is negated.

7.7 PMI POWER-UP/POWER-DOWN

The power-up/power-down protocol for the PMI bus in an LSI system is described in Chapter 6. The protocol used in a Unibus system is similar to that of the LSI system. The primary difference is that in an LSI system, the BPOK signal is negated by the power supply 3 ms after it is asserted, and in the Unibus system, the KDJ11-B must ignore the assertion of AC LO for a minimum of 2 ms after it is asserted. These delays allow the system software enough time to prepare for a power-down before the KDJ11-B can execute the power-down sequence.

In the Unibus system, the KDJ11-B receives DC LO as the DCOK signal, and the BPOK signal is isolated from AC LO by the UBA. The PMI memory interfaces to AC LO, but not to BPOK on the LSI bus. Therefore, when a Unibus device asserts AC LO to the UBA, it asserts BPOK for a mimimum of 2 ms before it allows AC LO to negate BPOK.

CHAPTER 8
ADDRESSING MODES

8.1 INTRODUCTION

The KDJ11-B utilizes the six addressing modes described below with the base instruction set to control or program the operations executed by the microprocessor. Included in this chapter are specific examples of how these addressing modes are used.

- Single-Operand Addressing – One part of the instruction word specifies the registers; the other part provides information for locating the operand.
- Double-Operand Addressing – One part of the instruction word specifies the registers; the remaining parts provide information for locating two operands.
- Direct Addressing – The operand is the content of the selected register.
- Deferred (Indirect) Addressing – The contents of the selected register is the address of the operand.
- Use of the PC as a General Purpose Register – The PC is different from other general purpose registers in one important respect. Whenever the processor retrieves an instruction, it automatically advances the PC by 2. By combining this automatic advancement of the PC with four of the basic addressing modes, the four special PC modes – immediate, absolute, relative, and relative-deferred – are created.
- Use of the General Purpose Registers as an SP – General purpose registers can be used for stack operations.

8.2 ADDRESSING MODES

Data stored in memory must be accessed and manipulated. Data handling is specified by a KDJ11-B instruction (MOV, ADD, etc.), and usually includes the following.

- The function to be performed (operation code)
- The general purpose register to be used when locating the source operand, and/or destination operand (where required)
- The addressing mode, which specifies how the selected registers are to be used

A large portion of the data handled by a computer is structured (character strings, arrays, lists, etc.). The KDJ11-B addressing modes provide for efficient and flexible handling of structured data.

A general purpose register may be used with an instruction in any of the following ways.

1. As an accumulator – The data to be manipulated resides in the register.

2. As a pointer – The contents of the register is the address of an operand, rather than the operand itself.

3. As a pointer that automatically steps through memory locations – Automatically stepping forward through consecutive locations is known as autoincrement addressing; automatically stepping backward is known as autodecrement addressing. These modes are particularly useful for processing tabular or array data.

4. As an index register – In this instance, the contents of the register and the word following the instruction are summed to produce the address of the operand. This allows easy access to variable entries in a list.

An important KDJ11-B feature that should be considered with the addressing modes is the following register arrangement.

- Two sets of six general purpose registers (R0–R5 and R0'–R5')
- A hardware SP register (R6) for each processor mode (kernel, supervisor, user)
- A PC register (R7)

Registers R0–R5 and R0'–R5' are not dedicated to any specific function. Their uses are determined by decoded instructions and include the following.

- They can be used for operand storage. For example, the contents of two registers can be added and stored in another register.
- They can contain the address of an operand or serve as pointers to the address of an operand.
- They can be used for the autoincrement or autodecrement features.
- They can be used as index registers for convenient data and program access.

The KDJ11-B also has instruction addressing mode combinations that facilitate temporary data storage structures. These can be used for convenient handling of data that must be accessed frequently. This is known as stack manipulation. The register that keeps track of stack manipulation is called the stack pointer, or SP. Any register can be used as an SP under program control. However, certain instructions associated with subroutine linkage and interrupt service automatically use R6 as a hardware stack pointer. For this reason, R6 is frequently referred to as the SP. The SP functions include the following.

- The SP keeps track of the latest entry on the stack.
- The SP moves down as items are added to the stack and moves up as items are removed. Therefore, the SP always points to the top of the stack.
- The hardware stack is used during trap or interrupt handling to store information, allowing an orderly return to the interrupted program.

R7 is used by the processor as its PC. It is recommended that R7 not be used as an SP or accumulator. Whenever an instruction is fetched from memory, the PC is automatically incremented by two to point to the next instruction word.

8.2.1 Single-Operand Addressing

The instruction format for all single-operand instructions (such as CLR, INC, TST) is shown in Figure 8-1. Bits <15:6> specify the operation code that defines the type of instruction to be executed. Bits <5:0> form a 6-bit field called the destination address field. The destination address field consists of two subfields, as follows.

- Bits <5:3> specify the destination mode. Bit 3 is set to indicate deferred (indirect) addressing.
- Bits <2:0> specify which of the eight general purpose registers is to be referenced by this instruction word.

8.2.2 Double-Operand Addressing

Operations that employ two operands (such as ADD, SUB, MOV, and CMP) are handled by instructions that specify two addresses. The first operand is called the source operand; the second is called the destination operand. Bit assignments in the source and destination address fields may specify different modes and different registers. The instruction format for the double-operand instruction is shown in Figure 8-2.

The source address field is used to select the source operand (the first operand). The destination is used similarly, and locates the second operand and the result. For example, the instruction ADD A, B adds the contents (source operand) of location A to the contents (destination operand) of location B. After execution, B contains the result of the addition and the contents of A is unchanged.

Examples throughout this chapter use the sample KDJ11-B instructions given in Table 8-1. (A complete list of KDJ11-B instructions appears in Chapter 9, Table 9-1).

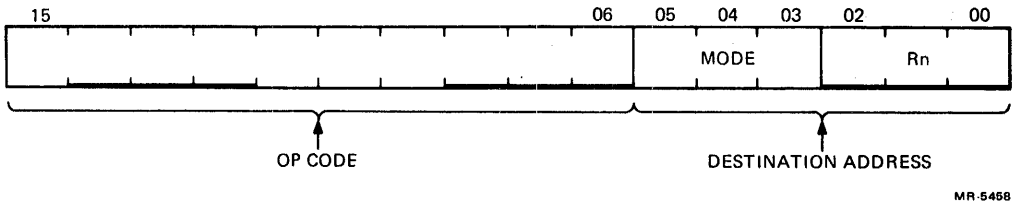


Figure 8-1 Single-Operand Addressing

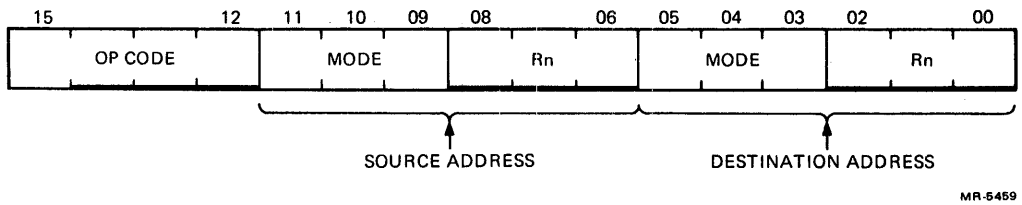


Figure 8-2 Double-Operand Addressing

Table 8-1 Sample KDJ11-B Instructions

Mnemonic	Description	Octal Code*
CLR	Clear – Zero the specified destination.	0050DD
CLRB	Clear byte – Zero the byte in the specified destination.	1050DD
INC	Increment – Add one to the contents of the destination.	0052DD
INCB	Increment byte – Add one to the contents of the destination byte.	1052DD
COM	Complement – Replace the contents of the destination by its logical complement; each 0 bit is set and each 1 bit is cleared.	0051DD
COMB	Complement byte – Replace the contents of the destination byte by its logical complement; each 0 bit is set and each 1 bit is cleared.	1051DD
ADD	Add – Add the source operand to the destination operand and store the result at the destination address.	06SSDD

* DD = Destination field (six bits)
SS = Source field (six bits)

8.2.3 Direct Addressing

The following summarizes the four basic modes used with direct addressing. These direct modes are illustrated in Figures 8-3 to 8-6.

Mode	Name	Assembler Syntax	Function
0	Register	Rn	Register contains operand.

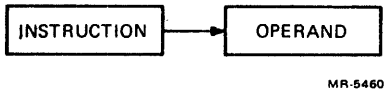


Figure 8-3 Mode 0, Register

Mode	Name	Assembler Syntax	Function
2	Autoincrement	(Rn)+	Register is used as a pointer to sequential data and then is incremented.

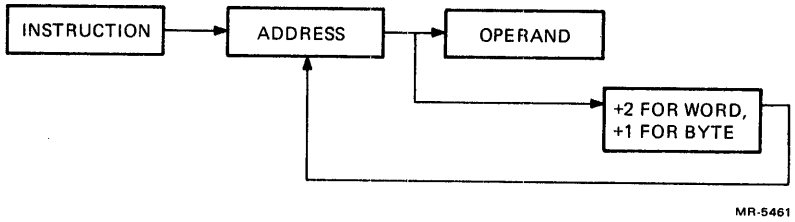


Figure 8-4 Mode 2, Autoincrement

Mode	Name	Assembler Syntax	Function
4	Autodecrement	-(Rn)	Register is decremented and then used as a pointer.

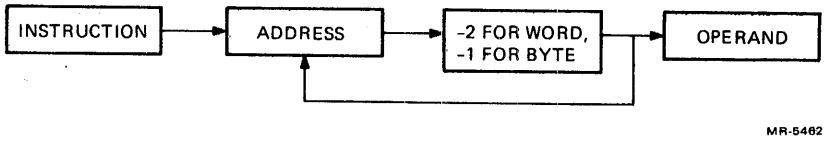


Figure 8-5 Mode 4, Autodecrement

Mode	Name	Assembler Syntax	Function
6	Index	X(Rn)	Value X is added to (Rn) to produce address of operand. Neither X nor (Rn) is modified.

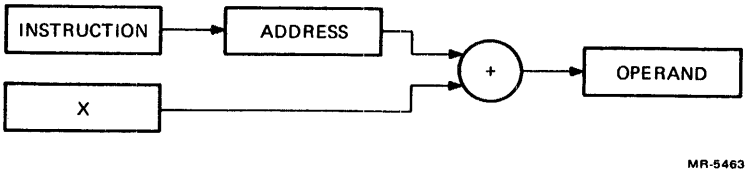


Figure 8-6 Mode 6, Index

8.2.3.1 Register Mode – With register mode (mode 0) any of the general registers may be used as simple accumulators, with the operand contained in the selected register. Since they are hardware registers (within the processor), the general registers operate at high speeds and provide speed advantages when used for operating on frequently accessed variables. The assembler interprets and assembles instructions of the form OPR Rn as register mode operations. Rn represents a general register name or number and OPR is used to represent a general instruction mnemonic. Assembler syntax requires that a general register be defined as follows.

R0 = %0 (% sign indicates register definition)
R1 = %1
R2 = %2, etc.

Registers are typically referred to by name as R0, R1, R2, R3, R4, R5, R6, and R7. However, R6 and R7 are also referred to as SP and PC, respectively. Three register mode operations are illustrated in Figures 8-7 to 8-9.

Register Mode Examples:

Symbolic	**Octal Code**	**Instruction Name**
INC R3	005203	Increment

Operation: Add one to the contents of R3.

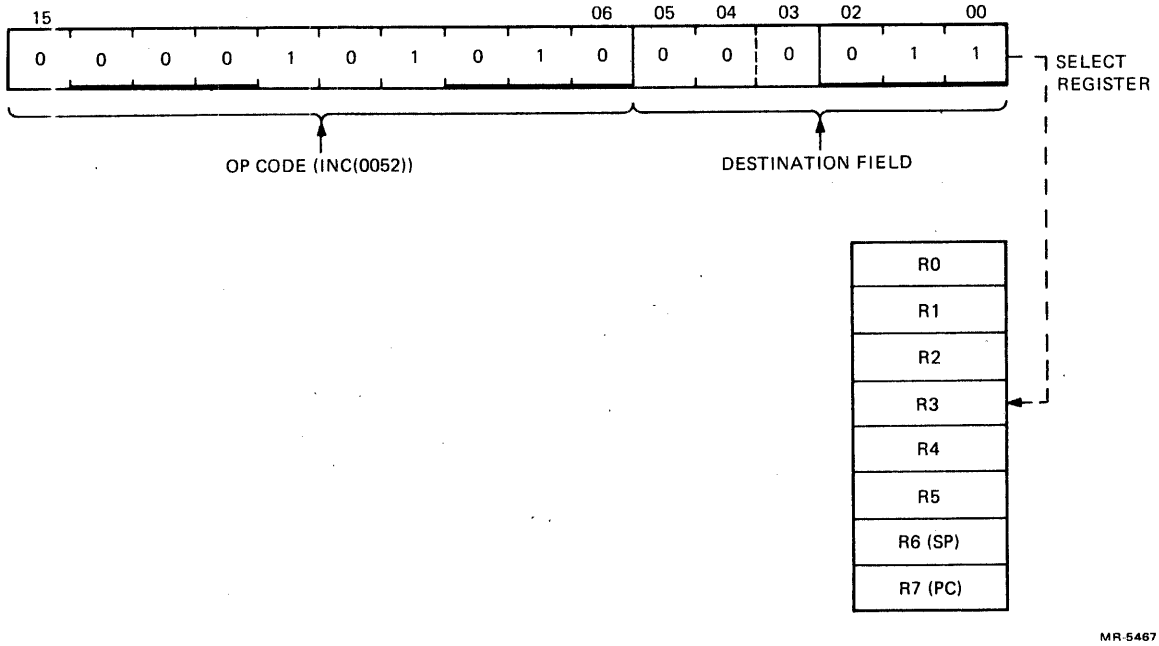


Figure 8-7 INC R3 Increment

Symbolic	Octal Code	Instruction Name
ADD R2, R4	060204	Add

Operation: Add the contents of R2 to the contents of R4.

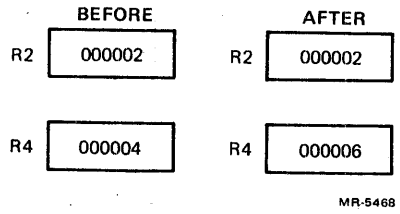


Figure 8-8 ADD R2,R4 Add

Symbolic	Octal Code	Instruction Name
COMB R4	105104	Complement byte

Operation: 1's complement bits <7:0> (byte) in R4. When general registers are used, byte instructions (with the exception of MOVB) operate only on bits <7:0>, that is, byte 0 of the register. MOVB to a register, unique for byte instructions, extends the most significant bit of the low-order byte (sign extension) into the high byte of the selected register. Otherwise, MOVB operates on bytes the same way MOV operates on words.

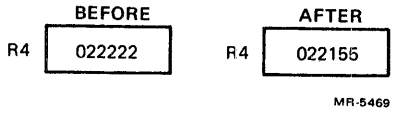


Figure 8-9 COMB R4 Complement Byte

8.2.3.2 Autoincrement Mode [OPR (Rn)+] – This mode (mode 2) provides for automatic stepping of a pointer through sequential elements of a table of operands. It assumes the contents of the selected general purpose register to be the address of the operand. Contents of registers are stepped (by one for byte instructions, by two for word instructions, always by two for R6 and R7) to address the next sequential location. The autoincrement mode is especially useful for array processing and stack processing. It accesses an element of a table and then steps the pointer to address the next operand in the table. Although autoincrement mode is most useful for table handling, it is completely general and may be used for a variety of purposes. Three autoincrement mode operations are illustrated in Figures 8-10 to 8-12.

Autoincrement Mode Examples:

Symbolic	Octal Code	Instruction Name
CLR (R5)+	005025	Clear

Operation: Use contents of R5 as the address of the operand. Clear the selected operand and then increment the contents of R5 by two.

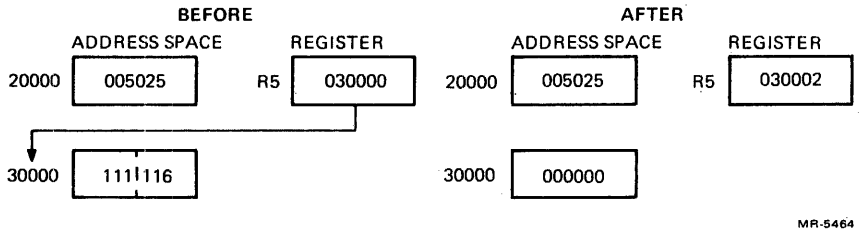


Figure 8-10 CLR (R5)+ Clear

Symbolic	Octal Code	Instruction Name
CLRB (R5)+	105025	Clear byte

Operation: Use contents of R5 as the address of the operand. Clear the selected byte operand and then increment the contents of R5 by one.

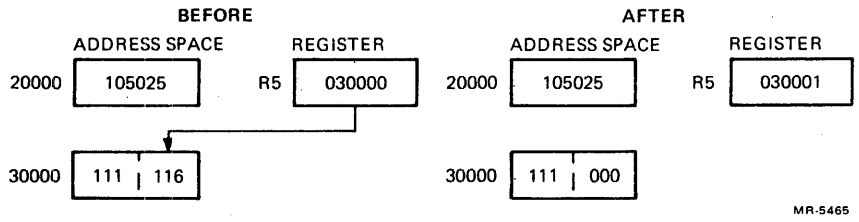


Figure 8-11 CLRB (R5)+ Clear Byte

Symbolic	Octal Code	Instruction Name
ADD (R2)+,R4	062204	Add

Operation: The contents of R2 is used as the address of the operand, which is added to the contents of R4. R2 is then incremented by two.

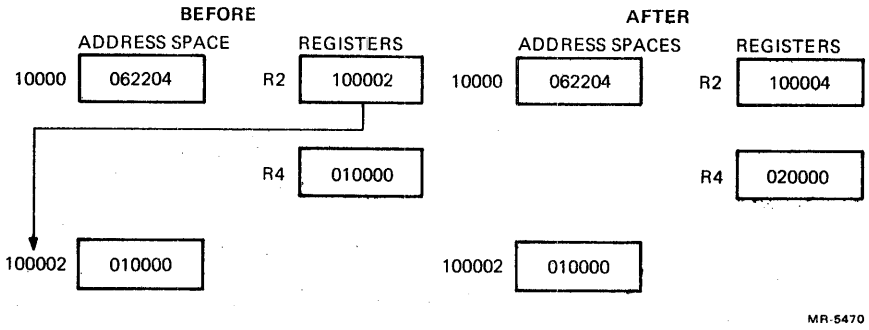


Figure 8-12 ADD (R2)+,R4 Add

8.2.3.3 Autodecrement Mode [OPR –(Rn)] – This mode (mode 4) is useful for processing data in a list in reverse direction. The contents of the selected general purpose register is decremented (by one for byte instructions, by two for word instructions) and then used as the address of the operand. The postincrement and predecrement features on the KDJ11-B are intended to facilitate hardware/software stack operations. Three autodecrement mode operations are illustrated in Figures 8-13 to 8-15.

Autodecrement Mode Examples:

Symbolic	Octal Code	Instruction Name
INC –(R0)	005240	Increment

Operation: The contents of R0 is decremented by two and used as the address of the operand. The operand is incremented by one.

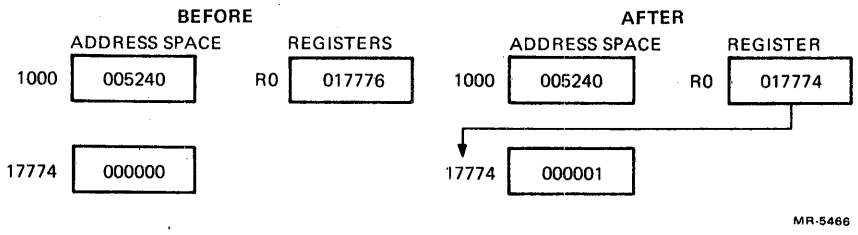


Figure 8-13 INC –(R0) Increment

Symbolic	Octal Code	Instruction Name
INCB –(R0)	105240	Increment byte

Operation: The contents of R0 is decremented by one and then used as the address of the operand. The operand byte is increased by one.

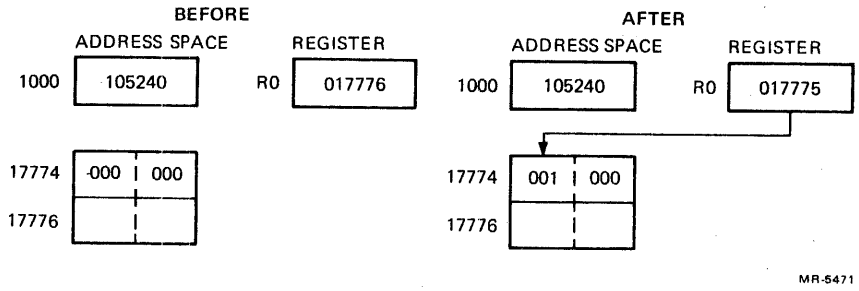


Figure 8-14 INCB –(R0) Increment Byte

Symbolic	Octal Code	Instruction Name
ADD –(R3),R0	064300	Add

Operation: The contents of R3 is decremented by two and then used as a pointer to an operand (source), which is added to the contents of R0 (destination operand).

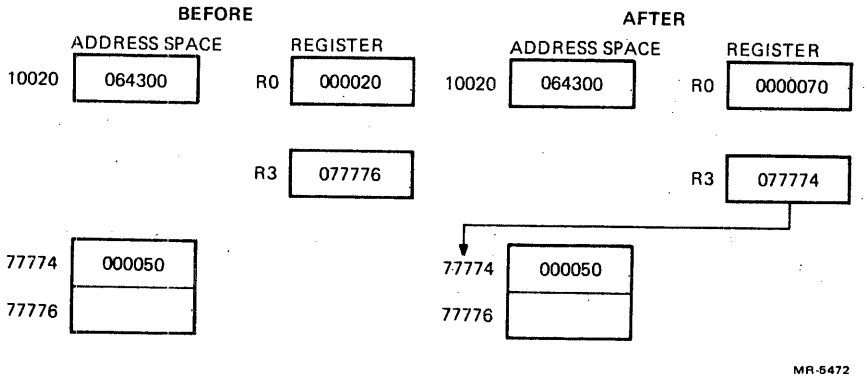


Figure 8-15 ADD –(R3),R0 Add

8.2.3.4 Index Mode [OPR X(Rn)] – In this mode (mode 6), the contents of the selected general purpose register and an index word following the instruction word are summed to form the address of the operand. The contents of the selected register may be used as a base for calculating a series of addresses, thus allowing random access to elements of data structures. The selected register can then be modified by a program to access data in the table. Index addressing instructions are of the form OPR X(Rn), where X is the indexed word located in the memory location following the instruction word, and Rn is the selected general purpose register. Three index mode operations are illustrated in Figures 8-16 to 8-18.

Index Mode Examples:

Symbolic	Octal Code	Instruction Name
CLR 200(R4)	005064 000200	Clear

Operation: The address of the operand is determined by adding 200 to the contents of R4. The operand location is then cleared.

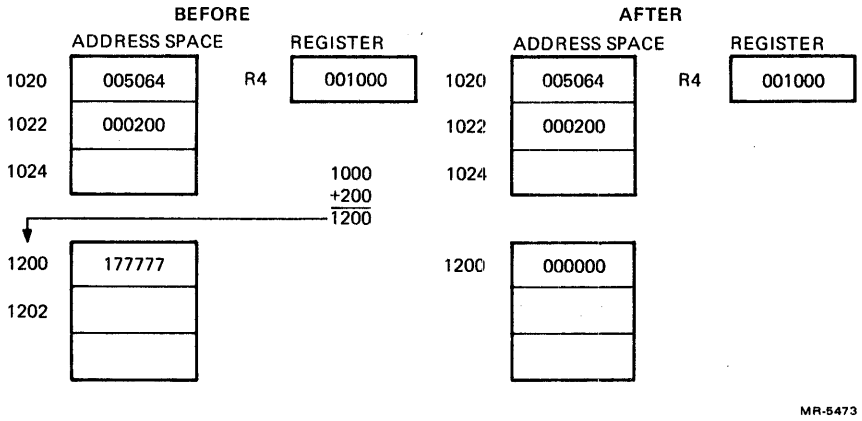


Figure 8-16 CLR 200(R4) Clear

Symbolic	Octal Code	Instruction Name
COMB 200(R1)	105161 000200	Complement byte

Operation: The contents of a location determined by adding 200 to the contents of R1 is 1's complemented, that is, logically complemented.

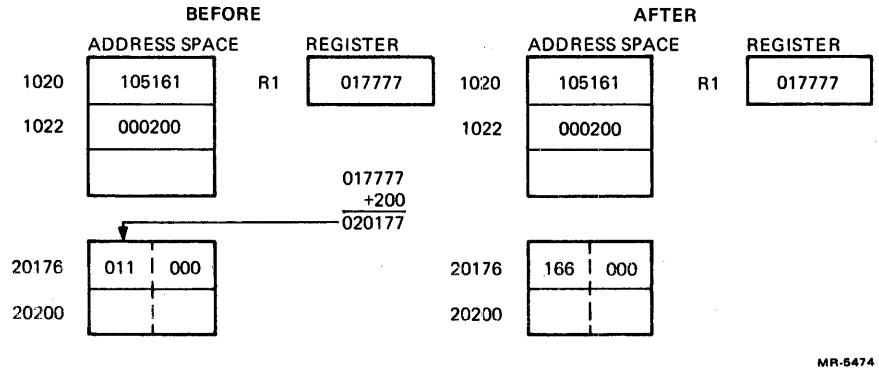


Figure 8-17 COMB 200(R1) Complement Byte

Symbolic	Octal Code	Instruction Name
ADD 30(R2),20(R5)	066265 000030 000020	Add

Operation: The contents of a location determined by adding 30 to the contents of R2 is added to the contents of a location determined by adding 20 to the contents of R5. The result is stored at the destination address, that is, 20(R5).

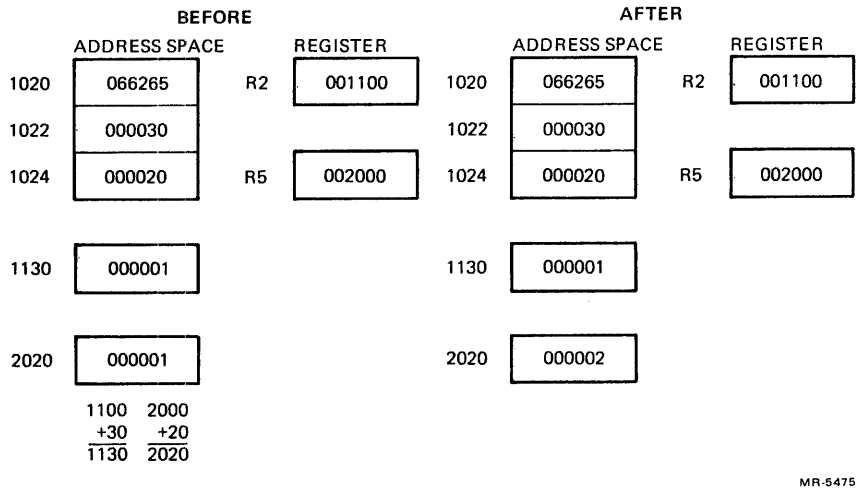


Figure 8-18 ADD 30(R2),20(R5) Add

8.2.4 Deferred (Indirect) Addressing

The four basic modes may also be used with deferred addressing. While in register mode the operand is the contents of the selected register, in register-deferred mode the contents of the selected register is the address of the operand.

In the three other deferred modes, the contents of the register selects the address of the operand rather than the operand itself. These modes are therefore used when a table consists of addresses rather than operands. The assembler syntax for indicating deferred addressing is @, or () when this is not ambiguous.

The following summarizes the deferred versions of the basic modes. These deferred modes are illustrated in Figures 8-19 to 8-22.

Mode	Name	Assembler Syntax	Function
1	Register-deferred	@Rn or (Rn)	Register contains the address of the operand.

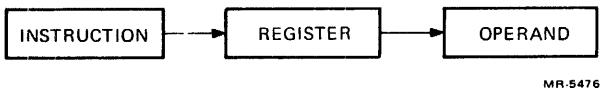


Figure 8-19 Mode 1, Register-Deferred

Mode	Name	Assembler Syntax	Function
3	Autoincrement-deferred	@(Rn)+	Register is first used as a pointer to a word containing the address of the operand, and then is incremented (always by two, even for byte instructions).

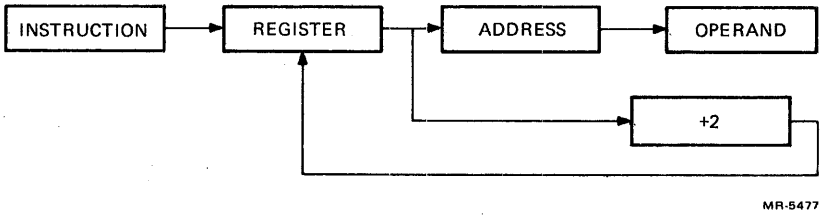


Figure 8-20 Mode 3, Autoincrement-Deferred

Mode	Name	Assembler Syntax	Function
5	Autodecrement-deferred	@-(Rn)	Register is decremented (always by two, even for byte instructions) and then used as a pointer to a word containing the address of the operand.

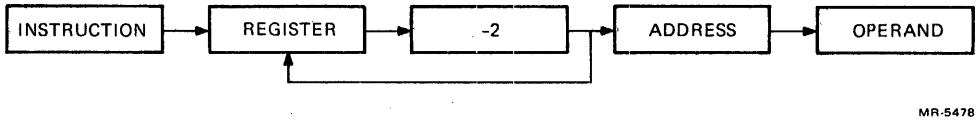


Figure 8-21 Mode 5, Autodecrement-Deferred

Mode	Name	Assembler Syntax	Function
7	Index-deferred	@X(Rn)	Value X (stored in a word following the instruction) and (Rn) are added. The sum is used as a pointer to a word containing the address of the operand. Neither X nor (Rn) is modified.

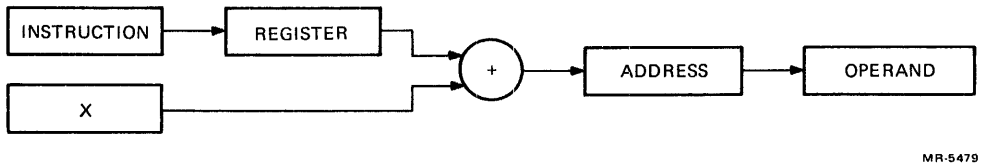


Figure 8-22 Mode 7, Index-Deferred

The following examples (Figures 8-23 to 8-26) further illustrate use of the deferred modes.

Register-Deferred Mode Example:

Symbolic	Octal Code	Instruction Name
CLR @R5	005015	Clear

Operation: The contents of a location specified in R5 is cleared.

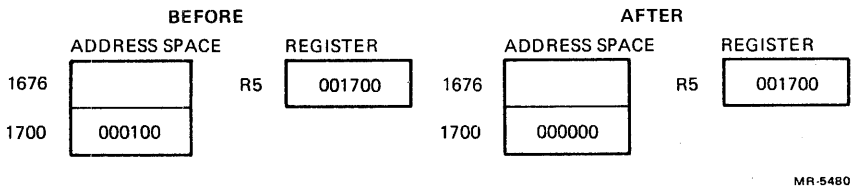


Figure 8-23 CLR @R5 Clear

Autoincrement-Deferred Mode Example:

Symbolic	Octal Code	Instruction Name
INC @(R2)+	005232	Increment

Operation: The contents of R2 is used as the address of the address of the operand. The operand is increased by one; the contents of R2 is incremented by two.

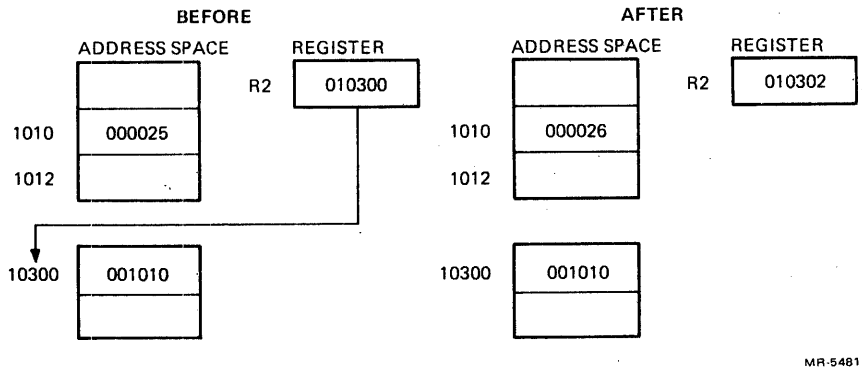


Figure 8-24 INC @(R2)+ Increment

Autodecrement-Deferred Mode Example:

Symbolic	Octal Code	Instruction Name
COM @–(R0)	005150	Complement

Operation: The contents of R0 is decremented by two and then used as the address of the address of the operand. The operand is 1's complemented, that is, logically complemented.

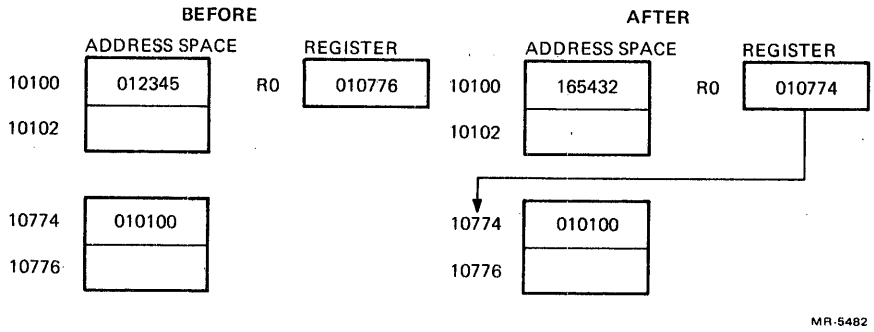


Figure 8-25 COM @–(R0) Complement

Index-Deferred Mode Example:

Symbolic	Octal Code	Instruction Name
ADD @1000(R2),R1	067201 001000	Add

Operation: Location 1000 and the contents of R2 are summed to produce the address of the address of the source operand, the contents of which are added to the contents of R1. The result is stored in R1.

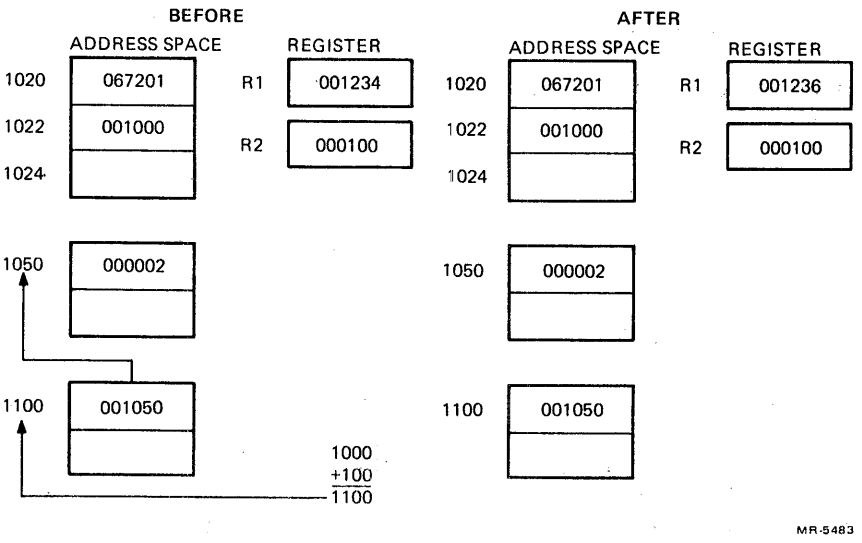


Figure 8-26 ADD @1000(R2),R1 Add

8.2.5 Use of the PC as a General Purpose Register

Although R7 is a general purpose register, it doubles in function as the PC for the KDJ11-B. Whenever the processor uses the PC to acquire a word from memory, the PC is automatically incremented by two to contain the address of the next word of the instruction being executed or the address of the next instruction to be executed. (When the program uses the PC to locate byte data, the PC is still incremented by two.)

The PC responds to all the standard KDJ11-B addressing modes. However, with four of these modes the PC can provide advantages for handling Position-Independent Code (PIC) and unstructured data. When utilizing the PC, these modes are termed immediate, absolute (or immediate-deferred), relative, and relative-deferred. They are summarized in the following chart.

Mode	Name	Assembler Syntax	Function
2	Immediate	#n	Operand follows instruction.
3	Absolute	@#A	Absolute address of operand follows instruction.
6	Relative	A	Relative address (index value) follows the instruction.
7	Relative-deferred	@A	Index value (stored in the word after the instruction) is the relative address for the address of the operand.

When a standard program is available for different users, it is often helpful to be able to load it into different areas of memory and run it in those areas. The KDJ11-B can accomplish the relocation of a program very efficiently through the use of PIC, which is written by using the PC addressing modes. If an instruction and its operands are moved in such a way that the relative distance between them is not altered, the same offset relative to the PC can be used in all positions in memory. Thus, PIC usually references locations relative to the current location.

The PC also greatly facilitates the handling of unstructured data. This is particularly true of the immediate and relative modes.

8.2.5.1 Immediate Mode [OPR #n,DD] – With the PC, immediate mode (mode 2) is equivalent in use to the autoincrement mode. It provides speed improvements for accessing constant operands by including the constant in the memory location immediately following the instruction word. An immediate mode operation is illustrated in Figure 8-27.

Immediate Mode Example:

Symbolic	Octal Code	Instruction Name
ADD #10,R0	062700 000010	Add

Operation: The value 10 is located in the second word of the instruction and is added to the contents of R0. Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus, the PC is used as a pointer to fetch the operand (the second word of the instruction) before it is incremented by two to point to the next instruction.

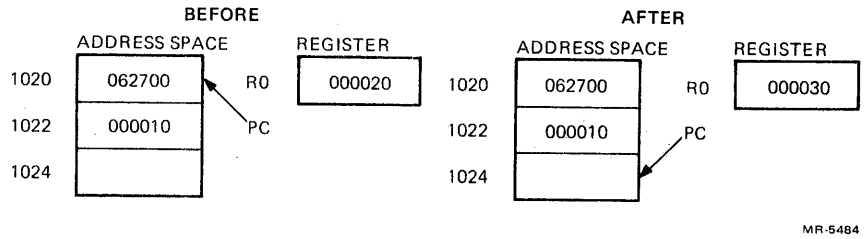


Figure 8-27 ADD #10,R0 Add

8.2.5.2 Absolute Mode [OPR @#A] – Using the PC, this mode (mode 3) is the equivalent of the immediate-deferred or autoincrement-deferred modes. The contents of the location following the instruction are taken as the address of the operand. Immediate data is interpreted as an absolute address, that is, an address that remains constant no matter where in memory the assembled instruction occurs. Two absolute mode operations are illustrated in Figures 8-28 and 8-29.

Absolute Mode Examples:

Symbolic	Octal Code	Instruction Name
CLR @#1100	005037 001100	Clear

Operation: Clear the contents of location 1100.

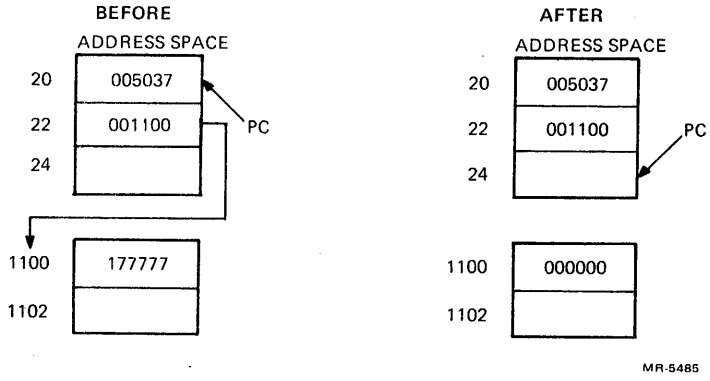


Figure 8-28 CLR @ #1100 Clear

Symbolic	Octal Code	Instruction Name
ADD @#2000,R3	063703 002000	Add

Operation: Add contents of location 2000 to R3.

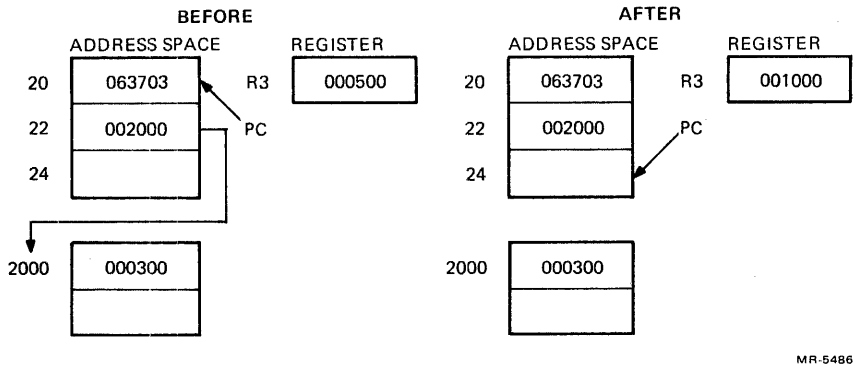


Figure 8-29 ADD @ #2000 Add

8.2.5.3 Relative Addressing Mode [OPR A or OPR X(PC)] – Usi ng R7, this mode (mode 6) is assembled as index mode. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand, but the number which, when added to the PC, becomes the address of the operand. This mode is useful for writing PIC since the location referenced is always fixed relative to the PC. When instructions are to be relocated, the operand is moved by the same amount. The instruction OPR X(PC) is interpreted as 'X is the location of A relative to the PC.' A relative mode operation is illustrated in Figure 8-30.

Relative Addressing Mode Example:

Symbolic	Octal Code	Instruction Name
INC A	005267 000054	Increment

Operation: To increment location A, the contents of the memory location immediately following the instruction word is added to the PC to produce address A. The contents of A is increased by one.

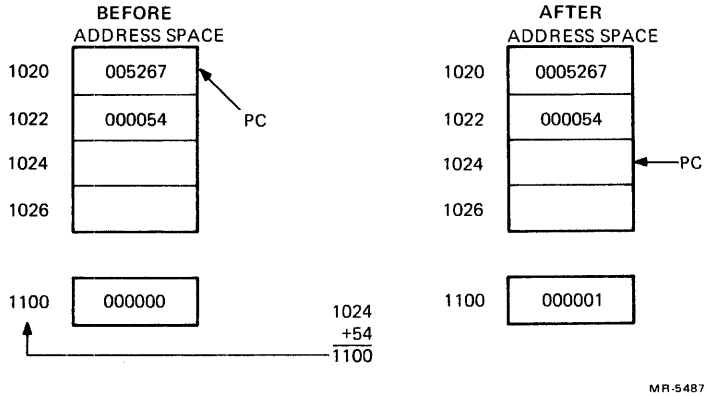


Figure 8-30 INC A Increment

8.2.5.4 Relative-Deferred Addressing Mode [OPR @A or OPR @X(PC)] – This mode (mode 7) is similar to relative mode, except that the second word of the instruction, when added to the PC, contains the address of the address of the operand, rather than the address of the operand. The instruction OPR @X(PC) is interpreted as 'X is the location containing the address of A, relative to the PC.' A relative-deferred mode operation is illustrated in Figure 8-31.

Relative-Deferred Addressing Mode Example:

Symbolic	Octal Code	Instruction Name
CLR @A	005077 000020	Clear

Operation: Add second word of instruction to the updated PC to produce the address of the address of the operand. Clear the operand.

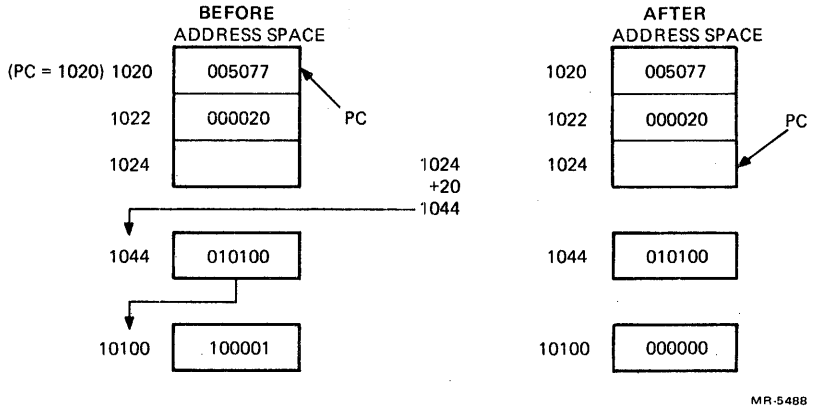


Figure 8-31 CLR @A Clear

8.2.6 Use of the General Purpose Registers as a Stack Pointer

The processor SP (R6) is, in most cases, the general register used for the stack operations related to program nesting. Autodecrement using R6 'pushes' data onto the stack, and autoincrement using R6 'pops' data off the stack. Since the SP is used by the processor for interrupt handling, it has a special attribute: Autoincrements and autodecrements are always done in steps of two. Byte operations using the SP in this way leave odd addresses (upper bytes) unmodified.

CHAPTER 9
BASE INSTRUCTION SET

9.1 INSTRUCTION SET

This chapter describes the KDJ11-B instruction set. The explanation of each instruction includes the instruction mnemonic, octal code, binary code, a diagram showing the format of the instruction, a symbolic notation describing its execution and effect on the condition codes, a description, special comments, and examples. Each explanation is headed by its mnemonic. When the word instruction has a byte equivalent, the byte mnemonic also appears.

The instruction set is listed by functional groups in Paragraph 9.4, and an alphabetical list is given in Table 9-1 below.

Table 9-1 Instruction Set

Mnemonic	Instruction	Op Code
ADC(B)	Add carry	■055DD
ADD	Add source to destination	06SSDD
ASH	Arithmetic shift	072RSS
ASHC	Arithmetic shift combined	073RSS
ASL(B)	Arithmetic shift left	■063DD
ASR(B)	Arithmetic shift right	■062DD
BCC	Branch if carry is clear	103000
BCS	Branch if carry is set	103400
BEQ	Branch if equal (to zero)	001400
BGE	Branch if greater than or equal (to zero)	002000
BGT	Branch if greater than (zero)	003000
BHI	Branch if higher	101000
BHIS	Branch if higher or same	103000
BIC(B)	Bit clear	■4SSDD
BIS(B)	Bit set	■5SSDD
BIT(B)	Bit test	■3SSDD
BLE	Branch if less than or equal (to zero)	003400
BLO	Branch if lower	103400
BLOS	Branch if lower or same	101400
BLT	Branch if less than (zero)	

Table 9-1 Instruction Set (Cont)

Mnemonic	Instruction	Op Code
BMI	Branch if minus	100400
BNE	Branch if not equal (to zero)	001000
BPL	Branch if plus	100000
BPT	Breakpoint trap	000003
BR	Branch (unconditional)	000400
BVC	Branch if overflow is clear	102000
BVS	Branch if overflow is set	102400
CCC	Clear all CC bits	000257
CLC	Clear C	000241
CLN	Clear N	000250
CLR(B)	Clear destination	■050DD
CLV	Clear V	000242
CLZ	Clear Z	000244
CMP(B)	Compare source to destination	■2SSDD
COM(B)	Complement destination	■051DD
CSM	Call to supervisor mode	0070DD
DEC(B)	Decrement destination	■053DD
DIV	Divide	071RSS
EMT	Emulator trap	104000–104377
HALT	Halt	000000
IOT	Input/output trap	000004
INC(B)	Increment destination	■052DD
JMP	Jump	0001DD
JSR	Jump to subroutine	004RDD
MARK	Mark	0064NN
MFPD	Move from previous data space	0065SS
MFPI	Move from previous instruction space	1065SS
MFPS	Move byte from PS	1067DD
MFPT	Move processor type	000007
MOV(B)	Move source to destination	■1SSDD
MTPD	Move to previous data space	1066SS
MTPI	Move to previous instruction space	0066SS
MTPS	Move byte to PS	1064SS
MUL	Multiply	070RSS
NEG(B)	Negate destination	■054DD
NOP	No operation	000240
RESET	Reset external bus	000005
ROL(B)	Rotate left	■061DD
ROR(B)	Rotate right	■060DD
RTI	Return from interrupt	000002

Table 9-1 Instruction Set (Cont)

Mnemonic	Instruction	Op Code
RTS	Return from subroutine	00020R
RTT	Return from interrupt	000006
SBC(B)	Subtract carry	■056DD
SCC	Set all CC bits	000277
SEC	Set C	000261
SEN	Set N	000270
SEV	Set V	000262
SEZ	Set Z	000264
SOB	Subtract one and branch (if ≠ 0)	077R00
SPL	Set priority level	00023N
SUB	Subtract source from destination	16SSDD
SWAB	Swap bytes	0003DD
SXT	Sign extend	0067DD
TRAP	Trap	104400–104777
TST(B)	Test destination	■057DD
TSTSET	Test destination, set low bit	0072DD
WAIT	Wait for interrupt	000001
WRTLCK	Read/lock destination	0073DD
XOR	Exclusive OR	074RDD

The diagram that accompanies each instruction shows the octal op code, binary op code, and bit assignments. Notice that in byte instructions, the most significant bit (bit 15) is always a one.

Symbols:

( ) = contents of

SS or src = source address

DD or dst = destination address

loc = location

← = becomes

↑ = 'is popped from stack'

↓ = 'is pushed onto stack'

∧ = Boolean AND

∨ = Boolean OR

⊻ = exclusive OR

~ = Boolean not

REG or R = register

B = byte

■ = 0 for word, 1 for byte

, = concatenated

9.2 INSTRUCTION FORMATS

The following formats include all instructions used in the KDJ11-B. Refer to individual instructions for more detailed information.

1. Single-Operand Group: (Figure 9-1)
   CLR, CLRB, COM, COMB, INC, INCB, DEC, DECB, NEG, NEGB, ADC, ADCB, SBC, SBCB, TST, TSTB, ROR, RORB, ROL, ROLB, ASR, ASRB, ASL, ASLB, JMP, SWAB, MFPS, MTPS, SXT, TSTSET, WRTLCK

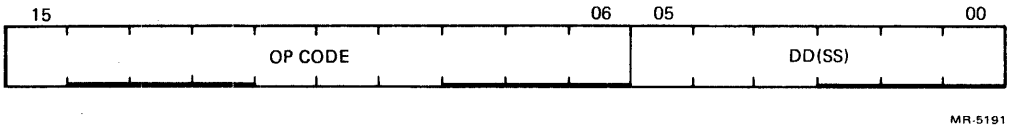


Figure 9-1 Single-Operand Group

2. Double-Operand Groups:

   a. Group 1: (Figure 9-2)
      BIT, BITB, BIC, BICB, BIS, BISB, ADD, SUB, MOV, MOVB, CMP, CMPB

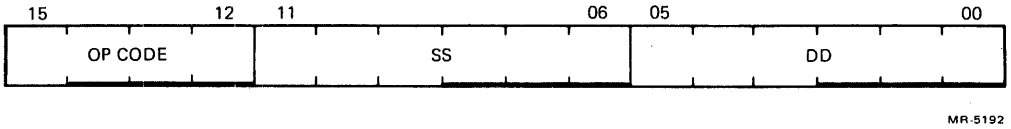


Figure 9-2 Double-Operand Group 1

   b. Group 2: (Figure 9-3)
      ASH, ASHC, DIV, MUL, XOR

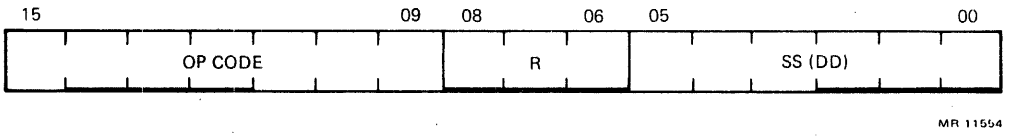


Figure 9-3 Double-Operand Group 2

3. Program Control Groups:

   a. Branch (all branch instructions) (Figure 9-4)

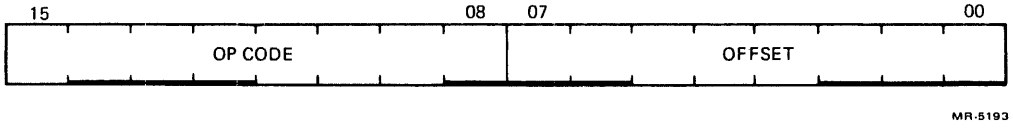


Figure 9-4 Program Control Group Branch

b. Jump (JMP) (Figure 9-5)

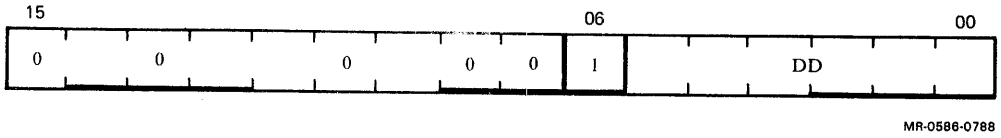


Figure 9-5 Program Control Group JMP

c. Jump to Subroutine (JSR) (Figure 9-6)

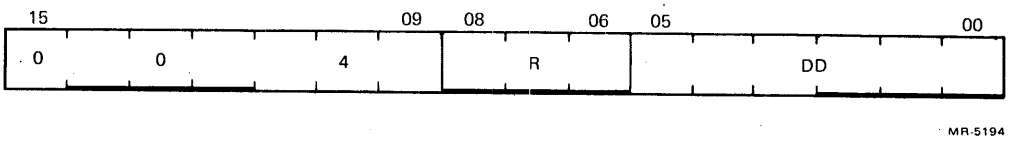


Figure 9-6 Program Control Group JSR

d. Subroutine Return (RTS) (Figure 9-7)

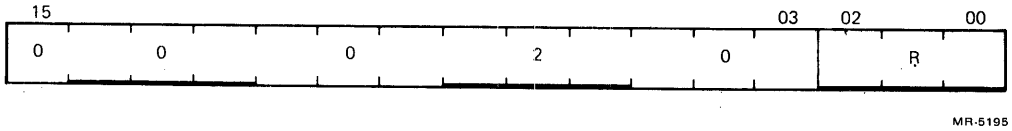


Figure 9-7 Program Control Group RTS

e. Traps (breakpoint, IOT, EMT, TRAP, BPT) (Figure 9-8)

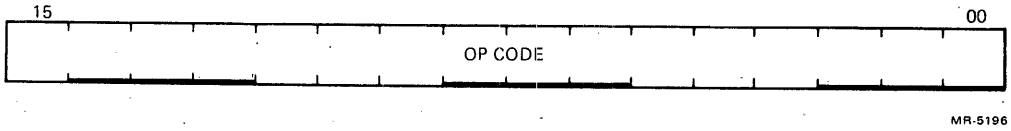


Figure 9-8 Program Control Group Traps

f. Subtract 1 and Branch (if = 0) (SOB) (Figure 9-9)

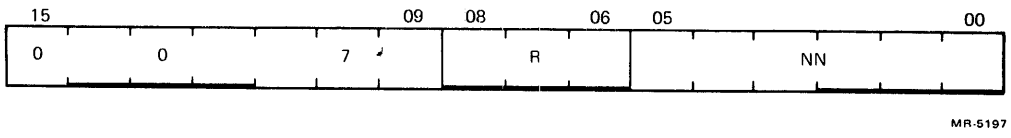


Figure 9-9 Program Control Group Subtract

g. Mark (Figure 9-10)

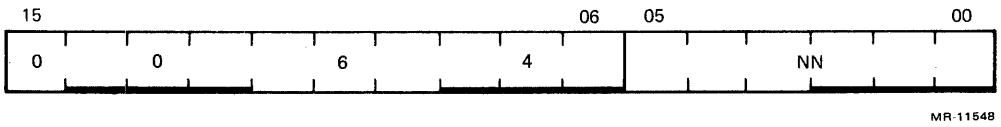


Figure 9-10 Mark

h. Call to Supervisor Mode (CSM) (Figure 9-11)

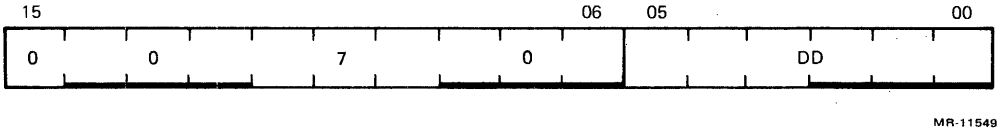


Figure 9-11 Call to Supervisor Mode

i. Set Priority Level (SPL) (Figure 9-12)

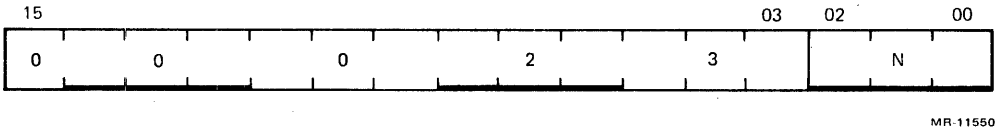


Figure 9-12 Set Priority Level

4. Operate Group: (Figure 9-13) HALT, WAIT, RTI, RESET, RTT, NOP, MFPT

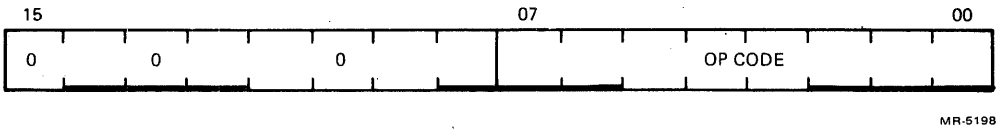


Figure 9-13 Operate Group

5. Condition Code Operators: (Figure 9-14) (all condition code instructions)

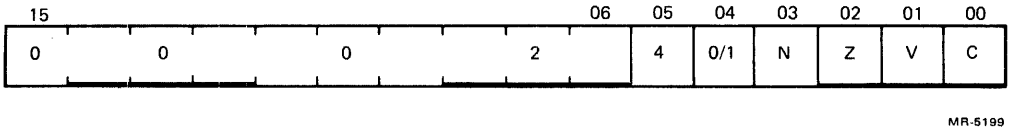


Figure 9-14 Condition Group

6. Move To/From Previous Instruction/Data Space Group: (Figure 9-15)

MTPD, MTPI, MFPD, MFPI

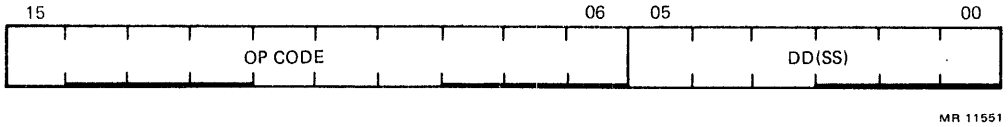


Figure 9-15 Move To and From Previous Instruction/Data Space Group

9.3 BYTE INSTRUCTIONS

The KDJ11-B includes a full complement of instructions that manipulate byte operands. Since all KDJ11-B addressing is byte-oriented, byte manipulation addressing is straightforward. Byte instructions with autoincrement or autodecrement direct addressing cause the specified register to be modified by one to point to the next byte of data. Byte operations in register mode access the low-order byte of the specified register. These provisions enable the KDJ11-B to perform as either a word or byte processor. The numbering scheme for word and byte addresses in memory is shown in Figure 9-16.

The most significant bit (bit 15) of the instruction word is set to indicate a byte instruction.

Example:

Symbolic	Octal Code	Instruction Name
CLR	0050DD	Clear word
CLRB	1050DD	Clear byte

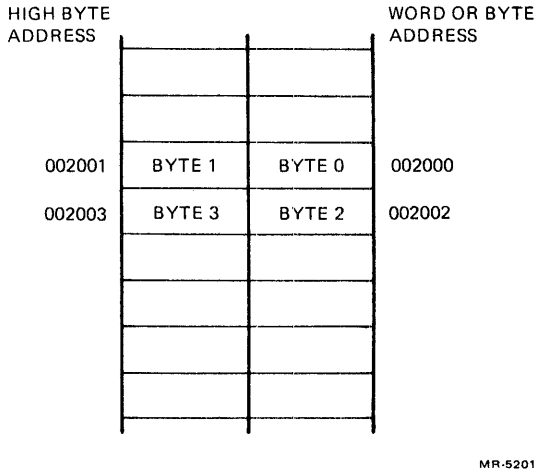


Figure 9-16 Byte Instructions

9.4 LIST OF INSTRUCTIONS

The following is a functional list of the KDJ11-B instruction set.

SINGLE-OPERAND

General

Mnemonic	Instruction	Op Code
CLR(B)	Clear destination	■ 050DD
COM(B)	Complement destination	■ 051DD
INC(B)	Increment destination	■ 052DD
DEC(B)	Decrement destination	■ 053DD
NEG(B)	Negate destination	■ 054DD
TST(B)	Test destination	■ 057DD
WRTLCK	Read/lock destination, write/unlock R0 into destination	0073DD
TSTSET	Test destination, set low bit	0072DD

Shift and Rotate

Mnemonic	Instruction	Op Code
ASR(B)	Arithmetic shift right	■ 062DD
ASL(B)	Arithmetic shift left	■ 063DD
ROR(B)	Rotate right	■ 060DD
ROL(B)	Rotate left	■ 061DD
SWAB	Swap bytes	0003DD

Multiple-Precision

Mnemonic	Instruction	Op Code
ADC(B)	Add carry	■ 055DD
SBC(B)	Subtract carry	■ 056DD
SXT	Sign extend	0067DD

PSW Operators

Mnemonic	Instruction	Op Code
MFPS	Move byte from PSW	1067DD
MTPS	Move byte to PSW	1064SS

DOUBLE-OPERAND

General

Mnemonic	Instruction	Op Code
MOV(B)	Move source to destination	■ 1SSDD
CMP(B)	Compare source to destination	■ 2SSDD
ADD	Add source to destination	06SSDD
SUB	Subtract source from destination	16SSDD
ASH	Arithmetic shift	072RSS
ASHC	Arithmetic shift combined	073RSS
MUL	Multiply	070RSS
DIV	Divide	071RSS

Logical

Mnemonic	Instruction	Op Code
BIT(B)	Bit test	■ 3SSDD
BIC(B)	Bit clear	■ 4SSDD
BIS(B)	Bit set	■ 5SSDD
XOR	Exclusive OR	074RDD

PROGRAM CONTROL

Branch

Mnemonic	Instruction	Op Code or Base Code
BR	Branch (unconditional)	000400
BNE	Branch if not equal (to zero)	001000
BEQ	Branch if equal (to zero)	001400
BPL	Branch if plus	100000
BMI	Branch if minus	100400
BVC	Branch if overflow is clear	102000
BVS	Branch if overflow is set	102400
BCC	Branch if carry is clear	103000
BCS	Branch if carry is set	103400

Signed Conditional Branch

Mnemonic	Instruction	Op Code or Base Code
BGE	Branch if greater than or equal (to zero)	002000
BLT	Branch if less than (zero)	002400
BGT	Branch if greater than (zero)	003000
BLE	Branch if less than or equal (to zero)	003400

Unsigned Conditional Branch

Mnemonic	Instruction	Op Code or Base Code
BHI	Branch if higher	101000
BLOS	Branch if lower or same	101400
BHIS	Branch if higher or same	103000
BLO	Branch if lower	103400

Jump and Subroutine

Mnemonic	Instruction	Op Code or Base Code
JMP	Jump	0001DD
JSR	Jump to subroutine	004RDD
RTS	Return from subroutine	00020R
SOB	Subtract one and branch (if $\neq$ 0)	077RDD

Trap and Interrupt

Mnemonic	Instruction	Op Code or Base Code
EMT	Emulator trap	104000–104377
TRAP	Trap	104400–104777
BPT	Breakpoint trap	000003
IOT	Input/output trap	000004
RTI	Return from interrupt	000002
RTT	Return from interrupt	000006

Miscellaneous Program Control

Mnemonic	Instruction	Op Code or Base Code
CSM	Call to supervisor mode	0070DD
MARK	Mark	0064NN
SPL	Set priority level	00023N

MISCELLANEOUS

Mnemonic	Instruction	Op Code or Base Code
HALT	Halt	000000
WAIT	Wait for interrupt	000001
RESET	Reset external bus	000005
MFPT	Move processor type	000007
MTPD	Move to previous data space	1066DD
MTPI	Move to previous instruction space	0066DD
MFPD	Move from previous data space	1065SS
MFPI	Move from previous instruction space	0065SS

CONDITION CODE OPERATORS

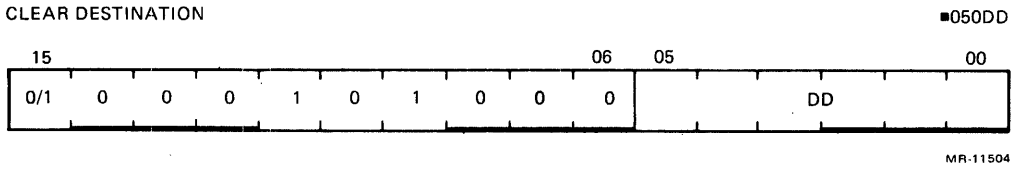
Mnemonic	Instruction	Op Code or Base Code
CLC	Clear C	000241
CLV	Clear V	000242
CLZ	Clear Z	000244
CLN	Clear N	000250
CCC	Clear all CC bits	000257
SEC	Set C	000261
SEV	Set V	000262
SEZ	Set Z	000264
SEN	Set N	000270
SCC	Set all CC bits	000277
NOP	No operation	000240

9.5 SINGLE-OPERAND INSTRUCTIONS

The KDJ11-B instructions that involve only one operand are described in the paragraphs that follow.

9.5.1 General

CLR
CLRB



Operation: (dst) ← 0

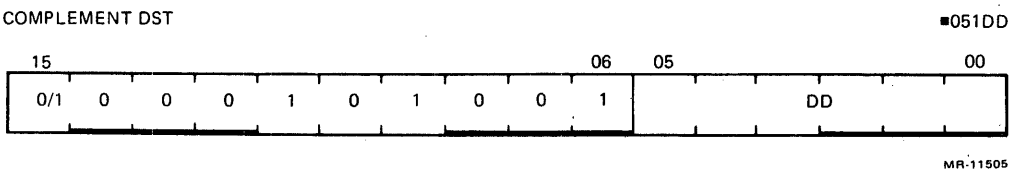
Condition Codes:
N: cleared
Z: set
V: cleared
C: cleared

Description: Word: The contents of the specified destination are replaced with 0s.
Byte: Same.

Example: CLR R1

Before	After
(R1) = 177777	(R1) = 000000
N Z V C	N Z V C
1 1 1 1	0 1 0 0

COM
COMB



Operation: (dst) ← ~ (dst)

Condition Codes:
N: set if most significant bit of result is set; cleared otherwise
Z: set if result is 0; cleared otherwise
V: cleared
C: set

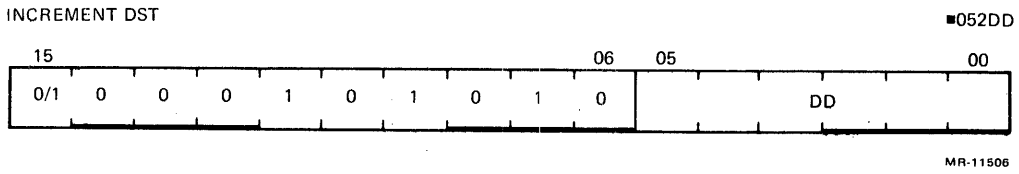
Description: Word: Replaces the contents of the destination address by its logical complement. (Each bit equal to 0 is set and each bit equal to 1 is cleared.)

Byte: Same.

Example: COM R0

Before	After
(R0) = 013333	(R0) = 164444
N Z V C 0 1 1 0	N Z V C 1 0 0 1

INC
INCB



Operation: (dst) ← (dst) + 1

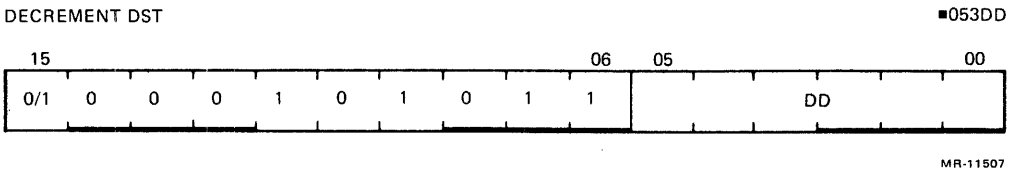
Condition Codes:
N: set if result is < 0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if (dst) held 077777; cleared otherwise
C: not affected

Description: Word: Add 1 to the contents of the destination.
Byte: Same.

Example: INC R2

Before	After
(R2) = 000333	(R2) = 000334
N Z V C 0 0 0 0	N Z V C 0 0 0 0

DEC
DECB



Operation: (dst) ← (dst) − 1

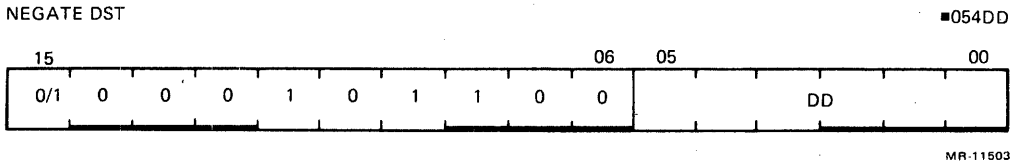
Condition Codes:
N: set if result is < 0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if (dst) was 100000; cleared otherwise
C: not affected

Description: Word: Subtract 1 from the contents of the destination.
Byte: Same.

Example: DEC R5

Before	After
(R5) = 000001	(R5) = 000000
N Z V C	N Z V C
1 0 0 0	0 1 0 0

NEG
NEGB



Operation: (dst) ← − (dst)

Condition Codes:
N: set if result is < 0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if result is 100000; cleared otherwise
C: cleared if result is 0; set otherwise

Description: Word: Replaces the contents of the destination address by its 2's complement. Note that 100000 is replaced by itself. (In 2's complement notation the most negative number has no positive counterpart.)

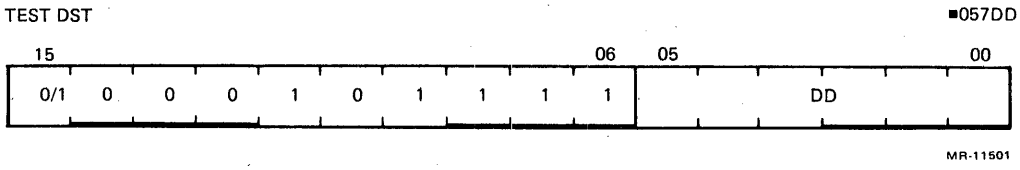
Byte: Same.

Example:

NEG R0

Before	After
(R0) = 000010	(R0) = 177770
N Z V C 0 0 0 0	N Z V C 1 0 0 1

TST
TSTB



Operation: (dst) ← (dst)

Condition Codes:

N: set if result is < 0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: cleared
C: cleared

Description:

Word: Sets the condition codes N and Z according to the contents of the destination address; the contents of dst remain unmodified.

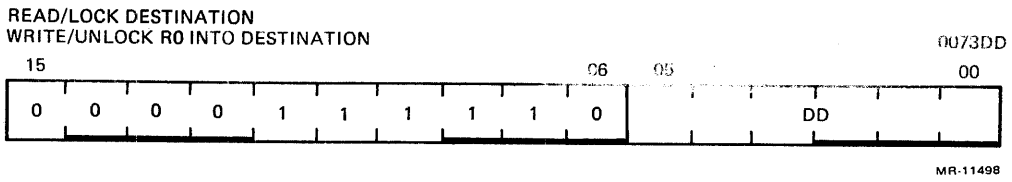
Byte: Same.

Example:

TST R1

Before	After
(R1) = 012340	(R1) = 012340
N Z V C 0 0 1 1	N Z V C 0 0 0 0

WRTLCK

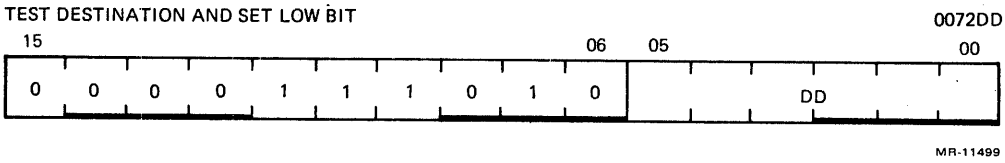


Operation: (dst) ← (R0)

Condition Codes:
N: set if R0 < 0
Z: set if R0 = 0
V: cleared
C: unchanged

Description: Writes contents of R0 into destination using bus lock. If mode is 0, traps to 10.

TSTSET



Operation: (R0) ← (dst), (dst) ← (dst) ∨ 000001 (octal)

Condition Codes:
N: set if R0 < 0
Z: set if R0 = 0
V: cleared
C: gets contents of old destination bit 0.

Description: Reads/locks destination word and stores it in R0. Writes/unlocks (R0) ∨ 1 into destination. If mode is 0, traps to 10.

9.5.2 Shifts and Rotates

Scaling data by factors of two is accomplished by the shift instructions:

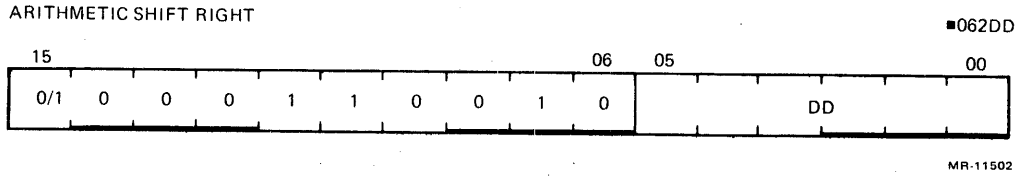
ASR – Arithmetic shift right

ASL – Arithmetic shift left

The sign bit (bit 15) of the operand is reproduced in shifts to the right. The low-order bit is filled with 0s in shifts to the left. Bits shifted out of the C-bit, as shown in the following instructions, are lost.

The rotate instructions operate on the destination word and the C-bit as though they formed a 17-bit 'circular buffer.' These instructions facilitate sequential bit testing and detailed bit manipulation.

ASR
ASRB



Operation: (dst) ← (dst) shifted one place to the right

Condition Codes:

N: set if high-order bit of result is set (result < 0); cleared otherwise

Z: set if result = 0; cleared otherwise

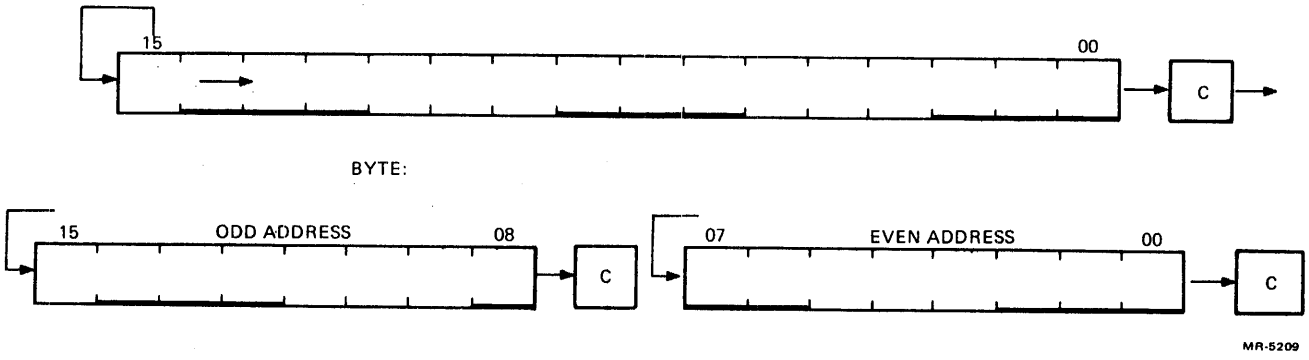
V: loaded from exclusive OR of N-bit and C-bit (as set by the completion of the shift operation)

C: loaded from low-order bit of destination

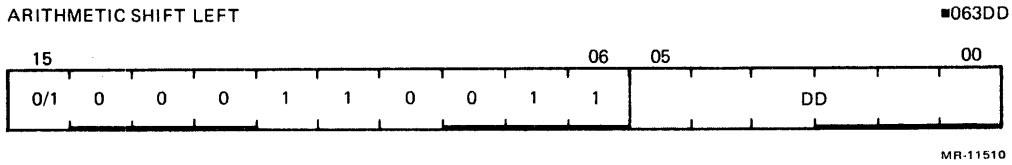
Description: Word: Shifts all bits of the destination right one place. Bit 15 is reproduced. The C-bit is loaded from bit 0 of the destination. ASR performs signed division of the destination by 2.

Byte: Same.

Example:



ASL
ASLB



Operation: (dst) ← (dst) shifted one place to the left

Condition Codes:

N: set if high-order bit of result is set (result < 0); cleared otherwise

Z: set if result = 0; cleared otherwise

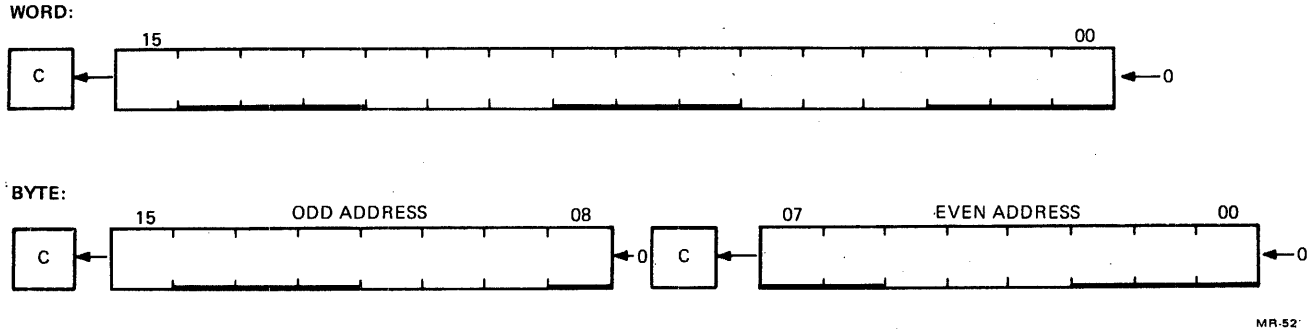
V: loaded with exclusive OR of N-bit and C-bit (as set by the completion of the shift operation)

C: loaded with high-order bit of destination

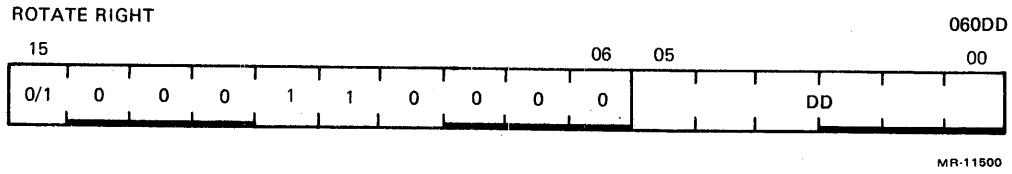
Description: Word: Shifts all bits of the destination left one place. Bit 0 is loaded with a 0. The C-bit of the PSW is loaded from the most significant bit of the destination. ASL performs a signed multiplication of the destination by 2 with overflow indication.

Byte: Same.

Example:



ROR
RORB



Operation: (dst) ← (dst) rotate right one place

Condition Codes:

N: set if high-order bit of result is set (result < 0); cleared otherwise

Z: set if all bits of result = 0; cleared otherwise

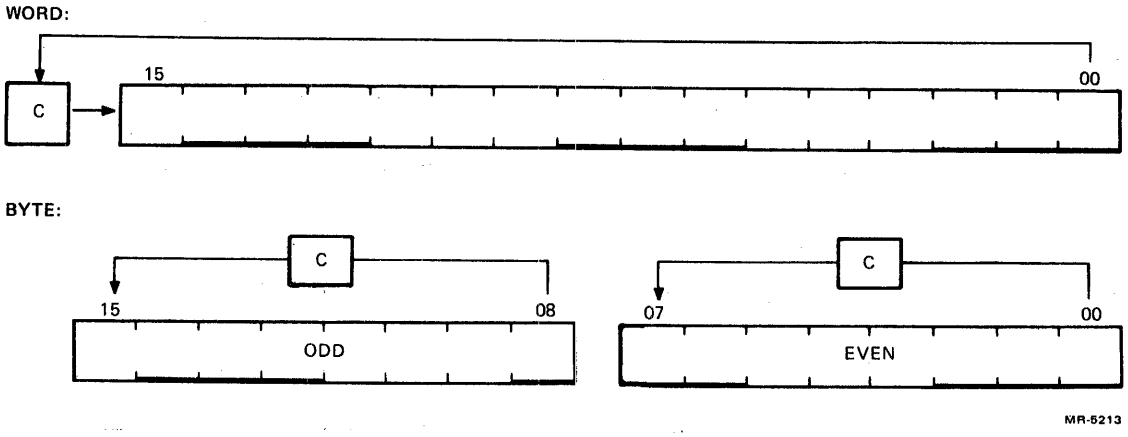
V: loaded with exclusive OR of N-bit and C-bit (as set by the completion of the rotate operation)

C: loaded with low-order bit of destination

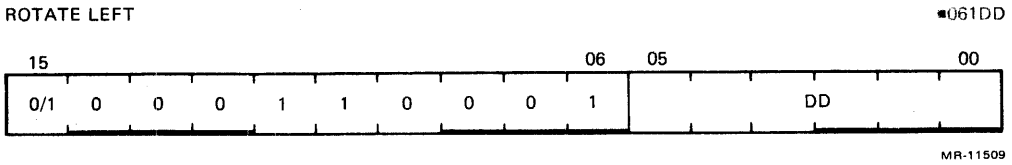
Description: Word: Rotates all bits of the destination right one place. Bit 0 is loaded into the C-bit and the previous contents of the C-bit are loaded into bit 15 of the destination.

Byte: Same, except the C-bit is loaded into MSB 7 or 15.

Example:



ROL
ROLB



Operation: (dst) ← (dst) rotate left one place

Condition Codes:

N: set if high-order bit of result word is set (result < 0); cleared otherwise

Z: set if all bits of result word = 0; cleared otherwise

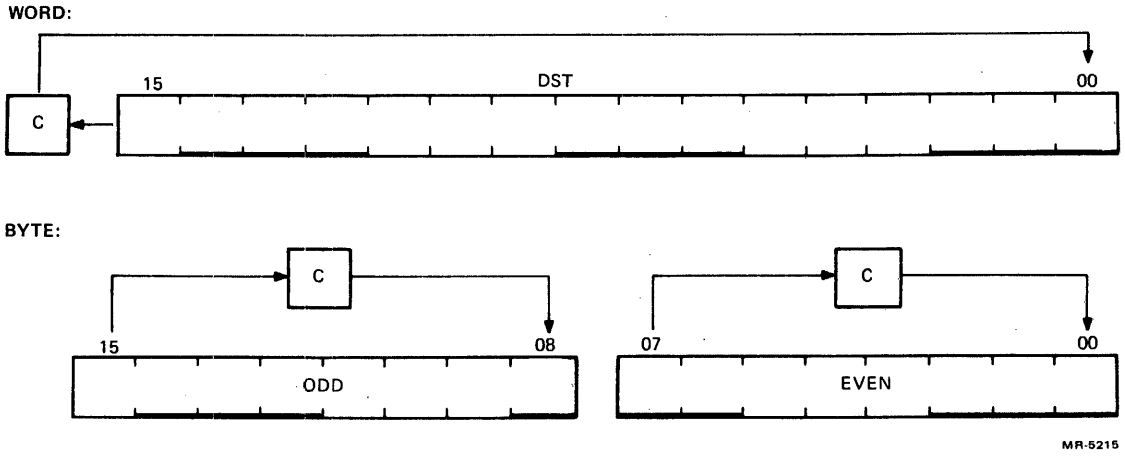
V: loaded with exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation)

C: loaded with high-order bit of destination

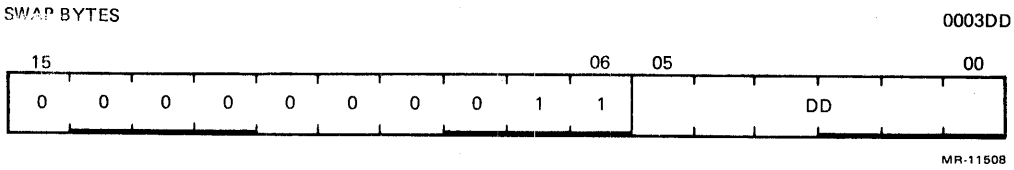
Description: Word: Rotates all bits of the destination left one place. Bit 15 is loaded into the C-bit of the PSW and the previous contents of the C-bit are loaded into bit 0 of the destination.

Byte: Same, except the C-bit is loaded into LSB 8 or 0.

Example:



SWAB



Operation: byte 1/byte 0 ← byte 0/byte 1

Condition Codes:

N: set if high-order bit of low-order byte (bit 7) of result is set; cleared otherwise

Z: set if low-order byte of result = 0; cleared otherwise

V: cleared

C: cleared

Description: Exchanges high-order byte and low-order byte of the destination word. (The destination must be a word address.)

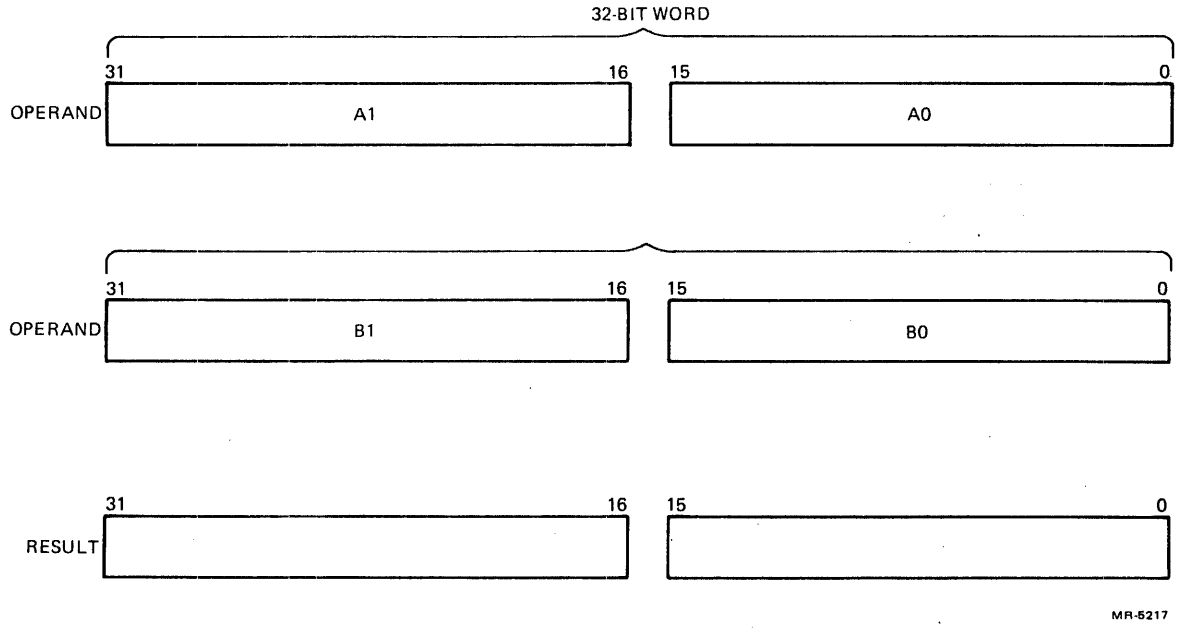
Example: SWAB R1

Before	After
(R1) = 077777	(R1) = 177577
N Z V C 1 1 1 1	N Z V C 0 0 0 0

9.5.3 Multiple-Precision

It is sometimes necessary to do arithmetic operations on operands considered as multiple words or bytes The KDJ11-B makes special provision for such operations with the instructions ADC (add carry) and SBC (subtract carry) and their byte equivalents.

For example, two 16-bit words may be combined into a 32-bit double-precision word and added o subtracted as shown below.



Example:

The addition of −1 and −1 could be performed as follows.

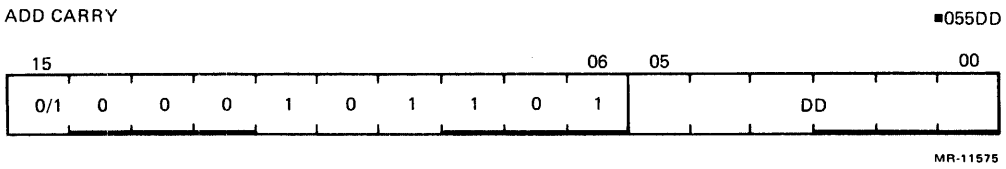
−1 = 37777777777

(R1) = 177777 (R2) = 177777 (R3) = 177777 (R4) = 177777

```
ADD R1,R2
ADC R3
ADD R4,R3
```

1. After (R1) and (R2) are added, 1 is loaded into the C-bit.
2. The ADC instruction adds the C-bit to (R3); (R3) = 0.
3. (R3) and (R4) are added.
4. The result is 37777777776, or −2.

ADC
ADCB



Operation: (dst) ← (dst) + (C-bit)

Condition Codes:
N: set if result < 0; cleared otherwise
Z: set if result = 0; cleared otherwise
V: set if (dst) was 077777 and (C) was 1; cleared otherwise
C: set if (dst) was 177777 and (C) was 1; cleared otherwise

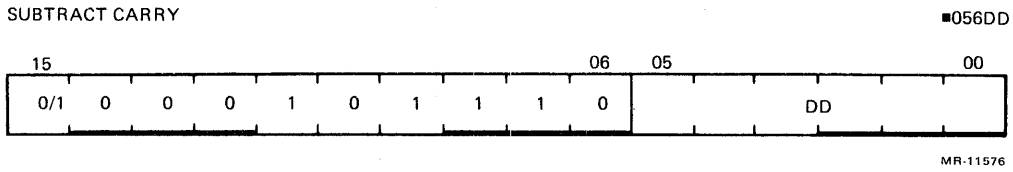
Description: Word: Adds the contents of the C-bit to the destination. This permits the carry from the addition of the low-order words to be carried to the high-order result.

Byte: Same.

Example: Double-precision addition may be done with the following instruction sequence.

```
ADD     A0,B0     ;add low-order parts
ADC     B1        ;add carry into high-order
ADD     A1,B1     ;add high-order parts
```

SBC
SBCB



Operation: (dst) ← (dst) − (C)

Condition Codes:
N: set if result < 0; cleared otherwise
Z: set if result = 0; cleared otherwise
V: set if (dst) was 100000; cleared otherwise
C: set if (dst) was 0 and C was 1; cleared otherwise

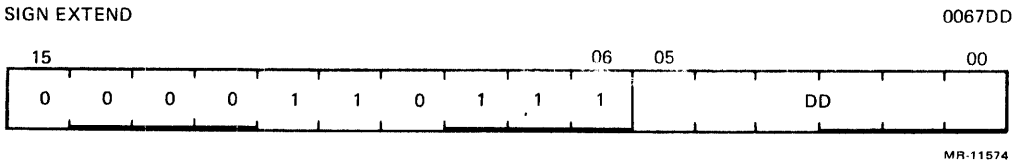
Description: Word: Subtracts the contents of the C-bit from the destination. This permits the carry from the subtraction of two low-order words to be subtracted from the high-order part of the result.

Byte: Same.

Example: Double-precision subtraction is done by:

```
SUB     A0,B0
SBC     B1
SUB     A1,B1
```

SXT



Operation: (dst) ← 0 if N-bit is clear
(dst) ← 1 if N-bit is set

Condition Codes: N: not affected
Z: set if N-bit is clear
V: cleared
C: not affected

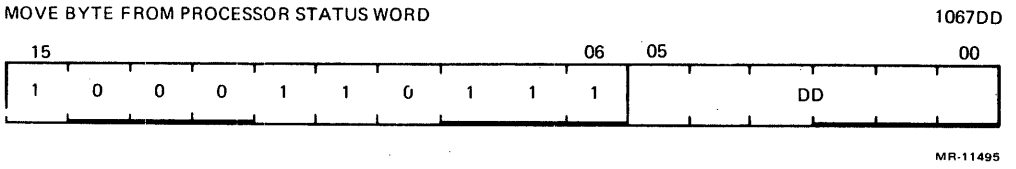
Description: If the condition code bit N is set, a −1 is placed in the destination operand; if the N-bit is clear, a 0 is placed in the destination operand. This instruction is particularly useful in multiple-precision arithmetic because it permits the sign to be extended through multiple words.

Example: SXT A

Before	After
(A) = 012345	(A) = 177777
N Z V C 1 0 0 0	N Z V C 1 0 0 0

9.5.4 PSW Operators

MFPS



Operation: (dst) ← PSW
dst lower 8 bits

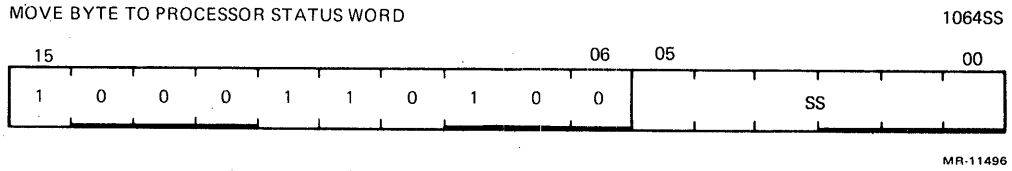
Condition Codes: N: set if PSW bit 7 = 1; cleared otherwise
Z: set if PSW <7:0> = 0; cleared otherwise
V: cleared
C: not affected

Description: The 8-bit contents of the PSW are moved to the effective destination. If the destination is mode 0, PSW bit 7 is sign-extended through the upper byte of the register. The destination operand address is treated as a byte address.

Example: MFPS R0

Before	After
(R0) = 0	(R0) = 000014
(PSW) = 000014	(PSW) = 000000

MTPS



Operation: PSW ← (src)

Condition Codes: Set according to effective SRC operand bits <3:0>

Description: The eight bits of the effective operand replace the current contents of the lower byte of the PSW. The source operand address is treated as a byte address. Note: The T-bit (PSW bit 4) cannot be set with this instruction. The SRC operand remains unchanged. This instruction can be used to change the priority bits (PSW <7:5>) in the PSW only in kernel mode. If not in kernel mode, PSW <7:5> cannot be changed.

Example: MTPS R1

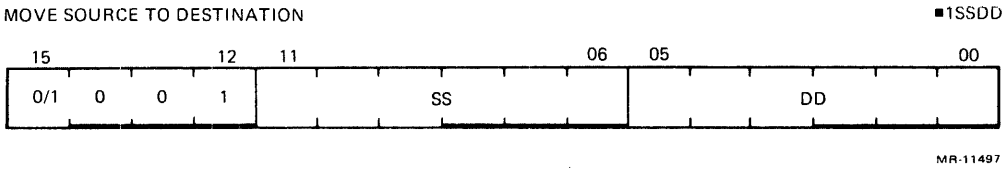
Before	After
(R1) = 000777	(R1) = 000777
(PSW) = XXX000	(PSW) = XXX357
N Z V C	N Z V C
0 0 0 0	1 1 1 1

9.6 DOUBLE-OPERAND INSTRUCTIONS

Double-operand instructions save instructions (and time), since they eliminate the need for load and save sequences such as those used in accumulator-oriented machines.

9.6.1 General

MOV
MOVB



Operation: (dst) ← (src)

Condition Codes:
N: set if (src) < 0; cleared otherwise
Z: set if (src) = 0; cleared otherwise
V: cleared
C: not affected

Description: Word: Moves the source operand to the destination location. The previous contents of the destination are lost. Contents of the source address are not affected.

Byte: Same as MOV. The MOVB to a register (unique among byte instructions) extends the most significant bit of the low-order byte (sign extension). Otherwise, MOVB operates on bytes exactly as MOV operates on words.

Example:

```
MOV XXX,R1          ;loads register 1 with the contents of memory location; XXX represents a programmer-defined mnemonic used to represent a memory location

MOV #20,R0          ;loads the number 20 into register 0; # indicates that the value 20 is the operand

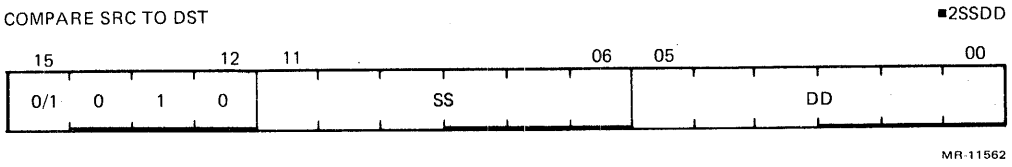
MOV @#20,-(R6)      ;pushes the operand contained in location 20 onto the stack

MOV (R6)+,@#177566  ;pops the operand off a stack and moves it into memory location 177566 (terminal print buffer)
```

```
MOV R1,R3                    ;performs an inter-register
                             transfer

MOVB @#177562,@#177566       ;moves a character from the
                             terminal keyboard buffer to
                             the terminal printer buffer
```

CMP
CMPB



Operation: (src) – (dst)

Condition Codes:

N: set if result < 0; cleared otherwise

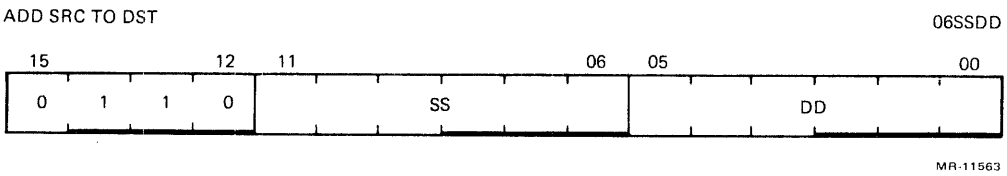
Z: set if result = 0; cleared otherwise

V: set if there was arithmetic overflow; that is, operands were of opposite signs and the sign of the destination was the same as the sign of the result; cleared otherwise

C: cleared if there was a carry from the most significant bit of the result; set otherwise

Description: Compares the source and destination operands and sets the condition codes, which may then be used for arithmetic and logical conditional branches. Both operands are not affected. The only action is to set the condition codes. The compare is customarily followed by a conditional branch instruction. Notice that unlike the subtract instruction, the order of operation is (src) – (dst), not (dst) – (src).

ADD



Operation: (dst) ← (src) + (dst)

Condition Codes:

N: set if result < 0; cleared otherwise

Z: set if result = 0; cleared otherwise

V: set if there was arithmetic overflow as a result of the operation, that is, both operands were of the same sign and the result was of the opposite sign; cleared otherwise

C: set if there was a carry from the most significant bit of the result; cleared otherwise

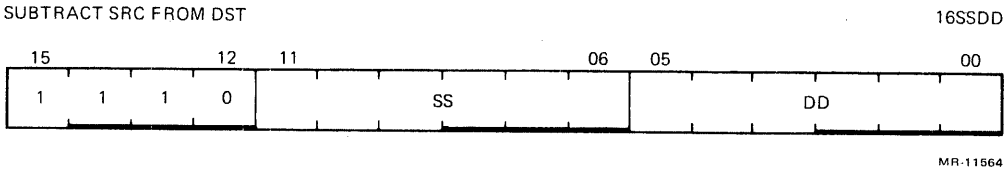
Description: Adds the source operand to the destination operand and stores the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. 2's complement addition is performed. Notice that there is no equivalent byte mode.

Example:

Add to register:	ADD 20,R0
Add to memory:	ADD R1,XXX
Add register to register:	ADD R1,R2
Add memory to memory:	ADD @#17750,XXX

(XXX is a programmer-defined mnemonic for a memory location.)

SUB



Operation: (dst) ← (dst) − (src)

Condition Codes:

N: set if result < 0; cleared otherwise

Z: set if result = 0; cleared otherwise

V: set if there was arithmetic overflow as a result of the operation, that is, if operands were of opposite signs and the sign of the source was the same as the sign of the result; cleared otherwise

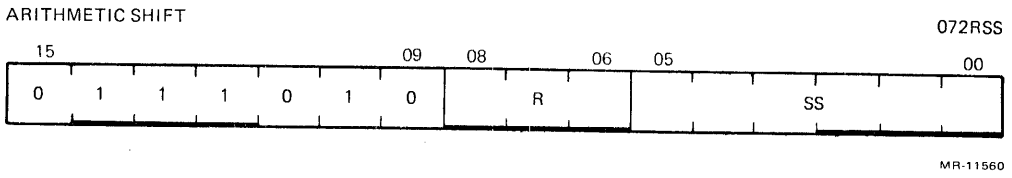
C: cleared if there was a carry from the most significant bit of the result; set otherwise

Description: Subtracts the source operand from the destination operand and leaves the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. In double-precision arithmetic the C-bit, when set, indicates a 'borrow.' Notice that there is no equivalent byte mode.

Example: SUB R1,R2

Before	After
(R1) = 011111	(R1) = 011111
(R2) = 012345	(R2) = 001234
N Z V C	N Z V C
1 1 1 1	0 0 0 0

ASH



Operation: R ← R shifted arithmetically NN places to the right or left where NN = (src)

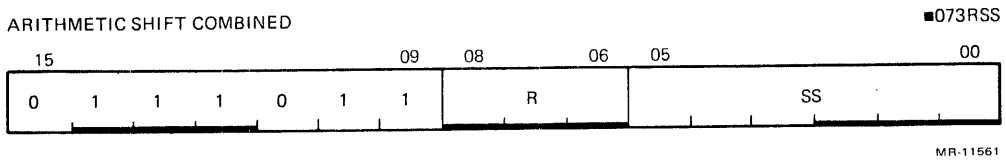
Condition Codes:
- N: set if result < 0
- Z: set if result = 0
- V: set if sign of register changed during shift
- C: loaded from last bit shifted out of register

Description: The contents of the register are shifted right or left the number of times specified by the source operand. The shift count is taken as the low-order six bits of the source operand. This number ranges from −32 to +31. Negative is a right shift and positive (less than +31) is a left shift.

NOTE

A shift count of +31 shifts the contents of the register to the right 31 times.

ASHC



Operation: R, R ∨ 1 ← R, R ∨ 1
The double word is shifted NN places to the right or left where NN = (src)

Condition Codes:
- N: set if result < 0
- Z: set if result = 0
- V: set if sign bit changes during shift
- C: loaded with high-order bit when left shift; loaded with low-order bit when right shift (loaded with the last bit shifted out of the 32-bit operand)

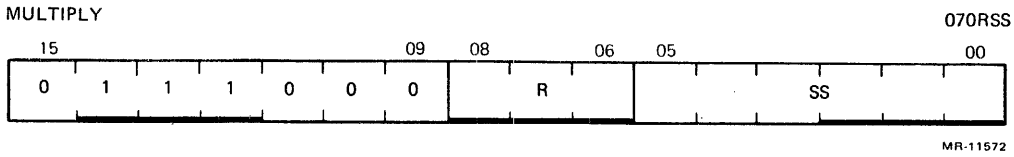
Description: The contents of the register and the register ORed with 1 are treated as one 32-bit word. R ∨ 1 (bits <15:0>) and R (bits <31:16>) are shifted right or left the number of times specified by the shift count. The shift count is taken as the low-order 6 bits of the source operand; the upper 11 bits of the source operand must be 0. This number ranges from −32 to +31. Negative is a right shift and positive is a left shift.

When the register chosen is an odd number, the register and the register ORed with 1 are the same. In this case, the right shift becomes a rotate. The 16-bit word is rotated right the number of times specified by the shift count.

NOTE

Bits <5:0> shift count. Bits <15:6> must be 0.

MUL

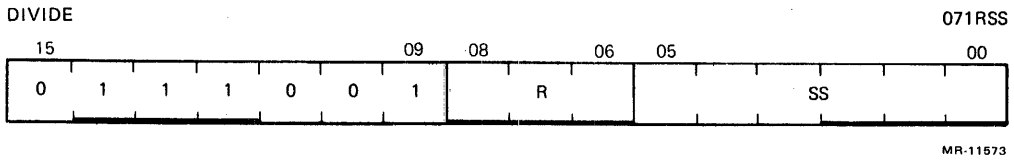


Operation: R, R ∨ 1 ← R × (src)

Condition Codes:
- N: set if product < 0
- Z: set if product = 0
- V: cleared
- C: set if the result is less than −2 ** 15 or greater than or equal to 2 **15 −1.

Description: The contents of the destination register and source taken as 2's complement integers are multiplied and stored in the destination register and the succeeding register, if R is even. If R is odd, only the low-order product is stored. Assembler syntax is: MUL S,R. Notice that the actual destination is R, R ∨ 1, which reduces to just R when R is odd.

DIV



Operation: R, R ∨ 1 ← R, R ∨ 1/(src)

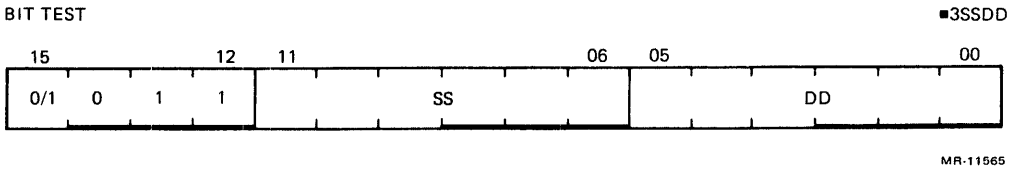
Condition Codes:
- N: set if quotient < 0
- Z: set if quotient = 0
- V: set if source = 0 or if the absolute value of the register is larger than the absolute value of the instruction in the source. (In this case the instruction is aborted because the quotient would exceed 15 bits.)
- C: set if divide by zero is attempted.

Description: The 32-bit 2's complement integer in R and R ∨ 1 is divided by the source operand. The quotient is left in R; the remainder is of the same sign as the dividend. R must be even.

9.6.2 Logical

These instructions have the same format as those in the double-operand arithmetic group. They permit operations on data at the bit level.

BIT
BITB



Operation: (src) ∧ (dst)

Condition Codes:
- N: set if high-order bit of result set; cleared otherwise
- Z: set if result = 0; cleared otherwise
- V: cleared
- C: not affected

Description: Performs logical AND comparison of the source and destination operands and modifies condition codes accordingly. Neither the source nor the destination is affected. The BIT instruction may be used to test whether any of the corresponding bits set in the destination are also set in the source, or whether all corresponding bits set in the destination are clear in the source.

Example:

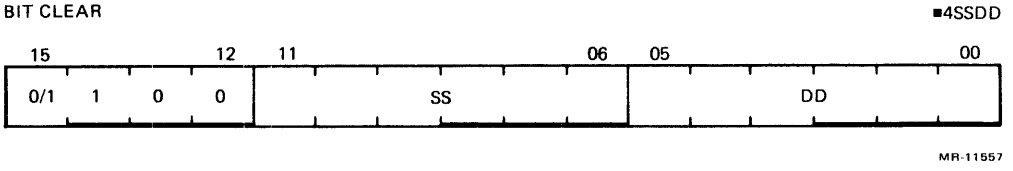
```
BIT #30,R3          ;test bits three and four of R3 to see if
                    ;both are off.

R3 = 0 000 000 000 011 000

Before              After

N Z V C             N Z V C
1 1 1 1             0 0 0 1
```

BIC
BICB



Operation: (dst) ← ~ (src) ∧ (dst)

Condition Codes:
- N: set if high-order bit of result set; cleared otherwise
- Z: set if result = 0; cleared otherwise
- V: cleared
- C: not affected

Description: Clears each bit in the destination that corresponds to a set bit in the source. The original contents of the destination are lost. The contents of the source are not affected.

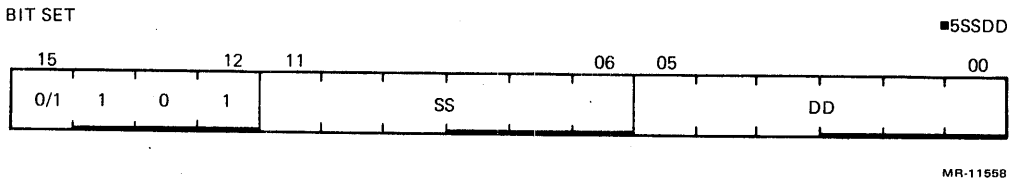
Example: BIC R3,R4

Before	After
(R3) = 001234	(R3) = 001234
(R4) = 001111	(R4) = 000101
N Z V C	N Z V C
1 1 1 1	0 0 0 1

Before: (R3) = 0 000 001 010 011 100
(R4) = 0 000 001 001 001 001

After: (R4) = 0 000 000 001 000 001

BIS
BISB



Operation: (dst) ← (src) ∨ (dst)

Condition Codes:
N: set if high-order bit of result set; cleared otherwise
Z: set if result = 0; cleared otherwise
V: cleared
C: not affected

Description: Performs an inclusive OR operation between the source and destination operands and leaves the result at the destination address, that is, corresponding bits set in the source are set in the destination. The contents of the destination are lost.

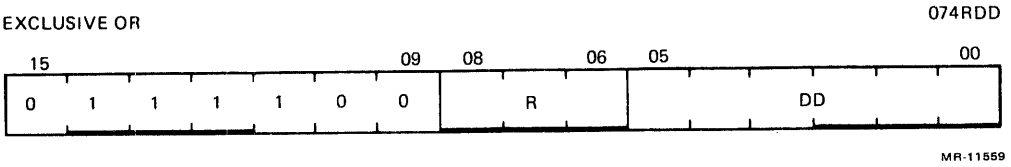
Example: BIS R0,R1

Before	After
(R0) = 001234	(R0) = 001234
(R1) = 001111	(R1) = 001335
N Z V C	N Z V C
0 0 0 0	0 0 0 0

Before: (R0) = 0 000 001 010 011 100
(R1) = 0 000 001 001 001 001

After: (R1) = 0 000 001 011 011 101

XOR



Operation: (dst) ← (reg) ∀ (dst)

Condition Codes:
N: set if result < 0; cleared otherwise
Z: set if result = 0; cleared otherwise
V: cleared
C: not affected

Description: The exclusive OR of the register and destination operand is stored in the destination address. The contents of the register are not affected. The assembler format is XOR R,D.

Example: XOR R0,R2

Before	After
(R0) = 001234	(R0) = 001234
(R2) = 001111	(R2) = 000325
N Z V C	N Z V C
1 1 1 1	0 0 0 1

Before: (R0) = 0 000 001 010 011 100
(R2) = 0 000 001 001 001 001

After: (R2) = 0 000 000 011 010 101

9.7 PROGRAM CONTROL INSTRUCTIONS

The following paragraphs describe the KDJ11-B instructions that affect program control.

9.7.1 Branches

These instructions cause a branch to a location defined by the sum of the offset (multiplied by 2) and the current contents of the program counter if:

1. The branch instruction is unconditional.
2. The branch instruction is conditional and the conditions are met after testing the condition codes (N Z V C).

The offset is the number of words from the current contents of the PC, forward or backward. Note that the current contents of the PC point to the word following the branch instruction.

Although the offset expresses a byte address, the PC is expressed in words. The offset is automatically multiplied by 2 and sign-extended to express words before it is added to the PC. Bit 7 is the sign of the offset. If it is set, the offset is negative and the branch is done in the backward direction. If it is not set, the offset is positive and the branch is done in the forward direction.

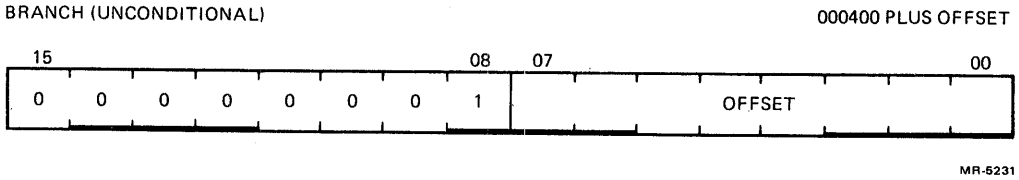
The 8-bit offset allows branching in the backward direction by 200 octal words (400 octal bytes) from the current PC, and in the forward direction by 177 octal words (376 octal bytes) from the current PC.

The KDJ11-B assembler typically handles address arithmetic for the user and computes and assembles the proper offset field for branch instructions in the form:

Bxx loc

Bxx is the branch instruction and loc is the address to which the branch is to be made. The assembler gives an error indication in the instruction if the permissible branch range is exceeded. Branch instructions have no effect on condition codes. Conditional branch instructions where the branch condition is not met are treated as NOPs.

BR



Operation: PC ← PC + (2 × offset)

Condition Codes: Not affected

Description: Provides a way of transferring program control within a range of −128 to +127 words with a one word instruction.

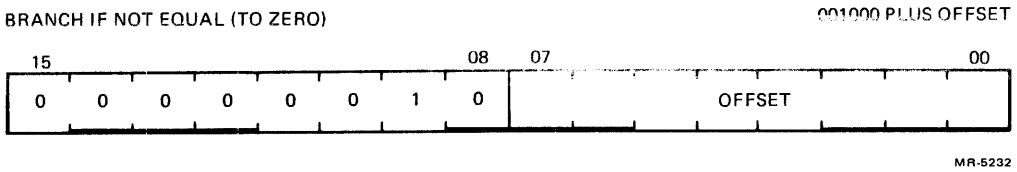
New PC address = updated PC + (2 × offset)

Updated PC = address of branch instruction +2

Example: With the branch instruction at location 500, the following offsets apply.

New PC Address	Offset Code	Offset (decimal)
474	375	−3
476	376	−2
500	377	−1
502	000	0
504	001	+1
506	002	+2

BNE



Operation: PC ← PC + (2 × offset) if Z = 0

Condition Codes: Not affected

Description: Tests the state of the Z-bit and causes a branch if the Z-bit is clear. BNE is the complementary operation of BEQ. It is used to test: (1) inequality following a CMP, (2) that some bits set in the destination were also in the source following a BIT operation, and (3) generally, that the result of the previous operation was not 0.

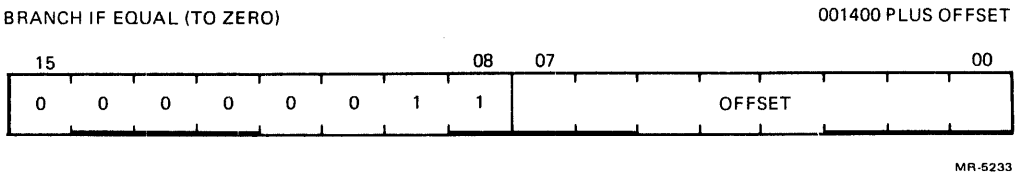
Example: Branch to C if A ≠ B

```
CMP A,B          ;compare A and B
BNE C            ;branch if they are not equal
```

Branch to C if A + B ≠ 0

```
ADD A,B          ;add A to B
BNE C            ;branch if the result is not
                 equal to 0
```

BEQ



Operation: PC ← PC + (2 × offset) if Z = 1

Condition Codes: Not affected

Description: Tests the state of the Z-bit and causes a branch if Z is set. It is used to test: (1) equality following a CMP operation, (2) that no bits set in the destination were also set in the source following a BIT operation, and (3) generally, that the result of the previous operation was 0.

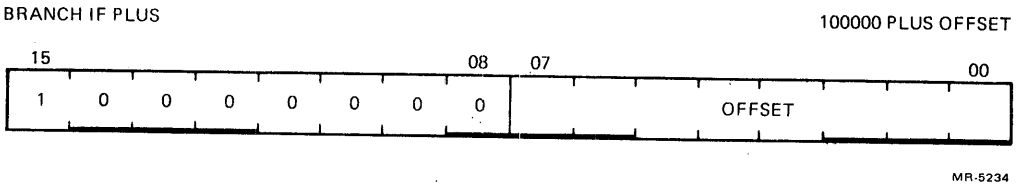
Example: Branch to C if A = B

```
CMP A,B          ;compare A and B
BEQ C            ;branch if they are equal
```

Branch to C if A + B = 0

```
ADD A,B          ;add A to B
BEQ C            ;branch if the result = 0
```

BPL

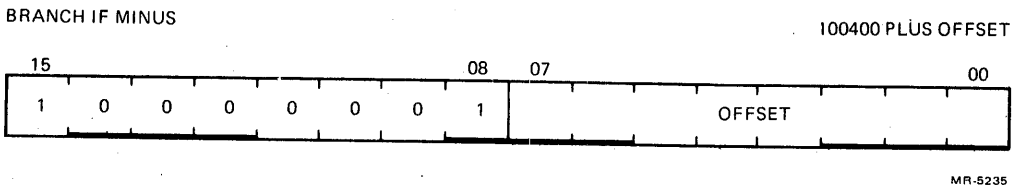


Operation: PC ← PC + (2 × offset) if N = 0

Condition Codes: Not affected

Description: Tests the state of the N-bit and causes a branch if N is clear (positive result). BPL is the complementary operation of BMI.

BMI

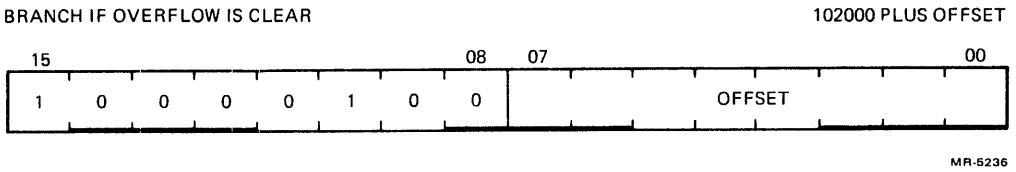


Operation: PC ← PC + (2 × offset) if N = 1

Condition Codes: Not affected

Description: Tests the state of the N-bit and causes a branch if N is set. It is used to test the sign (most significant bit) of the result of the previous operation), branching if negative. BMI is the complementary function of BPL.

BVC

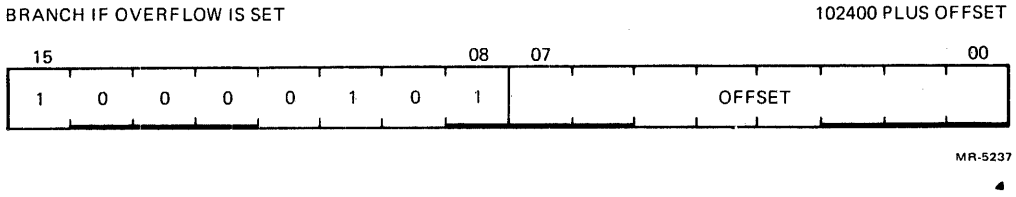


Operation: PC ← PC + (2 × offset) if V = 0

Condition Codes: Not affected

Description: Tests the state of the V-bit and causes a branch if the V-bit is clear. BVC is the complementary operation of BVS.

BVS

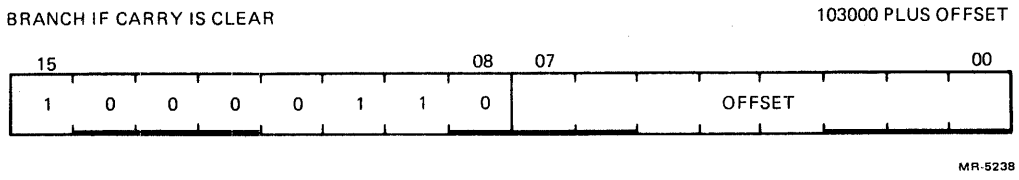


Operation: PC ← PC + (2 × offset) if V = 1

Condition Codes: Not affected

Description: Tests the state of the V-bit (overflow) and causes a branch if V is set. BVS is used to detect arithmetic overflow in the previous operation.

BCC

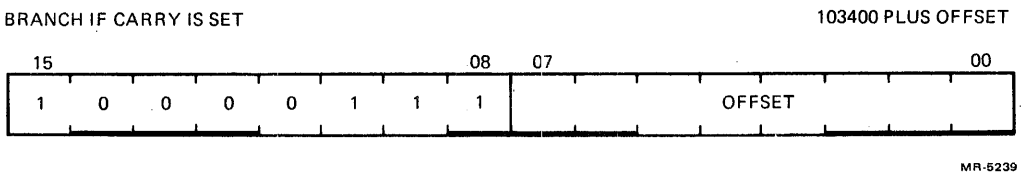


Operation: PC ← PC + (2 × offset) if C = 0

Condition Codes: Not affected

Description: Tests the state of the C-bit and causes a branch if C is clear. BCC is the complementary operation of BCS.

BCS



Operation: PC ← PC + (2 × offset) if C = 1

Condition Codes: Not affected

Description: Tests the state of the C-bit and causes a branch if C is set. It is used to test for a carry in the result of a previous operation.

9.7.2 Signed Conditional Branches

Particular combinations of the condition code bits are tested with the signed conditional branches. These instructions are used to test the results of instructions in which the operands were considered as signed (2's complement) values.

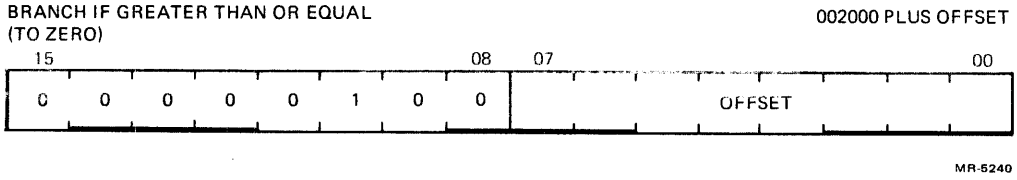
Note that the sense of signed comparisons differs from that of unsigned comparisons in that in signed, 16-bit, 2's complement arithmetic, the sequence of values is as follows.

largest positive	077777
	077776
	.
	.
	.
	000001
	000000
	177777
	177776
	.
	.
	.
smallest negative	100001
	100000

Whereas, in unsigned, 16-bit arithmetic, the sequence is considered to be:

highest	177777
	.
	.
	.
	.
	.
	.
	000002
	000001
lowest	000000

BGE

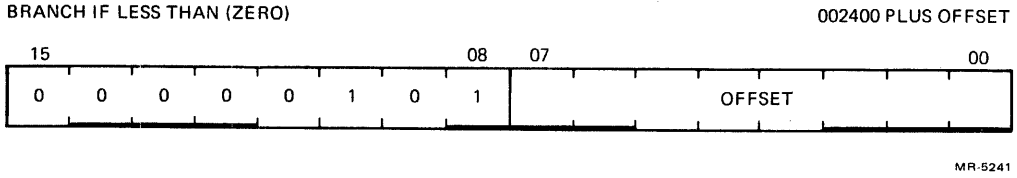


Operation: $PC \leftarrow PC + (2 \times offset)$ if $N \veebar V = 0$

Condition Codes: Not affected

Description: Causes a branch if N and V are either both clear or both set. BGE is the complementary operation of BLT. Thus, BGE will always cause a branch when it follows an operation that caused addition of two positive numbers. BGE will also cause a branch on a 0 result.

BLT

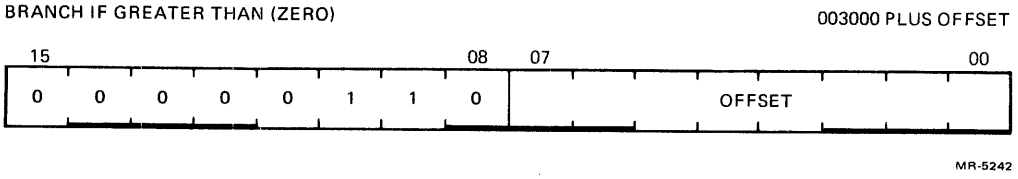


Operation: $PC \leftarrow PC + (2 \times offset)$ if $N \veebar V = 1$

Condition Codes: Not affected

Description: Causes a branch if the exclusive OR of the N- and V-bits is one. Thus, BLT will always branch following an operation that added two negative numbers, even if overflow occurred. In particular, BLT will always cause a branch if it follows a CMP instruction operating on a negative source and a positive destination (even if overflow occurred). Further, BLT will never cause a branch when it follows a CMP instruction operating on a positive source and negative destination. BLT will not cause a branch if the result of the previous operation was 0 (without overflow).

BGT

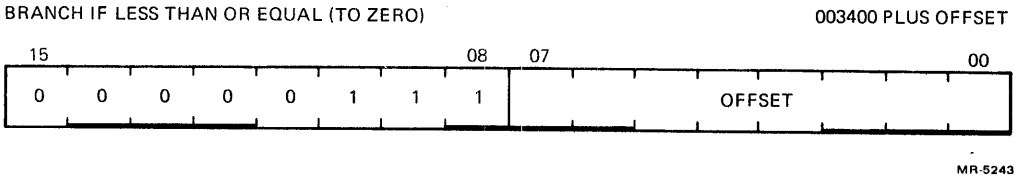


Operation: $PC \leftarrow PC + (2 \times offset)$ if $Z \vee (N \veebar V) = 0$

Condition Codes: Not affected

Description: Operation of BGT is similar to BGE, except that BGT will not cause a branch on a 0 result.

BLE



Operation: PC ← PC + (2 × offset) if Z ∨ (N ⊻ V) = 1

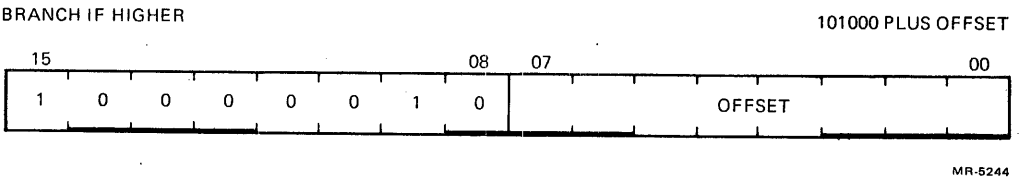
Condition Codes: Not affected

Description: Operation is similar to BLT, but in addition will cause a branch if the result of the previous operation was 0.

9.7.3 Unsigned Conditional Branches

The unsigned conditional branches provide a means for testing the result of comparison operations in which the operands are considered as unsigned values.

BHI

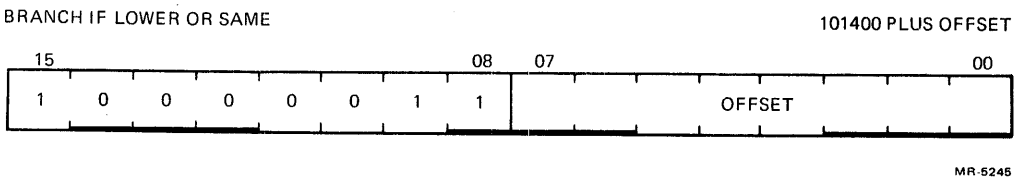


Operation: PC ← PC + (2 × offset) if C = 0 and Z = 0

Condition Codes: Not affected

Description: Causes a branch if the previous operation caused neither a carry nor a 0 result. This will happen in comparison (CMP) operations as long as the source has a higher unsigned value than the destination.

BLOS

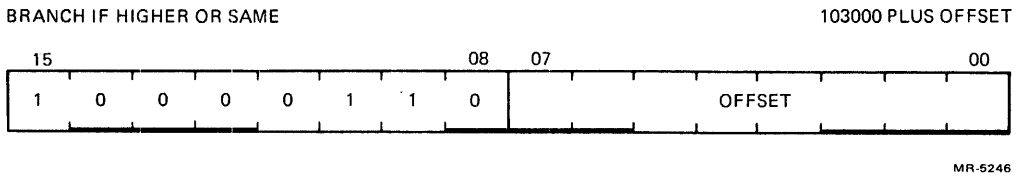


Operation: PC ← PC + (2 × offset) if C ∨ Z = 1

Condition Codes: Not affected

Description: Causes a branch if the previous operation caused either a carry or a 0 result. BLOS is the complementary operation of BHI. The branch will occur in comparison operations as long as the source is equal to or has a lower unsigned value than the destination.

BHIS

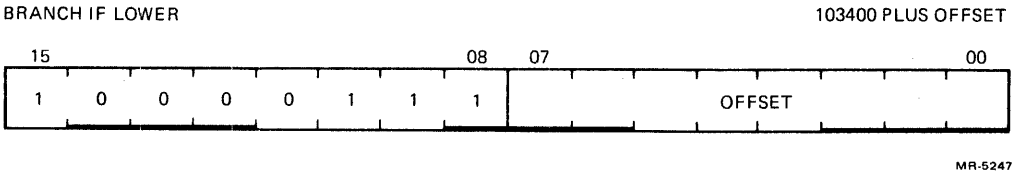


Operation: PC ← PC + (2 × offset) if C = 0

Condition Codes: Not affected

Description: BHIS is the same instruction as BCC. This mnemonic is included for convenience only.

BLO



Operation: PC ← PC + (2 × offset) if C = 1

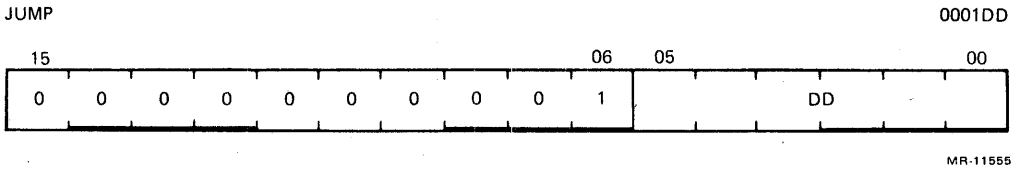
Condition Codes: Not affected

Description: BLO is the same instruction as BCS. This mnemonic is included for convenience only.

9.7.4 Jump and Subroutine Instructions

The subroutine call in the KDJ11-B provides for automatic nesting of subroutines, reentrancy, and multiple entry points. Subroutines may call other subroutines (or indeed themselves) to any level of nesting without making special provision for storage of return addresses at each level of subroutine call. The subroutine calling mechanism does not modify any fixed location in memory, and thus provides for reentrancy. This allows one copy of a subroutine to be shared among several interrupting processes.

JMP



Operation: PC ← (dst)

Condition Codes: Not affected

Description: JMP provides more flexible program branching than the branch instructions do. Control may be transferred to any location in memory (no range limitation) and can be accomplished with the full flexibility of the addressing modes, with the exception of register mode 0. Execution of a jump with mode 0 will cause an illegal instruction condition, and will cause the CPU to trap to vector address 4. (Program control cannot be transferred to a register.) Register-deferred mode is legal and will cause program control to be transferred to the address held in the specified register. Note that instructions are word data and must therefore be fetched from an even-numbered address.

Deferred-index mode JMP instructions permit transfer of control to the address contained in a selectable element of a table of dispatch vectors.

Example:

```
First:

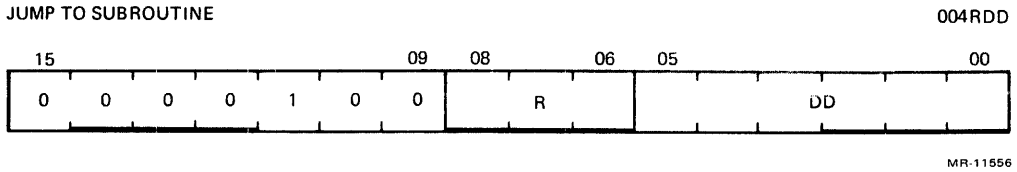
JMP FIRST           ;transfers to FIRST
. . . . .
. . . . .
JMP @LIST           ;transfers to location
                    pointed to at LIST
. . . . .

List:

FIRST               ;pointer to FIRST

JMP @(SP)+          ;transfer to location
                    pointed to by the top of
                    the stack, and remove the
                    pointer from the stack
```

JSR



Operation:

(tmp) ← (dst) (tmp is an internal processor register)

↓ (SP) ← reg (pushes register contents onto processor stack)

reg ← PC (PC holds location following JSR – this address now put in register)

PC ← (dst) (PC now points to subroutine destination)

Description:

In execution of the JSR, the old contents of the specified register (the linkage pointer) are automatically pushed onto the processor stack and new linkage information is placed in the register. Thus, subroutines nested within subroutines to any depth may all be called with the same linkage register. There is no need either to plan the maximum depth at which any particular subroutine will be called or to include instructions in each routine to save and restore the linkage pointer. Further, since all linkages are saved in a reentrant manner on the processor stack, execution of a subroutine may be interrupted. The same subroutine may be reentered and executed by an interrupt service routine. Execution of the initial subroutine can then be resumed when other requests are satisfied. This process (called nesting) can proceed to any level.

A subroutine called with a JSR reg,dst instruction can access the arguments following the call with either autoincrement addressing, (reg) +, if arguments are accessed sequentially; or by indexed addressing, X(reg), if accessed in random order. These addressing modes may also be deferred, @(reg)+ and @X(reg), if the parameters are operand addresses rather than the operands themselves.

JSR PC, dst is a special case of the KDJ11-B subroutine call suitable for subroutine calls that transmit parameters through the general registers. The SP and the PC are the only registers that may be modified by this call.

Another special case of the JSR instruction is JSR PC,@(SP) +, which exchanges the top element of the processor stack with the contents of the program counter. This instruction allows two routines to swap program control and resume operation from where they left off when they are recalled. Such routines are called coroutines.

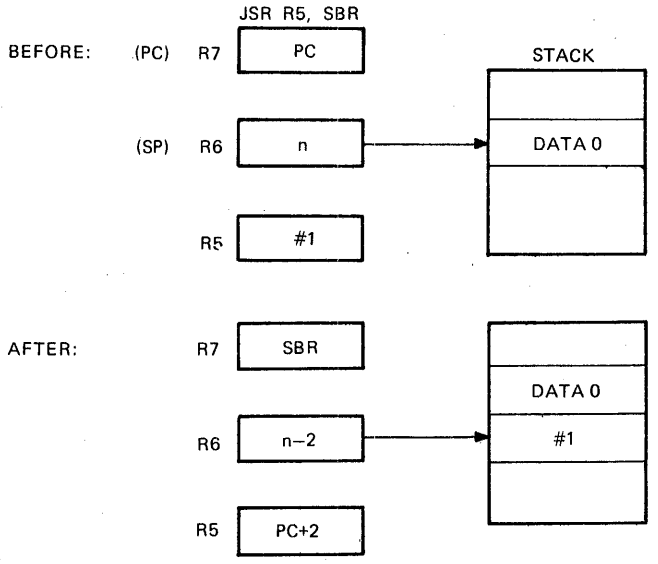
Return from a subroutine is done by the RTS instruction. RTS reg loads the contents of reg into the PC and pops the top element of the processor stack into the specified register.

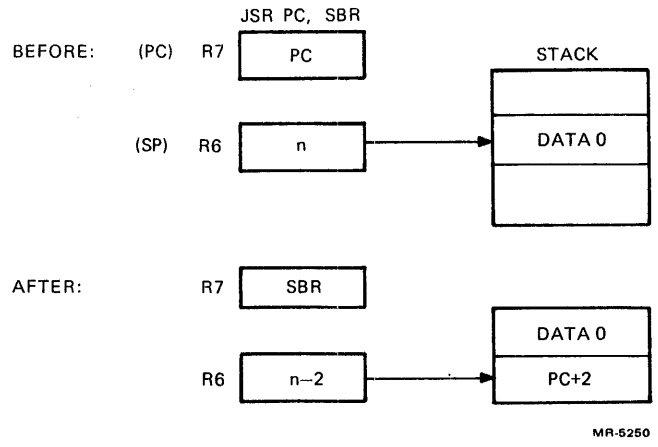
NOTE

JSR with register mode destination 0 is illegal and traps to 10.

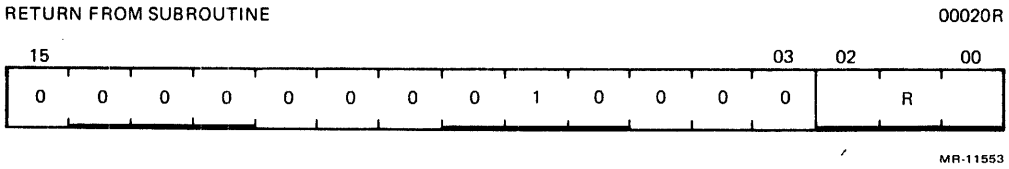
Example:

		R5	R6	R7
SBCALL:	JSR R5,SBR	#1	n	SBCALL
SBCALL+4:	ARG 1			
	ARG 2			
	.			
	.			
SBCALL+2+2M:	ARG M			
CONT:	Next Instruction	#1	n	CONT
	.			
	.			
SBR:	MOV (R5)+,dst 1	SBCALL+4	n–2	SBR
	MOV (R5)+,dst 2			
	.			
	MOV (R5)+,dst M	SBCALL+2+2M		
	Other Instructions	CONT		
EXIT:	RTS R5	CONT	n–2	EXIT





RTS

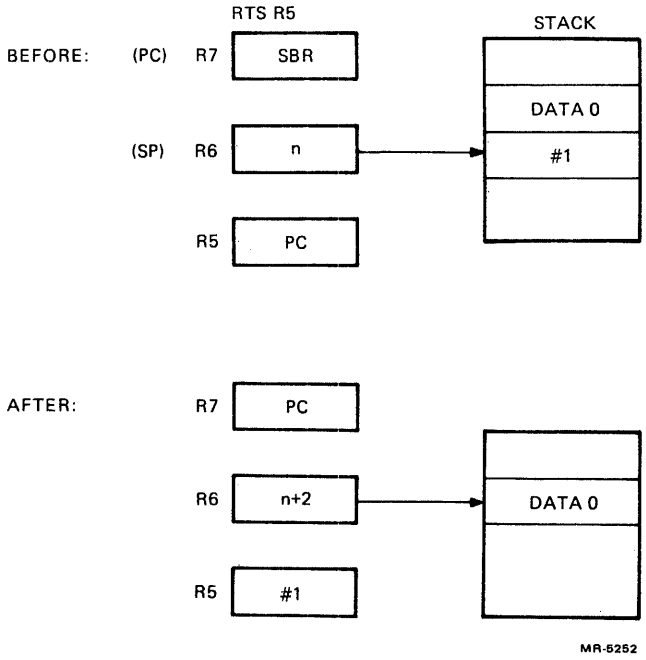


Operation: PC ← (reg)
(reg) ← (SP) ↑

Description: Loads the contents of the register into the PC and pops the top element of the processor stack into the specified register.

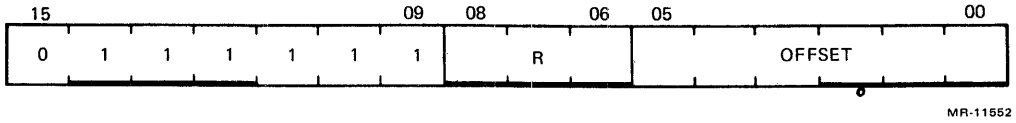
Return from a nonreentrant subroutine is typically made through the same register that was used in its call. Thus, a subroutine called with a JSR PC, dst exits with an RTS PC, and a subroutine called with a JSR R5, dst may pick up parameters with addressing modes (R5) +, X(R5), or @X(R5), and finally exits with an RTS R5.

Example: RTS R5



SOB

SUBTRACT ONE AND BRANCH (IF ≠ 0) 077RNN



Operation: (R) ← (R) − 1; if this result ≠ 0, then PC ← PC − (2 × offset); if (R) = 0 then PC ← PC

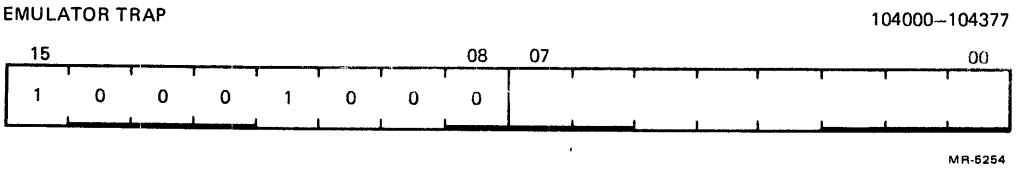
Condition Codes: Not affected

Description: The register is decremented. If the contents does not equal 0, twice the offset is subtracted from the PC (now pointing to the following word). The offset is interpreted as a 6-bit positive number. This instruction provides a fast, efficient method of loop control. The assembler syntax is SOB R,A where A is the address to which transfer is to be made if the decremented R is not equal to 0. Notice that the SOB instruction cannot be used to transfer control in the forward direction.

9.7.5 Traps

Trap instructions provide for calls to emulators, I/O monitors, debugging packages, and user-defined interpreters. A trap is effectively an interrupt generated by software. When a trap occurs, the contents of the current PC and PSW are pushed onto the processor stack and are replaced by the contents of a 2-word trap vector containing a new PC and new PSW. The return sequence from a trap involves executing an RTI or RTT instruction, which restores the old PC and old PSW by popping them from the stack. Trap instruction vectors are located at permanently assigned fixed addresses.

EMT



Operation:

↓ (SP) ← PSW
↓ (SP) ← PC
PC ← (30)
PSW ← (32)

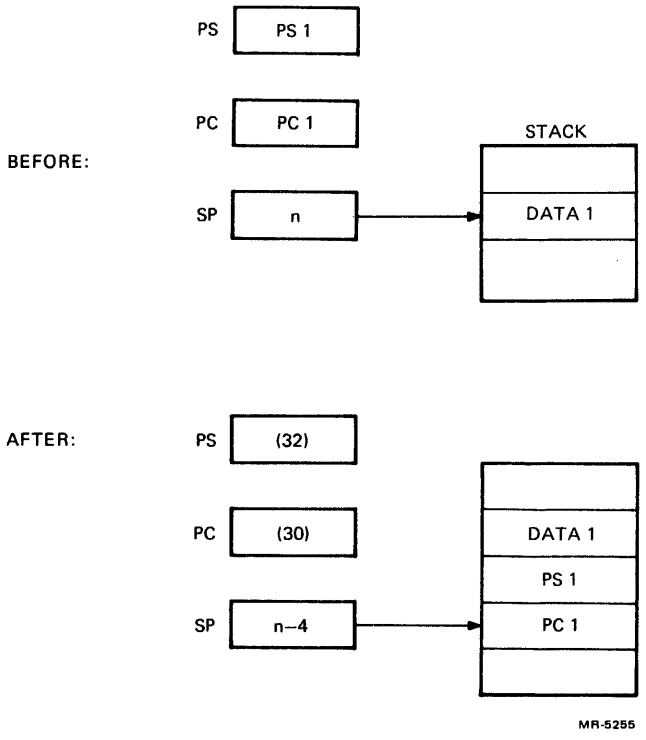
Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

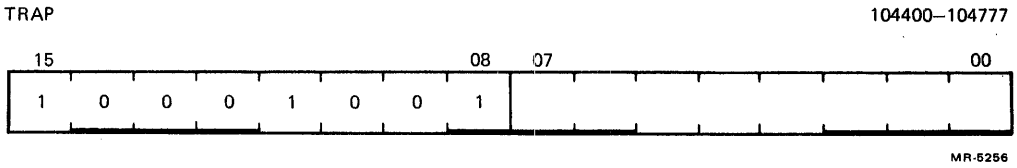
Description:

All operation codes from 104000 to 104377 are EMT instructions and may be used to transmit information to the emulating routine (e.g., the function to be performed). The trap vector for EMT is at address 30. The new PC is taken from the word at address 30; the new PSW is taken from the word at address 32.

NOTE
EMT is used frequently by Digital system software and is therefore not recommended for general use.



TRAP



Operation:

↓ (SP) ← PSW
↓ (SP) ← PC
PC ← (34)
PSW ← (36)

Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

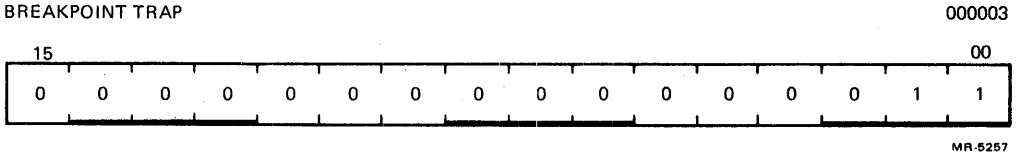
Description:

Operation codes from 104400 to 104777 are TRAP instructions. TRAPs and EMTs are identical in operation, except that the trap vector for TRAP is at address 34.

NOTE

Since Digital software makes frequent use of EMT, the TRAP instruction is recommended for general use.

BPT



Operation:

↓ (SP) ← PSW
↓ (SP) ← PC
PC ← (14)
PSW ← (16)

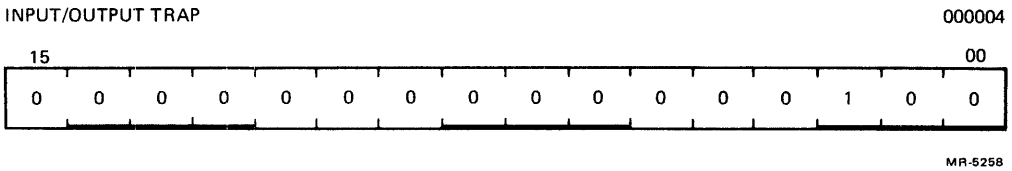
Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

Description:

Performs a trap sequence with a trap vector address of 14. Used to call debugging aids. The user is cautioned against employing code 000003 in programs run under these debugging aids. (No information is transmitted in the low byte.)

IOT



Operation:

↓ (SP) ← PSW
↓ (SP) ← PC
PC ← (20)
PSW ← (22)

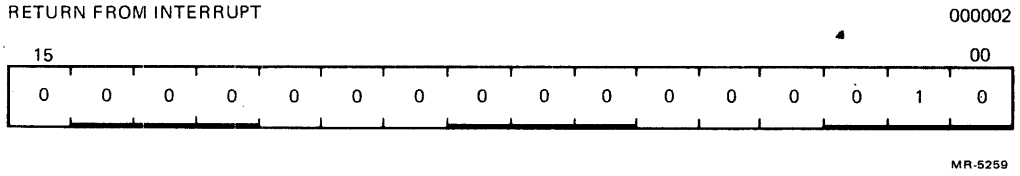
Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

Description:

Performs a trap sequence with a trap vector address of 20. (No information is transmitted in the low byte.)

RTI



Operation:

PC ← (SP) ↑
PSW ← (SP) ↑

Condition Codes:

N: loaded from processor stack
Z: loaded from processor stack
V: loaded from processor stack
C: loaded from processor stack

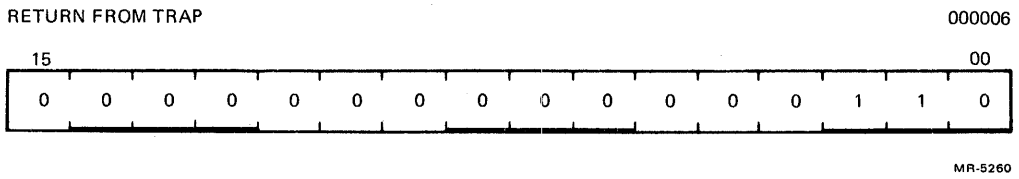
Description:

Used to exit from an interrupt or TRAP service routine. The PC and PSW are restored (popped) from the processor stack. If the RTI sets the T-bit in the PSW, a trace trap will occur prior to execution of the next instruction.

When executing in kernel mode, any legal mode can be stored in PSW <15:14, 13:12>. When executing in supervisor mode, only supervisor or user mode can be stored, and in user mode, only the user mode can be stored.

When executing in kernel mode, either a 1 or a 0 can be stored in PSW bit 11. When executing in supervisor mode, a stored 0 can be changed to a 1, but a stored 1 cannot be changed to a 0.

RTT



Operation:

PC ← (SP) ↑
PSW ← (SP) ↑

Condition Codes:

N: loaded from processor stack
Z: loaded from processor stack
V: loaded from processor stack
C: loaded from processor stack

Description:

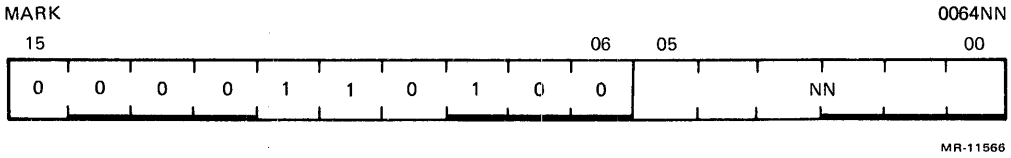
Operation is the same as RTI except that it inhibits a trace trap, whereas RTI permits a trace trap. If the new PSW has the T-bit set, a trap will occur after execution of the instruction following RTT.

When executing in kernel mode, any legal mode can be stored in PSW <15:14, 13:12>. When executing in supervisor mode, only supervisor or user mode can be stored, and in user mode, only the user mode can be stored.

When executing in kernel mode, either a 1 or a 0 can be stored in PSW bit 11. When executing in supervisor mode, a stored 0 can be changed to a 1, but a stored 1 cannot be changed to a 0.

9.7.6 Miscellaneous Program Controls

MARK



Operation:

SP ← PC + 2 × NN
PC ← R5
R5 ← (SP)+

(NN = number of parameters)

Condition Codes:

N: unaffected
Z: unaffected
V: unaffected
C: unaffected

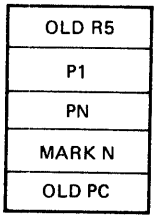
Description:

Used as part of the standard subroutine return convention. MARK facilitates the stack clean-up procedures involved in subroutine exit. Assembler format is: MARK N.

Example:

```
MOV R5,–(SP)          ;place old R5 on stack
MOV P1,–(SP)          ;place N parameters on
MOV P2,–(SP)          ;the stack to be used
                      ;there by the subroutine
MOV PN,–(SP)
MOV =MARKN,–(SP)      ;place the instruction
                      ;MARK N on the stack
MOV SP,R5             ;set up address at MARK N
                      ;instruction
JSR PC,SUB            ;jump to subroutine
```

At this point the stack is as follows.



MR-11569

The program is at the address SUB, which is the beginning of the subroutine.

```
SUB:                  ;execution of the
                      ;subroutine itself

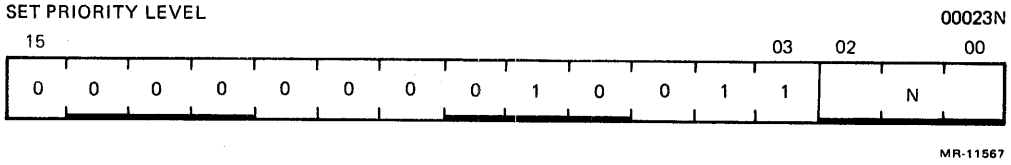
RTS R5                ;the return begins:
                      ;this causes the contents
                      ;of R5 to be placed in the
                      ;PC which then results in
                      ;the execution of the
                      ;instruction MARK N. The
                      ;contents of the old PC
                      ;are placed in R5.
```

MARK N causes: (1) the stack pointer to be adjusted to point to the old R5 value; (2) the value now in R5 (the old PC) to be placed in the PC; and (3) the contents of the old R5 to be popped into R5, thus completing the return from the subroutine.

NOTE

If memory management is in use, the stack must be mapped through both I and D space to execute the MARK instruction.

SPL



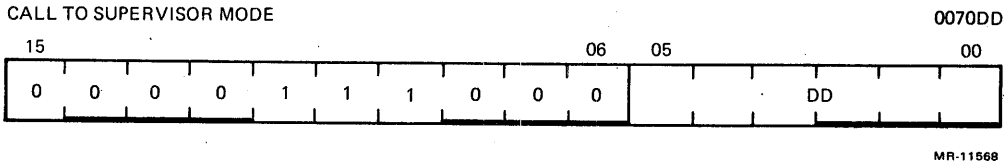
Operation: PSW bits <7:5> ← priority
(priority = N)

Condition Codes:
N: unaffected
Z: unaffected
V: unaffected
C: unaffected

Description: In kernel mode, the least significant three bits of the instruction are loaded into PSW bits <7:5>, thus causing a changed priority. The old priority is lost. In user or supervisor modes, SPL executes as an NOP.

Assembler syntax is: SPL N

CSM



Operation: If MMR3 bit 3 = 1, and
current mode = kernel, then
supervisor SP ← current mode SP
temp <15:4> ← PSW <15:4>
temp <3:0> ← 0
PSW <13:12> ← PSW <15:14>
PSW <15:14> ← 1
PSW 4 ← 0
−(SP) ← temp
−(SP) ← PC
−(SP) ← (dst)
PC ← (10);
otherwise, traps to 10 in kernel mode.

Condition Codes:
N: unaffected
Z: unaffected
V: unaffected
C: unaffected

Description: CSM may be executed in user or supervisor mode, but is an illegal instruction in kernel mode. CSM copies the current SP to the supervisor mode, switches to supervisor mode, stacks three words on the supervisor stack (the PSW with the condition codes cleared, the PC, and the argument word addressed by the operand), and sets the PC to the contents of location 10 (in supervisor space). The called program in supervisor space may return to the calling program by popping the argument word from the stack and executing RTI. On return, the condition codes are determined by the PSW on the stack. Hence, the called program in supervisor space may control the condition code values following return.

9.7.7 Reserved Instruction Traps

These are caused by attempts to execute instruction codes reserved for future processor expansion (reserved instructions) or instructions with illegal addressing modes (illegal instructions). Order codes not corresponding to any of the instructions described are considered to be reserved instructions. JMP and JSR with register mode destinations are illegal instructions; they trap to virtual address 10 in kernel data space. Reserved instructions trap to vector address 10 in kernel data space.

9.7.8 Trace Trap

Trace trap is enabled by bit 4 of the PSW and causes processor traps at the end of instruction execution. The instruction that is executed after the instruction that sets the T-bit proceeds to completion and then traps through the trap vector at address 14. The trace trap is a system debugging aid and is transparent to the general programmer.

NOTE

Bit 4 of the PSW can only be set indirectly by executing an RTI or RTT instruction with the desired PSW on the stack.

The following are special cases of the T-bit.

NOTE

The traced instruction is the instruction after the one that sets the T-bit.

1. An instruction that clears the T-bit – Upon fetching the traced instruction, an internal flag – the trace flag – is set. The trap still occurs at the end of this instruction. The PSW on the stack, however, has a clear T-bit.

2. An instruction that sets the T-bit – Since the T-bit is already set, setting it again has no effect. The trap still occurs.

3. An instruction that causes an instruction trap – The instruction trap is performed and the entire routine for the service trap is executed. If the service routine exits with an RTI, or in any other way restores the stacked PSW, the T-bit is set again, the instruction following the traced instruction is executed, and, unless it is one of the special cases noted previously, a trace trap occurs.

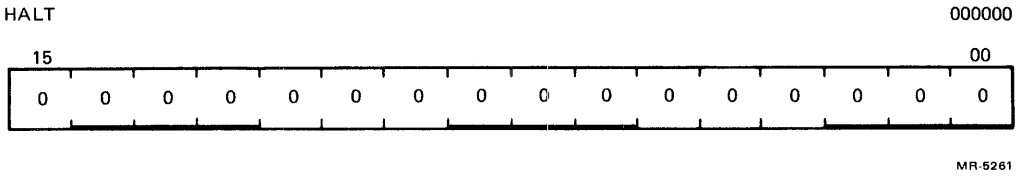
4. An instruction that causes a stack overflow – The instruction completes execution as usual. The stack overflow does not cause a trap. The trace trap vector is loaded into the PC and PSW and the old PC and PSW are pushed onto the stack. Stack overflow occurs again, and this time the trap is made.

5. An interrupt between setting the T-bit and fetching the traced instruction – The entire interrupt service routine is executed and then the T-bit is set again by the exiting RTI. The traced instruction is executed (if there have been no other interrupts), and, unless it is a special case noted above, a trace trap occurs.

6. Interrupt trap priorities – See Table 1-6.

9.8 MISCELLANEOUS INSTRUCTIONS

HALT



Operation:

↓ (SP) ← PSW
↓ (SP) ← PC
PC ← restart address
PSW ← 340

Condition Codes: Not affected

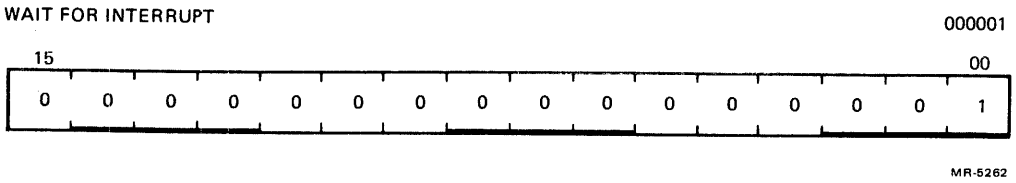
Description: The effect of HALT depends upon the CPU operating mode and the state of the trap-on-halt option (bit 3) in the maintenance register. Execution of the HALT instruction in kernel mode with the trap-on-halt option cleared causes the CPU to end the execution of instructions after the current instruction and enter the DCJ11 micro-ODT mode. Execution of the HALT instruction in kernel mode with the halt-on-trap option set, or at any time in supervisor or user modes, causes a trap through virtual address 4 and also sets bit 7 of the CPU error register.

NOTE

DMA activity may continue while the CPU is halted, even if the Halt switch is on.

The state of the halt-on-trap option has no effect on the operation of the Halt switch located on the operator console panel.

WAIT

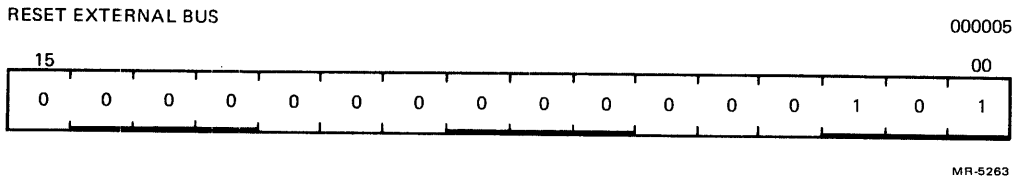


Condition Codes: Not affected

Description: The WAIT instruction allows the processor to relinquish the bus while it waits for an interrupt. During this time the processor does not compete for instructions or operands from memory. This may permit higher transfer rates between devices and memory, since there are no processor induced latencies by requests from the devices.

In WAIT, as in all instructions, the PC points to the next instruction following the WAIT instruction. Thus, when an interrupt causes the PC and PSW to be pushed onto the processor stack, the address of the next instruction following the WAIT is saved. The exit from the interrupt routine causes resumption of the interrupted process at the instruction following the WAIT. The WAIT instruction executes as an NOP in supervisor and user modes.

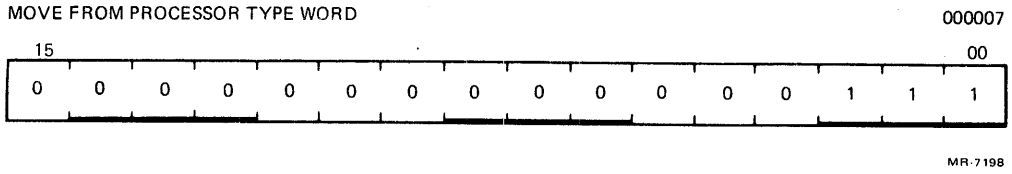
RESET



Condition Codes: Not affected

Description: The following sequence of events occurs: (1) a general purpose write cycle is performed and a general purpose code of 014 is generated; (2) operation is delayed for 69 microcycles; (3) a general purpose write is performed and a general purpose code of 214 is generated; and (4) operation is delayed for 600 microcycles. If not in kernel mode, RESET operates as an NOP.

MFPT



Operation: R0 ← 5

Condition Codes: Not affected

Description: The number 5 is placed in R0, indicating to the system software that the processor type is a CPU designed to use the DCJ11 microprocessor. The value returned by this instruction does not guarantee that the CPU is a KDJ11-B. The KDJ11-A CPU also returns the same value because it too uses the DCJ11 microprocessor. The system program should read the maintenance register and check bits <7:4> to determine the exact type of microprocessor being used. The specific values contained in the maintenance register are shown below:

	Maintenance Register Bits			
CPU Type	**7**	**6**	**5**	**4**
KDJ11-A	0	0	0	1
KDJ11-B	0	0	1	0

In addition, maintenance register bit 9 is used to further define the type of system. If this bit is set, the system is Unibus based and if this bit is cleared, the system is LSI bus based.

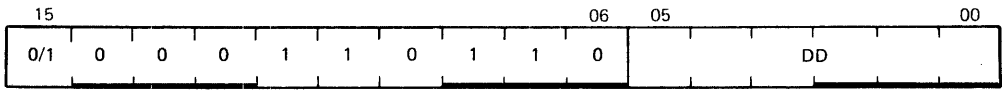
NOTE

The following PDP-11 CPUs implement the MFPT instruction. The chart shows the value returned to R0 when the instruction is executed. The other PDP-11 CPUs treat this instruction as a reserved instruction and trap through virtual address 10.

Contents of R0	**PDP-11 CPU Type**	**Microprocessor**
1	PDP-11/44	
3	KDF11-A, -B, -UA	DCF11
4	KXT11-AA, -AB, -CA	DCT11
5	KDJ11-A, -B	DCJ11

MTPD/MTPI

MOVE TO PREVIOUS DATA SPACE (BIT 15 = 1)
MOVE TO PREVIOUS INSTRUCTION SPACE (BIT 15 = 0) ■066DD



MR-11571

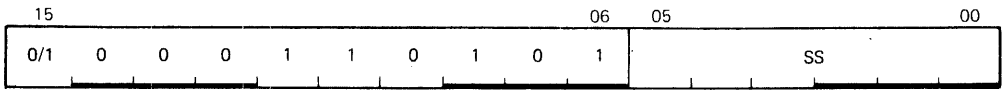
Operation:
(temp) ← (SP)+
(dst) ← (temp)

Condition Codes:
N: set if the source < 0
Z: set if the source = 0
V: cleared
Z: unaffected

Description:
The instruction pops a word off the current stack determined by PSW <15:14> and stores that word in an address in the previous space (PSW <13:12>). The destination address is computed using the current registers and memory map.

MFPD/MFPI

MOVE FROM PREVIOUS DATA SPACE (BIT 15 = 1)
MOVE FROM PREVIOUS INSTRUCTION SPACE (BIT 15 = 0) ■065SS



MR-11570

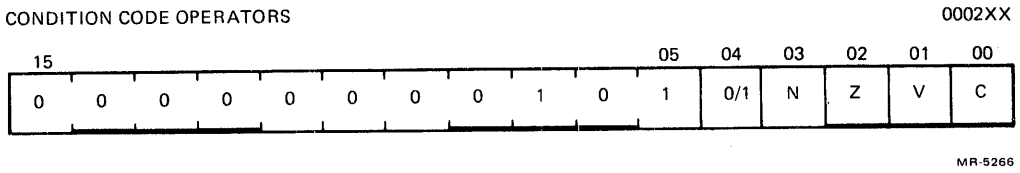
Operation:
(temp) ← (src)
–(SP) ← (temp)

Condition Codes:
N: set if the source < 0
Z: set if the source = 0
V: cleared
Z: unaffected

Description:
Pushes a word onto the current stack from an address in the previous space determined by PSW <13:12>. The source address is computed using the current registers and memory map. When MFPI is executed and both previous mode and current mode are user, the instruction functions as though it were MFPD.

9.9 CONDITION CODE OPERATORS

CLN SEN
CLZ SEZ
CLV SEV
CLC SEC
CCC SCC



Description:

Set and clear condition code bits. Selectable combinations of these bits may be cleared or set together. Condition code bits corresponding to bits in the condition code operator (bits <3:0>) are modified according to the sense of bit 4, the set/clear bit of the operator. That is, set the bit specified by bit 0, 1, 2, or 3, if bit 4 = 1. Clear corresponding bits if bit 4 = 0.

Mnemonic	Operation	Op Code
CLC	Clear C	000241
CLV	Clear V	000242
CLZ	Clear Z	000244
CLN	Clear N	000250
SEC	Set C	000261
SEV	Set V	000262
SEZ	Set Z	000264
SEN	Set N	000270
SCC	Set all CCs	000277
CCC	Clear all CCs	000257
	Clear V and C	000243
NOP	No operation	000240

Combinations of the above set and clear operations may be ORed together to form combined instructions.

CHAPTER 10
FLOATING-POINT ARITHMETIC

10.1 INTRODUCTION

The KDJ11-B executes 46 floating-point instructions. The floating-point instruction set is compatible with the FP11 instruction set for PDP-11 computers. Both single- and double-precision floating-point capabilities are available with other features, including floating-to-integer and integer-to-floating conversion.

10.2 FLOATING-POINT DATA FORMATS

Mathematically, a floating-point number may be defined as having the form (2 ** K) * f, where K is an integer and f is a fraction. For a nonvanishing number, K and f are uniquely determined by imposing the condition $1/2 < f < 1$. The fractional part (f) of the number is then said to be 'normalized.' For the number 0, f is assigned the value 0, and the value of K is indeterminate.

The floating-point data formats are derived from this mathematical representation for floating-point numbers. Two types of floating-point data are provided. In single-precision, or floating mode, the data is 32 bits long. In double-precision, or double mode, the data is 64 bits long. Sign magnitude notation is used.

10.2.1 Nonvanishing Floating-Point Numbers

The fractional part (f) is assumed normalized, so that its most significant bit must be 1. This 1 is the 'hidden' bit. It is not stored explicitly in the data word, but the microcode restores it before carrying out arithmetic operations. The floating and double modes reserve 23 and 55 bits, respectively, for f. These bits, with the hidden bit, imply effective word lengths of 24 bits and 56 bits.

Eight bits are reserved for storage of the exponent K in excess 200 notation (i.e., as K + 200 octal), giving a biased exponent. Thus, exponents from −128 to +127 may be represented by 0 to 377 (base 8), or 0 to 255 (base 10). For reasons given below, a biased exponent of 0 (the true exponent of −200 octal) is reserved for floating-point 0. Therefore, exponents are restricted to the range −127 to +127 inclusive (−177 to +177 octal) or, in excess 200 notation, 1 to 377.

The remaining bit of the floating-point word is the sign bit. The number is negative if the sign bit is a 1.

10.2.2 Floating-Point Zero

Because of the hidden bit, the fractional part is not available to distinguish between 0 and nonvanishing numbers whose fractional part is exactly 1/2. Therefore, the KDJ11-B reserves a biased exponent of 0 for this purpose, and any floating-point number with a biased exponent of 0 either traps or is treated as if it were an exact 0 in arithmetic operations. An exact or 'clean' 0 is represented by a word whose bits are all 0s. A 'dirty' 0 is a floating-point number with a biased exponent of 0 and a nonzero fractional part. An arithmetic operation for which the resulting true exponent exceeds 177 octal is regarded as producing a floating overflow; if the true exponent is less than −177 octal, the operation is regarded as producing a floating underflow. A biased exponent of 0 can thus arise from arithmetic operations as a special case of overflow (true exponent = −200 octal). (Recall that only eight bits are reserved for the biased exponent.) The fractional part of results obtained from such overflow and underflow is correct.

10.2.3 Undefined Variables

An undefined variable is any bit pattern with a sign bit of 1 and a biased exponent of 0. The term 'undefined variable' is used, for historical reasons, to indicate that these bit patterns are not assigned a corresponding floating-point arithmetic value. Note that the undefined variable is frequently referred to as −0 elsewhere in this chapter.

A design objective was to ensure that the undefined variable would not be stored as the result of any floating-point operation in a program run with the overflow and underflow interrupts disabled. This is achieved by storing an exact 0 on overflow and underflow if the corresponding interrupt is disabled. This feature, together with an ability to detect reference to the undefined variable (implemented by the FIUV bit discussed later), is intended to provide the user with a debugging aid: If −0 occurs, it did not result from a previous floating-point arithmetic instruction.

10.2.4 Floating-Point Data

Floating-point data is stored in words of memory as illustrated in Figures 10-1 and 10-2.

The KDJ11-B provides for conversion of floating-point to integer format and vice versa. The processor recognizes single-precision integer (I) and double-precision integer long (L) numbers, which are stored in standard 2's complement form. (See Figure 10-3.)

10.3 FLOATING-POINT STATUS REGISTER (FPS)

This register provides mode and interrupt control for the currently executing floating-point instruction and also reflects conditions resulting from the execution of the previous instruction. (See Figure 10-4.) In this discussion a set bit = 1 and a reset bit = 0. Three bits of the FPS register control the modes of operation as follows.

Single/Double – Floating-point numbers can be either single- or double-precision.

Long/Short – Integer numbers can be 16 bits or 32 bits.

Chop/Round – The result of a floating-point operation can be either 'chopped' or 'rounded.' The term 'chop' is used instead of 'truncate' to avoid confusion with truncation of series used in approximations for function subroutines.

The FPS register contains an error flag and four condition codes (5 bits): carry, overflow, zero, and negative, which are analogous to the CPU condition codes.

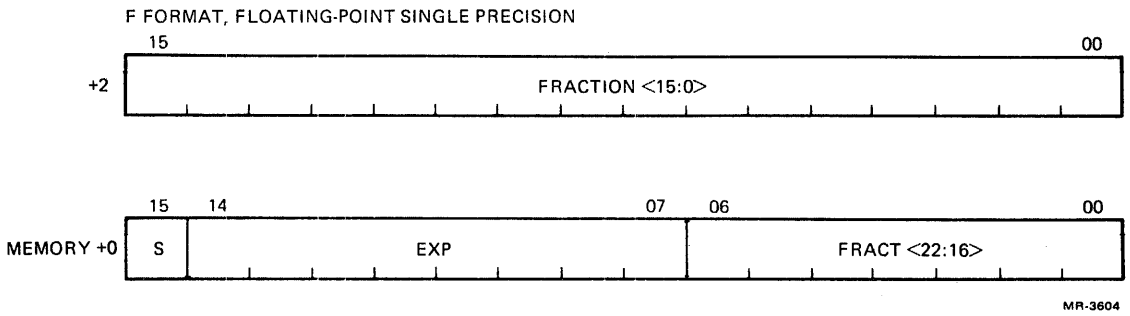


Figure 10-1 Single-Precision Format

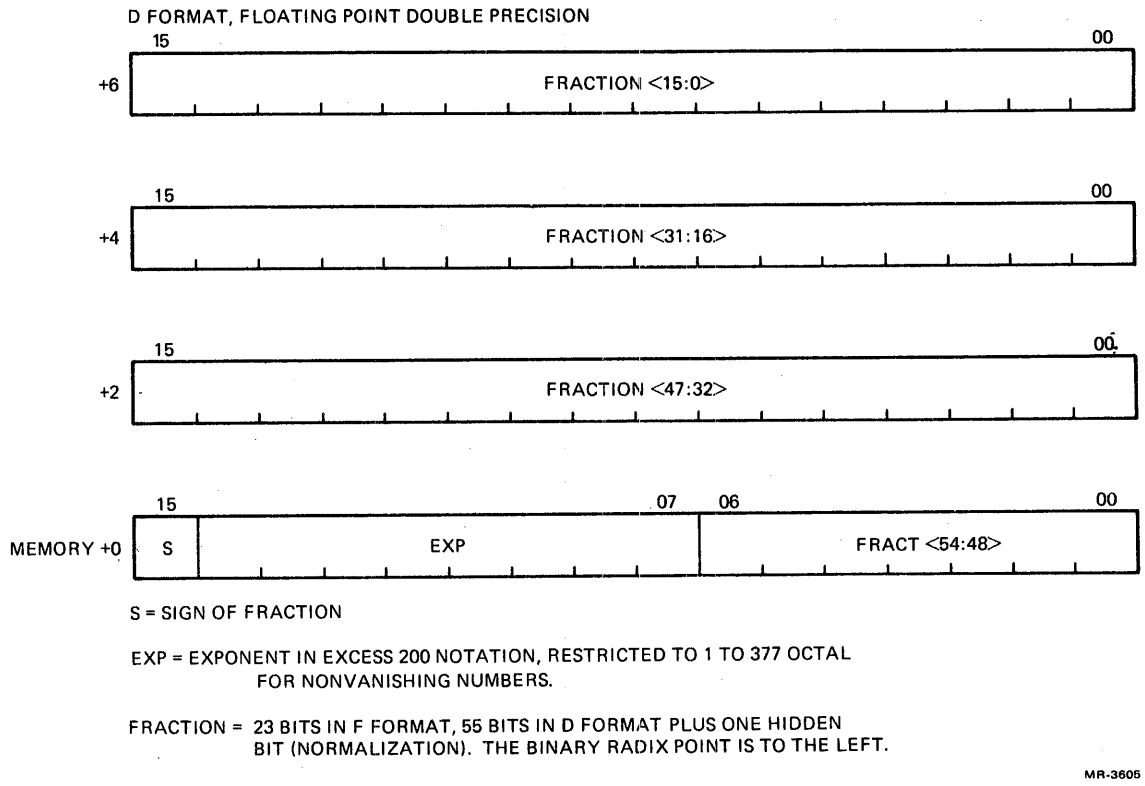


Figure 10-2 Double-Precision Format

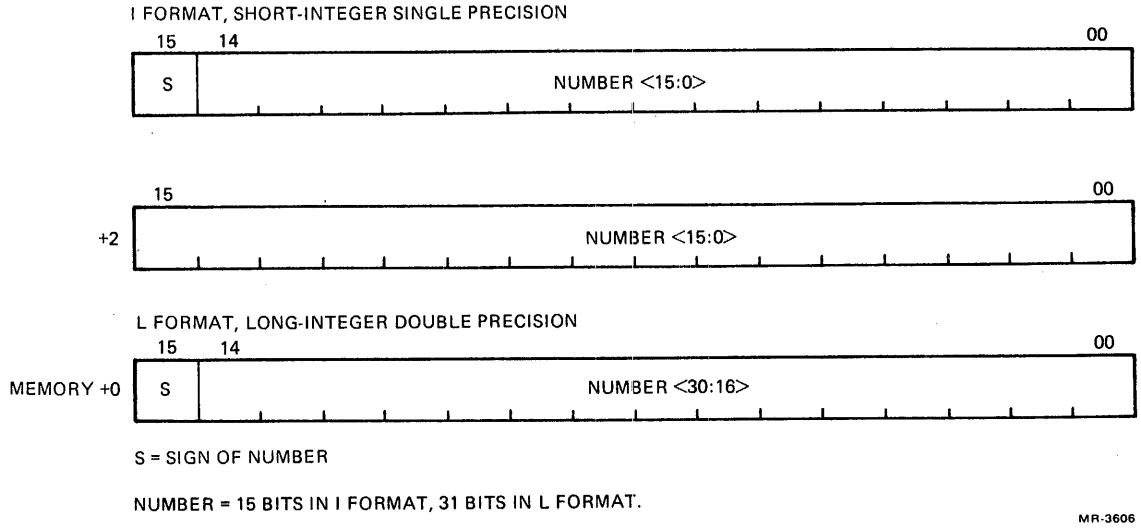


Figure 10-3 2's Complement Format

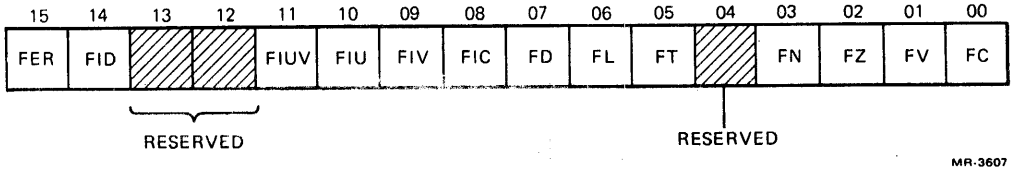


Figure 10-4 Floating-Point Status Register

The KDJ11-B recognizes the following six floating-point exceptions.

- Detection of the presence of the undefined variable in memory
- Floating overflow
- Floating underflow
- Failure of floating-to-integer conversion
- Attempt to divide by 0
- Illegal floating op code

For the first four of these exceptions, bits in the FPS register are available to individually enable and disable interrupts. An interrupt on the occurrence of either of the last two exceptions can be disabled only by setting a bit that disables interrupts on all six of the exceptions as a group.

Of the 13 FPS bits, 5 are set as part of the output of a floating-point instruction: the error flag and condition codes. Any of the mode and interrupt control bits may be set by the user; the LDFPS instruction is available for this purpose. These 13 bits are stored in the FPS register as shown in Figure 10-4. The FPS register bits are described in Table 10-1.

Table 10-1 FPS Register Bit Description

Bit	Name	Function
15	FER	The Floating ERror (FER) bit is set by the KDJ11-A if: 1. Division by zero occurs, 2. An illegal op code occurs, 3. Any one of the remaining floating-point exceptions occurs and the corresponding interrupt is enabled. Note that the above action is independent of whether the FID bit is set or clear. Note also that the KDJ11-A never resets the FER bit. Once the FER bit is set by the KDJ11-A, it can be cleared only by an LDFPS instruction. (The RESET instruction does not clear the FER bit.) This means that the FER bit is up to date only if the most recent floating-point instruction produced a floating-point exception.
14	FID	If the FID bit is set, all floating-point interrupts are disabled. **NOTE** **The FID bit is primarily a maintenance feature. It is normally clear and it must be clear if one wishes to assure that storage of −0 by the KDJ11-A is accompanied by an interrupt.** **Throughout the rest of this chapter, assume that the FID bit is clear in all discussions involving overflow, underflow, occurrence of −0, and integer conversion errors.**

Table 10-1 FPS Register Bit Description (Cont)

Bit	Name	Function
13	Reserved	Reserved for future use.
12	Reserved	Reserved for future use.
11	FIUV	An interrupt occurs if FIUV is set and a −0 is obtained from memory as an operand of ADD, SUB, MUL, DIV, CMP, MOD, NEG, ABS, TST, or any LOAD instruction. The interrupt occurs before execution on all instructions. When FIUV is reset, −0 can be loaded and used in any floating-point operation. Note that the interrupt is not activated by the presence of −0 in an AC operand of an arithmetic instruction. In particular, trap on −0 never occurs in mode 0. A result of −0 is not stored without the simultaneous occurrence of an interrupt.
10	FIU	When the FIU bit is set, floating underflow causes an interrupt. The fractional part of the result of the operation causing the interrupt is correct. The biased exponent is too large by 400, except for the special case of 0, which is correct. A special case is discussed later in the detailed description of the LDEXP instruction.
9	FIV	When the FIV bit is set, floating overflow causes an interrupt. The fractional part of the result of the operation causing the overflow is correct. The biased exponent is too small by 400. If the FIV bit is reset and overflow occurs, there is no interrupt. The KDJ11-A returns exact 0. Special cases of overflow are discussed later in the detailed descriptions of the MOD and LDEXP instructions.
8	FIC	When the FIC bit is set and a conversion to integer instruction fails, an interrupt occurs. When the interrupt occurs, the destination is set to 0 and all other registers are left untouched. If the FIC bit is reset, the result of the operation is the same as that detailed above, but no interrupt occurs. The conversion instruction fails if it generates an integer with more bits than can fit in the short or long integer word specified by the FL bit.
7	FD	The FD bit determines the precision that is used for floating-point calculations. When set, double-precision is assumed. When reset, single-precision is used.
6	FL	The FL bit is active in conversion between integer and floating-point formats. When set, the integer format assumed is double-precision 2's complement (i.e., 32 bits). When reset, the integer format assumed is single-precision 2's complement (i.e., 16 bits).

Table 10-1 FPS Register Bit Description (Cont)

Bit	Name	Function
5	FT	When the FT bit is set, the result of any arithmetic operation is chopped (truncated). When reset, the result is rounded.
4	Reserved	Reserved for future use.
3	FN	FN is set if the previous floating-point operation result was negative; otherwise it is reset.
2	FZ	FZ is set if the previous floating-point operation result was 0; otherwise it is reset.
1	FV	FV is set if the previous floating-point operation resulted in an exponent overflow; otherwise it is reset.
0	FC	FC is set if the previous floating-point operation resulted in a carry of the most significant bit.

10.4 FLOATING EXCEPTION CODE AND ADDRESS REGISTERS

One interrupt vector is assigned to take care of all floating-point exceptions (location 244). The six possible errors are coded in the 4-bit Floating Exception Code (FEC) register as follows.

Code	Exception
2	Floating op code error
4	Floating divide by zero error
6	Floating-to-integer or double-to-integer conversion error
8	Floating overflow error
10	Floating underflow error
12	Floating undefined variable error

The address of the instruction producing the exception is stored in the Floating Exception Address (FEA) register.

The FEC and FEA registers are updated only when one of the following occurs.

- Division by zero
- Illegal op code
- Any of the other four exceptions with the corresponding interrupt enabled

This implies that the FEC and FEA registers are updated only when the FER bit is set.

NOTES

1. **If one of the last four exceptions occurs with the corresponding interrupt disabled, the FEC and FEA are not updated.**

2. **If an exception occurs, inhibition of interrupts by the FID bit does not inhibit updating of the FEC and FEA.**

3. **The FEC and FEA are not updated if no exception occurs. This means that the STST (store status) instruction returns current information only if the most recent floating-point instruction produced an exception.**

4. **Unlike the FPS, no instructions are provided for storage into the FEC and FEA registers.**

10.5 FLOATING-POINT INSTRUCTION ADDRESSING

Floating-point instructions use the same type of addressing as the central processor instructions. A source or destination operand is specified by designating one of eight addressing modes and one of eight central processor general registers to be used in the specified mode. The modes of addressing are the same as those of the central processor, except in mode 0. In mode 0, the operand is located in the designated floating-point processor accumulator rather than in a central processor general register. The modes of addressing are as follows.

0 = Floating-point accumulator
1 = Deferred
2 = Autoincrement
3 = Autoincrement-deferred
4 = Autodecrement
5 = Autodecrement-deferred
6 = Index
7 = Index-deferred

Autoincrement and autodecrement operate on increments and decrements of 4 for F format, and 10 (octal) for D format.

In mode 0, all six floating-point accumulators (AC0–AC5) may be used as source or destination. Specifying floating-point accumulators AC6 or AC7 results in an illegal op code trap. In all other modes, which involve transfer of data to or from memory or the general registers, users are restricted to the first four floating-point accumulators (AC0–AC3). When reading or writing a floating-point number to or from memory, the low memory word contains the most significant word of the floating-point number, and the high memory word the least significant word.

10.6 ACCURACY

General comments on the accuracy of the KDJ11-B floating-point instructions are presented here. The descriptions of the individual instructions include the accuracy at which they operate. An instruction or operation is regarded as 'exact' if the result is identical to an infinite precision calculation involving the same operands. The a priori accuracy of the operands is thus ignored. All arithmetic instructions treat an operand whose biased exponent is 0 as an exact 0 (unless FIUV is enabled and the operand is −0, in which case an interrupt occurs). For all arithmetic operations except DIV, a 0 operand implies that the instruction is exact. The same statement holds for DIV if the 0 operand is the dividend. But if it is the divisor, division is undefined and an interrupt occurs.

For nonvanishing floating-point operands, the fractional part is binary normalized. It contains 24 bits or 56 bits for floating mode and double mode, respectively. For ADD, SUB, MUL, and DIV, two guard bits are necessary and sufficient for the general case, to guarantee return of a chopped or rounded result identical to the corresponding infinite precision operation chopped or rounded to the specified word length. Thus, with two guard bits, a chopped result has an error bound of one Least Significant Bit (LSB); a rounded result has an error bound of 1/2 LSB. These error bounds are realized by the KDJ11-B for all instructions.

In the rest of this chapter, an arithmetic result is called exact if no nonvanishing bits would be lost by chopping. The first bit lost in chopping is referred to as the 'rounding' bit. The value of a rounded result is related to the chopped result as follows.

1. If the rounding bit is 1, the rounded result is the chopped result incremented by an LSB.

2. If the rounding bit is 0, the rounded and chopped results are identical.

It follows that

1. If the result is exact:

   Rounded value = chopped value = exact value.

2. If the result is not exact, its magnitude is

   - always decreased by chopping,
   - decreased by rounding if the rounding bit is 0,
   - increased by rounding if the rounding bit is 1.

Occurrence of floating-point overflow and underflow is an error condition; the result of the calculation cannot be correctly stored because the exponent is too large to fit into the eight bits reserved for it. However, the internal hardware has produced the correct answer. In the case of underflow, replacement of the correct answer with 0 is a reasonable resolution of the problem for many applications. This is done by the KDJ11-B if the underflow interrupt is disabled. The error incurred by this action is an absolute rather than a relative error; it is bounded (in absolute value) by 2 ** −128. There is no such simple resolution for the case of overflow. The action taken, if the overflow interrupt is disabled, is described under FIV (bit 09) in Table 10-1.

The FIV and FIU bits (of the floating-point status word) provide users with an opportunity to implement their own correction of an overflow or underflow condition. If such a condition occurs and the corresponding interrupt is enabled, the microcode stores the fractional part and the low eight bits of the biased exponent. When the interrupt takes place, users can identify the cause by examination of the floating overflow (FV) bit or the FEC. The reader can readily verify that (for the standard arithmetic operations ADD, SUB, MUL, and DIV) the biased exponent returned by the instruction bears the following relation to the correct exponent.

- On overflow, it is too small by 400 (octal).
- On underflow, if the biased exponent is 0, it is correct. If the biased exponent is not 0, it is too large by 400 (octal).

Thus, with the interrupt enable, enough information is available to determine the correct answer. Users may, for example, rescale their variables (via STEXP and LDEXP) to continue a calculation. Note that the accuracy of the fractional part is unaffected by the occurrence of underflow or overflow.

10.7 FLOATING-POINT INSTRUCTIONS

Each instruction that references a floating-point number can operate on either single- or double-precision numbers, depending on the state of the FD mode bit. Similarly, there is an FL mode bit that determines whether a 32-bit integer (FL = 1) or a 16-bit integer (FL = 0) is used in conversion between integer and floating-point representations. FSRC and FDST operands use floating-point addressing modes (Figure 10-5); SRC and DST operands use CPU addressing modes.

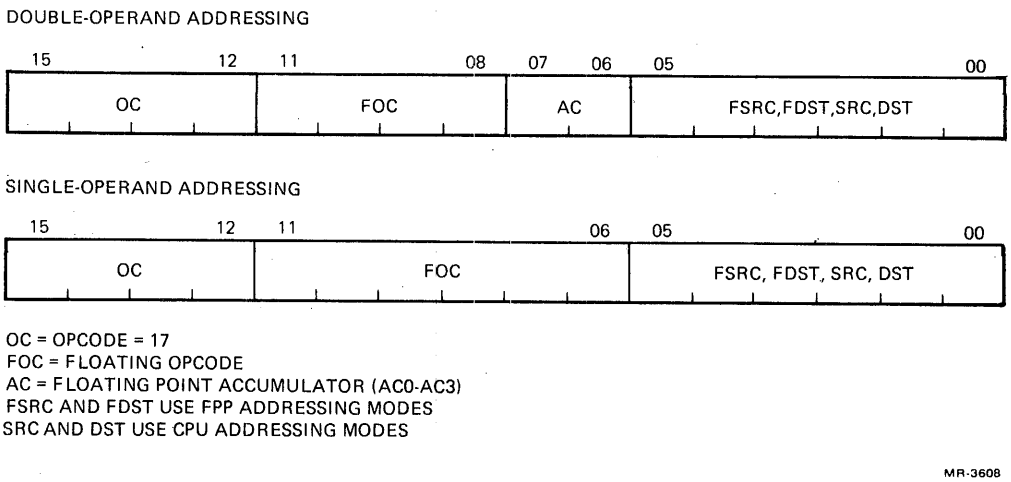


Figure 10-5 Floating-Point Addressing Modes

Terms Used in Instruction Definitions

OC = op code = 17

FOC = floating op code

AC = contents of accumulator, as specified by AC field of instruction

FSRC = address of floating-point source operand

FDST = address of floating-point destination operand

f = fraction

XL = largest fraction that can be represented:

1 − 2 ** (−24), FD = 0; single-precision
1 − 2 ** (−56), FD = 1; double-precision

XLL = smallest number that is not identically zero =

2 ** (−128)

XUL = largest number that can be represented =

2 ** (127) * XL

JL = largest integer that can be represented:

2 ** (15) − 1; FL = 0; short integer
2 ** (31) − 1; FL = 1; long integer

ABS (address) = absolute value of (address)

EXP (address) = biased exponent of (address)

.LT. = less than

.LE. = less than or equal to

.GT. = greater than

.GE. = greater than or equal to

LSB = least significant bit

Boolean Symbols

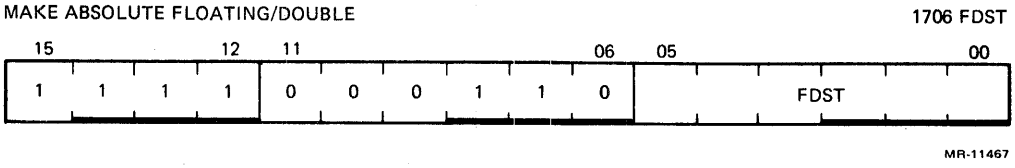
$\wedge$ = AND

$\vee$ = inclusive OR

$\forall$ = exclusive OR

$\sim$ = NOT

ABSF/ABSD



Format: ABSF FDST

Operation: If (FDST) < 0, (FDST) ← − (FDST).

If EXP(FDST) = 0, (FDST) ← exact 0.

For all other cases, (FDST) ← (FDST).

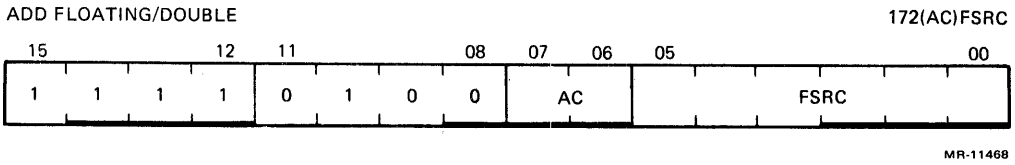
Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (FDST) = 0, else FZ ← 0
FN ← 0

Description: Set the contents of FDST to its absolute value.

Interrupts: If FIUV is enabled, trap on −0 occurs before execution. Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

ADDF/ADDD



Format: ADDF FSRC,AC

Operation: Let SUM = (AC) + (FSRC).

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all others cases, AC ← SUM.

Condition Codes:
FC ← 0
FV ← 1 if overflow occurs, else FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: Add the contents of FSRC to the contents of AC. The addition is carried out in single- or double-precision and is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for

- Overflow with interrupt disabled
- Underflow with interrupt disabled

For these exceptional cases, an exact 0 is stored in AC.

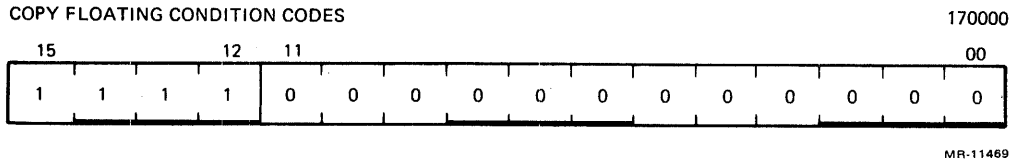
Interrupts: If FIUV is enabled, trap on −0 in FSRC occurs before execution. If overflow or underflow occurs, and if the corresponding interrupt is enabled, the trap occurs with the faulty result in AC. The fractional parts are correctly stored. The exponent part is too small by 400 for overflow. It is too large by 400 for underflow, except for the special case of 0, which is correct.

Accuracy: Errors due to overflow and underflow are described above. If neither occurs, then for oppositely signed operands with exponent difference of 0 or 1, the answer returned is exact if a loss of significance of one or more bits can occur. Note that these are the only cases for which loss of significance of more than one bit can occur. For all other cases the result is inexact with error bounds of

- LSB in chopping mode with either single- or double-precision
- 1/2 LSB in rounding mode with either single- or double-precision

Special Comment: The undefined variable −0 can occur only in conjunction with overflow or underflow. It is stored in AC only if the corresponding interrupt is enabled.

CFCC

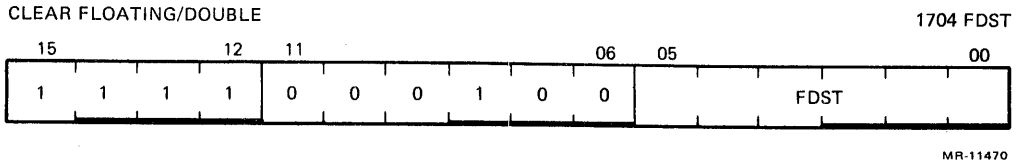


Format: CFCC

Operation:
C ← FC
V ← FV
Z ← FZ
N ← FN

Description: Copy the floating-point condition codes into the CPU condition codes.

CLRF/CLRD



Format: CLRF FDST

Operation: (FDST) ← exact 0

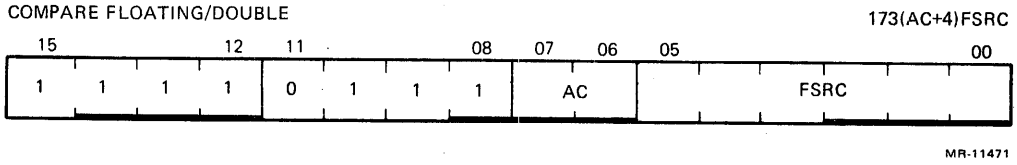
Condition Codes:
FC ← 0
FV ← 0
FZ ← 1
FN ← 0

Description: Set FDST to 0. Set FZ condition code and clear other condition code bits.

Interrupts: No interrupts occur. Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

CMPF/CMPD



Format: CMPF FSRC,AC

Operation: (FSRC) − (AC)

Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (FSRC) = 0, else FZ ← 0
FN ← 1 if (FSRC) < 0, else FN ← 0

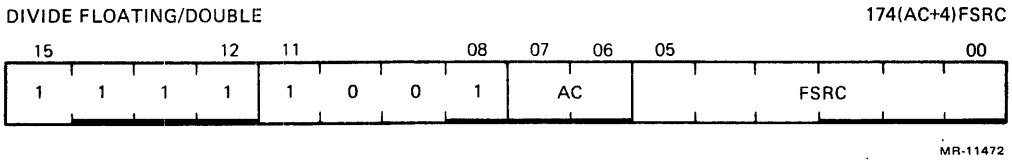
Description: Compare the contents of FSRC with the accumulator. Set the appropriate floating-point condition codes. FSRC and the accumulator are left unchanged except as noted below.

Interrupts: If FIUV is enabled, trap on −0 occurs before execution.

Accuracy: These instructions are exact.

Special Comment: An operand that has a biased exponent of 0 is treated as if it were an exact 0. In this case, where both operands are 0, the KDJ11-B stores an exact 0 in AC.

DIVF/DIVD



Format: DIVF FSRC,AC

Operation: If EXP(FSRC) = 0, (AC) ← (AC) and the instruction is aborted.

If EXP(AC) = 0, (AC) ← exact 0.

For all other cases, let QUOT = (AC)/(FSRC).

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all others cases, AC ← QUOT.

Condition Codes:
FC ← 0
FV ← 1 if overflow occurs, else FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: If either operand has a biased exponent of 0, it is treated as an exact 0. For FSRC this would imply division by 0; in this case, the instruction is aborted, the FEC register is set to 4, and an interrupt occurs. Otherwise, the quotient is developed to single- or double-precision with two guard bits for correct rounding. The quotient is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in the AC except for

- Overflow with interrupt disabled
- Underflow with interrupt disabled

For these exceptional cases, an exact 0 is stored in AC.

Interrupts: If FIUV is enabled, trap on −0 in FSRC occurs before execution. If (FSRC) = 0, interrupt traps occur on an attempt to divide by 0. If overflow or underflow occurs, and if the corresponding interrupt is enabled, the trap occurs with the faulty result in AC. The fractional parts are correctly stored. The exponent part is too small by 400 for overflow. It is too large by 400 for underflow, except for the special case of 0, which is correct.

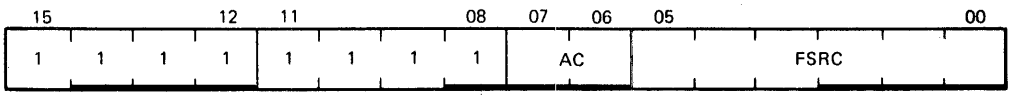
Accuracy: Errors due to overflow and underflow are described above. If none of these occurs, the error in the quotient is bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode.

Special Comment: The undefined variable −0 can occur only in conjunction with overflow or underflow. It is stored in AC only if the corresponding interrupt is enabled.

LDCDF/LDCFD

LOAD AND CONVERT FROM DOUBLE-TO-FLOATING
AND FROM FLOATING-TO-DOUBLE

177(AC+4)FSRC



MR-11473

Format: LDCDF FSRC,AC

Operation: If EXP(FSRC) = 0, AC ← exact 0.

If FD = 1, FT = 0, FIV = 0 and rounding causes overflow, AC ← exact 0.

In all other cases, AC ← Cxy(FSRC), where Cxy specifies conversion from floating mode x to floating mode y.

x = D, y = F if FD = 0 (single) LDCDF
y = F, y = D if FD = 1 (double) LDCFD

Condition Codes:
FC ← 0
FV ← 1 if conversion produces overflow, else
FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: If the current mode is floating mode (FD = 0), the source is assumed to be a double-precision number and is converted to single-precision. If the floating chop bit (FT) is set, the number is chopped; otherwise, the number is rounded.

If the current mode is double mode (FD = 1), the source is assumed to be a single-precision number and is loaded left-justified in AC. The lower half of AC is cleared.

Interrupts: If FIUV is enabled, trap on −0 occurs before execution. Overflow cannot occur for LDCFD.

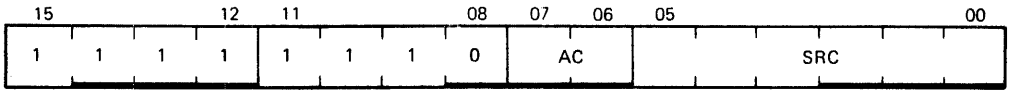
A trap occurs if FIV is enabled and if rounding with LDCDF causes overflow. AC ← overflowed result. This result must be +0 or −0. Underflow cannot occur.

Accuracy: LDCFD is an exact instruction. Except for overflow (see above), LDCDF incurs an error bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode.

LDCIF/LDCID/LDCLF/LDCLD

LOAD AND CONVERT INTEGER OR LONG INTEGER
TO FLOATING OR DOUBLE-PRECISION

177(AC)SRC



MR-11474

Format: LDCIF SRC,AC

Operation: AC ← Cjx(SRC), where Cjx specifies conversion from integer mode j to floating mode x.

j = I if FL = 0, j = L if FL = 1
x = F if FD = 0, x = D if FD = 1

Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: Conversion is performed on the contents of SRC from a 2's complement integer with precision j to a floating-point number of precision x. Note that j and x are determined by the state of the mode bits FL and FD.

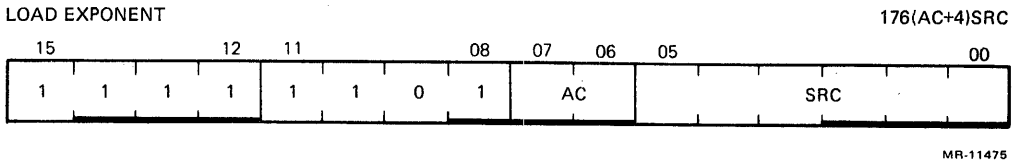
If a 32-bit integer is specified (L mode) and (SRC) has an addressing mode of 0 or immediate addressing mode is specified, the 16 bits of the source register are left-justified and the remaining 16 bits are loaded with 0s before conversion.

In the case of LDCLF, the fractional part of the floating-point representation is chopped or rounded to 24 bits for FT = 1 or 0, respectively.

Interrupts: None. SRC is not floating-point, so trap on −0 cannot occur.

Accuracy: LDCIF, LDCID, and LDCLD are exact instructions. The error incurred by LDCLF is bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode.

LDEXP



Format: LDEXP SRC,AR

Operation: (Note that 177 and 200, appearing throughout this instruction definition, are octal numbers.)

If $-200 < SRC < 200$, EXP(AC) ← SRC + 200 and the rest of AC is unchanged.

If (SRC) > 177 and FIV is enabled, EXP(AC) ← [(SRC) + 200]<7:0>.

If (SRC) > 177 and FIV is disabled, AC ← exact 0.

If (SRC) < −177 and FIU is enabled, EXP(AC) ← [(SRC) + 200]<7:0>.

If (SRC) < −177 and FIU is disabled, AC ← exact 0.

Condition Codes:
FC ← 0
FV ← 1 if (SRC) > 177, else FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: Change AC so that its unbiased exponent = (SRC). That is, convert (SRC) from 2's complement to excess 200 notation and insert it into the EXP field of AC. This is a meaningful operation only if ABS(SRC) LE 177.

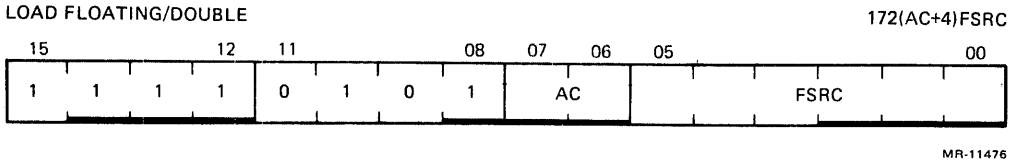
If SRC > 177, the result is treated as overflow. If SRC < −177, the result is treated as underflow.

Interrupts: No trap on −0 in AC occurs, even if FIUV is enabled. If SRC > 177 and FIV is enabled, trap on overflow occurs. If SRC < −177 and FIU is enabled, trap on underflow occurs.

Accuracy: Errors due to overflow and underflow are described above. If EXP(AC) = 0 and (SRC) = −200, AC changes from a floating-point number treated as 0 by all floating arithmetic operations to a nonzero number. This happens because the insertion of the 'hidden' bit in the microcode implementation of arithmetic instructions is triggered by a nonvanishing value of EXP.

For all other cases, LDEXP implements exactly the transformation of a floating-point number (2 ** K) * f into (2 ** (SRC)) * f where 1/2 .LE. ABS(f) .LT. 1.

LDF/LDD



Format: LDF FSRC,AC

Operation: AC ← (FSRC)

Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

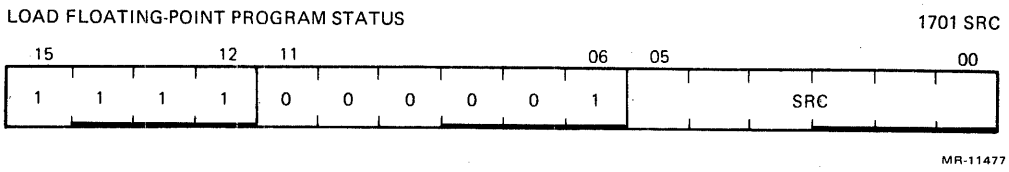
Description: Load single- or double-precision number into AC.

Interrupts: If FIUV is enabled, trap on −0 occurs before AC is loaded. Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

Special Comment: These instructions permit use of −0 in a subsequent floating-point instruction if FIUV is not enabled and (FSRC) = −0.

LDFPS



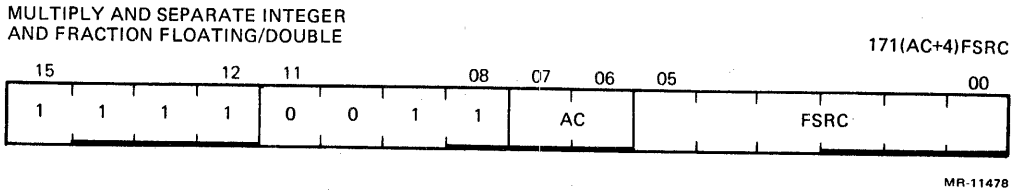
Format: LDFPS SRC

Operation: FPS ← (SRC)

Description: Load floating-point status register from SRC.

Special Comment: Users are cautioned not to use bits 13, 12, and 4 for their own purposes, since these bits are not recoverable by the STFPS instruction.

MODF/MODD



Format: MODF FSRC,AC

Description and Operation: This instruction generates the product of its two floating-point operands, separates the product into integer and fractional parts, and then stores one or both parts as floating-point numbers.

Let PROD = (AC) * (FSRC) so that in

Floating-point: ABS(PROD) = (2 ** K) * f, where

1/2 .LE. f .LT. 1, and EXP(PROD) = (200 + K).

Fixed-point binary: PROD = N + g, where

N = INT(PROD) = integer part of PROD, and

g = PROD − INT(PROD) = fractional part of PROD with 0 .LE. g .LT. 1.

Both N and g have the same sign as PROD. They are returned as follows.

If AC is an even-numbered accumulator (0 or 2), N is stored in AC + 1 (1 or 3), and g is stored in AC.

If AC is an odd-numbered accumulator, N is not stored and g is stored in AC.

These two statements can be combined as:

N is returned to AC ∨ 1 and g is returned to AC.

Five special cases occur, as indicated in the following formal description with L = 24 for floating mode and L = 56 for double mode.

1. If PROD overflows and FIV is enabled, AC ∨ 1 ← N, chopped to L bits, AC ← exact 0.

   Note that EXP(N) is too small by 400 and that −0 can be stored in AC ∨ 1.

   If FIV is not enabled, AC ∨ 1 ← exact 0, AC ← exact 0, and −0 will never be stored.

2. If 2 ** L .LE. ABS(PROD) and no overflow, AC ∨ 1 ← N, chopped to L bits, AC ← exact 0.

   The sign and EXP of N are correct, but low-order bit information is lost.

3. If 1 .LE. ABS(PROD) .LT. 2 ** L, AC ∨ 1 ← N, AC ← g.

   The integer part N is exact. The fractional part g is normalized and chopped or rounded in accordance with FT. Rounding may cause a return of + unity for the fractional part. For L = 24, the error in g is bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode. For L = 56, the error in g increases from the limits above as ABS(N) increases above 8, because only 59 bits of PROD are generated.

   If 2 ** p .LE. ABS(N) .LT. 2 ** (p + 1), with p > 2, the low order p − 2 bits of g may be in error.

4. If ABS(PROD) .LT. 1 and no underflow, AC ∨ 1 ← exact 0 and AC ← g.

   There is no error in the integer part. The error in the fractional part is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode. Rounding may cause a return of + unity for the fractional part.

5. If PROD underflows and FIU is enabled, AC ∨ 1 ← exact 0 and AC ← g.

   Errors are as in case 4, except that EXP(AC) is too large by 400_8 (if EXP = 0, it is correct). Interrupt occurs and −0 can be stored in AC.

   If FIU is not enabled, AC ∨ 1 ← exact 0 and AC ← exact 0.

   For this case the error in the fractional part is less than 2 ** (−128).

Condition Codes:

FC ← 0
FV ← 1 if PROD overflows, else FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Interrupts:

If FIUV is enabled, trap on −0 in FSRC occurs before execution. Overflow and underflow are described above.

Accuracy:

Described above.

Applications:

1. Binary-to-decimal conversion of a proper fraction. The following algorithm, using MOD, generates decimal digits D(1), D(2) · · · from left to right.

```
Initialize:   I ← 0;
              X ← number to be converted;
              ABS(X) < 1;
While:        X ≠ 0
Begin:        PROD ← X * 10;
              I ← I + 1;
              D(I) ← INT(PROD);
              X ← PROD − INT(PROD);
End.
```

This algorithm is exact. It is case 3 in the description because the number of nonvanishing bits in the fractional part of PROD never exceeds L, and hence neither chopping nor rounding can introduce error.

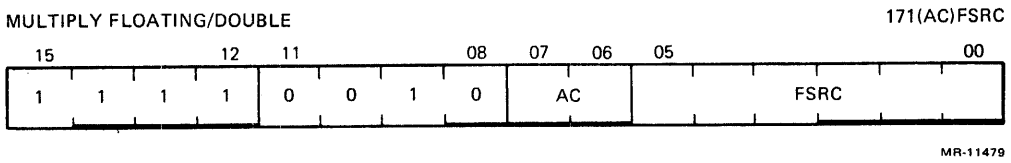
2. To reduce the argument of a trigonometric function.

   ARG * 2/PI = N + g. The two low bits of N identify the quadrant, and g is the argument reduced to the first quadrant. The accuracy of N + g is limited to L bits because of the factor 2/PI. The accuracy of the reduced argument thus depends on the size of N.

3. To evaluate the exponential function e ** x, obtain x * (log e base 2) = N + g, then e ** x = (2 ** N) * (e ** (g * ln 2)).

   The reduced argument is g * ln2 < 1 and the factor 2 ** N is an exact power of 2, which may be scaled in at the end via STEXP, ADD N to EXP and LDEXP. The accuracy of N + g is limited to L bits because of the factor (log e base 2). The accuracy of the reduced argument thus depends on the size of N.

MULF/MULD



Format: MULF FSRC,AC

Operation: Let PROD = (AC) * (FSRC).

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all others cases, AC ← PROD.

Condition Codes:
FC ← 0
FV ← 1 if overflow occurs, else FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: If the biased exponent of either operand is 0, (AC) ← exact 0. For all other cases PROD is generated to 48 bits for floating mode and 59 bits for double mode. The product is rounded or chopped for FT = 0 or 1, respectively, and is stored in AC except for

- Overflow with interrupt disabled
- Underflow with interrupt disabled

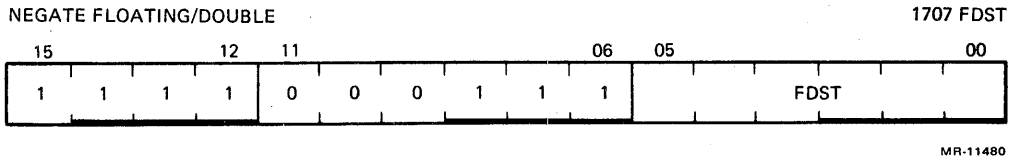
For these exceptional cases, an exact 0 is stored in AC.

Interrupts: If FIUV is enabled, trap on −0 in FSRC occurs before execution. If overflow or underflow occurs, and if the corresponding interrupt is enabled, the trap occurs with the faulty result in AC. The fractional parts are correctly stored. The exponent part is too small by 400 for overflow. It is too large by 400 for underflow, except for the special case of 0, which is correct.

Accuracy: Errors due to overflow and underflow are described above. If neither occurs, the error incurred is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode.

Special Comment: The undefined variable −0 can occur only in conjunction with overflow or underflow. It is stored in AC only if the corresponding interrupt is enabled.

NEGF/NEGD



Format: NEGF FDST

Operation: (FDST) ← − (FDST) if (FDST) = 0, else (FDST) ← exact 0.

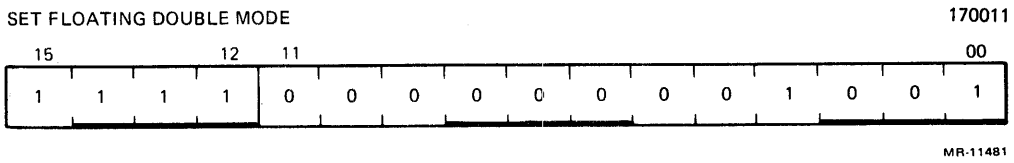
Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (FDST) = 0, else FZ ← 0
FN ← 1 if (FDST) < 0, else FN ← 0

Description: Negate the single- or double-precision number and store result in same location (FDST).

Interrupts: If FIUV is enabled, trap on −0 occurs before execution. Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

SETD

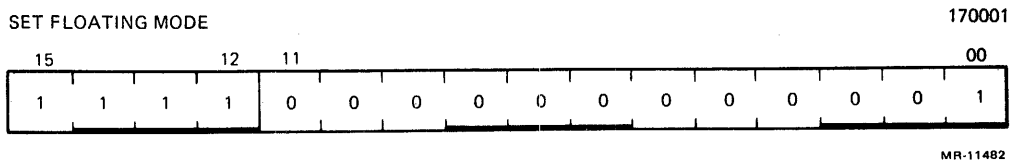


Format: SETD

Operation: FD ← 1

Description: Set the KDJ11-B in double-precision mode.

SETF

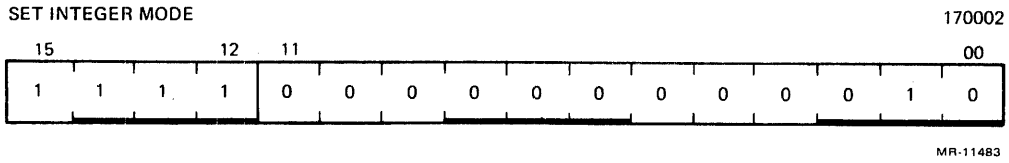


Format: SETF

Operation: FD ← 0

Description: Set the KDJ11-B in single-precision mode.

SETI

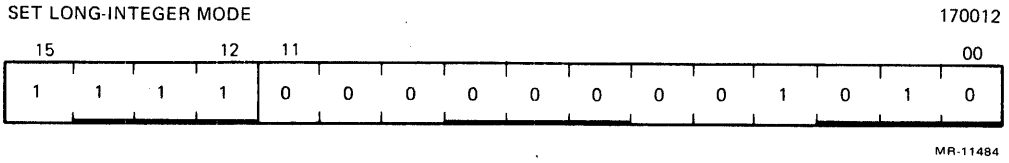


Format: SETI

Operation: FL ← 0

Description: Set the KDJ11-B for short-integer data.

SETL

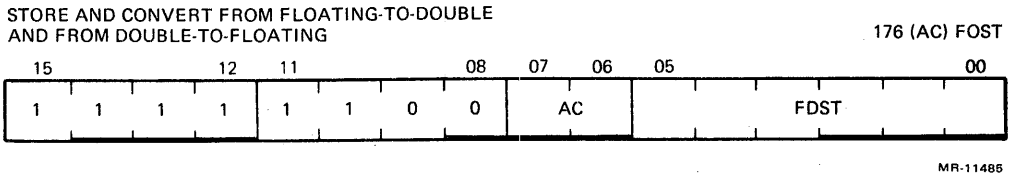


Format: SETL

Operation: FL ← 1

Description: Set the KDJ11-B for long-integer data.

STCFD/STCDF



Format: STCFD AC,FDST

Operation: If (AC) = 0, (FDST) ← exact 0.

If FD = 1, FT = 0, FIV = 0 and rounding causes overflow, (FDST) ← exact 0.

In all other cases, (FDST) ← Cxy(AC), where Cxy specifies conversion from floating mode x to floating mode y.

x = F, y = D if FD = 0 (single) STCFD
x = D, y = F if FD = 1 (double) STCDF

Condition Codes: FC ← 0
FV ← 1 if conversion produces overflow, else
FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: If the current mode is single-precision, the accumulator is stored left-justified in FDST and the lower half is cleared.

If the current mode is double-precision, the contents of the accumulator are converted to single-precision, chopped or rounded depending on the state of FT, and stored in FDST.

Interrupts: Trap on −0 does not occur even if FIUV is enabled because FSRC is an accumulator. Underflow cannot occur. Overflow cannot occur for STCFD.

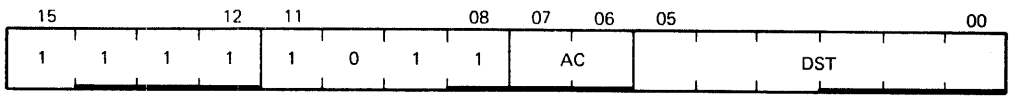
A trap occurs if FIV is enabled and if rounding with STCDF causes overflow. (FDST) ← overflowed result. This result must be +0 or −0.

Accuracy: STCFD is an exact instruction. Except for overflow (see above), STCDF incurs an error bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode.

STCFI/STCFL/STCDI/STCDL

STORE AND CONVERT FROM FLOATING OR DOUBLE
TO INTEGER OR LONG INTEGER

175(AC+4)DST



MR-11486

Format: STCFI AC,DST

Operation: (DST) ← Cxj(AC) if −JL − 1 < Cxj(AC) < JL + 1, else (DST) ← 0, where Cjx specifies conversion from floating mode j to integer mode x.

j = I if FL = 0, j = L if FL = 1
x = F if FD = 0, x = D if FD = 1

JL is the largest integer.

2 ** 15 − 1 for FL = 0
2 ** 32 − 1 for FL = 1

Condition Codes:
C, FC ← 0 if −JL − 1 < Cxj(AC) < JL + 1, else
C, FC ← 1
V, FV ← 0
Z, FZ ← 1 if (DST) = 0, else Z, FZ ← 0
N, FN ← 1 if (DST) < 0, else N, FN ← 0

Description: Conversion is performed from a floating-point representation of the data in the accumulator to an integer representation.

If the conversion is to a 32-bit word (L mode), and an addressing mode of 0 or immediate addressing mode is specified, only the most significant 16 bits are stored in the destination register.

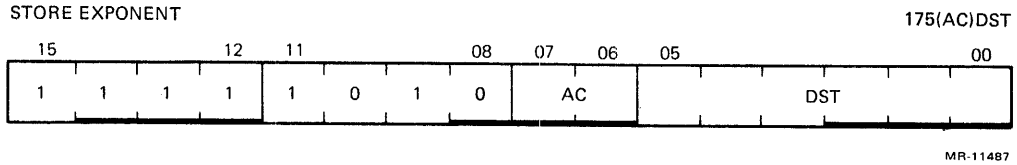
If the operation is out of the integer range selected by FL, FC is set to 1 and the contents of the DST are set to 0.

Numbers to be converted are always chopped (rather than rounded) before they are converted. This is true even when chop mode bit FT is cleared in the FPS register.

Interrupts: These instructions do not interrupt if FIUV is enabled, because the −0 (if present) is in AC, not in memory. If FIC is enabled, trap on conversion failure occurs.

Accuracy: These instructions store the integer part of the floating-point operand, which may not be the integer most closely approximating the operand. They are exact if the integer part is within the range implied by FL.

STEXP



Format: STEXP AC,DST

Operation: (DST) ← EXP(AC) − 200.

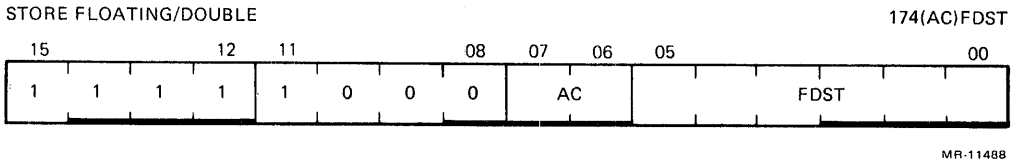
Condition Codes:
C, FC ← 0
V, FV ← 0
Z, FZ ← 1 if (DST) = 0, else Z, FZ ← 0
N, FN ← 1 if (DST) < 0, else N, FN ← 0

Description: Convert the AC exponent from excess 200 notation to 2's complement and store the result in DST.

Interrupts: This instruction does not trap on −0. Overflow and underflow cannot occur.

Accuracy: This instruction is exact.

STF/STD



Format: STF AC,FDST

Operation: (FDST) ← AC

Condition Codes:
FC ← FC
FV ← FV
FZ ← FZ
FN ← FN

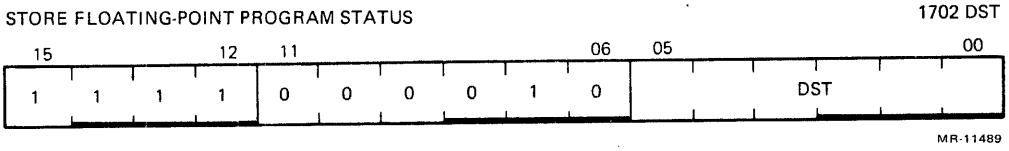
Description: Store single- or double-precision number from AC.

Interrupts: These instructions do not interrupt if FIUV is enabled, because the −0 (if present) is in AC, not in memory. Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

Special Comment: These instructions permit storage of a −0 in memory from AC. There are two conditions in which −0 can be stored in an AC of the KDJ11-B. One occurs when underflow or overflow is present and the corresponding interrupt is enabled. A second occurs when an LDF or LDD instruction is executed and the FIUV bit is disabled.

STFPS



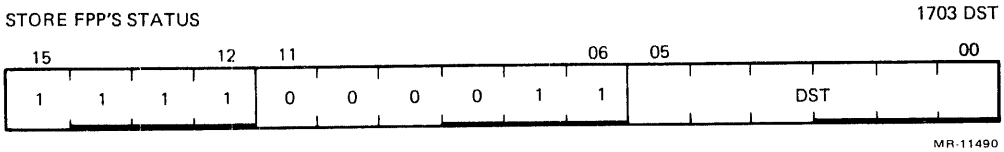
Format: STFPS DST

Operation: (DST) ← FPS

Description: Store the floating-point status register in DST.

Special Comment: Bits 13, 12, and 4 are loaded with 0. All other bits are the corresponding bits in the FPS.

STST



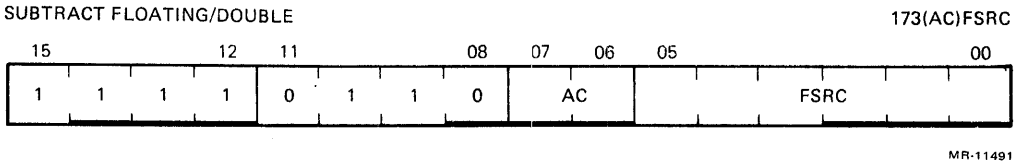
Format: STST DST

Operation: (DST) ← FEC (DST + 2) ← FEA.

Description: Store the FEC and FEA in DST and DST + 2. Note the following.

- If the destination mode specifies a general register or immediate addressing, only the FEC is saved.
- The information in these registers is current only if the most recently executed floating-point instruction caused a floating-point exception.

SUBF/SUBD



Format: SUBF FSRC,AC

Operation: Let DIFF = (AC) − (FSRC).

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all others cases, AC ← DIFF.

Condition Codes:
FC ← 0
FV ← 1 if overflow occurs, else FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0
FN ← 1 if (AC) < 0, else FN ← 0

Description: Subtract the contents of FSRC from the contents of AC. The subtraction is carried out in single- or double-precision and is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for

- Overflow with interrupt disabled
- Underflow with interrupt disabled

For these exceptional cases, an exact 0 is stored in AC.

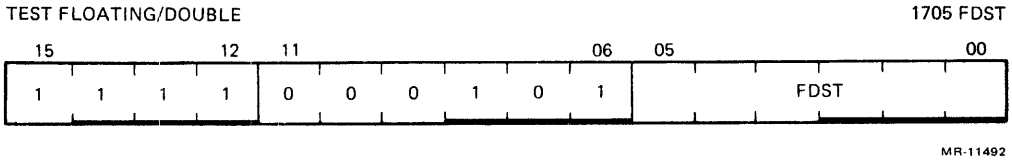
Interrupts: If FIUV is enabled, trap on −0 in FSRC occurs before execution. If overflow or underflow occurs, and if the corresponding interrupt is enabled, the trap occurs with the faulty result in AC. The fractional parts are correctly stored. The exponent part is too small by 400 for overflow. It is too large by 400 for underflow, except for the special case of 0, which is correct.

Accuracy: Errors due to overflow and underflow are described above. If neither occurs, then for like-signed operands with exponent difference of 0 or 1, the answer returned is exact if a loss of significance of one or more bits can occur. Note that these are the only cases for which loss of significance of more than one bit can occur. For all other cases the result is inexact with error bounds of

- LSB in chopping mode with either single- or double-precision
- 1/2 LSB in rounding mode with either single- or double-precision

Special Comment: The undefined variable −0 can occur only in conjunction with overflow or underflow. It is stored in AC only if the corresponding interrupt is enabled.

TSTF/TSTD



Format: TSTF FDST

Operation: (FDST)

Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (FDST) = 0, else FZ ← 0
FN ← 1 if (FDST) < 0, else FN ← 0

Description: Set the floating-point condition codes according to the contents of FDST.

Interrupts: If FIUV is set, trap on −0 occurs before execution. Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

CHAPTER 11
PROGRAMMING TECHNIQUES

11.1 INTRODUCTION

The KDJ11-B offers a great deal of programming flexibility and power. Utilizing the combination of the instruction set, the addressing modes, and the programming techniques, it is possible to develop new software or to utilize old programs effectively. The programming techniques in this chapter show the capabilities of the KDJ11-B. The techniques discussed involve PIC, stacks, subroutines, interrupts, reentrancy, coroutines, recursion, processor traps, programming peripherals, and conversion.

11.2 POSITION-INDEPENDENT CODE

The output of a MACRO-11 assembly is a relocatable object module. The task builder or linker binds one or more modules together to create an executable task image. Once built, a task can only be loaded and executed at the virtual address specified at link time. This is because the linker has had to modify some instructions to reflect the memory locations in which the program is to run. Such a body of code is considered position-dependent (i.e., dependent on the virtual addresses to which it is bound).

The KDJ11-B processor offers addressing modes that make it possible to write instructions that do not depend on the virtual addresses to which they are bound. This type of code is termed position-independent and can be loaded and executed at any virtual address. PIC can improve system efficiency, both in use of virtual address space and in conservation of physical memory.

In multiprogramming systems like RSX-11M, it is important that many tasks be able to share a single physical copy of common code (e.g., a library routine). To make optimum use of the virtual address space of a task, shared code should be position-independent. Code that is not position-independent can also be shared, but it must appear in the same virtual locations in every task using it. This restricts the placement of such code by the task builder and can result in the loss of virtual addressing space.

11.2.1 Use of Addressing Modes in the Construction of Position-Independent Code

The construction of PIC is closely linked to the proper use of addressing modes. The remainder of this explanation assumes the reader to be familiar with the addressing modes described in Chapter 6.

The following addressing modes, which involve only register references, are position-independent.

R	Register mode
(R)	Register-deferred mode
(R)+	Autoincrement mode
@(R)+	Autoincrement-deferred mode
−(R)	Autodecrement mode
@−(R)	Autodecrement-deferred mode

When employing these addressing modes, the user is guaranteed position independence, providing the contents of the registers are supplied independently of a particular virtual memory location.

The following two relative addressing modes are position-independent when a relocatable address is referenced from a relocatable instruction.

A	Relative mode
@A	Relative-deferred mode

Relative modes are not position-independent when an absolute address (that is, a nonrelocatable address) is referenced from a relocatable instruction. In such a case, absolute addressing (i.e., @#A) may be employed to make the reference position-independent.

Index modes can be either position-independent or position-dependent, according to their use in the program.

X(R)	Index mode
@X(R)	Index-deferred mode

If the base, X, is an absolute value (e.g., a control block offset), the reference is position-independent. The following is an example.

```
MOV     2(SP),R0        ;POSITION-INDEPENDENT
```

N=4

```
MOV     N(SP),R0        ;POSITION-INDEPENDENT
```

If, however, X is a relocatable address, the reference is position-dependent, as the following example shows.

```
CLR     ADDR(R1)        ;POSITION-DEPENDENT
```

Immediate mode can be either position-independent or not, according to its use. Immediate mode references are formatted as follows.

#N	Immediate mode

When an absolute expression defines the value of N, the code is position-independent. When a relocatable expression defines N, the code is position-dependent. That is, immediate mode references are position-independent only when N is an absolute value.

Absolute mode addressing is position-independent only in those cases where an absolute virtual location is being referenced. Absolute mode addressing references are formatted as follows.

@#A	Absolute mode

An example of a position-independent absolute reference is a reference to the PSW from a relocatable instruction, as in this example.

```
MOV     @#PSW,R0        ;RETRIEVE STATUS AND PLACE IN REGISTER
```

11.2.2 Comparison of Position-Dependent and Position-Independent Code

The RSX-11 library routine, PWRUP, is a FORTRAN-callable subroutine for establishing or removing a user power failure, Asynchronous System Trap (AST) entry point address. Embedded within the routine is the actual AST entry point that saves all registers, effects a call to the user-specified entry point, restores all registers on return, and executes an AST exit directive. The following examples are excerpts from this routine. The first example is modified to illustrate position-dependent references. The second example is the position-independent version.

Position-Dependent Code

```
PWRUP::
        CLR     -(SP)           ;ASSUME SUCCESS
        CALL    .X.PAA          ;PUSH (SAVE)
                                ;ARGUMENT ADDRESSES
                                ;ONTO STACK
        .WORD   1.,$PSW         ;CLEAR PSW, AND
                                ;SET R1=R2SP
        MOV     $OTSV,R4        ;GET OTS IMPURE
                                ;AREA POINTER
        MOV     (SP)+,R2        ;GET AST ENTRY
                                ;POINT ADDRESS
        BNE     10$             ;IF NONE SPECIFIED,
                                ;SPECIFY NO POWER
        CLR     -(SP)           ;RECOVERY AST SERVICE
        BR      20$             ;
10$:                            ;
        MOV     R2,F.PF(R4)     ;SET AST ENTRY POINT
        MOV     #BA,-(SP)       ;PUSH AST SERVICE
                                ;ADDRESS
20$:                            ;
        CALL    .X.EXT          ;ISSUE DIRECTIVE, EXIT.
        .BYTE   109.,2.         ;
        .
        .
        .
BA:     MOV     R0,-(SP)        ;PUSH (SAVE) R0
        MOV     R1,-(SP)        ;PUSH (SAVE) R1
        MOV     R2,-(SP)        ;PUSH (SAVE) R2
```

Position-Independent Code

```
PWRUP::
        CLR     -(SP)           ;ASSUME SUCCESS
        CALL    .X.PAA          ;PUSH ARGUMENT
                                ;ADDRESSES ONTO
                                ;STACK
        .WORD   1.,$PSW         ;CLEAR PSW, AND
                                ;SET R1=R2-SP.
        MOV     @#$OTSV,R4      ;GET OTS IMPURE
                                ;AREA POINTER
        MOV     (SP)+,R2        ;GET AST ENTRY
                                ;POINT ADDRESS
        BNE     10$             ;IF NONE SPECIFIED,
                                ;SPECIFY NO POWER
        CLR     -(SP)           ;RECOVERY AST SERVICE
        BR      20$
10$:                            ;
        MOV     R2,F.PF(R4)     ;SET AST ENTRY POINT
        MOV     PC,-(SP)        ;PUSH CURRENT LOCATION
        ADD     #BA-.,(SP)      ;COMPUTE ACTUAL LOCATION
                                ;OF AST
20$:
        CALL    .X.EXT          ;ISSUE DIRECTIVE, EXIT.
        BYTE    109.,2.
                                ;
                                ;ACTUAL AST SERVICE ROUTINE:
                                ;
                                ; 1)    SAVE REGISTERS
                                ; 2)    EFFECT A CALL TO SPECIFIED
                                ;       SUBROUTINE
                                ; 3)    RESTORE REGISTERS
                                ; 4)    ISSUE AST EXIT DIRECTIVE
                                ;
BA:     MOV     R0,-(SP)        ;PUSH  (SAVE) R0
        MOV     R1,-(SP)        ;PUSH  (SAVE) R1
        MOV     R2,-(SP)        ;PUSH  (SAVE) R2
```

The position-dependent version of the subroutine contains a relative reference to an absolute symbol ($OTSV) and a literal reference to a relocatable symbol (BA). Both references are bound by the task builder to fixed memory locations. Therefore, the routine does not execute properly as part of a resident library, if its location in virtual memory is not the same as the location specified at link time.

In the position-independent version, the reference to $OTSV has been changed to an absolute reference. In addition, the necessary code has been added to compute the virtual location of BA based upon the value of the PC. In this case, the value is obtained by adding the value of the PC to the fixed displacement between the current location and the specified symbol. Thus, execution of the modified routine is not affected by its location in the virtual address space of the image.

11.3 STACKS

The stack is part of the basic design architecture of the KDJ11-B. It is an area of memory set aside by the programmer or the operating system for temporary storage and linkage. It is handled on a Last In, First Out (LIFO) basis, where items are retrieved in reverse of the order in which they were stored. A stack starts at the highest location reserved for it and expands linearly downward to lower addresses as items are added.

It is not necessary to keep track of the actual locations into which data is being stacked. This is done automatically through an SP. To keep track of the last item added to the stack, a general register is used to store the memory address of the last item in the stack. Any register except R7 (the PC) may be used as an SP under program control; however, instructions associated with subroutine linkage and interrupt service automatically use R6 as a hardware stack pointer. For this reason, R6 is frequently referred to as the system SP. Stacks may be maintained in either full-word or byte units. This is true for a stack pointed to by any register except R6, which must be organized in full-word units. Byte stacks (Figure 11-1) require instructions capable of operating on bytes rather than full words.

11.3.1 Pushing onto a Stack

Items are added to a stack using the autodecrement addressing mode. Adding items to the stack is called 'pushing,' and is accomplished by the following instructions.

```
MOV     Source,-(SP)     ;MOV CONTENTS OF SOURCE WORD
                         ;ONTO THE STACK
                             OR
MOVB    Source,-(SP)     ;MOVB SOURCE BYTE ONTO
                         ;THE STACK
```

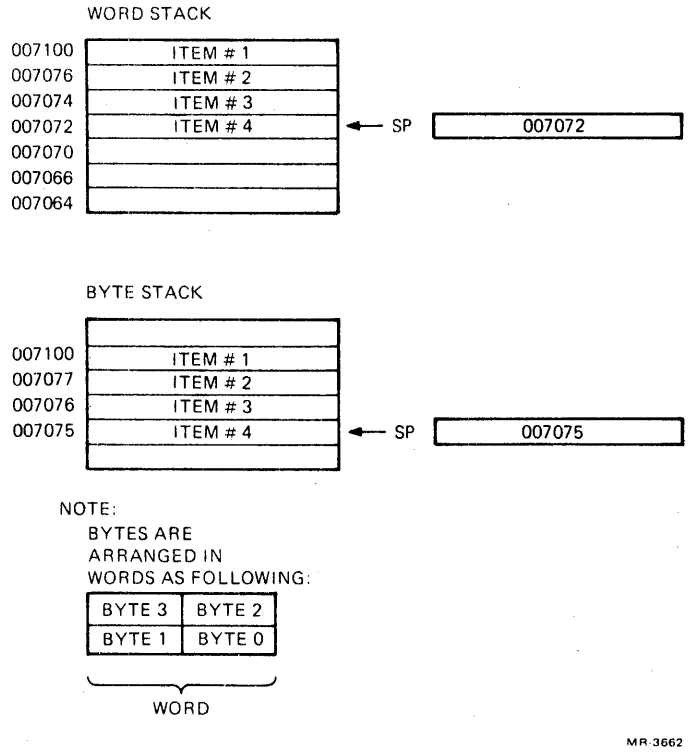


Figure 11-1 Word and Byte Stacks

11.3.2 Popping from a Stack

Removing data from the stack is called 'popping.' This operation is accomplished using the autoincrement mode.

```
MOV     (SP)+,Destination     ;MOV DESTINATION WORD
                              ;OFF THE STACK
                                   OR
MOVB    (SP)+,Destination     ;MOVB DESTINATION BYTE
                              ;OFF THE STACK
```

After an item has been popped, its stack location is considered free and available for other use. The SP points to the last-used location, implying that the next lower location is free. Thus, a stack may represent a pool of shareable temporary storage locations. Refer to Figure 11-2.

11.3.3 Deleting Items from a Stack

The following techniques may be used to delete items from a stack.

To delete one item from a byte stack:

```
INC SP or TSTB(SP)+
```

To delete two items from a word stack:

```
ADD#2,SP or TST(SP)+
```

To delete 50 items from a word stack:

```
ADD#100.,SP
```

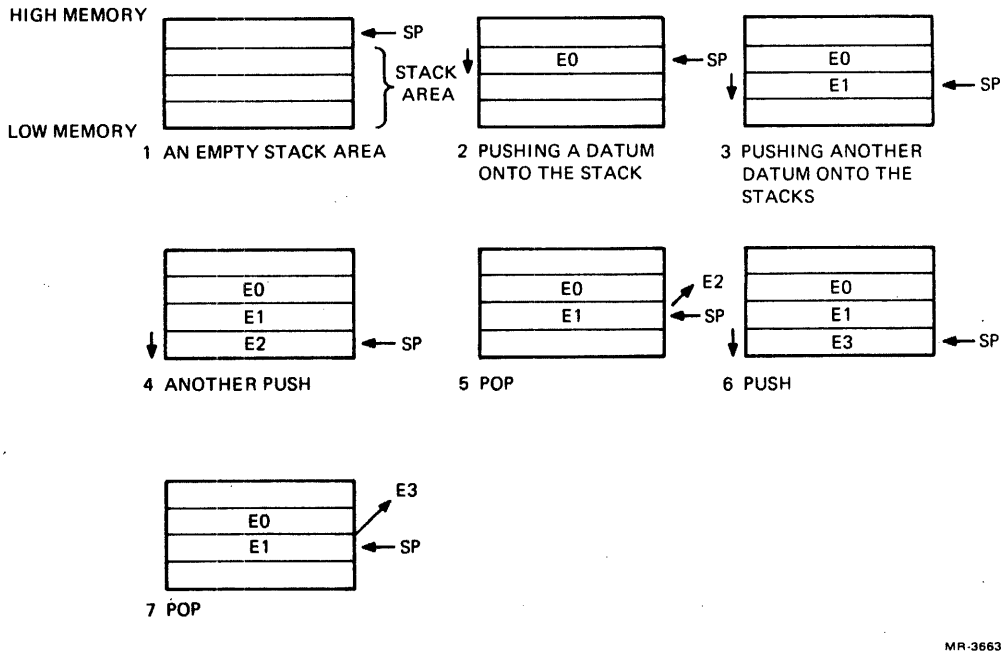


Figure 11-2 Push and Pop Operations

11.3.4 Stack Uses

A stack is used in the following ways.

1. Often, one of the general-purpose registers must be used in a subroutine or interrupt service routine and then be returned to its original value. The stack can be used to store the contents of the registers involved.

2. The stack is used in storing linkage information between a subroutine and its calling program. The JSR instruction, used in calling a subroutine, requires the specification of a linkage register along with the entry address of the subroutine. The content of this linkage register is stored on the stack, so as not to be lost, and the return address is moved from the PC to the linkage register. This provides a pointer back to the calling program so that successive arguments may be transmitted easily to the subroutine.

3. If no arguments need be passed by stacking them after the JSR instruction, the PC may be used as the linkage register. In this case, the result of the JSR is to move the return address in the calling program from the PC onto the stack and replace it with the entry address of the called subroutine.

4. In many cases, the operations performed by the subroutine can be applied directly to the data located on or pointed to by a stack without the need to move the data into the subroutine area.

   Example:

```
                        ;CALLING PROGRAM
MOV   SP,R1             ;R1 IS USED AS THE STACK
JSR   PC,SUBR           ;POINTER HERE.

                        ;SUBROUTINE
ADD   (R1)+,(R1)        ;ADD ITEM #1 TO #2, PLACE
                        ;RESULT IN ITEM #2,
                        ;R1 POINTS TO
                        ;ITEM #2 NOW
```

   Since arguments may be obtained from the stack by using some form of register-indexed addressing, it is sometimes useful to save a temporary copy of R6 in some other register that has been saved at the beginning of a subroutine. If R6 is saved in R5 at the beginning of the subroutine, R5 may be used to index the arguments. During this time, R6 is free to be incremented and decremented while being used as the SP. If R6 is used directly as the base for indexing and is not 'copied,' it may be difficult to keep track of its position in the argument list, since the base of the stack changes with every autoincrement/decrement.

   However, if the contents of R6 (SP) are saved in R5 before any arguments are pushed onto the stack, the position relative to R5 remains constant.

   Return from a subroutine also involves the stack, as the return instruction, RTS, must retrieve information stored there by the JSR.

   When a subroutine returns, it is necessary to 'clean up' the stack by eliminating or skipping over the subroutine arguments. One way this can be done is to insist that the subroutine keep the number of arguments as its first stack item. Returns from subroutines then involve calculating the amount by which to reset the SP, resetting the SP, and then storing the original contents of the register that was used as the SP copy.

5. Stack storage is used in trap and interrupt linkage. The PC and the PSW of the executing program are pushed on the stack.

6. When the system stack is being used, nesting of subroutines, interrupts, and traps to any level can occur until the stack overflows its legal limits.

7. The stack method is also available for temporary storage of any kind of data. It may be used as a LIFO list for storing inputs, intermediate results, etc.

11.3.5 Stack Use Examples

As an example of stack use, consider this situation. A subroutine (SUBR) wants to use registers 1 and 2, but these registers must be returned to the calling program with their contents unchanged. The subroutine could be written as follows.

Not Using the Stack

Address	Octal Code	Assembler Syntax	Comments
076322	010167 SUBR:	MOV R1,TEMP1	;SAVE R1
076324	000074	*	
076326	010267	MOV R2,TEMP2	;SAVE R2
076330	000072	*	
.	.	.	
.	.	.	
.	.	.	
076410	016701	MOV TEMP1,R1	;RESTORE R1
076412	000006	*	
076414	016702	MOV TEMP2,R2	;RESTORE R2
076416	000004	*	
076420	000207	RTS PC	
076422	000000	TEMP1:0	
076424	000000	TEMP2:0	

Using the Stack

Note that in this case, R3 is being used as an SP and has been previously set to point to the end of an unused block of memory.

Address	Octal Code	Assembler Syntax	Comments
010020	010143 SUBR:	MOV R1,–(R3)	;PUSH R1
010022	010243	MOV R2,–(R3)	;PUSH R2
.	.	.	
.	.	.	
.	.	.	
.	.	.	
010130	012302	MOV (R3)+,R2	;POP R2
010132	012301	MOV (R3)+,R1	;POP R1
010134	000207	RTS PC	

*Index constants

The second routine uses four fewer words of instruction code and two words of temporary stack storage. Another routine may use the same stack space at some later point. Thus, the ability to share temporary storage in the form of a stack is a way to save on memory usage.

As another example of stack use, consider the task of managing an input buffer from a terminal. As characters come in, the user may wish to delete characters from the line. This is accomplished very easily by maintaining a byte stack containing the input characters. Whenever a backspace is received, a character is popped off the stack and eliminated from consideration. In this example, popping characters to be eliminated can be done by using either the MOVB (move byte) or INC (increment) instructions.

Note that in this case the INC instruction is preferable to MOVB, since it accomplishes the task of eliminating the unwanted character from the stack by readjusting the SP without the need for a destination location. Note also, that the SP used in this example cannot be the system SP (R6) because R6 may point only to word (even) locations. Refer to Figure 11-3.

11.3.6 Subroutine Linkage

The contents of the linkage register are saved on the system stack when a JSR is executed. The effect is the same as executing a MOV reg,–(R6). Following the JSR instruction, the same register is loaded with the memory address (the contents of the current PC) and a jump is made to the entry location specified. Figure 11-4 shows the conditions before and after the subroutine instruction JSR R5, 1064 is executed.

Because hardware already uses general purpose register 6 to point to a stack for saving and restoring PC and PSW information, it is convenient to use that stack to save and restore intermediate results and to transmit arguments to and from subroutines. Using R6 this way permits nesting subroutines and interrupt service routines.

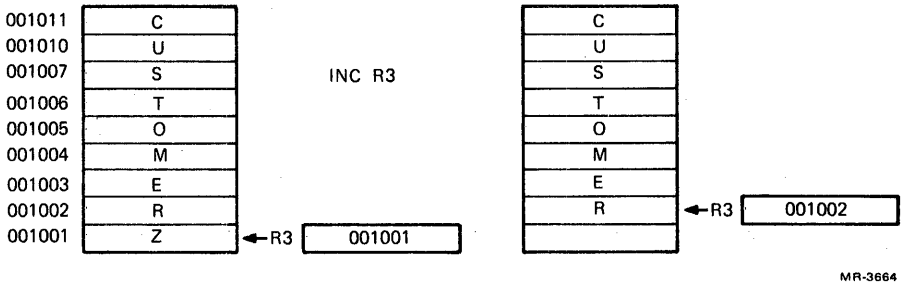


Figure 11-3 Byte Stack Used as a Character Buffer

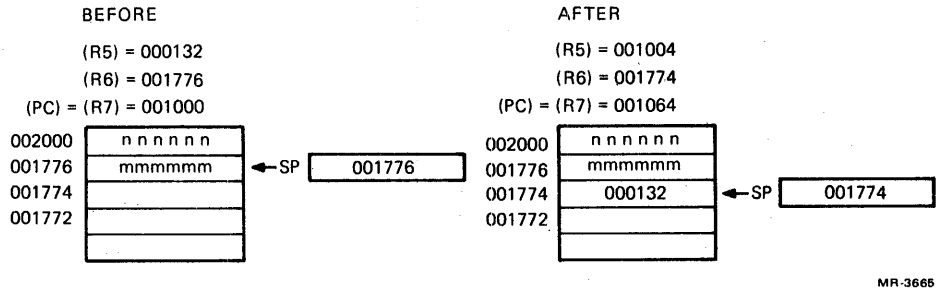


Figure 11-4 JSR Stack Condition Example

11.3.6.1 Return from a Subroutine – An RTS instruction provides for a return from the subroutine to the calling program. The RTS instruction must specify the same register the JSR instruction used in the subroutine call. When the RTS is executed, the register specified is moved to the PC, and the top of the stack is placed in the register specified. Thus, an RTS PC has the effect of returning to the address specified on the top of the stack.

11.3.6.2 Subroutine Advantages – The JSR instruction provides several advantages to the subroutine calling procedure.

1. Arguments can be passed quickly between the calling program and the subroutine.

2. If there are no arguments, or the arguments are in a general register or on the stack, the JSR PC,DST mode can be used so that none of the general purpose registers need to be used for linkage.

3. Many JSRs can be executed without the need to provide any saving procedure for the linkage information, since all linkage information is automatically pushed onto the stack in sequential order. Returns can be made by automatically popping this information from the stack in the order opposite to the JSRs.

This linkage address bookkeeping is called automatic nesting of subroutine calls. This feature enables construction of fast, efficient linkages in a simple, flexible manner. It also permits a routine to call itself.

11.3.7 Interrupts

An interrupt is similar to a subroutine call, except that it is initiated by the hardware rather than by the software. An interrupt can occur after the execution of an instruction.

Interrupt-driven techniques are used to reduce CPU waiting time. In direct program data transfer, the CPU loops to check the state of the done/ready flag (bit 7) in the peripheral interface. Using interrupts, the CPU can handle other functions until the peripheral initiates service by setting the done bit in its CSR. The CPU completes the instruction being executed, then acknowledges the interrupt, and vectors to an interrupt service routine. The service routine transfers the data and may perform calculations with it. After the interrupt service routine is complete, the computer resumes the program that was interrupted by the high-priority request.

11.3.7.1 Interrupt Service Routines – With interrupt service routines, linkage information is passed so that a return to the main program can be made. More information is necessary for an interrupt sequence than for a subroutine call because of the random nature of interrupts. The complete machine state of the program immediately prior to the occurrence of the interrupt must be preserved in order to return to the program without any noticeable effects. This information is stored in the PSW. Upon interrupt, the contents of the PC (address of next instruction) and the PSW are automatically pushed onto the R6 system stack. The effect is the same as executing:

```
MOV PS,–(SP)      ;PUSH PSW
MOV PC,–(SP)      ;PUSH PC
```

The new contents of the PC and PSW are loaded from two preassigned consecutive memory locations called vector addresses. The first word contains the interrupt service routine entry address (the address of the service routine program sequence). The second word contains the new PSW that will determine the machine status, including the operational mode and register set to be used by the interrupt service routine. The contents of the vector address is set under program control.

After the interrupt service routine is complete, an RTI is performed. The top two words of the stack are automatically popped and placed in the PC and PSW, respectively, thus resuming the interrupted program. Interrupt service programming is intimately involved with the concept of CPU and device priority levels.

11.3.7.2 Nesting – Interrupts can be nested in much the same manner that subroutines are nested. It is possible to nest any arbitrary mixture of subroutines and interrupts without any confusion. When the respective RTI and RTS instructions are used, the proper returns are automatic. Refer to Figure 11-5.

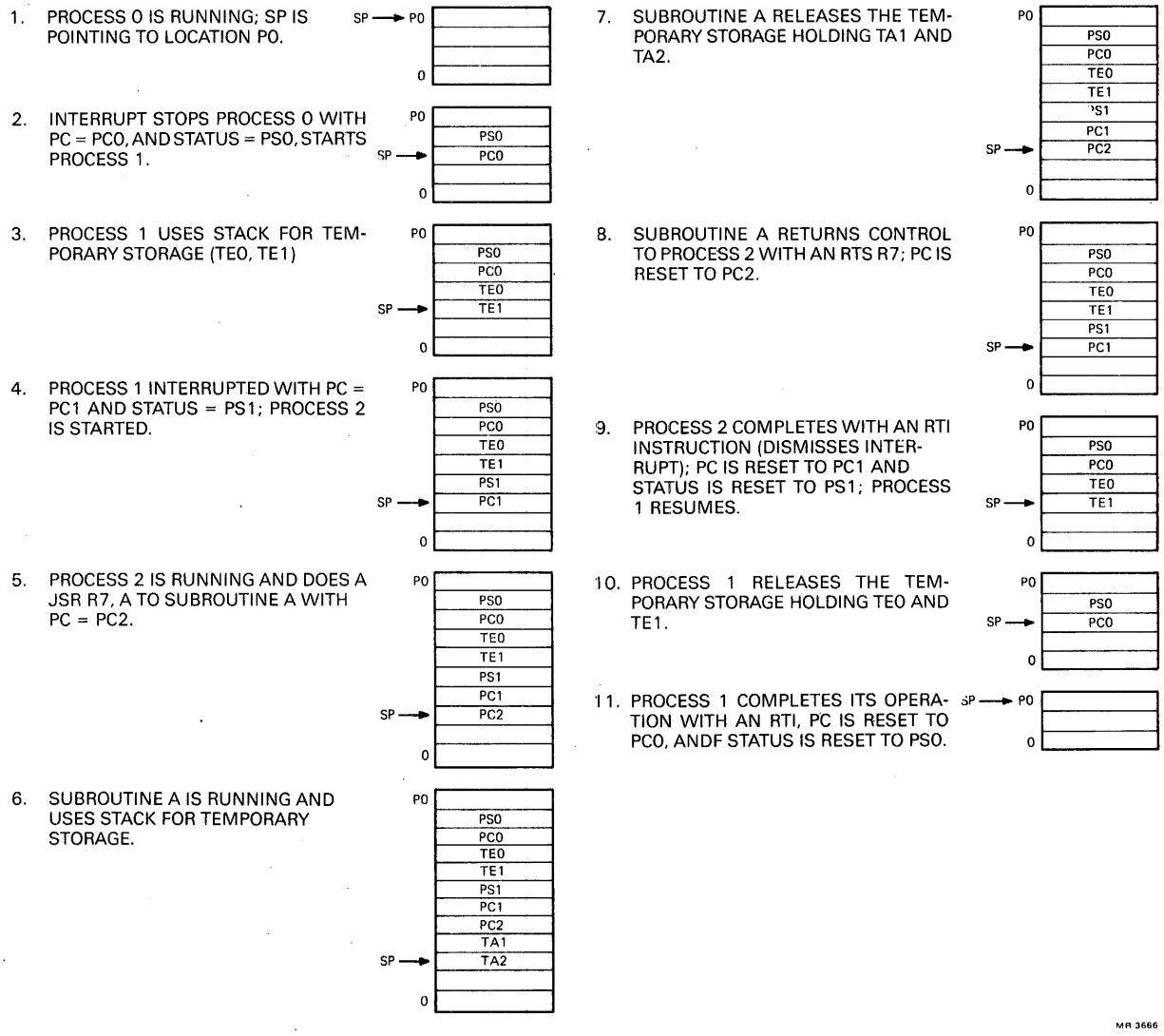


Figure 11-5 Nested Interrupt Service Routines and Subroutines

11.3.8 Reentrancy

Other advantages of the KDJ11-B stack organization occur in programming systems that handle several tasks. Multitask program environments range from simple single-user applications that manage a mixture of I/O interrupt service and background data processing (as in RT-11), to complex multiprogramming systems that manage an intricate mixture of executive and multiuser programming situations (as in RSX-11). In all these situations, using the stack as a programming technique provides flexibility and time/memory economy by allowing many tasks to use a single copy of the same routine with a simple straightforward way of keeping track of complex program linkages.

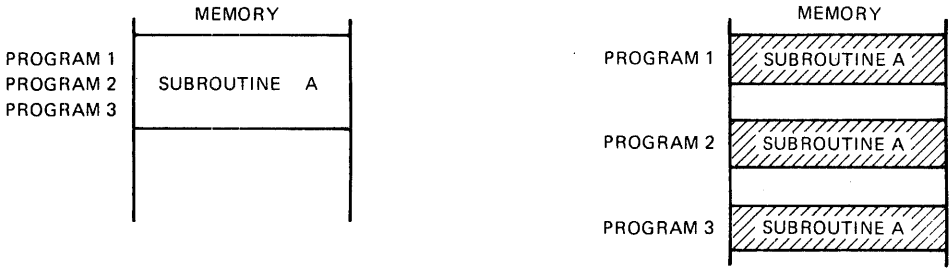
The ability to share a single copy of a program among users or among tasks is called reentrancy. Reentrant program routines differ from ordinary subroutines in that it is not necessary for reentrant routines to finish processing a given task before they can be used by another task. At any time, tasks can exist in various stages of completion in the same routine. Thus, the situation shown in Figure 11-6 may occur.

11.3.8.1 Reentrant Code – Reentrant routines must be written in pure code (that is, any code that consists exclusively of instructions and constants). The value of using pure code whenever possible is that the resulting code has the following characteristics.

- It is generally considered easier to debug than standard code.
- It can be kept in read-only memory (is read-only protected).

Using reentrant code, control of a routine can be shared as follows. Refer to Figure 11-7.

1. Task A requests processing by reentrant routine Q.
2. Task A temporarily gives up control of reentrant routine Q before it completes processing.
3. Task B starts processing the same copy of reentrant routine Q.
4. Task B completes processing by reentrant routine Q.
5. Task A regains use of reentrant routine Q and resumes where it stopped.



KDJ11-A APPROACH

PROGRAMS 1, 2, AND 3 CAN SHARE SUBROUTINE A.

CONVENTIONAL APPROACH

A SEPARATE COPY OF SUBROUTINE A MUST BE PROVIDED FOR EACH PROGRAM.

MR 3667

Figure 11-6 Reentrant Routines

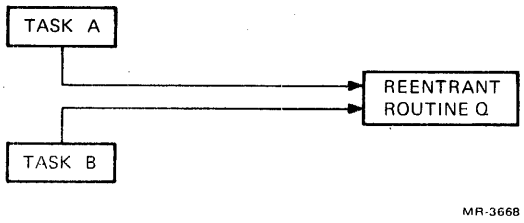


Figure 11-7 Sharing Control of a Routine

11.3.8.2 Writing Reentrant Code – In an operating system environment, when one task is executing and is interrupted to allow another task to run, a context switch occurs in which the PSW and current contents of the general purpose registers are saved and replaced by the appropriate values for the task being entered. Therefore, reentrant code must use the general purpose registers and the stack for any counters, pointers, or data to be modified or manipulated in the routine.

The context switch occurs whenever a new task is allowed to execute. It causes all of the general purpose registers, the PSW, and often, other task-related information to be saved in an impure area. It then reloads these registers and locations with the appropriate data for the task being entered. Notice that one consequence of this is that a new SP value is loaded into R6, thereby causing a new area to be used as the stack when the second task is entered.

The following should be observed when writing reentrant code.

1. All data should be in or pointed to by one of the general purpose registers.

2. A stack can be used for temporary storage of data or pointers to impure areas within the task space. The pointer to such a stack would be stored in a general purpose register.

3. Parameter addresses should be used by indexing and indirect reference rather than by putting them into instructions within the code.

4. When temporary storage is accessed within the program, it should be by indexed addresses, which can be set by the calling task in order to handle any possible recursion.

11.3.9 Coroutines

In some programming situations, several program segments or routines are highly interactive. Control is passed back and forth between the routines, each going through a period of suspension before being resumed. Since the routines maintain a symmetric relationship with each other, they are called coroutines.

Coroutines are two program sections, either one subordinate to the call of the other. The nature of the call is, 'I have processed all I can for now, so you can execute until you are ready to stop, then I will continue.' The coroutine call and return are identical, each being a jump to subroutine instruction with the destination address on top of the stack and the PC serving as the linkage register, as follows.

```
JSR PC,@(R6)+
```

11.3.9.1 Coroutine Calls – The coding of coroutine calls is made simple by the stack feature. Initially, the entry address of the coroutine is placed on the stack, and from that point the JSR PC,@(R6)+ instruction is used for both the call and the return statements. This JSR instruction results in an exchange of the contents of the PC and the top element of the stack, permitting the two routines to swap control and resume operation where each was terminated by the previous swap. An example is shown in Figure 11-8. Notice that the coroutine linkage cleans up the stack with each control transfer.

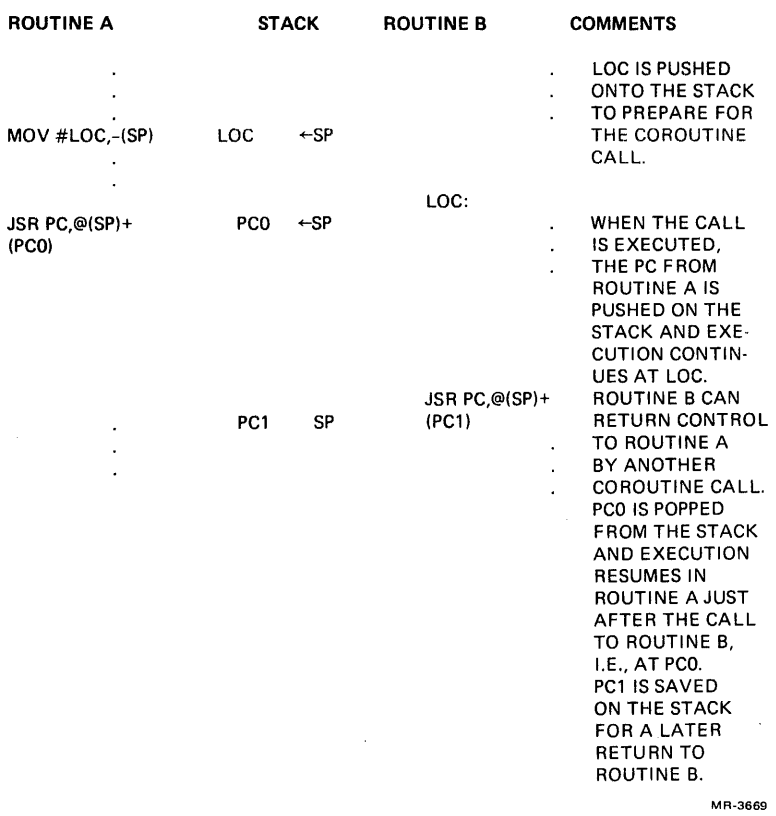


Figure 11-8 Coroutine Example

11.3.9.2 Coroutines Versus Subroutines – Coroutines can be compared to subroutines in the following ways.

- A subroutine is considered subordinate to the main or calling routine, but a coroutine is considered to be on the same level, as each coroutine calls the other when it has completed current processing.

- When called, a subroutine executes to the end of its code. When called again, the same code will execute before returning. A coroutine executes from the point after the last call of the other coroutine. Therefore, the same code will not be executed each time the coroutine is called. An example is shown in Figure 11-9.

- The call and return instructions for coroutines are the same.

  JSR PC,@(SP)+

  This one instruction also cleans up the stack with each call. The last coroutine call leaves an address on the stack that must be popped if no further calls are to be made. Refer to Paragraph 11.3.6.1 for information on the return from subroutine instruction.

- Each coroutine call returns to the coroutine code at the point after the last exit with no need for a specific entry point label, as would be required with subroutines.

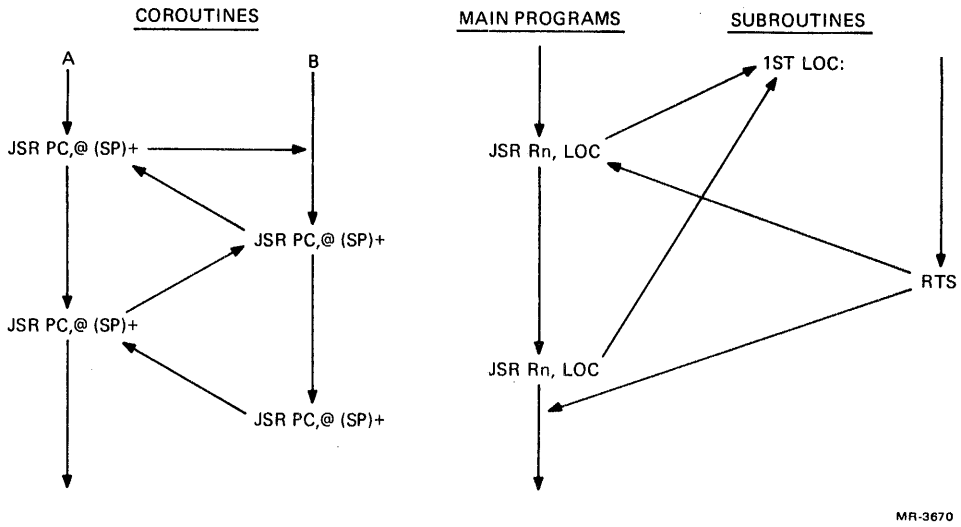


Figure 11-9 Coroutines Versus Subroutines

11.3.9.3 Using Coroutines – Coroutines should be used in the following situations.

- Whenever two tasks must be coordinated in their execution without obscuring the basic structure of the program. For example, in decoding a line of assembly language code, the results at any one position might indicate the next process to be entered. A detected label must be processed. If no label is present, the operator must be located, etc.

- To add clarity to the process being performed, to ease in the debugging phase, etc.

An assembler must perform a lexicographic scan of each assembly language statement during pass 1 of the assembly process. The various steps in such a scan should be separated from the main program flow to add to program clarity and to aid in debugging by isolating details. Subroutines are not satisfactory in this case, as too much information has to be passed to the subroutine each time it is called. Coroutines could be effectively used, with one routine performing as the assembly pass 1 routine and the other extracting one item at a time from the current input line. Figure 11-10 illustrates this example.

Coroutines can be utilized in I/O processing. Figure 11-10 shows coroutines used in double-buffered I/O using IOX. The flow of events may be described as follows.

```
Write 01
Read I1          concurrently,
Process I2
```

then

```
Write 02
Read I2          concurrently,
Process I1
```

Figure 11-11 illustrates a coroutine swapping interaction.

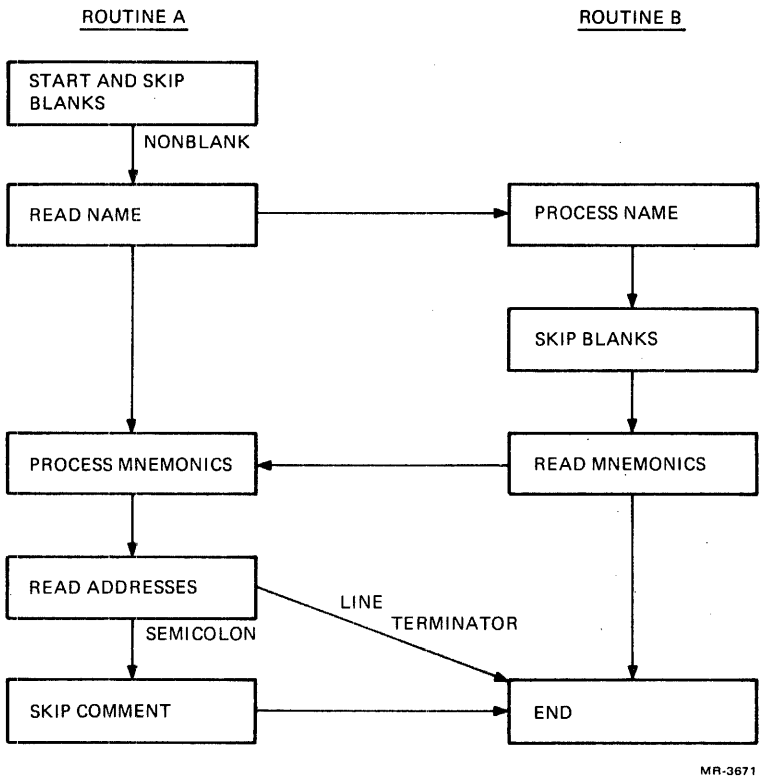


Figure 11-10 Coroutine Path

ROUTINE #1 IS OPERATING, IT THEN EXECUTES:

```
MOV #PC2,-(R6)
JSR PC,@(R6)+
```

WITH THE FOLLOWING RESULTS:

1. PC2 IS POPPED FROM THE STACK AND THE SP AUTOINCREMENTED.
2. SP IS AUTODECREMENTED AND THE OLD PC (I.E., PC1) IS PUSHED.
3. CONTROL IS TRANSFERRED TO THE LOCATION PC2 (I.E., ROUTINE #2).

ROUTINE #2 IS OPERATING, IT THEN EXECUTES:

```
JSR PC,@(R6)+
```

WITH THE RESULT THAT PC2 IS EXCHANGED FOR PC1 ON THE STACK AND CONTROL IS TRANSFERRED BACK TO ROUTINE #1.

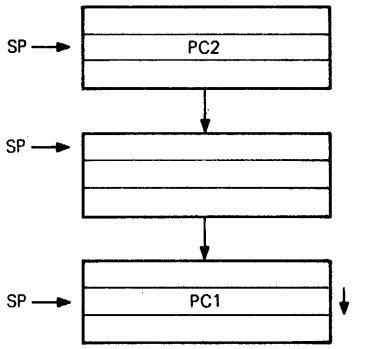


Figure 11-11 Coroutine Interaction

When routine 1 is operating, it executes

```
MOV #PC2,–(R6)
JSR PC,@(R6)+
```

with the following results.

1. PC2 is popped from the stack and the SP is autoincremented.
2. SP is autodecremented and the old PC (i.e., PC1) is pushed.
3. Control is transferred to the location PC2 (i.e., routine 2).

When routine 2 is operating, it executes

```
JSR PC,@(R6)+
```

with the result that PC2 is exchanged for PC1 on the stack and control is transferred back to routine 1.

11.3.10 Recursion

An interesting aspect of a stack facility, other than its providing for automatic handling of nested subroutines and interrupts, is that a program may call on itself as a subroutine – just as it can call on any other routine. Each new call causes the return linkage to be placed on the stack, which (as it is a LIFO queue) sets up a natural unraveling to each routine just after the point of departure. Typical flow for a recursive routine resembles that shown in Figure 11-12.

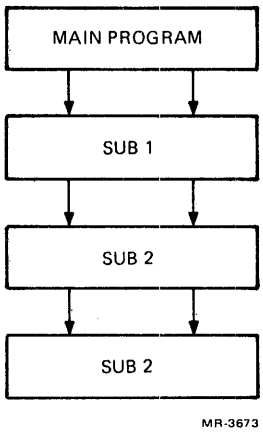


Figure 11-12 Recursive Routine Flow

The main program calls function 1, SUB 1, which calls function 2, SUB 2, which recurses once before returning.

Example:

```
DNCF:   ,
        ,
        ,
        BEQ 1$              ;TO EXIT RECURSIVE LOOP
        JSR R5,DNCF         ;RECURSE
1$      ,
        ,
        ,
        RTS R5              ;RETURN TO 1$ FOR
                            ;EACH CALL, THEN TO
                            ;MAIN PROGRAM
```

The routine DNCF calls itself until the variable tested becomes equal to 0. Then it exits to 1$, where the RTS instruction is executed, returning to the 1$ once for each recursive call and a final time to return to the main program.

In general, recursion techniques lead to slower programs than the corresponding interactive techniques, but recursion does produce shorter programs, and thus saves memory space. Both the brevity and clarity produced by recursion are important in assembly language programs.

Uses of Recursion – Recursion can be used in any routine in which the same process is required several times. For example, a function to be integrated may contain another function to be integrated, as in solving for XM, where

$$SM = 1 + F(X)$$

and

$$F(X) = G(X).$$

Another use for a recursive function could be in calculating a factorial function, because

$$FACT(N) = FACT(N - 1) * N.$$

Recursion should terminate when $N = 1$.

The macroprocessor within MACRO-11 is itself recursive, since it can process nested macrodefinitions and calls. For example, within a macrodefinition, other macros can be called. When a macro call is encountered within definition, the processor must work recursively (i.e., it must process one macro before it is finished with another and then continue with the previous one). The stack is used for a separate storage area for the variables associated with each call to the procedure.

As long as nested definitions of macros are available, it is possible for a macro to call itself. However, unless conditionals are used to terminate this expansion, an infinite loop may be generated.

11.3.11 Processor Traps

Certain errors and programming conditions cause the KDJ11-B processor to enter the service state and trap to a fixed location. A trap is an interrupt generated by hardware. Pending conditions are arbitrated according to a priority. The following list describes the priority from highest to lowest.

Condition	Description
Memory management violation* (MMUERR)	A memory management violation causes an abort and traps to location 250_8.
Timeout error* (BUSERR)	No response from a bus device during a bus transaction causes an abort and traps to location 4_8.
Parity error* (PARERR)	A parity error signal received by the processor during a bus transaction causes an abort and traps to location 114_8.
Trace (T) bit*	If PSW bit 4 is set at the end of instruction execution, the processor traps to location 14_8.
Stack overflow* (STKOVF)	If the KSP was pushed below 400_8 during instruction execution, the processor traps to location 4_8 at the end of the instruction.
Power fail* (PFAIL)	If the power OK bus signal (BPOKH) was negated during instruction execution, the processor traps to location 24_8 at the end of the instruction.
Interrupt level 7 (BIRQ7) Interrupt level 6 (BIRQ6) Interrupt level 5 (BIRQ5) Interrupt level 4 (BIRQ4)	If device interrupt requests are asserted and PSW <7:5> are properly set, the processor at the end of the present instruction execution initiates an interrupt vector sequence on the bus. These inputs are maskable by PSW <7:5>.
Halt line	If the BHALT L bus signal is asserted during the service state, the processor enters ODT mode.

PSW <7:5>	Levels Inhibited
7	All
6	6, 5, 4
5	5, 4
4	4
0–3	None

* Nonmaskable software cannot inhibit the condition. MMUERR, BUSERR, PARERR are mutually exclusive when the processor is executing a program.

11.3.11.1 Trap Instructions – Trap instructions provide for calls to emulators, I/O monitors, debugging packages, and user-defined interpreters. When a trap occurs, the contents of the current PC and PSW are pushed onto the processor stack and are replaced by the contents of a 2-word trap vector containing a new PC and new PSW. The return sequence from a trap involves executing an RTI or RTT instruction, which restores the old PC and old PSW by popping them from the stack. Trap vectors are located at permanently assigned fixed addresses.

The EMT (trap emulator) and TRAP instructions do not use the low-order byte of the word in their machine language representation. This allows user information to be transferred in the low-order byte. The new value of the PC, loaded from the vector address of the TRAP or EMT instructions, is typically the starting address of a routine to access and interpret this information. This routine is called a trap handler.

A trap handler must accomplish several tasks. It must save and restore all necessary general purpose registers, interpret the low byte of the trap instruction and call the indicated routine, serve as an interface between the calling program and this routine by handling any data that needs to be passed between them, and finally, cause the return to the main routine.

A trap handler can be useful as a patching technique. Jumping out to a patch area is often difficult because a 2-word jump must be performed. However, the 1-word TRAP instruction may be used to dispatch to patch areas. A sufficient number of slots for patching should first be reserved in the dispatch table of the trap handler. The jump can then be accomplished by placing the address of the patch area into the table and inserting the proper TRAP instruction where the patch is to be made.

11.3.11.2 Use of Macro Calls – The trap handler can be used in a program to dispatch execution to any one of several routines. Macros may be defined to cause the proper expansion of a call to one of these routines, as in the example below.

```
.MACRO SUB2 ARG
MOV ARG, R0
TRAP +1
.ENDM
```

When expanded, this macro sets up the one argument required by the routine in R0, and then causes the trap instruction with the number 1 in the lower byte. The trap handler should be written so that it recognizes a 1 as a call to SUB2. Notice that ARG here is being transmitted to SUB2 from the calling program. It may be data required by the routine or it may be a pointer to a longer list of arguments.

In an operating system environment like RT-11, the EMT instruction is used to call the system or monitor routines from a user program. The monitor of an operating system necessarily contains coding for many functions, such as I/O, file manipulation, etc. This coding is made accessible to the program through a series of macro calls that expand into EMT instructions with low bytes, indicating the routine or group of routines to which the desired routine belongs. Often a general purpose register is designated to be used to pass an identification code to further indicate to the trap handler which routine is desired. For example, the macro expansion for a resume execution command in RT-11 is as follows.

```
.MACRO .RSUM
CM3, 2.
.ENDM
```

CM3 is defined:

```
.MACRO CM3 CHAN, CODE
MOV #CODE *400,R0
.IIF NB          CHAN,BISB CHAN,R0
EMT 374
.ENDM
```

Note that the EMT low byte is 374. This is interpreted by the EMT handler to indicate a group of routines. Then the contents of R0 (high byte) is tested by the handler to identify exactly which routine within the group is being requested – in this case routine number 2. (The CM3 call of the .RSUM is set up to pass the identification code.)

11.3.12 Conversion Routines

Almost all assembly language programs require the translation of data or results from one form to another. Code that performs such a transformation is called a conversion routine. Several commonly used conversion routines follow.

Almost all assembly language programs involve some type of conversion routine. Octal-to-ASCII, octal-to-decimal, and decimal-to-ASCII are a few of the most widely used.

Arithmetic multiply and divide routines are fundamental to many conversion routines. Division is typically approached in one of two ways.

1. The division can be accomplished through a combination of rotates and subtractions.

   Example:

   Assume the following code and register data. To make the example easier, also assume a 3-bit word.

```
DIV:    MOV #3,-(SP)        ;SET UP DIGIT COUNTER
        CLR -(SP)           ;CLEAR RESULT
1$      ASL (SP)
        ASL R1
        ROL R0
        CMP R0,R3
        BLT 2$
        SUB R3,R0           ;R0 CONTAINS REMAINDER
        INC (SP)            ;INCREMENT RESULT
2$      DEC 2 (SP)          ;DECREMENT COUNTER
        BNE $1
```

   Therefore, to divide 7 by 2:

```
R0 = 000        remainder
R1 = 111        7 (multiplicand)
R3 = 010        2 (multiplier)
C bit = 0

Stack
011             counter
000             quotient
```

   Following through the coding, the quotient, remainder, and dividend all shift left, manipulating the most significant digit first, etc.

   At the conclusion of the routine:

```
R0 = 001        remainder
R1 = 000
R3 = 010

Stack
000             counter
011             quotient
```

2. The second method of division works by repeated subtraction of the powers of the divisor, keeping a count of the number of subtractions at each level.

Example:

To divide 221_{10} by 10, first try to subtract powers of 10 until a nonnegative value is obtained, counting the number of subtractions of each power.

```
  221
−1000
```

Negative, so go to the next lower power, and count for $10^3 = 0$.

```
  221
 −100

          121 count for 10^2= 1
         −100

           21 count        = 2
         −100
```

Negative, so reduce power, and count for $10^2 = 2$.

```
   21
  −10

          11 count for 10^1  = 1

          11
         −10

           1 count         = 2
         −10
```

Negative, so count for $10^1 = 2$.

No lower power, so remainder is 1.

Answer = 022, remainder 1.

Multiplication is also approached in one of two ways.

1. Multiplication can be done with a combination of rotates and additions.

   Example:

   Assume the following code and a 3-bit word.

```
            CLR R0              ;HIGH HALF OF ANSWER
            MOV #3,CNT          ;SET UP COUNTER
            MOV R1,MULT;        ;MULTIPLICAND

            MORE:               ROR R2
                                BCC NOW
                                ADD MULT,R0
                                ;IF INDICATED,
ADD
                                ;MULTIPLICAND
            NOW;                ROR R0
                                R04 R1
                                DEC CNT
                                BNE MORE
            MULT:               0
            CNT:                0
```

   The following conditions exist for 6 × 3.

```
R0 = 000        high-order half of result
R1 = 110        multiplicand
R3 = 011        multiplier
```

   After the routine is executed:

```
R0 = 010        high-order half of result
R1 = 010        low-order half of result
R2 = 100
CNT = 0
MULT = 110
```

2. The second method of multiplication is repetitive addition.

Example:

Multiplication of R0 by 50_8(101000).

```
MUL50:      MOV R0,–(SP)
            ASL R0
            ASL R0
            ADD (SP)+,R0
            ASL R0
            ASL R0
            ASL R0
            RETURN
```

If R0 contains 7:

R0 = 111

After execution:

R0 = 100011000
(7_8 * 50_8 = 430_8)

ASCII Conversions – The conversion of ASCII characters to the internal representation of a number, as well as the conversion of an internal number to ASCII in I/O operations, presents a challenge. The following routine takes the 16-bit word in R1 and stores the corresponding 6 ASCII characters in the buffer addressed by R2.

```
OUT:    MOV     #5,R0           ;LOOP COUNT
LOOP:   MOV     R1,–(SP)        ;COPY WORD INTO STACK
        BIC     #177770,@SP     ;ONE OCTAL VALUE
        ADD     #'0,@SP         ;CONVERT TO ASCII
        MOVB    (SP)+,–(R2)     ;STORE IN BUFFER
        ASR     R1              ;SHIFT
        ASR     R1              ;RIGHT
        ASR     R1              ;THREE
        DEC     R0              ;TEST IF DONE
        BNE     LOOP            ;NO, DO IT AGAIN
        BIC     #177776,R1      ;GET LAST BIT
        ADD     #'0,R1          ;CONVERT TO ASCII
        MOVB    R5,–(R2)        ;STORE IN BUFFER
        RTS     PC              ;DONE,RETURN
```

11.4 PROGRAMMING THE PROCESSOR STATUS WORD

The current processor status can be read and written using several programming techniques on the PSW. The PSW has an I/O address of 17 777 776. The KDJ11-B and other PDP-11 processors implement this address, whereas LSI-11 and LSI-11/2 processors do not. One technique is to use the I/O address as a source or destination address with any instruction.

```
CLR @#17777776
MOV @#17777776, R0
```

The first instruction clears the PSW and the second instruction moves the contents of the PSW to general register 0.

The PSW explicit address (17 777 776) can be accessed on a word or byte basis. The KDJ11-B recognizes the PSW odd address (17 777 777) and the access result is identical to an odd memory address reference.

Another technique is to use the two dedicated PSW instructions, MTPS and MFPS. These instructions only reference the even byte. If memory management is enabled, certain PSW bits are protected.

11.5 PROGRAMMING PERIPHERALS

Programming LSI-11 bus compatible modules (devices) is simple. A special class of instructions that deals with I/O operations is unnecessary. The bus structure permits a unified addressing structure in which control, status, and data registers for devices are directly addressed as memory locations. Therefore, all operations on these registers (such as information transfer and data manipulation) are performed by normal memory reference instructions.

The use of all memory reference instructions on device registers greatly increases the flexibility of I/O programming. For example, information in a device register can be compared directly with a value and a branch made on the result.

```
CMP RBUF,          #101
BEQ SERVICE
```

In this case, the program looks for 101 in the DLV11 receiver data buffer register (RBUF) and branches if it finds it. There is no need to transfer the information into an intermediate register for comparison.

When the character is of interest, a memory reference instruction can transfer the character into a user buffer in memory or to another peripheral device. The instruction MOV DRINBUF LOC transfers a character from the DRV11 data input buffer (DRINBUF) into a user-defined location.

All arithmetic operations can be performed on a peripheral device register. For example, the instruction ADD #10, DROUT BUF adds 10 to the DRV11 output buffer. All read/write device registers can be treated as accumulators. There is no need to funnel all data transfers, arithmetic operations, and comparisons through one or a small number of accumulator registers.

11.6 PDP-11 PROGRAMMING EXAMPLES

The programming examples that follow show how the instruction set, addressing modes, and programming techniques can be used to solve some simple problems. The format used is MACRO-11.

```
Program  Program
Address  Contents  Label   Op Code   Operand   Comments

                                               ;PROGRAMMING EXAMPLE
                                               ;SUBTRACT CONTENTS OF LOCS 700-710
                                               ;FROM CONTENTS OF LOCS 1000-1010

         000000            R0=%0
         000001            R1=%1
         000002            R2=%2
         000003            R3=%3
         000004            R4=%4
         000005            R5=%5
         000006            SP=%6
         000007            PC=%7

         000500            .=500
000500   012706  START:    MOV       #.,SP       ;INIT STACK POINTER
         000500
000504   012701            MOV       #700,R1
         000700
000510   012702            MOV       #712,R2
         000712
000514   012703            MOV       #1000,R3
         001000
000520   012704            MOV       #1012,R4
         001012
000524   005000            CLR       R0
000526   005005            CLR       R5
000430   062105  SUM1:     ADD       (R1)+,R5    ;START ADDING
000532   020102            CMP       R1,R2       ;FINISHED ADDING?
000534   001375            BNE       SUM1        ;IF NOT BRANCH BACK
000536   062300  SUM2:     ADD       (R3)+,R0    ;START ADDING
000540   020304            CMP       R3,R4       ;FINISHED ADDING?
000542   001375            BNE       SUM2        ;IF NOT BRANCH BACK

000544   160500  DIFF:     SUB       R5,R0       ;SUBTRACT RESULTS

000546   000000            HALT                  ;THAT'S ALL FOLKS

         000700            .=700
000700   000001            WORD 1,2,3,4,5
000702   000002
000704   000003
000706   000004
000710   000005

         001000            .=1000
001000   000004            WORD 4,5,6,7,8
001002   000005
001004   000006
001006   000007
001010   000010


         000500            END
```

Program Address	Program Contents	Label	Op Code	Operand	Comments
					;PROGRAM TO COUNT NEGATIVE NUMBERS
					;IN A TABLE
					;20. SIGNED WORDS
					;BEGINNING AT LOC VALUES
					;COUNT HOW MANY ARE NEGATIVE IN R0
			R0=%0		
			R1=%1		
			R2=%2		
			SP=%6		
			PC=%7		
			.=500		
		START:	MOV#.,SP		;SET UP STACK
			MOV #VALUE,R1		;SET UP POINTER
			MOV #VALUES+40.,R2		;SET UP COUNTER
			CLR R0		
		CHECK:	TST (R1)+		;TEST NUMBER
		BPL NEXT			;POSITIVE?
		INC R0			;NO, INCREMENT
					;COUNTER
		NEXT:	CMP R1,R2		;YES, FINISHED?
		BNE CHECK			;NO, GO BACK
		HALT			;YES, STOP
		VALUES:	0		
		.END			

Program Address	Program Contents	Label	Op Code	Operand	Comments
					;PROGRAM TO COUNT ABOVE AVERAGE QUIZ SCORES
					;LIST OF 16. QUIZ SCORES
					;BEGINNING AT LOC SCORES
					;KNOWN AVERAGE IN LOC AVERAGE
					;COUNT IN R0 SCORES ABOVE AVERAGE
			R0=%0		
			R1=%1		
			R2=%2		
			R3=%3		
			SP=%6		
			PC=%7		
			.=500		
		START:	MOV #.,SP		;SET UP STACK
			MOV #16.,R1		;SET UP COUNTER
			MOV #SCORES,R2		;SET UP POINTER
			MOV #AVERAGE,R3		
			CLR R0		
		CHECK:	CMP (R2)+,(R3)		;COMPARE SCORE AND AVERAGE
			BLE NO		;LESS THAN OR EQUAL
					;TO AVERAGE?
			INC R0		;NO, COUNT
		NO:	DEC R1		;YES, DECREMENT COUNTER
			BNE CHECK		;FINISHED? NO, CHECK
			HALT		;YES, STOP
		AVERAGE:	65.		
		SCORES*	25.,70.,100.,60.,80.,80.,40.		
			55.,75.,100.,65.,90.,70.,65.,70.		
			.END		

```
Program  Program
Address  Contents  Label   Op Code      Operand   Comments

                                                  ;PROGRAMMING EXAMPLE
                                                  ;ACCEPT (IMMEDIATE ECHO) AND
                                                  ;STORE 20. CHARS
                                                  ;FROM THE KEYBOARD, OUTPUT CR & LF
                                                  ;ECHO ENTIRE STRING FROM STORAGE

                         R0=%0
                         R1=%1
                         SP=%6
                         CR=15
                         LF=12
                         TKS=177560
                         TKB=TKS+2
                         TPS=TKB+2
                         TPB=TPS+2

                         .TITLE ECHO

                         .=1000
                 START:  MOV     #.,SP     ;INITIALIZE STACK POINTER
                 MOV     #SAVE+2,R0        ;SA OF BUFFER
                                           ;BEYOND CR & LF
                 MOV     #20.,R1           ;CHARACTER COUNT

                 IN:     TSTB    @#TKS     ;CHAR IN BUFFER?
                         BPL     IN        ;IF NOT BRANCH BACK
                                           ;AND WAIT
                 ECHO:   TSTB    @#TPS     ;CHECK TELEPRINTER
                                           ;READY STATUS
                 BPL     ECHO
                 MOVB    @#TKB,@#TPB       ;ECHO CHARACTER
                 MOVB    @#TKB,(R0)+       ;STORE CHARACTER AWAY
                 DEC     R1
                 BNE     IN                ;FINISHED INPUTTING?

                 MOV     #SAVE,R0          ;SA OF BUFFER INCLUDING
                                           ;CR & LF
                 MOV     #22.,R1           ;COUNTER OF BUFFER
                                           ;INCLUDING CR & LF

         OUT:    TSTB    @#TPS             ;CHECK TELEPRINTER
                                           ;READY STATUS
                 BPL     OUT
                 MOVB    (R0)+,@#TPB       ;OUTPUT CHARACTER
                 DEC     R1
                 BNE     OUT               ;FINISHED OUTPUTTING?
                 HALT

         SAVE:   .BYTE   CR,LF
                 .=.+20,
                 .END
```

Program Address	Program Contents	Label	Op Code	Operand	Comments
					;PROGRAMMING EXAMPLE
					;SUBROUTINE TO INPUT TEN VALUES
		INPUT:	MOV	#BUFFER,R0	;SET UP SA OF
					;STORAGE BUFFER
			MOV	#−10.,R1	;SET UP COUNTER
		IN:	TSTB	@#TKS	;TEST KYBD READY STATUS
			BPL	IN	
		OUT:	TSTB	@#TPS	;TEST TTO READY STATUS
			BPL	OUT	
			MOVB	@#TKB,@#TPB	;ECHO CHARACTER
			MOVB	@#TKB,(R0)+	;STORE CHARACTER
			INC	R1	;INC COUNTER
			BNE	IN	
			RTS	PC	;EXIT
					;PROGRAMMING EXAMPLE
					;SUBROUTINE TO SORT TEN VALUES
		SORT:	MOV	#−10.,R4	
		NEXT:	MOV	COUNT,R3	
			MOV	#BUFFER+9.,R0	
			ADD	R3,R0	
			MOVB	(R0)+,R1	
		LOOP:	CMPB	(R0)+,R1	
			BGE	GT	
		LT:	MOVB	−(R0),R2	
			MOVB	R1,(R0)+	
			MOV	R2,R1	
		GT:	INC	R3	
			BNE	LOOP	
		INSERT:	MOVB	R1,BUFFER+10.(R4)	
			INC	R4	
			INC	COUNT	
			BNE	NEXT	
			MOV	#−9.,COUNT	;RESTORE LOCATION COUNT
			RTS	PC	;EXIT
		COUNT:	.WORD	−9.	
		LINE1:	.ASCII	/INPUT ANY TEN SINGLE-DIGIT VALUES (0-9); I'LL/	
			.ASCII	/SORT AND OUTPUT THEM IN/	
		LINE2:	.ASCII	/SMALLEST TO LARGEST ORDER./	
		BUFFER:	.=.+10.		
			.END	INITSP	;FINISHED!!!

Program Address	Program Contents	Label	Op Code	Operand	Comments
					;PROGRAMMING EXAMPLE
					;SUBROUTINE EXAMPLE
					;INPUT TEN VALUES, SORT, AND
					;OUTPUT THEM IN SMALLEST TO LARGEST ORDER
			R0=%0		
			R1=%1		
			R2=%2		
			R3=%3		
			R4=%4		
			R5=%5		
			SP=%6		
			PC=%7		
			TKS=177560		
			(address of terminal control status register)		
			TKB=TKS+2 - (terminal data buffer register)		
			TPS=TKB+2 - (terminal output control and status registers)		
			TPB=TPS+2 - (terminal output data buffer)		
			.=3000		
		INITSP:	MOV #.,SP		;INITIALIZE STACK POINTER
			JSR PC,CRLF		;GO TO CRLF SUBROUTINE
			JSR R5, OUTPUT		;GO TO OUTPUT SUBROUTINE
			LINE1		;SA OF LINE 1 BUFFER
			69.		;NUMBER OF OUTPUTS
			JSR PC,CRLF		;GO TO CRLF SUBROUTINE
			JSR R5,OUTPUT		;GO TO OUTPUT SUBROUTINE
			LINE2		;SA OF LINE 2 BUFFER
			26.		;NUMBER OF OUTPUTS
			JSR PC,CRLF		;GO TO CRLF SUBROUTINE
			JSR PC,INPUT		;GO TO INPUT SUBROUTINE
			JSR PC,SORT		;GO TO SORT SUBROUTINE
			JSR PC,CRLF		;GO TO CRLF SUBROUTINE
			JSR R5,OUTPUT		;GO TO OUTPUT SUBROUTINE
			BUFFER		;INPUT BUFFER AREA
			10.		;NUMBER OF OUTPUTS
			JSR PC,CRLF		
			HALT		;THE END!!!

```
Program  Program
Address  Contents  Label    Op Code          Operand   Comments

                                                      ;PROGRAMMING EXAMPLE
                                                      ;SUBROUTINE TO OUTPUT A CR & LF

                   CRLF:    TSTB @#TPS                ;TEST TTO READY STATUS
                            BPL CRLF
                            MOVB #15,@#TPB            ;OUTPUT CARRIAGE RETURN
                   LNFD:    TSTB @#TPS                ;TEST TTO READY STATUS
                            BPL LNFD
                            MOVB #12,@#TPB            ;OUTPUT LINE FEED
                            RTS PC                    ;EXIT

                                                      ;SUBROUTINE TO OUTPUT A
                                                      ;VARIABLE LENGTH MESSAGE
                   OUTPUT:  MOV (R5)+,R0              ;PICK UP SA OF DATA BLOCK
                            MOV (R5)+,R1              ;PICK UP NUMBER OF OUTPUTS
                            NEG R1                    ;NEGATE IT
                   AGAIN:   TSTB @#TPS                ;TEST TTO READY STATUS
                            BPL AGAIN
                            MOVB (R0)+,@#TPB          ;OUTPUT CHARACTER
                            INC R1                    ;BUMP COUNTER
                            BNE AGAIN
                            RTS R5
```

11.7 LOOPING TECHNIQUES

Looping techniques are illustrated in the program segments below. The segments are used to clear a 50-word table.

1. Autoincrement (pointer address in general purpose register)

```
                   R0=%0
                   MOV #TBL,R0
LOOP:              CLR (R0)+
                   CMP R0,#TBL+100.
                   BNE LOOP
```

2. Autodecrement (pointer and limit values in general purpose register)

```
                   R0=%0
                   R1=%1
                   MOV #TBL,R0
                   MOV #TBL+100.,R1
LOOP:              CLR - (R1)
                   CMP R1,R0
                   BNE LOOP
```

3. Counter (decrementing a general purpose register containing count)

```
        R0=%0
        R1=%1
        MOV #TBL,R0
        MOV #50.,R1
LOOP:   CLR (R0)+
        DEC R1
        BNE LOOP
```

4. Index Register Modification (indexed mode, modifying index value)

```
        R0=%0
        CLR R0
LOOP:   CLR TBL (R0)
        ADD #2,R0
        CMP R0,#100.
        BNE LOOP
```

5. Faster Index Register Modification (storing values in general purpose register)

```
        R0=%0
        R1=%1
        R2=%2
        MOV #2,R1
        MOV #100.,R2
        CLR R0
LOOP:   CLR TBL (R0)
        ADD R1,R0
        CMP R0,R2
        BNE LOOP
```

6. Address Modification (indexed mode, modifying base address)

```
        R0=%0
        MOV #TBL,R0
LOOP:   CLR 0(R0)
        ADD #2,LOOP+2
        CMP LOOP+2,#100.
        BNE LOOP
```

APPENDIX A
ROM CODE DIFFERENCES

A.1 GENERAL

The KDJ11-B module uses two ROMs that contain the boot and diagnostic coding described in Chapter 4. The original version is designated as V6.0 and the revised or updated versions are V7.0 and V8.0. The user does not have to remove the module from the system for identification because the version number is shown in the upper right hand corner of the display whenever the setup mode is entered. The ROM part numbers associated with each version are shown in Table A-1. The differences between V6.0 and V7.0 are detailed in Paragraph A.2, while the differences between V7.0 and V8.0 are covered in Paragraph A.3.

Table A-1 ROM Part Numbers

Socket	V8.0 Set	V7.0 Set	V6.0 Set
Low byte E116	23-168E5	23-116E5-00	23-077E5-00
High byte E117	23-169E5	23-117E5-00	23-078E5-00

A.2 V6.0 AND V7.0 DIFFERENCES

A.2.1 Boot Support for Tape MSCP Devices (TK50/TU81)

V7.0 has a built-in tape MSCP boot program for the TK50/TU81 devices and the device name is MU. The tape MSCP boot and the disk MSCP boot are combined into one common boot program.

V6.0 does not have a tape MSCP boot program for the TK50/TU81 devices. Unibus systems could boot these devices if an M9312 type boot ROM for tape MSCP devices could be installed in the UBA module, but this type of boot ROM is not available.

A.2.2 Disk MSCP Automatic Boot Routine

In the V7.0 MSCP automatic boot, the program tries to boot removable media units from 0 to 255 and then to boot fixed media units from 0 to 255. The program attempts to boot each unit at the standard MSCP address and if this fails, the boot program attempts the same unit number from the first floating disk MSCP device (if it is present) before continuing to the next unit number. The routine always makes the first pass trying to boot the removable media units and the final pass trying the fixed media units.

In the V6.0 MSCP automatic boot (device name A), the program tries to boot removable media units from 0 to 7 and then to boot fixed media units from 0 to 7. It only tries to boot the drives attached to the controller at the standard address of 172 150. The MSCP automatic boot does not support unit numbers above 7 and it hangs if the controller has a response from a unit number greater than 7.

The first floating controller (when present) is at address 160 334, if there are no devices from 160 010 to 160 330. The main advantage of V7.0 is to allow the user to add a second disk MSCP device without making any entries into the translation table (as long as the controller address is set exactly according to the floating CSR address rules).

A.2.3 Dialog Mode Boot Command for Disk MSCP Boot

V7.0 of the dialog mode lets the user execute the boot command for a DU device and the ROM code tries to boot the selected unit number at the standard controller address. If the boot is not successful, the ROM code then tries to boot the same unit number at the first floating controller address (if it is present). When an error occurs on both controllers, the V7.0 ROM code prints out error messages for both controllers starting with the standard address. Nonexistent error messages are not printed unless the unit is nonexistent on both controllers. If the second controller does not exist at the proper floating address, the ROM code prints out messages associated with the standard controller only. When the translation table or the /A switch is used, only one controller is tried regardless of the existence of two or more controllers.

V6.0 of the boot routine tries the standard address only, unless otherwise directed by the translation table or the /A switch.

A.2.4 Disk MSCP Boot (DU)

The V7.0 disk MSCP boot always initializes the disk controllers when they are first accessed. The controller is left on-line, unless it is necessary to take it off-line. This allows the boot to operate faster in the automatic boot mode when many unit numbers and possibly multiple controllers are being tried. The controller is always turned off before control is transferred to the secondary boot. The V7.0 DU/MU boot requires a 16-Kword memory (minimum) and the V6.0 DU boot requires an 8-Kword memory (minimum).

In V6.0, the controller is initialized only when the SA register is not zero. The controllers are usually left on and are turned off before transferring control to the secondary boot. The controller is also turned off before checking for a valid boot block. Therefore, if the automatic boot sequence 'sees' a lot of nonbootable media before it gets to the device being booted, the boot code may be slow since it has to reinitialize the controller after each nonbootable unit is found.

A.2.5 8-Unit Restriction for MSCP Automatic Boot

V6.0 is restricted to units 0 through 7 and if the first unit on the controller is unit 8 or greater, the boot loops because the automatic boot program does not correctly handle unit numbers greater than 7. V7.0 can handle unit numbers from 0 through 255.

A.2.6 Irregular Monitoring of Keyboard During Automatic Boot Sequence

As the ROM code proceeds through the devices during the V6.0 automatic boot, it does not check the keyboard for a <CTRL> C unless a specific boot program does it. The keyboard is sometimes checked by a boot when the boot program is in a potentially long loop waiting for some action to occur. V7.0 checks the keyboard at least once between each boot in the automatic boot sequence.

A.2.7 Addition of Single-Letter Mnemonic in Automatic Boot List

A single-letter mnemonic (L) has been added to the boot command list for V7.0. The L command causes the automatic boot sequence to loop continuously until one of the selected devices is successfully booted. Normally, the last device in the automatic boot table is followed with the mnemonic E, which causes the sequence to exit at the end of the table, and if no device is successfully booted, the ROM code displays an error message and requests input before proceeding.

When the L follows the last device, the ROM code restarts the table at the beginning and continuously tries each device in the table until one is booted or the user types <CTRL> C to abort the sequence. This feature is useful for booting a fault-tolerant system that must be tried continuously until a successful boot occurs.

```
.=10000                                    ; Program is relocatable to another
                                           ; address.

START:    tstb    @#177560                 ; Has any characters been typed
          bpl     10$                      ; No-Go exit back to auto boot
                                           ; Yes-Check the character
          movb    @#177562,r5              ; Get the character from the RBUF
          bic     #177600,r5               ; Clear off all bits above bit 07
          cmp     r5,#3                    ; Is the character a CTRL C ?
          beq     20$                      ; Yes-Then return to ROM code with
                                           ; r5 set to 3 which will cause the
                                           ; boot sequence to be aborted.

10$:      mov     #301,r5                  ; Load r5 with value for drive error
          movb    #100,@#177611            ; This will fake out the ROM code
                                           ; and make it restart the auto boot
                                           ; sequence

20$       bic     #760,@#177520            ; Make sure the ROMs are selected in
                                           ; the BCSR
          jmp     @#165762                 ; Return to the ROM code.
                                           ; If r5 is 301 then restart the auto
                                           ; boot sequence. If r5 is 3 then
                                           ; abort the sequence and go to Dialog
                                           ; mode.
```

MR-17272

Figure A-1 Program for Continuous Loop

The L command is not included in V6.0, but the user can implement it by writing a small EEPROM boot to emulate the feature. The source code and the description of this program (to enable a continuous loop function for V6.0) are shown in Figure A-1. When this feature is implemented, it must be noted that there is no boot program using a device name of L, and if there is, the user has to delete or rename that boot before using the new program.

A.2.8 Setup Mode Disable

V7.0 includes a disable parameter on the list of parameters used by setup command 2. This command was added to prevent unauthorized entry into setup mode and it allows the user to disable entry into setup mode if the forced dialog mode is not selected. This change assumes that the forced dialog mode switch is controlled or that switch 5 on the module is on to prevent unauthorized entry into setup mode. When the ROM code is in dialog mode and setup mode is disabled, all references to the setup commands are eliminated, and typing SETUP causes an invalid command response from the ROM code. In V6.0, the setup mode can always be entered from dialog mode.

A.2.9 Disable All Testing Parameter

V7.0 includes a disable testing parameter on the list of parameters used by setup command 2. When this parameter is set or selected, it disables all memory and cache testing if the forced dialog mode is not selected. (The forced dialog mode causes the module to run the complete set of tests.) This reduces the testing time to approximately 70 or 85 ms. This parameter is not available in V6.0.

A.2.10 Edit/Create Command

In V7.0, the edit/create command of the setup mode uses a decimal value for the highest unit number entry on the EEPROM boots. V6.0 uses an octal number that is converted into a decimal number.

A.2.11 Initialize Command for the PMG Counter

The initialize command sets the PMG count value to 7 in V7.0. This value was set to 0 in V6.0. The recommended value for the PMG count is 7 for all modules that use V6.0.

NOTE

It is recommended that users of V6.0 change the PMG count value from the default value of 0 to a value of 7.

A.2.12 PMG Parameter Warning

V7.0 prints a warning message if the PMG count value is set to 0 by the user. The warning was created to prevent the user from operating the system with a PMG count value of 0. This ensures that the CPU is not locked out from the bus for excessively long periods of time, which could cause some loss of data if it is stalled for more than 250 ms. The message shows the PMG count value being changed and prints the warning with the parameter line being reprinted, allowing the user to change the PMG count value. The display also contains the current values associated with the selections available to the user (Figure A-2) and thus eliminates the need to consult a reference document. V6.0 prints only the parameter selected and the values the user may select (Figure A-2).

A.2.13 Setup Command 4 Printout

V7.0 prints descriptions of the single-letter mnemonics A, B, E, and L when they are used by setup command 4. V6.0 prints only the descriptions for A and E because there are no descriptions for B and L. The V7.0 descriptions are shown in Figure A-3.

A.2.14 MU (TK50/TU81) Device

V7.0 adds the device name MU for the TK50 or TU81 to the list of devices in the automatic boot selections table. This is also added to the list when the setup mode initialize command is executed. V6.0 does not have the MU device name. The setup command 4 automatic boot lists are shown in Table A-2 for both versions.

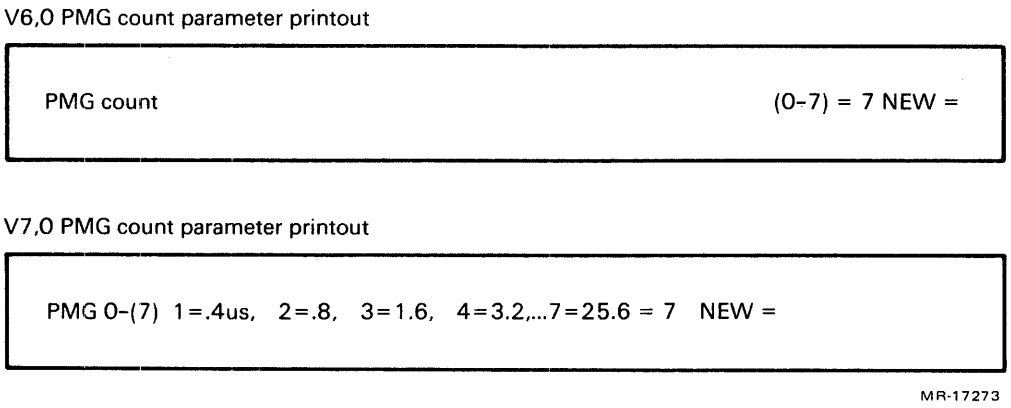


Figure A-2 PMG Count Value Warning Message

```
KDJ11-B Setup mode
Press the RETURN key for Help
Type a command then press the RETURN key: 4 <CR>

List/change the Automatic boot selections in the Setup table

A = MSCP automatic boot
B = External ROM boot
E = Exit automatic boot
L = Loop continuously

Boot 1 = A
Boot 2 = DLO
Boot 3 = MSO
Boot 4 = MUO
Boot 5 = E
Boot 6 = blank

Type CTRL Z to exit or press the RETURN key for No change

Boot 1 = A
Device name =
```

MR-17274

Figure A-3 Single-Letter Descriptions for Command 4

Table A-2 Setup Command 4 Automatic Boot Lists

V7.0	V6.0
A	A
DL0	DL0
MS0	MS0
MU0	A
A	

A.2.15 Setup Command 5

Setup command 5 is eliminated in V7.0. The setup command 5 description is reserved and if the command is selected, it is ignored.

In V6.0, this command allows different character sets in the console terminal to automatically be selected by the ROM code when the user changes from English to a local text or from local to English text. The command is no longer required since all text printed on the screen uses only the standard ASCII characters generally available on all terminals. Special characters used in some languages are imitated by fallback representations in standard ASCII.

A.2.16 Memory Initialization at Power-Up

V7.0 writes to all consecutive memory starting at location 0 at least once after the power-up sequence is complete. This feature is disabled if the disable testing option is enabled. This option does not apply to restarts. V6.0 may not write to locations above 248 Kbytes if the long memory test is disabled or <CTRL> C is typed.

A.2.17 Power-Up Option Set to 3 with Battery Backed Up Memory

In V7.0, if the selected power-up mode is 3 and the battery indicates that the voltages are lost with the ignore battery function turned off, the ROM code goes to the dialog mode regardless of the restart mode selection. For the same conditions in V6.0, the ROM code executes the restart mode selection if it is not mode 3 or it goes to the dialog mode. The battery OK signal is currently used only in Unibus systems.

A.2.18 Halt-on-Break

V7.0 sets the halt-on-break bit in the BCSR immediately after the "Testing in progress – Please wait" message is displayed. The halt-on-break feature, generally used in single-user environments, was not needed.

V6.0 does not set the halt-on-break bit in the BCSR until a break is received and discarded, any valid character is received except XON, or the ROM code gives up control of the CPU. This allows the ROM code to ignore any breaks that come as a result of a terminal being powered up.

A.2.19 Local Language Support

V7.0 supports local language translations by using the <CTRL> L command. Local language is not supported in V6.0.

A.2.20 Addition of Map Command Feature

V7.0 adds an additional feature to the map command. This feature determines the clock speed of the CPU by counting the number of SOB instructions that can be executed out of the cache memory during one 20 ms cycle of the internal DLART clock. This value is compared with a table of standard values and if it is within 0.1% of any standard value, that value is displayed. If it does not match a standard value, the actual value is displayed. The standard values are 15.206, 17, 18, 19, and 20. The speed is not calculated if any errors are detected during testing.

A.2.21 EEPROM Load Error Before Dialog Mode

In V7.0, if the setup mode is entered and an error occurs in loading the EEPROM data into memory, the dialog mode is restarted and no error message is generated. V6.0 does not check to verify the data is OK and setup mode cannot be entered without testing memory. In either case, a timeout may occur and trap to location 4 with an error message being generated.

A.2.22 Test Command Decimal Numbers

In V6.0 dialog mode, when the user selects a specific test with the test command, the ROM code selects a different test number. The valid test numbers are in the range of 31 to 70 (octal) with the exception of tests 64 to 66 and any Unibus test on LSI-11 bus systems. The only test numbers that may cause confusion are illegal test numbers that end in 8 or 9 using the decimal system. Table A-3 lists the selectable (illegal) test numbers and the actual test run by the ROM code.

Table A-3 ROM Code Test Selections

Selected Test	Actual Test
78	70
69	61
68	60
59	51
58	50
49	41 (Unibus only)
48	40 (Unibus only)
39	31 (Unibus only)

V6.0 *does* echo the correct and actual test being run. For example, if the user selects T 59, then V6.0 responds that it is looping on test 51. V7.0 corrects the problem.

A.2.23 Test Command Execution of a Single Test

In V7.0, if the test command is used and a specific test is selected, the memory size routine is run before the selected test is run. Some of the memory size parameters have been lost and need to be replaced. V6.0 runs only the selected test when a single test is selected.

A.2.24 Test Errors in Tests 72 to 75

In V6.0, if an error occurs in tests 72 to 75, the user has a choice of either rerunning the test or looping on the test. It does not matter what the user selects, however, because the ROM code unconditionally restarts from the beginning if the user selects a valid choice. For V7.0, the user is only allowed to rerun the test, but the ROM code still restarts the code from the beginning when this selection is made.

A.2.25 Bypass Errors for Test Failures

In V6.0, if an error occurs during testing, the user may bypass the test if the error is considered to be nonfatal. Many times it is difficult to determine if an error is fatal or nonfatal and sometimes, if an error is determined to be nonfatal, it may still cause a problem when overridden.

V7.0 considers all errors to be fatal and never provides the override command. However, the user can still override the error in the same way used for V6.0. To override, the user types <CTRL> O and then types 4 <RETURN>. If <CTRL> O is not typed, the 4 is ignored and not echoed.

A.2.26 Test 76 and 75 Error Messages

During the first two major tests, 76 and 75, the printout for errors has been changed. These tests have a simple printout because the normal printout routine has not been turned on at this time. V6.0 prints "Error 76" or "Error 75" and V7.0 prints "A 76" or "A 75." This change was made for local language applications since these printouts cannot be translated.

A.2.27 Starting Automatic Boot Sequence Message

V7.0 prints a message indicating when the automatic boot mode is selected and the sequence is starting. This message (Figure A-4) indicates that all the testing is complete and the ROM code is starting the automatic boot sequence.

NOTE

This does not apply to LSI-11 systems with the friendly format feature selected by setup command 2.

```
Testing in progress - Please wait
Memory size is 512 K Bytes
9 Step memory test
  Step 1 2 3 4 5 6 7 8 9

Starting system
```

MR-17275

Figure A-4 Automatic Boot Sequence Message

A.2.28 Device Name and Number After Message

V7.0 prints the device mnemonic and unit number after the "Starting system" message shown in Figure A-5. This has no affect on the printout when the user friendly printout feature is selected.

A.2.29 Incorrect Message for Invalid Unit Number

V6.0 responds with an incorrect message (Figure A-5) when the user types in a unit number greater than 255. V7.0 corrects this problem by printing a message (Figure A-6) indicating the invalid unit number.

A.2.30 Dialog Mode Header Message

V7.0 changes the dialog mode header message by deleting the brackets because they are not available on all terminals.

A.2.31 Map Command Message

V7.0 changes the & symbol to "and" for the map command message in setup mode because the symbol is not available on all terminals.

A.2.32 List Device Descriptions

V7.0 changes the descriptions for the device names in some of the mnemonics and also shows the TK25 and TS05 devices under the mnemonic MS for Unibus systems. The differences are not listed here because they are obvious. RA80/81/60, for example, is changed to RA80, RA81, and RA60.

A.2.33 Loss of the First Line in a List Header

In V6.0 dialog mode, when the user types the boot command without the device and then types <RETURN> ? to get a list of boot devices, the ROM code does not send a line feed before the header of the list and the header is lost in the right margin (Figure A-7). The list *is* typed out correctly. V7.0 corrects the problem and the message shown in Figure A-8 is displayed.

A.2.34 <CTRL> R and <CTRL> U Echo

V6.0 echoes the <CTRL> R and <CTRL> U commands as R and U, respectively. V7.0 does not echo these commands because the symbols are not available on all terminals. These commands still function the same way.

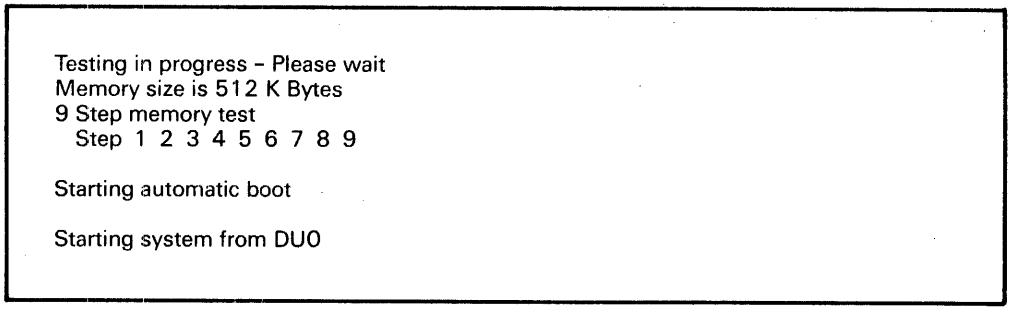


Figure A-5 V6.0 Incorrect Message

```
Commands are Help, Boot, List, Setup, Map, Test.
Type a command then press the RETURN key: B DL300 <CR>

Invalid unit number

Commands are Help, Boot, List, Setup, Map, Test.
Type a command then press the RETURN key:
```

MR-17277

Figure A-6 V7.0 Correct Error Message

```
Commands are: [Help, Boot, List, Setup, Map, Test]
Type a command then press the RETURN key: B <CR>

Enter the device name and unit numer then press the RETURN key: ?

name      numbers    Source     Device type

DU        0-255      CPU ROM    MSCP (RA80/81/60, RD51/52, RX50, RC25,...)
DL        0-3        CPU ROM    RL01/RL02
          etc.....
```

MR-17278

Figure A-7 V6.0 List Header Error

```
Commands are Help, Boot, List, Setup, Map, Test.
Type a command then press the RETURN key: B <CR>

Enter the device name and unit number then press the RETURN key: ?

Device    Unit       CPU ROM    RA80, RA81, RA60, RD51, RD52, RX50,RC25
name      numbers    CPU ROM    RL01, RL02

DU        0-255
DL        0-3
          etc.....
```

MR-17279

Figure A-8 V7.0 Correct List Header

A.2.35 Power-Up or Restart Mode Set to 3 (LSI Bus Only)

In V6.0, before executing a power recovery trap through location 24, the ROM code does the following.

1. Reads and stores the contents of location 24
2. Executes a read/write test on location 24
3. Restores the original contents of location 24

When the test is successfully completed, the ROM code loads the contents of location 26 into the PSW and jumps to the location specified in location 24. Since this location was tested, the ROM code cannot be present in the lower portion of memory.

V7.0 does not test location 24 and it *is* possible to have ROM code in the lower portion of memory. The ROM code loads the contents of location 26 into the PSW and jumps to the location specified in location 24.

A.2.36 Automatic Boot Misleading Error Message (LSI Bus Only)

In V6.0, R5 is cleared at the end of the MSCP disk sniffer boot and this causes all errors that occurred during the sniffer to appear to be correctable by the user. This minor problem only affects the message sent to users operating in the friendly mode. V7.0 corrects the problem.

A.2.37 APT Halt-on-Break Detect (LSI Bus Only)

V6.0 can detect breaks coming from an APT system. This feature allows LSI type systems that have the halt-on-break option disabled to halt, and enables the halt-on-break option if the APT is trying to down-line-load. V7.0 eliminates this feature because it is implemented in the manufacturing process. If the feature is not eliminated, there is a small chance that the system may be halted with halt-on-break disabled if the terminal is a VT5**X** terminal (or possibly other terminals), but not if it is a VT1**XX** or VT2**XX** terminal.

Note also that autobaud detect routines from remote hosts *can* cause halt-on-break when it is not desired.

A.2.38 B Mnemonic for ROM Boots (Unibus Only)

For V7.0 under the B mnemonic for ROM boots, the address located at 173 024 on the M9312 module in the Unibus system must be an even address. This is the only check of the address data. For V6.0 under the B mnemonic for ROM boots, the address located at 173 024 must be 165 000 or greater, but it *can* be odd. In either case, if all the conditions are not met, an invalid device message is reported.

A.2.39 Error in List Command When Unknown ROM is Found (Unibus Only)

In V6.0, the ROM board for the Unibus must respond to all addresses from 17 773 000 to 17 773 776 for the ROM code to transfer control using the B mnemonic, or else an invalid device message is reported. In V7.0, only address 17 773 024 must respond.

A.2.40 Power-Up or Restart Mode Set to 3 (Unibus Only)

In V6.0, the ROM code checks for the presence of Unibus memory and sets up the KMCR before executing a power recovery trap through location 24. Then the ROM code does the following.

1. Reads and stores the contents of location 24
2. Executes a read/write test on location 24
3. Restores the original contents of location 24

When the test is successfully completed, the ROM code loads the contents of location 26 into the PSW and jumps to the location specified in location 24. Since this location was tested, the ROM code cannot be present in the lower portion of memory.

V7.0 does not check for Unibus memory and assumes that by selecting mode 24 the system has the final configuration of memory already installed. Therefore, location 24 is not tested and it *is* possible to have ROM code in the lower portion of memory. The ROM code loads the contents of location 26 into the PSW and jumps to the location specified in location 24.

A.2.41 Saving KMCR Bits <5:0> in the EEPROM (Unibus Only)

In V7.0, when the setup table is written into the EEPROM, the contents of KMCR bits <5:0> are always copied into the EEPROM regardless of the power-up or restart modes. The EEPROM data is used to load the KMCR when the ODT or 24/26 modes are selected. The ROM code autosizes for Unibus memory when the automatic boot or dialog modes are selected.

In V6.0, KMCR bits <5:0> are copied into the EEPROM only when ODT is selected for the power-up or restart mode. The ODT mode is the only mode that does not autosize for Unibus memory and consequently must depend on the EEPROM to contain the correct KMCR information.

A.3 V7.0 AND V8.0 DIFFERENCES

This paragraph describes the changes made when V8.0 of the ROM code was created. The changes made in V7.0 (as described in Paragraph A.2) are still true for V8.0 except as noted below. Paragraph A.2 describes the differences between V6.0 and V7.0 only.

A.3.1 M9312 MultiROM Bootstrap Support (PDP-11/84 Only)

V6.0 and V7.0 do not support M9312 bootstrap programs, which require more than one ROM to implement (multiROM bootstraps). The only way these programs can be supported for V6.0 and V7.0 is to use a work-around program loaded into the EEPROM (refer to the *PDP-11/84 Technical Manual*). V8.0 corrects this problem and automatically supports M9312 multiROM bootstraps.

NOTE
This problem occurs only in PDP-11/84 systems.

A.3.2 Small Memory Automatic Boot Problem for RQDX3

V7.0 has a check in the MSCP initialize sequence that assumes the disk controller starts step 1 within 100 μseconds of a hard initialize command. This is not true of many RQDX3 controller modules at power-up. The problem happens in small memory systems (less than 1 Mbyte) and on large memory systems if some of the memory tests are bypassed. The problem occurs only at power-up. V8.0 allows at least 10 seconds for step 1 to start (as in V6.0).

A.3.3 RAnn Disk Spinup Time Delay for Automatic Boot

In V6.0 and V7.0, the disk MSCP bootstrap assumes that off-line error codes from the disk being booted are correct. If the disk is an RA**nn** on a UDA/KDA controller, and if the disk is spinning up or down, it may incorrectly identify a disk spinning up as being off-line (not available). This causes the ROM code to skip this unit and try another even though there is no problem.

V8.0 works around this problem with the following strategy. It identifies the controller as a KDA50, UDA50, or UDA50A. The identification is done in step 4 of the initialize sequence. If the device is not a KDA/UDA controller, the delay is not present. If the controller type is UDA/KDA and the response packet from the controller is an off-line code (3), the ROM code repeatedly tries to boot the device for a period of at least 60 seconds. RA**nn** devices need this delay time to spin up and be ready to respond to the host. If the RA**nn** is not ready to be booted after 60 seconds, the code reports the error and sets a flag preventing this delay time from occurring again unless the code is rebooted. The next device specified in the automatic boot setup table is then tried. The code responds to the terminal shortly if it cannot find a bootable device.

In a case where the code enters dialog mode, it is assumed that the user has the RA**nn** spun up and ready. The code does not wait 60 seconds for the device to spin up. The device promptly reports any errors if they occur. Some RA**nn** devices (possibly RA60) work adequately without this change.

CAUTION

When a system is configured with RAnn disks (and possibly with other non-RAnn MSCP disks), it is important to realize that the wait loop in V8.0 delays the automatic boot process for 60 seconds or more. It is recommended that the user remove A (disk MSCP automatic boot) from the boot table in the EEPROM using setup command 4. The user should replace it with the desired order of devices to be booted (i.e., DU0, DU2, etc.). This is especially true when booting fixed media devices, since the disk automatic boot ignores fixed media devices until it has tried all removable media devices.

Remember that the disk automatic boot tries each unit at the standard controller address and then at the first floating address. This is also true for individual unit numbers (i.e., DU2, DU0, etc.) unless the unit number is described in the translation table (set-up mode command 3).

A.3.4 Addition of RESET Instruction at Beginning of Code

V8.0 executes a RESET instruction (bus reset) at the beginning of the code. V6.0 and V7.0 do not include this instruction. This change provides a bus reset after POK is asserted.

A.3.5 Addition of New Setup Command 5

V8.0 adds a new setup mode command 5, similar to the setup mode command 5 in V6.0. V7.0 does not have a setup mode command 5. This new command in V8.0 allows the user to store up to 20 bytes of information in the EEPROM. The data is stored in the same place in the EEPROM as for V6.0. However, the information stored there is never sent to the console as it was in V6.0. The data must be entered as octal numbers in the range of 0 to 377. This command may be used to store serial numbers, etc. The setup mode initialize command resets this data to 0. The ROM code does not use this data for any purpose at all.

A.3.6 Memory Test Coverage Problem

V6.0 and V7.0 test only the first 4 Kwords of memory when running test 50. V8.0 corrects this and checks all available consecutive memory. Test 50 checks two locations for floating 1s and 0s and does byte testing.

A.3.7 List Command Device Descriptions

Some of the messages printed during the V8.0 list command are new. The changes are given in Table A-4.

A.3.8 Manufacturing Test Loop Problem

The manufacturing test loop in V7.0 does not execute all of its tests. V8.0 corrects the problem. V6.0 always worked correctly. This change affects only those manufacturing sites that use the feature. The manufacturing test loop can only be selected by using the switchpack on the CPU module.

Table A-4 New List Command Device Descriptions

Message Type	From:	To:	Comments
DU	RD51, RD52, RC25, RA80, RA81, RA60	RD**nn**, RX**nn**, RC25, RA**nn**	
XH	DECnet DEQNA	DECnet Ethernet	11/73 or 11/83 only
XE	DECnet DEQNA	DECnet Ethernet	11/84 only (if ROM present)

APPENDIX B
SETUP PARAMETER WORKSHEETS

B.1 PURPOSE

The purpose of these worksheets is to report and confirm the setup parameters contained in the setup EEPROM on the KDJ11-B CPU module.

B.2 FUNCTION

The worksheets are to be filled out when the KDJ11-B module is installed and should contain all pertinent information on the setup parameters selected. When complete, they should be left with the system as a reference and may also be used to program a replacement module in the future.

The original data should be written in ink and any new data should be added in pencil. The user sets the configuration as follows.

1. Setup command 7 lists the original values to ensure that the changes are being programmed correctly, and setup command 1 is used to exit.

2. Setup command 9 copies any changes the user makes to the setup table in the EEPROM.

3. Setup command 14 writes a boot program from memory into the EEPROM.

Item	Parameter	Selections	Original	Currrent
A	Enable halt-on-break	(0) = No (1) = Yes	= 1	
B	Disable user friendly format	(0) = No (1) = Yes	= 1	
C	ANSI video terminal	(0) = No (1) = Yes	= 1	
D	Power-up	(0) = Dialog (1) = Automatic (2) = ODT (3) = 24	= 0	
E	Restart	Same as power-up	= 2	
F	Ignore battery	(0) = No (1) = Yes	= 0	
G	PMG count	Select from 0–7	= 5	
H	Disable clock CSR	(0) = No (1) = Yes	= 1	
I	Force clock interrupts	(0) = No (1) = Yes	= 1	
J	Clock frequency	(0) = Power supply (1) = 50 Hz (2) = 60 Hz (3) = 800 Hz	= 3	
K	Enable EEC test	(0) = No (1) = Yes	= 1	
L	Disable long memory test	(0) = No (1) = Yes	= 0	
M	Disable ROM	(0) = No (1) = Disable 165 (2) = Disable 173 (3) = Disable both	= 3	
N	Enable trap-on-halt	(0) = No (1) = Yes	= 1	
O	Allow alternate boot block	(0) = No (1) = Yes	= 0	

Type <CTRL> Z to exit or press <RETURN> to proceed.

Setup Command 3 Selections

Bootstrap		Original	Current
TT1			
Device name	=		
Unit #	=		
CSR address	=		
TT2			
Device name	=		
Unit #	=		
CSR address	=		
TT3			
Device name	=		
Unit #	=		
CSR address	=		
TT4			
Device name	=		
Unit #	=		
CSR address	=		
TT5			
Device name	=		
Unit #	=		
CSR address	=		
TT6			
Device name	=		
Unit #	=		
CSR address	=		
TT7			
Device name	=		
Unit #	=		
CSR address	=		
TT8			
Device name	=		
Unit #	=		
CSR address	=		
TT9			
Device name	=		
Unit #	=		
CSR address	=		

Setup Command 4 Selections

Program		Original	Current
Boot 1	=		
Device name	=		
Boot 2	=		
Device name	=		
Boot 3	=		
Device name	=		
Boot 4	=		
Device name	=		
Boot 5	=		
Device name	=		
Boot 6	=		
Device name	=		

Setup Command 5 Selections

Selection	Original	Current	Selection	Original	Current
Non-English			**English**		
0 =			0 =		
1 =			1 =		
2 =			2 =		
3 =			3 =		
4 =			4 =		
5 =			5 =		
6 =			6 =		
7 =			7 =		
8 =			8 =		
9 =			9 =		

Setup Command 6 Selections

Switches			Original	Current
2	3	4		
On	On	On	Special =	
On	On	Off	SB1 =	
On	Off	On	SB2 =	
On	Off	Off	SB3 =	
Off	On	On	SB4 =	
Off	On	Off	SB5 =	
Off	Off	On	SB6 =	
Off	Off	Off	Normal =	

APPENDIX C
MNEMONICS

AMUX – A-multiplexer

APF – active page field

ASCII – American Standard Code for Information Interchange

AST – asynchronous system trap

BCR – boot and diagnostic facility configuration register

BCSR – boot and diagnostic control/status register

BMUX – B-multiplexer

CCR – cache control register

CDP – cache data path

CDR – configuration and display register

CMUX – C-multiplexer

CSM – call supervisor mode

CSR – control/status register

DATBI – data block in

DATI/DATIP – data in/data in pause

DATO/DATOB – data out/data out byte

DCSR – diagnostic control/status register

DLART – digital-link asynchronous receiver/transmitter

DMA – direct memory access

EEPROM – electrically erasable programmable ROM

EIA – Electronic Industries Association

EIS – extended instruction set

EPROM – erasable programmable ROM

ESC – escape

FEA – floating exception address

FEC – floating exception code

FER – floating error

FPA – floating-point accelerator

FPS – floating-point status register

HMR – hit/miss register

IACK – interrupt acknowledge

I/O – input/output

IRQ – interrupt request

KMCR – KTJ11 memory configuration register

KSP – kernel stack pointer

LE – latch enable

LIFO – last in, first out

LRU – least recently used

LSB – least significant bit

LTC – line time clock

MMR – memory management register

MSCP – mass storage control protocol

MSER – memory system error register

MUX – multiplexer

NOP – non-I/O

NPR – nonprocessor request

NXM – nonexistent memory

ODT – on-line debugging technique

OE – output enable

PA – physical address

PAF – page address field

PAR – page address register

PC – program counter

PCR – page control register

PDR – page descriptor register

PIC – position-independent code

PIRQ – program interrupt request

PLF – page length field

PMG – processor mastership grant

PMI – private memory interconnect

PSW – processor status word

RAM – random-access memory

RBUF – receiver buffer

RCSR – receiver control/status register

ROM – read-only memory

SLU – serial line unit

SP – stack pointer

SSP – supervisor stack pointer

UBA – UNIBUS adapter module

USP – user stack pointer

VA – virtual address

XBUF – transmitter buffer

XCSR – transmitter control/status register

INDEX

A

A-multiplexer, 5-30
ABSF/ABSD, 10-11
Absolute addressing mode, 8-17
AC bus loading, 6-21
Accuracy, 10-8
ADC, 9-22
ADD, 9-28
ADDF/ADDD, 10-11
Address decode, 5-32
Addressing modes, 8-1
Advance to next test, 4-25
Alternate boot block, 2-12
ANSI video terminal, 2-9
ASH, 9-29
ASHC, 9-30
ASL, 9-18
ASR, 9-17
Autodecrement mode, 8-9
Autodecrement-deferred, 8-12
Autoincrement mode, 8-8
Autoincrement-deferred, 8-12
Automatic (Mode 1), 4-2
Automatic boot sequence, 4-8
Automatic mode, 2-10

B

B-multiplexer, 5-28
Back-up/restart, 1-23
Bank select address codes, 5-4
Basic transactions, 5-12
Baud rate, 2-4
Baud rate selection, 1-35
BCC, 9-38
BCS, 9-39
BEQ, 9-36
BEVNT L, 6-21
BGT, 9-40
BHALT L, 6-19
BHI, 9-41
BHIS, 9-42
BIC, 9-32
BIS, 9-33
Bit, 9-32
BLE, 9-41
BLO, 9-42
Block data in (DABTI), 1-35
BLOS, 9-41
BLT, 9-40
BMI, 9-37
BNE, 9-36
Boot mode commands, 4-3
Boot program, 4-3
Bootstrap
    error LED display, 4-17
    programs available, 4-12
    translation table, 4-8
BPL, 9-37
BPT, 9-49
BR, 9-35
Break-detected interrupt request, 5-41
Bus arbitrator, 5-10
Bus cycle protocol, 6-4
Bus device NPR, 7-6
Bus distribution, 5-37
Bus master, 6-2
Bus read, 5-7
Bus termination, 6-23
Bus timeout, 1-10
Bus write, 5-8
BVC, 9-38
BVS, 9-38
Byte allocation, 5-18
Byte instructions, 9-6

C

C-multiplexer, 5-28
C/D interface, 1-34
Cache, 1-1
Cache control, 5-26
    logic, 5-30
    register, 1-30
    signals, 5-24
Cache data parity logic, 5-25
Cache data path, 5-30
Cache data RAM, 5-24
Cache memory, 1-27, 5-24
Cache memory (test 62), 4-20
Cache operation with memory
    (test 51), 4-21
Cache parity, 5-25
Cache response, 1-28
CCC, 9-59
CFCC, 10-13
CLC, 9-59
CLN, 9-59
Clock CSR, 2-11
Clock interrupts, force 2-11
Clock select, 2-11
Clock start logic, 5-35
CLR, 9-12
CLRF/CLRD, 10-13
CLV, 9-59
CLZ, 9-59
CMP, 9-27
CMPF/CMPD, 10-13
COM, 9-12
Complete memory data/byte
    exercise (test 50), 4-21
Condition code operators, 9-11
Configuration and display circuits, 5-42
Configuration
    parameters, 2-8
    register, 1-43
    requirements, 2-1
Connectors J2 and J3, 2-3
Console enable, 2-6
Console interrupt arbitration logic,
    5-41
Console message, 4-10
Console ODT, 3-1
Console serial line unit (SLU), 5-38
Console SLU test 1 (test 66), 4-19
Console SLU test 2 (test 65), 4-19
Console SLU test 3 (test 64), 4-19
Contact (pin) identification, 2-18
Control store, 5-18
Control store outputs, 5-20
Control/status register, 1-40
Conversion routines, 11-21
Coroutines, 11-14
CPU and MMU (test 76), 4-18
CPU error register, 1-7
CPU or halt switch (test 77), 4-18
CPU ROM checksum and PCR
    (test 70), 4-19
CSM, 9-53
<CTRL> C command, 4-16
Current transaction, 5-2
Cycle decoder, 5-33
Cycle encoder, 5-12
    status, 5-14

D

D space group, 1-16
DADR bus bits, 5-43
Data in (DATI), 1-34
Data out (DATO), 1-35
Data path controller, 5-12
Data transfer bus cycles, 6-3
DATI bus cycle, 6-5
DATIO (B) bus cycle, 6-10
DATO (B) bus cycle, 6-7
DC bus loading, 6-21
DC350/394
    accesses, 5-18
    gate array, 5-28
DC351 gate array, 5-34
DEC, 9-14
Default, 5-17
Deferred (indirect) addressing, 8-12
Destination operand, 8-3
DEVCD outputs, 5-33
Device addressing, 6-4
Device priority, 6-14
Diagnostic tests, 4-16
    error message, 4-24
Dialog (Mode 0), 4-1
Dialog mode, 2-9
Direct addressing, 8-4
Direct memory access (DMA), 6-12
Disable all testing, 2-12
Disable ROM, 2-11
Disable setup mode, 2-12
Disable UBA ROM, 2-12
Distributed arbitration, 6-14
DIV, 9-31
DIVF/DIVD, 10-14
DMA monitor, 5-18
DMA tag
    data path, 5-35
    store, 5-26
DMA transaction, 6-12
Double-operand addressing, 8-3

E

ECC test, 2-12
EEPROM, 4-1
    Create a boot program, 4-12
    Delete a boot program, 4-12
    Load a boot program, 4-12
    Save a boot program, 4-13
EEPROM checksum (test 71), 4-19
Electrical characteristics, 6-21
EMT, 9-48
Enable 18-bit mode, 2-12
Enable trap-on-halt, 2-12
Enable UBA cache, 2-12
Enable Unibus memory test, 2-12
Enter ROM ODT, 4-14
EPROMs, 4-1
Error message, 4-28
Error number, 4-24
Exit (test 30), 4-23
Exit command, 4-5
Exit standalone mode (test 56), 4-20
Extended LSI-11 bus signals, 6-1
External read/write, 5-17
External transactions, 5-10

F

Floating exception register
    address, 10-6
    code, 10-6
Floating-point accelerator (FPA), 1-2, 5-44
Floating-point accumulators, 10-8
Floating-point data, 10-2
Floating-point exceptions, 10-6
Floating-point instruction (test 60), 4-18
Floating-point instructions, 10-9
Floating-point number, 10-1
Floating-point status register (FPS), 5-44, 10-2
Flush cache, 1-30
Flush counter, 5-35
Forced dialog mode, 4-2
FPA operation, 5-44
FPS register bits, 10-2
Functional blocks, 5-1

G

General purpose read, 5-8
    codes, 5-8
General purpose registers, 1-2
General purpose write, 5-8
    codes, 5-9

H

H9277-A backplane, 2-1
H9278-A backplane, 2-1
Halt, 9-51
Halt-on-break, 2-9
Help command, 4-2
High byte parity (P1), 5-22
High byte parity bit, 5-25
Hit logic, 5-35
Hit/miss register, 1-32

I

I space group, 1-16
IADR bits, 5-42
Ignore battery, 2-10
Immediate mode, 8-16
INC, 9-13
Index bits, 5-22
Index field, 1-27
Index mode, 8-11
Index-deferred, 8-13
Initialization, 5-2, 6-18
Input signals, 5-4
Installation procedure, 2-21
Instruction set list, 9-1
Internal bus control
    network, 5-37
    signals, 5-21
Internal transactions, 5-12
Interrupt-driven techniques, 11-10
Interrupt protocol, 6-15
Interrupts, 1-8, 6-12
Interrupt service routines, 11-10
Interrupt vector, 5-18
Interrupt vector timeout, 1-10
IOT, 9-50

J

JMP, 9-43
JSR, 9-44
Jump and subroutine, 9-10
Jumper wires, 2-1

K

Kernel, 1-2
Kernel protection, 1-3
Kernel stack, 1-3

L

Label bits, 5-22
LDCDF/LDCFD, 10-15
LDCIF/LDCID/LDCLF/LDCLD, 10-16
LDEXP, 10-17
LDF/LDD, 10-18
LDFPS, 10-18
LED display, 4-16
Line clock (test 61), 4-20
Line time clock register, 1-45
List command, 4-4
Logical, 9-9
Long memory test, 2-11
Loop on test, 4-24
Looping techniques, 11-32
Low byte parity (P0), 5-23
Low byte parity bit, 5-22
LSI-11
- bus signals, 2-19
- bus systems, 6-24
- compatible options, 2-13
- control signals, 5-38

LSI-11 based system, 2-13
LSI/Unibus, 5-18

M

Main memory parity error, 1-29, 5-32, 5-35
Maintenance register, 1-44
MAIO coding, 5-2
Map command, 4-15
Map memory and I/O page, 4-25
MARK, 9-51
Master/slave relationship, 6-2
MBS1 H and MBS0 H signals, 5-3
Memory address shorts (test 46), 4-22
Memory location 0 (test 53), 4-21
Memory locations 0 to 4K words (test 52), 4-21
Memory management registers, 1-16
- register 0, 1-20
- register 1, 1-20
- register 2, 1-20
- register 3, 1-22

Memory mapping, 1-11
Memory parity/ECC (test 47), 4-21
Memory refresh, 6-19
Memory sizing routine (test 54), 4-20
Memory system error register, 1-32
Memory system registers, 1-30
MFPD/MFPI, 9-58
MFPS, 9-24
MFPT, 9-57
Microprocessor, 5-2
Miscellaneous CPU and EIS (test 67), 4-19
MMU aborts (test 63), 4-19
Mode 24, 2-10
24/26 (Mode 3), 4-2
Modes, 1-2
MODF/MODD, 10-19
MOV, 9-26
MSV11-J memory modules, 2-1
MTPD/MTPI, 9-58
MTPS, 9-25
MUL, 9-31
MULF/MULD, 10-22
Multiple faults, 1-23
Multiple-backplane systems, 6-24
Multiple-precision, 9-8

N

NEG, 9-14
NEGF/NEGD, 10-23
Nesting, 11-11
Next address multiplexer, 5-17
No SACK timeout, 1-10
NOP, 5-7
ODT
- commands, 3-3
- entry conditions, 3-2
- mode, 2-10
- timeout, 3-8

ODT (Mode 2), 4-2
Optional commands, 4-24
Oscillator, 5-15
- control signals, 5-16

Output messages, 4-29

P

Page address register, 1-18
Page control register, 1-42
Page descriptor register, 1-18
Parity error, 1-29
- detection, 1-29

Parity generator, 5-30
Parity interrupt and abort, 5-31
Physical address, 1-14
PMG count, 2-10
PMI bus control signals, 5-38
PMI bus master signals, 7-1
- Block Mode, 7-1
- Bus Cycle, 7-1
- PMI Byte, 7-1
- PMI Write Strobe, 7-1

PMI cycle request, 5-12
PMI data transfers, 7-9
    block data in, 7-11
    data in/data in pause, 7-9
    data out/data out byte, 7-13
PMI interrupt protocol, 7-15
PMI operation, 7-4
    bus device interrupt, 7-6
PMI power-up/down, 7-15
PMI signal assignments, 2-18
PMI slave signals, 7-3
    PMI High Byte Data Parity, 7-3
    PMI Low Byte Data Parity, 7-3
    PMI Read Strobe, 7-3
    PMI Slave Buffer Full, 7-3
    PMI Slave Selected, 7-3
PMI turned on and check UBA reboot bit (test 74), 4-18
PMI Unibus adapter signals, 7-4
    PMI Busy, 7-4
    PMI Unibus Map Enable, 7-4
    PMI Unibus Memory, 7-4
    PMI Unibus System, 7-4
    PMI Unibus Timeout, 7-4
Popping from a stack, 11-6
Position-defined arbitration, 6-14
Position-dependent code, 11-3
Position-independent code, 8-16, 11-1
Power status protocol, 6-18
Power-up modes, 2-9
Power-up to Mode 3: 24 Mode (test 72), 4-18
Power-up to Mode 2: ODT (test 73), 4-18
Power-up/power-down, 6-19
Predicted parity, 5-30
Primary bootstrap programs, 4-26
Primary control system, 5-1
Private memory interconnect (PMI), 1-34, 7-1
Processor status word (PSW), 1-4
Processor traps, 11-19
Program control, 9-7
Program counter, 1-4
Program interrupt request register, 1-7
Programming techniques, 11-1
Protection modes, 1-1
PSW operators, 9-8
Pushing onto a stack, 11-5

Q

Qualifiers, 4-3

R

Read interrupt vector, 5-10
Receiver buffer register, 1-37
Receiver control/status register, 1-36
Recursion, 11-17
Reentrancy, 11-12
Reentrant code, 11-12
Register mode, 8-6
Register-deferred, 8-13
Relative addressing mode, 8-18
Relative-deferred addressing mode, 8-19
Rerun test, 4-24
Reset, 9-56
Resident boot programs, 4-5
Restart, 2-10
Restricted LSI-11 options, 2-15
ROL, 9-20
ROM code, 4-1
    disable, 2-11
ROM ODT commands, 4-15
ROR, 9-19
RTI, 9-50
RTS, 9-46
RTT, 9-51

S

SBC, 9-23
SCC, 9-59
SEC, 9-59
Secondary functional blocks, 5-2
Select configuration parameters (2), 4-6
Selection of NA <1:0> status, 5-17
SEN, 9-59
Serial line interface, 1-35
Serial line unit, 1-2
SETD, 10-23
SETF, 10-23
SETI, 10-24
SETL, 10-24
Setup mode, 4-5
    disable, 2-12
Setup mode command 2, 2-7
Setup table, 4-12
SEV, 9-59
SEZ, 9-59
Shift and rotate, 9-8
Signed conditional branch, 9-9
Single-backplane systems, 6-25
Single-operand addressing, 8-3
SOB, 9-47
Source operand, 8-3

SPL, 9-53
Stack, 11-5
    uses, 11-8
Stack pointer, 1-4
Stale data, 1-32
Standalone mode, 5-18
Standard factory configuration, 2-2
Status LED display, 2-7
Status registers, clearing, 1-22
STCFD/STCDF, 10-25
STCFI/STCFL/STCDI/STCDL, 10-26
STEXP, 10-27
STF/STD, 10-27
STFPS, 10-28
Stretched bus read, 5-7
STST, 10-28
SUB, 9-28
SUBF/SUBD, 10-29
Subroutine advantages, 11-10
Sunset loop, 1-8
Supervisor, 1-2
Supervisory program, 1-10
SWAB, 9-21
Switchpack, 2-2
Switchpack boot selections, 4-11
SXT, 9-24

T

Tag comparator, 5-30
Tag field, 1-27
Tag parity bit (P2), 5-22
Tag RAM, 5-23
Tag store data, 5-22
Tag valid bit (V), 5-22
Time-multiplexed data/address bus, 5-6
Timeout logic, 5-36
Transactions, 5-3
Transmitter buffer register, 1-39
Transmitter control/status register, 1-38
Trap, 9-49
Trap and interrupt, 9-10
Trap-on-halt, enable, 2-12
TST, 9-15
TSTF/TSTD, 10-30
TSTSET, 9-16
Turn on MMU, run CPU and MMU (test 75), 4-18
Typical displays, 4-25
Typical usage, 1-22

U

UBA address overflow (test 40), 4-22
UBA boot ROM (test 45), 4-21
UBA cache data (test 37), 4-23
UBA cache, enable, 2-12
UBA cache LRU (test 36), 4-23
UBA cache parity error (test 34), 4-23
UBA cache tag store (test 35), 4-23
UBA floating address/data (test 41), 4-22
UBA map registers data path (test 44), 4-21
UBA mapped diagnostic data (test 42), 4-22
UBA register response (test 55), 4-20
UBA unmapped diagnostic data (test 43), 4-22
Unibus adapter module (UBA), 1-34
    signals, 7-2
    compatible options, 2-16
Unibus map enable, 5-34
Unibus memory address shorts (test 31), 4-23
Unibus memory data/byte exercise (test 33), 4-23
Unibus memory parity (test 32), 4-23
Unibus system, 2-16
Unsigned conditional branch, 9-10
User, 1-2
User friendly
    format, 2-9
    mode, 4-28
User program, 1-11

V

Valid tag bits, 5-26
Version V6.0, 4-1, A-1
Version V7.0, 4-1, A-1, A-11
Version V8.0, 4-1, A-1, A-11
Virtual address, 1-14

W

WAIT, 9-56
WRTLCK, 9-16
XOR, 9-34

READER'S COMMENTS

Your comments and suggestions will help us in our efforts to improve the quality and usefulness of our publications.

1. **Which of the following most closely describes your job?** **1** ⊂a⊃ ⊂b⊃ ⊂c⊃ ⊂d⊃ ⊂e⊃ ⊂f⊃ ⊂g⊃ ⊂h⊃ ⊂i⊃

   (a) Administrative support (d) Scientist/Engineer (g) Educator/Trainer
   (b) Programmer/Analyst (e) Systems Manager (h) Computer Operator
   (c) Software support (f) Sales (i) Other _____

2. **How many years of experience do you have with computers?** **2** ⊂a⊃ ⊂b⊃ ⊂c⊃ ⊂d⊃ ⊂e⊃

   (a) Less than 1 (b) 1 to 3 (c) 4 to 6 (d) 7 to 9 (e) 10 or more

3. **What did you like *most* about this manual?**

4. **What did you like *least* about this manual?**

5. **How do you rate this manual?**

   Indicate your opinion of the quality of the manual. For each aspect of quality, darken your response on the five-point scale, where (1) = POOR and (5) = EXCELLENT

(a)	Accuracy	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(b)	Completeness	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(c)	Usefulness of Examples/Figures	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(d)	Clearness of Language	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(e)	Helpfulness of Index/Table of Contents	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(f)	Consistency in Presenting Information	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(g)	Logical Organization	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(h)	Visual Appeal	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(i)	Relevance of Information	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(j)	Ease of Learning	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(k)	Ease of Use	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(l)	YOUR OVERALL IMPRESSION	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(m)	Quality Relative to Other Digital Manuals	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(n)	Quality Relative to Other Companies' Manuals	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃

6. **List any errors you found in the manual. (Reference page, table, or figure numbers.)**

7. **Do you have any additional comments?**

Name _____ Company _____

Title _____ Department _____

Street _____ City _____ State/Country _____ Zip _____

Telephone No. _____ Date _____

FOLD HERE AND TAPE

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Educational Services/Quality Assurance
12 Crosby Drive BUO/E08
Bedford, MA 01730

FOLD HERE

Digital Equipment Corporation • Marlboro, MA 01752