KDJ11-E CPU Module

digital

User Guide

Order Number: EK-KDJ1E-UG-001

KDJ11-E CPU Module User Guide

Order Number EK-KDJ1E-UG-001

Revision/Update Information: This is a new manual.

Digital Equipment Corporation

January 1991

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

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013.

Copyright © Digital Equipment Corporation 1991

All Rights Reserved.
Printed in U.S.A.

The following are trademarks of Digital Equipment Corporation: DECsystem, PDP, PDP–11, PDP–11/03, PDP–11/05, PDP–11/10, PDP–11/15, PDP–11/20, PDP–11/23, PDP–11/23+, PDP–11/24, PDP–11/35, PDP–11/40, PDP–11/44, PDP–11/45, PDP–11/50, PDP–11/53, PDP–11/60, PDP–11/70, PDP–11/73, PDP–11/83, PDP–11/84, PDP–11/93, PDP–11/94, Q-bus, Q22-bus, RSX–11, RSX–11M, RT–11, UNIBUS, VT100, VT220, VT330, and the DIGITAL logo.

This document was prepared and published by Educational Services Development and Publishing, Digital Equipment Corporation.

Contents

2 Configuration

3 Console On-Line Debugging Technique (ODT)

4 Boot ROMs and Diagnostics

5 Extended LSI-11 Bus

8 Base Instruction Set

9 Floating-Point Arithmetic

10 Programming Techniques

A Setup Parameters Worksheet

Index

Examples

Figures

Tables

About This Manual

This user guide contains descriptions of the KDJ11-E CPU module architecture, configuration, system requirements, and programming.

Audience

This manual is intended for Digital Customer Services, self-maintenance, and Original Equipment Manufacturing (OEM) personnel.

Organization

This manual contains ten chapters, an appendix, and an index.

- Chapter 1, Architecture, describes the KDJ11-E CPU module architecture.
- Chapter 2, Configuration, describes the configuration requirements when configuring and installing the KDJ11-E CPU module into an LSI-11 based system.
- Chapter 3, Console On-Line Debugging Technique (ODT), describes ODT.
- Chapter 4, Boot ROMs and Diagnostics, describes diagnostics that test the CPU, UBA and memory; the chapter also includes hard copy ROM commands.
- Chapter 5, Extended LSI-11 Bus, describes the LSI-11 bus.
- Chapter 6, Private Memory Interconnect Bus, describes PMI protocol, and PMI bus signals.
- Chapter 7, Addressing Modes, describes the addressing modes.
- Chapter 8, Base Instruction Set, describes the KDJ11-E instruction set.
- Chapter 9, Floating-Point Arithmetic, describes floating-point instructions.
- Chapter 10, Programming Techniques, describes utilizing the combination of the instruction set, addressing mode and programming techniques to develop new software or to utilize old programs effectively.
- Appendix A, Setup Parameters Worksheet, contains two worksheets for each mode (video terminal or hardcopy) to record the original setup parameter selections and the new setup parameters selections contained in the EEPROM of the KDJ11-E module.

Conventions

The following conventions are used in this manual:

Convention	Meaning
Caution	Contains information to prevent damage to equipment

Convention	Meaning
Note	Contains general information
<x:y>	Represents a bit field, a set of lines, or a set of signals, ranging from x through y. For example, R0 <7:4> indicates bits 7 through 4 in a general-purpose register R0.
R/W	Read/Write
R	Read
RO	Read-only

1
Architecture

1.1 KDJ11-E CPU Module Description

The KDJ11-E is a quad-height processor module for LSI-11 bus systems. It is designed for use in high-speed, real-time applications, and multiuser, multitasking environments. The module can also operate as a CPU in PDP–11 UNIBUS systems, when 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 using 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-E module.

The KDJ11-E 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. Each section can also 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.

Features

The KDJ11-E system module is a high performance CPU module with the following features:

- DCJ11 microprocessor
- Floating-point accelerator (FPJ11)
- 22-bit memory management
- Programmable line frequency clock
- Console serial line unit

- Seven serial line units (SLUs)
- Boot and diagnostic ROMs
- Time of Year (TOY) clock
- Two or four Mbytes onboard parity memory
- Console programmable setup features
- Comprehensive self-test capability
- LED status indicator

The boot and diagnostic ROM features include:

- Booting of the user's software on various devices
- Memory size display
- Time and date of TOY clock
- Boot device selection
- Ability to define parameters for SLUs
- Self-test selection
- User boot area on EEPROM
- Support for hard copy terminals and video display terminals

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

Figure 1–1 shows the user-addressable registers, classified as general purpose, system control, memory system, floating-point and Memory Management Registers (MMRs).

The KDJ11-E 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.

1.2 DCJ11-A Microprocessor 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 provides complete executive protection in a multiprogram environment.

There are 16 general purpose registers (Table 1–1), but only 8 are addressable 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 registers, autodecrement registers, and stack pointers.

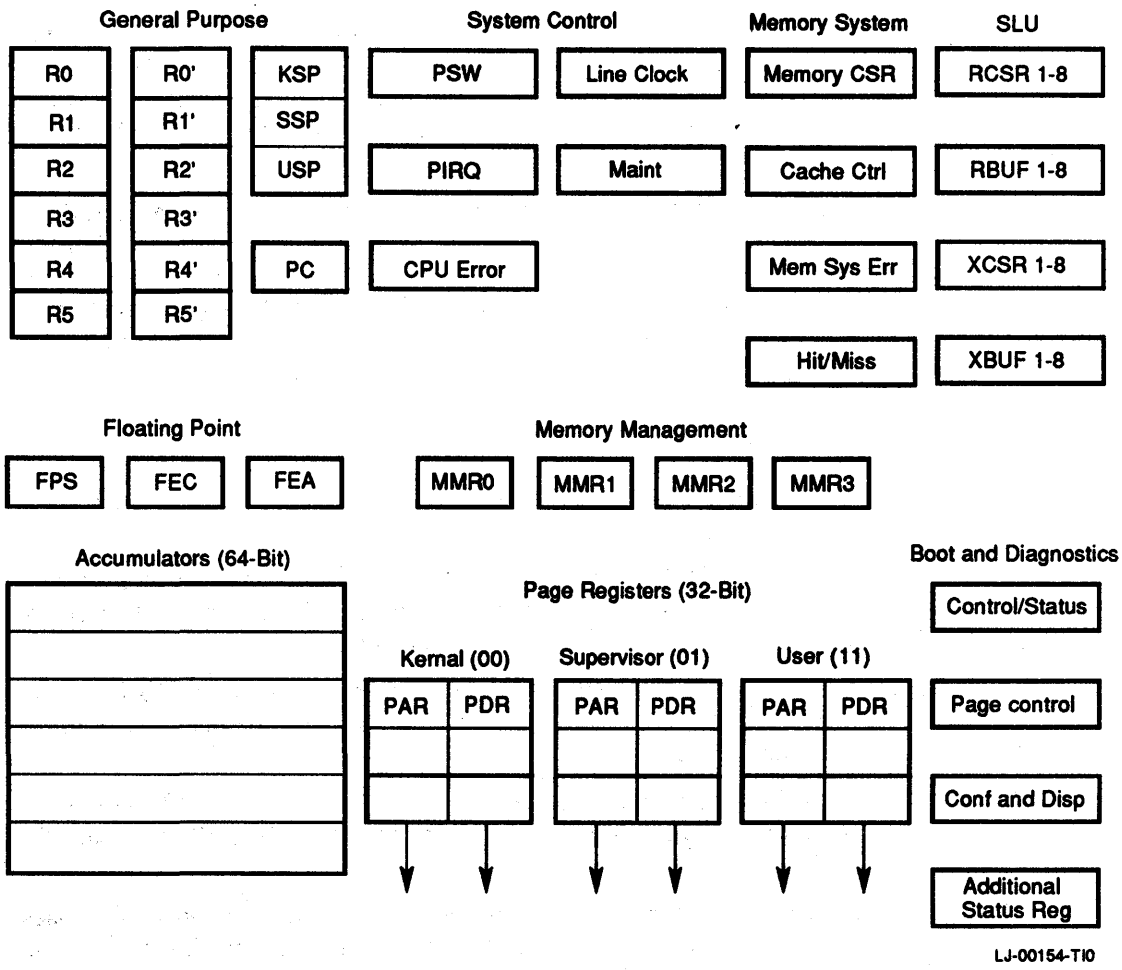


Figure 1–1 Programming Model

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), program interrupt request (PIRQ), and 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 kernel mode, the HALT, RESET, and SPL instructions are executed as specified. In supervisor or user mode, the HALT instruction causes a trap through location 4, but the RESET and SPL instructions are treated as NOPs.
- In kernel mode, the RTI and RTT instructions can freely change bits <15:11> and <7:5> of the PSW register. In supervisor or user mode, these instructions can change only bits <15:11> of the PSW register.
- In kernel mode, the MTPS instruction can change bits <7:5> of the PSW register. In supervisor or user mode, 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 what the contents of the PSW are at the time the interrupt or trap occurs.
- Kernel stack references are checked for stack overflow, but 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. 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.

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)

1.2.5 Program Counter

The program counter (PC) contains the 16-bit address of the next instruction 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 normally it is not used as an accumulator.

1.2.6 Processor Status Word (17777776)

The processor status word (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. Figure 1–2 shows the PSW register. Table 1–3 describes the PSW bits.

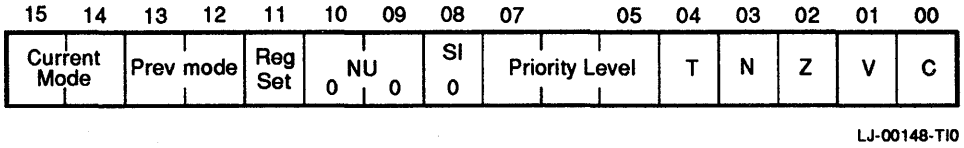


Figure 1–2 Processor Status Word Register

Table 1–3 Processor Status Word Bit Description

Bits	Name	Status	Function
<15:14>	Current mode	R/W	Indicates the current operating mode and is coded as follows:

Bit 15	14	Mode
0	0	Kernel
0	1	Supervisor
1	0	Illegal
1	1	User

Table 1–3 (Cont.) Processor Status Word Bit Description

Bits	Name	Status	Function
<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>	Unused	R	Read as 0s.
8	Suspended information	R/W	Reserved.
<7:5>	Priority	R/W	Indicates the current priority level of the processor and is coded as follows:

Bit 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[1]	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 0.
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.

[1]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 (17777766)

The error register, at address 17777766, identifies the source of any abort or trap that caused a trap through location 4. The CPU error register is cleared when it is written. It is also cleared at power-up or by console start. The CPU error register is unaffected by a RESET instruction. Figure 1–3 shows the register format. Table 1–4 provides the CPU error register bit descriptions.

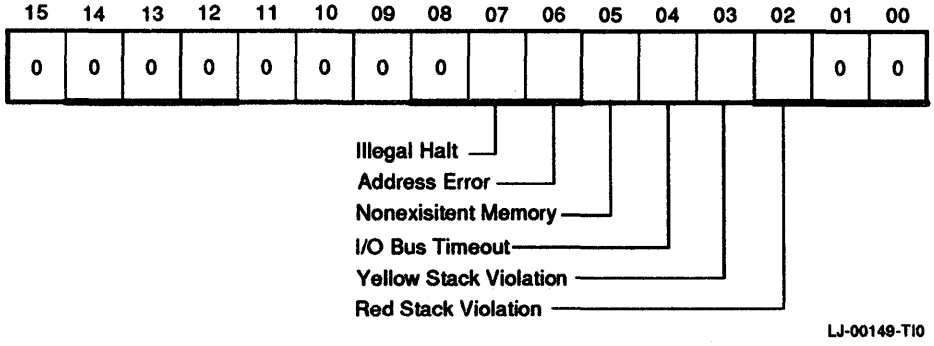


Figure 1–3 CPU Error Register

Table 1–4 CPU Error Register Bit Descriptions

Bits	Name	Status	Function
7	Illegal HALT	RO	Set when execution of a HALT instruction is attempted in user or supervisor mode.
6	Address Error	RO	Set when word access to an odd byte address or an instruction fetch from an internal register is attempted.
5	Nonexistent Memory	RO	Set when a reference to main memory times out.
4	I/O Bus Timeout	RO	Set when a reference to the I/O page times out.
3	Yellow Stack Violation	RO	Set on a yellow zone stack overflow trap.
2	Red Stack Violation	RO	Set on a red zone stack overflow trap.

1.2.8 Program Interrupt Request Register (17777772)

The program interrupt request register (PIRQ), at location 17777772, implements a software interrupt facility. When a program interrupt request is granted, the processor traps through location 240. It is the responsibility of the interrupt service routines to clear the appropriate bit in PIRQ before exiting. PIRQ is cleared at power-up by a console start and by the RESET instruction. Figure 1–4 shows the register. Table 1–5 provides the program interrupt register bit descriptions.

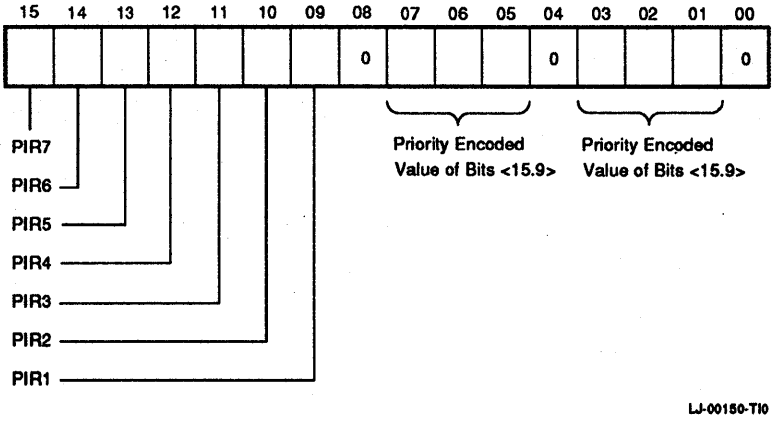


Figure 1–4 Program Interrupt Request Register (17777772)

Table 1–5 Program Interrupt Register Bit Descriptions

Bits	Name	Function
<15:09>	PIR 7–1	Each bit, when set, provides one of seven levels of software interrupt, corresponding to interrupt priority levels 7 through 1.
08	Unused	Read as 0.
<07:05>	Priority encoded value of bits <15:09>	These three bits are set by the CPU to the encoded value of the highest pending interrupt request (bits <15:09>).
04	Unused	Read as 0.
<03:01>	Priority encoded value of bits <15:09>	The function of these bits is identical to bits <07:05>.

1.3 Interrupts

The KDJ11-E module uses a variety of trap, hardware, and software interrupts. Table 1–6 lists the KDJ11-E interrupts in order of their priority. 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-E to temporarily suspend the execution of the current program, and service the device or condition that caused the interrupt or trap. The KDJ11-E has eight levels of interrupt priority and the current priority level is defined by bits <07:05> 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 independently of the processor priority. These nonmaskable interrupts have their own priority structure (Table 1–6).

Table 1–6 KDJ11-E Interrupts

Interrupt	Internal /External	Vector Address	Priority Level
Red stack trap (CPU error register, bit 2)	Internal	4	NM[1]
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)	Internal	114	NM
Trace (T-bit) trap (PSW, bit 4)	Internal	14	NM
Yellow stack trap (CPU error register, bit 3)	Internal	4	NM
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
SLU 1, receive	Internal	300/400	4
SLU 2, receive	Internal	310/410	4
SLU 3, receive	Internal	320/420	4
SLU 4, receive	Internal	330/430	4
SLU 5, receive	Internal	340/440	4
SLU 6, receive	Internal	350/450	4
SLU 7, receive	Internal	360/460	4
SLU 8, receive(Console)	Internal	60	4
SLU 1, transmit	Internal	304/404	4
SLU 2, transmit	Internal	314/414	4
SLU 3, transmit	Internal	324/424	4

[1]NM = Nonmaskable

Table 1–6 (Cont.) KDJ11-E Interrupts

Interrupt	Internal /External	Vector Address	Priority Level
SLU 4, transmit	Internal	334/434	4
SLU 5, transmit	Internal	344/444	4
SLU 6, transmit	Internal	354/454	4
SLU 7, transmit	Internal	364/464	4
SLU 8, transmit(Console)	Internal	64	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)[2]	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.		

[2]The halt line usually has the lowest priority. However, it has the 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.

1.3.1 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.

1.3.2 Red Stack Aborts

A red stack abort happens when an abort sequence 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 ensures 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 12.8 μseconds after the KDJ11-E 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-E traps through virtual address 4 of the kernel space. In a UNIBUS system, the KDJ11-E 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 12.8 μseconds after the KDJ11-E acknowledges an interrupt by asserting the BIAK L bus signal. The timeout is ignored by the KDJ11-E and it continues as if the interrupt request did not occur. In a UNIBUS system, the KDJ11-E 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 (selection acknowledge) timeout occurs when the BSACK L bus signal is not asserted within 12.8 μseconds after the KDJ11-E grants a DMA request by asserting BDMG L. The timeout is ignored by the KDJ11-E and it continues as if the DMA request did not occur.

1.4 Memory Management

KDJ11-E memory management provides the hardware for complete memory management and protection. It is designed to be a memory management facility for accessing all of the 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 supervisory program includes the following tasks:

- 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 controlling 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 of the system executive program.

The following are the basic characteristics of KDJ11-E 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 has 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 17760000 to 17777777.

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 in the upper 4K block (Figure 1–5).

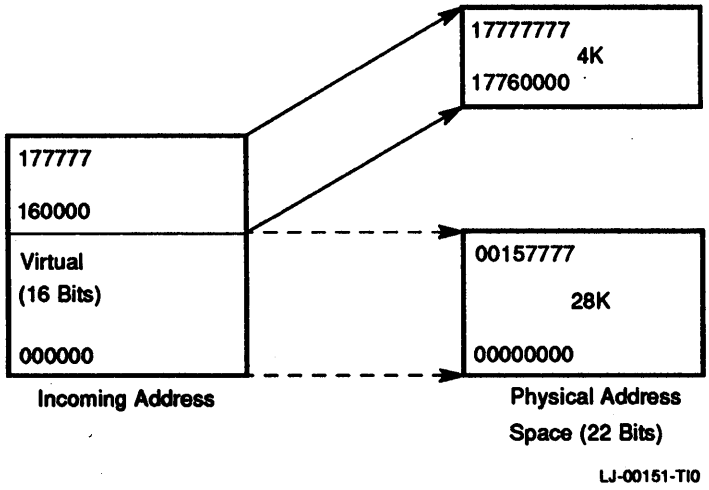


Figure 1–5 16-Bit Mapping

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 seen 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 (Figure 1–7).

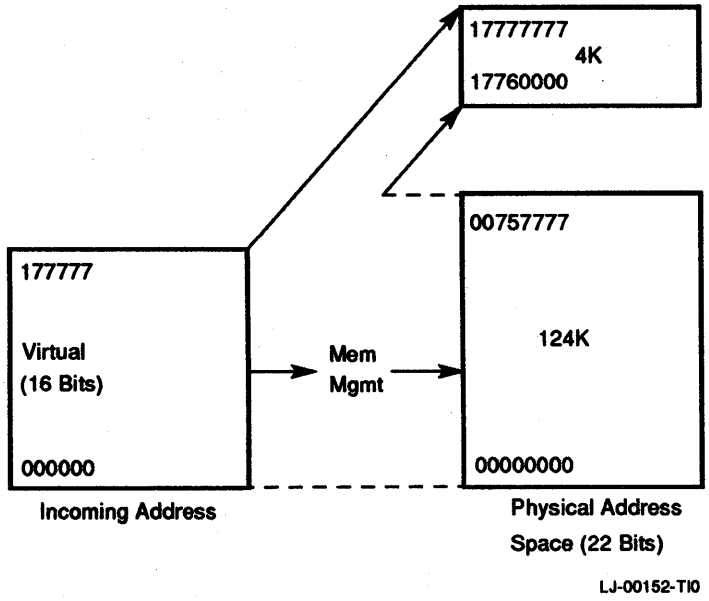


Figure 1–6 18-Bit Mapping

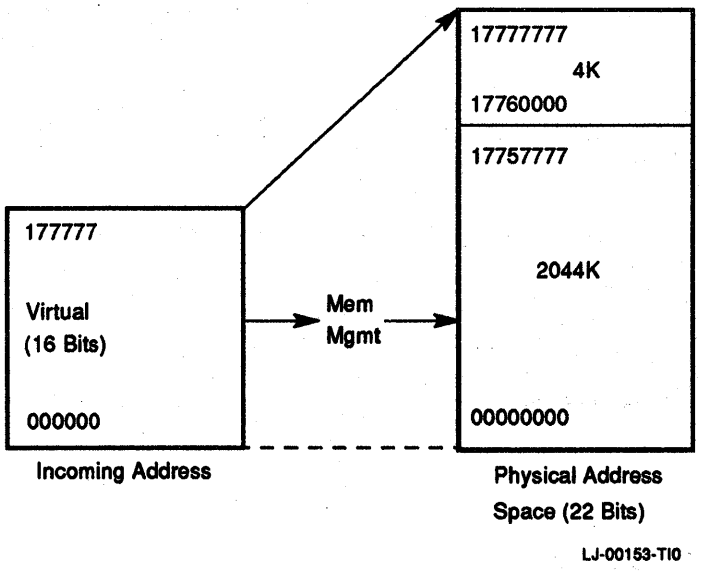


Figure 1–7 22-Bit Mapping

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-E (Table 1–7).

Table 1–7 KDJ11-E 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, -11/60, -11/23
22-Bit	On	PDP–11/94, -11/84, -11/70, -11/44, -11/24, PDP–11/93, -11/83, -11/73, -11/53, -11/23+

1.4.2.1 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 (Figure 1–8). Using memory management, memory can be dynamically allocated in pages, each composed of 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 (kernel, supervisor, or user mode) and by whether the reference is for instructions or data.

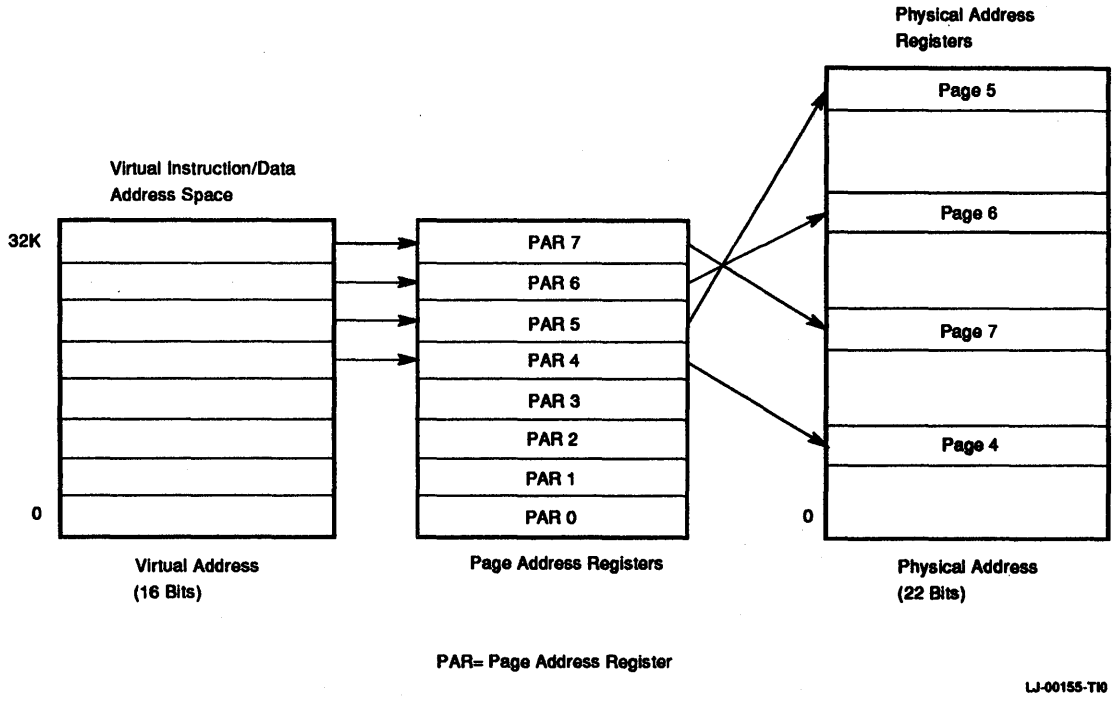


Figure 1–8 Virtual Address Mapping Into Physical Address

1.4.3 Interrupts Under Memory Management

Memory management relocates all addresses. When it is enabled, all traps, aborts, and interrupt vectors are mapped using the data-space mapping registers in kernel mode. 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 the data-space registers in kernel mode.

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.

1.4.3.1 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 (Figure 1–9), and the appropriate PAR set.

The virtual address (VA) consists of the following fields:

- 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 (Figure 1–10).

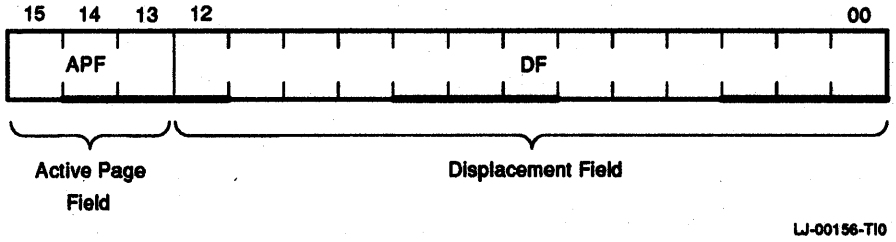


Figure 1–9 Interpretation of a Virtual Address

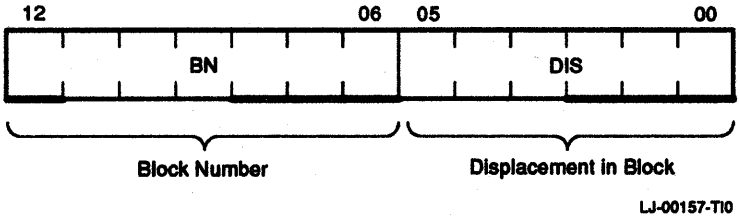


Figure 1–10 Displacement Field of a Virtual Address

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, if PAF is 3 it indicates a starting address of 96 (3 × 32 words in physical memory).

Figure 1–11 illustrates the construction of the physical address (PA). The logical sequence involved in constructing a PA is as follows:

1. The APF of the VA selects one of eight page address registers (PAR0–PAR7) from the appropriate set. The set used depends on the space being referenced and the protection mode being used.
2. The PAF of the selected PAR contains the starting address of the currently active page as a block number in physical memory.
3. 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.

4. 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.

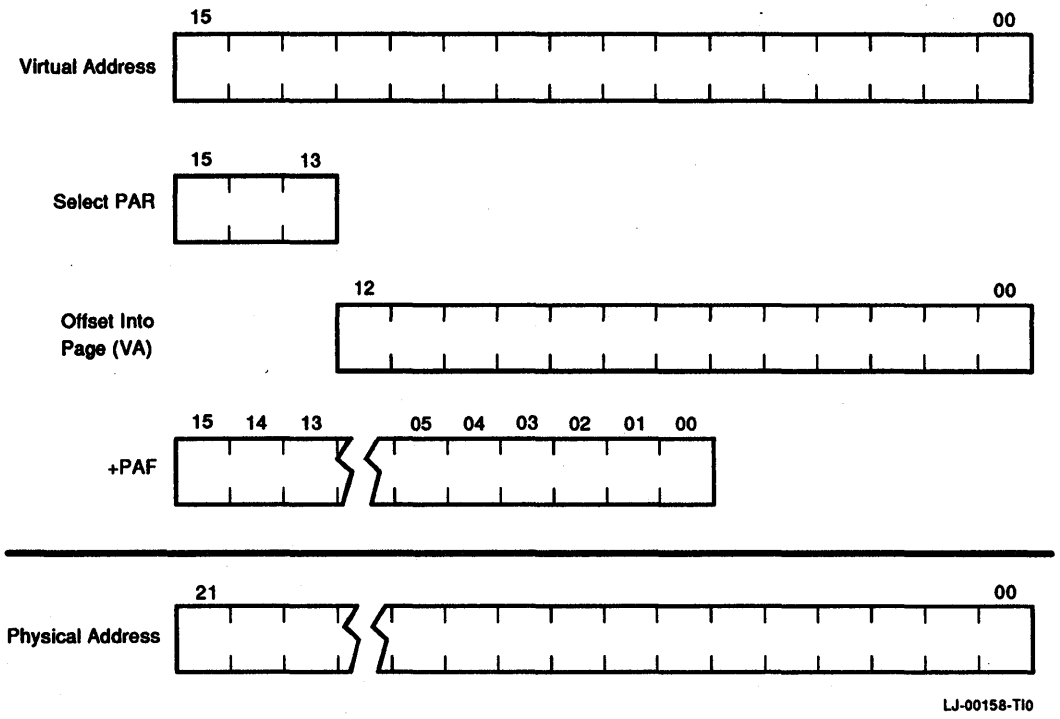


Figure 1–11 Construction of a Physical Address

1.4.4 Memory Management Registers

Memory management implements three sets of thirty-two 16-bit registers (Figure 1–12). One set of registers is used in kernel mode, another in supervisor mode, and the other 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 is described in the following section. 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 in the uppermost 8 Kbytes of physical address space, which is designated as the I/O page. Table 1–8 lists the addresses allocated to the MMRs.

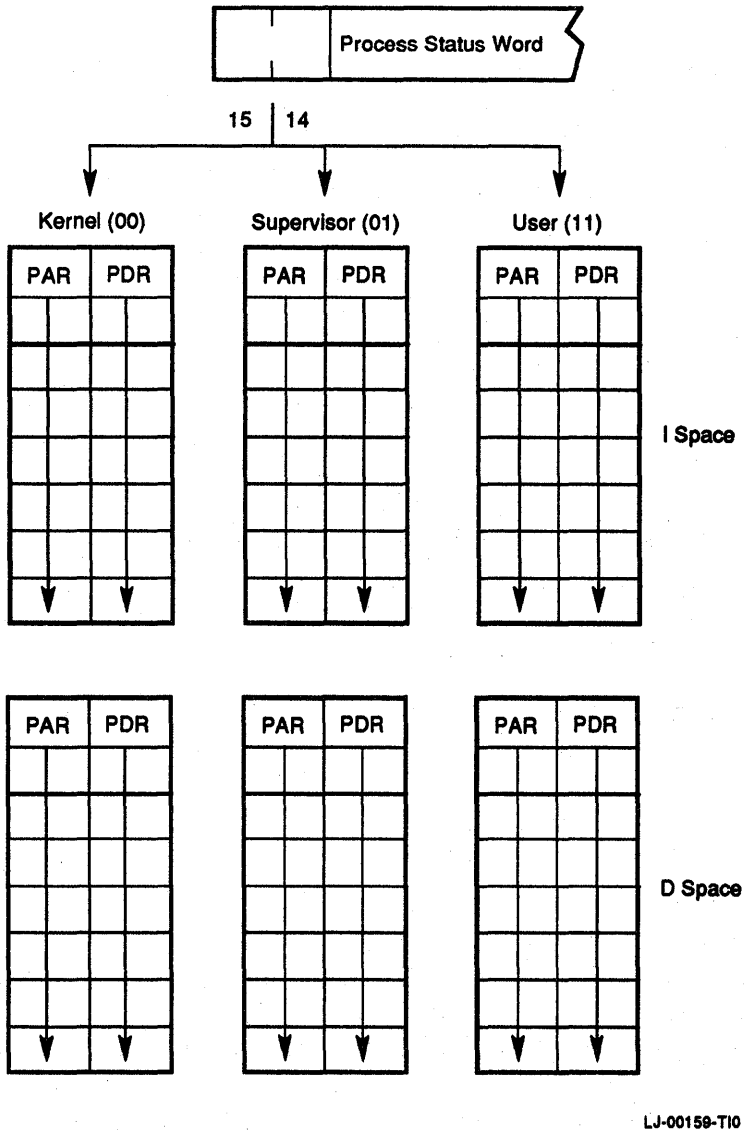


Figure 1–12 Active Page Register

Table 1–8 Memory Management Register Address

Register	Address
Memory management register 0 (MMR0)	17777572
Memory management register 1 (MMR1)	17777574
Memory management register 2 (MMR2)	17777576
Memory management register 3 (MMR3)	17772516
User I space descriptor register (UISDR0)	17777600
.	.

Table 1–8 (Cont.) Memory Management Register Address

Register	Address
.	.
.	.
User I space descriptor register (UISDR7)	17777616
User D space descriptor register (UDSDR0)	17777620
.	.
.	.
.	.
User D space descriptor register (UDSDR7)	17777636
User I space address register (UISAR0)	17777640
.	.
.	.
.	.
User I space address register (UISAR7)	17777656
User D space address register (UDSAR0)	17777660
.	.
.	.
.	.
User D space address register (UDSAR7)	17777676
Supervisor I space descriptor register (SISDR0)	17772200
.	.
.	.
.	.
Supervisor I space descriptor register (SISDR7)	17772216
Supervisor D space descriptor register (SDSDR0)	17772220
.	.
.	.
.	.
Supervisor D space descriptor register (SDSDR7)	17772236
Supervisor I space address register (SISAR0)	17772240
.	.
.	.
.	.
Supervisor I space address register (SISAR7)	17772256
Supervisor D space address register (SDSAR0)	17772260
.	.
.	.

Table 1–8 (Cont.) Memory Management Register Address

Register	Address
.	.
Supervisor D space address register (SDSAR7)	17772276
Kernel I space descriptor register (KISDR0)	17772300
.	.
.	.
.	.
Kernel I space descriptor register (KISDR7)	17772316
Kernel D space descriptor register (KDSDR0)	17772320
.	.
.	.
.	.
Kernel D space descriptor register (KDSDR7)	17772336
Kernel I space address register (KISAR0)	17772340
.	.
.	.
.	.
Kernel I space address register (KISAR7)	17772356
Kernel D space address register (KDSAR0)	17772360
.	.
.	.
.	.
Kernel D space address register (KDSAR7)	17772376

1.4.4.1 Page Address Register (PAR)

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.

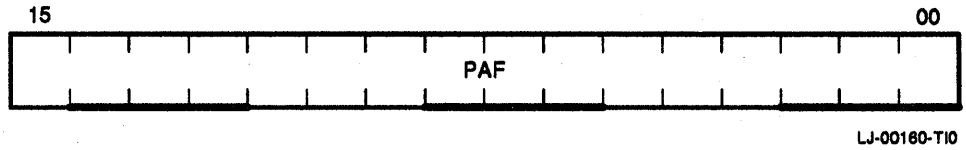


Figure 1–13 Page Address Register

1.4.4.2 Page Descriptor Register (PDR)

The PDR contains information relative to page expansion, page length, and access control. Figure 1–14 shows the page descriptor register. Table 1–9 describes the bits.

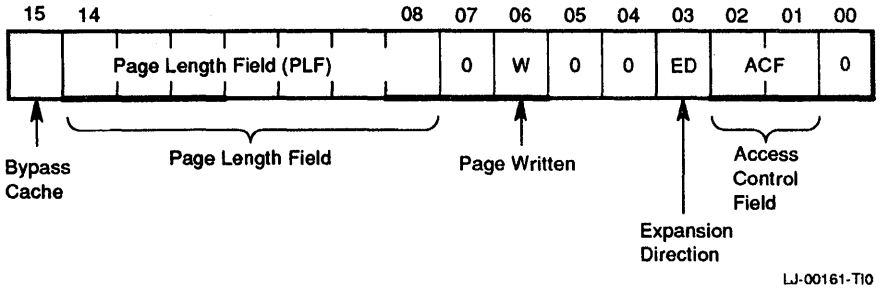


Figure 1–14 Page Descriptor Register

Table 1–9 Page Descriptor Register Bit Description

Bit	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	Unused	RO	Read as 0.
6	Page written	RO	The written into bit (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>	Unused	RO	Read as 0.
3	Expansion direction	R/W	This bit specifies the direction in which the page expands. If it equals 0, the page expands upward from block number 0 to include blocks with higher addresses; if it equals 1, the page expands downward from block number 127 to include blocks with lower addresses.

Table 1–9 (Cont.) Page Descriptor Register Bit Description

Bit	Name	Status	Function
<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	Unused	RO	Read as 0.

1.4.5 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.

1.4.5.1 Memory Management Register 0 (17777572)

MMR0 provides control and records memory management unit status. The register contains abort and status flags (Figure 1–15). Table 1–10 describes the bits.

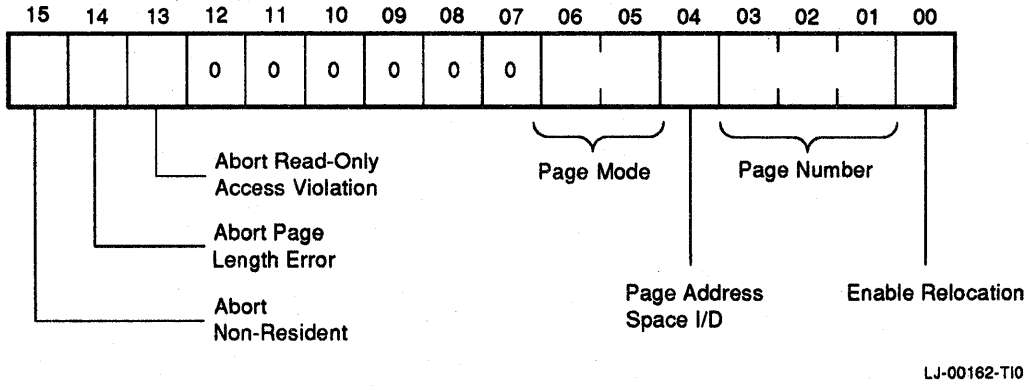


Figure 1–15 Memory Management Register 0 (MMR0)

Table 1–10 MMR0 Bit Description

Bit	Name	Status	Function
15[1]	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[1]	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[1]	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 0s.
<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.

[1]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.

1.4.6 Memory Management Register 1 (17777574)

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.

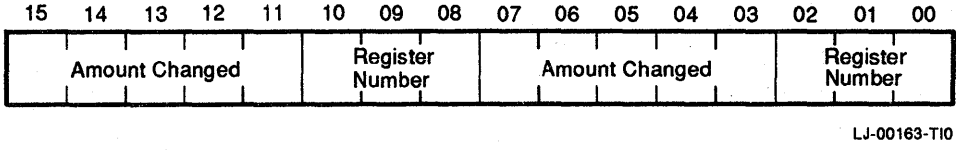


Figure 1–16 Memory Management Register 1 (MMR1)

1.4.6.1 Memory Management Register 2 (17777576)

MMR2 is loaded with the program counter of the current instruction and is frozen when any abort condition is posted in MMR0.

1.4.6.2 Memory Management Register 3 (17772516)

Memory management register 3 (MMR3) at address 17772516 enables or disables D space, 22-bit mapping, the Call Supervisor Mode (CSM) instruction, and the I/O map (when applicable). MMR3 is cleared at power-up by a console start and by a RESET instruction. Figure 1–17 shows the register format. Table 1–11 provides memory management register 3 bit descriptions.

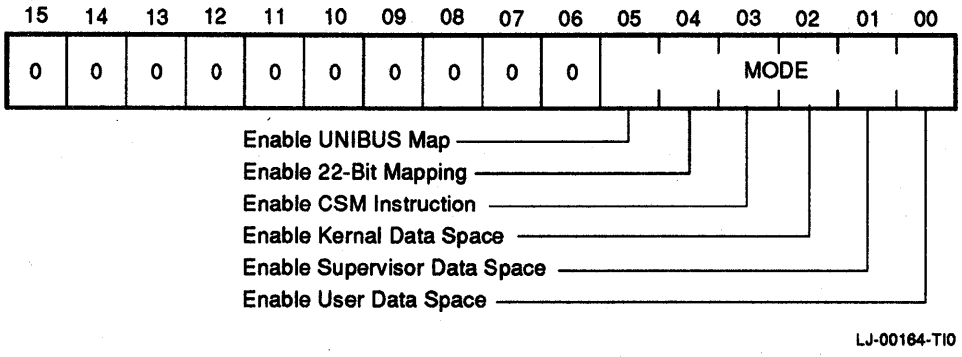


Figure 1–17 Memory Management Register 3 Format (17772516)

Table 1–11 Memory Management Register 3 Bit Descriptions

Bits	Name	Status	Function
<15:06>	Unused		Read as 0s.
05	Enable UNIBUS Map	R/W	This bit enables the I/O map for the UNIBUS adapter (UNIBUS systems only).
04	Enable 22-bit mapping	R/W	This bit, when set, selects 22-bit memory addressing. When this bit is clear, 18-bit addressing is selected. (Only when MMR0 bit <0> is set is 18- or 22-bit addressing actually enabled.)
03	Enable CSM instruction	R/W	This bit enables recognition of the Call Supervisor Mode (CSM) instruction.
<2:0>	Enable data space	R/W	These three bits enable data space mapping for kernel, supervisor, and user mode, respectively.

1.4.6.3 Instruction Back-Up and Restart Recovery

The process of backing up and restarting a partially completed instruction involves the following steps:

1. Performing the appropriate memory management tasks to alleviate the cause of the abort (for example, 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 and 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.

1.4.6.4 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.

1.4.6.5 Multiple Faults

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

1.4.7 Common Usage Examples

The memory management 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 versatility offered by the memory management means that both single-user and multiprogramming systems have complete freedom to make whatever memory management decisions best suit their individual needs. Though there are more common methods of using the memory management, there is no limit to the ways to use these facilities can be used.

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.

1.4.7.1 Typical Memory Page

When the memory management is enabled, 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 1 to 128 blocks and is pointed to by the PAF of the corresponding PAR (Figure 1–18).

The memory segment 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 (that is, written) in the page
- Read-only protection
- Upward expansion

These attributes were determined according to the following scheme:

1. The page address register (PAR6) and page descriptor register (PDR6) were selected by the APF of the VA. (Bits <15:13> of the VA = 6.)

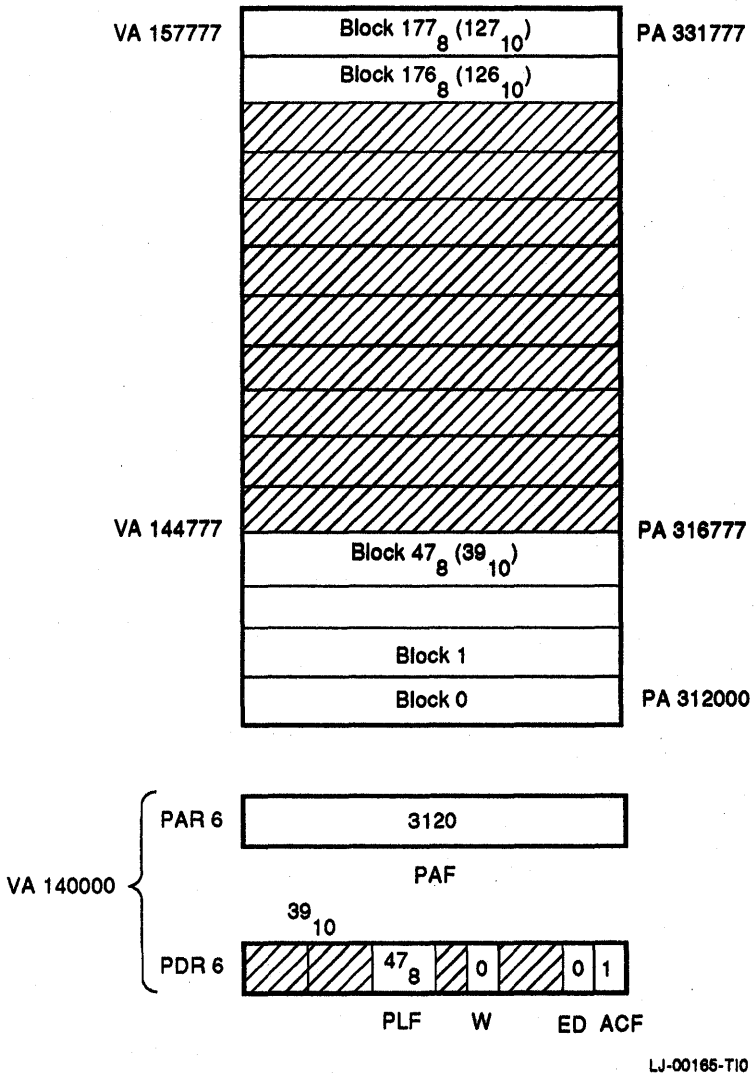


Figure 1–18 Typical Memory Page (17772516)

2. The initial address of the page was determined from the PAF of PAR6 (312000 = 31208 blocks 40 (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 (47 + 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 shown in Figure 1–11.
5. The W-bit indicates that no locations in this page have been modified (that is, 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 (that is, 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.

1.4.7.2 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 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.

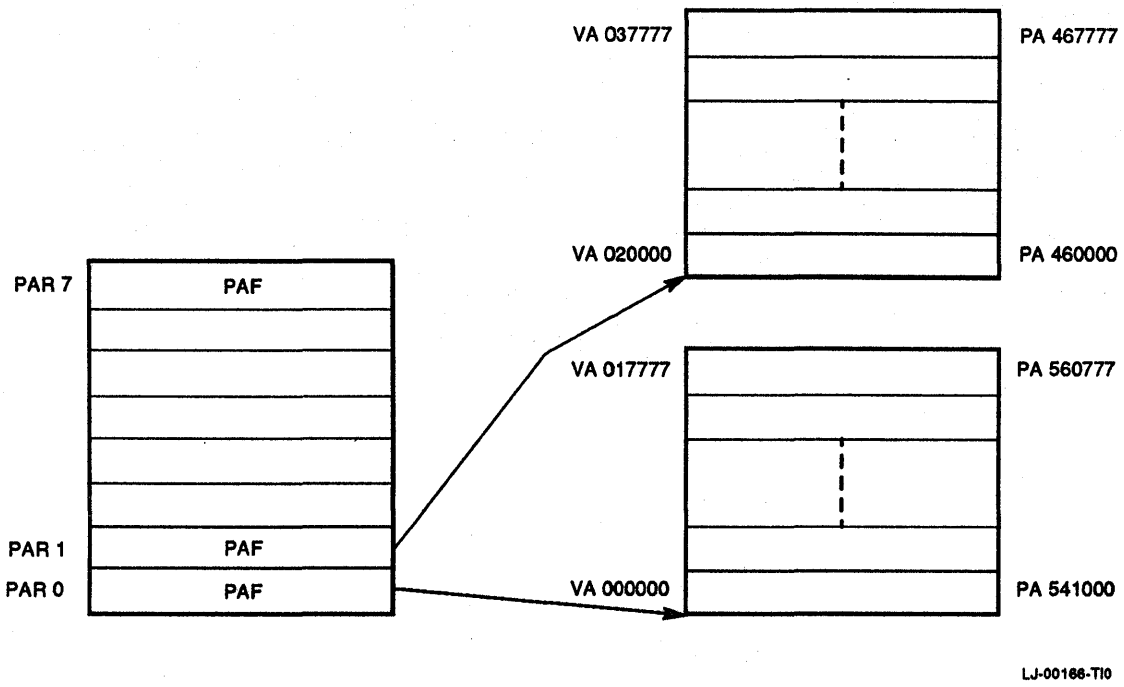


Figure 1–19 Nonconsecutive Memory Pages

1.4.7.3 Stack Memory Pages

When constructing programs, it is often desirable to isolate all program variables from pure code (that is, 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 shows 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 the 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.

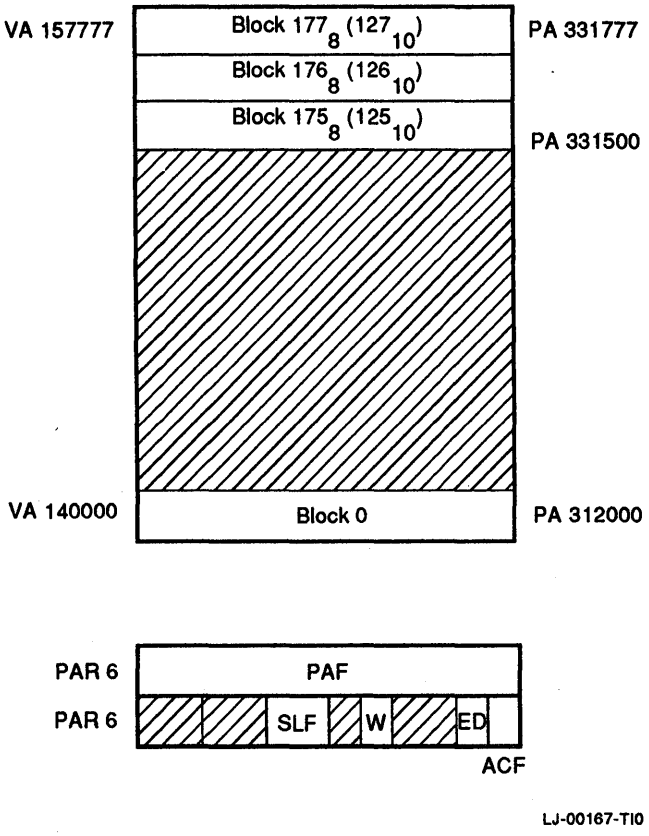


Figure 1–20 Typical Stack Memory Page

1.4.8 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 which are 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.

1.5 KDJ11-E Memory System Implementation

Technological advances implemented on the KDJ11-E allow the onboard memory to perform at cache-like speeds, eliminating the need for cache. The KDJ11-E cache register set has been implemented for compatibility purposes only and all bits should remain cleared at all times.

1.5.1 Memory System Error Register (17777744)

The memory system error register (MSER), at address 17777744, reflects the status of cache and main memory parity errors. Figure 1–21 shows the register format. Table 1–12 provides memory system error register bit descriptions.

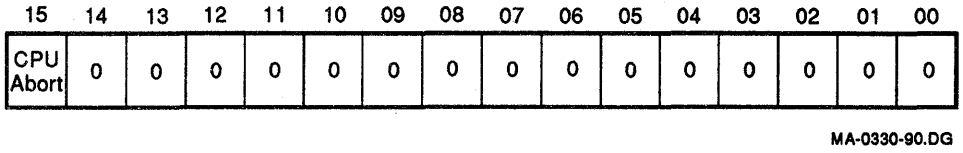


Figure 1–21 Memory System Error Register Format (17777744)

Table 1–12 Memory System Error Register Bit Descriptions

Bits	Name	Status	Function
15	CPU abort	RO	This bit is set if a main memory parity error results in an instruction abort (only during the demand read cycle). Main memory parity errors always cause an abort.
<14:00>	Unused		Read as 0s.

Main memory parity errors always cause the CPU to abort the current instruction, set MSER <15>, and trap through vector location 114.

1.5.2 Cache Control Register (17777746)

The cache control register (CCR) is at address 17777746. This register is used for compatibility with former designs only. All bits should be cleared at all times. Figure 1–22 shows the register format. Table 1–13 describes the cache control register bits.

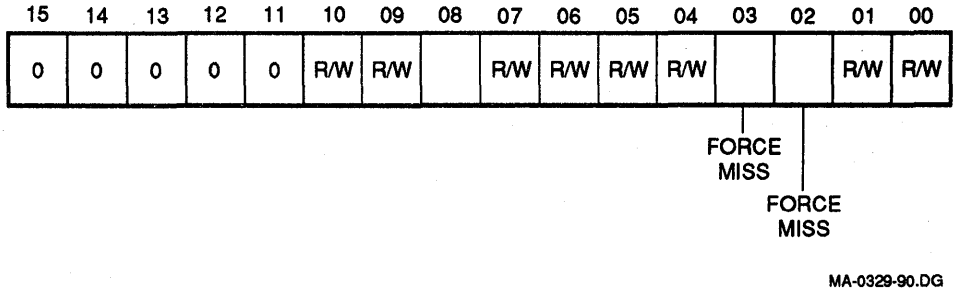


Figure 1–22 Cache Control Register Format (17777746)

NOTE
During normal operations all bits should be cleared at all times.

Table 1–13 Cache Control Register Bit Descriptions

Bits	Name	Function
<3:2>	Force Miss	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.
<14:00>	Unused	Read as 0s.

1.5.3 Hit/Miss Register (17777752)

This register, at address 17777752, is for compatibility purposes only.

1.5.4 Parity CSR Register (17772100)

The parity control status register (CSR) is used to control parity checking. Parity checking should only be enabled after the entire memory has been written to and the right parity has been generated. Wrong parity can also be written for test purposes (WWPAR). Figure 1–23 shows the register format. Table 1–14 describes the parity CSR register (17772100).

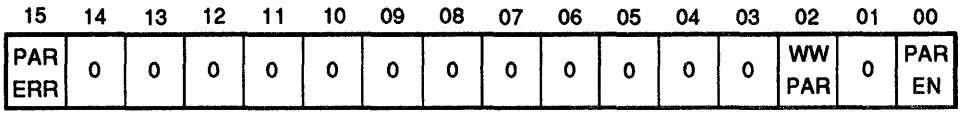


Figure 1–23 Parity CSR Register (17772100)

Table 1–14 Parity CSR Register (17772100)

Bits	Name	Status	Function
15	PAR ER	R/W	Parity error. Bit 15 is set when a parity error is detected in the main memory. Bit 15 is reset by a power-up, INIT command. This bit can also be written.
<14:03>	Unused	–	Read as 0.
02	WWPAR	R/W	Write wrong parity. When set, "wrong" parity bits are written in to the main memory with each write access made. This provokes a parity error with each read access that follows.
01	Unused	–	Read as 0.
00	PAREN	R/W	Parity checking enable. When set, parity checking is enabled. When enabled, parity errors detected will cause a parity trap. This bit is reset by a power-up or INIT command.

1.6 Private Memory Interconnect

The private memory interconnect (PMI) is a unique Q22-bus protocol that provides a high performance data path between the KDJ11-E module and KTJ11-B (UNIBUS adapter module) in UNIBUS systems. 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 6.

1.6.1 PMI Protocol

In a Q22-bus system, the KDJ11-E 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-E CPU is the default PMI master and the KTJ11-B UNIBUS adapter is the default UNIBUS master. When the CPU, as PMI master, references an I/O page address on the UNIBUS, the UNIBUS adapter responds as the slave on the PMI 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 by 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 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 BR4 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 memory, the PMI I/O page or UNIBUS adapter I/O page.

1.6.2 PMI Data Transfers

There are three general categories for the PMI data transfer cycles:

- Data In/Data In Pause (DATI/DATIP)
- Block Data In (DATBI)
- Data Out/Data Out Byte (DATO/DATOB)

These three cycles 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 memory. When the PMI bus master accesses the I/O page it can only read one word.

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 memory. The PMI bus master cannot use the DATBI cycle when accessing the I/O page.

The PMI bus master can only start DATBI transfers on even word boundaries. This means that address bits <1:0> must be equal to 0s. 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 1s.

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 KDJ11-E Serial Line Units

The KDJ11-E has eight serial line units (SLUs). As a factory setting, SLU 8 is configured as the console port, with a base address/vector of 177560/60. Optionally, SLU 8 can follow in ascending order after SLUs 1 - 7 as the eighth SLU or it can be set independently from the other seven SLUs as the console/SLU.

NOTE
Modem control is not available.

Table 1–15 describes SLU 8.

Table 1–15 Console/SLU Panel - SLU 8

Switch 1 Setting	Function	Address	Vector
Off	Console enabled	177560	60
On	Console disabled	176570 (176670)	370 (470)

NOTE
Console SLU 8 has fixed parameters of eight data bits, no parity, and one stop bit. The baud rate is determined by switches 6, 7, and 8.

SLU lines 1-7 can be set independently from each other. See Chapter 4 for information on using the setup menu.

1.7.1 Silo Buffer Length

The serial interfaces located on the KDJ11-E are "DL" type interfaces. Each line on the KDJ11-E has a five-character silo. The silos were implemented on the KDJ11-E to prevent overrun problems from occurring on heavily loaded systems. Serial characters that are not picked by the operating systems under peak loading, are stored in these silo buffers. If the operating system does not get around to these characters the silo buffer will slowly be filled as other characters are being received.

The serial line interfaces are controlled by a dedicated processor and each line has a five-character silo. This is a compromise between overrun protection and operating transparency. It is possible, however, to change the silo size on each line individually to any value between two and 65 characters by modifying the firmware on the KDJ11-E.

NOTE
It is explicitly emphasized that Digital Equipment Corporation does not support any modules in which the original code in the EPROMs is modified. Therefore, this modification should be performed by appropriately trained personnel only.

To modify the firmware on the KDJ11-E, the following is needed:

- An EPROM programming device with the capability of editing individual bytes.
- An empty EPROM that is the same type (27128) as the one on the module.

For each individual channel, there are three locations within the SLU EPROM that contain the data that determines the buffer size. All three locations for each channel must contain the same data. (For the location of the SLU EPROM see Figure 1–24).

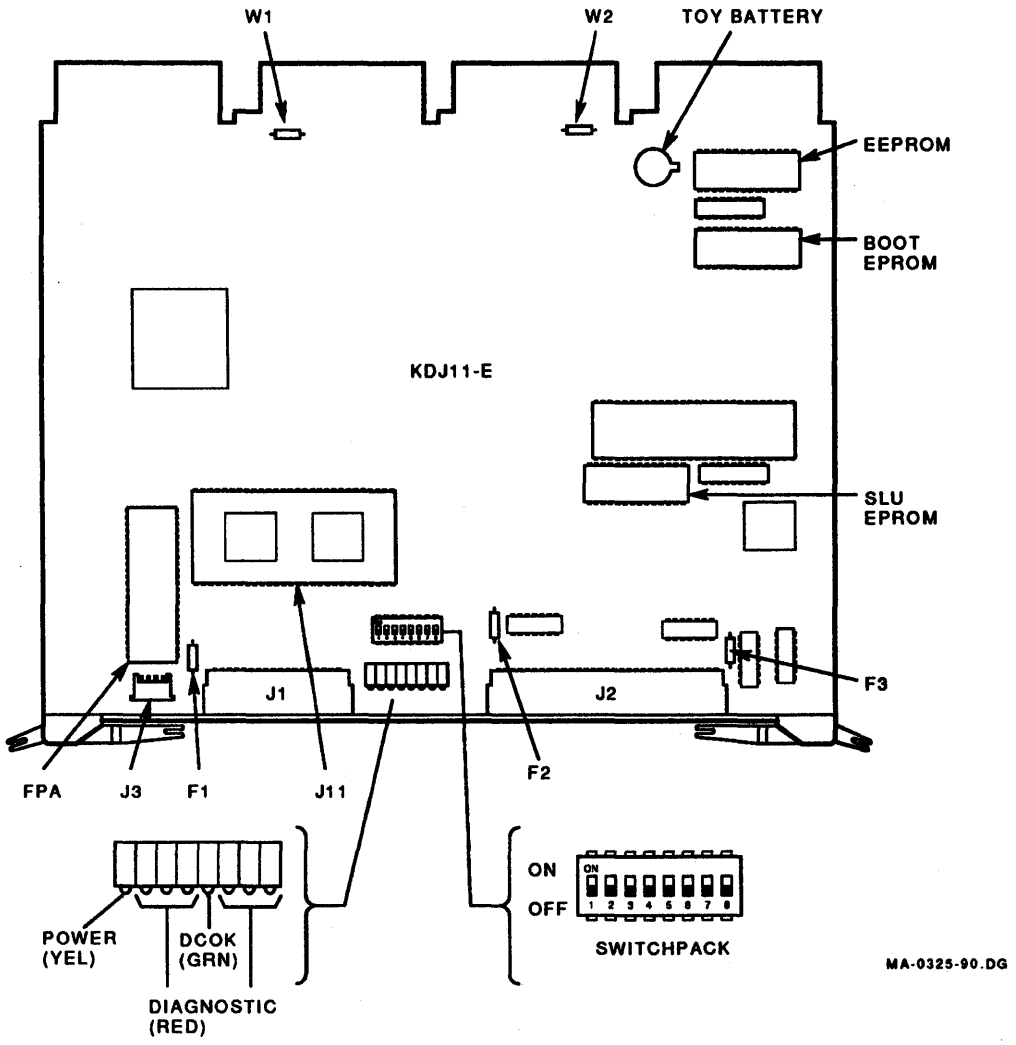


Figure 1–24 KDJ11-E Jumper and DIP Switch Locations

Example for Channel 1:

Reference Field Address / Data			Pointer Address		Actual Parameter Address / Data	
070H:	CD 02	=	02CDH	=	02CDH:	08
072H:	E6 02	=	02E6H	=	02E6H:	08
074H:	49 06	=	0649H	=	0649H:	08

The addresses 70H, 72H, 74H point to where in the EEPROM the parameters are located, and this location is where the new values are to be entered. A similar procedure is used for the other seven channels. The fixed reference fields have the following addresses:

Reference Field	Channel
070H, 072H, 074H	1
078H, 07AH, 07CH	2
080H, 082H, 084H	3
088H, 08AH, 08CH	4
090H, 092H, 094H	5
098H, 09AH, 09CH	6
0A0H, 0A2H, 0A4H	7
08AH, 0AAH, EACH	8

All addresses are specified in a hexadecimal format. To determine from the parameter data what the silo buffer size equals, use the following formula:

(Parameter data/2) + 1 = Silo Buffer Length

For example, if the Parameter data equals 8 (factory setting) then the silo buffer size equals five.

8 divided by 2 equals 4, plus 1 equals 5.

To determine what the parameter should be, insert the desired silo buffer size into the following formula:

(Silo Buffer Length - 1) x 2 = Parameter data

For example, if the desired buffer size equals 65, then the parameter data equals 128 decimal (80H).

65 minus 1 equals 64, times 2 equals 128

The valid range of parameters lies between 2 and 80H (128 decimal) which corresponds to effective buffer lengths of 2 to 65 characters. All three parameters per channel spread throughout the EEPROM must be of the same value.

1.7.2 Serial Line Unit Registers

Each serial line unit has four registers:

- Receiver status register
- Receiver data buffer register
- Transmitter status register
- Transmitter data buffer register

These registers are described in the following sections.

1.7.2.1 Receiver Status Register (1777xxx0)

Figure 1–25 shows the Receiver Status Register (RCSR) format at address 1777xxx0. Table 1–16 provides the receiver status register bit descriptions.

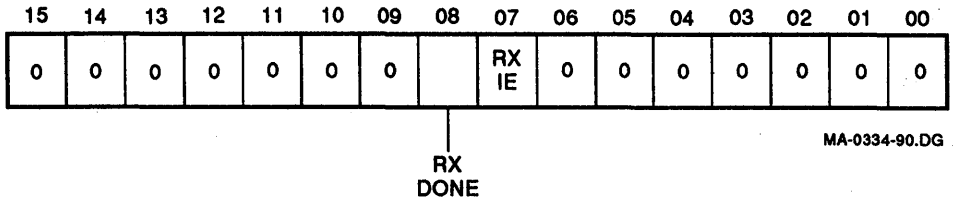


Figure 1–25 Receiver Status Register Format (1777xxx0)

Table 1–16 Receiver Status Register Bit Descriptions

Bits	Name	Status	Function
<15:08>	Unused		Read as 0s.
07	RX DONE	RO	Receiver Done. This bit is set when an entire character has been received and is ready to be read from the RBUF register. This bit is automatically cleared when RBUF is read. It is also cleared by power-up.
06	RX IE	R/W	Receiver Interrupt Enable. This bit is cleared by power-up and bus initialization. If both RCVR DONE and RCVR INT ENB are set, a program interrupt is requested.
<05:00>	Unused		Read as 0s.

1.7.2.2 Receiver Data Buffer Register (1777xxx2)

Figure 1–26 shows the receiver data buffer register (RBUF) format at address 1777xxx2. Table 1–17 provides the receiver data buffer register bit descriptions.

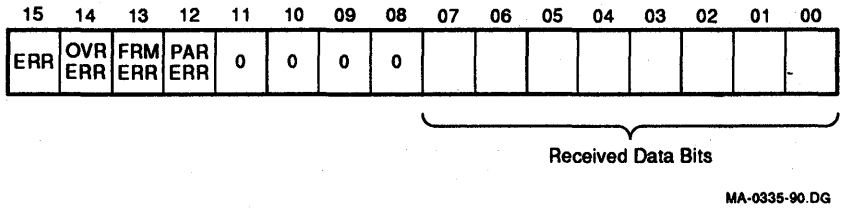


Figure 1–26 Receiver Data Buffer Register Format (1777xxx2)

Table 1–17 Receiver Data Buffer Register Bit Descriptions

Bits	Name	Status	Function
15	ERR	RO	Error. This bit is set if RBUF 14 or 13 is set. ERR is cleared if these two bits are cleared. This bit cannot generate a program interrupt.

Table 1–17 (Cont.) Receiver Data Buffer Register Bit Descriptions

Bits	Name	Status	Function
14	OVR ERR	RO	Overrun Error. This bit is set if a previously received character is not read before being overwritten by the present character.
13	FRM ERR	RO	Framing Error. This bit is set if the present character has no valid stop bit. This bit is used to detect a break.

NOTE
Error conditions remain present until the next character is received. At that point, the error bits are updated. The error bits are not necessarily cleared by power-up.

12	Receiver Parity Error		This bit is set when the parity received does not match the parity expected.
<11:08>	Unused		Read as 0s.
<07:00>	Received Data Bits	RO	These read-only bits contain the received character.

1.7.2.3 Transmitter Status Register (1777xxx4)

Figure 1–27 shows the transmitter status register (XCSR) format at address 1777xxx4. Table 1–18 provides the transmitter status register bit descriptions.

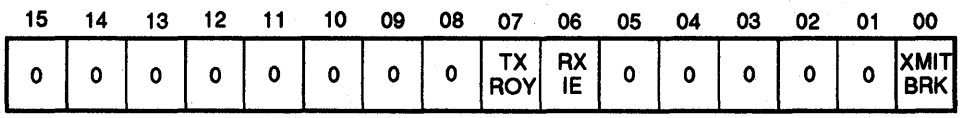


Figure 1–27 Transmitter Status Register Format (1777xxx4)

Table 1–18 Transmitter Status Register Bit Descriptions

Bits	Name	Status	Function
<15:08>	Unused		Read as 0s.
07	TX RDY	RO	Transmitter Ready. This bit is cleared when XBUF is loaded. The bit sets when XBUF can receive another character. XMT RDY is set by power-up and by bus initialization.
06	TX IE	R/W	Transmitter Interrupt Enable. This bit is cleared by power-up and by bus initialization. If both TX RDY and TX IE are set, a program interrupt is requested.
<05:01>	Unused		Read as 0s.
00	XMIT BRK	R/W	Transmit Break. When this bit is set, the serial output is forced to the space condition. XMIT BRK is cleared by power-up and by bus initialization.

1.7.2.4 Transmitter Data Buffer Register (1777xxx6)

Figure 1–28 shows the transmitter data buffer register (XBUF) format at address 1777xxx6. Table 1–19 contains the bit descriptions.

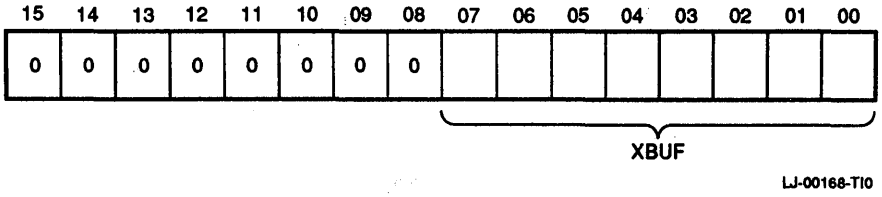


Figure 1–28 Transmitter Data Buffer Register Format (1777xxx6)

Table 1–19 Transmitter Data Buffer Register Bit Descriptions

Bits	Name	Status	Function
<15:08>	Unused		Read as 0s.
<07:00>	XBUF	WO	These eight bits are used to load the transmitted character.

Table 1–20 describes the console/SLU register settings for a data base address set at 176500.

Table 1–20 Console/SLU Register Settings - Base Address 176500

SLU	Address	Register	Vector
1	176500	RCSR1	300
	176502	RBUF1	
	176504	XCSR1	304
	176506	XBUF1	
2	176510	RCSR2	310
	176512	RBUF2	
	176514	XCSR2	314
	176516	XBUF2	
3	176520	RCSR3	320
	176522	RBUF3	
	176524	XCSR3	324
	176526	XBUF3	
4	176530	RCSR4	330
	176532	RBUF4	
	176534	XCSR4	334
	176536	XBUF4	
5	176540	RCSR5	340

Table 1–20 (Cont.) Console/SLU Register Settings - Base Address 176500

SLU	Address	Register	Vector
	176542	RBUF5	
	176544	XCSR5	344
	176546	XBUF5	
6	176550	RCSR6	350
	176552	RBUF6	
	176554	XCSR6	354
	176556	XBUF6	
7	176560	RCSR7	360
	176562	RBUF7	
	176564	XCSR7	364
	176566	XBUF7	
8	177560	RCSR8	60
	177562	RBUF8	
	177564	XCSR8	64
	177566	XBUF8	

Table 1–21 describes the console/SLU register settings for a data base address set at 176600.

Table 1–21 Console/SLU Register Settings - Base Address 176600

SLU	Address	Register	Vector
1	176600	RCSR1	400
	176602	RBUF1	
	176604	XCSR1	404
	176606	XBUF1	
2	176610	RCSR2	410
	176612	RBUF2	
	176614	XCSR2	414
	176616	XBUF2	
3	176620	RCSR3	420
	176622	RBUF3	
	176624	XCSR3	424
	176626	XBUF3	
4	176630	RCSR4	430
	176632	RBUF4	
	176634	XCSR4	434

Table 1–21 (Cont.) Console/SLU Register Settings - Base Address 176600

SLU	Address	Register	Vector
	176636	XBUF4	
5	176640	RCSR5	440
	176642	RBUF5	
	176644	XCSR5	444
	176646	XBUF5	
6	176650	RCSR6	450
	176652	RBUF6	
	176654	XCSR6	454
	176656	XBUF6	
7	176660	RCSR7	460
	176662	RBUF7	
	176664	XCSR7	464
	176666	XBUF7	
8	177560	RCSR8	60
	177562	RBUF8	
	177564	XCSR8	64
	177566	XBUF8	

1.7.3 Break Response

The KDJ11-E console serial line unit may be configured either to perform a halt operation or to have no response when a break condition is received. A halt operation will cause the processor to halt and enter the octal debugging technique (ODT) microcode. The halt-on-break option is selected using the halt-on-break parameter in the setup menu.

1.8 Boot and Diagnostic Register Set

The following registers are used by the KDJ11-E ROM code:

- Control/status register
- Page control register
- Configuration and display register
- Additional status register
- Maintenance register

These registers are described in the following paragraphs.

1.8.1 Control/Status Register (17777520)

The control/status register (CSR), at address 17777520, is both word and byte addressable. Figure 1–29 shows the register format. The CSR allows the boot and diagnostic ROM programs to test battery backup status, to set parameters for the line clock, to enable the console halt-on-break feature, and to enter or exit from standalone mode.

The CSR also allows these programs to selectively disable the response of the boot and diagnostic ROMs at addresses 17765000 through 17765776 or at addresses 17773000 through 17773776, or both, and to control read/write access to the EEPROM memory.

Programs that access the I/O page can use the CSR to alter the line clock parameters, to enable or disable the halt-on-break feature, and to control access to the EPROM and EEPROM memories.

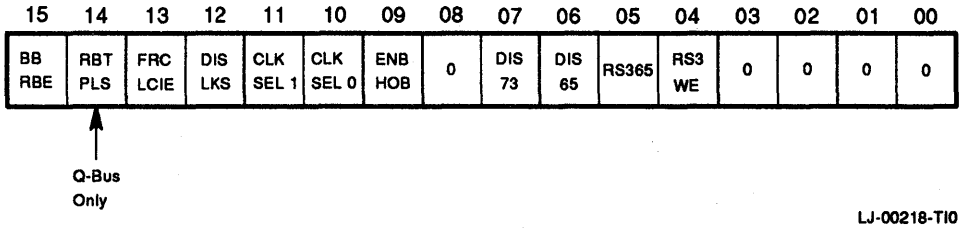


Figure 1–29 Control/Status Register Format (17777520)

Table 1–22 provides the control/status register bit descriptions.

Table 1–22 Control/Status Register Bit Descriptions

Bits	Name	Status	Function
15	BB RBE		Battery Backup Reboot Enable. When set, this bit indicates that battery backup failed to maintain voltages to the memory system during the previous power failure. When this bit is clear, it indicates that the system does not feature battery backup, or that battery backup maintained voltages during the previous power failure. This signal is received from backplane pin BH1 and latched when DC OK is asserted.
14	RBT PLS Q-bus only		Reboot Pulse. This read-only bit is set (1) when the DCOK input is pulsed 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.

Table 1–22 (Cont.) Control/Status Register Bit Descriptions

Bits	Name	Status	Function
13	FRC LCIE		Force Line Clock Interrupt Enable. If this bit is set, assertion of the signal selected by CSR <11:10> and <1:0> will unconditionally request interrupts. If FRC LCIE is clear, assertion of the selected signal will request interrupts only if the line clock status register bit 6 (LCIE) is set under program control. FRC LCIE is cleared by the negation of DCOK.
12	DIS LKS		Line Clock Status Register Disable. If this bit is set, the line clock status register (LKS) is disabled. If this bit is clear, LKS is enabled and responds to bus address 17777546. LKS DIS is cleared by the negation of DCOK.
11 10	CLK SEL1 CLK SEL0		Clock Select Bits 1 and 0. These two bits select the source of the line clock interrupt request: **CLK SEL1 / CLK SEL0 / Source of Interrupt** 0 / 0 / Power supply 0 / 1 / On-board 50 Hz 1 / 0 / On-board 60 Hz 1 / 1 / On-board 800 Hz Both bits are cleared by the negation of DCOK.
09	ENB HOB	R/W	Enable Halt-on-Break. When this bit is set, the console serial line unit halt-on-break feature is enabled. When this bit is clear, the feature is disabled. ENB HOB is cleared by the negation of DCOK.
08	Unused		Read as 0.
07	DIS 73	R/W	Disable 17773000. When this bit is set, response of the 16-bit ROM memory to addresses 17773000 through 17773776 is disabled, allowing the operation of an external ROM that uses those addresses. When DIS 73 is clear, the 16-bit ROMs respond to those addresses, using the high byte of the page control register as the most significant address bits. DIS 73 is cleared by the negation of DCOK.

Table 1–22 (Cont.) Control/Status Register Bit Descriptions

Bits	Name	Status	Function
06	DIS 65	R/W	Disable 17765000. When this bit is set, response of the boot and diagnostic 16-bit and 8-bit ROM memory to addresses 17765000 to 17765776 is disabled; this allows the operation of external ROM which uses those addresses. When DIS 65 is clear, the ROM memory selected by CSR bit 5 responds to those addresses, using the low byte of the page control register as the most significant address bits. DIS 65 is cleared by the negation of DCOK.
05	RS365		When this read/write bit is set (1), the EEPROM responds to addresses between 17765000 and 17765776, provided that bit 6 of the CSR is reset. When this bit 5 is reset (0), then the ROM at these addresses is selected instead. In both cases, the high byte of the ROM address is made up of the low byte of the PCR. The bit is reset through the negation of the DCOK inputs.
04	RS3 WE		When this read/write bit is set and the CSR bit 5 is set with the CSR bit 6 reset, then the program is able to write into the EEPROM. This bit is reset through power-up and initialization routines.
<03:00>	Unused		Read as 0s.

1.8.2 Page Control Register (17777522)

The page control register (PCR) is a read/write register that can be addressed by words and bytes. Only the bits <15:9> and <7:1> can be used. The remaining bits will always be read as 0s. This register is reset through the negation of the DCOK input.

The PCR bits <15:9> become the ROM address bits <15:9> for the address area from 17773000 to 17773776 and together with the current address bits <8:0>, they make up a 16-bit address for the boot EPROM . The PCR bits <7:1> becomes the EEPROM address bits <15:9> for the addresses from 17765000 TO 17765766 and together with the current address bits <8:0> they make up a 16-bit address for the EEPROM.

The control/status register (CSR) bits <7:4> control the access to the boot EPROM and EEPROM.

Figure 1–30 shows the register format. Table 1–23 describes the page control register bits.

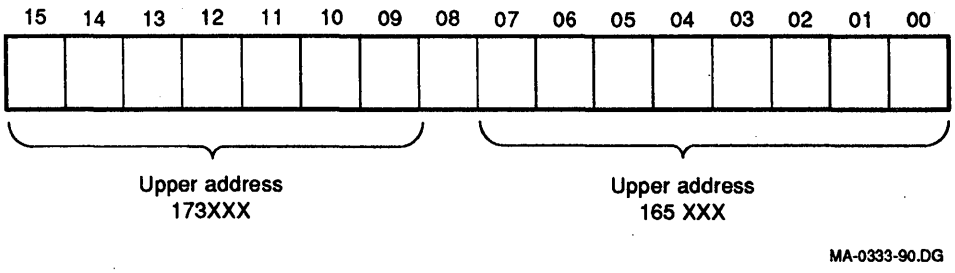


Figure 1–30 Page Control Register Format (17777522)

Table 1–23 Page Control Register Bit Descriptions

Bits	Name	Function
<15:09>	Upper address 173xxx	Used for address area 17773000 to 17773776.
08	Unused	Read as 0.
<07:00>	Upper address 165xxx	Used for address area 17765000 to 17766766.

1.8.3 Configuration and Display Register (17777524)

The configuration and display register (CDR) consists of two independent registers which are pointed to by the same address:

NOTE
Switch in the *off* position is read as 1.

- Read-only boot and diagnostic configuration register.

  The read-only boot and diagnostic configuration register specifies the status of the configuration switches (8 to 1) that are located on the module. When switches 8 to 1 on the KDJ11-E are set to the off position, the state of the register bits can be controlled using external switches. Figure 1–31 shows the register format.

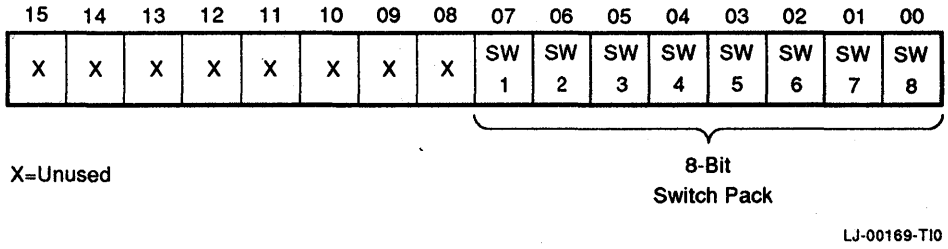


Figure 1–31 Boot and Diagnostic Configuration Register Format (17777524 - Read-Only)

- Write-only boot and diagnostic display register.

  The write-only boot and diagnostic display register (BDR), at address 17777524, allows the boot diagnostic programs to light the LEDs on the KDJ11-E module. These display bits are also available on external connector J02. Bits <05:00> are cleared on power-up (all LEDs on) by the negation of DCOK. Figure 1–32 shows the register format.

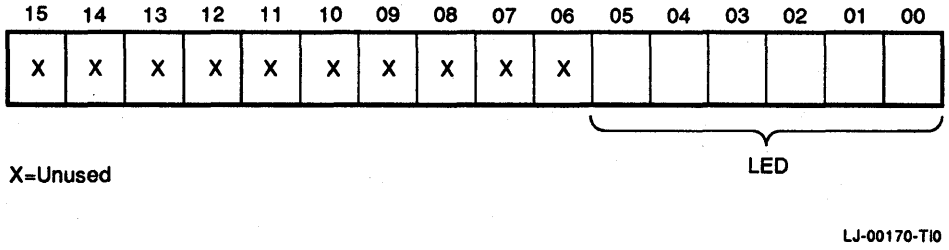


Figure 1–32 Boot and Diagnostic Display Register Format (17777524 - Write-Only)

Table 1–24 describes the display register bits.

Table 1–24 Display Register Bit Descriptions

Bits	Name	Function
<15:06>	Unused	Read as 0s.
<05:00>	LED 5–0	These bits enable the boot and diagnostic programs to light the LEDs located at the top of the CPU module. Clearing any of these bits lights the corresponding LED.

1.8.4 Additional Status Register (17777526)

The additional status register controls board-internal functional units such as memory and interfaces. It is used by the boot and test firmware for testing and configuration purposes. Basic processor functions are performed here. Figure 1–33 shows the register format.

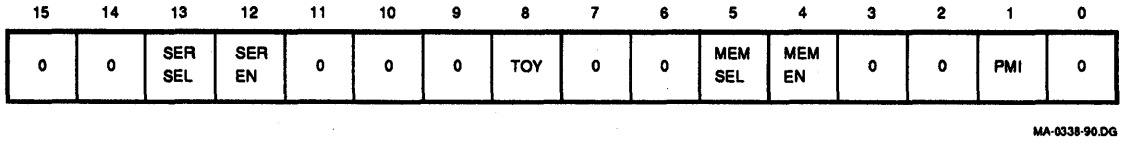


Figure 1–33 Additional Status Register (17777526)

Table 1–25 describes the additional status register.

Table 1–25 Additional Status Register

Bits	Status	Function
<15:14>	Unused	Read as 0s.

Table 1–25 (Cont.) Additional Status Register

Bits	Status	Function
<13:12>	R/W	Selects interfaces internally as follows:

SERSEL 13	SEREN 12	Serial lines address/vector
0	1	SLUs on 176500/300
1	1	SLUs on 176600/400
0	0	No serial lines selected.

Bits	Status	Function
<11:09>	Unused	Read as 0s.
8	R/W	This bit is used for serial communication to the TOY.
<07:06>	Unused	Read as 0s.
<05:04>	R/W	Selects internal memory as follows:

MEMSEL	MEMSEN	Function
0	0	Disabled
0	1	0-2 Mbyte memory
1	1	0-4 Mbyte memory

<03:02>	Unused	Read as 0s.
1	R/W	Flag for PMI-Cycle (set by system).
0	Unused	Read as 0.

1.9 Line Frequency Clock and Status Register (17777546)

The line clock provides the system with timing information at fixed intervals determined by the LTC line (BEVENT) or by one of the on-board KDJ11-E frequency signals. The signals are programmed by boot and diagnostic controller status register bits <11:10>. Typically, LTC cycles at the AC line frequency, producing intervals of 16.7 ms (60 Hz line) or 20.0 ms (50 Hz line). The three on-board frequencies are 50 Hz, 60 Hz and 800 Hz.

The LKS, at address 17777546, allows line clock interrupts to be enabled and disabled under program control. Alternatively, line clock interrupts can be unconditionally enabled by setting CSR <13> (FRC LCIE). Program recognition of the clock status register can be disabled by setting CSR <12> (LKS DIS).

The normal KDJ11-E configuration is FRC LCIE and LKS DIS both clear. These bits are set up by the boot and diagnostic ROM programs from the KDJ11-E configuration data. Figure 1–34 shows the clock status register format. Table 1–26 describes the clock status register bits.

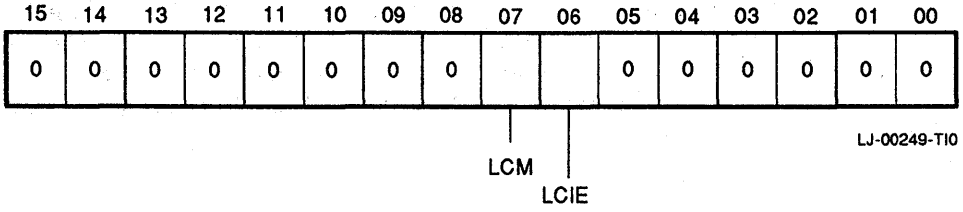


Figure 1–34 Clock Status Register Format (17777546)

Table 1–26 Clock Status Register Bit Descriptions

Bits	Name	Status	Function
<15:08>	Unused		Read as 0s.
07	LCM	R/W	Line Clock Monitor. This bit is set by the leading edge of the external BEVENT line (or of one of the three on-board clock frequencies), and by bus initialization. LCM is cleared automatically on processor interrupts acknowledge. It is also cleared by writes to the LKS with bit 7 = 0.
06	LCIE	R/W	Line Clock Interrupt Enable. This bit, when set, causes the set condition of LCM (LKS bit 7) to initiate a program interrupt request at a priority level of 6. When LCIE is clear, line clock interrupts are disabled. LCIE is cleared by power-up and by bus initialization. LCIE is held set INIT. LCIE is held set when CSR bit 13 (FRC LCIE) is set.
<05:00>	Unused		Read as 0s.

1.9.1 Maintenance Register (17777750)

The DCJ11 microcode addresses the maintenance register at address 17777750 and reads BPOK H, the power-up option code, and the halt/trap option bit. Other bits in the maintenance register, not used by DCJ11 microcode, contain information on the module type and system parameters which is useful to operating system and diagnostic software. Figure 1–35 shows the register format.

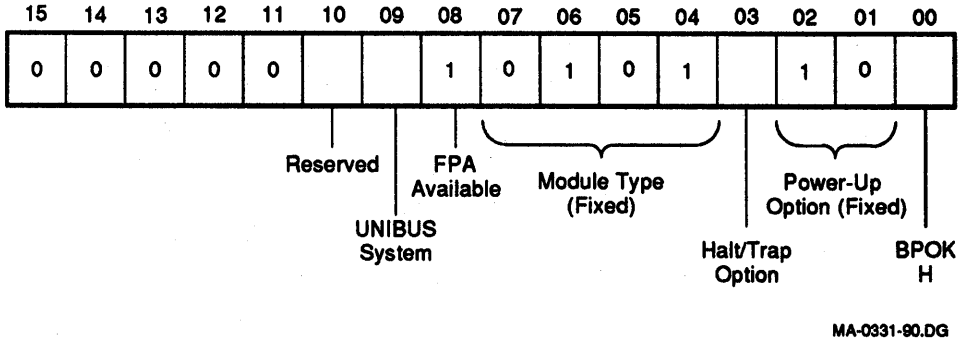


Figure 1–35 Maintenance Register Format (17777750)

The power-up option code is hard-wired for standard bootstrap operation (code 2). The PSW is set to 340, and the processor begins program execution at address 173000. The boot and diagnostic code, which starts at address 173000, configures the KDJ11-E. The code runs standalone diagnostics before acting on the user-specified power-up option stored as part of the EEPROM configuration data. Table 1–27 describes the maintenance register bits.

Table 1–27 Maintenance Register Bit Descriptions

Bits	Name	Status	Function
<15:11>	Unused		Reserved for future use. Read as 0s.
10	Unused		Reserved for future use. Read as 0.
09	UNIBUS System	RO	This bit reflects the status of the externally applied UNIBUS adapter line. A 1 indicates that the system includes a UNIBUS adapter.
08	FPA Available	RO	Read as 1.
<07:04>	Module Type		This 4-bit code is hard-wired as a 5, indicating a KDJ11-E module.
03	Halt/Trap	R/W	This read/write bit determines the response of a processor to a kernel mode halt instruction. Setting the bit selects the trap option, causing the CPU to trap to location 4. Clearing the bit selects the halt option, causing the CPU to halt and enter ODT. This bit is cleared by the negation of DCOK and is set by the boot and diagnostic ROM code if the trap option is selected by a bit in the configuration RAM. The trap option is not intended for normal use and is reserved for controller applications.
<02:01>	Power-up code		This 2-bit code is hard-wired as a 2. At power-up, the processor sets the PC to 173000 and sets the PSW to 340. It then starts program execution at location 173000, which is the starting location for the KDJ11-E boot and diagnostic ROM program. These programs test the KDJ11-E module and then implement the user-selected power-up option specified in the configuration data.
00	BPOK H		This bit is set (1) if the PMI bus signal BPOK H is asserted, indicating that AC power is acceptable.

1.10 Time of Year Clock (TOY Clock)

The TOY clock is used to establish the time and date after a power down. It is not used during normal operation. (Normal timekeeping for the KDJ11-E is performed with the Line Time clock.) The TOY clock uses 24-hour mode.

In the absence of power, battery backup is derived from an on-board lithium battery.

The TOY clock provides:

- Hundredths of seconds
- Seconds
- Minutes
- Hours
- Day of week

- Month
- Year

Adjustments for months with less than 31 days and leap year are automatic. The time and date can be read and set from dialog mode or by programming the time chip.

1.10.1 Programming Information

Communication between the KDJ11-E module and the TOY clock is through bit 8 of the additional status register (17777526).

Communication between the KDJ11-E module and TOY clock is established by pattern recognition of a unique serial bit stream of 64 bits (Bits 0 - 63).

Table 1–28 shows the recognition pattern.

Table 1–28 Recognition Pattern

15 14 13	12 11 10	9 8 7	6 5 4	3 2 1 0		Character Recognition Word
0 0 1	1 1 0	1 0 1	1 0 0	0 1 0 1	035305	1 Bits 0-15
0 1 0	1 1 1	0 0 1	0 1 0	0 0 1 1	056243	2 Bits 16-31
0 0 1	1 1 0	1 0 1	1 0 0	0 1 0 1	035305	3 Bits 32-47
0 1 0	1 1 1	0 0 1	0 1 0	0 0 1 1	056243	4 Bits 48-63

1.10.1.1 Clear the Comparison Register Pointer

Prior to sending the recognition pattern to the TOY clock, the comparison register pointer must be cleared. To clear the pointer read the additional status register (17777526).

1.10.1.2 Establish the Pattern Recognition

The communication with the Time Chip is established by pattern recognition of a serial bit stream of 64 bits (Table 1–28), which must be matched by executing 64 consecutive write cycles containing the proper data.

All information must shift one bit at a time through bit 8 of the additional status register (177526).

Writing starts with bit 0 of the first word and ends with bit 15 of the last word.

1.10.1.3 Updating Information in the TOY Clock

Information for the TOY clock can be read or updated after the recognition pattern sequence has been established. When the first write cycle is executed, it is compared to bit 1 of the 64-bit comparison register. If a match is found, the pointer increments to the next location of the comparison register. If a match is not found, the pointer does not advance and all subsequent write cycles are ignored. If a read cycle occurs at any time during pattern recognition, the present sequence is aborted and the comparison register pointer is reset. On the KDJ11-E module the port to the Time Chip is bit 8 of the additional status register (177526). Writing starts with bit 0 of the first word and ends with bit 15 of the last word (Table 1–28). After recognition is established, the next 64 read or write cycles either extract or update data in the Time Chip.

The Time Chip information is contained in eight registers of eight bits each, which are sequentially accessed one bit at a time after the 64-bit pattern recognition sequence has been completed. When updating the Time Chip registers, each must be handled in groups of eight bits. Writing and reading individual bits within a register could produce erroneous results. The read/write registers are defined in the following tables. Reading and writing the registers is always accomplished by stepping through all eight registers, starting with bit 0 of register 0 and ending with bit 7 of register 7.

Here is the format of the eight 8-bit registers.

Register 0

Byte Number 0
Range 00-99
Function: Hundredths of Seconds

7	6	5	4	3	2	1	0
0.1 Sec				0.01 Sec			
Digit A				Digit B			
Range 0-9				Range 0-9			

Register 1

Byte Number 1
Range: 00-59
Function: Seconds

7	6	5	4	3	2	1	0
0	10 Sec			Sec			
	Digit A			Digit B			
	Range 0-5			Range 0-9			

Register 2

Byte Number 2
Range: 00-59
Function: Minutes

7	6	5	4	3	2	1	0
0	10 Min				Min		
	Digit A				Digit B		
	Range 0-5				Range 0-9		

Register 3

Byte Number 3
Range: 00-23
Function: Hours

7	6	5	4	3	2	1	0
0	0	10 Hrs			Hours		
	Digit A				Digit B		
	Range 0-2				Range 0-9		

Register 4

Byte Number 4
Range: 01-07
Function: Day Of Week

7	6	5	4	3	2	1	0
0	0	0	0	0	Day		
	Digit A				Digit B		
	Range 0-0				Range 1-7		

Note:
The KDJ11-E ROM Code assigns day 1 as Monday.

Register 5

Byte Number 5
Range: 01-31
Function: Day in Month

7	6	5	4	3	2	1	0
0	0	10	DATE		DATE		
		Digit			Digit		
		A			B		
		Range 0-3			Range 0-9		

NOTE
The last date of the month is automatically adjusted for months with less than 31 days, including leap years.

Register 6

Byte Number 6
Range: 01-12
Function: Month

7	6	5	4	3	2	1	0
0	0	0			Month		
		Digit			Digit		
		A			B		
		Range 1-1			Range 0-9		

Register 7

Byte Number 7
Range: 00-99
Function: Year

7	6	5	4	3	2	1	0
10 Year					Year		
		Digit			Digit		
		A			B		
		Range 0-9			Range 0-9		

2
Configuration

2.1 Introduction

This chapter discusses the configuration requirements and other factors to consider when configuring the KDJ11-E 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/B 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.

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. Understand the system differences if an existing system is being upgraded.

2.2 Module Configuration

The KDJ11-E module has two jumpers and eight switches mounted in a switchpack on the module as shown in Figure 2–1.

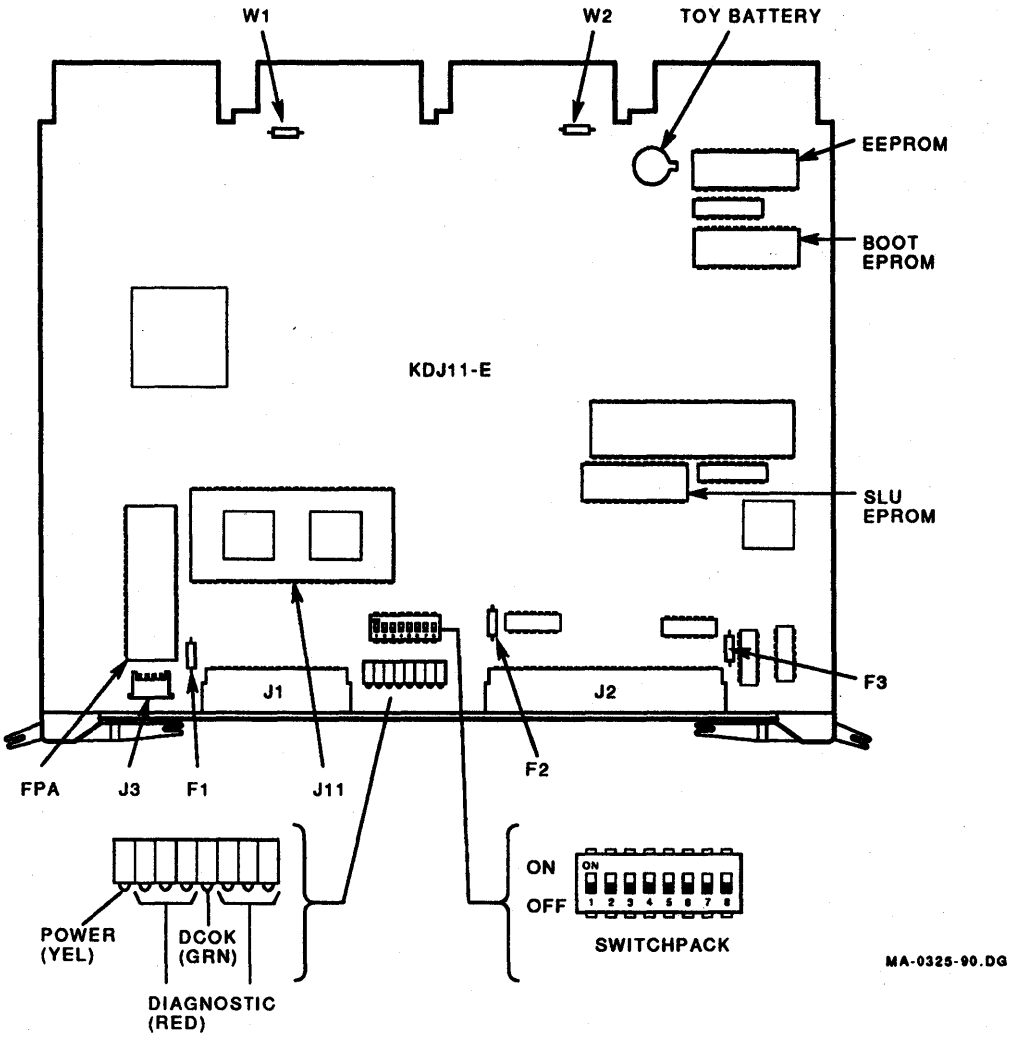


Figure 2–1 KDJ11-E Jumper and DIP Switch Locations

2.2.1 Jumpers For +5 V Power Source Selection

The KDJ11-E CPU module has two jumpers (W1 and W2), which will select the +5 V power source. The +5 V power source for the KDJ11-E can be obtained from one of the following two sources:

- Module pins AA2, BA2, CA2, DA2 (W1 & W2 installed)
- Connector J3 (W1 & W2 removed)

NOTE
Only one source may be selected.

The two jumpers, W1 and W2, connect the +5 V power source from backplane pins AA2, BA2, CA2 and DA2 to the +5 V logic on the KDJ11-E CPU module.

The KDJ11-E CPU module uses either connector J3 or backplane pins AA2, BA2, CA2, and DA2 as its source for +5 V power.

CAUTION
If connector J3 is used as the source for +5 V, jumpers W1 and W2 *must* be removed. Otherwise the two +5 V sources will be shorted together which can result in damage to the power supply.

2.2.2 Switchpack

The switchpack contains eight individual switches that provide the following functions (Figure 2–2):

- Console baud rate - selected by switches 6, 7 and 8.
- ROM mode - selected by switches 2, 3, and 4.
- Force dialog mode - selected by switch 5.
- Console SLU enable/disable - condition determined by switch 1.

The switches are wired to connector J2. The functions provided by these switches can be controlled remotely by an external switchpack connected to J2. When using an external switchpack, switches 1 through 8 on the KDJ11-E should be placed in the off position. A switch placed in the ON position on the CPU will override the selection of the corresponding switch located on the rear SLU I/O panel. Then the user can remotely turn a switch on by grounding the corresponding pin on the connector cabling. The switch connections for connector J2 are shown in Figure 2–3. The functions controlled by the switches are described in the paragraphs that follow.

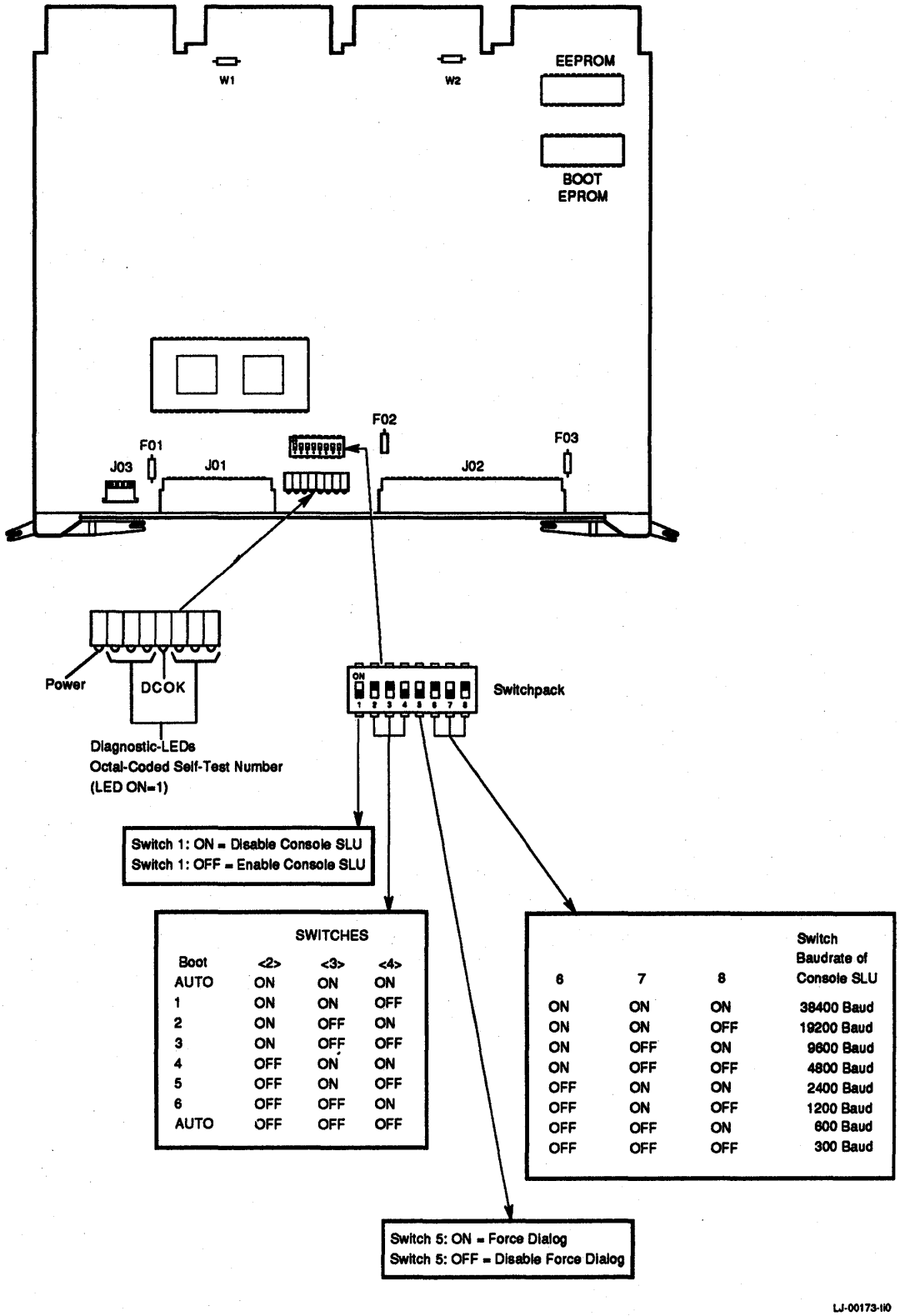


Figure 2–2 KDJ11-E Switch Configuration

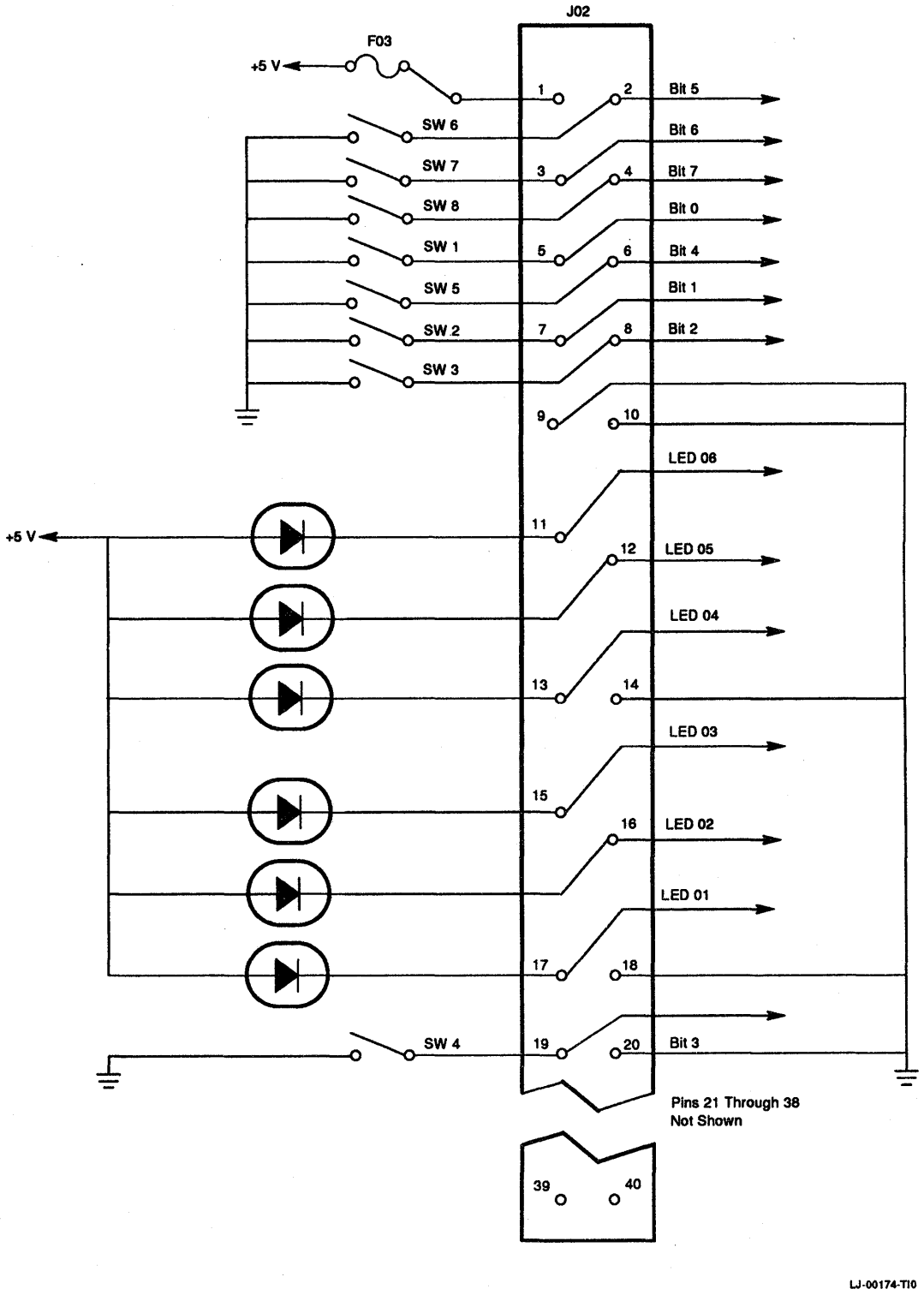


Figure 2–3 Switch Connections for Connector J2

2.2.2.1 Baud Rate Selection

The baud rate for the console SLU is selected by switches 6, 7 and 8. The baud rate selections and the switch conditions are listed in Table 2–1.

Table 2–1 Baud Rate Selections

Switches 6	7	8	Baud Rate
On	On	On	38,400
On	On	Off	19,200
On	Off	On	9600
On	Off	Off	4800
Off	On	On	2400
Off	On	Off	1200
Off	Off	On	600
Off	Off	Off	300

2.2.2.2 Force Dialog Mode

Force dialog mode allows the user to establish a dialog with the system via the console terminal by using the dialog commands that are described in Chapter 4.

If the force dialog switch (S5 on the switch pack) is on, the ROM code enters dialog mode at power-up or restart regardless of the selections in the EEPROM.

With switch 5 in the off position, (force dialog disabled) the action at power-up or restart is predetermined by the set-up in the ROM code.

2.2.2.3 ROM Mode

ROM Mode is a special automatic boot mode that is entered as a power-up/restart option to boot specific devices when one or more of switches 2 through 4 on the KDJ11-E CPU module are set to the ON position.

When switches 2 through 4 are set to one of the six combinations shown in Table 2–2 and force dialog mode is not selected, ROM mode is entered. The mode attempts to boot only the one device selected by this command. If the boot is unsuccessful, the ROM code prints out the normal error message and enters dialog mode.

Table 2–2 ROM Mode Switch Settings

Switches			Settings			Description
2	3	4	off	off	off	Normal Automatic Boot Mode.
2	3	4	on	on	off	Device 1 in list/change boot parameters.
2	3	4	on	off	on	Device 2 in list/change boot parameters.
2	3	4	on	off	off	Device 3 in list/change boot parameters.
2	3	4	off	on	on	Device 4 in list/change boot parameters.
2	3	4	off	on	off	Device 5 in list/change boot parameters.
2	3	4	off	off	on	Device 6 in list/change boot parameters.
2	3	4	on	on	on	Normal Auto Boot Mode

2.2.2.4 Console/SLU Enable - Disable

The console/SLU is enabled by setting switch 1 to the off position. If the switch is in the ON position, the system console/SLU is disabled.

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 necessary requirements.

The CPU contains EEPROMs which store the programs (ROM code) that comprehensively test the CPU, UBA and memory at power-up. The ROM code:

- Boots the user's software on various devices
- Provides memory size display
- Provides time and date of TOY clock
- Provides boot device selection
- Provides the ability to define parameters for SLUs
- Provides self-test selection
- Provides user boot area on EEPROM
- Provides support for:
  - — Hard copy terminals
  - — Video display terminals

The CPU automatically starts the ROM code each time you power up the system or restart it with the **Restart/Run/Halt** switch on the front panel. The action the ROM code takes is determined by the parameters stored in the EEPROM.

The parameters in the EEPROM determine the tests to be run, the general mode entered after testing is complete, and the final configuration of certain registers on the CPU and UBA module (UNIBUS systems only) before the system software is started. Parameters in the EEPROM can easily be changed through a program in the ROM code called *Setup Mode* without removing the CPU module. The EEPROM can also store customer bootstrap programs.

The ROM code runs tests selected by parameters in the EEPROM. After testing is complete, parameters in the EEPROM determine what action is to be taken next by the ROM code. Typically, the ROM code will automatically load and start a program from the user's disk or tape. This is commonly referred to as booting a program or automatic boot mode.

For a detailed description of EEPROM configuration parameters, refer to Chapter 4.

2.4 System Installation

The KDJ11-E 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.

2.4.1 LSI-11 Based Systems

An LSI-11 based system can be custom designed using the KDJ11-E module and compatible LSI-11 components.

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

When building a custom LSI-11 system in a BA23 enclosure, the following rules apply:

1. Slot 1 is reserved for the KDJ11-E.
2. Slots 2 and 3 can accept one dual or one quad Q-bus option.
3. Slots 4 through 8 can accept one quad or two dual Q-bus options.
4. All open A and B rows in slots 2 through 8 and all open C and D rows in slots 4 through 8 must be filled with a grant card if any modules follow in the backplane.
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 which are 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 rules 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.

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-E CPU module, the KTJ11-B UBA and compatible UNIBUS components. This type of system must be installed in the H9277-A or H9277-B backplane.

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.

2.5 Module Contact Finger Identification

The LSI-11 type modules, including the KDJ11-E, all use the same contact (pin) identification system. The contacts used on a quad-height module are identified in Figure 2–4. 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-E bus signals are identified along with the LSI-11 bus signals in Table 2–3 and the pins are identified as follows:

AV2

A = Row identifier side (Row A)
V = Pin identifier (Pin V)
2 = 1 = component side/2 = solder side

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 5.

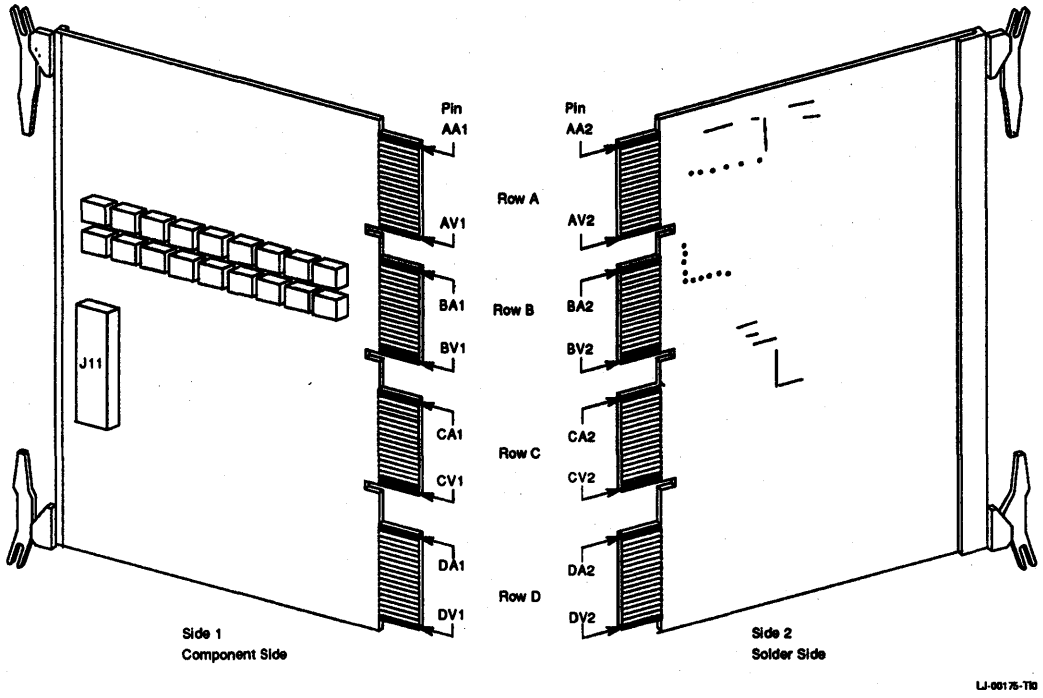


Figure 2–4 KDJ11-E Module Contacts

Table 2–3 KDJ11-E Module and LSI-11 Bus Signals

Component side			Solder Side		
Pin	**LSI-11 Bus**	**KDJ11-E**	**Pin**	**LSI-11 Bus**	**KDJ11-E**
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[1]	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

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

Table 2–3 (Cont.) KDJ11-E Module and LSI-11 Bus Signals

Component side			Solder Side		
Pin	**LSI-11 Bus**	**KDJ11-E**	**Pin**	**LSI-11 Bus**	**KDJ11-E**
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 KDJ11-E 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-E CPU module and the KTJ11-B UBA. The module signals are identified in Table 2–4.

Also refer to Table 2–5 through Table 2–7 for reference information concerning the KDJ11-E connector pin assignments J1 through J3.

Table 2–4 Module PMI Signal Assignments

Component Side		Solder Side	
Pin	**KDJ11-E**	**Pin**	**KDJ11-E**
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

Table 2–4 (Cont.) Module PMI Signal Assignments

Component Side		Solder Side	
Pin	**KDJ11-E**	**Pin**	**KDJ11-E**
DU1	Not used	DU2	Not used
DV1	Not used	DV2	Not used

Table 2–5 J1, Connector Pin Assignments

Pin	Signal	Used For
J01 pin 1	N/C	Serial Line Unit 5
J01 pin 2	GND	Serial Line Unit 5
J01 pin 3	TX	Serial Line Unit 5
J01 pin 4	GND	Serial Line Unit 5
J01 pin 5	GND	Serial Line Unit 5
J01 pin 6	N/C	Serial Line Unit 5
J01 pin 7	RX-	Serial Line Unit 5
J01 pin 8	RX+	Serial Line Unit 5
J01 pin 9	GND	Serial Line Unit 5
J01 pin 10	+12VF2	Serial Line Unit 5
J01 pin 11	N/C	Serial Line Unit 6
J01 pin 12	GND	Serial Line Unit 6
J01 pin 13	TX	Serial Line Unit 6
J01 pin 14	GND	Serial Line Unit 6
J01 pin 15	GND	Serial Line Unit 6
J01 pin 16	N/C	Serial Line Unit 6
J01 pin 17	RX-	Serial Line Unit 6
J01 pin 18	RX+	Serial Line Unit 6
J01 pin 19	GND	Serial Line Unit 6
J01 pin 20	+12VF2	Serial Line Unit 6
J01 pin 21	N/C	Serial Line Unit 7
J01 pin 22	GND	Serial Line Unit 7
J01 pin 23	TX	Serial Line Unit 7
J01 pin 24	GND	Serial Line Unit 7
J01 pin 25	GND	Serial Line Unit 7
J01 pin 26	N/C	Serial Line Unit 7
J01 pin 27	RX-	Serial Line Unit 7
J01 pin 28	RX+	Serial Line Unit 7
J01 pin 29	GND	Serial Line Unit 7

Table 2–5 (Cont.) J1, Connector Pin Assignments

Pin	Signal	Used For
J01 pin 30	+12VF2	Serial Line Unit 7
J01 pin 31	N/C	Serial Line Unit 8/Console
J01 pin 32	GND	Serial Line Unit 8/Console
J01 pin 33	TX	Serial Line Unit 8/Console
J01 pin 34	GND	Serial Line Unit 8/Console
J01 pin 35	GND	Serial Line Unit 8/Console
J01 pin 36	N/C	Serial Line Unit 8/Console
J01 pin 37	RX-	Serial Line Unit 8/Console
J01 pin 38	RX+	Serial Line Unit 8/Console
J01 pin 39	GND	Serial Line Unit 8/Console
J01 pin 40	+12VF2	Serial Line Unit 8/Console

Table 2–6 J2, Connector Pin Assignments

pin	Signal	Used For
J02 pin 1	+5 V	+5 Volts for Console/SLU Panel
J02 pin 2	Switch 6	Remote Console/SLU Connection
J02 pin 3	Switch 7	Remote Console/SLU Connection
J02 pin 4	Switch 8	Remote Console/SLU Connection
J02 pin 5	Switch 1	Remote Console/SLU Connection
J02 pin 6	Switch 5	Remote Console/SLU Connection
J02 pin 7	Switch 2	Remote Console/SLU Connection
J02 pin 8	Switch 3	Remote Console/SLU Connection
J02 pin 9	GND	Remote Console/SLU Connection
J02 pin 10	GND	Remote Console/SLU Connection
J02 pin 11	Led 06	Remote Console/SLU Connection
J02 pin 12	Led 05	Remote Console/SLU Connection
J02 pin 13	Led 04	Remote Console/SLU Connection
J02 pin 14	GND	Remote Console/SLU Connection
J02 pin 15	Led 03	Remote Console/SLU Connection
J02 pin 16	Led 02	Remote Console/SLU Connection
J02 pin 17	Led 01	Remote Console/SLU Connection
J02 pin 18	GND	Remote Console/SLU Connection
J02 pin 19	Switch 4	Remote Console/SLU Connection
J02 pin 20	GND	Remote Console/SLU Connection
J02 pin 21	N/C	Serial Line Unit 1

Table 2–6 (Cont.) J2, Connector Pin Assignments

pin	Signal	Used For
J02 pin 22	GND	Serial Line Unit 1
J02 pin 23	TX	Serial Line Unit 1
J02 pin 24	GND	Serial Line Unit 1
J02 pin 25	GND	Serial Line Unit 1
J02 pin 26	N/C	Serial Line Unit 1
J02 pin 27	RX-	Serial Line Unit 1
J02 pin 28	RX+	Serial Line Unit 1
J02 pin 29	GND	Serial Line Unit 1
J02 pin 30	+12VF1	Serial Line Unit 1
J02 pin 31	N/C	Serial Line Unit 2
J02 pin 32	GND	Serial Line Unit 2
J02 pin 33	TX	Serial Line Unit 2
J02 pin 34	GND	Serial Line Unit 2
J02 pin 35	GND	Serial Line Unit 2
J02 pin 36	N/C	Serial Line Unit 2
J02 pin 37	RX-	Serial Line Unit 2
J02 pin 38	RX+	Serial Line Unit 2
J02 pin 39	GND	Serial Line Unit 2
J02 pin 40	+12VF1	Serial Line Unit 2
J02 pin 41	N/C	Serial Line Unit 3
J02 pin 42	GND	Serial Line Unit 3
J02 pin 43	TX	Serial Line Unit 3
J02 pin 44	GND	Serial Line Unit 3
J02 pin 45	GND	Serial Line Unit 3
J02 pin 46	N/C	Serial Line Unit 3
J02 pin 47	RX-	Serial Line Unit 3
J02 pin 48	RX+	Serial Line Unit 3
J02 pin 49	GND	Serial Line Unit 3
J02 pin 50	+12VF1	Serial Line Unit 3
J02 pin 51	N/C	Serial Line Unit 4
J02 pin 52	GND	Serial Line Unit 3
J02 pin 53	TX	Serial Line Unit 3
J02 pin 54	GND	Serial Line Unit 3
J02 pin 55	GND	Serial Line Unit 3
J02 pin 56	N/C	Serial Line Unit 3
J02 pin 57	RX-	Serial Line Unit 3

Table 2–6 (Cont.) J2, Connector Pin Assignments

pin	Signal	Used For
J02 pin 58	RX+	Serial Line Unit 3
J02 pin 59	GND	Serial Line Unit 3
J02 pin 60	+12VF1	Serial Line Unit 3

Table 2–7 J3, Connector Pin Assignments

pin	Signal	Used For
J01 pin 1	+5 V	Power
J02 pin 2	+5 V	Power
J03 pin 2	+5 V	Power
J04 pin 2	+5 V	Power

2.6 Module Installation Procedure

To install or replace the KDJ11-E module or any LSI-11 option used in the system, follow these procedures.

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

2.7 Specifications

Identification	M8981
Size	Quad
Dimensions	26.5 cm × 22.8 cm (10.5 in × 8.9 in)
Power Consumption	+5 V ±5% at 4.5 A (maximum) +12 V ±5% at 0.6 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).

3
Console On-Line Debugging Technique (ODT)

3.1 Introduction

The console on-line debugging technique (console ODT), allows the KDJ11-E to respond to commands and information entered via a console terminal connected to the module. The console interface uses addresses 17777560 through 17777566 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 (ODT) is called the microcode on-line debugging technique, or micro-ODT. The KDJ11-E 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-E provides a console serial line interface unit (SLU) on the module. This allows the console to communicate with the KDJ11-E. The console SLU 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 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.

3.2 Console ODT Entry Conditions

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

- During execution of a HALT instruction in kernel mode, provided the trap option is not selected in the maintenance register (address 17777750, bit 3). The trap option is reset by the negation of DCOK.
- During 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, 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 17777562) 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 a line feed <LF> 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 17777560 is constantly being tested for a 1 by a DATI by the processor. If bit 7 is a 0, the processor keeps testing.

3.3 Console ODT Command Set

The console ODT commands are a subset of the commands used in the ODT-11 software and the same command characters are used. ODT has 10 internal states; 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 (that is, not stripped) by the 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 (for example, <CR>) have the parity bit equal to 0. All commands are echoed except for <LF>.

The following descriptions of the console ODT commands include examples of what is displayed on the console terminal in response to the commands entered by the user.

NOTE
For the novice user, these paragraphs should be scanned first for familiarization and then reread for detail. The word "location," used in the following paragraphs, refers to a bus address, processor register, or PSW.

Table 3–1 lists the console ODT commands.

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	S	Opens the PSW; must follow an S 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

NOTE
<CTRL> J serves as line feed <LF> on some terminals.

3.3.1 Slash (/) Command (ASCII 057)

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

Example:	@00001000/012525	
Where:	@ = ODT prompt character	
	00001000	= octal location in the Q22-bus address space desired by the user (leading 0s are not required)
	/	= command to open and print contents of location
	012525	= contents of octal location 1000

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

Example:	@1000/**012525** 1234 <CR>	**@/001234<SPACE>**
Where:	first line	= new data of 1234 entered into location 1000 and location closed with <CR>.
	second line	= a / entered without a location specifier opens the previous location to reveal the new contents.

3.3.2 Carriage Return (<CR>) Command (ASCII 15)

The carriage return (<CR>) command is used to close an open location. If the contents of a location are to be changed, the user must precede the carriage return (<CR>) with the new data. If no change is desired, a carriage return (<CR>) closes the location without altering its contents.

Example:
```
@R1/004321 <CR> <LF>
@
```

Processor register R1 was opened and no change was desired, so the user issued a carriage return (<CR>).

Example:
```
@R1/004321 1234 <CR> <LF>
@
```

In this case, the user desired to change R1. The new data, 1234, was entered before the carriage return (<CR>). ODT deposited the new data into the open location.

3.3.3 Line Feed (<LF>) Command (ASCII 12)

The line feed (<LF>) 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 line feed (<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 line feed (<LF>). If no data is entered, the location is closed without being altered.

NOTE
<CTRL> J serves as line feed (<LF>) on some terminals.

Example:
```
@R2/123456 <LF>
@R3/054321
```

In this case, the user entered a line feed (<LF>) with no data preceding it. In response, ODT closed R2, then opened R3. When a user has the last register, R7, open, and issues a line feed (<LF>), ODT rolls over to the first register, R0. ODT opens location 0 if the last location in the I/O page (17777776) is open and the user issues a line feed (<LF>).

3.3.4 Internal Register Designator ($) (ASCII 044) or (R) (ASCII 122)

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

Examples:
```
@$0/000123

@R7/000123 <LF>
@R0/054321
```

If more than one character (digit or S) follows the R or dollar sign ($), ODT uses the last character as the register designator. However, an exception is that if the last three digits are either 077 or 477, ODT opens the PSW rather than R7.

3.3.5 Processor Status Word Designator (S) (ASCII 123)

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

Example: **@RS/100377 0**
@/000010

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 (for example, ODT-11) that use the T-bit for single-stepping from being accidentally harmed by the user. If the user issues a line feed (<LF>) while the PSW is open, the word is closed and ODT prints an at sign (@). No new location is opened.

3.3.6 Go (G) Command (ASCII 107)

The go (G) 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

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.

2. Load R7 (PC) with the entered data. If no data is entered, 0 is used. (In the preceding 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 micro-seconds, negating BINIT L, and then waiting for 110 micro-seconds.

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, the user issues a G and then successive P commands, all with the BHALT L bus signal asserted.

3.3.7 Proceed (P) Command (ASCII 120)

The proceed (P) command is used to resume execution of a program and it 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 reenters ODT. Upon entry, the contents of the PC (R7) are printed. In this fashion, a user can single-step through a program and get a PC trace displayed on the terminal.

3.3.8 Binary Dump (<CTRL> <SHIFT> S) Command (ASCII 23)

The binary dump (<CTRL> <SHIFT> S) 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, and prevent 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. The binary dump command is intended to more efficiently display a portion of the memory, as compared to the slash (/) and line feed (<LF>) commands.

The binary dump 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, the user should reset the terminal and type an "a" at least three times to ensure that console ODT is ready to accept commands again. The command protocol is as follows:

1. After a prompt character, ODT receives a binary dump (<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 an at sign (@).

3.4 ODT Address Specification

The KDJ11-E 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 , the user must enter 17777560, not 177560 or 777560.

3.4.1 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, 17777776. Processor registers R0 through R7 do not respond (that is, timeout occurs) to bus addresses 17777700 through 17777707 if referenced in ODT.

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

Example: **@17777572/000001**

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.

3.4.2 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>). If a program operating in kernel mode (PSW <15:14> equals 00) is halted and R6 is open, the KSP is accessed.

Similarly, if a program is operating in user mode (PSW <15:14> equals 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
```

USP has been opened.

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

@R6/123456 <CR>

@RS/000000 140000 <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.

3.4.3 Entering Octal Digits

When the user is specifying an address, the console ODT uses the last eight digits if more than eight digits have been entered. When the user is specifying data, the console ODT uses the last six octal digits if more than six were entered. The user does not need to enter leading 0s for either address or data; the console ODT forces 0s as the default. If an odd address is entered, the console ODT responds to the error by printing a question mark (?) followed by an at sign (@).

3.4.4 ODT Timeout

If the user specifies a nonexistent address or causes a parity error, ODT responds to the bus timeout by printing a question mark (?) followed by an at sign (@).

3.4.5 General Registers

Whenever R0 through R5 are referenced in the 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 <CR>
```

R4 in register set 0 has been opened.

```
@RS/000000 4000 <CR>
@R4/177777 <CR>
@RS/004000 0 <CR>
@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.

4
Boot ROMs and Diagnostics

4.1 Operation Overview

NOTE
All reference to KTJ11-B or UBA through out this chapter shall refer only to UNIBUS systems.

NOTE
All reference to UNIBUS systems shown in example printouts through out this chapter are displayed for UNIBUS systems and are omitted for Q-bus systems.

The CPU contains EEPROMs which store the programs (ROM code) that comprehensively test the CPU, UBA and memory at power-up. The ROM code also:

- Boots the user's software on various devices
- Provides memory size display
- Provides time and date from TOY clock
- Provides boot device selection
- Provides the ability to define parameters for SLUs
- Provides self-test selection
- Provides user boot area on EEPROM
- Provides support for:
  - — Hard copy terminals
  - — Video display terminals

The CPU automatically starts the ROM code each time the system is powered up or restarted with the Restart switch on the front panel. The action the ROM code takes is determined by the parameters stored in the EEPROM.

The parameters in the EEPROM determine the tests to be run, the general mode entered after testing is complete, and the final configuration of certain registers on the CPU and UBA module before the system software is started. Parameters in the EEPROM can easily be changed through a program in the ROM code called *Setup Mode* without removing the CPU or UBA modules. The EEPROM can also store customer bootstrap programs.

The ROM code runs tests selected by parameters in the EEPROM. After testing is complete, parameters in the EEPROM determine what action is to be taken next by the ROM code. Typically, the ROM code will automatically load and start a program from the user's disk or tape. This is commonly referred to as booting a program or automatic boot mode.

After the software is started, the ROM code is not entered again until the system is powered up or restarted. In some cases, after testing is complete, the ROM code enters *Dialog Mode* which allows you to select the actions entered through keyboard commands using the console terminal.

Dialog mode allows you to:

- Boot a device
- List the boot programs available
- Run ROM resident tests
- List a map of the I/O page locations
- Enter setup mode to list or change all parameters in the EEPROM
- Modify time and date in TOY clock

Section 4.2 describes hard copy terminal support.

Section 4.3 describes video terminal support.

4.2 Hard Copy Terminal Support

This section describes the hard copy ROM commands.

When dialog mode is entered, the ROM code displays the main menu at the console terminal and waits for you to select a command.

Example 4–1 shows an example of the main menu.

```
                  ❶
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
Unibus System❷
Memory    2048 KW❸
EEprom       4 KW❹
Time     15:44:37 30-May-90 Wed❺

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

Example 4–1 Main Menu

The ROM code heading contains:

❶ Version number of the ROM code and date created

❷ Indicates UNIBUS system

❸ Memory size in Kilowords

❹ EEPROM area for USERBOOT programs

❺ Time and date from TOY Clock

❻ Command line for selecting one of the seven hard copy ROM commands:

Boot	Map
Diagnostic	Setup
Help	TOY
List	

These commands are described in the following sections.

4.2.1 Boot Command

The boot command allows a device to be booted. The primary boot program normally reads 256 words from the device into memory starting at location 0. If the secondary bootstrap "block 0" is loaded without errors, then the ROM code transfers control to location 0 with the MMU off, R0 equal to the unit number of the device booted and R1 equal to the base address of the device CSR. For some devices, R1 is the base address plus an offset.

The format for the boot command is:

```
B XXN
```

Where:

B	Is the boot command.
XX	Is the two letter mnemonic representing the device to be booted. The device name must be alpha characters.
N	Is the unit number to be booted.

When the ROM code has a device name, it searches for the first boot program with the same device name. The ROM code looks for matches from these sources in the following sequence:

1. CPU EEPROM (user boot)
2. CPU EPROM
3. UBA module (UNIBUS system only)
4. M9312 module (UNIBUS system only)

There are two optional switches that can be used with the boot command:

Switch	Description
/U	Tells the ROM code to search for the boot program in the UBA ROMs first, then the M9312 (if present). This overrides the standard sequence of searching first in the EEPROM, then the CPU EPROM.
/A	Overrides the default address and allows you to enter a new address.

Table 4–1 provides examples of how the ROM code interprets user input.

Table 4–1 ROM Code Interpretation of User Input

User Input	ROM Code Action
B DL1	Boots DL1.
B DU7	Boots DU unit 7.
B B	Transfers control to an external boot module.
B/A 160100 DK0	Boots RK05 with a CSR address of 160100.
B D U 0	Invalid format. No space allowed in the device name DU.
BDU0	Invalid format. There must be a space between the boot command and the device name.

NOTE
For a complete list of all the boot programs, execute the list command from the main menu.

4.2.1.1 Transferring Control to Non-Digital Boot Modules (UNIBUS System Implementation)

The single-letter device name **B** implements a method of supporting non-Digital boot devices on the UNIBUS.

B causes the ROM code to transfer control to the address contained in location 17773024 of a ROM on the UNIBUS (if any) as long as the value in location 17773024 is not odd.

When the CPU ROM passes control:

- The CPU ROMs and the UBA ROMs are disabled.
- R0 contains a unit number.
- R1 contains 0.

The ROM code types out an *invalid device message* if:

- The address in location 17773024 on the UNIBUS is odd.
- The boot module does not respond to all addresses from 17773000 to 17773776.

Typically the single-letter device name **B** is used when you have a module which has a switch pack that responds at address 17773024 similar to a M9312 module. Usually the start address of the program desired is set in the switch pack on the module.

4.2.1.2 Transferring Control to Non-Digital Boot Modules (Q-bus System Implementation)

The single-letter device name **B** implements a method of supporting non-Digital boot devices on the Q-bus.

On Q-bus systems, the letter **B** causes ROM to disable the CPU ROM and check location 17773000 for the existence of a ROM on the Q-bus. If a ROM is present and location 17773000 of the ROM is not 0 (HALT instruction), ROM will pass control to address 17773000 with MMU off and MMR3 set to 0.

4.2.1.3 Error Detection During the Boot Command

The ROM code boot programs attempt to detect errors during the boot process and take the appropriate action. Table 4–2 lists the possible errors that the ROM code tries to detect. Not all errors are applicable for all boot programs.

Table 4–2 Boot Command Errors

LED Code	Description
21	Drive error
20	Controller error
17	Boot device selection was invalid
16	Invalid unit number selected
15	Non-existent drive
14	Non-existent controller
13	No tape
12	No disk
11	Invalid boot block
10	Drive not ready
07	No bootable device found (while in Auto Boot Mode)

NOTE

After successful completion of the loading of a secondary bootstrap, the display is set to 00. Before transferring control to the secondary boot, the ROM code prints out *Starting System*. At this time, parameters saved in the ROMs are loaded into the CPU registers.

The following is an example of a boot command:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
Unibus System
Memory     2048 KW
EEprom        4 KW
Time      15:41:52 16-May-90 Wed

Commands are: [Boot, Diagnostic, Help, List, Map, Setup, Toy]
Type a command then press the RETURN key: B DL0

   Trying
   Starting System
```

Example 4–2 Boot Command

4.2.2 Diagnostic Command

The diagnostic command allows you to test the CPU, the on-board memory SLUs, and the KTJ11-B (UNIBUS system only). Tests can be run individually or as a group (Test 30, All Selected Tests).

To execute the Diagnostic Command:

1. At the command line on the Main Menu, type D.
2. Press Return.

You are then prompted for the following information:

Prompt	Action
Test Number	Select a test from the list displayed on the terminal
	Test 30, All Selected Tests, runs all tests selected in Setup Mode Command 3. See Section 4.2.6.3 for more information.
	Test 32, Serial line Unit Loopback Test requires that loopback connectors be installed on all SLUs on the KDJ11-E.
Repeat Counter	Type the number of desired iterations in decimal, or type 0 to run the test(s) continuously.

Testing starts after you select the test number and number of iterations. The ROM code displays the test number, description of the test, error count, and iteration number.

If continuous testing is selected, no information is printed except errors. This allows the Diagnostic Command to run for extended periods of time without requiring additional printer paper.

To terminate testing, type Ctrl/C or Ctrl/P.

In the following example, test 67 is selected to be run once.

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
Unibus System
Memory    2048 KW
EEprom       4 KW
Time     15:44:37 30-May-90 Wed

Commands are: [Boot, Diagnostic, Help, List, Map, Setup, Toy]
Type a command then press the RETURN key: D

KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Single diagnostic test repeat

67 CPU Test
66 MMU Test
65 Pre-Console Test
64 MSER Test
63 CCR r/w Test
62 HIT/MISS-Reg Test
61 LTC Speed Test
60 Add-Stat-Reg Test
57 CPU-Err-Reg Test
55 UBA reg. resp. Test(1)
54 Address 0 Test
53 Pre-Memory (0-4KW) Test
52 FPA Register Test
51 FPA Function Test
50 Int Mem Address Test
47 Int Mem Data Test
46 PIRQ-Reg Test
45 LTC Int Test
44 Lines Config. Test
43 Serial Lines Test
40 Memory parity Test
37 UBA map reg Test(1)
36 UBA NPR Cycle Test(1)
32 Loopback SLU Test
31 Extended Memory Test
30 All Selected Tests

Type CTRL Z to exit

Test number    = 67                                New = 67

Type 0 for endless loop; break loop with CTRL C

Repeat counter = 000001                            New = 1

67 CPU Test  No Errors found
```

Example 4–3 Diagnostic Command

(1) Selected tests intended for UNIBUS systems only

NOTE
If the repeat counter is set to 0, to run continuously, only errors will be printed.

4.2.3 Help Command

The help command prints out a brief description of all the commands. At the end of this command, you are returned to the main menu.

To execute the help command:

1. At the command line on the main menu, type H.
2. Press Return.

The following is an example of the help command:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
Unibus system
Memory     2048 KW
EEprom        4 KW
Time      15:44:37 30-May-90 Wed

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

KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Command      Description

Help         Type this message
Boot         Load and start a program from a device
Diagnostic   Execute a self-test single or repetitive
List         List boot programs
Map          Map memory and I/O page
Setup        Enter Setup mode
Toy          Set time and date

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

Example 4–4 Help Command

4.2.4 List Command

The list command prints out a list of all available boot programs found in the CPU EPROM, the CPU EEPROM (user boot), ROM sockets on the UNIBUS adapter (if present), or an M9312 (if present).

To execute the list command:

1. At the command line on the main menu, type L.
2. Press Return.

The following is an example of the List Command (for Q-bus system):

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Memory     2048 KW
EEPROM        4 KW
Time      15:44:37 30-May-90 Wed

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

KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
❶      ❷
Device  Unit      ❸      ❹
Name    Numbers Source Device Type

AB      0       USR
TT      0       USR
DD      0-  1   ROM    TU58
DK      0-  7   ROM    RK05
MS      0-  1   ROM    TK25, TS04/05/11, TU80
NE      0- 15   ROM    DECNET DLV11-E
NF      0- 15   ROM    DECNET DLV11-F
NU      0- 15   ROM    DECNET DUV11
DU      0-255   ROM    MSCP (RAxx, RDxx, RX50, RC25, ...)
DL      0-  3   ROM    RL01/RL02
DX      0-  1   ROM    RX01
DY      0-  3   ROM    RX02
MU      0-255   ROM    TMSCP (TK50, TU81, ...)

Press RETURN key when ready to continue
```

Example 4–5 List Command (Q-bus System)

❶ *Device Name* is a two-letter mnemonic. The device name must be alpha characters. At input, ROM converts all lowercase letters to uppercase letters.

❷ *Unit Numbers* range is the allowable range of unit numbers valid for a particular boot program.

❸ *Source* lists where the actual boot program is located:

Physical Location of Boot Program	Source
CPU EPROM	ROM
ROM Sockets on the UNIBUS adapter module	UBA (UNIBUS system only)

Physical Location of Boot Program	Source
M9312	M93 (UNIBUS system only)
User Boot Area	USR

❹ *Device Type* is a description of the device to be booted.

At the completion of the list command, you are returned to the main menu.

4.2.5 Map Command

The map command prints out all addresses in the I/O page that respond. The I/O page starts at address 17760000.

In addition, all addresses that are on the CPU or on the UBA that respond, are briefly described. There is no description for optional device addresses that respond. At completion of the map command, you are returned to the main menu.

To execute the map command:

1. At the command line on the main menu, type M.
2. Press Return.

The following is an example of the Map Command:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
Unibus System
Memory     2048 KW
EEprom        4 KW
Time      15:44:37 30-May-90 Wed

Commands are: [Boot, Diagnostic, Help, List, Map, Setup, Toy]
Type a command then press the RETURN key: M

KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

I/O page Map
Starting     Ending
Address      address

17765000    17765776    CPU ROM or EEPROM
17772100                Memory CSR
17772200    17772276    Supervisor I and D PDR/PAR's
17772300    17772376    Kernel I and D PDR/PAR's
17772516                MMR3
17773000    17773776    CPU ROM
17776500    17776566    SLU's
17777200    17777376    UBA map REG's
17777520    17777526    CSR, PCR, BCR/BDR ASR
17777546                Clock CSR
17777560    17777566    Console SLU
17777572    17777576    MMR0,1,2
17777600    17777676    User I and D PDR/PAR's

Press RETURN key when ready to continue
```

Example 4–6 Map Command

4.2.6 Setup Command

The setup command has fourteen commands (Example 4–7). These commands allow you to list, change, or list and change all parameters stored in the EEPROM. Setup also allows you to create or edit USERBOOT programs stored in the EEPROM.

The EEPROM contains information needed by the ROM code to configure the KDJ11-E (CPU) and the KTJ11-B (UBA) and to determine the boot device, diagnostic test selections, and restart modes.

NOTE
Changes made under setup mode are ignored unless they are saved in EEPROM using setup mode command 8.

To execute the setup command:

1. At the command line on the main menu, type S.
2. Press Return.

The following is an example of the setup command:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
Unibus System
Memory     2048 KW
EEprom        4 KW
Time      15:44:37 30-May-90 Wed

Commands are: [Boot, Diagnostic, Help, List, Map, Setup, Toy]
Type a command then press the RETURN key: S

KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Setup Mode Commands

1   Exit
2   Select configuration parameters
3   Select diagnostic configuration
4   Select serial line parameters
5   Select boot parameters
6   List available boot programs
7   Factory setting
8   Save the setup table in the EEPROM
9   Load EEPROM data into the setup table
10  Load EEPROM boot program into memory
11  Edit or create EEPROM boot program
12  Save a boot program in the EEPROM
13  Delete a saved EEPROM boot program
14  Enter ROM ODT

Commands are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
Type a command then press the RETURN key:
```

Example 4–7 Setup Command

4.2.6.1 Setup Mode Command 1 - Exit

Setup mode command 1 exits setup mode and returns to the main menu. You can also return to the main menu by pressing Ctrl/C.

To execute the setup mode command 1:

1. At the command line on the setup menu, type 1.
2. Press Return.

The following is an example of setup mode command 1:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
Unibus System
Memory     2048 KW
EEprom        4 KW
Time      15:44:37 16-May-90 Wed

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

Example 4–8 Setup Mode Command 1

4.2.6.2 Setup Mode Command 2 - Select Configuration Parameters

The setup mode command 2 prints out the current status of various parameters and allows you to change them.

When setup mode command 2 is executed, the ROM code prints out the current status of all parameters, repeats the first parameter, then waits for your input.

There are two methods you can use to position the program at the parameter you want to change:

- Press Return until you are positioned at the parameter to be changed.
- To go directly to the parameter to be changed, enter the letter to the left of the parameter.

To change a parameter, enter the new value and press Return. The ROM code proceeds to the next parameter.

NOTE
Changes made under setup mode are ignored unless they are saved in the EEPROM using setup mode command 8 (Section 4.2.6.9).

To execute setup mode command 2:

1. At the command line on the setup menu, type 2.
2. Press Return.

The following is an example of setup mode command 2:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

A  Memory Intern                (0) =   2MB  (1) =   4MB      = 1

B  Rom on 173000                (0) =    No  (1) =   Yes      = 1

C  Rom on 165000                (0) =    No  (1) =   Yes      = 0

D  Power-up Mode                (0) = Dialog
                                (1) =   Odt
                                (2) = Trap24
                                (3) =  Auto                   = 0

E  Restart Mode                 (0) = Dialog
                                (1) =   Odt
                                (2) = Trap24
                                (3) =  Auto                   = 0

F  Power-on Self-tests          (0) =    No  (1) =   Yes      = 1

G  Alternate Boot Block         (0) =    No  (1) =   Yes      = 0

H  LTC Register                 (0) =    No  (1) =   Yes      = 1

I  Force Clock Interrupt        (0) =    No  (1) =   Yes      = 0

J  Clock Frequency              (0) =   P/S
                                (1) =  50Hz
                                (2) =  60Hz
                                (3) = 800Hz                   = 2

K  Halt on Break                (0) =    No  (1) =   Yes      = 0

L  Trap on Halt                 (0) =    No  (1) =   Yes      = 0

M  Ignore Battery               (0) =    No  (1) =   Yes      = 0

N  Lines on                     (0) =   DIS
                                (1) = 176500
                                (2) = 176600                  = 1

O  Disable UBA ROM❶             (0) =    No  (1) =   Yes      = 1

P  Enable UBA 18-Bit Mode❶      (0) =    No  (1) =   Yes      = 0

Type CTRL Z to exit or press Return key to proceed
```

Example 4–9 Setup Mode Command 2

❶ UNIBUS system only

A description of each parameter that can be changed using setup code command 2 follows:

A - Memory Intern

The KDJ11-E CPU module contains the following on-board memory:

KDJ11-EA Contains 2 Mbyte of on-board memory.

KDJ11-EB Contains 4 Mbyte of on-board memory.

If the amount of memory selected does not match the memory on-board, the message *Mem mismatch* prints. This parameter has no effect on the KDJ11-EA.

The KDJ11-EB contains 4 Mbyte of on-board memory. For some applications it may be desirable to disable the top 2 Mbyte of memory.

Table 4–3 describes the variations of the memory intern parameter.

Table 4–3 Memory Intern Parameter Variations

Setting	Description
0	2 Mbyte
1	4 Mbyte

B - ROM on 173000

This parameter sets/resets bit 7 of the CSR at address 17777520.

The KDJ11-E ROM code responds to the addresses from 17773000 through 17773777. This address range is automatically enabled at power up or after the restart switch is enabled regardless of the setting of this parameter.

Table 4–4 describes the variations of the ROM on 173000 parameter.

Table 4–4 ROM on 173000 Parameter Variations

Value	Description
0	KDJ11-E ROM code disabled.
1	KDJ11-E responds to addresses 17773000 - 17773777. **Factory setting.**

C - ROM on 165000

This parameter allows you to disable the ROM. It sets/resets bit 6 of the CSR at address 17777520.

Table 4–5 describes the variations of this parameter.

Table 4–5 ROM on 165000 Parameter Variations

Value	Description
0	The KDJ11-E does not respond to addresses 17765000 - 17765777. **Factory setting.**
1	Enables internal EEPROM or the boot EPROM to respond to addresses 17765000 - 17765777.

D - Power-up Mode and E - Restart Mode

There are four mode choices for either the power-up mode or the restart mode:

- Dialog mode
- ODT mode
- Trap 24 mode
- Automatic boot mode

When the ROM code is started, it checks a status bit to determine if the unit is powering up or if the restart switch was activated. The ROM code then uses the appropriate mode selected. You can define the action taken by the ROM code at power-up or restart to be the same or different.

- Dialog mode

  In dialog mode, all selected tests are executed at power-up unless power-up self-tests are disabled. Dialog mode allows you to:

  - Boot a device
  - List boot programs available
  - Run ROM resident tests
  - Print a map of all I/O page locations
  - Enter setup mode to list and change all parameters in the EEPROM.
  - Enter time and date for TOY clock.

- ODT mode

  At completion of a very limited set of tests, the ROM code executes a halt instruction and passes control to J11 micro-ODT. This mode is used in debug environments. The ROM code does not change any locations in memory before entering ODT mode (Chapter 3).

- Trap 24 Mode

  At the completion of a limited set of tests, the ROM code loads the PSW with the contents of location 26 and then transfers control to the address in location 24. This mode is used when power fail recovery is desired.

- Automatic boot mode

  In automatic boot mode, all selected tests are executed at power-up unless power-up self-tests are disabled.

  NOTE
  If the force dialog switch (S5 on the KDJ11-E CPU module) is on, the ROM code enters dialog Mode at power-up or restart regardless of the selections in the EEPROM.

  The ROM code enters an automatic boot routine that tries to boot a previously selected device or devices. The list of devices can be from one to six devices long. Each device is tried sequentially until a successful boot occurs or the end of the boot table is reached.

  ROM mode is a special automatic boot mode that is entered as a power-up/restart option to boot specific devices when one or more of switches 2 through 4 on the KDJ11-E CPU module are set.

  When switches 2 through 4 of the KDJ11-E CPU module are set to one of the six combinations shown in Table 4–6 and force dialog mode is not selected, ROM mode is entered. The mode attempts to boot only the one device selected by this command. If the boot is unsuccessful, the ROM code prints out the normal error message and enters dialog mode.

Table 4–6 ROM Mode Switch Settings

Switches	Settings	Description
2 3 4	off off off	Normal Automatic Boot Mode. (Does not enter ROM mode.)
2 3 4	on on off	Device 1 in list/change boot parameters.
2 3 4	on off on	Device 2 in list/change boot parameters.
2 3 4	on off off	Device 3 in list/change boot parameters.
2 3 4	off on on	Device 4 in list/change boot parameters.
2 3 4	off on off	Device 5 in list/change boot parameters.
2 3 4	off off on	Device 6 in list/change boot parameters.

NOTE
See Section 4.2.6.5 to add or delete devices from the boot device block.

F - Power-on Self-tests

After a power-up sequence, the diagnostic tests contained in the ROM code are executed. Control is passed to the ROM code and a comprehensive set of diagnostic tests check the KDJ11-E and UNIBUS adapter. Upon completion of the on-board diagnostics, control is passed to the previously selected power-up mode option.

NOTE
The force dialog switch (S5) on the KDJ11-E CPU module must be off.

Table 4–7 describes the variations of the power-on self-tests parameter.

Table 4–7 Power-On Self-Tests Parameter Variations

Value	Description
0	No self-tests performed after a power-up.
1	Self-tests performed after a power-up. **Factory setting.**

NOTE
Power-on self-tests are not executed if:

- **If power-up mode is set to Trap 24.**
- **If power-up mode is set to ODT.**
- **If ROM code is entered by depressing the RESTART switch on the front panel.**

G - Alternate Boot Block

When you attempt to boot a device, the ROM code does not transfer control to the booted device unless the device looks *bootable*.

Some *poking around* is done by the ROM code to ensure control is never passed to unbootable media. For example, a blank pack may have been mounted in a disk drive by mistake.

The boot block on all bootable PDP–11 software distributed by Digital Equipment Corporation has the following format:

- Location 0 can range from 240 to 277. Normally this location contains a no operation command (240).
- Location 2 can range from 400 to 777. This is an unconditional branch instruction.

Table 4–8 describes the variations of the alternate boot block parameter.

Table 4–8 Alternate Boot Block Parameter Variations

Value	Description
0	Standard format bootblock. **Factory setting.** If this parameter is set to 0, the ROM code looks for memory location 0 to be a value of 240 to 277 and for memory location 2 to be 400 to 777. If the data found is within those ranges, the ROM code assumes that bootable media is present. At this point, the ROM code passes control to the secondary boot program starting at memory address 0. If the data in memory location 0 or 2 is not within the range specified above, the ROM code will type out an error message indicating that the media is not bootable. (Boot block error)
1	Nonstandard format for the boot block. No poking around is done by the ROM code to determine if the media is bootable. All the ROM code does is verify that memory location 0 does not contain a halt instruction (000000). When this parameter is set to 1 the ROM code looks for location 0 of the boot block to be any nonzero number. If a nonzero number is found, control is passed unconditionally to the secondary boot starting at memory location 0. This allows you to boot media that is in nonstandard format.

NOTE
The ROM code checks location 0 or 2 after the boot block (normally block 0) is loaded into memory.

NOTE
USERBOOT programs are not checked to see if they are bootable.

H - LTC Register

Table 4–9 describes the variations of the LTC register parameter.

Table 4–9 LTC Register Parameter Variations

Value	Description
0	Disables the KDJ11-E line time clock register to allow you to place a customer time clock at this address.
1	Enables the KDJ11-E line time clock register. **Factory setting.**

I - Force Clock Interrupt

Table 4–10 describes the variations of the force clock interrupt parameter.

Table 4–10 Force Clock Interrupt Parameter Variations

Value	Description
0	**Factory setting.** The clock can request interrupts only if: • The clock CSR is enabled, parameter H LTC Register = 1 • The interrupt enable bit, CSR bit 6, is set at address 17777546. • The processor priority is 5 or less.
1	The clock unconditionally requests interrupts when the processor priority is 5 or less.

NOTE
If the force clock interrupt is selected, always disable the clock CSR, because the CSR has no control over the clock.

J - Clock Frequency

The clock frequency parameter determines the source of the clock to be used.

Table 4–11 describes the variations of this parameter.

Table 4–11 Clock Frequency Parameter Variations

Value	Source
0	Clock sourced from power supply, at backplane pin BR1. The power supply drives this signal at 50 or 60 Hz. **Factory setting.**
1	Clock sourced internally at 50 Hz from KDJ11-E.
2	Clock sourced internally at 60 Hz from KDJ11-E.
3	Clock sourced internally at 800 Hz from KDJ11-E.

K - Halt-on-Break

Table 4–12 describes the variations of the halt-on-break parameter.

Table 4–12 Halt-on-Break Parameter Variations

Value	Description
0	Console breaks are ignored from the Break key on terminal. **Factory setting.**
1	Enables the processor to halt if the console SLU detects a break condition from the Break key on terminal.

L - Trap on Halt

Table 4–13 describes the variations of the trap-on-halt parameter.

Table 4–13 Trap-on-Halt Parameter Variations

Value	Description
0	**Factory setting.** The processor enters J11 micro-ODT if a halt instruction is executed in kernel mode.
1	The processor traps to location 4 if a halt instruction is executed in kernel mode.

M - Ignore Battery

The ignore battery parameter is used only when the current power up or restart mode is set to 24 (3).

Table 4–14 describes the variations of this parameter.

Table 4–14 Ignore Battery Parameter Variations

Value	Description
0	**Factory setting.** The battery OK signal (BOK) must be present to execute Trap 24. Battery OK indicates that the memory contents were not corrupted as a result of a power failure. If BOK is not set, Trap 24 mode is not executed and dialog mode is entered.
1	Trap 24 mode is executed regardless of the status of the battery OK bit. This mode is used if you have custom battery back-up hardware. In this case, the battery OK bit may not reflect the actual state of the memory on the KDJ11-E.

N - Lines On

The lines on parameter allows you to select the starting address/vector of serial lines 1 through 7. The priority level is set to 4 and cannot be changed.

To select other parameters for SLUs 1 through 7 (such as baud rate, number of data bits, stop bits or parity), use setup mode command 4 (Section 4.2.6.4).

Table 4–15 describes the variations of the lines on parameter.

NOTE
After selecting and saving the desired starting address/vector for this parameter, the system must be powered down and then rebooted for the change to occur.

Table 4–15 Lines On Parameter Variations

Value	Description
0	Disables SLUs 1 through 7. SLUs do not respond to any address and cannot be accessed.
1	SLUs respond to the following addresses/vectors:

UNIBUS Address	Vector	SLU Number
176500	300	1
176510	310	2
176520	320	3
176530	330	4
176540	340	5
176550	350	6
176560	360	7
176570	*370*	*8*
		Line 8 is included in the list **only** if the KDJ11-E is disabled. The KDJ11-E is disabled if switch 1, disable console serial line unit, located on the KDJ11-E is set to the on position. In this case a console SLU, a DL11 for example, must be provided by the user.

Value	Description
2	SLUs respond to the following addresses/vectors:

UNIBUS Address	Vector	SLU Number
176600	400	1
176610	410	2
176620	420	3
176630	430	4
176640	440	5
176650	450	6
176660	460	7
176670	*470*	*8*
		Line 8 is included in the list **only** if the console/SLU is disabled. The console/SLU is disabled if switch 1, disable console serial line unit, located on the KDJ11-E, is set to the on position. In this case a console SLU, a DL11 for example, must be provided by the user.

O - Disable UBA ROM (UNIBUS Systems Only)

Table 4–16 describes the variations of the disable UBA ROM parameter.

Table 4–16 Disable UBA ROM Parameter Variations

Value	Description
0	Enables the UBA ROMs. **Factory setting.**
1	Disables the UBA ROMs. This allows other ROM boards on the UNIBUS to show up in the UBA ROM address range of 17773000 to 17773776.

NOTE

If the ROM code is booting directly from an M9312 type boot ROM located on the M9312 module, the ROM code automatically disables the CPU ROM in the 17773nnn address range and the ROMs on the UBA module and uses the ROMs on the M9312 module. This action is taken regardless of the status of the disable UBA ROM parameter and the disable ROM parameter.

P - Enable UBA 18-Bit Mode (UNIBUS system Only)

Table 4–17 describes the variations of the enable UBA 18-bit mode parameter.

Table 4–17 Enable UBA 18-Bit Mode Parameter Variations

Value	Description
0	Selects 22-bit addressing. **Factory setting.**
1	Selects 18-bit addressing.

4.2.6.3 Setup Mode Command 3 - Select Diagnostic Configuration

This command allows you to select or deselect individual tests (Example 4–10). The tests selected here are run when power-on self-tests (Section 4.2.6.2) and the diagnostic command (Section 4.2.2) are enabled.

NOTE

To run the diagnostic command, changes to the self-test menu do not need to be saved.

To execute setup mode command 3:

1. At the command line on the Setup Menu, type 3.
2. Press Return.

The following is an example of Setup Mode Command 3:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
A  Nr. 67 CPU Test                  (0) =   No  (1) =  Yes     = 1
B  Nr. 66 MMU Test                  (0) =   No  (1) =  Yes     = 1
C  Nr. 65 Pre-Console Test          (0) =   No  (1) =  Yes     = 1
D  Nr. 64 MSER Test                 (0) =   No  (1) =  Yes     = 1
E  Nr. 63 CCR r/w Test              (0) =   No  (1) =  Yes     = 1
F  Nr. 62 HIT/MISS-Reg Test         (0) =   No  (1) =  Yes     = 1
G  Nr. 61 LTC Speed Test            (0) =   No  (1) =  Yes     = 1
H  Nr. 60 Add-Stat-Reg Test         (0) =   No  (1) =  Yes     = 1
I  Nr. 57 CPU-Err-Reg Test          (0) =   No  (1) =  Yes     = 1
J  Nr. 55 UBA reg. resp. Test❶      (0) =   No  (1) =  Yes     = 1
K  Nr. 54 Address 0 Test            (0) =   No  (1) =  Yes     = 1
L  Nr. 53 Pre-Memory (0-4KW) Test   (0) =   No  (1) =  Yes     = 1
M  Nr. 52 FPA Register Test         (0) =   No  (1) =  Yes     = 1
N  Nr. 51 FPA Function Test         (0) =   No  (1) =  Yes     = 1
O  Nr. 50 Int Mem Address Test      (0) =   No  (1) =  Yes     = 1
P  Nr. 47 Int Mem Data Test         (0) =   No  (1) =  Yes     = 1
Q  Nr. 46 PIRQ-Reg Test             (0) =   No  (1) =  Yes     = 1
R  Nr. 45 LTC Int Test              (0) =   No  (1) =  Yes     = 1
S  Nr. 44 Lines Config. Test        (0) =   No  (1) =  Yes     = 1
T  Nr. 43 Serial Lines Test         (0) =   No  (1) =  Yes     = 1
U  Nr. 40 Memory parity Test        (0) =   No  (1) =  Yes     = 1
V  Nr. 37 UBA map reg Test❶         (0) =   No  (1) =  Yes     = 1
W  Nr. 36 UBA NPR Cycle Test        (0) =   No  (1) =  Yes     = 1
X  Nr. 32 Loopback SLU Test         (0) =   No  (1) =  Yes     = 0
Y  Nr. 31 Extended Memory Test      (0) =   No  (1) =  Yes     = 0
Z  Nr. 30 All Selected Tests        (0) =   No  (1) =  Yes     = 0
Type CTRL Z to exit or press Return key to proceed
```

Example 4–10 Setup Mode Command 3 - Select Diagnostic Configuration

❶ UNIBUS system only

4.2.6.4 Setup Mode Command 4 - Select Serial Line Parameters

This command prints out the current status of selectable parameters for SLUs 1 through 7 and then allows you to change them if desired.

The following parameters can be modified for SLUs 1 through 7:

Parameter	Values
Baud rate	300, 600, 1200, 2400, 4800, 9600, 19,200, 38,400
Number of data bits	7 or 8
Number of stop bits	1 or 2
Parity	Even, odd, or disabled

NOTE

The following parameters are fixed for the console/SLU:

Baud rate	Set by using switches 6, 7, 8 on the KDJ11-E.
Number of data bits	8
Number of stop bits	1
Parity	None

To execute the setup mode command 4:

1. At the command line on the setup menu, type 4.
2. Press Return.

The following is an example of Setup Mode Command 4:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

A  Line 1  Baudrate              (0) =   300
                                 (1) =   600
                                 (2) =  1200
                                 (3) =  2400
                                 (4) =  4800
                                 (5) =  9600
                                 (6) = 19200
                                 (7) = 38400                    = 5

   Line 1  Data bits             (0) =  8      (1) =  7         = 0

   Line 1  Stop bits             (0) =  2      (1) =  1         = 1

   Line 1  Parity                (0) = Even
                                 (1) =  Odd
                                 (2) =  Dis                     = 2
.
.
.
.
.
G  Line 7  Baudrate              (0) =   300
                                 (1) =   600
                                 (2) =  1200
                                 (3) =  2400
                                 (4) =  4800
                                 (5) =  9600
                                 (6) = 19200
                                 (7) = 38400                    = 5

   Line 7  Data bits             (0) =  8      (1) =  7         = 0

   Line 7  Stop bits             (0) =  2      (1) =  1         = 1

   Line 7  Parity                (0) = Even
                                 (1) =  Odd
                                 (2) =  Dis                     = 2

Type <CTRL> Z to exit or press Return key to proceed
```

Example 4–11 Setup Mode Command 4

3. If you do not want to change any parameters, enter Ctrl/Z to return to the setup menu.
4. If you want to change parameters, there are two methods you can use to position the program at the parameter you want to change:
   a. Press Return until positioned at the parameter to be changed.
   b. Type one of the following letters:
      A To change parameters for line 1
      B To change parameters for line 2
      C To change parameters for line 3
      D To change parameters for line 4
      E To change parameters for line 5
      F To change parameters for line 6
      G To change parameters for line 7
5. Press Return. The ROM code proceeds to the next parameter.

4.2.6.5 Setup Mode Command 5 - Select Boot Parameters

This command allows you to list current boot parameters for the power-up/restart mode of Autoboot and to modify the Autoboot table.

To execute the setup mode command 5:

1. At the command line on the setup menu, type 5.
2. Press Return.

The following is an example of setup mode command 5:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

List/change boot parameter
                                          Device Unit Address
Boot 1: Switches 2,3,4 on  on  off  = DM ROM   0  Default
Boot 2: Switches 2,3,4 on  off on   = DL ROM   0  Default
Boot 3: Switches 2,3,4 on  off off  = TT USR   0  000000
Boot 4: Switches 2,3,4 off on  on   = DL UBA   0  Default
Boot 5: Switches 2,3,4 off on  off  = blank
Boot 6: Switches 2,3,4 off off on   = blank

Type CTRL Z to exit or press Return key to proceed
```

Example 4–12 Setup Mode Command 5 - Boot Parameters Menu

3. If you want to return to the setup menu, enter Ctrl/Z.
4. To proceed, press Return.

The following is an example:

```
Boot 1: Switches 2,3,4 on  on  off  = DM ROM   0  Default
❶Device name    =
❷Boot location  (0) = USR  (1) = ROM  (2) = UBA(UNIBUS system only)
  (3) = M93(UNIBUS system only)  = 1
❸Unit number    = 0
❹Address        =

Boot 2: Switches 2,3,4 on  off on   = DL ROM   0  Default
Device name    =
```

❶ Device Name

The ROM code prompts you for a device name. The previous mnemonic is displayed. You can change the device by typing a new two-letter mnemonic associated with the device to be selected. Pressing Return leaves the parameter unchanged.

Table 4–18 lists the default boot programs in the CPU ROM. Your system may contain boot programs that are not on this list.

Table 4–18 Default Boot Programs

Device Name	Unit Numbers	Source	Device Type
DU	0-255	ROM	MSCP (RAxx, RDxx, RX50, RC25, ...)
DL	0-3	ROM	RL01/RL02
DX	0-1	ROM	RX01
DY	0-3	ROM	RX02
MS	0-1	ROM	TK25, TS04/05/11, TU80 (Q-bus only)
MT	0-1	ROM	TU10, TE10, TS03 (UNIBUS only)
MU	0-255	ROM	TMSCP (TK50, TU81, ...)
NE	0-15	ROM	DECNET DLV11-E (Q-bus only)
NF	0-15	ROM	DECNET DLV11-F (Q-bus only)
NU	0-15	ROM	DECNET DUV11 (Q-bus only)
XH	0	ROM	ETHERNET (Q-bus only)

NOTE
To print a complete list of all available bootstraps and associated mnemonics, use the list command.

❷ Boot Location

The ROM code requires you to enter the location of the boot code. The default is the the standard on-board ROM.

Where:

USR	User previously created a User Boot. Boot code is located in the User Boot area.
ROM	Standard default area.
UBA[1]	M9312-type ROMs located in the ROM sockets of the UNIBUS Adapter Module.
M93[1]	M9312-type ROMs located in the ROM sockets of the optional M9312 module on the UNIBUS.

[1]UNIBUS system only

❸ Unit Number

The ROM code prompts for the unit number. Type in the unit number of the device to be booted.

❹ Address

To select the default address press Return.

Enter an address if the device CSR is set at a nonstandard or floating address.

The ROM code continues to prompt for all six boot entries shown in Example 4–12.

If you do not want to change any items in Example 4–12, press Ctrl/Z to return to the setup command menu. You can also skip any entry by pressing Return for each entry you want to skip.

The ROM code continues to prompt for all six entries in Example 4–12.

Autoboot Mode

Autoboot mode allows you to select the devices to be tried in the automatic boot sequence. You can create a list that defines the devices and the order in which they are tried. One entry is needed to define a device and its unit number. If the same device is used more than once with different unit numbers, then one entry is needed for each unit number. The ROM code attempts to boot the devices you have defined in Example 4–12, starting with boot 1. If the autoboot is unsuccessful, an error message is printed and the ROM code enters dialog mode.

ROM Mode

ROM Mode is entered when Autoboot is selected as a power-up/restart option and one or more of switches 2 through 4 of the KDJ11-E CPU module are set.

When switches 2 through 4 of the KDJ11-E are set to one of the six combinations shown in Table 4–19 and force dialog mode is not selected, the ROM code enters a special autoboot mode called ROM mode. ROM mode attempts to boot only the one device selected by this command. If the boot is unsuccessful, the ROM code prints out the normal error message and enters dialog mode.

Table 4–19 ROM Mode

Switches	Settings	Description
2 3 4	off off off	Normal Automatic Boot Mode.
2 3 4	on on off	Device 1 in list/change boot parameters.
2 3 4	on off on	Device 2 in list/change boot parameters.
2 3 4	on off off	Device 3 in list/change boot parameters.
2 3 4	off on on	Device 4 in list/change boot parameters.
2 3 4	off on off	Device 5 in list/change boot parameters.
2 3 4	off off on	Device 6 in list/change boot parameters.
2 3 4	on on on	Normal Automatic Boot Mode

4.2.6.6 Setup Mode Command 6 - List Available Boot Programs (for UNIBUS System)

To execute the setup mode command 6:

1. At the command line on the setup menu, type 6.
2. Press Return.

The following is an example of setup mode command 6: (UNIBUS System)

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
❶       ❷
Device  Unit      ❸      ❹
Name    Numbers Source Device Type

AB      0       USR
TT      0       USR
DU      0-255   ROM    MSCP (RAxx, RDxx, RX50, RC25, ...)
DL      0-  3   ROM    RL01/RL02
DX      0-  1   ROM    RX01
DY      0-  3   ROM    RX02
MT      0-  1   ROM    TU10, TE10, TS03
MU      0-255   ROM    TMSCP (TK50, TU81, ...)
DL      0-  3   UBA    RL01/RL02

Press RETURN key when ready to continue
```

Example 4–13 Setup Mode Command 6 (UNIBUS System)

❶ *Device name* is a two-letter mnemonic. The device name must be alpha characters. At input, the ROM code converts all lowercase letters to uppercase letters.

❷ *Unit numbers* is the allowable range of unit numbers that is valid for a particular boot program. The range varies from 0 to 255, depending on the device. If the unit numbers range information is blank, the ROM code assumes the range limit is 0 to 255.

❸ *Source* lists where the actual boot program is located:

Physical Location of Boot ROM	Source
User boot area	USR
CPU EPROM	ROM
ROM sockets on the UNIBUS adapter module[1]	UBA
M9312[1]	M93

[1]UNIBUS system only

❹ *Device Type* is a description of the device to be booted. It is the name on the outside of the device to be booted. For example, the description for a device name DL is RL02, which is the name on the outside of the physical device.

The mnemonic for each ROM found on either the UBA or the M9312 is checked against the list of mnemonics in the ROM code. If the mnemonic matches an item in this list, the ROM code prints out a description of that device. If no match is found, the description is left blank for that mnemonic.

4.2.6.7 Setup Mode Command 6 — List Available Boot Programs (Q-bus System)

To execute the setup mode command 6:

1. At the command line on the setup menu, type 6.
2. Press Return.

The following is an example of setup mode command 6:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
❶       ❷
Device  Unit     ❸      ❹
Name    Numbers Source Device Type

AB      0       USR
TT      0       USR
DD      0- 1    ROM    TU58
DK      0- 7    ROM    RK05
MS      0- 1    ROM    TK25, TS04/05/11, TU80
NE      0- 15   ROM    DECNET DLV11-E
NF      0- 15   ROM    DECNET DLV11-F
NU      0- 15   ROM    DECNET DUV11
DU      0-255   ROM    MSCP (RAxx, RDxx, RX50, RC25, ...)
DL      0- 3    ROM    RL01/RL02
DX      0- 1    ROM    RX01
DY      0- 3    ROM    RX02
MU      0-255   ROM    TMSCP (TK50, TU81, ...)

Press RETURN key when ready to continue
```

Example 4–14 Setup Mode Command 6 (Q-bus System)

❶ *Device Name* is a two-letter mnemonic. The device name must be alpha characters. At input, ROM converts all lowercase letters to uppercase letters.

❷ *Unit Numbers* is the allowable range of unit numbers that is valid for a particular boot program. The range varies from 0 to 255, depending on the device. If the unit numbers range information is blank, ROM assumes the range limit is 0 to 255.

❸ *Source* lists where the actual boot program is located:

Physical Location of Boot ROM	Source
CPU EPROM	ROM
USERBOOT area	USR

❹ *Device Type* is a description of the device to be booted. It is the name on the outside of the device to be booted. For example, the description for a device name *DL* is RL02, which is the name on the outside of the physical device.

4.2.6.8 Setup Mode Command 7 - Factory Setting

This command initializes the current contents of the values in the setup menu table to the default factory settings.

To execute the setup mode command 7:

1. At the command line on the setup menu, type 7.
2. Press Return.

The following is an example of setup mode command 7:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Factory setting
```

Example 4–15 Setup Mode Command 7

3. Execute setup mode command 8, if you want to save the values in the setup table into the EEPROM.

4.2.6.9 Setup Mode Command 8 - Save the Setup Table in the EEPROM

NOTE
All changes made in setup mode will be lost if setup mode command 8 is executed.

This command permanently saves the current parameters of the setup table in the EEPROM.

To execute the setup mode command 8:

1. At the command line on the setup menu, type 8.
2. Press Return.

The following is an example of setup mode command 8:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Saving setup table in the EEPROM, please wait
```

Example 4–16 Setup Mode Command 8

4.2.6.10 Setup Mode Command 9 - Load EEPROM Data into the Setup Table

This command restores the setup table in memory with the values stored in the EEPROM during the last save. You can also restore the setup table after making temporary changes.

To execute the setup mode command 9:

1. At the command line on the setup menu, type 9.
2. Press Return.

The following is an example of setup mode command 9:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Load EEPROM data into the setup table
```

Example 4–17 Setup Mode Command 9

4.2.6.11 Setup Mode Command 10 - Load EEPROM Boot Program Into Memory

When this command is executed, the ROM code loads a previously created user boot program. The ROM code asks for the device name of an EEPROM boot to be loaded in memory.

To execute setup mode command 10:

1. At the command line on the setup menu, type 10.
2. Press Return.
3. Enter the two-letter mnemonic of the boot program to load:

```
Device name        =          New = TT
```

The following is an example of setup mode command 10:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Loading EEprom boot program

Type CTRL Z to exit or press Return key to proceed
```

Example 4–18 Setup Mode Command 10

NOTE
Setup mode command 10 only loads the program. To examine or edit the userboot program, use setup mode command 11.

4.2.6.12 Setup Mode Command 11 - Edit or Create EEPROM Boot Program

This command is used to either create a new EEPROM boot program or to edit a program previously loaded using setup mode command 10.

You can change or enter the:

- Device name
- Beginning address of the user boot program
- Ending address of the user boot program
- Start address
- Highest unit number
- Device description

When these changes are complete, the ROM code enters ROM ODT which is a ROM code version of J11 micro-ODT. When this command is first entered, it lists the available space in the EEPROM for boots.

CAUTION
To ensure the new or edited user boot program is not lost, you must save it using setup mode command 12.

To execute setup mode command 11:

1. At the command line on the setup menu, type 11.
2. Press Return.

The following is an example of setup mode command 11:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

User boot editor

Type CTRL Z to exit or press Return key to proceed

11630 Bytes free in the EEPROM

❶Device name                                    = TT            New =

❷Beginning address                              = 01000         New =

❸Last byte address                              = 01010         New =

❹Start address                                  = 01000         New =

❺Highest unit number                            = 000           New =

❻Device description                             =               New = test

❼Enter ROM ODT

xxxxxx/ = open word location xxxxxx if address even; byte if odd
RETURN  = close location
. or LF = close location and open next
  -     = close location and open previous

ROM ODT >
```

Example 4–19 Setup Mode Command 11

❶ *Device name* is a two-letter mnemonic for the boot program to be created.

❷ *Beginning address* is the first location of the program in memory. The address range is 1000 to 17544.

The ROM code prints out the old starting address and prompts you for a new starting address. Type in a new address or press Return to accept the old starting address.

❸ *Last byte address* is the address of the last byte of code used in memory. If in doubt, use the last address of data + 2 for this value. Do not use a much larger number to avoid wasting EEPROM space.

The ROM code prints out the old ending address and prompts you for a new ending address. Type in a new address or press Return to accept the old ending address.

❹ *Start address* is the address that the ROM code passes control to. The start address does not have to be the same as the beginning address but it must be even and a value in the range defined by the beginning and ending addresses.

❺ *Highest unit number* defines the allowable range of valid unit numbers for this device. If the value is set to 3, the allowable range is 0 to 3. If a unit number is typed in at boot time and it is not in range, an invalid unit number error occurs.

❻ *Device description* is an optional but recommended description of the device name. The name should be the name that is physically marked on the outside of the device (for example, RA82).

NOTE
After entering the device description, ROM ODT is automatically entered.

❼ *Enter ROM ODT* - At this time, a new program can be entered or an existing program can be edited/examined using ROM ODT. ROM ODT uses the commands in Table 4–20.

CAUTION
After creating or editing a user boot program, use setup mode command 12 to save it. If you do not save the program, it is lost.

To exit ROM ODT mode, press Ctrl/Z.

Table 4–20 ROM ODT Commands

Command	Symbol	Description
Slash	/	Prints contents of specified location or if no address is defined, then the contents of the last location that was opened prints. If the location opened is an odd number, then only the contents of the byte prints. If location is even, the mode is word. If location is odd, the mode is byte. Leading zero's are assumed. Only bits 15 through zero of the address are used.
Return	<CR>	Closes an open location.
Line Feed	<LF>	Closes an open location and opens the next location. If the mode is word, the address is incremented by 2. If the mode is bytes, the address is incremented by 1.
Period	.	Alternate character for line feed. This command is useful when the terminal is a VT2xx series terminal. It is also convenient to use with the keypad.
Minus		Alternate character for the up arrow on the cursor control keypad. This command is useful when the terminal is a VT2xx series terminal. It is also convenient to use with the keypad.
Delete	DELETE	Deletes the previously typed character.
CTRL Z	^Z	Exits ROM ODT and returns to Setup mode.

4.2.6.13 Setup Mode Command 12 - Save a Boot Program in the EEPROM

This command allows you to save the user boot program created or edited in setup mode command 11 into the EEPROM. This is the only command that actually writes a boot into the EEPROM.

When saving a boot program into memory, the device name of the program **must not** match the name of an existing program in the EEPROM. If the program name already exists, you must delete that program first or change the name of the program to be saved.

NOTE
The save procedure can take up to two minutes at the full length of 8 Kbytes.

To execute the setup mode command 12:

1. At the command line on the setup menu, type 12.
2. Press Return.

Example 4–20 shows the setup mode command 12:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Saving boot program

Are you sure  Y/N ? Y

Saving Boot,wait-
```

Example 4–20 Setup Mode Command 12 - Save a Boot Program in the EEPROM

4.2.6.14 Setup Mode Command 13 - Delete a Saved EEPROM Boot Program

This command allows you to delete an EEPROM boot. If this command is executed the ROM code asks for the device name of the EEPROM boot to be deleted. After the device name is input, the ROM code looks for the first boot program in the EEPROM. If it finds the boot program, the ROM code deletes it.

To execute setup mode command 13:

1. At the command line on the setup menu, type 13.
2. Press Return.

   The following is an example of setup mode command 13:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Del saved boot

Type CTRL Z to exit or press Return key to proceed

Device name       = TT        New = JP
```

Example 4–21 Setup Mode Command 13

3. Press Return. The following is displayed:

```
Are you sure?  Y/N?
```

4.2.6.15 Setup Mode Command 14 - Enter ROM ODT

This command enters ROM ODT. The ROM code opens up the address defined by the beginning address of the program. ROM ODT is not the same as J11 micro-ODT. The only purpose of ROM ODT is to allow the user to create or edit a small bootstrap program to be stored in the EEPROM.

In ROM ODT, the only allowable addresses that can be examined are the addresses of memory from 0-28 KW (0-00157776). Any other addresses and any attempt to access the I/O page or any registers are not allowed.

Rom ODT uses the commands listed in Table 4–20.

To execute setup mode command 14:

1. At the command line on the setup menu, type 14.
2. Press Return.

The following is an example of setup mode command 14:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Enter ROM ODT

xxxxxx/ = open word location xxxxxx if address even; byte if odd
RETURN  = close location
. or LF = close location and open next
  -     = close location and open previous

ROM ODT >
```

Example 4–22 Setup Mode Command 14

3. To exit ROM ODT mode and return to the setup menu, press Ctrl/Z.

4.2.7 TOY Command

The TOY command allows you to change the time and date of the TOY clock. The time is in 24-hour format.

To execute the TOY command:

1. Enter T.
2. Press Return.

The following is an example of the TOY command:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
UNIBUS System
Memory    2048 KW
EEprom       4 KW
Time     15:44:37 30-May-90 Wed

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

KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Set time and date

Use following format:
For time: HH:MM:SS         For date: DD-MMM-YY

Time:  15:45:20
Input:
Date:  16-May-90
Input:

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

Example 4–23 TOY Command

4.3 Video Terminal Support

This section describes how to operate the KDJ11-E ROM code using video terminal support.

4.3.1 Moving Through Menus

Table 4–21 describes how to move through or execute instructions in the menus.

Table 4–21 Moving Through Menus

Key	Function
Return	Executes a Do or an Edit.
Space bar	Selects the next value for a parameter.
Backspace	Selects the previous value for a parameter.
Tab	In setup menu, moves the cursor to the Save parameter. To save changes, press Return.
	In the self-test Menu, moves the cursor to the Monitor parameter. To return to setup menu, press Return.
	In the user boot menu, moves the cursor to the Exit parameter. To return to the setup menu, press Return.
	In the Map Menu, returns you to the setup menu.
↑	Moves the cursor up to the next parameter.
↓	Moves the cursor down to the next parameter.
→	Moves the cursor to the next parameter on the right.
←	Moves the cursor to the next parameter on the left.
H	Displays a list of all boot programs while in the boot device block.
Ctrl/C or Ctrl/P	Ends self-test and returns you to the setup menu.

4.3.2 Types of Function Fields - Video Terminal

Each menu contains function fields which allow you to perform a specific operation. There are three types of function fields which are described in Table 4–22.

Table 4–22 Types of Function Fields

Field	Function
Edit	This is an executable function. When entered, Edit brings you to a submenu such as Self-test, User Boot, or Map. You can make modifications through the submenu. Not all fields can be edited.
Do	This is an executable function. When selected, Do executes a specific operation after pressing Return.
Addresses	Allows you to enter addresses directly in an octal format.

4.3.3 Setup Menu

The setup menu lists all the parameters in the EEPROM including boot parameters. Use the setup menu to modify these parameters. Example 4–24 shows a setup menu.

```
                                                                    ❹
KDJ11-E Monitor  Version 1.06 30-Jul-1990      Disable UBA ROM[1]          No
(C) Digital Equipment Corporation 1990         Enable UBA 18-Bit Mode[1]   No
UNIBUS System                                  Memory Intern               4MB
Memory     2044 KW                             Rom on 173000               Yes
EEprom        4 KW                             Rom on 165000               No
❶Time      16:17:45 18-May-90 Fri              Power-up Mode               Dialog
                                               Restart Mode                Dialog
❷Nr Device Unit Address                        Power-on Self-tests         Yes
 1 DU ROM   0  Default Do                       Select Self-tests           Edit
 2 DL ROM   0  Default Do                       User Boot                   Edit
 3 TT USR   0  000000  Do                       Alternate Boot Block        No
 4                     Do                       LTC Register                Yes
 5                     Do                       Force Clock Interrupt       No
 6                     Do                       Clock Frequency             P/S
                                                Halt on Break               No
❸Lines  Address/Vec  Baud   Data  Stop  Par     Trap on Halt                No
Line 1  176500/300   9600   8     1     Dis     Ignore Battery              No
Line 2  176510/310   9600   8     1     Dis     Lines on                    176500
Line 3  176520/320   9600   8     1     Dis
Line 4  176530/330   9600   8     1     Dis
Line 5  176540/340   9600   8     1     Dis     Map                         Do
Line 6  176550/350   9600   8     1     Dis     Factory Setting             Do
Line 7  176560/360   9600   8     1     Dis     Save                        Do
```

Example 4–24 Setup Menu

❶ The time and date from the TOY clock is displayed here. To change the time and date:

1. Move the cursor to the Time field and press Return.
2. Enter the new time and date in the following format:

   NOTE
   Use the right arrow key to move from the time field to the date field.

   HH:MM:SS DD-MMM-YY

3. Press Return. The day is automatically entered.

❷ The boot device block contains six lines which are divided into five sections:

Nr	Represents the sequence number.
Dev	A two letter mnemonic which represents a boot program and its location.
Unit	represents the unit number to be booted from. The space bar moves you to the next unit number.
Address	Default is the standard address. If the device is set at a nonstandard address, enter a new address.
Do	Executes the boot for that line.

Each line of this boot device block provides you with 12 different boot devices with corresponding units and the source where the actual boot program is located. As a default setting, three boot devices have been set at the factory. To change the boot device, move the cursor to the device and press the space bar to display a different device.

[1] UNIBUS system only

A list of resident-supported boot devices can be displayed by positioning the cursor anywhere within the boot device block and pressing the H key. An example follows:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Device  Unit
Name    Numbers Source Device Type

AB      0       USR
DU      0-255   ROM    MSCP (RAxx, RDxx, RX50, RC25, ...)
DL      0-  3   ROM    RL01/RL02
DX      0-  1   ROM    RX01
DY      0-  3   ROM    RX02
MS      0-  1   ROM    TK25, TS04/05/11, TU80
MT      0-  1   ROM    TU10, TE10, TS03
MU      0-255   ROM    TMSCP (TK50, TU81, ...)
XH      0       ROM    ETHERNET
DL      0-  3   UBA    RL01/RL02

Press RETURN key when ready to continue
```

Example 4–25 Resident-Supported Boot Devices

❸ The SLU setup block individually sets the baud rate and character format for each channel as follows:

Baud rate range	300-38400
Data bits	Eight bits[1] Seven Bits
Stop bits	Two stop bits One stop bit[1]
Parity	Even Odd No parity[1]

[1]Default setting

NOTE
The status of the setup is only read during power-up or during an initialization (INIT). Changes in the SLU settings are not implemented until completion of a successful boot.

❹ Configuration parameters allow you to enable or disable various functions or change the values of others. Table 4–23 describes each parameter.

Table 4–23 Setup Menu Configuration Parameters

Parameter	Description	Values
Disable UBA ROM[1]	Controls the ROMs located on the UBA.	**Yes** disables the four ROM sockets. **No (factory setting)** enables the UBA ROMs. It is ignored when you try to boot the UBA or the M9312 boot ROMs.

[1]UNIBUS systems only

Table 4–23 (Cont.) Setup Menu Configuration Parameters

Parameter	Description	Values
Enable UBA 18-bit mode[1]	Selects 18- or 22-bit addressing modes. Its status is copied into bit 5 of the UBA KTJ11 Memory Configuration Register (KMCR).	**Yes** enables the memory to use 18-bit mode. **No (factory setting)** enables the memory to use 22-bit mode.
Memory intern	Allows you to disable the top 2 Mbyte of memory on a 4 Mbyte board. If the amount of memory selected does not match the memory onboard, the message *Mem mismatch* displays. This parameter has no effect on a 2 Mbyte board.	2 Mbyte 4 Mbyte
ROM on 173000	Enables or disables the CPU ROM code at address 1730000.	**Yes (factory setting)** enables the internal boot EPROM on the KDJ11-E. When enabled, the boot EPROM occupies the address area from 173000 to 173777. This area consists of 512 words which represents one page out of the EPROM. This parameter sets/resets bit 7 of the CSR at address 17777520 just before transferring control to another boot program. The page number can be selected through the bits 15-9 of the PCR register. **No** disables the internal boot EPROM on the KDJ11-E.
Rom on 165000	Enables or disables the CPU ROM code at address 165000.	**Yes** enables the internal EEPROM or boot EPROM depending on bit 6 of the CSR register (17777520). The ROM code uses an address area from 165000 to 165776 for EEPROM or for the boot EPROM, depending on bit 6 of the CSR register. **No (factory setting)** disables the internal EEPROM.
Power-up Mode	When the ROM code is started, it checks a status bit to determine if the unit is powering up or if the front panel RESTART switch was activated. The ROM code then uses the appropriate mode selected. There are four power-up modes which you can select.	**Dialog Mode (factory setting)** - Force dialog must be disabled (S5 off). The setup menu is entered when dialog mode is selected.

[1]UNIBUS systems only

Table 4–23 (Cont.) Setup Menu Configuration Parameters

Parameter	Description	Values
		Auto Mode - At the completion of the diagnostics the ROM code enters an automatic boot routine that tries to boot a previously selected device or devices. The list of devices can be from 1 to 6 devices long. Each device is tried sequentially until a successful boot occurs or the end of the boot table is reached. *ROM Mode* is a special automatic boot mode that is entered as a power-up/ restart option to boot specific devices when one or more of switches 2 through 4 on the KDJ11-E are set. When switches 2 through 4 of the KDJ11-E are set to one of the six combinations shown in Table 4–6 and force dialog mode is not selected, ROM mode is entered. The mode attempts to boot only the one device selected by this command. If the boot is unsuccessful, the ROM code prints out the normal error message and enters dialog mode. See Table 4–6 for switch settings. **ODT Mode** - At completion of a very limited set of tests, the ROM code executes a halt instruction and passes control to J11 micro-ODT. This mode is used in debug environments. The ROM code does not change any locations in memory before entering ODT mode. **Trap 24 Mode** - The ROM code loads the PSW with the contents of locate 26 and then transfers control to the address located in location 24. This mode is used when power-fail recovery is desired. The ROM code does not change any locations in memory before executing mode 24.
Restart mode	See power-up mode.	See power-up mode.
Power-on self-tests	Enables or disables power-on self-tests.	**Yes (factory setting)** executes all self-tests during a power-on. The force dialog switch (S5) must be set to off. During a restart the self-tests are not performed. **No** disables all self-tests.
Select self-tests	Allows you to enter the self-tests menu.	**Edit**
User boot	Allows you to enter the user boot menu.	**Edit**
Alternate boot block	After the boot block of a device is loaded into memory, the ROM code looks at word locations 0 and 2 to see if the device looks bootable. If the data is not correct, the ROM code types out an error message indicating that the media is not bootable.	**Yes** sets the ROM code to expect location 0 to be any nonzero number. **No (factory setting)** sets the ROM code to look for location 0 to be a value of 240 to 277 and for location 2 to be 400 to 777.

Table 4–23 (Cont.) Setup Menu Configuration Parameters

Parameter	Description	Values
LTC register	Enables/disables the LTC register.	**Yes (factory setting)** enables the clock CSR at address 17777546. **No** disables the clock CSR at address 17777546.
Force clock interrupt	Allows you to unconditionally force LTC interrupts.	**Yes** enables the clock to unconditionally request interrupts when the processor priority is 5 or less. **No (factory setting)** enables the clock to request interrupts only if the clock CSR is enabled, clock CSR bit 6 is 1, and the processor priority is 5 or less.
Clock frequency	Determines the source of the clock to be used.	**PS** determines the source of the clock to be from backplane pin BR1. The power supply normally drives this signal at 50 or 60 Hz. **50 Hz (factory setting)** determines the source of the clock to be from the KDJ11-E at 50 Hz. **60 Hz** determines the source of the clock to be from the KDJ11-E at 60 Hz. **800 Hz** determines the source of the clock to be from the KDJ11-E at 800 Hz.
Halt-on-break	Enables the processor to halt when the break key is pressed.	**Yes** halts the processor when the break key is pressed. **No (factory setting)** tells the processor to ignore any break request.
Trap-on-halt	Enables or disables a trap-on-halt.	**Yes** - If a halt instruction is executed in kernel mode, the processor traps to location 4 if a halt. **No (factory setting)** - If a halt instruction is executed in kernel mode, the processor enters J11 micro-ODT.
Ignore battery	This is used only when the current power-up or restart mode is set to 24.	**Yes** executes mode 24 regardless of the status of the battery. **No (factory setting)** - The battery OK signal must be present to execute mode 24. Battery OK indicates that the memory contents were not corrupted as a result of a power failure. If BOK is set, Trap 24 mode is not executed and dialog mode is entered.
Lines on	Changes the addresses of the serial interfaces 1-7 from 176500-176560 to 176600-176660.	**176500 (factory setting)** selects address 176500, vector 300. **176600** selects address 176600, vector 400. **DIS** disables all SLUs. **NOTE** **After selecting and saving the desired starting address, the system must be powered down and then rebooted for the change to occur.**
Map	Maps all locations in the I/O page.	To execute the map parameter, move the cursor to **Do** and press Return. Upon execution, the map menu is displayed (Example 4–28). After the valid I/O page addresses are displayed, the setup menu is displayed.
Factory setting	Resets all parameters to their factory settings.	Move the cursor to **Do** and press Return to execute the factory setting parameter.
Save	Saves all modifications made to the setup menu.	Press the Tab key to move the cursor to **Do**. Press Return to execute Save. You are prompted "Are you sure? Y/N". Enter Y to save the modifications. Enter N if you do not want to save the modifications.

NOTE
If you want to save all function modifications, you must execute Save *before* leaving the setup menu.

4.3.4 Self-Test Menu

When you execute Select Self-tests on the setup menu, the self-test menu is displayed (Example 4–26). This menu is used to determine which tests are executed on power-up and also allows individual tests to be selected and executed. The factory settings are shown in the following example:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

67 CPU Test                     Yes   44 Lines Config. Test       Yes
66 MMU Test                     Yes   43 Serial Lines Test        Yes
65 Pre-Console Test             Yes   40 Memory parity Test       Yes
64 MSER Test                    Yes   37 UBA Map Reg Test❶        Yes
63 CCR r/w Test                 Yes   36 UBA NPR Cycles Test❶     Yes
62 HIT/MISS-Reg Test            Yes
61 LTC Speed Test               Yes
60 Add-Stat-Reg Test            Yes   32 Loopback SLU Test        No
57 CPU-Err-Reg Test             Yes   31 Extended Memory Test     No
55 UBA Reg. Resp. Test❶         Yes
54 Address 0 Test               Yes
53 Pre-Memory (0-4KW) Test      Yes
52 FPA Register Test            Yes
51 FPA Function Test            Yes
50 Int Mem Address Test         Yes
47 Int Mem Data Test            Yes
46 PIRQ-Reg Test                Yes
45 LTC Int Test                 Yes    30 All Selected Tests      No

Monitor     Do                Test 00     Repeat 00000            Do
```

Example 4–26 Self-Test Menu

❶ UNIBUS system only

NOTE
Test 30, All Selected Tests, runs all tests selected (parameter = Yes) as a group.

Test 32, Loopback SLU Test, requires loopback connectors installed on all SLUs on the KDJ11-E.

4.3.4.1 Selecting or Deselecting Tests Executed Upon Power-Up

To select or deselect tests that are executed upon power-up:

1. Move the cursor to the test to be changed.
2. Press Return to select Yes or No.
3. Press Tab to select the Monitor field.
4. Press Return to return to the setup menu.
5. Press Tab to select the Save field.
6. Press Return to save the self-test parameters.

4.3.4.2 Selecting and Executing an Individual Test

To select and execute an individual test:

1. Move the cursor to the Test field.
2. Type in the test number of the test to be run.
3. Move the cursor to the Repeat field.
4. Type in the number of iterations to run the test or enter 0 to run the tests continuously.
5. Move the cursor to Do.
6. Press Return to execute testing.

NOTE
Ctrl/C terminates testing.

4.3.4.3 Selecting and Executing a Group of Tests (Test 30)

To select and execute a group of tests using Test 30:

1. Move the cursor to each test to be changed.
2. Press Return to select Yes or No.
3. Move the cursor to the Test field.
4. Enter 30 to select Test 30, All Selected Tests.
5. Move the cursor to the Repeat field.
6. Type in the number of iterations to run the tests or enter 0 to run the tests continuously.
7. Move the cursor to Do.
8. Press Return to execute testing.

NOTE
Ctrl/C terminates testing.

4.3.5 User Boot Menu

Example 4–27 shows a sample user boot menu. The addressing scheme shown is for clarity only. Normally the boot routine resides here.

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

                                                    ❸Exit                Do
❶Device name          DK   ❷Beginning addr   1000  ❹Save boot           Do
❺Dev Description  RK05     ❻Last byte addr   1016  ❼Load saved boot    Do
❽Highest Unit          7   ❾Start addr       1000  ❿Del saved boot     Do

Addrs.  000000  000002  000004  000006  000010  000012  000014  000016⓫
001000  012737  000005  177404  105737  177404  100375  000774  000000
001020  000000  000000  000000  000000  000000  000000  000000  000000
001040  000000  000000  000000  000000  000000  000000  000000  000000
001060  000000  000000  000000  000000  000000  000000  000000  000000
001100  000000  000000  000000  000000  000000  000000  000000  000000
001120  000000  000000  000000  000000  000000  000000  000000  000000
001140  000000  000000  000000  000000  000000  000000  000000  000000
001160  000000  000000  000000  000000  000000  000000  000000  000000
001200  000000  000000  000000  000000  000000  000000  000000  000000
001220  000000  000000  000000  000000  000000  000000  000000  000000
001240  000000  000000  000000  000000  000000  000000  000000  000000
001260  000000  000000  000000  000000  000000  000000  000000  000000
001300  000000  000000  000000  000000  000000  000000  000000  000000
001320  000000  000000  000000  000000  000000  000000  000000  000000
001340  000000  000000  000000  000000  000000  000000  000000  000000
001360  000000  000000  000000  000000  000000  000000  000000  000000
```

Example 4–27 User Boot Menu

❶ *Device name* is a two-letter mnemonic name for the boot program.

❷ *Beginning addr* refers to the the first address of the boot routine. The address range is 1000-17544. The ROM code prints out the old beginning address and prompts you for the new beginning address. Type in a new address or press Return to accept the old beginning address.

❸ *Exit* returns you to the Setup Menu.

❹ *Save boot* saves the boot program in the EEPROM. This function can take up to two minutes for the full length of 8 Kbytes.

❺ *Dev Description* is an optional but recommended description of the device name. The name is usually the name that is physically marked on the outside of the device (for example, RA82).

❻ *Last byte addr* is the address of the last byte of code used in memory.

❼ *Load saved boot* reloads previously saved boot program. This allows additional changes to be made to the boot routine.

❽ *Highest Unit* defines the allowable range of valid unit numbers for this device. If the value is set to 3, the allowable range is 0 to 3. If a unit number is typed in at boot time and it is not in range, an invalid unit number error occurs.

❾ *Start addr* is the address that the ROM code passes control to.

❿ *Del saved boot* deletes the boot program from the EEPROM.

NOTE
The beginning address and last byte address refer to physical addresses in memory for the program. Permitted values are between 1000 and 17544.

Pressing Return closes a location. To select a different address, use the cursor control arrows.

4.3.6 Map Menu

The map menu displays all addresses in the I/O page that respond. The I/O page is from addresses 17760000 through 17777776. In addition, all addresses on the CPU or the UBA that respond, are described. There is no description for optional device addresses that respond.

The setup menu is displayed at the completion of the Map function. The ROM code waits for you to press Return rather than scrolling data forward on video screen terminals. The ROM code always assumes the terminal can display at least 24 lines of 80-column data.

A sample map menu is shown in Example 4–28.

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

I/O page Map
Starting    Ending
Address     address

17765000   17765776   CPU ROM or EEPROM
17772100              Memory CSR
17772200   17772276   Supervisor I and D PDR/PAR's
17772300   17772376   Kernel I and D PDR/PAR's
17772516              MMR3
17773000   17773776   CPU ROM
17776500   17776566   SLU's
17777200   17777376   UBA map REG's
17777520   17777526   CSR, PCR, BCR/BDR ASR
17777546              Clock CSR
17777560   17777566   Console SLU
17777572   17777576   MMR0,1,2
17777600   17777676   User I and D PDR/PAR's

Press RETURN key when ready to continue

KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

I/O page Map
Starting    Ending
Address     address

17777730   17777736   DCSR, DDR, KMCR
17777744   17777752   MSER, CCR, MREG, Hit/Miss
17777766              CPU Error
17777772              PIRQ
17777776              PSW

Press RETURN key when ready to continue
```

Example 4–28 Map Menu

4.4 Diagnostic Programs

The KDJ11-E supports two types of diagnostic programs. Table 4–24 describes these programs.

Table 4–24 Diagnostic Programs

Diagnostic Program	Function	
DECX11	DECX11 is a system level exerciser. DECX11 tests all parts of the KDJ11-E CPU module. The following DECX11 modules should be configured for a basic system:	
	MON E	Monitor E supports the memory management unit and mapping in UNIBUS systems.
	MON Q	Monitor Q supports the memory management unit in Q-bus systems.
	KWA	Line time clock.
	CPA	KDJ11-E instruction set.
	CPB	KDJ11-E extended instruction set.
	FPB	Floating point unit.
	DLA	Serial line units.
Read-only memory (ROM) resident diagnostics	CPU ROM resident startup diagnostics test various functions specific to the CPU and UBA modules.	

A comprehensive set of diagnostics can be executed during a system power-up by selecting the Power-On Self-Test on the setup menu. A failure during diagnostic execution halts the testing and displays one of the following:

- If tests 30-67 fail, an error code and an error message on the console terminal (if connected).
- A test number in the KDJ11-E CPU module diagnostic LEDs.

4.4.1 KDJ11-E Self-Test

The KDJ11-E self-test allows you to comprehensively test:

- CPU
- Memory management
- On-board memory
- Serial line units
- Console/SLU bulkhead (if equipped)
- UNIBUS signals
- KTJ11-B

The self-test menu allows you to select and execute tests individually or as a group using either video terminal mode or hard copy terminal mode.

- To select and execute self-tests using the video terminal mode, proceed to Section 4.4.1.1.
- To select and execute self-tests using the hard copy terminal mode, proceed to Section 4.4.1.2.

4.4.1.1 Video Terminal Mode

NOTE

Test 30, All Selected Tests, runs all tests selected (parameter = Yes) as a group.

Test 32, Loopback SLU Test, requires loopback connectors installed on all SLUs on the console/SLU panel (if equipped).

Selecting or Deselecting Tests Executed Upon Power-Up

To select or deselect tests that are executed upon power-up:

1. Move the cursor to the test to be changed.
2. Press Return to select Yes or No.
3. Press Tab to select the Monitor field.
4. Press Return to return to the setup menu.
5. Press Tab to select the Save field.
6. Press Return to save the self-test parameters.

Selecting and Executing Individual Tests

To select and execute an individual test:

1. Move the cursor to the Test field.
2. Type in the test number of the test to be run.
3. Move the cursor to the Repeat field.
4. Type in the number of iterations to run the test or enter 0 to run the tests continuously.
5. Move the cursor to Do.
6. Press Return to execute testing.

NOTE

Ctrl/C terminates testing. To return to the setup menu, move the cursor to the Monitor field and press Return.

Selecting and Executing a Group of Tests

To select and execute a group of tests using test 30:

1. Move the cursor to each test to be changed.
2. Press Return to select Yes or No.
3. Move the cursor to the Test field.
4. Enter 30 to select test 30, All Selected Tests.
5. Move the cursor to the Repeat field.
6. Type in the number of iterations to run the tests or enter 0 to run the tests continuously.
7. Move the cursor to Do.
8. Press Return to execute testing.

NOTE

Ctrl/C terminates testing. To return to the setup menu, move the cursor to the Monitor field and press Return.

An example of the self-test menu follows:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

67 CPU Test                        Yes  44 Lines Config. Test          Yes
66 MMU Test                        Yes  43 Serial Lines Test           Yes
65 Pre-Console Test                Yes  40 Memory parity Test          Yes
64 MSER Test                       Yes  37 UBA map reg Test❶           Yes
63 CCR r/w Test                    Yes  36 UBA NPR cycles Test❶        Yes
62 HIT/MISS-Reg Test               Yes
61 LTC Speed Test                  Yes
60 Add-Stat-Reg Test               Yes  32 Loopback SLU Test            No
57 CPU-Err-Reg Test                Yes  31 Extended Memory Test         No
55 UBA reg. resp. Test❶           Yes
54 Address 0 Test                  Yes
53 Pre-Memory (0-4KW) Test         Yes
52 FPA Register Test               Yes
51 FPA Function Test               Yes
50 Int Mem Address Test            Yes
47 Int Mem Data Test               Yes
46 PIRQ-Reg Test                   Yes
45 LTC Int Test                    Yes  30 All Selected Tests           No

Monitor    Do              Test 00   Repeat 00000                      Do
```

Example 4–29 Self-Test Menu

❶ UNIBUS systems only

4.4.1.2 Hard Copy Terminal Mode

Selecting and Executing Individual Tests

To select and execute an individual test:

1. Enter **D** on the main menu,.
2. Press Return to execute the diagnostic command. The self-test menu is displayed (Example 4–30).

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
Licensed to Digital Equipment Corporation
❶
67 CPU Test
66 MMU Test
65 Pre-console Test
64 MSER Test
63 CCR r/w Test
62 HIT/MISS-Reg Test
61 LTC Speed Test
60 Add-Stat-Reg Test
57 CPU-Err-Reg Test
55 UBA Reg. Resp. Test[1]
54 Address 0 Test
53 Pre-Memory (0-4KW) Test
52 FPA Register Test
51 FPA Function Test
50 Int Mem Address Test
47 Int Mem Data Test
46 PIRQ-Reg Test
45 LTC Int Test
44 Lines Config. Test
43 Serial Lines Test
40 Memory parity Test
37 UBA Map Reg Test[1]
36 UBA NPR Cycle Test[1]
32 Loopback SLU Test ❷
30 All Selected Tests

Test number = 67❸    New =❹

Repeat counter = 000000 ❺    New =❻
```

Example 4–30 Self-Test Menu

❶ List of tests that can be selected.

❷ **32 Loopback SLU Test** requires that turnarounds are installed on the SLU's (1-7).

❸ **Test number** lists the currently selected test.

❹ **New** allows you to change the currently selected test by entering a new test number.

❺ **Repeat counter** lists the currently selected number of times to repeat a test.

❻ **New** allows you to change the currently selected number of times to repeat a test by entering a new number.

3. Enter the number of the new test to be run.
4. Enter the number of the times the new test is to be repeated. Enter **0** if you want the test to run continuously.
5. Press Return to start the testing.

[1] UNIBUS systems only

During the testing, the test number, description of the test, error count and repeat number are displayed and printed. If continuous testing is selected, no information will be printed except for errors to allow the Diagnostic Command to run for an extended period of time.

6. Press Ctrl/C or Ctrl/P to stop the testing.

Selecting and Executing a Group of Tests

To select and execute a group of tests:

1. Type **S** on the command line on the main menu.
2. Press Return to execute the Setup command. The setup menu is displayed (Example 4–31).

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990

Setup Mode Commands

1 Exit
2 Select configuration parameters
3 Select diagnostic configuration
4 Select serial line parameters
5 Select boot parameters
6 List available boot programs
7 Factory setting
8 Save the setup table in the EEPROM
9 Load EEPROM data into the setup table
10 Load EEPROM boot program into memory
11 Edit or create EEPROM boot program
12 Save a boot program in the EEPROM
13 Delete a saved EEPROM boot program
14      Enter ROM ODT
Commands are: [1, 2, 3, 4, 5, 6, 7,8 ,9 10, 11, 12, 13, 14]
type a command then press the RETURN key:
```

Example 4–31 Setup Menu

3. Type **3** on the command line of the setup menu.
4. Press Return to execute the setup mode command 3. The following list is displayed:

```
KDJ11-E Monitor  Version 1.06 30-Jul-1990
(C) Digital Equipment Corporation 1990
A Nr. 67 CPU Test                        (0) = No   (1) = Yes   = 1
B Nr. 66 MMU Test                        (0) = No   (1) = Yes   = 1
C Nr. 65 Pre-Console Test                (0) = No   (1) = Yes   = 1
D Nr. 64 MSER Test                       (0) = No   (1) = Yes   = 1
E Nr. 63 CCR r/w Test                    (0) = No   (1) = Yes   = 1
F Nr. 62 HIT/MISS-Reg Test               (0) = No   (1) = Yes   = 1
G Nr. 61 LTC Speed Test                  (0) = No   (1) = Yes   = 1
H Nr. 60 Add-Stat_Reg Test               (0) = No   (1) = Yes   = 1
I Nr. 57 CPU-Err-Reg Test                (0) = No   (1) = Yes   = 1
J Nr. 55 UBA Reg. Resp. Test❶            (0) = No   (1) = Yes   = 1
K Nr. 54 Address 0 Test                  (0) = No   (1) = Yes   = 1
L Nr. 53 Pre-Memory (0-4KW) Test         (0) = No   (1) = Yes   = 1
M Nr. 52 FPA Register Test               (0) = No   (1) = Yes   = 1
N Nr. 51 FPA Function Test               (0) = No   (1) = Yes   = 1
O Nr. 50 Int Mem Address Test            (0) = No   (1) = Yes   = 1
P Nr. 47 Int Mem Data Test               (0) = No   (1) = Yes   = 1
Q Nr. 46 PIRQ-Reg Test                   (0) = No   (1) = Yes   = 1
R Nr. 45 LTC Int Test                    (0) = No   (1) = Yes   = 1
S Nr. 44 Lines Config. Test              (0) = No   (1) = Yes   = 1
T Nr. 43 Serial Lines Test               (0) = No   (1) = Yes   = 1
U Nr. 40 Memory parity Test              (0) = No   (1) = Yes   = 1
V Nr. 37 UBA Map Reg Test❶               (0) = No   (1) = Yes   = 1
W Nr. 36 UBA NPR Cycle Test❶             (0) = No   (1) = Yes   = 1
X Nr. 32 Loopback SLU Test               (0) = No   (1) = Yes   = 1
Y Nr. 31 Extended Memory Test            (0) = No   (1) = Yes   = 1
Z Nr. 30 All Selected Tests              (0) = No   (1) = Yes   = 1
Type CTRL\Z to exit or press Return key to proceed
```

Example 4–32 Selecting Individual Tests

❶ UNIBUS systems only

5. To deselect a test, enter **0** at the appropriate test number.
6. To select a test, enter **1** at the appropriate test number.
7. Press Return to return to the main menu.
8. Enter **D** on the main menu to select and execute the diagnostic command. The self-test menu is displayed (Example 4–30).
9. Enter **30** as the number of the new test to be run. Test 30 runs the group of tests previously selected in setup command 3.

10. Enter the number of the times test 30 is to be repeated. Enter **0** if you want test 30 to run continuously.
11. Press Return to start the testing.

    During the testing, the test number, description of the test, error count, and repeat number are displayed and printed. If continuous testing is selected, no information will be printed except for errors to allow the diagnostic command to run for an extended period of time.
12. Ctrl/C or Ctrl/P stops the testing.

Table 4–25 describes the test numbers, status display numbers, and boot error numbers.

Table 4–26 provides recommendations to correct any errors that can occur during testing.

Table 4–25 LED Display Messages and Descriptions

Number	Function	Description
77	Test	CPU or halt switch. When the CPU is powered up or restarted the DCOK signal causes the display to be set to 77. If the CPU hangs with the display set to 77 then either the halt switch is on or the CPU does not have enough logic functioning to execute an instruction out of the ROM.
76	Test	CPU and MMU. Set the LED display to 76 to indicate the first instruction has been executed without hanging up the processor. Enter standalone mode, set PSW to priority 7 and turn off MMU. Clear PCU and set up SP. Jump to the high page of the ROM if not already there (173xxx). Execute a few simple CPU tests. General register writes and reads, branch instructions, and a simple JSR instruction.
75	Test	CPU ROM Checksum and Page Control Register (17777522). Use the user PAR for stack operations and enable upper and lower page of EPROM so that page 0 can be accessed on 173xxx and 165xxx. Test the function of the PCR register. The following steps are performed: • Disable 173xxx and jump to 165xxx • Enable 173xxx, disable 165xxx, and jump back to higher page • Enable 165xxx and jump to lower page • Increment PCR high and compare the page number on 173774 • Jump back to higher page and use lower page for comparison
74	Test	Turn on MMU, run CPU and MMU. Test all MMU registers: • Test all PARs using a floating 0 and 1 pattern • Check with this pattern all R/W bits of PDR <14:8, 3:1> • Write PAR address into PAR registers to verify address uniqueness • Compare content of MMR2 with the PC of current instruction • Check R/W bits of MMR3 <5:0> Enable the memory management unit.

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
73	Test	Determine if system is restarting or powering up. Determine if the ROM code has started by the CPU being powered up or restarted, by use of the Restart switch on the front panel. **UNIBUS Systems** Test the state of bit 7, Reboot Pulse, of the Memory Configuration Register. (17777734) Bit 7 = 1 System is being started by use of the Restart switch on the front panel. Bit 7 = 0 System is being powered up. **Q-bus Systems** Test the state of bit 14, Reboot Pulse, of the Control/Status Register (17777520). Bit 14 = 1 System is being started by use of the Restart switch on the front panel. Bit 14 = 0 System is being powered up.
72	Test	EEPROM Checksum. This test will verify EEPROM checksum. If a checksum error is found dialog mode is entered. The user should do a factory setting to write a good EEPROM checksum. The ROM code will branch in the following way: A. Check the state of the Force Dialog switch. If switch 5 is in the ON position go to Dialog Mode no matter what the Restart or Power up option was set to. B. Compare EEPROM Checksum, on error jump to Dialog Mode. C. If Trap 24 was selected as a Power-up or Restart Option go to Test 70. D. If ODT was selected as a Power-up or Restart Option go to Test 71. E. Determine what diagnostic selections previously selected.
71	Status Display	This is not a test. It is the display that indicates that the selected Power-up/Restart mode is ODT. A limited set of diagnostics is run and the ROM code executes a HALT instruction and passes control to the micro-ODT code. The EEPROM parameter are installed and the MMU is disabled. If the operator proceeds from ODT without changing any registers the ROM code will continue to run selected tests and enter dialog mode when complete.

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
70	Status Display	System Start Mode Trap 24. This is not a test. If the selected startup /restart mode is 24/26 then check the status of the ignore battery status parameter in the EEPROM. *** Ignore Battery parameter = No. If the Battery Backup Reboot Enable Bit in the Control Status Register (177777520 bit 15) is cleared indicating that the battery backup maintained voltages during the previous power fail. The ROM code loads the contents of location 26 into the PSW and then transfers control to the address specified by the contents of location 24. The EEPROM parameters are installed and the MMU mapping is disabled. If BBRE is set, Battery Backup failed to maintain memory voltages, go to Dialog Mode. *** Ignore Battery parameter = Yes. Unconditionally the ROM code loads the contents of location 26 into the PSW and then transfers control to the address specified by the contents of location 24. The EEPROM parameters are installed and the MMU mapping is disabled.

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
67	Test	Miscellaneous CPU. There are 10 CPU diagnostics implemented to test the CPU instruction set: Test A Branch instruction test BEQ, BNE, BMI, BPL, BVS, BVC, BCS, BCC, BLT, BGT, BLE, BGE, BLOS, BHI, BR Test B Single operand instruction destination mode 0 CLR, COM, NEG, ROR, ROL, ADC, SBC, ASR, ASL, INC, DEC, TST Test C Byte single operand instruction destination mode 0 CLRB, COMB, NEGB, SWAB, RORB, ROLB, ADCB, SBCB, ASRB, ASLB, INCB, DECB, TSTB Test D Double operand word instruction all source modes destination mode 0. MOV, CMP, ADD, SUB, BIC, BIS, BIT, XOR Source mode 0 Source mode 1 Source mode 2 Source mode 3 Source mode 4 Source mode 5 Source mode 6 Source mode 7 Test E Jump instruction destination modes 1, 3, 6 JMP Mode 1 Mode 3 Mode 6 Test F TSTB and TST destination mode 1, 2, 4, PC rel addr TST, TSTB Mode 1 Mode 2 Mode 4 PC relative addr

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
		Test G Byte double operand instruction destination mode 0 BICB, BITB, BISB, MOVB, CMPB Source mode 1 Source mode 2 Source mode 4 PC relative addr Test H Single and Double Operand Instructions Mode 1-7 ADC, ADD, ASL, ASR, BIC, BIS, BIT, CLR, CMP, COM, DEC, INC, MOV, NEG, ROL, ROR, SUB, TST Test I Byte instruction destination modes 1-7 CLRB, COMB, SBCB, INCB, NEGB, DECB, SWAB, ASLB, RORB, ASRB, ROLB, ADCB, CMPB, BISB, BICB, BITB Test J JSR instruction destination mode 7
66	Test	MMU Modes and Aborts. Test the memory management unit in kernel, supervisor and user mode. All three modes are read and write with byte/word access. Test memory management violation and error bits. Test that the protection bits in the PDRs will cause aborts when the conditions are violated. Verify that abort occurs through virtual address 250. Verify that MMR1 properly records changes in the general purpose registers affected by the abort.
65	Pre-console	Test the console transmitter ready bit. Send a nonprinting character to console and wait in a time-out loop for the ready bit. Identify the terminal type... Hardcopy of Video Terminal. The ROM code supports a VT100, VT220, VT330. All other terminal types will be set up as hard copy. From now on, a message including test number and a short description will be displayed for the next tests on the terminal.
64	Test	MSER Test read/write access of the Memory System Errror Register (17777744).
63	Test	CCR R/W/Timeout. Rotate pattern through the Cache Control Register (17777746) to check R/W bits.
62	Test	Hit/Miss Register. Test force miss condition. Produce a stream of MISS cycles to check the function of the Hit/Miss Register (17777752).
61	Test	LTC Speed. Test the Line Time Clock. Test the On-board 50HZ, 60HZ and 800HZ clock source. A internal software loop is used to verify clock speeds. The interrupt logic is not tested here.
60	Test	Additional Status Register Test (17777526). Tests the functionality of ASR bits 12 and 13, serial line address/vector selection logic. Test the functionality of ASR bits 4 and 5, internal memory address encoding.

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
57	Test	CPU Error Conditions (CPU Error Register). Test the CPU Error Register (17777766). Verify the error detection logic is functioning. Error conditions will be created to test error detection logic. The following errors will be tested: • Bit 6: word access to an odd byte address • Bit 5: access to a nonexistent memory address • Bit 4: access to a nonexistent I/O address NOTE: Address 1776600 is used for a timeout address. If 1776600 exists on the system this test will fail. Verify bits 3..0 are read only.
56	Test	Reserved
55	Test	UBA register response test ***UNIBUS Systems Only*** Test that the DCSR, KMCR and DDR respond properly on the UBA. Test the KMCR with a data pattern. Enable UBA diagnostic mode and read the following UNIBUS lines. UNIBUS Address Lines UNIBUS Data Lines 15:00 UNIBUS Control Lines: C0, C1, PB, SSYN, MSYN
54	Test	Verify that memory exists at location 0. Rotate pattern through memory location 0. Test read/modify/write with bus lock.
53	Test	Test memory from 0 to 4 KW. This test fully checks out the first 4 KW of memory before the main memory tests are loaded and run. The following steps are performed: • Write pattern (101010..) in ascending order into memory to verify data • Complement, read, and compare data in descending order • Write physical address as data into memory in ascending order to verify address uniqueness • Compare memory address data in descending order
52	Test	FPA Register. Check the bits of the floating-point register. Rotate a bit through the register AC0-AC5 and execute instructions in double-precision mode. The results are stored in memory and compared with the pattern. An error will occur if no floating-point accelerator is available.
51	Test	FPA Function. Clear AC0-AC5 and memory space, set double precision mode. Test the following instructions. LDFPS, STFPS, SETD, CLRD, STD, LDD, LDCID, ADDD, DIVD, MOD, SUBD, STCDI, CFCC

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
50	Test	Internal Memory Address Test. This test verifies the address uniqueness of the whole internal memory. The segment address is written as data on each segment address of internal memory, starting from physical 0 to the end of internal memory. The data is compared in descending order. During the whole test parity abort is enabled.
47	Test	Internal Memory Data Test. This test will verify all of memory with data patterns. Write pattern in ascending order into memory. Compare all data in descending order. Complement all data in ascending order. Compare complemented data in descending order. Parity is enabled for this test. If a memory error occurs the following information will be printed: The memory address Expected pattern Pattern read
46	Test	PIRQ Interrupt Level Test. This test checks the software interrupt facility of the CPU. A trap table is installed in RAM to detect unexpected interrupts.

PIRQ Level	PSW Interrupt Level	Expect Interrupt Vector
1	0	240
2	1	240
3	2	240
4	3	240
5	4	240
6	5	240
7	6	240

Next test sets PIRQ and PSW interrupt to the same level. Interrupt should only occur after decrementing PSW level by one step. Repeat with all interrupt levels.

Last tests starts with the lowest PSW interupt level, enable all PIRQ interrupt, count and compare interrupt events. Repeat the loop with all PSW interrupt levels.

45	Test	LTC Interrupt Test. Test the line time clock interrupt facility for the 50, 60, 800 Hz clock source. Test the ability of the clock to interrupt to location 100 and that it can interrupt at the correct BR Level. On an interrupt, once the clock is in sync, check that bit 7 (LCM) is cleared on an interrupt.
44	Test	SLU Configuration. Check transmitter ready bit and look for correct configuration of all Serial Lines Units.

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
43	Test	SLU Function. Test the receiver and transmitter interrupt ability of all the Serial Line Units. Show received characters with corresponding port number if turnaround connectors installed.
42	Test	Reserved
41	Test	Reserved
40	Test	Memory Parity. Memory Parity Error bit is tested by creating parity errors. Memory is tested in 4 KW segments.
37	Test	UBA Mapping Registers Test ***UNIBUS Systems Only*** Test all 32 UNIBUS Map Register pairs with a rotating 1s and 0s pattern and an address uniqueness pattern. All 32 register pairs are tested.
36	Test	UBA NPR Read/Write Cycles ***UNIBUS Systems Only*** With mapping disabled execute a floating 1s and 0s test through a floating address pattern using diagnostic Data In and Out cycles for 124K words of memory if present. With mapping enabled, execute a floating 1s and 0s test through a floating address pattern using diagnostic Data In and Out cycles for up to 2044K words of memory if present. This test floats a 1 and 0 across both inputs to the UBA address adder. This tests that the UNIBUS can write and read all 0s and 1s.
35	Test	Not Used
34	Test	Not Used
32	Test	Serial Line Unit Test **Install Turnaround Connectors** This test assumes all lines have turnaround connectors installed. This test uses the EEPROM Configuration Parameter (Baudrate, Data Bits, Stop Bits, Parity) to check the data rate of the serial lines. It calculates first a counter value dependent on the EEPROM parameter. Then it checks the port to see whether all loop back connectors are installed. The routine installs the RX vectors for the receiver interrupt routine and starts to transmit a fixed number of characters. The transmitter time is compared with the calculated data rate. An LTC counter is running in the background, to stop the test if no RX interrupt happens in a defined time. All data received is compared with the data transmitted and the error bits are monitored.
27		Not Used
26		Not Used
25		Reserved for the MDM in "A" series enclosures only. Not used by ROM code. This code is driven by the MDM module on UNIBUS systems. Blower under/over speed. This is for "A" Series UNIBUS systems only.
24		Not Used
23		Not Used
22		Not Used
21	Boot Error	Drive error
20	Boot Error	Controller error

Table 4–25 (Cont.) LED Display Messages and Descriptions

Number	Function	Description
17	Boot Error	Boot device selection was invalid
16	Boot Error	Invalid unit number selected
15	Boot Error	Nonexistent drive
14	Boot Error	Nonexistent controller
13	Boot Error	No tape
12	Boot Error	No disk
11	Boot Error	Invalid boot block
10	Boot Error	Drive not ready
07	Boot Error	No bootable device found while in Auto Boot Mode
06		Not used
05		Not used
04	Status Display	Dialog mode
03	Status Display	UBA ROM boot in progress
02	Status Display	EEPROM boot in progress
01	Status Display	CPU ROM boot in progress
00	Status Display	Boot successful Control transferred from ROM code to booted device. The message "Starting System" is displayed. The display blanks when it receives a code of 00.

Table 4–26 Error Messages

LED Display	Probable Cause	Recommended Action
77	1. Halt switch/button in halt position. 2. CPU module failure. 3. UNIBUS failure.	1. Check run/halt switch. 2. Replace the CPU module. 3. Replace UBA; verify devices on UNIBUS.
76	CPU module failure.	Replace the CPU module.
75	CPU module failure.	Replace the CPU module.
74	CPU module failure.	Replace the CPU module.
73	1. UBA module failure. 2. CPU module failure.	1. Replace the UBA. 2. Replace the CPU module.
72	CPU module failure.	Replace the CPU module.
70	1. UBA failure. 2. CPU module failure.	1. Replace the UBA. 2. Replace the CPU module.
67	CPU module failure.	Replace the CPU module.

Table 4–26 (Cont.) Error Messages

LED Display	Probable Cause	Recommended Action
66	CPU module failure.	Replace the CPU module.
65	CPU module failure.	Replace the CPU module.
64	CPU module failure.	Replace the CPU module.
63	CPU module failure.	Replace the CPU module.
62	CPU module failure.	Replace the CPU module.
61	Line Time Clock failure.	Replace the CPU module.
57	CPU module failure.	Replace the CPU module.
55	1. UBA failure. 2. CPU module failure.	1. Replace the UBA. 2. Replace the CPU module.
54	CPU module failure.	Replace the CPU module.
53	CPU module failure.	Replace the CPU module.
52	CPU module failure.	Replace the CPU module.
51	CPU module failure.	Replace the CPU module.
50	CPU module failure.	Replace the CPU module.
47	CPU module failure.	Replace the CPU module.
46	CPU module failure.	Replace the CPU module.
45	CPU module failure.	Replace the CPU module.
44	CPU module failure.	Replace the CPU module.
43	1. CPU module failure. 2. SLU panel failure.[1] 3. Cables from SLU to CPU.[1]	1. Replace the CPU module. 2. Replace the SLU panel. 3. Check the cables from SLU to CPU.
40	CPU module failure.	Replace the CPU module.
37[2]	UBA failure.	Replace the UBA.
36[2]	UBA failure.	Replace the UBA.
32	1. CPU module failure. 2. Turnarounds. 3. Cables from SLU to CPU.	1. Replace CPU module. 2. Check turnarounds. 3. Check cables from SLU to CPU.
31	CPU module failure.	Replace CPU module.
25[2]	1. KDJ11-E: Blower assembly. 2. KDJ11-E: Fan assembly.	1. KDJ11-E: replace blower assembly. 2. KDJ11-E: replace fan assembly.
21	Boot error indicating that the media you are trying to to boot from is not bootable.	Reboot from another media.
20	Boot error.	Make sure that the NPG jumper (UNIBUS system only) was removed if the device is a direct memory access (DMA) controller. Consult the device's technical manual for more information.

[1]If equipped

[2]UNIBUS systems only

Table 4–26 (Cont.) Error Messages

LED Display	Probable Cause	Recommended Action
17	Boot error indicating that the mnemonic typed in for the boot device is either incorrect or the boot ROM for that device is not installed.	Enter dialog mode and list the valid devices.
16	Boot error indicating an invalid unit number selection.	The unit number after the mnemonic is not within the acceptable range for that device. See that device's technical manual for help.
15	Boot error indicating a nonexistent drive.	The drive number you are trying to boot from is not on the system. Enter a drive number that is on the system.
14	Boot error indicating a nonexistent controller.	The controller for the device you are trying to boot from is not on the UNIBUS or is addressed incorrectly.
13	Boot error indicating that no tape is installed in the drive.	Install a tape.
12	Boot error indicating that no media is in the drive or the drive LOAD button is not in.	Boot from a drive in which media is installed. Push the drive LOAD button in.
11	Boot error indicating that the bootstrap data from the device does not conform to the boot block specifications.	Make sure that media is bootable. Change setup mode to accept nonstandard boot blocks.
10	Boot error indicating no media is present in the drive or the disk drive has not completed its spinup function.	Boot from a drive in which media is installed. Wait until the disk drive has completed its spinup function.
7	Boot error indicating no bootable device is found in automatic boot mode.	Check devices in auto boot list.

CAUTION
Do not bypass errors unless all user software has been removed or write protected.

4.4.2 KDJ11-E CPU Module Fault Isolation

The KDJ11-E module contains:

One green power OK LED	Indicates DCOK status
One yellow LED	Indicates that dc power to the CPU module is present
Six red LEDs	Corresponds to the status display shown in Figure 4–1

If the CPU module is the suspected problem, before replacing the module:

- Ensure that jumpers W1 and W2 are configured correctly (Figure 4–1).

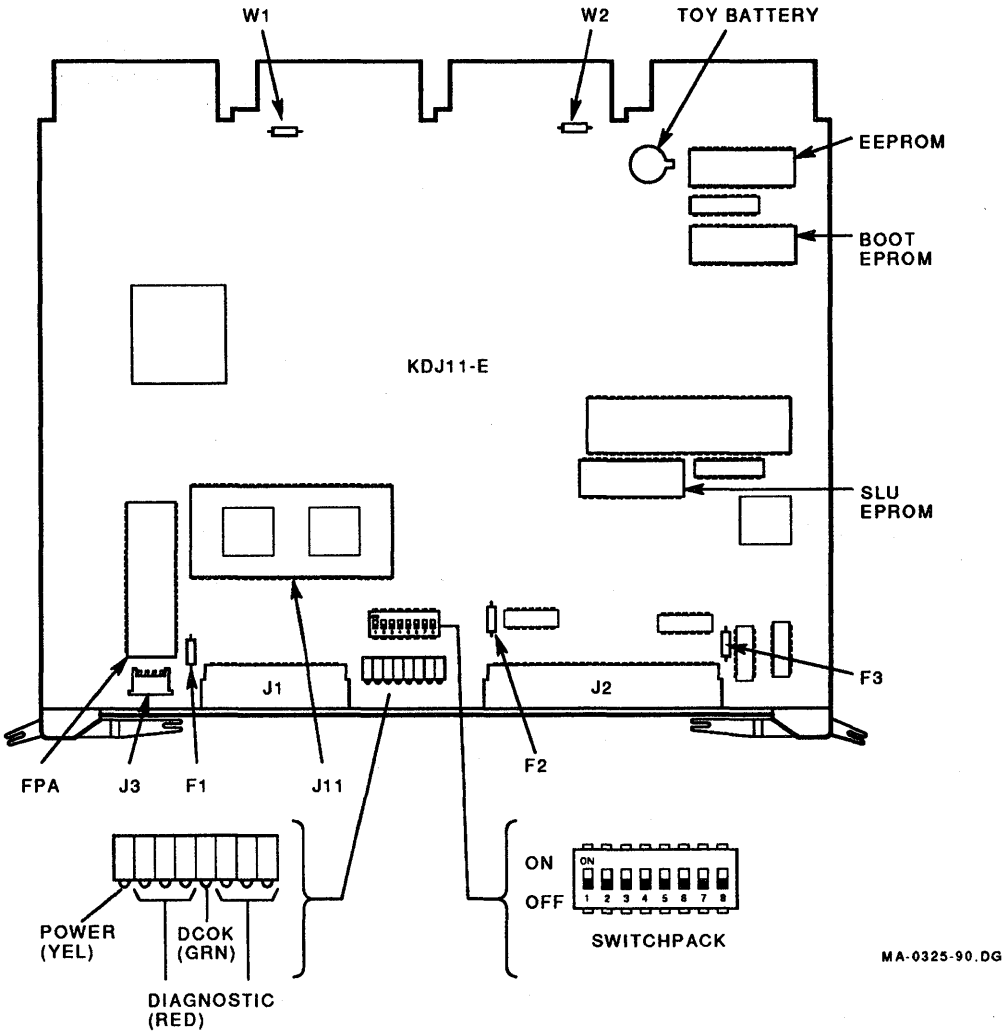


Figure 4–1 KDJ11-E CPU Module Layout

- Ensure that the dual in-line package (DIP) switches are configured correctly (Figure 4–1).
- The three 1A fuses (F01, F02 and F03). F01 and F02 protect +12 V on the connectors JO1 (pins 30, 40, 50, and 60) and JO2 (pins 10, 20, 30, and 40). F03 protects +5 V on JO2 (pin 1).

Table 4–27 describes some CPU failure symptoms, the probable causes, and corrective actions.

Table 4–27 CPU Troubleshooting

Symptom	Possible Cause	Corrective Action
TOY fails.	Defective battery.	Replace battery.
SLU ports 1-4 fail.	Defective fuse.	Replace F2.
SLU ports 5-8 fail.	Defective fuse.	Replace F1.
No +5V on J02 pin 1.	Defective fuse.	Replace F3.
Power LED is not lit.	No +5 V present on the CPU module.	Verify the presence of +5 V and ensure that W1 & W2 are configured correctly.

5
Extended LSI-11 Bus

5.1 Introduction

The processor, memory and I/O devices communicate through 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 two signal lines. The signal lines are functionally divided as listed in Table 5–1 . See 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 5–1 Summary Of Signal Line Functions

Quality	Function	Bus Signal Mnemonic
16	Data/address lines	BDAL <15:00>
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 through 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 by 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 by 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-E 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 (that is, 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 12.8 μs. The KDJ11-E has a bus timer that restarts the clock when no device responds to BDIN L or BDOUT L within 12.8 μs. An immediate trap to location 4 occurs. The slowest peripheral or memory device must respond in less than 12.8 μs to prevent a bus timeout error.

5.2 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 <21:00> 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 <21:00> 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 <21:00>

4. Signal outputs of receivers are referred to with the prefix character R, for receive. For example:

   RSYNC, RWTBT, RBS7, RDAL <21:00>

Whenever timing is important, the designations in items 3 and 4, listed previously, 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.

5.3 Data Transfer Bus Cycles

Data is transferred between a bus master and slave device to accomplish various functions. Table 5–2 describes the data transfer bus cycles and their functions.

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 5–3 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 5–2 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 5–3 Data Transfer Bus Signals

Mnemonic	Description	Function
BDAL <21:00> 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	

5.3.1 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 which are performed vary slightly, depending on the type of data transfer desired.

5.3.1.1 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 <21:00> 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 <21:00>, 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.

5.3.1.2 DATI

The DATI bus cycle is a read operation that inputs data from the slave device to the bus master. Figure 5–1 shows the operations performed by the bus master and slave device during a DATI bus cycle. Figure 5–2 shows the DATI bus cycle timing. 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

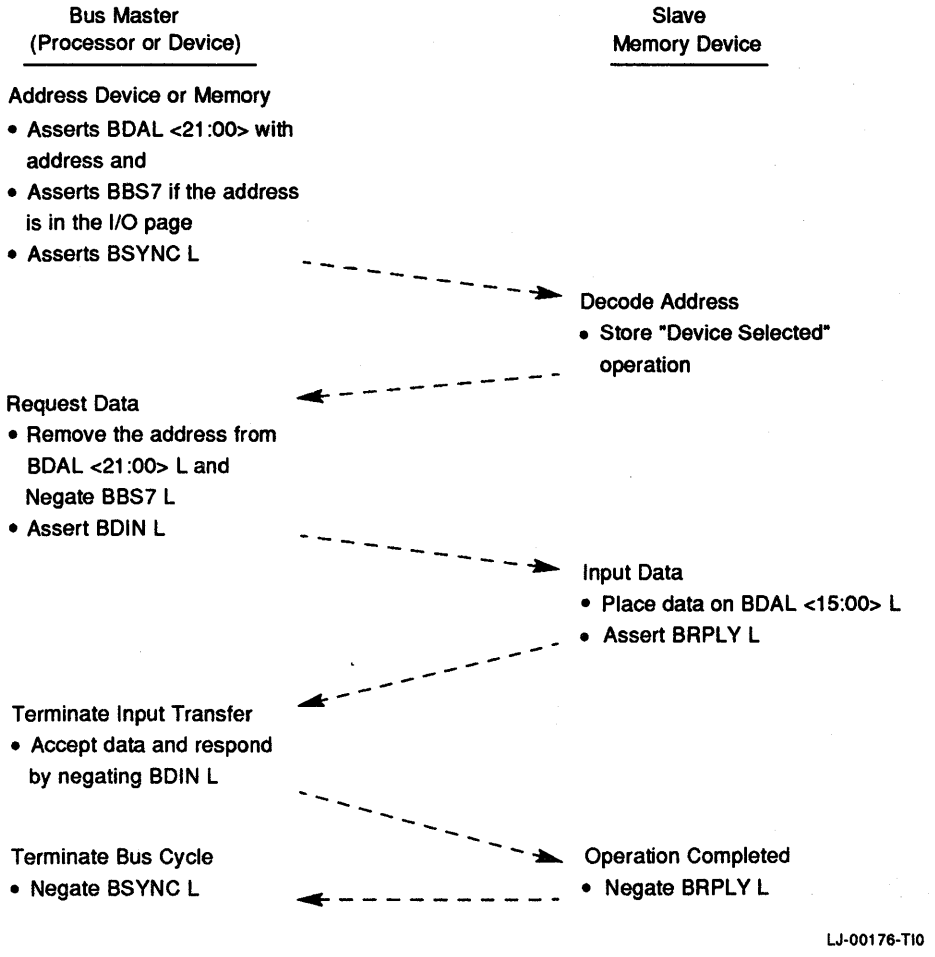
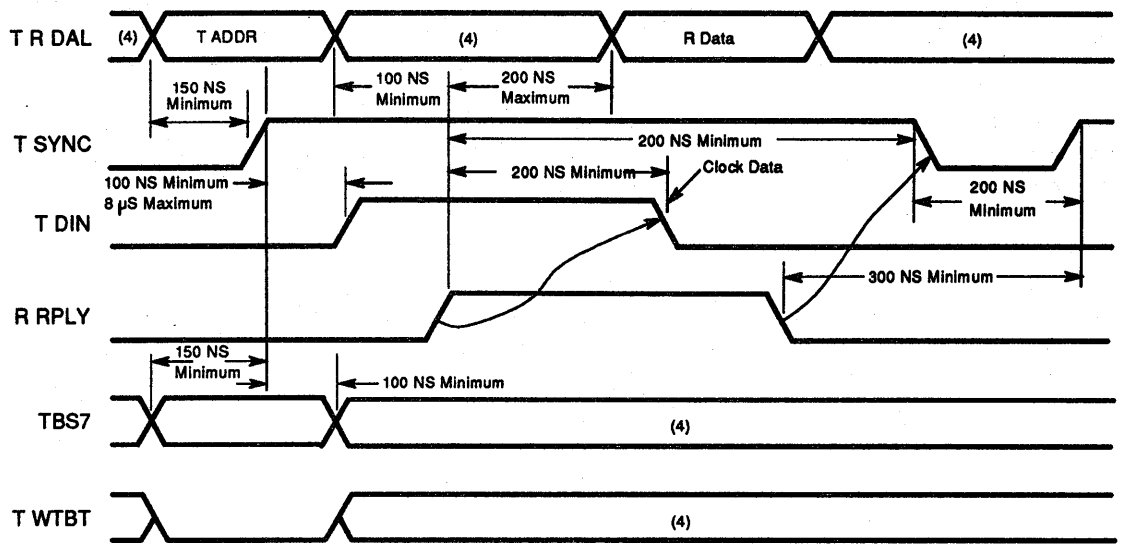
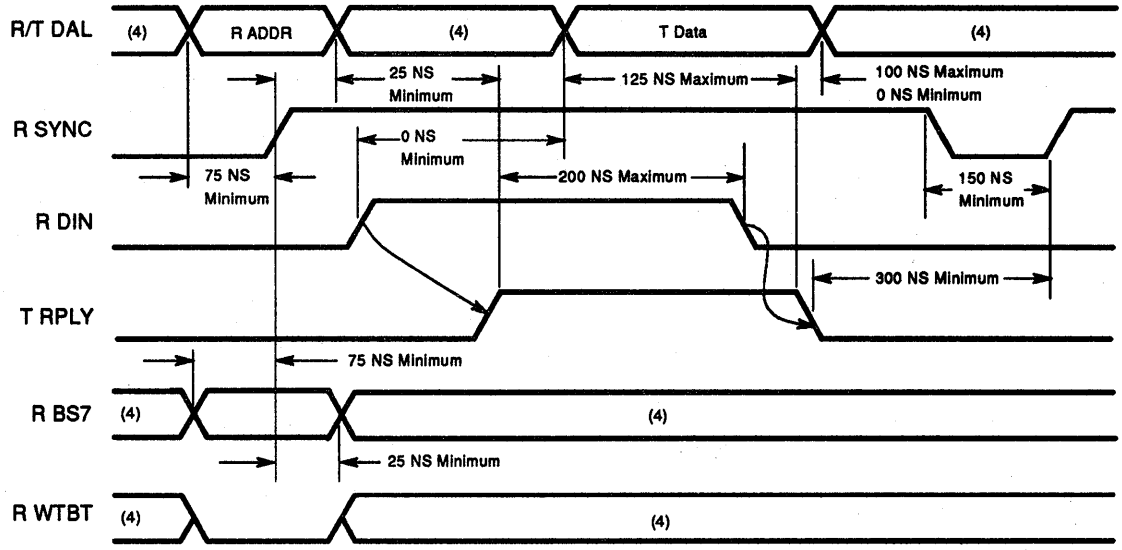


Figure 5–1 DATI Bus Cycle



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.

LJ-00177-TI0

Figure 5–2 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 <15:00> 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.

5.3.1.3 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. Figure 5–3 shows the operations performed by the bus master and slave device during a DATO(B) bus cycle. Figure 5–4 shows the DATO(B) bus cycle timing.

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 <15:00> 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 <15:08> L) is selected. Otherwise, the low byte (BDAL <07:00> 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 <21:00> 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.

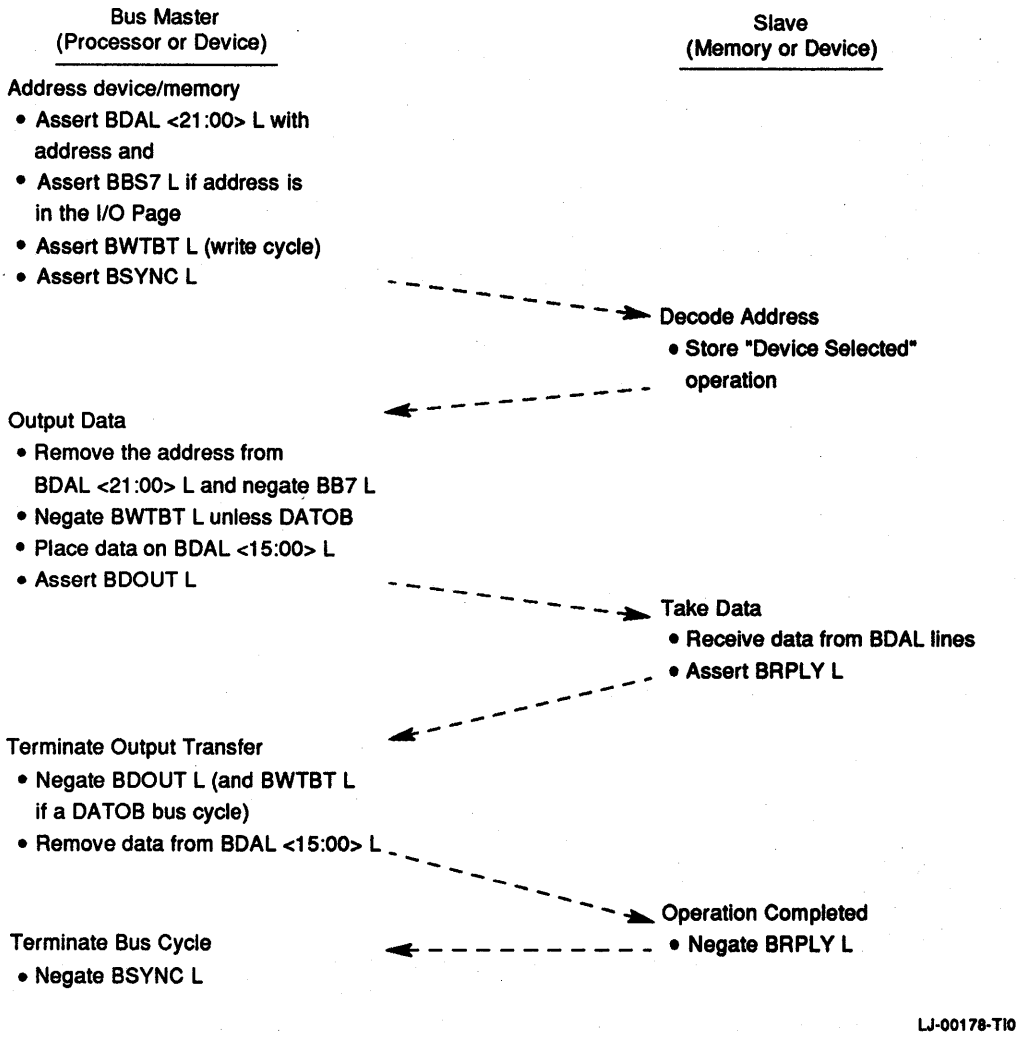
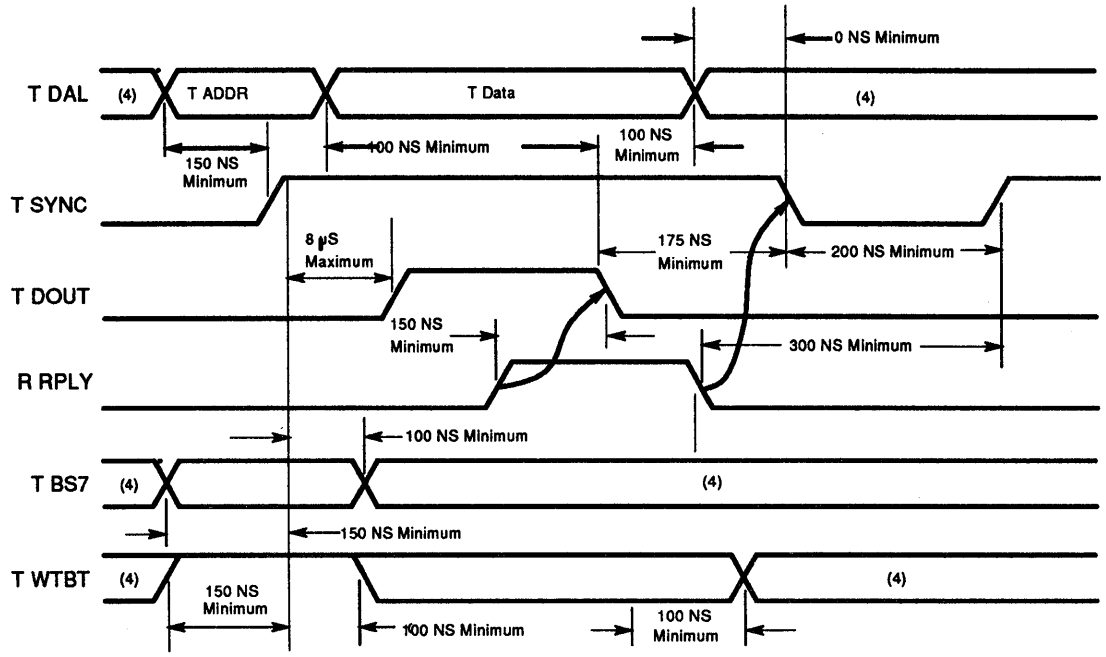
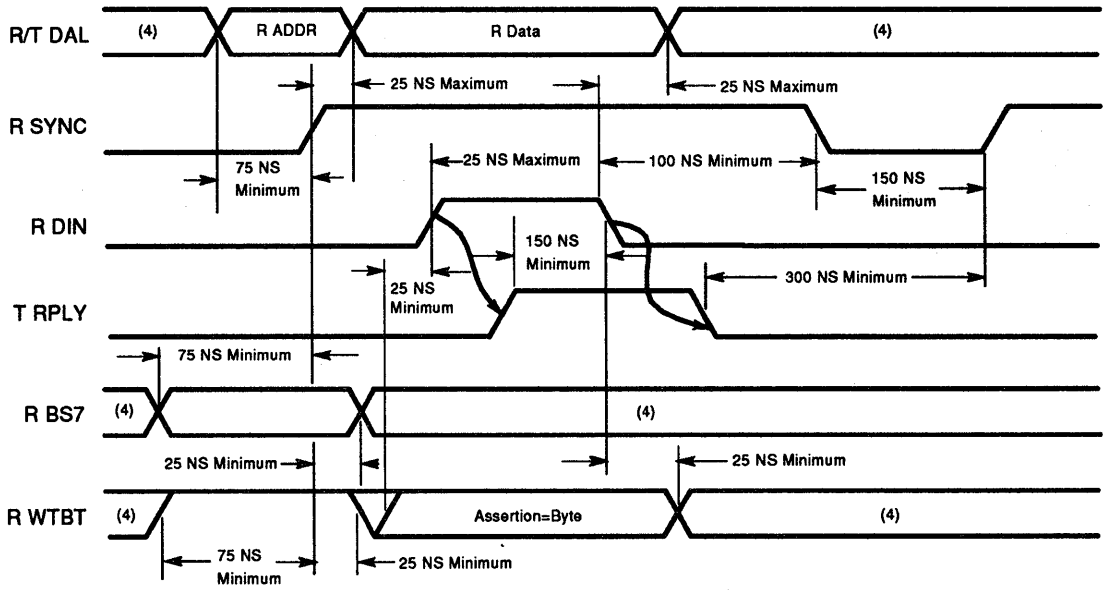


Figure 5–3 DATO or DATO(B) Bus Cycle



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.

LJ-00179-TI0

Figure 5–4 DATO or DATO(B) Bus Cycle Timing

5.3.1.4 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 Section 5.3.1.2, 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). Figure 5–5 shows the operations performed by the bus master and slave device during a DATIO or DATIO(B) bus cycle. Figure 5–6 shows the DATIO and DATIO(B) bus cycle timing.

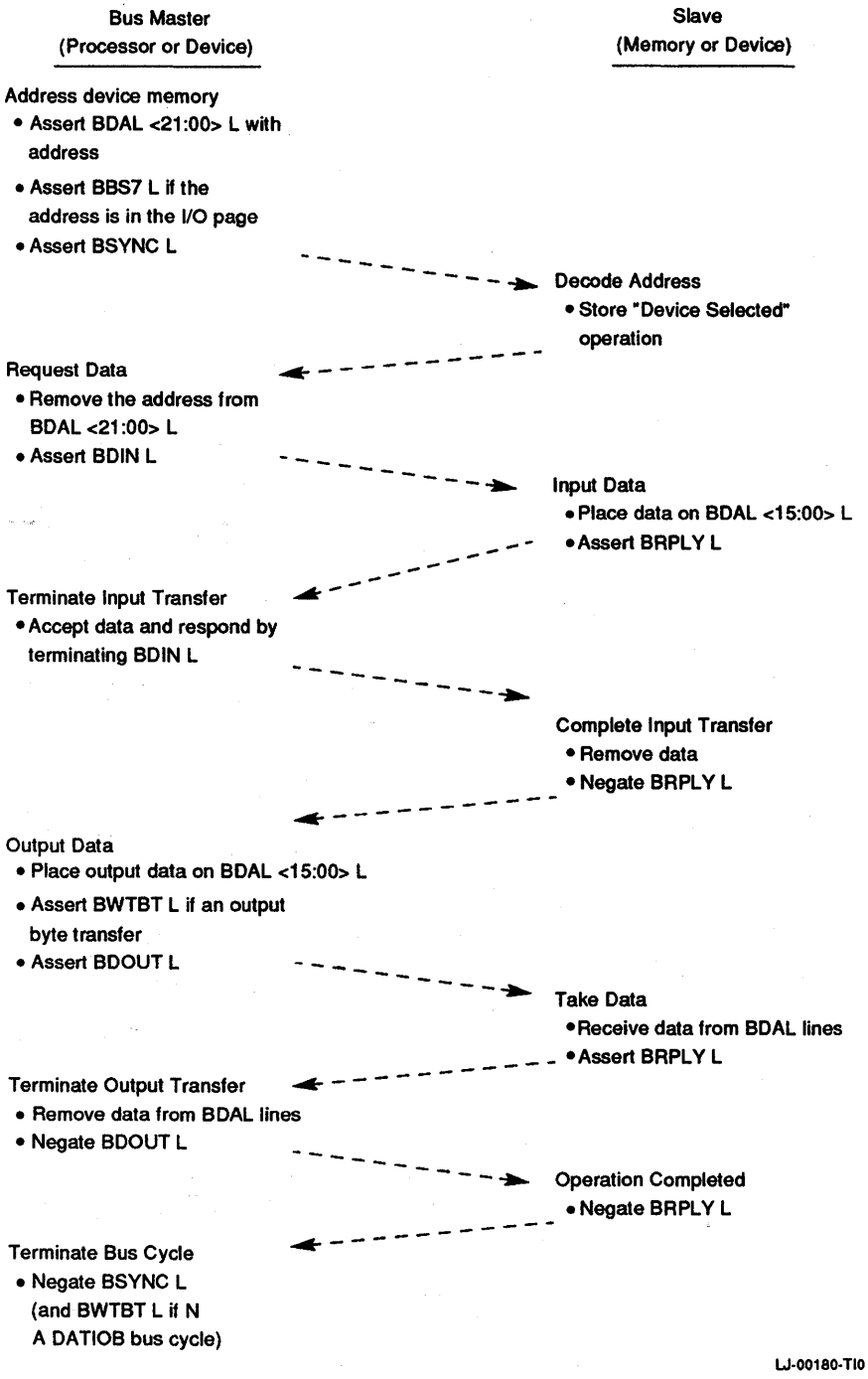
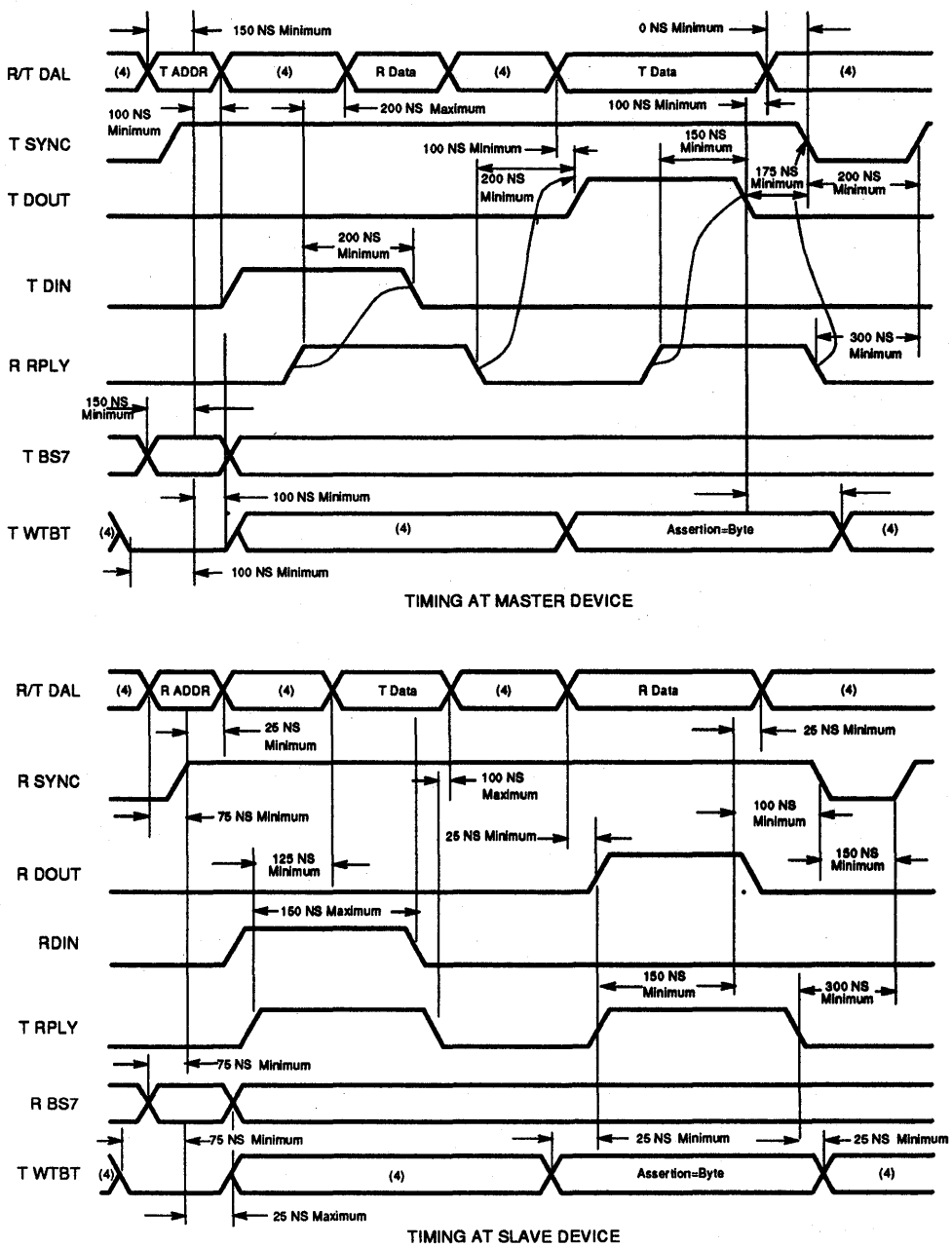


Figure 5–5 DATIO or DATIO(B) Bus Cycle



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.

LJ-00181-TI0

Figure 5–6 DATIO or DATIO(B) Bus Cycle Timing

5.4 Direct Memory Access

Direct Memory Access (DMA) capability allows direct data transfers between I/O devices and memory. This is useful when using mass storage devices (for example, 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. Figure 5–7 shows the operations performed by the processor and bus master during the DMA request/grant sequence. Figure 5–8 shows the DMA request/grant bus cycle timing.

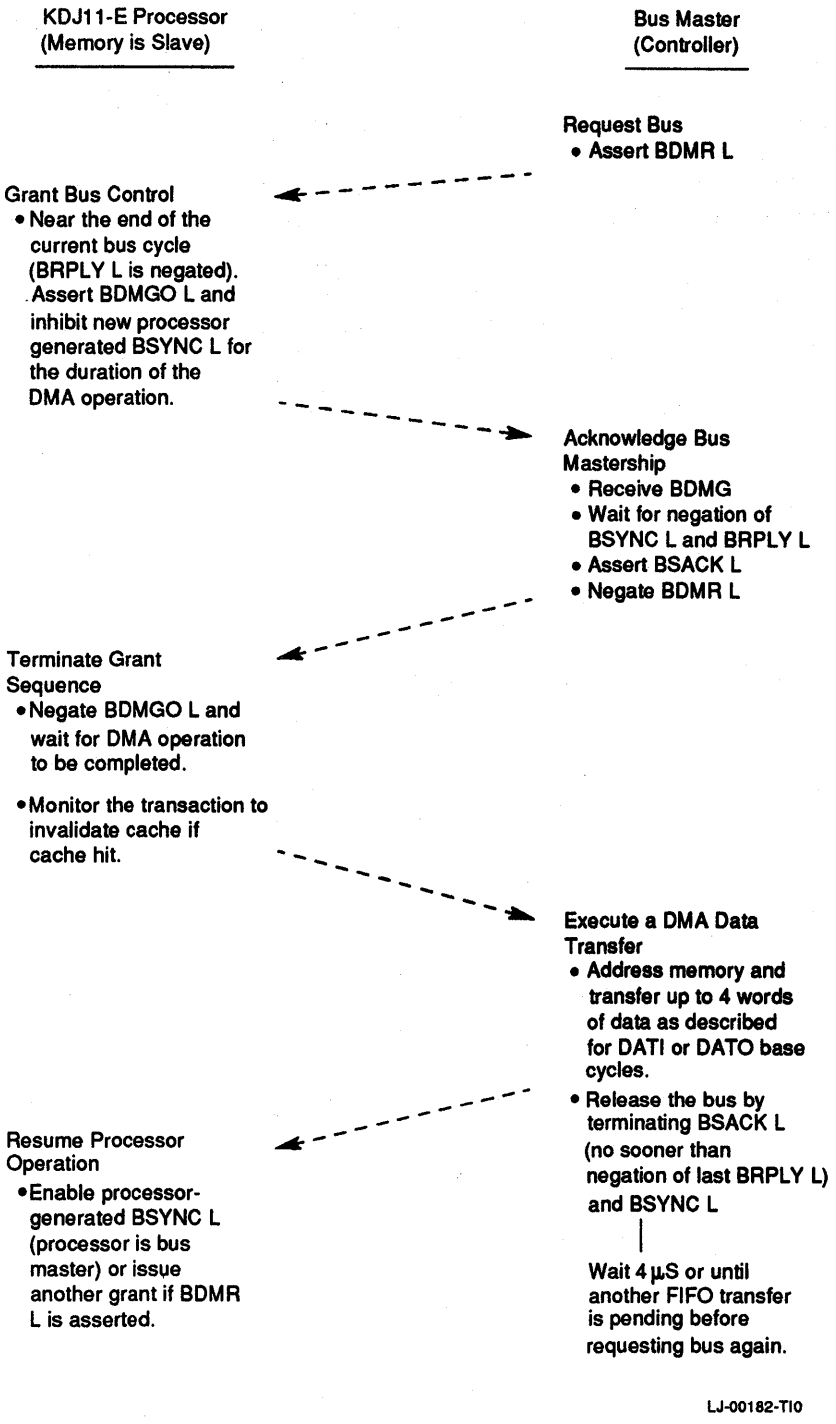


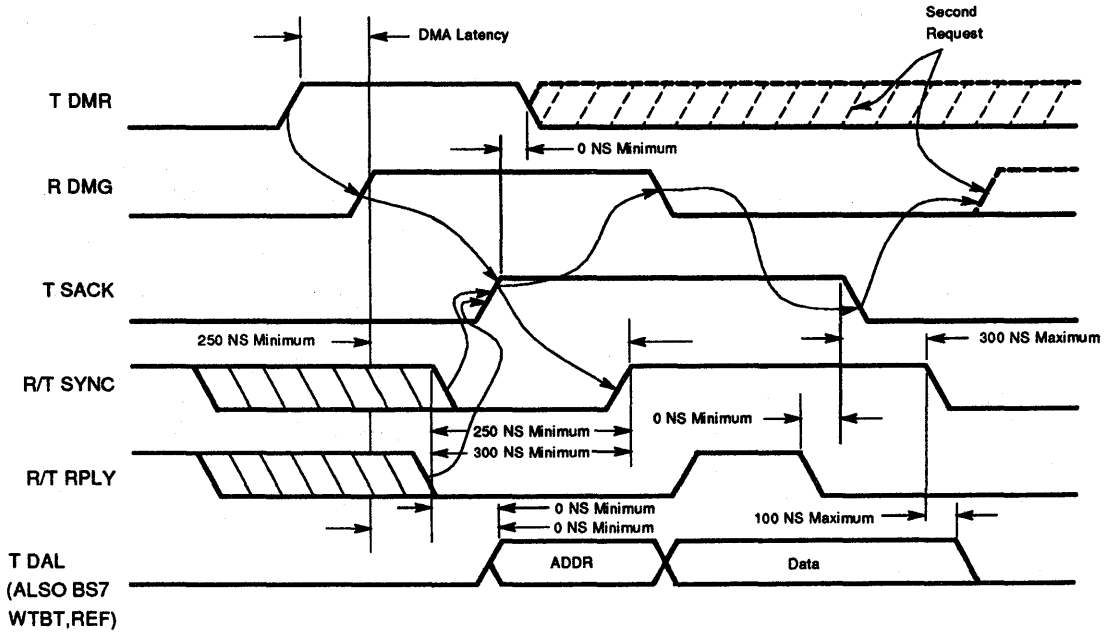
Figure 5–7 DMA Request/Grant Sequence

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-E 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-E uses a no SACK timer that clears BDMGO L if BSACK L is not received from the DMA device within 12.8 μseconds.

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.



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.

LJ-00183-TI0

Figure 5–8 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.

5.5 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 777. 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 <07:05> can be programmed to a priority level from 0 to 7. 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 LSI-11 bus signals used in interrupt transactions are listed here.

Signal	Name
BIRQ4 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 <15:00> L	Data/address lines
BDIN L	Data input strobe
BRPLY L	Reply

5.5.1 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-E 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 <07:05>). Single-level interrupt (position-defined) devices that interrupt on BIRQ4 can also be used in KDJ11-E systems, but must be placed in a bus slot following the last bus slot in which a position-independent device is installed.

5.5.2 Interrupt Protocol

Interrupt protocol has three phases: the interrupt request phase, the interrupt acknowledge and priority arbitration phase, and the interrupt vector transfer phase. Figure 5–9 shows the operations performed by the processor and interrupting device. Figure 5–10 shows the interrupt protocol timing.

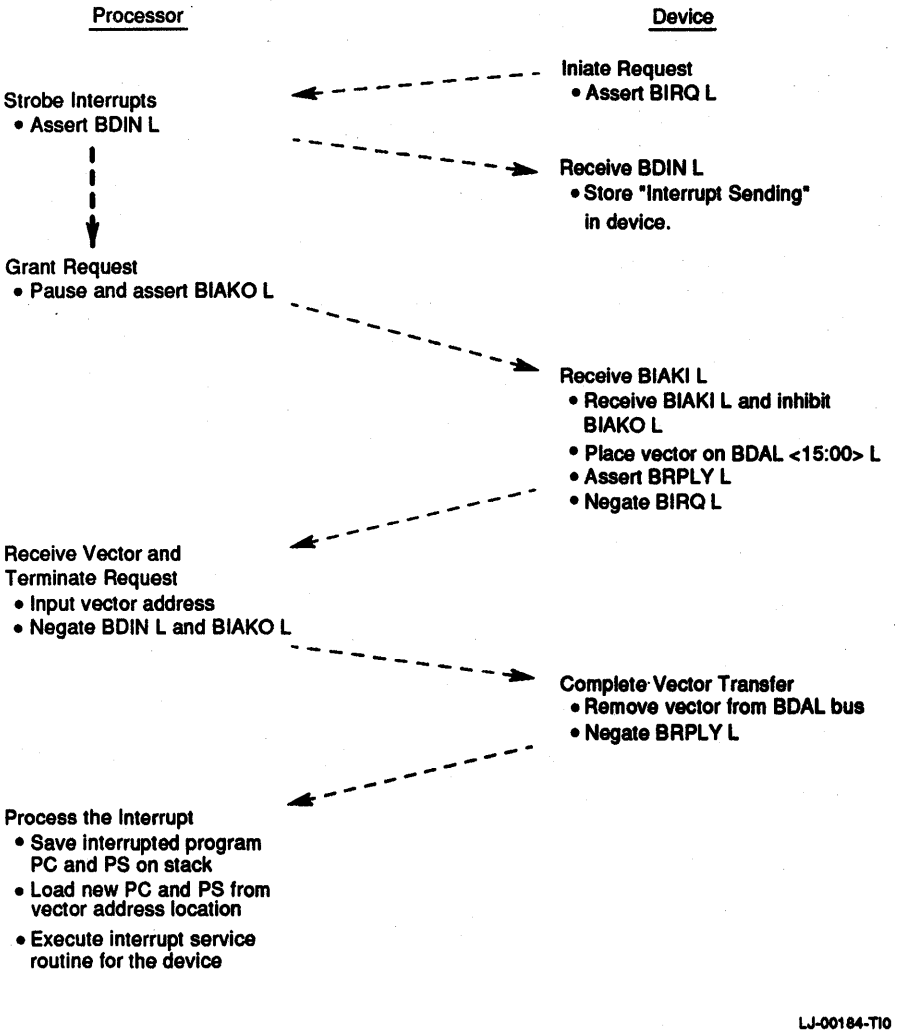
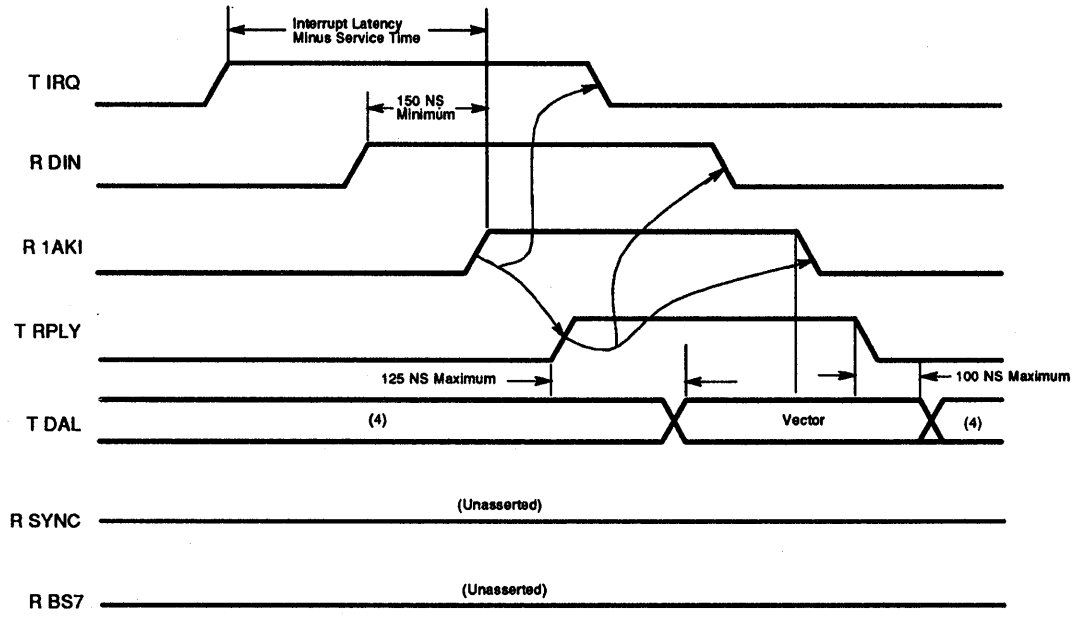


Figure 5–9 Interrupt Request/Acknowledge Sequence



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.

LJ-00185-TI0

Figure 5–10 Interrupt Protocol Timing

The interrupt request phase begins when a device meets its specific conditions for interrupt requests (for example, 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-E acknowledges interrupts under the following conditions:

1. The device interrupt priority is higher than the current priority level stored in PSW <07:05>.
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 acknowledgment 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 5–4 presents the Interrupt Request (IRQ) lines that a device at each level must assert in order to request an interrupt. Table 5–4 also 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 774, none of the remaining TDAL lines are used.

Table 5–4 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	

5.5.3 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.

Figure 5–11 shows the position-independent configuration. 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 Section 5.5.2. 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.

Figure 5–12 shows the position-dependent configuration. 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 the position-dependent configuration, each device must assert only 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.

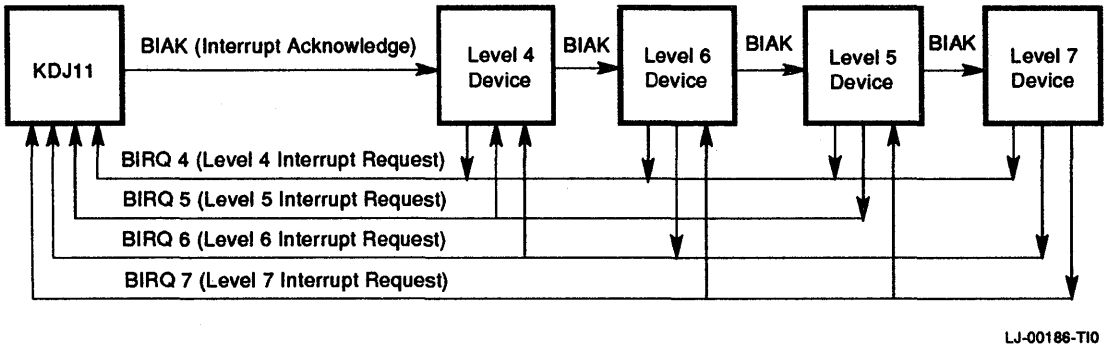


Figure 5–11 Position-Independent Configuration

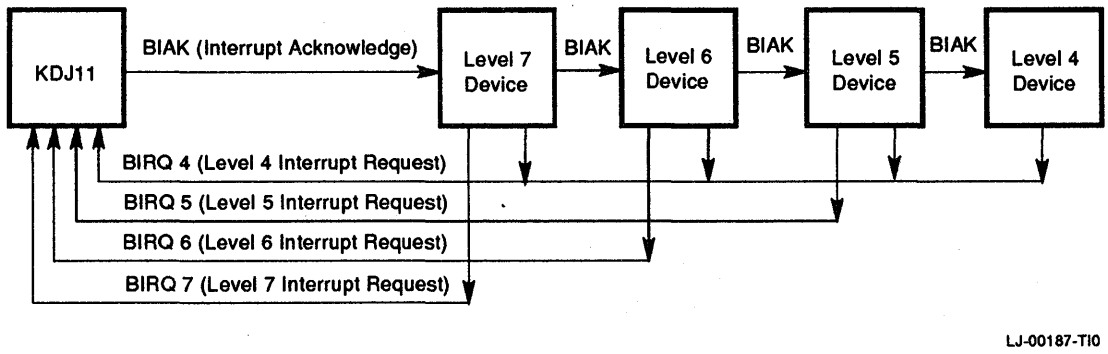


Figure 5–12 Position-Dependent Configuration

5.6 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

5.6.1 Halt

The 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.

5.6.2 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

5.6.3 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 in the following sections:

5.6.3.1 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.

5.6.3.2 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.

5.6.3.3 Power-Up

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 minimum of 3 ms. BDCOK H must remain asserted 4 ms (minimum) after the negation of BPOK H.

5.6.3.4 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.

5.6.4 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 100 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).

5.7 Bus Electrical Characteristics

This paragraph contains information about the electrical characteristics of the LSI-11 bus.

5.7.1 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)

5.7.2 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.

5.7.3 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.

5.7.4 120 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. Because bus drivers, receivers, and wiring connected to the bus have finite resistance and nonzero reactance, the transmission line impedance becomes nonuniform, and therefore 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-E processor (as well as all standard Digital-supplied LSI-11 interfaces) connects to the bus via special drivers and receivers described in Section 5.7.5 and Section 5.7.6.

The KDJ11-E processor provides resistive (250 Ω) pull-up on all bussed lines to 3.4 Vdc for this wired-OR interconnecting scheme.

5.7.5 Bus Drivers

Devices driving the 120 Ω LSI-11 bus must have open collector outputs and meet the specifications that follow.

DC Specifications[1]

- 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.

5.7.6 Bus Receivers

Devices that receive signals from the 120 Ω LSI-11 bus must meet the following requirements.

DC Specifications[2]

- 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.

5.7.7 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. Figure 5–13 shows the terminating resistors. The KDJ11-E contains terminating resistor networks which provide the 120 Ω (terminations for the data/address, synchronization, and control lines) at the processor end of the bus.

---

[1] These conditions must be met at worst-case supply voltage, temperature, and input signal levels

[2] These conditions must be met at worst-case supply voltage, temperature, and output signal levels

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.

5.7.7.1 Bus Interconnection Wiring

The bus interface for the module connectors is provided by one, two, or three backplanes, depending on the system configuration.

5.7.7.2 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, and so on), 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.

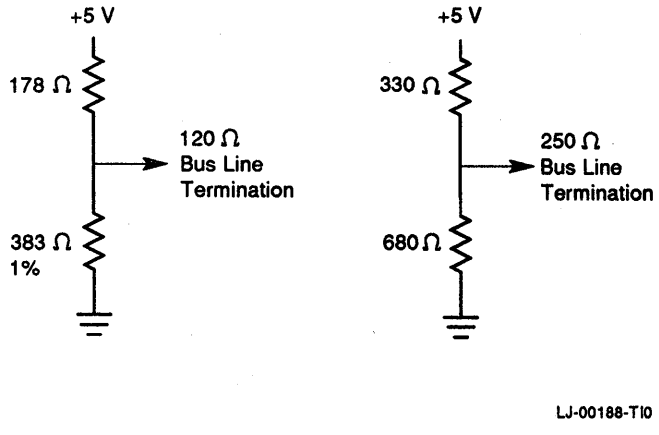


Figure 5–13 Bus Line Termination

5.7.7.3 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.

5.7.7.4 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.

5.7.7.5 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.

5.8 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 comprises 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.

5.8.1 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. Figure 5–14 shows a configuration diagram for a single-backplane.

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 .

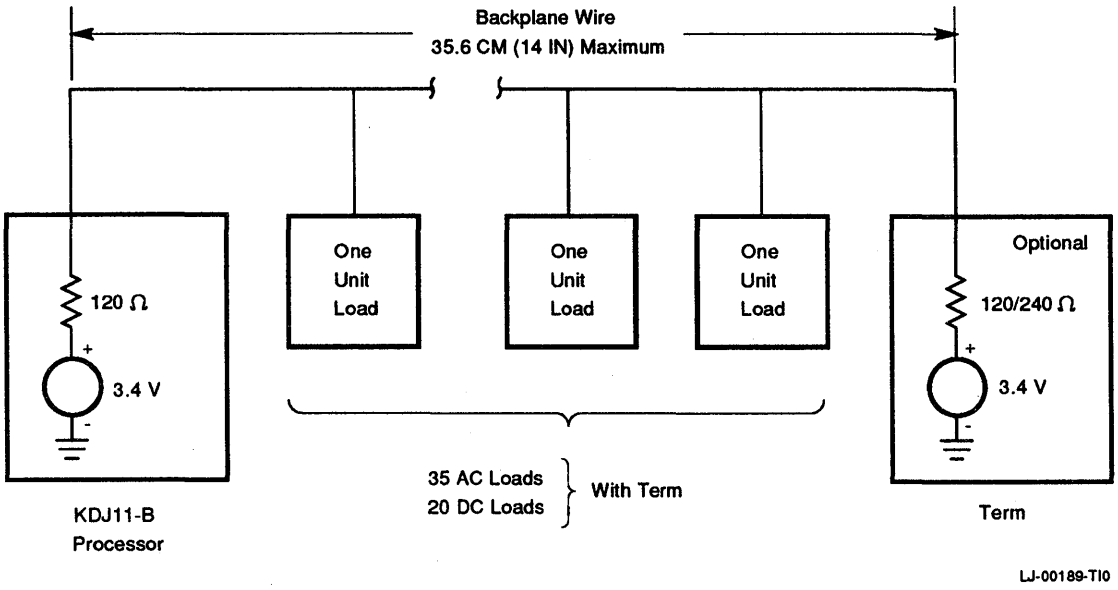


Figure 5–14 Single–Backplane Configuration

5.8.2 Rules for Configuring Multiple-Backplane Systems

Multiple-backplane systems can contain a maximum of three backplanes. Figure 5–15 shows a configuration diagram for a multiple-backplane system.

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).

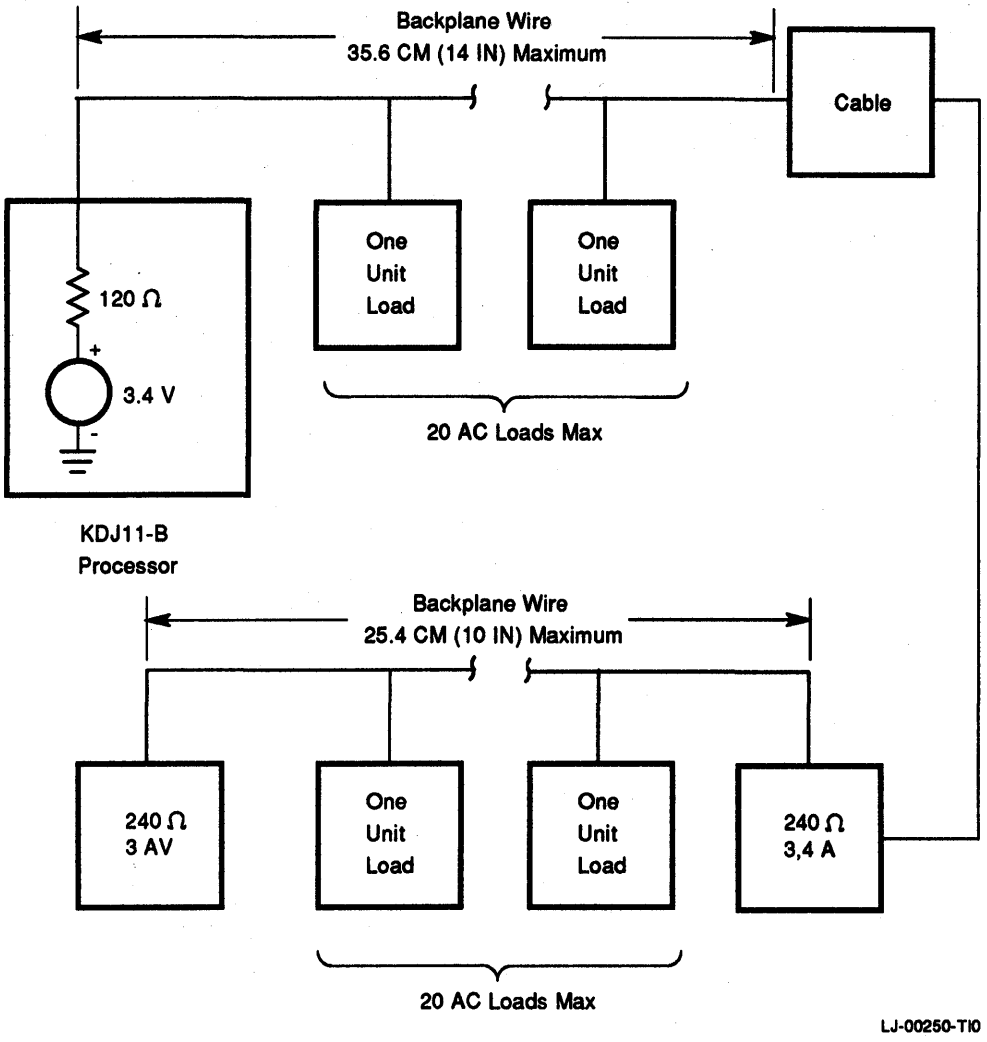


Figure 5–15 Multiple-Backplane Configuration

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 cables 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 .

5.8.3 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.

6
Private Memory Interconnect Bus

6.1 Description

The PMI bus provides a high performance communications path between the KDJ11-E CPU module 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.

6.2 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 LSI bus signals that are used with the PMI protocol use the A and B rows of the backplane defined as the LSI bus.

6.2.1 PMI Bus Master Signals

The PMI bus master controls the PMI bus cycles by using the nonmultiplexed control signals described in Table 6–1. These signals are asserted low and negated high.

6.2.2 PMI Slave Signals

The PMI slave responds to the bus master by the nonmultiplexed signals listed in Table 6–2. These signals are asserted low and negated high by any device that is capable of being a slave.

6.2.3 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-E and the UBA. These signals are nonmultiplexed as described in Table 6–3 and are not used in any LSI based system.

6.2.4 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 6–4.

Table 6–1 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 6–2 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>.

Table 6–2 (Cont.) PMI Slave Signals

Pin	Mnemonic	Function
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 6–3 PMI UNIBUS Adapter Signals

Pin	Mnemonic	Function
DD1	PMAPE L	**PMI UNIBUS Map Enable** The KDJ11-E 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-E 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-E 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-E 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-E 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

Table 6–3 (Cont.) PMI UNIBUS Adapter Signals

Pin	Mnemonic	Function
CR1	PBSY L	**PMI Busy** This signal is asserted by the PMI bus master (KDJ11-E or UBA) when it gains control of the PMI bus. The PMI bus master negates this signal when it relinquishes PMI mastership. The KDJ11-E is the bus master at power-up and when the bus is idle.

Table 6–4 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-E 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-E 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 AN2	BIAKI L BIAKO L	**Interrupt Acknowledge In** **Interrupt Acknowledge Out** These signals are only used in PMI UNIBUS systems during the interrupt grant cycles. The KDJ11-E asserts the BIAKI L signal and the UBA acknowledges it by asserting one of the UNIBUS bus grant signals.

Table 6–4 (Cont.) LSI Bus Signals

Pin	Mnemonic	Function
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.

6.3 PMI Operation in an LSI-11 System

The KDJ11-E is the default bus master in an LSI-11 system. Any bus device that has the appropriate circuits can become the bus master and can control data transfers over the LSI-11 bus. The KDJ11-E 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 5.

If the KDJ11-E 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-E 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-E asserts the BDMG L signal after it asserts the BRPLY L signal.
3. In a PMI DATIP cycle, the KDJ11-E negates the BRPLY L signal before the PMI slave removes the data from the bus.
4. In a PMI DATOB cycle, the KDJ11-E 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-E can regain bus mastership only after BSYNC L and BSACK L have been negated by the DMA device.

6.4 PMI Operation in a UNIBUS System

In a UNIBUS system the KDJ11-E 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 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.

6.4.1 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-E 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-E bus arbitrator asserts the DMA grant (DMGO) after receiving the DMR input and after the negation of BSACK by the UBA.
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-E.
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 selection 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-E 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-E bus arbitrator negates DMGO after BSACK is asserted. Since the UBA provides the timeout function, the KDJ11-E maintains DMGO until it receives BSACK.
10. The UBA negates NPG after the KDJ11-E 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-E 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-E is the next PMI bus master, the UBA or the current bus master clears the bus. The PMI control data negates PBSY to relinquish control of the PMI bus.

6.4.2 PMI Bus Device Interrupt

Any UNIBUS device that is capable of being a bus master can issue a BR7 through BR4 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-E 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-E 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-E 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-E 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-E after the negation of SACK and the assertion of PBSY.
12. The KDJ11-E 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-E 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-E, and the interrupt cycle is aborted.
15. The UNIBUS master negates BBSY after it clears the UNIBUS.

6.5 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.

6.5.1 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 memory. The PMI bus master detects an I/O page reference by the assertion of TBS7.

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 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-E is the PMI master, then PMI cycles are performed with the UBA responding as a slave, and follow the routine listed in steps 1 through 6.

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.
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-E 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.

6.5.2 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. The PMI bus master detects an I/O page reference by the assertion of TBS7.

The PMI bus master can only start DATBI transfers on even word boundaries. This means that address bits <1:0> must be equal to 0s. 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 1s. 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 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-E is the PMI master, the PMI cycles are performed with the UBA responding as a slave, and follow the routine listed in steps 1 through 6.
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.
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-E 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.

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.

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.

6.5.3 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 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-E is the PMI master, the PMI cycles are performed with the UBA responding as a slave, and follow the routine listed in steps 1 through 6.
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).
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.

6.6 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 5. The UNIBUS requirements for the request and grant are described in Section 6.4.2. The transfer of the interrupt vector from the requesting UNIBUS device to the KDJ11-E 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-E latches the vector data within 200 ns after the UBA-asserted BRPLY.
9. The KDJ11-E negates BDIN and BIAKO after it latches the vector data.
10. The UBA negates BRPLY after BIAKO is negated.

6.7 PMI Power-Up/Power-Down

The power-up/power-down protocol for the PMI bus in an LSI system is described in Chapter 5. 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-E 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-E can execute the power-down sequence.

In the UNIBUS system, the KDJ11-E receives DC LO as the DCOK signal, and the BPOK signal is isolated from AC LO by the UBA. When a UNIBUS device asserts AC LO to the UBA, it asserts BPOK for a minimum of 2 ms before it allows AC LO to negate BPOK.

7
Addressing Modes

7.1 Introduction

The KDJ11-E uses the six addressing modes described here 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 Program Counter (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 two. 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 a stack pointer (SP) - General purpose registers can be used for stack operations.

7.2 Addressing Modes

Data stored in memory must be accessed and manipulated. Data handling is specified by a KDJ11-E instruction (MOV, ADD, and so on), and usually includes the following:

- The function to be performed (operation code)
- The general purpose register to be used when locating the source operand, or destination operand (where required), or both.
- 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-E 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-E 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.

KDJ11-E 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 (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.

7.2.1 Single-Operand Addressing

The instruction format for all single-operand instructions (such as CLR, INC, TST) is shown in Figure 7–1. 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 (indirect) deferred addressing.
- Bits <2:0> specify which of the eight general purpose registers is to be referenced by this instruction word.

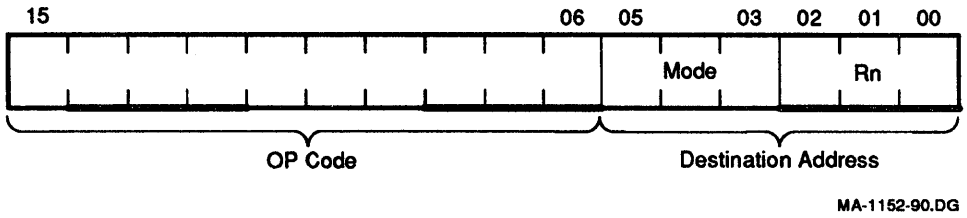


Figure 7–1 Single-Operand Addressing

7.2.2 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. Figure 7–2 shows the instruction format for the double-operand instruction .

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-E instructions given in Table 7–1. (A complete list of KDJ11-E instructions appears in Chapter 8 Table 8–1.

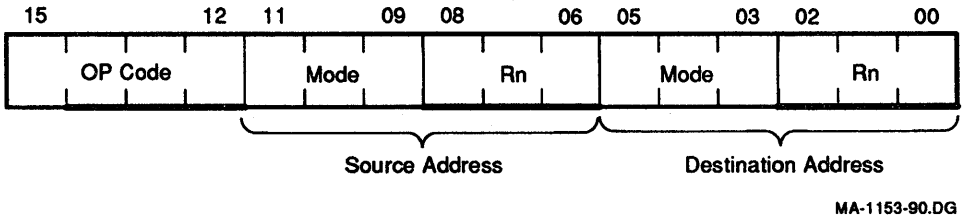


Figure 7–2 Double-Operand Addressing

Table 7–1 Sample KDJ11-E Instructions

Mnemonic	Description	Octal Code[1]
CLR	Clear - Zero the specified destination.	0050DD
CLRB	Clear byte - Zero the byte in the specified destination.	1050DD

[1]DD = Destination field (six bits)
SS = Source field (six bits)

Table 7–1 (Cont.) Sample KDJ11-E Instructions

Mnemonic	Description	Octal Code[1]
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

[1]DD = Destination field (six bits)
SS = Source field (six bits)

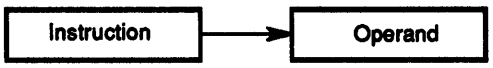
7.2.3 Direct Addressing

There are four basic modes used with direct addressing:

- Register mode (mode 0)
- Autoincrement mode (mode 2)
- Autodecrement mode (mode 4)
- Index mode (mode 6)

These direct modes are illustrated in Figure 7–3 through Figure 7–6, and are summarized in the following paragraphs.

Mode	Name	Assembler Syntax	Function
0	Register	Rn	Register is operand.



MA-1154-90

Figure 7–3 Mode 0, Register

Mode	Name	Assembler Syntax	Function
2	Autoincrement	(Rn+)	Register is used as a pointer to sequential data and then is incremented

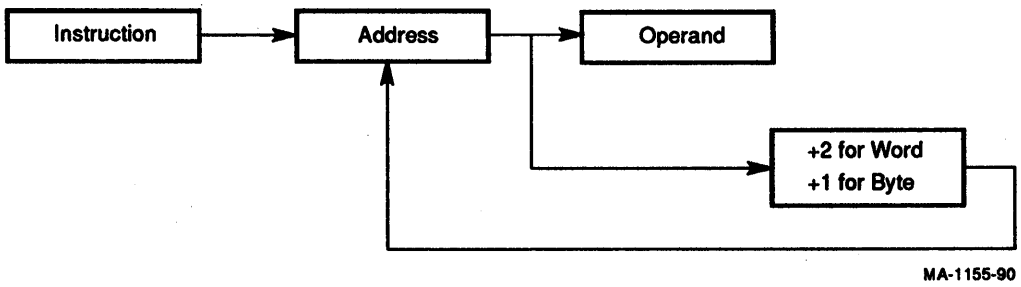


Figure 7–4 Mode 2, Autoincrement

Mode	Name	Assembler Syntax	Function
4	Autodecrement	-(Rn)	Register is decremented and then used as a pointer

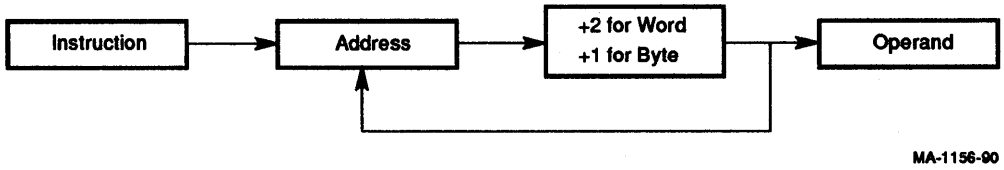


Figure 7–5 Mode 4, Autodecrement

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

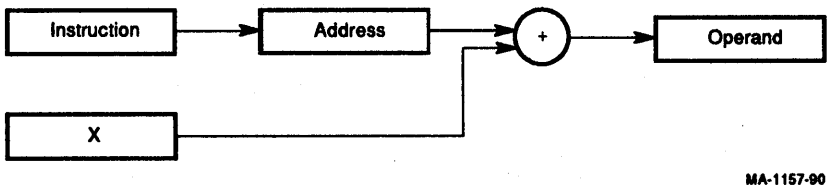


Figure 7–6 Mode 6, Index

7.2.3.1 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 a general register be defined as follows:

R0 = %0 (% sign indicates register definition)
R1 = %1
R2 = %2, and so on.

Registers are typically referred to by names 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 Figure 7–7 through Figure 7–9.

Register Mode Examples:

Symbolic	Octal Code	Instruction Name
INC R3	005203	Increment

Operation: Add one to the contents of R3.

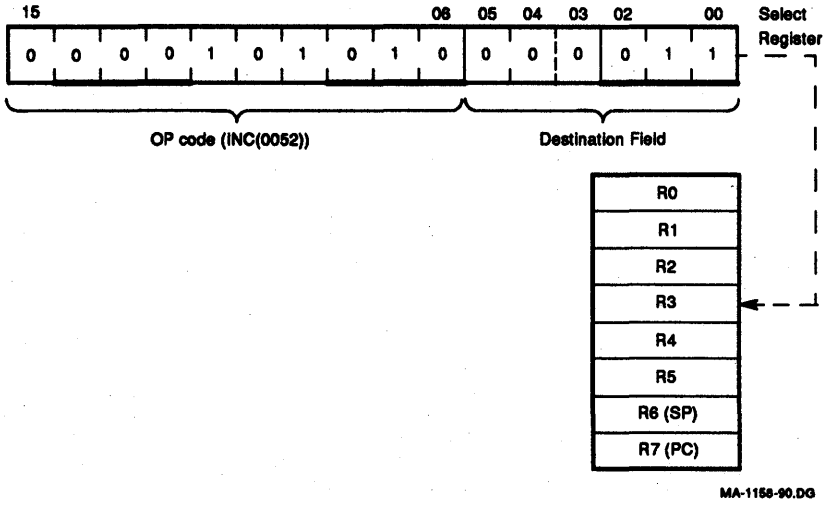


Figure 7–7 INC R3

Symbolic	Octal Code	Instruction Name
ADD R2,R4	060204	Add

Operation: Add the contents of R2 to the contents of R4.

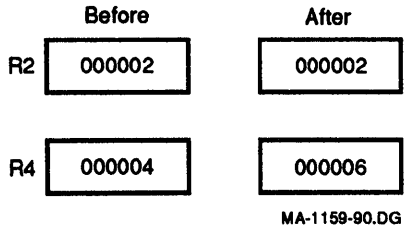


Figure 7–8 ADD R2,R4

Symbolic	Octal Code	Instruction Name
COMB R4	105104	Complement byte

Operation: 1's complement <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.

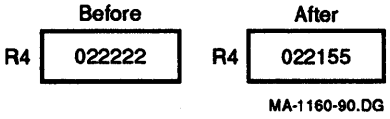


Figure 7–9 COMB R4

7.2.3.2 Autoincrement Mode [OPR (Rn)+]

The autoincrement mode (mode 2) provides for automatic stepping of a pointer through sequential elements of a table of operands. It assumes that the contents of the selected general purpose register is the address of the operand. Contents of registers are stepped (by one for byte instructions, by two for word instructions, always 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 Figure 7–10 through Figure 7–12.

Autoincrement Mode Examples:

Symbolic	Octal Code	Instruction Name
CLR (R5)+	005025	Clear

Operation: Use the contents R5 as the address of the operand. Clear the selected operand and then increment the contents of R5 by two.

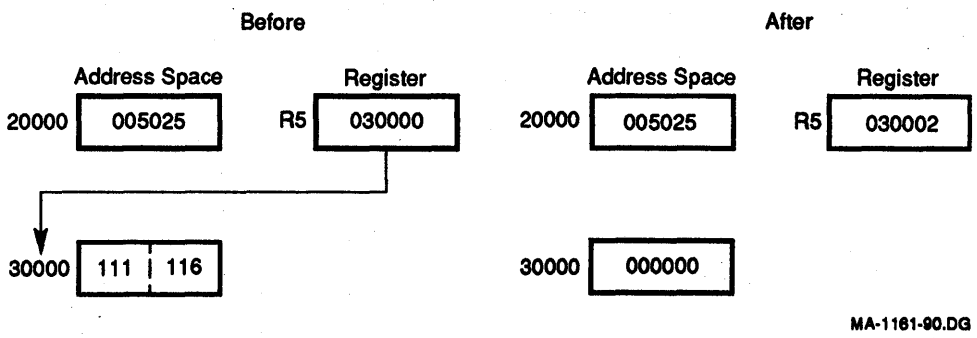


Figure 7–10 CLR (R5)+

Symbolic	Octal Code	Instruction Name
CLRB (R5)+	105025	Clear byte

Operation: Use the contents R5 as the address of the operand. Clear the selected byte operand and then increment the contents of R5 by one.

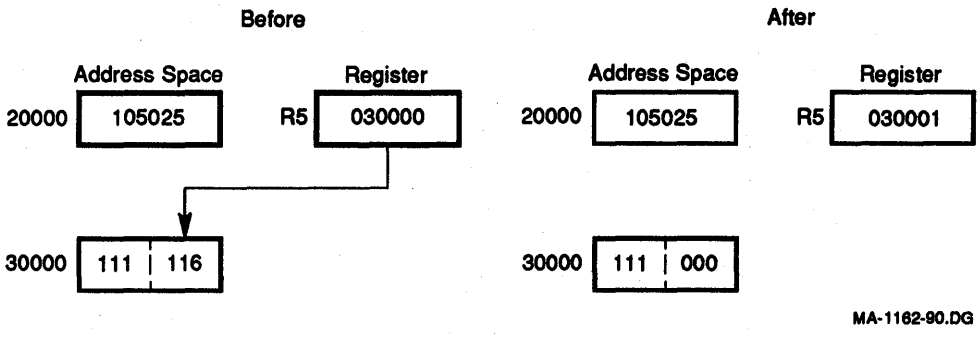


Figure 7–11 CLRB (R5)+

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.

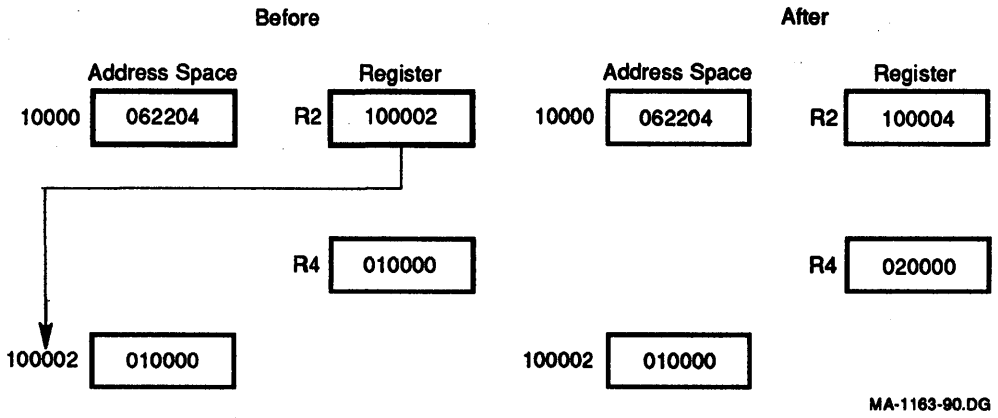


Figure 7–12 ADD(R2)+,R4

7.2.3.3 Autodecrement Mode [OPR -(Rn)]

The autodecrement 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-E are intended to facilitate hardware/software operations. Three autodecrement mode operations are illustrated in Figure 7–13 through Figure 7–15.

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.

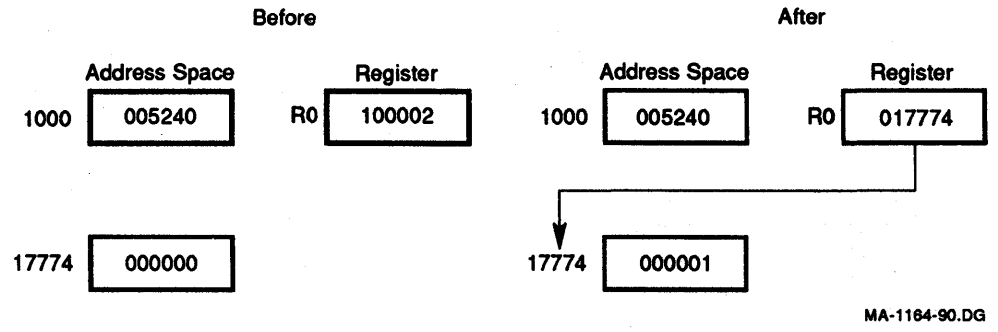


Figure 7–13 INC -(R0)

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.

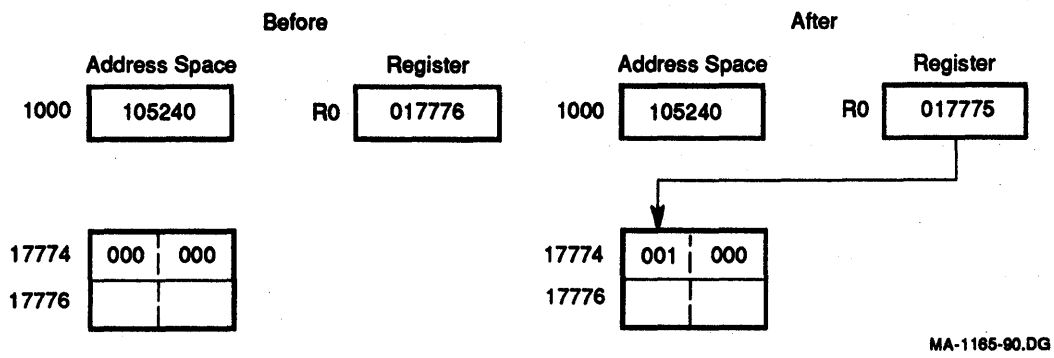


Figure 7–14 INCB -(R0)

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).

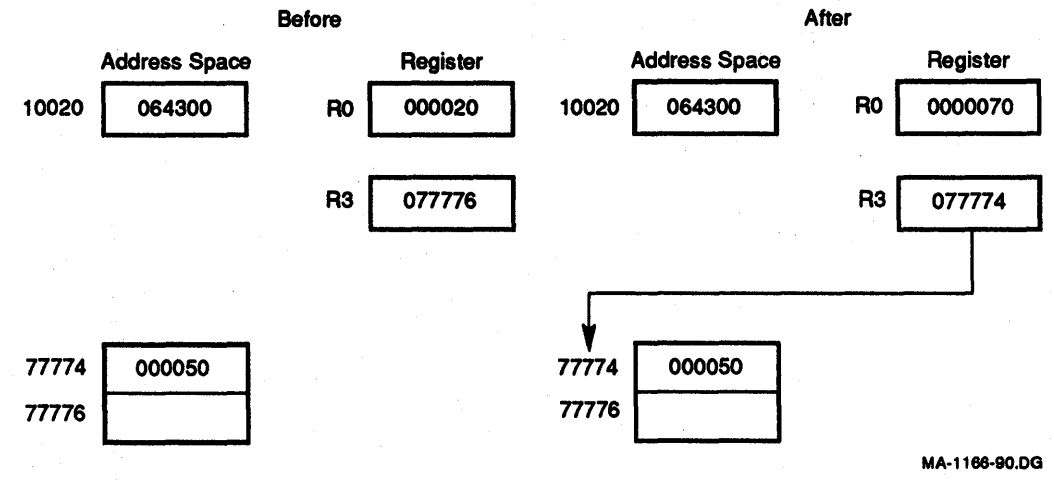


Figure 7–15 ADD -(R3),R0

7.2.3.4 Index Mode [OPR X(Rn)]

In the index 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 maybe 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 Figure 7–16 through Figure 7–18.

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.

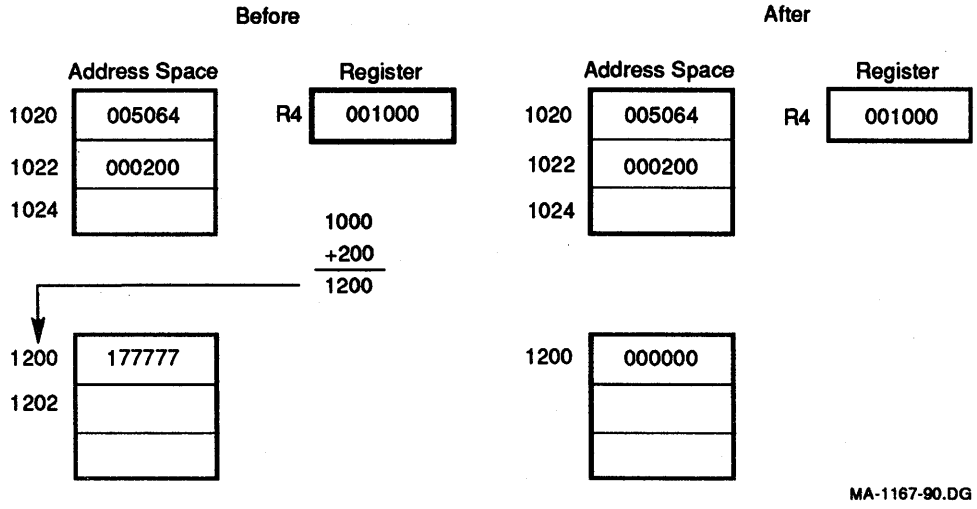


Figure 7–16 CLR 200(R4)

Symbolic	Octal Code	Instruction Name
COMB 200(R1)	105161 000200	Complement byte

Operation: The contents of a location, determined by adding 200 to R1, is replaced by its logical 1's complement. Each 0 bit is set and each 1 bit is cleared.

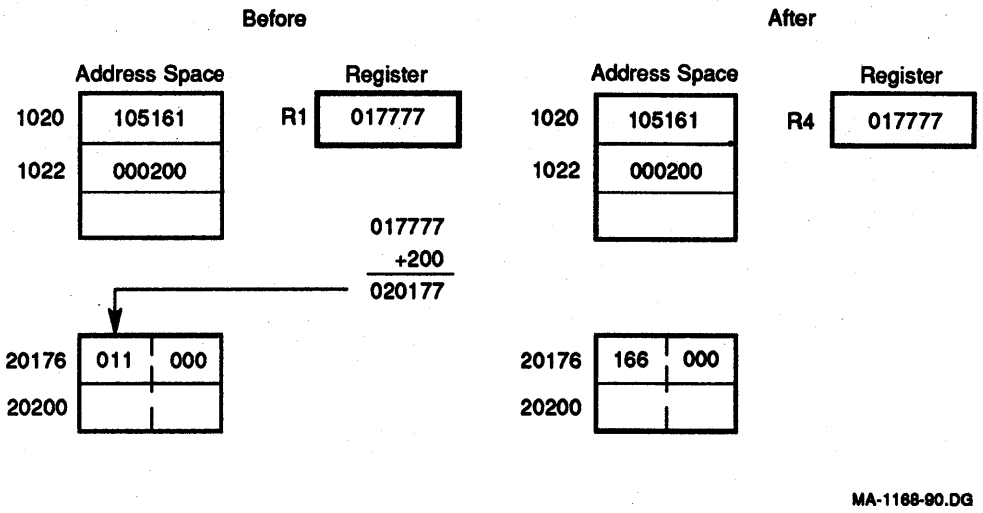


Figure 7–17 COMB 200(R1)

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).

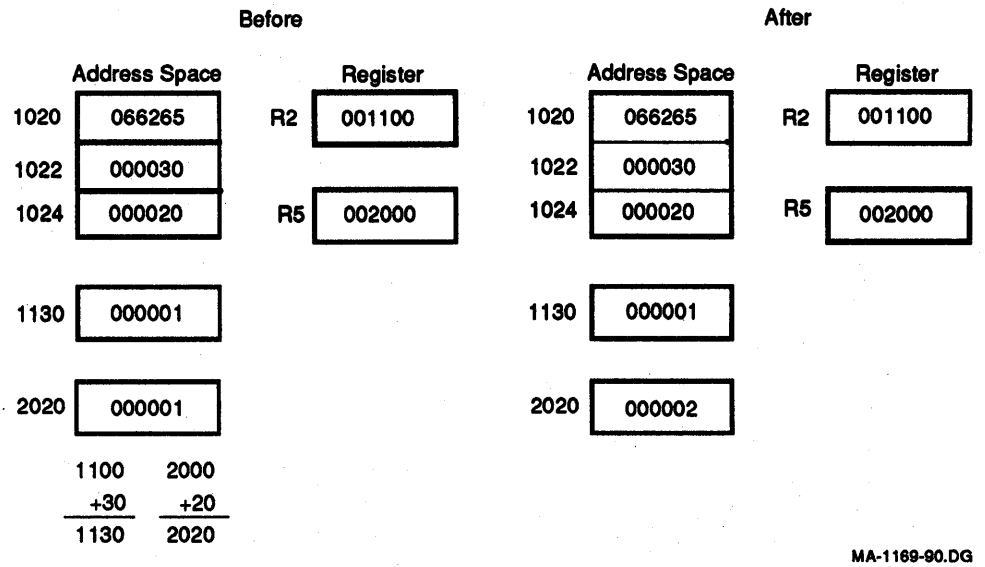


Figure 7–18 ADD 30(R2),20(R5)

7.2.4 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 other three 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 an at sign (@), or parentheses ( ).

The following section summarizes the deferred versions of the basic modes. These deferred modes are illustrated in Figure 7–19 through Figure 7–22.

Mode	Name	Assembler Syntax	Function
1	Register-deferred	@Rn or (Rn)	Register contains the address of the operand

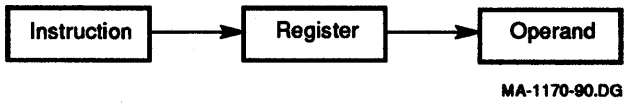


Figure 7–19 Mode 1, Register-Deferred

mode	Name	Assembler Syntax	Function
3	Autoincrement-deferred	@(Rn)+	Register is used as a pointer to a word containing the address of the operand, and then is incremented (always by two, even for byte instructions)

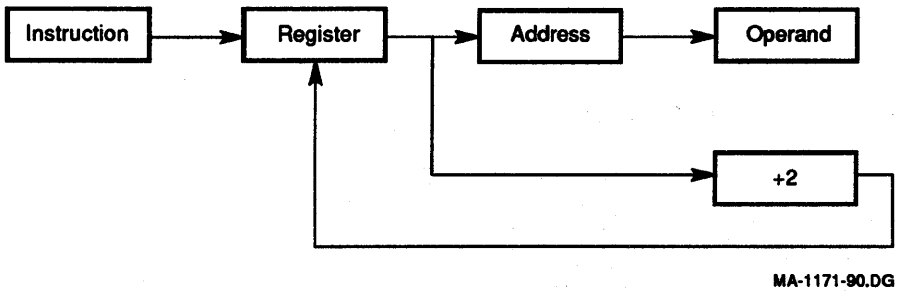


Figure 7–20 Mode 3, Autoincrement-Deferred

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.

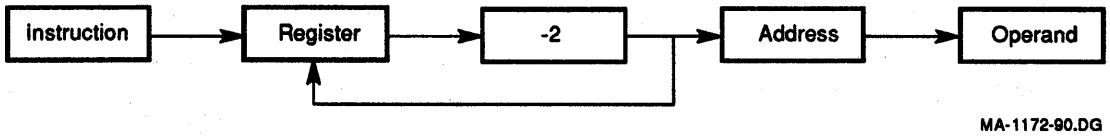


Figure 7–21 Mode 5, Autodecrement-Deferred

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.

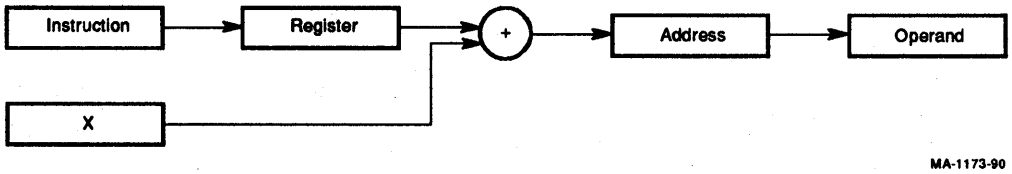


Figure 7–22 Mode 7, Index-Deferred

The following examples shows in Figure 7–23 through Figure 7–26, further illustrate use of the deferred modes.

Register-Deferred Mode Example:

Symbolic	Octal Code	Instruction
CLR @R5	005015	Clear

Operation: The contents of a location specified in R5 is cleared.

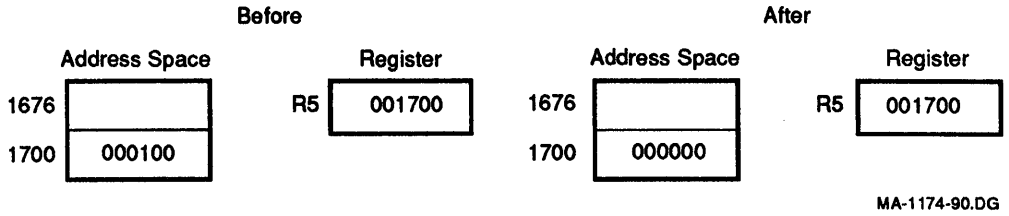


Figure 7–23 CLR @R5

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 operand. The operand is increased by one; the contents of R2 is incremented by two.

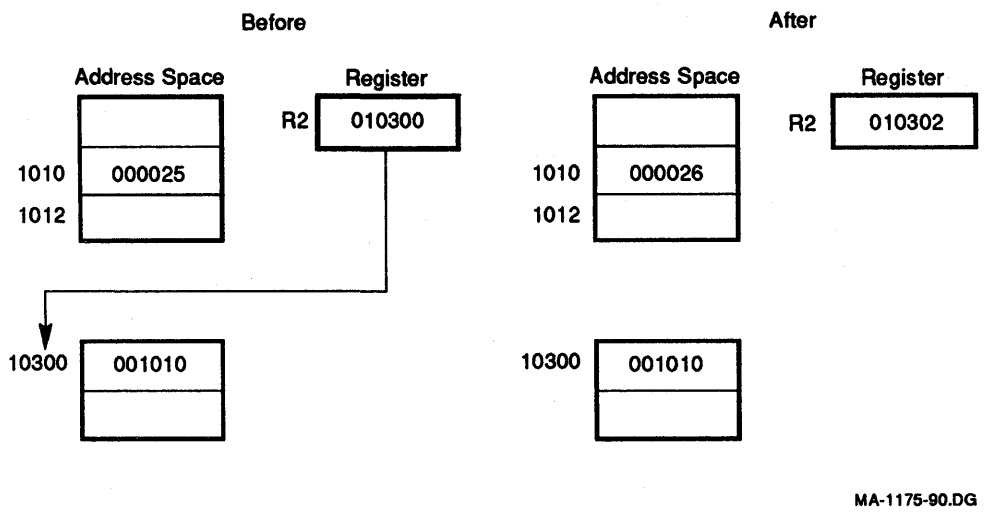


Figure 7–24 INC @(R2)+

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.

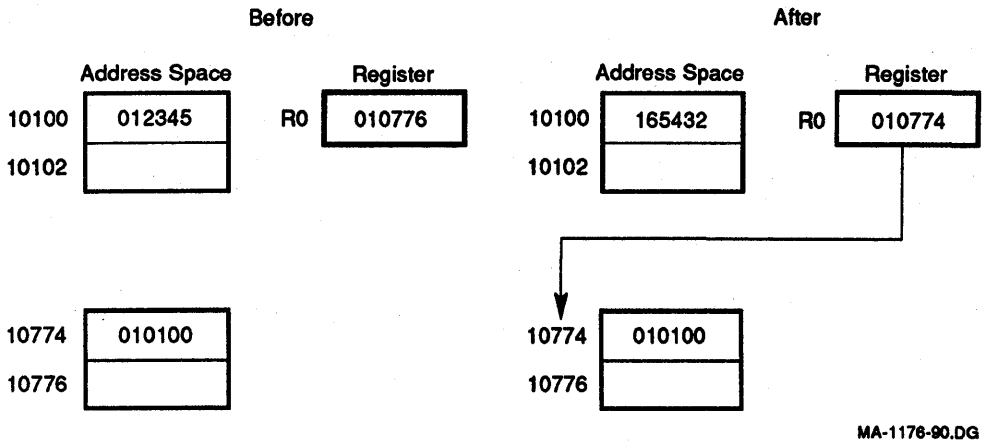


Figure 7–25 COM @-(R0)

Index-Deferred Mode Example:

Symbolic	Octal Code	Instruction Name
ADD @10000(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.

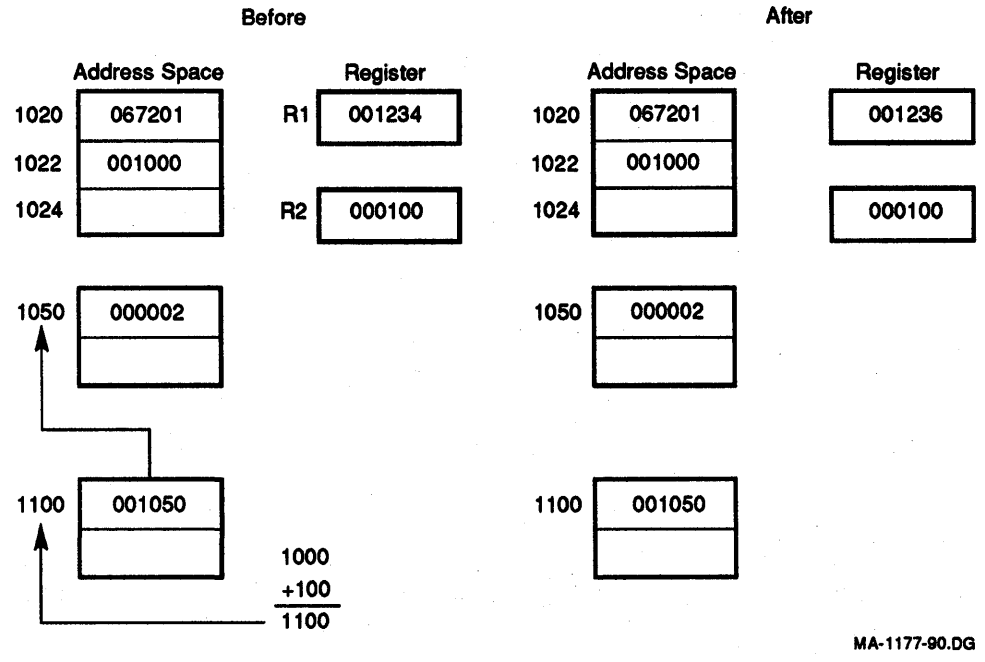


Figure 7–26 ADD @1000(R2),R1

7.2.5 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-E. Whenever the processor uses the PC to acquire a word from memory, the PC is automatically incremented by two to point to 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 incremented by two.)

The PC responds to all the standard KDJ11-E 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 the 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-E can accomplish the relocation of a program very efficiently through the the use of the 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.

7.2.5.1 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. Figure 7–27 illustrates an immediate mode operation.

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 the next instruction.

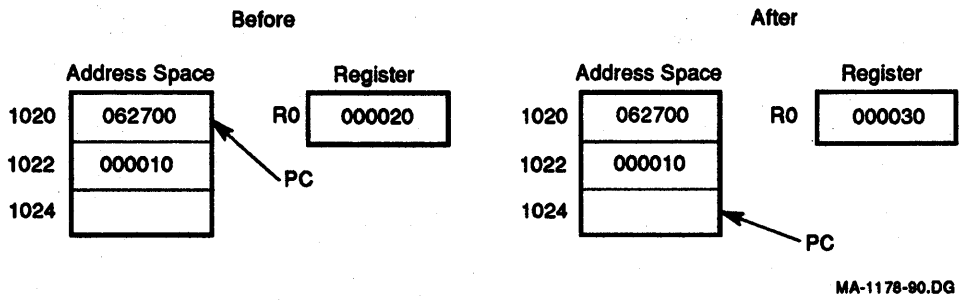


Figure 7–27 ADD #10,R0

7.2.5.2 Absolute Mode [OPR @#A]

Using the PC, the absolute 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. Figure 7–28 and Figure 7–29 illustrate two absolute mode operations.

Absolute Mode Examples:

Symbolic	Octal Code	Instruction Name
CLR @#1100	005037 001100	Clear

Operation: Clear the contents of location 1100.

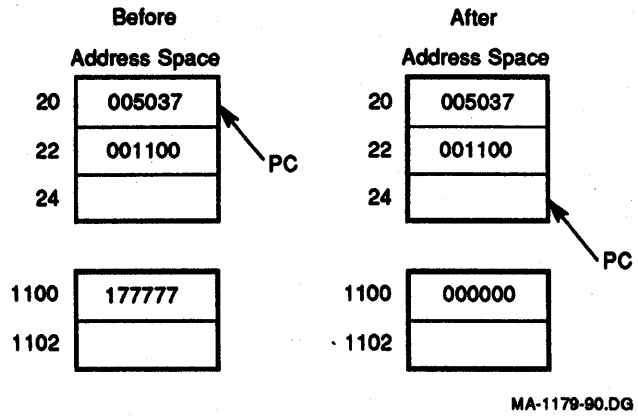


Figure 7–28 CLR @#1100

Symbolic	Octal Code	Instruction Name
Add @#2000,R3	063703 002000	Add

Operation: Add the contents of location 2000 to R3.

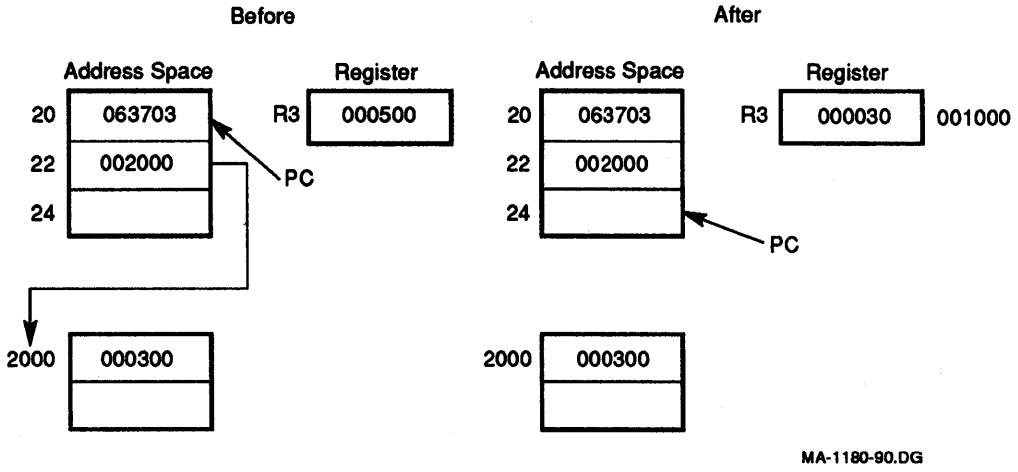


Figure 7–29 ADD @#2000

7.2.5.3 Relative Addressing Mode [OPR A or OPR X(PC)]

Using R7, the relative addressing 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 instruction OPR X(PC) is interpreted, "X is the location of A relative to the PC". Figure 7–30 illustrates a relative mode operation.

Relative Addressing Mode Example:

Symbolic	Octal Code	Instruction
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.

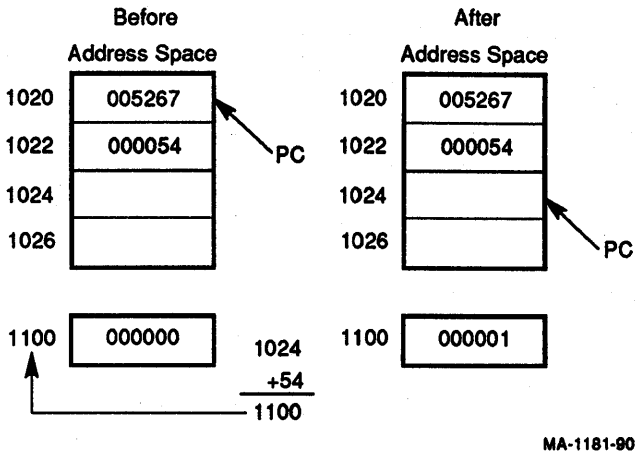


Figure 7–30 INC A

7.2.5.4 Relative-Deferred Addressing Mode [OPR @A or OPR @X(PC)]

The relative-deferred addressing 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." Figure 7–31 illustrates a relative-deferred mode operation.

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.

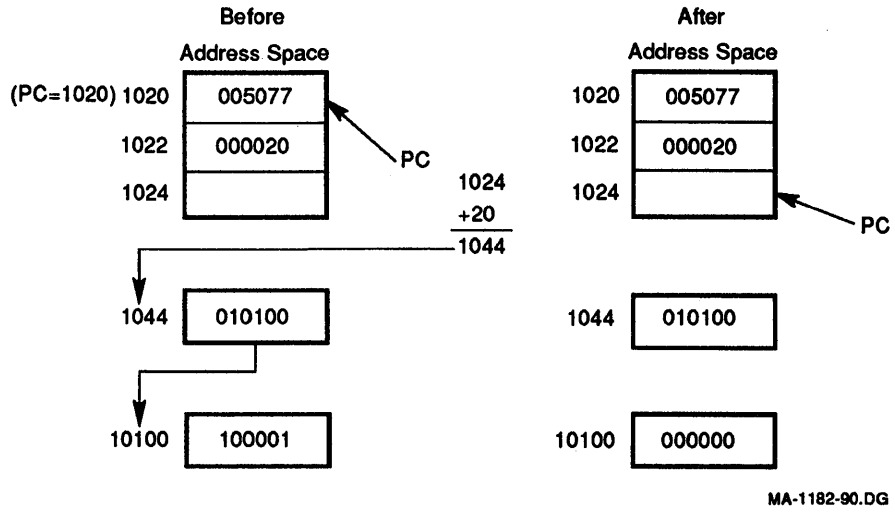


Figure 7–31 CLR @A

7.2.6 Use of the General Purpose Registers as a Stack Pointer

The processor SP (R6) is, in most cases, the general purpose register used for the stack operations related to program nesting. Autodecrement using R6 "pushes" data onto the stack, and autoincrementing 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.

8
Base Instruction Set

8.1 Instruction Set

This chapter describes the KDJ11-E base instruction set. The chapter includes an explanation of each 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 Section 8.4, and an alphabetical list is provided in Table 8–1.

Table 8–1 Instruction Set

Mnemonic	Instruction	Op Code
ADC(B)	Add carry	B055DD
ADD	Add source to destination	06SSDD
ASH	Arithmetic shift	072RSS
ASHC	Arithmetic shift combined	073RSS
ASL(B)	Arithmetic shift left	B063DD
ASR(B)	Arithmetic shift right	B062DD
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	B4SSDD
BIS(B)	Bit set	B5SSDD
BIT(B)	Bit test	B3SSDD
BLE	Branch if less than or equal (to zero)	003400

Table 8–1 (Cont.) Instruction Set

Mnemonic	Instruction	Op Code
BLO	Branch if lower	103400
BLOS	Branch if lower or same	103400
BLT	Branch if less than (zero)	101400
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	B050DD
CLV	Clear V	000242
CLZ	Clear Z	000244
CMP(B)	Compare source to destination	B2SSDD
COM(B)	Complement destination	B051DD
CSM	Call to supervisor mode	0070DD
DEC(B)	Decrement destination	B053DD
DIV	Divide	071RSS
EMT	Emulator trap	104000-104377
HALT	Halt	000000
IOT	Input/output trap	000004
INC(B)	Increment destination	B052DD
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	B1SSDD

Table 8–1 (Cont.) Instruction Set

Mnemonic	Instruction	Op Code
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	B054DD
NOP	No operation	000240
RESET	Reset external bus	000005
ROL(B)	Rotate left	B061DD
ROR(B)	Rotate right	B060DD
RTI	Return from interrupt	000002
RTS	Return from subroutine	00020R
RTT	Return from interrupt	000006
SBC(B)	Subtract carry	B056DD
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 $\neq$ 0)	077R00
SPL	Set priority level	00023N
SUB	Subtract source from destination	16SSDD
SWAB	Swap bytes	0003DD
SXT	Sigh extend	0067DD
TRAP	Trap	104400-104777
TST(B)	Test destination	B057DD
TSTSET	Test destination, set low bit	0072DD
WAIT	Wait for interrupt	000001
WRTLCK	Write interlocked	0073DD
XOR	Exclusive OR	074RDD

The diagram that accompanies each instruction shows the octal op code, and bit assignments.

NOTE
In byte instructions, the most significant bit (bit 15) is always a 1.

Symbols:	
( ) = contents of	∨ = Boolean OR
SS or src = source address	∀ = exclusive OR
DD or DST = destination address	~ = Boolean not
loc = location	REG or R = register
← = becomes	B = byte
↑ = "is popped from stack"	B = 0 for word, 1 for byte
↓ = "is pushed onto stack"	, = concatenated
∧ = Boolean AND	

8.2 Instruction Formats

The following formats include all instructions used in the KDJ11-E. Refer to individual instructions for more detailed information.

1. Single-Operand Group: 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

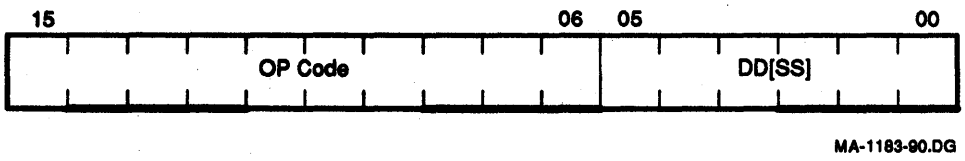


Figure 8–1 Single-Operand Group

2. Double-Operand Groups:
   a. Group 1: BIT, BITB, BIC, BICB, BIS, BISB ADD, SUB, MOV, MOVB, CMP, CMPB

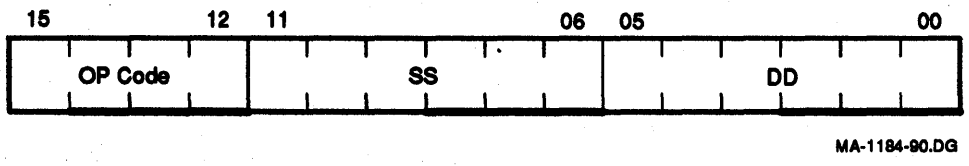


Figure 8–2 Double-Operand Group 1

b. Group 2: ASH, ASHC, DIV, MUL, XOR

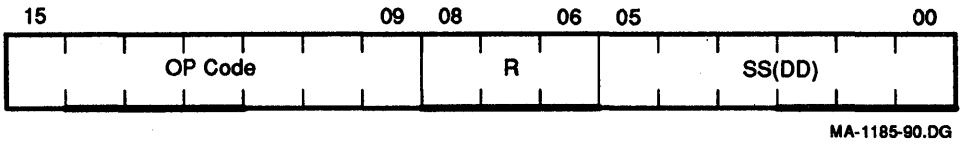


Figure 8–3 Double-Operand Group 2

3. Program Control Groups:

a. Branch (all branch instructions)

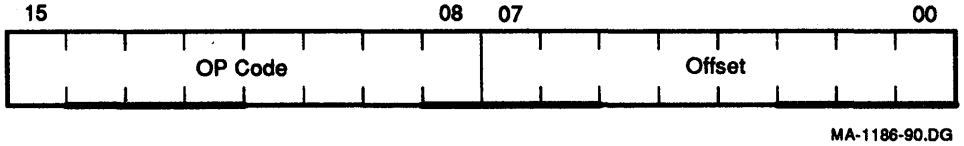


Figure 8–4 Program Control Group Branch

b. Jump (JMP)

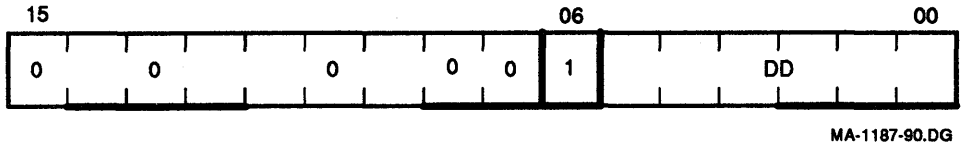


Figure 8–5 Program Control Group JMP

c. Jump to Subroutine (JSR)

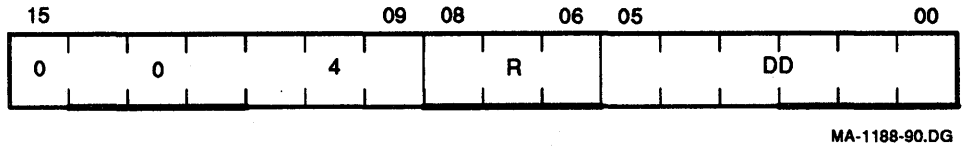


Figure 8–6 Program Control Group JSR

d. Subroutine Return (RTS)

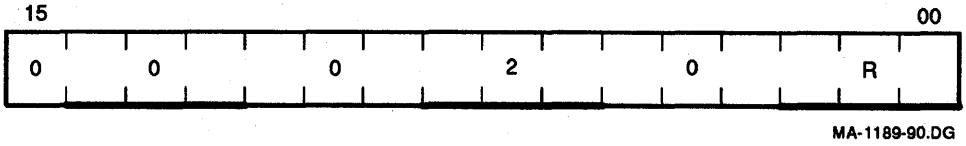


Figure 8–7 Program Control Group RTS

e. Traps (breakpoint, IOT, EMT, TRAP, BPT)

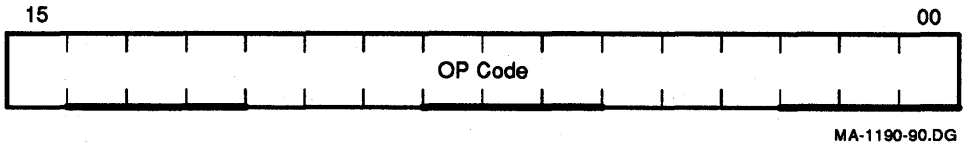


Figure 8–8 Program Control Group Traps

f. Subtract 1 and Branch (if = 0) (SOB)

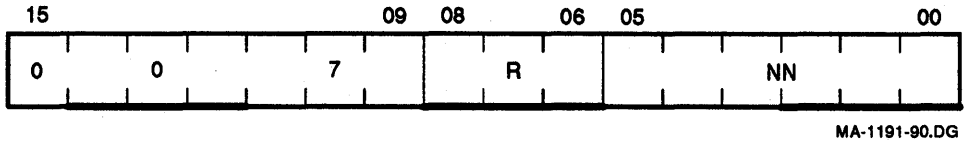


Figure 8–9 Program Control Group Subtract

g. Mark

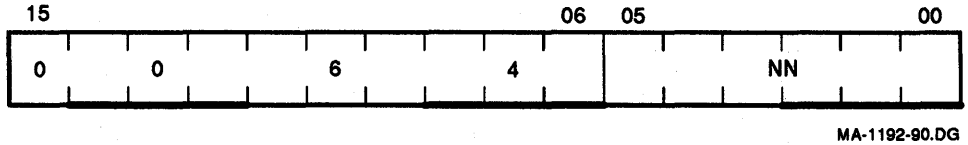


Figure 8–10 Mark

h. Call to Supervisor Mode (CSM)

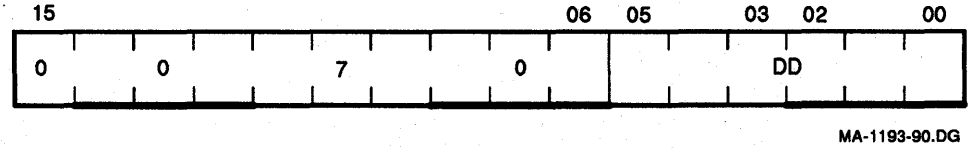


Figure 8–11 Call to Supervisor Mode

i. Set Priority Level (SPL)

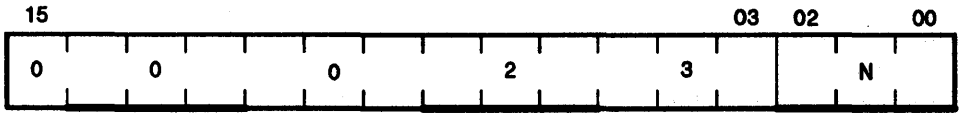


Figure 8–12 Set Priority Level

4. Operate Group: **HALT, WAIT, RTI, RESET, RTT, NOP, MFPT**

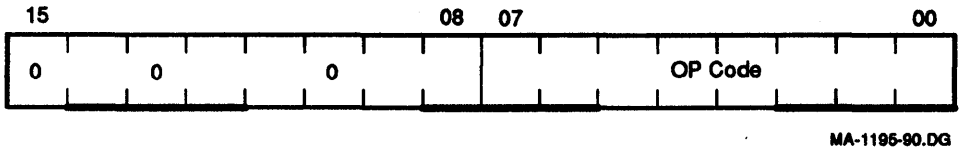


Figure 8–13 Operate Group

5. Condition Code Operators: (all condition code instructions)

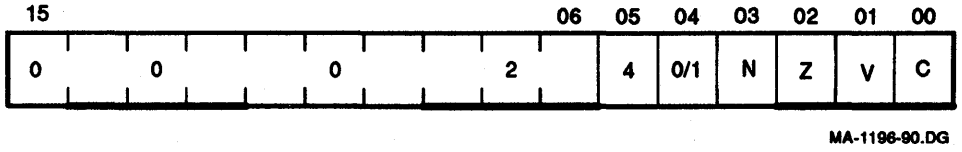


Figure 8–14 Condition Group

6. Move To and From Previous Instruction/Data Space Group: **MTPD, MTP1 MFPD, MFPI**

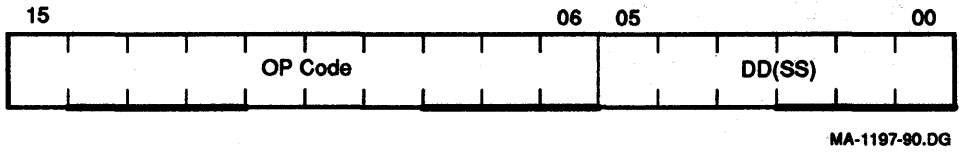


Figure 8–15 Move To and From Previous Instruction/Data Space Group

8.3 Byte Instructions

The KDJ11-E includes a full complement of instructions that manipulate byte operands. Since all KDJ11-E 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-E to perform as either a word or byte processor. Figure 8–16 shows the numbering scheme for word and byte addresses in memory.

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

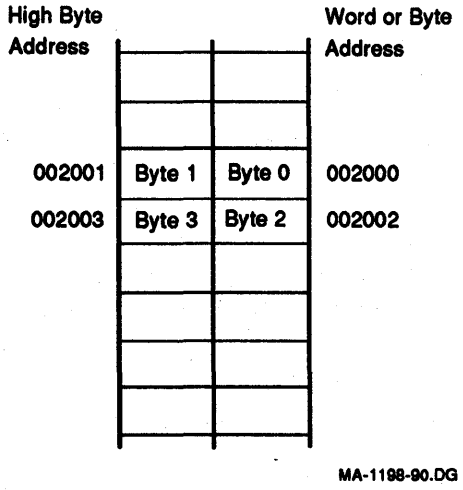


Figure 8–16 Byte Instructions

8.4 List of Instructions

The following section provides a functional list of the KDJ11-E instruction set.

8.4.1 Single-Operand

General

Mnemonic	Instruction	Op Code
CLR(B)	Clear destination	B050DD
COM(B)	Complement destination	B051DD
INC(B)	Increment destination	B052DD
DEC(B)	Decrement destination	B053DD
NEG(B)	Negate destination	B054DD
TST(B)	Test Destination	B057DD
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	B062DD
ASL(B)	Arithmetic shift left	B063DD
ROR(B)	Rotate right	B060DD
ROL(B)	Rotate left	B061DD
SWAB	Swap bytes	0003DD

Multiple-Precision

Mnemonic	Instruction	Op Code
ADC(B)	Add carry	B055DD
SBC(B)	Subtract carry	B056DD
SXT	Sign extend	0067DD

PSW Operators

Mnemonic	Instruction	Op Code
MFPS	Move byte from PSW	1067DD
MTPS	Move byte to PSW	1064SS

8.4.2 Double-Operand

General

Mnemonic	Instruction	Op Code
MOVE(B)	Move source to destination	B1SSDD
CMP(B)	Compare source to destination	B2SSDD
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	B3SSDD
BIC(B)	Bit clear	B4SSDD
BIS(B)	Bit set	B5SSDD
XOR	Exclusive OR	074RDD

8.4.3 Program Control

Mnemonic	Instruction	Op Code or Base Code
Branch		
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	0045DD
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

8.4.4 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

8.4.5 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

8.5 Single-Operand Instructions

The KDJ11-E instructions that involve only one operand are described in the paragraphs that follow.

8.5.1 General

CLR
CLRB

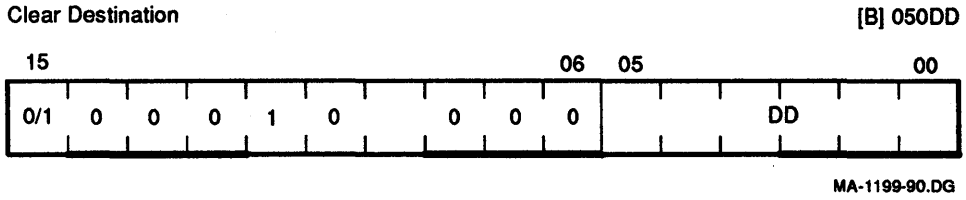


Figure 8–17 Clear Destination

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

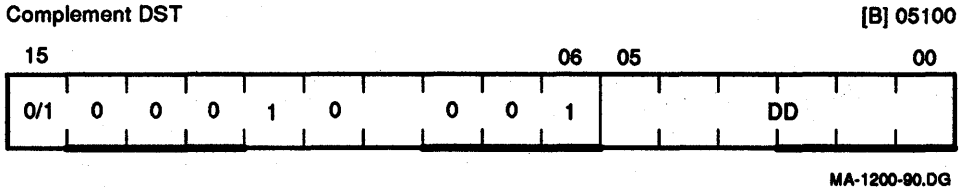


Figure 8–18 Complement Destination

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: Replace 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 R

Before	After
(R) = 013333	(R) = 164444
N Z V C	N Z V C
0 1 1 0	1 0 0 1

INC
INCB

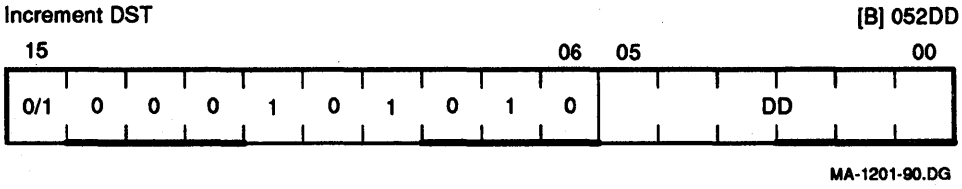


Figure 8–19 Increment Destination

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	N Z V C
0 0 0 0	0 0 0 0

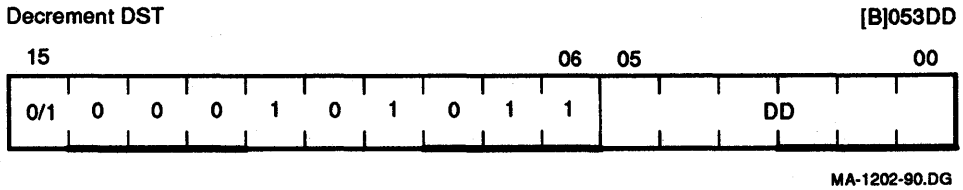


Figure 8–20 Decrement Destination

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

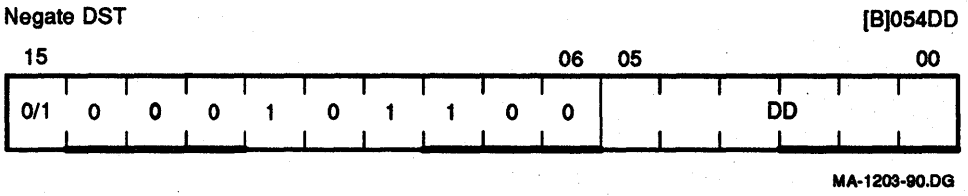


Figure 8–21 Negate Destination

Operation:	(DST) ⟵ – (DST)
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: 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 R

Before	After
(R) = 000010	(R) = 177770
N Z V C	N Z V C
0 0 0 0	1 0 0 1

TST
TSTB

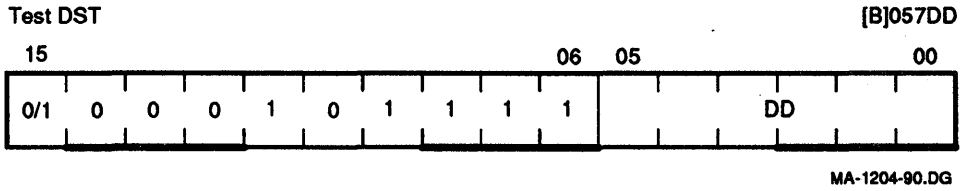


Figure 8–22 Test Destination

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	N Z V C
0 0 1 1	0 0 0 0

WRTLCK

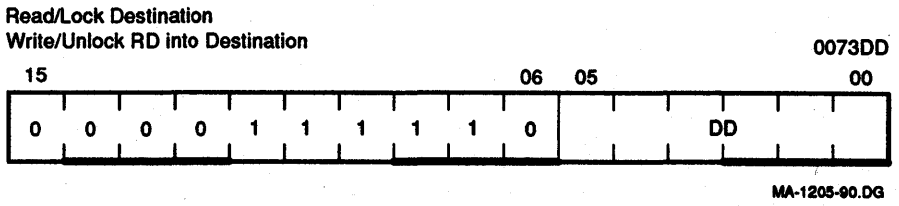


Figure 8–23 Read/Lock Destination, Write/Unlock R0 into Destination

Operation: (DST) ⟵ (R)

Condition Codes:
N: set if R < 0
Z: set if R = 0
V: cleared
C: unchanged

Description: Writes contents of R into destination using bus lock. If mode is 0, traps to 10.

TSTSET

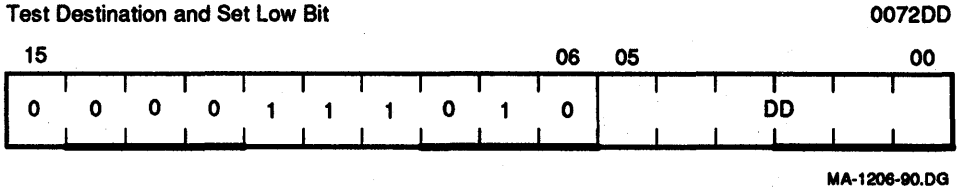


Figure 8–24 Test Destination and Set Low Bit

Operation: (R) ⟵ (DST),(DST) ⟵ (DST) ∨ 000001 (octal)

Condition Codes:
N: set if R < 0
Z: set if R = 0
V: cleared
C: gets contents of old destination bit 0.

Description: Reads/locks destination word and stores it in R. Writes/unlocks (R) V 1 into destination. If mode is 0, traps to 10.

8.5.2 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

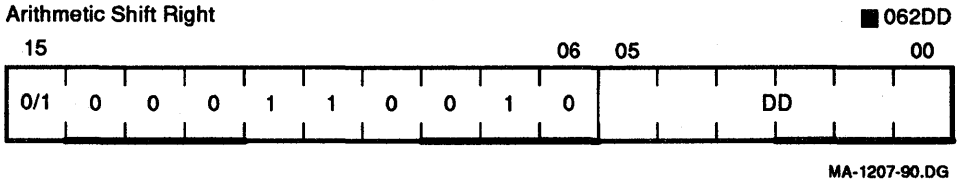


Figure 8–25 Arithmetic Shift Right

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.

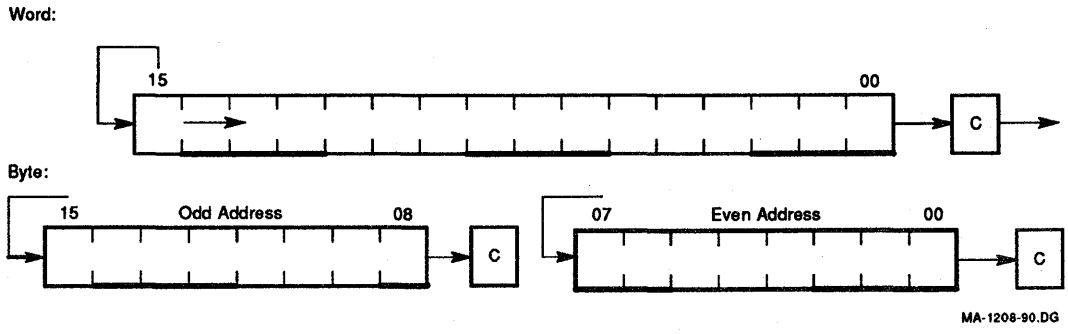


Figure 8–26 Example: Arithmetic Shift Right

ASL
ASLB

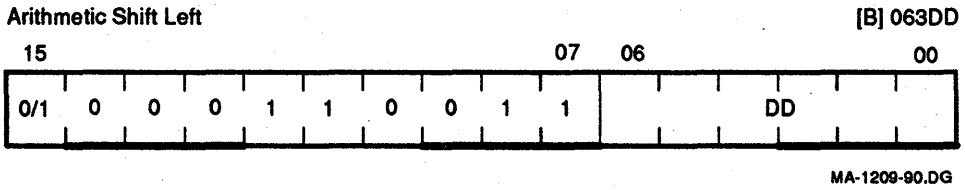


Figure 8–27 Arithmetic Shift Left

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.

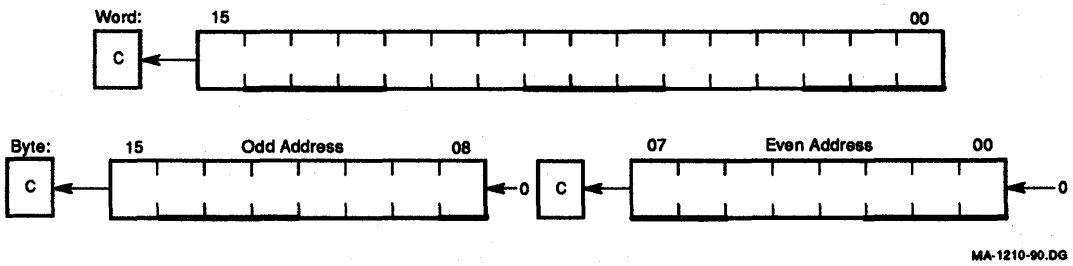


Figure 8–28 Example: Arithmetic Shift Left

ROR
RORB

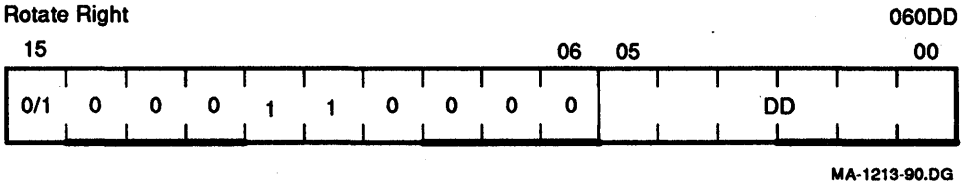


Figure 8–29 Rotate Right

Operation:	(DST) ⟵ (DST) rotate one place to the right
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.

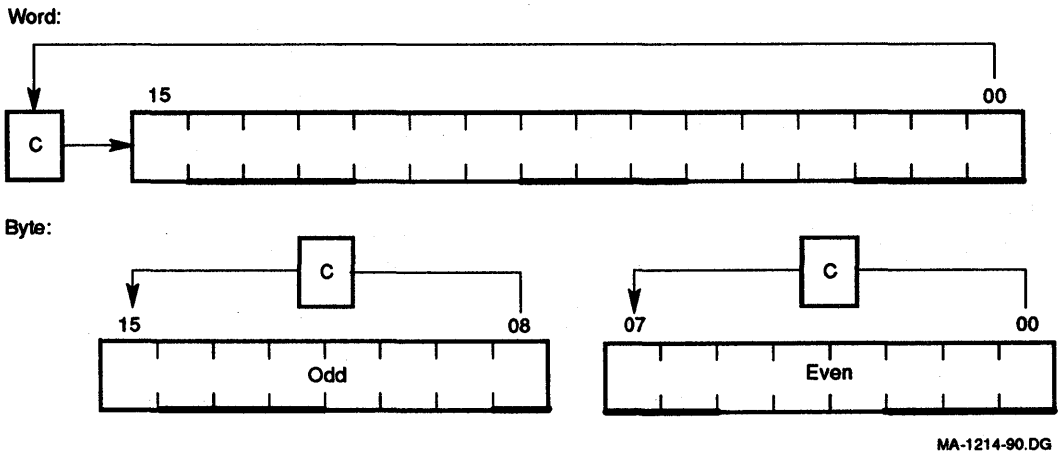


Figure 8–30 Example: Rotate Right

ROL
ROLB

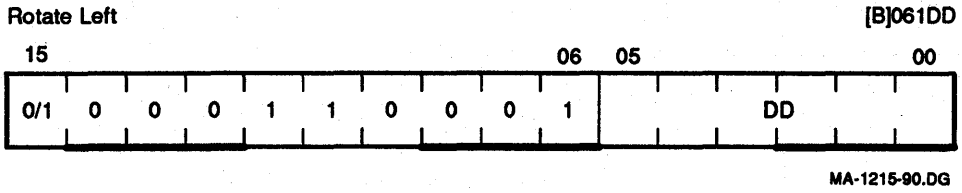


Figure 8–31 Rotate Left

Operation:	(DST) ← (DST) rotate one place to the left
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.

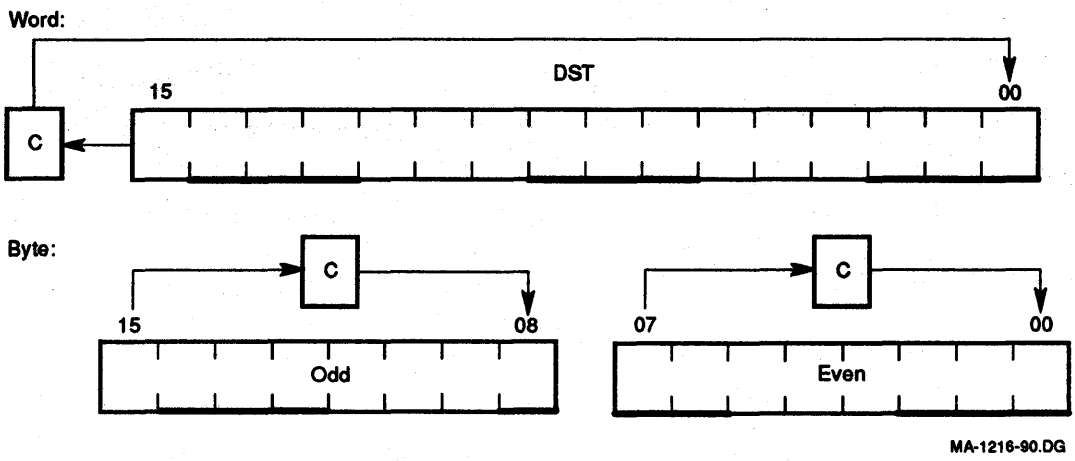


Figure 8–32 Example: Rotate Left

SWAB

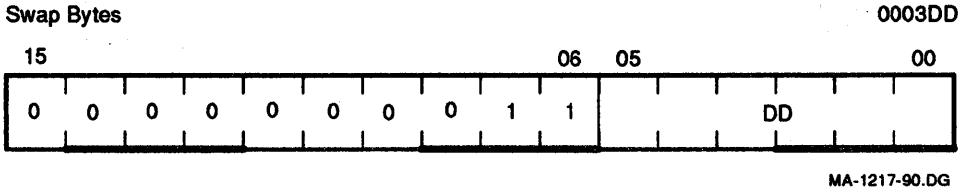


Figure 8–33 Swap Bytes

Operation:	byte 1/byte 0←—byte 0/byte 1
Condition Codes:	N: set if high-order bit of low-order byte (bit 7) of the 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	N Z V C
1 1 1 1	0 0 0 0

8.5.3 Multiple-Precision

It is sometimes necessary to do arithmetic operations on operands considered as multiple words or bytes. The KDJ11-E 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 or subtracted as shown below.

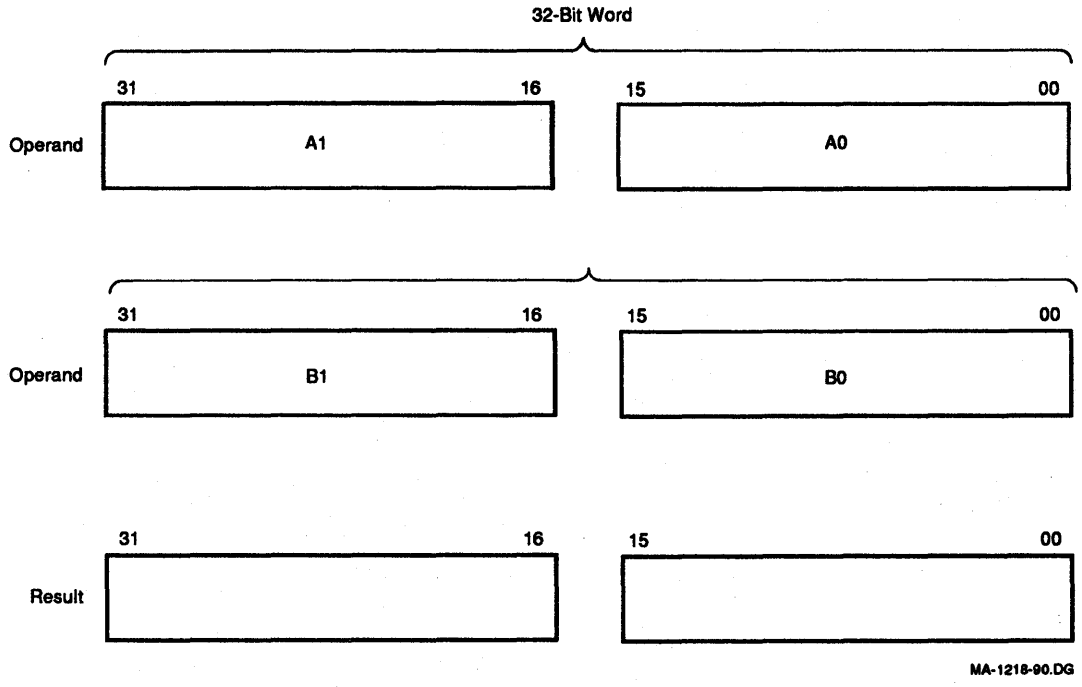


Figure 8–34 Multiple-Precision

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

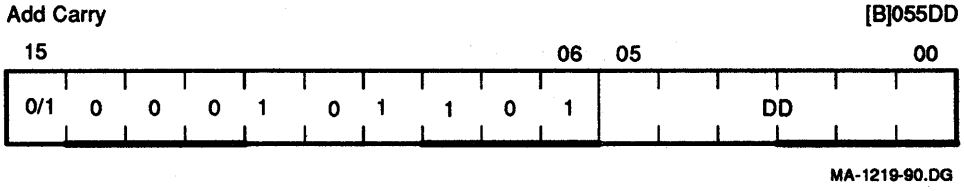


Figure 8–35 Add Carry

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

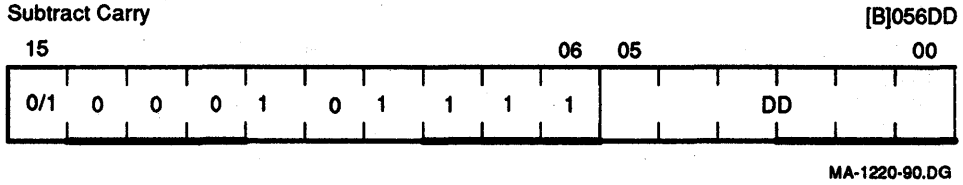


Figure 8–36 Subtract Carry

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 may be done with the following instruction sequence:

```
SUB     A0,B0
SBC     B1
SUB     A1,B1
```

SXT

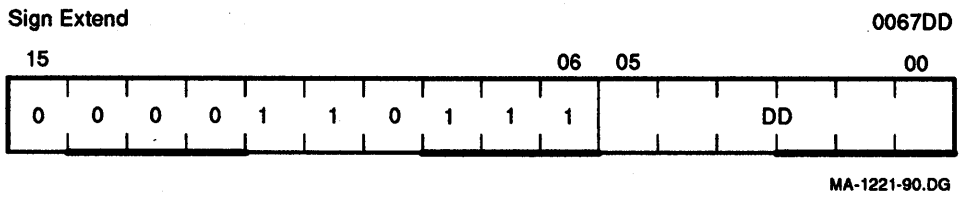


Figure 8–37 Sign Extend

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	N Z V C
1 0 0 0	1 0 0 0

8.5.4 PSW Operators

The MFPS and MTPS are op codes used directly with the processor status word.

MFPS

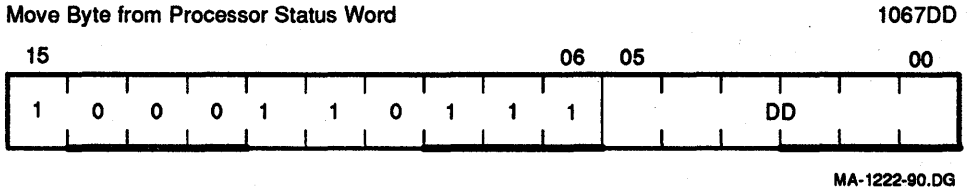


Figure 8–38 Move Byte from Processor Status Word

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 R

Before	After
(R)= 0	(R)= 000014
(PSW)= 000014	(PSW)= 000000

MTPS

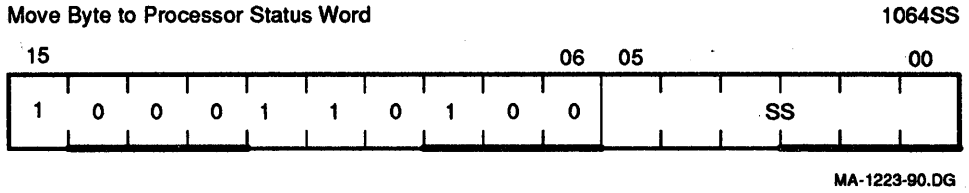


Figure 8–39 Move Byte to Processor Status Word

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

8.6 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.

8.6.1 General

MOV
MOVB

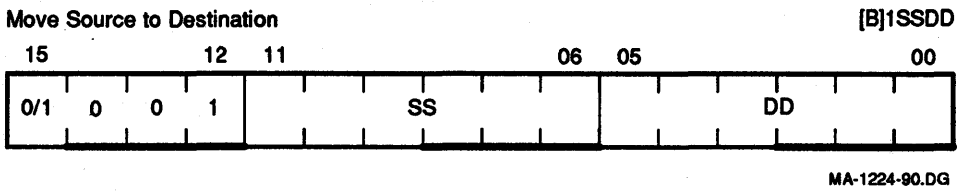


Figure 8–40 Move Source to Destination

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,R           ;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

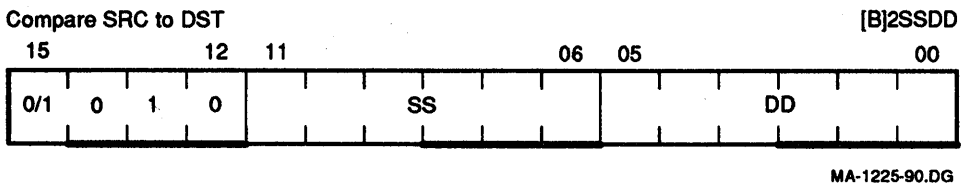


Figure 8–41 Compare SRC to Destination

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

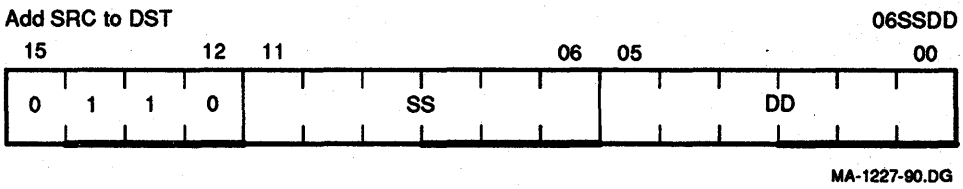


Figure 8–42 Add SRC to Destination

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,R
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

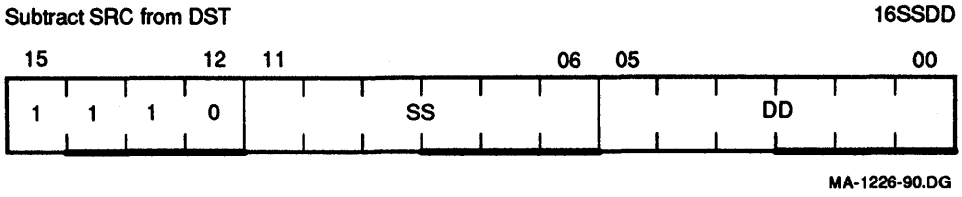


Figure 8–43 Subtract SRC from DST

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

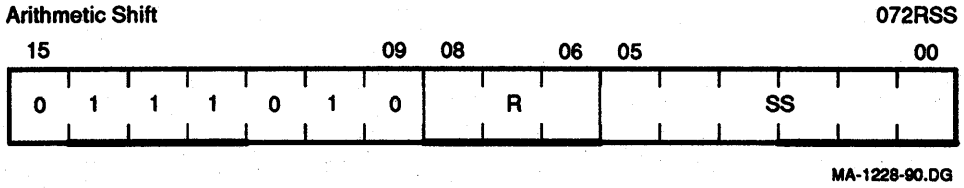


Figure 8–44 Arithmetic Shift

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

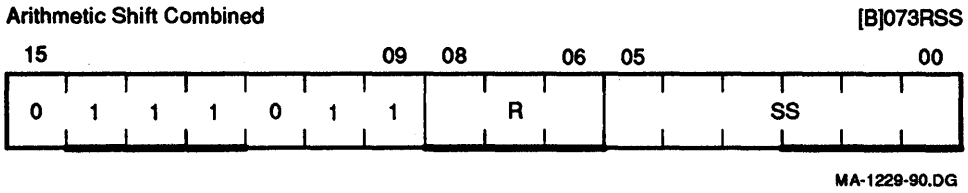


Figure 8–45 Arithmetic Shift Combined

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

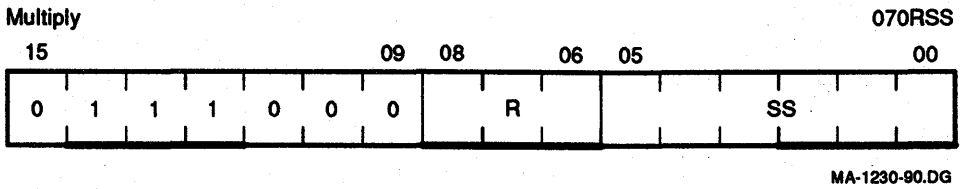


Figure 8–46 Multiply

Operation:	R, R ∨ 1←—R x (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

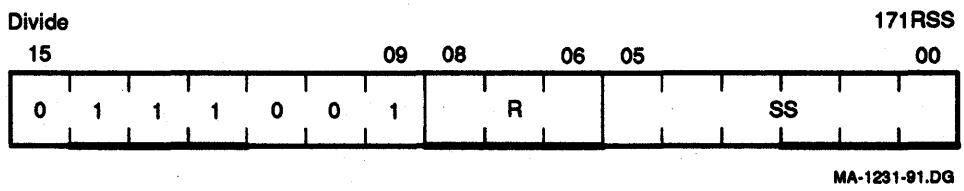


Figure 8–47 Divide

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.

8.6.2 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

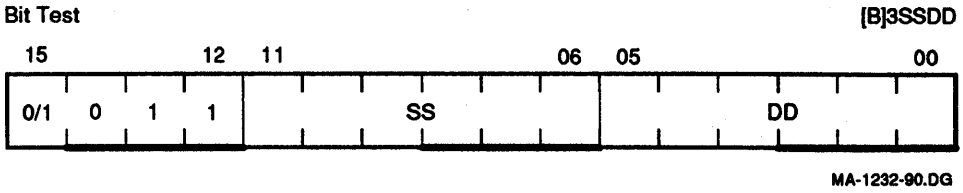


Figure 8–48 Bit Test

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

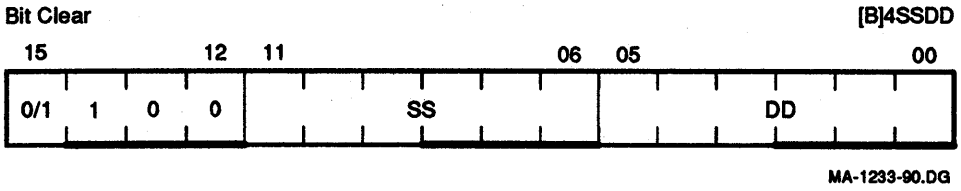


Figure 8–49 Bit Clear

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

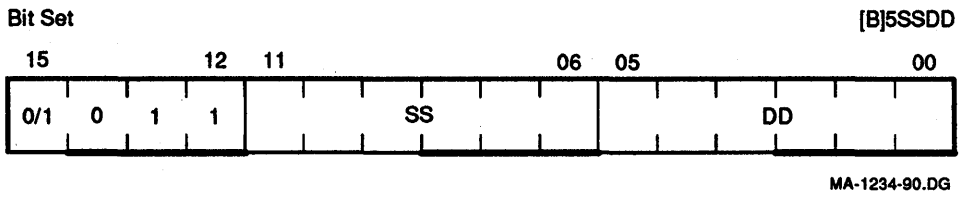


Figure 8–50 Bit Set

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 R,R1

Before	After
(R) = 001234	(R) = 001234
(R1) = 001111	(R1) = 001335
N Z V C	N Z V C
0 0 0 0	0 0 0 0

Before: (R) = 0 000 001 010 011 100
(R1) = 0 000 001 001 001 001

After: (R1) = 0 000 001 011 011 101

XOR

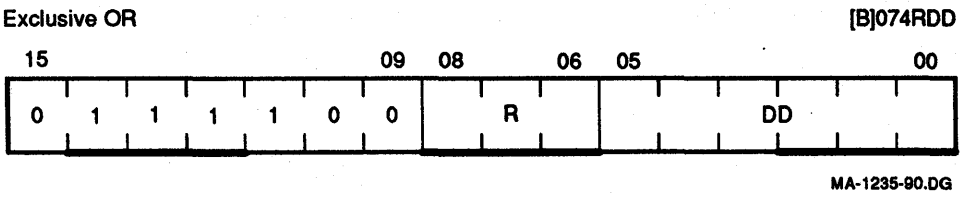


Figure 8–51 Exclusive OR

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 R,R2

Before	After
(R) = 001234	(R) = 001234
(R2) = 001111	(R2) = 000325
N Z V C	N Z V C
1 1 1 1	0 0 0 1

Before: (R) = 0 000 001 010 011 100
(R2) = 0 000 001 001 001 001

After: (R2) = 0 000 000 011 010 101

8.7 Program Control Instructions

The following paragraphs describe the KDJ11-E instructions that affect program control.

8.7.1 Branches

Program control 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-E assembler typically handles address arithmetic for the user and computes and assembles the proper offset field for branch instructions in the following format:

```
Bxx loc
```

Bxx is the branch instruction and loc is the address where 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

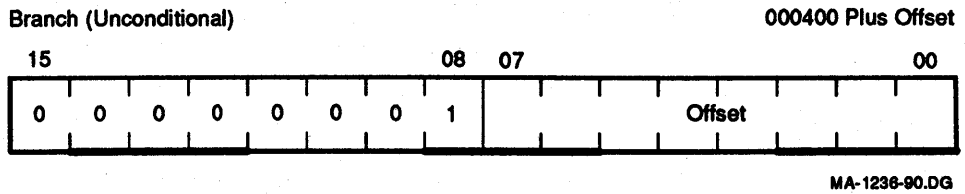


Figure 8–52 Branch (Unconditional)

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

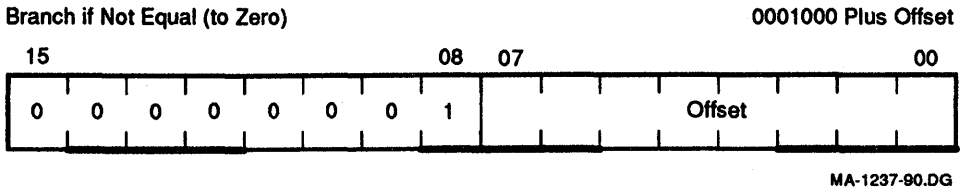


Figure 8–53 Branch If Not Equal (to Zero)

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

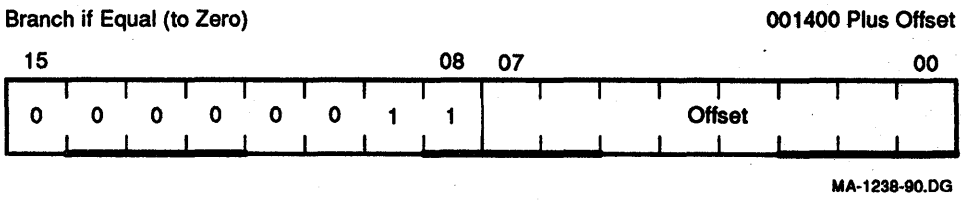


Figure 8–54 Branch If Equal (to Zero)

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

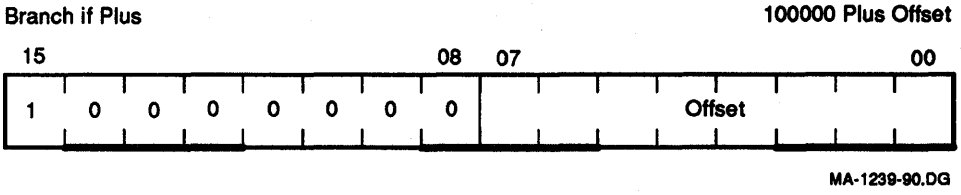


Figure 8–55 Branch If Plus

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

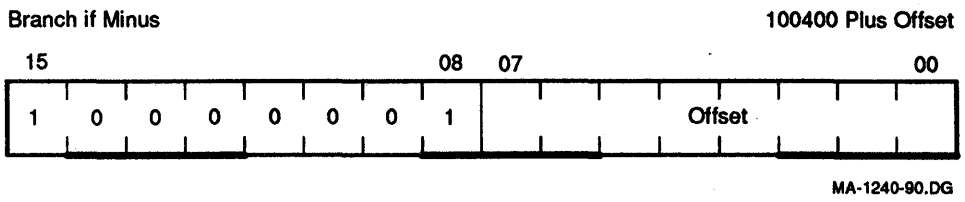


Figure 8–56 Branch if Minus

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

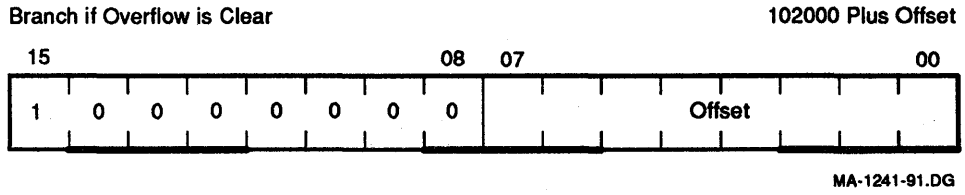


Figure 8–57 Branch if Overflow is Clear

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

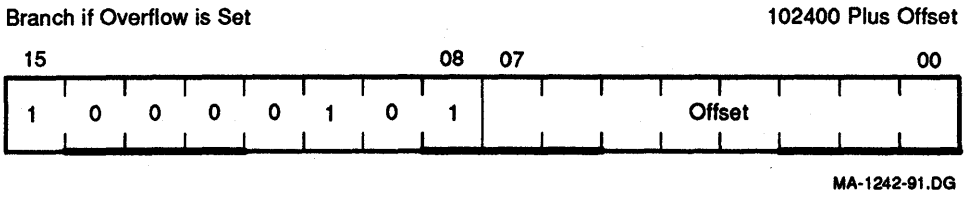


Figure 8–58 Branch if Overflow is Set

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

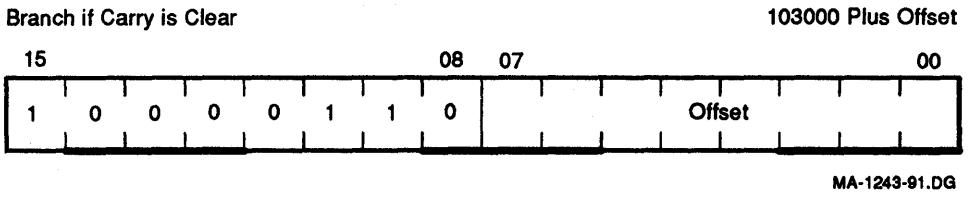


Figure 8–59 Branch if Carry is Clear

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

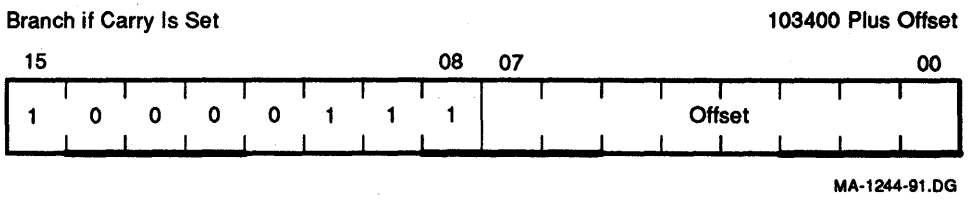


Figure 8–60 Branch if Carry is Set

Operation: PC ⟵ PC + (2 X 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.

8.7.2 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	077777
positive	077776
	.
	.
	.
	000001
	000000
	177777
	177776
	.
	.
	.
smallest	100001
negative	100000

Whereas, in unsigned, 16-bit arithmetic, the sequence is considered to be:

highest	177777
	.
	.
	.
	.
	.
	000002
	000001
lowest	000000

BGE

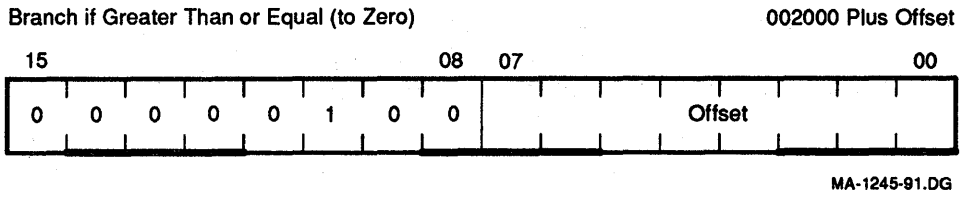


Figure 8–61 Branch if Greater Than or Equal (to Zero)

Operation: PC ⟵ PC + (2 × offset) if N ∀ 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

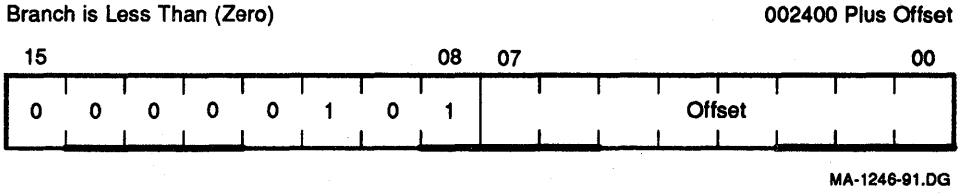


Figure 8–62 Branch If Less Than (Zero)

Operation: PC ⟵ PC + (2 × offset) if N ∀ V = 1

Condition Codes: Not affected

Description: Causes a branch if the exclusive OR of the N- and V-bits is 1. 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

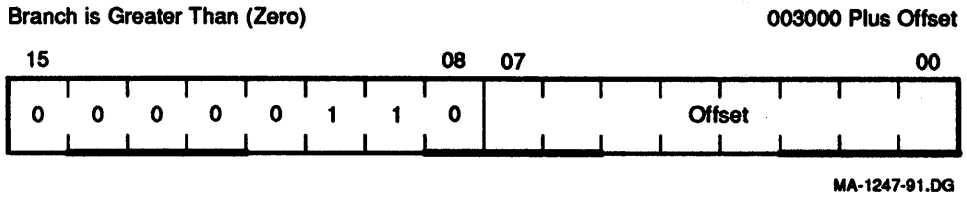


Figure 8–63 Branch If Greater Than Zero

Operation: PC ⟵ PC + (2 × offset) if Z (N ∀ 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

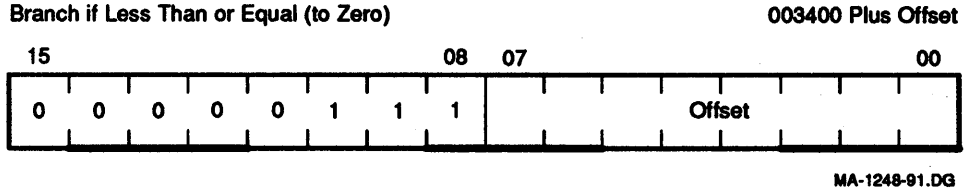


Figure 8–64 Branch If Less Than or Equal (to Zero)

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.

8.7.3 Unsigned Conditional Branches

The unsigned conditional branches provide a means for testing the result of comparison operations in which the operands are considered to be unsigned values.

BHI

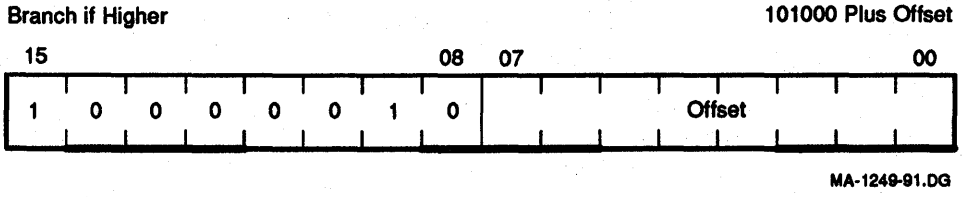


Figure 8–65 Branch if Higher

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

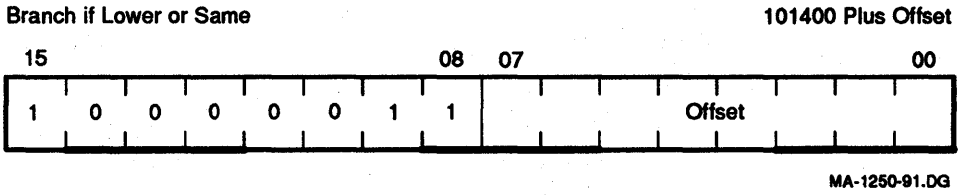


Figure 8–66 Branch if Lower or Same

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

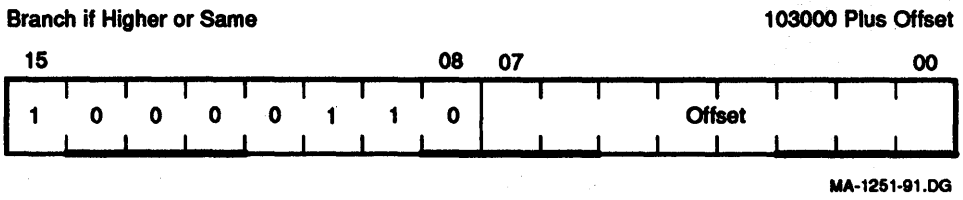


Figure 8–67 Branch If Higher or Same

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

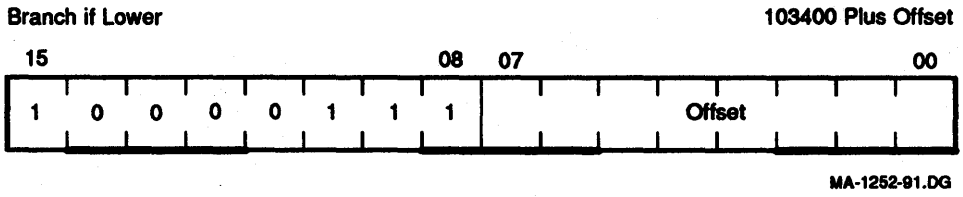


Figure 8–68 Branch If Lower

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.

8.7.4 Jump and Subroutine Instructions

The subroutine call in the KDJ11-E 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

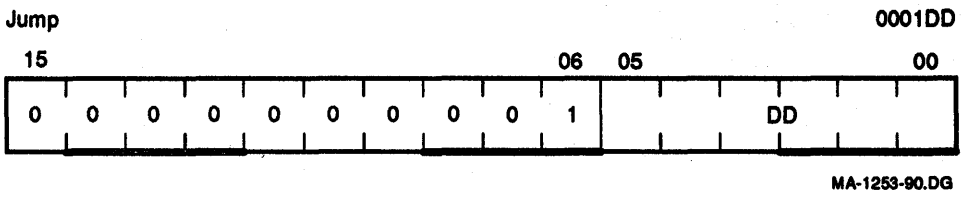


Figure 8–69 Jump

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

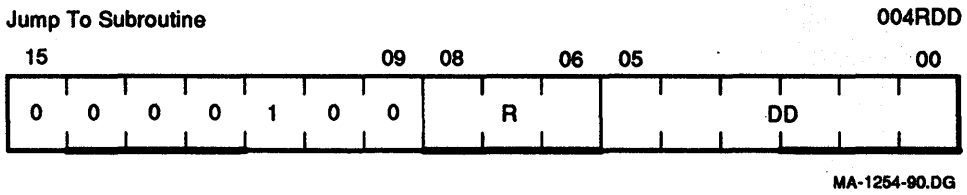


Figure 8–70 Jump to Subroutine

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-E 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

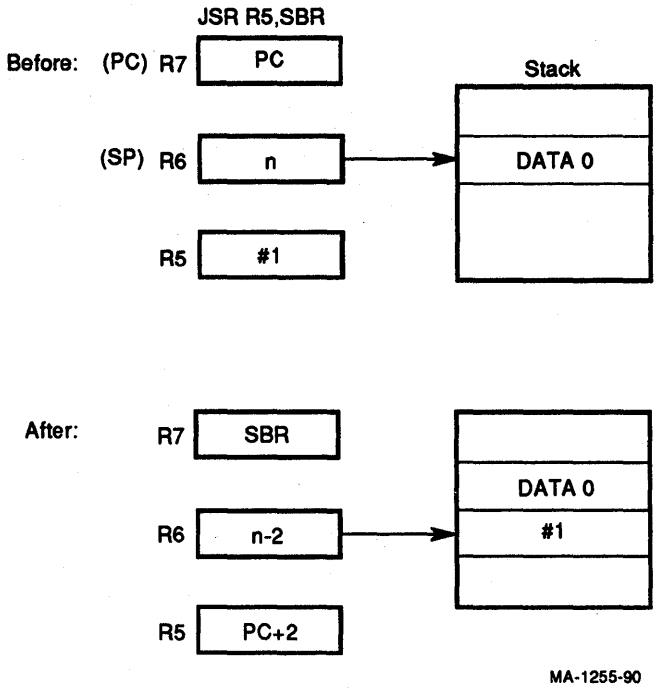


Figure 8–71 Example: Jump to Subroutine

RTS

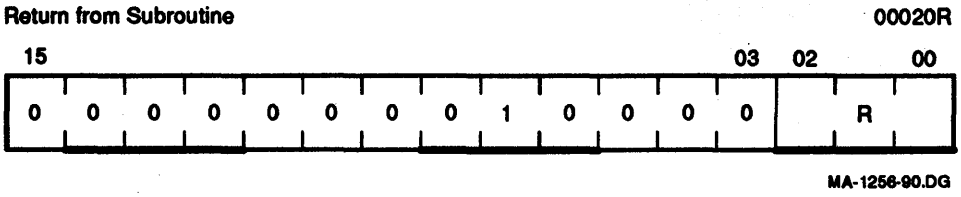


Figure 8–72 Return from Subroutine

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

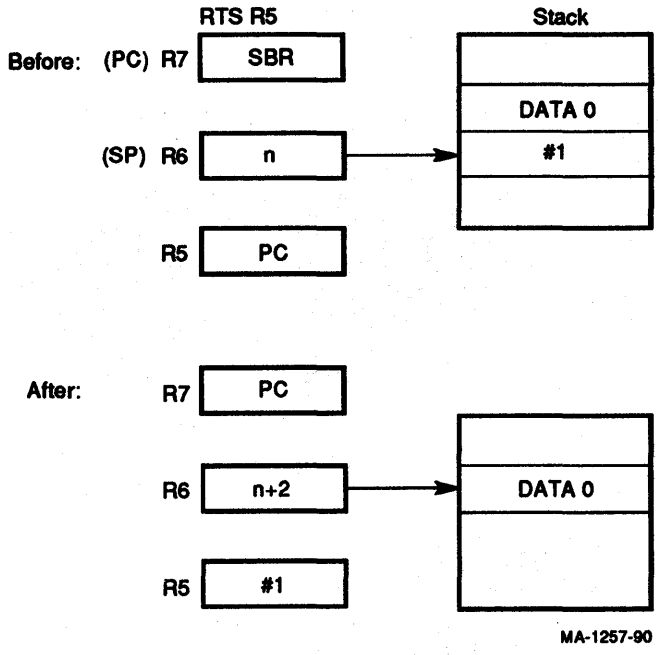


Figure 8–73 Example: RTS R5

SOB

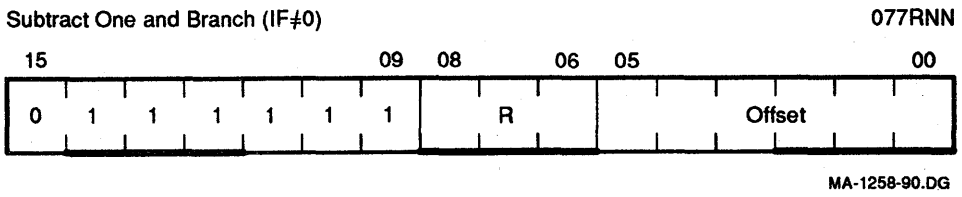


Figure 8–74 Subtract One and Branch

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.

8.7.5 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

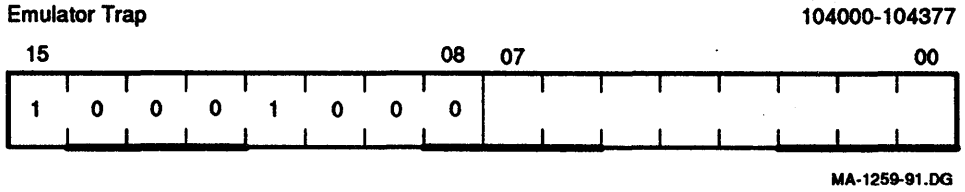


Figure 8–75 Emulator Trap

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 (for example, 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.

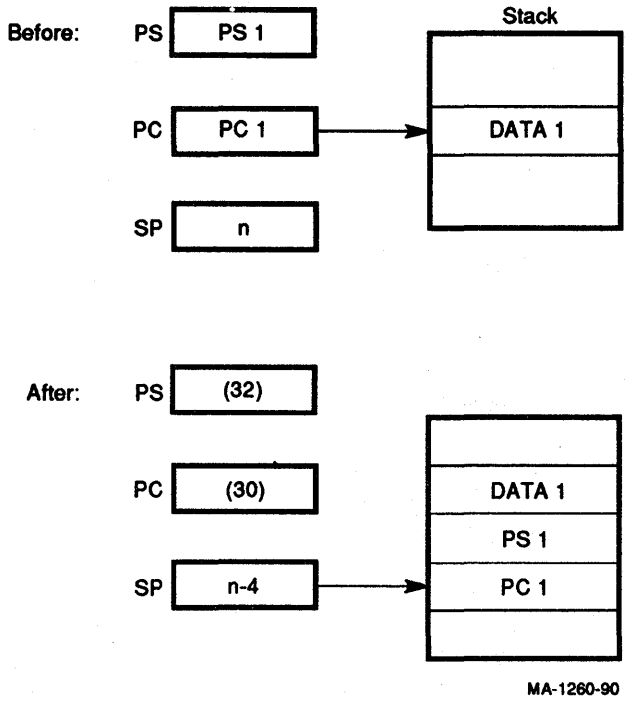


Figure 8–76 Example: Emulator Trap

TRAP

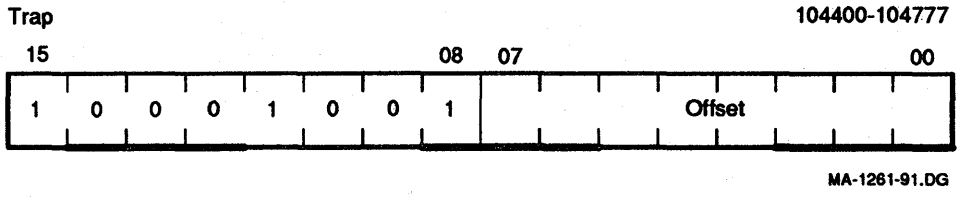


Figure 8–77 Trap

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

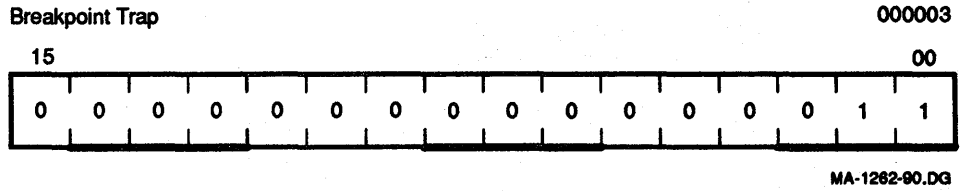


Figure 8–78 Breakpoint Trap

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

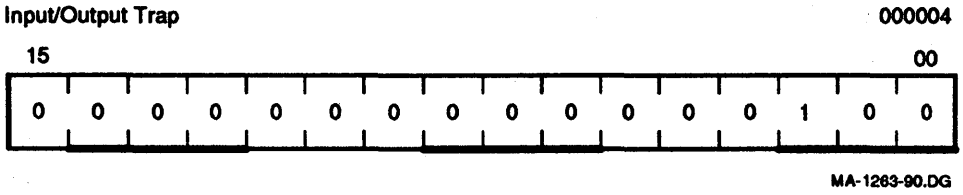


Figure 8–79 Input/Output Trap

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

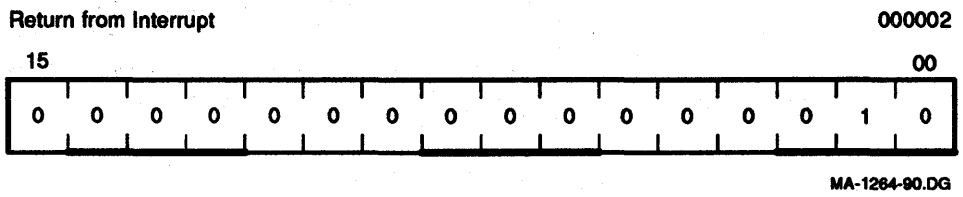


Figure 8–80 Return from Interrupt

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

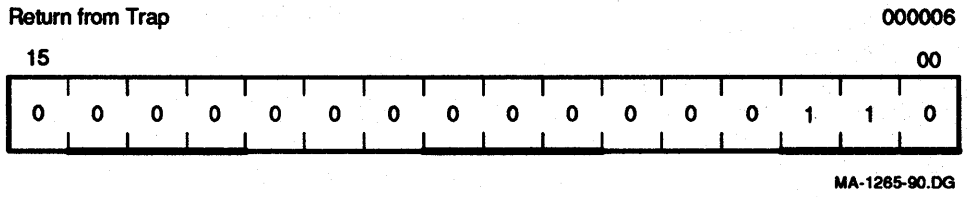


Figure 8–81 Return from Trap

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.

8.7.5.1 Miscellaneous Program Controls

MARK

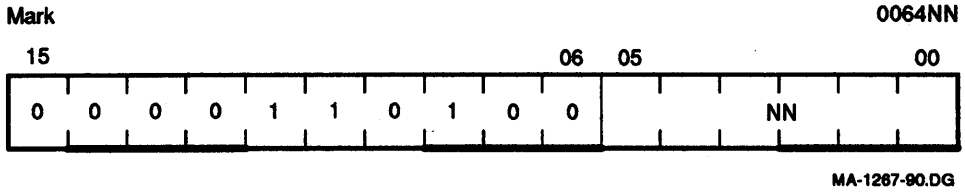


Figure 8–82 Mark

Operation:
SP ⟵ PC + 2 X 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:

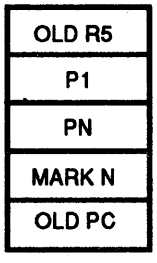


Figure 8–83 Example: Mark

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.
```

NOTE
If memory management is in use, the stack must be mapped through both I and D space to execute the MARK instruction.

SPL

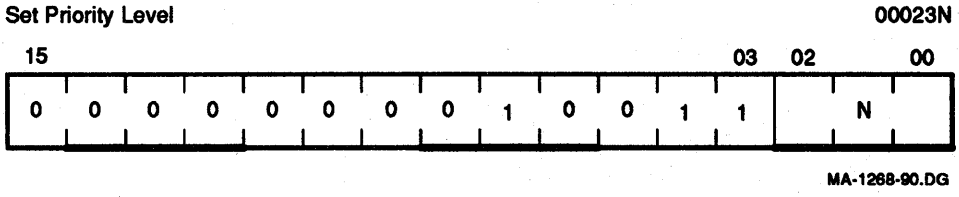


Figure 8–84 Set Priority Level

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

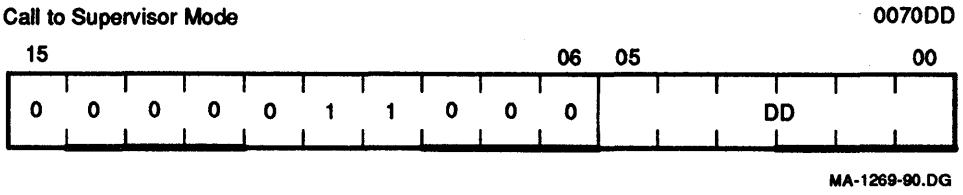


Figure 8–85 Call to Supervisor Mode

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.

8.7.6 Reserved Instruction Traps

Reserved instruction traps 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.

8.7.7 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.

NOTE
The traced instruction is the instruction after the one that sets the T-bit.

The following are special cases of 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 in steps 1 through 4, a trace trap occurs.
6. Interrupt trap priorities—See Table 1–6.

8.8 Miscellaneous Instructions

HALT

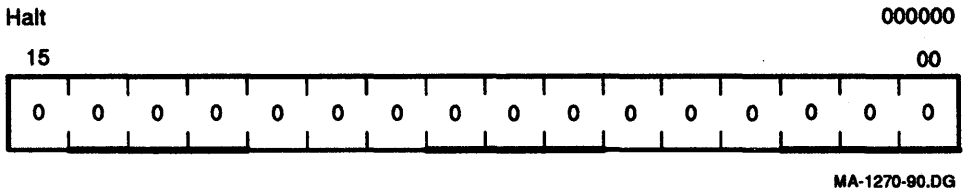


Figure 8–86 Halt

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.

NOTE
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

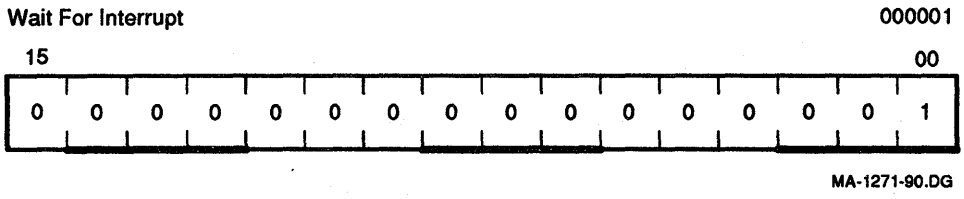


Figure 8–87 Wait for Interrupt

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

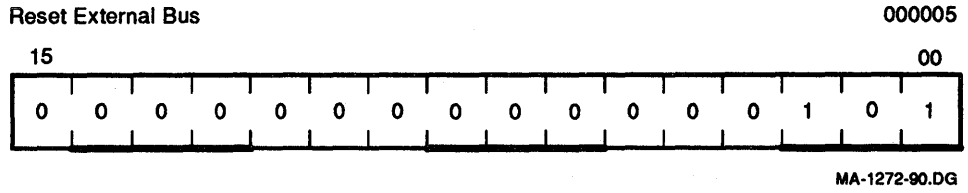


Figure 8–88 Reset External Bus

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

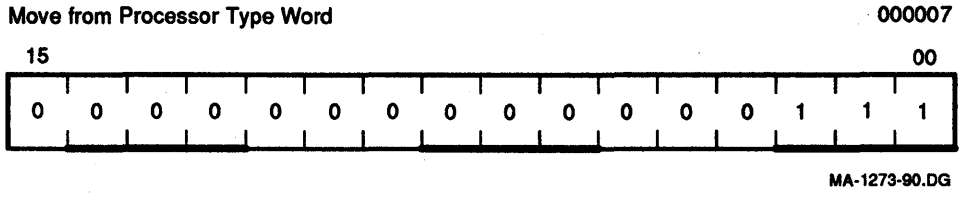


Figure 8–89 Move from Processor Type Word

Operation: R ⟵ 5

Condition Codes: Not affected

Description: The number 5 is placed in R, indicating to the operating system software that the processor type is a KDJ11-E.

MTPD/MTPI

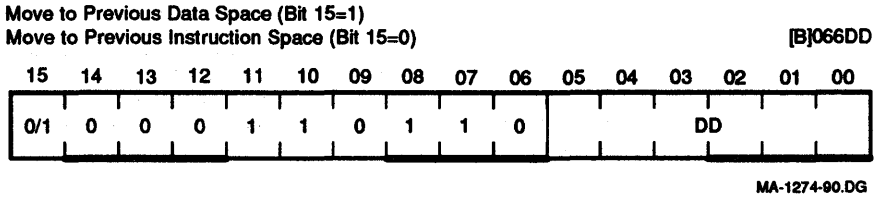


Figure 8–90 Move to Previous Data Space (Bit 15 = 1) Move to Previous Instruction Space (Bit 15 = 0)

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

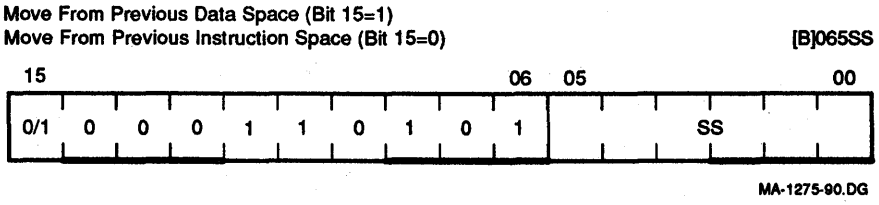


Figure 8–91 Move from Previous Data Space (Bit 15 = 1) Move from Previous Instruction Space (Bit 15 = 0)

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.

8.9 Condition Code Operators

The op codes that can be used to set/clear condition codes, either independently or ORed together, are as follows:

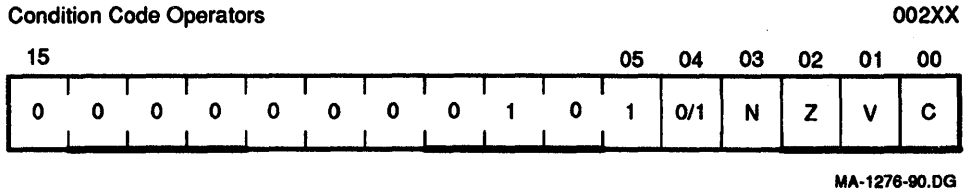


Figure 8–92 Condition Code Operators

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.

9
Floating-Point Arithmetic

9.1 Introduction

The KDJ11-E 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.

9.2 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.

9.2.1 Nonvanishing Floating-Point Numbers

The fractional part (f) is assumed to be 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 (that is, 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 described in the following section, 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.

9.2.2 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-E 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.

9.2.3 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.

9.2.4 Floating-Point Data

Floating-point data is stored in words of memory as illustrated in Figure 9–1 and Figure 9–2.

The KDJ11-E 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 (Figure 9–3).

9.3 Floating-Point Status Register (FPS)

The floating-point status register (FPS) provides mode and interrupt control for the currently executing floating-point instruction and also reflects conditions resulting from the execution of the previous instruction (Figure 9–4). 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.

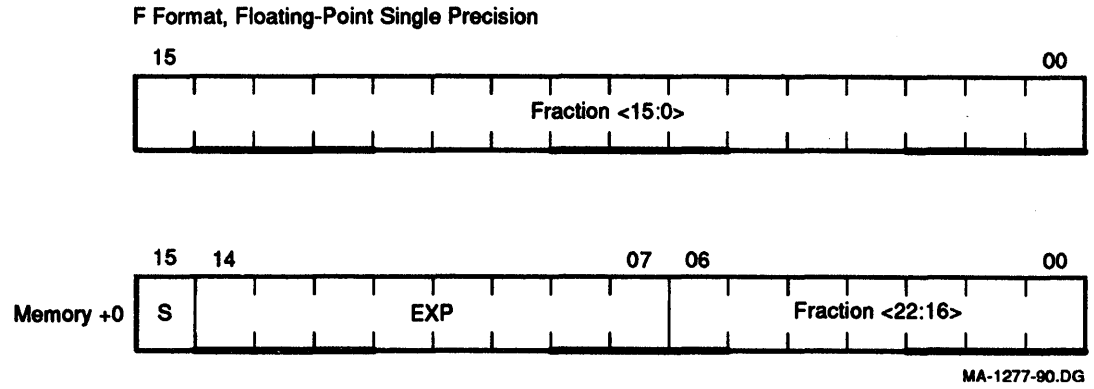


Figure 9–1 Single-Precision Format

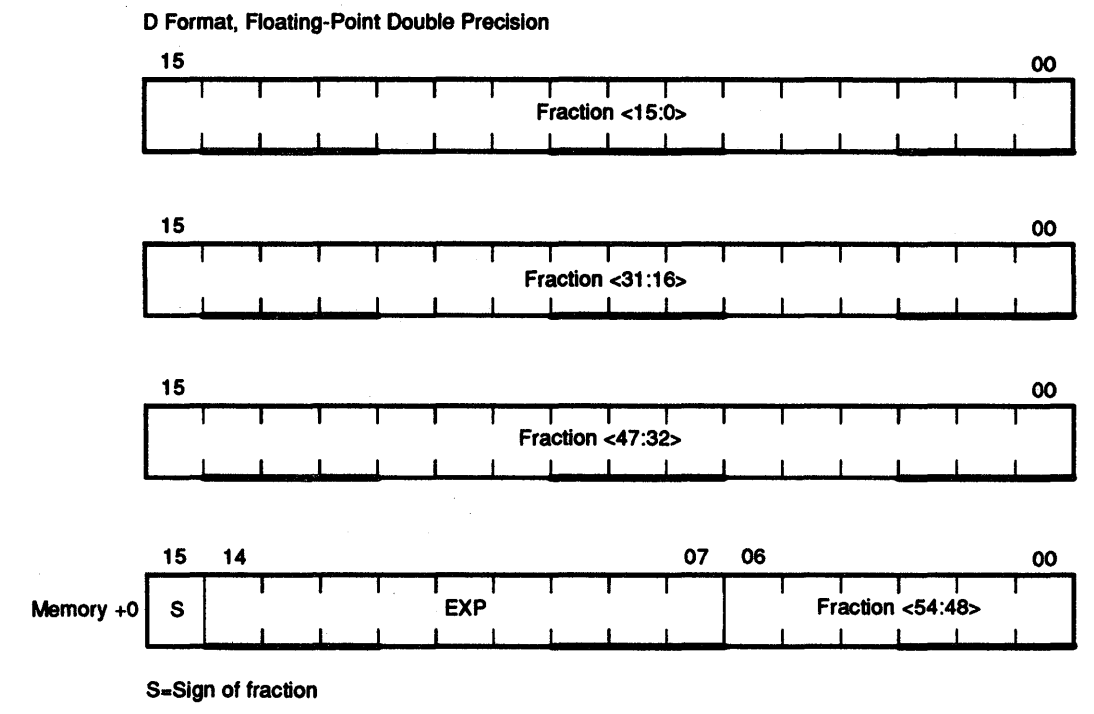


Figure 9–2 Double-Precision Format

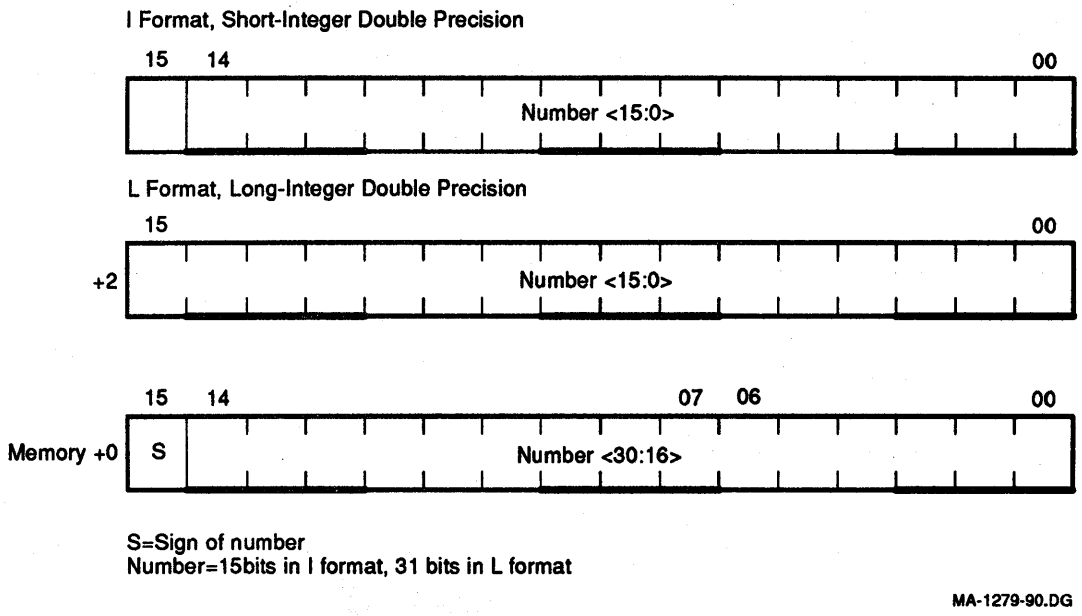


Figure 9–3 2's Complement Format

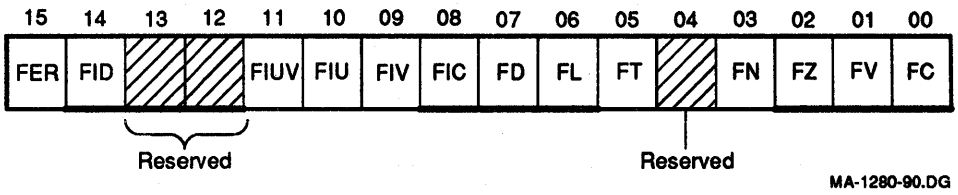


Figure 9–4 Floating-Point Status Register

The KDJ11-E 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 (Figure 9–4). Table 9–1 describes the FPS register bits.

Table 9–1 FPS Register Bit Descriptions

Bit	Name	Function
15	FER	The floating error (FER) bit is set by the DCJ11 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 DCJ11 never resets the FER bit. Once the FER bit is set by the DCJ11, 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 must be clear if you want to ensure that storage of –0 by the DCJ11 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.

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 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 accumulator 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 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 DCJ11 returns exact 0. Special cases of overflow are discussed later in the detailed descriptions of the MOD and LDEXP instructions.

Table 9–1 (Cont.) FPS Register Bit Descriptions

Bit	Name	Function
8	FIC	When the FIC bit is set and a conversion to integer instruction fails, an interrupt occurs. When the interrupt occurs, 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 previously, 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 doubled-precision 2's complement (32 bits). When reset, the integer format assumed is single-precision 2's complement (16 bits).
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 was 0; otherwise it is reset.
1	FV	FV is set if the previous floating-point operation 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.

9.4 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.

NOTE

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 store status (STST) 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.

9.5 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 contains the least significant word.

9.6 Accuracy

General comments on the accuracy of the KDJ11-E 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 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. This guarantees 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-E 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 a LSB.
2. If the rounding bit is 0, the rounded and chopped results are identical.

It follows that if the result is exact:

Rounded value = chopped value = exact value.

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-E 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 9–1.

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 (by using STEXP and LDEXP) to continue a calculation. Note that the accuracy of the fractional part is unaffected by the occurrence of underflow or overflow.

9.7 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 9–5); SRC and DST operands use CPU addressing modes.

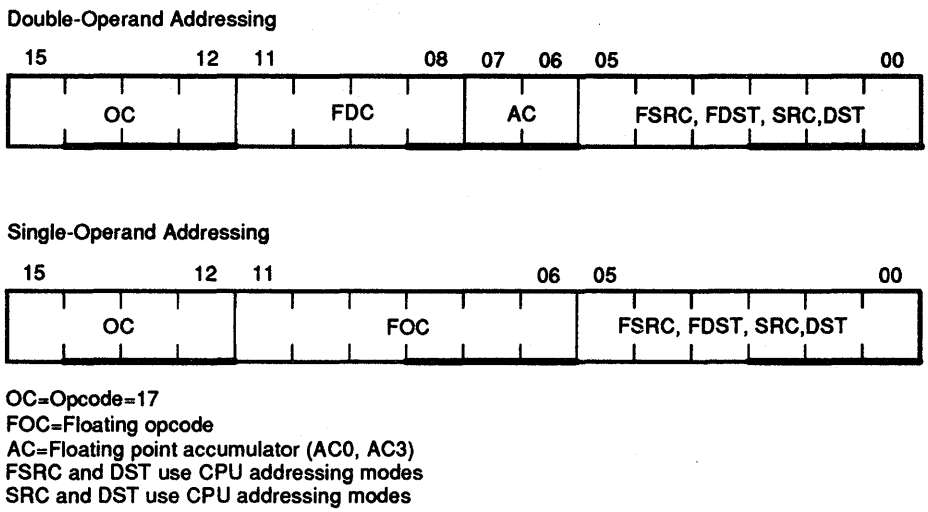


Figure 9–5 Floating-Point Addressing Modes

9.7.1 Terms Used in Instruction Definitions

The following paragraphs provide a list of the terms used in instruction definitions, as well as explanations of and figures for each instruction.

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 identical to 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

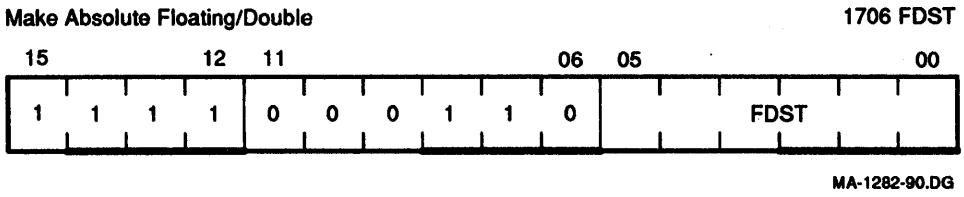


Figure 9–6 Make Absolute Floating/Double

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

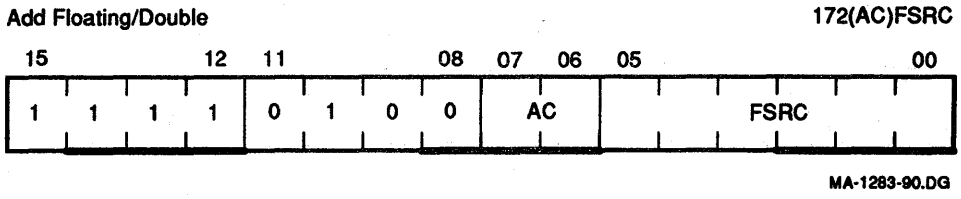


Figure 9–7 Add Floating/Double

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

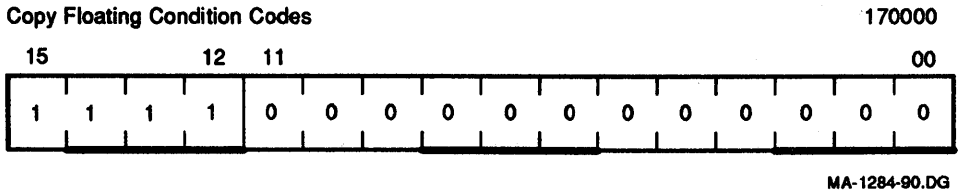


Figure 9–8 Copy Floating Condition Codes

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

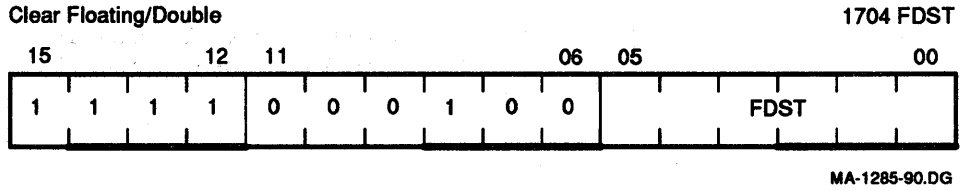


Figure 9–9 Clear Floating/Double

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

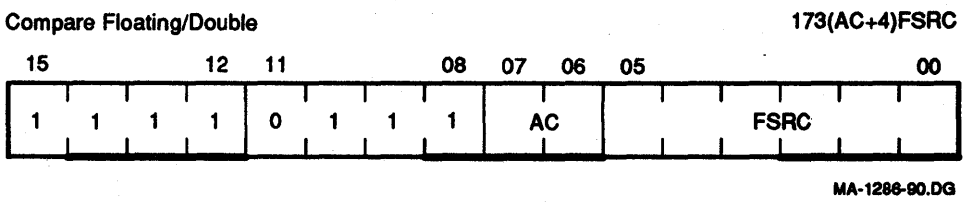


Figure 9–10 Compare Floating/Double

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 later.
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-E stores an exact 0 in AC.

DIVF/DIVD

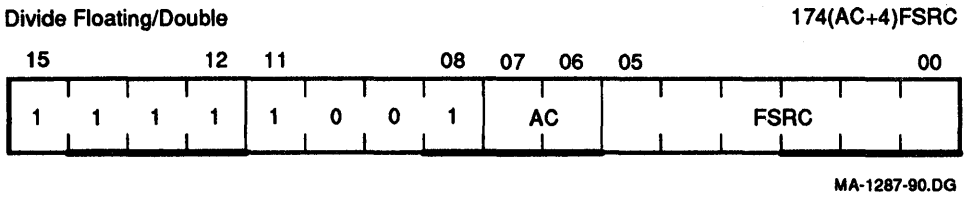


Figure 9–11 Divide Floating/Double

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 previously. If no underflow or overflow errors occur, 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

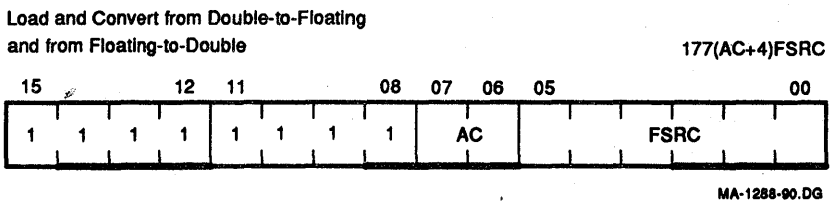


Figure 9–12 Load and Convert From Double-To-Floating and from Floating-to-Double

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

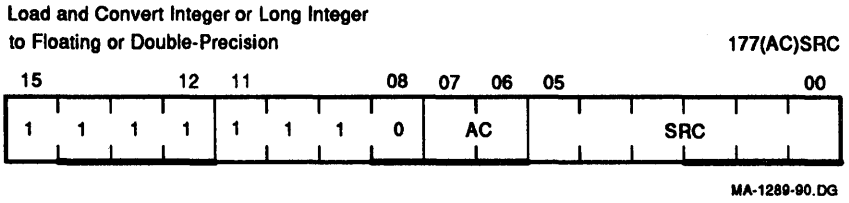


Figure 9–13 Load and Convert Integer or Long Integer to Floating or Double-Precision

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

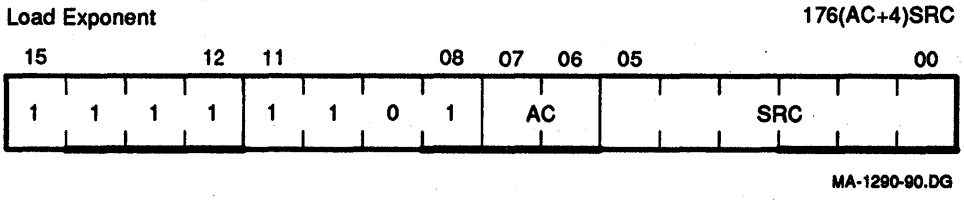


Figure 9–14 Load Exponent

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 previously. 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

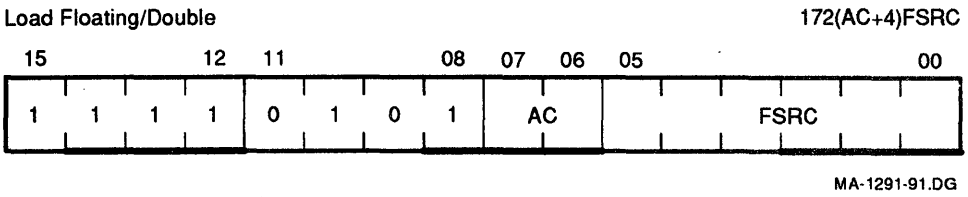


Figure 9–15 Load Floating/Double

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

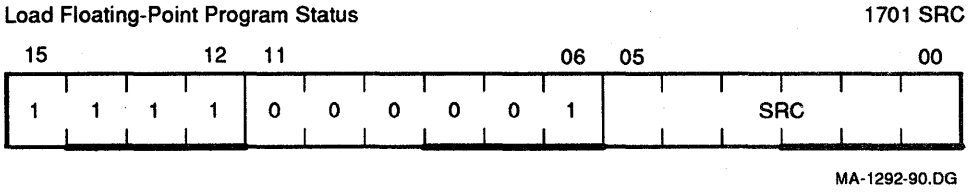


Figure 9–16 Load Floating-Point Program Status

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

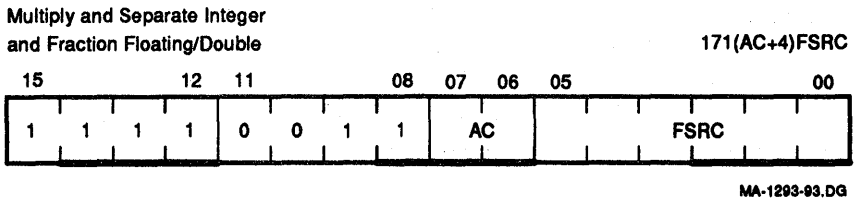


Figure 9–17 Multiply and Separate Integer and Fraction Floating/Double

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 previously.

Accuracy:

Described previously.

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

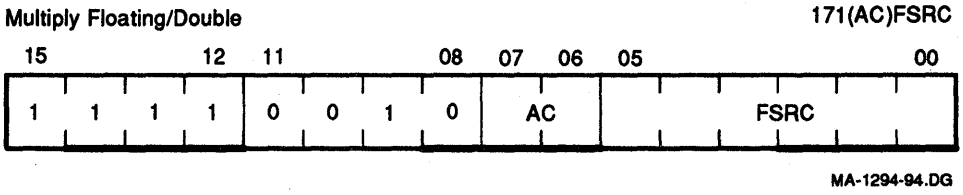


Figure 9–18 Multiply Floating/Double

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

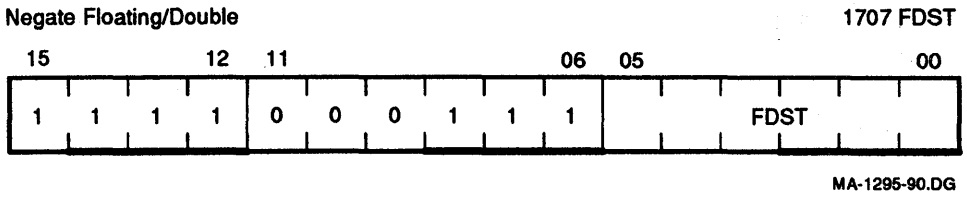


Figure 9–19 Negate Floating/Double

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

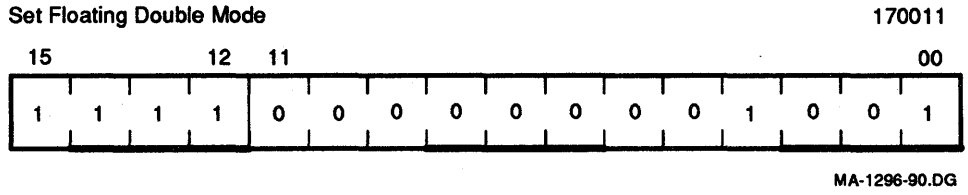


Figure 9–20 Set Floating Double mode

Format:	SETD
Operation:	FD ⟵ 1
Description:	Set the KDJ11-E in double-precision mode.

SETF

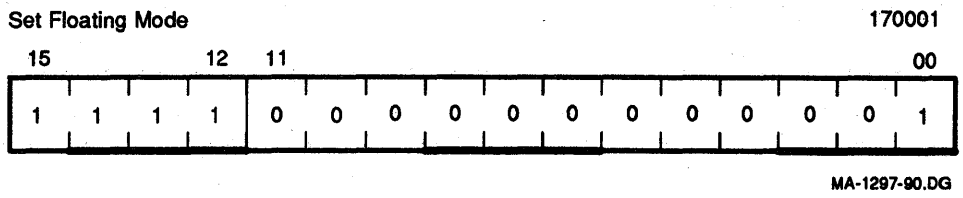


Figure 9–21 Set Floating Mode

Format: SETF

Operation: FD ← 0

Description: Set the KDJ11-E in single-precision mode.

SETI

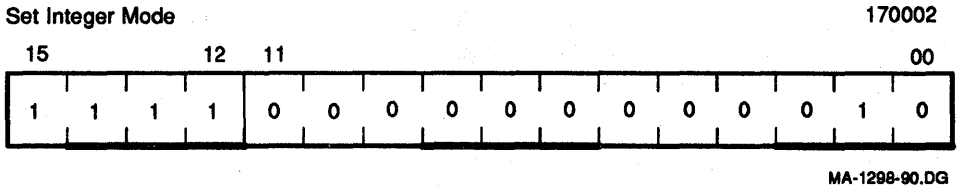


Figure 9–22 Set Integer Mode

Format: SETI

Operation: FL ← 0

Description: Set the KDJ11-E for short-integer data.

SETL

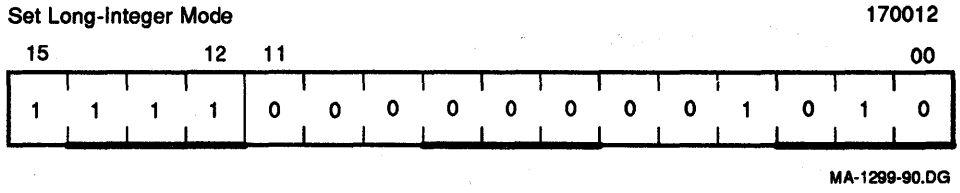


Figure 9–23 Set Long Integer Mode

Format: SETL

Operation: FL ← 1

Description: Set the KDJ11-E for long-integer data.

STCFD/STCDF

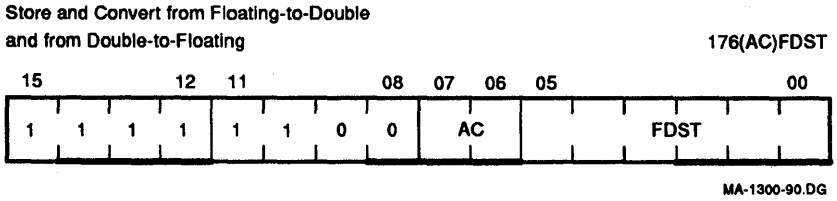


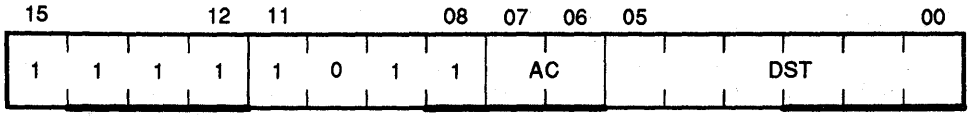
Figure 9–24 Store and Convert from Floating-To-Double and from Double-To-Floating

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



LJ-00431-TI0

Figure 9–25 Store and Convert from Floating-to-Double To Integer Or Long Integer

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

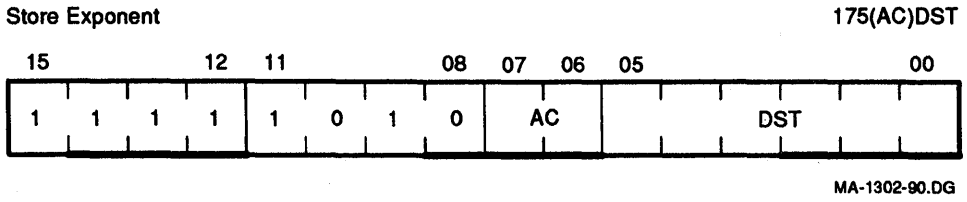


Figure 9–26 Store Exponent

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

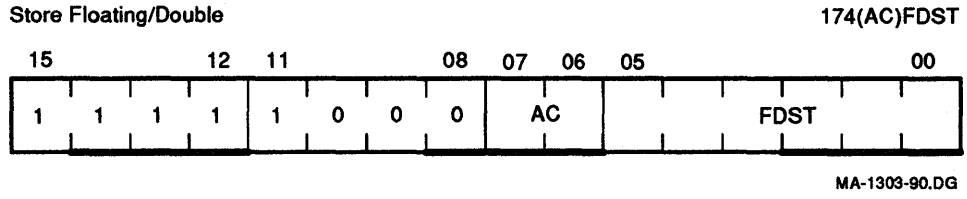


Figure 9–27 Store Floating/Double

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-E. 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

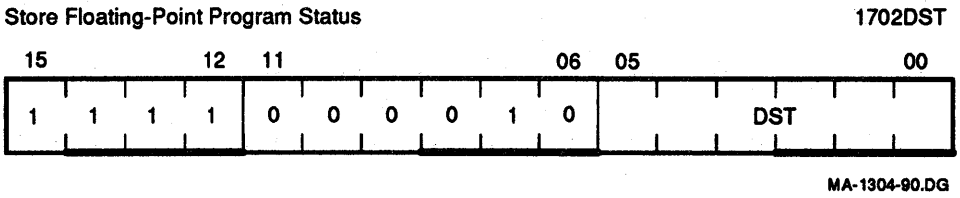


Figure 9–28 Store Floating-Point Program Status

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

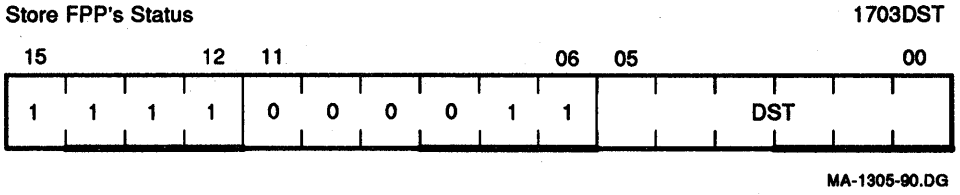


Figure 9–29 Store FPP's Status

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

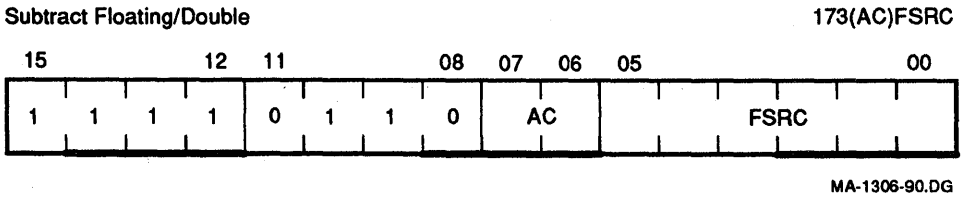


Figure 9–30 Subtract Floating/Double

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

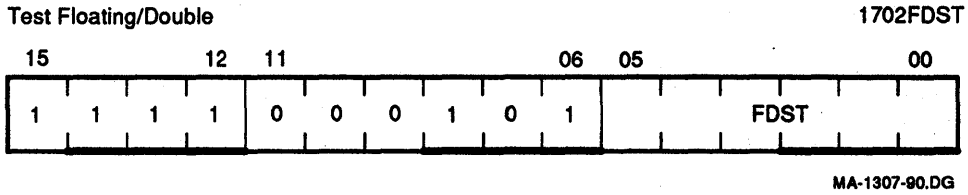


Figure 9–31 Test Floating/Double

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.

10
Programming Techniques

10.1 Introduction

The KDJ11-E offers a great deal of programming flexibility and power. Using the combination of the instruction set, the addressing modes, and the programming techniques, it is possible to develop new software or to use old programs effectively. The programming techniques in this chapter show the capabilities of the KDJ11-E. The techniques discussed involve PIC, stacks, subroutines, interrupts, reentrancy, coroutines, recursion, processor traps, programming peripherals, and conversion.

10.2 Position-Independent Code (PIC)

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 (that is, dependent on the virtual address to which it is bound).

The KDJ11-E 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 (for example, 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.

10.2.1 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 is familiar with the addressing modes described in Chapter 7.

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 (@#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 (for example, 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
```

10.2.2 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=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$:                                ;
        MOV         R2,F.PF(R4)     ;SET AST ENTRY POINT
        MOV         #BA, -(SP)      ;PUSH AST SERVICE
                                    ;ADDRESS
20$:                                ;
        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.

10.3 Stacks

The stack is part of the basic design architecture of the KDJ11-E. 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 10–1) require instructions capable of operating on bytes rather than full words.

10.3.1 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
```

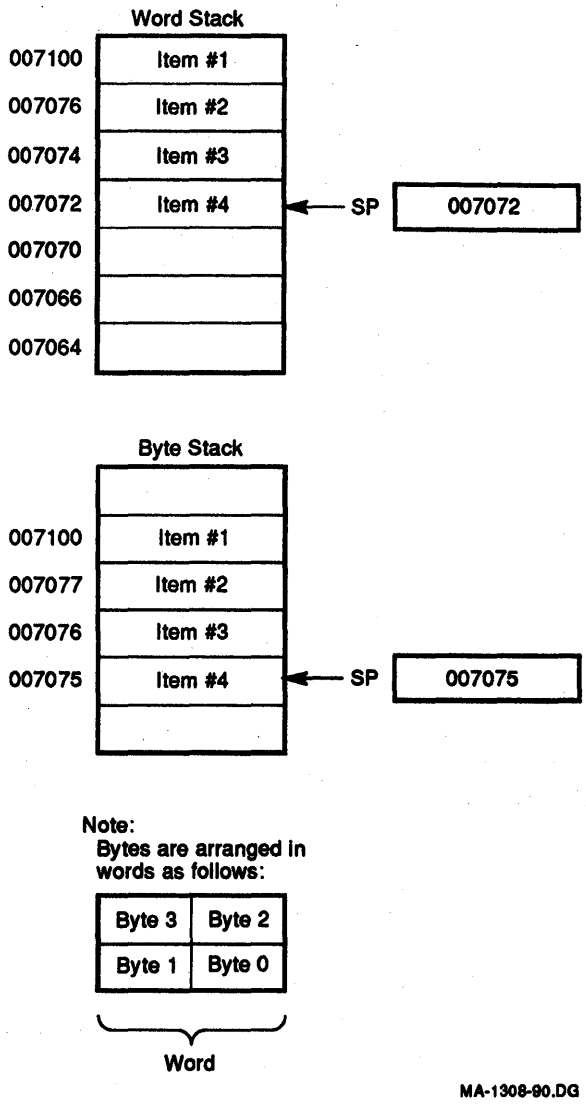


Figure 10–1 Word and Byte Stacks

10.3.2 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. Figure 10–2 illustrates the push and pop operations.

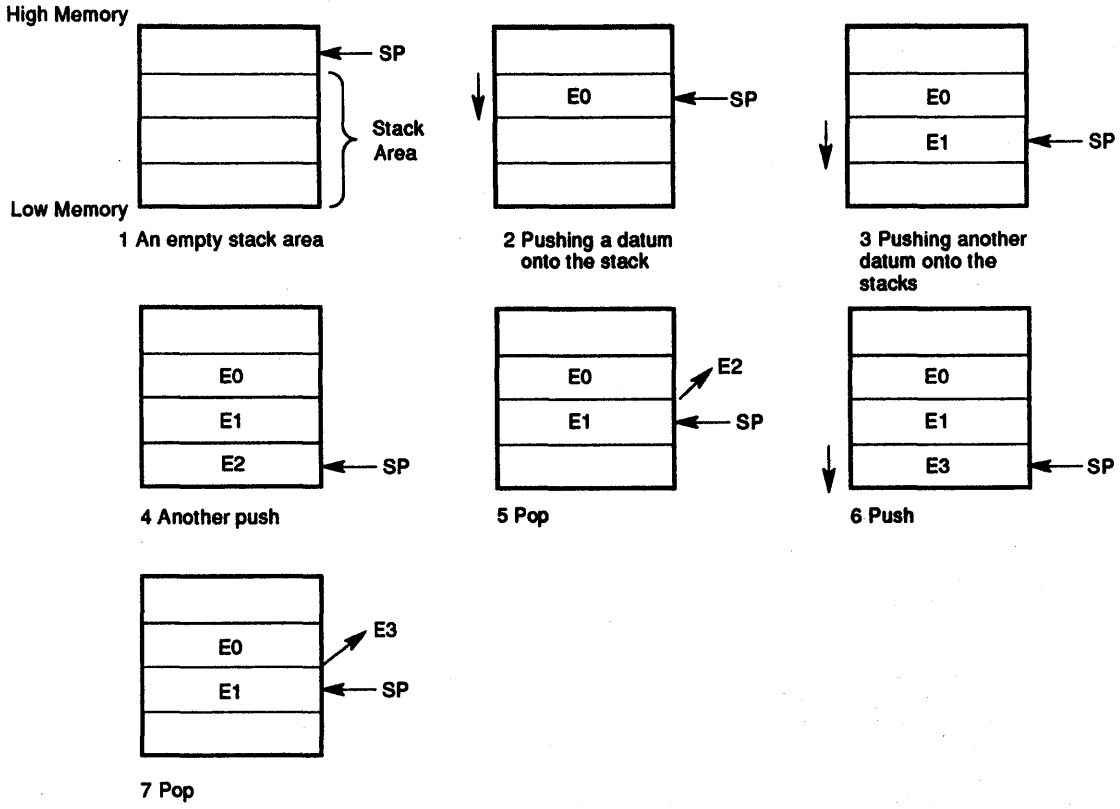


Figure 10–2 Push and Pop Operations

10.3.3 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

10.3.4 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 to 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, and so on.

10.3.5 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	MOV R1,TEMP1	;SAVE R1
	SUBR:		
076324	000074	*[1]	
076326	010267	MOV R2,TEMP2	;SAVE R2
076330	000072	*[1]	
.	.	.	
.	.	.	
.	.	.	
076410	016701	MOV TEMP1,R1	;RESTORE R1
076412	000006	*[1]	
076414	016702	MOV TEMP2,R2	;RESTORE R2
076416	000004	*[1]	
076420	000207	RTS PC	
076422	000000	TEMP1:0	
076424	000000	TEMP2:0	

[1]Index constants

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	

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. See Figure 10–3.

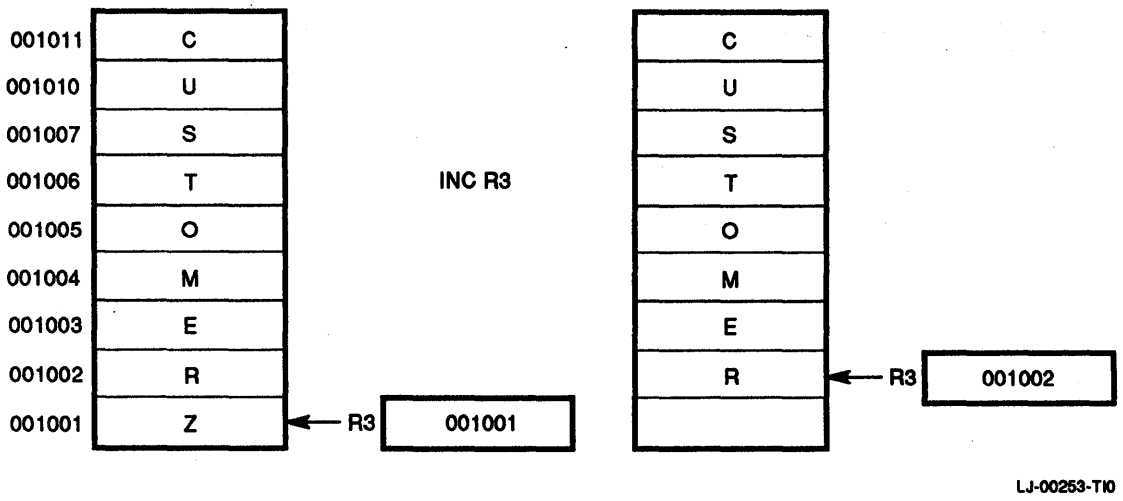


Figure 10–3 Byte Stack Used as a Character Buffer

10.3.6 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 10–4 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.

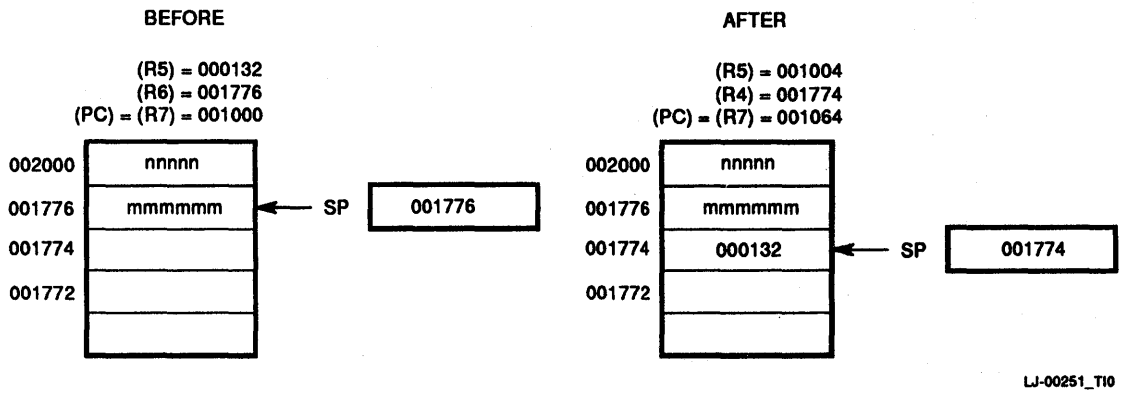


Figure 10–4 JSR Stack Condition Example

10.3.6.1 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.

10.3.6.2 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.

10.3.7 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.

10.3.7.1 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.

10.3.7.2 Nesting

Interrupts can be nested in much the same manner that subroutines are nested (Figure 10–5). It is possible to nest any arbitrary mixture of subroutines and interrupts with out any confusion. When the respective RTI and RTS instructions are used, the proper returns are automatic.

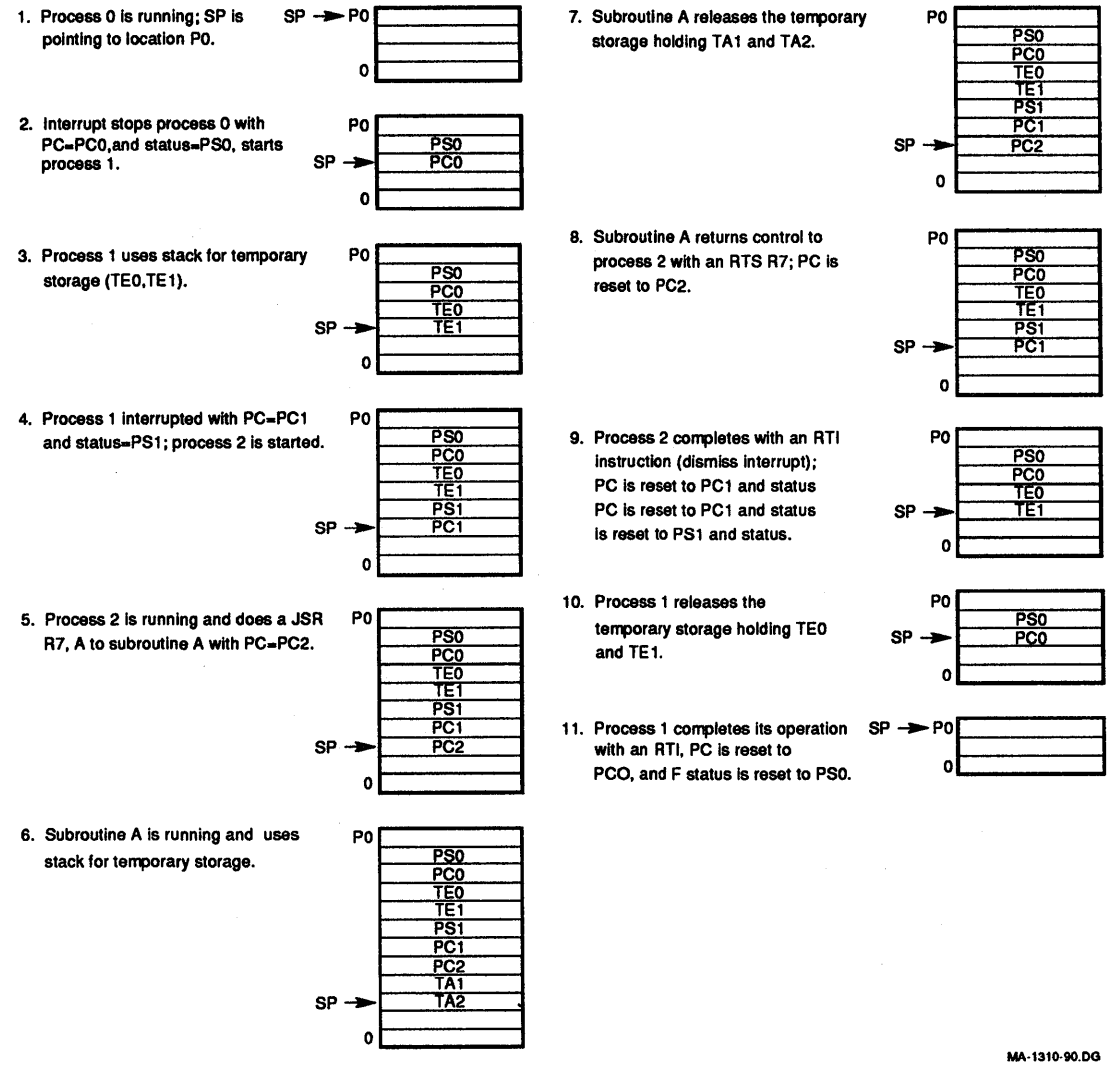


Figure 10–5 Nested Interrupt Service Routines and Subroutines

10.3.8 Reentrancy

Other advantages of the KDJ11-E 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 10–6 may occur.

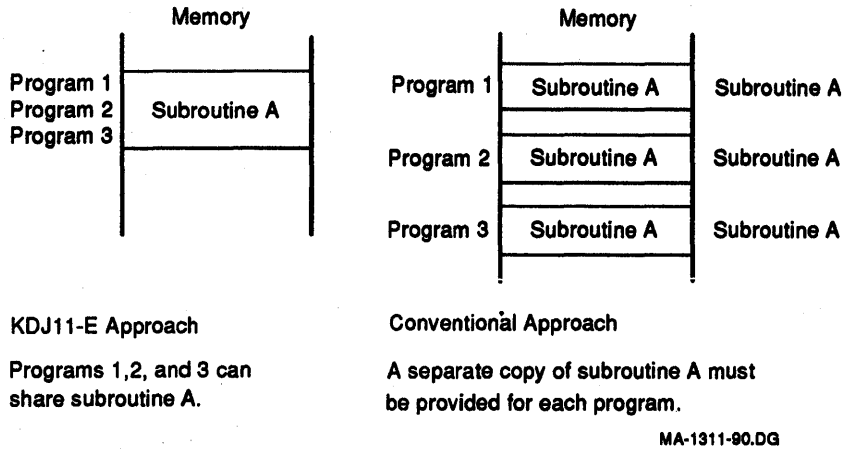


Figure 10–6 Reentrant Routines

10.3.8.1 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 (Figure 10–7).

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.

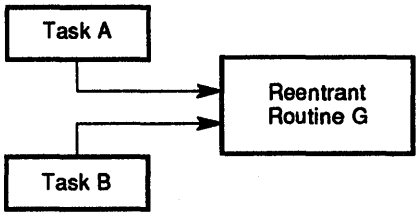


Figure 10–7 Sharing Control of a Routine

10.3.8.2 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 guidelines should be followed 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.

10.3.9 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)+

10.3.9.1 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 10–8. Notice that the coroutine linkage cleans up the stack with each control transfer.

ROUTINE A	STACK		ROUTINE B	COMMENTS
MOV #LOC-(SP)	LOC	← SP		LOC is pushed onto the stack to prepare for the coroutine call.
JSR PC,@(SP)+ (PC0)	PC0	← SP	LOC:	When the call is executed the PC from routine A is pushed on the stack and execution continues at LOC.
	PC1	SP	JSR PC,@(SP)+ (PC1)	Routine B can return control to routine A by another coroutine call. PC0 is popped from the stack and execution resumes in routine A just after the call to routine B, (that is, at PC0). PC1 is saved on the stack for a later return to routine B.

MA-1348-90.DG

Figure 10–8 Coroutine Example

10.3.9.2 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 10–9.

- 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. See Section 10.3.6.1 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.

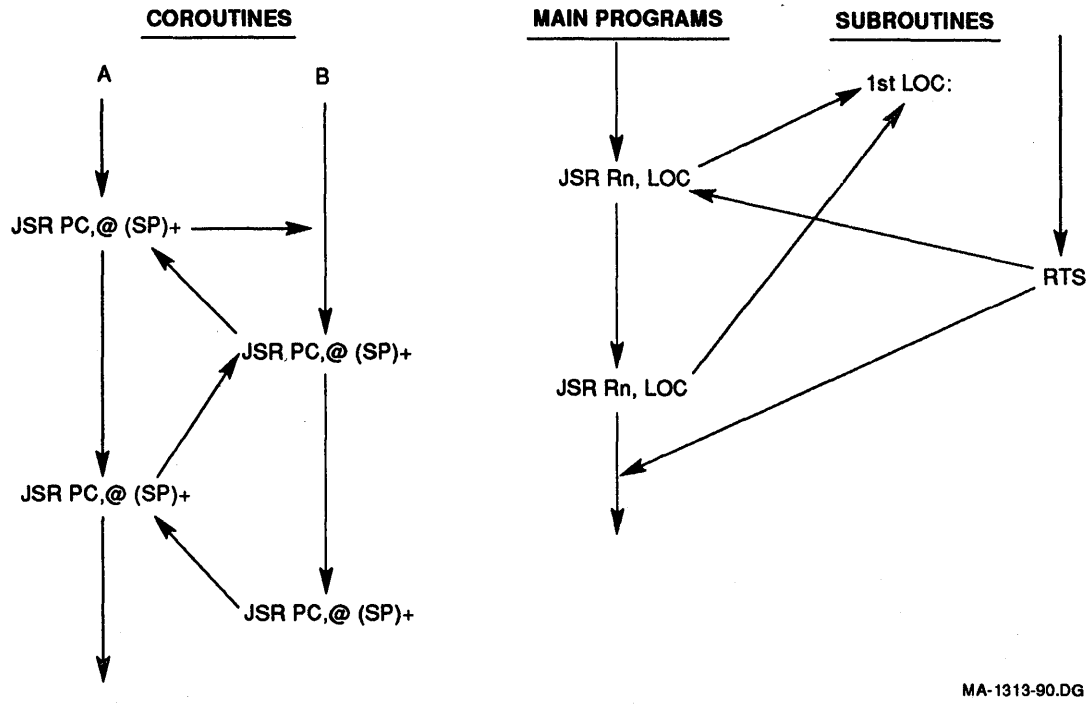


Figure 10–9 Coroutines Versus Subroutines

10.3.9.3 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, and so on.
- To add clarity to the process being performed, to ease in the debugging phase, and so on.

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 10–10 illustrates this example.

Coroutines can be utilized in I/O processing. Figure 10–10 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 10–11 illustrates a coroutine swapping interaction.

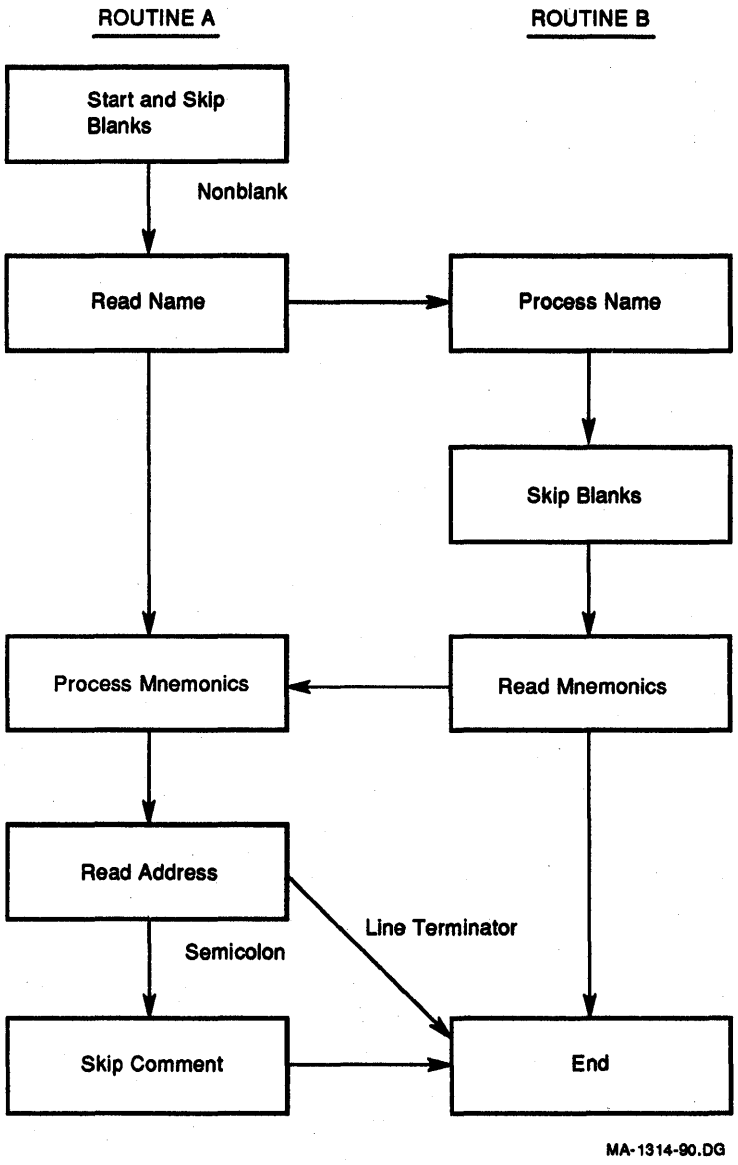


Figure 10–10 Coroutine Path

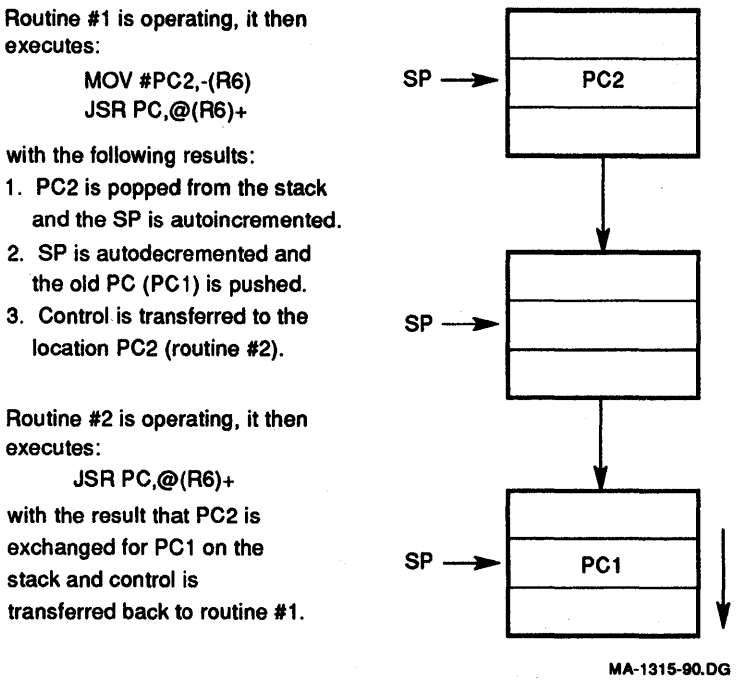


Figure 10–11 Coroutine Interaction

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 (PC1) is pushed.
3. Control is transferred to the location PC2 (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.

10.3.10 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. Figure 10–12 shows a typical flow for a recursive routine.

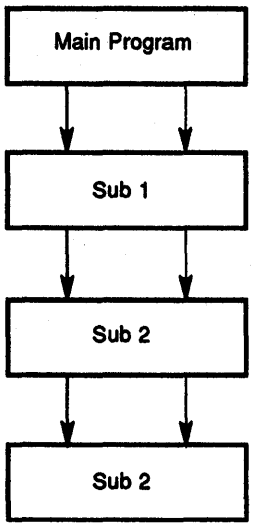


Figure 10–12 Recursive Routine Flow

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             ;RECURSE
        R5,DNCF
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 (that is, 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.

10.3.11 Processor Traps

Certain errors and programming conditions cause the KDJ11-E 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[1](MMUERR)	A memory management violation causes an abort and traps to location 250_8.
Timeout error[1] (BUSERR)	No response from a bus device during a bus transaction causes an abort and traps to location 4_8.
Parity error[1] (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[1]	If PSW bit 4 is set at the end of instruction execution, the processor traps to location 14_8.
Stack overflow[1] (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.

[1]Nonmaskable software cannot inhibit the condition. MMUERR, BUSERR PARERR are mutually exclusive when the processor is executing a program.

Condition	Description
Power fail[1] (PFAIL)	If the power OK bus signal (BPOK H) 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

[1]Nonmaskable software cannot inhibit the condition. MMUERR, BUSERR PARERR are mutually exclusive when the processor is executing a program.

10.3.11.1 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.

10.3.11.2 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 following example:

```
.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, and so on. 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.)

10.3.12 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:

   The following example uses 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, and so on.

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:

   The following example uses a 3-bit word:

```
        CLR R0          ;HIGH HALF OF ANSWER
        MOV #3,CNT      ;SET UP COUNTER
        MOV             ;MULTIPLICAND
        R1,MULT;

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 X 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
```

10.4 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 17777776. The KDJ11-E 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-E 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.

10.5 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.

10.6 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 Address	Program Counter	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				

Program Address	Program Counter	Label	Op Code	Operand	Comments
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 Counter	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  Program                Op
Address  Counter  Label  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                  ;SET UP POINTER
                        #SCORES,R2
                        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 Address	Program Counter	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

Program Address	Program Counter	Label	Op Code	Operand	Comments
		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 Counter	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

Program Address	Program Counter	Label	Op Code	Operand	Comments
					;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:	.ANCII/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 Counter	Label	Op Code	Operand	Comments
					;PROGRAMMING EXAMPLE
					;SUBROUTINE EXAMPLE
					;INPUT TEN VALUES, SORT, AND
					;OUTPUT THEM IN SMALLEST TO LARGEST ORDER

```
Program  Program                Op
Address  Counter  Label  Code      Operand  Comments
                        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                Op
Address  Counter  Label  Code      Operand  Comments
                                            ;PROGRAMMING EXAMPLE
                                            ;SUBROUTINE TO OUTPUT A CR & LF
                 CRLF:  TSTB @#TPS          ;TEST TTO READY STATUS
                        BPL CRLF
```

Program Address	Program Counter	Label	Op Code	Operand	Comments
			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		

10.7 Looping Techniques

Looping techniques are illustrated in the program segments that follow. The segments are used to clear a 50-word table.

1. Autoincrement (pointer address in general purpose register)

```
                    R0=%0
                    MOVE #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
```

A
Setup Parameters Worksheet

Two worksheets for each mode (video terminal or hardcopy) are provided for you to record the original setup parameter selections and the new setup parameters selections contained in the EEPROM of the KDJ11-E CPU module:

- Fill out the original worksheet when you install a KDJ11-E CPU module.
- Fill out the new worksheet when you change the parameter selections.

The information on these worksheets is used for programming any future replacement KDJ11-E CPU module.

Leave the worksheets with the system for future use.

Refer to Chapter 4 for more information on setup.

A.1 Original Setup Menu Worksheet - Video Terminal Support

KDJ11-E Monitor Version 1.06		Disable UBA ROM	Yes/No
Licensed to Digital Equipment Corporation			
Unibus System		Enable UBA 18-Bit Mode	Yes/No
Memory		Memory Intern	Do not change
EEprom		Rom on 173000	Yes/No
Time		Rom on 165000	Yes/No
		Power up Mode	Rom/Auto/ODT/Trap 24
Boot	Dev. Unit Address	Restart Mode	Rom/Auto/ODT/Trap 24
1		Power-on Self-tests	Yes/No
		Select Self-tests	Edit
2		User Boot	Edit
3		Alternate Boot Block	Yes/No
		LTC Register	Yes/No
4		Force Clock Interrupt	Yes/No
5		Clock Frequency	PS/50Hz/60Hz/800Hz
		Halt on Break	Yes/No
6		Trap on Halt	Yes/No
		Ignore Battery	Yes/No
		Lines on	176500/176600/DIS

Lines	Address / Vec	Baud	Data	Stop	Par
Line 1	/				
Line 2	/				
Line 3	/				
Line 4	/				
Line 5	/				
Line 6	/				
Line 7	/				

A.2 New Setup Menu Worksheet - Video Terminal Support

KDJ11-E Monitor Version 1.06

Licensed to Digital Equipment Corporation

Unibus System

Memory

EEprom

Time

Boot	Dev.	Unit	Address
1			
2			
3			
4			
5			
6			

Disable UBA ROM	Yes/No
Enable UBA 18-Bit Mode	Yes/No
Memory Intern	Do not change
Rom on 173000	Yes/No
Rom on 165000	Yes/No
Power up Mode	Rom/Auto/ODT/Trap 24
Restart Mode	Rom/Auto/ODT/Trap 24
Power-on Self-tests	Yes/No
Select Self-tests	Edit
User Boot	Edit
Alternate Boot Block	Yes/No
LTC Register	Yes/No
Force Clock Interrupt	Yes/No
Clock Frequency	PS/50Hz/60Hz/800Hz
Halt on Break	Yes/No
Trap on Halt	Yes/No
Ignore Battery	Yes/No
Lines on	176500/176600/DIS

Lines	Address / Vec	Baud	Data	Stop	Par
Line 1	/				
Line 2	/				
Line 3	/				
Line 4	/				
Line 5	/				
Line 6	/				
Line 7	/				

A.3 Original Worksheet - Hard Copy Printer Support

```
KDJ11-E Monitor  Version 1.06 30-July-1990
(C) Digital Equipment Corporation 1990

A  Memory Intern                (0) =   2MB  (1) =   4MB      =

B  Rom on 173000                (0) =    No  (1) =   Yes      =

C  Rom on 165000                (0) =    No  (1) =   Yes      =

D  Power-up Mode                (0) = Dialog
                                (1) =   Odt
                                (2) = Trap24
                                (3) =  Auto                   =

E  Restart Mode                 (0) = Dialog
                                (1) =   Odt
                                (2) = Trap24
                                (3) =  Auto                   =

F  Power-on Self-tests          (0) =    No  (1) =   Yes      =

G  Alternate Boot Block         (0) =    No  (1) =   Yes      =

H  LTC Register                 (0) =    No  (1) =   Yes      =

I  Force Clock Interrupt        (0) =    No  (1) =   Yes      =

J  Clock Frequency              (0) =   P/S
                                (1) =  50Hz
                                (2) =  60Hz
                                (3) = 800Hz                   =

K  Halt on Break                (0) =    No  (1) =   Yes      =

L  Trap on Halt                 (0) =    No  (1) =   Yes      =

M  Ignore Battery               (0) =    No  (1) =   Yes      =

N  Lines on                     (0) =   DIS
                                (1) = 176500
                                (2) = 176600                  =

O  Disable UBA ROM              (0) =    No  (1) =   Yes      =

P  Enable UBA 18-Bit Mode       (0) =    No  (1) =   Yes      =
```

A.4 New Worksheet - Hard Copy Printer Support

```
KDJ11-E Monitor  Version 1.06 30-July-1990
(C) Digital Equipment Corporation 1990

A  Memory Intern                (0) =   2MB  (1) =   4MB      =

B  Rom on 173000                (0) =    No  (1) =   Yes      =

C  Rom on 165000                (0) =    No  (1) =   Yes      =

D  Power-up Mode                (0) = Dialog
                                (1) =   Odt
                                (2) = Trap24
                                (3) =  Auto                   =

E  Restart Mode                 (0) = Dialog
                                (1) =   Odt
                                (2) = Trap24
                                (3) =  Auto                   =

F  Power-on Self-tests          (0) =    No  (1) =   Yes      =

G  Alternate Boot Block         (0) =    No  (1) =   Yes      =

H  LTC Register                 (0) =    No  (1) =   Yes      =

I  Force Clock Interrupt        (0) =    No  (1) =   Yes      =

J  Clock Frequency              (0) =   P/S
                                (1) =  50Hz
                                (2) =  60Hz
                                (3) = 800Hz                   =

K  Halt on Break                (0) =    No  (1) =   Yes      =

L  Trap on Halt                 (0) =    No  (1) =   Yes      =

M  Ignore Battery               (0) =    No  (1) =   Yes      =

N  Lines on                     (0) =   DIS
                                (1) = 176500
                                (2) = 176600                  =

O  Disable UBA ROM              (0) =    No  (1) =   Yes      =

P  Enable UBA 18-Bit Mode       (0) =    No  (1) =   Yes      =
```

Index

D

E

F

N

O

P

R

S