December 1977

This manual provides the information required to develop, assemble, load, and debug microprograms for the KMC11 auxiliary microprocessor. It includes descriptions of the KMC11 architecture, microinstruction repertoire, macro syntax and expansion, loader utility, KMCDA debugging aid, and special programming techniques.

KMC11
Programmer's Manual

Order No. AA-5244B-TC

SUPERSESSION/UPDATE INFORMATION: This document completely supersedes the document of the same name, Order No. AA-5244A-TC, published November 1977.

OPERATING SYSTEM AND VERSION:
RSX-11M V3.0
RSX-11D V6.2
IAS V2.0

SOFTWARE VERSION:
KMC11 MACRO V01
KMC11 LOADER V01
KMC11 DEBUGGER V01

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

Preliminary (First) Printing, November 1977
Second Printing, December 1977

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

FIGURES (Cont.)

TABLES

PREFACE

MANUAL OBJECTIVES AND READER CLASS ASSUMPTIONS

The objective of this manual is to provide experienced programmers with the information needed to develop, assemble, load, and debug microprograms for the KMC11-A auxiliary microprocessor.

The level of technical detail presented in this manual assumes that the reader is proficient in developing MACRO-11 assembly language programs and using the RSX-11 Task Builder to create an executable task image. In addition, the reader is assumed to be familiar with PDP-11 processor architecture and UNIBUS interfacing and have an in-depth knowledge of PDP-11 programming techniques.

STRUCTURE OF THIS MANUAL

This manual consists of seven chapters and an appendix.

Chapter 1 is a concise overview of the KMC11 and a summary of its operating environment. This chapter also contains microprogram development criteria, a list of reference documents, and an explanation of the notations used in this manual.

Chapter 2 describes the KMC11 microprocessor architecture with emphasis on the internal registers and data paths.

Chapter 3 provides detailed descriptions of the KMC11 microinstruction repertorie; a bit map precedes the description of each microinstruction. This chapter is subdivided into two major parts corresponding to the two classes of microinstructions: Move class and Branch class.

Chapter 4 defines the KMC11 macro syntax, the macros, and expansion of the macros. This chapter also lists the reserved symbols and outlines the general procedures to operate the KMC-11 MACRO assembler.

Chapter 5 describes the KMC11 loader utility. The basic KMC11 loader subroutine in this chapter provides the basis for the user to develop his own operating system specific loader. The KMC11 Loader (KMCLDR) running on RSX-11M is also described in this chapter for users who utilize this DIGITAL-developed component.

Chapter 6 provides the programmer with the detailed information necessary to use the KMC11 Debugging Aid (KMCDA); this chapter also includes details on task building KMCDA.

Chapter 7 considers special programming characteristics and techniques; some are unique to the KMC11 microprocessor and some are unique to KMC11 microprocessors operating in the multiprocessor environment.

Appendix A provides information ancillary to that contained in Chapter 7 on preventing overwriting data in the KMC11 Control and Status Registers and on ensuring data bit settling.

CHAPTER 1

INTRODUCTION

1.1 PURPOSE OF MANUAL

This manual provides the information needed by an experienced programmer to assemble, load, and debug microprograms for the KMC11-A auxiliary microprocessor. It therefore describes the KMC11 software tools that aid the programmer in developing, loading, and debugging a KMC11 microprogram.

1.2 KMC11 GENERAL DESCRIPTION

The KMC11 is an auxiliary processor designed for use on PDP-11 computer systems. It operates in parallel with the main CPU and has an architecture specifically suited to data movement, character processing, address arithmetic, and other functions necessary for controlling I/O devices, formatting data, and processing communications protocols. The KMC11 can be used in conjunction with all UNIBUS-based PDP-11 processors, from the PDP-11/04 to the PDP-11/70.

The functions performed by the KMC11 are determined primarily by the microprogram in the KMC11 control memory. This control memory is writable and may be changed whenever desired by the PDP-11 processor. In normal operation, the PDP-11 operating system would load the microprogram into the KMC11 control memory as part of system initialization; it would remain in the control memory until the system is subsequently reinitialized.

Software support of the KMC11 by a PDP-11 operating system consists of two parts:

1. the PDP-11 operating system driver, and

2. the KMC11 microprogram.

The microprogram must be tailored to the specific processing to be performed by the KMC11. The operating system driver interfaces the microprogram to the rest of the PDP-11 software. Communication between the microprogram and the operating system uses the KMC11 control status registers and is entirely defined by the software. Different applications may require different types of microcode/software interfaces. Operating system support for the KMC11 should always be considered in terms of the specific KMC11 microprogram employed.

1.2.1 Controlling Peripherals over the UNIBUS

A typical application of the KMC11 is the control of several peripheral devices attached to the UNIBUS. These devices, for example the DZ11 8-line asynchronous multiplexer, typically operate by programmed I/O, interrupting the PDP-11 processor for each character input or output. However, with the addition of a KMC11, this processor overhead can be substantially reduced and I/O throughput increased.

The PDP-11 processor communicates with the KMC11 on a message basis, with the KMC11 interrupting the processor at the end of each message. The KMC11 has direct memory access to message buffers in PDP-11 memory by means of NPR transfers. Data is transferred between the KMC11 and associated peripherals on a character-by-character basis through NPR transfers that address the peripheral's control, status, and data registers (CSRs). NPR transfers are conducted over the UNIBUS with no direct connection between the KMC11 and associated peripherals. The peripherals are operated with their interrupts disabled. The KMC11 periodically scans the peripheral status registers to determine when characters may be transferred. A single KMC11 can simultaneously control many transfers between peripherals and memory, keeping track of their status by using its 1024-byte data memory.

In addition to assembling and disassembling messages, the KMC11 microprogram can perform formatting, special character recognition, error checking, and other protocol functions.

1.2.2 Controlling Peripherals Attached to the External Connector

For high-speed operation, the KMC11 can be connected directly to a specially designed peripheral device, such as a DMC11 synchronous line unit.

In this mode of operation, the KMC11 communicates with the PDP-11 processor and memory as described above, but has a direct path to the high-speed peripheral via the KMC11's external connector. The external connector implements a simple bidirectional data port. Transfers are entirely under microprogram control.

1.3 OPERATING ENVIRONMENT

The KMC11 microcode can operate on any UNIBUS-based system (PDP-11/04 to PDP-11/70) with whatever hardware the user program requires so long as the means exist to load the previously developed microprogram.

To develop the KMC11 microprogram, the hardware, software, and software tools listed in Sections 1.3.1 to 1.3.3 are required.

1.3.1 KMC11 Microprogramming Tools Minimum Hardware Requirements

Any valid RSX-11M, RSX-11D, or IAS hardware configuration and one KMC11 auxiliary microprocessor comprise the minimum hardware requirements.

1.3.2 KMC11 Software Tools

The KMC11 microprogramming tools are used to aid a programmer in developing and debugging a KMC11 microprogram; the tools enable the programmer to assemble, load and debug the new microprogram.

The KMC11 software tools consist of three parts:

1. A MACRO-11 prefix file consisting of macro definitions of KMC11 instructions. The prefix file is assembled using MACRO-11 together with a file containing customer written microinstructions in the form of macro calls.

2. A utility program to load the writable control memory of a KMC11-A from a file containing an image of a KMC11 microprogram. The utility program runs as a privileged task under the supporting operating system. The input to the program is a file created by the Operating System Task Builder from the output of the MACRO-11 assembly.

3. A utility program to enable a programmer to debug interactively a microprogram running on a KMC11. The utility program runs as a privileged task under the supporting operating system and utilizes the maintenance features of the KMC11 hardware. The user may examine and modify the contents of the microprocessor internal registers, data memory and control memory. The user may start, stop, or single step the microprogram; or he may direct the KMC11 to execute a single microinstruction from the console. The user may optionally set up to eight breakpoints in the microprogram. Breakpoints are user-selected locations at which microprogram execution is to be halted temporarily to permit interaction between the microprogram and the user. If breakpoint support is to be used, the 16 highest locations in the KMC11 control memory must be reserved for the utility program. All addresses and data input and output to the user are in octal.

1.3.3 Prerequisite Software

The KMC11 microprogramming tools operate with one of the following PDP-11 systems:

RSX-11M, Version 3.0
RSX-11D, Version 6.2
IAS, Version 1.1

1.4 MICROPROGRAM DEVELOPMENT CONSIDERATIONS

Software support of a KMC11 by a PDP-11 operating system consists of two parts: the KMC11 microprogram and the PDP-11 operating system. The operating system driver interfaces the microprogram with the rest of the PDP-11 software. The microprogram must be tailored to the specific processing to be performed by the KMC11.

Development or acquisition of the microprogram can be through one of the following ways:

1. Users can use DIGITAL-developed microprograms and drivers supplied with DIGITAL's PDP-11 operating systems.

2. Users can use DIGITAL-developed microprograms in their own software environment. These users will develop their own operating system drivers.

3. Users can develop their own microprograms tailored to specific applications. These users can also develop their own operating system drivers. They may utilize the DIGITAL software tools described in Section 1.3.2.

4. Users can use DIGITAL's Computer Special Systems (CSS) to design and develop custom microprograms. CSS services avoid the need to develop in-house expertise and experience in the details of KMC11 microprogramming, yet provide the benefits of custom-microcode.

If KMC11 microcode is to be developed by the user, personnel should be senior-level and should have experience in programming I/O routines in several different minicomputer or microprocessor assembly languages. In addition, the personnel should be familiar with the problems of correctly synchronizing multiple processors.

1.5 REFERENCE DOCUMENTS

The following documents supplement the information in this manual. Some will aid in understanding the use and programming of the peripheral devices mentioned in this manual; others provide installation, operational and maintenance information for the KMC11.

Title	Document Number
KMC11 General Purpose Microprocessor User's Manual	EK-KMC11-OP
KMC11 General Purpose Microprocessor Maintenance Manual	EK-KMC11-MM
COMM IOP-DUP Programming Manual	AA-5670A-TC
COMM IOP-DZ Programming Manual	AA-5127A-TC
DMC11-DA Network Link Synchronous Line Unit Maintenance Manual	EK-DMCLU-MM
PDP-11 Peripherals Handbook	EB05961
RSX-11M MACRO-11 Reference Manual	DEC-11-OMMAA
RSX-11M Task Builder Reference Manual	DEC-11-OMTBA

1.6 NOTATIONS

The following notations are used throughout this manual. These notations represent actions or processes that express Boolean functions, programming conventions, and line printer or I/O terminal actions.

$\wedge$	Boolean AND	↑	Prompt
$\vee$	Boolean OR	CR	Carriage Return
$\veebar$	Boolean Exclusive OR	LF	Line Feed
~	Negation	[ ]	Bracketed Argument is Optional
→	Becomes		

CHAPTER 2

KMC11 MICROPROCESSOR ARCHITECTURE

2.1 CPU STRUCTURES

The KMC11 microprocessor is a parallel 16-bit general-purpose microcomputer. Its architecture and instruction set are optimized for the character processing environment specific to network communications and other I/O-oriented systems. The KMC11 instruction cycle has a period ranging from 300 to 330 ns, and microinstructions are executed from a 1K x 16-bit writeable control store referred to as the control RAM or CRAM. Microinstructions executed from the CRAM can access all KMC11 internal registers as well as data memory.

Within the instruction cycle period, the KMC11 can (1) execute internal register-to-register transfers; (2) initiate bus requests (interrupt the main CPU); (3) initiate NPR transactions; and (4) perform transfers between an internal register and data memory. In addition, the KMC11 can perform a wide range of arithmetic and logical functions on internally transferred data. (The main CPU is defined as the resident PDP-11 processor and its associated memory.)

All internal registers and data paths are 8 bits wide. In addition, the Control and Status Registers (CSRs) are both word and byte addressable from the UNIBUS, but byte addressable only from the KMC11.

The microprogram can read from or write into all internal registers and memories, except the microinstruction memory (CRAM), the memory address register (MAR), and the program counter (PC). The CRAM can be addressed for microinstruction execution only through the PC, and the MAR and PC can only be written into.

The KMC11 microprocessor executes two major classes of microinstructions: the Branch class and the Move class. Branch class microinstructions implement conditional and unconditional program jumps as well as subroutine entry and return. Move class instructions provide for interregister and intermemory data transfers and for logical and arithmetic operations on the transferred data.

Various configurations of each microinstruction class are used to construct the macroinstruction set described in Chapter 4. The detailed descriptions of each microinstruction class are presented in Chapter 3.

A major characteristic of the KMC11 is that the CRAM is separate from the data memory. A microprogram executed from the CRAM has free access to the data memory and all internal registers through microinstruction execution. However, a CRAM location cannot be modified through microinstruction execution.

Figure 2-1 is an overall block diagram of the KMC11. It shows the structure of the processor and the data paths connecting that

structure. This section summarizes the structural components accessed through internal buses, through direct microinstruction execution, and over the UNIBUS, as well as the Arithmetic Logic Unit (ALU), which is accessible to all internal registers.

2.1.1 INBUS/OUTBUS and INBUS*/OUTBUS* Accessed Registers

The INBUS/OUTBUS and the INBUS*/OUTBUS* (extended) serve as the data and address link between the KMC11 and its external environment. These busses link a group of internal registers and serve as the interface between the KMC11 and the UNIBUS as well as the KMC11 and an external device. They are accessed through assigned fields in the microinstruction. The registers addressed over these busses include the 16 bytes of the multiport RAM, the NPR control register, the μPMISC register, and registers associated with an external device. These registers are described in detail in Sections 2.1.1.1 through 2.1.1.4.

2.1.1.1 **Multiport RAM** - The multiport RAM is a 16-byte random-access memory divided into two 8-byte sections that are addressed through the INBUS*/OUTBUS* and the INBUS/OUTBUS. One section contains the CSRs for the KMC11 microprocessor, and the other serves as an address and data port for NPR transfers. As shown in Figure 2-1, the multiport RAM interfaces directly with the UNIBUS for the transfer of data words or bytes and the 16 low-order bits of a UNIBUS address. This memory is byte addressable from the microprocessor; the two 8-byte sections can be accessed by the microprogram for both read and write operations. The UNIBUS CSRs are accessed through the INBUS*/OUTBUS* and NPR interface through the INBUS/OUTBUS. From the UNIBUS, the CSRs are both word and byte addressable and can be read from and written into by the main CPU or another NPR device.

2.1.1.2 **NPR Control Register** - The NPR control register is an 8-bit formatted register that enables the microprogram to control input and output NPR transactions. This register also contains the two high-order bits of the UNIBUS address for an input NPR transaction. Note that an input NPR transaction involves the transfer of data from main CPU memory to the microprocessor. Conversely, an output NPR transaction involves transfer of data from the multiport RAM to main CPU memory.

The NPR register also stores the state of MAR bits 8 and 10, which can be read by the microprogram for detection of page and memory overflow. This register is addressed as a register in the INBUS*/OUTBUS* category.

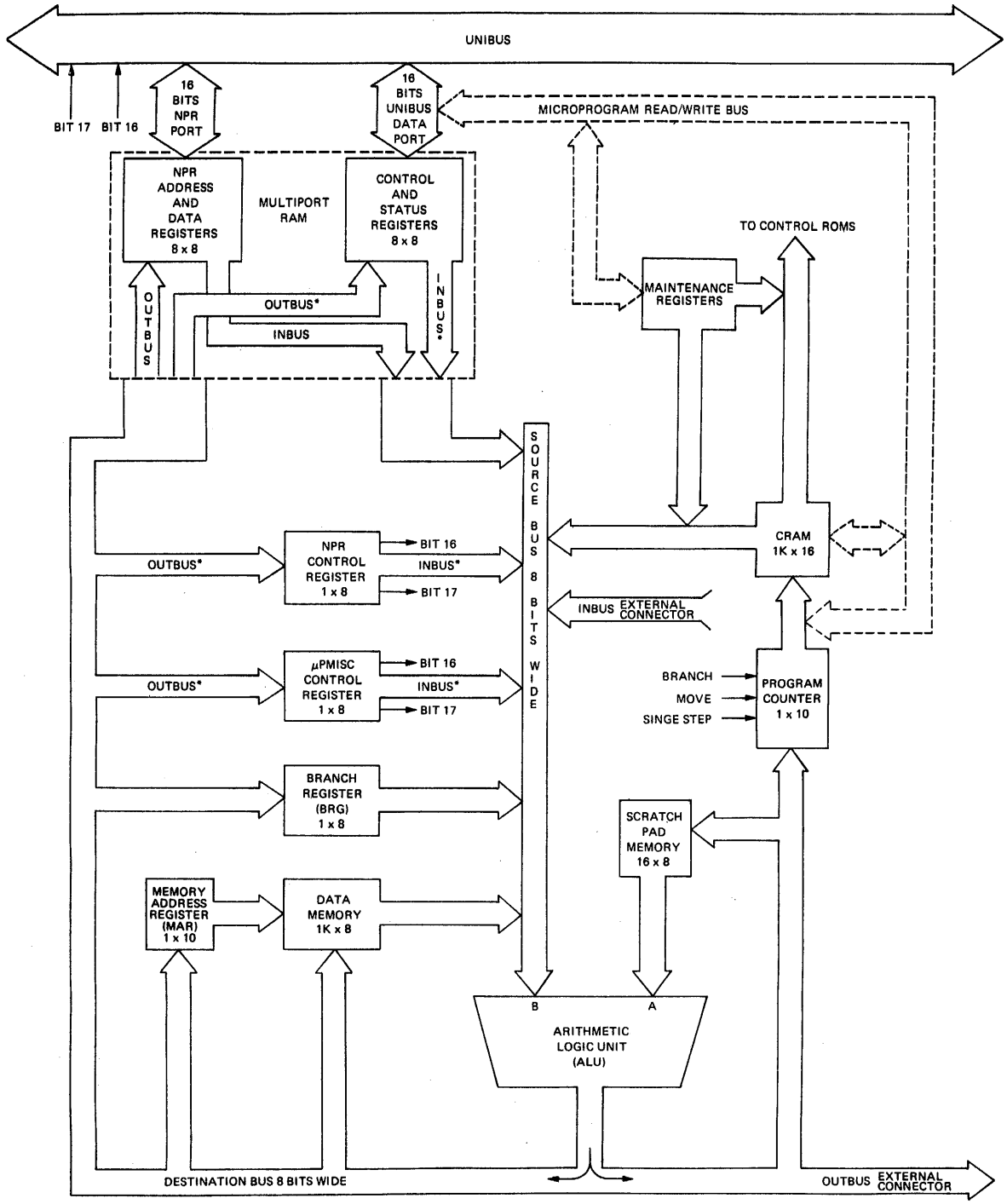


Figure 2-1 KMC11 Register and Data Path Structure

2.1.1.3 **Microprocessor Miscellaneous Control Register** - The microprocessor miscellaneous control (μPMISC) register is an 8-bit register that contains various control and function bits necessary for operation of the KMC11. For example, this register contains (1) an internal timing bit that times 50-μs intervals, and (2) a bit that flags the addressing of a nonexistent main CPU memory location by the microprocessor.

Specific bits are also available to the microprogram to control initiation of bus requests over the UNIBUS and to specify one of two vectors for appropriate interrupt processing. In addition, this register contains the two high-order UNIBUS address bits for an output NPR transaction. This register is addressed as a register in the INBUS*/OUTBUS* category.

2.1.1.4 **External Connector** - The KMC11 microprocessor is equipped with an external connector that supports eight parallel data input lines and eight parallel data output lines. These lines provide the KMC11 with direct access to a high-speed peripheral device such as a DMC11 line unit.

Eight internal 8-bit register addresses are assigned to the supported line unit. As shown in Figure 2-1, these registers (physically located in the external line unit) are read from the input bus and written from the output bus through the external connector. A typical line unit capable of being used with the KMC11 microprocessor is the Network Link Synchronous Line Unit (DMC11-DA). (The title and document number identifying the maintenance manual for this device are listed in Section 1.5.)

2.1.2 Components Accessed Through Direct Microinstruction Execution

Specific KMC11 registers are addressed as either sources or destinations of data or loaded through uniquely coded fields in the pertinent microinstruction. These registers include the branch register (BRG), data memory, the memory address register (MAR), the program counter (PC), and the scratch pad. These components are described in detail in Sections 2.1.2.1 through 2.1.2.4.

2.1.2.1 **Branch Register** - The branch register (BRG) is an 8-bit register used by Branch class microinstructions to specify branch conditions and to derive a branch address. The Move class microinstruction also uses the BRG as a data source and destination. A specific Move class microinstruction can perform a 1-bit right shift on the BRG. Through this Move microinstruction, the BRG can be set up to implement a subsequent conditional branch based on the state of certain BRG bits. The BRG also can be used as a general-purpose storage register by the microprogram.

2.1.2.2 **Data Memory and Memory Address Register** - As previously indicated, a major feature of the KMC11 architecture is the use of a separate instruction and data memory. The KMC11 data memory is a 1K x 8-bit RAM that is read/write accessible to the microprogram. The memory is addressed by a 10-bit memory address register (MAR) that is write-accessible only to the microprogram and cannot be read directly.

Through specific Move class instructions, the MAR can be loaded with an 8-bit address for page offsets (256-byte page) and the two high-order bits to designate one of four pages. A full 10-bit address is loaded in two steps: the eight low-order bits and the two high-order bits. This loading of the MAR can be performed in any sequence desired. In addition, the full 10-bit contents can be incremented to address the next data memory location.

Since the microprogram cannot read the MAR, a page overflow bit and a memory overflow bit are stored in the NPR register. The microprogram uses the page overflow bit (bit 8 of the MAR) to detect page overflow and the memory overflow bit to detect total data memory overflow. In addition, the MAR can be incremented across page boundaries.

2.1.2.3 **Program Counter** - The KMC11 PC has two modes of operation, determined by the class of microinstruction currently being executed. When a Move class instruction is executed, the content of the PC is incremented by one at the conclusion of current microinstruction execution to address the next sequential microinstruction.

If a Branch class instruction is executed and designated branch conditions are satisfied, the result of ALU operation on the two microinstruction-specified operands or the value of a single operand is written into the PC to address the next microinstruction to be executed. Figure 2-1 shows that the microprogram can write into the PC only through the ALU, as a consequence of a Branch class microinstruction execution.

2.1.2.4 **Scratch Pad Memory** - The KMC11 scratch pad memory is a high-speed, read-write, random-access memory made up of sixteen 8-bit bytes, and addressed by the microprogram over the destination bus (Figure 2-1). Scratch pad memory output provides the sole input to the A-side of the ALU.

With Branch class microinstructions, the contents of a scratch pad location can serve as an operand for an arithmetic or logic operation with a second operand such as a data memory location or the BRG to derive a jump address. By using Move class microinstructions, data can be transferred between all internal registers and memories except for the CRAM, the PC, and the MAR. A single Move class instruction cannot be used to exchange the contents of two scratch pad locations. Rather, the contents of one location must first be moved to an internal register that inputs to the B-side of the ALU (such as the BRG). A second Move transfers it to the desired scratch pad location.

2.1.3 Components Accessed from the UNIBUS

The KMC11 structural components that are accessed only from the UNIBUS include the Control RAM (CRAM) and the two maintenance registers. The functions of these components are described in detail in this section.

2.1.3.1 **Control RAM** - The KMC11 control RAM (CRAM) is a 1K x 16-bit random-access memory characterized by an extremely high speed access cycle resulting in a microinstruction execution time ranging from 300 ns for most microinstructions to 330 ns for microinstructions accessing the multiport RAM. As previously stated, the microprogram

which controls the functioning of the microprocessor, is stored in and executed from the CRAM.

A loader residing in the main CPU loads the microprogram into the CRAM. A loader that operates as a task under RSX-11/IAS is described in detail in Chapter 5. Techniques for interfacing this loader with other operating systems are also presented in Chapter 5.

A loader transfers a microprogram stored in main CPU memory over the UNIBUS one microinstruction at a time for storage in the KMC11 CSRs. Each instruction is accompanied by the CRAM address for that instruction. For each microinstruction transferred, the loader sets two control bits in the CSR maintenance register: one to load the microinstruction address and another to store the instruction in the CRAM address designated.

Conversely, a given CRAM location can be read over the UNIBUS from the main CPU through use of the KMC11 CSRs. In this case only a PC address is stored in the CSRs and specific control bits configured. A main CPU program can then read the addressed CRAM location from the CSRs.

As shown in Figure 2-1, these transfers are accomplished from the CSRs over the microprogram read/write bus. This bus, although internal to the KMC11, cannot be accessed by the microprogram; rather, it is a facility employed by the main CPU for loading, debugging, and maintaining the KMC11 microprogram. Details on the structure and use of the microprogram read/write bus are covered in Section 2.2.2.

2.1.3.2 **Maintenance Registers** - One maintenance register is used to store a test microinstruction for subsequent execution, and the second serves to address a CRAM location for microinstruction loading. For this discussion, the first is referred to as the maintenance register and the second, the maintenance address register.

The microprogramming debugging function is performed through the maintenance register. During debugging, the KMC11 debugging aid (Chapter 6) must be able to read the registers and data memory internal to the KMC11. The debugging aid primarily resides in and is executed from main CPU memory with breakpoint support code in the KMC11.

In actual practice, the debugging aid, through use of specific CSR control bits, can store a microinstruction in the maintenance register, and then execute it as though it came from the CRAM. For example, a series of instructions could move the content of a data memory location addressed by the MAR to the CSRs (INBUS*/OUTBUS*) for retrieval and examination by the debugging aid.

During CRAM load time, the maintenance address register is loaded from the UNIBUS to address the CRAM location, which stores an associated microinstruction. This register is also used by the KMC11 debugger when a CRAM location is examined or modified. (See Section 6.1.1.)

2.1.4 Arithmetic/Logic Unit

The KMC11 arithmetic/logic unit (ALU) has two sets of 8-bit data input lines and one set of 8-bit data output lines (Figure 2-1). The two sets of input lines are referred to as the A-input and the B-input. The output lines are extended to form a destination bus to all

internal registers and memories in the microprocessor. The A-input accepts data from the scratchpad memory only; the B-input accepts data from the data, CRAM, and multiport memories as well as the NPR, BRG, and μPMISC registers.

For Branch class instructions, the KMC11 ALU implements jump address derivations, indexed jumps, and conditional jumps based on an arithmetic or logical condition. For Move class instructions, the ALU performs logical and arithmetic operations on registers and the contents of memory locations; the results are stored in a specified register or memory over the destination bus. As shown in Figure 2-1, the CRAM cannot be accessed through the ALU.

The ALU also asserts the Carry (C) bit and the Zero (Z) bit in response to the various arithmetic and logic operations implemented by Move class microinstructions. The C-bit is asserted as a binary one when a given addition produces a carry term, and as a zero when a given subtraction produces a borrow term. Similarly, when a given arithmetic or logic operation produces a result of all ones (377 octal), the Z-bit is asserted as a binary one.

2.2 DATA PATHS

Within the structure of the KMC11, there are four defined data paths:

1. the source bus,
2. the destination bus,
3. the UNIBUS interface, and
4. the microprogram read/write bus.

The first two paths in this list are concerned with the transfer of data between KMC11 internal registers and memories. The third provides for transfer of data over the UNIBUS between the KMC11 and the main CPU. Finally, the fourth data path listed, the microprogram read/write bus, facilitates the loading of microinstructions from the UNIBUS to the CRAM or to the maintenance registers. The functional descriptions of these buses (Sections 2.2.2 through 2.2.5) are preceded by a discussion of the concept of source destination data transfers as implemented by the KMC11 microprocessor (Section 2.2.1).

2.2.1 Source Destination Data Transfer

Data transfer between internal registers and memories is conducted exclusively over the source bus and destination bus (Figure 2-1). A data source is always an input to the B-side of the ALU, and a destination is always the internal register or memory designated as the recipient of ALU output.

In general, the data source specified in the microinstruction is one of two operands to be operated on by the ALU in a manner designated by the instruction; the second operand is the contents of a scratch pad location. For data memory, the location containing the source operand is the location addressed by the current contents of the MAR. When the INBUS/OUTBUS or INBUS*/OUTBUS* is designated as the source of an operand for input into the B-side of the ALU, the operand is addressed by a field in the microinstruction. Where relevant, a scratch pad address is also specified by a microinstruction field.

For certain configurations of Branch class instructions, the source operand and the scratch pad location addressed by the pertinent microinstruction are operated upon by the ALU to derive a jump address. This derived address would point to a page offset in the CRAM containing the next microinstruction to be executed if the branch conditions are satisfied. Note that page address bits 8 and 9 are also contained in the microinstruction to form the full 10-bit address.

For Move class instructions, the source operand and the scratch pad location addressed by the pertinent microinstruction can be operated upon by the ALU with the result transferred to the destination designated by the microinstruction. Alternatively, the source operand can be transferred directly to the designated destination without change. Note that the use of internal registers and memories as destinations of data has relevance only to Move class instructions since the PC is the only possible destination for the results of Branch instruction execution.

Specific internal registers such as the scratch pads, the NPR control register, the μPMISC register, the CSRs, and the NPR data and address registers require a subaddress field in the microinstruction for proper referencing. Therefore, when any of these registers is used as a data source or destination, a field in the microinstruction is reserved for the specific address. When data memory is designated as a data destination, the memory location to contain the transferred data is addressed by the current content of the MAR. Since the MAR can be a data destination, it can be set to address a desired data memory location by a Move instruction (Figure 2-1).

2.2.2 Source Bus

Figure 2-1 shows that the KMC11 internal source bus is a parallel 8-bit data path that accepts data from one of eight separate sources as designated by the currently executed microinstruction. These sources include the BRG, the registers addressed through the INBUS and INBUS*, the external connector, the data memory, and the low-order byte of the current microinstruction. This bus connects to the B-side of the ALU providing for input of a source operand from the selected source register or memory location.

At microinstruction execution time, the state of the instruction source field specifies the source operand, thereby selecting the specific register or memory as the source of data input to the B-side of the ALU.

2.2.3 Destination Bus

Figure 2-1 shows that the destination bus is also a parallel 8-bit data path that conveys the output of the ALU to all internal registers and memories. The output bus routes data to six separate destinations: (1) BRG, (2) MAR, (3) data memory, (4) OUTBUS and OUTBUS* registers including the NPR and μPMISC registers, the external connector, the CSRs and the NPR data and address registers, (5) scratch pad, and (6) PC.

Move class microinstructions can designate any of these registers and memories, with the exception of the PC, as a data destination. As previously noted, the PC serves as a data destination only for the results of a Branch class microinstruction execution.

2.2.4 UNIBUS Interface

As shown in Figure 2-1, the destination bus, in conjunction with the source bus, provides the path not only for internal data transfers, but also for transfer of data between internal registers and memories and the main CPU. The interface between the KMC11 microprocessor and the UNIBUS is the multiport RAM. Through the multiport RAM, data is transferred over the UNIBUS between the KMC11 and the main CPU by way of the CSRs or under NPR control.

I/O data routed through the CSRs is controlled by the KMC11 and the main CPU through the use of a microprogram-defined scheme. In a typical microprogram implementation such as COMM I/OP, data received and transmitted through the CSRs takes the form of structured commands.

Note that the programmer must design his CSR-accessing algorithms in a manner that implements the orderly discipline necessary to the parallel operation of a KMC11 and an associated PDP-11. This discipline must take into account all the detailed programming considerations described in Chapter 7.

CAUTION

Failure to implement a properly disciplined KMC11 CSR accessing algorithm can lead to timing problems and cause unreliable system operation.

In COMM I/OP, one set of these commands permits a user program in the main CPU to configure and initialize a KMC11 microprogram as well as assign and deassign main CPU buffer space accessed by that program. Similarly, a second command set permits a KMC11 microprogram to transfer certain control information and communicate normal and error completions to the main CPU. These commands are described in detail in the COMM IOP-DUP and -DZ programming manuals. (Refer to Section 1.5 for document numbers.)

All data transfers between the KMC11 and the main CPU, with the exception of user-designated control information normally passed through the CSRs, are typically performed by NPRs. An NPR transaction can access any UNIBUS address including main CPU memory and any I/O control register in the I/O page. These NPR transfers are conducted under the control of the KMC11. As indicated in Figure 2-1, NPR transfers occur over the UNIBUS from the KMC11 to a UNIBUS address and from a UNIBUS address to the KMC11. An NPR that transfers data from the KMC11 to a UNIBUS address is designated an output NPR, and the converse, an input NPR. The INBUS/OUTBUS section of the multiport RAM is partitioned to store the 16 low-order bits of a UNIBUS address and the associated 16 data bits for both input and output NPRs. Actual control of an NPR transfer, which is performed asynchronously with respect to the microprogram, is exercised by the NPR control register. As previously indicated, the extended bus address bits (bits 16 and 17) for an output NPR are contained in the μPMISC register and for an input NPR these bits are contained in the NPR register.

2.2.5 Microprogram Read/Write Bus

Unlike the source bus, the destination bus, and the UNIBUS interface, the microprogram Read/Write bus is accessible only to the main CPU over the UNIBUS and cannot be accessed in any manner by the KMC11 microprogram. The purpose of this data path is to provide for loading the microprogram into the CRAM from the main CPU. In addition, a user program in the main CPU (for example, a debugger) can use this path to write into and read from the maintenance registers as well as read from a CRAM location addressed by the PC.

Figure 2-1 shows that this data path originates at the KMC11 UNIBUS data interface connecting to the INBUS/OUTBUS section of the multiport RAM. In the process of loading microinstructions into the CRAM or maintenance registers or reading from these sources, the data is stored in the INBUS*/OUTBUS* section of the multiport RAM. Data is also routed onto or off the microprogram read/write bus directly onto or from the UNIBUS. Note that data is moved over this path by the user program through programmmed data transfers involving 16-bit address and data words only. As previously stated, control of this data path is maintained by a user program through specifically assigned bits in the high byte of CSR0.

2.3 REGISTER AND MEMORY FORMATS

The operational components of the KMC11 microprocessor that are assigned a fixed format for control purposes include the following: (1) the KMC11 CSRs, (2) the NPR data and address registers, (3) the NPR control register, (4) the μPMISC register, and (5) the BRG.

The functions performed by the remaining KMC11 components, for example, the CRAM, the scratch pad, the data memory, and the maintenance instruction register, do not require assigned formats. An illustration of each format along with a detailed description of the function performed by each field within a register or memory is presented for each pertinent KMC11 component in Sections 2.3.1 through 2.3.4.

2.3.1 KMC11 CSR Format

As shown in Figure 2-2, the 8-byte partition of the multiport RAM, which is addressed through the INBUS*/OUTBUS*, contains the UNIBUS CSRs for the KMC11 microprocessor. These registers are assigned the following UNIBUS addresses in the floating I/O page: 76XXX0, 76XXX1, 76XXX2, 76XXX3, 76XXX4, 76XXX5, 76XXX6, and 76XXX7, with the even addresses forming the word boundaries. These CSRs are both word and byte addressable from the UNIBUS. For purposes of identity (Figure 2-2), the CSR bytes are referred to by the names BSEL0 through BSEL7, and on word boundaries by the names SEL0, SEL2, SEL4, and SEL6. However, from the KMC11, the CSRs are byte addressable only.

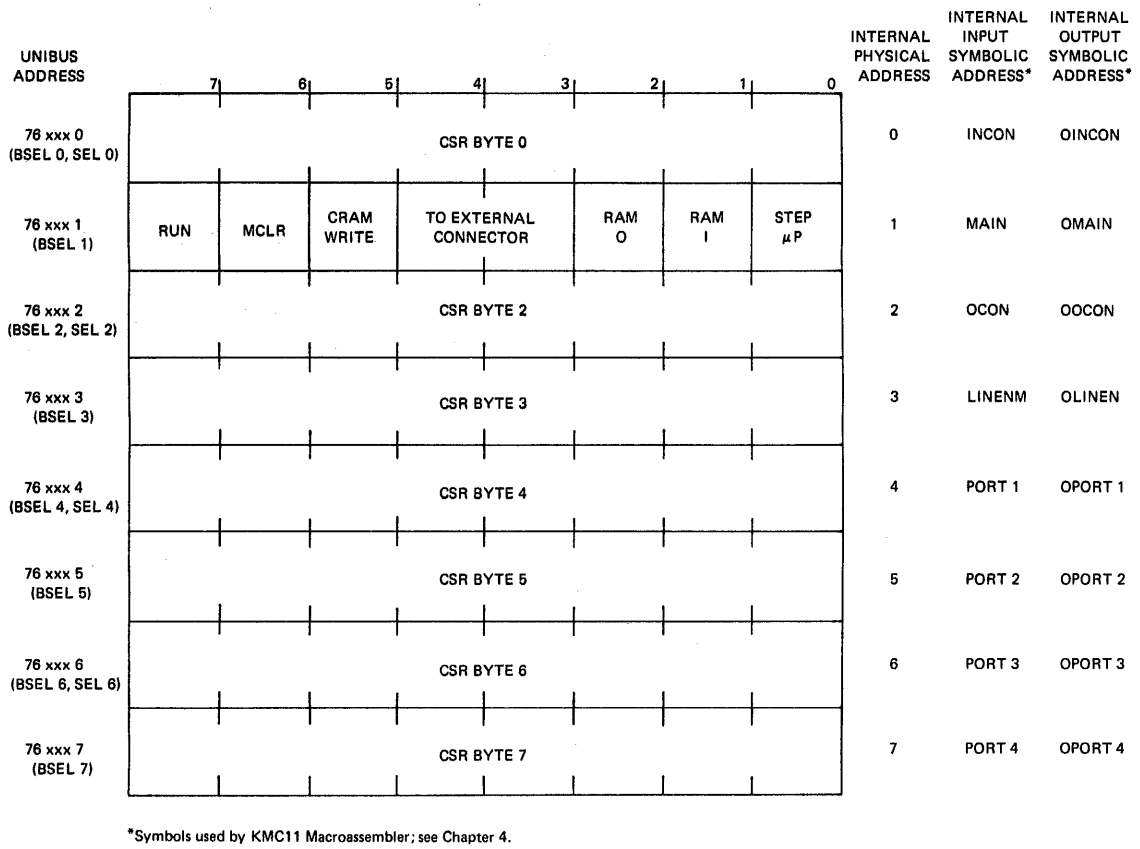


Figure 2-2 KMC11 CSR Format

Figure 2-2 shows that the only CSR having a fixed or hardware-defined format is CSR 1. CSR 1, like the remaining CSRs, exist as read/write locations in the multiport RAM. In addition, the CSR 1 control bits have associated hardware logic elements that implement the functions performed by these bits. The user program can set and clear these bits and the associated logic elements over the UNIBUS through programmed data transfers. The microprogram can also access the multiport RAM through the internal data paths to set and clear CSR 1 control bits. However, the states of the associated logic elements are not altered when the microprogram changes the state of a corresponding bit. Therefore, the microprogram can never assume that the state of a CSR 1 control bit reflects the state of the associated logic element.

Formats for the remaining CSRs are a function of the specific application microprogram; therefore, the descriptions that follow are concerned only with CSR 1.

NOTE

The symbolic addresses assigned the CSR registers shown in Figure 2-2 are those defined by the KMC11 Macroassembler described in Chapter 4.

The user program sets bit 6 of CSR 1, the MCLR (master clear) bit, and bit 7 (the RUN bit) at KMC11 initialization time to clear all condition-sensitive logic and to place the microprocessor in the run state.

Setting the MCLR bit performs the following actions on the KMC11 microprocessor:

1. clears all hardware logic related to the maintenance bits in CSR1,
2. clears all flip-flops related to the NPR register with the exception of the flip-flop associated with NPR REQ (Figure 2-4),
3. clears all flip-flops related to the μPMISC register,
4. clears the Z-bit and the C-bit,
5. clears the BRG, the MAR, and the PC,
6. clears the maintenance instruction register, and
7. clears the RUN bit and zeros all timing signals.

However, setting the MCLR bit has no effect on the following KMC11 components:

1. the multiport RAM,
2. the CRAM,
3. data memory,
4. the scratch pad, and
5. the flip-flop associated with the NPR REQ bit.

The NPR REQ bit is cleared independently due to its function in the NPR process by the UNIBUS control signal DATA BUS INIT asserted by the main CPU. Delays implemented through ACLO and PGM CLK (Section 2.3.3) are not cleared but expire at the end of the delay associated with these signals.

With MCLR set, a PDP-11 program can then immediately set the RUN bit through execution of an instruction. The specific conventions concerning the setting and clearing of the RUN bit are detailed in Chapter 7. Detailed examples of the use of the MCLR and RUN bits at initialization time are contained in the COMM IOP-DUP and COMM IOP-DZ programming manuals. (Refer to Section 1.5 for document numbers.)

Bits 2 and 5, RAM 0 and CRAM WRITE (Figure 2-2), are used together to load a CRAM address and a microinstruction over the microinstruction read/write bus into the maintenance address register and CRAM, respectively. These bits provide the mechanism for loading a complete microprogram into the KMC11 CRAM and for changing unique microinstructions.

In this process, the user sets RAM 0 to one, and loads (1) the address of the CRAM location to which the microinstruction is to be written into SEL4 (this action loads the maintenance address register but does not change the PC content) and (2) the actual instruction into SEL6. The user then sets CRAM WRITE to one, and the instruction is loaded

into the CRAM location addressed by the new value of the maintenance address register.

At this point, both the RAM O and CRAM WRITE bits may be cleared by the PDP-11 program prior to performing the next microinstruction write. However, instead of clearing both RAM O and CRAM WRITE, CRAM WRITE alone can be cleared, and the SEL6 read to verify that the microinstruction was correctly written into the proper location. In this case, the CSR SEL6 contains the contents of the CRAM location addressed by the maintenance address register. SEL4 also contains the contents of the CRAM location written into. However, after RAM O is cleared, SEL4 contains the address of the last CRAM location written. Similarly, the user can read a selected CRAM location by loading the desired PC address into SEL4, setting RAM O, and then reading SEL6.

As an example of this CRAM write/read process, consider the storing of a microinstruction that transfers the contents of the INBUS* byte 7 to location 4 of the scratch pad into CRAM location 1754 and then verifies the write:

```
MOVB    #4,BSEL1        ;Set RAM O to one

MOV     #1754,SEL4      ;Load address

MOV     #123146,SEL6    ;Load microinstruction

BISB    #40,BSEL1       ;Set CRAM WRITE

BICB    #40,BSEL1       ;Clear CRAM WRITE

CMP     #123146,SEL6    ;Verify that instruction is loaded
```

After clearing CRAM WRITE, the user can examine the contents of SEL4 and SEL6 directly from the main CPU console. This method of single instruction writing and reading is useful during the microprogram debugging process for changing the contents of specific CRAM locations and verifying that the change is correct. RAM O must be cleared prior to writing the next instruction.

For example, when loading a complete microprogram, the first location in that program could be the base address to be incremented for each microinstruction stored in the CRAM. The microprogram could be stored in a main CPU buffer to be accesssed for loading by programmed data transfers in the autoincrement mode. Note that RAM O and CRAM WRITE should be cleared after each microinstruction is loaded (Chapter 6).

Combinations of bits 0 and 1 of BSEL1, RAM I and STEP μP are used to write a microinstruction from the main CPU into the KMC11 maintenance instruction register. As described in Section 2.1.10, a debugging aid resident in the main CPU can store a microinstruction in this register from the UNIBUS and then execute that instruction in place of a specific instruction in the CRAM.

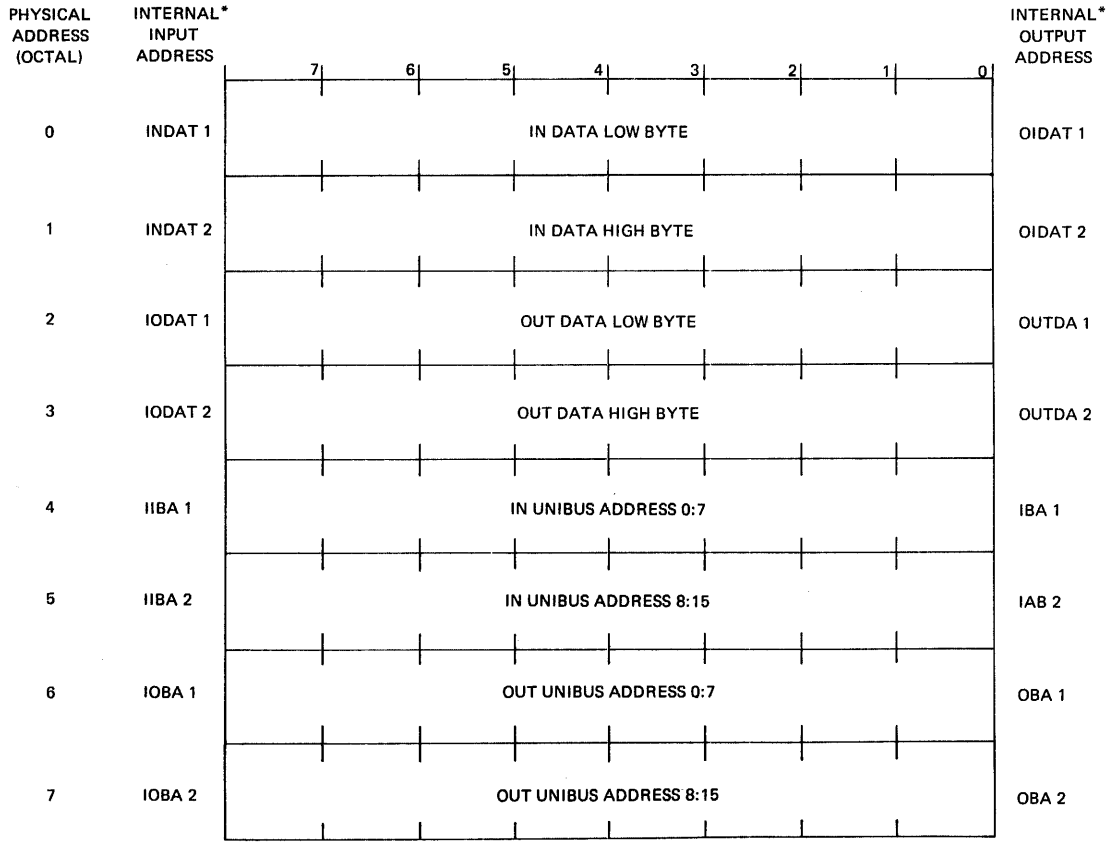
The first step in loading a microinstruction into the maintenance instruction register involves clearing the RUN bit (Bit 7 of BSEL1). Next, the microinstruction is loaded into SEL6, and RAM I is set to load the microinstruction into the maintenance instruction register. Finally, setting STEP μP causes the instruction to be executed from the maintenance instruction register rather than the CRAM. The PC is also incremented each time STEP μP bit is set. With RAM I cleared, setting STEP μP causes the microinstruction in the next CRAM location to be executed. In addition, the contents of the maintenance instruction register can be read from either SEL4 or SEL6. The user must first clear the STEP P bit, and then set RAM I along with RAM O and read the contents of SEL4 or SEL6.

As shown in Figure 2-2, bits 3 and 4 of BSEL1 are labeled "to external connector." These bits can serve as status/control bits for an external device.

2.3.2 NPR Address and Data and NPR Control Register Formats

The NPR transactions performed by the KMC11 directly involve the NPR address and data registers, which are addressed through the INBUS/OUTBUS and the NPR control register. Note that the NPR register is addressed through the INBUS*/OUTBUS*. As shown in Figure 2-3, the NPR address and data registers serve as the buffer for the 16 low-order UNIBUS address bits and the 16-bit data words for both in-NPR and out-NPR transactions. The NPR transactions performed by the KMC11 microprocessor are controlled by the NPR control register (Figure 2-4).

Before executing an NPR transaction, the KMC11 microprogram must store the 16 low-order bits of the UNIBUS address and the associated 16-bit data word (if the transaction is an out-NPR) in the NPR address and data registers. For an in-NPR, the source of UNIBUS address bits 16 and 17 is the NPR register, and for an out-NPR, the source of these bits is the μPMISC register. (See Figures 2-4 and 2-5.) The microprogram accesses the NPR address and data registers by referencing the NPR register and using the internal symbolic or physical addresses shown in Figure 2-3.



*Symbols used by KMC11 Macroassembler; see Chapter 4.

Figure 2-3 NPR Data and Address Register

INTERNAL PHYSICAL ADDRESS (OCTAL)	INTERNAL† INPUT ADDRESS	7	6	5	4	3	2	1	0	INTERNAL† OUTPUT ADDRESS
		*	**	**	*	*	*	*	***	
10	NPR	BYTE XFER	MAR 10	MAR 8	OUT NPR	INBA 17	INBA 16	NOT LAST XFER	NPR RQ	ONPR

* KMC11 can read and write these bits.
** KMC11 can only read these bits.
*** KMC11 can only set this bit to a one or read it. It is cleared automatically at completion of NPR.
† Symbols used by KMC11 Macroassembler; see Chapter 4.

Figure 2-4 NPR Control Register

With the 18 bits of the UNIBUS address and the related data stored appropriately in the INBUS/OUTBUS, the KMC11 microprogram can then set up the NPR control register for an out-NPR transaction. All the pertinent bits in the NPR control register can be set to the required state by moving a single 8-bit binary array appropriately configured into the register. Note that bits 5 and 6 (MAR8 and MAR10) can be read only by the microprogram and are unaffected by writing. The function performed by each NPR control bit is described in detail below.

Bit 0 of the NPR control register, NPR RQ, provides the start and end control for an NPR transaction. When set by the microprogram, an NPR transaction cycle is initiated. During this cycle, the UNIBUS physical address specified is accessed independent of microprogram operation. As a consequence, the microprogram can continue operation during an NPR transaction.

Upon completion of an NPR transaction, the NPR RQ bit is automatically cleared to designate completion of the transaction. If the transaction is an in-NPR, this action informs the microprogram that retrieved data is ready to be read from the NPR data registers INDAT1 and INDAT2 (register 0 and 1 of the INBUS). If the transaction is an out-NPR, the clearing of NPR RQ informs the microprogram that the data in the NPR data registers OUTDA1 and OUTDA2 (registers 2 and 3 of the INBUS) have been stored in the location specified by the UNIBUS address. If a nonexistent memory location is accessed by an NPR transaction, the transaction is terminated after 20 µs by the microprocessor, the NON EX MEM bit in the µPMISC register is set (Section 2.3.3), and the NPR RQ bit is automatically cleared.

The state of NPR register bit 4 (the OUT NPR bit) designates whether a given NPR transaction is to store data in or retrieve data from a specified UNIBUS address. When this bit is set to one, the transaction will be an out-NPR, and when zero, an in-NPR.

Bits 2 and 3 of the NPR register INBA 16 and INBA 17 are the high-order bits of the extended UNIBUS address for an in-NPR. The state of these bits is automatically placed on the UNIBUS along with the 16 low-order bits contained in bytes 4 and 5 of the OUTBUS/INBUS (Figure 2-3) during an in-NPR transaction to form the full 18-bit UNIBUS address. The corresponding bits for an out-NPR are contained in the same bit positions in the µPMISC register (Section 2.3.3).

Bit 7, BYTE XFER, applies only to out-NPR transactions. If a single byte is to be transferred during an in-NPR transaction, the microprogram must perform the NPR for the entire word and then retrieve the pertinent byte from either INDAT1 (low byte) or INDAT2 (high byte) (Figure 2-3).

CAUTION

Setting the BYTE XFER bit during an in-NPR transaction could have a detrimental effect on system operation.

When this bit is at zero, all out-NPR transactions are conducted on word boundaries. However, when set to one, bit 0 of the UNIBUS address determines whether the transaction involves the low byte or the high byte. For example, with BYTE XFER set, where address bit zero is zero, the contents of the NPR data register symbolically addressed as OUTDA1 is transferred to the low-byte position of the location specified by the UNIBUS address. Similarly, if address bit zero is one, the contents of OUTDA2 is transferred to the high-byte position of the location specified by the UNIBUS address. When performing a single-byte out-NPR, the programmer should be sure that the data is placed in the proper OUTBUS register: OUTDA1 for the low byte and OUTDA2 for the high byte (Figure 2-3).

NPR transactions can be conducted by the KMC11 microprogram singly or as a series of transactions executed sequentially. When NPR transactions are conducted singly, control or mastership of the UNIBUS is automatically relinquished when the transaction is completed. However, when conducting multiple sequential NPR transactions, it may be advantageous, for processing efficiency, to maintain bus control until the last transaction is completed. Bit 1 of the NPR register, the NOT LAST XFER bit, maintains this bus control throughout a series of NPR transactions. The KMC11 microprogram sets this bit along with other pertinent control bits when starting the first transaction in the series. The microprogram must clear this bit to relinquish bus control with the first instruction following the start of the last transaction in the series.

CAUTION

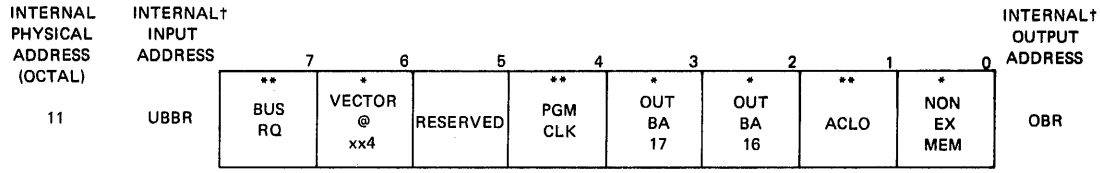
The maximum number of sequential NPRs that can be executed is determined by the specific PDP-11 system configuration. Exceeding this maximum can cause latency problems with such associated peripherals as mass storage devices and communications controllers.

The bit labeled MAR8 (Figure 2-4) always reflects the current state of bit 8 in the MAR. The bit labeled MAR10 in the NPR register does not reflect a bit state in the MAR, but rather can serve as an overflow indicator for the MAR. MAR10 is set to one when the MAR is incremented from 1777 octal to zero. If the MAR is incremented beyond 1777 a second time, MAR10 is set to zero. MAR10 is automatically cleared when the two high-order bits of the MAR are loaded (Section 3.1.1).

2.3.3 μPMISC Register Format

The μPMISC register is an internal register containing various function and control bits necessary for the operation of the KMC11 microprocessor. The format for this register is illustrated in Figure 2-5, and the detailed functions performed are presented below.

Bit 0, the NON EX MEM bit, is set by microprocessor hardware when a nonexistent UNIBUS location is addressed by the microprogram during an NPR transaction. The microprogram is responsible for monitoring the state of this bit and clearing it when set. If the microprogram does not do this on each NPR, it cannot identify a specific NPR that has failed.



* KMC11 can read and write these bits.
** KMC11 can only set these bits to one, they are reset automatically at the completion of a bus request, a power fail sequence, or by expiration of the pertinent delay period, whichever is applicable.
† Symbols used by KMC11 Macroassembler; see Chapter 4.

Figure 2-5 Microprocessor Miscellaneous (μPMISC) Register

Bit 1, ACLO, is set by the microprogram to initiate a power fail sequence at the main CPU. This bit clears automatically upon completion of the sequence. The period of this sequence is 500 μs.

Bits 2 and 3, OUT BA 16 and OUT BA 17, comprise the extended address bits for an out-NPR transaction. These bits are automatically transferred onto the UNIBUS during an out-NPR. The microprogram is responsible for the maintenance of these bits.

When the microprogram writes a one to PGM CLK, the PGM CLK bit goes to a zero and remains in that state for 50 μs. At the end of that period, it returns to the one state and remains in that state until the microprogram again writes a one to it. The 50-μs timing period provided by this bit is useful to the microprogram during operations such as communications line polling.

Another use of the PGM CLK bit is as a watch-dog timer. In this case, the microprogram writes a one to PGM CLK (the PGM CLK bit will go to the zero state for 50 μs) and then during the 50-μs zero state of that bit, the microprogram writes a series of ones to PGM CLK. The zero state of that bit will be assured for 50 μs after the writing of the last one; if the time period between writing ones exceeds 50 μs, the state of the PGM CLK bit will return to the one state.

It should be noted that the microprogram cannot stop the PGM CLK bit during any on-going timeout period. However, the microprogram can reset this bit at any time to start a new 50 μs timeout period.

Bit 7, BUS RQ, is set by the KMC11 microprogram to inform the main CPU that it is ready to send or receive data over the UNIBUS on a programmed-data-transfer basis. Setting this bit interrupts the main CPU. The vector location used by the main CPU to process the

interrupt is designated by bit 6, VECTOR @ XX4. When set to one, this bit implies a vector to location XX4, and when set to zero, implies a vector to XX0. The XX portion of a vector address is defined by switches on the KMC11 module. (See KMC11 General Purpose Microprocessor User's Manual, EK-KMC11-OP.) If the system designer requires an interrupt enable capability, he can designate a bit in a CSR as the interrupt enable bit for interpretation by the microprogram.

2.3.4 Branch Register Format

The Branch register (BRG) can be considered as both a formatted and an unformatted register in that it can serve as a general-purpose storage register as well as a branch designator for Branch class instructions. Figure 2-6 depicts the BRG as a formatted register.

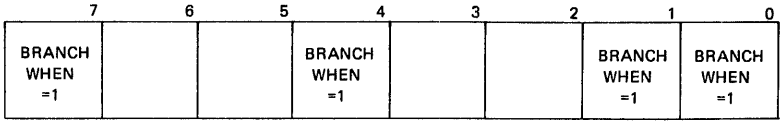


Figure 2-6 Branch Register

Use of the BRG as a formatted register is a function of Branch class microinstructions. When the BRG is used in this manner, a Branch class microinstruction can be configured to branch when BRG bit 0, bit 1, bit 4, or bit 7 is set to one. A branch cannot be made on the remaining BRG bits.

As an example of how the BRG can be used to implement decision-making microprogram branches, consider the execution of an NPR transaction. As previously indicated, the NPR RQ bit (bit 0 of the NPR register) is automatically cleared to zero when the transaction is complete. As a test for NPR completion, the microprogram can periodically move the contents of the NPR register into the BRG and execute a branch on the state of bit 0. If bit 0 is one, the transaction is still under way and a branch out of the test sequence is made. When NPR register bit 0 is zero, the next NPR transaction can begin. This technique can also be applied to the μPMISC register bits NON EX MEM (bit 0), ACLO (bit 1), PGM CLK (bit 4), and BUS RQ (bit 7).

The BRG can also be right-shifted under program control. In this process, the contents of the BRG are right-shifted one bit position, and bit 0 of the ALU output resulting from microinstruction execution is placed in BRG bit position 7. This capability serves to implement a single instruction one bit right rotate as well as permitting a test of the state of a specific BRG bit.

CHAPTER 3

KMC11 MICROINSTRUCTION REPERTOIRE

KMC11 microinstructions fall into two major categories: the Move class instruction and the Branch class instruction. Move class instructions provide the mechanism for performing data transfers between internal registers; Branch class instructions implement the conditional and unconditional branch functions. The basic functions of both Move and Branch class instructions include the ability to perform a designated arithmetic or logical operation on the source operands. For Move class instructions, the memory address register (MAR) can be incremented as a consequence of instruction execution or the resultant destination data can be stored in the MAR as well as the destination register.

The level of detail presented in this chapter is sufficient for a systems programmer who has experience with PDP-11 architecture, specifically the UNIBUS, to quickly understand the functions and application of the KMC11 microinstruction set. Systems programmers desiring further information on the PDP-11 UNIBUS should refer to Chapters 5 and 6 of the PDP-11 Peripherals Handbook, EB05961.

3.1 MOVE CLASS MICROINSTRUCTIONS

Figure 3-1 summarizes Move class microinstructions. Excluding the permutations provided by the 16 ALU functions, there are 40 basic Move class microinstructions. Note that Figure 3-1 presents the mnemonics for each Move class microinstruction field along with the binary equivalent and a brief description of the function performed during microinstruction execution by that field. In day-to-day microprogramming activity, the programmer can use Figure 3-1 as a reference.

A unique feature of Move class microinstructions is that the MAR control field can be used as needed by any instruction in the total set. Therefore, the functions and use of the MAR Control field are presented first, followed by a detailed description of each basic Move class microinstruction format and function. The Move class microinstructions are organized according to data destination.

3.1.1 MAR Control Field

As shown in Figure 3-1, the MAR control field occupies bits 11 and 12 of each KMC11 Move class microinstruction. For each Move microinstruction, the two MAR control bits implement one of four functions. These functions and the associated arguments (in parentheses) to the KMC11 macroassembler are as follows:

1. Leave contents of MAR unchanged (no argument needed).

2. Load MAR bits 0 to 7 (LDMAR).

3. Load MAR bits 8 and 9 (LDMAPG).

4. Increment MAR (INCMAR).

Syntactically within a Move class macroinstruction, a MAR control field argument will generally be the last argument in a microinstruction mnemonic. In a given Move class macroinstruction, the absence of an MAR control field mnemonic causes the KMC11 macroassembler to assemble zeros into the MAR control field so that the MAR remains unchanged during execution of that instruction.

When the eight low-order bits of the MAR are to be loaded, the content of the destination bus is moved to corresponding MAR bit positions. When the two high-order bit positions are loaded, bits 0 and 1 of the destination bus are moved to bits 8 and 9 of the MAR.

In actual practice, the four states of MAR bits 8 and 9 designate one of four 256-word pages in data memory. Bits 0 to 7 identify the specific word addressed (page offset) within the designated page.

Finally, the MAR can be incremented by one through execution of any Move class microinstruction. In this operation, any other functions performed by the executed microinstruction have no effect on the MAR. This incrementing action is performed during the last phase of the microinstruction cycle, after the register-to-register transfer is complete. When performing the increment MAR function, the microprogram can consider the data memory as a 1K set of contiguous and cyclical memory locations. (MAR address 1777+1 →MAR address 0.)

3.1.2 Move Class Microinstructions: Formats and Functions

The detailed information for Move class microinstructions is organized by data destination. The detailed information presented for each microinstruction in the Move class includes the following:

1. the microinstruction name,

2. the macroassembler mnemonic,

3. the format,

4. a detailed functional description, and

5. a description of microinstruction arguments.

As previously indicated, certain Move class instructions can perform an arithmetic or logical operation on the source operands; the result is stored in the specified destination register. Figure 2-1 shows that these operations are performed by the KMC11 Arithmetic/Logic Unit (ALU) and that this unit accepts two input operands. On the B-side of the ALU, the operand is always from the KMC11 source bus; on the A-side, the operand is always from the scratch pad. The KMC11 ALU performs a total of 16 arithmetic and logical functions (Table 3-1).

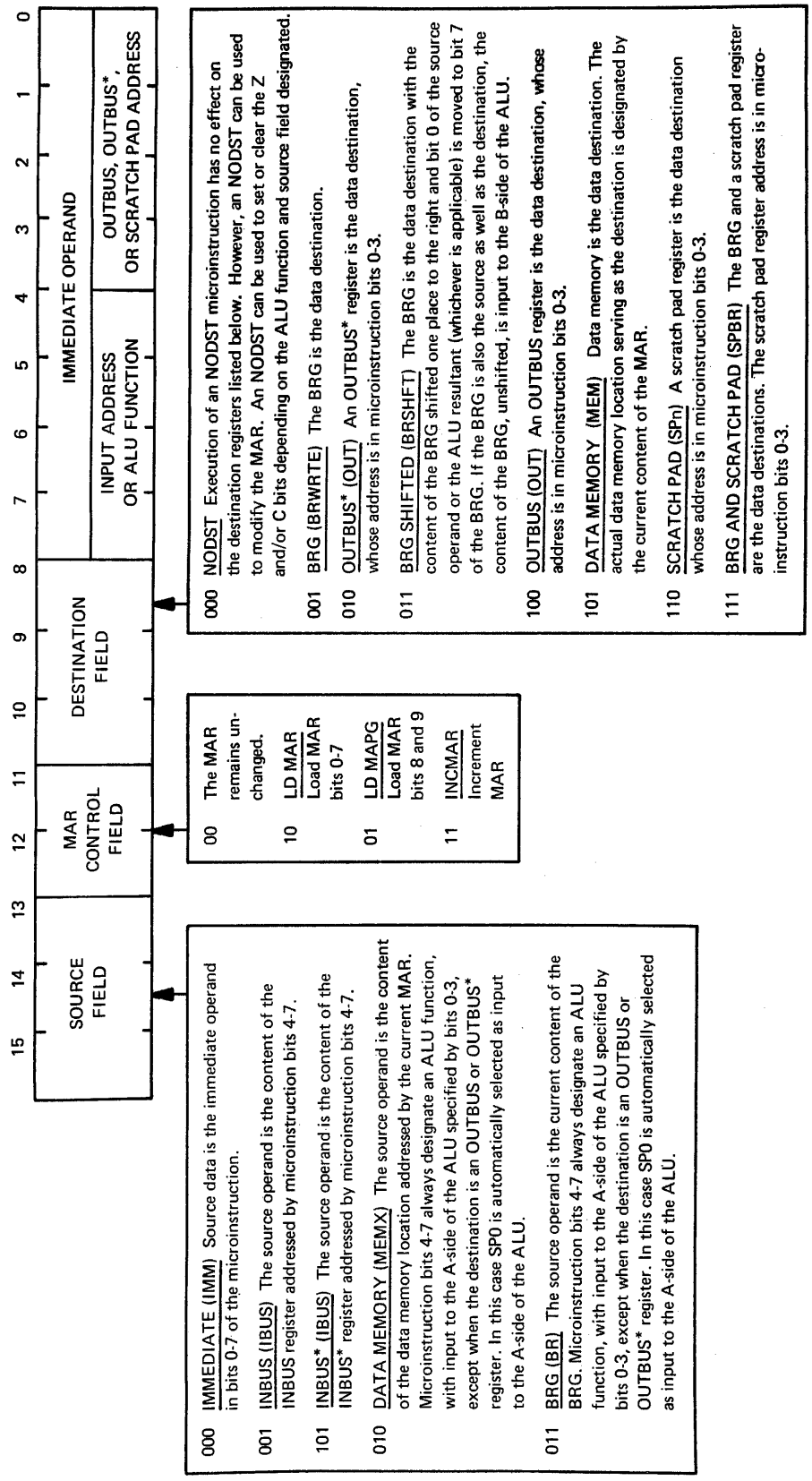


Figure 3-1 Move Class Microinstructions

Table 3-1
Arithmetic/Logic Unit Functions

ALU Function Mnemonic	ALU Function Field Binary Equivalent	ALU Functions Performed	Function Notation	C-Bit Affected?
ADD	0000	Add A and B	A+B	Yes
ADDC	0001	Add A and B with carry	A+B+C	Yes
SUBC	0010	Subtract B from A with borrow	A-B-~C	Yes
INCA	0011	Increment A	A+1	Yes
APLUSC	0100	A plus carry	A+C	Yes
TWOA	0101	A plus A	A+A	Yes
TWOAC	0110	A plus A plus carry	A+A+C	Yes
DECA	0111	Decrement A	A-1	Yes
SELA	1000	Select A side of ALU	A	No
SELB	1001	Select B side of ALU	B	No
AORNB	1010	A OR NOT B	AV~B	No
AANDB	1011	A AND B	AΛB	No
AORB	1100	A OR B	AVB	No
AXORB	1101	A exclusive OR B	A¥B	No
SUB	1110	Subtract B from A (two's complement)	A-B	Yes
SUBOC	1111	Subtract B from A (one's complement)	A-B-1	Yes

NOTE

The ALU function SUBC is actually a subtract with complement of the C-bit to allow double-precision subtraction.

Depending on the function performed, execution of a Move class microinstruction involving an ALU function can affect the state of the C-bit. However, all Move class microinstructions can affect the state of the Z-bit. The specific effects on these bits by Move class microinstruction execution are as follows:

1. C-bit:

   a. The C-bit is set to one when the sum produced by any ALU add function results in a carry, and cleared to zero when a carry is not produced.

b. The C-bit is cleared to zero when the difference produced by any ALU subtract function results in a borrow or a sign change and set to one when a borrow or a sign change is not produced.

c. The C-bit is unaffected when a logical ALU function is performed.

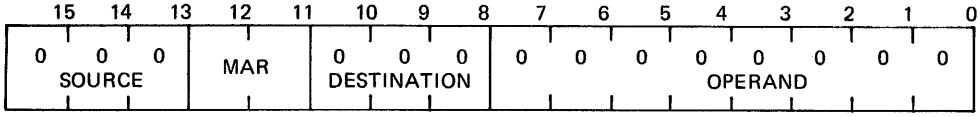
2. The Z-bit is set to one when the output of the ALU, through execution of any Move microinstruction, results in a value of all ones (377, octal). If the result is not equal to 377(octal) the Z-bit is set to zero.

The state of the C-bit is used to determine the result of a compare transaction. For the ALU function mnemonic SUB (binary code 1110), subtract B from A (two's complement); the C-bit will be set to one when A is greater than or equal to B and will be cleared to zero when A is less than B. For the ALU function mnemonic SUBOC (binary code 1111), subtract B from A (one's complement); the C-bit will be set to one when A is greater than B and will be cleared to zero when A is less than or equal to B.

The state of the Z-bit is used to determine the equality of two quantities followed by a branch on Z-bit set to one. For the ALU function SUBOC (binary code 1111), subtract B from A (one's complement); the Z-bit will be set when A is equal to B and will be cleared to zero when A is not equal to B.

3.1.2.1 Destination NODST

NO DESTINATION
(NODST)



This instruction, with the MAR control field set to zero, performs a null operation. A microinstruction cycle is executed, and no internal registers are changed but the Z-bit is cleared. In this configuration the NODST Immediate microinstruction can be used to implement program delays of any length.

However, this instruction can also be used to increment or load the MAR without affecting the contents of any other internal register using the arguments INCMAR, LDMAR, and LDMAPG. For example, the MAR can be incremented by

```
NODST INCMAR
```

the lower eight bits loaded by

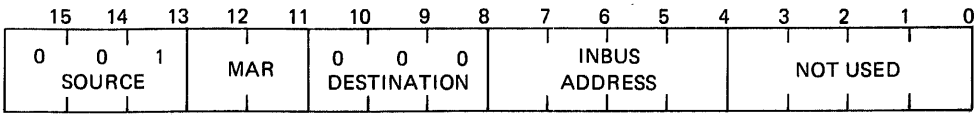
```
NODST IMM,opr,LDMAR
```

and the upper two bits by

```
NODST IMM,opr,LDMAPG
```

The argument for LDMAR, namely opr, would be a value in the range 0 to 377 (octal) and contained in bits 0 to 7 of the microinstruction. On the other hand, the argument for LDMAPG could be any value up to 377 (octal) as long as bit positions 0 and 1 of the microinstruction argument correspond to the desired values of MAR bits 8 and 9.

TEST INBUS
(NODST IBUS, adri)

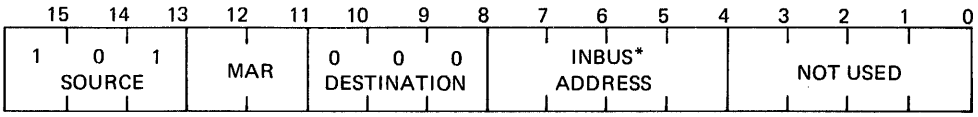


This microinstruction permits a microprogram to determine if the value of an INBUS register is equal to or not equal to 377(octal). Execution of this microinstruction may be followed by a Branch class microinstruction that tests and branches on the state of the ALU Z-bit (Section 3.2). If the Z-bit is equal to 1, the contents of the selected INBUS register have an octal value of 377. If the Z-bit is equal to 0, the register has an octal value not equal to 377. The contents of the addressed INBUS register are not changed by this test.

The symbolic addresses for INBUS registers, as specified by the argument adri, are listed in Table 3-2 along with the internal physical address of the corresponding register.

For detailed information on the first eight INBUS registers, refer to Section 2.3.2 and Figure 2-3. The eight INBUS registers 10 through 17 are assigned to support a high-speed peripheral device such as the DMC11-DA Synchronous Line Unit. The titles and document numbers identifying the maintenance and user manuals for this device are listed in Section 1.5.

TEST INBUS*
(NODST IBUS, adri)



This microinstruction permits a microprogram to determine if the value of an INBUS* register is equal to or not equal to 377 (octal). Execution of this microinstruction may be followed by a Branch class microinstruction that tests and branches on the state of the ALU Z-bit (Section 3.2). If the Z-bit is equal to 1, the contents of the selected INBUS* register have an exact octal value of 377. If the Z-bit is equal to 0, the contents of the selected register have an octal value not equal to 377. The contents of the addressed INBUS* register are not affected by this test.

The symbolic addresses for the INBUS* registers as specified by the argument adri are listed in Table 3-3 along with the internal physical address of the corresponding register.

Table 3-2
INBUS Register Symbolic Addresses

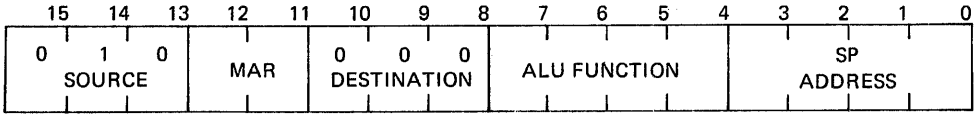
Symbol	Register Function	Octal Address
INDAT1	In-NPR data low byte	0
INDAT2	In-NPR data high byte	1
IODAT1	Out-NPR data low byte	2
IODAT2	Out-NPR data high byte	3
IIBA1	In-NPR UNIBUS address low byte	4
IIBA2	In-NPR UNIBUS address high byte	5
IOBA1	Out-NPR UNIBUS address low byte	6
IOBA2	Out-NPR UNIBUS address high byte	7
XREG0	User-specified	10
XREG1	User-specified	11
XREG2	User-specified	12
XREG3	User-specified	13
XREG4	User-specified	14
XREG5	User-specified	15
XREG6	User-specified	16
XREG7	User-specified	17

Table 3-3
INBUS* Register Symbolic Addresses

Symbol	Register Function	Octal Address
INCON	CSR0	0
MAIN	CSR1 (Maintenance Register)	1
OCON	CSR2	2
LINENM	CSR3	3
PORT1	CSR4	4
PORT2	CSR5	5
PORT3	CSR6	6
PORT4	CSR7	7
NPR	NPR Register	10
UBBR	μPMISC Register	11

Note that the first eight symbolic INBUS* addresses comprise the KMC11 CSRs and that the symbols NPR and UBBR address the NPR and μPMISC registers. See Sections 2.3.1, 2.3.2, and 2.3.3 for detailed operational information on the CSRs, the NPR register, and the μPMISC register, respectively.

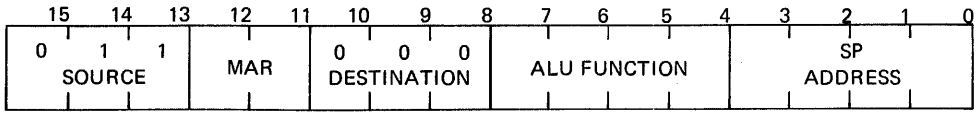
TEST MEMORY AND SCRATCH PAD
(NODST MEMX, func, SPn)



With this microinstruction, the microprogram can test the result of a designated ALU function (Table 3-1) on the contents of the data memory addressed by the current MAR and the contents of the addressed scratch pad location. If the result is equal to the octal value 377, the Z-bit is set to one. If the ALU function designated by the argument func implements an add or subtract operation, the C-bit is affected as described in Section 3.1.2. For example, the microinstruction NODST MEMX,ADD,SP3 adds the contents of the data memory location addressed by the current MAR to the contents of SP3, with the result capable of affecting the state of both the C-bit and Z-bit. The contents of a memory and a scratch pad location involved in this test are unaffected by the test.

The pertinent mnemonics comprising the arguments for func are listed in Table 3-1 along with the corresponding binary code and the arithmetic/logic function initiated by the mnemonic. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location to be involved in the microinstruction execution.

TEST BRG AND SCRATCH PAD
(NODST BR, func, SPn)

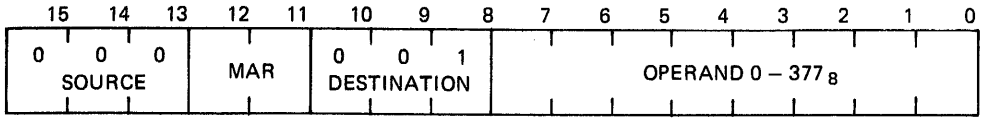


With this microinstruction, the microprogram can test the result of a designated ALU function (Table 3-1) on the contents of the Branch Register (BRG) and the addressed scratch pad location. If the result is equal to the octal value 377, the Z-bit is set to one; if the result is not equal to this value, the Z-bit is cleared to zero. If the ALU function designated by the argument for func implements an add or subtract function, the C-bit is affected as described in Section 3.1.2. The contents of the BRG and a scratch pad location involved in this test are unaffected by the test.

The pertinent mnemonics comprising the arguments for func are listed in Table 3-1 along with the corresponding binary code and the arithmetic/logic function implemented by the mnemonic. In the argument SPn, the lowercase n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location to be involved in the microinstruction execution.

3.1.2.2 Destination BRG

MOVE IMMEDIATE TO BRG
(BRWRTE IMM, opr)

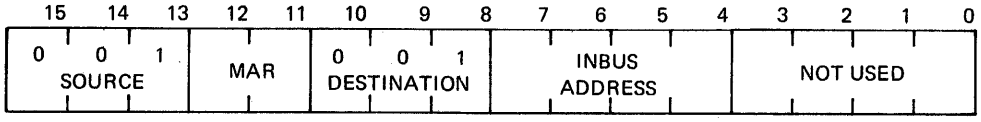


The microinstruction Move Immediate to BRG permits an 8-bit operand to be stored in the BRG. The actual value of the operand, as specified by the argument opr, can be an octal value in the range 0 to 377 or it can be the contents of any symbolic address designated by the symbol table for the pertinent microprogram.

NOTE

If the operand is equal to 377 (octal), the Z-bit is set to one.

MOVE INBUS TO BRG
(BRWRTE IBUS, adri)

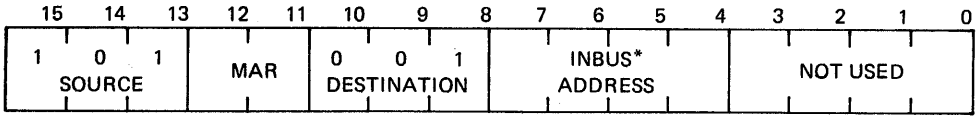


The microinstruction Move INBUS to BRG moves the contents of a designated INBUS register to the BRG with the contents of the source INBUS register remaining unchanged. The INBUS symbolic addresses represented by the argument adri are listed in Table 3-3.

NOTE

If the contents of the INBUS register addressed by the microinstruction are equal to the 377 (octal), the Z-bit is set to one.

MOVE INBUS* TO BRG
(BRWRTE IBUS, adri)

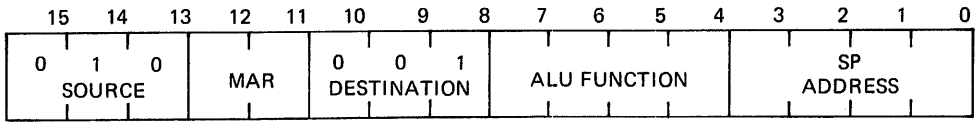


The microinstruction Move INBUS* to BRG moves the contents of a designated INBUS* register to the BRG, with the contents of the source INBUS* register remaining unchanged. The INBUS* symbolic addresses represented by the argument adri are listed in Table 3-3.

NOTE

If the contents of the INBUS* Register addressed by the microinstruction are equal to 377 (octal), the Z-bit is set to one.

MOVE RESULTS OF MEMORY AND SCRATCH PAD TO BRG
(BRWRTE MEMX, func, SPn)



This microinstruction can go beyond a simple source-to-destination transfer to permit the microprogram to perform an ALU function (Table 3-1) on the contents of the data memory location addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn. The result of the designated ALU function on these values is then stored in the BRG. An example of such a microinstruction is as follows:

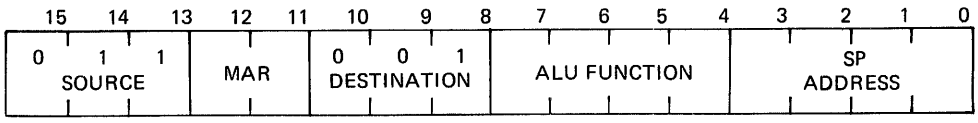
```
BRWRTE MEMX,SUB,SP5
```

The contents of the specific memory and scratch pad location involved remain unchanged. Similarly, a direct source-to-destination transfer of data from the memory location addressed by the current MAR to the BRG can be implemented with the ALU function Select B (Table 3-1). This microinstruction would take the following form:

```
BRWRTE MEMX,SELB
```

The mnemonics for the argument func are listed in Table 3-1 along with the arithmetic/logic function implemented by each mnemonic. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this microinstruction.

MOVE RESULTS OF BRG AND SCRATCH PAD TO BRG
(BRWRTE BR, func, SPn)



As with the previous Move class microinstruction, this microinstruction also goes beyond a simple source-to-destination transfer to perform a designated ALU function on the contents of the BRG and the contents of the addressed scratch pad location with the result being stored in the BRG. The result produced after executing this instruction is stored in the BRG, and the contents of the addressed scratch pad location remain unchanged.

A direct source-to-destination transfer of data from the scratch pad to the BRG can be implemented by using the ALU function Select A

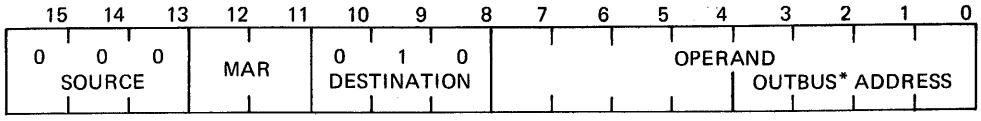
(SELA, Table 3-1). This microinstruction would take the following form:

```
BRWRTE SELA,SPn
```

The pertinent mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this microinstruction.

3.1.2.3 **Destination OUTBUS*** - Microinstructions with the destination OUTBUS* and OUTBUS share the same destination name (specifically OUT). The KMC11 macroassembler distinguishes between these two Move instruction sets by analyzing the names of the OUTBUS* and OUTBUS registers.

MOVE IMMEDIATE TO OUTBUS*
(OUT IMM, opr, adro)



The microinstruction Move Immediate to OUTBUS* permits the microprogram to move an 8-bit operand directly to the OUTBUS* register addressed by the four low-order bits of the immediate operand. To understand this instruction, consider the arguments opr and adro. The argument opr is an octal number or a symbol having a value in the range 0 to 377. Conversely, the argument adro is one of the symbolic addresses for an OUTBUS* register as listed in Table 3-4. Consequently, the four low-order bits of the operand opr must have a value equal to the value of the physical address corresponding to the pertinent OUTBUS* symbolic address.

For example, if, the argument opr in a Move Immediate to OUTBUS* has the octal value 364, then the argument adro must be the OUTBUS* register symbolic address OPORT1 (Table 3-4). This microinstruction would take the following form:

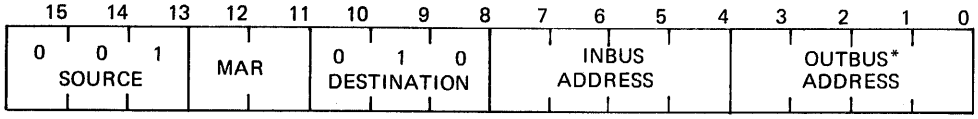
```
OUT IMM,364,OPORT1
```

If a conflict occurs during assembly between the value of the four low-order bits of an operand and the value of adro (the corresponding octal code for an OUTBUS* register symbolic address), an assembly error is posted (Chapter 4).

Table 3-4
OUTBUS* Register Symbolic Addresses

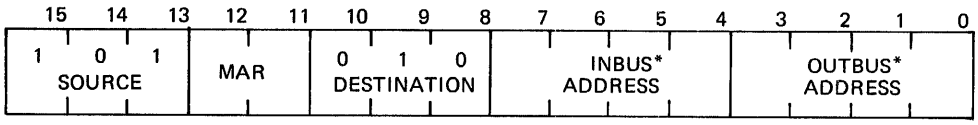
Symbol	Register Function	Octal Address
OICOM	CSR0	0
OMAIN	CSR1 (Maintenance Register)	1
OOCON	CSR2	2
OLINEN	CSR3	3
OPORT1	CSR4	4
OPORT2	CSR5	5
OPORT3	CSR6	6
OPORT4	CSR7	7
ONPR	NPR Register	10
OBR	μPMISC Register	11

MOVE INBUS TO OUTBUS*
(OUT IBUS, adri, adro)



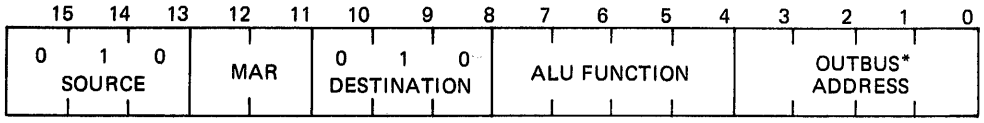
This microinstruction provides the microprogram with the ability to transfer data directly from an INBUS register to an OUTBUS* register, with the contents of the source INBUS register remaining unchanged. The argument adri is always one of the INBUS register symbolic addresses listed in Table 3-2. Similarly, the argument adro is one of the OUTBUS* symbolic addresses listed in Table 3-4.

MOVE INBUS* TO OUTBUS*
(OUT IBUS, adri, adro)



This microinstruction provides the microprogram with the ability to transfer data directly from an INBUS* register to an OUTBUS* register with the contents of the source INBUS* register remaining unchanged. The argument adri is one of the INBUS* register symbolic addresses listed in Table 3-3. Similarly, the argument adro is always one of the OUTBUS* symbolic registers listed in Table 3-4.

MOVE RESULTS OF MEMORY AND SP0 TO OUTBUS*
(OUT MEMX, func, adro)



This microinstruction permits a microprogram to perform a designated ALU function on two operands, one being the contents of the data memory location addressed by current MAR and the other being the contents of scratch pad location zero. The result of the designated ALU function on these operands is then stored in the OUTBUS* register addressed by the instruction. A direct source-to-destination transfer of data from data memory to the addressed OUTBUS* register can be implemented by using the ALU function Select B (SELB, Table 3-1). This microinstruction would take the following form:

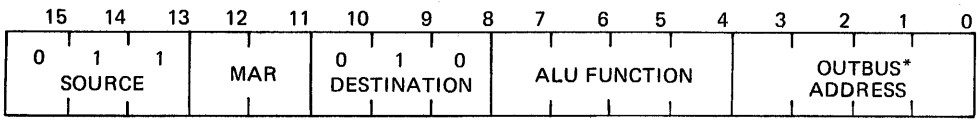
```
OUT MEMX,SELB,adro
```

Similarly, a direct transfer of the contents of scratch pad location zero (SP0) can be implemented using the ALU function Select A (SELA). This microinstruction would take the following form:

```
OUT SELA,adro
```

The complete set of arithmetic/logic functions and corresponding mnemonics for the argument func are listed in Table 3-1. Symbolic addresses for the argument adro, which are the addresses of the ten OUTBUS* registers, are listed in Table 3-4.

MOVE RESULTS OF BRG AND SP0 TO OUTBUS*
(OUT BR, func, adro)



With this microinstruction, a microprogram can perform a designated ALU function on two operands; one is the contents of the BRG and the other is the contents of scratch pad location zero (SP0). The result produced by the designated ALU function is then stored in the addressed OUTBUS* register. The contents of the BRG and SP0 remain unchanged.

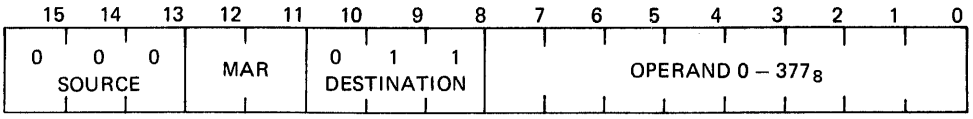
A direct source-to-destination transfer between the BRG and a designated OUTBUS* register can be implemented by using the ALU function Select B (SELB, Table 3-1). This microinstruction would be in the following form:

```
OUT BR,SELB,adro
```

The complete set of arithmetic/logic functions and corresponding mnemonics for the argument func is listed in Table 3-1. Symbolic addresses for the argument adro, which are the addresses for the ten OUTBUS* registers, are listed in Table 3-4.

3.1.2.4 Destination BRG Right-Shifted

RIGHT SHIFT BRG ONE PLACE AND MOVE OPERAND BIT 0 TO BRG BIT 7
(BRSHFT IMM, opr)



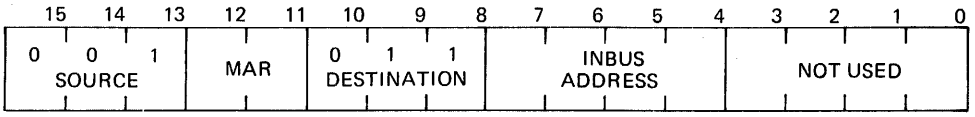
Execution of this microinstruction is performed functionally in two steps. First the contents of the BRG are shifted right one place, and second, bit 0 of the immediate operand is moved to bit position 7 of the BRG. Bit 0 of the BRG, as it was prior to the right-shift operation, is lost. A typical use of this instruction is to shift zeros into the high-order end of the BRG. Such an instruction would take the following form:

```
BRSHFT IMM,0
```

or

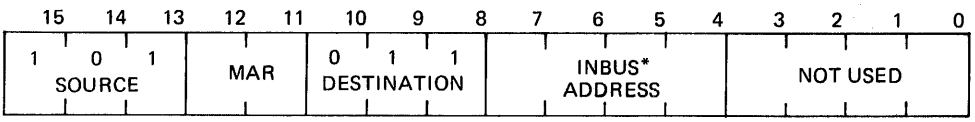
```
BRSHFT
```

RIGHT SHIFT BRG ONE PLACE AND MOVE INBUS BIT 0 TO BRG BIT 7
(BRSHFT IBUS, adri)



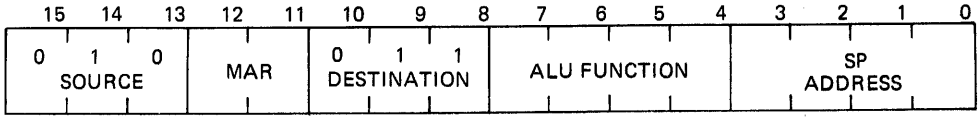
Execution of this microinstruction results in the contents of the BRG being shifted one place to the right. Bit 0 of the INBUS register addressed by the symbolic address designated by the argument adri is then moved to bit position 7 of the BRG. Bit 0 of the BRG as it was prior to the right-shift operation is lost. The contents of the addressed INBUS register remain unchanged. The INBUS symbolic addresses for the argument adri are listed in Table 3-2.

RIGHT SHIFT BRG ONE PLACE AND MOVE INBUS* BIT 0 TO BRG BIT 7
(BRSHFT IBUS, adri)



Execution of this microinstruction shifts the contents of the BRG one place to the right. Bit 0 of the INBUS* register addressed by the symbolic address designated by the argument adri is then moved to bit position 7 of the BRG. Bit 0 of the BRG as it was prior to the right shift is lost. The contents of the addressed INBUS register remain unchanged. The INBUS* symbolic addresses for the argument ardi are listed in Table 3-3.

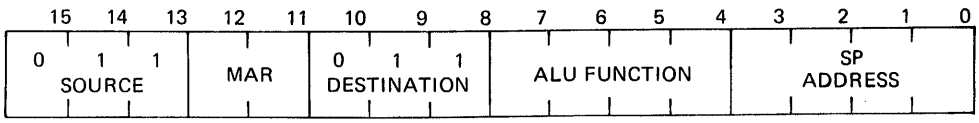
RIGHT SHIFT BRG ONE PLACE AND MOVE ALU OUTPUT BIT 0 (MEMORY AND SPn) TO BRG BIT 7
(BRSHFT MEMX, func, SPn)



This microinstruction functionally executes in three steps. First, the BRG is right-shifted one place with bit 0 of the BRG being lost. Next, the contents of the data memory location addressed by the current MAR and the contents of the addressed scratch pad location are operated on according to the designated ALU function. Finally, bit 0 of the resulting ALU output is moved to bit position 7 of the BRG. The contents of the addressed data memory location and scratch pad location remain unchanged.

Bit 0 of the data memory location addressed by the current MAR can be moved to bit position 7 of the BRG by using the ALU function Select B (SELB, Table 3-1). Similarly, bit 0 of the scratch pad register addressed by the argument SPn (where n is the octal address of the designated register) can be moved to bit 7 of the BRG by using the ALU function Select A (SELA). The symbols for the argument func are listed in Table 3-1 along with the function performed and the corresponding binary code.

RIGHT SHIFT BRG ONE PLACE AND MOVE ALU OUTPUT BIT 0 (BRG AND SPn) TO BRG BIT 7
(BRSHFT BR, func, SPn)



In this microinstruction, the BRG is both the source and the destination of data. As a consequence, the first step in the execution of this instruction involves placing the unshifted contents of the BRG onto the source bus (Figure 2-1) for input to the B-side of the ALU. Following this action, the contents of the BRG are right-shifted one place. The designated ALU function is then performed on the unshifted BRG and the addressed scratch pad location. Bit 0 of the resulting ALU output is moved to bit position 7 of the BRG.

One of the major functions of this microinstruction is to provide a single-bit right rotate of the BRG, which is implemented when the ALU function Select B (SELB, Table 3-1) is used. Since the BRG is the source, its contents are placed on the source bus unshifted for input to the ALU. With the BRG also the destination, it is then right-shifted one place. Such an instruction would take the following form:

```
BRSHFT BR,SELB
```

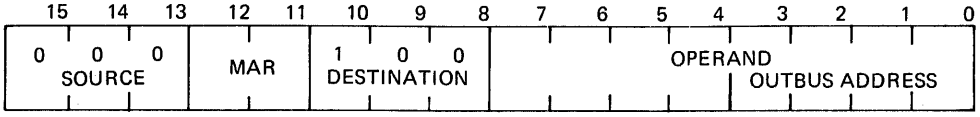
With SELB the designated ALU function, the B-side is selected and bit 0 of the unshifted BRG is moved to bit position 7 of the BRG. The result is a one-instruction one-bit right rotate of the BRG.

The mnemonics for the argument func are listed in Table 3-1 along with the arithmetic/logic function implemented and the corresponding binary code. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of scratch pad location involved in the execution of this microinstruction.

3.1.2.5 Destination OUTBUS

Microinstructions with the destination OUTBUS and OUTBUS* share the same destination name (specifically OUT). The KMC11 macroassembler distinguishes between these two Move instruction sets by analyzing the names of the OUTBUS and OUTBUS* registers.

MOVE IMMEDIATE TO OUTBUS
(OUT IMM, opr, adro)



The microinstruction Move Immediate to OUTBUS permits the microprogram to move the 8-bit operand directly to the OUTBUS register addressed by the four low-order bits of the immediate operand. To understand this instruction, consider the arguments opr and adro. The argument opr is an octal value in the range 0 to 377. Conversely, the argument adro is of the symbolic addresses of an OUTBUS register as listed in Table 3-5. Consequently, the four low-order bits of the operand must have a value equal to the physical address corresponding to the pertinent OUTBUS symbolic address.

For example, if the argument opr has the octal value 305 in a Move Immediate to OUTBUS, then the argument for adro must be the OUTBUS register symbolic address IBA2. This microinstruction would take the following form:

```
OUT IMM,305,IBA2
```

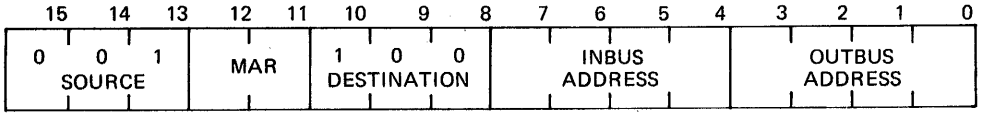
If a conflict occurs between the value of the four low-order bits of an operand and the corresponding octal code for an OUTBUS register symbolic address, then an assembly error will be posted (Chapter 4).

For detailed information on the first eight OUTBUS registers, refer to Section 2.3.2 and Figure 2-3. The eight OUTBUS registers 10 through 17 are assigned to support a high-speed peripheral device such as the DMC11-DA Synchronous Line Unit. The title and document numbers identifying the maintenance and user manuals for this device are listed in Section 1.5.

Table 3-5
OUTBUS Register Symbolic Addresses

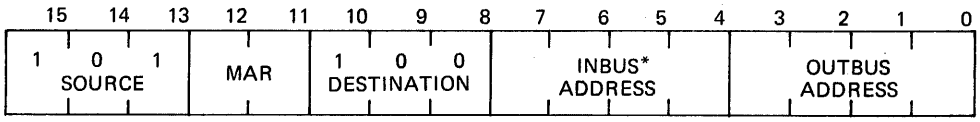
Symbolic Address	Register Function	Octal Address
OIDAT1	In-NPR data low byte	0
OIDAT2	In-NPR data high byte	1
OUTDA1	Out-NPR data low byte	2
OUTDA2	Out-NPR data high byte	3
IBA1	In-NPR UNIBUS address low byte	4
IBA2	In-NPR UNIBUS address high byte	5
OBA1	Out-NPR UNIBUS address low byte	6
OBA2	Out-NPR UNIBUS address high byte	7
OXREG0	User-specified	10
OXREG1	User-specified	11
OXREG2	User-specified	12
OXREG3	User-specified	13
OXREG4	User-specified	14
OXREG5	User-specified	15
OXREG6	User-specified	16
OXREG7	User-specified	17

MOVE INBUS TO OUTBUS
(OUT IBUS, adri, adro)



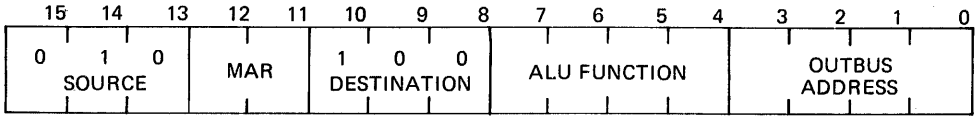
This microinstruction provides the microprogram with the ability to transfer directly from an INBUS register to an OUTBUS register. The argument adri is one of the INBUS register symbolic addresses listed in Table 3-2. Similarly, the argument adro is one of the OUTBUS symbolic addresses listed in Table 3-5. The contents of the addressed INBUS register are not affected by execution of this instruction.

MOVE INBUS* TO OUTBUS
(OUT IBUS, adri, adro)



This microinstruction provides the microprogram with the ability to transfer data directly from an INBUS* register to an OUTBUS register. The argument adri is one of the INBUS* register symbolic addresses listed in Table 3-3. Similarly, the argument adro is one of the OUTBUS symbolic registers listed in Table 3-5. The contents of the addressed INBUS* register are not affected by execution of this instruction.

MOVE RESULTS OF MEMORY AND SP0 TO OUTBUS
(OUT MEMX, func, adro)



This microinstruction permits a microprogram to perform a designated ALU function on two operands, one being the contents of the data memory location addressed by the current MAR, and the second being the contents of scratch pad location zero. The result produced by the designated ALU function is then stored in the OUTBUS register addressed by the instruction.

A direct source-to-destination transfer of data from data memory to the addressed OUTBUS register can be implemented by using the ALU function Select B (SELB, Table 3-1). This microinstruction would take the following form:

```
OUTPUT MEMX,SELB,adro
```

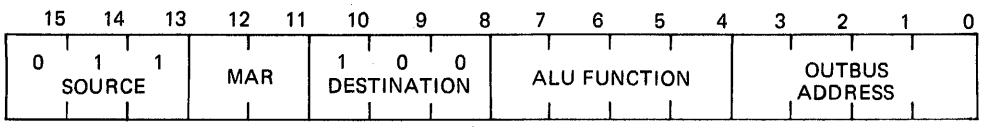
Similarly, a direct transfer of the contents of scratch pad location zero (SP0) can be implemented using the ALU function Select A (SELA). This microinstruction would take the following form:

```
OUTPUT SELA,adro
```

The contents of the addressed data memory location and SP0 are not affected by execution of this instruction.

The complete set of arithmetic/logic functions and corresponding mnemonics for the argument func is listed in Table 3-1. Symbolic addresses for the argument adro, which are the addresses of the OUTBUS registers, are listed in Table 3-5.

MOVE RESULTS OF BRG AND SP0 TO OUTBUS
(OUT BR, func, adro)



With this microinstruction, a microprogram can perform a designated ALU function on two operands; one being the contents of the BRG and the other the contents of scratch pad location zero (SP0). The result produced by the designated ALU function is then stored in the addressed OUTBUS register.

A direct source-to-destination transfer between the BRG and the designated OUTBUS register can be implemented by using the ALU function Select B (SELB, Table 3-1) this microinstruction would take the following form:

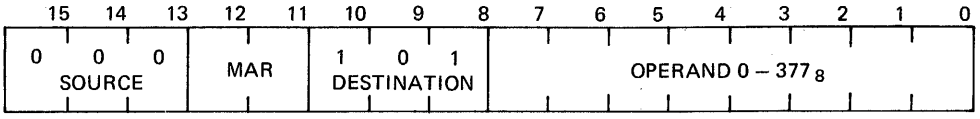
```
OUTPUT BR,SELB,adro
```

The complete set of arithmetic/logic functions and corresponding mnemonics is listed in Table 3-1. Symbolic addresses for the argument adro, which are the addresses of the OUTBUS registers, are listed in Table 3-5.

The contents of the BRG and SP0 are not affected by execution of this instruction.

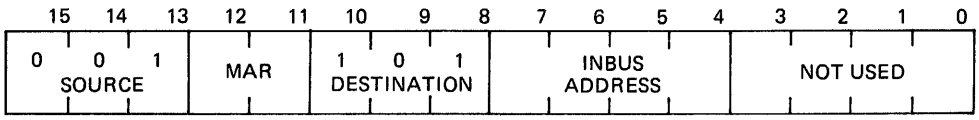
3.1.2.6 **Destination Data Memory** - As previously described, the KMC11 data memory is addressed by the Memory Address Register (MAR), which in turn is accessed from the destination bus. Since the destination for all five microinstructions in this group is data memory, the actual data memory location comprising the data destination is that location addressed by the current MAR. For all the microinstructions in this group, when the MAR control field is used, the address in the MAR will be the value prior to execution of the current instruction. The function initiated by the MAR control field in a pertinent microinstruction is performed after the result produced by execution of the instruction is stored in data memory.

MOVE IMMEDIATE TO MEMORY
(MEM IMM, opr)



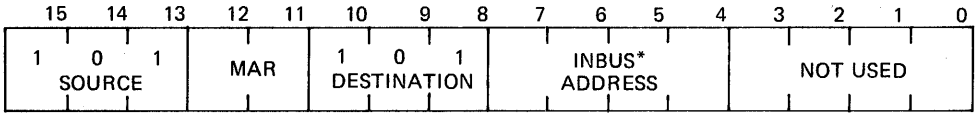
The microinstruction Move Immediate to Data Memory permits the microprogram to move an 8-bit operand directly to the data memory location addressed by the current content of the MAR. The argument opr can be a number or a symbol having a value in the range 0 to 377 (octal).

MOVE INBUS TO MEMORY
(MEM IBUS, adri)



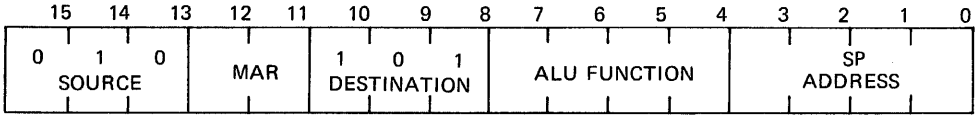
Execution of this microinstruction enables the microprogram to transfer data directly from an addressed INBUS register to the data memory location addressed by the current MAR. The argument adri is one of the INBUS register symbolic addresses listed in Table 3-2. The contents of the addressed INBUS register are unaffected by execution of this instruction.

MOVE INBUS* TO MEMORY
(MEM IBUS, adri)



Execution of this instruction enables the microprogram to transfer data directly from an addressed INBUS* register to the data memory location addressed by the current MAR. The argument adri is one of the INBUS* register symbolic addresses listed in Table 3-3. The contents of the addressed INBUS* register are unaffected by execution of this instruction.

MOVE RESULTS OF MEMORY AND SP TO MEMORY
(MEM MEMX, func, SPn)



This microinstruction permits a microprogram to perform an ALU function on two operands, one being the contents of the data memory location addressed by the current MAR, and the second being the contents of the scratch pad location addressed by the argument SPn. The result produced by the designated ALU function on these two operands is then stored in the same data memory location from which the memory operand was fetched with the prior content of that location being lost. Such an instruction would take the following form:

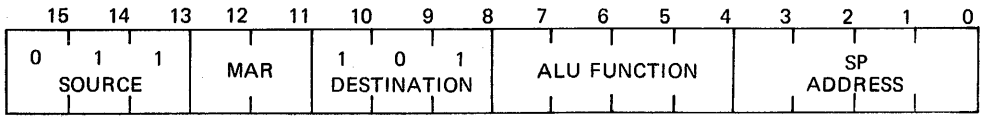
```
MEM MEMX,ADD,SP5
```

The contents of the addressed scratch pad location are not affected by execution of this instruction. A direct transfer of data from the addressed scratch pad location to the memory location addressed by the current MAR can be performed using the ALU function Select A (SELA, Table 3-1). This microinstruction would take the following form:

```
MEM SELA,SPn
```

The complete set of arithmetic/logic functions and the corresponding mnemonics for the argument func is listed in Table 3-1. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of scratch pad locations involved in the execution of this microinstruction.

MOVE RESULTS OF BRG AND SP TO MEMORY
(MEM BR, func, SPn)



As with the previous microinstruction, this microinstruction permits the microprogram to perform a designated ALU function on two operands. One operand is the current contents of the BRG, and the second the contents of the addressed scratch pad location.

The result produced by the designated ALU function on these two operands is then stored in the data memory location addressed by the current MAR. The contents of the BRG and the addressed scratch pad location are unaffected by execution of this microinstruction.

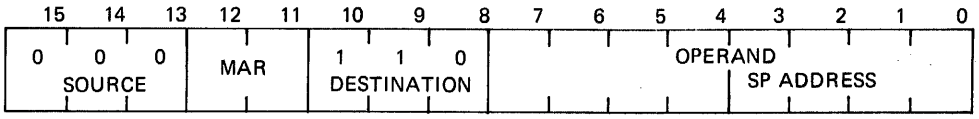
A direct transfer of data from the BRG to the data memory location addressed by the current MAR can be performed by using the ALU function Select B (SELB, Table 3-1). This microinstruction would take the following form:

```
MEM BR,SELB
```

The complete set of arithmetic/logic functions and the corresponding mnemonics for the argument func is listed in Table 3-1. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this instruction.

3.1.2.7 Destination Scratch Pad

MOVE IMMEDIATE TO SCRATCH PAD
(SP IMM, opr, SPn)

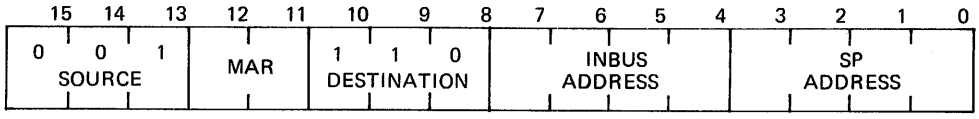


The microinstruction Move Immediate to Scratch Pad permits the microprogram to move an eight-bit operand directly to the scratch pad location addressed by the low-order four bits of the immediate operand. To understand how this microinstruction decodes to address the destination scratch pad location consider the arguments opr and SPn. The argument opr can be an octal number or a symbol having a value in the range 0 to 377, and the argument SPn must be an octal number in the range 0 to 17. If, for example, the argument for opr has an octal value of 377, then the argument SPn must have the octal value 17 to address the last register in the scratch pad as the data destination. Such a microinstruction would take the following form:

```
SP IMM,377,SP17
```

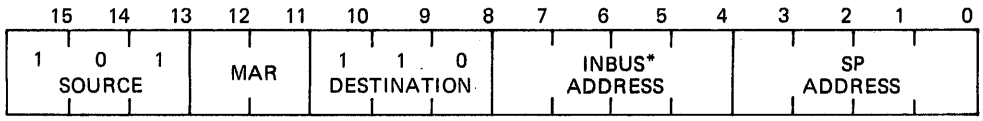
If a conflict occurs during assembly between the value of the four low-order bits of an operand opr and the value of SPn, then an assembly error will be posted (Chapter 4).

MOVE INBUS TO SP
(SP IBUS, adri, SPn)



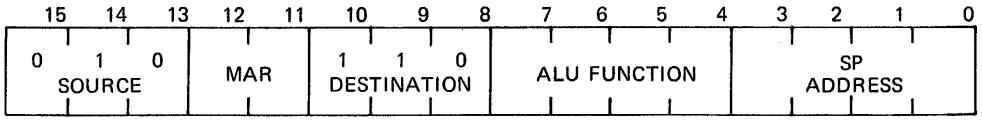
This microinstruction provides the microprogram with the ability to transfer data directly from an addressed INBUS register to an addressed scratch pad location. The argument adri is one of the INBUS register symbolic addresses listed in Table 3-2. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location designated the data destination. The contents of the INBUS register addressed as the data source are not affected by execution of this microinstruction.

MOVE INBUS* TO SP
(SP IBUS, adri, SPn)



This microinstruction provides the microprogram with the ability to transfer data directly from an addressed INBUS* register to an addressed scratch pad location. The argument adri is one of the INBUS* register symbolic addresses listed in Table 3-3. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location designated as the destination. The contents of the source INBUS* register are not affected by execution of this microinstruction.

MOVE RESULTS OF MEMORY AND SPn TO SPn
(SP MEMX, func, SPn)



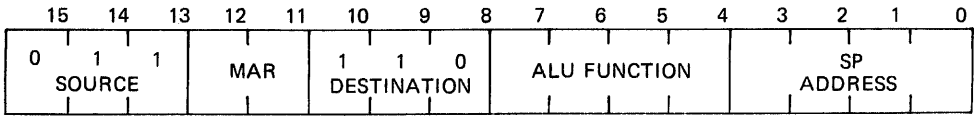
This microinstruction permits the microprogram to perform a designated ALU function on two operands: one is the contents of the data memory location addressed by the current MAR, and the other the contents of the scratch pad location addressed by the argument SPn. The result of the designated ALU function on these operands is then stored in the scratch pad location addressed by the instruction. The contents of the addressed data memory location are unchanged by execution of this instruction.

If required, a direct source-to-destination transfer between the data memory location addressed by the current MAR and the scratch pad location addressed by SPn can be performed by using the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

```
SP MEMX,SELB,SPn
```

The mnemonics for the argument func are listed in Table 3-1 along with the arithmetic/logic function implemented by each mnemonic. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this microinstruction.

MOVE RESULTS OF BRG AND SPn TO SPn
(SP BR, func, SPn)



This microinstruction permits the microprogram to perform a designated ALU function on two operands: one is the current contents of the BRG, and the other the contents of the scratch pad location addressed by the argument SPn. The result of the designated ALU function on these operands is then stored in the scratch pad location addressed by the instruction. The current contents of the BRG are unchanged by execution of this instruction.

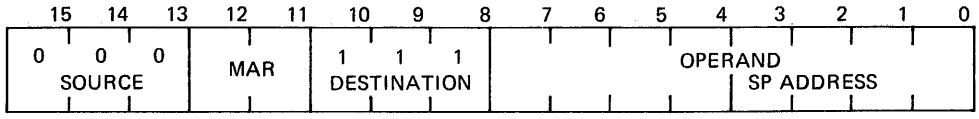
If required, a direct source-to-destination transfer between the BRG and the scratch pad location addressed by SPn can be performed by using the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

```
SP BR,SELB,SPn
```

The mnemonics for the argument func are listed in Table 3-1 along with the arithmetic/logic function implemented by each mnemonic. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this microinstruction.

3.1.2.8 Destination Scratch Pad and BRG

MOVE IMMEDIATE TO SP AND BRG
(SPBR IMM, opr, SPn)



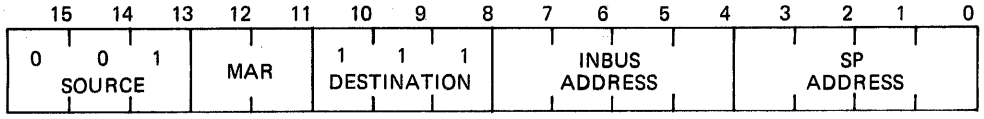
The microinstruction Move Immediate to Scratch Pad and BRG permits the microprogram to move an 8-bit operand simultaneously to the BRG and the scratch pad location addressed by the four low-order bits of the immediate operand. To understand how this microinstruction functions to address the destination scratch pad address, consider the arguments opr and SPn. The argument opr can be an octal number or a symbol having a value in the range 0 to 377. The argument SPn must be an octal number in the range 0 to 17.

If, for example, the argument opr is 230 (octal), then the argument SPn must be 10 (octal). Such a microinstruction would take the following form:

```
SPBR IMM,230,SP10
```

Executing this instruction moves the octal value 230 to the BRG and SP10. If a conflict occurs during assembly between the value of the four low-order bits of an operand opr and the value of SPn, then an assembly error will be posted (Chapter 4).

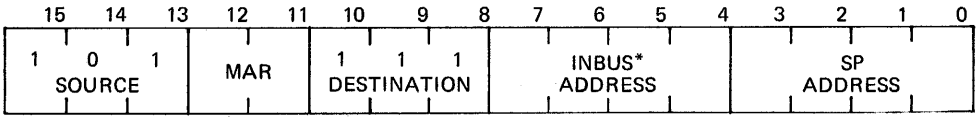
MOVE INBUS TO SP AND BRG
(SPBR IBUS, adri, SPn)



This microinstruction enables the microprogram to directly transfer data from an addressed INBUS register simultaneously to the BRG and an addressed scratch pad location.

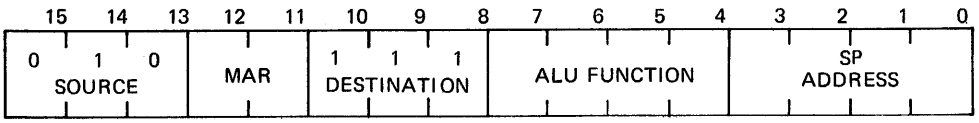
The argument adri is one of the INBUS register symbolic addresses listed in Table 3-2. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17. The contents of the INBUS register addressed as the data source are not affected by execution of this microinstruction.

MOVE INBUS* TO SP AND BRG
(SPBR IBUS, adri, SPn)



This microinstruction enables the microprogram to directly transfer data from an addressed INBUS* register simultaneously to the BRG and an addressed scratch pad location. The argument adri is one of the INBUS* register symbolic addresses listed in Table 3-3. For the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17. The contents of the INBUS* register addressed as the data source are not affected by execution of this microinstruction.

MOVE RESULTS OF MEMORY AND SP TO SP AND BRG
(SPBR MEMX, func, SPn)



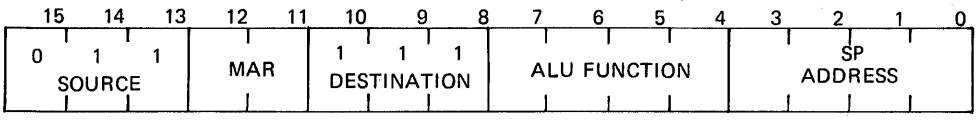
This microinstruction enables a microprogram to perform a designated ALU function on two source operands: one is the contents of the data memory location addressed by the current MAR and the second the contents of the scratch pad location addressed by the argument SPn. The result produced by the designated ALU function on these two operands is then simultaneously stored in the BRG and the scratch pad location addressed by the argument SPn. The contents of the data memory location accessed as one of the source operands is unaffected by execution of this microinstruction.

A direct source-to-destination transfer of the contents of the data memory location addressed by the current MAR to the BRG and the scratch pad location addressed by SPn can be implemented by using the ALU function Select B (SELB, Table 3-1). This microinstruction would take the following form:

```
SPBR MEMX,SELB,SPn
```

The complete set of arithmetic/logic functions and corresponding mnemonics for the argument func is listed in Table 3-1. For SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this instruction.

MOVE RESULTS OF BRG AND SP TO SP AND BRG
(SPBR BR, func, SPn)



This microinstruction enables a microprogram to perform a designated ALU function on two source operands: one is the contents of the BRG, and the second the contents of the scratch pad location addressed by the argument SPn. The result produced by the designated ALU function on these two operands is then simultaneously stored in the BRG and the scratch pad location addressed by SPn.

The complete set of arithmetic/logic functions and corresponding mnemonics for the argument func is listed in Table 3-1. For SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this instruction.

3.2 BRANCH CLASS MICROINSTRUCTIONS

Figure 3-2 summarizes KMC11 Branch class microinstructions. Excluding the permutations provided by the 16 ALU functions, there are 21 basic Branch class microinstructions. Figure 3-2 shows the derivation of all 21 permutations of Branch class microinstructions. In day-to-day microprogramming activity, the programmer can use Figure 3-2, in conjunction with Table 3-1, as a handy recall mechanism..

The narrative that follows describes the functions and use of the branch address field. The format and function for each basic Branch class microinstruction are organized according to the three states of the source field.

3.2.1 Branch Address Field

As shown in Figure 3-2, the eight low-order bits of a microinstruction branch address are either immediate or derived from the results of an ALU function (Table 3-1) on a data memory location or on the BRG and a scratch pad location. However, the two high-order bits required to form a Control RAM (CRAM) address must be contained as a 2-bit field (bits 11 and 12) of each Branch class microinstruction executed.

Figures 3-1 and 3-2 show that these bit positions coincide with the bit positions occupied by the MAR control field in a Move class microinstruction.

The four states of branch address bits 8 and 9 of the PC (bits 11 and 12 of the microinstruction) define page boundaries in the KMC11 CRAM. Page offset within each of four 256-word CRAM pages is defined by the actual contents of microinstruction bits 0 to 7 or the ALU designated results specified by source field and microinstruction bits 0 to 7. Within the microinstruction syntax implemented by the KMC11 macroassembler (Chapter 4), the argument for branch address bits 8 and 9 (Pn) is always the last argument in the pertinent assembly mnemonic and is one of the symbols listed in Table 3-6.

NOTE

Page boundaries are relevant only to Branch class microinstructions since the PC is always incremented as a 10-bit register with execution of a Move class microinstruction. However, in a given conditional branch microinstruction where the branch condition is not met, the PC is also incremented as a 10-bit register.

Table 3-6
Symbolic Values of the Argument OPARG1

Symbol for Argument Pn	Binary Equivalent of Page Bits 8 and 9	CRAM Page
P0	00	Page zero
P1	01	Page one
P2	10	Page two
P3	11	Page three

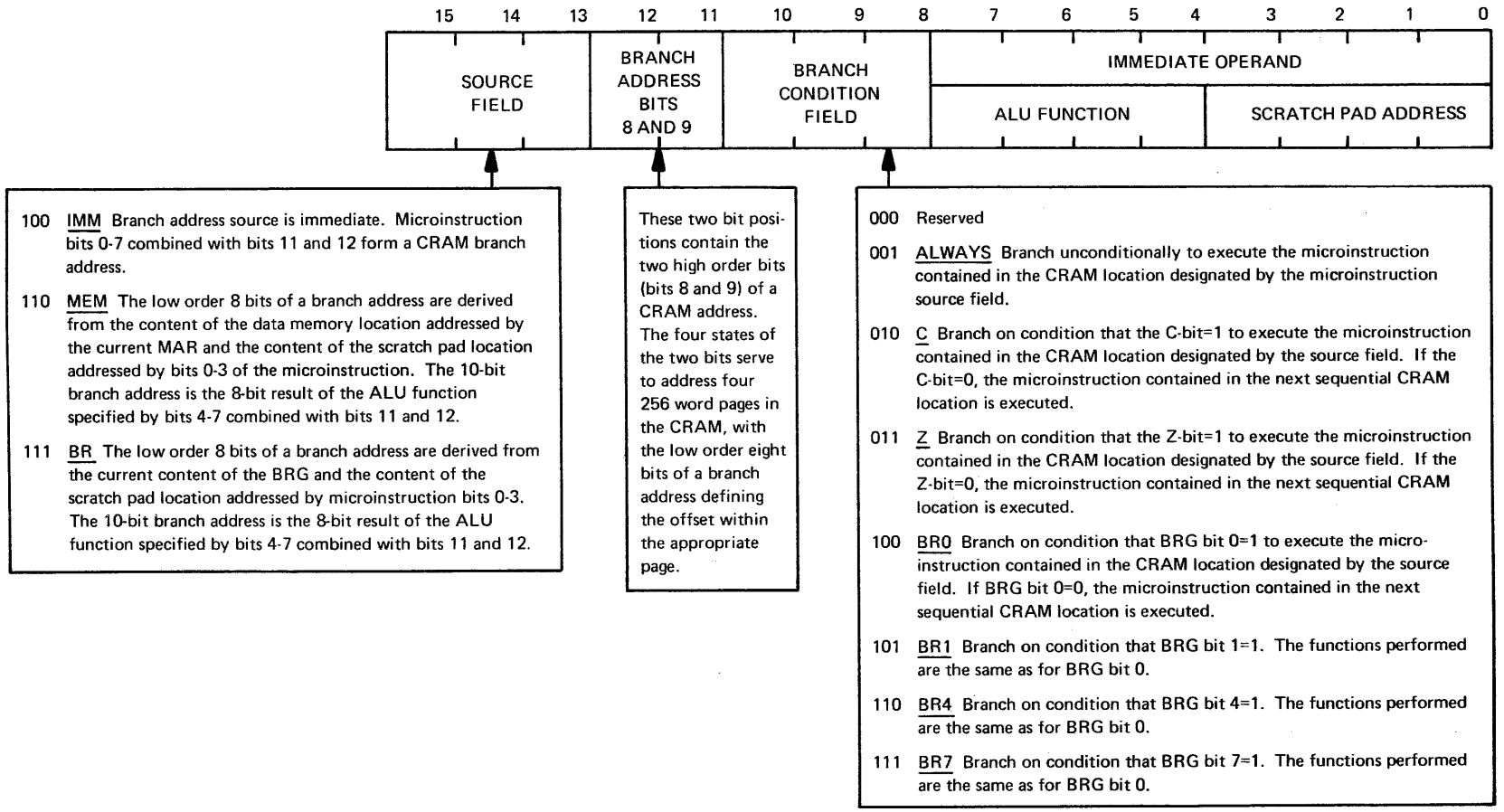


Figure 3-2 Branch Class Microinstructions

When a microprogram is assembled under the KMC11 macroassembler, one of the paging symbols P0 through P3 for the argument Pn should, where required, be included as the last argument in each Branch class microinstruction having a source other than Immediate (Figure 3-2). Otherwise, the assembler will default to page zero. For Branch class instructions whose source is Immediate, the branch address must be a microprogram label with paging performed during assembly.

3.2.2 Branch Class Microinstructions: Formats and Functions

The detailed information for Branch class microinstructions is organized according to microinstruction source. For each Branch class microinstruction in the total set, this detailed information includes the following:

1. the microinstruction name,
2. the macroassembler mnemonic,
3. the microinstruction format,
4. a detailed functional description, and
5. a description of microinstruction arguments.

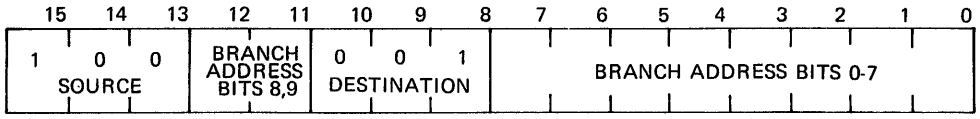
Branch class microinstructions fall into two categories:

1. microinstructions having an immediate source where the branch address is formed by fields within the microinstruction, and
2. microinstructions that derive the branch address from an ALU function performed on the contents of a scratch pad location and on the contents of the BRG or a data memory location.

The 16 arithmetic/logic functions performed by the KMC11 ALU are listed in Table 3-1. Note that execution of Branch class microinstructions has no effect on the state of the C-bit or the Z-bit.

3.2.2.1 **Source Immediate** - All Branch Immediate microinstructions share one characteristic: the branch address must be a valid microprogram label whose binary value is contained within the microinstruction. This characteristic is a direct function of the KMC11 macroassembler and must be adhered to in all microprograms assembled by this utility.

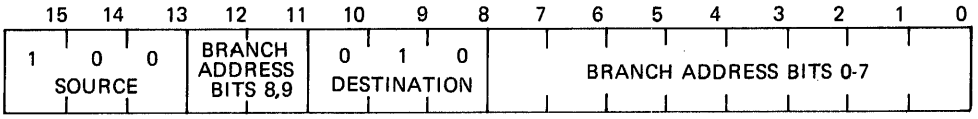
UNCONDITIONAL BRANCH
(ALWAYS label)



An Unconditional Branch microinstruction permits the microprogram to execute a branch to any location in the CRAM. The binary value of a labeled address is contained in the microinstruction and is used to

set the PC to address the branch location. The CRAM page is specified by the state of the branch address field (bits 11 and 12) of the microinstruction with the page offset contained in bits 0 to 7 of the same instruction. The next microinstruction executed is the one at the addressed location.

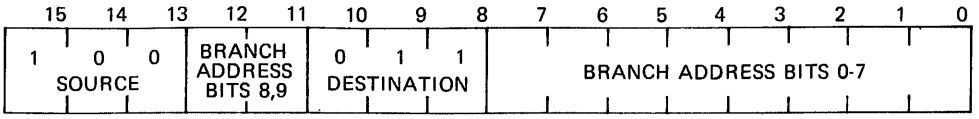
BRANCH ON C-BIT SET
(C label)



This microinstruction, when executed, branches to the labeled CRAM location specified in the microinstruction only if the C-bit is set to one. When the C-bit is set to one, the binary value of a labeled address contained in the microinstruction is used to set the PC to address the branch location. The CRAM page is specified by the state of the branch address field (bits 11 and 12) of the microinstruction with the page offset contained in bits 0 to 7 of the same instruction. (See Section 3.1.2 for details on the C-bit.) The next microinstruction executed is the one at the addressed location.

If the C-bit is equal to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line.

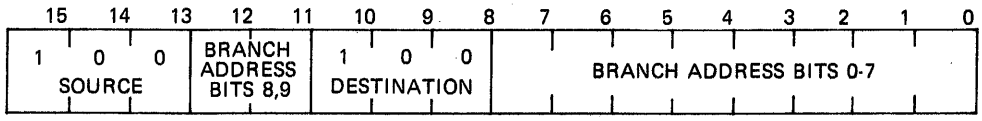
BRANCH ON Z-BIT SET
(Z label)



This microinstruction, when executed, branches to the CRAM location specified in the microinstruction only if the Z-bit is set to one. When the Z-bit is set to one, the binary value of a labeled address contained in the microinstruction is used to set the PC to address the branch location. The CRAM page is specified by the state of the branch address field (bits 11 and 12) of the microinstruction with the page offset contained in bits 0 to 7 of the same instruction. (See Section 3.1.2 for details on the Z-bit.) The next microinstruction executed is the one at the addressed location.

If the Z-bit is equal to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line.

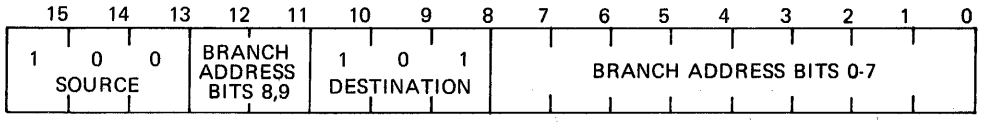
BRANCH ON BRG BIT 0 SET
(BR0 label)



This microinstruction, when executed, branches to the CRAM location specified in the microinstruction only if bit 0 of the BRG is equal to one. When BRG bit 0 is set to one, the binary value of a labeled address contained in the microinstruction is used to set the PC to address the branch location. The CRAM page is specified by the state of the branch address field (bits 11 and 12) of the microinstruction, with the page offset contained in bits 0 to 7 of the same instruction. The next instruction executed is the one at the addressed location.

If BRG bit 0 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line.

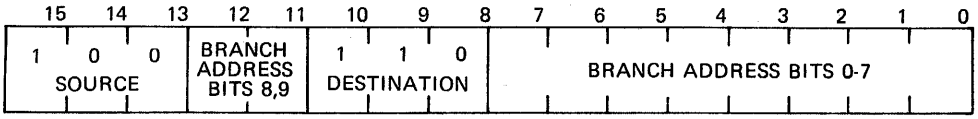
BRANCH ON BRG BIT 1 SET
(BR1 label)



This microinstruction, when executed, branches to the labeled CRAM location specified in the microinstruction only if bit 1 of the BRG is equal to one. When BRG bit 1 is set to one, the binary value of a labeled address contained in the microinstruction is used to set the PC to address the branch location. The CRAM page is specified by the branch address field (bits 11 and 12) of the microinstruction, with the page offset contained in bits 0 to 7 of the same instruction. The next instruction executed is the one at the addressed location.

If BRG bit 1 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line.

BRANCH ON BRG BIT 4 SET
(BR4 label)

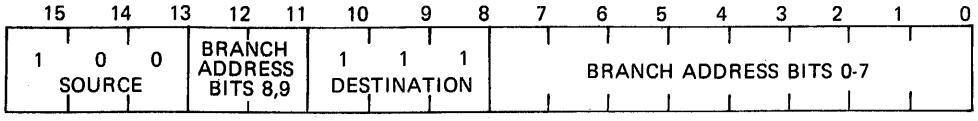


This microinstruction, when executed, branches to the CRAM location specified in the microinstruction only if BRG bit 4 is equal to one. When BRG bit 4 is equal to one, the binary value of a labeled address contained in the microinstruction is used to set the PC to address the branch location. The CRAM page is specified by the branch address field (bits 11 and 12) of the microinstruction, with the page offset

contained in bits 0 to 7 of the same instruction. The next instruction executed is the one at the addressed location.

If BRG bit 4 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line.

BRANCH ON BRG BIT 7 SET
(BR7 label)



This microinstruction, when executed, branches to the CRAM location specified in the microinstruction only if BRG bit 7 is equal to one. When BRG bit 7 is equal to one, the binary value of the labeled address contained in the microinstruction is used to set the PC to address the branch location. The CRAM page is specified by the branch address field (bits 11 and 12) of the microinstruction, with the page offset contained in bits 0 to 7 of the same instruction. The next instruction executed is the one at the addressed location.

If BRG bit 7 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line.

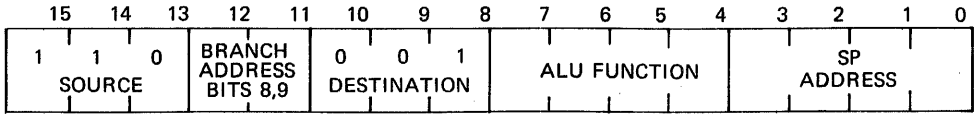
3.2.2.2 **Source Data Memory** - In Branch class microinstructions having data memory as the source, the 8-bit page offset for a branch address can be obtained in one of two ways:

1. directly from the data memory location addressed by the current MAR; or

2. from the result of an ALU function performed on the contents of a data memory location and a scratch pad location.

The page address bits of a complete CRAM address are obtained from the microinstruction branch address field as described in Section 3.2.1.

UNCONDITIONAL BRANCH TO ADDRESS DERIVED FROM MEMORY AND SP
(.ALWAY MEMX, func, SPn, Pn)



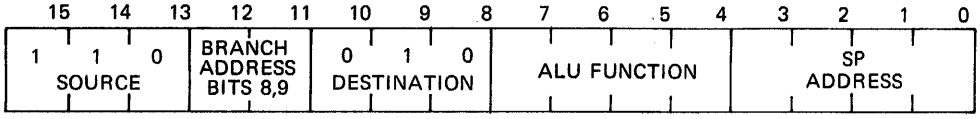
Executing this microinstruction sets the PC to unconditionally branch to a location within a designated CRAM page. The page offset for this location can be derived from the results of a designated ALU function performed on the data memory location addressed by the current MAR and the scratch pad location addressed by the microinstruction. The contents of the addressed data memory location and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this instruction.

The page offset for the branch address can also be the contents of the data memory location addressed by the current MAR using the ALU function Select B (SELB, Table 3-1). The argument Pn specifies the CRAM page. Such an instruction would take the following form:

```
.ALWAY MEMX,SELB,Pn
```

The mnemonics comprising the arguments for the field func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this microinstruction.

BRANCH ON C-BIT SET TO ADDRESS DERIVED FROM MEMORY AND SP
(.C MEMX, func, SPn, Pn)



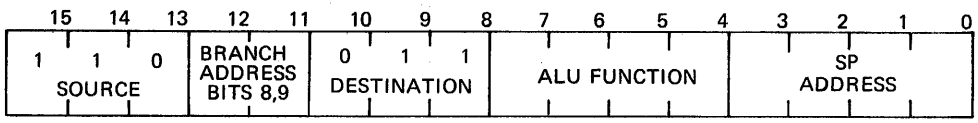
Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if the state of the C-bit is equal to one. If the C-bit is equal to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the contents of the data memory location addressed by the current MAR and the scratch pad location addressed by the microinstruction. The contents of the addressed data memory location and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this instruction.

The page offset for the branch address can also be the contents of the data memory location addressed by the current MAR using the ALU function Select B (SELB, Table 3-1). The argument Pn specifies the CRAM page. Such an instruction would take the following form:

```
.C MEMX,SELB,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this instruction.

BRANCH ON Z-BIT SET TO ADDRESS DERIVED FROM MEMORY AND SP
(.Z MEMX, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if the state of the Z-bit is equal to one. If the Z-bit is equal to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line.

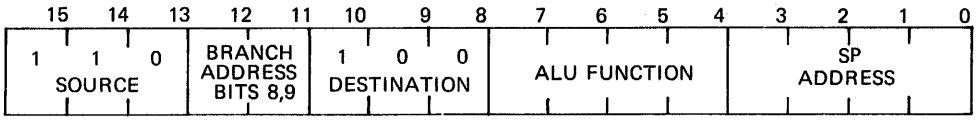
The page offset for the branch location can be derived from the results of a designated ALU function performed on the contents of the data memory location addressed by the current MAR and the scratch pad location addressed by the microinstruction. The contents of the addressed data memory location and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this instruction.

The page offset for a branch location can also be the contents of the data memory location addressed by the current MAR using the ALU function Select B (SELB, Table 3-1). The argument Pn specifies the CRAM page. Such an instruction would take the following form:

```
.Z MEMX,SELB,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

BRANCH ON BRG BIT 0 SET TO ADDRESS DERIVED FROM MEMORY AND SP
(.BR0 MEMX, func, SPn, Pn)



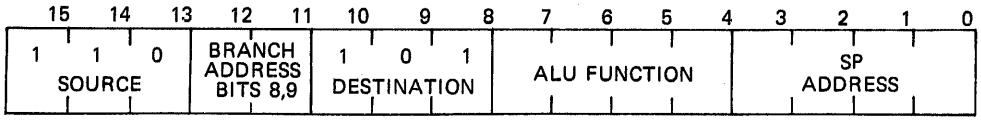
Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 0 of the BRG is equal to one. If BRG bit 0 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line. The page offset for the branch location can be derived from the results of a designated ALU function peformed on the contents of the data memory location addressed by the current MAR and the scratch pad location addressed by the microinstruction. The contents of the addressed data memory location and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this instruction.

The page offset for a branch location can also be the contents of the data memory location addressed by the current MAR using the ALU function Select B (SELB, Table 3-1). The argument Pn specifies the CRAM page. Such an instruction would take the following form:

```
.BR0 MEMX,SELB,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this instruction.

BRANCH ON BRG BIT 1 SET TO ADDRESS DERIVED FROM MEMORY AND SP
(.BR1 MEMX, func, SPn, Pn)



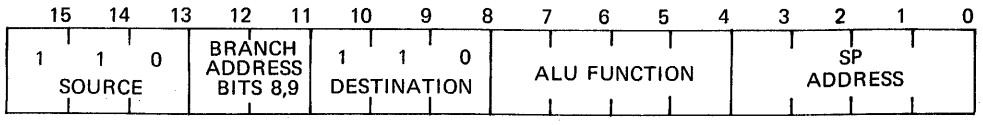
Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 1 of the BRG is equal to one. If BRG bit 1 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the data memory location addressed by the current MAR and the scratch pad location addressed by the microinstruction. The contents of the addressed data memory location and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this instruction.

The page offset for a branch location can also be the contents of the data memory location addressed by the current MAR using the ALU function Select B (SELB, Table 3-1). The argument Pn specifies the CRAM page. Such an instruction would take the following form:

```
.BR1 MEMX,SELB,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

BRANCH ON BRG BIT 4 SET TO ADDRESS DERIVED FROM MEMORY AND SP
(.BR4 MEMX, func, SPn, Pn)



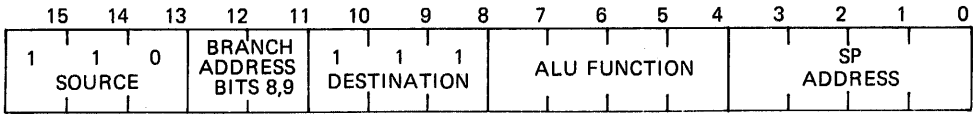
Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 4 of the BRG is equal to one. If BRG bit 4 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the data memory location addressed by the current MAR and the scratch pad location addressed by the microinstruction. The contents of the addressed data memory location and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this instruction.

The page offset for a branch location can also be the contents of the data memory location addressed by the current MAR using the ALU function Select B (SELB, Table 3-1). The argument Pn specifies the CRAM page. Such an instruction would take the following form:

```
.BR4 MEMX,SELB,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this instruction.

BRANCH ON BRG BIT 7 SET TO ADDRESS DERIVED FROM MEMORY AND SP
(.BR7 MEMX, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 7 of the BRG is equal to one. If BRG bit 7 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the data memory location addressed by the current MAR and the scratch pad location addressed by the microinstruction. The contents of the addressed data memory location and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this instruction.

The page offset for a branch location can also be the contents of the data memory location addressed by the current MAR using the ALU function Select B (SELB, Table 3-1). The argument Pn specifies the CRAM page. Such an instruction would take the following form:

```
.BR7 MEMX,SELB,Pn
```

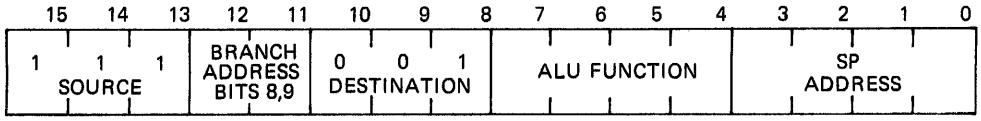
The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in the execution of this instruction.

3.2.2.3 **Source BRG** - Branch class microinstructions having the BRG as a source can obtain an 8-bit CRAM page offset in one of three ways:

1. from the result of an ALU function performed on the contents of the BRG and a scratch pad location;
2. directly from the current BRG; or
3. directly from a scratch pad location.

The page address bits of a complete CRAM address are obtained from the microinstruction branch address field as described in Section 3.2.1.

UNCONDITIONAL BRANCH TO ADDRESS DERIVED FROM BRG AND SP
(.ALWAY BR, func, SPn, Pn)



Executing this microinstruction sets the PC unconditionally to branch to a location within a designated CRAM page. The page offset for the branch address can be derived from the results of a designated ALU function performed on the current contents of the BRG, the scratch pad location addressed by the microinstruction. The contents of the BRG and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this microinstruction.

The page offset for a branch address can also be the current contents of the BRG alone through use of the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

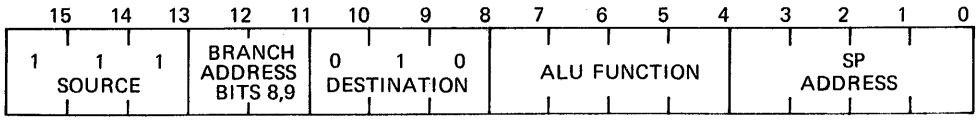
```
.ALWAY BR,SELB,Pn
```

Similarly, using the ALU function Select A (SELA) implements the use of the contents of the addressed scratch pad location as the page offset within the CRAM page specified by Pn. Such an instruction would take the following form:

```
.ALWAY SELA,SPn,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

BRANCH ON C-BIT SET TO ADDRESS DERIVED FROM BRG AND SP
(.C BR, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if the state of the C-bit is equal to one. If the C-bit is equal to zero, the PC is incremented by one to execute the next sequential instruction within the microprogram line. The page offset for the branch address can be derived from the results of a designated ALU function performed on the current contents of the BRG and the scratch pad location addressed by the microinstruction. The contents of the BRG and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this microinstruction.

The page offset for a branch address can also be the current contents of the BRG alone through use of the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

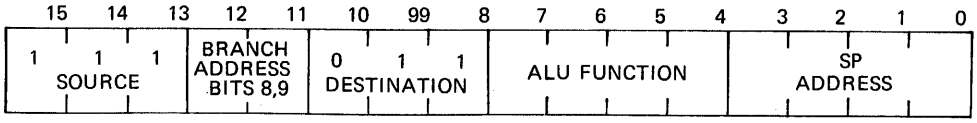
```
.C BR,SELB,Pn
```

Similarly, using the ALU function Select A (SELA) implements the use of the contents of the addressed scratch pad location as the page offset within the CRAM page specified by Pn. Such an instruction would take the following form:

```
.C SELA,SPn,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

BRANCH ON Z-BIT SET TO ADDRESS DERIVED FROM BRG AND SP
(.Z BR, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if the state of the Z-bit is equal to one. If the Z-bit is equal to zero, the PC is incremented by one to execute the next sequential instruction within the program line. The page offset for a branch address can be derived from the results of a designated ALU function performed on the current contents of the BRG and the scratch pad location addressed by the microinstruction. The contents of the BRG and the addressed scratch pad location and the state of the C-bit and Z-bit are not changed by execution of this microinstruction.

The page offset for a branch address can also be the current contents of the BRG through use of the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

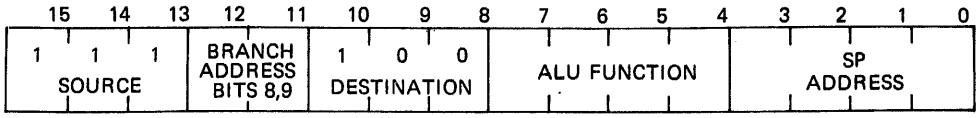
```
.Z BR,SELB,Pn
```

Similarly, using the ALU function Select A (SELA) implements the use of the contents of the addressed scratch pad location as the page offset within the CRAM page specified by Pn. Such an instruction would take the following form:

```
.Z SELA,SPn,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the microinstruction involved in the execution of this microinstruction.

BRANCH ON BRG BIT 0 SET TO ADDRESS DERIVED FROM BRG AND SP
(.BR0 BR, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 0 of the BRG is equal to one. If BRG bit 0 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the BRG and the scratch pad location addressed by the instruction. The contents of the BRG and the addressed scratch pad location and the state of the C-bit and Z-bit are unchanged by execution of this microinstruction.

The page offset for a branch address can also be the current contents of the BRG through use of the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

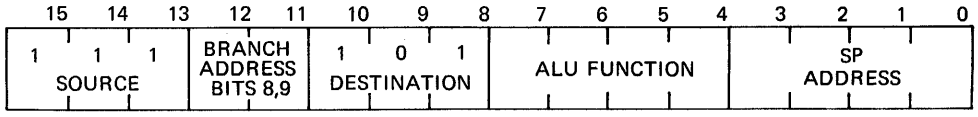
```
.BR0 BR,SELB,Pn
```

Similarly, using the ALU function Select A (SELA) implements the use of the contents of the addressed scratch pad location as the page offset within the CRAM page specified by Pn. Such an instruction would take the following form:

```
.BR0 SELA,SPn,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

BRANCH ON BRG BIT 1 SET TO ADDRESS DERIVED FROM BRG AND SP
(.BR1 BR, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 1 of the BRG is equal to one. If BRG bit 1 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the BRG and the scratch pad location addressed by the instruction. The contents of the BRG and the addressed scratch pad location and the state of the C-bit and Z-bit are unchanged by execution of this microinstruction.

The page offset for a branch address can also be the current contents of the BRG through use of the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

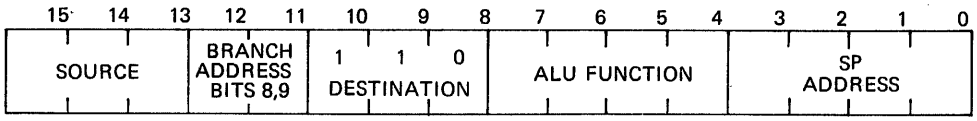
```
.BR1 BR,SELB,Pn
```

Similarly, using the ALU function Select A (SELA) implements the use of the contents of the addressed scratch pad location as the page offset within the CRAM page specified by Pn. Such an instruction would take the following form:

```
.BR1 SELA,SPn,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

BRANCH ON BRG BIT 4 SET TO ADDRESS DERIVED FROM BRG AND SP
(.BR4 BR, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 4 of the BRG is equal to one. If BRG bit 4 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the BRG and the scratch pad location addressed by the instruction. The contents of the BRG and the addressed scratch pad location and the state of the C-bit and Z-bit are unchanged by execution of this microinstruction.

The page offset for a branch address can also be the current contents of the BRG through use of the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

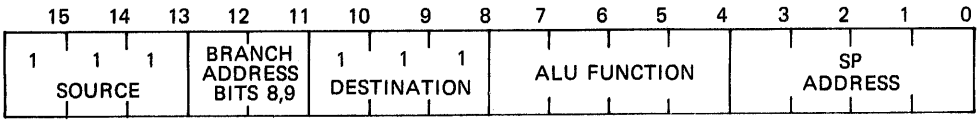
```
.BR4 BR,SELB,Pn
```

Similarly, using the ALU function Select A (SELA) implements the use of the contents of the addressed scratch pad location as the page offset within the CRAM page specified by Pn. Such an instruction would take the following form:

```
.BR4 SELA,SPn,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

BRANCH ON BRG BIT 7 SET TO ADDRESS DERIVED FROM BRG AND SP
(.BR7 BR, func, SPn, Pn)



Executing this microinstruction sets the PC to branch to a location within a designated CRAM page only if bit 7 of the BRG is equal to one. If BRG bit 7 is equal to zero, the PC is incremented by one to execute the next sequential instruction in the program line. The page offset for the branch location can be derived from the results of a designated ALU function performed on the BRG and the scratch pad location addressed by the instruction. The contents of the BRG and the addressed scratch pad location and the state of the C-bit and Z-bit are unchanged by execution of this microinstruction.

The page offset for a branch address can also be the current contents of the BRG through use of the ALU function Select B (SELB, Table 3-1). Such an instruction would take the following form:

```
.BR7 BR,SELB,Pn
```

Similarly, using the ALU function Select A (SELA) implements the use of the contents of the addressed scratch pad location as the page offset within the CRAM page specified by Pn. Such an instruction would take the following form:

```
.BR7 SELA,SPn,Pn
```

The mnemonics comprising the arguments for func are listed in Table 3-1. In the argument SPn, the lowercase letter n represents an octal value in the range 0 to 17, which specifies the physical address of the scratch pad location involved in execution of this microinstruction.

CHAPTER 4

KMC11 MACRO INSTRUCTIONS

This chapter provides the user with the information he needs to define and expand the desired macros to develop programs coded in MACRO-11 assembly language. The information is presented with the assumption that the user has a working knowledge of PDP-11 programming requirements and techniques and of the MACRO-11 assembly language. Detailed information on the PDP-11, the UNIBUS, and the MACRO-11 assembly language is contained in the PDP-11 Peripherals Handbook, EB05961, and the IAS/RSX-11 MACRO-11 Reference Manual, DEC-11-OIMRA-A.

4.1 MICROPROCESSOR REGISTER DEFINITIONS

4.1.1 NPR Control Register

The NPR transactions performed by the KMC11 microprocessor are controlled by the NPR control register.

The NPR bit definitions are as follows:

```
DATI=    1          ;WORD INPUT NPR
DATIH=   3          ;WORD INPUT NPR WITH BUS HOLD
DATO=   21          ;WORD OUTPUT NPR
DATOH=  23          ;WORD OUTPUT NPR WITH BUS HOLD
DATOB= 221          ;BYTE OUTPUT NPR
DATOBH=223          ;BYTE OUTPUT NPR WITH BUS HOLD
```

When performing multiple sequential NPR transactions, it is advantageous, for processing efficiency, to maintain bus control until the last transaction is complete; setting bit 1 (NOT LAST XFER) of the NPR control register guarantees maintaining bus control (i.e., 3, 23, or 223 maintains bus control until the last transaction). The microprogram must clear this bit to relinquish bus control after starting the last transaction of the series.

CAUTION

The maximum number of sequential NPRs that can be executed is determined by the specific KMC11 system configuration. Exceeding this maximum can cause latency problems with associated peripherals such as mass storage devices and communications controllers.

Note that DATOB=221 and DATOBH=223 set bit 7 (BYTE XFER) of the NPR control register and apply only to out-NPR transactions. When this bit is set, bit 0 of the UNIBUS address determines whether the transaction involves the low byte or the high byte.

CAUTION

Setting the BYTE XFER bit during an in-NPR transaction constitutes an illegal operation and should not be done.

The NPR control register is described in Sections 2.1.2 and 2.3.2.

4.2 MACRO INSTRUCTION SYNTAX

4.2.1 Macro Arguments

Macro arguments are essentially free of fixed syntactical rules; that is, there is no predefined order for listing arguments. Multiple arguments within a macro definition are separated from each other by the legal separators listed in Table 4-1. For convenience, the convention of having the arguments ordered as source, ALU function or INBUS address, scratch pad or OUTBUS, and MAR control or page offset has been adopted in this manual.

4.2.2 Source Field Mnemonics

The mnemonics for the source data are as follows:

IMM	Immediate. The source data is the immediate operand (bits 0-7 of the microinstruction). The assembler automatically ANDs all immediate operand bits with 377 (octal) to mask the high operand bits.
IBUS	INBUS or INBUS*. The source data is the contents of the INBUS or INBUS* register addressed by bits 4-7 of the microinstruction.
MEMX	Data Memory. The source data is the contents of the memory location addressed by the current MAR.
BR	Branch Register. The source data is the current contents of the branch register.
MEMI	Data Memory. Same as MEMX except the argument INCMAR may be omitted and the MAR will still be incremented.

4.2.3 INBUS* and INBUS Register Symbolic Addresses

The INBUS* register symbolic addresses and functions are listed in Table 4-2. The INBUS register symbolic addresses and functions are listed in Table 4-3.

4.2.4 Arithmetic/Logic Unit (ALU) Functions

The mnemonics for the ALU functions and the ALU function implemented by each mnemonic are listed in Table 4-4.

4.2.5 OUTBUS* and OUTBUS Register Symbolic Addresses

The OUTBUS* register symbolic addresses and functions are listed in Table 4-5. The OUTBUS register symbolic addresses and functions are listed in Table 4-6.

4.2.6 Scratch Pad Locations

The 16 KMC11 scratch pad memory locations are addressed by the microprogram according to the following definitions:

```
SP0     SP4     SP10    SP14
SP1     SP5     SP11    SP15
SP2     SP6     SP12    SP16
SP3     SP7     SP13    SP17
```

The scratch pad memory is described in Section 2.1.2.4 and Chapter 3.

4.2.7 Memory Address Register (MAR) Field Definitions

The KMC11 MAR is write-accessible only to the microprogram; the MAR field definitions are as follows:

```
LDMAPG          ;LOAD THE 2 MOST SIGNIFICANT BITS OF MAR
LDMAR           ;LOAD THE 8 LEAST SIGNIFICANT BITS OF MAR
INCMAR          ;INCREMENT MAR
```

The above arguments can be optionally used as the last argument in any Move class instruction.

When data memory is used as a source for a move and the MAR is being incremented, the programmer can omit the argument INCMAR and use the macro MEMI instead of MEMX for the source argument. Similarly, when data memory is used as the destination for a move and the MAR is being incremented, the programmer may omit the argument INCMAR and use the macro MEMINC.

4.2.8 Data Memory Page Definitions

The KMC11 Data Memory page (four 256-byte pages) definitions are as follows:

```
P0        ;PAGE 0
P1        ;PAGE 1
P2        ;PAGE 2
P3        ;PAGE 3
```

The data memory is described in detail in Sections 2.1.1.2 and 2.2.

4.3 MICROINSTRUCTION SYNTAX

The microinstructions comprising the KMC11 macro library are divided into two categories:

1. Move class instructions

2. Branch class instructions

Move class instructions provide interregister and intermemory data transfers and are based on destination; Branch class microinstructions provide unconditional or conditional program jumps to program subroutines and are based on conditions. Tables 4-7 through 4-15 list the macroassembler mnemonics for Move class microinstructions; Tables 4-16, 4-17, and 4-18 list macroassembler mnemonics for Branch microinstructions. Table 4-19 lists macroassembler mnemonics for calling subroutines and returning to the main program from subroutines. Table 4-19 also gives the mnemonics for comparing values contained in data memory and the BRG with values in scratch pad locations. These tables briefly describe the function implemented by each instruction as well as the arguments applicable to each macro.

An argument for loading the MAR is usually the last argument in a microinstruction mnemonic. In a given microinstruction, the absence of this argument leaves the MAR unchanged during execution of that microinstruction. Tables 4-7 through 4-15, which define Move class instructions, are based on data destinations.

Generally, NODST Move instructions do not provide for data transfers, but provide program nulls for testing the contents of internal registers or testing the result of a designated Arithmetic/Logic Unit (ALU) function. The one exception is the use of a NODST instruction to load the MAR.

Tables 4-16, 4-17, and 4-18, which define Branch class instructions, are based on data source.

The Chapter 3 section number referenced below the macroassembler mnemonic for each microinstruction in Tables 4-7 through 4-18, gives a detailed description of the microinstruction and the related arguments. The format (bit map) of each microinstruction is also illustrated in those sections.

Table 4-1
Legal Separating and Delimiting Characters Used in Macro Definitions

Character	Function
Tab	A tab is a legal separator between instruction fields.
, (Comma)	A comma is a legal separator between arguments. It is also a legal separator between instruction fields; however, a tab is the preferred separator between instruction fields.
Space	A space is a legal separator between instruction fields; however, a tab is the preferred separator.
<...> (Paired angle brackets)	Paired angle brackets are used to enclose arguments; they must be used when the arguments themselves contain separating characters.

Table 4-2
INBUS* Register Symbolic Addresses*

Symbol	Register Function	Physical Address (octal)
INCON	CSR0	0
MAIN	CSR1 (Maintenance Register)	1
OCON	CSR2	2
LINENM	CSR3	3
PORT1	CSR4	4
PORT2	CSR5	5
PORT3	CSR6	6
PORT4	CSR7	7
NPR	NPR Control Register	10
UBBR	μPMISC Register	11

Note that the first eight symbolic IBUS addresses comprise the KMC11 CSRs and that the symbols NPR and UBBR address the NPR and μPMISC registers. For detailed operational information on the CSRs, the NPR control register, and the μPMISC register refer to Sections 2.3.1, 2.3.2, and 2.3.3.

Table 4-3
INBUS Register Symbolic Addresses*

Symbol	Register Function	Physical Address (octal)
INDAT1	In-NPR data low byte	0
INDAT2	In-NPR data high byte	1
IODAT1	Out-NPR data low byte	2
IODAT2	Out-NPR data high byte	3
IIBA1	In-NPR UNIBUS address low byte	4
IIBA2	In-NPR UNIBUS address high byte	5
IOBA1	Out-NPR UNIBUS address low byte	6
IOBA2	Out-NPR UNIBUS address high byte	7
XREG0	User-specified	10
XREG1	User-specified	11
XREG2	User-specified	12
XREG3	User-specified	13
XREG4	User-specified	14
XREG5	User-specified	15
XREG6	User-specified	16
XREG7	User-specified	17

*For detailed information on the first eight INBUS registers refer to Section 2.3.2 and Figure 2-3. The eight INBUS registers 10 through 17 are assigned to support a high-speed peripheral device such as the DMC11-DA Synchronous Line Unit. The title and document numbers identifying the maintenance and user manuals for this device are listed in Section 1.5.

Table 4-4
Arithmetic/Logic Unit Functions

ALU Function Mnemonic	ALU Function Field Binary Equivalent	Arithmetic/Logic Functions Performed
ADD	0000	Add A and B*
ADDC	0001	Add A and B with carry*
SUBC	0010	Subtract B from A with borrow*
INCA**	0011	Increment A*
APLUSC**	0100	A plus carry*
TWOA**	0101	A plus A*
TWOAC**	0110	A plus A plus carry*
DECA**	0111	Decrement A*
SELA**	1000	Select A side of ALU
SELB	1001	Select B side of ALU
AORNB	1010	A OR NOT B
AANDB	1011	A AND B
AORB	1100	A OR B
AXORB	1101	A exclusive OR B
SUB	1110	Subtract B from A* (two's complement)
SUBOC	1111	Subtract B from A* (one's complement)

* When used in a Move class microinstruction, these ALU functions clock the C-bit.

** These ALU functions do not require a source argument.

Table 4-5
OUTBUS* Register Symbolic Addresses

Symbol	Register Function	Physical Address (octal)
OICON	CSR0	0
OMAIN	CSR1 (Maintenance Register)	1
OOCON	CSR2	2
OLINEN	CSR3	3
OPORT1	CSR4	4
OPORT2	CSR5	5
OPORT3	CSR6	6
OPORT4	CSR7	7
ONPR	NPR Control Register	10
OBR	μPMISC Register	11

Table 4-6
OUTBUS Symbolic Addresses

Symbolic Address	Register Function	Physical Address (octal)
OIDAT1	In-NPR data low byte	0
OIDAT2	In-NPR data high byte	1
OUTDA1	Out-NPR data low byte	2
OUTDA2	Out-NPR data high byte	3
IBA1	In-NPR UNIBUS address low byte	4
IBA2	In-NPR UNIBUS address high byte	5
OBA1	Out-NPR UNIBUS address low byte	6
OBA2	Out-NPR UNIBUS address high byte	7
OXREG0	User-specified	10
OXREG1	User-specified	11
OXREG2	User-specified	12
OXREG3	User-specified	13

(continued on next page)

Table 4-6 (Cont.)
OUTBUS Symbolic Addresses

Symbolic Address	Register Function	Physical Address (octal)
OXREG4	User-specified	14
OXREG5	User-specified	15
OXREG6	User-specified	16
OXREG7	User-specified	17

Table 4-7
Move Instruction Destination: Branch Address Register

Macroassembler Mnemonic	Description
BRWRTE IMM,opr,mar (Section 3.1.2.2)	Stores an 8-bit operand in the BRG. The argument opr can be any number in the range 0-377 (octal) or it can be the contents of any symbolic address designated by the symbol table for the microprogram. For example, BRWRTE IMM,1777 will load 377 (octal) into the BRG since the assembler masks the operand to eight bits.
BRWRTE IBUS,adri,mar (Section 3.1.2.2)	Moves the contents of the addresses INBUS or INBUS* register, adri, to the BRG and leaves the contents of the INBUS or INBUS* register unchanged. The INBUS symbolic addresses are listed in Table 4-3. The INBUS* symbolic addresses are listed in Table 4-2.
BRWRTE MEMX,func,SPn,mar (Section 3.1.2.2)	Performs the specified ALU function, func, on the contents of the data memory location addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn. The result of the ALU function on these values is then stored in the BRG. The contents of the addressed memory location and the scratch pad location remain unchanged. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic.

(continued on next page)

Table 4-7 (Cont.)
Move Instruction Destination: Branch Address Register

Macroassembler Mnemonic	Description
BRWRTE MEMX,func,SPn,mar (Section 3.1.2.2) (Cont.)	For the argument SPn, the n represents an octal number in the range 0-17, which specifies the address of the desired scratch pad location. A direct source-to-destination data transfer from the addressed memory location to the BRG can be achieved with the ALU function Select B. This microinstruction would take the following form: BRWRTE MEMX,SELB
BRWRTE BR,func,SPn,mar (Section 3.1.2.2)	Same as the microinstruction BRWRTE MEMX,func,SPn,mar except it performs the specified ALU function on the contents of the BRG and the contents of the scratch pad location addressed by the argument SPn. If specified by the form BRWRTE SELA,SPn the direct source-to-destination data transfer is from the specified scratch pad location, SPn, to the BRG.

Table 4-8
Move Instruction Destination: OUTBUS*

Macroassembler Mnemonic	Description
OUT IMM,opr,adro,mar (Section 3.1.2.3)	Moves an 8-bit operand directly to the OUTBUS* register addressed by the four low-order bits of the immediate operand. Argument opr is any number or symbol in the range 0-377 (octal). Argument adro must be a symbolic address for one of the OUTBUS* registers listed in Table 4-5.

(continued on next page)

Table 4-8 (Cont.)
Move Instruction Destination: OUTBUS*

Macroassembler Mnemonic	Description
OUT IMM,opr,adro,mar (Section 3.1.2.3) (Cont.)	NOTE The four low-order bits of the argument opr must have a value equal to that of the argument adro. For example, if the argument opr is 311 (octal), then the argument adro must be OUTBUS* register symbolic address OBR. This microinstruction would take the following form: OUT IMM,311,OBR If a conflict occurs between the value of the four low-order bits of the immediate operand and the octal code for an OUTBUS* register symbolic address, an assembly error is posted. The following is a cross reference between 8-bit operands and the appropriate opr and adro arguments. 8-Bit Operand — Argument adro bbb bb0 000 — OICOM bbb bb0 001 — OMAIN bbb bb0 010 — OOCON bbb bb0 011 — OLINEN bbb bb0 100 — OPORT1 bbb bb0 101 — OPORT2 bbb bb0 110 — OPORT3 bbb bb0 111 — OPORT4 bbb bb1 000 — ONPR bbb bb1 001 — OBR where b = any binary bit
OUT IBUS,adri,adro,mar (Section 3.1.2.3)	Moves data directly from the addressed INBUS or INBUS* register to the addressed OUTBUS* register. The argument adri must be one of the INBUS register symbolic addresses listed in Table 4-3 or one of the INBUS* register symbolic addresses listed in Table 4-2. The argument adro must be one of the OUTBUS* register symbolic addresses listed in Table 4-5.

(continued on next page)

Table 4-8 (Cont.)
Move Instruction Destination: OUTBUS*

Macroassembler Mnemonic	Description
OUT MEMX,func,adro,mar (Section 3.1.2.3)	Performs the specified ALU function, func, on the contents of the data memory location addressed by the current MAR and the contents of scratch pad location 0. The result of the ALU function on these values is then stored in the OUTBUS* register addressed by the argument adro. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. The argument adro must be one of the OUTBUS* symbolic addresses listed in Table 4-5. A direct source-to-destination data transfer from the specified memory location to the addressed OUTBUS* register can be achieved with the ALU function Select B; this microinstruction would take the following form: OUT MEMX,SELB,adro Similarly, a direct scratch pad location 0-to-destination data transfer can be achieved with ALU function Select A; this microinstruction would take the following form: OUT SELA,adro
OUT BR,func,adro,mar (Section 3.1.2.3)	Same as the microinstruction OUT MEMX,func,adro,mar except it performs the specified ALU function on the contents of the BRG and the contents of scratch pad location 0. If specified by the form OUT BR,SELB,adro the direct source-to-destination data transfer will be from the OUTBUS* register addressed by the argument adro.

Table 4-9
Move Instruction Destination: Branch Address Register Right-Shifted

Macroassembler Mnemonic	Description
BRSHFT IMM,opr (Section 3.1.2.4)	Shifts the contents of the BRG right one place; bit 0 of the argument opr is then moved to bit position 7 of the BRG. The original (prior to right-shifting) BRG bit 0 is discarded. BRSHFT is equivalent to BRSHFT IMM,0.
BRSHFT IBUS,adri,mar (Section 3.1.2.4)	Shifts the contents of the BRG right one place; bit 0 of the INBUS or INBUS* register addressed by the argument adri is then moved to bit position 7 of the BRG. The original (prior to right-shifting) BRG bit 0 is discarded. The INBUS symbolic addresses are listed in Table 4-3. The INBUS* symbolic addresses are listed in Table 4-2.
BRSHFT MEMX,func,SPn,mar (Section 3.1.2.4)	Shifts the contents of the BRG right one place; the contents of the data memory location addressed by the current MAR and the contents of the scratch pad addressed by the argument SPn are then operated on according to the ALU function specified by func. Bit 0 of the resulting ALU output is moved to bit position 7 of the BRG. The original (prior to right-shifting) BRG bit 0 is discarded. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents an octal number in the range 0-17, which specifies the address of the desired scratch pad location. Bit 0 of the data memory location addressed by the current MAR can be moved to bit position 7 of the BRG by the following form: BRSHFT MEMX,SELB Similarly, bit 0 of the scratch pad location addressed by the argument SPn can be moved to bit position 7 of the BRG by the following form: BRSHFT MEMX,SELA,SPn

(continued on next page)

Table 4-9 (Cont.)
Move Instruction Destination: Branch Address Register Right-Shifted

Macroassembler Mnemonic	Description
BRSHFT BR,func,SPn,mar (Section 3.1.2.4)	Places the unshifted contents of the BRG onto the source bus for input to the B-side of the ALU; the contents of the BRG are then shifted right one place (bit 0 is discarded). The unshifted contents of the BRG and the contents of the scratch pad addressed by the argument SPn are operated on according to the ALU function specified by func. Bit 0 of the resulting ALU output is moved to bit position 7 of the BRG. The mnemonics for the argument func are listed in Table 4-4 long with the ALU function implemented by each mnemonic. For the argument SPn, the n represents the octal number in the range 0-17, which specifies the address of the desired scratch pad location. A major function of this microinstruction is a single-bit right-rotate of the BRG. Since the BRG is both the source and destination, a single-bit right-rotate of the BRG is accomplished by the following form: BROTAT

Table 4-10
Move Instruction Destination: OUTBUS

Macroassembler Mnemonic	Description
OUT IMM,opr,adro,mar (Section 3.1.2.5)	Moves an 8-bit operand directly to the OUTBUS register addressed by the four low-order bits of the immediate operand. Argument opr is an octal number or symbol in the range 0-377. Argument adro must be a symbolic address for one of the OUTBUS registers listed in Table 4-6.

(continued on next page)

Table 4-10 (Cont.)
Move Instruction Destination: OUTBUS

<table>
<tr><th>Macroassembler Mnemonic</th><th>Description</th></tr>
<tr><td>OUT IMM,opr,adro,mar
(Section 3.1.2.5)
(Cont.)</td><td>
NOTE

The four low-order bits of the argument opr must have a value equal to that of argument adro. For example, if the argument opr is 372 (octal), then the argument adro must be OUTBUS register symbolic address OUTDA1. This microinstruction would take the following form:

OUT IMM,372,OUTDA1

If a conflict occurs between the value of the four low-order bits of the immediate operand and the octal code for an OUTBUS register symbolic address, an assembly error will be posted.

The following is a cross reference between 8-bit operands and the appropriate opr and adro arguments.

<table>
<tr><th>8-Bit Operand</th><th>Argument adro</th></tr>
<tr><td>bbb bb0 000</td><td>OIDAT1</td></tr>
<tr><td>bbb bb0 001</td><td>OIDAT2</td></tr>
<tr><td>bbb bb0 010</td><td>OUTDA1</td></tr>
<tr><td>bbb bb0 011</td><td>OUTDA2</td></tr>
<tr><td>bbb bb0 100</td><td>IBA1</td></tr>
<tr><td>bbb bb0 101</td><td>IBA2</td></tr>
<tr><td>bbb bb0 110</td><td>OBA1</td></tr>
<tr><td>bbb bb0 111</td><td>OBA2</td></tr>
<tr><td>bbb bb1 000</td><td>OXREG0</td></tr>
<tr><td>bbb bb1 001</td><td>OXREG1</td></tr>
<tr><td>bbb bb1 010</td><td>OXREG2</td></tr>
<tr><td>bbb bb1 011</td><td>OXREG3</td></tr>
<tr><td>bbb bb1 100</td><td>OXREG4</td></tr>
<tr><td>bbb bb1 101</td><td>OXREG5</td></tr>
<tr><td>bbb bb1 110</td><td>OXREG6</td></tr>
<tr><td>bbb bb1 111</td><td>OXREG7</td></tr>
</table>

where

b = any binary bit
</td></tr>
<tr><td>OUT IBUS,adri,adro,mar
(Section 3.1.2.5)</td><td>Moves data directly from the addressed INBUS or INBUS* register to the addressed OUTBUS register.</td></tr>
</table>

(continued on next page)

Table 4-10 (Cont.)
Move Instruction Destination: OUTBUS

Macroassembler Mnemonic	Description
OUT IBUS,adri,adro,mar (Section 3.1.2.5) (Cont.)	The argument adri must be one of the INBUS register symbolic addresses listed in Table 4-3 or one of the INBUS* register symbolic addresses listed in Table 4-2. The argument adro must be one of the OUTBUS register symbolic addresses listed in Table 4-6.
OUT MEMX,func,adro,mar (Section 3.1.2.5)	Performs the specified ALU function, func, on the contents of the data memory location addressed by the current MAR and the contents of scratch pad location 0. The result of the ALU function on those values is then stored in the OUTBUS register addressed by the argument adro. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. The argument adro must be one of the OUTBUS registers listed in Table 4-6. A direct source-to-destination data transfer from the specified memory location to the addressed OUTBUS register can be achieved with the ALU function Select B. This microinstruction would take the following form: OUT MEMX,SELB,adro A direct source-to-destination data transfer from scratch pad location 0 to the addressed OUTBUS register can be achieved with the ALU function Select A. This microinstruction would take the following form: OUT SELA,adro
OUT BR,func,adro,mar (Section 3.1.2.5)	Same as the microinstruction OUT MEMX,func,adro except it performs the specified ALU function on the contents of the specified BRG and the contents of scratch pad location 0. If specified by the form OUT BR,SELB,adro the direct source-to-destination data transfer will be from the BRG to the OUTBUS register addressed by the argument adro.

Table 4-11
Move Instruction Destination: Data Memory

Macroassembler Mnemonic	Description
MEM IMM,opr,mar (Section 3.1.2.6)	Moves an 8-bit operand directly to the data memory location addressed by the current contents of the MAR. The argument opr can be any number or symbol in the range 0-377 (octal).
MEM IBUS,adri,mar (Section 3.1.2.6)	Moves the contents of the addressed INBUS or INBUS* register, adri, to the data memory location addressed by the current MAR and leaves the contents of the INBUS unchanged. The INBUS symbolic addresses are listed in Table 4-3 and the INBUS* symbolic addresses are listed in Table 4-2.
MEM MEMX,func,SPn,mar (Section 3.1.2.6)	Performs the specified ALU function, func, on the contents of the data memory location addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn. The result of the ALU function on these values is then stored in the same data memory location and the original contents of that data memory location are discarded. The contents of the addressed scratch pad location remain unchanged. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. A direct source-to-destination data transfer from the addressed scratch pad location to the data memory location addressed by the current MAR can be achieved with the ALU function Select A. This microinstruction would take the following form: MEM SELA,SPn

(continued on next page)

Table 4-11 (Cont.)
Move Instruction Destination: Data Memory

Macroassembler Mnemonic	Description
MEM BR,func,SPn,mar (Section 3.1.2.6)	Same as the microinstruction MEM MEMX,func,SPn,mar except it performs the specified ALU function on the contents of the BRG and the contents of the scratch pad location addressed by the argument SPn. If specified by the form MEM BR,SELB the direct source-to-destination data transfer will be from the BRG to the data memory location addressed by the current MAR.

Table 4-12
Move Instruction Destination: Scratch Pad

Macroassembler Mnemonic	Description
SP IMM,opr,SPn,mar (Section 3.1.2.7)	Moves an 8-bit operand directly to the scratch pad location addressed by the four low-order bits of the immediate operand. Argument opr is any number or symbol in the range 0-377 (octal). For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. NOTE The four low-order bits of the argument opr must have a value equal to the value of the argument SPn. For example, if the argument opr is 317 (octal), then the argument SPn must have the value 17 (octal). This microinstruction would take the following form: SP IMM,317,SP17 If a conflict occurs between the value of the four low-order bits of the immediate operand and the value of SPn, an assembly error will be posted.

(continued on next page)

Table 4-12 (Cont.)
Move Instruction Destination: Scratch Pad

Macroassembler Mnemonic	Description
SP IBUS,adri,SPn,mar (Section 3.1.2.7)	Moves data directly from the addressed INBUS or INBUS* register to the addressed scratch pad location. The argument adri must be one of the INBUS register symbolic addresses listed in Table 4-3 or one of the INBUS* register symbolic addresses listed in Table 4-2. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. The contents of the addressed INBUS or INBUS* remain unchanged.
SP MEMX,func,SPn,mar (Section 3.1.2.7)	Performs the specified ALU function, func, on the contents of the data memory location addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn. The result of the ALU function on these values is then stored in the same scratch pad location and the original contents of that scratch pad location are discarded. The contents of the addressed data memory location remain unchanged. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. A direct source-to-destination data transfer from the data memory location addressed by the current MAR to the addressed scratch pad location can be achieved with the ALU function Select B. This microinstruction would take the following form: SP MEMX,SELB,SPn

(continued on next page)

Table 4-12 (Cont.)
Move Instruction Destination: Scratch Pad

Macroassembler Mnemonic	Description
SP BR,func,SPn,mar (Section 3.1.2.7)	Same as the microinstruction SP MEMX,func,SPn,mar except it performs the specified ALU function on the contents of the BRG and the contents of the scratch pad location addressed by the argument SPn. If specified by the form SP BR,SELB,SPn the direct source-to-destination data transfer will be from the BRG to the scratch pad location addressed by the argument SPn.

Table 4-13
Move Instruction Destination: Scratch Pad and Branch Address Register

Macroassembler Mnemonic	Description
SPBR IMM,opr,SPn,mar (Section 3.1.2.8)	Moves an 8-bit operand directly to the BRG and to the scratch pad location addressed by the four low-order bits of the immediate operand. Argument opr is any number or symbol in the range 0-377 (octal). For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. NOTE The four low-order bits of the argument opr must have a value equal to that of the argument SPn. For example, if the argument opr is 230 (octal), then the argument SPn must be 10(octal). This microinstruction would take the following form: SPBR IMM,230,SP10 If a conflict occurs between the value of the four low-order bits of the immediate operand and that of SPn, an assembly error will be posted.

(continued on next page)

Table 4-13 (Cont.)
Move Instruction Destination: Scratch Pad and Branch Address Register

Macroassembler Mnemonic	Description
SPBR IBUS,adri,SPn,mar (Section 3.1.2.8)	Moves data directly from the addressed INBUS or INBUS* register to the BRG and the addressed scratch pad location. The argument adri must be one of the INBUS register symbolic addresses listed in Table 4-3 or one of the INBUS* register symbolic addresses listed in Table 4-2. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. The contents of the addressed INBUS or INBUS* register remain unchanged.
SPBR MEMX,func,SPn,mar (Section 3.1.2.8)	Performs the specified ALU function, func, on the contents of the data memory location addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn. The result of the ALU function on these values is then stored in the BRG and in the same scratch pad location and the original contents of that scratch pad location are discarded. The contents of the addressed data memory location remain unchanged. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. A direct source-to-destination data transfer from the data memory location addressed by the current MAR to the addressed scratch pad location can be achieved with the ALU function Select B; this microinstruction would take the following form: SPBR MEMX,SELB,SPn

(continued on next page)

Table 4-13 (Cont.)
Move Instruction Destination: Scratch Pad and Branch Address Register

Macroassembler Mnemonic	Description
SPBR BR,func,SPn,mar (Section 3.1.2.8)	Same as the microinstruction SPBR MEMX,func,SPn,mar except it performs the specified ALU function on the contents of the BRG and the contents of the scratch pad location addressed by the argument SPn. If specified by the form SPBR BR,SELB,SPn the direct source-to-destination data transfer is from the BRG to the scratch pad location addressed by the argument SPn.

Table 4-14
Move Instruction Destination: NODST (No Destination)

Macroassembler Mnemonic	Description
NODST (Section 3.1.2.1)	Performs a null operation; i.e., a microinstruction is executed, but all internal registers are unchanged. Used to implement a program delay of any length.
NODST INCMAR (Section 3.1.2.1)	Increments the MAR by one.
NODST IMM,opr,LDMAR (Section 3.1.2.1)	Loads the lower eight bits, bits 0-7, of the MAR with the value of opr, 0-255 (decimal), corresponding to the specific memory word addressed (page offset) within a designated page; e.g., NODST IMM,413,LDMAR loads bits 0-7 of the MAR with 13. If the Immediate argument is equal to 377 (octal), the Z-bit is set to one; otherwise it is cleared to zero. This instruction does not affect any other register.
NODST IMM,opr,LDMAPG (Section 3.1.2.1)	Loads the upper two bits, bits 8 and 9, of the MAR with the value of the lower two bits, bits 0 and 1, of the argument opr, corresponding to one of four 256-word pages in data memory; e.g., NODST IMM,113,LDMAPG loads both bits 8 and 9 of the MAR with 1. If the Immediate argument is equal to 377 (octal), the Z-bit is set to one; otherwise it is cleared to zero. This instruction does not affect any other register.

(continued on next page)

Table 4-14 (Cont.)
Move Instruction Destination: NODST (No Destination)

Macroassembler Mnemonic	Description
NODST IBUS,adri,mar (Section 3.1.2.1)	Tests the INBUS or INBUS* addressed by the argument adri as to whether the contents are equal to or not equal to 377 (octal). If the result is equal to 377 (octal), the Z-bit is set to one; otherwise it is cleared to zero. The contents of the INBUS or INBUS* register are unchanged. The INBUS symbolic addresses are listed in Table 4-3. The INBUS* symbolic addresses are listed in Table 4-2.
NODST MEMX,func,SPn,mar (Section 3.1.2.1)	Tests the result of the designated ALU function, func, on the contents of the data memory addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn as to whether the result is equal to or less than 377 (octal). If the result is equal to 377 (octal), the Z-bit is set to one. If the result is less than 377 (octal), the Z-bit is cleared to zero. If the ALU function is a logic function (i.e., A∨B, A∧B, A∀B, or A∨∿B) or if the A side or B side of the ALU is selected, the C-bit is unaffected. If the ALU function is an arithmetic function (i.e., add or increment) that results in a carry, the C-bit is set to one. If the ALU function is an arithmetic function (i.e., subtract or decrement) that results in a borrow, the C-bit is cleared to zero. The contents of the addressed memory location and the scratch pad location remain unchanged. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location.

(continued on next page)

Table 4-14 (Cont.)
Move Instruction Destination: NODST (No Destination)

Macroassembler Mnemonic	Description
NODST BR,func,SPn,mar (Section 3.1.2.1)	Tests the result of the designated ALU function, func, on the contents of the BRG and the contents of the scratch pad location addressed by the argument SPn as to whether the result is equal to or less than 377 (octal). If the result is equal to 377 (octal), the Z-bit is set to one. If the result is less than 377 (octal), the Z-bit is cleared to zero. If the ALU function is a logic function (i.e., A∨B, A∧B, A∀B, or A∨∿B) or if the A side or B side of the ALU is selected, the C-bit is unaffected. If the ALU function is an arithmetic function (i.e., add or increment) that results in a carry, the C-bit is set to one. If the ALU function is an arithmetic function that results in a borrow, the C-bit is cleared to zero. The contents of the BRG and the addressed scratch pad location remain unchanged. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the desired scratch pad location.

Table 4-15
Move Instruction: Increment and Load MAR

Macroassembler Mnemonic	Description
INCMA (Section 3.1.2.1)	Increments the MAR by one.
LDMA IMM,opr (Section 3.1.2.1)	Loads the lower eight bits, bits 0-7, of the MAR with the value of opr, 0-255 (decimal), corresponding to the specific memory word addressed (page offset) within a designated page.

(continued on next page)

Table 4-15 (Cont.)
Move Instruction: Increment and Load MAR

Macroassembler Mnemonic	Description
LDMAP IMM,opr (Section 3.1.2.1)	Loads the upper two bits, bits 8 and 9, of the MAR with the value (0-3) of the upper two bits, bits 8 and 9, of the argument opr corresponding to one of four 256-word pages in data memory. The instruction LDMAP IMM,1400 is equivalent to the instruction NODST IMM,3,LDMAPG

Table 4-16
Branch Instruction Source: Immediate

Macroassembler Mnemonic	Description
NOTE In all immediate branches, the value of label is divided by two to obtain the corresponding microlocation. (The reason for this is that MACRO-11 counts bytes because the PDP-11 is byte addressed while the KMC11 CRAM is word addressed.) The result is then split between the offset field and the page field. The CRAM page is specified by bits 11 and 12 and the page offset is specified by bits 0-8 of the branch address field of the microinstruction.	
ALWAYS label (Section 3.2.2.1)	Causes the microprogram to execute a branch to the binary labeled CRAM location, label.
C label (Section 3.2.2.1)	If the C-bit is set to one, causes the microprogram to execute a branch to the binary labeled CRAM location, label.

(continued on next page)

Table 4-16 (Cont.)
Branch Instruction Source: Immediate

Macroassembler Mnemonic	Description
C label (Section 3.2.2.1) (Cont.)	If the C-bit is cleared to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line. Refer to Chapter 3 for a detailed description on setting and clearing the C-bit.
Z label (Section 3.2.2.1)	Same as the microinstruction C label except it is dependent on whether the Z-bit is set or cleared. Refer to Chapter 3 for a detailed description on setting and clearing the Z-bit.
BR0 label (Section 3.2.2.1)	If BRG bit 0 is set to one, causes the microprogram to execute a branch to the binary labeled CRAM location, label. If BRG bit 0 is cleared to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram line. Refer to Chapter 2 for a detailed description on setting and clearing BRG bits 0, 1, 4, and 7.
BR1 label (Section 3.2.2.1)	Same as the microinstruction BR0 label except it is dependent on whether BRG bit 1 is set or cleared.
BR4 label (Section 3.2.2.1)	Same as the microinstruction BR0 label except it is dependent on whether BRG bit 4 is set or cleared.
BR7 label (Section 3.2.2.1)	Same as the microinstruction BR0 label except it is dependent on whether BRG bit 7 is set or cleared.

Table 4-17
Branch Instruction Source: Data Memory

Macroassembler Mnemonic	Description
.ALWAY MEMX,func,SPn,Pn (Section 3.2.2.2)	Causes the microprogram to unconditionally execute a branch to a specific location (page offset) within a designated CRAM page. The page offset is derived from the results of the specified ALU function, func, performed on the contents of the data memory location addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn. The contents of the addressed memory and scratch pad locations remain unchanged as do the states of the C-bit and the Z-bit. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. For the argument Pn, the n represents a number in the range 0-3, which specifies the desired page number of the CRAM. The page offset can also be derived from the contents of the data memory location addressed by the current MAR. This microinstruction would take the following form: .ALWAY MEMX,SELB,Pn
.C MEMX,func,SPn,Pn (Section 3.2.2.2)	If the C-bit is set to one, causes the microprogram to execute a branch to a specific location (page offset) within a designated CRAM page. The page offset is derived from the results of the specified ALU function, func, performed on the contents of the data memory location addressed by the current MAR and the contents of the scratch pad location addressed by the argument SPn. The contents of the addressed data memory and scratch pad locations remain unchanged as do the states of the C-bit and Z-bit.

(continued on next page)

Table 4-17 (Cont.)
Branch Instruction Source: Data Memory

Macroassembler Mnemonic	Description
.C MEMX,func,SPn,Pn (Section 3.2.2.2) (Cont.)	The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. For the argument Pn, the n represents a number in the range 0-3, which specifies the desired page number of the CRAM. The page offset can also be derived from the contents of the data memory location addressed by the current MAR. This microinstruction would take the following form: .C MEMX,SELB,Pn If the C-bit is cleared to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram.
.Z MEMX,func,SPn,Pn (Section 3.2.2.2)	Same as the microinstruction .C MEMX, func,SPn,Pn except it is dependent on whether the Z-bit is set or cleared.
.BR0 MEMX,func,SPn,Pn (Section 3.2.2.2)	Same as the microinstruction .C MEMX, func,SPn,Pn except it is dependent on whether bit 0 of the BRG is set or cleared.
.BR1 MEMX,func,SPn,Pn (Section 3.2.2.2)	Same as the microinstruction .C MEMX, func,SPn,Pn except it is dependent on whether bit 1 of the BRG is set or cleared.
.BR4 MEMX,func,SPn,Pn (Section 3.2.2.2)	Same as the microinstruction .C MEMX, func,SPn,Pn except it is dependent on whether bit 4 of the BRG is set or cleared.
.BR7 MEMX,func,SPn,Pn (Section 3.2.2.2)	Same as the microinstruction .C MEMX, func,SPn,Pn except it is dependent on whether bit 7 of the BRG is set or cleared.

Table 4-18
Branch Instruction Source: Branch Address Register

Macroassembler Mnemonic	Description
.ALWAY BR,func,SPn,Pn (Section 3.2.2.3)	Causes the microprogram to unconditionally execute a branch to a specific location (page offset) within a designated CRAM page. The page offset is derived from the results of the specified ALU function, func, performed on the contents of the BRG and the contents of the scratch pad location addressed by the argument SPn. The contents of the BRG and the addressed scratch pad location remain unchanged as do the states of the C-bit and the Z-bit. The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. For the argument Pn, the n represents a number in the range 0-3, which specifies the desired page number of the CRAM. The page offset can also be derived from the contents of the BRG or from the contents of the scratch pad location addressed by the microinstruction. These microinstructions take the following forms, respectively: .ALWAY BRG,SELB,Pn .ALWAY SELA,SPn,Pn
.C BR,func,SPn,Pn (Section 3.2.2.3)	If the C-bit is set to one, causes the microprogram to execute a branch to a specific location (page offset) within a designated CRAM page. The page offset is derived from the results of the specified ALU function, func, performed on the contents of the BRG and the contents of the scratch pad location addressed by the argument SPn. The contents of the BRG and the addressed scratch pad location remain unchanged as do the states of the C-bit and the Z-bit.

(continued on next page)

Table 4-18 (Cont.)
Branch Instruction Source: Branch Address Register

Macroassembler Mnemonic	Description
.C BR,func,SPn,Pn (Section 3.2.2.3) (Cont.)	The mnemonics for the argument func are listed in Table 4-4 along with the ALU function implemented by each mnemonic. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. For the argument Pn, the n represents a number in the range 0-3, which specifies the desired page number of the CRAM. The page offset can also be derived from the contents of the BRG or from the contents of the scratch pad location addressed by the microinstruction. These microinstructions take the following forms, respectively: .C BRG,SELB,Pn .C SELA,SPn,Pn If the C-bit is cleared to zero, the PC is incremented by one to execute the next sequential instruction in the microprogram.
.Z BR,func,SPn,Pn (Section 3.2.2.3)	Same as the microinstruction .C BR, func,SPn,Pn except it is dependent on whether the Z-bit is set or cleared.
.BR0 BR,func,SPn,Pn (Section 3.2.2.3)	Same as the microinstruction .C BR, func,SPn,Pn except it is dependent on whether bit 0 of the BRG is set or cleared.
.BR1 BR,func,SPn,Pn (Section 3.2.2.3)	Same as the microinstruction .C BR, func,SPn,Pn except it is dependent on whether bit 1 of the BRG is set or cleared.
.BR4 BR,func,SPn,Pn (Section 3.2.2.3)	Same as the microinstruction .C BR, func,SPn,Pn except it is dependent on whether bit 4 of the BRG is set or cleared.
.BR7 BR,func,SPn,Pn (Section 3.2.2.3)	Same as the microinstruction .C BR, func,SPn,Pn except it is dependent on whether bit 7 of the BRG is set or cleared.

Table 4-19
Compare Values and Subroutine Calls and Returns

Macroassembler Mnemonic	Description
COMP MEMX,SPn	Compares the contents of the data memory addressed by the current MAR with the contents of the scratch pad location addressed by the argument SPn to determine if they are equal. If they are equal, the Z-bit is set to one; if the contents of the addressed data memory location are less than the contents of the addressed scratch pad location, the C-bit is set to one. The contents of the addressed data memory and the scratch pad locations remain unchanged. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location.
COMP BR,SPn	Compares the contents of the BRG with the contents of the scratch pad location addressed by the argument SPn to determine if they are equal. If they are equal, the Z-bit is set to one; if the contents of the BRG are less than the contents of the addressed scratch pad location, the C-bit is set to one. The contents of the BRG and the addressed scratch pad location remain unchanged. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location.
BRADDR label	Stores the eight low-order bits of the labeled function address (the page offset) in the BRG. This macro is useful for creating dispatch tables. In the following example, the base address is in the BRG, and the index is in SP0:

(continued on next page)

Table 4-19 (Cont.)
Compare Values and Subroutine Calls and Returns

Macroassembler Mnemonic	Description
BRADDR label (Cont.)	BRADDR TAB .ALWAY BR,ADD,SP0,P0 TAB: ALWAYS LABEL1 ALWAYS LABEL2 ALWAYS LABEL3 . . . ALWAYS LABELN This example assumes that TAB is on CRAM page 0, and that SP0 contains the appropriate offset (i.e., 0, 1, 2, 3 or n).
MEMADR label	Same as BRADDR, except stores the page offset in memory.
CALLSB SPn,subrtn label [,optBRGval]	Calls and performs the labeled subroutine. The eight low-order bits of the return address are stored in the referenced scratch pad location. This assumes that the labeled subroutine returns to the page where the next instruction is located. Refer to RTNSUB. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired scratch pad location. For each level of call, another scratch pad location is required to store the return address.
CALLSR SPn,subrtn label, return label[,optBRGval]	Same as the microinstruction CALLSB SPn, subrtn label [,optBRGval]. It is used when the labeled subroutine does not return to the page where the next instruction is located or when the user wishes to return to a location other than that following the call.

(continued on next page)

Table 4-19 (Cont.)
Compare Values and Subroutine Calls and Returns

Macroassembler Mnemonic	Description
CALLSR SPn,subrtn label, return label[,optBRGval] (Cont.)	Note that CALLSB SPn,FOO,5 is equivalent to BRADDR .+10 SP BR,SELB,SPn BRWRTE IMM,5 ALWAYS FOO and CALLSR SPn,FOO,RETN,10 is equivalent to BRADDR RETN SP BR,SELB,SPn BRWRTE IMM,10 ALWAYS FOO This assumes RETN is on the page where subroutine FOO returns.
RTNSUB SPn,Pn	Causes the microprogram to return to the scratch pad location addressed by the arguments SPn and Pn. For the argument SPn, the n represents a number in the range 0-17 (octal), which specifies the address of the desired return scratch pad location. For the argument Pn, the n represents a number in the range 0-3, which specifies the desired return page number of the CRAM. Note that RTNSUB SPn,Pn is equivalent to .ALWAY SELA,SPn,Pn

(continued on next page)

Table 4-19 (Cont.)
Compare Values and Subroutine Calls and Returns

Macroassembler Mnemonic	Description
RTNSUB SPn,Pn (Cont.)	NOTE To save CRAM space, the page with the most calls to a given subroutine should be selected over pages with fewer calls to the given subroutine. For example, if page 0 has one call to SUBR and page 1 has two calls to subroutine SUBR, the appropriate form is as follows: RTNSUB SPn,P1 The following forms are appropriate for calling SUBR on any page and then branching to page 1 (with the most calls to SUBR) if B is the next microinstruction to be executed: page 0 → CALLSR SPn,SUBR,A B: next instruction page 1 → A: ALWAYS B

4.4 EXAMPLES OF KMC11 INSTRUCTION MACRO EXPANSIONS

KMC11 instruction macro expansions are summarized in Figure 4-1. Figure 4-2 contains examples of KMC11 instruction macros.

MOVE INSTRUCTIONS:

OUT	IBUS,in addr [src,]alu funct,out addr[,mar] IMM,opr
LDMA LDMAP MEM BRWRTE BRSHFT NODST	IBUS,in addr [src,]alu funct[,sp addr][,mar] IMM,opr
SP	IBUS,in addr
SPBR	[src,]alu funct,sp addr[,mar]
INCMA	
BRADDR MEMADR	label[,mar]
COMP	src,sp addr
MEMINC	IBUS,in addr [src,]alu funct[,sp addr] IMM,opr
CALLSB	sp addr,SUBRTN label[,optBRGval]
CALLSR	sp addr,SUBRTN label,return label[,optBRGval]
RTNSUB	sp addr,page #

BRANCH INSTRUCTIONS:

ALWAYS BR0 BR1 BR4 BR7 C Z	addr
.ALWAY .BR0 .BR1 .BR4 .BR7 .C .Z	IBUS,in addr [src,]alu funct[,sp addr],page # IMM,opr

Figure 4-1 Summary of KMC11 Instruction Macros

MOVE INSTRUCTION EXAMPLES:

```
  OUT          SELA,ONPR              ;move contents of SP0 to the NPR
                                      ;control register

  OUT          BR,AANDB,OBR           ;move to the  PMISC register the
                                      ;logical AND of SP0 and the BRG

  OUT          MEMX,ADD,OBA1,INCMAR   ;add SP0 with memory and store
or                                    ;in the low byte of the output
  OUT          MEMI,ADD,OBA1          ;buffer address register then
                                      ;increment the MAR

  OUT          IMM,200,IOCON,LDMAPG   ;move 200 to input control register
                                      ;CSR0 and load the high-order 2
                                      ;bits (page) of the MAR with zeros

  MEM          MEMX,ADD,SP3,INCMAR    ;add contents of memory location
or                                    ;addressed by current MAR with
  MEMINC       MEMX,ADD,SP3           ;contents of SP3 and store result
or                                    ;in memory then increment the MAR
  MEM          MEMI,ADD,SP3
or
  MEMINC       MEMI,ADD,SP3

  BRWRTE       IMM,200,LDMAR          ;move the 8-bit immediate operand
                                      ;to the BRG and load the low-order
                                      ;8 bits (page offset) of the MAR
                                      ;with octal 200

  BRWRTE       MEMX,SELB              ;direct source-to-destination
                                      ;data transfer from the memory
                                      ;location addressed by the current
                                      ;MAR to the BRG

  BRWRTE       MEMX,ADD,SP3           ;add contents of memory location
                                      ;addressed by current MAR with
                                      ;contents of SP3 and store result
                                      ;in the BRG

  BRWRTE       BR,ADD,SP3             ;add contents of BRG to the con-
                                      ;tents of SP3 and store result in
                                      ;the BRG

  BRWRTE       SELA,SP3               ;direct source-to-destination
                                      ;data transfer from SP3 to the
                                      ;BRG

  NODST        IMM,3,LDMAR            ;load the low-order 8 bits
or                                    ;(page offset) of the MAR with 3
  LDMA         IMM,3

  NODST        INCMAR                 ;increment the MAR
or
  INCMA

  NODST        MEMX,SUB,SP4,LDMAPG    ;load the high-order 2 bits of
or                                    ;the MAR with the low-order 2
  LDMAP        MEMX,SUB,SP4           ;bits of the result from sub-
                                      ;tracting the contents of the
                                      ;memory contents addressed by
                                      ;current MAR from SP4
```

Figure 4-2 Examples of KMC11 Macro Expansions

```
BRANCH INSTRUCTION EXAMPLES:

ALWAYS          155,P2              ;branch to CRAM page 2,
                                    ;page offset 155

BR0             155,P2              ;branch to CRAM page 2, page
                                    ;offset 155 if BRG bit 0
                                    ;set to one

Z               40$                 ;if the previous move instruction
                                    ;had a resultant of 377 octal,
                                    ;branch to label 40$

.ALWAY          BR,AORB,SP0,P2      ;branch to page 2 at the offset
                                    ;determined by the logical
                                    ;OR of the contents of the BRG
                                    ;and the contents of SP0

.BR0            BR,AORB,SP0,P2      ;branch to page 2 at the offset
                                    ;determined by the logical OR
                                    ;of the contents of the BRG and
                                    ;the contents of SP0 if BRG bit
                                    ;0 set to one

.C              BR,AORB,SP0,P2      ;if the carry bit is set branch
                                    ;to page 2 offset determined by
                                    ;the logical OR of the BRG and
                                    ;SP0
```

Figure 4-2 (Cont.) Examples of KMC11 Macro Expansions

4.5 RESERVED SYMBOLS

The following symbols are reserved for the macroassembler:

AANDB	IIBA1	OCON	PORT3	SP5
ADD	IIBA2	OIDAT1	PORT4	SP6
ADDC	IMM	OIDAT2	P0	SP7
ALCOND	INCA	OINCON	P1	START
AORB	INCMAR	OLINEN	P2	SUB
AORNB	INCON	OMAIN	P3	SUBC
APLUSC	INDAT1	ONPR	SELA	SUBTC
AXORB	INDAT2	OOCON	SELB	TWOA
BR	IOBA1	OPORT1	SHFTBR	TWOAC
BR0CON	IOBA2	OPORT2	SPBRX	UBBR
BR1CON	IODAT1	OPORT3	SPX	WRMEM
BR4CON	IODAT2	OPORT4	SP0	WROUT
BR7CON	JUMP	OUTDA1	SP1	WROUTX
CCOND	LDMAPG	OUTDA2	SP10	WRTEBR
DATI	LDMAR	OXREG0	SP11	XREG0
DATIH	LINENM	OXREG1	SP12	XREG1
DATO	MAIN	OXREG2	SP13	XREG2
DATOB	MEMI	OXREG3	SP14	XREG3
DATOBH	MEMX	OXREG4	SP15	XREG4
DATOH	MOVE	OXREG5	SP16	XREG5
DECA	NPR	OXREG6	SP17	XREG6
IBA1	OBA1	OXREG7	SP2	XREG7
IBA2	OBA2	PORT1	SP3	ZCOND
IBUS	OBR	PORT2	SP4	

The programmer (user) is cautioned not to employ these reserved symbols when constructing user-defined symbols because an assembly error will result.

4.6 OPERATING INSTRUCTIONS

The user should follow the general rules and procedures discussed in this section when operating the assembler.

The macro definition file KMCMAC.MAC should precede any user microcode macro calls. For example, if the user has microcode macro calls on a file with filename MICRO.MAC, he should answer the assembler prompt with

```
object file, listing file = KMCMAC,MICRO
```

The user must supply

```
.END
```

at the end of the microcode program.

For example, using RSX-11M,

```
>RUN $MAC
MAC> object file, listing file = KMCMAC, microcode input file

MAC>^Z
```

The user should then link or task build the object file according to the format that the loader expects.

CHAPTER 5

KMC11 LOADER

5.1 INTRODUCTION

The KMC11 loader is a utility used during microprogram development and loading. It enables the user to load the KMC11 CRAM from a file when the file name is input at the console. This utility runs as a privileged task under the RSX-11M, RSX-11D, and IAS operating systems. It runs when the microprogram is being loaded into the KMC11 CRAM during system initialization or reconfiguration. If the user wants to develop his own driver which incorporates a built-in loader that loads from an area of PDP-11 memory, the KMC11 basic loader subroutine in Section 5.2 is an example that can be followed. The KMC11 Loader (KMCLDR) is described in Section 5.3, which also gives some examples of its use under RSX-11M.

In this chapter, the KMC11 is used as the reference point for all transfers of information between the PDP-11 processor program and the KMC11 microprocessor. An OUT-transfer transfers information from the KMC11 to the PDP-11 program; an IN-transfer transfers information from the KMC11 to the PDP-11 program.

Eight byte-sized Control and Status Registers (CSRs) are used for the exchange of control and status information between the PDP-11 program and the KMC11. The eight CSRs are byte or word addressable from the UNIBUS. The UNIBUS addresses are 76xxx0 through 76xxx7 (BSEL0 through BSEL7) with the even addresses forming the word boundaries (SEL0, SEL2, SEL4, and SEL6).

The only CSR having a fixed or hardware-defined format is CSR1 (BSEL1): the maintenance register (Figure 5-1). The other seven CSRs are undefined and the user can program them to satisfy specific requirements.

The maintenance register (BSEL1) is used primarily for initializing and servicing the KMC11; two BSEL1 bits (RAM 0 and CRAM WRITE), however, are used when loading the KMC11 CRAM (Figure 3-1).

RAM 0, when set, modifies the source paths for SEL4 to be the CRAM maintenance address register, enabling CRAM read and/or write via SEL6 of the specified locations. A write is accomplished by loading the new CRAM data into SEL6 and asserting CRAM WRITE. CRAM read is accomplished by reading SEL6.

CRAM WRITE, when set, allows the contents of SEL6 to be loaded into the CRAM at the address specified by SEL4. Note that RAM 0 must also be set to accomplish the loading procedure.

The CSRs are described in detail in Section 2.3.1; the CSRs and all bits of CSR1 are described in detail in the KMC11 General Purpose Microprocessor User's Manual, EK-KMC11-OP.

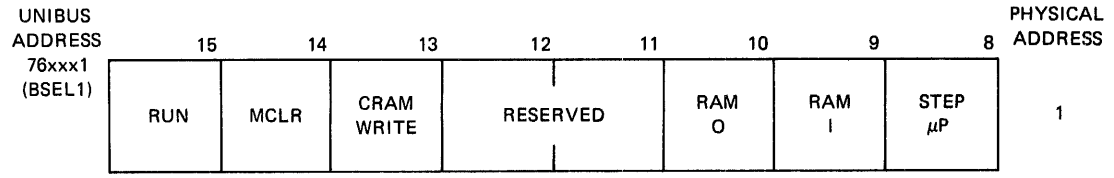


Figure 5-1 Control and Status Registers CSR1 Bit Map

5.2 KMC11 BASIC LOADER SUBROUTINE

The following procedure for loading the KMC11 CRAM fully utilizes the KMC11 hardware and provides for future compatibility:

1. Write CSR0 with only bit 10 (BSEL1 bit 2) (RAM OUTPUT) set.
2. Load the right-justified PC into CSR4.
3. Load CRAM data into CSR6.
4. Set CSR0 bit 13 (BSEL1 bit 5) (CRAM WRITE).
5. Clear CSR0.
6. Repeat Steps 2 through 5 as necessary to load the required instructions.

The following procedure is used to verify the CRAM:

1. Write CSR0 with only bit 10 (BSEL1 bit 2) set.
2. Load the right-justified PC into CSR4.
3. Read CRAM data from CSR6.
4. Clear CSR0.
5. Repeat Steps 2 through 4 as necessary to verify the instructions.

The KMC11 Basic Loader Subroutine can be incorporated into a user-developed driver. (See Figure 5-2.)

```
; WRITE THE RAM


; INPUTS:
;       R0 = NUMBER OF WORDS TO WRITE
;       R3 = CSR ADDRESS OF KMC-11
;       R5 = CRAM ADDRESS AT WHICH TO START LOADING
;       BUFF = BUFFER CONTAINING MICRO-INSTRUCTIONS
;       S.LOAD (STATUS) = FLAG TO INDICATE A LOAD (1) OR COMPARE (0)
;               IS TO BE PERFORMED

WTRAM:
        MOV     #BUFF,R4        ;GET BUFFER ADDRESS
10$:    BIT     #S.LOAD,STATUS  ;LOAD KMC?
        BEQ     15$             ;NO,JUST COMPARE
        MOV     #2000,(R3)      ;SELECT CRAM
        MOV     R5,4(R3)        ;LOAD ADDRESS
        MOV     (R4),6(R3)      ;PUT THE DATA IN THE REGISTER
        BIS     #20000,(R3)     ;CLOCK IT IN
```

Figure 5-2 KMC11 Basic Loader Subroutines

```
15$:    CLR     (R3)            ;CLEAR CSR 0
        CLR     4(R3)           ;CLEAR CSR 4
        CLR     6(R3)           ;CLEAR DATA PORT
        MOV     #2000,(R3)      ;READ CURRENT CRAM LOCATION
        MOV     R5,4(R3)        ;LOAD ADDRESS AGAIN
        CMP     (R4),6(R3)      ;EQUAL TO WHAT JUST WRITTEN THERE?
        BNE     25$             ;NO,RAM WRITE ERROR
20$:    ADD     #2,R4           ;ADDRESS NEXT WORD IN INPUT FILE
        INC     R5              ;INCREMENT THE RAM ADDRESS
        DEC     R0              ;ONE LESS WORD
        BNE     10$             ;KEEP GOING
        CCC                     ;CLEAR CONDITION CODE
        RETURN
25$:    ERROR ROUTINE
          .
          .
          .
        BR      20$             ;CONTINUE
```

Figure 5-2 (Cont.) KMC11 Basic Loader Subroutines

5.3 KMC11 LOADER UTILITY PROGRAM

Figure 5-3 is a printout example of the KMC11 loader running on RSX-11M, and Figure 5-4 is an error printout example. In both examples, the underscored text is system-generated and the nonunderscored text is user-generated.

```
>RUN KMCLDR
KMC LOADER
CSR? 170
FILE NAME? COMIOPDZ
LOAD OR COMPARE? L
KMC LOAD COMPLETE
>
```

Figure 5-3 KMC-11 Loader Printout Example

```
>RUN KMCLDR
KMC LOADER
CSR? 170
FILE NAME? COMIOPDZ.TSK
LOAD OR COMPARE? C
***KMC COMPARE ERROR AT 000050 SOURCE=101020    KMC RAM=101024 ***
KMC COMPARE COMPLETE
>
```

Figure 5-4 KMC-11 Loader Error Printout Example

In Figures 5-3 and 5-4, the user-generated answer (170) to the system-generated question "CSR?" is ORed by the system hardware with 760000 to obtain the address of CSR0, i.e., 760170. The default for the device is SY (system device) and the default for user code is the current user identification code. In Figures 5-3 and 5-4, the file name is COMIOPDZ; there are no defaults for file name. In Figure 5-4, the file type is .TSK; the default for file type is .TSK. For file version, the default for an input file is the highest-numbered existing version.

During a load operation, a compare is automatically performed, and a message is typed for all errors. An error printout during a load operation indicates a faulty CRAM location; DIGITAL Field Service should be called to correct the situation.

A compare is also useful during debugging to obtain a listing of all modified locations when CRAM locations have been changed. The error printout example in Figure 5-4 indicates this use of the loader. Alternately, an error could indicate a faulty CRAM location if the user has not modified the CRAM since loading.

5.3.1 Loader Assembly

To assemble the loader, the user should type the following statement after the prompt, which is underlined for clarity:

```
> MAC KMCLDR=[1,1]EXEMC/ML,[user UIC],KMCLDR
```

NOTE

The KMC11 loader must be assembled and the microcode must be built on the same version of RSX-11M.

5.3.2 Loader and Microcode Task Building

To task build the loader, the user should type the following statement after the underlined prompts:

```
>TKB KMCLDR/PR=KMCLDR
```

To task build the microcode, the user should type the following statements after the underlined prompts:

```
TKB>file name/-HD/-MM=file name.OBJ
TKB>/
ENTER OPTIONS:
TKB>STACK=0
TKB>PAR=:0:1000
TKB>//
```

NOTE

File name .OBJ is the output of the assembler. (See Chapter 4.)

The output of the task builder results in a file with at least two label blocks of 512 bytes each, followed by the microcode instructions. These label blocks are stripped (ignored or skipped) by the KMCLDR and should also be skipped if a user-designed utility is used to read this file.

Detailed task building instructions are contained in the RSX-11M Task Builder Reference Manual, DEC-11-OMTBA.

CHAPTER 6

KMC11 DEBUGGING AID

This chapter gives the programmer the information he needs to use the KMC11 Debugging Aid. The KMC11 Debugging Aid, identified by the mnemonic KMCDA, provides the microprogram developer with the ability to isolate and correct microprogram errors. KMCDA uses a minimal amount of KMC11 microinstruction memory (CRAM) space, and it uses that space only when operating in the breakpoint mode.

For all KMC11 debugging operations, KMCDA is resident in the associated main CPU with the microprogram manipulation implemented by the debugging commands being executed through the KMC11 CSRs. KMCDA runs as a privileged task on all the RSX-11 operating systems as well as on IAS. KMCDA is configured to operate from any command terminal or TTY supported by RSX-11 operating systems and IAS.

KMCDA is distributed as an object module on the distribution kit. As the first step in using KMCDA, the microprogram uses the utility PIP to move the object module KMCDA.OBJ and the command file KMCDA.CMD to the system device (SY). Next, the user task builds KMCDA by typing the following:

```
TKB @KMCDA
```

To run KMCDA, the programmer types

```
RUN KMCDA
```

KMCDA responds with the following question:

```
CSR?
```

and the programmer types in the CSR address for the KMC11 being debugged. For example,

```
CSR?160170
```

NOTE

The address given in the above example (160170) could have been entered as 170 because KMCDA assumes that the three high-order bits are ones.

KMCDA then responds with the prompt symbol ^(up arrow). At this point debugging operations can begin. When debugging operations are finished, KMCDA can be exited by typing the following command after the prompt:

```
^X
```

6.1 COMMAND CATEGORIES

Commands executed by KMCDA fall into six functional categories:

1. Examine and modify CRAM contents. With commands in this category, the programmer can open and examine the contents of selected CRAM locations either sequentially or randomly, and where required, modify the contents of the location being examined.

2. Control microprogram execution. Using these commands the programmer can initiate microprogram execution, set and clear breakpoint, single step the microprocessor, and exit KMCDA.

3. Examine and modify CSRs. With these commands, the contents of each register comprising the KMC11 CSRs (the INBUS* registers 0 through 7, see Figure 2-2 and Table 3-3) can be examined and modified.

4. Examine internal registers and data memory. Commands in this category permit the examination of any internal register including the BRG, the scratch pad registers, the INBUS and INBUS* including the NPR, μPMISC and Line Unit registers, and designated sets of data memory locations.

5. Use utility commands. These commands provide the programmer with a variety of utility functions necessary to debugging activities. These commands usually support such functions as replacing microinstructions, listing breakpoint, modifying the MAR and the data memory and the scratch pad, zeroing the CRAM, and calculating branch offsets.

The command structure for KMCDA is patterned after the structure for IAS/RSX-11 ODT. Consequently, prior experience in using ODT is helpful when using KMCDA. Note that like ODT, KMCDA will perform additions and subtractions of octal numbers that are parts of a value in a command string. The only exception to this are values to be used to modify the CSRs. This feature is valuable when dealing with two octal numbers of large magnitude such as two high-order addresses. The following sections provide the necessary detailed information for using the debugging commands within each command category.

6.1.1 Examine and Modify CRAM

The ability to examine and modify a microinstruction is a basic program debugging requirement. Using the commands in this category, the programmer can open any designated CRAM location, examine the contents of that location, modify it as required, and close it. In addition, the option also exists to close the current location and open the next sequential location or the prior location.

There are two basic command types in this category. In the first type, the CRAM location of interest is accessed by the assembly listing address, which is divided by two by KMCDA to obtain the physical address. In the second type, the CRAM location of interest is addressed directly by the physical address. The command to examine and modify an assembled CRAM location takes the following form:

```
n/[m]<close>
```

where

n is an even octal integer having a value in the range 0 to 3776. It designates the microinstruction address, as assembled by the KMC11 microassembler, that is to be examined and if necessary modified.

NOTE

This address appears on the listing from MACRO-11 and is twice the value of the corresponding control RAM address.

If n is specified as an odd octal integer in this range or exceeds this range, KMCDA posts an error by displaying the symbol ?.

The command to examine and modify a CRAM physical location takes the following form:

n\[m]<close>

where

n is an even or odd octal integer having a value in the range 0 to 1777. It designates the physical address of the location within the CRAM to be examined and if necessary modified. If n exceeds this value, KMCDA posts an error by displaying the symbol ?.

For both command types,

[m] is an optional octal integer equivalent to the binary form of a microinstruction that is to replace the microinstruction existing at the CRAM location addressed by n. If m is omitted from the command string (null), the contents of the microinstruction location addressed by n remain unchanged.

<close> must be one of the following:

<CR> (carriage return) causes the CRAM location addressed by n to be closed with no other location being opened.

<LF> (line feed) causes the CRAM location addressed by n to be closed and the next sequential location opened and displayed for examination. The next sequential location can be changed by entering a pertinent value for [m]. In addition, following [m] by <LF> closes location n and opens location n+1. This sequence can be continued as required.

<^> (up arrow) causes the CRAM location addressed by n to be closed and the next prior location opened and displayed for examination. The next prior location can be changed by entering a pertinent value for [m]. In addition, following [m] by an <^> closes location n and opens location n-1. This sequence can be continued as required.

KMCDA does not limit the programmer as to the sequence of closings within CRAM boundaries. An <LF> or an <^> can be entered in any described pattern, and a <CR> can be executed at any point in an examination.

An example of these two types of commands is as follows:

```
        MACRO-11 Listing

00000   BRWRTE   IMM,5
00002   SP       BR,SELB,SP0
```

To examine with KMCDA:

```
0/000405<CR>      or 0\000405<CR>
2/063220<CR>      or 1\063220<CR>
```

6.1.2 Execution Control Commands

Executive control commands are used in the management of breakpoints and the pursuant analysis and control of microprogram operation and single stepping. The specific type of commands in this category are as follows:

1. Set breakpoints.
2. Clear breakpoints.
3. Begin execution of microprogram.
4. Proceed from breakpoint.
5. Step microprogram.

When a microprogram is to be debugged using breakpoints, the programmer must reserve the last 16 CRAM locations for breakpoint handling microinstructions. These microinstructions are described in detail in Section 6.2.

6.1.2.1 **Set Breakpoints** - The set breakpoint command takes the following form:

```
n;[m]B
```

where

n is an even octal integer in the range 2 to 3776. It designates the address of the microinstruction location, as assembled by the KMC11 macroassembler, that is to be a breakpoint location. If n is an odd octal integer or exceeds 3776, KMCDA posts an error by displaying the symbol ?.

[m] is an optional octal integer in the range 0 to 7. It identifies the specific breakpoint within the total set of eight breakpoints. If [m] is not included as part of the command string, KMCDA will assign the identity number of the next unassigned breakpoint with the lowest identity number to identify the breakpoint currently being set. For example, with no breakpoints set

```
^774;0B   sets breakpoint 0
^362;3B   sets breakpoint 3
^142;4B   sets breakpoint 4
^2062;B   sets breakpoint 1
```

If a breakpoint has been previously set, a subsequent command can reset it. For example,

```
^176;4B
```

resets breakpoint 4 to a new CRAM address.

6.1.2.2 **Clear Breakpoints** - The command to clear breakpoint takes the following form:

```
[n]B
```

where

[n] is the optional identity number of the breakpoint to be cleared. For example,

```
^3B
```

clears breakpoint 3.

If [n] is not specified, all eight breakpoints are cleared, for example,

```
^B
```

6.1.2.3 **Begin Execution of Microprogram** - This command permits the programmer to begin execution of a microprogram at any valid CRAM location. The command takes the following form:

```
[n]G
```

where

[n] is an optional even octal integer in the range 0 to 3776. It designates the microinstruction address, as assembled by the KMC11 macroassembler, to be the location from which the microprogram is to begin execution. KMCDA then divides the value of n by two to derive the actual CRAM address. For example,

```
2006G
```

would begin microprogram execution of CRAM location 1003 (octal) or 515 (decimal).

If [n] is not used and the command is

```
G
```

microprogram execution begins at the CRAM location addressed by the current PC.

If any breakpoints are pending at the start of microprogram execution, no further prompts will be issued until a breakpoint is encountered. If there are no breakpoints pending, KMCDA indicates return to the prompt mode by displaying an ^ (up arrow) and the microprogram is permitted to run until the next command is entered. At this point the KMC11 is halted at an indeterminate location.

6.1.2.4 **Proceed from Breakpoint** - With this command, the user can start the microprogram at the last encountered breakpoint and continue execution while encountering that breakpoint a specified number of times. After encountering the last breakpoint the specified number of times, KMCDA halts the microprogram. This command takes the following form:

```
[n]P
```

where

[n] is an optional positive octal integer in the range 1 through 77777. It specifies the number of times the last encountered breakpoint will be encountered. For example, for the command

```
15P
```

the last encountered breakpoint is passed for a count of 14 (octal) and on the fifteenth time the microprogram is halted at the pertinent breakpoint location. Assuming, for example, that the breakpoint of interest is breakpoint 4, KMCDA will then display the following:

```
MB4:000176
```

If [n] is not specified, the proceed counter in KMCDA is set to one and the microprogram proceeds as if the command 1P was issued. For details on breakpoint display (MB), see Section 6.1.5.1.

If the microprogram is halted at a location other than a breakpoint, issuing a P command is equivalent to issuing a G command.

6.1.2.5 **Single Step** - This command allows transfer of control to a designated microprogram location followed by execution of a specified number of microinstructions beginning at the location to which control transferred. This command takes the following form:

```
[n];[m]S
```

where

[n] is an optional even octal integer in the range 0 to 3776. It designates the address of the microinstruction location, as assembled by the KMC11 macroassembler, from which microprogram execution is to start. If [n] is not present, KMCDA begins microinstruction execution at the CRAM location addressed by the current PC. If the value of [n] exceeds the specified range or is an odd octal integer in the specified range, KMCDA posts an error by displaying the symbol ?.

[m] is an optional octal integer in the range 1 to 1777. It specifies the number of instructions, including the instruction at the starting location [n], to be executed before halting. If [m] is not specified, the instruction addressed by [n], or the PC where [n] is not specified, is executed. Similarly, when neither [n] nor [m] is specified, the single step function is performed.

6.1.3 Examine and Modify CSRs

As described in Section 2.2.4 and Figure 2-1, the UNIBUS interface for the KMC11 consists of eight 8-bit Status and Control Registers (CSRs). The commands in this category allow examination and, if necessary, modification of the KMC11 CSRs.

This category includes a single basic command with two optional variations. In the first variation, the CSR of interest can be opened, examined, and closed. In the second variation, the CSR of interest can be opened, examined, modified, and then closed. This basic command takes the following forms:

```
$[n]/<close>
```

and

```
$[n]/[m]<close>
```

where

[n] is an optional octal integer in the range 0 to 7. It specifies the physical address of the CSR to be examined or modified. If [n] is not specified, CSR 0 is accessed.

[m] is an optional octal integer equivalent to an 8-bit binary array that is to replace the contents of the CSR accessed by [n].

For both variations of this command, the field <close> must be one of the following:

<CR> (carriage return) causes the CSR addressed by [n] to be closed with no other CSR being opened.

<LF> (line feed) causes the CSR addressed by [n] to be closed and the next sequential CSR opened and displayed for inspection. The displayed CSR can be modified at this point by entering a pertinent value for [m]. In addition, following [m] by an <LF> closes location n and opens location n+1. This sequence can be continued until the last CSR is opened.

<^> (up arrow) causes the CSR addressed by [n] to be closed and the next prior location opened and displayed.

6.1.4 Examine Internal Registers and Data Memory

The internal registers and data memory have three basic command types. One type displays the contents of the BRG and the 16 bytes of the scratch pad for examination. A second command type displays the contents of the INBUS and INBUS* for examination. The third type displays selected portions of data memory for examination. In all types, the specified contents are printed out or presented as a visual display, depending on the type of terminal being used for microprogram debugging.

6.1.4.1 Examine BRG and Scratch Pad - This command takes the following form:

```
$R
```

As an example of how the pertinent registers are displayed consider this annotated conversation with KMCDA. Command is input following KMCDA prompt, and the following is displayed:

```
^$R
004 : 000 000 103 232 000 101 010 001 * 110 004 000 100 000 312 305 315
^
BRG              SP0 to 7                        SP8 to 15
```

Note that the prompt symbol ^ following the display of the BRG and scratch pad contents indicates that KMCDA is ready to receive the next debugging command.

6.1.4.2 **Examine INBUS and INBUS*** - This command takes the following form:

```
$I
```

As an example of how the pertinent registers are displayed, consider this annotated conversation with KMCDA:

Command is input following KMCDA prompt, and the following is displayed:

```
                        INBUS REGISTERS
                                                  LU REGISTERS
↑$I
010 000 007 000 000 000 304 340 X 377 377 377 377 377 377 377 377
004 000 004 000 000 000 000 000 X 000 020
^
                                      NPR  μPMISC
                                  REGISTER  REGISTER
                        INBUS* REGISTERS
```

Note that the line unit (LU) registers are displayed as all ones (377, octal) when no LU is connected. This value represents the actual state of LU registers when read from a KMC11-based system not equipped with an optional LU.

6.1.4.3 **Examine Data Memory** - This command takes the following form:

```
n;mL
```

where

- n is an octal integer in the range 0 to 1777. It is the starting address of the series of sequential data memory locations to be displayed.
- m is an octal integer in the same range. It specifies the number of data memory locations whose contents are to be displayed. In actual practice, if the value of m is less than n (for example 100:40L), m is taken as the octal count of the number of data memory locations to be displayed. In this example, the contents of data memory locations from 100 to 137(octal) would be displayed. However, when the value

of m is greater than n (for example 100;220L), m is interpreted by KMCDA as the upper boundary of the data memory area to be displayed. In this example, the contents of 120 octal locations (from location 100 to 217) would be displayed.

The examine memory command has a characteristic whereby it displays memory locations in groups of 20 octal locations only. If the quantity specified in the command is not an even multiple of 20, this memory command characteristic will cause multiples of 20 locations to be displayed so that the specified quantity is included in the display. For example the command

```
150;10L
```

would display the contents of 20 octal locations starting at location 150 and ranging to location 167. Similarly, the command

```
150;21L
```

would display the contents of 40 octal locations starting at location 150 and going to location 207. In the same manner, the command

```
100;222L
```

would display the contents of 140 locations starting at location 100 and going to location 237. However, the command

```
100;100+120L
```

would display the contents of 120 locations starting at location 100 and going to location 217. Note the use of addition in the ending address field.

An example of a data memory display is shown below along with the command that initiated the display:

```
^0;50L
0000 : 000 000 000 000 000 000 000 003 * 000 173 117 241 342 136 000 000
0020 : 000 001 000 325 115 041 111 032 * 000 144 117 241 342 136 000 000
0040 : 000 004 000 275 115 001 005 001 * 000 175 117 241 342 136 000 000
^
```

Inspection of this display shows that the command specified the display of data memory locations 0 through 50 (octal). KMCDA actually displayed locations 0 through 57 or a total of 60 octal locations.

6.1.5 Utility Commands

Utility commands provide the various utility functions that are supportive to microprogram debugging activities.

These functions include the following:

1. Display a designated breakpoint.
2. Execute a designated microinstruction.
3. Load a predesignated value into the data memory location addressed by the current MAR.

4. Load a predesignated value into the MAR or display the current MAR.

5. Load a predesignated value into a prescribed scratch pad location or into the BRG.

6. Set all locations in the data memory to zero.

7. Calculate the branch offset from a prescribed CRAM location and configure the octal equivalent of the required unconditional Branch class instruction.

The formats for the commands that implement these utility functions are listed below.

6.1.5.1 **Display a Breakpoint** - With this command, the programmer can call the KMCDA utility that displays the CRAM address, the microinstruction at that address, and the outstanding proceed count, if any, for all active breakpoints.

The format for the command which calls the display breakpoint utility is as follows:

```
$B
```

KMCDA responds with

```
n:A I C
```

where

n is a breakpoint number 0 to 7

A is an octal integer in the range 0 to 3776. It is the CRAM address at which the breakpoint is exercised times two.

I is an octal integer that corresponds to the instruction at the breakpoint address A. If breakpoint n has been set without an intervening proceed command being executed, I will be displayed as all zeros or as the instruction at the location to which the breakpoint was previously assigned.

C is the proceed count for breakpoint n. When greater than zero it indicates an outstanding proceed count of C.

The commands to set and clear breakpoints are described in Sections 6.1.2.1 and 6.1.2.2 respectively. Detailed information on breakpoint handling is given in Section 6.2.

6.1.5.2 **Execute a Microinstruction** - Using this utility, the programmer can execute any single KMC11 microinstruction. If a Move class instruction is executed in this manner, the PC is incremented by one.

Similarly, if a Branch class instruction is executed by this utility, the PC will be incremented by one or changed to the address branched to. Whatever instruction is executed in this manner, the integrity of the instruction set stored in the CRAM is unaffected. However, the contents of data memory, the MAR, and any of the KMC11 addressable internal registers can be affected by execution of specific Move class instructions.

NOTE

If the microinstruction executed modifies the CSRs, the modification will be lost.

The command to invoke this utility takes the following form:

nE

where

n is an octal value which is equivalent to the binary form of the instruction to be executed.

For example:

10200E

causes 10200(octal) to be executed as the microinstruction

LDMA IMM,200

which moves the value 200 to the eight low-order (page offset) bits of the MAR.

Similarly,

63222E

causes the contents of the BRG to be moved to scratch pad register 2 as the microinstruction

SP BR,SELB,SP2

6.1.5.3 **Load Data Memory** - This utility permits the loading of data into any data memory location desired or to the location addressed by the current MAR. The command to invoke this utility takes the following form:

[n;]mW

where

[n;] is the optional octal value in the range 0 to 1777, which is the data memory address into which data is to be loaded.

m is an octal value in the range 0 to 377, which is the data to be loaded into the data memory location addressed by [n;]. If the optional qualifier [n;] is not supplied so that the command takes the form

mW

then the data, m, will be loaded into the data memory location addressed by the current MAR. Note that when [n;] is supplied, the state of the current MAR is restored by KMCDA by a proceed command.

6.1.5.4 **Load or Display the Memory Address Register (MAR)** - With this utility, the programmer can load a value in the MAR to address any one of the 1024 bytes in the data memory or display the current MAR. The command to load the MAR takes the following form:

nA

where

n is an octal integer in the range 0 to 1777, which is the address to be loaded into the MAR. For example,

1000A

sets the MAR to address the data memory decimal location 512.

The command to display the MAR takes the following form:

$A

6.1.5.5 **Load Scratch Pad or BRG** - Using this utility, a programmer can load a selected value in a specified scratch pad location or load a selected value in the BRG.

The command to load a selected value in a specified scratch pad location takes the following form:

n;mC

where

n is the address of the desired scratch pad location.

m is an integer in the range 0 to 377 (octal) to be loaded into SPn.

The command to load a selected value in the BRG takes the following form:

mC

where

m is an integer in the range 0 to 377 (octal) to be loaded into the BRG.

6.1.5.6 **Zero Data Memory** - This utility sets the contents of each data memory location to binary zero. The current content of the MAR remains unchanged. The command to invoke this utility takes the following form:

```
Z
```

6.1.5.7 **Calculate Offset** - With this utility, the programmer can use KMCDA to calculate the unconditional immediate branch instruction required to branch to a specified address. KMCDA will display the octal form of the unconditional branch to the location addressed by the calculated offset. The command to perform this function takes the following form:

```
nO
```

where

n is an even octal integer in the range 0 to 1776. It designates the microinstruction address as it appears on the MACRO-11 listing for which KMCDA is to calculate an offset. For example, typing,

```
4320
```

causes KMCDA to respond with

```
100615
```

which is the octal form of the KMC11 microinstruction

```
ALWAYS LABEL
```

where LABEL has a tag value of octal 432

NOTE

The tag value of 432 (octal) corresponds to a control RAM address of 215 (octal).

This utility is extremely useful for making on-line changes to existing program logic.

6.2 BREAKPOINT HANDLING

The use of breakpoints permits the programmer to temporarily halt microprogram execution at a predetermined CRAM location so that the pertinent internal registers can be examined. If a malfunction is discovered, the program can be modified and microprogram execution can be resumed and again halted at the same location for subsequent examination. The following sections present the requirements and techniques and constraints to be applied when performing breakpoint analysis.

6.2.1 Reserved CRAM Requirements

As previously indicated, KMCDA resides in the memory space of the associated PDP-11 and monitors the microprogram from the UNIBUS through the KMC11 CSRs. When a microprogram debugging operation involves the use of breakpoints, the last 16 CRAM locations must be set aside for use by a breakpoint routine, which implements up to eight breakpoints. The source microcode comprising this routine is as follows:

```
NBRKS   = 8.
.SAV=.
.=START+<<1024.-<NBRK$*2>>*2>
$NBRKS=0
.REPT   NBRKS
.IRP    $$NBRK,<\$NBRKS>
OUT     IMM,$$NBRK*20+1,OMAIN
ALWAYS  .
$NBRKS=$NBRKS+1
.ENDM
.ENDM
.=.SAV
```

NOTE

The file name for this routine is BRKPNT.MAC.

This routine must be assembled with the microprogram to be debugged when breakpoints are to be used.

6.2.2 Breakpoint Location Constraints

Since KMCDA operates external to the environment it monitors, information critical to microprogram execution, specifically the state of the KMC11 C-bit and Z-bit, is not available to KMCDA. Although the structure of KMCDA provides for careful maintenance of the KMC11 PC, and MAR, it cannot keep track of the states of the C-bit and Z-bit. On this basis then, breakpoints should never be set at a microprogram location containing a Branch class microinstruction that is conditional on the state of the C-bit or Z-bit.

If a breakpoint is inadvertently set at a Branch class microinstruction conditional on the state of the C-bit or Z-bit, proper execution of that instruction cannot be guaranteed. Therefore, setting of breakpoints at locations containing this type of instruction must be avoided.

6.2.3 Proceed Counter

When a Go (G) command with outstanding breakpoints or a proceed from breakpoint (nP) command is issued, KMCDA initiates execution of the microprogram and enters the wait for breakpoint mode. When the next outstanding breakpoint having a proceed count greater than zero is encountered, that counter is decremented, and microprogram execution is resumed. When an outstanding breakpoint is encountered having a proceed count equal to or less than zero, microinstruction execution is halted. KMCDA then displays the following message:

```
MBn:xxxxxx
```

where MB is the acronym for microbreak, n is the identity number of the breakpoint, and xxxxxx is an octal integer that is the CRAM address of that breakpoint location. KMCDA then enters the command decode mode.

At this juncture, the distinction between the nG and the nP commands with respect to breakpoints should be clarified. When an nG command is issued, microprogram execution is started at location n and proceeds until the next outstanding breakpoint is encountered. If the proceed counter for that breakpoint is greater than zero, it is decremented and microprogram execution continues until the next outstanding breakpoint is encountered. If the proceed count for the currently encountered breakpoint is less than or equal to zero, execution is halted at that breakpoint and KMCDA enters the command decode mode. Note that the nG command will not initiate execution from a breakpoint unless n is equal to the breakpoint address.

The purpose of the nP command, however, is specifically to proceed from a breakpoint and halt at that breakpoint after encountering it n times. In addition, an nP command will always initiate execution from the last encountered breakpoint even after execution of a series of single step commands.

If a G or a P command is executed and a breakpoint is not encountered, KMCDA can be reentered by placing all data switches on the PDP-11 console switch register in the up position. At that point KMCDA will display

```
FE
```

followed by

With the prompt symbol displayed KMC11 is halted at an indeterminate location. To continue operation from that location use the G command. If any commands except G or P are executed, the status of the C-bit and the Z-bit may be modified.

The PDP-11 must have a Console Switch Register (address 777570) in order for the debugger to use this feature. If this register is not on the PDP-11 being used, KMCDA must be reassembled with the RSX-11M system general configuration file RSXMC.MAC ensuring that the symbol S$$WRG is not defined. The command KMCDA=RSXMC.KMCDA is used to assemble the debugger.

6.3 EXAMPLE OF A KMCDA CONVERSATION

```
>INS KMCDA                          ;Install KMCDA as RSX11-M task
>RUN KMCLDR                         ;Run the microcode loader
KMC LOADER
CSR? 170                            ;Inform loader of KMC11-A CSR location
FILE NAME? COMIOPDUP                ;Assign file name
LOAD OR COMPARE? L                  ;Specify load operation
KMC LOAD COMPLETE
>KMC                                ;Run debugger
CSR?170                             ;Inform debugger of KMC11-A CSR address
*3764/000000 1121                   ;Enter microcode to implement breakpoints 5,6,and 7
003766 / 000000 114773
003770 / 000000 1141
003772 / 000000 114775
003774 / 000000 1161
003776 / 000000 114777
*$B                                 ;Display active breakpoints - none

*$I                                 ;Examine INBUS and INBUS* command
002 102 022 004 310 340 310 340 X 377 377 377 377 377 377 377 377
000 000 000 000 000 000 000 000 X 014 120
*$0/000000                          ;Examine CSR 0

2:000000                            ;LF to examine CSR 2

4:000000                            ;LF to examine CSR 4

6:000000                            ;LF to examine CSR 6
        ?                           ;LF to examine CSR 8 - no CSR 8
*$R                                 ;Display BRG and scratch pad registers
111 : 057 102 200 000 000 105 107 176 * 002 002 010 207 001 110 030 022
*;L                                 ;Contents of BRG and scratch pads 0 through 16
0000 : 000 000 300 340 002 206 001 000 * 000 240 102 000 000 000 000 340
*Z                                  ;Display octal 20 locations of data memory
*0;0+100L                          ;Contents of first octal 20 locations of data memory
0000 : 000 000 000 000 000 000 000 000 * 000 000 000 000 000 000 000 000
0020 : 000 000 000 000 000 000 000 000 * 000 000 000 000 000 000 000 000
0040 : 000 000 000 000 000 000 000 000 * 000 000 000 000 000 000 000 000
0060 : 000 000 000 000 000 000 000 000 * 000 000 000 000 000 000 000 000
*$A 001777                          ;Clear all locations in data memory
*B                                  ;Display contents of data memory from location 0 to
*530;5B                            ;Contents of data memory from 0 to 100 octal
*$B                                 ;Display contents of MAR
5:000530 000000 000000              ;Clear all breakpoints
                                    ;Set breakpoint 5 to CRAM location 530
*0G                                 ;Display breakpoints
                                    ;At location 530, content=0, proceed count=0

                                    ;Begin microprogram execution at location zero
```

```
RUN SNCFCC

KMC11-DUP11 THROUGHPUT TEST

TEST IS RUN ON 2-16 LINES IN A CRISS-CROSS PATTERN
USING BIT-STUFF PROTOCOL, VARIABLE MESSAGES.

ENTER NUMBER OF LINES TO TEST (2-16) 2
ENTER LENGTH OF EACH PASS IN MINUTES (1-99) 1
MB:000530
@$I
022 102 022 004 310 340 000 000 X 377 377 377 377 377 377 377 377
220 000 003 000 000 000 300 340 X 000 034
@P
MB5:000530
@$B
5:000530 137140 177777
@S
@S
@$R
003 : 120 002 201 000 000 057 107 176 * 002 002 001 055 000 110 316 022

@111C
@3;333C
@$R
111 : 120 002 201 333 000 057 107 176 * 002 002 001 055 000 110 316 022
@;L
0000 : 000 000 303 340 002 206 001 000 * 000 000 000 000 000 000 000 000
@$A 000055
@2A
@$A 000002
@222^
@4;44^
@;L
0000 : 000 000 222 340 044 206 001 000 * 000 000 000 000 000 000 000 000

@5300  100654

@100654E
@X
```

```
;Run KMC11-DUP11 throughput test

;Breakpoint 5 encountered, microprogram halts at 530
;Examine INBUS and INBUS* registers
;Contents of INBUS and INBUS* registers displayed

;Proceed from breakpoint 5 (location 530)
;Breakpoint 5 encounted, microprogram halts at 530
;Display breakpoint
;At location 530, content=137140, proceed count=1777
;Single step to next instruction (531)
;Single step to next instruction (532)
;Examine BRG and scratch pad registers
;Contents of BRG and scratch pad registers

;Load octal value 111 into BRG
;Load octal value 333 into SP3
;Examine BRG and scratch pad registers
;Contents of BRG and scratch pad = BRG=111, SP3=333
;Display data memory locations 0 to 20 octal
;Contents of data memory locations 0 to 20 octal
;Examine contents of MAR, MAR=55 octal
;Set MAR to address data memory location two
;MAR now addresses data memory location two
;Load octal value 222 into data memory location two
;Load octal value 44 into data memory location 4
;Display data memory locations 0 to 20 octal
;Contents of data memory locations 0 to 20, 2=222,
; 4=044
;Calculate offset; derive branch instruction to
;location 530
;Execute derived branch instruction
;Exit KMCDA
```

CHAPTER 7

SPECIAL PROGRAMMING CHARACTERISTICS

The KMC11-A option on a PDP-11 system can be viewed as a peripheral device as well as a second processor residing on the UNIBUS. Considering the latter view, that of the KMC11 as an auxiliary processor, complete with memory and interfaced to the UNIBUS, the KMC11 improves the performance of the PDP-11 system by performing time-consuming system functions in parallel with the PDP-11 CPU. The KMC11 has a 1024 16-bit word writeable control memory containing the microprogram that is loaded by the PDP-11 processor. Eight bytes of control and status registers (CSRs) and associated interrupt logic provide communication between the PDP-11 program and the KMC11 microprogram.

Since the KMC11 microprogram is writeable and can be changed whenever desired by the PDP-11, there are some programming characteristics that require special consideration. The information in this chapter delineates and describes those special programming characteristics.

A number of programming characteristics are common to a multiprocessor configuration, in this case, the parallel operation of a KMC11 and a PDP-11 main CPU. Most of the KMC11 programming characteristics described in this chapter are typical of the type of multiprocessor interactions and race conditions normally encountered. However, several conditions are unique to the KMC11; these conditions relate to internal CSR discipline and CSR bit settling times.

7.1 CSR DISCIPLINE

Control and Status Register (CSR) accessing discipline is critical to the proper operation of a KMC11/PDP-11 multiprocessor configuration. This discipline has three major aspects. First, the CSRs must be set to all zeros prior to initialing KMC11 microprogram operation. The second concerns the internal read-modify-write accessing of a KMC11 CSR by the microprogram relative to a collateral activity by the main CPU over the UNIBUS. The third involves a similar accessing of the same CSR by the main CPU over the UNIBUS relative to a collateral activity by the microprogram. To ensure reliable access of the KMC11 CSRs, the KMC11/PDP-11 CSR discipline should be designed to eliminate the possibility of simultaneous modification of a CSR by the PDP-11 program and microprogram.

7.1.1 Initializing the CSRs

As indicated in Section 2.3.1, the logic elements associated with specific KMC11 RAM CSR bits in BSEL1 are cleared when the PDP-11 program sets the MCLR (master clear) bit. However, this action has no

effect on the state of the CSR RAM bits associated with these logic elements or on the remaining RAM bits comprising the KMC11 CSRs (BSEL0, and BSEL2 to BSEL7).

Proper operation of a KMC11/PDP-11 multiprocessor configuration requires that the eight CSR bytes be initialized to some known state prior to run time. This initialization of the KMC11 CSRs can be done at startup time by the microprogram or the associated PDP-11 program. The programming entity chosen to perform this task depends on the specific configuration. In one method of initializing the KMC11 CSRs, the PDP-11 program performs the following actions:

1. Sets CSR1 bit 6 (MCLR).
2. Initializes CSRs 0 and 2 through 6.
3. Sets CSR1 bit 7 (RUN).

7.1.2 Microprogram Modification of CSRs

It is possible for the microprogram to overwrite the data transferred over the UNIBUS to a KMC11 CSR resulting in the loss of the transferred data and disruption of multiprocessor interaction. As an example of this situation, consider this step-by-step description of a CSR read-modify-write by the microprogram:

Step 1. BSEL0(CSR0) = 40

Step 2. Microprogram moves contents of BSEL0 to SP0.

Step 3. Microprogram executes a write immediate to BRG of 200.

Step 4. Microprogram ORs contents of BRG with SP0 and writes the result into BSEL0.

At this point BSEL0 contains the octal value 240. If between Steps 1 and 4 the PDP-11 program tries to transfer data over the UNIBUS to the same CSR (for example BISB #100, BSELO), the transferred data will be lost. Preventing such an overwrite and subsequent loss of data is delineated in Appendix A, Section A.1.

7.1.3 UNIBUS Modification of the CSRs

In a manner similar to a microprogram overwrite of data transferred to a CSR over the UNIBUS, it is possible for data transferred to a CSR by the microprogram to be overwritten from the UNIBUS. As an example of this situation, consider this step-by-step description of a CSR read-modify-write from the UNIBUS:

Step 1. BSEL0 (CSR0) = 40

Step 2. The PDP-11 program executes the instruction BISB #100, BSEL0. This instruction is executed in three steps.

Step 2A. The contents of BSEL0 are transferred to a PDP-11 internal register.

Step 2B. The octal value 100 is ORed with the contents of the internal register.

Step 2C. The results of Step 2B are written back into BSEL0.

Step 3. The final content of BSEL0 is 140 (octal).

If the microprogram attempts a write operation between Steps 2A and 2C, the results of the microprogram write to BSEL0 will be lost. Preventing such an overwrite and subsequent loss of data is delineated in Appendix A, Section A.1.

7.2 MULTIPORT RAM LOCKOUT

As shown in Section 2.1.1 of this manual, the multiport RAM is a 16-byte memory that serves as the UNIBUS interface between the KMC11 and associated PDP-11 for CSRs and NPR transactions. The multiport RAM has a single write port that can be written into by both the KMC11 microprogram and the associated PDP-11 program. Since the microprogram and the PDP-11 program can write to the same port (port A), lockouts on the ports can occur. That is, the microprogram is inhibited from writing into any multiport RAM location while the UNIBUS is writing into or reading from a CSR. In addition, the microprogram is inhibited from reading from or writing into any multiport RAM location during the portion of an NPR transaction cycle in which the KMC11 is bus master.

The KMC11 is designed to automatically suspend microprogram execution while either of these conditions is occurring. These lockouts affect microprogram timing only and are otherwise transparent to the microprogram. The actual time periods involved are a function of UNIBUS timing and the associated PDP-11 as well as system memory and the peripheral devices that the KMC11 communicates with in a given configuration. If microprogram execution speed is critical, the microprogram should be written to minimize the possibility of these lockouts.

7.3 CSR BIT SETTLING TIME

The settling for the memory bits comprising the multiport RAM varies from bit-to-bit over a narrow range due to the nature of the high-speed semiconductor circuits used and the environmental conditions that the circuits operate under. Consequently, if the microprogram attempts to read a CSR immediately after the PDP-11 program has written into that same CSR from the UNIBUS, the data read by the microprogram could be incomplete. As an example of this condition, consider the following step-by-step sequence:

Step 1. BSEL0 = 0

Step 2. The PDP-11 program executes the instruction MOVB #377, BSEL0.

Step 3. BSEL0 now contains the octal value 377.

If the microprogram reads BSEL0 prior to Step 2, the correct value (0) is read; and, if the microprogram reads BSEL0 after Step 3, the correct value (377) is read.

If, however, the microprogram reads BSEL0 during Step 2, the result read can be any one of the 256 combinations of the eight BSEL0 bits and could likely be an incorrect value. An incorrect value will be

seen only when the microprogram reads a CSR within 20 ns after data was stored in that CSR from the UNIBUS.

NOTE

The converse of the above condition (i.e., the PDP-11 reads the CSR when the microprogram writes that CSR) is not of concern because the UNIBUS is locked out during that situation. Refer to Section 7.2 for further details.

Any problems attending the above condition can be eliminated in a KMC11 microprogram by making sure that multibit flag fields are not referenced in a single microinstruction. A method for eliminating the possibility of erroneous results due to differential bit settling rates is delineated by the annotated program example shown in Appendix A, Section A.2.

7.4 µPMISC AND NPR REGISTER CONSTRAINTS

The µPMISC and NPR control registers contain various function and control bits necessary for proper, efficient operation of the KMC11 microprocessor. If during a given read-modify-write operation, the state of certain of these bits change, redundant functions or undesired timeouts will occur or bus requests will be vectored to the incorrect location. An example is given below as to how these problems can occur unless precautions are taken.

To avoid the possibility of a change of state of these µPMISC register bits during a microprogram read-modify-write operation, a zero must always be written to the µPMISC register BUS RQ, ACLO, and PGM CLK bits and the NPR control register NPR RQ bit regardless of their prior states -- unless the function performed by one of these bits is required. If that function is required, the bit performing it should be set to one.

During an in-NPR transaction (i.e., NPR control register bits 0 and 4, NPR RQ and OUT-NPR, equal one and zero, respectively), the state of NPR control register bits 2, 3, and 7 (INBA 17, INBA 18, and BYTE XFER) should not be changed. Bits INBA 17 and INBA 18 are the high-order bits of the UNIBUS address for an in-NPR transaction and their states must be maintained during the transaction. The BYTE XFER bit applies only to an out-NPR, but its state should not be changed during an in-NPR transaction because microprogram operation would be adversely affected.

During an out-NPR transaction (i.e., NPR control register bits 0 and 4, NPR RQ and OUT NPR, both are equal to one), the state of NPR control register bit 7 (BYTE XFER) and µPMISC register bits 2 and 3 (OUT BA 17 and OUT BA 18) should not be changed. Bits OUT BA 17 and OUT BA 18 are the high-order bits of the UNIBUS address for an out-NPR transaction and their states must be maintained during the transaction. The BYTE XFER bit must not be changed during either an in-NPR or out-NPR transaction; during an out-NPR transaction, if BYTE XFER equals zero, transactions will be on word boundaries and if BYTE XFER equals one, transactions will be by bytes. Complete details relative to out-NPR byte transactions are contained in Chapter 2.

To allow for valid monitoring of the µPMISC register NON EX MEM bit, the µPMISC register should never be written into during an NPR transaction; i.e., when NPR control register bit 0 (NPR RQ) equals 1. If a series of NPR transactions is to be executed, the state of the NON EX MEM bit should either be preserved or checked after the completion of each NPR transaction.

When the microprogram writes to the NPR control register, it must always write a zero to the NPR RQ bit unless an NPR transaction is to be initiated.

To avoid the possibility of a bus request being vectored to the wrong location, the µPMISC register VECTOR @ XX4 bit must not be changed if a bus request is pending; i.e., BUS RQ set to one.

To avoid a spurious bus request or an undesired program timeout or programmed pseudo-power failure at the CPU, a zero must always be written to the µPMISC register BUS RQ, PGM CLK, and ACLO bits.

Suppose, for example, the following events occur:

1. BUS RQ = 1.
2. Microprogram reads µPMISC register.
3. Microprogram ORs in OUTBA 16:17.
4. Microprogram writes a change to the µPMISC register (BUS RQ is set to one).

If BUS RQ is automatically cleared as a result of completion of a bus request between Steps 1 and 4, a spurious bus request will occur. The following example microinstructions would prevent a spurious bus request, as described above, from occurring:

```
SP        IBUS,UBBR,SP0        ;read µPMISC register
BRWRTE    IMM,101              ;mask to save
                               ;NXM and xx4,
                               ;clear all others and
                               ;store results in SP0
SP        BR,AANDB,SP0         ;perform the clear
BRWRTE    IMM,14               ;mask to set OUTBA 16:17
OUT       BR,AORB,OBR          ;OR mask with contents
                               ;of SP0 and write into
                               ;µPMISC register
```

Similar microinstructions apply to the NPR control register NPR RQ bit.

In summary, when accessing the µPMISC register, a zero must be written to the BUS RQ, PGM CLK, and ACLO bits, unless the function performed by that bit is specifically required; when accessing the NPR control register, a zero must be written to the NPR RQ bit, unless that function is specifically required. When writing the µPMISC register, the VECTOR @ XX4 bit must not be changed while a bus request is in progress.

APPENDIX A

SPECIAL PROGRAMMING TECHNIQUES

A.1 PREVENTING LOSS OF DATA BY OVERWRITING WHEN THE MICROPROGRAM OR THE PDP-11 MODIFIES THE CSRs

The possibility of overwriting data when the microprogram or the PDP-11 modifies the CSRs is eliminated by observing the following conditions for reading and writing the BSEL0 and BSEL2 status and control bits (Figure A-1) and by using the CSR protocols in the example program in this Appendix.

BSEL0

RQI — This bit is written by the PDP-11 program and read by the microprogram. When set, it indicates that the PDP-11 is ready for and requesting use of BSEL3, SEL4 and SEL6 for data transfer to the microprocessor.

IEO — This bit is set by the PDP-11 program to indicate that an output interrupt is enabled. (RDYO sets.)

IEI — This bit set by the PDP-11 program to indicate that an input interrupt is enabled. (RDYI sets.)

BSEL2

RDYO — This bit is read and written by the microprogram and is also read and written by the PDP-11, but is written by the PDP-11 program only when the microprogram is not writing it. When set, indicates that the microprocessor has data to give to the PDP-11. This bit is cleared by the PDP-11 program.

RDYI — This bit is set by the microprogram upon finding RQI set to indicate that the requested CSRs are free. It is cleared by the PDP-11 program to indicate that the PDP-11 has completed and input and the CSRs can be read by the microprogram.

NOTE

RDYO and RDYI are set mutually exclusive. BSEL0 is never written by the microprocessor and BSEL2 has been arranged such that at any state only the microprocessor or the PDP-11 would be writing it.

COMMAND TYPE CODE
These three bits indicate the type of information being passed in the other CSRs. They are set by the PDP-11 on RDYI transfers and by the microprocessor on RDYO transfers.

Complete details for implementing PDP-11 code to prevent overwriting of data when either the PDP-11 or the microprogram modifies the CSRs are contained in the COMM IOP-DUP Programming Manual, AA-5670A-TC, and the COMM IOP-DZ Programming Manual, AA-5127A-TC.

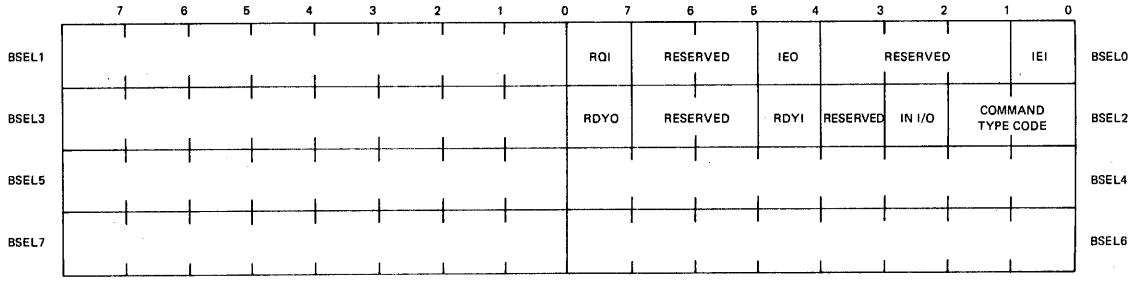


Figure A-1 Suggested Format for UNIBUS CSRs

A.2 ENSURING THAT CSR BITS HAVE SETTLED

The following example program procedures demonstrate the way to ensure that CSR bits have settled when the PDP-11 writes and the microprogram subsequently reads a CSR. Note especially the annotation A2 for bit settling time assurance techniques. Note at annotation A1 that RDYO is set as the second to last step in setting up the CSRs or posting a completion to the PDP-11. The last step is generating the interrupt, if requested.

```
        .SBTTL  IDLE - IDLE LOOP
IDLE:   SPBR    IBUS,UBBR,SP0   ;READ THE BUS REQUEST REGISTER AND
                                ;STORE THE IMAGE IN SP0 AND THE BRG
        BR4     TIMER           ;BRANCH IF THE TIMER HAS EXPIRED
IDLE1:  LDMA    IMM,P,PORT      ;LOAD MAR TO POINT TO PORT STATUS
        LDMAP   IMM,P,PORT      ;LOAD MAR HIGH
        .ALWAY  MEMX,SELB,0     ;TIMER HAS NOT EXPIRED YET, CHECK THE
                                ;DATA PORT TO SEE IF ANY PROCESSING IS
                                ; REQUIRED RAM CONTAINS THE ADDRESS OF
                                ;THE APPROPRIATE SERVICE ROUTINE
                                ; RQISET ==> WAITING FOR RQI TO SET
                                ; RDICLR ==> WAITING FOR RDYI TO CLEAR
                                ; RDOSET ==> WAITING FOR A COMPLETION
                                ; RDOCLR ==> WAITING FOR RDYO TO CLEAR

        .SBTTL  DATA PORT PROCESSING ROUTINES
;+
; **RQOCLR-WAITING FOR READY OUT TO BE CLEARED BY THE PDP-11**
;
; INPUTS:
;       MAR = PORT STATUS WORD (P,PORT)
; OUTPUTS:
;       IF READY OUT IS FOUND TO BE CLEARED THE INPUT CONTROL CSR IS
;       EXAMINED NEXT TO SEE IF THERE ARE ANY PENDING INPUT REQUESTS
;       FROM THE PDP-11.  IF NOT, THE COMPLETION QUEUE IS CHECKED FOR
;       ANY PENDING DONES.
;
;       THIS ROUTINE ALSO CHECKS IF INTERRUPT ENABLE HAS BEEN SET IF
;       IT WAS NOT SET WHEN RDYI WAS.
;
;       NOTE: THERE EXISTS THE POSSIBILITY OF AN INTERRUPT BEING GENERATED
;       IF IEO IS CLEARED AT ANY TIME AFTER IT IS TESTED BY THE MICRO-
;       PROCESSOR (APPROX A 1.5US WINDOW)
;-
RDOCLR:
        BRWRTE  IBUS,INCON              ;READ INPUT CONTROL CSR
        BR4     RDOST2                  ;USER SET INTERRUPT ENABLE OUTPUT
;
; ENTER HERE IF AN OUTPUT INTERRUPT HAS ALREADY BEEN GENERATED
;
RDOCL1:
        BRWRTE  IBUS,OCON               ;READ OUTPUT CONTROL CSR
        BR7     IDLE                    ;READY OUT STILL SET
        BRWRTE  IMM,0                   ;CLEAR OUTPT CONTROL CSR
        OUT     BR,SELB,OOCON           ;..
        MEMADR  RQISET                  ;LOOK FOR RQI NEXT
        ALWAYS  IDLE                    ;BACK TO IDLE LOOP

;+
; **RDOSET-MICROPROCESSOR COMPLETION POSTING**
;
; INPUTS:
;       MAR = PORT STATUS WORD (P,PORT)
; OUTPUTS:
;       CHECK THE COMPLETION SILO TO SEE IF ANY COMPLETIONS ARE PENDING.
;       IF THERE ARE POST THE COMPLETION TO THE PDP-11 OTHERWISE
;       CHECK TO SEE IF THE PDP-11 HAS ANY INPUT DATA
;-
RDOSET:
        MEMADR  RQISET                  ;LOOK FOR RQI NEXT
        LDMA    IMM,P,SLOT      ;SET MAR TO COMPLETION SILO NEXT OUT POINTER
        LDMA    MEMX,SELB       ;POINT COMPLETION TO NEXT OUT ENTRY
        Z       IDLE            ;THE POINTER IS ZERO THEREFORE THE SILO
                                ;IS EMPTY
```

```
; A COMPLETION OUTPUT IS PENDING IN THE COMPLETION SILO. MAR POINTS TO
; THE NEXT ENTRY
        OUT     MEMI,SELB,OLINEN ;WRITE THE LINE NUMBER BYTE

; READ THE SECOND WORD OF THE COMPLETION SILO AND SET UP CSR 4

        OUT     MEMI,SELB,OPORT1 ;WRITE PORT BYTE 1
        OUT     MEMI,SELB,OPORT2 ;AND PORT BYTE 2

; READ THE THIRD WORD OF THE COMPLETION SILO AND SET UP CSR 6

        OUT     MEMI,SELB,OPORT3 ;WRITE PORT BYTE 3
        OUT     MEMI,SELB,OPORT4 ;AND PORT BYTE 4

        OUT     MEMI,SELB,OOCON ;WRITE THE NEW OUTPUT CNTRL CSR          (A1)

; INCREMENT THE NEXT OUT POINTER

        LDMA    IMM,P.SLIN      ;SET MAR TO POINT TO COMPLETION SILO
                                ;NEXT IN OFFSET
        SP      MEMI,SELB,SP1   ;SAVE THE NEXT IN POINTER IN SP1
        SP      MEMX,SELB,SP0   ;SAVE THE NEXT OUT POINTER IN SP0
        MEM     IMM,P.NPR       ;ASSUME THE SILO IS GOING TO WRAP AROUND
        BRWRTE  IMM,SILOED      ;OFFSET TO LAST SILO ENTRY
        COMP    BR,SP0          ;COMPARE CURRENT OUT POINTER WITH END
                                ;OF SILO
        Z       50$             ;IT WRAPPED AROUND - ALREADY SET UP
        BRWRTE  IMM,SENTRY      ;GET THE SIZE OF A SILO ENTRY
        MEM     BR,ADD,SP0      ;INCREMENT NEXT OUT POINTER AND SAVE IT

; IF SILO IS NOW EMPTY "ZERO" THE NEXT OUT POINTER

50$:    COMP    MEMX,SP1        ;COMPARE OUT POINTER TO IN POINTER
        Z       70$             ;THEY ARE THE SAME
        ALWAYS  80$             ;THEY ARE DIFFERENT
70$:    MEM     IMM,377         ;SILO IS EMPTY - SET NEXT OUT POINTER
                                ;TO A LOGICAL ZERO (-1)
80$:    LDMA    IMM,P.PORT      ;SET MAR TO POINT TO PORT STATUS
        SPBR    IBUS,INCON,SP0  ;READ INPUT CONTROL CSR
        BR4     RDOST2          ;OUTPUT INTERRUPT REQUESTED
        MEMADR  RDOCLR          ;STATE TO WAITING FOR READY OUT CLEARING
        ALWAYS  IDLE            ;BACK TO IDLE LOOP
RDOST2: MEMADR  RDOCL1          ;STATE TO WAITING FOR READY OUT CLEARING
        BRWRTE  IMM,300         ;MASK FOR BUS REQUEST AND XX4
RDOST3: OUT     BR,SELB,OBR     ;GENERATE AN INTERRUPT
        ALWAYS  IDLE            ;BACK TO IDLE LOOP

;+
; **RQISET-PROCESS INPUT FROM THE PDP-11**
;
; INPUTS:
;       MAR = PORT STATUS WORD (P,PORT)
; OUTPUTS:
;       CHECK TO SEE IF REQUEST IN HAS BEEN SET BY THE PDP-11. IF SO,
;       SET READY IN AND SET THE PORT STATUS TO WAIT FOR THE PDP-11 TO
;       CLEAR REQUEST IN.
;-
RQISET: BRWRTE  IBUS,INCON      ;READ INPUT CONTROL CSR
        BR7     10$             ;REQUEST IN SET
        ALWAYS  RDOSET          ;SEE IF ANY COMPLETIONS TO POST
10$:    SP      IMM,20,SP0      ;MASK TO SET READY IN
        OUT     SELA,OOCON      ;SET IN OUPUT CONTROL CSR
        BR0     RQIST1          ;INTERRUPT ENABLE IS SET
        MEMADR  RDICLR          ;STATE TO WAITING FOR RDYI TO CLEAR
        ALWAYS  IDLE            ;BACK TO IDLE LOOP
RQIST1: MEMADR  RDICL1          ;STATE TO WAITING FOR RDYI TO CLEAR
        BRWRTE  IMM,200         ;MASK FOR BUS REQUEST AND XX0
        ALWAYS  RDOST3          ;GENERATE AN INTERRUPT
```

```
;+
; **RDICLR=PDP=11 HAS CLEARED READY IN (DATA PORTS HAVE BEEN SET UP)**
;
; INPUTS:
;       MAR = PORT STATUS WORD (P,PORT)
; OUTPUTS:
;       CHECK TO SEE IF THE PDP=11 HAS CLEARED READY IN SIGNIFYING
;       THAT IT HAS SET UP THE DATA PORTS.  IF SO THEN DISPATCH TO THE
;       PROPER ROUTINE TO HANDLE THE REQUEST BASED ON
;       BIT 0&1 OF THE OUTPUT CONTROL CSR
;-

; RDYI CLEAR ROUTINE IS ENTERED HERE IF INTERRUPT ENABLE WAS NOT
; SET WHEN THE COMMIOP SET READY IN. IF IN THE MEANTIME INTERRUPT ENABLE
; WAS SET, IT WILL BE SEEN HERE AND AN INTERRUPT WILL BE GENERATED.

        .ENABL  LSB
RDICLR:
        BRWRTE  IBUS,OCON       ;READ OUTPUT CONTROL CSR
        BR4     5$              ;READY IN STILL SET
        ALWAYS  10$             ;PDP=11 CLEARED RDYI, DONT BOTHER
                                ;CHECKING FOR IEI JUST PROCESS THE DATA
5$:     BRWRTE  IBUS,INCON      ;READ INPUT CONTROL CSR
        BR0     RQIST1          ;INTERRUPT REQUESTED
RDICL1:
        BRWRTE  IBUS,OCON       ;READ OUTPUT CONTROL CSR                    (A2)
        BR4     IDLE            ;RDYI STILL SET
10$:    MEMADR  RDOSET          ;STATE TO WAIT FOR COMPLETIONS

; READY IN CLEAR

        BRWRTE  IMM,P,LADR      ;GET ADDRESS OF LINE TABLE ADDR TABLE
        SP      BR,SELB,SP5     ;SAVE IT IN SP5
        SP      IBUS,LINENM,SP,LN       ;READ THE LINE NUMBER
        BRWRTE  TWOA,SP,LN      ;MULTIPY IT BY TWO
        LDMA    BR,ADD,SP5      ;POINT TO ENTRY IN TABLE FOR THIS LINE
        SP      MEMI,SELB,SP,RM0  ;GET THE ADDRESS OF THIS LINE'S TABLE
                                ;(LOW) AND SAVE IT IN SP,RM0
        SP      MEMX,SELB,SP,RM1,LDMAPG ;GET THE ADDRESS (HIGH) AND
                                ;POINT THE MAR HI TO THIS ADDRESS
        LDMA    SELA,SP,RM0     ;SET MAR LOW
15$:
        BRWRTE  IMM,14,INCMAR   ;SET UP DUP CSR BY SETTING ADDR BITS 16=17
        OUT     BR,SELB,OBR,INCMAR ;WRITE EXTENDED MEM BITS FOR OUT NPR
        OUTPUT  MEMI,SELB,OBA1  ;WRITE OUT LOW BYTE OF CSR ADDRESS
        OUTPUT  MEMI,SELB,OBA2  ;WRITE OUT HIGH BYTE
        LDMA    SELA,SP,RM0     ;POINT BACK TO START OF LINE TABLE
        BRWRTE  IBUS,OCON       ;GET ORIGINAL IMAGE OF INPUT CONTROL CSR    (A2)
                                ;READ FROM BSEL2 TO AVOID RAM BIT SET
                                ;PROBLEM.  BY THIS TIME ALL BITS WILL
                                ;HAVE SETTLED
        BR1     20$             ;BIT 1 SET
        BR0     CONIN           ;BIT 1=0 AND BIT 0=1 => CONTROL IN REQUEST
        ALWAYS  BAIN            ;BIT 0&1=0 => BUFFER ADDRESS IN REQUEST
20$:    BR0     BASEIN          ;BIT 1=1 BIT 0=1 => BASE IN REQUEST
                                ;BIT 1=1 BIT 0=0 => ILLEGAL REQUEST

; ABOVE THREE ROUTINES RETURN HERE AFTER REQUEST HAS BEEN PROCESSED

RQICL2: BRWRTE  IMM,0           ;CLEAR OUT CONTROL CSR
        OUT     BR,SELB,OOCON   ;..
        SPBR    IBUS,UBBR,SP0   ;READ THE MISC REGISTER
        BR0     NXMERR          ;IF BIT 0 SET, THEN A NON=EXISTENT
                                ;MEMORY ERROR OCCURRED, REPORT IT.
        ALWAYS  IDLE            ;OTHERWISE, BACK TO IDLE LOOP
```

(A1) RDYO (Ready Out) is set after the other CSRs have been setup.

(A2) In case Ready In was cleared when the I/O bits were set, re-reading BSEL2 will assure that all I/O bits have settled.

INDEX

KMC11 Programmer's
Manual
AA-5244B-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754