C00-3288-33

HOPKINS COMPUTER RESEARCH REPORTS

REPORT 33

SEPTEMBER 1974

EVALUATION OF JHU MICROMACHINE EMULATION OF THE PDP-11

BY

STEPHEN BRADLEY, JR.

AND

CHARLES NEUHAUSER

MASTER

RESEARCH PROGRAM IN COMPUTER SYSTEMS ARCHITECTURE

COMPUTER SCIENCE PROGRAM

THE JOHNS HOPKINS UNIVERSITY

BALTIMORE, MARYLAND

EVALUATION OF JHU MICROMACHINE EMULATION OF THE PDP-11*

ABSTRACT

A skeleton emulation of a DEC PDP-11 series machine on the JHU micromachine (see Report #28.1) is given and evaluated. The report contains the following sections:

1) Short description of the microassembly language
2) Skeleton emulation of the PDP-11
3) Description of emulator and techniques used
4) Timing estimate and evaluation of micromachine performance

This emulation indicates that the instruction execution rate of the emulated machine is about one-half to one-third that of a PDP-11/20. Architectural improvements are recommended which will allow the emulated machine to execute PDP-11/20 code in approximately real time.

Stephen Bradley Jr.
Charles Neuhauser
Electrical Engineering Department
The Johns Hopkins University
Baltimore, Maryland 21218

*This work was supported by the U.S. Atomic Energy Commission under Contract AT (11-1 3288).

CONTENTS

I. COMMENTS ON AND EVALUATION OF PDP-11 EMULATION

1. Introduction

This study was undertaken to evaluate the effectiveness of the JHU micromachine [1] as a host machine for the emulation of conventional machines. We chose the PDP-11 as a benchmark because this machine is powerful architecturally and has proven difficult to emulate on existing host machines. Only a skeleton emulation is given, but all memory accessing mode operations are fully coded.

The following assumptions concerning the host machine were used in the construction of the PDP-11 emulator:

1) When a microinstruction consists of both an A- and T-machine sub-instruction we assume that both the A- and T-machines execute simultaneously, and that data modifications by one machine are not available to the other until the following cycle.

2) It follows from assumption 1) above that the A-machine may not branch conditionally upon T-machine results until the following instruction cycle.

3) Main memory operates as follows:

   a) One cycle must intervene between a request for main memory data and its usage.

   b) Two cycles must intervene between subsequent accesses to main memory.

   c) On main memory accesses registers containing address and data to be sent (on writes) are stored internally and may be changed in the following cycle.

4) Main memory accesses were assumed to be full word, that is 32 bits in width.

2. Timing Estimate

In order to estimate emulator timing the following PDP-11 instructions have been coded: ADD, INC, ASR, MUL, MOV, and BEQ. Figure 1 shows the flow of control through the the major microcode routines used by the emulator. For each routine and sub-routine the number of memory references are indicated. The following notation is used:

I	- number of microinstruction fetches from control memory,
D	- number of data fetches from control memory, and
E	- number of references to external memory.

In some cases a routine issues a call to one of three sub-routines (WADCON, WRITES, WRITEW) which handle the accessing of main memory.

Where these sub-routines are used by a main emulator routine the I, D, and E counts are included in the I, D, and E counts of the calling routine, and only the number of sub-routine calls is indicated.

In general the timing for a given instruction may be calculated by summing the contributions of the following microcoded routines:

1) IFETCH
2) DECODE (0 to 3 applications of TBLEn as required)
3) SMODE and subsequent execution of SMODn
4) DMODRD or DMODWT and subsequent execution of DMODnR or DMODnW
5) OP (emulation of instruction data transformation)

Timing in the actual PDP-11 consists of summing:

1) Base timing of the instruction (includes the instruction fetch)
2) Source mode fetch timing
3) Destination fetch timing (actually a read-modify-write cycle)

Below is a tabulation of the PDP-11/20 cycle times and the corresponding number of host machine control store accesses (I and D). Base timing estimates for the emulator include time for IFETCH, DECODE, mode selection, and data transformation.

BASE TIMING

Ins.	PDP	EMULATOR		
	usec	I	D	
ADD	2.3	35	12	
INC	2.3	41	16	
ASR	2.3	38	13	
MOV	2.3	25	12	
BEQ	2.6	31	12	
MUL	8.9	57	15	(PDP-11/45 timing estimate)

SOURCE MODE TIMING

Mode	PDP	EMULATOR	
	usec	I	D
0	0.0	1	1
1	1.5	10	1
2	1.5	11	2
3	1.5	12	1
4	2.7	20	2
5	2.7	20	2
6	2.7	20	2
7	3.9	32	3

DESTINATION MODE TIMING

Mode	PDP usec	EMULATOR I	EMULATOR D
0	0.0	1	1
1	1.4	10	1
2	1.4	11	2
3	1.4	12	2
4	2.6	20	2
5	2.6	20	2
6	2.6	20	3
7	3.8	29	3

As an example, we use the above table to calculate the time of a common PDP-11 instruction ADD in which both source and destination are mode 6 (Indexed).

PDP-11/20 time: 2.3 + 2.7 +2.6 = 7.6 usec

Emulator time: 35 + 20 +20 = 75 I-fetch cycles and
12 + 2 + 3 = 17 D-fetch cycles

3. Evaluation of Host Machine Performance

3.1 Timing

Major time expenditures in emulator operation are as follows in decreasing order:

1) Resource matching,
2) Accesses to control store for data,
3) Decoding, and
4) Condition code setting.

Matching of emulator resources to target machine needs consumes the greatest portion of emulator time in the PDP-11 emulator. This problem is particularly accute with respect to the acessing of main memory. Routines concerned with accessing main memory spend most of their time translating a PDP-11 address into the appropriate host machine address and extracting or inserting the 16 bit (or 8 bit) data word referenced by the emulator into the 32 bit host machine word. Approximately two thirds of the emulator's cycle time is concerned with this sort of manipulation. A considerable savings in time could be realized by providing an address mapping unit to translate between target and emulator resource requirements. Such a unit could be located either on the host bus or within the main memory. To be most effective the main memory should also be operated in a read-modify-write cycle to eliminate memory transfers to the host machine when only a write is required.

Assuming that emulator time is directly related to control store accesses, we see that about 20% of emulator time is devoted to data accesses to control store (as opposed to instruction fetches). A possible solution to this problem would involve 'banking' of control store to allow simultaneous assess to control store for instruction and data fetches.

Depending on the target machine instruction being executed between 5% and 20% of emulator time is spent in the decoding process. However, it is not clear that architectural changes could significantly reduce this expenditure of time .

Evaluation and setting of condition codes consumes about 10% of emulator timing. The introduction of short bit setting and unsetting instructions into the T-machine could save some time here. In general, however, it seems better to approach this problem by deferring the processing of target machine condition codes as long as possible. This is done by storing the host machine codes, data transformation results and possibly instruction type on each target machine cycle. When target machine code requires a decision based on condition codes the stored target machine 'state' is translated properly to direct the decision.

3.2 Coding

Initial inspection of the host machine code leads to the following conclusions:

1) Long instructions (except insert and extract) are rarely used, and might be profitably eliminated.

2) Compare instructions are rarely used and might be eliminated, being replaced by a SUB-TEST sequence.

3) Host machine condition codes need to be set only on ALU type operations.

4) There is an arithmetic mismatch between the 32 bit arithmetic unit and the 16 bit operands of the PDP-11. Addition of a sign-extend instruction to the instruction set would help reduce the space-time penalties imposed by thei situation. It might be argued that all operands could be kept permanently left justified, but the addresses in the PDP-11 general registers must be kept right justified for convenience in address arithmetic.

5) Manipulation of condition codes is difficult. An instruction in the T-machine to specifically load the condition codes would be useful.

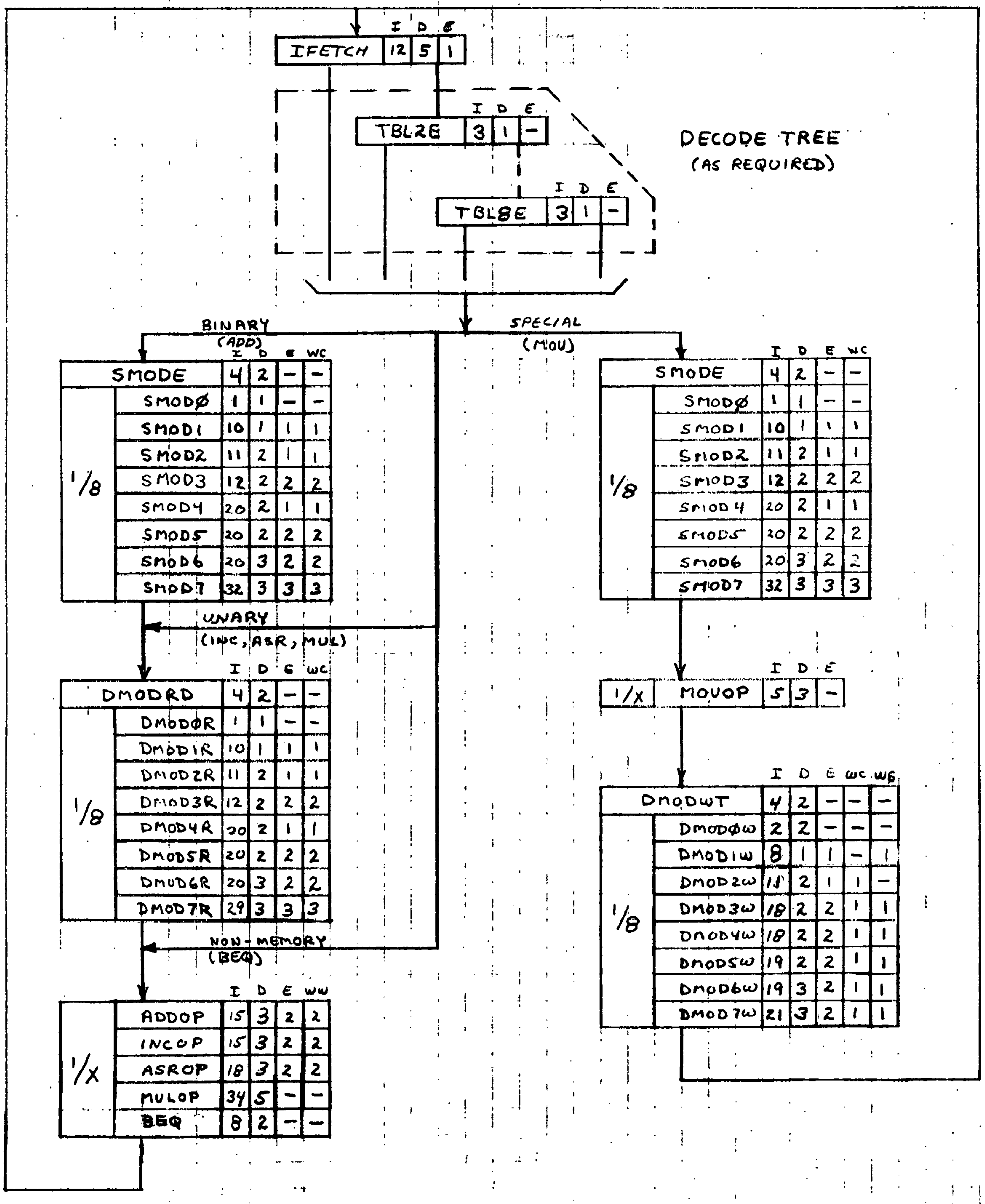
6) Although it was found that one of the best ways to save cycles is to duplicate code, the entire emulator should easily fit into the available 4K of micromemory.

4. Conclusion

Assuming a 200 nsec control memory access time for the host machine, the emulator describe here exhibits an instruction execution rate of between one-half and one-third that of a PDP-11/20. Reorganizing main memory so that it may be matched efficiently to the target machine should improve emulator performance greatly, and real time emulation of the PDP-11/20 should be possible.

5. References

[1] C. Neuhauser; "An Emulation Oriented, Dynamic Microprogrammable Processor (Version II)"; Hopkins Computer Research Report #28.1



	I	D	E
IFETCH	12	5	1
TBL2E	3	1	-
TBL8E	3	1	-

BINARY (ADD)

	I	D	E	WC
SMODE	4	2	-	-
SMODØ	1	1	-	-
SMOD1	10	1	1	1
SMOD2	11	2	1	1
SMOD3	12	2	2	2
SMOD4	20	2	1	1
SMOD5	20	2	2	2
SMOD6	20	3	2	2
SMOD7	32	3	3	3

UNARY (INC, ASR, MUL)

	I	D	E	WC
DMODRD	4	2	-	-
DMODØR	1	1	-	-
DMOD1R	10	1	1	1
DMOD2R	11	2	1	1
DMOD3R	12	2	2	2
DMOD4R	20	2	1	1
DMOD5R	20	2	2	2
DMOD6R	20	3	2	2
DMOD7R	29	3	3	3

NON-MEMORY (BEQ)

	I	D	E	WW
ADDOP	15	3	2	2
INCOP	15	3	2	2
ASROP	18	3	2	2
MULOP	34	5	-	-
BEQ	8	2	-	-

SPECIAL (MOV)

	I	D	E	WC
SMODE	4	2	-	-
SMODØ	1	1	-	-
SMOD1	10	1	1	1
SMOD2	11	2	1	1
SMOD3	12	2	2	2
SMOD4	20	2	1	1
SMOD5	20	2	2	2
SMOD6	20	3	2	2
SMOD7	32	3	3	3

	I	D	E
1/X MOVOP	5	3	-

	I	D	E	WC	WS
DMODWT	4	2	-	-	-
DMODØW	2	2	-	-	-
DMOD1W	8	1	1	-	1
DMOD2W	18	2	1	1	-
DMOD3W	18	2	2	1	1
DMOD4W	18	2	2	1	1
DMOD5W	19	2	2	1	1
DMOD6W	19	3	2	1	1
DMOD7W	21	3	2	1	1

NOTATION:

			I	D	E
I - MICROINSTRUCTION FETCH*	WC	- CALL TO <u>WADCON</u>	7	-	1
D - CONTROL STORE DATA FETCH*	WS	- CALL TO <u>WRITES</u>	8	1	1
E - EXTERNAL MEMORY FETCH*	WW	- CALL TO <u>WRITEW</u>	6	-	2

*SUMMARIES FOR I, D, AND E INCLUDE I, D, E COUNTS FOR WC, WS, WW

PDP-11 EMULATOR TIMING   FIG. 1   CJN

II DESCRIPTION OF THE EMULATOR

1. General

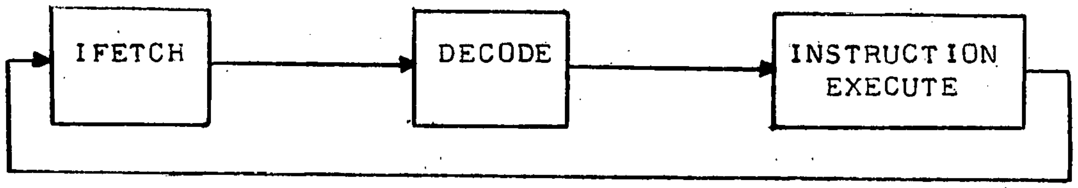
The general structure of the emulator consists of three parts:

1) The instruction fetch (IFETCH) section,

2) The instruction decode section, including decode tables, and

3) Individual instruction handling routines, including those routines common to a certain class or classes of PDP-11 instructions.

The flow of control is as follows:



It is emphasized that this emulation is not a complete PDP-11 emulation. In particular, only the following instructions, thought to be representative of various types, were microcoded in full:

Instruction	Classes
MOV	Logical, Binary
ADD	Arithmetic, Binary
INC	Arithmetic, Unary
ASR	Shift, Unary
BEQ	Branch
MUL	Fixed Point Multiply, Register-Operand

Although a great number of specific instructions were not emulated, the structure of the emulator has been well defined, so that the addition of other PDP-11 instructions to the emulator is quite simple. Adding an instruction consists of invoking the routines available in the emulator for instruction decoding, operand address generation, operand storage of results, etc., and then microcoding the instruction dependent data transformation and the proper setting of the PDP-11 condition codes. Thus the amount of coding needed to add new instructions is in many cases quite small.

It should be noted that at the time of this writing the design of the micromachine and the microlanguage was still evolving, so the objective of this emulation was not so much to produce precise working code as to evaluate the performance and suitability of the micromachine and its resources in emulating a minicomputer as architecturally powerful as the PDP-11. Specifically, two changes or proposed changes to the micromachine have significant effects on the emulator:

1) Alteration of the main memory address structure from word addressable to byte or half word addressable.

2) Modification of the main micromachine cycle so that the A-half of the microinstruction is performed subsequent to the T-half, instead of in parallel with it.

The effect of 1) would be extremely beneficial, cutting the execution time of some PDP-11 instructions by at least 50%. A significant improvement in emulator performance would be realized. Several subroutines that handle main memory fetches and stores could be eliminated. This fact and 2) above would necessitate some recoding.

2. Differences Between 'Emulated' and 'Target' Machines

How 'faithfully' does the emulation match the target machine? A number of PDP-11 features were not included in this first attempt, the most important feature being a priority interrupt type of interaction with the devices on the micromachine's external bus. It is fairly clear, however, that there should be no great difficulty in adding the necessary features to the emulator without any gross alterations of its structure. We shall indicate some of these features.

2.1 Hardware Interrupts (From I/O devices)

External device interrupt handling would be facilitated by the addition of an interrupt handling unit on the external bus. At each IFETCH, the micromachine would check to see if the handler indicated a device had interrupted, and would then read a register in the handler to identify which devices had interrupted and their associated priorities were.

2.2 I/O Device and Special References

The topmost 4K words of the address space spanned by 18 bit PDP-11 addresses are reserved for I/O devices and special internal processor registers such as the Processor Status (PS) or Stack Limit (SL) registers. In the absence of special address traslation hardware, this 18 bit address is generated from a 16 bit address by setting bits 16 and 17 to a value resulting from the logical 'and' of bits 13, 14, and 15. To handle this in the emulator, all main memory access routines simply check every address for this condition. Should the condition be true, then the address must be compared against a list of addresses of internal registers. These registers, along with the general registers, are stored in micromemory. If there is no match, then the reference is to an external device, and a transfer would then be made to a routine that handles such communication.

2.3 Condition Code Handling

It was found that the differences in the representation of condition codes in the PDP-11 (i.e. N, Z, V, C bits in the PS) vs. those of the micromachine required too many cycles of manipulation to resolve. Therefore, the micromachine codes are simply stored at a location in micromemory (CCODES) at the appropriate times and represent the PDP-11 codes. At present only the leftmost four bits of the micromachine codes are stored. It may be desirable to store them all, since at present, conditions of overflow and zero result are not simultaneously detectable. The remainder of the processor status word is stored separately in micromemory (PS).

2.4 Possible PDP 11/45 Emulation

Space was reserved in micromemory for the eight additional general registers used by the 11/45, but in general no attempt was made to provide for the more sophisticated features of the machine such as the floating point unit, the memory management unit, or the multiprogramming mode control feature. The simplest implementation of an 11/45 requires an interaction between almost every instruction and the processor status to determine which physical register set to use, which mode the machine is operating in, etc. Multiprogramming protection features must also be accounted for, especially on interrupts. All of this obviously adds overhead.

2.5 Common Routines for Unimplemented Instructions

As has been claimed above, the routines included in the emulator facilitate the addition of many of the PDP-11 instructions. Only the instruction dependent data transformations and setting of the condition codes need be written. There are some exceptions however. A set of routine needs to be coded to handle destination operands for the instructions JMP and JSR. However, these routines would be nearly identical to the destination mode routines already coded (DMODRD). The differences are:

1) Mode 0 is illegal for these instructions, and

2) The final data fetch for the instruction is not done, but rather the address of the datum is used as the operand (it is loaded into the PC).

Also, a common routine for handling interrupt initiated accesses to main memory should probably be coded. It would handle the loading of the new PC and PS from the interrupt vector and the stacking of the old PC and PS. This could be used in conjunction with instructions such as BPT, IOT, TRAP, EMT (essentially software interrupts) as well as device interrupts and break conditions (e.g. stack errors). Finally, although full decoding was implemented for them, no PDP-11 byte instructions were emulated. All that is required in the present context, however, is the coding of main memory access routines to handle bytes as 16 bit words are now handled, and the modification of code that implements the auto-increment and auto-decrement addressing modes to add or subtract the correct amount for byte addressing.

2.6 Stack Limit Test

No stack limit testing is done in this emulation, nor is the system stack pointer (R6) checked to allow only word organization of the stack. This could be handled by the main memory access routines.

3. Detailed Structure of Emulator

There are eight registers in the micromachine. Their uses are indicated below. Because the registers are a relatively scarce machine resource, it was necessary to use them for different purposes in different situations. Thus the uses given for registers 2-7 below are subject to change throughout the emulator.

Register	Symbolic Name	Usage
0	MAR	Micromachine memory address register
1	IR	PDP-11 instruction register - holds PDP-11 instruction
2	SR	PDP-11 source register - holds source addresses and operands
3	DR	PDP-11 destination register - holds destination addresses and operands
4	R4	General use - main memory addresses
5	R5	Holds return address in subroutine calls
6	R6	Micromemory stack pointer for threaded code
7	R7	General use and main memory operands

3.1 PDP-11 Registers

The PDP-11 general registers are kept in the lowest locations in micromemory, so that the register number is the micromemory address of the register. The 16 bit contents are right justified in micromemory.

3.2 IFETCH Section

The IFETCH portion of the emulator implements the fetching of the next PDP-11 instruction. The PC is read from micromemory and checked for legality (it mustn't be odd). The byte address is converted to a word address via a right shift two places. The two rightmost bits of the byte address are saved in ICODE bits 22 and 23 of the MAR (see fig. 2). After the fetch of the instruction from main memory, a branch on the ICODE bit determines whether the low or high order 16 bits of the fetched word is the actual instruction. The PC is incremented by two.

The destination register field of the instruction is extracted for later use. Bits 12-15 of the IR are used as an offset into the first decode table, and an indirect jump through this table begins the decoding process.

3.3 Decode Section

The decoding of PDP-11 instructions is implemented by a group of eight decode tables in micromemory. Associated with each table is a corresponding table entry routine which loads the address of the table base into a register and inserts an offset derived from a selected bit field in the IR to form the address of the proper entry in the table. An indirect jump transfers control to the proper routine. Each entry in all of the decode tables is an address of either a specific instruction handling routine or another table entry routine. Thus the decoding of an instruction consists of a series of indirect jumps through decode tables until the specific routine for that instruction is reached. Starting the tables on proper boundaries in memory saves a cycle in each of the table entry routines. This complicated decoding scheme is the result of the fact that decoding of PDP-11 instructions does not involve only a fixed length opcode field; in some cases the entire 16 bits must be examined (see fig. 3). Decoding proceeds as follows:

Instruction Type	Example	Decode Table Sequence
Binary (Double Operand)	MOV	1 => instruction
Binary Byte	MOVB	1 => instruction
Unary (Single Operand)	INC	1 => 2 => 4 => instruction
Unary Byte	INCB	1 => 7 => 8 => instruction
Register-Operand	MUL	1 => 3 => instruction
Branch (a)	BEQ	1 => 2 => instruction
JMP, RTS, SWAB, CCOP, SPL	JMP	1 => 2 => 5 => instruction
JSR	JSR	1 => 2 => instruction
BRANCH (b), EMT, TRAP	BPL	1 => 7 => instruction
OPERATE	HALT	1 => 2 => 5 => 6 => instruction

The only decoding not completed by the tables is differentiation of RTS from SPL. The longest decode is required by the operate instructions. These are rather infrequently used instructions, with the exception of RTI. Perhaps something special could be done to speed decoding for these.

3.4 Address Calculation

It is useful at this point to enumerate several non-disjoint classes of PDP-11 instructions (byte instructions are omitted for simplicity).

Class	Members
Double Operand	MOV, CMP, BIT, BIC, ADD, SUB
Register-Operand	MUL, DIV, ASH, ASHC, XOR
Operate	HALT, WAIT, RTI, BPT, IOT, RESET, RTI
I-class	CMP, TST, BIT, MUL, DIV, MFP, ASH, ASHC
O-class	MOV, CLR, MTP
P-class	COM, INC, DEC, NEG, ADC, SBC, ROR, ROL, ASR, ASL, SXT, SWAB, ADD, SUB, BIS, BIC, XOR, MOVB (Mode 0)

The last three classes are important as to their implications on the structure of the emulator. For binary (double operand) instructions, the source operand is always fetched. For I-class instructions, the destination operand is read, but the result of the instruction is not a modification of the destination location. For O-class instructions the destination is never fetched but only written into. P-class instructions, the most common type, read the destination operand, perform some transformation on it, and then store the result back in this location. Note that, for P-class instructions, once the destination address has been computed in order to read the destination, it is unnecessary to perform the address calculations again to store the result; the address must simply be preserved while the transformation is accomplished.

The preceding analysis gives rise to a set of three different address computation routines:

Name	Function
SMODE	- computes address of and fetches source operand for binary instructions.
DMODRD	- computes addres of and fetches destination operand for I and P-class instructions.
DMODWT	- computes address of and stores destination operand for O-class instructions only.

Each of these routines is similar in structure; the main difference between them is the main memory access subroutine which is called. Each routine uses the appropriate mode field from the IR to index through a table of routines that handle each possible mode. The mode routine performs the appropriate main memory or micromemory references in the correct sequence; some call one of the main memory access subroutines. Each mode routine also follows conventions as to operand placement, so that the code that implements particular instructions has no dependence on the addressing mode(s).

There are also three main memory access routines:

Name	Function
WADCON -	Given a byte address, this routine returns a 16 bit operand that corresponds to the given address (i.e. a general main memory fetch).
WRITES -	Given a byte address and a 16 bit operand, the routine writes the operand to the corresponding main memory address. This is used by the O-class instructions.
WRITEW -	Given a byte address and a left justified 16 bit operand, the routine writes the operand to the corresponding main memory address. This routine is used by the P-class instructions and is called directly from the code that performs the instruction data transformation.

There is one additional routine used by P-class instructions in certain cases:

MOD0 - Performs the same function as WRITEW, but handles only the case of destination mode 0. The result is stored in one of the PDP-11 general registers in micromemory.

3.5 Individual Instruction Implementation - Threaded Code

Each instruction is implemented in a convenient, uniform way that allows for ease of coding and of following the flow of control through the emulator. A technique called threaded code is used which eliminates a subroutine call type of implementation in favor of one in which each routine calls the next one in succession rather than returning to some main routine. By convention, the first microinstruction of each PDP-11 instruction execution routine sets up a stack in micromemory which immediately follows the instruction. The stack consists of a list of addresses of routines to be sequentially invoked in the execution of the instruction. An indirect load of the MAR calls the first routine. It and each subsequent routine return to the sequince by doing a POP from R6 (the stack pointer) into the MAR. The last address on the list for each instruction is that of IFETCH, so that the emulator cycle is completed. The only difference in the stacks between instructions in the same class is in most cases simply the address of the routine that does the actual data manipulation and sets the condition codes (e.g. INCOP vs. ASROP).

3.5.1 Individual Routines

The code for implementing the individual instruction routines is for the most part self-explanatory, but a few remarks are in order:

1) In many cases, to take advantage of the micromachine condition codes on logical and especially arithmetic operations, the 16-bit operands are left justified first, and then operated on to give a result which is also left justified. This is taken into account in the WRITEW routine.

2) In the multiply (MUL) instruction implementation, two 16 bit operands are multiplied to give a 32 bit result. The multiplier is left justified so that after the proper number of multiply steps and shifts, the entire 32 bit result is contained in micromachine register 4.

3.5.3 References to Non-existent Labels

Other than non-emulated instructions, the following labels are not coded: ODDPC, RBYTE, WBYTE, WBYTES, RESVD.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCODE								ICODE								STATE				MAR											

- CONDITION CODES SET BY T-MACHINE

C C	C	H	L	D	P	B		
0 0							- ZERO	
0 1							- LESS THAN	
1 0							- GREATER THAN	
1 1							- OVERFLOW	
	0						CARRY	
	1							
		0					HIGH BIT (BIT 31)	
		1						
			0				LOW BIT (BIT Ø)	
			1					
				0			DIFFERENT	BITS 0-31 NOT SAME
				1				BITS 0-31 SAME
					0		PARITY	
					1			
						0	BUS REQ STATUS	NO HOST MACHINE REQUEST IN PROGRESS
						1		HOST MACHINE REQUEST IN PROGRESS

ICODES - INDICATOR CODES SET BY PROGRAMMER BUT MACHINE TESTABLE

R	DI	-	-	
0				- HOST MACHINE HALTED
1				- HOST MACHINE RUNNING
	0			- ENABLE INTERRUPTS
	1			- DISABLE INTERRUPTS

MAR

MICROINSTRUCTION REGISTER
POINTS TO NEXT MICROINSTRUCTION
FETCH ADDRESS

MAR REGISTER FORMAT

FIGURE 2

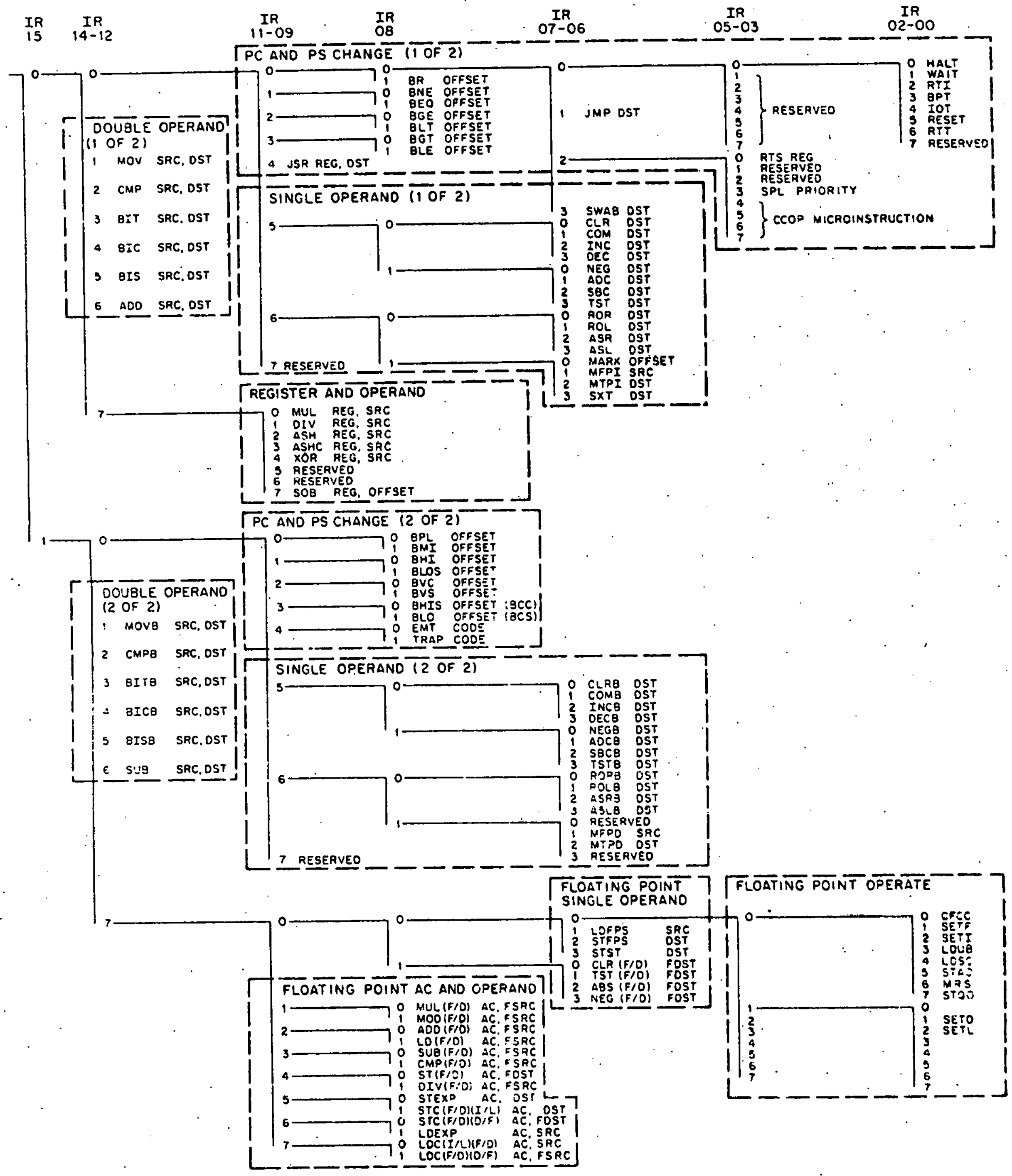


FIGURE 3

PDP-11 INSTRUCTION CODE

III INFORMAL DESCRIPTION OF THE MICROASSEMBLY LANGUAGE

1. Introduction

For the purposes of evaluating the micromachine we have specified a simple microassembly language. Since this language will not be used in the actual laboratory system we will only give an informal description here.

Internally the micromachine consists of three submachines each receiving control information from the current microinstruction and acting independently except when data dependent conflicts occur. These machines and their function are as follows:

1) T-machine - functional processing of register data with logical and arithmetic operations.

2) A-machine - handling of communications between control store, the registers and external devices. The A-machine also performs elementary calculations oriented toward address formation.

3) I-machine - fetching of the next microinstruction and conditional testing.

In terms of the hardware representation, microinstructins are 32 bits in length. For convenience we consider each microinstruction to be divided into a left half (14 bits) and a right half (18 bits). In general, the left half of the instruction specifies a T-machine operation and the right half specifies an A-machine operation. Occasionally, one or both halves may be used to specify an I-machine operation. In addition, there are cases when the entire instruction is used to specify only a T-machine operation, in which case, the right half of the instruction is interpreted as immediate data. Thus we have the following general forms of instruction format:

```
<label>: <T-machine spec> / <A-machine spec> ; <comment>
<label>: <I-machine spec> / <A-machine spec> ; <comment>
<label>: <T-machine spec> / <I-machine spec> ; <comment>
<label>: <I-machine spec> / <I-machine spec> ; <comment>
<label>: <long T-machine spec> ; <comment>
```

Label and comment fields are optional. If a spec field is blank then the assumption is that a NOP is specified.

2. T-machine Instruction Specification

T-machine operations perform the following functions:

1) Logical
2) Arithmetic
3) Shift/Rotate

4) Extended Arithmetic
5) Insert/Extract

Insert/Extract operations are long format instructions, in addition the other instruction types may be either long format or short format depending upon whether immediate data is used or not. Short format instructions may, in some cases, specify immediate data. The particular format used is identified by the symbol following the the T-machine opcode:

1) ~~ø~~ - short format instruction, no immediate data
2) S - short format instruction, immediate data included
3) L - long format instruction, immediate data from right half

Thus, except for INSERT/EXTRACT instructions, the T-machine specifications may be as follows:

```
<opcode>  <regA>,<regB>
<opcode>S <regA>,<immediate data>
<opcode>L <regA>,<immediate data>
```

2.1 Logical Opcodes

Logical instructions may use either one or two register specifications as required. The opcodes are as follows:

CLR	clear register
NOP	no operation
OR	logical 'or'
XOR	logical 'exclusive or'
XNOR	logical 'exclusive nor'
NOR	logical 'nor'
AND	logical 'and'
NAND	logical 'nand'
COM	logical 'not'
XFR	register transfer
CXFR	complemented register transfer
TEST	register unchanged but condition codes set
ONE	set register to ones

2.2 Arithmetic Opcodes

Arithmetic operations consist of additon and subtraction. In addition, and arithmetic operation may be used for comparison purposes by following the opcode specification with a 'C' in which case the micromachine performs the indicated arithmetic operation but does not store the result in the destination register. Usually two registers are specified (a destination, <regA>, and a source, <regB>) in operations involving small immediate data values one register may be replaced by an immediate data specification in the range -8 to decimal. The arithmetic opcodes are:

ADD	two's complement addition

ADC	two's complement addition with carry
SUB	two's complement subtraction
SBB	two's complement subtraction with borrow

2.3 Shift/Rotate Opcodes

In a Shift/Rotate instruction the <regA> specification identifies the register to be modified. The regB specification may either specifiy the immediate shift amount (-8 to or identify a register holding the shift amount. On long format instructions the right half immediate data field provides the shift amount. Shift/Rotate opcodes are as follows:

LSS	left shift single (logical)
LRS	left rotate single
RSS	right shift single (logical)
RAS	right shift arithmetic single
LSD	left shift double (logical)
LRD	left rotate double
RSD	right shift double (logical)
RAD	right shift arithmetic double

2.4 Extended Arithmetic Opcodes

Extended arithmetic operations use either the short format with two register specifications or the long format with immediate data from the right half. The extended arithmetic opcodes are:

MULT	multiply step
DIVD	divide step
XSS	form excess sixes
DEL	delay cycle
BTD	binary to decimal conversion step
DTB	decimal to binary cnversion step
DAD	decimal addition
DSU	decimal subtraction

2.5 Insert/Extract Opcodes

Insert/Extract instructions have a separate format:

<opcode> <regA>,<regB>,<rotate amount>,<mask data>

The rotate amounte is specified in decimal (0 to 32) and the mask data is immediate data from the right half field of the microinstruction. Register specification <regA> denotes the destination and <regB> indicates the source.

The two opcodes are:

INS	insert
EXT	extract

3. A-machine Instruction Specification

A-machine instructions are used by the microprogrammer to access control store and external memory, perform simple address calculations and provide short loops in the microprogram. Specifically the operation codes may be divided into the following classes:

1) Direct memory access,
2) Indirect memory access,
3) Pointer modification, and
4) Stacking operations.

3.1 Direct Memory Access Opcodes

The format for direct memory access operations is as follows:

```
<opcode> <regC>,<address>
```

The specification, <regC>, represents the source or destination register for the operation and <address> represents a 12 bit address in control store. The three opcodes are:

LR	load register from control store
SR	store register in control store
RI	load register with address immediate

3.2 Indirect Memory Access Opcodes

Indirect memory access operations are used to move data between the registers, control memory and the external memory system. These operations have the following format:

```
<opcode> <regC>,<regD>,<subopcode>,<immediate data>
```

By convention, the <regC> specification will identify the destination for the memory move operation and <regD> will identify the source. Depending upon the opcode the register contents may be used directly or as a pointer to control or external memory. After each operation the contents of either, neither or both registers is incremented by the amount specified in the <immediate data> field as specified by the <subopcode> specification. Immediate data may be in the range from -8 to decimal. Opcodes are:

RR	register to register
RM	register to control memory
RE	register to external memory
MR	control memory to register
MM	control memory to control memory
ME	control memory to external memory
ER	external memory to register
EM	external memory to control memory

Subopcode specifications are:

MN	modify neither pointer register
MS	modify source pointer register
MD	modify destination pointer register
MB	modify both pointer registers

3.3 Pointer Modification Opcodes

The pointer modification instructions allow the microprogrammer to perform simple address calculations and to control the fetching of the next microinstruction based on the outcome of the operation. The two possible formats of these instructions are:

```
<opcode1> <regC>,<regD>,<subopcode>,<address modifier>

<opcode2> <regC>,<immediate data>,<subopcode>,<address modifier>
```

Instructions with the <opcode1> specification use the two registers specified in the address calculation, while the instructions with <opcode2> specifications use a single register and immediate data in the range of -8 to decimal. The <subopcode> specification defines a test on the result of the address calculation. When the test is 'true' the address modifier (-8 to is added to the address of the following microinstruction to provide the address where the next microinstruction is to be fetched from.

The specifications for <opcode1> are:

ADD	two's complement addition of registers
SUB	two's complement subtraction of registers

The specifications for <opcode2> are:

INC	two's complement increment of a register
DEC	two's complement decrement of a register

The specifications for the <subopcode> test are:

NL	no looping
LLT	loop if less than zero
LLE	loop if less than or equal to zero
LZ	loop if zero
LNZ	loop if not zero
LGE	loop if greater than or equal to zero
LGT	loop if greater than zero
AL	always loop

3.4 Stacking Operation Opcodes

Stacking operations are used by the programmer to manipulate stacks in control memory. The format is:

```
<opcode> <regC>,<regD>,<subopcode>,<immediate data>
```

On Stacking operations the <regC> specification identifies a register which is the source or destination of the stack data. The <regD> specification identifies a register containing control information. The <subopcode> specification is used to indicate whether or not data transfer takes place and whether or not the stack pointer is to be tested against. The opcodes are:

POP	control memory to register
PUSH	register to control memory

The sub opcodes are:

NLNT	no limit test, no transfer
NLT	no limit test, transfer
LNT	limit test, no transfer
LT	limit test, transfer

4. I-machine Instruction Specifications

Asside from the pointer modification instruction described above there are two I-machine instructions. One, the conditional instruction, is used in the left half specification to conditionally control the execution of the right half instruction. The other, the branch instruction, is used in the right half specification to cause conditional branching by short relative amounts.

The formats for these instructions are as follows:

```
COND  <test spec>
COND  <mask>,<test code>

BRN   <test spec>,<address modifier>
BRN   <mask>,<test code>,<address modifier>
```

In specifying the test the microprogrammer may either give an eleven bit quantity which contains both the mask and the test code, or specify the mask and use one of the test codes listed below.

CAT	test condition codes for all, execute if true
CAF	test condition codes for all, execute if false
COT	test condition codes for any, execute if true
COF	test condition codes for any, execute if false
IAT	test indicator codes for all, execute if true
IAF	test indicator codes for all, execute if false
IAT	test indicator codes for any, execute if true
IOT	test indicator codes for any, execute if false

5. Miscellaneous

In specifying microinstructions an empty left or right half specification is used to indicate a 'NOP' instruction to the T- and

A-machines respectively.

Constants are either decimal or octal as specified in the formats above. Decimal constants are written directly, while octal quantities are preceeded by a zero (e.g. 077 is the octal representation of 63 decimal).

Some of the assembler directives used are

.WORD	reserve a word and initialize to constant
.BLKW	reserve a block of words
.=	set location counter
.END	end of program
%	register definition
=	direct assignment

In addition labels are used in branch instructions, in direct memory access and in point modification instructions, with the resulting <address> or <address modifier> field calculated by the assembler.

PDP-11 EMULATION CODE

```
            MAR  = %0                  ;REGISTER DEFINITIONS
            IR   = %1
            SR   = %2
            DR   = %3
            R4   = %4
            R5   = %5
            R6   = %6
            R7   = %7

    LOWON   =      0110                ;MASK FOR LOW BIT TEST
    ZERO    =      0772                ;MASK FOR ZERO TEST
    DON     =      0040                ;MASK FOR DIFFERENCE TEST
    NZERO   =      0776                ;MASK FOR NOT ZERO TEST
    NOOVF   =      03000               ;MASK FOR NO OVERFLOW TEST
    WADD    =      0200                ;MASK FOR WORD ADDRESS TEST
    BADD    -      0100                ;MASK FOR BYTE ADDRESS TEST

    LOW16   =      0177777             ;MASK FOR LOW 16 CONTIGUOUS BITS
    HIGH16  =      037777600000        ;MASK FOR HIGH 16 CONTIGUOUS BITS
    BABITS  =      060000000           ;MASK TO STORE BITS IN ICODES
    BIT02   =      07                  ;MASK FOR LOW 3 CONTIGUOUS BITS
    BIT03   =      017                 ;MASK FOR LOW 4 CONTIGUOUS BITS
    ADDMSK  =      030000177777        ;MASK FOR WORD ADDRESS COMPUTATION

            .=0                        ;LAYOUT OF PDP-11 REGISTERS IN CONTROL STORE
    PDPR0:  .WORD     0                ;REGISTER 0
    PDPR1:  .WORD     0                ;REGISTER 1
    PDPR2:  .WORD     0                ;REGISTER 2
    PDPR3:  .WORD     0                ;REGISTER 3
    PDPR4:  .WORD     0                ;REGISTER 4
    PDPR5:  .WORD     0                ;REGISTER 5
    PDPSP:  .WORD     0                ;STACK POINTER (KERNEL)
    PDPPC:  .WORD     0                ;PROGRAM COUNTER
            .BLKW     8                ;RESERVE NEXT 8 LOCATIONS FOR POSSIBLE
                                       ;PDP-11/45 EMULATION
    CCODES: .WORD     0                ;CONDITION CODE STORAGE
    PS:     .WORD     0                ;PROCESSOR STATUS WORD STORAGE
                                       ;(THE USUAL PS WITHOUT CCODES)
            .BLKW     10               ;RESERVED FOR SPECIAL INTERNAL REGISTERS
                                       ;(FOR FUTURE USE)
```

```
        .=01000                                         ;START AT LOCATION 01000
IFETCH: CLR  DR                  /LR  R7,PDPPC          ;GET PC
        TEST R7                  /RR  R4,R7,MD,+2       ;R4 <= PC+2
        COND LOWON               /RI  MAR,ODDPC         ;IS PC ODD?
        INS  MAR,R7,22,BABITS                           ;SET LOWER 2 BITS OF BYTE
                                                        ;ADDRESS INTO MAR<23:22>
        RSSS R7,2                /SR  R4,PDPPC          ;CONVERT TO WORD ADDRESS
                                                        ;PC <= PC+2
        ADDS DR,+7               /ER  IR,R7,MN          ;FETCH THE WORD
        XFR  SR,DR               /BRN WADD,IAT,1$       ;HIGH OR LOW 16 BITS
        EXT  IR,IR,0,LOW16                              ;LOW 16 BITS
        AND  DR,IR               /RI  R5,TBL1           ;SELECT DEST. FIELD
                                                        ;LOAD DECODE TABLE BASE
        LSSS SR,+6               /MR DR,DR,MN           ;READ OUT DEST. REGISTER
        INS  R5,IR,20,BIT03                             ;GET BITS 12-15 OF IR
        AND  SR,IR               /MR  MAR,R5,MN         ;JUMP VIA DECODE TABLES
1$:     EXT  IR,IR,16,LOW16                             ;HIGH 16 BITS
        AND  DR,IR               /RI  R5,TBL1           ;SELECT DEST. FIELD
                                                        ;LOAD DECODE TABLE BASE
        LSSS SR,+6               /MR  DR,DR,MN          ;READ OUT DEST. REGISTER
        INS  R5,IR,20,BIT03                             ;GET BITS 12-15 OF IR
        AND  SR,IR               /MR  MAR,R5,MN         ;JUMP VIA DECODE TABLES
```

```
TBL1:   .WORD   TBL2E     ;FIRST DECODE TABLE STARTS ON 16 WORD BOUNDARY
        .WORD   MOV       ;OFFSET IS BITS 12-15 OF IR
        .WORD   CMP
        .WORD   BIT
        .WORD   BIC
        .WORD   BIS
        .WORD   ADD
        .WORD   TBL3E     ;REGISTER-OPERAND TYPE
        .WORD   TBL7E
        .WORD   MOVB
        .WORD   CMPB
        .WORD   BITB
        .WORD   BICB
        .WORD   BISB
        .WORD   SUB
        .WORD   RESVD     ;NOTE -- FLOATING POINT ON 11/45

TBL2:   .WORD   TBL5E     ;SECOND DECODE TABLE
        .WORD   BR        ;STARTS ON 16 WORD BOUNDARY OFFSET IS
        .WORD   BNE       ;OFFSET IS BITS 8-11 OF IR
        .WORD   BEQ
        .WORD   BGE
        .WORD   BLT
        .WORD   BGT
        .WORD   BLE
        .WORD   JSR
        .WORD   JSR
        .WORD   TBL4E     ;SINGLE OPERAND INSTRUCTIONS
        .WORD   TBL4E
        .WORD   TBL4E
        .WORD   TBL4E
        .WORD   RESVD
        .WORD   RESVD

TBL3:   .WORD   MUL       ;THIRD DECODE TABLE
        .WORD   DIV       ;STARTS ON 8 WORD BOUNDARY
        .WORD   ASH       ;OFFSET IS BITS 9-11 OF IR
        .WORD   ASHC      ;HANDLES REGISTER-OPERAND INSTRUCTIONS
        .WORD   XOR
        .WORD   RESVD
        .WORD   RESVD
        .WORD   SOB
```

```
TBL4:    .WORD   ROR        ;FOURTH DECODE TABLE
         .WORD   ROL        ;STARTS ON 16 WORD BOUNDARY
         .WORD   ASR        ;OFFSET IS BITS 6-9 OF IR
         .WORD   ASL
         .WORD   MARK
         .WORD   MFPI
         .WORD   MTPI
         .WORD   SXT
         .WORD   CLR
         .WORD   COM
         .WORD   INC
         .WORD   DEC
         .WORD   NEG
         .WORD   ADC
         .WORD   SBC
         .WORD   TST

TBL5:    .WORD   TBL6E      ;FIFTH DECODE TABLE
         .WORD   RESVD      ;STARTS ON 16 WORD BOUNDARY
         .WORD   RESVD      ;OFFSET IS BITS 4-7 OF IR
         .WORD   RESVD
         .WORD   JMP
         .WORD   JMP
         .WORD   JMP
         .WORD   JMP
         .WORD   RTS        ;RESERVED IF BIT 3=1
         .WORD   SPL        ;RESERVED IF BIT 3=0
         .WORD   CCOP
         .WORD   CCOP
         .WORD   SWAB
         .WORD   SWAB
         .WORD   SWAB
         .WORD   SWAB

TBL6:    .WORD   HALT       ;SIXTH DECODE TABLE
         .WORD   WAIT       ;STARTS ON 16 WORD BOUNDARY
         .WORD   RTI        ;OFFSET IS BITS 0-3 OF IR
         .WORD   BPT
         .WORD   IOT
         .WORD   RESET
         .WORD   RTT
         .WORD   RESVD
         .WORD   RESVD
         .WORD   RESVD
         .WORD   RESVD
         .WORD   RESVD
         .WORD   RESVD
         .WORD   RESVD
         .WORD   RESVD
         .WORD   RESVD
```

```
TBL7:   .WORD   BPL         ;SEVENTH DECODE TABLE
        .WORD   BMI         ;STARTS ON 16 WORD BOUNDARY
        .WORD   BHI         ;OFFSET IS BITS 8-11 OF IR
        .WORD   BLOS
        .WORD   BVC
        .WORD   BVS
        .WORD   BHIS
        .WORD   BLO
        .WORD   EMT
        .WORD   TRAP
        .WORD   TBL8E       ;SINGLE OPERAND (BYTE)
        .WORD   TBL8E
        .WORD   TBL8E
        .WORD   TBL8E
        .WORD   RESVD
        .WORD   RESVD

TBL8:   .WORD   RORB        ;EIGHTH DECODE TABLE
        .WORD   ROLB        ;STARTS ON 16 WORD BOUNDARY
        .WORD   ASRB        ;OFFSET IS BITS 6-9 OF IR
        .WORD   ASLB
        .WORD   RESVD
        .WORD   MFPD
        .WORD   MTPD
        .WORD   RESVD
        .WORD   CLRB
        .WORD   COMB
        .WORD   INCB
        .WORD   DECB
        .WORD   NEGB
        .WORD   ADCB
        .WORD   SBCB
        .WORD   TSTB
```

```
TBL2E:                          /RI   R5,TBL2       ;LOAD TABLE BASE
        INS  R5,IR,24,BIT03                         ;GET BITS 8-11 OR IR
                                /MR   MAR,R5,MN     ;INDIRECT JUMP THRU TABLE

TBL3E:                          /RI   R5,TBL3       ;LOAD TABLE BASE
        INS  R5,IR,23,BIT02                         ;GET BITS 9-11 OF IR
                                /MR   MAR,R5,MN     ;INDIRECT JUMP THRU TABLE

TBL4E:                          /MR   R5,TBL4       ;LOAD TABLE BASE
        INS  R5,IR,26,BIT03                         ;GET;BITS 6-9 OF IR
                                /MR   MAR,R5,MN     ;INDIRECT JUMP THRU TABLE

TBL5E:                          /RI   R5,TBL5       ;LOAD TABLE BASE
        INS  R5,IR,28,BIT03                         ;GET BITS 4-7 OF IR
                                /MR   MAR,R5,MN     ;INDIRECT JUMP THRU TABLE

TBL6E:                          /RI   R5,TBL6       ;LOAD TABLE BASE
        INS  R5,IR,0,BIT03                          ;GET BITS 0-3 OR IR
                                /MR   MAR,R5,MN     ;INDIRECT JUMP THRU TABLE

TBL7E:                          /RI   R5,TBL7       ;LOAD TABLE BASE
        INS  R5,IR,24,BIT03                         ;GET BITS 8-11 OF IR
                                /MR   MAR,R5,MN     ;INDIRECT JUMP THRU TABLE

TBL8E:                          /RI   R5,TBL8       ;LOAD TABLE BASE
        INS  R5,IR,26,BIT03                         ;GET BITS 6-9 OR IR
                                /MR   MAR,R5,MN     ;INDIRECT JUMP THRU TABLE
```

```
SMODE:   EXT  R7,IR,23,BIT02                        ;GET SOURCE MODE
         RSSS SR,+6        /RI  R5,SMDBAS           ;LOAD TABLE BASE
         ADD  R5,R7        /MR  R4,SR,MN            ;ADD OFFSET, FETCH REG[SR]
         ADDS R6,+1        /MR  MAR,R5,MN           ;JUMP THRU TABLE

SMDBAS: .WORD     SMOD0
        .WORD     SMOD1
        .WORD     SMOD2
        .WORD     SMOD3
        .WORD     SMOD4
        .WORD     SMOD5
        .WORD     SMOD6
        .WORD     SMOD7

SMOD0:   XFR  SR,R4        /POP MAR,R6,NLT          ;SR <= OPERAND; RETURN

SMOD1:   XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND
         XFR  SR,R7        /POP MAR,R6,NLT          ;SR <= OPERAND; RETURN

SMOD2:   XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND
         XFR  SR,R7        /RM  SR,R4,MN            ;STORE BACK INCREMENTED VALUE
                                                    ;SR <= OPERAND
                           /POP MAR,R6,NLT          ;RETURN

SMOD3:   XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND ADDRESS
         XFR  R4,R7        /RM  SR,R4,MN            ;STORE BACK INCREMENTED VALUE
         XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND
         XFR  SR,R7        /POP MAR,R6,NLT          ;SR <= OPERAND; RETURN

SMOD4:   SUBS R4,2         /
                           /RM  SR,R4,MN            ;STORE AUTO-DECREMENTED VALUE
         XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND
         XFR  SR,R7        /POP MAR,R6,NLT          ;SR <= OPERAND; RETURN

SMOD5:   SUBS R4,2         /
                           /RM  SR,R4,MN            ;STORE AUTO-DECREMENTED VALUE
         XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND ADDRESS
         XFR  R4,R7        /
         XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND
         XFR  SR,R7        /POP MAR,R6,NLT          ;SR <= OPERAND; RETURN

SMOD6:   XFR  SR,R4        /LR  R4,PDPPC            ;GET PC
         XFR  R5,MAR       /RI  MAR,WADCON          ;GET INDEX WORD
         ADD  SR,R7        /SR  PDPPC,R4            ;PC <= PC+2
         XFR  R4,SR        /                        ;FORM OPERAND ADDRESS
         XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND
         XFR  SR,R7        /POP MAR,R6,NLT          ;SR <= OPERAND; RETURN

SMOD7:   XFR  SR,R4        /LR  R4,PDPPC            ;GET PC
         XFR  R5,MAR       /RI  MAR,WADCON          ;GET INDEX WORD
         ADD  SR,R7        /SR  PDPPC,R4            ;PC <= PC+2
         XFR  R4,SR        /                        ;FORM ADDRESS
         XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND ADDRESS
         XFR  R4,R7        /
         XFR  R5,MAR       /RI  MAR,WADCON          ;FETCH OPERAND
         XFR  SR,R7        /POP MAR,R6,NLT          ;SR <= OPERAND; RETURN
```

```
DMODRD: EXT   R7,IR,29,BIT02                                 ;GET DEST. MODE
                                   /RI   R5,DMRBAS           ;LOAD TABLE BASE
        ADD   R5,R7                /MR   R4,DR,MN            ;ADD OFFSET; FETCH REG[DR]
        ADDS  R6,+1                /MR   MAR,R5,MN           ;JUMP THRU TABLE

DMRBAS: .WORD     DMOD0R
        .WORD     DMOD1R
        .WORD     DMOD2R
        .WORD     DMOD3R
        .WORD     DMOD4R
        .WORD     DMOD5R
        .WORD     DMOD6R
        .WORD     DMOD7R

DMOD0R: XFR   DR,R4                /POP  MAR,R6,NLT          ;DR <= OPERAND; RETURN

DMOD1R: XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND
        XFR   DR,R7                /POP  MAR,R6,NLT          ;DR <= OPERAND; RETURN

DMOD2R: XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERANDD
        XFR   DR,R7                /RM   DR,R4,MN            ;STORE BACK INCREMENTED VALUE
                                                             ;DR <= OPERAND
                                   /POP  MAR,R6,NLT          ;RETURN

DMOD3R: XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND ADDRESS
        XFR   R4,R7                /RM   DR,R4,MN            ;STORE INCREMENTED VALUE
        XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND
        XFR   DR,R7                /POP  MAR,R6,NLT          ;DR <= OPERAND; RETURN

DMOD4R: SUBS  R4,2                 /                         ;AUTO-DECREMENT
                                   /RM   DR,R4,MN            ;STORE DECREMENTED VALUE
        XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND
        XFR   DR,R7                /POP  MAR,R6,NLT          ;DR <= OPERAND; RETURN

DMOD5R: SUBS  R4,2                 /
                                   /RM   DR,R4,MN            ;STORE DECREMENTED VALUE
        XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND ADDRESS
        XFR   R4,R7                /
        XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND
        XFR   DR,R7                /POP  MAR,R6,NLT          ;DR <= OPERAND; RETURN

DMOD6R: XFR   DR,R4                /LR   R4,PDPPC            ;GET PC
        XFR   R5,MAR               /RI   MAR,WADCON          ;GET INDEX WORD
        ADD   DR,R7                /SR   PDPPC,R4            ;PC <= PC+2
        XFR   R4,DR                /                         ;FORM OPERAND ADDRESS
        XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND
        XFR   DR,R7                /POP  MAR,R6,NLT          ;DR <= OPERAND; RETURN

DMOD7R: XFR   DR,R4                /LR   R4,PDPPC            ;GET PC
        XFR   R5,MAR               /RI   MAR,WADCON          ;GET INDEX WORD
        ADD   DR,R7                /SR   PDPPC,R4            ;PC <= PC+2
        XFR   R4,DR                /                         ;FORM ADDRESS
        XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND ADDRESS
        XFR   R4,R7                /
        XFR   R5,MAR               /RI   MAR,WADCON          ;FETCH OPERAND
        XFR   DR,R7                /POP  MAR,R6,NLT          ;DR <= OPERAND; RETURN
```

```
DMODWT:                                ;USED ONLY BY MOV, CLR, AND MTP
        EXT  R7,IR,29,BIT02                                 ;GET DEST. MODE
                                       /RI  R5,DMWBAS       ;LOAD TABLE BASE
        ADD  R5,R7                     /MR  R4,DR,MN        ;ADD OFFSET, FETCH REG[DR]
        ADDS R6,+1                     /MR  MAR,R5,MN       ;JUMP THRU TABLE

DMWBAS: .WORD    DMOD0W
        .WORD    WRITES
        .WORD    DMOD2W
        .WORD    DMOD3W
        .WORD    DMOD4W
        .WORD    DMOD5W
        .WORD    DMOD6W
        .WORD    DMOD7W

DMOD0W:                                /RM  DR,SR,MN        ;REG[DR] <= OPERAND
                                       /POP MAR,R6,NLT      ;RETURN

DMOD2W: ADDS R4,+2                     /                    ;AUTO-INCREMENT
                                       /RM  DR,R4,MS,-2     ;STORE BACK INCREMENTED VALUE
                                       /RI  MAR,WRITES      ;STORE OPERAND

DMOD3W: XFR  R5,MAR                    /RI  MAR,WADCON      ;FETCH OPERAND ADDRESS
        XFR  R4,R7                     /RM  DR,R4,MN        ;AUTO-INCREMENT
                                       /RI  MAR,WRITES      ;STORE OPERAND

DMOD4W: SUBS R4,+2                     /                    ;AUTO-DECREMENT
                                       /RM  DR,R4,MN        ;STORE DECREMENTED VALUE
                                       /RI  MAR,WRITES      ;STORE OPERAND

DMOD5W: SUBS R4,+2                     /                    ;AUTO-DECREMENT
                                       /RM  DR,R4,MN        ;STORE DECREMENTED VALUE
        XFR R5,MAR                     /RI  MAR,WADCON      ;FETCH OPERAND ADDRESS
        XFR  R4,R7                     /RI  MAR,WRITES      ;STORE OPERAND

DMOD6W: XFR  DR,R4                     /LR  R4,PDPPC        ;GET PC
        XFR  R5,MAR                    /RI  MAR,WADCON      ;GET INDEX WORD
        ADD  DR,R7                     /SR  PDPDC,R4        ;PC <= PC+2
        XFR  R4,DR                     /RI  MAR,WRITES      ;STORE OPERAND

DMOD7W: XFR  DR,R4                     /LR  R4,PDPPC        ;GET PC
        XFR  R5,MAR                    /RI  MAR,WADCON      ;GET INDEX WORD
        ADD  DR,R7                     /SR  PDPPC,R4        ;PC <= PC+2
        XFR  R4,DR                     /                    ;FORM ADDRESS
        XFR  R5,MAR                    /RI  MAR,WADCON      ;FETCH OPERAND ADDRESS
        XFR  R4,R7                     /RI  MAR,WRITES      ;STORE OPERAND
```

```
WADCON:                           ;WORD ADDRESS CONVERSION ROUTINE
                                  ;READ WORD FROM MAIN MEMORY
        INS  MAR,R4,22,BABITS                        ;SET LOWER TWO BITS OF BYTE
                                                     ;ADDRESS IN MAR<23:22>
        COND BADD,IAT          /RI  MAR,RBYTE        ;ODD (BYTE) ADDRESS?
        EXT  R4,R4,30,ADDMSK                         ;MAKE INTO A WORD ADDRESS
        LRSS R4,+2             /ER  R7,R4,MN         ;FETCH WORD (32 BITS)
        ADDS R4,+2             /BRN WADD,IAT,3$      ;LOW OR HIGH 16 BITS
                                                     ;INCREMENT ORIGINAL ADDRESS
        EXT  R7,R7,0,LOW16                           ;LOW 16 BITS
        XFR  MAR,R5            /                     ;RETURN
3$:     EXT  R7,R7,16,LOW16                          ;HIGH 16 BITS
        XFR  MAR,R5            /                     ;RETURN


WRITES:                           ;WRITE WORD TO MAIN MEMORY (SPECIAL)
        INS  MAR,R4,22,BABITS                        ;SET LOWER TWO BITS OF BYTE
                                                     ;ADDRESS INTO MAR<23:22>
        COND BADD,IAT          /RI  MAR,WBYTES       ;ODD (BYTE) ADDRESS?
        EXT  R4,R4,30,ADDMSK                         ;MAKE WORD ADDRESS
                               /ER  R7,R4,MN         ;FETCH WORD (32 BITS)
                               /BRN WADD,IAT,4$      ;LOW OR HIGH 16 BITS TO
                                                     ;BE WRITTEN
        INS  R7,SR,0,LOW16                           ;LOW 16 BITS
                               /RE  R4,R7,MN         ;WRITE BACK WORD
                               /POP MAR,R6,NLT       ;RETURN
4$:     INS  R7,SR,16,HIGH16                         ;HIGH 16 BITS
                               /RE  R4,R7,MN         ;WRITE BACK WORD
                               /POP MAR,R6,NLT       ;RETURN


WRITEW:                           ;WRITE WORD (16 BITS) TO MAIN MEMORY
        INS  MAR,R4,22,BABITS                        ;SET LOWER TWO BITS OF BYTE
                                                     ;ADDRESS INTO MAR<23:22>
        COND BADD,IAT          /RI  MAR,WBYTE        ;ODD (BYTE) ADDRESS?
        EXT  R4,R4,30,ADDMSK                         ;GET WORD ADDRESS
                               /ER  R7,R4,MN         ;FETCH WORD (32 BITS)
        ADDS R6,+1             /BRN WADD,IAT,5$      ;LOW OR HIGH 16 BITS TO
                                                     ;BE WRITTEN
        INS  R7,DR,16,LOW16                          ;LOW 16 BITS
        XFR  MAR,R5            /RE  R4,R7,MN         ;WRITE BACK WORD; RETURN
5$:     INS  R7,DR,0,HIGH16                          ;HIGH 16 BITS
        XFR  MAR,R5            /RE  R4,R7,MN         ;WRITE BACK WORD; RETURN


MOD0C:  EXT  R5,R7,4,BIT03                           ;EXTRACT CCODES FROM SAVED MAR
MOD0:                          /SR  R5,CCODES        ;STORE PDP-11 CCODES
        EXT  R4,IR,0,BIT02                           ;EXTRACT DR FIELD FROM IR
        RSSL DR,16                                   ;RIGHT SHIFT RESULT BY 16
        ADDS R6,+1             /RM  R4,DR,MN         ;STORE RESULT IN REG[DR]
                               /POP MAR,R6,NLT       ;RETURN
```

```
MOV:                                                    ;MOV INSTRUCTION
         XFR  R6,MAR            /LR   MAR,MOVST         ;SET UP TOP OF STACK
MOVST:   .WORD    SMODE                                 ;GET SOURCE
         .WORD    MOVOP                                 ;SET CONDITION CODES
         .WORD    DMODWT                                ;STORE RESULT IN DESTINATION
         .WORD    IFETCH                                ;FETCH NEXT INSTRUCTION

MOVOP:   EXT  R7,SR,16,HIGH16                           ;SET HOST MACHINE CCODES
         XFR  R7,MAR            /LR   R5,CCODES         ;SAVE HOST MACHINE CCODES
                                                        ;FETCH PDP-11 CCODES
         INS  R5,R7,4,015                               ;DON'T CHANGE PDP-11 'C' BIT
         ADDS R6,+1             /SR   R5,CCODES         ;STORE PDP-11 CCODES
                                /POP  MAR,R6,NLT        ;RETURN



ADD:                                                    ;ADD INSTRUCTION
         XFR  R6,MAR            /LR MAR,ADDST           ;SET TOP OF STACK

ADDST:   .WORD    SMODE                                 ;GET SOURCE
         .WORD    DMODRD                                ;GET DESTINATION
         .WORD    ADDOP                                 ;ADD, STORE RESULT IN DEST.
         .WORD    IFETCH                                ;FETCH NEXT INSTRUCTION

ADDOP:   LSSL SR,16                                     ;LEFT JUSTIFY SOURCE OPERAND
         LSSL DR,16                                     ;LEFT JUSTIFY DEST. OPERAND
         ADD  DR,SR             /LR   R5,DMDMSK         ;DO THE ADD; LOAD MASK
                                                        ;FOR DESTINATION MODE
         AND  R5,IR             /RR   R7,MAR,MN         ;SAVE HOST MACHINE CCODES
         SUBS R4,+2             /BRN  ZERO,MOD0C        ;DEST. MODE=0?
         XFR  R5,MAR            /RI   MAR,WRITEW        ;NO, CALL SUBROUTINE TO WRITE
                                                        ;RESULT TO MAIN MEMORY
         EXT  R5,R7,4,BIT03                             ;EXTRACT CCODES FROM SAVED MAR
         ADDS R6,+1             /SR   R5,CCODES         ;STORE PDP-11 CCODES
                                /POP  MAR,R6,NLT        ;RETURN
DMDMSK:  .WORD  070
```

```
INC:                                                    ;INC INSTRUCTION
        XFR  R6,MAR             /LR  MAR,INCST          ;SET TOP OF STACK

INCST:  .WORD    DMODRD                                 ;GET OPERAND
        .WORD    INCOP                                  ;INC & STORE RESULT; SET COI
        .WORD    IFETCH                                 ;FETCH NEXT INSTRUCTION

INCOP:  LSSL DR,16                                      ;LEFT JUSTIFY OPERAND
        ADDL DR,+0200000                                ;DO THE INCREMENT
        XFR  R7,MAR             /LR  R5,CCODES          ;SAVE HOST MACHINE CCODES
                                                        ;FETCH PDP-11 CCODES
        INS  R5,R7,4,015                                ;DON'T CHANGE PDP-11 'C' BIT
        EXT  R7,IR,29,BIT02                             ;EXTRACT DEST. MODE FROM IR
        COND ZERO               /RI  MAR,MODO           ;DESTINATION MODE=0?
        SUBS R4,+2              /SR  R5,CCODES          ;STORE PDP-11 CCODES
        XFR R5,MAR              /RI  MAR,WRITEW         ;NO, CALL ROUTINE TO WRITE
                                                        ;RESULT TO MAIN MEMORY
                                /POP MAR,R6,NLT         ;RETURN



ASR:                                                    ;ASR INSTRUCTION
        XFR  R6,MAR             /LR  MAR,ASRST          ;SET TOP OF STACK

ASRST:  .WORD    DMODRD                                 ;GET OPERAND
        .WORD    ASROP                                  ;SHIFT, STORE RESULT, CODE SET
        .WORD    IFETCH                                 ;FETCH NEXT INSTRUCTION

ASROP:  LSSL DR,16                                      ;LEFT JUSTIFY OPERAND
        RASS DR,+1              /                       ;DO SHIFT
        EXT  R5,MAR,4,015                               ;EXTRACT CCODES
        INS  R5,DR,18,02                                ;INSERT THE SHIFTED OUT BIT
                                                        ;INTO 'C' OF PDP-11 CCODES
        EXT  R7,IR,29,BIT02                             ;EXTRACT DEST. MODE FROM IR
        COND ZERO               /RI  MAR,MODO           ;DESTINATION MODE=0?
        SUBS R4,+2              /SR  R5,CCODES          ;STORE PDP-11 CCODES
        XFR  R5,MAR             /RI  MAR,WRITEW         ;WRITE BACK RESULT
                                /POP MAR,R6,NLT         ;RETURN



BEQ:                                                    ;BEQ INSTRUCTION
                                /LR  R5,CCODES          ;GET PDP-11 CCODES
        INS  MAR,R5,16,03600000                         ;INSERT INTO ICODES BITS
                                                        ;16-19 OF MAR
        COND NZERO              /RI  MAR,IFETCH         ;BRANCH CONDITION FAILS
        EXT  R4,IR,0,0377                               ;EXTRACT OFFSET FROM IR
        LSSS R4,+1              /LR  R7,PDPPC           ;CONVERT OFFSET TO BYTES
                                                        ;GET PC
        ADD  R7,R4              /LR  R4,IFETCH          ;ADD OFFSET TO PC
        EXT  R7,R7,0,LOW16                              ;16 BITS ONLY
        XFR  MAR,R4             /SR  R7,PDPPC           ;STORE NEW PC; RET. TO IFETCH
```

```
MUL:                                                   ;MUL INSTRUCTION
        XFR  R6,MAR            /LR  MAR,MULST          ;SET TOP OF STACK

MULST:  .WORD    DMODRD                                ;GET DEST. OPERAND
        .WORD    MULOP                                 ;MULTIPLY
        .WORD    IFETCH                                ;FETCH NEXT INSTRUCTION

MULOP:  RSSS SR,+6             /RR  R5,DR,MN           ;FINISH EXTRACTING SR FIELD
                                                       ;R5 <= MULTIPLICAND
        CLR  R4                /MR  R7,SR,MN           ;SET UP R4 FOR MULTIPLY
                                                       ;FETCH REG[SR] (MULTIPLIER)
        LSSL R7,16                                     ;LEFT JUSTIFY MULTIPLIER
        ADDS R6,+1             /LR  DR,FIFTEEN         ;SET MULTIPLY STEP COUNT
MULSTP: MULT R4,R7             /DEC DR,1,LGT,MULSTP
                                                       ;EXECUTE FIFTEEN STEPS
        TEST R5                /                       ;MULTIPLICAND POS OR NEG
        RASS R4,+1             /BRN LOWON,NEG          ;SINGLE RIGHT ARITHMETIC
                                                       ;SHIFT IN ANY CASE
POS:    RASS R4,+1             /RI  MAR,CCSET          ;MULTIPLICAND POSITIVE: NO
                                                       ;CORRECTION REQUIRED
NEG:    SUB  R4,R7             /                       ;MULTIPLICAND NEGATIVE:
                                                       ;MAKE CORRECTION
        RASS R4,+1             /BRN NOOVF,CCSET        ;OVERFLOW ON SUBTRACTION?
        COM  R7                /                       ;THERE WAS OVERFLOW:
                                                       ;FLIP SIGN OF MULTIPLIER
        INS  R4,R7,0,020000000000                      ;AND MAKE IT THE SIGN
                                                       ;OF THE RESULT
CCSET:  EXT  DR,MAR,4,BIT03                            ;SAVE CONDITION CODES SET
                                                       ;BY 32 BIT RESULT NOW IN R4
        EXT  R5,R4,0,LOW16                             ;EXTRACT LOW ORDER 16 BITS
                                                       ;OF PRODUCT AND PLACE IN R5
        EXT  R7,R4,17,0377777                          ;SAVE MOST SIGNIFICANT 17
                                                       ;BITS OF PRODUCT
        EXT  R4,R4,16,LOW16                            ;EXTRACT HIGH ORDER 16 BITS
        INS  R7 ,R7,15,037777400000                    ;REPLICATE BITS FOR
                                                       ;SIGNIFICANCE CHECK
        CLR  R7                /BRN DON,STORCC         ;ALL 0'S OR 1'S => DON'T
                                                       ;SET 'C' BIT IN CCODES
        ANDL DR,+2                                     ;TWO 16 BIT WORDS REQUIRED
                                                       ;FOR PRODUCT SO SET 'C'
STORCC: ADDS R7,+1             /SR  DR,CCODES          ;STORE CONDITION CODES
        OR   R7,SR             /RM  SR,R4,MN           ;STORE HIGH ORDER RESULT
                                                       ;IN REG[SR]
                               /RM  R7,R5,MN           ;STORE LOW ORDER RESULT
                                                       ;IN REG[SR OR 1]
                               /POP MAR,R6,NLT         ;RETURN
FIFTEN: .WORD 017
        .END
```