CODE COMPACTNESS AND EXECUTION EFFICIENCY OF VIRTUAL MACHINE PROGRAMMING LANGUAGE ON PDP-11/40E

BY

Ching-Chao Liu

A Research Project Report
submitted to
Department of Computer Science
Oregon State University

in partial fulfillment of
the requirements for the
degree of
Master of Science

September 20, 1979

Code Compactness and Exection Efficiency of
Virtual Machine Programming Language on PDP-11/40E

Table of Contents

1. Introduction To Microprogramming

A digital computer is characterized by the instruction set it executes. Each machine instruction is a bit pattern which is to be interpreted by the central processing unit CPU. The traditional way to interpret a machine instruction is to use random hardware. A contemporary approach is to install a program inside the CPU to interpret the machine instructions. This program is called a microprogram or an emulator. The process of interpreting an instruction set is called emulation. The machine that is characterized by the instruction set is called a virtual machine or a target machine. The real hardware which runs the microprogram is called the host machine.

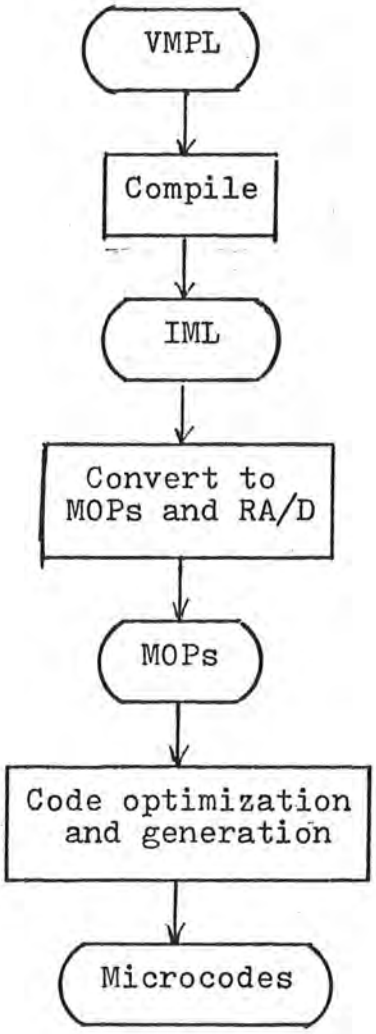
2. The Virtual Machine Programming Language

The virtual machine programming language VMPL [1] [2] is a high level microprogramming language designed for writing virtual machine instruction set interpreters. VMPL is block structured. Each VMPL program consists of a global declaration section and several procedures with their own local declarations. Various data types are provided in VMPL, such as memory, register, flag, field, etc. Two keywords, "GLOBAL" and "LOCAL", are used to define the scope of the variables. Another two keywords, "PERMANENT" and "TEMPORARY", are used to define the priorities of allocation/deallocation of the host

machine resources to the virtual machine resources. Some descriptions of the characteristics of the virtual machine, such as the width of a word, the arithmetic mode, etc., must also be included in the global declaration.

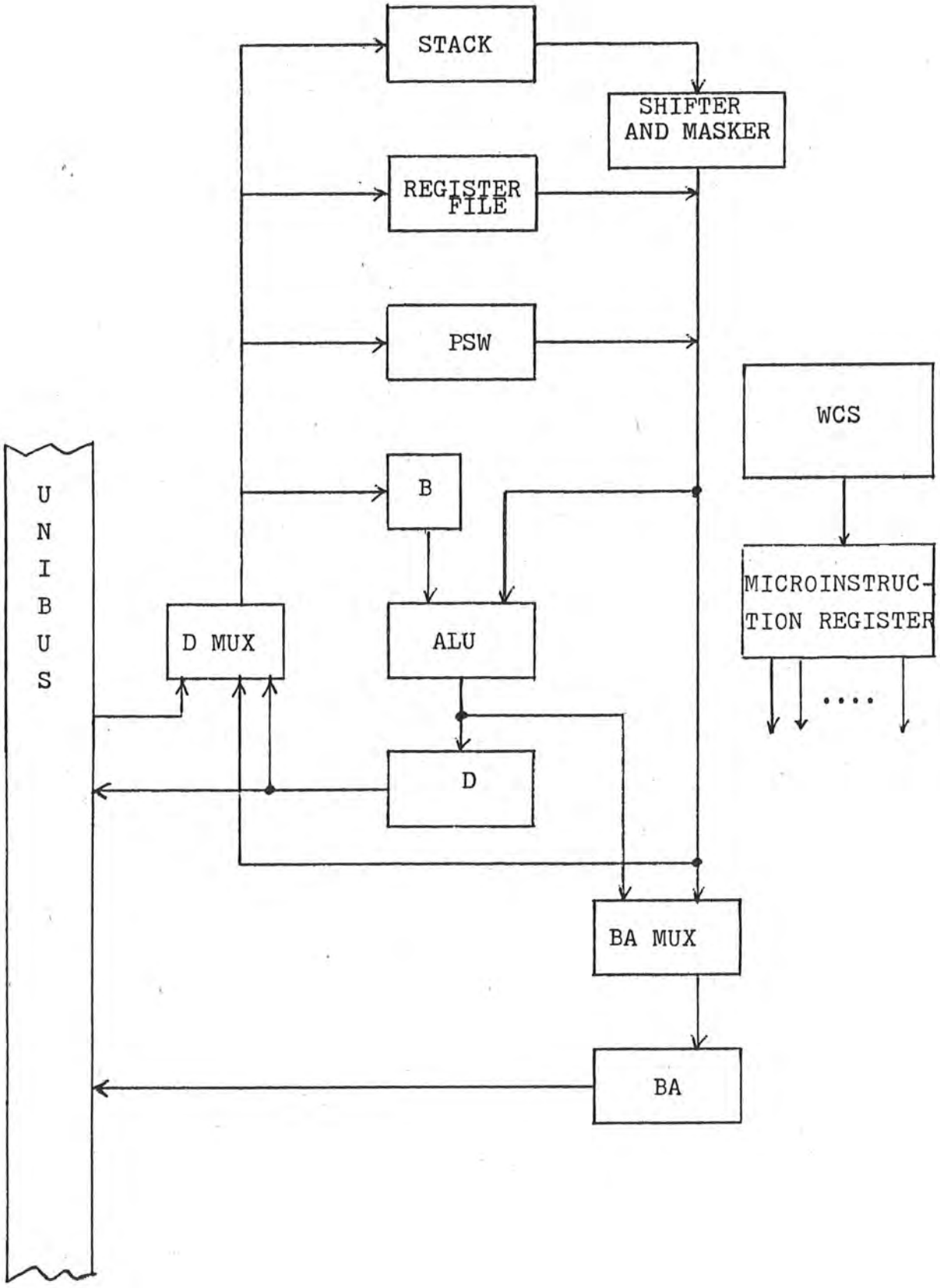
Inside each procedure, there are a section of local declarations and a section of executable statements. The "USE" declaration is designed to explicitly import the global variables inside the scope of the local environment. This feature enhances the reliability of the microprograms written in VMPL. A set of primitive operators is provided to operate on the variables. Each statement may have at most one operator in it. The VMPL compiler will detect potential parallelism between individual operations and may pack them into a single microinstruction.

Microprograms written in VMPL are compiled into an intermediate form, called intermediate language IML [2], which is machine independent and, thus, portable to different host machines. Various tags are used in IML to ease further processing. The IML is then converted into a sequence of machine dependent microoperations MOPs [3]. A register allocation/deallocation RA/D scheme is provided to map the virtual machine resources into host machine registers and memories. Finally, the machine dependent MOPs are optimized and packed into microinstructions, and the microcodes are generated. The full process of VMPL compilation is shown below:



3. The Architecture of PDP-11/40E

The architecture of PDP-11/40E CPU as viewed by a micro-programmer consists of a writable control store WCS where the microprogram is stored, a stack, a register file, a shift and mask unit, an arithmetic and logical unit ALU, and various registers [4]. The logic diagram of the PDP-11/40E is shown below:

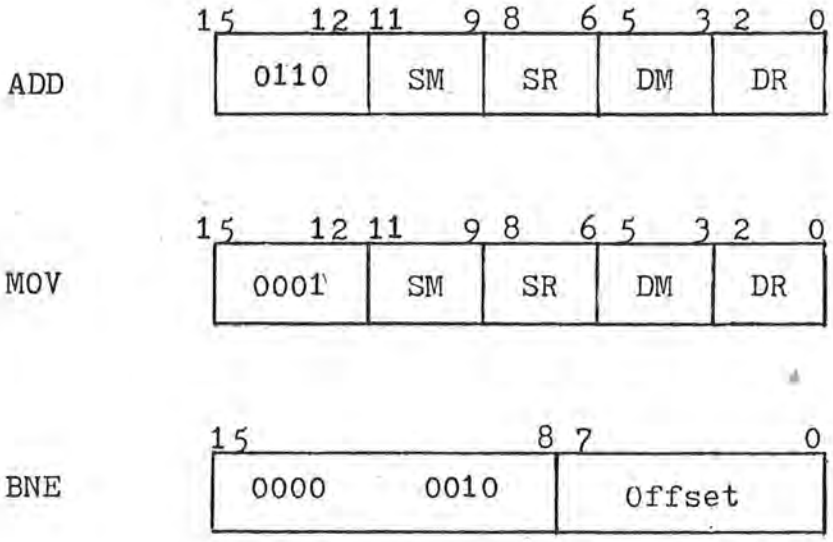

STACK
SHIFTER AND MASKER
REGISTER FILE
PSW
WCS
B
MICROINSTRUC-TION REGISTER
U N I B U S
D MUX
ALU
D
BA MUX
BA

The PDP-11/40E is a multiphase CPU.There are three possible cycle times ranging between 140 nsec and 300 nsec. The shortest cycle is called P1 cycle and ends with a pulse, called the P1 pulse, 140 nsec after start of the microinstruction. The second cycle is called P2 cycle and ends with the P2 pulse at 200 nsec. The longest cycle is called P3 cycle and contains both a P2 pulse at 200 nsec and a final P3 pulse at 300 nsec.

4. Description of The Project

This project is to implement three PDP-11 instructions, namely ADD, MOV, and BNE, by using VMPL. The compactness and execution efficiency of the code generated from the VMPL system are compared with that of the original PDP-11 emulator written in MICRO/40 microassembly language.

The format of the three instructions [5] are as follows:



SM : source mode

SR : source register

DM : destination mode

DR : destination register

The PDP-11/40E provides eight user-addressable registers R0 through R7 where R7 is designated as the program counter. It also allows eight addressing modes. The details of the addressing modes are listed in appendix C.

The VMPL version of the three instructions uses the first eight memory locations as the eight general purpose registers, and they are not shown in the declaration part. Only the working registers are declared. The listing of PDP-11/40E emulator in VMPL is in appendix A and the listing of its counterpart in MICRO/40 is in appendix B.

To facilitate the comparison of execution time,the memory is assumed to have the read access time of 500ns, the write access time of 200ns, and the cycle time 1000ns.

5. Detailed Comparison

5.1 Instruction Fetch

5.1.A VMPL

	Instruction			Timing
FET	MOVE8	14	BA	P1
				MEMR
	MOVE4	UNIBUS	R15	P1
	MOVE8	R15	BA	P1
				MEMR
	MOVE4	UNIBUS	R14	P1

Execution time $\equiv T_v$

$= 1780$ ns

# of instructions $\equiv N_v = 4$

5.1.B MICRO/40

Instruction	Timing
FET01 : P1 ; ba $\leftarrow$ rpc ; dati ; clkoff ; goto FET03	P1 MEMR
FET03 : P1 ; b,ir,rir $\leftarrow$ unibus ; goto FET04	P1

Execution time $\equiv T_m = 2$ P1 + MEMR= 780 ns

# of instructions $\equiv N_m = 2$

5.1.C Comparison

$$\text{Execution time ratio} \equiv \frac{\text{execution time of VMPL}}{\text{execution time of MICRO/40}}$$

$$= \frac{T_v}{T_m} = 2.3$$

$$\text{Code compactness ratio} \equiv \frac{\text{\# of instructions in VMPL}}{\text{\# of instructions in MICRO/40}}$$

$$= \frac{N_v}{N_m} = 2$$

5.2 Increment Program Counter and Decoding

5.2.A VMPL

Instruction		Timing
MOVE7 2	B	P3

```
      ADD R15  B   D ; MOVE5 D     R15                    P3
      MOVE2  14              BA                           P1
      MOVE9  R15             D                            P2
                                                          MEMW
      NOOP                                                P1
DECD  PUSH1  R14             TOS                          P2
      RSMK   TOS     OP      D                            P2
      MOVE5  D               R13                          P3
      PUSH1  R13             TOS                          P2
      LMASK1 TOS     3       EUBC                         P2
      NOOP   XUPF    P.001                                P1
```

T_v = 2520 ns

N_v = 11

5.2.B MICRO/40

Instruction	Timing
FET04 : P2 ; ba,d ⇐ rpc + 2 ; but 37 ; dati	P2
FET05 : P1 ; rpc ⇐ d	P1

Execution time T_m = 340 ns

# of instructions N_m = 2

5.2.C Comparison

Execution time ratio : $\frac{T_v}{T_m} = 7.4$

Code compactness ratio : $\frac{N_v}{N_m} = 5.5$

5.3 Interpretation of BNE Instruction

5.3.A VMPL

	Instruction				Timing
BNE	PUSH1	Z		TOS	P2
	LMASK1	TOS	2	EUBC	P2
	NOOP	XUPF	P.003		P1
BR	PUSH1	R14		TOS	P2
	RSMK	TOS	MAGN	D	P2
	MOVE5	D		R10	P3
	PUSH1	R14		TOS	P2
	RMASK1	TOS	7	EUBC	P2
	NOOP	XUPF	P.004		P1
	BRCH	L.01	P.004		*
	MOVE7	65024		B	P3
	OR R10 B D ; MOVE5 D R10				P3
L.01	MOVE8	14		BA	P1
	MOVE4	UNIBUS		R15	P1
	MOVE1	R10		B	P1
	ADD R15 B D ; MOVE5 D R15				P3
	MOVE2	14		BA	P1
	MOVE9	R15		D	P2
					MEMW
	NOOP				P1
	MOVE12	OFFSET		BA	P1
	MOVE9	R10		D	P2

* Pseudo code,not counted in calculation.

```
                                                        MEMW
      NOOP                                               P1
NBR   UNJP   FET                                         P1
```

Execution time:

No branch : T_V = 680 ns

Branch forward : T_V = 3200 ns

Branch backward : T_V = 3800 ns

# of instructions :

N_V = 22

5.3.B MICRO/40

Instruction	Timing
NBR00 : P1 ; disp.ir ; but 16	P1
SOB06 : P1 ; disp.ir ; goto FET02	P1
BRA00 : P2 ; d ← rpc + b<el>	P2
BRA01 : P1 ; rpc ← d ; but 16	P1
BRA02 : P3 ; d ← rpc + b<el> ; rpc ← d ; goto FET02	P3

Execution time :

No branch : T_m = 280 ns

Branch : T_m = 640 ns

# of instructions :

N_m = 5

5.3.C Comparison

Execution time ratio :

No branch : $\frac{T_v}{T_m} = 2.4$

Branch forward : $\frac{T_v}{T_m} = 5.0$

Branch backward : $\frac{T_v}{T_m} = 5.9$

Code compactness ratio : $\frac{N_v}{N_m} = 4.4$

5.4 Interpretation of Source Operand Fetching

5.4.1 Mode 1

5.4.1.A VMPL

Instruction			Timing
MOVE8	R12	BA	P1
			MEMR
MOVE4	UNIBUS	R11	P1
RETURN	RETADR	EUBC	P2
NOOP1	XUPF 0		P1
MOVE8	R11	BA	P1
			MEMR
MOVE4	UNIBUS	R10	P1

T_v = 1900 ns

N_v = 6

5.4.1.B MICRO/40

Instruction	Timing
SRC00 : P1 ; ba ← rsf ; dati ; allow.odd	P1

SRC14 : P1 ; clkoff ; disp.rir ; but 35 P1,MEMR*

SRC15 : P1 ; b,rsrc←unibus ; goto DOP15 P1

T_m = 780 ns

N_m = 3

*Memory cycle in parallel with CPU cycle. The actual time taken is the longest of the two.

5.4.1.C Comparison

Execution time ratio : $\frac{T_v}{T_m} = 2.4$

Code compactness ratio : $\frac{N_v}{N_m} = 2$

5.4.2 Mode 2

5.4.2.A VMPL

Instruction				Timing
MOVE8	R12		BA	P1
				MEMR
MOVE4	UNIBUS		R11	P1
MOVE7	2		B	P3
ADD	R11	B	D	P2
MOVE5	D		R15	P3
MOVE2	R12		BA	P1
MOVE9	R15		D	P2
				MEMW
NOOP				P1

```
RETURN RETADR EUBC                                    P2
NOOP1  XUPF   0                                       P1
MOVE8  R11            BA                              P1
                                                      MEM
MOVE4  UNIBUS         R10                             P1
```

T_v = 3560 ns

N_v = 12

5.4.2.B MICRO/40

Instruction	Timing
SRC01 : P2 ; ba ← rsf ; dati ; allow.odd ; d ← rsf + c[3]	P2
SRC03 : P1 ; rsf ← d ; clkoff ; but 35 ; goto SRC15	P1,MEMR
SRC15 : P1 ; b,rsrc ← unibus ; goto DOP15	P1

T_m = 840 ns

N_m = 3

5.4.2.C Comparison

Execution time ratio : $\frac{T_v}{T_m} = 4.2$

Code compactness ratio : $\frac{N_v}{Nm} = 4$

5.4.3 Mode 3

5.4.3.A VMPL

Instruction	Timing
MOVE8 R12 BA	P1

```
                                                          MEMR
MOVE4   UNIBUS          R11                               P1
MOVE7   2               B                                 P3
ADD     R11     B       D                                 P2
MOVE5   D               R15                               P3
MOVE2   R12             BA                                P1
MOVE9   R15             D                                 P2
                                                          MEMW
NOOP                                                      P1
MOVE8   R11             BA                                P1
                                                          MEMR
MOVE4   UNIBUS          R11                               P1
RETURN  RETADR          EUBC                              P2
NOOP1   XUPF    0                                         P1
MOVE8   R11             BA                                P1
                                                          MEMR
```

T_V = 4680 ns

N_V = 14

5.4.3.B MICRO/40

Instruction	Timing
SRC04 : P3 ; ba ← rsf ; dati ; d ← rsf + 2 ;	P3
rsf ← d ; clkoff	
	MEMR
SRC12 : P1 ; b,rsrc ← unibus	P1
SRC13 : P1 ; ba ← rsrc ; dati ; allow.odd ; goto SRC14	P1
SRC14 : P1 ; clkoff ; disp.rir ; but 35	P1,MEMR

SRC15 : P1 ; b,rsrc ← unibus ; goto DOP15 P1

T_m = 1580 ns

N_m = 5

5.4.3.C Comparison

Execution time ratio : $\frac{T_v}{T_m}$ = 3.0

Code compactness ratio : $\frac{N_v}{N_m}$ = 2.8

5.4.4 Mode 4

5.4.4.A VMPL

Instruction				Timing
MOVE8	R12		BA	P1
				MEMR
MOVE4	UNIBUS		R15	P1
MOVE7	2		B	P3
SUB	R15	B	D	P2
MOVE5	D		R11	P3
MOVE2	R12		BA	P1
MOVE9	R11		D	P2
				MEMW
NOOP				P1
RETURN	RETADR		EUBC	P2
NOOP1	XUPF	0		P1
MOVE8	R11		BA	P1

			MEMR
MOVE4	UNIBUS	R10	P1

T_v = 3680 ns

N_v = 12

5.4.4.B MICRO/40

Instruction	Timing
SRC02 : P2 ; d,ba ← rsf − c[3] ; dati ; allow.odd ; goto SRC14	P2
SRC14 : P1 ; clkoff ; disp.rir ; but 35	P1,MEMR
SRC15 : P1 ; b,src ← unibus ; goto DOP15	P1

T_m = 840 ns

N_m = 3

5.4.4.C Comparison

Execution time ratio: $\frac{T_v}{T_m} = 4.2$

Code compactness ratio : $\frac{N_v}{N_m} = 4$

5.4.5 Mode 5

5.4.5.A VMPL

Instruction			Timing
MOVE8	R12	BA	P1
			MEMR
MOVE4	UNIBUS	R15	P1
MOVE7	2	B	P3

```
SUB   R15  B   D ; MOVE5  D      R15                    P3
MOVE2  R12           BA                                 P1
MOVE9  R15           D                                  P2
                                                        MEMW
NOOP                                                    P1
MOVE8  R15           BA                                 P1
                                                        MEMR
MOVE4  UNIBUS        R11                                P1
RETURN RETADR        EUBC                               P2
NOOP1  XUPF   0                                         P1
MOVE8  R11           BA                                 P1
                                                        MRMR
MOVE4  UNIBUS        R10                                P1
```

T_V = 4480 ns

N_V = 13

5.4.5.B MICRO/40

Instruction	Timing
SRC05 : P3 ; d,ba←rsf - 2 ; dati ;	P3
rsf←d ; clkoff ; goto SRC12	
	MEMR
SRC12 : P1 ; b,src←unibus	P1
SRC13 : P1 ; ba←rsrc ; dati ; allow.odd ; goto SRC14	P1
SRC14 : P1 ; clkoff ; disp.rir ; but 35	P1,MEMR
SRC15 : P1 ; b,rsrc←unibus ; goto DOP15	P1

T_m = 1940 ns

N_m = 5

5.4.5.C Comparison

Execution time ratio : $\frac{T_v}{T_m}$ = 2.3

Code compactness ratio : $\frac{N_v}{N_m}$ = 2.6

5.4.6 Mode 6

5.4.6.A VMPL

Instruction			Timing
MOVE8	14	BA	P1
			MEMR
MOVE4	UNIBUS	R15	P1
MOVE8	R15	BA	P1
			MEMR
MOVE4	UNIBUS	R11	P1
MOVE7	2	B	P3
ADD	R15 B D ; MOVE5	D R15	P3
MOVE2	14	BA	P1
MOVE9	R15	D	P2
			MEMW
NOOP			P1
MOVE10	REG	D	P2
MOVE5	D	R12	P3
MOVE8	R12	BA	P1

```
                                                              MEMR
    MOVE4  UNIBUS          R15                                P1
    MOVE1  R15             B                                  P1
    ADD  R11  B  D ; MOVE5  D      R11                        P3
    RETURN RETADR          EUBC                               P2
    NOOP1  XUPF   0                                           P1
    MOVE8  R11             BA                                 P1
                                                              MEMR
    MOVE4  UNIBUS          R10                                P1
```

T_v = 5920 ns

N_v = 19

5.4.6.B MICRO/40

Instruction	Timing
SRC06 : P3 ; d ← rpc + 2 ; rpc ← d ; clkoff	P3,MEMR
SRC07 : P1 ; b,rsrc ← unibus	P1
SRC08 : P2 ; ba ← rsf + b ; dati ; allow.odd ; goto SRC14	P2
SRC14 : P1 ; clkoff ; disp.rir ; but 35	P1,MEMR
SRC15 : P1 ; b,rsrc ← unibus ; goto DOP15	P1

T_m = 1640 ns

N_m = 5

5.4.6.C Comparison

Execution time ratio : $\frac{T_v}{T_m} = 3.6$

Code compactness ratio : $\frac{N_v}{N_m} = 3.8$

5.4.7 Mode 7

5.4.7.A VMPL

Instruction				Timing
MOVE8	14	BA		P1
				MEMR
MOVE4	UNIBUS	R15		P1
MOVE8	R15	BA		P1
				MEMR
MOVE4	UNIBUS	R11		P1
MOVE7	2	B		P3
ADD	R15 B D ; MOVE5	D	R15	P3
MOVE2	14	BA		P1
MOVE9	R15	D		P2
				MEMW
NOOP				P1
MOVE10	REG	D		P2
MOVE5	D	R12		P3
MOVE8	R12	BA		P1
				MEMR
MOVE4	UNIBUS	R15		P1
MOVE1	R15	B		P1
ADD	R11 B D ; MOVE5	D	R11	P3
MOVE8	R11	BA		P1

```
                                                          MEMR
    MOVE4  UNIBUS          R11                             P1
    RETURN RETADR          EUBC                            P2
    NOOP1  XUPF   0                                        P1
    MOVE8  R11             BA                              P1
                                                          MEMR
    MOVE4  UNIBUS          R10                             P1
```

T_v = 6140 ns

N_v = 21

5.4.7.B MICRO/40

Instruction	Timing
SRC09 : P3 ; d←rpc + 2 ; rpc←d ; clkoff	P3,MEMR
SRC10 : P1 ; b,rsrc←unibus	P1
SRC11 : P2 ; ba←rsf + b ; dati ; clkoff ; goto SRC12	P2
	MEMR
SRC12 : P1 ; b,rsrc←unibus	P1
SRC!# : P1 ; ba←rsrc ; dati ; allow.odd ; goto SRC14	P1
SRC14 : P1 ; clkoff ; disp.rir ; but 35	P1,MEMR
SRC15 : P1 ; b,rsrc←unibus ; goto DOP15	P1

T_m = 2640 ns

N_m = 7

5.4.7.C Comparison

Execution time ratio : $\frac{T_v}{T_m}$ = 2.3

Code compactness ratio : $\frac{N_v}{N_m} = 3$

5.5 Destination Operand Fetch

Destination operand fetch is similar to source operand fetch and will not be described here.

5.6 Interpretation of ADD Operation

5.6.A VMPL

Instruction								Timing
ADD	R15	B	D	; MOVE5	D		R15	P3
MOVE2	R11			BA				P1
MOVE9	R15			D				P2
								MEMW
NOOP								P1

T_v = 980 ns

N_v = 4

5.6.B MICRO/40

Instruction	Timing
DOP08 : P2 ; d←rsrc ; f/ir/ ;	P2
dato/datob ; goto DOP12	
DOP12 : P1 ; n.z.v.c. ; clkoff ;	P1,MEMW
inhibit ; disp.d ; but 16	
DOP20 : P1 ; disp.d ; goto FET02	P1

T_m = 540 ns

N_m = 3

5.6.C Comparison

Execution time ratio : $\frac{T_v}{T_m}$ = 1.8

Code compactness ratio : $\frac{N_v}{N_m}$ = 1.3

5.7 Interpretation of MOV Operation

5.7.A VMPL

Instruction			Timing
MOVE2	R11	BA	P1
MOVE9	R10	D	P2
			MEMW
NOOP			P1
MOVE12	MAR	BA	P1
MOVE9	R11	D	P2
			MEMW
NOOP			P1
UNJP	FET		P1

T_v = 1820 ns

N_v = 7

5.7.B MICRO/40

Instruction	Timing
DOP08 : P2 ; d←rsrc ; f/ir/ ; dato/datob ;	P2

goto DOP12

DOP12 : P1 ;n.z.v.c. ; clkoff ; inhibit ; P1,MEMW

disp.d ; but 16

DOP20 : P1 ; disp.d ; goto FET02 P1

T_m = 840 ns

N_m = 3

5.7.3 Comparison

Execution time ration : $\frac{T_v}{T_m}$ = 3.4

Code compactness ration : $\frac{N_v}{N_m}$ = 2.3

5.8 Summarized Results

Summarized results are shown in Table 1.

5.9 Total Emulator Length and Execution Time

5.9.A VMPL

There are 111 P1 cycle instructions, 77 P2 cycle instructions, and 38 P3 cycle instructions, a total of 226 instructions in VMPL version. Among them, there are 21 memory read and 22 memory write references. The total microinstruction time is 57240 ns.

5.9.B MICRO/40

There are 30 P1 cycle instructions, 19 P2 cycle instructions, and 14 P3 cycle instructions, a total of 63

Table 1. Summarized Results

		Execution time ratio $\frac{Tv}{Tm}$	Code compactness ratio $\frac{Nv}{Nm}$
Instruction fetch		2.3	2.0
Increment PC and decoding		7.4	5.5
Interpretation of BNE instruction	no branch	2.4	4.4
	branch forward	5.0	4.4
	branch backward	5.9	4.4
Source operand fetch	mode 1	2.4	2.0
	mode 2	4.2	4.0
	mode 3	3.0	2.8
	mode 4	4.2	4.0
	mode 5	2.3	2.6
	mode 6	3.6	3.8
	mode 7	2.3	3.0
Interpretation of ADD operation		1.8	1.3
Interpretation of MOV operation		3.4	2.3
Average		3.6	3.3
Standard deviation		1.6	1.2

instructions. Among them, there are 23 memory read and 2 memory write references. The total microinstruction time is 24100 ns.

5.9.C Comparison

Execution time ratio = 2.4

Code compactness ratio = 3.6

6. Summary

The VMPL version as compared to the MICRO/40 version is less compact and takes more time to execute. One reason is that PDP-11/40E architecture has some hardware features that can be used to speed up the interpretation of PDP-11 instruction set. The MICRO/40 version takes full advantage of the hardware, but the VMPL version doesn't. Another reason is that the VMPL system does not generate efficient code as it could. A more detailed study is presented below, and suggestions are made to improve the VMPL system.

6.1 Hardware Related Features

(1) In PDP-11 instruction set, the SR and DR fields contain the indexes to the register file. The architecture of PDP-11/40E allows direct addressing of the register file from these fields of the instruction register. The MICRO/40 makes full use of this hardware feature and is more compact and efficient.

This hardware feature is instruction set dependent. The VMPL can not take advantage of it because the VMPL is a general modeling system.

Besides, the VMPL does not have the register array data type. For emulators in VMPL, if the registers are to be allocated in CPU, the microprogram must extract SR or DR field, decode it, and branch according to the index. As a result, multiple copies of codes must be written for each register in the register file, and the codes become too large to be handled by the current VMPL compiler system. In this project, registers are put in memory so that they may be addressed through the memory address register and the codes are shorter. However, for each reference to a register, the VMPL version takes two more micro-instructions and one more memory reference as compared to its MICRO/40 counterpart.

The study strongly suggests that a register array data type be included in the VMPL language. When binding the register array of the virtual machine to host machine resources, the register array should be allocated and deallocated as a whole block. In addition, there must be hardware that can be used as a general mechanism to address the register file of the host machine. On PDP-11/40E, the lower order four bits of BA register could be used for this purpose. This addressing feature should

be properly described in the host machine model so that the code generator could use it to generate better codes.

(2) The PDP-11/40E decoding hardware can do OP code decoding, addressing mode decoding, and condition codes testing in one step. This feature makes the codes of the MICRO/40 version six times faster and four times more compact than that of the VMPL version which uses the shift and mask unit to decode the instruction. This is clearly shown in Table 1 that the decoding is the bottle neck of efficiency of emulators in VMPL.

This study reveals that a general decoding hardware is needed on a user microprogrammable host machine. A possible implementation is to use a random access memory RAM. The VMPL compiler could generate a decoding table for the RAM with OP codes as addresses and the corresponding entry points as contents. This table can be loaded into the decoding RAM during emulating time and fast decoding can be achieved.

(3) The MICRO/40 version makes use of the hardware feature of direct ALU function select from the OP code field of the instruction register. This allows the MICRO/40 version to use one piece of code to interpret several instructions such as ADD and MOV. Hence, it is more compact than the VMPL version.

(4) In microprogramming PDP-11/40E, there exist three kinds of parallelism:

(i) CPU operations in parallel with memory operation.

(ii) one micro-operation in parallel with another micro-operation in the CPU.

(iii) microinstruction execution in parallel with next microinstruction fetch.

The VMPL only detects the second type of parallelism and packs parallel micro-operations into one microinstruction if there are no resource conflicts.

The MICRO/40 version uses all three kinds of parallelism and is more efficient.

Suitable descriptions of potential parallelism of host machine should be put in the machine model so that the VMPL can detect parallelism and generate more efficient code.

(5) The MICRO/40 version uses the sign extension hardware to do sign extension. The VMPL version does the sign extension by decoding the sign bit, then ORing the extension bit, which is much slower.

Improvement could be made by including the sign extension hardware in the host machine model. In addition, a sign extension operator should be included in the VMPL so

that we can map the virtual machine operator to the host machine hardware and generate efficient code.

(6) PDP-11/40E has a ROM which contains all the constants used in interpreting PDP-11 instruction set. The MICRO/40 makes full use of this ROM and feeds the constant directly into the ALU. On the other hand, the VMPL version has to derive the constant from the EMIT field of the micro-instruction and clock it into the B register before it can be fed into the ALU.

To take advantage of the constant ROM, it should be modelled in the host machine model.

(7) The bus structure of the PDP-11/40E requires that after each write cycle there must be a P1 or a P3 cycle to release the bus, yet it may not be a read cycle for the bus will not be able to handle it. To guarantee that this constraint is not violated, in VMPL version a NOOP is always inserted after each write cycle which is sometimes unnecessary.

6.2 Software Related Features

(1) In VMPL system, the RA/D scheme generates extra memory read/write operations to swap the virtual machine variables back and forth between the host machine registers and host machine memory. Techniques can be developed to

improve the RA/D scheme.

One possible way is to redefine the meaning of the "PERMANENT" and "TEMPORARY" feature in the VMPL declaration part. Let the "PERMANENT" mean that the variable will be referenced beyond current instruction cycle, and the "TEMPORARY" mean that the variable will be limited to be referenced in current instruction cycle. Then, the temporary variable can be mapped to a working register of the host machine and microinstructions not needed to copy it into a general purpose register. Whenever a temporary variable is released, the register could be freely assigned to another variable without saving its content back into memory. Another important point is that proper choosing of the initial state of register assignments may greatly reduce the allocation/deallocation frequency, and thus increase the efficiency and the compactness of the code. The current RA/D scheme assigns all blanks as the initial state of register assignment. A possibly better way is to assign the most frequently referenced variables to the registers initially.

(2) As shown in the source operand fetch, the code of MICRO/40 version is more compact for the common instructions are overlapped by using the "goto" statement. While VMPL is a well structured language, these codes are duplicated for each mode of operand fetch which makes the VMPL version less

compact. In VMPL, common codes may be grouped into subroutines to achieve code compactness. However, some execution efficiency will be lost due to subroutine linkage.

Another interesting point to notice is that the execution time ratio of the whole emulator is considerably smaller than the average figure shown in Table 1. The reason is that all operand fetches are through the subprocedure ADR in VMPL which results in an almost equal number of memory read references as that of the MICRO/40 version. The MICRO/40 uses a piece of code for source operand fetch.

In general, the VMPL is an easy-to-use microprogramming system in terms of programming effort and debugging effort. But it is not efficient enough. As summarized above, improvements on both the VMPL compiler itself and the microprogrammable hardware are needed to increase the efficiency of the codes generated from the VMPL system.

References

[1] T. G. Lewis, Kamran Malik, and Perng-Yi Ma, "Firmware Engineering Using A High-level Microprogramming System To Implement Virtual Instruction Set Processors", Research Report, Oregon State University.

[2] Kamran Malik, "Design A High-level Microprogramming Language", Doctoral Thesis, Oregon State University.

[3] Perng-Yi Ma, "Optimizing The Microcode Produced By A High-level Microprogramming Language", Doctoral Thesis, Oregon State University.

[4] S. H. Fuller, G. T. Almes, W. H. Broadley, C. L. Forgy, P. L. Karlton, V. R. Lesser, and J. R. Teter, "PDP-11/40E Microprogramming Reference Manual", Carnegie-Mellon University.

[5] Digital Equipment Corporation, "PDP-11/40 Processor Handbook".

Appendix A

Emulator of PDP-11 in VMPL

A.1 Source VMPL Listing

```
[*
  PDP-11 EMULATOR
*]
EMULATOR: PDP11S ;
DCL WORDSIZE 16 ;
DCL ARITHMETIC 2 ;
DCL GLOBAL PERMANENT MEMORY MEM:[1024] ;
DCL GLOBAL TEMPORARY IR,MAR,MDR,MOD,REG ;
DCL GLOBAL PERMANENT FLAG N,Z,O,C ;
DCL FIELD OP(11, 3,-11), SMOD(9,11,-8), SREG(6,8,-5),
          DMOD(3,5,-2), DREG(0,2,1), MAGN(0,6,1) ;
[*
  INSTRUCTION FETCH
*]
PROC: FET ;
 DCL GLOBAL USE IR,MEM,MDR ;
≡
 MDR=MEM[14] ;
 IR=MEM[MDR] ;
 MDR=MDR+2 ;
 MEM[14]=MDR ;
≡
[*
  INSTRUCTION DECODING
*]
PROC: DECD ;
 DCL GLOBAL USE IR ;
 DCL LOCAL TEMPORARY OPCD ;
```

```
≡
 OPCD=OP(IR) ;
 SELECT (OPCD,8) FROM ;
  (0,BNE) ;
  (1,NOTIMP) ;
  (2,MOV) ;
  (3,MOV) ;
  (4,ADD) ;
  (5,ADD) ;
  (6,NOTIMP) ;
  (7,NOTIMP) ;
 ENDSELECT ;
≡
[*
  EFFECTIVE ADDRESS
*]
SPROC: ADR ;
 DCL GLOBAL EXPECT MOD,REG ;
 DCL GLOBAL RETURN MAR ;
 DCL SPROC USE MDR ;
≡
 SELECT (MOD,8) FROM ;
  (0,MOD0) ;
  (1,MOD1) ;
  (2,MOD2) ;
  (3,MOD3) ;
  (4,MOD4) ;
  (5,MOD5) ;
  (6,MOD6) ;
  (7,MOD7) ;
 ENDSELECT ;
```

```
MOD0:
 BEGIN ;
  MAR=REG ;
  RETURN ;
 END ;
MOD1:
 BEGIN ;
  MAR=MEM[REG] ;
  RETURN ;
 END ;
MOD2:
 BEGIN ;
  MAR=MEM[REG] ;
  MDR=MAR+2 ;
  MEM[REG]=MDR ;
  RETURN ;
 END ;
MOD3:
 BEGIN ;
  MAR=MEM[REG] ;
  MDR=MAR+2 ;
  MEM[REG]=MDR ;
  MAR=MEM[MAR] ;
  RETURN ;
 END ;
MOD4:
 BEGIN ;
  MDR=MEM[REG] ;
  MAR=MDR-2 ;
  MEM[REG]=MAR ;
  RETURN ;
 END ;
```

```
MOD5:
 BEGIN ;
  MDR=MEM[REG] ;
  MDR=MDR-2 ;
  MEM[REG]=MDR ;
  MAR=MEM[MDR] ;
  RETURN ;
 END ;
MOD6:
 BEGIN ;
  MDR=MEM[14] ;
  MAR=MEM[MDR] ;
  MDR=MDR+2 ;
  MEM[14] ;
  MDR=MEM[REG] ;
  MAR=MAR+MDR ;
  RETURN ;
 END ;
MOD7:
 BEGIN ;
  MDR=MEM[14] ;
  MAR=MEM[MDR] ;
  MDR=MDR+2 ;
  MEM[14]=MDR ;
  MDR=MEM[REG] ;
  MAR=MAR+MDR ;
  MAR=MEM[MAR] ;
  RETURN ;
 END ;
≡
[*
  BNE INSTRUCTION
*]
```

```
PROC: BNE ;
 DCL GLOBAL USE IR,MDR,MEM ;
 DCL LOCAL TEMPORARY OFFSET ;
≡
 SELECT (Z,2) FROM ;
  (0,BR) ;
  (1,NBR) ;
 ENDSELECT ;

 BR:
  BEGIN ;
   OFFSET=MAGN(IR) ;
   IFTRUE (IR,7) THEN ;
    BEGIN ;
     OFFSET=OFFSET .OR. 65024 ;
    END ;
   ENDIF ;
   MDR=MEM[14] ;
   MDR=MDR+OFFSET ;
   MEM[14]=MDR ;
  END ;

 NBR:
  LEAVE FET ;
≡
[*
  MOV INSTRUCTION
*]
PROC: MOV ;
 DCL GLOBAL USE MOD,REG,MAR,IR,MEM ;
 DCL LOCAL TEMPORARY RSRC ;
≡
 MOD=SMOD(IR) ;
```

```
 REG=SREG(IR) ;
 EXECUTE ADR ;
 RSRC=MEM[MAR] ;
 MOD=DMOD(IR) ;
 REG=DREG(IR) ;
 EXECUTE ADR ;
 MEM[MAR]=RSRC ;
 LEAVE FET ;
≡
[*
  ADD INSTRUCTION
*]
PROC: ADD ;
 DCL GLOBAL USE MOD,REG,MAR,MDR,IR,MEM ;
 DCL LOCAL TEMPORARY RSRC ;
≡
 MOD=SMOD(IR) ;
 REG=SREG(IR) ;
 EXECUTE ADR ;
 RSRC=MEM[MAR] ;
 MOD=DMOD(IR) ;
 REG=DREG(IR) ;
 EXECUTE ADR ;
 MDR=MEM[MAR] ;
 MDR=MDR+RSRC ;
 MEM[MAR]=MDR ;
 LEAVE FET ;
≡
[*
  OTHER INSTRUCTIONS ARE NOT IMPLEMENTED
*]
PROC: NOTIMP ;
```

```
≡
 LEAVE FET ;
≡
ENDEMULATOR ;
```

A.2 IML Listing

```
00A  PDP11S
00D  ,,,16
00E  TWO
221  MEM,1024,16
210  IR,,16
210  MAR,,16
210  MDR,,16
210  MOD,,16
210  REG,,16
224  N,,1,3
224  Z,,1,4
224  O,,1,2
224  C,,1,1
005  OP,,,11,13,-11
005  SMOD,,,9,11,-8
005  SREG,,,6,8,-5
005  DMOD,,,3,5,-2
005  DREG,,,0,2,1
005  MAGN,,,0,6,1
00B  PROGRAMSTART
00G  ADR
207  MOD
207  REG
208  MAR
406  MDR
00H
```

```
                SLCT    MOD     C8
        *                       C0      SMOD0
        *                       C1      SMOD1
        *                       C2      SMOD2
        *                       C3      SMOD3
        *                       C4      SMOD4
        *                       C5      SMOD5
        *                       C6      SMOD6
        *                       C7      SMOD7
        *
        MOD0
        00J
                MOVE    REG     MAR
                RET
        00K
        MOD1
        00J
                RMOVE   MEM     REG     MAR
                RET
        00K
        MOD2
        00J
                RMOVE   MEM     REG     MAR
                ADD     MAR     C2      MDR
                WMOVE   MEM     REG     MDR
                RET
        00K
        MOD3
        00J
                RMOVE   MEM     REG     MAR
                ADD     MAR     C2      MDR
                WMOVE   MEM     REG     MDR
                RMOVE   MEM     MAR     MAR
00      00K
        MOD4
        00J
```

```
        RMOVE   MEM     REG     MDR
        SUB     MDR     C2      MAR
        WMOVE   MEM     REG     MAR
        RET
00K
MOD5
00J
        RMOVE   MEM     REG     MDR
        SUB     MDR     C2      MDR
        WMOVE   MEM     REG     MDR
        RMOVE   MEM     MDR     MAR
        RET
00K
MOD6
00J
        RMOVE   MEM     C14     MDR
        RMOVE   MEM     MDR     MAR
        ADD     MDR     C2      MDR
        WMOVE   MEM     C14     MDR
        RMOVE   MEM     REG     MDR
        ADD     MAR     MDR     MAR
        RET
00K
MOD7
00J
        RMOVE   MEM     C14     MDR
        RMOVE   MEM     MDR     MAR
        ADD     MDR     C2      MDR
        WMOVE   MEM     C14     MDR
        RMOVE   MEM     REG     MDR
        ADD     MAR     MDR     MAR
        RMOVE   MEM     MAR     MAR
        RET
00K
00I
00F  FET
```

```
206  IR
206  MEM
206  MDR
00H
        RMOVE    MEM     C14     MDR
        RMOVE    MEM      MDR    IR
        ADD      MDR     C2      MDR
        WMOVE    MEM     C14     MDR
00I
00F  DECD
206  IR
110  OPCD,,16
00H
        EXTR     OP       IR     OPCD
        SLCT     OPCD    C8
*                        C0      SBNE
*                        C1      SNOTIMP
*                        C2      SMOV
*                        C3      SMOV
*                        C4      SADD
*                        C5      SADD
*                        C6      SNOTIMP
*                        C7      SNOTIMP
*
00I
00F  BNE
206  IR
206  MDR
206  MEM
110  OFFSET,,16
00H
        SLCT     Z       C2
*                        C0      SBR
*                        C1      SNBR
*
```

```
BR
00J
        EXTR    MAGN    IR      OFFSET
        CONDF   .IR,7   TL.001
00J
        OR      OFFSET  C65024  OFFSET
00K
L.001
        RMOVE   MEM     C14     MDR
        ADD     MDR     OFFSET  MDR
        WMOVE   MEM     C14     MDR
00K
NBR
        BRCH    BFET
00I
00F  MOV
206  MOD
206  REG
206  MAR
206  IR
206  MEM
110  RSRC,,16
00H
        EXTR    SMOD    IR      MOD
        EXTR    SREG    IR      REG
        XEQ     ADR
*
        RMOVE   MEM     MAR     RSRC
        EXTR    DMOD    IR      MOD
        EXTR    DREG    IR      REG
        XEQ
*
        WMOVE   MEM     MAR     RSRC
        BRCH    BFET
00I
```

```
00F  ADD
206  MOD
206  REG
206  MAR
206  MDR
206  IR
206  MEM
110  RSRC,,16
00H
        EXTR     SMOD     IR       MOD
        EXTR     SREG     IR       REG
        XEQ      ADR
*
        RMOVE    MEM      MAR      RSRC
        EXTR     DMOD     IR       MOD
        EXTR     DREG     IR       REG
        XEQ
*
        RMOVE    MEM      MAR      MDR
        ADD      MDR      RSRC     MDR
        WMOVE    MEM      MAR      MDR
        BRCH     BFET
00I
00F  NOTIMP
00H
        BRCH     BFET
00I
00C  PROGRAMEND
```

A.3 Object listing

```
FET   MOVE8   14                BA
      MOVE4   UNIBUS            R15
      MOVE8   R15               BA
      MOVE4   UNIBUS            R14
      MOVE7   2                 B
      ADD     R15     B         D     ; MOVE5  D          R15
      MOVE2   14                BA
      MOVE9   R15               D
      NOOP
DECD  PUSH1   R14               TOS
      RSMK    TOS     OP        D
      MOVE5   D                 R13
      PUSH1   R13               TOS
      LMASK1  TOS     3         EUBC
      NOOP    XUPF    P.001
      UNJP    BNE     P.001     3
      UNJP    NOTIMP  P.001+01  3
      UNJP    MOV     P.001+02  3
      UNJP    MOV     P.001+03  3
      UNJP    ADD     P.001+04  3
      UNJP    ADD     P.001+05  3
      UNJP    NOTIMP  P.001+06  3
      UNJP    NOTIMP  P.001+07  3
      MOVE11  MOD               BA
      MOVE4   UNIBUS            R13
ADR   PUSH1   R13               TOS
      LMASK1  TOS     3         EUBC
```

```
        NOOP    XUPF    P.002
        UNJP    MOD0    P.002     3
        UNJP    MOD1    P.002+01  3
        UNJP    MOD2    P.002+02  3
        UNJP    MOD3    P.002+03  3
        UNJP    MOD4    P.002+04  3
        UNJP    MOD5    P.002+05  3
        UNJP    MOD6    P.002+06  3
        UNJP    MOD7    P.002+07  3
MOD0    MOVE11  REG               BA
        MOVE4   UNIBUS            R12
        MOVE3   R12               D
        MOVE5   D                 R11
        RETURN  RETADR            EUBC
        NOOP1   XUPF    0
MOD1    MOVE10  REG               D
        MOVE5   D                 R12
        MOVE8   R12               BA
        MOVE4   UNIBUS            R11
        RETURN  RETADR            EUBC
        NOOP1   XUPF    0
MOD2    MOVE10  REG               D
        MOVE5   D                 R12
        MOVE8   R12               BA
        MOVE4   UNIBUS            R11
        MOVE7   2                 B
        ADD     R11     B         D
        MOVE5   D                 R15
        MOVE2   R12               BA
        MOVE9   R15               D
        NOOP
        RETURN  RETADR            EUBC
        NOOP1   XUPF    0
MOD3    MOVE10  REG               D
```

```
        MOVE5   D               R12
        MOVE8   R12             BA
        MOVE4   UNIBUS          R11
        MOVE7   2               B
        ADD     R11     B       D
        MOVE5   D               R15
        MOVE2   R12             BA
        MOVE9   R15             D
        NOOP
        MOVE8   R11             BA
        MOVE4   UNIBUS          R11
        RETURN  RETADR          EUBC
        NOOP1   XUPF    0
MOD4    MOVE10  REG             D
        MOVE5   D               R12
        MOVE8   R12             BA
        MOVE4   UNIBUS          R15
        MOVE7   2               B
        SUB     R15     B       D
        MOVE5   D               R11
        MOVE2   R12             BA
        MOVE9   R11             D
        NOOP
        RETURN  RETADR          EUBC
        NOOP1   XUPF    0
MOD5    MOVE10  REG             D
        MOVE5   D               R12
        MOVE8   R12             BA
        MOVE4   UNIBUS          R15
        MOVE7   2               B
        SUB     R15     B       D       ; MOVE5  D          R15
        MOVE2   R12             BA
        MOVE9   R15             D
        NOOP
```

```
        MOVE8   R15             BA
        MOVE4   UNIBUS          R11
        RETURN  RETADR          EUBC
        NOOP1   XUPF    0
MOD6    MOVE8   14              BA
        MOVE4   UNIBUS          R15
        MOVE8   R15             BA
        MOVE4   UNIBUS          R11
        MOVE7   2               B
        ADD     R15     B       D       ; MOVE5  D      R15
        MOVE2   14              BA
        MOVE9   R15             D
        NOOP
        MOVE10  REG             D
        MOVE5   D               R12
        MOVE8   R12             BA
        MOVE4   UNIBUS          R15
        MOVE1   R15             B
        ADD     R11     B       D       ; MOVE5  D      R11
        RETURN  RETADR          EUBC
        NOOP1   XUPF    0
MOD7    MOVE8   14              BA
        MOVE4   UNIBUS          R15
        MOVE8   R15             BA
        MOVE4   UNIBUS          R11
        MOVE7   2               B
        ADD     R15     B       D       ; MOVE5  D      R15
        MOVE2   14              BA
        MOVE9   R15             D
        NOOP
        MOVE10  REG             D
        MOVE5   D               R12
        MOVE8   R12             BA
        MOVE4   UNIBUS          R15
```

```
        MOVE1   R15             B
        ADD     R11     B       D       ; MOVE5  D      R11
        MOVE8   R11             BA
        MOVE4   UNIBUS          R11
        RETURN  RETADR          EUBC
        NOOP1   XUPF    0
BNE     PUSH1   Z               TOS
        LMASK1  TOS     2       EUBC
        NOOP    XUPF    P.003
        UNJP    BR      P.003   2
        UNJP    NBR     P.003+01 2
BR      PUSH1   R14             TOS
        RSMK    TOS     MAGN    D
        MOVE5   D               R10
        PUSH1   R14             TOS
        RMASK1  TOS     7       EUBC
        NOOP    XUPF    P.004
        BRCH    L.01    P.004   1
        MOVE7   65024           B
        OR      R10     B       D       ; MOVE5  D      R10
L.01    MOVE8   14              BA
        MOVE4   UNIBUS          R15
        MOVE1   R10             B
        ADD     R15     B       D       ; MOVE5  D      R15
        MOVE2   14              BA
        MOVE9   R15             D
        NOOP
        MOVE12  OFFSET          BA
        MOVE9   R10             D
        NOOP
NBR     UNJP    FET
MOV     PUSH1   R14             TOS
        RSMK    TOS     SMOD    D
        MOVE5   D               R13
```

```
PUSH1   R14             TOS
RSMK    TOS     SREG    D
MOVE5   D               R12
PUSH
MOVE12  MAR             BA
MOVE9   R11             D
NOOP
MOVE12  REG             BA
MOVE9   R12             D
NOOP
CALL    ADR
MOVE8   R11             BA
MOVE4   UNIBUS          R10
PUSH1   R14             TOS
RSMK    TOS     DMOD    D
MOVE5   D               R13
PUSH1   R14             TOS
RSMK    TOS     DREG    D
MOVE5   D               R12
PUSH
MOVE12  RSRC            BA
MOVE9   R10             D
NOOP
MOVE12  MAR             BA
MOVE9   R11             D
NOOP
MOVE12  REG             BA
MOVE9   R12             D
NOOP
CALL    ADR
MOVE2   R11             BA
MOVE9   R10             D
NOOP
MOVE12  MAR             BA
MOVE9   R11             D
```

```
        NOOP
        UNJP    FET
ADD     PUSH1   R14             TOS
        LSMK    TOS     SMOD    D
        MOVE5   D               R13
        PUSH1   R14             TOS
        LSMK    TOS     SREG    D
        MOVE5   D               R12
        PUSH
        MOVE12  REG             BA
        MOVE9   R12             D
        NOOP
        CALL    ADR
        MOVE8   R11             BA
        MOVE4   UNIBUS          R10
        PUSH1   R14             TOS
        LSMK    TOS     DMOD    D
        MOVE5   D               R13
        PUSH1   R14             TOS
        RSMK    TOS     DREG    D
        MOVE5   D               R12
        PUSH
        MOVE12  RSRC            BA
        MOVE9   R10             D
        NOOP
        MOVE12  MAR             BA
        MOVE9   R11             D
        NOOP
        MOVE12  REG             BA
        MOVE9   R12             D
        NOOP
        CALL    ADR
        MOVE8   R11             BA
        MOVE4   UNIBUS          R15
```

```
        MOVE11  RSRC            BA
        MOVE4   UNIBUS          R10
        MOVE1   R10             B
        ADD     R15     B       D       ; MOVE5  D          R15
        MOVE2   R11             BA
        MOVE9   R15             D
        NOOP
        MOVE12  MAR             BA
        MOVE9   R11             D
        NOOP
        UNJP    FET
NOTIMP  UNJP    FET
```

Appendix B

Emulator of PDP-11 in MICRO/40

```
!  This is a transcription of the standard PDP-11 emulator in
!  MICRO/40.  The sequence and labels of the micro instructions
!  closely follow the flowcharts in the KD11-A Processor
!  Engineering Drawings.
!  This microcode is available on the CMU-10A as
!  ROM256.MIC[N200MU00] and can be used as a starting point
!  when constructions a modified PDP-11 instruction set on the
!  PDP-11/40E microprocessor.
!
!  Orisinal Version: Summer 1975  Rajan Modi
!  Revision 1:       Autumn 1975  Guy Almes

require defs.mic

f/ir/         := dad=14$        ! alu function of ir
allow.odd     := dad=1$         ! allow an odd address
check.ovflo   := dad=6$         ! check stack overflow
ovflo/odd     := dad=7$
dato/datob    := dad=17; dato$
inhibit       := dad=12$        ! inhibit dato/clkoff
clkoff/ovlap  := clk=0$         ! clkoff if overlap fetch
awbby         := bus=2$         ! await bus busy

disp.rir    := sdm=0; srx=1; rif=13$  ! set data to be displayed
disp.rpc    := sdm=0; srx=1; rif=7$
disp.vect   := sdm=0; srx=1; rif=14$
disp.adrsc  := sdm=0; srx=1; rif=17$
disp.rdf    := sdm=0; srx=4$
disp.d      := sdm=2$
disp.bus    := sdm=1$

begin  goto FET02
```

```
        .........
        FET02: p1; ba <- rpc; dati; clkoff; goto FET03
        .........

    !  instruction fetch

    FET01: p1; ba <- rpc; dati; clkoff; goto FET03
    FET03: p1; b,ir,rir <- unibus; goto FET04
    .........

!  increment PC and decode the instruction

FET04: p2; ba,d <- rpc+2; but 37; dati ! if ovlap fetch
FET05: p1; rpc <- d
.........

        !  double operand with register to register operation

        D0P02: p1; b <- rdf; but 31
        D0P03: p2; d <- rsf; f/ir/; but 27
              .........
              D0P19: p1; rdf <- d; n.z.v.c.; clkoff/ovlap;-
                                             goto FET02
              .........

!  branch instructions
!  no branch

NBR00: p1; disp.rir; but 16
S0B06: p1; disp.rir; goto FET02
.........

!  branch
BRA00: p2; d <- rpc+b<el>
```

```
BRA01: p1; rpc <- d; but 16
BRA02: p3; d <- rpc+b<el>; rpc <- d; goto FET02
.........

! source mode 1 operand fetch

SRC00: p1; ba <- rsf; dati; allow.odd
SRC14: p1; clkoff; disp.rir; but 35
SRC15: p1; b,rsrc <- unibus; goto DOP15
.........

!  source mode 2

SRC01: p2; ba <- rsf; dati; allow.odd; d <- rsf+c[3]
SRC03: p1; rsf <- d; clkoff; but 35; goto SRC15
.........

!  source mode 3

SRC04: p3; ba <- rsf; dati; d <- rsf+2; rsf <- d; clkoff
SRC12: p1; b,rsrc <- unibus
SRC13: p1; ba <- rsrc; dati; allow.odd; goto SRC14
.........

!  source mode 4

SRC02: p2; d,ba <- rsf-c[3]; dati; allow.odd; goto SRC14

!  source mode 5

SRC05: p3; d,ba <- rsf-2; dati; rsf <- d; clkoff; goto SRC12
.........

!  source mode 6
```

```
SRC06: p3; d <- rpc+2; rpc <- d; clkoff
SRC07: p1; b,rsrc <- unibus
SRC08: p2; ba <- rsf+b; dati; allow.odd; goto SRC14
.........

!  source mode 7

SRC09: p3; d <- rpc+2; rpc <- d; clkoff
SRC10: p1; b,rsrc <- unibus
SRC11: p2; ba <- rsf+b; dati; clkoff; goto SRC12
.........

!  destination mode 1

DST00: p1; ba <- rdf; datip; ovflo/odd
DST14: p1; disp.rir; clkoff; but 33
DST15: p1; b,rdst <- unibus
.....

        .........

        DOP12: p1; n.z.v.c.; clkoff; inhibit; disp.d; but 16
        DOP20: p1; disp.d; goto FET02

        .........

        ! double operand with source mode 0

        DOP07: p2; d <- rsf; f/ir/; dato/datob; goto DOP12

        ! double operand with source mode not 0

        DOP08: p2; d <- rsrc; f/ir/; dato/datob; goto DOP12

        .........
```

```
          ..........

!  destination mode 2

DST01: p2; ba <- rdf; datip; ovflo/odd; d <- rdf+c[3]
DST03: p1; rdf <- d; clkoff; but 33; goto DST15

..........

!  destination mode 3

DST04: p3; ba <- rdf; dati; d <- rdf+2; rdf <- d; clkoff
DST12: p1; b,rdst <- unibus
DST13: p1; ba <- rdst; datip; allow.odd; goto DST14

..........

!  destination mode 4

DST02: p2; d,ba <- rdf-c[3]; datip; ovflo/odd; goto DST03

!  destination mode 5

DST05: p3; d,ba <- rdf-2; dati; rdf <- d; clkoff

..........

!  destination mode 6

DST07: p3; d <- rpc+2; rpc <- d; clkoff; but 17
DST09: p1; b,rdst <- unibus

       ..........
```

```
      DST10: p2; ba <- rdf+b;datip; ovflo/odd; goto DST14
      DST11: p2; ba <- rdf+b; dati; clkoff; goto DST12
      ..........

!  destination mode 7

DST06: p3; ba <- rpc; dati; d <- rpc+2; rpc <- d; clkoff;-
                                          but 17; goto DST09

!  MOV Rm,Rn

MOV18: p2; d <- rsf; but 20; goto MOV20
..........

!  MOV Rm,@Rn

MOV00: p2; d,ba <- rdf; ovflo/odd; but 22
MOV07: p1; rdf <- d
      ..........
      MOV17: p2; d <- rsf; dato
      ..........
MOV22: p3; n.z.v.c.; disp.d; inhibit; clkoff; but 16;-
                                                goto DOP20
..........

!  MOV Rm,(Rn)+

MOV01: p2; ba <- rdf; d <- rdf+c[3]; ovflo/odd; but 22;-
                                                  goto MOV07
..........

!  MOV Rm,@(Rn)+
```

```
MOV03: p3; ba <- rdf; d <- rdf+2; dati; rdf <- d; clkoff
MOV11: p1; b,rdst <- unibus; but 22
MOV12: p1; ba <- rdst; allow.odd; goto MOV16
.........

!  MOV Rm,-(Rn)

MOV02: p2; d,ba <- rdf-c[3]; ovflo/odd; but 22; goto MOV07

!  MOV Rm,@-(Rn)

MOV04: p3; d,ba <- rdf-2; dati; rdf <- d; clkoff; goto MOV11
.........

!  MOV Rm,X(Rn)

MOV06: p3; d <- rpc+2; rpc <- d; clkoff; but 17
MOV08: p1; b,rdst <- unibus; but 21
       .........
       MOV09: p2; ba <- rdf+b; ovflo/odd; goto MOV16
       .........

!  MOV Rm,@X(Rn)

MOV05: p3; ba <- rpc; dati; d <- rpc+2; rpc <- d; clkoff;-
                                        but 17; goto MOV08
.........

finis
```

Appendix C

Addressing Modes of PDP-11 Instruction Set

Mode	Assembler Syntax	Function
0	Rn	Register contains operand.
1	@Rn	Register contains the address of the operand.
2	(Rn)+	Register is used as a pointer to sequential data then incremented.
3	@(Rn)+	Register is first used as a pointer to a word containing the address of the operand, then incremented.
4	-(Rn)	Register is decremented and then used as a pointer.
5	@-(Rn)	Register is decremented and then used as a pointer to a word containing the address of the operand.
6	X(Rn)	Value X (stored in a word following the instruction) is added to (Rn) to produce address of operand.
7	@X(Rn)	Value X and (Rn) are added and the sum is used as a pointer to a word containing the address of the operand.