AN ABSTRACT OF THE THESIS OF

Perng-Yi Ma for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on October 27, 1978.

Title: OPTIMIZING THE MICROCODE PRODUCED BY A HIGH LEVEL MICROPROGRAMMING LANGUAGE

Abstract approved: Redacted for privacy
(Theodore G. Lewis)

The purpose of this research is to develop methods to translate a certain machine independent intermediate language (IML) to efficient horizontal microprograms for a class of microprogrammable machines. This IML has been developed by Malik (12) and is compiled directly from a high level microprogramming language used to implement a microprogrammed interpreter.

An IML-host machine interface design that allows easy modification for language portability should be a primary objective; i.e., the interface design must be of sufficient power and versatility to generate efficient code for a variety of host machines. Transportability is accomplished by the use of a Field Description Model (FDM) and Macro Table which are used to describe the most machine to the translator system.

A register allocation scheme and control flow analysis are employed to allocate the symbolic variables of

the IML to the general purpose registers of the host machine. Again, with the aid of the FDM, a set of 5-tuple micro-operations (MOP: OP, I/O, field, phase) is obtained. Then an optimization algorithm is used to detect the parallelism of MOPs, and generate efficient code for a horizontal microprogrammable machine. This research terminated with a study of the effects of the above methods upon the quality of microcode produced for a specific commercial computer.

Optimizing the Microcode Produced by a High Level Microprogramming Language

by

Perng-Yi Ma

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed October 27, 1978

Commencement June 1979

APPROVED:

Redacted for privacy

Associate Professor of Computer Science Department
in Charge of Major

Redacted for privacy

Head of Department of Electrical and Computer Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented October 27, 1978

Typed by Clara Homyer for Perng-Yi Ma

TABLE OF CONTENTS

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Acronyms and symbols

ALU	arithmetic logic unit of a computer
CPU	central process unit of a computer
FDM	Field Description Model
FS	final state of a SLC
GPR	general purpose registers of the host machine
IESG	executable statement group of IML
IISG	information statement group of IML
IML	host machine independent intermediate language
IS	initial state of a SLC
MDIL	host machine dependent intermediate language
MET	Macro Expansion Table
MI	microinstruction
MOP	microoperation
NR	the number of general purpose registers in the host machine
RA/D scheme	register allocation and deallocation scheme
SLC	straight line code
β	$M_i \beta M_j$ means M_i is data independent of M_j
$//$	$M_i // M_j$ means M_i is parallel M_j
$><$	$M_i >< M_j$ means M_i is invertible with M_j
$\|MI\|$	the number of MOPs in MI

OPTIMIZING THE MICROCODE PRODUCED BY A HIGH LEVEL MICROPROGRAMMING LANGUAGE

CHAPTER I

INTRODUCTION

1-1 Motivation

Recent research in computer systems organization has shown the need for microprogramming tools (1, 3, 5, 6, 19, 20, 21, 22). Such tools must be able to aid the development of emulators and special purpose processors for high speed applications. For example, the emulation of the IBM 370/158 instruction set is accomplished by a microprogram resident in the control memory of the IBM 370 host.

A microprogram executes from the control memory of a machine which is called the host computer in this research. The host computer emulates a virtual computer by simulating a target instruction set. The terms "target" and "virtual" are often used interchangeably, and designate the same level in a multi-level system as shown in Figure 1-1.

The resident microprogram at the emulator level of Figure 1-1 must be implemented in much the same fashion as any other computer program. Therefore, it is only logical to apply the lessons learned from software engineering to this task. That is, the notions of structured programming, high level languages, and machine independence directly

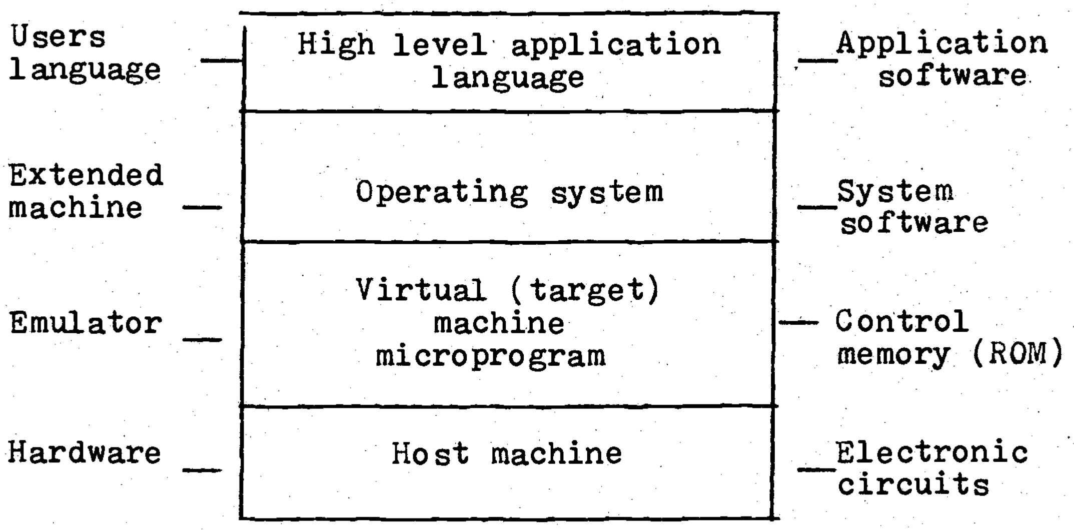


Figure 1-1. A Multi-level Computer System

apply to the problem of reliable, efficient microcode production (15). However, software engineering is extremely difficult to achieve when dealing with microprograms due to the following problems:

Problem #1. Host machines widely vary in their architecture. They may be broadly classified as either horizontal (more than one microoperation may be simultaneously executed from one microinstruction) or vertical (single microoperations per microinstruction typically encoded much like machine code). See references 2, 12, 15) for a detailed discussion of microprogrammable host machines.

Problem #2. Horizontal microinstruction formats offer added speed of machine operation only if concurrent microoperations can be detected and combined into a single microinstruction. A microprogram is said to be optimized if the resulting code is of minimum length (length is equal to the number of microinstructions). DeWitt (7) has proven the NP-completeness of code optimization for machines with horizontal formats. Thus, the approach taken in this research is to concentrate on fast, efficient algorithms that compact the code, but do not guarantee absolute minimum length of code sequence.

Problem #3. Portability. The production of portable, yet compact code for a family of microprogrammable host machines is a topic largely ignored by others. However, the time and effort needed to produce an emulator should not be wasted when changing the host. Indeed, the emulation should be transferable to a number of different host machines with little added effort. A portable emulator is one that can be moved from one machine to another and, more importantly, enables the host designer to work in parallel with the firmware designer. Thus, the virtual machine emulator and host machine hardware are constructed in concert, rather than in an ad hoc fashion.

These and other problems are solved in part by use of a high-level programming language specifically designed to write emulators. A proposed high-level language for implementing emulators is described by Malik (12). Malik's language is compiled into a portable intermediate form called IML (see Appendix A). The IML version of a virtual machine is then passed on to a translator-portability system for retrofitting to a specific host machine. <u>It is the translation of the IML described by Malik (13) that concerns this investigation</u>.

1-2 Significance of the Research

Most recent research in microprogramming is concerned with the quality of the code generation. Microprogram optimization refers to either reduction of the size of control store or reduction of the execution time of microprograms. Sizeable reductions in the execution time of microprograms may be obtained for horizontal microinstructions. This is due to the ability of horizontal microinstructions to combine more than one microoperation into a single microinstruction. All of the proposed algorithms detect parallelism of microoperations and then allocate microoperations to the smallest number of microinstructions possible. Two parallel microoperations are defined to be any two microoperations that can be executed without conflict. We discuss the kinds of conflicts that can arise in Chapter V.

Early work in code optimization is reviewed by Agrewala (1) with the conclusion that very few techniques exist that can be applied in a practical environment. A more recent overview in this area is given by Davidson (5), who found that there have been no published results showing the usefulness of any of these methods with large amounts of production microcode.

DeWitt (7) examined some compilers and algorithms proposed as "good" optimization algorithms (19, 21, 22) and found that these algorithms fail to produce the optimal sequence of microinstructions because they do not consider the interaction between register allocation and micro-operation concurrency. Furthermore, he found that micro-operation concurrency is sometimes determined by the format of the control word as well as by the host hardware. The importance of DeWitt's translating system is that the elevated code generation to the level of symbolic variables so that he could solve the combined problem of optimization and register allocation. In addition he opened the door to portability by supplying:

1) a model capable of describing a wide variety of microprogrammable machines, and
2) a register allocation/deallocation scheme integrated with code generation.

DeWitt's methodology is too general to run on a real machine, because his model does not define the host machine microcode, and the control flow interface problem is not taken into account.

<u>The major significance of this research, then, is</u>

to extend the results of DeWitt, add new techniques for solving the portability problem, and reveal the effectiveness of these methods when placed in use.

1-3 Thesis Introduction

The purpose of this thesis is to solve the problems associated with the translation of a machine independent intermediate language (IML) into an efficient microcode for a variety of microprogrammable machines. The IML defined by Malik (12) is directly compiled from a high level machine independent microprogramming language designed specifically for the realization of some virtual machine. The goals of the resulting system are:

A. Efficiency - The translator must produce the smallest number of horizontal microinstructions practical. This is accomplished by a compaction algorithm described in Chapter V.

B. Portability - An arbitrary machine can be used as the host. The system must be portable so that it is easy to retrofit it to any machine. This is accomplished by the Field Description Model discussed in Chapter II.

To realize these goals the following tasks must be done in this research:

1) Devise a model which describes all information needed by the system about the host machine.

2) Design a portable interface to map the machine independent IML into a machine dependent symbolic intermediate language (MDIL).

3) Implement register allocation/deallocation scheme to map symbolic variables in MDIL to machine unit names.

4) Develop a compaction algorithm to detect concurrency of statements which have been register allocated and to generate compact host binary microcode.

The next section provides an overview of the whole system by showing the implementation of a PDP8 virtual machine on a PDP11/40E host.

1-4 General Structure of the System

Based on the analysis of the goals and tasks proposed in the last section, the general structure of a machine independent translation system is described in Figure 1-2. The system requires three passes over the source code to produce compact host microcode. There are two inputs to the system. One is the machine independent intermediate language (IML) which is the realization of some virtual machine. The other is the description of the host machine. The output is the final version of a virtual machine ready to be loaded into a host control store as an optimized sequence of microinstructions which will execute some

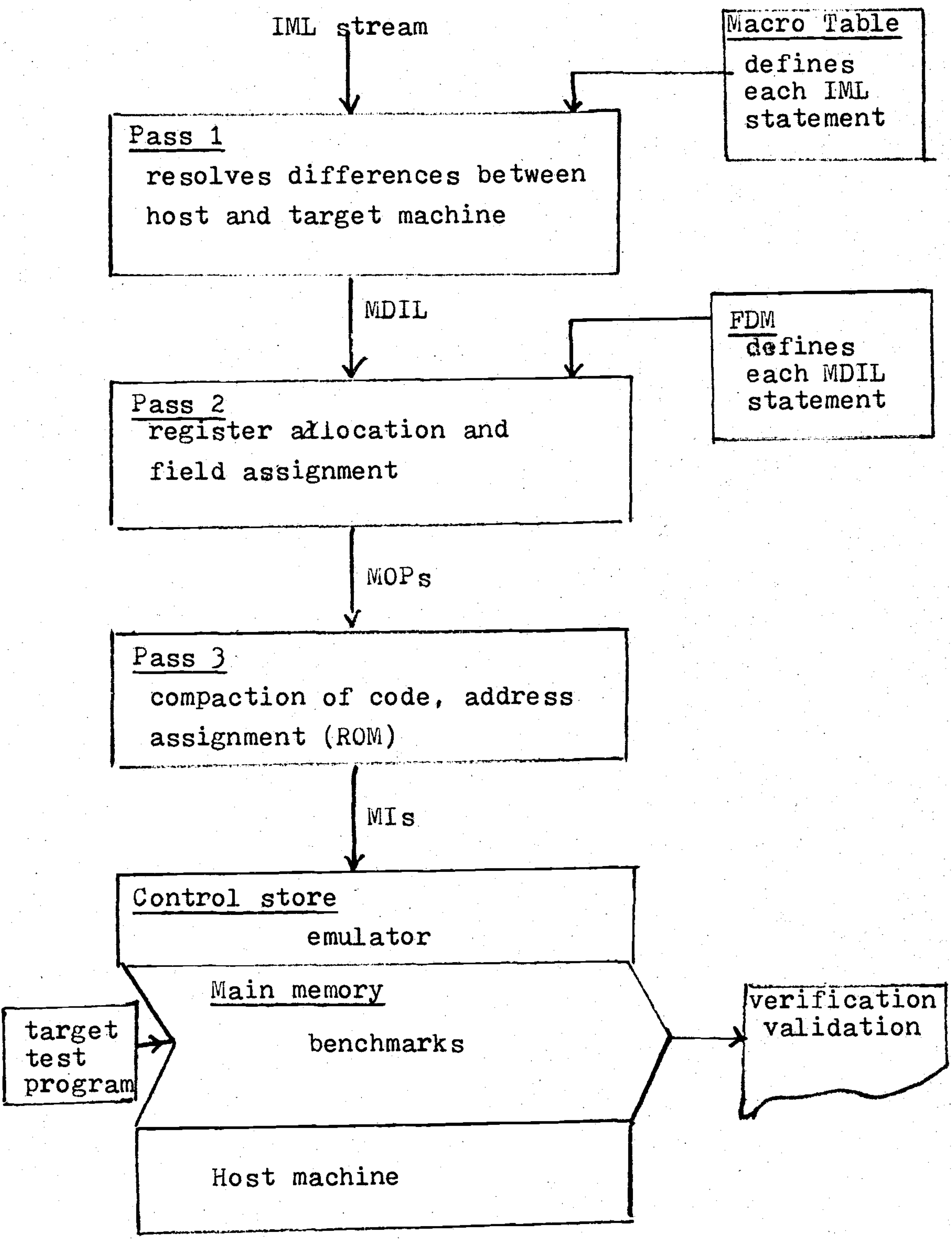

IML stream
Macro Table
defines each IML statement
Pass 1
resolves differences between host and target machine
MDIL
FDM
defines each MDIL statement
Pass 2
register allocation and field assignment
MOPs
Pass 3
compaction of code, address assignment (ROM)
MIs
Control store
emulator
Main memory
benchmarks
target test program
verification validation
Host machine

Figure 1-2. Structure of the Translation System

virtual machine program stored in the host's main memory. Suppose a PDP8 emulator written in Malik's high-level microprogramming language and translated into an IML stream is input to the system. The IML stream input to pass 1 is divided into two parts. One, called the intermediate executable statement group (denoted by IESG), contains a set of executable IML codes to describe the functional behavior of the target machine. This IML program is further divided into blocks. Each block is a single entry-multiple exit IML code. Variables defined in each block are either global (universal to the whole emulator program) or local (available only within the current block). The second part of the IML input, called the intermediate information statement group (denoted by IISG), describes the target machine hardware information and lists the variables used by each block.

For the PDP8 emulation, some typical parts of the IISG appear as shown in Figure 1-3. It provides partial hardware information of the target machine and lists one block of variables. This block is used to calculate the effective address of PDP8 target machine.

Note that in Figure 1-3, global variables are used to simulate the registers of the PDP8. For example, the PDP8 has a memory of 4096x12-bit words called MEM, a program counter called PC, and other registers, e.g., MAR, IR.

```
     IML (IISG Section)            Comments

00A  PDP8                          name of the emulator.

00D  ...12                         target machine has 12-bit words.

00E  TWO                           target machine is 2's complement.

221                                target machine memory is 4096x12
                                   bit words.
     .
     .
     .

00G  EFTADR                        block name for effective address
                                   computation.

207  MEM                           global variables used by the
                                   emulation to simulate the
207  IR                            registers of the PDP8 target
                                   machine.
207  PC

208  MAR

120  ADR,,7                        local variables with 7.12, and
                                   12 bit precision, respectively
120  PCTEMP,,12

120  MART,,12

     .
     .
     .
```

Figure 1-3. Partial IISG of PDP8 Emulator

The emulation also uses local variables such as the temporary program counter, PCTEMP, and temporary memory address register MART. These are used by the emulation to calculate the effective address prior to an operand fetch by the target PDP8 machine.

The executable IML codes of the "effective address block" are partially illustrated in Figure 1-4. These codes are given in quadruple notation.

The executable section of IML code is produced by the high-level language translator in a form to aid in optimization by pass 1, 2, and 3. For example, temporary variables are tagged (+, -) to indicate whether use will continue or not. This helps the register allocator.

The two-part IML stream is input to pass 1 as shown in Figure 1-2. The Macro Table (provided by the user) is consulted during pass 1 in order to expand each IML statement into a host-machine dependent macro. This process is illustrated for the PDP8 emulation by expanding the first four executable IML statements of Figure 1-4.

IML Macro Table	Comments
EXTR PGEADR IR +T.003	The first IML code of Figure 1-4. The following three codes are its macro expansion.
PUSH1 *1+IR TOS	Copy the IR into the top of the stack (TOS) of the PDP11/40E. Pass 1 tags IR as a global symbolic variable (denoted by sign "1") that will be used later (denoted by the sign "+").

```
    IML *IESG Section)                        Comments

00G EFTADR                                    Name of executable block
                                              for address calculation.

    EXTR   PGEADR  IR       +T.003            Get PGEADR from IR, put
                                              into temporary register
                                              designated as T.003.

    MOVE   -T.003  ADR                        Copy to ADR.  The "-" in-
                                              dicates that T.003 will
                                              no longer be used in this
                                              block.  (The "-" in the
                                              previous line indicates
                                              later use.)  These tags
                                              (+, -) are cues to be
                                              used by the register
                                              allocator.

    CONDF  .IR,7   TL.001                     Test bit 7 of IR, and
                                              branch to label L.001 if
                                              zero.  The label is
                                              designated "T" to indicate
                                              a True/False branch.

    SUB    PC      c1       PCTEMP            Decrement PC by constant
                                              1, and store it in
                                              PCTEMP.

    EXTR   CRNTPG  PCTEMP   +T.004            Extract CRNTPG (current
                                              page number) from PCTEMP
                                              and place into active
                                              temporary variable T.004.

    MOVE   -T.004  PCTEMP                     Copy from temporary var-
                                              iable T.004 (made inactive
                                              "-") into PCTEMP.

    OR     PCTEMP  ADR      MAR               Inclusive OR PCTEMP with
                                              ADR and store into MAR.
    .
    .
    .
```

Figure 1-4. Partial IESG of PDP8 Emulator

IML Macro Table				Comments
RSMK	TOS	PGEADR	D	Right-shift and mask the TOS word with PGEADR as a mask and store into host register D.
MOVE5	D		*2+T.003	Move host register D to temporary variable T.003. Pass 1 tags T.003 as a local symbolic variable (indicated by the "2") that will be used later.

The macro expanded version of EXTR still uses symbolic variables PGEADR, T.003, and IR. However, the macro also introduces PDP11/40E host machine registers. For example, the D register is the output from the ALU. The TOS register is actually a 16-word pushdown stack in the PDP11/40E host.

MOVE	-T.003	ADR	The second IML code of Figure 1-4.
MOVE3	*2-T.003	D	Copy from T.003 to host D. Pass 1 tags T.003 as a local variable that will not be used subsequently in this block (indicated by "-"). When the MOVE3 is done, the register allocated to T.003 may be reallocated to another variable.
MOVE5	D	*2+ADR	Copy from host D to symbolic ADR. Pass 1 tags ADR as a local variable that will be used later.

The macro above uses two different forms of MOVE because the PDP11/40E microoperation for MOVE commands when copying from D differ from those when copying to D.

CONDF	.IR,7	TL.001		The third IML code of Figure 1-4.
PUSH1	*1+IR	TOS		Save symbolic IR to TOS.
RMASK1	TOS	7	EUBC	Copy bit 7 of TOS word to host register EUBC, bit zero.
NOOP	XUPF	P.001		Copy symbolic address P.001 into base address register XUPF for purposes of branching, later.
BRCH	L.001	P.001	1	Branch depends upon the bit 7 of IR(0.1).

The CONDF code is performed by testing bit 7 of the symbolic variable IR. If a "1" is placed in the EUBC (a hardware register on the PDP11/40E host) the BRCH micro-operation fails to cause a branch to L.001. On the other hand, if a zero is placed in EUBC, the branch to P.001 is taken. In pass 3, the actual value of P.001 is determined along with L.001.

SUB	PC	c1	PCTEMP	The fourth IML code of Figure 1-4.
MOVE7	16		B	Copy constant 16 to register B. 16 is obtained by shifting a one by 4 bits due to a 12-bit target word on a 26-bit host. Hence, c1=16, is put into B.
SUB	*1-PC	B D		Subtract register B from PC and put into register D. Pass 2 tags PC as a global symbolic variable that will not be used subsequently in this block. When the subtract is done, the register allocated to PC may be reallocated to another variable.

SUB PC	c1	PCTEMP	The fourth IML code of Figure 1-4.
MOVE5	D	*2+PCTEMP	Copy register D to PCTEMP. Pass 1 tags PCTEMP as a local variable that will be used later, hence the "+" sign. This register may not be reallocated as permitted by the PC variable, in this block.

Register B is a host register for input to the ALU. Thus, host registers A and B are used for binary micro-operations on the PDP11/40E host.

The macro expansion above illustrates the use of tags placed in the IML stream by pass 1 as well as the macro expansion process.

Macro expansion of each block continues until the IML stream is exhausted. The result is a set of host machine dependent codes (MDIL) with partially symbolic variable.

Several problems remain before the output from pass 1 can be used on the PDP11/40E. First, we must allocate the symbolic variables to the general purpose registers of the actual host machine. Then, we can assign the binary microcode to each symbolic assembler code. Finally, we must resolve addresses (L.001). This additional step is done in pass 2.

In pass 2, the FDM (field description model) is used to define each MDIL instruction. This yields executable microoperations which will run on an actual host. FDM is actually a set of primitive operations used to describe the host machine control memory. Each primitive operation is defined by a 5-tuples in the form <OP, I, O, F, P>.

OP: operation code of this primitive operation.

I/O: host machine resources used as the inputs and outputs by this OP.

T: timing period of the machine needed to execute the <OP,I,O>.

F: a set of fields in the host machine micro-instruction format used to execute the <OP, I,O>.

For example, one of the primitive operations in the FDM of PDP11/40E is:

OP: SUB

I : One of the general purpose registers and register B of PDP11/40E.

O : register D of PDP11/40E.

T : pulse P2

F : Field RIF — Determined by register used by variable.

Field SRX=1 — Use RIF(0:3) as the address of register. This tells the host which register to use in the subtraction.

Field SBM=0 — Copy register B to B multiplexer in preparation for the subtract. This inputs B to the ALC.

Field SALU=6 The ALU is told to SUB.

Field DAD=8 The ALU is told to SUB.

Field CLK=2 The SUB is to occur during the second clock pulse of the microinstruction.

Field XUPF Determined by the next address.

Field CD=1 Copy result from ALU to register D.

The rest of the fields are not used.

This primitive operation can be used to define the MDIL code:

```
SUB    *1-PC   B   D    ; subtract register B from PC
                          and store in register D.
```

The FDM of each primitive operation is stored in a table and used by pass 2. Note that any host machine may be described by an appropriate FDM table. Hence, the portability of the system depends on the flexibility of this table.

The remaining chapters give generalized algorithms for producing compact, portable microprograms on a class of horizontal microprogrammable machines (pass 3). The PDP8/PDP11/40E example used throughout will illustrate that the techniques are quite general and apply to other high-level languages and host machines.

The results from pass 3 have been omitted from this introduction, but a complete PDP8 emulation is given in Chapter VI. For results of the compaction and register allocation algorithms see Chapter VI and Appendix E.

Chapter II develops the FDM (field description model) to describe general host machines. The purpose of this model is to describe an arbitrary horizontal microprogrammable host machine to the IML translator. Thus, portability is obtained if any other machine is used as the host, without altering the translation system. However, code efficiency is obtained only if the model can support sufficient host information to decode the IML and produce "compact" microcode. Microcode efficiency is the subject of Chapter V.

Chapter III solves problems that arise from the architectural differences between the virtual machine realized by the IML input stream and the host machine described by the FDM model. These problems include differences in the word size, memory size, arithmetic mode, hardware mismatch, and operation format mismatch. Portability and efficiency may be traded off in an attempt to solve these problems.

The purpose of Chapter IV (pass 2) is to assign binary microcode to each statement in the MDIL stream. Before this process can be completed all symbolic variables have to be allocated to the general purpose registers (GPR) of the host machine. In general, the number of variables in the program is greater than the number of registers of the host machine. In this case, one of the "less active" variables allocated to a register must be deallocated. "Load" and "store" operations are used to move operands between memory and the central processor's working registers.

The block structure of the MDIL stream from pass 2 is divided into a set of straight line code segments (SLC). The "state" of a GPR is defined for each SLC as the assignment of operands to the GPR. In loops, some extra load and store operations are needed to force the states of the GPRs equal to the initial state of the loop immediately before a backward branch operation. In this pass, an efficient register allocation/deallocation scheme and control flow interface scheme are developed to keep the number of "load" and "store" operations as small as possible.

After all symbolic variables have been allocated to the GPR registers, the microinstruction field value and timing phase are assigned to each statement. This produces a set of microoperations (MOP) in a 5-tuple representation. <OP, I, O, F, P>, for each SLC in each block of MDIL.

The 5-tuples obtained from pass 2 may be exchangeable with one another due to their independence. This fact is used to detect whether a particular MOP can move toward the beginning of the SLC. Whenever a 5-tuple is moved forward in the SLC possible concurrency is checked. Chapter V (pass 3) examines the 5-tuples of each SLC to detect and combine concurrent 5-tuples into fewer microinstructions. Thus, a compaction algorithm is developed to allocate the sequences of microoperations into compact concurrent microinstruction.

The optimization of microoperations produced from a portable high level language is known to be an NP-complete problem (7). Invertibility (defined as the situation where two MOPs are data independent with each other) is the cause of the NP-complete optimization problem, but data dependency among MOPs limits their invertibility. After some restrictions are put on the allocation of MOPs, as O(mn) algorithm is developed which may not produce optimum code, but produces the "best" possible code when it applies to the real machine.

In Chapter VI we explore the quality of the linear time compaction algorithm and show that it is close to the best that can be done with real machines.

CHAPTER II

THE FIELD DESCRIPTION MODEL

2-1 Introduction

The purpose of this chapter is to develop a model used to describe arbitrary microprogrammable host machines in order to get both portability and efficiency from the translation system when machine independent IML is translated to a host machine microcode. By portability we mean that when other host machine is used, only this model is changed. Effective translation can take place if the model supplies all information about the host machine which will be needed to translate the virtual machine into microcode for a subsequent host machine. The following goals are set up for designing this model:

1) The format of this model is machine independent so that it easily fits other machines.
2) The model is comprehensive in that it includes all host machine information needed in the system and it can describe the IML well.
3) This model provides an easy way to detect the conflicts between any two operations.

Section 2-2 surveys earlier research done in this area. Section 2-3 gives a brief analysis of a microprogrammable machine used as an example host. Section 2-4 describes how the Field Description Model is developed to suit the system. The use of this model is illustrated in section 2-5 and 2-6.

2-2 Previous research review

Two different models proposed by Dasgupta (3) and DeWitt (6), respectively, have previously been used to describe an arbitrary host machine and its corresponding concurrency of microoperations.

2-2-1 Dasgupta Model

In Dasgupta's model (3), the host machine is described in terms of a sequency of microoperations. Each microoperation is denoted by the 5-tuple.

m= ⟨OP, SC, SK, U, V⟩

where

"OP" designates a primitive operation,

"SC," "SK" denote the data source and sink sets respectively for "OP,"

"U" denotes the set of operational units and/or paths required to execute m,

"V" is a timing period in which m is executed.

One criterion used to detect the concurrency of microoperations is: If there is no source/sink conflict and no operational unit conflict between two operations, they can be combined into one microinstruction.

This model is hardware oriented. All necessary machine units associated with the microoperation are given in the 5-tuple. The model is inadequate as a portable translator model for the following reasons:

1) Because of architectural complexity of the host machines, it is not easy to display all physical operational units which are used to execute the operation.
2) Detection of the operational unit conflicts is another complexity, if the model cannot display all hardware units.
3) Some counter examples given by DeWitt show that even if there is no hardware unit conflict between two operations, they still cannot be executed in one microinstruction.

2-2-2 DeWitt Model

DeWitt (6) found that the concurrency permitted by microoperations is sometimes determined not simply by the hardware configurations but also by the format of the control word chosen by the designer. This observation motivated the control word model for determining parallel operations. This model describes a host machine, a set of blocks B, and a set of configurations C. Each block (which corresponds to the first three tuples of the Dasgupta Model) describes a set of microoperations or a field in the microinstruction. Each configuration describes a legal combination of microoperations. The set C contains a description of all the legal microinstructions for the machine. Thus, in order to determine whether two or more microoperations can be executed concurrently, the corresponding block for each operation is identified first and the set C is examined to determine if a configuration C_j exists in such a way that each block is an element of C_j. In conclusion, this model utilizes a logical approach for describing the concurrency available in host machines rather than a physical approach as in the Dasgupta Model. The factor determining success of the Control Word Model is whether this model can successfully describe all the legal microinstructions a machine can execute.

This model provides a correct method to determine the concurrency of microoperations, but there are still some problems it does not solve. Among these problems are:

1) In using this model, one has to determine the independent block first, then check for concurrency of blocks in order to get a "legal" configuration. DeWitt does not give a method for finding concurrency of the blocks. This might be a heavy burden for a user who is not familiar with the host machine.

2) This model does not supply the binary microcode of each microoperation.

3) The model in the DeWitt system is not used to map the machine independent code to machine

dependent code.

These two models fail to satisfy the needs of our translation system, but lead to a modified model called the Field Description Model described in the next section.

2-3 General Description of the Host Machine

To summarize all host information into a fixed format model to suit the translation system is challenging work because of the substantial architectural differences in a variety of microprogrammable machines. In this section an example host machine is briefly analyzed and critical features extracted and used in the model.

2-3-1 Hardware Description

In order to describe the IISG of the IML, the following hardware information of the host machine must be known:

1) Word size and memory size.
2) Arithmetic mode.
3) Status registers used to display flag settings, e.g., carry, overflow.
4) Storage devices
   a. Primary memory used to store virtual machine executable programs,
   b. Control memory used to store the final version of virtual machine,
   c. General purpose registers (GPRs) used to hold the variables declared in the IISG,

d. Working registers used to perform ALU operations (in most machines, working register and the GPR are the same), and

e. Any other machine units.

5) Hardware configuration and stack. The IML will supply information about a stack, if it exists in the virtual machine.

6) The method used to determine the next micro-address.

▶▶ Example 2-1:

The example host machine is the PDP11/40E, and the following hardware information is extracted:

Items	Information
word size	16 bits
arithmetic mode	2's complement
flags setting	carry, overflow, negative and zeros
storage devices	
main memory	
control memory	1024 words RAM, 256 words ROM and 32 words PROM
general purpose register (GPR)	16 words
working register	GPR is used as the working register
other machine units names	registers EUBC, UPF, EUPF, TOS, BA, B, D, etc.

Items	Information
hardware configuration	16 words stack, processor status register (used to set flags), shifter, masker, etc.

2-3-2 Software Description

From the functional behavior viewpoint, a microprogrammable machine is simply a machine consisting of a set of primitive operations encoded and stored in a control memory. When one of these operations is executed, a set of hardware units is activated to process the data during a certain timing period with reference to the machine cycle. This set of primitive operations is used to emulate the statement in IESG of IML.

The efficient emulation of IML involves the following questions:

1) How are primitive operations chosen to describe the IML?

2) How is hardware unit information used in the corresponding operation supplied?

3) How is the binary microcode associated with the primitive operation?

2-4 Field Description Model

Each host machine has a unique microinstruction format which consists of a set of fields. Each microoperation has fixed fields in the MI format where binary microcodes are assigned. The set of fields of each microoperation can be considered as the logical operational unit and used as the residence in the execution of this microoperation. If the physical operational unit in Dasgupta's model is replaced by this logical operational unit, the shortcomings given in the last section to explain why the illustrated models fail to satisfy the needs of our system can be alleviated. This modified model can get the following advantages immediately:

1) All the necessary fields used to execute the microoperation are easily illustrated in the microinstruction format.

2) The binary microcode is obtained directly from the value of each field.

Further, in Chapter V, we successfully develop a rule to detect the concurrency of microoperations given this modified model. These enhancements motivated the development of the Field Description Model that meets the objectives proposed in the first section.

2-4-1 Definition of the FDM

The Field Description Model (FDM) represents the host machine as a set of microoperations (MOPs).

$$\text{FDM}=\{M_i,\ 1 \leq i \leq n\}$$

Each MOP, M_i, which is identified by a unique index i is denoted by a set of five tuples,

$$M_i = \{\text{OP, I, O, F, P}\}$$

and each tuple is expanded by specifying its domain. Each domain enumerates all the legal values which the component can assume. The tuple components are:

OP: Designates the primitive operation to be performed.

I : Denotes the resources used as the input to the OP.

O : Denotes the resource used as the output to the OP.

F : Denotes the set of fields which are occupied in the microinstruction format when OP,I,O is executing.

P : Denotes the set of timing phases at which the ⟨OP,I,O⟩ is executing.

The following example will illustrate this idea.

▶▶ Example 2-2:

One of the MOPs in the FDM of the PDP11/40E (7,8) is described by:

$$M_i = \langle ADD,\ I,\ O,\ F,\ P\rangle$$

where

1) The domain of I is register B and the set of the general purpose registers.

2) The domain of output is register D.

3) The domain of timing is pulse 2.

4) The domain of field is as follows: (The meaning of each field is described in Appendix B)

Field 1 specifies one register from the set GPR.

Field 13 specifies the next address.

Field 6=9 (specifies the operation ADD).

Field 2=1 (allows Field 1 to be used as a source of general register address).

Field 5=0 (B register→B mux).

Field 12=2 (this MOP is activated in pulse 2).

Field 19=1 (allows clocking the ALU into D register).

The remaining fields are not used in this MOP.

5) The domain of OP is operation ADD. ◄◄

A complete FDM of the PDP11/40E is described in Appendix B. There are 41 MOPs in the model which are used to describe the host machine and decode most statements in IESG of IML. For each MOP, there are some items in tuples I, O, and F which cannot be determined when the model is built. For instance, in example 2-2, one GPR is to be used as the input, so field 1 is undetermined. The selection of the register used as the input is determined from the register allocation/deallocation scheme in Chapter IV. The determination of field 1 and field 13 are shown in Chapter IV and Chapter V, respectively.

2-4-2 General Rule to Build the FDM

The general rule to determine the FDM is described in Figure 2-1 and Algorithm 2-1 which implies the following steps in the selection of the five tuples.

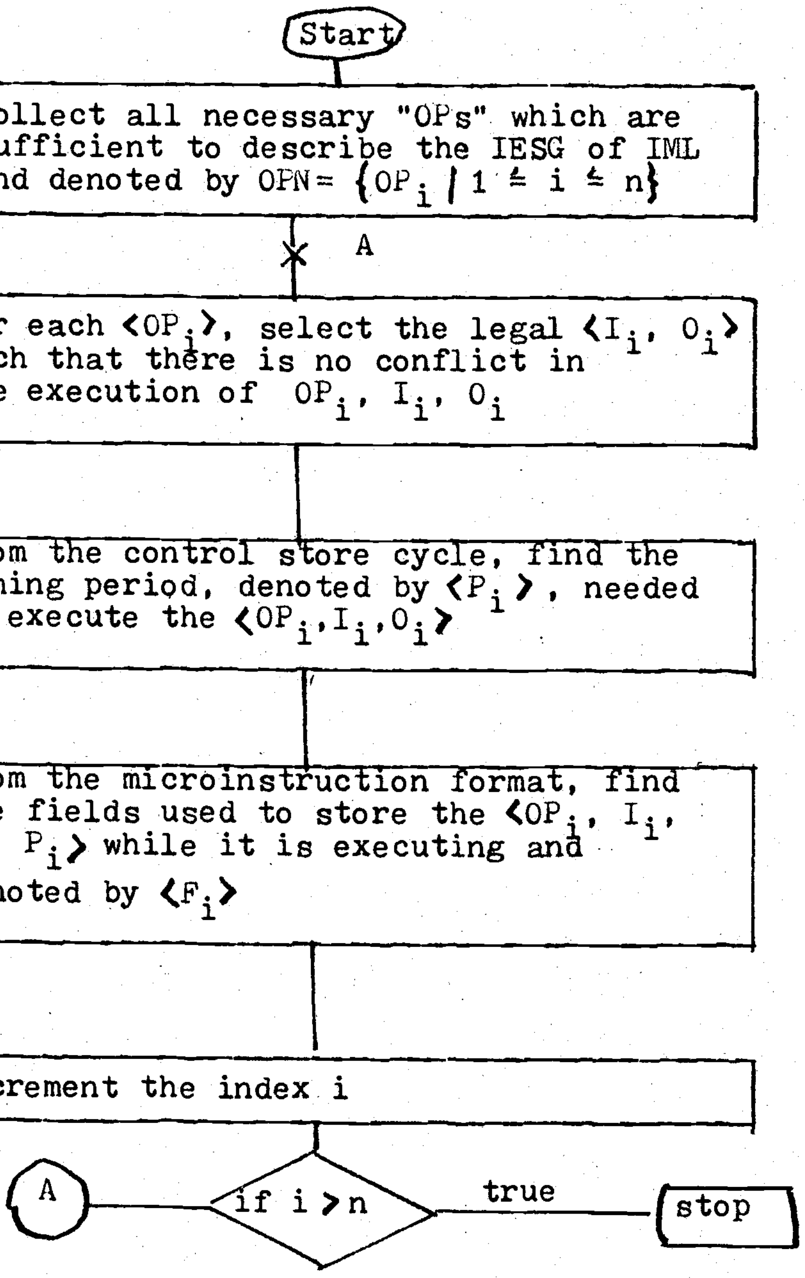

Start
collect all necessary "OPs" which are sufficient to describe the IESG of IML and denoted by OPN= $\{OP_i \mid 1 \leq i \leq n\}$
A
for each $\langle OP_i \rangle$, select the legal $\langle I_i, O_i \rangle$ such that there is no conflict in the execution of OP_i, I_i, O_i
from the control store cycle, find the timing period, denoted by $\langle P_i \rangle$, needed to execute the $\langle OP_i, I_i, O_i \rangle$
from the microinstruction format, find the fields used to store the $\langle OP_i$, I_i, O_i, $P_i \rangle$ while it is executing and denoted by $\langle F_i \rangle$
increment the index i
A
if $i > n$
true
stop

Figure 2-1. Functional Flow Chart of the Generation of the FDM

Algorithm 2-1. General Rule to Determine the FDM

Comment: The Field Description Model (FDM) is built by the user to supply the host machine primitive operations.

```
BEGIN
CALL ALGORITHM 2-2 TO OBTAIN ALL NECESSARY "OPs" WHICH ARE
SUFFICIENT TO DESCRIBE THE IESG OF IML
SELECT THE LEGAL <I,O> ASSOCIATED WITH EACH "OP" SUCH THAT
THERE IS NO CONFLICT IN THE EXECUTION OF <OP,I,O>
IF THE RESOURCES USED AS <I,O> ARE THE MACHINE UNIT NAMES
   THEN ASSIGN THE MACHINE UNIT NAMES TO <I,O> DIRECTLY
   ELSE (These resources used as the <I,O> cannot be
        determined now)
        ASSIGN THE CORRESPONDING MNEMONIC VARIABLE TO <I,O>
        (This variable will be determined in Chapter IV)
CALL ALGORITHM 2-3 TO DIVIDE LOGICALLY THE CONTROL STORE
CYCLE INTO A SET OF PHASES AND EACH <OP,I,O> IS ASSIGNED TO
THE CORRESPONDING PHASES(S)
FROM THE MICROINSTRUCTION FORMAT, FIND THE FIELDS USED TO
EXECUTE THE <OP,I,O> AND DETERMINED THE VALUE OF EACH FIELD
IF THE FIELD VALUE CAN BE DETERMINED FROM <OP,I,O>
   THEN FIELD VALUE IS ASSIGNED TO THE CORRESPONDING
        NUMERICAL VALUE
   ELSE FIELD VALUE IS ASSIGNED TO AN ALPHABETIC VALUE
        AND WILL BE DETERMINED IN PASS 2
BASED ON THE MACHINE CONSTRAINT, GET A RULE TO DETECT THE
CONCURRENCY OF MOPs (This idea is illustrated in Chapter V)
END.
```

OP,I,O Selection

The "OP" selection directly influences the efficiency of the FDM. From the objective viewpoint, the basic function of the model is to map the IML into machine dependent code. This mapping is one-to-one for simple IML operations, and, one-to-many for complex IML operations. The set of operations in the FDM must be able to express simple operations in the IML. The general rules for choosing the "OP" used in the FDM are described in Algorithm 2-2. The I/O resources must be selected so that there are no conflicts in the execution of ⟨OP,I,O⟩. The following example will illustrate this idea.

▶▲Example 2-3:

In the PDP11/40E (7,8), addition is one of the ALU operations. The input resources to the arithmetic logic unit are BIN and AIN, respectively. The choice of an output register must consider possible I/o conflicts if register B and one register from the set of general purpose registers (GPR) are used as inputs.

If one register from the set of GPRs is used as the output resource, then conflict may occur within this MOP. For example, the statement

R2 + B ⟶ R3 ; add R2 and register B to R3

is not allowed by the PDP11/40E host in one microinstruction due to the conflict between R2 and R3. In this case, Algorithm 2-2. Selection of Tuple "OP" in FDM

```
BEGIN
COMPARE THE OPERATIONS (OPs) IN THE MI FORMAT OF THE HM
WITH THE STMT IN IESG OF IML
CASE "OP" OF:
       IN IESG AND IN HM:  THIS "OP" IS USED IN THE FDM
       IN HM BUT NOT IN IESG:  THIS "OP"IS NOT USED IN THE
       FDM
       IN IESG BUT IN HM:  BEGIN
                          *IF THIS "OP" IS NOT DECODED BY
                           PASS 1 (i.e. This "OP" is used
                           as the simple IML code)
                             THEN DECODE THIS "OP" INTO A
                                  SET OF MACHINE OPERATIONS
                                  AND PUT THEM IN THE FDM
                           END
END.
```

*Some complex IML stmts are decoded by the translation system. The detail is in Chapter III.

Algorithm 2-3. Selection of Tuple "F" in FDM

```
BEGIN
IF THE CONTROL CYCLE IS PHYSICALLY DIVIDED INTO SEVERAL
PHASES AND ASSIGNED TO EACH MICROOPERATION
   THEN THE LOGICAL PHASE=THE PHYSICAL PHASE
   ELSE BASED ON THE SEQUENCE OF THE MICROOPERATIONS APPEAR
      IN THE MICROINSTRUCTION, THE CONTROL STORE CYCLE IS
      LOGICALLY DIVIDED INTO A SET OF PHASES AND EACH
      PRIMITIVE OPERATION IS ASSIGNED TO THE CORRESPONDING
      PHASE
END,
```

a set of GPRs cannot be used as the output resource. Instead, register D is used as the output resource to make sure this MOP is executable and causes no conflict in <OP, I, O>.◀◀

Each <OP,I,O7 is a primitive operation and from the characteristics of horizontal microprogrammable machines, more than one of these primitive operations may be executed in the same microinstruction. In order to construct this kind of microinstruction, we must consider "residence conflicts" and possible "timing" conflicts.

Timing Tuple Assignment

The execution of a microinstruction is controlled by the fixed control store cycle. Within this cycle, most machines provide multiple phases (polyphases) of timing periods for each microinstruction. In this research the control cycle is logically broken into several distinct phases and control signals are issued at each phase. According to the sequence of the <OP,I,O> appearing in the microinstruction, each primitive operation is assigned to one or more logical phases. The general rule is described in Algorithm 2-3. The following example will illustrate this idea.

▶▶Example 2-4:

In the Mathilda machine (18), the microinstruction is implemented in a polyphase manner. The logical phases of microinstruction execution are the following:

1) Performing data transport on the main data path.

2) Executing shift and other operations.

3) Calculating the address of the next microinstruction to be executed.

Another example is the Microdata 3200 machine (16), where each 135 nano-second clock is needed to get and execute a single 32-bit microinstruction from control store. This control cycle is logically divided into three phases, which are:

P1: Test evaluation condition.

P2: Action of the current instruction.

P3: Branch, on the basis of the test value from P1.

Thus, all microoperations of these machines can be logically assigned to three phases. ◄►

Field Tuple Selection

The choice of the set of fields associated with the ⟨OP,I,O⟩ is obtained directly from the microinstruction format of the host machine. The value of each field is classified as one of two kinds. One is the commercial value already defined. The other is the alphabetical value determined later. The following example explains this idea.

►►Example 2-5:

In the PDP11/40E (7,8), the eighty bit microinstruction format is divided into 27 fields. The field tuple associated with each <OP,I,O> uses these 27 fields directly. In reference example 2-1, seven fields are used in the field tuple of this MOP and classified into two kinds:

1) The field value has already been defined.

   Field 2 is set to use the GPR as the input resource.

   Field 5 is set to use the register B.

   Field 6 is set to use the OP ADD.

   Field 12 is set to use clock 2.

   Field 19 is set to clock register D.

2) The field has not yet been defined.

   Field 1 is a function of GPR selection.

   Field 13 is determined by the next microaddress.◄◄

2-5 Discussion of the FDM

The FDM is a modified Dasgupta model in which the logical operational unit is a set of fields replacing the physical operational unit referenced by the microinstruction format. The FDM overcomes the disadvantages listed in section 2-2, and includes other important features as follows:

1) The field tuple implicitly limits the number of MOPs in the MI. The fields associated with

each MOP, and the number of MOPs in one MI are inherently constrained by the host machine. When the number of MOPs in a MI is equal to the length of the MI, all fields in MI are occupied, making it impossible to add another MOP to this MI. This feature is used to advantage in the code compaction algorithm of Chapter V.

2) Because of the architectural complexity of host machines, it is hard to display all physical operational units for each MOP. This adds difficulty to the detection of physical unit conflicts. But in the FDM, all physical units used in one MOP can be expressed in terms of the logical operational unit. Then, all physical operational unit conflicts between MOPs can be detected from their logical operational unit.

▶▶Example 2-6:

In the PDP11/40E (7,8), the RD bus has three potential resources: 1) GPR, 2) the processor status word, and 3) the extension. Each of the three can independently gate a word onto the RD bus. Usually two resources gated onto the RD bus would result in an error. When the following operations are involved,

M1: 400⟶D, P2 ; copy constant 400 to register D

; in pulse P2

M2: D ⟶ R6, P3 ; copy register D to R6 in pulse P3.

R6 is set to a constant value, 400. With reference to Figure 2-2, the physical operational units used in M1 are the RD bus and AIN. In M2, it appears as if only the Bus is used as the physical unit. If this assumption were true, then M1 and M2 would be executed in one MI during clock cycle 3, the I/O conflict being avoided by the timing pulse. But, in fact the new value of R6 is its old value with bit 8 set instead of 400. Why? This idiosyncrasy is handled by the FDM by switching M1 and M2 to the following:

M1 <MOVE6, 400, D, F1, P2> ; same actions as the

M2 <MOVE5, D, R6, F2, P3 > ; previous statements

Domain of F1 is:

Field 6=0
Field 14=1
Field 14=15
Field 16=25
Field 17=0
Field 18=400
Field 19=1

Domain of F2 is:

Field I=6
Field 2=1
Field 4=2
Field 11=3

(The set of fields used in M1 or M2 is defined from Appendix B.)

The logical operational unit of M2 is examined.

From the host machine manual as field 1 and field 2 are set and the corresponding register is being clocked, the RD bus is activated again. This implies that the RD bus is used as the physical unit in M2. Hence there is a physical operational unit conflict so that the potential concurrency cannot be permitted. As is seen, the fields used in one MOP can express hardware characteristics of a host computer.

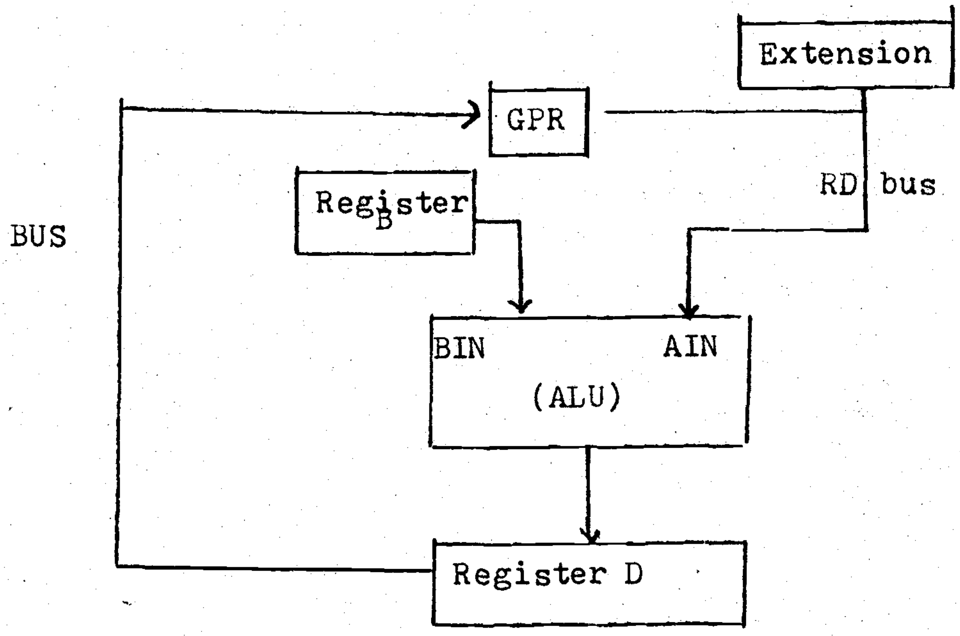


Figure 2-2. Simple Diagram of PDP11/40E CPU

Furthermore, the physical operational unit conflict can be detected from the field tuples. In F2, as field 1 and field 2 are set it implies that the RD bus is activated. In F1, as field 14 is set it implies that the emit value is sent to the RD bus. In the detection of RD bus conflicts, we only check these three fields by the following rule:

```
IF (f(1,14)=1) and (f(2,1) are set)
   THEN there is an RD bus conflict between M1
        and M2
   ELSE no conflict
```

where f(i,j) means field j in MOP_i.▼▼

3) Some field can be shared by more than one MOP in one microinstruction (MI) and will not cause a conflict. This feature can be used to detect whether two MOPs can be executed in one MI even if there is a physical operational unit conflict in the same timing phase. To understand this point, the fields in the MI format are grouped into two categories first, then an example is given to illuminate this feature.

There are two kinds of fields in the MI format denoted by F_A and F_B, respectively.

$F_A = \{f_i \mid$ if $f_i\}$ is used by more than one MOP in the same MI and the values assigned to these fields are the same, it will cause no conflict.

For example, the literal field can be used by more than one MOP in the same MI only if the value assigned to this field is the same. Obviously, if this kind of field is used by more than one MOP and the field value is not the same, it causes a conflict.

$F_B = \{f_i \mid$ if f_i is used by more than one MOP in the same MI, it will cause a conflict even if the field value is the same.$\}$

For example, when the machine has only one ALU operational unit, if two MOPs try to execute the same ALU operation, this field will cause a conflict in detection of parallelism.

▸▸Example 2-7:

```
Case 1:  M1:  R2 + B   D, P2 ; add R2 and register B to
                             ; register D in pulse P2
         M2:  D  R3, P3      ; copy register D to R3 in
                             ; pulse P3
```

Refer to Figure 2-2, the physical operational units used in M1 are the RD bus and the ALU. Based on example 2-6, the RD bus and BUS are used in M2. Because of the RD bus conflict, the potential concurrency cannot be permitted.

```
Case 2:  M3:  R2 + B   D, P2 ; add R2 and register B to
                             ; register D in pulse P2
         M4:  D  R2, P3      ; copy register D into R2
                             ; in pulse P3
```

For the same reason as in case 1, the conflict of RD bus still exists between M3 and M4. But, the execution of M3 and M4 in one MI is permitted by the machine. This permission can be obtained by examining the field tuples. The field tuples used in each MOP are:

F1 and F3	F2	F4
f(1,1)=f(3,1)=2	f(2,1)=3	f(4,1)=2
f(1,1)=f(3,2)=1	f(2,2)=1	f(4,2)=1
f(1,5)=f(3,5)=0	f(2,4)=2	f(4,4)=2
f(1,6)=f(3,6)=9	f(2,11)=3	f(4,11)=3
f(1,19)=f(3,29)=1		

(The set of fields used in each MOP is defined from Appendix B.)

Further, field 1 and field 2 are classified as the elements in set F_A. As is seen in case 2, f(3,1)=f(4,1), f(3,2)=f(4,2); i.e., there is no logical operational unit conflict, so that concurrency is allowed even if the physical unit conflict exists. (The physical unit conflict on RD bus still gets the correct result which is from R2 ORed R2). An examination of case 1 shows that f(1,1)=2, f(2,1)=3, so this logical operational unit conflict does not permit the concurrency of M1 and M2. (The conflict on RD bus gives a wrong result which is from R2 ORed R3). ◄◄

4) The logical operational unit supplies the binary microcode for each MOP so that the FDM tuple can be used in the real machine instead of the abstract machine.

From the above discussion, it is obvious that the logical operational unit of the FDM has much potential to detect the concurrency of MOPs. Based on the 5-tuple format, a code compaction algorithm which is developed in Chapter V can save up to 20% instruction count when it is applied to the real machine.

2-6 Conclusion

Refer to Figure 2-3, a simple illustration of the system, to show the use of the FDM. The FDM developed in this chapter provides the following facilities for this system:

1) In pass 1 (Chapter III), Tuple OP supplies the basic host machine operations used to decode the statements in IESG of IML.
2) In pass 2 (Chapter IV), Tuple F provides the field value for each primitive operation and Tuple P assigns the timing phase to this operation. The output is mapped to a set of MOPs in 5-tuple representation.
3) In pass 3 (Chapter V), the 5-tuple format provides a very efficient way to perform the optimization of MOPs.

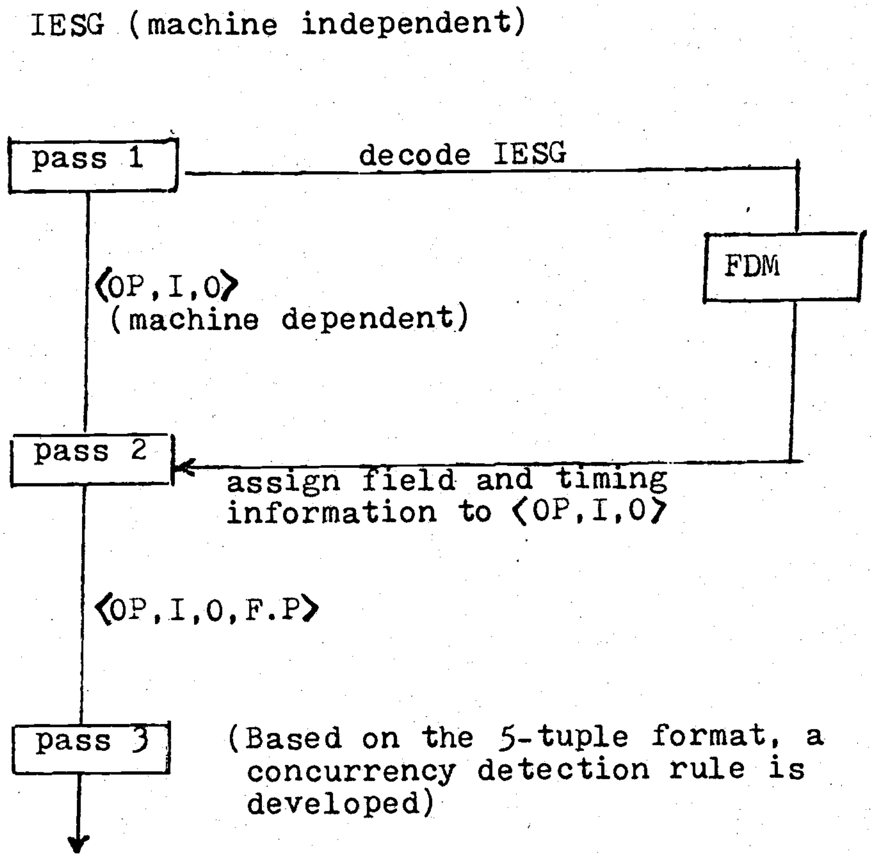

IESG (machine independent)
pass 1
decode IESG
FDM
⟨OP,I,O⟩
(machine dependent)
pass 2
assign field and timing information to ⟨OP,I,O⟩
⟨OP,I,O,F.P⟩
pass 3
(Based on the 5-tuple format, a concurrency detection rule is developed)

Figure 2-3. Use of the FDM

CHAPTER III

PASS 1

3-1 Introduction

The purpose of this chapter is to present a solution to the interface problems associated with the mapping of a machine independent intermediate language (IML) to a host machine dependent intermediate code (MDIL). The IML is directly compiled from a high level machine independent microprogramming language developed for the realization of some virtual machine. /The host machine information is described in the Field Description Model (FDM) which was developed in the previous chapter.

A machine independent interface system is needed for portability, but, because of the architectural differences between the virtual machine and the host machine, such a portable interface system can hardly take advantage of the host machine to produce efficient object code. In order to squeeze both of these goals into the system, the problems arising from the mapping are solved by the system designer and the user.

Section 3-2 discusses problems arising from attempting to handle different host machines. Section 3-3 makes a suitable assignment of responsibility for solving these problems to the user and the designer. Section 3-4 then shows how these problems are solved.

3-2 Problems Arising from the Differences Between Machines

From a low level designer's viewpoint, all characteristics of the virtual machine are described in the IML. (The detail description of the IML is illustrated in Appendix A.) The information statement group, IISG, of IML describes the virtual machine hardware characteristics. The executable statement group, IESG, of IML, which is used to describe the virtual machine functional behavior, consists of a set of blocks. Each block is a single entry-multiple exit collection of host machine independent codes. Variables defined in each block are either global (universal to the whole emulator program) or local (available only within the current block).

Virtual machine and host machine differences stem from:

1) The word size and the memory size. These differences influence machine performance.
2) Arithmetic mode used
   The negative number representation and the subtraction operation may cause incompatibility between virtual machine and host machine.
3) Hardware configurations
   If some hardware unit exists in the target machine but not in host machine, an extra

mapping is needed.

▶▶Example 3-1:

In the IML, if the statement

223,,,S1,S2,S3,S4

is given in the IISG, the tag indicates that a stack pointer exists in the target machine. The other information, S1, S2, S3, S4 indicates the push-pop sequence associated with the stack. If the host machine does not support a hardware stack, the code generation procedure must provide a software routine to implement an algorithm to simulate the stack operation.◀◀

4) Operation format

The host machine operations defined by the FDM are called the basic machine codes.

IML operations in IESG are divided into two kinds. The operations in one group are called the simple IML codes. The group of complex IML codes is the IML codes which cannot directly map into the basic machine codes.

▶▶Example 3-2:

a) ADD *GPR B D

This is a basic machine code from the FDM which means to add GPR and register B to D.

b) ADD SRC1 SRC2 Dest

This is a simple IML code which means to add SRC1 and SRC2 to Dest. There may be different ways to implement this in different host machines.

```
c) LOOP  SRC1  SRC2  SRC3  ; loop for SRC1=
                             SRC2 to
                           ; SRC3 by 1.
```

Since most machines do not provide the corresponding primitive operation to decode the "LOOP" directly. This complex IML code needs additional modification described in subroutine EXPANS (section 3-4) before it can be mapped into a machine code. ◀ ▼

The translation system must:

1) handle the problem of word size differences and/or different arithmetic modes,

2) simulate the hardware units existing in the target machine but not in the host machine,

3) decode the complex IML code,

4) implement a mapping from the simple IML code to basic machine code.

3-3 Information Supplied by the User

Before going into more detail, two objectives proposed in the previous chapter are to be traded-off here. One is to get an efficient object code. The other is to get a portable translation system. If all the tasks arising from the differences between machines are implemented by the translation system designer, the translation process can be made machine independent, but it can hardly take advantage of the host machine. The result may be production of inefficient microcode. On the contrary, if all these tasks are implemented in the host machine microcode by the user, we can easily take advantage of the machine to get efficient object code. But this is a tedious and error-prove implementation methodology rejected at the outset because portability is lost.

A Macro Expansion Table (MET) written in the basic host machine code is built by the user to simulate simple IML code. The target machine hardware units which do not exist in the host must be simulated, also. The remaining tasks, including the decode of the complex IML code and the simulation of problems from the word size and arithmetic mode differences, are done by the system designer in pass 1.

▶▶Example 3-3:

```
↑ADD    SRC1    SRC2    Dest
```

This is a simple IML code to perform addition of SRC1 and

SRC2 to Dest and set the host machine flags, carry (C), overflow (O), negative (N), and zero (Z). The corresponding MET to do this IML code on a PDP11/40E is as follows:

```
        MOVE1   SRC1        B      ; move SRC1 to register B
        ADD     SRC2   B    D      ; add SRC2 and register B to
                                   ; register D
        MOVE 5 D            Dest ; move register D to Dest
        Flag                       ; set host flags C,O,N, and Z
```

where register B and D are the PDP11/40E units. All four codes and their corresponding format are defined in the FDM (see Appendix B). SRC1, SRC2, and Dest are still symbolic variables and are allocated into registers in Chapter IV. Another example is:

```
        MOVE   .PS,0   varc
```

Where PS is a status register of the host machine which is used to display flags carry, overflow, negative and zero from the associated bits in PS. ".PS,0" means the bit 0 of register PS. This simple IML code moves the bit 0 in PS to varc. The corresponding MET is:

```
        PUSH     PS                TOS
        RSMK     TOS   0, 15, 0    D
        MOVE5    D                 varc
```

Where "0,15,0" is the constant to be shifted and/or masked. The content in the top of stack, TOS, is masked out the left fifteen bits (field LML=0, field RML=15) and shifted zero bit (field SC=0).

A complete example of MET of PDP11/40E is illustrated in Appendix C.

When the user decides which machine is to be the host machine to the system, the following tasks must be accomplished.

1) Build a FDM as described in Chapter II.
2) Build the MET for the corresponding simple IML code.
3) Simulate all hardware units which exist only in the target machine.

The remaining tasks will be done by the system in pass 1.

3-4 Pass 1

With the aid of user supplied host machine information, pass 1 maps the machine independent IML into a machine dependent intermediate language (MDIL). The functional flow chart and the general structure of this pass are shown in Figure 3-1 and Algorithm 3-1, respectively. Refer to Figure 3-1, the following paragraph is to illustrate the detail function of each subroutine.

*** Subroutine IISG ***

This subroutine is used to collect the virtual machine hardware information, and assign a main memory location of the host machine to each variable declared as either global or local variables in IML. The virtual

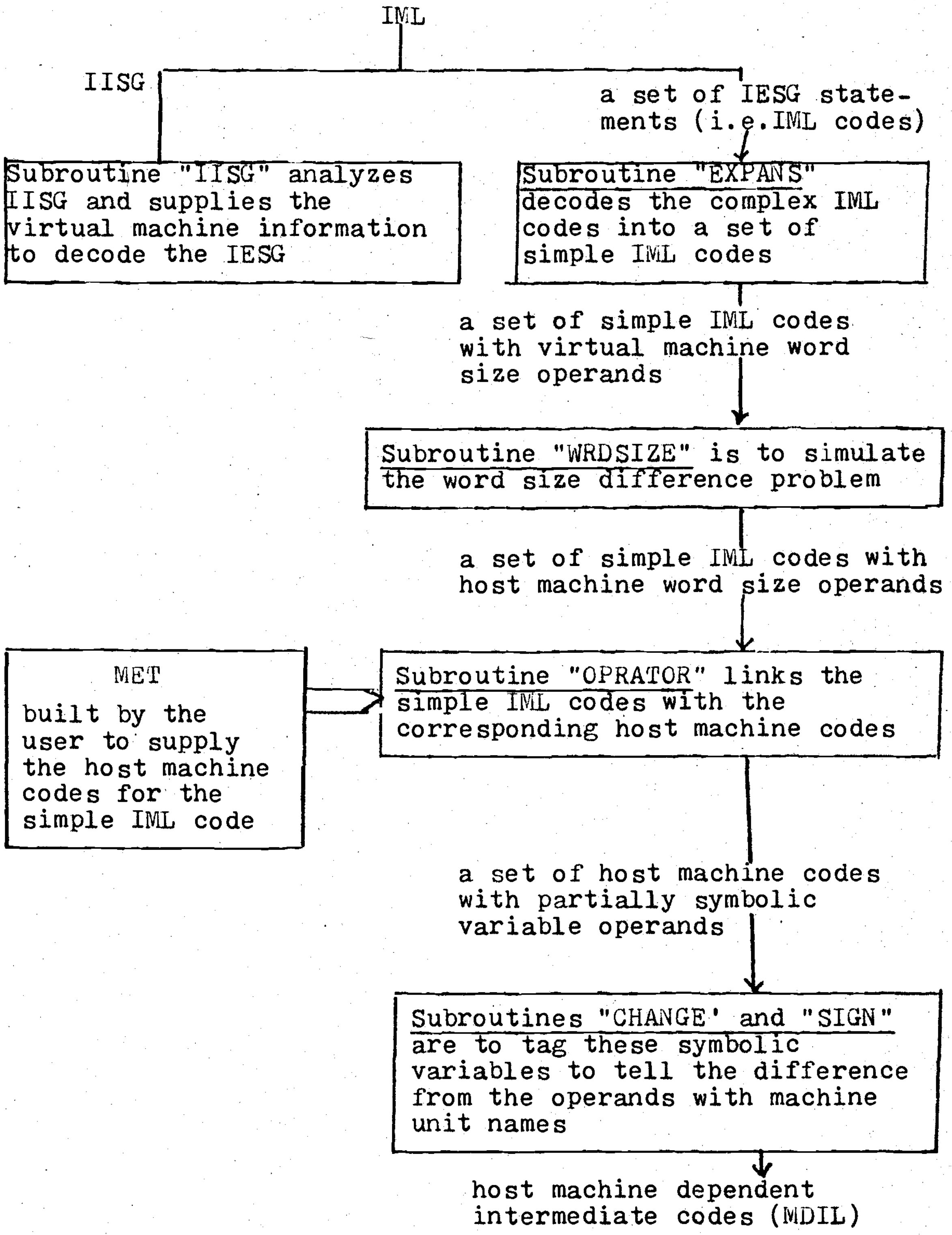

IML
IISG
a set of IESG statements (i.e.IML codes)
Subroutine "IISG" analyzes IISG and supplies the virtual machine information to decode the IESG
Subroutine "EXPANS" decodes the complex IML codes into a set of simple IML codes
a set of simple IML codes with virtual machine word size operands
Subroutine "WRDSIZE" is to simulate the word size difference problem
a set of simple IML codes with host machine word size operands
MET
built by the user to supply the host machine codes for the simple IML code
Subroutine "OPRATOR" links the simple IML codes with the corresponding host machine codes
a set of host machine codes with partially symbolic variable operands
Subroutines "CHANGE' and "SIGN" are to tag these symbolic variables to tell the difference from the operands with machine unit names
host machine dependent intermediate codes (MDIL)

Figure 3-1. Functional Flow Chart of Pass 1

Algorithm 3-1. General Structure of Pass 1

Comment: Pass 1 maps the machine independent IML code to the host machine dependent code (MDIL). The host machine information is included in the FDM. To each simple IML code there is a corresponding set of host machine codes in the Macro Expansion Table (MET). Subroutines IISG, EXPANS, WRDSZE, OPERATOR, CHANGE, and SIGN are used.

```
BEGIN
CALL SUBROUTINE IISG TO DECODE THE IISG TO GET THE VIRTUAL
MACHINE HARDWARE INFORMATION
READ A STATEMENT OF IESG AND DECIDE IT
IF IT IS A COMPLEX IML CODE
   THEN CALL SUBROUTINE EXPANS TO DECODE IT INTO A SET OF
      SIMPLE IML CODES
IF THERE IS A WORDSIZE DIFFERENCE BETWEEN VIRTUAL MACHINE
AND THE HOST MACHINE
   THEN CALL SUBROUTINE WRDSIZE TO RESOLVE THE DIFFERENCE
IF THERE IS AN ARITHMETIC MODE DIFFERENCE
   THEN MODIFY THE ASSOCIATED OPERATIONS
CALL SUBROUTINE OPRATOR TO LINK THE SIMPLE IML TO THE MET
AND DECODE IT INTO A SET OF BASIC MACHINE CODES
CALL SUBROUTINES CHANGE AND SIGN TO ADD THE SPECIAL SYMBOL
TO THE VARIABLES WHICH ARE TO BE REGISTER ALLOCATED
END.
```

machine information is collected in Table 3-1 and will be used later.

▶▶Example 3-4:

Consider the following partial description of the PDP8 target machine in IISG:

```
00A  PDP-8                ; name of virtual machine
00D  ..,12                ; 12-bit words
00E  TWO                  ; two's complement arith-
                            metic
221  MEM,4096,12          ; 4096x12-bit main memory
220  ACCM                 ; accumulator is a global
                            variable
 .
 .
 .
214  LNK,,1,1             ; link bit register is one-
                            bit long
 .
 .
 .
005  OPCODE,,9,11,-9      ; opcode is a field in bit
                            position
 .                        ; 9 through 11 that is
 .                          shifted
 .                        ; right 9 places (-9) when
                            used
00G  EFTADR
207  PC
 .
 .
 .
120  ADR
120  MART
 .
 .
 .
```

Table 3-1. Virtual Machine Information from IISG

Item	Usage
PROGNAM	program name
WDSZE	word size
ARTH MOD	arithmetic mode
MEMDIM MEMSZE	memory dimension x memory size
SUNA	subblock name
EXNA	external block name
FGNA, FGST	flat name and its flag setting
IDNA, IDADR	global and local variable name and their location in host memory
CHAR, BALUE	field variable name and its associated constant
Block Information	block name, block index and the global variables in this block
OTHERS	stack information if it exists in the target machine

In the partial PDP8 emulator above, the global variables are MEM and PC; the local variables are ADR and MART.

The following vector is mapped into main memory locations supplied by the user of the translation system:

Variable name	Host memory location	Comment
ACCM	2000	PDP8 accumulator
PC	1000	PDP8 program counter
.		
.		
.		
ADR	2003	PDP8 effective address
.		
.		
.		

Flag name	Corresponding flag	Comment
LNK	carry	PDP8 carry register
.	.	
.	.	
.	.	

Field variable	Value range	Comment
OPCODE	9,11,9	PDP8 opcode field
.		
.		
.		

*** Subroutine EXPANS ***

As was mentioned in the last section, most machines do not supply the corresponding machine primitive operations to decode the complex IML code directly. In order to reduce the burden from the user, an intermediate step is needed to do the transformation from the complex IML code into a set of simple IML codes. Then, the user provides only the machine codes (MET) for the simple IML code, not for this complex IML code. Refer to Figure 3-1, where subroutine EXPANS is used to expand the complex IML code into the simple IML codes.

►►Example 3-5:

```
   LOOP  SRC1  SRC2  SRC3    ; loop for SRC1=SRC2 to SRC3
                             ; by 1
```

This complex IML code "LOOP" is decoded into the following set of simple IML codes:

```
      MOVE  SRC2  SRC1        ; copy (SRC2) to (SRC1)
L.001 COMP  SRC3  SRC1        ; compare (SRC3):(SRC1) and
                              ; set host flags
      CONDT N     LL.002      ; if true, skip to L.002
      INC   SRC1              ; otherwise increment SRC1
       :
      BRCH  FL.001            ; and jump back to L.001
L.002 (next IML code)
```

The user has only to provide the MET for the above simple IML codes instead of decoding the operation "LOOP."

Another example is the complex IML code "ADD" with flag carry setting:

```
       ADD   SRC1   SRC2   Dest flag C
```

which is used to perform addition and set virtual machine flag carry. This flag is declared as a variable name, varc, in the IML emulator. In the host machine, the set of carry flag can be shown from the bit 0 of PS register.

The corresponding set of simple IML codes is:

```
        ADD  SRC1  SRC2  Dest  ; the comment is described
        MOVE .PS,0       varc  ; in example 3-3.◀◀
```

*** Subroutine WRDSZE ***

Refer to Figure 3-1, this subroutine is used to solve the problems of word size difference between virtual machine and the host machine. This assumes that host microprogrammable computers can provide the facility to set flags.

In case the word size of the host machine is greater than the word size of virtual machine, the host machine flag-setting facilities can be used to set virtual machine flags by left-justifying the host machine register, zero filling the remaining bits of each register.

▶▶Example 3-6:

Suppose the target machine is the PDP8 (12 bits), and the host machine is the 16-bit PDP11/40E. All variables declared in the IML emulator for the PDP8 are to be loaded into the 12 most significant bits of each PDP11/40E register. This is done by modifying the appropriate IML codes. For example, the IML increment code,

```
        INC  SRC1               ; add one to SRC1
```

is expanded into,

```
        ADD  SRC1  c16  SRC1    ; add constant 16 to SRC1 and
                                ; put into SRC1
```

where the constant one has been shifted left four bits to get 16. This is then mapped into machine code, as further illustrated by the following examples:

```
    DEC  SRC1                  ; subtract one from SRC1
```

is expanded by :

```
    SUB  SRC1  c16  SRC1       ; subtract 16 from SRC1
```

and,

```
    NOT  SRC1       Dest       ; one's complement SRC1
```

is expanded by:

```
    NOT  SRC1                  ; one's complement the top
                               ; 12 bits

    AND  Dest c65520 Dest      ; and then fill-in the lower 4
                               ; bits
```

In addition to arithmetic and logical modifications, the operands may need to be changed.

Before;

```
    CONDT  .PC,7  L.001   ; test bit 7 of the variable
```

PC and after;

```
    CONDT  .PC,11 L.001   ; test bit 7+4=11 of PC
```

i.e., the 7th bit of PC is left shifted to the 11th bit in host machine. Constants are modified by 2** (word size difference).

Before;

```
    MOVE  c8     AB       ; copy 8 into AB
```

and after;

```
    MOVE c128    AB       ; copy 2^4* (8) into AB
```

The other IML codes that need to be modified when conforming to larger host machine words are;

SHR, SHL, SLCT, and EXTR. ◄ ◄

If the host machine does not provide a facility to set flags, the problem of target-host mismatching must be solved by the user. Further, as a virtual machine program is loaded into the host main memory, each 12-bit word must be shifted before loading it into the 16-bit host machine memory.

In case the virtual machine word size is an integer multiple, n, of the host machine size, before the IML variable can be mapped into either host memory or a host register, this variable has to be bound into n segments. Each segment is the host machine word size. Then, n registers and n memory locations for each variable are needed when the load/store operation is used between the host machine memory and GPR. When a statement in IML is taking into account this kind of word size problem, we have to

1) decode the statement which includes each operand in the virtual machine word size into a set of IML statements which include each operand in the host machine word size.
2) modify the load/store operation so that one IML variable is associated with n host registers and n host memory locations.

The following example will illustrate this point.

▶▶Example 3-7:

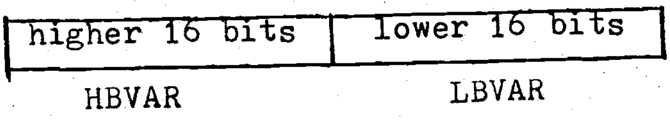
Assume the virtual machine wordsize is 32 bits and the host machine is the 16 bit PDP11/40E. Each variable declared in the IML emulator for the virtual machine is to be loaded into two host registers. This is done by the following steps.

For example the IML addition statement;

```
ADD  SRC1  SRC2  Dest (stmt 1); add SRC1 and SRC2 to Dest,
                                ; and each operand is in the
                                ; virtual machine word size
```

Step 1: Bind each variable into two segments. One is the higher 16 bits of variable, denoted by HBVAR, the other is the lower 16 bits of variable, denoted by LBVAR, i.e.,

variable in 32 bits



Step 2: Decode stmt 1 into another set of IML statements in which each operand is in the host machine word size.

Stmt 1 is expanded by:

```
    ADD  LBSRC1  LBSRC2  LBDest ; add lower 16 bits of SRC1
                                ; and SRC2 to Dest, and set
                                ; host machine flags

    CONDF Carry          L.001 (stmt 2); if no carry, go to
                                ; L.001

    INC   HBSRC1                ; increment higher 16 bits
                                ; of SRC1 by one
```

```
L.001 ADD  HBSRC1  HBSRC2  HBDest ; add higher 16 bits of
                                  ; SRC1 and SRC2 to Dest
```

The above codes are another set of IML statements, and each operand is in the host machine word size.

Step 3: The Macro Expansion Table is used to expand each statement into a set of machine code (Here, we skip the expansion of stmt 2).

```
      MOVE1  LBSRC1    B (stmt 3) ; move LBSRC1 into
                                    register B

      ADD    LBSRC2 B  D          ; LBSRC2+B→D

      MOVE5  D         LBDest     ; move the result into
                                    LBDest

      FLAG

      CONDF  carry     L.001      ; check carry flag

      INC    HBSRC1     (stmt 4)  ; increment HBSRC1 by one

L.001 MOVE1  HBSRC1    B

      ADD    HBSRC2 B  D

      MOVE5  D         HBDest
```

The above codes are a set of machine codes and each operand is either a machine unit name (for example, register B or D) or a symbolic variable in the host machine word size (for example, LBSRC1, LBDest, or HBSRC2).

Step 4: The load/store operation which is used to transfer the variable between host memory and GPR must have the following function:

"As the variable LBVAR is to be loaded into GPR, the load operation will load LBVAR into R_h and HBVAR into R_{h+1}

together. Similarly, either R_h or R_{h+1} is to be deallocated. Both the contents of R_h and R_{h+1} will be stored in the memory." For example, in stmt 3 of step 3, as LBSRC1 is to be allocated into the GPR, we allocate LBSRC1 into R_0 and HBSRC1 into R_1. In stmt 4, as the variable HBSRC1 is first read, we know it is in R_1 already. Later, if either R_0 or R_1 is to be deallocated, both the contents of R_0 and R_1 will be stored back in host machine memory. The other examples are illustrated in Appendix D. ◀ ◀

*** Subroutine OPRATOR ***

Refer to Figure 3-1, this subroutine is used to map the simple IML code to a set of basic host machine codes. To each simple IML code, there is a corresponding set of machine codes which are stored in MET as provided by the user. This subroutine provides a link to connect them.

▶▶Example 3-8:

In the second case of example 3-5, a complex IML code is decoded into two simple IML codes. Then, as shown in example 3-3, each simple IML code as defined by its associated set of basic machine codes stored in MET, is mapped into the basic codes of the host machine by Macro Expansion Table. For example, an IML addition corresponds to seven basic machine codes. When the proper variable names are substituted into the codes, we get the following

MDIL code:

Before expansion we have;

```
   ADD  ACCM  MDR  ACCM  C   ; IML addition and set virtual
                             ; machine carry flag
```

which becomes after expansion:

```
   MOVE1 ACCM             B   ; move from ACCM to host machine
                              ; register B

   ADD   MDR      B       D   ; add MDR and register B to
                              ; register D

   MOVE5  D            ACCM   ; move from register D to ACCM

   FLAG                       ; set carry flag

   PUSH3  PS            TOS   ; move register PS to the top
                              ; of stack

   RSMK   TOS 0,15,0    D     ; see example 3-3

   MOVE5  D             LNK   ; move from register D to LNK ◀◀
```

In the above example, registers B, PS, TOS, and D are the machine unit names. Symbols ACCM, LNK, and MDR are the variables declared in IML which are to be allocated to the general purpose registers in pass 2.

*** Subroutines CHANGE and SIGN ***

In order to tell the difference between variables declared in IML and host machine unit names, these two subroutines of Figure 3-1 assign the symbol (*) (1 or 2) (+ or -) to the IML variables which need be register allocated. Each block which is defined in section 3-2 is used as the basic unit when the assignment is processed.

A detailed definition of this symbol is shown in Table 3-2.

Table 3-2. TAGs of the Variable

(*)(n)(sign)(variable)	Explanation
*1+variable	It is a global variable and will be used later in this block.
*1-variable	This global variable will not be used in the current block, but it may be used in the next blocks.
*2+variable	It is a local variable and will be used later in this current block.
*2-variable	This local variable will not be used any more.

.Code '*' means the variable is to be register allocated.
.Code 'n' is either 1 or 2.
.Code 'sign' is either '+' or '-'.
.Code 'variable' is the variable name to be processed.

►►Example 3-9:

Assuming that codes of example 3-8 consist of a single block. ACCM and LNK are global variables, and MDR is a local variable, then the final result of pass 1 yields:

```
MOVE1    *1+ACCM     B         ; for comments see
                               ;
ADD      *2-MDR  B   D         ; example 3-8
MOVE5    D           *1-ACCM   ;
FLAG
PUSH3    PS          TOS       ;
RSMK     TOS 0,15,0  D         ;
MOVE5    D           *1-LNK    ;
```

Each statement described above is a host machine code defined directly from the FDM model. Operand tagged with symbol "*" is the symbolic variable which will be allocated into the general purpose registers in pass 2.

With the aid of the Macro Expansion Table supplied by the user, pass 1 produces a set of host machine dependent intermediate codes (MDIL) consisted of a set of blocks that can be the input of pass 2.

CHAPTER IV

PASS 2

4-1 Introduction

Pass 2 accepts a set of single entry-multiple exit segments called control blocks which are directly from the output of pass 1. Each block is a collection of MDIL statements consisting of machine dependent, executable statements with partially symbolic operands. The purposes of pass 2 are to allocate the symbolic operand to one of the general purpose register (GPRs) of the actual host machine and assign the corresponding host binary microcode to each statement of MDIL.

In general, the number of symbolic variable operands in a given program is greater than the number of registers in the host machine. Thus, the register must be shared by more than one symbolic operand. Register allocation/deallocation is a major factor in producing efficient code. "Active" operands are held in the registers and swapped to main memory when they become latent or "passive." As the number of swaps increases, the efficiency of the executable code decreases.

Within the block, more than one branch statement may jump to the same label statement. Thus, different symbolic variables may use the register *at the same time* which in turn involves the control flow interface problem (see section 4-4). This interface problem can be made less burdensome by structuring the blocks of MDIL code. Each block is analyzed for its flow of control governed by two legal control structures--the branch statement and the label statement. These two statements divide the block into a set of straight line codes (SLC) which are sets of single entry-single exit statements.

We define the "state" of a SLC as the assignment of operands to GPRs for the given SLC. Upon entry to the SLC we must define an initial state IS_i for SLC_i, and we define the final state FS_i as the state of SLC_i when register allocation is completed.

When the RA/D scheme is applied, the SLC is used as the basic unit of program segment. At the end of each SLC, this scheme will continue with the next SLC after the initial state of the following SLC is determined. During the execution of the RA/D scheme on each statement, the host machine field values and their timing phase are assigned to each MOP.

The functional flow chart and the general structure of pass 2 are described in Figure 4-1 and Algorithm 4-1, respectively, which tell how each branch statement and label statement separate the block into SLC segments and lead to the associated tasks with each SLC.

The general terminology of pass 2 is described in

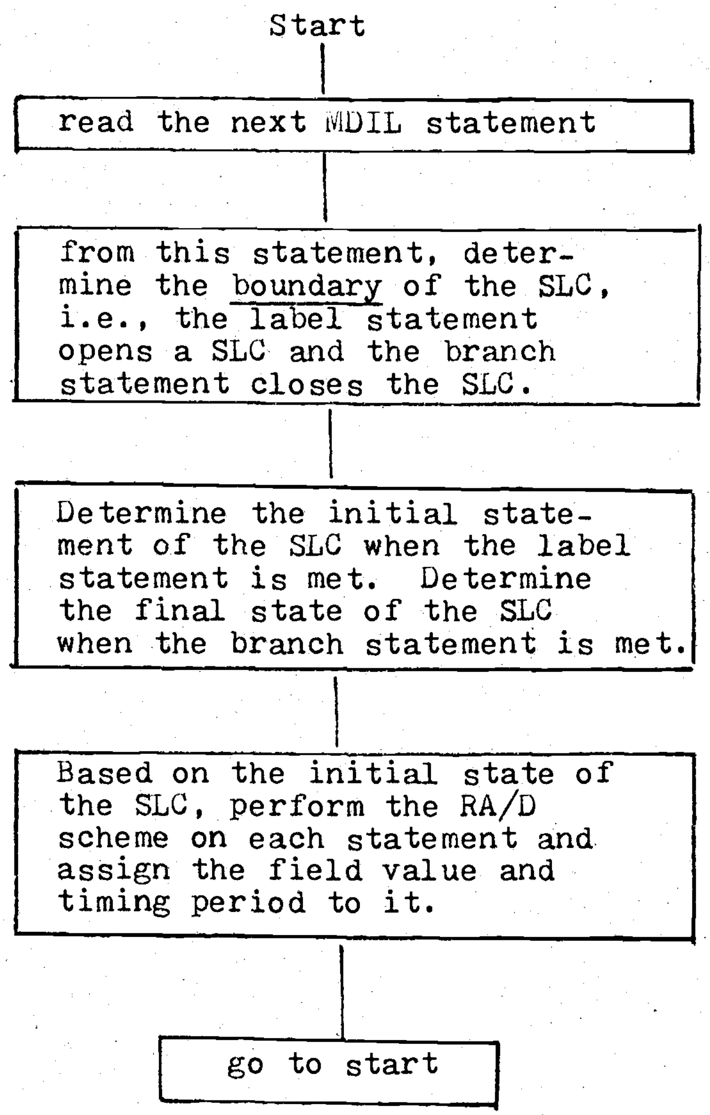

Start
read the next MDIL statement
from this statement, determine the boundary of the SLC, i.e., the label statement opens a SLC and the branch statement closes the SLC.
Determine the initial statement of the SLC when the label statement is met. Determine the final state of the SLC when the branch statement is met.
Based on the initial state of the SLC, perform the RA/D scheme on each statement and assign the field value and timing period to it.
go to start

Figure 4-1. Simplified Flow Chart of Pass 2

Algorithm 4-1

Program: General Structure of Pass 2

Data: I is index of SLC.
IS(I) is the initial state of SLC(I).
FS(I) is the final state of SLC(I).

Pseudo code:

```
        BEGIN
(START) FETCH NEXT STMT
        IF THE CURRENT STMT IS A LABEL STMT (the beginning
           of SLC(I)
           THEN BEGIN
                FILL THE LABEL TABLE (see Algorithm 4-3)
                IF THE PREVIOUS STMT IS NOT A BRANCH STMT
                   THEN DETERMINE FS(I-1) (see Algorithm
                        4-5, 4-7, 4-8)
                DETERMINE IS(I) (see Algorithm 4-4)
                GO TO AA
                END
           ELSE BEGIN
                IF THE PREVIOUS STMT IS A BRANCH STMT (the
                   end of SLC(I-1)
                   THEN DETERMINE IS(I)
       (AA)     IF THE CURRENT STMT IS A BRANCH STMT (the
                   end of SLC(I))
                   THEN BEGIN
                        FILL THE LABEL TABLE
                        BASED ON THE POINTER TO DETERMINE
                        FS(I) (see Algorithm 4-3)
                        END
           ELSE BEGIN
                   PERFORM RA/D SCHEME ON THE STMT (see
                   Algorithm 4-2)
                   ASSIGN FIELD AND PHASE TUPLES TO THE STMT
                   END
           GO TO START
           END
END.
```

section 4-2. The details of the register allocation scheme and field value computation are given in section 4-3. The control flow interface problem is discussed in section 4-4. The initial state and the final state of a SLC are described in section 4-5 and section 4-6, respectively.

4-2 Definitions and Terminology

Some general components of pass 2 are introduced first, and other special terms are explained in more detail when they are used in later sections.

1) OPND= {OPND1, OPND2} is a set of operands, where OPND1 is a set of machine unit names, and OPND2 is a set of symbolic variables to be register allocated.

2) GPR= {R(1), R(2).... R(NR)} is a set of host machine general purpose registers used to hold the operand values during execution of the statement. R(J) is defined as jth register in the set of GPRs, where $1 \leq J \leq NR$.

3) VML is a set of variable memory locations which are in the host machine main memory and are used to hold the variable values when deallocated from the general purpose registers.

4) A program consists of a set BK= BK_1, BK_1.... BK_{BNK} of blocks. Each block starts with a special code BKS, and is a single entry-multiple

exit collection of straight line codes.

5) A straight line code, SLC, is a single entry-single exit set of statements. There is an index I to each SLC, denoted by SLC(I), which orders the SLC in the program. SLC(I) and SLC(K) are said to be in sequential order. I<K, we say SLC(I) precedes SLC(K).

6) Each statement of a SLC segment is given as:
LB(I), OP(I), ODA(I,1), ODA(I,2), ODA(I,3)
where I is the index of the statements in the program, and LB(I) is the label of the statement.
OP(I) is the MOP name which can be found from Field Description Model.
ODA(I,1) and ODA(I,2) are the elements of set OPND and are used as the source inputs of OP*I).
OPA(I,3) is from set OPND and used as the output destination of OP(I).
Symbolic variables can be used as operands of SLC statements.

7) The label statement is defined if LB(I) is not empty. The branch statement is defined if OP(I) is a branch operation and ODA(I,1) is a label name. Branches are either forward or backward branches depending on the direction of the branch.

8) The state of register GPR(J) during the execution

of SLC(I) is denoted by:

SR(I,J)= SA(J), ST(J), TY(J), PT(J)

R(J) is the jth register in the GPR.

SA(J) is the variable name currently held in R(J).

ST(J) is the status of the variable in R(J).

TY(J) is the type of this variable.

PT(J) is the position of the variable in the statement.

The detailed description is shown in Table 4-1. The states of GPR in SLC(I), denoted by S(I), are a set of states of R(J), where J=1 to NR, and are represented by:

$$S(I) = \bigcup_{J=1}^{NR} SR(I,J)$$

9) The operation which is used to load and store variables between main memory and the central processor exists both in the original IML and pass 2 level, but they are processed in different ways.

a) In the IML level, operation RMOVE and WMOVE are used for reading and writing into the variable memory of the virtual machine (VM). The format is:

RMOVE SRC1 SRC2 Dest ; Dest ← Mem(SRC2)

VMOVE SRC1 SRC2 Dest ; Mem(SRC2) ← Dest

SRC2 is the address value of the memory, and Mem. (SRC2) is the content of this

Table 4-1. Components of SR(I,J)

	Action
Status ST(J)	
Active	The value of the variable in R(J) is different from the content of the same variable stored in VML.
Passive	The value of this variable is the same between the VML and the register.
Position PT(J)	
Source	This variable is used as source in the statement.
Dest	This variable is used as destination in the statement.
Type TY(J)	
1	This global variable will be used later in this current block.
3	This global variable will not be used in the current block.
2	This local variable will be used later in this current block.
none	This local variable will not be used in the current block.
Reference	SLC(I-1) and SL(I) are in sequential order, if SLC(I-1) has an unconditional branch then the final state of SLC(I-1) cannot be used by SLC(I), but can be considered as a reference state. In this case, such variables are assigned to type reference which means the register does not really contain the variable.

address.

In pass 2, the variable memory of virtual machine is mapped into the host main memory and the operations RMOVE and WMOVE are decoded into a set of basic machine codes. The following example will illustrate how the RMOVE and WMOVE are implemented by the set of PDP11/40E microcodes.

▶▶Example 4-1:

```
 RMOVE  Mem  PC  IR  ; IR<-Mem(PC)
```

This means the memory content of PC is read into a register IR (instruction register). The corresponding DIL codes are:

```
 MOVE8 *1+PC   BA    ; copy the address of
                     ; PC to Bus address
                     ; register, set
                     ; DATI, and then turn
                     ; off processor clock

 MOVE4 unibus *1-IR; copy the value of PC
                     ; to IR
```

This means the address of PC is moved to the bus address register (BA), and then the memory content of this address is moved to the register which holds the IR. In the statement MOVE8, the first operand is the address value of the variable instead of its content.

Similarly, an example of a WMOVE operation:

```
WMOVE Mem  MAR  -T.001 ; Mem(MAR ← -T.001
```

The corresponding DIL codes are:

```
MOVE2  *1+MAR    BA ; copy the address of
                    ; MAR to register BA

MOVE9  *2-T.001 D   ; copy the value of
                    ; (-T.001) to register
                    ; D, set DATO, and then
                    ; turn off processor
                    ; clock
NOOP                ; no operation
```

This means the address of MAR is moved to BA. Then the content of -T.001 is moved to register D, and the machine stores the content of D into the address which is in BA.◄◄

b) In the register allocation/deallocation scheme (the level of pass 2), MEMREAD and MEMWRITE statements are used to communicate between a GPR and the main memory of the host machine. In the most general case, the host machine cannot implement these statements in one machine cycle. However, the execution procedure is different in various machines. The general format of the MEMREAD statement used in this chapter is:

```
MEMREAD variable register ; register ←
                          ; Mem(variable)
```

This means the content of the variable is loaded into the register. The variable is declared in the IML level and is assigned a host memory address. This statement is decoded into the PDP11/40E microcodes:

```
MOVE11 variable  BA ; copy the address
                    ; of "variable" to
                    ; BA register, set
                    ; DATI, then turn
                    ; off processor
                    ; clock.

MOVE4  unibus register
                    ; copy the value of
                    ; "variable" to
                    ; "register"
```

It is useful to compare the difference between the operation RMOVE and the statement MEMREAD as given above. One is from the IML level; the other is from the pass 2 level. The first operand of statement MOVE8 is stored in the register, but, in statement MOVE11, it is displayed by an emit value.

In the example 4-1, MEMREAD statement cannot be used when the address value of PC is loaded into the register. The statement:

```
MEMREAD PC register ;register← Mem(PC)
```

means to load the contents of PC into a register. This feature should be carefully considered in the scheme and field value computation. The general format of the MEMWRITE statement is:

```
   MEMWRITE register variable ; Mem(re-
                              ; gister)
                              ; variable
```

and the corresponding PDP11/40E microcodes are:

```
   MOVE12 variable BA ; copy the address
                      ; of "variable" to
                      ; register BA

   MOVE9  register D  ; copy the value of
                      ; "register" to
                      ; register D, set
                      ; DATO, and then
                      ; turn off processor
                      ; clock

   NOOP               ; no operation
```

For the same reason, the reader may compare the difference between WMOVE in IML and MEMWRITE in the pass 2 level.

4-3 Register Allocation/Deallocation Scheme

The input to pass 2 from pass 1 of the translation system is a set of machine dependent, executable statements, in which some operands still reference symbolic variables. Before the binary microcode can be completely assigned to any one statement, the symbolic variable operands must be allocated to the general purpose registers of the actual host machine. In general, the number of GPRs in the host machine is less than the number of variables in the program.

That means these variables cannot stay in the GPR forever, and some variables must be stored in the host machine memory and loaded into the GPR when they are recalled. There need to be some extra MEMREAD or MEMWRITE statements to move operands between the GPR and host machine memory. These "extra" memory references influence the efficiency of object code.

The general idea of the RA/D scheme is to keep the variables in the corresponding registers as long as possible until no available register is free for the next new variable. When the set of general purpose registers is full of variables, the register deallocation process is used to free a register for the new variable. A decision must then be made as to which old variable in the registers should be replaced first so that the number of MEMREAD or MEMWRITE statements is kept as small as possible. The efficiency of the RA/D scheme is highly dependent on the priority assignment of variables.

4-3-1 Replacement Priority Assignment

The replacement priority is determined by the status and type of each variable. When an "active" status variable is to be deallocated, a MEMWRITE statement is needed to store this variable in the host machine memory. However, an extra MEMWRITE statement is not necessary for a "passive" status variable. Combinations of status and type, and the replacement priority of variables are described in Table 4-2.

There is one kind of variable which cannot be deallocated, regardless of the priority of the variable. The register which holds the first operand of a statement cannot be deallocated until the second operand of this statement is register allocated. The following example will illustrate this idea:

Example 4-2:

This statement

```
ADD  *2-AB  *1+BC  *1+BC   ; AB+BC→BC
```

is to be register allocated. In the worst case, assume that after R1 is allocated to variable AB, all registers are full, and R1 containing the variable AB has the highest priority to be deallocated. If R1 is not protected, the output will be:

```
MEMREAD  AB  R1        ; R1←Mem(AB)
MEMREAD  BC  R1        ; R1←Mem(BC)
ADD      R1  R1  R1    ; R1+R1→R1
```

In the third statement both the first and second R1 hold the value of variable BC and this gives an incorrect result. Thus, it is necessary to protect the register which holds the first operand of one statement from deallocation. This restriction can be dismissed after the second operand of this statement is register allocated.

Table 4-2. Replacement Priority Assignment

*priority	type	status	action
1	none	passive	Local variable with passive status will not be used in the rest of the current block.
2	ref	do not care	This variable does not actually exist in the register.
3	none	active	Same as (1) but with active status.
4	3	passive	Global variable with passive status will not be used in the rest of the current block, but may be used in the next blocks.
5	3	active	Same as (4) but with active status.
6	2	passive	Local variable with passive status will be used in the rest of the current block.
7	1	passive	Global variable with passive status will be used in the rest of the current block.
8	2	active	Same as (6) but with active status.
9	1	active	Same as (7) but with active status.

*The smaller value in this column has the higher priority to be deallocated.

This limitation will be good for any machine as long as the number of GPRs is greater than one.◄◄

Refer to Algorithm 4-1, the RA/D scheme is divided into the following Algorithms.

4-3-2 RA/D Algorithm

The whole process which is described in Algorithm 4-2 can be described by the variation of the state of GPR when the operand is register allocating. Each SLC is treated independently of other SLCs when the RA/D scheme is applied. Within the SLC, the scheme is performed operand by operand; then, statement by statement.

4-3-3 Tuple5 Scheme

When the FDM is given by a user, the microinstruction format is divided into separate fields, and the value of the field which is assigned to each MOP is classified in two ways. One is by the numerical value which has already been defined. The other is by the alphabetical value which will be determined in this section.

Now, we use the FDM of PDP11/40E (Appendix B) and some examples to illustrate the function of Tuple5. The set of undetermined field values in FDM are described in Table 4-3.

Algorithm 4-2

Program RA/D Scheme
Data: ODA(M,K) is the kth operand of stmt M in SLC(I) and
is decoded by:

SY(1) is the first character of the operand.
SY(2) is the second character of the operand.
SY(3) is the third character of the operand.
SY(4) are the remaining characters of the operand.
R(J) is the jth register in GPR, $1 \le J \le NR$.
SA(J) is the variable name held by R(J)
ST(J) is the status of SA(J).
TY(J) is the type of SA(J).
(The detail definition and function of these program parameters are described in section 4-2.)

Pseudo code:

```
            BEGIN
(FETCH)     FETCH NEXT OPERAND, ODA(M,K)
            IF ODA(M,K) IS A MACHINE UNIT NAME
               THEN GO TO FETCH
               ELSE BEGIN (This symbolic operand is to be
                           allocated to GPR)
                    CALL ALGORITHM 4-6 TO DETERMINE NS
                    IF ODA(M,K) IS IN THE GPR ALREADY, SAY
                    R(J)
                       THEN BEGIN (Determine the state
                                   variable SA(J), ST(J),
                                   TY(J))
                            SA(J) IS NOT CHANGED
                            CALL SUBROUTINE TYPE TO DETER-
                            MINE TY(J)
                            IF K=3 (This operand is destin-
                            ation)
                               THEN ST(J)=ACTIVE
                               ELSE ST(J) IS NOT CHANGED
                            END
                       ELSE BEGIN (This operand is not in
                                   the set of GPR)
                            IF THERE IS A FREE REGISTER,
                               R(J), IN GPR
(FREE)   THEN BEGIN
             IF K=3 (This operand is destination)
                THEN BEGIN
                     SA(J)=ODA(M,K)
                     ST(J)=ACTIVE
                     CALL SUBROUTINE TYPE TO SOLVE TY(J)
                     END
```

Algorithm 4-2 (continued)

```
                   ELSE BEGIN (This operand is source)
                        MEMREAD  ODA(M,K)  R(J)
                        (load the operand into R(J)
                        SA(J)=ODA(M,K)
                        ST(J)=PASSIVE
                        CALL SUBROUTINE TYPE TO SOLVE TY(J)
                        END
                   END
              ELSE BEGIN (There is no free register in GPR)
                   FROM TABLE 4-2, DEALLOCATE THE HIGHEST
                   PRIORITY VARIABLE IN GPR, SAY R(J)
                   IF ST(J)=ACTIVE
                      THEN "MEMWRITE  R(J)  SA(J)
                      IF ST(J)=ACTIVE
                      THEN "MEMWRITE  R(J)  SA(J)*
                           (store the content of R(J) into
                           memory)
                      GO TO FREE
                   END
          END
      END
   END.
```

Subroutine TYPE

```
BEGIN
SEPARATE ODA(M,K) INTO SY(1), SY(2), AND SY(4)
IF SY(3)="+" (ODA(M,K) will be used later in the block)
   THEN TY(J)=SY(2)
   ELSE BEGIN (ODA(M,K) will not be used any more)
        IF SY(2)="2" (ODA(M,K) is a local variable)
           THEN TY(J)=NONE
           ELSE TY(J)="3"
        END
END.
```

Table 4.3. Undetermined Field of PDP11/40 FDM

Case	Format in the FDM/Field value determination
1	OP SRC1(GPR) SRC2 Dest(*GPR) Field(1)=function (the register used in the operand *GPR)
2	OP *emit Dest Field(18)=function (the constant used in *emit)
3	OP SRC1 $CT Dest or OP B TOS,CT D Field(15), Field(16), or Field(17) is a function of CT.
4	OP SRC1 $FF,LL,CT Dest Field(15), Field(16), and Field(17) are a function of FF, LL, and CT.
5	OP variable Dest Field(18)=function (address value of the variable)
6	Field(13)=function (next MOP address)

▶▶Example 4-2:

In case 3 of Table 4-3, one MOP in FDM is:

OP:RMASK

Input: TOS $CT

which means to mask out the right (16-CT) bits of TOS.

Now, in pass 2, the following MOP is to be field value assigned:

RMASK TOS 5 B

CT=5, field 16=CT-1=4.◀◀

▶▶Example 4-3:

In case 4 of Table 4-3, the format of MOP RSMK in FDM is:

FF LL CT
9, 11, -9

```
OP:RSMK
I : TOS $FF,LL,CT
field 15=LL-CT
field 16=15-FF+CT
field 17=CT
```

which means to right shift TOS CT bits, and then mask.

In pass 2, the following MOP is to be field value assigned:

RSMK TOS PGEADR D

Where PGEADR is a variable name which is associated with a bits range to be shifted or masked, the bits range associated with this variable is 0,6,0. Comparing PGEADR in pass 2 with FF,LL,CT in the format of the FDM, we have

FF=0, LL=6, and CT=0. The following field values are assigned to this MOP:

field 15=6, field 16=25, field 17-0. ◀ ◀

Example 4-4:

In case 5 of Table 4-3, the field value of the following MOP is to be assigned:

MOVE10 PC D

and the address value of PC is allocated to a fixed value in VML, say, PC=1000, then field 18=1000. ▲ ◀

4-4 Problems Arising from the Control Flow Interface

Before describing the RA/D scheme entering the next SLC or the next block, the interface problems are first considered.

1) The interface problems within the block

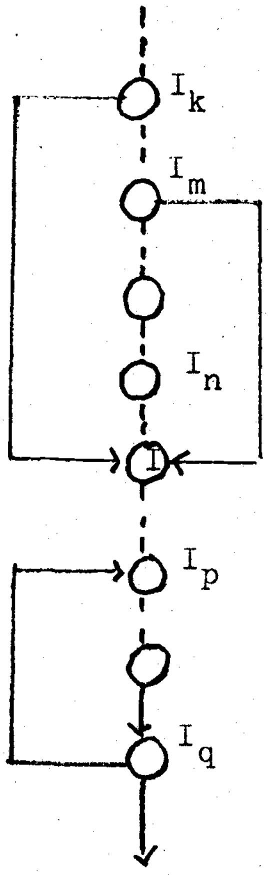
Figure 4-2 illustrates two typical examples. One is the forward branch case. The other is the backward branch case.

a) The forward branch case:

The final states (FS) of SLC(I_k), SLC(I_m), and SLC(I_n) have been determined already and will influence the initial state (IS) of SLC(I). Which state of GPR can be used as the IS of this SLC?

b) The backward branch case:

The IS of SLC(I_p) has been determined already.



1. Each circle means a SLC.

2. IS(I) is to be determined.

3. FS(I_q) is to be determined.

4. Each character, I_k, I_m ....
   I, or, I_q is a SLC index.

Figure 4-2. Forward Branch and Backward Branch

The FS of the SLC(I_q) is to be determined and depends on the IS(I_p). This backward branch region may be executed many times. How do we get the efficient interface to determine this FS?

2) Interface problems between blocks
Each block has a single entry point which is the first statement of the block and a set of its own local variables. When the interface occurs a problem arises in addition to the problems mentioned in condition (1). This is insuring that the local variables in FS of one block must not be used as the IS of the other block.

From the above analysis, it is evident that the interface problems can be solved by correctly finding the initial state and the final state of a SLC.

In order to find the initial state, the label statement has to record all SLCs which support the forward branch to this label. To find the final state, the direction of the branch statement has to be determined. There is a label table, described in Table 4-4, which is set up by the label statement and the branch statement in Algorithm 4-3, and used to record all information associated with each label. Based on this label table, the initial state and the final state of SLC are determined in the

Table 4-4. Label Table

Components	Functions
Label vector LBL	This is a label name vector which is used to record all labels according to the sequence in which the label appears in the whole program. LBL(1) is a label name with index I in the label vector.
SQ(I)	It is assigned to zero if the label appears in the label statement, and it is assigned to one if the label appears in branch statement. From this vector, the direction of branch statement can be determined.
SB(I)	It is used to count the number of forward branch statements to this label.
BS(I,J) J=1 to SB(I)	It is a matrix which is used to record the indexes of SLCs which support the forward branch statement to this label.
BWL(I)	It is used to count the number of backward branch statement to this label.
BWLB(I,J) J=1 to BWL(I)	It is used to record the indexes of SLCs which support the backward branch statement to this label.
BI(I)	If the label name is a block name then it is used to record the block index.
SLCD(I)	It is an index of the SLC which contains the label statement with label name LBL(I).

Algorithm 4-3.

Program: Label Table Determination

Data: LBL(J) is the label name.
SB(J) is the forward branch(f,b) counter of LBL(J).
BS(J,I) records all f.b. SLCs to LBL(J).
BWL(J) is the backward branch (b.b) counter of LBL(J).
BWLB(J,I) records all b.b. SLCs to LBL(J).
BI(J) tells if LBL(J) is a block name or not.
SQ(J) tells the direction of the branch.
SLCD(J) is the index of a SLC which contains LBL(J).

(The details are described in Table 4-4.)

Pseudo code:

```
BEGIN
IF THE LABEL NAME IS FROM THE LABEL STMT
   THEN BEGIN
           IF THIS LABEL IS IN THE LABEL TABLE
              THEN GO TO ASSIGN
              ELSE BEGIN
                      STORE THIS LABEL IN LBL(M)
                      IF LBL(M) IS A BLOCK NAME
                         THEN BI(M)=BLOCK INDEX
                         ELSE BI(M)=0
                      SB(M)=0 (set f.b. counter)
(ASSIGN)              SQ(M)=1 (label name appears in the label
                               position)
                      BWL(M)=0 (set b.b. counter)
                      SLCD(M)=CURRENT SLC INDEX
                      END
  ELSE BEGIN (it is from the branch stmt)
          IF THIS LABEL IS IN THE LABEL TABLE
          THEN GO TO TEST
          ELSE BEGIN
                  STORE THIS LABEL IN LBL(J)
                  SET SQ(J)=0, SB(J)=0
                  IF LBL(J) IS A BLOCK NAME
                     THEN BI(J)=BLOCK INDEX
                     ELSE BI(J)=0
(TEST)               IF SQ(J)=0 (it is a forward branch)
                        THEN BEGIN
                                SB(J) + 1 (INC the f.b. counter)
                                BS(J,SB(J))=CURRENT SLC INDEX
                                END
```

```
ELSE BEGIN
     BWL(J)=BWL(J)+1 (INC the b.b. counter)
     BWLB(J,BWL(J))+CURRENT SLC INDEX
     END
SET POINTED TO TELL THE BRANCH STATUS
(ref. to Algorithm 4-1, this pointer is
 used to determine FS)
END
```

next sections.

4-5 Initial State of SLC

The initial state of SLC(I), denoted by IS(I), is defined as the state of GPR immediately before entering this SLC(I). The IS of a SLC is actually determined from the FS of other SLCs, and used as the basis to perform the register allocation/deallocation scheme on the current SLC. To get a reliable IS is extremely important for pass 2.

Based on the above discussion, the IS(I) can be determined as follows:

From the label table, vector SB(label) tells the number of forward branches to this SLC(I), and the matrix BS(I,J), J=1, SB(label), lists all indexes of SLCs which supply the forward branch to this SLC. Now, with the assumption that:

SB(label=n,

and the indexes in BS are I_1, I_2.... I_n.

Case 1 if n=0 which means no forward branch to this SLC or SLC(I) is not a label SLC then IS(I)=FS(I-1).

Case 2:if n≠0, and SLC(I-1) is not an unconditional branch SLC then IS(I) can be expressed by $IS(I)=f_1(FS(I_1)$ ....$FS(I_n)$, FS(I-1)).

if SLC(I-1) is an unconditional branch SLC, then $IS(I)=f_2(FS(I_1)....FS(1_n))$.

To simplify the description, we have

$$IS(I)=f(FS(I_i)....FS(I_m))\text{-----------------------}(1)$$

Where the number of m is n or n+1.

Each FS or IS is a state of GPR. The further analysis follows:

$$IS(I)=\bigcup_{J=}^{NR} ISR(I,J)$$

$$FS(K) \bigcup_{J=}^{NR} FSR(K,J)$$

Where FSR(K,J) is the state of the jth register in the FS of SLC(K) and can be expressed by:

$$FSR(K,J)= \{FSA(K,J),\ FST(K,J),\ FTY(K,J)\}$$

FSA(K,J) is a variable name which is in the register J of the FS of SLC(K).

FST(K,J) is the status of the variable FSA(K,J).

FTY(K,J) is the type of the variable FSA(K,J).

Similarly, we have

$$ISR(I,J)= \{ISA(I,J),\ IST(I,J),\ ITY(I,J)\}$$

and the same explanation for each component of ISR(I,J).

Now, equation (1) is abbreviated as:

$$IS(I) = f(\sum_{k=1}^{m} FS(I_k))\ \text{---------------------}(2)$$

$$ISR(I,J)= f_j(\sum_{k=1}^{m} FSR(I_k,J))$$

The IS(I) of register J is determined by all the FSs of register J. The problem in finding the IS(I) is to solve the function f_j. Algorithm 4-4 is used to solve function f_j.

4-6 Final State of SLC

Refer to Figure 4-3 and 4-4. The branch statement which is the last statement of a SLC will bring a state to the sink SLC and leave a state to the next SLC. These two states may not be the same. The FS problem is actually to find these two states at the end of the current SLC. Some terminology will be used in this section.

1) The state immediately before the branch occurs in SLC(I) is denoted by CS(I).
2) After the branch statement, the state which will be brought to the sink SLC is called branch final state and denoted by FS(I). The state which will enter the next SLC is called the sequential final state and denoted by S(I).
3) Forward branch SLC is defined as a SLC in which the last statement of the SLC is a forward branch statement.
4) Backward branch SLC is defined as a SLC in which the last statement of the SLC is a backward branch statement.

The final state of a SLC may be from either the forward branch SLC or the backward branch SLC. They are determined as follows:

Algorithm 4-4

Program: Initial State of SLC(I)

Data: 1) There are m SLCs with indexes I_k, k=1 to m, forward branch to SLC(I). ISA(I,J), IST(I,J), ITY(I,J), FSA(I_k,J), FST(I_k,J), and FTY(I ,J) are defined in section 4-5.

2) A null state of register means no variable is assigned to this register and all information of this register is marked out.

3) $\overline{\text{Type}}$ means the complement of the type of the variable. If this variable is global variable, then $\overline{\text{type 1}}$ = type 3, and $\overline{\text{type 3}}$ = type 3. If this variable is a local variable, then $\overline{\text{type 2}}$ = type none, and $\overline{\text{type none}}$ = type none, and the type reference does not have the complement operation.

4) Operator π is defined as:

$$\pi A_i = \begin{cases} \text{passive, if all } A_i\text{'s are passive.} \\ \text{active, if one of } A_i \text{ is active.} \end{cases}$$

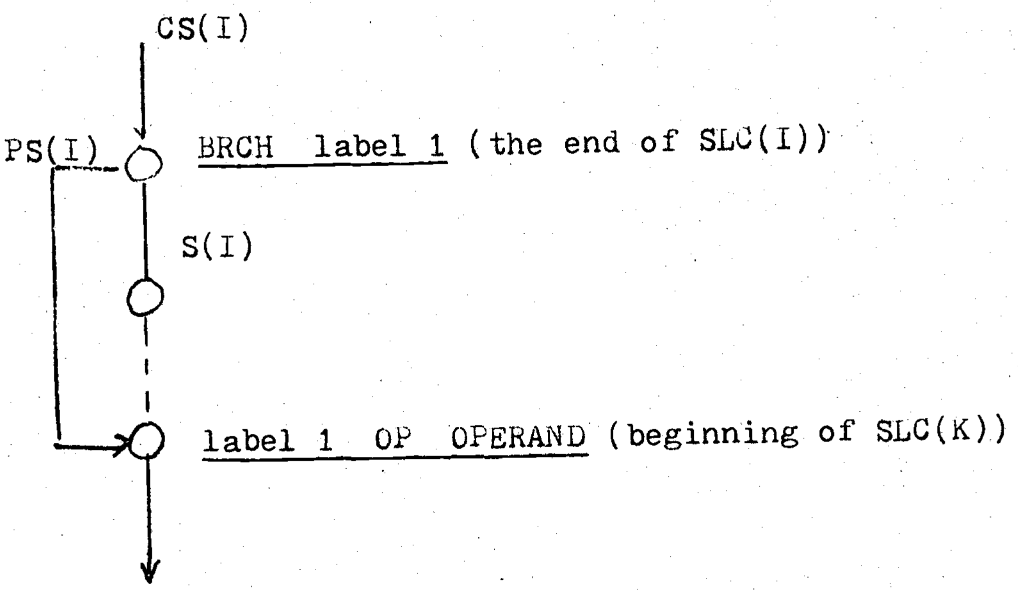
5) Vector VAR(L), where L=1 to VA, is defined in each block. 'VAR(L) to SLC(I)' means the vector stores all the variables which will not use any more from the SLC(I) to the end of the block.

Pseudo code:

```
BEGIN
IF ALL FSA ($I_k$,J), $1 \le k \le m$, ARE EQUAL (All variables in R(J)
   from the different SLCs, $I_1$, $I_2$,...and $I_m$, are the same)
   THEN BEGIN
        ISA(I,J)=FSA($I_k$,J) (Determine the variable in R(J)
        of IS(I))
        IST(I,J)=$\pi_k$ FST($I_k$,J) (Determine the status of this
        variable)
             THEN ITY(I,J)=$\overline{\text{FTY}(I_k,J)}$
             ELSE ITY(I,J)=FTY($I_k$,J)
        END
```

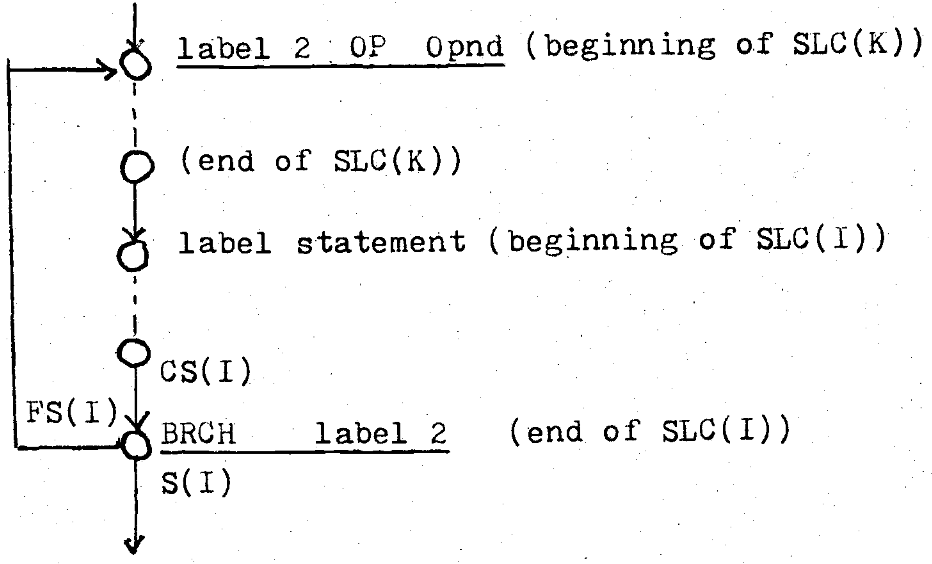
Algorithm 4-4 (continued)

```
ELSE BEGIN (One of the variables in R(J) from SLCs, $I_1$....
     $I_m$ is different from others)
     ISR(I,J) IS SET TO BE A NULL STATE
     FOR ALL k, $1 \leq k \leq m$
     IF FST($I_k$,J) = ACTIVE
        THEN "MEMWRITE  R(J)  FSA($I_k$,J)"  IS INSERTED
             AT THE END OF SLC($I_k$)
     END
IF SLC(I) IS THE FIRST SLC OF A BLOCK (Local variables of
   the previous block are not available here)
   THEN BEGIN
     IF ITY(I,J)=TYPE NONE (Reset the R(J) holding the
        local variable)
        THEN ISR(I,J) IS SET TO BE A NULL STATE
        ELSE BEGIN
             IF ISA(I,J) WILL BE USED IN THIS BLOCK
                THEN ITY(I,J)=TYPE 1
             END
        END
END.
```


CS(I)
PS(I)
BRCH label 1 (the end of SLC(I))
S(I)
label 1 OP OPERAND (beginning of SLC(K))

1. Each circle means a statement.
2. SLC(I) is a forward branch SLC.
3. SLC(K) is a sink SLC to SLC(I).
4. CS(I), FS(I), and S(I) are defined in section 4-6.

Figure 4-3. Final State of Forward Branch SLC


label 2 OP Opnd (beginning of SLC(K))
(end of SLC(K))
label statement (beginning of SLC(I))
CS(I)
FS(I)
BRCH label 2 (end of SLC(I))
S(I)

1. Each circle means a statement.

2. SLC(I) is a backward branch SLC.

3. SLC(K) is a sink SLC to SLC(I).

4. CS(I), FS(I), and S(I) are defined in section 4-6.

Figure 4-4. Final States of Backward Branch SLC

4-6-1 Final State of the Forward Branch SLC

The method used to determine the FS of the forward branch SLC (Figure 4-3) does not depend on the sink and can come directly from the current SLC. Algorithm 4-5 is used to describe the determination of this FS.

4-6-2 Next Initial State of Sink SLC

When a SLC(I) backwards branches to a SLC(K) (Figure 4-4), the state immediately before the branch statement must be the same as the initial state of the sink SLC, and the state just after the branch statement will go to the SLC(I=1).

The first problem to be determined is what initial state of SLC(K) will be used as a reference state by CS(I). From the last section, IS(K) is the state right before entering the SLC(K), but it does not involve any RA/D action about the SLC(K). The next initial state of SLC(K), denoted by NS(K), is introduced here.

When the R(J) is first allocated in the whole process of the RA/D scheme performed on SLC(K), the operand assigned to R(J) and its associated information is denoted by NSR(K,J) and expressed by:

$$NSR(K,J)= NSA(K,J), NST(K,J), NTY(K,J), NPT(K,J)$$

and NS(K) is defined as the set of NSR(K,J), J=1 to NR and expressed by $NS(K)= \bigcup_{J=1}^{NR} NSR(K,J)$. (For details see item 8 in section 4-2).

Algorithm 4-5

Program: FS of a Forward Branch SLC

Data: (Refer to Figure 4-3 and section 4-6)

1) I is the index of SLC(I).
2) J is the index of GPR, $1 \le J \le NR$.
3) FS(I), CS(I), and S(I) are the states associated with SLC(I). (see section 4-6)
4) "a null state" is defined in Algorithm 4-4.
5) FSR(I,J), CSR(I,J), SR(I,J) are defined in section 4-2 and section 4-5.

Pseudo code:

```
BEGIN
IF THE SLC FORWARD BRANCHES TO THE SAME BLOCK
(Determining the branch final state)
   THEN FS(I)=CS(I)  (FS is the same as the state before
                      the branch statement)
   ELSE BEGIN (branches to other block)
        FOR ALL J, 1≤J≤NR
        IF CTY(I,J)=TYPE 1 or TYPE 3
           THEN BEGIN
                FSA(I,J)=CSA(I,J)
                FST(I,J)=CST(I,J)
                FTY(I,J)=TYPE 3
                END
           ELSE FSR(I,J) IS SET TO BE A NULL STATE (Local
                variable only good within the current block)
        END
IF THE NEXT SLC IS IN THE SAME BLOCK (Determine the sequen-
   tial final state)
   THEN BEGIN
        FOR ALL J, 1≤J≤NR
        IF CTY(I,J)=TYPE 3
           THEN SR(I,J)=CSR(I,J)
           ELSE SR(I,J) IS SET TO BE A NULL STATE
        END

END
```

Some MEMREAD and MEMWRITE statements are needed in the generation of NS(K) from IS(K). This is simply described as follows:

Case a: if NSA(K,J)=ISA(K,J) then no MEMREAD/WRITE statement is needed.

Case b: if NSA(K,J) ≠ ISA(K,J), the possible conditions are:

IST(K,J)	NPT(K,J)	Condition
active	source	1
active	dest	2
passive	source	3
passive	dest	4

The statements that may be used are:

```
MEMWRITE  R(J)       ISA(K,J) -------------- (a)
MEMREAD   NSA(K,J)   R(J) ------------------ (b)
```

In condition 1, statements (a), and (b) are used.

In condition 2, statement (a) is used.

In condition 3, statement (b) is used.

In condition 4, none of the statements is used.

In the worst case, statements (a) and (b) are used to generate NSA(K,J) from ISA(K,J). If CS(I) uses the IS(K) as the reference state, these two statements cannot be moved out of the branch region. In the case of a loop, it will waste much time to execute these statements. If NS(K) is used as the reference state, these two statements do not need to be executed when the backward branch occurs.

However, if the statement (a) is still in the region, it will destroy the content of ISA(K,J) in the host machine memory. The conclusion is that if the MEMWRITE statement used to generate the NSA(K,J) from the ISA(K,J) can be moved out of the branch region, then NS(K) can be used as the reference state in the determination of FS(I). "A statement can be moved out of the region" means that this statement is data independent of all those statements ahead of it in the region. If we can prove that all the statements ahead of statement (a) do not contain R(J), ISA(K,J), this statement can be moved out of the region. The following paragraph will illustrate this point.

If NSA(K,J)=ISA(K,J), no MEMREAD or MEMWRITE is needed. Now, in the worst case of NSA(K,J)≠ISA(K,J), statements (a) and (b) are used. The basic idea of the RA/D scheme is that when it is performed on a variable which has been assigned to a register already, the same register is used by this variable. If ISA(K,J) has been used before it is deallocated, it must be the same as NSA(K,J). Our assumption, however, is that NSA(K,J)≠ ISA(K,J), so that ISA(K,J) in statement (a) is used for the first time in SLC(K). From the definition of NS(K), R(J) is first used when NSA(K,J) is assigned, R(J) and ISA(K,J) are both used for the first time in statement (a). It can be moved out of the region.

In statement (b), NSA(K,J) cannot be moved out unless the same variable is not in a different register in NS(K). This condition implies that each NSA(K,J) which appears in the mapping from ISR(K,J) to NSR(K,J) is used for the first time in SLC(K). In the case where this condition is not true, i.e., NSA(K,J)=NSA(K,J'), for J≠J', we have the following contradiction:

> From statement (a), NSA(K,J) is in R(J). After some calculations, NSA(K,J) has to be stored back in VML and another variable is allocated into R(J). The statement (c) is used if NST(K,J) is active.
>
> ```
>    MEMWRITE  R(J)  NSA(K,J) --------------------(c)
> ```
>
> and, then, for some reasons, NSA(K,J) is to be loaded again, and R(J') has the highest priority to be replaced. In the worst case,
>
> ```
>    MEMWRITE  R(J')     ISA(K,J') ---------------(d)
>    MEMREAD   NSA(J,J') R(J') -------------------(e)
> ```
>
> are used to generate NSA(K,J') in R(J'). Since NSA(K,J')=NSA(K,J), statement (c) blocks statement (e), but statement (d) can still be moved out.

This special example does not occur very often. If it does happen, the only result is inefficiency, not an error. NS(K) is used as the reference state by CS(I) to determine FS(I).

There is another special case where ISR(K,J)= CSR(I,J), but NSR(K,J) is empty. It will cause many unnecessary MEMREAD/WRITE statements if CS(I) uses NS(K) as the reference states. In this case, NSR(K,J) is set equal to ISR(K,J) before the determination of FS(I). Algorithm 4-6 is used to generate NS(K).

4-6-3 Final State of Backward Branch SLC

Refer to Figure 4-4. When the backward branch occurs, the state CS(I), which is right before the branch statement, must be set equal to the next initial state, NS(K), of the sink SLC. The state S(I) which is after the branch statement will go to SLC(I+1). The branch region between the branch statement and the sink may be a loop. Correct and efficient interface design is a major concern.

Algorithm 4-7 is used to solve the branch final state, FS(I). The sequential final state, S(I), is solved in Algorithm 4-8.

4-7 Conclusion

The outputs of pass 2 are a set of SLCs and a label reference table. Each SLC is a set of MOPs, which all operands are, in machine unit names. The timing phase is assigned, and all field values are determined except the next address value. The label reference table, which lists all labels and corresponding locations, is used to determine the next address value. The address field value assignment and the optimization process will be solved in the next chapter.

Algorithm 4-6

Program: NS of SLC(K)

Data:
1) NP(J) is set when R(J) is first allocated and will not be reset until entering the next SLC.
2) ODA(M,N) which is to be register allocated is an operand of a statement M in SLC(K).
3) Refer to Figure 4-4, SLC(K) is sink SLC and SLC(I) is a backward branch SLC.
SY1), SY(2), SY(3), and SY(4) are defined in Algorithm 4-2. Subroutine TYPE is defined in Algorithm 4-2.

Pseudo code:

```
 BEGIN
 IF THIS ALGORITHM IS CALLED FROM RA/D SCHEME
    THEN BEGIN
            ODA(M,N) IS SEPARATED INTO SY(1), SY(2), SY(3)
            AND SY(4)
            IF NP(J)=) (R(J) has not been allocated to
                       operand yet)
               THEN BEGIN
                    NSA(K,J)=SY(4), NP(J)=1
                    IF N=3 (ODA(M,N) is used as the destina-
                       tion)
                       THEN BEGIN (set the state variable of
                         R(J))
                            NST(K,J)=ACTIVE
                            NPT(K,J)=DEST
                            CALL SUBROUTINE TYPE TO SOLVE
                              NTY(K,J)
                            END
                       ELSE BEGIN (This operand is source)
                            NST(K,J)=PASSIVE
                            NPT(K,J)=SOURCE
                            CALL SUBROUTINE TYPE TO SOLVE
                               NTY(K,J)
                            END
            END
    ELSE RETURN (R(J) has been allocated to operand already)
    END
ELSE BEGIN
    IF CSR(I,J)=ISR(K,J) AND NSR(K,J) IS EMPTY
       THEN NSR(K,J)=ISR(K,J)
    END
 END
```

Algorithm 4-7

Program: Branch Final State of Backward Branch SLC.

Data: 1) Refer to Figure 4-4, SLC(I) branches SLC(K).
2) CS(I), FS(I), and NS(K) are defined in section 4-6

Pseudo code:

```
BEGIN
IF CSA(I,J)=NSA(K,J) (case 1)
   THEN BEGIN
        FSA(I,J)=CSA(I,J)
        FTY(I,J)=CTY(I,J)
        IF CST(I,J)=ACTIVE, AND NST(K,J)=PASSIVE
           THEN BEGIN (extra case 1)
                IF THERE IS NO DEALLOCATION PROCESS HAPPENS
                TO R(J) FROM SLC(K) TO SLC(I) (i.e. R(J)
                holds only this variable CAS(I,J) in this
                region)
                   THEN FST(I,J)=CST(I,J)
                   ELSE"MEMWRITE  R(J)   CSA(I,J)"
                       IS INSERTED AT THE END OF SLC(I)
                       FST(I,J)=PASSIVE
                END
        END
   ELSE BEGIN (case 2)
        IF CST(I,J)=PASSIVE, AND NPT(K,J)=DEST, OR
           CST(I,J)=PASSIVE, AND NPT(K,J)=EMPTY (R(J)did
           not hold variable in NSR(K,J)) (cond. a)
           THEN FSR(I,J)=CSR(I,J)
           ELSE BEGIN
                IF CST(I,J)=ACTIVE, AND NPT(K,J)=EMPTY
                   (R(J) did not hold variable in NSR(K,J))
                   THEN BEGIN (extra case 2)
                        IF R(J) HOLDS ONLY THE VARIABLE
                           CSA(I,J) FROM SLC(K) TO SLC(I)
                           (i.e. there is no deallocation
                           process which happens in this
                           region)
                        THEN FSR(I,J)=CSR(I,J)
                        ELSE BEGIN
                           "MEMWRITE  R(J)  CSA(I,J)"
                           IS INSERTED AT THE END OF SLC(I)
                           FSR(I,J)=CSR(I,J)
                           FST(I,J)=PASSIVE
                           END
                        END
                   END
```

Algorithm 4-7 continued

```
          ELSE BEGIN
             IF CST(I,J)=ACTIVE, AND NPT(K,J)=DEST (cond.b)
                THEN BEGIN
                   FSA(I,J)=CSA(I,J)
                   FTY(I,J)=CTY(I,J)
                   "MEMWRITE  R(J)  CSA(I,J)" IS
                   INSERTED AT THE END OF SLC(I)
                   FST(I,J)=PASSIVE
                   END
          ELSE BEGIN
                   FSA(I,J)=NSA(I,J)
                   FTY(I,J)=NTY(I,J)
                   FST(I,J)=PASSIVE
                   IF CST(I,J)=ACTIVE, AND
                      NPT(K,J)=SOURCE (cond. c)
                      THEN BEGIN
                           MEMWRITE  R(J)  CSA(I,J)
                           MEMREAD  NSA(K,J)  R(J)
                           ARE INSERTED AT THE END OF
                           SLC(I)
                           END
                      ELSE"MEMREAD  NSA(K,J)  R(J)"
                           IS INSERTED AT THE END OF
                           SLC(I) (cond.d)
                   END
               END
END.
```

Algorithm 4-8

Program: Sequential Final State of Backward Branch SLC.

Data: 1) Case 1 and condition a, b, c and d of case 2 are directly from Algorithm 4-7.
2) Refer to Figure 4-4, SLC(I) branches to SLC(K).
3) $\overline{\text{FTY}(I,J)}$, "set to be a null state," and VAR(L) are defined in Algorithm 4-4.
4) FS(I) has been determined in Algorithm 4-7 already.

Pseudo code:

```
BEGIN
REFER TO ALGORITHM 4-7
IF IT IS IN COND. C, D OF CASE 2 (it is described in
Algo. 4-7)
   THEN BEGIN
      IF SLC(I) AND SLC(K) ARE IN THE SAME BLOCK
         THEN BEGIN
              SR(I,J)=FSR(I,J)
              IF FSA(I,J) IS IN VAR(L)
                 THEN TY(J)=$\overline{\text{FTY}(I,J)}$
                 ELSE TY(J)=FTY(I,J)
              END
         ELSE BEGIN
              IF FTY(I,J)=TYPE 2 OR NONE (note:  FSA(I,J)
                 is a actually from NSA(K,J) in different
                 block)
                 THEN SR(I,J) IS SET TO BE A NULL STATE
                 ELSE BEGIN
                      SR(I,J)=FSR(I,J)
                      IF FSA(I,J) IS A GLOBAL VARIABLE OF
                         THE BLOCK WHICH CONTAINS THE
                         SLC(I) AND IT WILL BE USED BEHIND
                         SLC(I)
                         THEN TY(J)=TYPE 1
                         ELSE TY(J)=TYPE 3
                      END
              END
      END
   ELSE SR(I,J)=FSR(I,J) (case 1, and cond. a, b of case 2)
END.
```

CHAPTER V

PASS 3

5-1 Introduction

The inputs to pass 3 are a set of SLCs and a label reference table which are directly from the output of pass 2. Each SLC is a set of MOPs which is represented by M_i 5-tuples, $(OP_i, I_i, O_i, F_i, P_i)$, and is made machine dependent by specifying the architecture of a particular real microprogrammable machine. All field values in the field tuple F_i are defined except the address field which will be determined with the aid of the label reference table.

The purposes of this chapter are to develop techniques for combining sequences of M_iMOPs into shorter concurrent microinstructions, or what we abbreviate as MIs, and to move the redundant MOPs from the loop region.

We say the MI sequence is optimized if it is impossible to rearrange the sequence of M_iMOPs contained in the sequence of MI instructions, in a manner that will produce fewer microinstructions. DeWitt (7) has proved that this kind of absolute minimal reduction problem is an NP-complete problem. We find that the rules which are used to detect the parallelism of MOPs are dependent on the machine constraint. In this chapter, we show why the optimization problem is NP-complete and then derive general rules to detect the parallelism of MOPs and examine a special case of PDP11/40E machine to illustrate the machine dependency. Then, by seeking a near-optimal solution rather than the absolute optimum solution, we have been successful with a slower algorithm of complexity $O(mn)$, where m is a pragmatically determined constant less than n. While we have been unable to do so, it is noted that if we could apply a sort algorithm of complexity $O(n \log_2 n)$ to produce a near-optimal solution, then we could get a faster algorithm. This reduction would place the near optimal reduction problem in the class of sorting problems and yield extremely fast code optimization algorithms. The problem, then, is to produce the shortest possible sequence of microinstructions MI_1, MI_2,...MI_k from a compiler-generated sequence of microoperations, M_1, M_2,...M_n. The optimization algorithm which is used here to solve this problem is applied separately each SLC. The proposed algorithm runs in linear time to produce a reasonable approximation to the best possible code in most cases.

The general terminology used through this chapter is described in section 5-2. The general structure of pass 3 is illustrated in Algorithm 5-1 which leads to the following tasks: 1) Two important relationships among MOPs, invertibility and parallelism, are described in section 5-3 and section 5-4, respectively; 2) Based on this description,

Algorithm 5-1

Program: General Structure of Pass 3.

Data:
1) SLC(P) is to be compacted.
2) Forward branch is abbreviated as f.b.
   Backward branch is abbreviated as b.b.
3) The label name of the SLC is called LABEL, if any.
4) Subroutine OPTM is to describe the purpose of this pass, and is illustrated in Algorithm 5-2.

Pseudo code:

```
BEGIN
(START) FETCH NEXT SLC(P)
IF THERE ARE f.b. AND b.b TO SLC(P)
   THEN BEGIN
        TASK 1:  GENERATE A NEW LABEL NAME CALLED 'NEWLBL'
        TASK 2:  CALL SUBR OPTM TO COMPACT SLC(P)(see
                   Algorithm 5-2)
        TASK 3:  THE LABEL NAME 'LABEL' IS USED AS THE ENTRY
                 POINT OF SLC(P) FOR f.b. AND IS LOCATED ON
                 THE LABEL POSITION OF THE FIRST MOP OF THIS
                 SLC
        TASK 4:  THE NEW LABEL NAME 'NEWLBL' IS USED AS THE
                 ENTRY POINT FOR THE b.b. AND IS LOCATED IN
                 THE LABEL POSITION OF THE FIRST MOP RIGHT
                 AFTER THE MEMREAD/WRITE STATEMENTS
        TASK 5:  ANY b.b STATEMENT INVOLVED THE LABEL NAME
                 'LABEL' IS MODIFIED BY 'NEWLBL'
        GO TO START
        END
IF THERE IS ONLY A b.b. TO SLC(P)
   THEN BEGIN
        DO TASK 2
        DO TASK 4, BUT THE SENTENCE 'THE NEW LABEL NAME
        'NEWLBL' IS CHANGED BY 'THE LABEL NAME 'LABEL'
        GO TO START
        END
IF THERE IS ONLY A f.b. TO SLC(P)
   THEN DO TASK 2 AND TASK 3, GO TO START
PERFORM TASK 2, GO TO START
END.
```

the allocation problem of MOPs is illustrated in section 5-5.

5-2 General Terminology

The following terminologies assume a sequence of MOPs, M_1, M_2,...M_n are mapped into a sequence of microinstructions, MI_1, MI_2,... MI_k, $k \leq n$.

1) A SLC is the basic unit to be optimized and is represented by SLC=$\{M_1, M_2, \ldots M_n\}$, where M_i is a microoperation. We say M_i precedes M_j, denoted by $M_i < M_j$, if $i<j$.

2) We say a sequence of MOPs is executed in serial, denoted by $\{M_i\}, \{M_j\}, \{M_k\} \ldots$, if the MOPs are executed in separate control store cycles. Two MOPs, M_i and M_j, are executed concurrently, denoted by $\{M_i, M_j\}$, if they are executed in the same control store cycle.

3) A microinstruction MI is a set of concurrently executable MOPs denoted by MI= $\{M_i, M_j, \ldots M_k, \ldots\}$.

4) M_i and M_j are said to be parallel, denoted by $M_i /// M_j$, if for all inputs the sequential execution of $\{M_i\}, \{M_j\}$, results in the same output as the concurrent execution of microinstruction $MI_k = \{M_i, M_j\}$.

5) We say two MOPs, M_i, M_j in SLC and $M_i < M_j$ have

I/O conflicts if one MOP depends on the data produced by the other MOP or alters the data needed by the other MOP. Assume I_i, O_i, is in M_i and I_j, O_j is in M_j. If $I_i \cap O_j \neq 0$, $I_j \cap O_i \neq 0$, or $O_i \cap O_j \neq 0$, there is an I/O conflict between these two MOPs.

Now, we can pose the problem in more exact terms. Optimization of a sequence of MOPs in a loop-free SLC, is a conflict-free partition of the MOPs into sets, say MI_1, $MI_2, \ldots MI_k$, in such a way that no other partition results in fewer MIs; e.g., k cannot be reduced.

5-3 The Parallelism and Invertibility of MOPs

Based on the 5-tuple format of MOPs, two important relationships, parallelism and invertibility, are determined in this section. It will be easier to understand these relationships if we examine how the 5-tuple of a MOP affects:

1) I/O resources, 2) timing phase, and 3) field tuples.

5-3-1 I/O Resources

Consider two MOPs M_i, M_j, where M_i precedes M_j(denoted by $M_i < M_j$):

M_i : $\{OP_i, I_i, O_i\}$

M_j : $\{OP_j, I_j, O_j\}$

There are 4 cases in I/O intersection. (see

Table 5-1) In row 2, 3, or 4 of Table 5-1, there are two conditions for parallel execution (see the fourth column of Table 5-1). If the parallel action occurs above the dash line, it is different from the sequential action. Otherwise, the parallel action is the same as the sequential action.

The first nonempty intersection in Table 5-1 will not influence parallel execution, but the last three nonempty intersections do influence the parallel execution. Therefore, depending upon the values of A, B, or C in column 2 of Table 5-2, there are eight possible combinations. The only combination of interest, however, is the case where all intersections are empty. If A=B=C=0, then M_i, M_j are said to be data independent, denoted by $M_i \not\rho M_j$. This leads to a very important factor in the optimization problem. For example, consider the sequence of MOPs, N_1, M_2, M_3, with $M_1 \leq M_2 \leq M_3$ and additional properties that M_1 not $//M_2$, M_2 not $//M_3$but $M_1//M_3$. If we can change the position of M_2 and M_3 then we say M_2 and M_3 are invertible. We can invert two MOPs only when their execution is the same for both sequences. For example, sequential execution of M_1, M_2, M_3 produces the same result as the execution of M_1, M_3, M_2. We may take advantage of invertibility by combining M_1, M_3 into MI_1 leaving M_3 assigned to MI_2 to give an optimized partition for r=2. M_i, M_{i+1} are said to be invertible, denoted by $M_i >< M_{i+1}$, if $M_i \not\rho M_{i+1}$.

Table 5-1. I/O Intersection

Row	Nonempty intersection	Sequential action	Parallel action
1	$I_i \cap I_j$	Data sharing from common resource	Same as sequential
2	$A=I_i \cap O_j$	I_i transfers to O_i then O_j modified I_i	*If M_j is executed first, O_j reset I_i before I_i transfers to O_i Same as sequential
3	$B=O_i \cap I_j$	Data passes from M_i to M_j	*O_i has no chance to set I_j if M_j is executed first. Same as sequential
4	$C=O_i \cap O_j$	O_i is modified by O_j	*If M_j is executed first, O_j cannot modify O_i Same as sequential

- M_i, M are in sequential order, and M_i precedes M_j.
- M_i is denoted by $\langle OP_i, I_i, O_i \rangle$.
- M_j is denoted by $\langle OP_j, I_j, O_j \rangle$.

5-3-2 Timing Phase

An MI is considered to be a polyphase instruction in the designing of the FDM. The control store cycle is logically divided into several timing phases. (The detail is given in Chapter II.) The possibilities for timing intersections are discussed.

Assume $M_i \not< M_j$ and the time interval to initiate and execute M_j is T_j. The relationship between T_i, T_j is shown in Figure 5-1.

$T_i \cap T_j = 0$ implies $T_i < T_j$ or $T_i > T_j$

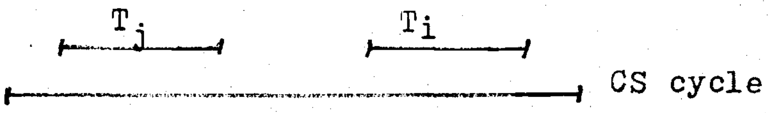
$T_i \cap T_j \neq 0$ implies $T_i = T_j, T_i \lesssim T_j$ or $T_i \gtrsim T_j$

We can see M_i precedes M_j in the sequential form, but in the parallel form M_i may not precede M_j. What we must do is to find an algorithm to detect whether the parallel execution of $MI_k = \{M_i, M_j\}$ can produce the same output as the sequential execution of $MI_k = M_i$, $MI_{k+1} = M_j$, for all inputs.

Consider Table 5-1 again. It is simple to determine the results of sequential execution, but parallel execution may or may not produce the same results as sequential action. If we add timing to the table and divide the fourth column in Table 5-1 into two parts, we get the results shown in Table 5-2. The entries of Table 5-2 show the conditions of timing which allow concurrency.

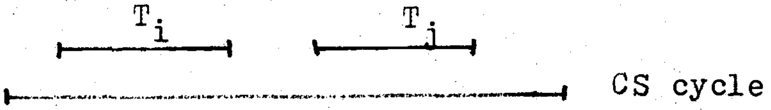
From the above discussion, it is obvious to see the I/O and the timing tuples play important roles in the

A) T_i T_j



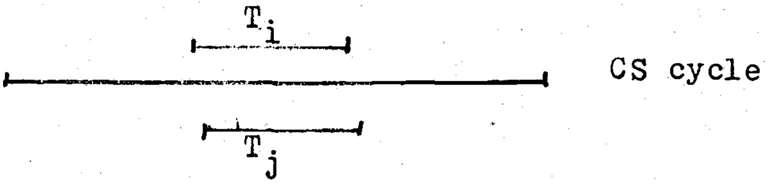
If both M_i and M_j can be executed in one CS cycle then M_j precedes M_i.

B) $T_i < T_j$



If M_i and M_j are executed in one CS cycle, then M_i still precedes M_j.

C) $T_i = T_j$



If $M_i // M_j$, then M_i and M_j execute in the same interval.

Figure 5-1. Timing Conflicts in a Polyphase Microinstruction

Table 5-2. $\langle I, O, T\rangle$ Conflict Detection

Nonempty intersection	Parallel and sequential execution leave same result	Parallel and sequential execution leave different result
$I_i \cap I_j$	independent of timing	
$I_i \cap O_j$	$T_i < T_j$	$T_i \geq T_j$
$O_i \cap I_j$	$T_i < T_j$	$T_i \geq T_j$
$O_i \cap O_j$	$T_i < T_j$	$T_i \geq T_j$

determination of the MOPs. Before going into the general rules to detect parallelism, a more exact explanation of field conflict is given.

5-3-3 Field Tuple

As mentioned in Chapter II, there are two kinds of fields in MI format, denoted by F_A, F_B, respectively.

$F_A = \{f_i /$ If f_i is used by more than one MOP in the same MI and the values assigned to these fields are the same, it will cause no conflict.$\}$

$F_B = \{f_i /$ If f_i is used by more than one MOP in the same MI, it will cause the conflict even if the field value is the same.$\}$

M_i, M_j are in SLC. F_i, F_j are the field tuple to Mi, Mj, respectively, and it is assumed:

$$F_i \cap F_j = F_k = \{f_i \mid \text{a set of fields}\} \neq 0$$

If $\forall f_i \in F_k \ni f_i \in F_A$ and the values of each f_i are the same, then $F_i \cap F_j$ is defined to be zero.

In other words, if one of $f_i \in F_k$ belongs to F_B, then

$$F_i \cap F_j \neq 0,$$

or if $\forall f_i \in F_k \ni f_i \in F_A$, but one pair of f_i has the different value, then

$$F_i \cap F_j \neq 0.$$

5-4 The Detection of Parallelism of MOPs

The machine constraints on the primitive operations may be different from computer to computer. The parallelism detection rule can never be machine independent. Here, we divide the discussion into two parts. One is statement of the general rules which are available to every machine. The second is an explanation of the machine constraints which must be faced. Then some examples are used to explain the machine limitations.

General rules

Every microinstruction is completed within a control store cycle. The method used to analyze the timing phase within the cycle is described in section 5-3-2. The following rules are used:

Given M_i, M_j in SLC and $M_i < M_j$. M_i and M_j are denoted by:

M_i: $\{OP_i, I_i, O_i, F_i, P_i\}$

M_j: $\{OP_j, I_j, O_j, F_j, P_j\}$

1) If $M_i \not\beta M_{i+i}$ then $M_i >< M_{i+1}$.

2) As $P_i < P_j$.

If $F_i \cap F_j=0$ then $M_i//M_j$.

3) As $P_i \geq P_j$

If $(F_i \cap F_j=0)$ and $(M_i \not\beta M_j)$ then $M_i//M_j$.

▶▶Example 5-2:

In the PDP11/40E machine (8,9), the CL3 cycle generates P2 and P3 pulses. Then each pulse is assigned to the corresponding MOP. There are three cases used to illustrate the general rules.

Case 1:	M1:	R2→D, P2	:	copy R2 to register D
	M2:	D→R3, P3	:	copy register D to R3
	M3:	R3+B→D, P2	:	add R3 and register B to register D
	M4:	D→R4, P3	:	copy register D to R4

M2 and M3 are examined to detect the parallelism.

From example 2-5, we know $F_2 \cap F_3=0$, but M_2 not β M_3. This implies M_2 not M . (If M_2 and M_3 are executed in one MI, and M_3 is executed prior to M_2, it will give a wrong result.)

```
Case 2:  M5:  emit →stack, P3 ; copy content value "emit"
                              ; to stack
         M6:  R3   D, P2      ; copy R3 to register D
```

From the third rule, ($F_5 \cap F_6=0$) and $M_5 \not\gamma M_6$) imply M5//M6.

```
Case 3:  M7:  R3+B →D, P2     ; add R3 and B to register D
         M8:  D →R3, P3       ; copy register D to R3
```

The pulses used by M_7 and M_8 are P2 and P3, respectively. $F_7 \cap F_8=0$ implies $M_7//M_8$ which is independent of I/O conflict. ◀ ◀

Machine Constraints

1) If more than one control store cycle is provided by the machine, this will cause some machine constraints on the general rules.

Example 5-2:

In the PDP11/40E machine (8,9), there are three machine cycles listed in Figure 5-2: a) CL1 cycle generates pulse P1; b) CL2 cycle generates pulse P2; and c) CL3 cycle generates pulse P2 and pulse P3.

The constraint is "Different microinstructions must use different control store cycles and MOPs in different cycles may not execute together." This implies that a MOP in CL1 can never execute together with MOPs in CL2. Before the general rule can be used, one has to determine that these two MOPs belong to the same control store cycle.

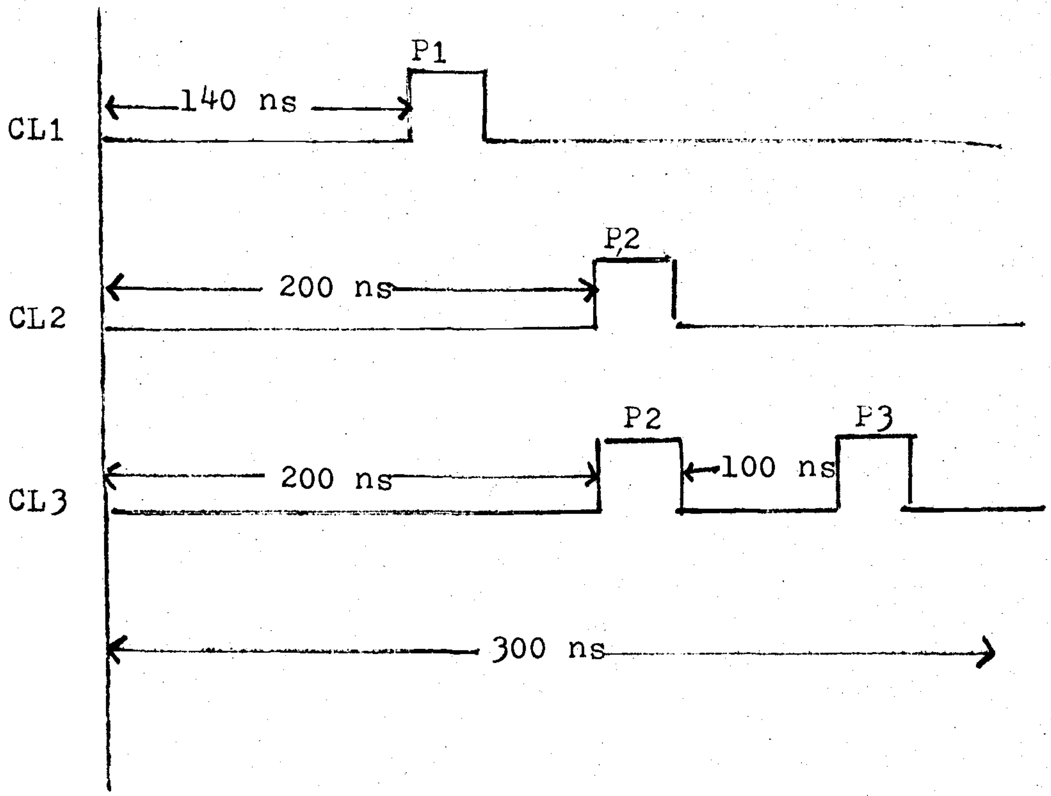


Figure 5-2. PDP11/40E Processor Clock

Case 4: M_9: PUSH, P1 ; push the stack

M_{10}: R3 →D, P2 ; copy R3 to register D

M_9 is in cycle CL1 and M_{10} is in cycle CL2 imply M_9 cannot be parallel with M_{10} even if the general rule is good in this case.

Examine M_5 and M_6 in example 5-1. They both belong to cycle CL3. The general rule is applied to get the parallelism result.◄◄

2) There are some MOPs used for special purposes such that the general rules cannot apply to them.

►►Example 5-3:

In the FDM of the PDP11/40E, the MOP FLAG is used to set the best machine flags for the previous ALU operation. MOP FLAG must be the next one after the ALU operation. It cannot move the position even if invertibility is true.

MOP NOOP, which is used in the N-way branch operation and provides the branch address, has its own fixed position. It cannot be moved and/or parallel with other MOPs even if the general rule is applied here.◄◄

The MOPs used for these special purpose and the extra machine constraint conditions cannot make the parallelism detection rules completely machine independent. In order to keep the system portable, they are packed into a subroutine. If the rules are changed for another machine, this subroutine must be rebuilt.

5-5 MOP Allocation and Movement

The purpose of this section is to develop algorithms used to allocate the MOPs into the MI and move the MEMREAD and/or MEMWRITE statements which are used to generate NS(K) from IS(K) in the sink SLC out of the backward branch region.

5-5-1 Theoretical Constraints on Optimization

The optimization problem is known to be NP-complete (7). Thus, it is not likely that there exists a nonexponential algorithm to solve this kind of problem with a deterministic Turing Machine. First of all, we examine why the optimization problem is NP-complete.

The definition of parallelism and invertibility of a pair of MOPs was described previously. Now, we extend the definitions to microinstruction.

MOP M_k is said to be parallel with MI, if $M_j \forall MI_k \exists$ $M_k//M_j$. Also MOP M_k is said to be invertible with MI, if $\forall M_j \in MI \exists M_k >< M_j$.

Given a SLC= $\{M_1, M_2 \ldots M_k \ldots M_n\}$, assume $\{M_1, M_2 \ldots M_{k-1}\}$ is partitioned into $MI_i \ldots MI_i$. As we allocated M_k, relationship between MOP and MI is : (refer to Table 5-3)

Case 1: M_k not $><MI_i$, and M_k not$//MI_i$

Case 2: M_k not$><MI_i$, and $M_k//MI_i$

Case 3: $M_k >< MI_i$, and M_k not//MI_i

Case 4: $M_k >< MI_i$, and $M_k//MI_i$

Table 5-3. Possible Positions of MOPs in the Allocation Problem

Possible position case number	MI_{i+1}	MI_i	$MI_i \ldots MI_{i-1}$
Case 1	X		
Case 2	X	X	
Case 3	X		Δ
Case 4	X	X	Δ

X: MOP can be in this position.

Δ: Check M_k with the MI ahead of the current one and determine which case it belongs to.

If M_k is invertible with MI_i (Case 3 or 4 of Table 5-3), it may be moved past MI_i and the same test applied to MI_{i-1}. On the other hand, if M_k is not invertible with MI_i (Case 1 or 2), it is blocked by this MI. In this case, M_k is placed in the subsequent MI_{i+1} or the current MI_i, respectively.

In Case 3 and 4 of Table 5-3, we have to check the MOP ahead of the current MI_i. Again we face four cases. If M_k is invertible with all MIs from MI_i back to MI_1, there are (i+1) possible positions for M_k; one position is ahead of MI_1, one is after MI_i. The other i-1 positions are between any pair of successive MIs. In the remaining cases, if M_k is // and invertible with all MIs, there are 2i+1) possible positions for M_k.

Let us consider the worst case:

$S= \{M_1 \ldots M_n\}$, assume every MOP is invertible with every other, but not parallel. M_1 is allocated in MI_1, M_j is to be determined, $2 \leq j \leq n$.

$j=2$, there are 2! possible positions for M_2, $\{M_1\}$, $\{M_2\}$, or $\{M_2\}\{M_1\}$.

$j=3$, there are 3! possible positions for M_3.

.

.

.

$j=n$, there are n! possible positions for M_n.

Totally, there are $\sum_1^n k!$ possible positions in which to allocate these n MOPs.

Clearly, this is a very special case, since if we know in advance that there is no parallelism among MOPs, it is not necessary to check these positions. We just use n MIs to allocate the n MOPs. The problem is that all the relationships are not known until we check the last MOP in SLC. The allocation of MOPs depends not only on the MOPs ahead of it, but on the MOPs after it. The best position of MOPs cannot be decided until every possible combination of MOPs is checked. We can see that <u>invertibility</u> causes the problem to be NP-complete.

On the other hand, the data dependency among MOPs is obvious and limits the invertibility considerably. In this case, it is hard for a MOP to cross too many MOPs ahead of it. A limitation of the times of comparing a MOP with other MOPs is necessary.

5-5-2 Linear Order Compaction Algorithm

In order to get a practical and efficient algorithm, we impose the following restrictions.

1) The position of MOP M_k is computed by searching backward over the previous microinstructions leading up to MOP M_k.

2) In each case of Table 5-3, we make the following decision.

Case 1: $\{M_k\} \rightarrow MI_{i+1}$

Case 2: $\{M_k\} \rightarrow MI_i$

In the next two cases, M_k is limited to make m comparisons with the previous MOPs. In other words, M_k can compare with h MIs from MI_{i-1} to MI_{i-h} where h is a number of MIs and $\sum_{j=1}^{h} |MI_{i-j}|$ is nearest to m. ($|MI_k|$ means number of MOPs in MI_k).

Case 3: If M_k is invertible with all MIs but not parallel, then $\{M_k\} \rightarrow MI_{i+1}$.

Case 4: Compare M_k with MI_{i-j}, $1 \leq j \leq h$, until we find the MI nearest to MI_1 that can accept M_k.

We restrict the invertibility problem as described above and use the relationship of // and >< between MOP and MI to get Algorithm 5-2. But, there is a special case in which this limitation cannot be put on the algorithm. As mentioned in Chapter IV, a SLC(I) backwards branches to SLC(K). The MEMWRITE statements which are used to generate the NS(K) from IS(K) will have to be moved out of the branch region. Otherwise, errors will occur. Algorithm 5-3 which is a subroutine to Algorithm 5-2 is used to move these statements out of the branch region.

Now, we consider the computational complexity of this algorithm, using the number of comparisons between pairs of MOPs as a measure of this complexity. There are n MOPs in SLC $\{M_1, M_2 \ldots M_k \ldots M_n\}$. Assume MOP M_k is to be determined for $2 \leq k \leq n$ and $M_1, M_2 \ldots M_{k-1}$ is partitioned into $MI_1, MI_2, \ldots MI_{j-1}$ already.

1) In case 1 and 2 of Table 5-3, M_k is assigned to MI_{j+1} of MI_j. In the worst case, we compare only M_k with all the MOPs in MI_j.
2) In case 3 of Table 5-3, as $k > m$, we check M_k with MI_{j-i}, i+1, 2,...h until >< does not exist. In the worst case, M_k is invertible with h MIs ahead of it. We need m comparisons before we get the position of M_k. As $k \leq m$, at most k comparisons are necessary.

Algorithm 5-2

Program: O(mn) Compaction Algorithm

Data: 1) SLC(P), M_1, M_1,...M_k...M_n is to be processed.

2) When M_k is allocating into MI, we assume M_1...M_{k-1} has been allocated to MI_1...MI_j already.

3) n is the number of MOPs in SLC(P).

4) m is the maximum number of comparisons which is allowed by the algorithm when a MOP is allocating to MI.

5) k is the current MOP index.

6) j is the current MI index.

7) S is the counter to count the number of comparisons when M_k is allocating.

8) /MI/ is the number of MOPs in MI.

9) ><(invertibility) and // (parallelism) are determined from section 5-3 and section 5-4.

Pseudo code:

```
BEGIN
(STRT) SET THE COMPARISON COUNTER S TO ZERO
FETCH NEXT MOP, $M_k$
IF ALL MOPs IN SLC(P) ARE ALLOCATED ALREADY INTO MIs THEN
   RETURN
   ELSE BEGIN
        IF THERE IS A b.b. TO SLC(P)
           THEN BEGIN
                IF $M_k$ IS A MEMREAD/WRITE STATEMENT
                   THEN CALL ALGORITHM 5-3
                        GO TO STRT
                   ELSE GO TO A
                END
           ELSE BEGIN
(A)             $S=S+|MI_j|$
                IF $M_k//MI_j$
                   THEN BEGIN
                        IF $M_k >< MI_j$
                   THEN kk=j (kk is set to the current MI
                              index)
                        GO TO C
                   ELSE ALLOCATED $M_k$ INTO $MI_j$,
                        GO TO STRT
           END
```

Algorithm 5-2 continued)

```
        ELSE BEGIN
             IF $M_k >< MI_j$
                THEN BEGIN
(C)                  IF S>m (The number of comparisons
                             exceeds the limitation)
                        THEN GO TO B
                        ELSE BEGIN
                             j=j-1 (decrement the current
                                    MI index)
                             IF j=0
                               THEN GO TO B
                               ELSE GO TO A
                             END
                     END
                ELSE BEGIN
(B)                  IF kk=0 ($M_k$ has never been parallel
                              with any $MI_{kk}$, where $kk \leq j$)
                        THEN BEGIN
                             ALLOCATE $M_k$ into $MI_{j+1}$
                             j=j+1 (set the new MI index)
                             GO TO STRT
                             END
                        ELSE ALLOCATE $M_k$ INTO $MI_{kk}$ ,
                             GO TO STRT
                     END
               END
           END
       END
END.
```

Algorithm 5-3

Program Movement

Data: 1) This algorithm is called from Algo. 5-2.
2) Label MOP is the MOP contained the label statement.
3) M_k is the MEMREAD/WRITE MOP to be moved out of the branch region.

Pseudo code:

```
BEGIN
CHECK $M_k$ WITH $MI_j$, $MI_{j-1}$,...$MI_q$($MI_q$ is the MI contained
                                      the label MOP)
IF $M_k$ IS INVERTIBLE WITH ALL THESE MIs
   THEN BEGIN
        T=j
        WHILE $t \geq q$ (change the index of MI)
        DO BEGIN
           $MI_{t+1} \leftarrow MI_t$
           $t \leftarrow t-1$
           END
        END
        ALLOCATE $M_k$ INTO $MI_q$ ($M_k$ is moved out the branch
                                  region)
        GO TO D
        END
   ELSE ALLOCATE $M_k$ INTO $MI_{j+1}$($M_k$ may not be used to gener-
                                  ate NS(P) from IS(P))
 (D)$j \leftarrow j+1$ (set new MI index)
        RETURN
 END.
```

3) In case 4 of Table 5-3, as $k > m$, we check M_k with MI_{j-i}, $i=1,2,..h$ until //, ><, or neither exist. Then we assign M_k to MI_{j-i} where i is as large as possible. The worst case occurs when M_k is // and with h MIs preceding it; i.e., we need m comparisons before the allocation of M_k. As $k \leq m$, at most k comparisons are necessary.

These four cases may occur alternatively but in the worst case, as $k > m$, M_k requires a total of m comparisons before allocation. Indeed, if this occurs for each of MOPs, $M_{m+1}, \ldots M_n$, the total number of comparisons is $T(n)=1+2+ \ldots m+(n-m)m$. Therefore, the algorithm complexity is $O(mn)$.

This algorithm fails to produce the absolute optimization code, but runs in linear time $O(mn)$. The value of m will be determined pragmatically in the next chapter.

CHAPTER VI

EXAMPLE AND CONCLUSION

6-1 Example

This chapter discusses an example used to describe the entire performance of the translating system. The general structure of this example is shown in Figure 6-1. The target machine, PDP8, is realized by IML in two parts. One is described by IISG (Appendix E-1); the other is described by IESG (Appendix E-2). The host machine used is the PDP11/40E. The FDM and the MET of the host are described in Appendix B and Appendix C, respectively.

IISG is decoded into OP(IISG) which, in turn, together with IESG and MET are the inputs to pass 1. The output of pass 1, Appendix E-3, is a set of host machine executable codes partly in the form of symbolic variables. These codes together with the FDM are the input to pass 2. The output of pass 2, Appendix E-4, is a set of MOPs and each MOP is in a 5-tuple representation. The output of pass 3, Appendix E-5, is a set of compacted codes and the host binary microcode associated with each MOP. Finally, three different benchmarks of PDP8 are tested and the result is shown in Appendix E-6.

This example shows that the system successfully translates the IML into the PDP11/40E microcode. The performance of each pass is evaluated in section 6-2 to show

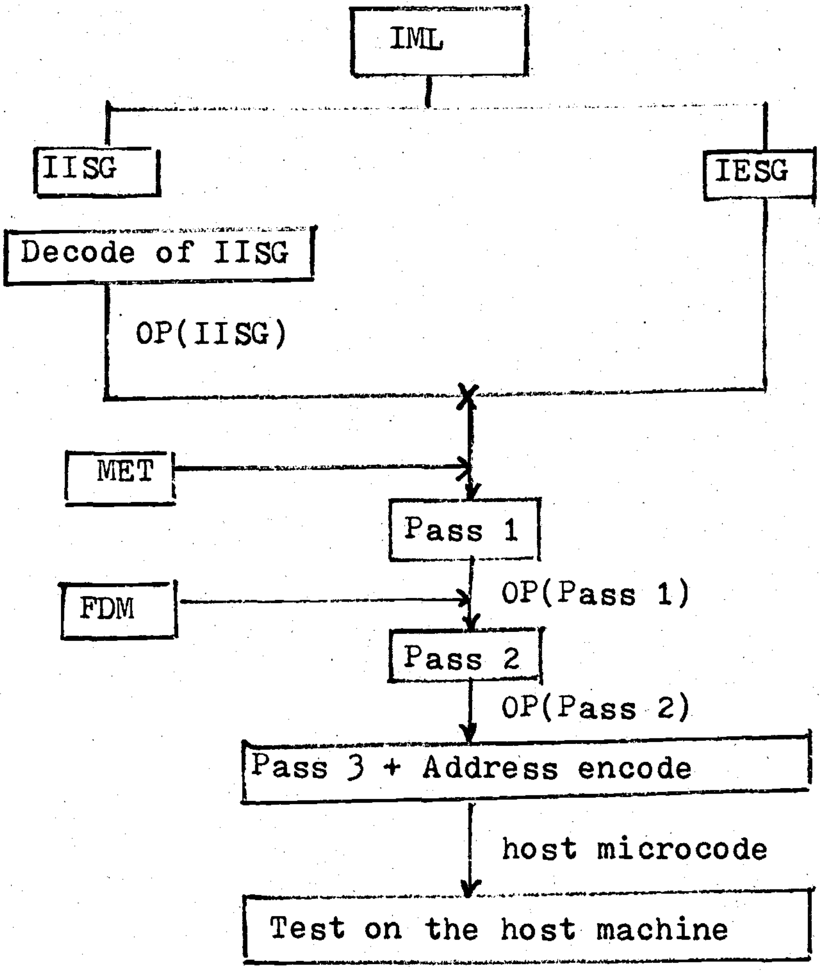


Figure 6-1. General Structure of Example 6-1

the efficiency of the system. In this translator, there are some limitations from the host machine constraint and part of the system have not yet been programmed. These factors will be described in section 6-3.

6-2 Performance Evaluation of Passes

6-2-1 Pass 1

Pass 1 increases the number of IML codes, M, to the number of MDIL codes, N. This increase number, N-M, which is used to solve the problems of the difference between the virtual machine and the host machine, is highly dependent on the choice of the host machine. Since extra machine codes (MDIL) are needed to match the difference between the host machine and the virtual machine, for instance, in the example 3-7 of Chapter 3, the word size problem causes eleven machine codes to describe that IML code which needs only three machine codes if there is no word size difference.

In the whole translation system, (refer to Figure 6-2), pass 2 is used to allocate the register to the variable in MDIL and the output is K MOPs. The increase in number, K-N, is needed to handle the load and/or store operations (the details are in Chapter IV). Pass 3 compacts these K MOPs into J MIs, where $J \leq K$ (the details are in Chapter V). Pass 1 is one of the factors that influences the system's efficiency (with respect to the

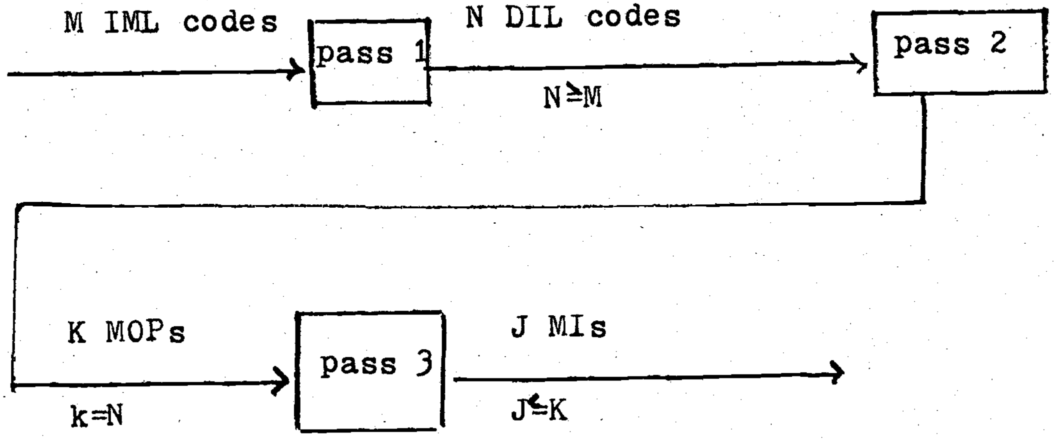


Figure 6-2. The Variation of the Number of Codes in the Whole System

number of codes increase). In order to minimize the value N-M, the user may often use the "equivalent" machine to emulate the target machine. For example:

1) The operations of the host machine are similar to the IML statements.

2) The hardware configuration of the host machine can describe the corresponding configuration in the target.

3) The arithmetic mode and the word size are the same for the host and the target.

6-2-2 Pass 2

The main purpose of pass 2 is to allocate the symbolic variables declared in the VMPL emulator program into the set of GPRs of the host machine. As mentioned before, pass 2 causes extra load/store operations which directly influence the system's efficiency. The performance of pass 2 with respect to the number of GPRs is to be evaluated. Some related work is discussed first.

Rannem, et al. (17) described an experiment performed for 15 small computers as follows:

1) Gather normalized execution times and memory space requirements for three simple benchmark kernels written in the macro assembly level of each computer.
2) Choose two different kinds of equations that have six standard machine parameters as the independent variables and execution time (T) and memory space (S) as the dependent variables.
3) Perform a standard regression fit of these equations to the observed data for time and space to estimate the equation coefficients.
4) Finally, for each kernel, there are two performance measures, S, and T, which are the functions of the six machine parameters.

Among these six performance equations, he found that the execution time of Kernel 3 is significantly dependent on the number of GPRs, and concluded that substantial changes in performance are not achieved by increasing the number of registers beyond 6 or 8.

Lunde, et al. (11) used the DEC-10 ISP (instruction set processor) to analyze 36 test programs written in high level languages from a scientific environment and 5 compilers, three of which were written in macro assembly language and the rest in a HLL. Lunde's analysis program was used to detect register lives, classify them and find the number of "live registers" at each time during program execution. The results suggest that programs might run almost equally time-efficiently on an ISP having fewer registers, but the same structure otherwise.

Reducing the number of GPRs in ISP will increase the execution time because of redundant register store and reload operations. The result shows that the average increase caused by a reduction to 8 registers is 7.9% and the authors conclude that eight registers would be sufficient for a general register ISP similar to the DEC system 10.

The example in section 6-1 shows that the input of pass 2 consists of 174 microoperations in 32 SLCs containing 7 global variables, 3 local variables and 13 local temporary variables. The host machine used is the PDP11/40E. An experiment is made by varying the number of different registers and measuring the length of code produced. (See the result in Table 6-1). As is seen, when the number of registers, N, is greater than or equal to 9, there is little change or increase in instruction count. If we reduce the value of N, it will increase the instruction count. For example, as N is reduced to 8, the

Table 6-1. Evaluation of Pass 2 (number of registers w.r.t. the length of code produced)

n	OP_n	$f=OP_n-OP_9$	f/OP_9	$f_2=OP_n-IP$	f_2/IP
3	267	76	39.8	93	53.4%
4	262	71	37.1%	88	50.6%
5	248	57	29.8%	74	42.5%
6	219	28	14.6%	45	25.8%
7	215	24	7.3%	41	23.6%
8	203	12	6.3%	29	16.7%
9	191	0	0	17	8.9%
10	191	0	0	17	8.9%

- The number of input codes in 174 in 32 LSCs.
- The number of variables is 23.
- n is the number of registers.
- OP_n is the number of output codes when the number of register is n.

increase in relative instruction count is 6.3% which is close to Lunde's result. It seems that eight or nine registers would be a good size for general purpose emulation.

The other feature of pass 2 is seen in the last column of Table 6-1. The inefficiency rate (IR) is defined as:

$$IR=(\# \text{ of } OP_n - \# \text{ of } IP)/(\# \text{ of } IP)$$

As shown, as the value of N decreases, the value of IR

Table 6-2. Testing O(m) Algorithm on the Husson's Machine

m	L	# of MIs	m	L	# of MIs
**	2	30	**	2	5
2	2	31	2	2	6
**	3	21	**	3	4
3	3	23	3	3	4
**	4	18	**	4	3
4	4	18	4	4	3
**	5	18	**	5	3
5	5	18	5	5	3
**	**	18	**	**	3
(56 MOPs in SLC)			(9 MOPS in SLC)		

- **: no limitation on this constraint.
- m : the number of comparisons.
- L : the length of MOPs in MI.

increases. When N is reduced from 9 to 3, the value of IR is increased from 9% to 53%. We conclude that pass 2 works well: i.e., it can produce up to 44% savings. The limitations are due to the host machine, not the algorithm.

6-2-3 Pass 3

Pass 3 uses a pragmatic rule to detect the concurrency of MOPs and an O(mn) algorithm to allocate the MOPs into the MIs. (Note: MOP is defined directly from the FDM). Where n is the total number of MOPs to be processed, m is the maximum number of comparisons allowed in the algorithm. The evaluation of pass 3 performance is used to answer such questions as: What width of the MI would be sufficient if a machine is designed? What is the best value of m in the O(mn) algorithm?

Two test examples, one containing 9 MOPs in a SLC, the other containing 56 MOPs in a SLC, are encoded on the Husson machine (10). The number of comparisons, m, and the limitation of the number of MOPs in one MI, L, are considered as the dependent variable in pass 3.

Different values of m and L are tested and the results are displayed in Table 6-2. As is seen, there is no change in the number of MIs when the value of L is greater or equal to 4 and the average concurrent MOPs in one MI is 3. It seems that four MOPs is the limiting width of a MI for a microprogrammable machine. Beyond this number, data dependency among MOPs limits the compaction of MOPs into MIs.

Next, the value of m is to be determined. Review Table 6-2 again. If the value of m is set equal to the value of L, the number of compacted output MIs is very

Table 6-3. Testing O(mn) Algorithm on the PDP11/40 Machine

# of m	# of MOPs reduced	# of OP
3	38	153
4	38	153
5	38	153
6	38	153

. The number of IPs is 191.

. The width of MI is 2.

. m is the number of comparisons.

close to the number of optimized MIs when the value of m is not limited. We conclude that the "best" peephole size of m is twice the width of the MI.

Now, the example in section 6-1 is examined. The width of the MI which is determined from the FDM is two. We checked all 41 MOPs in the FDM and found that, at the most, two MOPs can be combined in the legal condition.

Different values of m are tested in pass 3, as is shown in Table 6-3. There is no change as the value of m is greater than 4 (which is twice the MI width). The average number of concurrent MOPs in one MI is 1.24. Compare this value with the previous examples. It is significantly decreased. The reason for the decrease is that concurrency detection among MOPs is highly machine dependent. The last example is actually run on the

PDP11/40E, and the previous examples are based on Husson's abstract machine.

Pass 3 can produce 20% savings in the instruction count. Thus, this algorithm does better than the machine can support.

6-3 Conclusions

A translating system has been developed in this research to meet the goals set up in chapter one and run correctly on PDP11/40E. Some important features of this system are:

1) The FDM successfully plays the role of general model for all host machine information.

2) The RA/D scheme handles the control flow interface problems and produces as great a savings as host machine constraints will permit in practice, e.g. the number of GPRs used in the machine limit machine performance.

3) The optimization (Compaction) algorithm can save up to 20% instruction count but is limited by the real machine, rather than the theoretical NP-complete bound.

From the performance evaluation, we have:

1) The width of a MI should not exceed 4. Beyond this value, data dependency will limit the compaction of MOPs.

2) The number of comparisons, m, in O(mn) is twice the MI width. (Compare O(mn) and $O(n^2)$, as n is larger). Thus $m \doteq 8$.

3) The number of GPRs used in the machine is 8 to 10. Beyond this value, there will not be any significant change in the instruction count.

There are some limitations to this translation system, from the host machine constraint. If another host is used, the subroutines containing these limitations will be changed. Further, part of the system has not yet been programmed. The unfinished tasks and host limitations are described as follows:

in pass 1:

1) There are some statements in IESG of IML that have not yet been programmatically decoded; for instance, the statements LOOP, MPY, and DIV.

2) To each simple IML code, there is a corresponding set of machine codes in the Macro Expansion Table (MET). Each machine code is taken directly from the Field Description Model (FDM). These FDM and MET are host machine dependent and provided by the user.

In pass 2:

1) The size of GPR and the algorithm used to compute the field value are machine dependent.

2) Algorithm 4-7 is to determine FS(I) when SLC(I)

backward branches to SLC(K). There are two parts in this algorithm, denoted by extra case 1 and extra case 2, which have not been programmed.

In pass 3:

1) From Chapter V, the MOPs used for the special purposes and some machine constraints can never make the parallelism detection rule of MOPs machine independent. This rule will be designed by the user when the other host is used.

2) The next microaddress determination is dependent on the host machine.

3) The loader used to load the VM benchmark into the host machine memory is machine dependent.

BIBLIOGRAPHY

1. Agerwala, T. "Microprogram Optimization: A Survey," IEEE Trans. Comput., Vol. C-25, Oct. 1976, pp. 962-973.

2. Agrawala, A. K., and T. G. Rauscher. Foundations of Microprogramming Architecture, Software, and Applications, Academic Press, Inc., 1976.

3. Dasgupta, S. Parallelism in Microprogramming System, Ph.D. Thesis, University of Alberta, Aus. 1976, Tech. Rept., Dept. of Computing Sci.

4. Dasgupta, S., and J. Tartar. "The Identification of Maximal Parallelism in Straight Line Microprograms," IEEE Trans. Comput., Vol. C-25, Oct. 1976, pp. 986-991.

5. Davidson, S., and B. D. Shriver. "An Overview of Firmware Engineering," Computer, Vol. 11, No. 5, May 1978, pp. 21-33.

6. DeWitt, D. J. "A Control Word Model for Detecting Conflicts Between Microprograms," Proc. 8th Annu. Workshop on Microprogramming, pp. 6-13.

7. DeWitt, D. J. "A Machine Independent Approach to the Production of Optimal Horizontal Microcode," Ph.D. Dissertation, The University of Michigan, 1976.

8. Fuller, S. H., et al. PDP11/40E Microprogramming Reference Manual, Dept. Computer Sci., Carnegie-Mellon Univ., Jan. 1976.

9. Fuller, S. H., et al. The PDP11/40E Maintenance Manual, Dept. Computer Sci., Carnegie-Mellon Univ., June, 1977.

10. Husson, S. S. Microprogramming: Principles and Practice, Prentice Hall, Englewood Cliffs, New Jersey, 1970.

11. Lunde, A., G. Bell, D. Sieworek, and S. H. Fuller, "Empirical Evaluation of Some Feature of Instruction Set Processor Architecture," Comm. ACM 20, 3(March, 1977), 143-155.

12. Malik, K. Optimizing the Design of a High Level Language for Microprogramming, Unpublished Ph.D. Dissertation, Oregon State University.

13. Malik, K., and T. G. Lewis. "Description of IML," Dept. of Computer Sci., Oregon State Univ. (unpublished paper).

14. Malik, K., and T. G. Lewis. "High Level Microprogramming Language," COMPCON, 1978, pp. 88-91.

15. Mallet, P. W., and T. G. Lewis. "Considerations for Implementing a High Level Microprogramming Language Translation System," Computer, Vol. 8, No. 8, Aug. 1975, pp. 40-52.

16. Microdata 3200 Microprogramming Manual (preliminary), Revision 2, June 21, 1973.

17. Rannem, S., V. Hamacher, and S. Zaky. "Relating Small Computer Performance to Design Parameters," Infotech International, 1977, pp. 250-270.

18. Shriver, B. D. "A Description of the MATHILDA System," Dept. of Computer Sci. Report, Univ. of Arhus, Arhus, Denmark, April, 1973.

19. Tabendeh, M., and C. V. Ramamoorthy. "Execution Time (and Memory) Optimization in Microprograms," Preprints Supplement, 7th Annu. Workshop on Microprogramming, pp. S19-S27.

20. Tsuchiya, M., and C. V. Ramamoorthy. "A High Level Language for Horizontal Microprogramming," IEEE Trans. Comput., Vol. C-23, Aug. 1974, pp. 791-802.

21. Tsuchiya, M., and M. J. Gonzalez. "An Approach to Optimization of Horizontal Microprograms," Proceedings of the Seventh Workshop on Microprogramming, Palo Alto, California, Sept. 1974.

22. Yau, S. S., A. C. Schowe, and M. Tsuchiya. "On Storage Optimization of Horizontal Microprograms," Preprints, 7th Annu. Workshop on Microprogramming, pp. 98-106.

APPENDIX

APPENDIX A

MACHINE INDEPENDENT

INTERMEDIATE LANGUAGE

A program written in VMPL gets translated by the META-VMPL compiler into an abstract intermediate language (IML). The various statements of the intermediate language are discussed here. In discussing the intermediate language, reference to VMPL statements has been made, since IML is highly dependent on VMPL.

INTRODUCTION

Basically there are two kinds of statements in IML. One group is associated with the declaration statements of VMPL and is known as the intermediate information statement group (IISG). The other group is associated with the actual executable statements of VMPL and is known as the intermediate executable statement group (IESG). I will now discuss both these groups in detail.

IISG

An IISG statement is made up of five objects. The basic format of the statement:

DECLARATIONTAG IDENTIFIER, DIMENSION, LENGTH, OTHER-INFORMATION where

A uniform numbering system for the tags has been adopted. Assuming the tag is of the form $\alpha \beta \gamma$ then.

α		
	0	None of the others
	1	LOCAL
	2	GLOBAL
	3	Internal procedure (IPROC)
	4	Sub-procedure (SPROC)

β-		
	0	None of the others
	1	TEMPORARY
	2	PERMANENT

γ-		
	0	SIMPLE
	1	MEMORY
	2	STACK
	3	PSTACK
	4	FLAG
	5	FIELD
	6	USE
	7	EXPECT
	8	RETURN
	9	EXTERNAL

A	Name of emulator
B	Program start
C	Program end
D	WORDSIZE
F	ARITHMETIC
G	Sub-procedure name
H	Block code start
I	Block code end
J	
	Unused (presently)

Examples:

00D	Wordsize
221	Global permanent memory
214	Global temporary flag
00H	Block code starts

OTHERINFORMATION

This is only associated with a few tags. Since its format for each of them varies, so they will be discussed individually.

a) 005 - N_1, N_2, N_3

The tag indicates that this is a field declaration. N_1, N_2 and N_3 are integer numbers and are the three numbers associated with the FIELD declaration of VMPL.

b) 2(2/1)3 - S_1, S_2, S_3, S_4

The tag indicates that this is a stack pointer (PSTACK) declaration and the other information i.e.

S_1, S_2, S_3, S_4 indicates the push-pop sequence associated with the stack. S_1, S_2, S_3 and S_4 are all distinct symbols and can be ↑, ↓, +, - .

c) 2/2/1)9 - α

The tag indicates an EXTERNAL variable. α can be a 'p' indicating an external procedure or it can be an 'F' indicating it is an external flag.

d) 2(2/1) - β

The tag indicates a global flag declaration. β can be

0 - None of the others, a general flag

1 - Indicates special flag C - carry.

2 - Special flag O - overflow

3 - Special flag N - negative

4 - Special flag Z - zero

IESG

The IESG statements are based on quadruples with an operation and three operands. All three operands are optional in that some statements have none, some one, some two and some all three operands. First the overall format is discussed and then the individual statements are discussed.

FORMATS

A label starts in column 1 and always exists by itself in a line. A star (*) in the first column indicates a continuation of the previous statement. It is only used for translating two types of VMPL statements. If the line with the star is empty it indicates the end of the continuation. All other statements start in column 7 or 8. The various column designations are:

8-14	Operation
17-23	Operand one
26-32	Operand two
35-41	Operand three
42-46	Flag settings
7	Operation modifiers
16,25,34	Operand modifiers

OPERATION MODIFIERS

The two operation modifiers are:

% - indicates that the arithmetic operation is to be done according to the mode (1's or 2's) declared in the ARITHMETIC declare statement (tag - OOE).

↑ - indicates that the flags (host) are to be set and will be used by the following statement.

OPERAND MODIFIERS

The operand modifiers are:

. - indicates the operand is a bit operand. The format of the operand is:

ID, NUMBER

where NUMBER refers to the bit of ID in question.

/ - indicates concatenated operand. The format of the operand is:

ID_1, ID_2

where ID_1 and ID_2 are identifier names.

+ - indicates the temporary (operand) is needed.
- - indicates the temporary is not needed.
C - indicates a (constant) integer is the operand.
P - indicates the operand is a parameter identifier.
T - label for first branch in IF-THEN-ELSE statement.
E - label for second branch in IF-THEN-ELSE statement.
G - label for a GOTO statement.
F - label for a FOR statement.
L - label for a LEAVE statement.

STATEMENTS

There are seven classes of statements. Each class is treated separately.

1 - This class has as its OPERATION either an arithmetic or a logical operation. The general form:

```
OPERATION SRC1 SRC2 DEST
```

and it means:

```
DEST ←SRC1 (OPERATION) SRC2
```

The operations available are:

```
ADD, SUB, MPY, DVD, AND, OR, XOR
```

The not operation has the form

```
OPERATION SRC1 DEST
```

and it means

```
DEST ←(OPERATION)SRC1
```

2 - There are only two statements in this class which have the operation SHL (shift left) or SHR (shift right). The format is:

```
OPERATION SRC1  COUNT,(1/0) DEST
```

meaning 1 or 0 and store the result in DEST.

3 - These statements are for reading and writing into the variable MEMORY of VMPL. The operations are RMOVE

(read from) and WMOVE (write into). The format is:

```
OPERATION SRC1 SRC2 DEST
```

which means:

```
if operation is RMOVE
   DEST ⟵ SRC1 (SRC2)
else if operation is WMOVE
   SRC1 (SRC2) ⟵ DEST
```

4 - This class deals with the various branch operations.

a. - COMP SRC1 SRC2

is done to set various host flags. The operation requires us to do:

```
SRC1 - SRC2
```

along with the flag settings.

b. - The direct branch statement is:

```
BRCH label
```

meaning go to the label.

c. - Testing flags which usually follows the COMP statement is of the form:

```
OPERATION *FLAG LABEL
```

where operation can be CONDF (condition is false) or CONDT (condition is true). The statement means to branch to the label based on the setting of the flag and the operation, i.e.,

```
CONDF  Ç  ZETA
```

means go to ZETA if C (carry) is not set.

5 - This class includes the following statements:

```
a - INC SRC1             means  SRC1     SRC1 + 1
b - DEC SRC1                    SRC1     SRC2   1
c - SET SRC1                    SRC1     all 1's
d - CLR SRC1                    SRC1     0
e - MOVE SRC1 DEST              DEST     SRC1
f - PUSH SRC1                   Push SRC1 into STACK
g - POP DEST                    Pop from STACK into DEST
h - EXTR  FD  SRC1 Dest
```

FD is declared in IISG as a set of integer numbers, N1, N2, and N3. The 'EXTR' stmt means bit positions N1 through N2 of SRC1 are extracted and shifted right/N3/ bits if N3 is negative, otherwise, shifted left /N3/ bits.

6 - This contains two statements which are translated from the FOR and SELECT statement.

```
a. - LOOP  SRC1  SRC2  SRC3

     means

           FOR  SRC1 = SRC2 TO SRC3

b. -       SLCT SRC1   SRC2

     *          SRC3   Label 1

     *          SRC5   Label 2

     *

     means

           SELECT (SRC1, SRC2) FROM;

                  (SRC3, Label 1);

                  (SRC4, Label 2);

           ENDSELECT;
```

7 - The statements in this class are:

a - HALT means halt

b - XEQ SRC1 PAR1

```
XEQ   SRC1  PAR1
*     PAR2
*
```

means

```
      EXECUTE SRC1 (PAR1, PAR2)
```

c - RET means return from the sub-procedure.

. ____ x ____ x ____

* Flag can also be a bit variable and will be of the form, 'SRC1, SRC2 which means that a reference is made to the SRC2 bit of SRC1.

APPENDIX B

The FDM of PDP11/40

```
FIELD DESCRIPTION MODEL



   FIELD(1):RIF[0:3]              FIELD(2):SRX[4:7]
   FIELD(3):SBAM[13]              FIELD(4):SDM[14:15]
   FIELD(5):SBM[16:19]          →FIELD(6):SALU[24:28] ←
   FIELD(7):SPS[29:31]            FIELD(8):DAD[32:35]
   FIELD(9):EUS[36:38]            FIELD(10):CBA[39]
   FIELD(11):WR[42:43]            FIELD(12):CLK[46:47] ←
   FIELD(13):XUPF+UPF[48:58]      FIELD(14):DEST+MSC[59:63]
   FIELD(15):LML[64:67]           FIELD(16):RML[68:71]
   FIELD(17):SC[72:75]            FIELD(18):EMIT[64:79]
   FIELD(19):CD[40:43]            FIELD(20):CB[41]
   FIELD(21):CLKOFF[45]           FIELD(22):PPE[77]
MOP      1    ADD        *GPR       B          D          P2
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     5=     0
FIELD     6=     9 ←
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    19=     1
THE REST FIELDS ARE NOT USED
MOP      2    SUB        *GPR       B          D          P2
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     5=     0
FIELD     6=     6
FIELD     8=     8
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    19=     1
THE REST FIELDS ARE NOT USED
MOP      3    AND        *GPR       B          D          P2
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     5=     0
FIELD     6=    27
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    19=     1
THE REST FIELDS ARE NOT USED
MOP      4    OR         *GPR       B          D          P2
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     5=     0
FIELD     6=    30
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    19=     1
THE REST FIELDS ARE NOT USED
MOP      5    SUB1       *EMIT      B          D          P2
FIELD     5=     0
FIELD     6=     6
FIELD     8=     8
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=     1
FIELD    18 WILL BE DETERMINED BY EMIT
FIELD    19=     1
```

```
MOP     6    XOR       *GPR      B         D         P2
FIELD    1 WILL BE DETERMINED BY GPR
FIELD    2=     1
FIELD    5=     0
FIELD    6=    22
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   19=     1
THE REST FIELDS ARE NOT USED
MOP     7    INC       *GPR                D         P2
FIELD    1 WILL BE DETERMINED BY GPR
FIELD    2=     1
FIELD    6=     0
FIELD    8=     8
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   19=     1
THE REST FIELDS ARE NOT USED
MOP     8    NOT       *GPR                D         P2
FIELD    1 WILL BE DETERMINED BY GPR
FIELD    2=     1
FIELD    6=    16
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   19=     1
THE REST FIELDS ARE NOT USED
MOP     9    DEC       *GPR                D         P2
FIELD    1 WILL BE DETERMINED BY GPR
FIELD    2=     1
FIELD    6=    15
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   19=     1
THE REST FIELDS ARE NOT USED
MOP    10    CLR                           D         P2
FIELD    6=    19
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   19=     1
THE REST FIELDS ARE NOT USED
MOP    11    SET                           D         P2
FIELD    6=    28
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   19=     1
THE REST FIELDS ARE NOT USED
MOP    12    MOVE1     *GPR                B         P1
FIELD    1 WILL BE DETERMINED BY GPR
FIELD    2=     1
FIELD    4=     0
FIELD   12=     1
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   20=     1
THE REST FIELDS ARE NOT USED
MOP    13    MOVE2     *GPR                84        P1
FIELD    1 WILL BE DETERMINED BY GPR
FIELD    2=     1
FIELD    3=     1
FIELD   10=     1
FIELD   12=     1
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
THE REST FIELDS ARE NOT USED
```

```
MOP     14     MOVE3      *GPR                    D            P2
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     6=     0
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    19=     1
THE REST FIELDS ARE NOT USED
MOP     15     MOVE4      UNIBUS                  *GPR         P1
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     4=     1
FIELD    11=     3
FIELD    12=     1
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
THE REST FIELDS ARE NOT USED
MOP     16     MOVE5      D                       *GPR         P3
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     4=     2
FIELD    11=     3
FIELD    12=     3
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
THE REST FIELDS ARE NOT USED
MOP     17     MOVE6      *EMIT                   D            P2
FIELD     6=     0
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=     1
FIELD    18 WILL BE DETERMINED BY EMIT
FIELD    19=     1
THE REST FIELDS ARE NOT USED
MOP     18     MOVE7      *EMIT                   B            P3
FIELD     4=     0
FIELD    12=     3
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=     1
FIELD    18 WILL BE DETERMINED BY EMIT
FIELD    20=     1
THE REST FIELDS ARE NOT USED
MOP     19     PUSH1      *GPR                    TOS          P2
FIELD     1 WILL BE DETERMINED BY GPR
FIELD     2=     1
FIELD     4=     0
FIELD    12=     2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=     8
FIELD    22=     1
THE REST FIELDS ARE NOT USED
MOP     20     PUSH2      *EMIT                   TOS          P1
FIELD    12=     1
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=     4
FIELD    18 WILL BE DETERMINED BY EMIT
FIELD    22=     1
THE REST FIELDS ARE NOT USED
MOP     21     PUSH3      PS                      TOS          P3
FIELD     4=     0
FIELD     7=     6
FIELD    12=     3
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=     8
FIELD    22=     1
```

```
THE REST FIELDS ARE NOT USED
MOP      22     POP          TOS                        D          P2
FIELD     6=      0
FIELD    12=      2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=      6
FIELD    15=     15
FIELD    16=     15
FIELD    17=      0
FIELD    19=      1
FIELD    22=      1
THE REST FIELDS ARE NOT USED
MOP      23     LMASK        TOS         $CT            B          P3
FIELD     4=      0
FIELD    12=      3
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=      6
FIELD    15 WILL BE DETERMINED BY CT-01
FIELD    16=     15
FIELD    17=      0
FIELD    20=      1
FIELD    22=      1
THE REST FIELDS ARE NOT USED
MOP      24     RMASK        TOS         $CT            B          P3
FIELD     4=      0
FIELD    12=      3
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=      6
FIELD    15=     15
FIELD    16 WILL BE DETERMINED BY CT-01
FIELD    17=      0
FIELD    20=      1
FIELD    22=      1
THE REST FIELDS ARE NOT USED
MOP      25     FLAG         C,V,N,Z                               P1
FIELD     7=      3
FIELD    12=      1
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
THE REST FIELDS ARE NOT USED
MOP      26     BRCH        *LABEL
FIELD    12=      1
FIELD    13 WILL BE DETERMINED BY LABEL
THE REST FIELDS ARE NOT USED              N1  N2  N3
MOP      27     RSMK         TOS         $FF,LL,CT D               P2
FIELD     6=      0    X                   9   11  -9
FIELD    12=      2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=      6
FIELD    15 WILL BE DETERMINED BY LL-CT  11+9
FIELD    16 WILL BE DETERMINED BY 15-FF+CT    15 - 9 - 9
FIELD    17 WILL BE DETERMINED BY CT
FIELD    19=      1
FIELD    22=      1
THE REST FIELDS ARE NOT USED
MOP      28     LSMK         TOS         $FF,LL,CT D               P2
FIELD     6=      0
FIELD    12=      2
FIELD    13 WILL BE DETERMINED BY NEXT ADDR
FIELD    14=      6
FIELD    15 WILL BE DETERMINED BY LL+CT
FIELD    16 WILL BE DETERMINED BY 15-FF-CT
FIELD    17 WILL BE DETERMINED BY 16-CT
FIELD    19=      1
FIELD   22=    1
THE REST FIELDS ARE NOT USED
MOP    29    MOVE8     *GPR                BA        P1
FIELD    1 WILL BE CETERMINEC BY GPR
FIELD    2=    1
FIELD    3=    1
FIELD    9=    1
FIELD   10=    1
FIELD   12=    1
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   21=    1
THE REST FIELDS ARE NOT USED
MOP    30    NOOP      XUPF   --------->   *LABEL
FIELD   12=    1
FIELD   13 WILL BE CETERMINED BY LABEL
THE REST FIELDS ARE NOT USED
MOP    31    LMASK1    TOS       $CT       EUBC      P2
FIELD   12=    2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=    7
FIELD   15 WILL BE DETERMINEC BY CT-01
FIELD   16=   15
FIELD   17=    0
FIELD   22=    1
THE REST FIELDS ARE NOT USED
MOP    32    RMASK1    TOS       $CT       EUBC      P2
FIELD   12=    2
FIELD   13 WILL BE DETERMINEC BY NEXT ADDR
FIELD   14=    7
FIELD   15=    0
FIELD   16=   15
FIELD   17 WILL BE DETERMINEC BY CT
FIELD   22=    1
THE REST FIELDS ARE NOT USED
MOP    33    ORLSM     B         TOS,CT    D         P2
FIELD    5=    0
FIELD    6=   30
FIELD   12=    2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=    6
FIELD   15=   15
FIELD   16 WILL BE DETERMINED BY 15-CT
FIELD   17 WILL BE DETERMINED BY 16-CT
FIELD   19=    1
FIELD   22=    1
THE REST FIELDS ARE NOT USED
MOP    34    ORSM      B         TOS,CT    D         P2
FIELD    5=    0
FIELD    6=   30
FIELD   12=    2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=    6
FIELD   15 WILL BE CETERMINED BY 15-CT
FIELD   16=   15
FIELD   17=    0
FIELD   19=    1
FIELD   22=    1
THE REST FIELDS ARE NOT USED
MOP    35    MOVE9     *GPR                D         P2
FIELD    1 WILL BE CETERMINEC BY GPR
FIELD    2=    1
FIELD    6=    0
FIELD    9=    5
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   19=     1
FIELD   21=     1
THE REST FIELDS ARE NOT USED
MOP     36     MOVE11     *VAR                        BA          P1
FIELD    3=     1
FIELD    9=     1
FIELD   10=     1
FIELD   12=     1
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=     1
FIELD   18 WILL BE DETERMINED BY VAR
FIELD   21=     1
THE REST FIELDS ARE NOT USED
MOP     37     MOVE12     *VAR                        BA          P1
FIELD    3=     1
FIELD   10=     1
FIELD   12=     1
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=     1
FIELD   18 WILL BE DETERMINED BY VAR
THE REST FIELDS ARE NOT USED
MOP     38     MOVE10     *VAR                        D           P2
FIELD    6=     0
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=     1
FIELD   18 WILL BE DETERMINED BY VAR
FIELD   19=     1
THE REST FIELDS ARE NOT USED
MOP     39     CALL       *LABEL                                  P2
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=     3
FIELD   18=     0
FIELD   22=     1
THE REST FIELDS ARE NOT USED
MOP     40     RETURN     RETADR                      EUBC        P2
FIELD   12=     2
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=     7
FIELD   15=    15
FIELD   16=    15
FIELD   17=     0
FIELD   22=     1
THE REST FIELDS ARE NOT USED
MOP     41     PUSH                                               P1
FIELD   12=     1
FIELD   13 WILL BE DETERMINED BY NEXT ADDR
FIELD   14=    11
THE REST FIELDS ARE NOT USED
```

APPENDIX C

The MET of PDP11/40

```
EXAMPLE OF MACRO EXPANSION TABLE OF PDP11/40
```

$F \supseteq A+B$

```
SIMPLE IML CODE
      ↑ADD        SRC1        SRC2        DEST
THE CORRESPONDING MOPS
          MOVE1       SRC2                        B
          ADD         SRC1        B               D
          MOVE5       D                           DEST
          FLAG


SIMPLE IML CODE
       AND        SRC1        SRC2        DEST
THE CORRESPONDING MOPS
          MOVE1       SRC2                        B
          AND         SRC1        B               D
          MOVE5       D                           DEST


SIMPLE IML CODE
       NOT        SRC1                    DEST
THE CORRESPONDING MOPS
          NOT         SRC1                        D
          MOVE5       D                           DEST


SIMPLE IML CODE
      ↑SUB        SRC1        SRC2        DEST
THE CORRESPONDING MOPS
          MOVE1       SRC2                        B
          SUB         SRC1        B               D
          MOVE5       D                           DEST
          FLAG


SIMPLE IML CODE
       XOR        SRC1        SRC2        DEST
THE CORRESPONDING MOPS
          MOVE1       SRC2                        B
          XOR         SRC1        B               D
          MOVE5       D                           DEST


SIMPLE IML CODE
       OR         SRC1        SRC2        DEST
THE CORRESPONDING MOPS
          MOVE1       SRC2                        B
          OR          SRC1        B               D
          MOVE5       D                           DEST


SIMPLE IML CODE
       SHR        SRC         5,1         DEST
THE CORRESPONDING MOPS
          PUSH2       65535                       TOS
          LMASK     TOS       5          B
          PUSH1     SRC                  TOS
          ORSM      B         TOS,5      D
          MOVE5     D                    DEST


SIMPLE IML CODE
       SHL      SRC      6,1     DEST
THE CORRESPONDING MOPS
          PUSH2     65535                TOS
          LMASK     TOS       6          B
          PUSH1     SRC                  TOS
          ORLSM     B         TOS,6      D
          MOVE5     D                    DEST


SIMPLE IML CODE
       SHL      SRC      3,0     DEST
THE CORRESPONDING MOPS
          PUSH1     SRC                  TOS
          LSMK      TOS       NEWCHARA00D
          MOVE5     D                    DEST


SIMPLE IML CODE
       SHR      SRC      4,0     DEST
THE CORRESPONDING MOPS
          PUSH1     SRC                  TOS
          RSMK      TOS       NEWCHARA01D
          MOVE5     D                    DEST


SIMPLE IML CODE
       RMOVE    MEM      SRC     DEST
THE CORRESPONDING MOPS
          MOVE3     SRC                  BA
          MOVE4     UNIBUS               DEST


SIMPLE IML CODE
       WMOVE    MEM      SRC     DEST
THE CORRESPONDING MOPS
          MOVE2     SRC                  BA
          MOVE9     DEST                 D
          NOOP


SIMPLE IML CODE
       DEC      SRC1
THE CORRESPONDING MOPS
          DEC       SRC1                 D
          MOVE5     D                    SRC1


SIMPLE IML CODE
       SET      SRC1
```

```
THE CORRESPONDING MOPS
          SET                           D
          MOVE5     D                   SRC1


SIMPLE IML CODE
       INC      SRC1
THE CORRESPONDING MOPS
          INC       SRC1                D
          MOVE5     D                   SRC1


SIMPLE IML CODE
       CLR      SRC1
THE CORRESPONDING MOPS
          CLR                           D
          MOVE5     D                   SRC1


SIMPLE IML CODE
       MOVE     SRC1     DEST
THE CORRESPONDING MOPS
          MOVE3     SRC1                D
          MOVE5     D                   DEST


SIMPLE IML CODE
       EXTR     CRNTPG   SRC      DEST
THE CORRESPONDING MOPS
          PUSH1     SRC                 TOS
          RSMK      TOS       CRNTPG    D
          MOVE5     D                   DEST


SIMPLE IML CODE
      ↑COMP     SRC1     DEST
THE CORRESPONDING MOPS
          MOVE1     DEST                B
          SUB       SRC1      B         D
          FLAG


SIMPLE IML CODE
      ↑COMP     SRC1    C8
THE CORRESPONDING MOPS
          MOVE7     8                   B
          SUB       SRC1      B         D
          FLAG


SIMPLE IML CODE
       CONDF    .SRC,7   FLABEL2
THE CORRESPONDING MOPS
          PUSH1     SRC                 TOS
          RMASK1    TOS       7         EUBC
          NOOP      XUPF      P.001
          BRCH      LABEL2    P.001     1
```

```
SIMPLE IML CODE
      CONDF     N        FLABEL2
THE CORRESPONDING MOPS
          PUSH3     PS                  TOS
          RMASK1    TOS       3         EUBC
          NOOP      XUPF      P.002
          BRCH      LABEL2    P.002     1


SIMPLE IML CODE
      CONDT     C        TLABEL1
THE CORRESPONDING MOPS
          PUSH3     PS                  TOS
          RMASK1    TOS       0         EUBC
          NOOP      XUPF      P.003
          BRCH      LABEL1    P.003+1   1


SIMPLE IML CODE
      CONDT     .SR,8    LLABEL4
THE CORRESPONDING MOPS
          PUSH1     SR                  TOS
          RMASK1    TOS       8         EUBC
          NOOP      XUPF      P.004
          JUMP      LABEL4    P.004+1   1


SIMPLE IML CODE
      BRCH      LLABEL
THE CORRESPONDING MOPS
          BRCH      LABEL


SIMPLE IML CODE
      SLCT      SRC1     C3
*                        C0        SSUBR1
*                        C1        SSUBR2
*                        C2        SSUBR3
*
THE CORRESPONDING MOPS
          PUSH1     SRC1                TOS
          LMASK1    TOS       2         EUBC
          NOOP      XUPF      P.005
          UNJP      SUBR1     P.005     2
          UNJP      SUBR2     P.005+01  2
          UNJP      SUBR3     P.005+02  2


SIMPLE IML CODE
      XEQ       SUBR
*
THE CORRESPONDING MOPS
          PUSH
          CALL      SUBR
```

```
SIMPLE IML CODE
       RET
THE CORRESPONDING MCPS
          RETURN    RETADR                   EUBC
          N0001     XUPF      0
```

APPENDIX D

Case Where Virtual Machine Word Size is Integer Multiple of Host Machine Word Size

APPENDIX D

THE VM WORDSIZE IS 32 BITS AND THE HM IS 16 BITS.TO SOLVE THIS KIND OF WORDSIZE DIFFERENCE PROBLEM, THE VARIABLE BASED ON THE VM WORDSIZE HAS TO BE BINDED INTO SEVERAL VARIABLES BASED ON THE HM WORDSIZE. THEN,THE IML STATEMENT WHICH THE VARIABLES ARE DECLARED IN THE VM WORDSIZE IS EXPANDED INTO A SET OF IML STATEMENTS WHICH THE VARIABLES ARE BASED ON THE HM WORDSIZE. IN THIS EXAMPLE, THE LOWER 16 BITS OF VARIABLE,AB,IS DENOTED BY AB0,AND THE HIGHER 16 BITS OF THE VARIABLE IS DENOTED BY AB1.

```
    THIS IML CODE IS BASED ON VM WORDSIZE
          ADD       AB        CD        EF
    THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
         ↑ADD       AB0       CD0       EF0
          CONDF     C         L.00X
          INC       AB1
    L.00X ADD       AB1       CD1       EF1


    THIS IML CODE IS BASED ON VM WORDSIZE
          AND       AB        CD        EF
    THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
          AND       AB0       CD0       EF0
          AND       AB1       CD1       EF1


    THIS IML CODE IS BASED ON VM WORDSIZE
          XOR       AB        CD        EF
    THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
          XOR       AB0       CD0       EF0
          XOR       AB1       CD1       EF1


    THIS IML CODE IS BASED ON VM WORDSIZE
          OR        AB        CD        EF
    THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
          OR        AB0       CD0       EF0
          OR        AB1       CD1       EF1


    THIS IML CODE IS BASED ON VM WORDSIZE
          SUB       AB        CD        EF
    THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
          NOT       CD0                 CD0
          NOT       CD1                 CD1
         ↑INC       CD0
          CONDF     C         L.00Z
          INC       CD1
    L.00Z↑ADD       AB0       CD0       EF0
          CONDF     C         L.00W
          INC       AB1
    L.00W ADD       AB1       CD1       EF1


    THIS IML CODE IS BASED ON VM WORDSIZE
```

```
       NOT      AB               CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       NOT      AB0              CD0
       NOT      AB1              CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       RMOVE     MEM     AB       CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       RMOVE    MEM      AB0      CD0
       RMOVE    MEM      AB1      CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       WMOVE     MEM     AB       CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       WMOVE    MEM      AB0      CD0
       WMOVE    MEM      AB1      CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       CLR      AB
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       CLR      AB0
       CLR      AB1


THIS IML CODE IS BASED ON VM WORDSIZE
       DEC      AB
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      ↑DEC      AB0
       CONDF    C        L.00G
       INC      AB1
L.00G  DEC      AB1


THIS IML CODE IS BASED ON VM WORDSIZE
       SET      AB
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       SET      AB0
       SET      AB1


THIS IML CODE IS BASED ON VM WORDSIZE
       MOVE     AB               CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       MOVE     AB0              CD0
       MOVE     AB1              CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       MOVE    C1234             CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       MOVE    C1234             CD0
       MOVE    C0                CD1
```

```
THIS IML CODE IS BASED ON VM WORDSIZE
       MOVE    C1234567           CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       MOVE    C54919             CD0
       MOVE    C18                CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       INC      AB
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      ↑INC      AB0
       CONDF    C        L.00G
       INC      AB1
L.00G  (NEXT    IML)


THIS IML CODE IS BASED ON VM WORDSIZE
      ↑COMP     AB       CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       NOT      CD0               CD0
       NOT      CD1               CD1
      ↑INC      CD0
       CONDF    C        L.00Z
       INC      CD1
L.00Z ↑ADD      AB0      CD0      TEMP0
       CONDF    C        L.00W
       INC      AB1
L.00W ↑ADD      AB1      CD1      TEMP1


THIS IML CODE IS BASED ON VM WORDSIZE
       CONDF   .AB,4     LABEL2
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       CONDF   .AB0,4    LABEL2


THIS IML CODE IS BASED ON VM WORDSIZE
       CONDT   .AB,23    LABEL1
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       CONDT   .AB1,7    LABEL1


THIS IML CODE IS BASED ON VM WORDSIZE
       SHR      AB       19,1     CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       SHR      AB1      3,1      CD0
       MOVE    C65535             CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       SHR      AB       18,0     CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       SHR      AB1      2,0      CD0
       MOVE    C0                 CD1
```

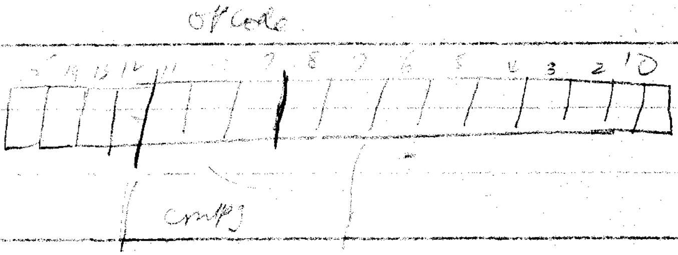
```
THIS IML CODE IS BASED ON VM WORDSIZE
      SHR      AB       5,0      CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      SHR      AB0      5,0      CD0
      EXTR     CHARA00  AB1      CD1
      OR       CD1      CD0      CD0
      SHR      AB1      5,0      CD1


THIS IML CODE IS BASED ON VM WORDSIZE
      SHR      AB       6,1      CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      SHR      AB0      6,0      CD0
      EXTR     CHARA01  AB1      CD1
      OR       CD1      CD0      CD0
      SHR      AB1      6,1      CD1


THIS IML CODE IS BASED ON VM WORDSIZE
      SHL      AB       5,0      CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      SHL      AB1      5,0      CD1
      SHR      AB0      11,0     CD0
      OR       CD1      CD0      CD1
      SHL      AB0      5,0      CD0


THIS IML CODE IS BASED ON VM WORDSIZE
      SHL      AB       6,1      CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      SHL      AB1      6,0      CD1
      SHR      AB0      10,0     CD0
      OR       CD1      CD0      CD1
      SHL      AB0      6,1      CD0


THIS IML CODE IS BASED ON VM WORDSIZE
      SHL      AB       18,0     CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      SHL      AB0      2,0      CD1
      MOVE    C0                 CD0


THIS IML CODE IS BASED ON VM WORDSIZE
      SHL      AB       18,1     CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      SHL      AB0      2,1      CD1
      MOVE    C65535             CD0


THIS IML CODE IS BASED ON VM WORDSIZE
      EXTR     CHAR1    AB       CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
      EXTR     CHARA02  AB0      CD0
      MOVE    C0                 CD1
```

```
THIS IML CODE IS BASED ON VM WORDSIZE
       EXTR      CHAR2     AB        CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       EXTR      CHARA03   AB1       CD1
       MOVE     C0                   CD0


THIS IML CODE IS BASED ON VM WORDSIZE
       EXTR      CHAR3     AB        CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       EXTR      CHARA04   AB0       CD0
       EXTR      CHARA05   AB1       CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       EXTR      CHAR4     AB        CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       SHR       AB0       5,0       TEMP0
       EXTR      CHARA06   AB1       TEMP1
       OR        TEMP1     TEMP0     TEMP0
       SHR       AB1       5,0       TEMP1
       EXTR      CHARA08   TEMP0     CD0
       EXTR      CHARA09   TEMP1     CD1


THIS IML CODE IS BASED ON VM WORDSIZE
       EXTR      CHAR5     AB        CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
       SHL       AB1       7,0       TEMP1
       SHR       AB0       9,0       TEMP0
       OR        TEMP1     TEMP0     TEMP1
       SHL       AB0       7,0       TEMP0
       EXTR      CHARA11   TEMP1     CD1
       MOVE     C0                   CD0
CHAR1      14    3    0
CHAR2      27   19    0
CHAR3      28   12    0
CHAR4      25   14   -5
CHAR5      21   14    7
CHARA00     4    0   11
CHARA01     5    0   10
CHARA02    14    3    0
CHARA03    27   19    0
CHARA04    15   12    0
CHARA05    12    0    0
CHARA06     4    0   11
CHARA07    20    9    0
CHARA08    15    9    0
CHARA09     4    0    0
CHARA10    28   21    0
CHARA11    28   21    0
```

APPENDIX E-1

IISG of Emulator PDP8

```
00A PDP8
00D ,,,12
00E TWO
221 MEM,4096,12
220 ACCM,,12
220 PC,,12
220 MAR,,12
210 IR,,12
210 MDR,,12
210 OPCD,,3
214 LNK,,1,1
229 IOINST,,,P
229 DATASW,,12
005 OPCODE,,,9,11,-9
005 CRNTPG,,,7,11,0
005 PGEADR,,,0,6,0
005 ROTFLD,,,1,3,-1
005 OSC,,,3,8,-3
005 OSB,,,0,2,0
306 C
00B PROGRAMSTART
00F INF
206 MEM
206 IR
206 PC
00F INSTDC
206 IR
206 OPCD
00G EFTADR
207 MEM
207 IR
207 PC
208 MAR
120 ADR,,7
120 PCTEMP,,12
120 MART,,12
00F MRI
206 MAR
206 MEM
206 MDR
206 OPCD
406 EFTADR
00F AND
206 ACCM
206 MDR
00F TAD
206 ACCM
206 MDR
206 LNK
00F ISZ
206 ACCM
206 MAR
206 PC
00F DCA
206 MEM
206 ACCM
206 MAR
00F JMS
206 MEM
206 MAR
206 PC
00F JMP
206 PC
```



```
206 MAR
00F IO
206 IR
406 IOINST
120 OS,,5
120 OS,,3
00F OPT
206 IR
00F OPR1
206 IR
206 ACCM
206 LNK
120 ROTACT,,3
00F RAL
205 LNK
206 ACCM
00F RTL
206 LNK
206 ACCM
00F RAR
206 ACCM
206 LNK
00F RTR
205 LNK
206 ACCM
00F OPR2
206 IR
206 ACCM
206 PC
206 LNK
206 DATASW
120 COUNT,,2
120 CHECK,,2
00C PROGRAMEND
```

APPENDIX E-2

IESG of Emulator PDP8

```
00A PDP8
00B PROGRAMSTART
00F INF
00H
        RMOVE    MEM      PC      +T.001
        MOVE     -T.001   IR
        INC      PC
00I
00F INSTDC
00H
        EXTR     OPCODE   IR      +T.002
        MOVE     -T.002   OPCD
        SLCT     OPCD     C8
*                         C0      SMRI
*                         C1      SMRI
*                         C2      SMRI
*                         C3      SDCA
*                         C4      SJMS
*                         C5      SJMP
*                         C6      SIO
*                         C7      SOPT
*
00I
00G EFTADR
00H
        EXTR     PGEADR   IR      +T.003
        MOVE     -T.003   ADR
        CONDF    .IR,7    TL.001
00J
        SUB      PC       C1       PCTEMP ⇐
        EXTR     CRNTPG   PCTEMP  +T.004
        MOVE     -T.004   PCTEMP
        OR       PCTEMP   ADR      MAR
00K
        BRCH     EL.002
L.001
00J
        MOVE     ADR      MAR
00K
L.002
        CONDF    .IR,8    TL.003
00J
        RMOVE    MEM      MAR     +T.005
        MOVE     -T.005   MART
       +COMP     MAR      C8                N
        CONDT    N        TL.004
00J
       +COMP     MAR      C16               N
        CONDF    N        TL.005
00J
        ADD      MART     C1      +T.007
        MOVE     +T.007   MART
        WMOVE    MEM      MAR     -T.007
00K
L.005
00K
L.004
        MOVE     MART     MAR
00K
L.003
        RET
00I
00F MRI
```

```
00H
        XEQ       EFTADR

*
        RMOVE     MEM       MAR       +T.010
        MOVE      -T.010    MDR
        SLCT      OPCD      C3
*                           C0        SAND
*                           C1        STAD
*                           C2        SISZ
*
00I
00F AND
00H
        AND       ACCM      MDR       ACCM
        BRCH      BINF
00I
00F TAD
00H
       *ADD       ACCM      MDR       ACCM      C
        BRCH      BINF
00I
00F ISZ
00H
        RMOVE     MEM       MAR       +T.011
        ADD       -T.011    C1        +T.012
        WMOVE     MEM       MAR       -T.012
        RMOVE     MEM       MAR       +T.013
       *COMP      -T.013    C0                  Z
        CONDF     Z         TL.006
00J
        INC       PC
00K
L.006
        BRCH      EINF
00I
00F DCA
00H
        XEQ       EFTADR

*
        WMOVE     MEM       MAR       ACCM
        CLR       ACCM
        BRCH      BINF
00I
00F JMS
00H
        XEQ       EFTADR

*
        WMOVE     MEM       MAR       PC
        INC       MAR
        MOVE      MAR       PC
        BRCH      BINF
00I
00F JMP
00H
        XEQ       EFTADR

*
        MOVE      MAR       PC
        BRCH      BINF
00I
```

```
00F IO
00H
       NOOP
       BRCH    BINF
00I
00F OPT
00H
       NOOP1
00I
00C PROGRAMEND
```

APPENDIX E-3

Output of Pass 1

```
OUTPUT OF PASS1
BKS
INF         MOVE8      *1+PC                   BA
            MOVE4      UNIBUS                  *2+T.001
            MOVE3      *2-T.001                D
            MOVE5      D                       *1-IR
            MOVE7      16                      B
            ADD        *1+PC      B            D
            MOVE5      D                       *1-PC
BKS
INSTDC      PUSH1      *1-IR                   TOS
            RSMK       TOS        OPCODE       D
            MOVE5      D                       *2+T.002
            MOVE3      *2-T.002                D
            MOVE5      D                       *1+OPCD
            PUSH1      *1-OPCD                 TOS
            RSMK       TOS        NEWCHARA00D
            MOVE5      D          Test         *2+T.00X
            PUSH1      *2-T.00X   bit 3        TOS       copies bit 3 of
            LMASK1     TOS        3            EUBC      TOS to bit 15
            NOOP       XUPF       P.001                  of EUBC
            UNJP       MRI        P.001        3
            UNJP       MRI        P.001+01     3
            UNJP       MRI        P.001+02     3
            UNJP       DCA        P.001+03     3
            UNJP       JMS        P.001+04     3
            UNJP       JMP        P.001+05     3
            UNJP       IO         P.001+06     3
            UNJP       OPT        P.001+07     3
BKS
EFTADR      PUSH1      *1+IR                   TOS
            RSMK       TOS        PGEADR       D
            MOVE5      D                       *2+T.003
            MOVE3      *2-T.003                D
            MOVE5      D                       *2+ADR
            PUSH1      *1+IR                   TOS
            RMASK1     TOS        11           EUBC
            NOOP       XUPF       P.002
            BRCH       L.001      P.002        1
            MOVE7      16                      B
            SUB        *1-PC      B            D
            MOVE5      D                       *2+PCTEMP
            PUSH1      *2+PCTEMP               TOS
            RSMK       TOS        CRNTPG       D
            MOVE5      D                       *2+T.004
            MOVE3      *2-T.004                D
            MOVE5      D                       *2+PCTEMP
            MOVE1      *2+ADR                  B
            OR         *2-PCTEMP  B            D
            MOVE5      D                       *1+MAR
            BRUN       L.002
L.001       MOVE3      *2-ADR                  D
            MOVE5      D                       *1+MAR
L.002       PUSH1      *1-IR                   TOS
            RMASK1     TOS        12           EUBC
            NOOP       XUPF       P.003
            BRCH       L.003      P.003        1
            MOVE8      *1+MAR                  BA
            MOVE4      UNIBUS                  *2+T.005
            MOVE3      *2-T.005                D
            MOVE5      D                       *2+MART
            MOVE7      128                     B
```

```
          SUB        *1+MAR     B          D
          FLAG
          PUSH3      PS                    TOS
          RMASK1     TOS        3          EUBC
          NOOP       XUPF       P.004
          BRCH       L.004      P.004+1    1
          MOVE7      256                   B
          SUB        *1+MAR     B          D
          FLAG
          PUSH3      PS                    TOS
          RMASK1     TOS        3          EUBC
          NOOP       XUPF       P.005
          BRCH       L.005      P.005      1
          MOVE7      16                    B
          ADD        *2+MART    B          D
          MOVE5      D                     *2+T.007
          MOVE3      *2+T.007              D
          MOVE5      D                     *2+MART
          MOVE2      *1+MAR                BA
          MOVE9      *2-T.007              D
          NOOP
L.005     NOOP
L.004     MOVE3      *2-MART               D
          MOVE5      D                     *1-MAR
L.003     RETURN     RETADR                EUBC
          NOOP1      XUPF       0
BKS
MRI       PUSH
          CALL       EFTADR
          MOVE8      *1-MAR                BA
          MOVE4      UNIBUS                *2+T.010
          MOVE3      *2-T.010              D
          MOVE5      D                     *1-MDR
          PUSH1      *1-OPCD               TOS
          RSMK       TOS        NEWCHARA01D
          MOVE5      D                     *2+T.00X
          PUSH1      *2-T.00X              TOS
          LMASK1     TOS        2          EUBC
          NOOP       XUPF       P.006
          UNJP       AND        P.006      2
          UNJP       TAD        P.006+01   2
          UNJP       ISZ        P.006+02   2
BKS
AND       MOVE1      *1-MDR                B
          AND        *1+ACCM    B          D
          MOVE5      D                     *1-ACCM
          UNJP       INF
BKS
TAD       MOVE1      *1-MDR                B
          ADD        *1+ACCM    B          D
          MOVE5      D                     *1-ACCM
          FLAG
          PUSH3      PS                    TOS
          RSMK       TOS        NEWCHARA02D
          MOVE5      D                     *1+LNK
          PUSH1      *1+LNK                TOS
          LSMK       TOS        NEWCHARA03D
          MOVE5      D                     *1-LNK
          UNJP       INF
BKS
ISZ       MOVE8      *1+MAR                BA
          MOVE4      UNIBUS                *2+T.011
          MOVE7      16                    B
```

```
           SUB       *1+MAR    B          D
           FLAG
           PUSH3     PS                   TOS
           RMASK1    TOS       3          EUBC
           NOOP      XUPF      P.004
           BRCH      L.004     P.004+1    1
           MOVE7     256                  B
           SUB       *1+MAR    B          D
           FLAG
           PUSH3     PS                   TOS
           RMASK1    TOS       3          EUBC
           NOOP      XUPF      P.005
           BRCH      L.005     P.005      1
           MOVE7     16                   B
           ADD       *2+MART   B          D
           MOVE5     D                    *2+T.007
           MOVE3     *2+T.007             D
           MOVE5     D                    *2+MART
           MOVE2     *1+MAR               BA
           MOVE9     *2-T.007             D
           NOOP
L.005      NOOP
L.004      MOVE3     *2-MART              D
           MOVE5     D                    *1-MAR
L.003      RETURN    RETADR               EUBC
           NOOP1     XUPF      0
BKS
MRI        PUSH
           CALL      EFTADR
           MOVE8     *1-MAR               BA
           MOVE4     UNIBUS               *2+T.010
           MOVE3     *2-T.010             D
           MOVE5     D                    *1-MDR
           PUSH1     *1-OPCD              TOS
           RSMK      TOS       NEWCHARA010D
           MOVE5     D                    *2+T.00X
           PUSH1     *2-T.00X             TOS
           LMASK1    TOS       2          EUBC
           NOOP      XUPF      P.006
           UNJP      AND       P.006      2
           UNJP      TAD       P.006+01   2
           UNJP      ISZ       P.006+02   2
BKS
AND        MOVE1     *1-MDR               B
           AND       *1+ACCM   B          D
           MOVE5     D                    *1-ACCM
           UNJP      INF
BKS
TAD        MOVE1     *1-MDR               B
           ADD       *1+ACCM   B          D
           MOVE5     D                    *1-ACCM
           FLAG
           PUSH3     PS                   TOS
           RSMK      TOS       NEWCHARA020
           MOVE5     D                    *1+LNK
           PUSH1     *1+LNK               TOS
           LSMK      TOS       NEWCHARA030
           MOVE5     D                    *1-LNK
           UNJP      INF
BKS
ISZ        MOVE8     *1+MAR               BA
           MOVE4     UNIBUS               *2+T.011
           MOVE7     16                   B
```

```
          ADD       *2-T.011  B         D
          MOVE5     D                   *2+T.012
          MOVE2     *1+MAR              BA
          MOVE9     *2-T.012            D
          NOOP
          MOVE8     *1-MAR              BA
          MOVE4     UNIBUS              *2+T.013
          MOVE7     0                   B
          SUB       *2-T.013  B         D
          FLAG
          PUSH3     PS                  TOS
          RMASK1    TOS       2         EUBC
          NOOP      XUPF      P.007
          BRCH      L.006     P.007     1
          MOVE7     16                  B
          ADD       *1+PC     B         D
          MOVE5     D                   *1-PC
L.006     UNJP      INF
BKS
DCA       PUSH
          CALL      EFTADR
          MOVE2     *1-MAR              BA
          MOVE9     *1+ACCM             D
          NOOP
          CLR                           D
          MOVE5     D                   *1-ACCM
          UNJP      INF
BKS
JMS       PUSH
          CALL      EFTADR
          MOVE2     *1+MAR              BA
          MOVE9     *1+PC               D
          NOOP
          MOVE7     16                  B
          ADD       *1+MAR    B         D
          MOVE5     D                   *1+MAR
          MOVE3     *1-MAR              D
          MOVE5     D                   *1-PC
          UNJP      INF
BKS
JMP       PUSH
          CALL      EFTADR
          MOVE3     *1-MAR              D
          MOVE5     D                   *1-PC
          UNJP      INF
BKS
IO        NOOP
          UNJP      INF
BKS
OPT       NOOP1
THE NUMBER OF CODES  174
```

APPENDIX E-4

Output of Pass 2

```
OUTPUT OF PASS2



          MOVE10     PC                   D         P2
FIELD 6=    0
FIELD12=    2
FIELD14=    1
FIELD18= 2048
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5      D                    R13       P3
FIELD 1=    2
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
INF       MOVE8      R13                  BA        P1
FIELD 1=    2
FIELD 2=    1
FIELD 3=    1
FIELD 9=    1
FIELD10=    1
FIELD12=    1
FIELD21=    1
THE REST FIFLDS ARE NOT USED
          MOVE4      UNIBUS               R12       P1
FIELD 1=    3
FIELD 2=    1
FIELD 4=    1
FIELD11=    3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE3      R12                  D         P2
FIELD 1=    3
FIELD 2=    1
FIELD 6=    0
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5      D                    R12       P3
FIELD 1=    3
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE7      16                   B         P3
FIELD 4=    0
FIELD12=    3
FIELD14=    1
FIELD18=   16
FIELD20=    1
THE REST FIELDS ARE NOT USED
          ADD        R13       B          D         P2
FIELD 1=    2
FIELD 2=    1
FIELD 5=    0
FIELD 6=    9
FIELD12=    2
```

```
FIELD19=     1
THE REST FIELDS ARE NOT USED
          MOVE5      D                      R13        P3
FIELD 1=     2
FIELD 2=     1
FIELD 4=     2
FIELD11=     3
FIELD12=     3
THE REST FIELDS ARE NOT USED
INSTDC    PUSH1      R12                    TOS        P2
FIELD 1=     3
FIELD 2=     1
FIELD 4=     0
FIELD12=     2
FIELD14=     8
FIELD22=     1
THE REST FIELDS ARE NOT USED
          RSMK       TOS       OPCODE       D          P2
FIELD 6=     0
FIELD12=     2
FIELD14=     6
FIELD15=     6
FIELD16=    11
FIELD17=     9
FIELD19=     1
FIELD22=     1
THE REST FIELDS ARE NOT USED
          MOVE5      D                      R11        P3
FIELD 1=     4
FIELD 2=     1
FIELD 4=     2
FIELD11=     3
FIELD12=     3
THE REST FIELDS ARE NOT USED
          MOVE3      R11                    D          P2
FIELD 1=     4
FIELD 2=     1
FIELD 6=     0
FIELD12=     2
FIELD19=     1
THE REST FIELDS ARE NOT USED
          MOVE5      D                      R11        P3
FIELD 1=     4
FIELD 2=     1
FIELD 4=     2
FIELD11=     3
FIELD12=     3
THE REST FIELDS ARE NOT USED
          PUSH1      R11                    TOS        P2
FIELD 1=     4
FIELD 2=     1
FIELD 4=     0
FIELD12=     2
FIELD14=     8
FIELD22=     1
THE REST FIELDS ARE NOT USED
          RSMK       TOS       NEWCHARA000             P2
FIELD 6=     0
FIELD12=     2
FIELD14=     6
FIELD15=    11
FIELD16=    15
FIELD17=     4
```

```
FIELD19=    1
FIELD22=    1
THE REST FIELDS ARE NOT USED
          MOVE5      D                    R10        P3
FIELD 1=    5
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          PUSH1      R10                  TOS        P2
FIELD 1=    5
FIELD 2=    1
FIELD 4=    0
FIELD12=    2
FIELD14=    8
FIELD22=    1
THE REST FIELDS ARE NOT USED
          LMASK1     TOS       3          EUBC       P2
FIELD12=    2
FIELD14=    7
FIELD15=    2
FIELD16=   15
FIELD17=    0
FIELD22=    1
THE REST FIELDS ARE NOT USED
          NOOP       XUPF      P.001
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       MRI       P.001      3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       MRI       P.001+01   3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       MRI       P.001+02   3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       DCA       P.001+03   3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       JMS       P.001+04   3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       JMP       P.001+05   3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       IO        P.001+06   3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       OPT       P.001+07   3
FIELD12=    1
THE REST FIELDS ARE NOT USED
EFTADR    PUSH1      R12                  TOS        P2
FIELD 1=    3
FIELD 2=    1
FIELD 4=    0
FIELD12=    2
FIELD14=    8
FIELD22=    1
THE REST FIELDS ARE NOT USED
          RSMK       TOS       PGEADR     D          P2
FIELD 6=    0
```

```
FIELD12=    2
FIELD14=    6
FIELD15=   10
FIELD16=   11
FIELD17=    0
FIELD19=    1
FIELD22=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R10        P3
FIELD 1=    5
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE3     R10                 D          P2
FIELD 1=    5
FIELD 2=    1
FIELD 6=    0
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R10        P3
FIELD 1=    5
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          PUSH1     R12                 TOS        P2
FIELD 1=    3
FIELD 2=    1
FIELD 4=    0
FIELD12=    2
FIELD14=    8
FIELD22=    1
THE REST FIELDS ARE NOT USED
          RMASK1    TOS       11        EUBC       P2
FIELD12=    2
FIELD14=    7
FIELD15=    0
FIELD16=   15
FIELD17=   11
FIELD22=    1
THE REST FIELDS ARE NOT USED
          NOOP      XUPF      P.002
FIELD12=    1
THE REST FIELDS ARE NOT USED
          BRCH      L.001     P.002     1
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE7     16                  B          P3
FIELD 4=    0
FIELD12=    3
FIELD14=    1
FIELD18=   16
FIELD20=    1
THE REST FIELDS ARE NOT USED
          SUB       R13       B         D          P2
FIELD 1=    2
FIELD 2=    1
FIELD 5=    0
FIELD 6=    6
```

```
      FIELD 8=    8
      FIELD12=    2
      FIELD19=    1
      THE REST FIELDS ARE NOT USED
                MOVE5      D                    R9         P3
      FIELD 1=    9
      FIELD 2=    1
      FIELD 4=    2
      FIELD11=    3
      FIELD12=    3
      THE REST FIELDS ARE NOT USED
                PUSH1      R9                   TOS        P2
      FIELD 1=    9
      FIELD 2=    1
      FIELD 4=    0
      FIELD12=    2
      FIELD14=    8
      FIELD22=    1
      THE REST FIELDS ARE NOT USED
                RSMK       TOS       CRNTPG     D          P2
      FIELD 6=    0
      FIELD12=    2
      FIELD14=    6
      FIELD15=   15
      FIELD16=    4
      FIELD17=    0
      FIELD19=    1
      FIELD22=    1
      THE REST FIELDS ARE NOT USED
                MOVE5      D                    R8         P3
      FIELD 1=    8
      FIELD 2=    1
      FIELD 4=    2
      FIELD11=    3
      FIELD12=    3
      THE REST FIELDS ARE NOT USED
                MOVE3      R8                   D          P2
      FIELD 1=    8
      FIELD 2=    1
      FIELD 6=    0
      FIELD12=    2
      FIELD19=    1
      THE REST FIELDS ARE NOT USED
                MOVE5      D                    R9         P3
      FIELD 1=    9
      FIELD 2=    1
      FIELD 4=    2
      FIELD11=    3
      FIELD12=    3
      THE REST FIELDS ARE NOT USED
                MOVE1      R10                  B          P1
      FIELD 1=    5
      FIELD 2=    1
      FIELD 4=    0
      FIELD12=    1
      FIELD20=    1
      THE REST FIFLDS ARE NOT USED
                OR         R9        B          D          P2
      FIELD 1=    9
      FIELD 2=    1
      FIELD 5=    0
      FIELD 6=   30
      FIELD12=    2
```

```
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R8        P3
FIELD 1=    8
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE12    PCTEMP              BA        P1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
FIELD14=    1
FIELD18= 3079
THE REST FIELDS ARE NOT USED
          MOVE9     R9                  D         P2
FIELD 1=    9
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIFLDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
          BRUN      L.002
FIELD12=    1
THE REST FIELDS ARE NOT USED
L.001     MOVE3     R10                 D         P2
FIELD 1=    5
FIELD 2=    1
FIELD 6=    0
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R8        P3
FIELD 1=    8
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
L.002     PUSH1     R12                 TOS       P2
FIELD 1=    3
FIELD 2=    1
FIELD 4=    0
FIELD12=    2
FIELD14=    8
FIELD22=    1
THE REST FIELDS ARE NOT USED
          RMASK1    TOS       12        EUBC      P2
FIELD12=    2
FIELD14=    7
FIELD15=    0
FIELD16=   15
FIELD17=   12
FIELD22=    1
THE REST FIELDS ARE NOT USED
          NOOP      XUPF      F.003
FIELD12=    1
```

```
THE REST FIELDS ARE NOT USED
        BRCH    L.003   P.003   1
FIELD12=   1
THE REST FIELDS ARE NOT USED
        MOVE8   R8              BA      P1
FIELD 1=   8
FIELD 2=   1
FIELD 3=   1
FIELD 9=   1
FIELD10=   1
FIELD12=   1
FIELD21=   1
THE REST FIELDS ARE NOT USED
        MOVE4   UNIBUS          R9      P1
FIELD 1=   9
FIELD 2=   1
FIELD 4=   1
FIELD11=   3
FIELD12=   1
THE REST FIELDS ARE NOT USED
        MOVE3   R9              D       P2
FIELD 1=   9
FIELD 2=   1
FIELD 6=   0
FIELD12=   2
FIELD19=   1
THE REST FIELDS ARE NOT USED
        MOVE5   D               R9      P3
FIELD 1=   9
FIELD 2=   1
FIELD 4=   2
FIELD11=   3
FIELD12=   3
THE REST FIELDS ARE NOT USED
        MOVE7   128             B       P3
FIELD 4=   0
FIELD12=   3
FIELD14=   1
FIELD18= 128
FIELD20=   1
THE REST FIELDS ARE NOT USED
        SUB     R8      B       D       P2
FIELD 1=   8
FIELD 2=   1
FIELD 5=   0
FIELD 6=   6
FIELD 8=   8
FIELD12=   2
FIELD19=   1
THE REST FIELDS ARE NOT USED
        FLAG                            P1
FIELD 7=   3
FIELD12=   1
THE REST FIELDS ARE NOT USED
        PUSH3   PS              TOS     P3
FIELD 4=   0
FIELD 7=   6
FIELD12=   3
FIELD14=   8
FIELD22=   1
THE REST FIELDS ARE NOT USED
        RMASK1  TOS     3       EUBC    P2
FIELD12=   2
```

```
FIELD14=    7
FIELD15=    0
FIELD16=   15
FIELD17=    3
FIELD22=    1
THE REST FIELDS ARE NOT USED
        NOOP    XUPF    P.004
FIELD12=    1
THE REST FIELDS ARE NOT USED
        BRCH    L.004   P.004+1 1
FIELD12=    1
THE REST FIELDS ARE NOT USED
        MOVE7   256             B       P3
FIELD 4=    0
FIELD12=    3
FIELD14=    1
FIELD18=  256
FIELD20=    1
THE REST FIELDS ARE NOT USED
        SUB     R8      B       D       P2
FIELD 1=    8
FIELD 2=    1
FIELD 5=    0
FIELD 6=    6
FIELD 8=    8
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
        FLAG                            P1
FIELD 7=    3
FIELD12=    1
THE REST FIELDS ARE NOT USED
        PUSH3   PS              TOS     P3
FIELD 4=    0
FIELD 7=    6
FIELD12=    3
FIELD14=    8
FIELD22=    1
THE REST FIELDS ARE NOT USED
        RMASK1  TOS     3       EUBC    P2
FIELD12=    2
FIELD14=    7
FIELD15=    0
FIELD16=   15
FIELD17=    3
FIELD22=    1
THE REST FIELDS ARE NOT USED
        NOOP    XUPF    P.005
FIELD12=    1
THE REST FIELDS ARE NOT USED
        BRCH    L.005   P.005   1
FIELD12=    1
THE REST FIELDS ARE NOT USED
        MOVE7   16              B       P3
FIELD 4=    0
FIELD12=    3
FIELD14=    1
FIELD18=   16
FIELD20=    1
THE REST FIELDS ARE NOT USED
        ADD     R9      B       D       P2
FIELD 1=    9
FIELD 2=    1
```

```
 FIELD 5=    0
 FIELD 6=    9
 FIELD12=    2
 FIELD19=    1
 THE REST FIELDS ARE NOT USED
          MOVE5     D                             R7        P3
 FIELD 1=   10
 FIELD 2=    1
 FIELD 4=    2
 FIELD11=    3
 FIELD12=    3
 THE REST FIELDS ARE NOT USED
          MOVE3     R7                            D         P2
 FIELD 1=   10
 FIELD 2=    1
 FIELD 6=    0
 FIELD12=    2
 FIELD19=    1
 THE REST FIELDS ARE NOT USED
          MOVE5     D                             R9        P3
 FIELD 1=    9
 FIELD 2=    1
 FIELD 4=    2
 FIELD11=    3
 FIELD12=    3
 THE REST FIELDS ARE NOT USED
          MOVE2     R8                            BA        P1
 FIELD 1=    8
 FIELD 2=    1
 FIELD 3=    1
 FIELD10=    1
 FIELD12=    1
 THE REST FIELDS ARE NOT USED
          MOVE9     R7                            D         P2
 FIELD 1=   10
 FIELD 2=    1
 FIELD 6=    0
 FIELD 9=    5
 FIELD12=    2
 FIELD19=    1
 FIELD21=    1
 THE REST FIELDS ARE NOT USED
          NOOP
 FIELD12=    1
 THE REST FIELDS ARE NOT USED
 L.005    NOOP
 FIELD12=    1
 THE REST FIELDS ARE NOT USED
 L.004    MOVE3     R9                            D         P2
 FIELD 1=    9
 FIELD 2=    1
 FIELD 6=    0
 FIELD12=    2
 FIELD19=    1
 THE REST FIELDS ARE NOT USED
          MOVE5     D                             R8        P3
 FIELD 1=    8
 FIELD 2=    1
 FIELD 4=    2
 FIELD11=    3
 FIELD12=    3
 THE REST FIELDS ARE NOT USED
          MOVE12    MART                          BA        P1
```

```
FIELD 3=    1
FIELD10=    1
FIELD12=    1
FIELD14=    1
FIELD18= 3080
THE REST FIELDS ARE NOT USED
          MOVE9      R9                   D         P2
FIELD 1=    9
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
L.003     RETURN     RETADR               EUBC      P2
FIELD12=    2
FIELD14=    7
FIELD15=   15
FIELD16=   15
FIELD17=    0
FIELD22=    1
THE REST FIELDS ARE NOT USED
          NOOP1      XUPF      0
FIELD12=    1
THE REST FIELDS ARE NOT USED
PRI       PUSH                                      P1
FIELD12=    1
FIELD14=   11
THE REST FIELDS ARE NOT USED
          CALL       EFTADR                         P2
FIELD12=    2
FIELD14=    3
FIELD22=    1
THE REST FIELDS ARE NOT USED
          MOVE8      R8                   BA        P1
FIELD 1=    8
FIELD 2=    1
FIELD 3=    1
FIELD 9=    1
FIELD10=    1
FIELD12=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          MOVE4      UNIBUS               R10       P1
FIELD 1=    5
FIELD 2=    1
FIELD 4=    1
FIELD11=    3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE3      R10                  D         P2
FIELD 1=    5
FIELD 2=    1
FIELD 6=    0
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5      D                    R10       P3
FIELD 1=    5
```

```
       FIELD 2=     1
       FIELD 4=     2
       FIELD11=     3
       FIELD12=     3
       THE REST FIELDS ARE NOT USED
                 PUSH1      R11                  TOS       P2
       FIELD 1=     4
       FIELD 2=     1
       FIELD 4=     0
       FIELD12=     2
       FIELD14=     8
       FIELD22=     1
       THE REST FIELDS ARE NOT USED
                 RSMK       TOS       NEWCHARA01D          P2
       FIELD 6=     0
       FIELD12=     2
       FIELD14=     6
       FIELD15=    11
       FIELD16=    15
       FIELD17=     4
       FIELD19=     1
       FIELD22=     1
       THE REST FIELDS ARE NOT USED
                 MOVE5      D                    R9        P3
       FIELD 1=     9
       FIELD 2=     1
       FIELD 4=     2
       FIELD11=     3
       FIELD12=     3
       THE REST FIELDS ARE NOT USED
                 PUSH1      R9                   TOS       P2
       FIELD 1=     9
       FIELD 2=     1
       FIELD 4=     0
       FIELD12=     2
       FIELD14=     8
       FIELD22=     1
       THE REST FIELDS ARE NOT USED
                 LMASK1     TOS       2          EU3C      P2
       FIELD12=     2
       FIELD14=     7
       FIELD15=     1
       FIELD16=    15
       FIELD17=     0
       FIELD22=     1
       THE REST FIELDS ARE NOT USED
                 NOOP       XUPF      P.006
       FIELD12=     1
       THE REST FIELDS ARE NOT USED
                 UNJP       AND       P.006      2
       FIELD12=     1
       THE REST FIELDS ARE NOT USED
                 UNJP       TAD       P.006+01   2
       FIELD12=     1
       THE REST FIELDS ARE NOT USED
                 UNJP       ISZ       P.006+02   2
       FIELD12=     1
       THE REST FIELDS ARE NOT USED
       AND       MOVE1      R10                  9         P1
       FIELD 1=     5
       FIELD 2=     1
       FIELD 4=     0
       FIELD12=     1
```

```
FIELD20=    1
THE REST FIELDS ARE NOT USED
          AND        P15       B          D         P2
FIELD 1=    0
FIELD 2=    1
FIELD 5=    0
FIELD 6=   27
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5      D                    R15       P3
FIELD 1=    0
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE12     IR                   BA        P1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
FIELD14=    1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
          MOVE9      R12                  D         P2
FIELD 1=    3
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP       INF
FIELD12=    1
THE REST FIELDS ARE NOT USED
TAD       MOVE1      R10                  B         P1
FIELD 1=    5
FIELD 2=    1
FIELD 4=    0
FIELD12=    1
FIELD20=  ' 1
THE REST FIELDS ARE NOT USED
          ADD        R15       B          D         P2
FIELD 1=    0
FIELD 2=    1
FIELD 5=    0
FIELD 6=    9
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5      D                    R15       P3
FIELD 1=    0
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          FLAG                                      P1
FIELD 7=    3
```

```
FIELD12=    1
THE REST FIELDS ARE NOT USED
            PUSH3      PS                       TOS        P3
FIELD 4=    0
FIELD 7=    6
FIELD12=    3
FIELD14=    8
FIELD22=    1
THE REST FIFLDS ARE NOT USED
            RSMK       TOS      NEWCHARA020                P2
FIELD 6=    0
FIELD12=    2
FIELD14=    6
FIELD15=    0
FIELD16=   15
FIELD17=    0
FIELD19=    1
FIELD22=    1
THE REST FIELDS ARE NOT USED
            MOVE5      D                        R14        P3
FIELD 1=    1
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIFLDS ARE NOT USED
            PUSH1      R14                      TOS        P2
FIELD 1=    1
FIELD 2=    1
FIELD 4=    0
FIELD12=    2
FIELD14=    8
FIELD22=    1
THE REST FIFLDS ARE NOT USED
            LSMK       TOS      NEWCHARA03D                P2
FIELD 6=    0
FIELD12=    2
FIELD14=    6
FIELD15=   15
FIELD16=   11
FIELD17=   12
FIELD19=    1
FIELD22=    1
THE REST FIELDS ARE NOT USED
            MOVE5      D                        R14        P3
FIELD 1=    1
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
            MOVE12     IR                       BA         P1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
FIELD14=    1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
            MOVE9      R12                      D          P2
FIELD 1=    3
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
```

```
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP      INF
FIELD12=    1
THE REST FIELDS ARE NOT USED
ISZ       MOVE8     R8                  BA        P1
FIELD 1=    8
FIELD 2=    1
FIELD 3=    1
FIELD 9=    1
FIELD10=    1
FIELD12=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          MOVE4     UNIBUS              R9        P1
FIELD 1=    9
FIELD 2=    1
FIELD 4=    1
FIELD11=    3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE7     16                  8         P3
FIELD 4=    0
FIELD12=    3
FIELD14=    1
FIELD18=   16
FIELD20=    1
THE REST FIELDS ARE NOT USED
          ADD       R9        B         D         P2
FIELD 1=    9
FIELD 2=    1
FIELD 5=    0
FIELD 6=    9
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R9        P3
FIELD 1=    9
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE2     R8                  BA        P1
FIELD 1=    8
FIELD 2=    1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE9     R9                  D         P2
FIELD 1=    9
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
FIELD19=    1
FIELD21=    1
```

```
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE8     R8                            BA        P1
FIELD 1=    8
FIELD 2=    1
FIELD 3=    1
FIELD 9=    1
FIELD10=    1
FIELD12=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          MOVE4     UNIBUS                        R9        P1
FIELD 1=    9
FIELD 2=    1
FIELD 4=    1
FIELD11=    3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE7     0                             B         P3
FIELD 4=    0
FIELD12=    3
FIELD14=    1
FIELD18=    0
FIELD20=    1
THE REST FIELDS ARE NOT USED
          SUB       R9        B                   D         P2
FIELD 1=    9
FIELD 2=    1
FIELD 5=    0
FIELD 6=    6
FIELD 8=    8
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          FLAG                                              P1
FIELD 7=    3
FIELD12=    1
THE REST FIELDS ARE NOT USED
          PUSH3     FS                            TOS       P3
FIELD 4=    0
FIELD 7=    6
FIELD12=    3
FIELD14=    8
FIELD22=    1
THE REST FIELDS ARE NOT USED
          RMASK1    TOS       2                   EUBC      P2
FIELD12=    2
FIELD14=    7
FIELD15=    0
FIELD16=   15
FIELD17=    2
FIELD22=    1
THE REST FIELDS ARE NOT USED
          NOOP      XUPF      P.007
FIELD12=    1
THE REST FIELDS ARE NOT USED
          BRCH      L.006     P.007     1
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE7     16                            B         P3
FIELD 4=    0
```

```
FIELD12=    3
FIELD14=    1
FIELD18=   16
FIELD20=    1
THE REST FIELDS ARE NOT USED
          ADD       R13       B         D         P2
FIELD 1=    2
FIELD 2=    1
FIELD 5=    0
FIELD 6=    9
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R13       P3
FIELD 1=    2
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE12    IR                  BA        P1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
FIELD14=    1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
          MOVE9     R12                 D         P2
FIELD 1=    3
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
L.006     UNJP      INF
FIELD12=    1
THE REST FIELDS ARE NOT USED
DCA       PUSH                                    P1
FIELD12=    1
FIELD14=   11
THE REST FIELDS ARE NOT USED
          CALL      EFTADR                        P2
FIELD12=    2
FIELD14=    3
FIELD22=    1
THE REST FIELDS ARE NOT USED
          MOVE2     R8                  BA        P1
FIELD 1=    8
FIELD 2=    1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE9     R15                 D         P2
FIELD 1=    0
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
```

```
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
        NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
        CLR                    D       P2
FIELD 6=   19
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
        MOVE5   D              R15     P3
FIELD 1=    0
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
        MOVE12  IR             BA      P1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
FIELD14=    1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
        MOVE9   R12            D       P2
FIELD 1=    3
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
        NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
        UNJP    INF
FIELD12=    1
THE REST FIELDS ARE NOT USED
JMS     PUSH                           P1
FIELD12=    1
FIELD14=   11
THE REST FIELDS ARE NOT USED
        CALL    EFTADR                 P2
FIELD12=    2
FIELD14=    3
FIELD22=    1
THE REST FIELDS ARE NOT USED
        MOVE2   R8             BA      P1
FIELD 1=    8
FIELD 2=    1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
THE REST FIELDS ARE NOT USED
        MOVE9   R13            D       P2
FIELD 1=    2
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
```

```
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
          MOVE7     16                  B         P3
FIELD 4=    0
FIELD12=    3
FIELD14=    1
FIELD18=   16
FIELD20=    1
THE REST FIELDS ARE NOT USED
          ADD       R8        B         D         P2
FIELD 1=    8
FIELD 2=    1
FIELD 5=    0
FIELD 6=    9
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R8        P3
FIELD 1=    8
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE3     R8                  D         P2
FIELD 1=    8
FIELD 2=    1
FIELD 6=    0
FIELD12=    2
FIELD19=    1
THE REST FIELDS ARE NOT USED
          MOVE5     D                   R13       P3
FIELD 1=    2
FIELD 2=    1
FIELD 4=    2
FIELD11=    3
FIELD12=    3
THE REST FIELDS ARE NOT USED
          MOVE12    IR                  BA        P1
FIELD 3=    1
FIELD10=    1
FIELD12=    1
FIELD14=    1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
          MOVE9     R12                 D         P2
FIELD 1=    3
FIELD 2=    1
FIELD 6=    0
FIELD 9=    5
FIELD12=    2
FIELD19=    1
FIELD21=    1
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=    1
THE REST FIELDS ARE NOT USED
          UNJP      INF
FIELD12=    1
```

```
THE REST FIELDS ARE NOT USED
JMP     PUSH                                    P1
FIELD12=   1
FIELD14=  11
THE REST FIELDS ARE NOT USED
        CALL    EFTADR                          P2
FIELD12=   2
FIELD14=   3
FIELD22=   1
THE REST FIELDS ARE NOT USED
        MOVE3   R8                      D       P2
FIELD 1=   8
FIELD 2=   1
FIELD 6=   0
FIELD12=   2
FIELD19=   1
THE REST FIELDS ARE NOT USED
        MOVE5   D                       R13     P3
FIELD 1=   2
FIELD 2=   1
FIELD 4=   2
FIELD11=   3
FIELD12=   3
THE REST FIELDS ARE NOT USED
        MOVE12  IR                      BA      P1
FIELD 3=   1
FIELD10=   1
FIELD12=   1
FIELD14=   1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
        MOVE9   R12                     D       P2
FIELD 1=   3
FIELD 2=   1
FIELD 6=   0
FIELD 9=   5
FIELD12=   2
FIELD19=   1
FIELD21=   1
THE REST FIELDS ARE NOT USED
        NOOP
FIELD12=   1
THE REST FIELDS ARE NOT USED
        UNJP    INF
FIELD12=   1
THE REST FIELDS ARE NOT USED
IO      NOOP
FIELD12=   1
THE REST FIELDS ARE NOT USED
        MOVE12  IR                      BA      P1
FIELD 3=   1
FIELD10=   1
FIELD12=   1
FIELD14=   1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
        MOVE9   R12                     D       P2
FIELD 1=   3
FIELD 2=   1
FIELD 6=   0
FIELD 9=   5
FIELD12=   2
FIELD19=   1
```

```
FIELD21=     1
THE REST FIELDS ARE NOT USED
          NOOP
FIELD12=     1
THE REST FIELDS ARE NOT USED
          UNJP       INF
FIELD12=     1
THE REST FIELDS ARE NOT USED
OPT       NOOP1
FIELD12=     1
THE REST FIELDS ARE NOT USED
THE NUMBER OF CODES  191
```

APPENDIX E-5

Output of Pass 3

```
MOP     1    CNT ADDR 2000     ( 1279)    NEXT ADS      1278                    MOVE10    PC                  D

GROUP4          4000

GROUP3          6376

GROUP2        100400

GROUP1             0

GROUP0             0
MOP     2    CNT ADDR 2001     ( 1278)    NEXT ADS      1277                    MOVE5     D                   R13

GROUP4             0

GROUP3          2375

GROUP2        146000

GROUP1             0

GROUP0        100022
MOP     3    CNT ADDR 2002     ( 1277)    NEXT ADS      1276     INF            MOVE8     R13                 BA

GROUP4             0

GROUP3          2374

GROUP2         60220

GROUP1             0

GROUP0         20022
MOP     4    CNT ADDR 2003     ( 1276)    NEXT ADS      1275                    MOVE4     UNIBUS              R12

GROUP4             0

GROUP3          2373

GROUP2         46000

GROUP1             0

GROUP0         40023
MOP     5    CNT ADDR 2004     ( 1275)    NEXT ADS      1274                    CPTM

GROUP4             0

GROUP3          2372

GROUP2        146400

GROUP1             0

GROUP0        100023
MOP     6    CNT ADDR 2005     ( 1274)    NEXT ADS      1273                    MOVE7     16                  8

GROUP4            20

GROUP3          6371

GROUP2        141000
```

```
GROUP1             0

GROUP0             0
MOP    7    CNT ADDR 2006    ( 1273)    NEXT ADS     1272                  OPTM

GROUP4             0

GROUP3          2370

GROUP2        146400

GROUP1          4400

GROUP0        100022
MOP    8    CNT ADDR 2007    ( 1272)    NEXT ADS     1271     INSTDC      PUSH1      R12                TOS

GROUP4         20000

GROUP3         42367

GROUP2        100000

GROUP1             0

GROUP0            23
MOP    9    CNT ADDR 2010    ( 1271)    NEXT ADS     1270                 RSMK       TOS       OPCODE   D

GROUP4         24666

GROUP3         32366

GROUP2        100400

GROUP1             0

GROUP0             0
MOP   10    CNT ADDR 2011    ( 1270)    NEXT ADS     1269                 OPTM

GROUP4             0

GROUP3          2365

GROUP2        146000

GROUP1             0

GROUP0        100024
MOP   11    CNT ADDR 2012    ( 1269)    NEXT ADS     1268                 PUSH1      R11                TOS

GROUP4         20000

GROUP3         42364

GROUP2        100000

GROUP1             0

GROUP0            24
MOP   12    CNT ADDR 2013    ( 1268)    NEXT ADS     1267                 RSMK       TOS       NEWCHARA00D

GROUP4         22373
```

```
GROUP3       32363

GROUP2      100400

GROUP1           0

GROUP0           0
MOP   13    CNT ADDR 2014    ( 1267)    NEXT ADS     1266                  MOVE5     D                   R10

GROUP4           0

GROUP3        2362

GROUP2      146000

GROUP1           0

GROUP0      100025
MOP   14    CNT ADDR 2015    ( 1266)    NEXT ADS     1265                  PUSH1     R10                 TOS

GROUP4       20000

GROUP3       42361

GROUP2      100000

GROUP1           0

GROUP0          25
MOP   15    CNT ADDR 2016    ( 1265)    NEXT ADS     1264                  LMASK1    TOS       3         EUBC

GROUP4       20362

GROUP3       36360

GROUP2      100000

GROUP1           0

GROUP0           0
MOP   16    CNT ADDR 2017    ( 1264)    NEXT ADS     1791                  NOOP      XUPF      P.001

GROUP4           0

GROUP3        3377

GROUP2       40000

GROUP1           0

GROUP0           0
MOP   25    CNT ADDR 2020    ( 1263)    NEXT ADS     1262    EFTADR        PUSH1     R12                 TOS

GROUP4       20000

GROUP3       42356

GROUP2      100000

GROUP1           0

GROUP0          23
MOP   26    CNT ADDR 2021    ( 1262)    NEXT ADS     1261                  PSMK      TOS       PGEADR    D
```

```
GROUP4         20272

GROUP3         32355

GROUP2        100400

GROUP1             0

GROUP0             0
MOP   27    CNT ADDR 2022    ( 1261)    NEXT ADS      1260                OPTM

GROUP4             0

GROUP3          2354

GROUP2        146000

GROUP1             0

GROUP0        100025
MOP   28    CNT ADDR 2023    ( 1260)    NEXT ADS      1259                PUSH1     R12                 TOS

GROUP4         20000

GROUP3         42353

GROUP2        100000

GROUP1             0

GROUP0            23
MOP   29    CNT ADDR 2024    ( 1259)    NEXT ADS      1258                RMASK1    TOS       11        EUBC

GROUP4         25760

GROUP3         36352

GROUP2        100000

GROUP1             0

GROUP0             0
MOP   30    CNT ADDR 2025    ( 1258)    NEXT ADS      1783                NOOP      XUPF      P.002

GROUP4             0

GROUP3          3367

GROUP2         40000

GROUP1             0

GROUP0             0
MOP   32    CNT ADDR 3011    ( 1782)    NEXT ADS      1256                MOVE7     16                  B

GROUP4            20

GROUP3          6350

GROUP2        141000

GROUP1             0
```

```
GROUP0             0
MOP   33    CNT ADDR 2027    ( 1256)     NEXT ADS     1255              SUB      R13      B       D

GROUP4             0

GROUP3          2347

GROUP2        100410

GROUP1          3000

GROUP0            22
MOP   34    CNT ADDR 2030    ( 1255)     NEXT ADS     1254              MOVE5    D                R9

GROUP4             0

GROUP3          2346

GROUP2        146000

GROUP1             0

GROUP0        100031
MOP   35    CNT ADDR 2031    ( 1254)     NEXT ADS     1253              PUSH1    R9               TOS

GROUP4         20000

GROUP3         42345

GROUP2        100000

GROUP1             0

GROUP0            31
MOP   36    CNT ADDR 2032    ( 1253)     NEXT ADS     1252              RSMK     TOS      CRNTPG  D

GROUP4         20117

GROUP3         32344

GROUP2        100400

GROUP1             0

GROUP0             0
MOP   37    CNT ADDR 2033    ( 1252)     NEXT ADS     1251              OPTM

GROUP4             0

GROUP3          2343

GROUP2        146000

GROUP1             0

GROUP0        100030
MOP   38    CNT ADDR 2034    ( 1251)     NEXT ADS     1250              MOVE5    D                R9

GROUP4             0

GROUP3          2342
```

```
GROUP2      146000

GROUP1           0

GROUP0      100031
MOP    39    CNT ADDR 2035     ( 1250)     NEXT ADS      1249                 MOVE1     R10                      B

GROUP4           0

GROUP3        2341

GROUP2       41000

GROUP1           0

GROUP0          25
MOP    40    CNT ADDR 2036     ( 1249)     NEXT ADS      1248                 OR        R9        3              D

GROUP4           0

GROUP3        2340

GROUP2      100400

GROUP1       17000

GROUP0          31
MOP    41    CNT ADDR 2037     ( 1248)     NEXT ADS      1247                 MOVE5     D                        R8

GROUP4           0

GROUP3        2337

GROUP2      146000

GROUP1           0

GROUP0      100030
MOP    42    CNT ADDR 2040     ( 1247)     NEXT ADS      1246                 MOVE12    PCTEMP                   BA

GROUP4        6007

GROUP3        6336

GROUP2       40200

GROUP1           0

GROUP0       20000
MOP    43    CNT ADDR 2041     ( 1246)     NEXT ADS      1245                 MOVE9     R9                       D

GROUP4           0

GROUP3        2335

GROUP2      120520

GROUP1           0

GROUP0          31
MOP    44    CNT ADDR 2042     ( 1245)     NEXT ADS      1241                 NOOP

GROUP4           0
```

```
    GROUP3          2331

    GROUP2         40000

    GROUP1             0

    GROUP0             0
    MOP   46    CNT ADDR 3010    ( 1783)    NEXT ADS    1242    L.001    MOVE3    R10                D

    GROUP4             0

    GROUP3          2332

    GROUP2        100400

    GROUP1             0

    GROUP0            25
    MOP   47    CNT ADDR 2045    ( 1242)    NEXT ADS    1241             MOVE5    D                  R8

    GROUP4             0

    GROUP3          2331

    GROUP2        146300

    GROUP1             0

    GROUP0        100030
    MOP   48    CNT ADDR 2046    ( 1241)    NEXT ADS    1240    L.002    PUSH1    R12                TOS

    GROUP4         20000

    GROUP3         42330

    GROUP2        100000

    GROUP1             0

    GROUP0            23
    MOP   49    CNT ADDR 2047    ( 1240)    NEXT ADS    1239             RMASK1   TOS      12        EUBC

    GROUP4         26360

    GROUP3         36327

    GROUP2        100000

    GROUP1             0

    GROUP0             0
    MOP   50    CNT ADDR 2050    ( 1239)    NEXT ADS    1781             NOOP     XUPF     P.003

    GROUP4             0

    GROUP3          3365

    GROUP2         40000

    GROUP1             0

    GROUP0             0
```

```
MOP    52     CNT ADDR 3013      ( 1780)      NEXT ADS        1237                      MOVE8      R8                                BA

GROUP4                0

GROUP3             2325

GROUP2            60220

GROUP1                0

GROUP0            20030
MOP    53     CNT ADDR 2052      ( 1237)      NEXT ADS        1236                      MOVE4      UNIBUS                            R9

GROUP4                0

GROUP3             2324

GROUP2            46000

GROUP1                0

GROUP0            40031
MOP    54     CNT ADDR 2053      ( 1236)      NEXT ADS        1235                      CPTM

GROUP4                0

GROUP3             2323

GROUP2           146400

GROUP1                0

GROUP0           100031
MOP    55     CNT ADDR 2054      ( 1235)      NEXT ADS        1234                      MOVE7      128                               B

GROUP4              200

GROUP3             6322

GROUP2           141000

GROUP1                0

GROUP0                0
MOP    56     CNT ADDR 2055      ( 1234)      NEXT ADS        1233                      SUB        R8          B                     D

GROUP4                0

GROUP3             2321

GROUP2           100410

GROUP1             3000

GROUP0               30
MOP    57     CNT ADDR 2056      ( 1233)      NEXT ADS        1232                      FLAG

GROUP4                0

GROUP3             2320

GROUP2            40000
```

```
GROUP1         60000

GROUP0             0
MOP    58    CNT ADDR 2057     ( 1232)     NEXT ADS      1231                 PUSH3      PS                   TOS

GROUP4         20000

GROUP3         42317

GROUP2        140000

GROUP1        140000

GROUP0             0
MOP    59    CNT ADDR 2060     ( 1231)     NEXT ADS      1230                 FMASK1     TOS        3         EUEC

GROUP4         21760

GROUP3         36316

GROUP2        100000

GROUP1             0

GROUP0             0
MOP    60    CNT ADDR 2061     ( 1230)     NEXT ADS      1779                 NOOP       XUPF       P.004

GROUP4             0

GROUP3          3363

GROUP2         40000

GROUP1             0

GROUP0             0
MOP    62    CNT ADDR 3014     ( 1779)     NEXT ADS      1228                 MOVE7      256                  B

GROUP4           400

GROUP3          6314

GROUP2        141000

GROUP1             0

GROUP0             0
MOP    63    CNT ADDR 2063     ( 1228)     NEXT ADS      1227                 SUB        R8         B         D

GROUP4             0

GROUP3          2313

GROUP2        100410

GROUP1          3000

GROUP0            30
MOP    64    CNT ADDR 2064     ( 1227)     NEXT ADS      1226                 FLAG

GROUP4             0

GROUP3          2312
```

```
GROUP2        40000

GROUP1        60000

GROUP0            0
MOP   65    CNT ADDR 2065    ( 1226)    NEXT ADS    1225             PUSH3    PS                TOS

GROUP4        20000

GROUP3        42311

GROUP2       140000

GROUP1       140000

GROUP0            0
MOP   66    CNT ADDR 2066    ( 1225)    NEXT ADS    1224             FMASK1   TOS      3        EUBC

GROUP4        21760

GROUP3        36310

GROUP2       100000

GROUP1            0

GROUP0            0
MOP   67    CNT ADDR 2067    ( 1224)    NEXT ADS    1777             NOOP     XUPF     P.005

GROUP4            0

GROUP3         3361

GROUP2        40000

GROUP1            0

GROUP0            0
MOP   69    CNT ADDR 3017    ( 1776)    NEXT ADS    1222             MOVEZ    16                B

GROUP4           20

GROUP3         6306

GROUP2       141000

GROUP1            0

GROUP0            0
MOP   70    CNT ADDR 2071    ( 1222)    NEXT ADS    1221             ADD      R9       B        D

GROUP4            0

GROUP3         2305

GROUP2       100400

GROUP1         4400

GROUP0           31
MOP   71    CNT ADDR 2072    ( 1221)    NEXT ADS    1220             OPTM
```

```
GROUP4             0

GROUP3          2304

GROUP2        146000

GROUP1             0

GROUP0        100032
MOP   72    CNT ADDR 2073      ( 1220)     NEXT ADS      1219                   MOVE5      D                   R9

GROUP4             0

GROUP3          2303

GROUP2        146000

GROUP1             0

GROUP0        100031
MOP   73    CNT ADDR 2074      ( 1219)     NEXT ADS      1218                   MOVE2      R8                  BA

GROUP4             0

GROUP3          2302

GROUP2         40200

GROUP1             0

GROUP0         20030
MOP   74    CNT ADDR 2075      ( 1218)     NEXT ADS      1217                   MOVE9      R7                  D

GROUP4             0

GROUP3          2301

GROUP2        120520

GROUP1             0

GROUP0            32
MOP   75    CNT ADDR 2076      ( 1217)     NEXT ADS      1777                   NOOP

GROUP4             0

GROUP3          3361

GROUP2         40000

GROUP1             0

GROUP0             0
MOP   76    CNT ADDR 3016      ( 1777)     NEXT ADS      1778     L.005         NOOP

GROUP4             0

GROUP3          3362

GROUP2         40000

GROUP1             0
```

```
GROUP0            0
MOP   77    CNT ADDR 3015    ( 1778)     NEXT ADS      1214     L.004      MOVE3     R9                   D

GROUP4            0

GROUP3         2276

GROUP2       100400

GROUP1            0

GROUP0           31
MOP   78    CNT ADDR 2101    ( 1214)     NEXT ADS      1213                MOVE5     D                    R8

GROUP4            0

GROUP3         2275

GROUP2       146000

GROUP1            0

GROUP0       100030
MOP   79    CNT ADDR 2102    ( 1213)     NEXT ADS      1212                MOVE12    MART                 BA

GROUP4         6010

GROUP3         6274

GROUP2        40200

GROUP1            0

GROUP0        20000
MOP   80    CNT ADDR 2103    ( 1212)     NEXT ADS      1211                MOVE9     R9                   D

GROUP4            0

GROUP3         2273

GROUP2       120520

GROUP1            0

GROUP0           31
MOP   81    CNT ADDR 2104    ( 1211)     NEXT ADS      1781                NOOP

GROUP4            0

GROUP3         3365

GROUP2        40000

GROUP1            0

GROUP0            0
MOP   82    CNT ADDR 3012    ( 1781)     NEXT ADS      1209     L.003      RETURN    RETADR               EUBC

GROUP4        20377

GROUP3        36271

GROUP2       100000
```

```
GROUP1          0

GROUP0          0
MOP   83    CNT ADDR 2106    ( 1209)    NEXT ADS      255                   NOOP1     XUPF      0

GROUP4          0

GROUP3        377

GROUP2      40000

GROUP1          0

GROUP0          0
MOP   84    CNT ADDR 3000    ( 1791)    NEXT ADS      1207    MRI       PUSH

GROUP4          0

GROUP3      56267

GROUP2      40000

GROUP1          0

GROUP0          0
MOP   85    CNT ADDR 2110    ( 1207)    NEXT ADS      1263                  CALL      EFTADR

GROUP4       2111

GROUP3      16357

GROUP2     100000

GROUP1          0

GROUP0          0
MOP   86    CNT ADDR 2111    ( 1206)    NEXT ADS      1205                  MOVE8     R8                  BA

GROUP4          0

GROUP3       2265

GROUP2      60220

GROUP1          0

GROUP0      20030
MOP   87    CNT ADDR 2112    ( 1205)    NEXT ADS      1204                  MOVE4     UNIBUS              R10

GROUP4          0

GROUP3       2264

GROUP2      46000

GROUP1          0

GROUP0      40025
MOP   88    CNT ADDR 2113    ( 1204)    NEXT ADS      1203                  OPTM

GROUP4          0
```

```
GROUP3          2263

GROUP2        146400

GROUP1             0

GROUP0        100025
MOP   89    CNT ADDR 2114    ( 1203)    NEXT ADS      1202             PUSH1     R11                 TOS

GROUP4         20000

GROUP3         42262

GROUP2        100000

GROUP1             0

GROUP0            24
MOP   90    CNT ADDR 2115    ( 1202)    NEXT ADS      1201             RSMK      TOS       NEWCHARA010

GROUP4         22373

GROUP3         32261

GROUP2        100400

GROUP1             0

GROUP0             0
MOP   91    CNT ADDR 2116    ( 1201)    NEXT ADS      1200             MOVE5     0                   R9

GROUP4             0

GROUP3          2260

GROUP2        146000

GROUP1             0

GROUP0        100031
MOP   92    CNT ADDR 2117    ( 1200)    NEXT ADS      1199             PUSH1     R9                  TOS

GROUP4         20000

GROUP3         42257

GROUP2        100000

GROUP1             0

GROUP0            31
MOP   93    CNT ADDR 2120    ( 1199)    NEXT ADS      1198             LMASK1    TOS       2         EUBC

GROUP4         20361

GROUP3         36256

GROUP2        100000

GROUP1             0

GROUP0             0
MOP   94    CNT ADDR 2121    ( 1198)    NEXT ADS      1775             NOOP      XUPF      P.006
```

```
GROUP4             0

GROUP3          3357

GROUP2         40000

GROUP1             0

GROUP0             0
MOP   98    CNT ADDR 3020    ( 1775)    NEXT ADS     1196    AND       MOVE1     R10                B

GROUP4             0

GROUP3          2254

GROUP2         41000

GROUP1             0

GROUP0            25
MOP   99    CNT ADDR 2123    ( 1196)    NEXT ADS     1195                CPTM

GROUP4             0

GROUP3          2253

GROUP2        146400

GROUP1         15400

GROUP0        100020
MOP  100    CNT ADDR 2124    ( 1195)    NEXT ADS     1194                MOVE12    IR                 BA

GROUP4          6002

GROUP3          6252

GROUP2         40200

GROUP1             0

GROUP0         20000
MOP  101    CNT ADDR 2125    ( 1194)    NEXT ADS     1193                MOVE9     R12                D

GROUP4             0

GROUP3          2251

GROUP2        120520

GROUP1             0

GROUP0            23
MOP  102    CNT ADDR 2126    ( 1193)    NEXT ADS     1277                NOOP

GROUP4             0

GROUP3          2375

GROUP2         40000

GROUP1             0
```

```
GROUP0              0
MOP  104     CNT ADDR 3021      ( 1774)     NEXT ADS       1190     TAD          MOVE1      R10                                B

GROUP4              0

GROUP3           2246

GROUP2          41000

GROUP1              0

GROUP0             25
MOP  105     CNT ADDR 2131      ( 1190)     NEXT ADS       1189                  OPTM

GROUP4              0

GROUP3           2245

GROUP2         146400

GROUP1           4400

GROUP0         100020
MOP  106     CNT ADDR 2132      ( 1189)     NEXT ADS       1188                  FLAG

GROUP4              0

GROUP3           2244

GROUP2          40000

GROUP1          60000

GROUP0              0
MOP  107     CNT ADDR 2133      ( 1188)     NEXT ADS       1187                  PUSH3      PS                                 TOS

GROUP4          20000

GROUP3          42243

GROUP2         140000

GROUP1         140000

GROUP0              0
MOP  108     CNT ADDR 2134      ( 1187)     NEXT ADS       1186                  RSMK       TOS         NEWCHARA02D

GROUP4          20360

GROUP3          32242

GROUP2         100400

GROUP1              0

GROUP0              0
MOP  109     CNT ADDR 2135      ( 1186)     NEXT ADS       1185                  MOVE5      D                                  R14

GROUP4              0

GROUP3           2241
```

```
GROUP2        146000

GROUP1             0

GROUP0        100021
MOP  110     CNT ADDR 2136     ( 1185)     NEXT ADS     1184                 PUSH1      R14                  TOS

GROUP4         20000

GROUP3         42240

GROUP2        100000

GROUP1             0

GROUP0            21
MOP  111     CNT ADDR 2137     ( 1184)     NEXT ADS     1183                 LSMK       TOS        NEWCHARA03D

GROUP4         26277

GROUP3         32237

GROUP2        100400

GROUP1             0

GROUP0             0
MOP  112     CNT ADDR 2140     ( 1183)     NEXT ADS     1182                 MOVE5      D                    R14

GROUP4             0

GROUP3          2236

GROUP2        146000

GROUP1             0

GROUP0        100021
MOP  113     CNT ADDR 2141     ( 1182)     NEXT ADS     1181                 MOVE12     IR                   BA

GROUP4          6002

GROUP3          6235

GROUP2         40200

GROUP1             0

GROUP0         20000
MOP  114     CNT ADDR 2142     ( 1181)     NEXT ADS     1180                 MOVE9      R12                  D

GROUP4             0

GROUP3          2234

GROUP2        120520

GROUP1             0

GROUP0            23
MOP  115     CNT ADDR 2143     ( 1180)     NEXT ADS     1277                 NOOP

GROUP4             0
```

```
 GROUP3          2375

 GROUP2         40000

 GROUP1             0

 GROUP0             0
MOP  117     CNT ADDR 3022     ( 1773)     NEXT ADS      1177    ISZ       MOVE8     R8                  BA

 GROUP4             0

 GROUP3          2231

 GROUP2         60220

 GROUP1             0

 GROUP0         20030
MOP  118     CNT ADDR 2146     ( 1177)     NEXT ADS      1176              MOVE4     UNIBUS              R9

 GROUP4             0

 GROUP3          2230

 GROUP2         46000

 GROUP1             0

 GROUP0         40031
MOP  119     CNT ADDR 2147     ( 1176)     NEXT ADS      1175              MOVE7     16                  B

 GROUP4            20

 GROUP3          6227

 GROUP2        141000

 GROUP1             0

 GROUP0             0
MOP  120     CNT ADDR 2150     ( 1175)     NEXT ADS      1174              OPTM

 GROUP4             0

 GROUP3          2226

 GROUP2        146400

 GROUP1          4400

 GROUP0        100031
MOP  121     CNT ADDR 2151     ( 1174)     NEXT ADS      1173              MOVE2     R8                  BA

 GROUP4             0

 GROUP3          2225

 GROUP2         40200

 GROUP1             0

 GROUP0         20030
```

```
MOP  122     CNT ADDR 2152     ( 1173)      NEXT ADS       1172                   MOVE9      R9                    D

GROUP4               0

GROUP3            2224

GROUP2          120520

GROUP1               0

GROUP0              31
MOP  123     CNT ADDR 2153     ( 1172)      NEXT ADS       1171                   NOOP

GROUP4               0

GROUP3            2223

GROUP2           40000

GROUP1               0

GROUP0               0
MOP  124     CNT ADDR 2154     ( 1171)      NEXT ADS       1170                   MOVE8      R8                    BA

GROUP4               0

GROUP3            2222

GROUP2           60220

GROUP1               0

GROUP0           20030
MOP  125     CNT ACDR 2155     ( 1170)      NEXT ADS       1169                   MOVE4      UNIEUS                R9

GROUP4               0

GROUP3            2221

GROUP2           46000

GROUP1               0

GROUP0           40031
MOP  126     CNT ACOF 2156     ( 1169)      NEXT ADS       1168                   MOVE7      0                     B

GROUP4               0

GROUP3            6220

GROUP2          141000

GROUP1               0

GROUP0               0
MOP  127     CNT ACDF 2157     ( 1168)      NEXT ADS       1167                   SUB        R9         B          D

GROUP4               0

GROUP3            2217

GROUP2          100410
```

```
GROUP1             3000

GROUP0               31
MOP  128    CNT ACCR 2160     ( 1167)     NEXT ADS      1166                 FLAG

GROUP4                0

GROUP3             2216

GROUP2            40000

GROUP1            60000

GROUP0                0
MOP  129    CNT ACCR 2161     ( 1166)     NEXT ADS      1165                 FUSH3     PS                       TOS

GROUP4            20000

GROUP3            42215

GROUP2           140000

GROUP1           140000

GROUP0                0
MOP  130    CNT ACCR 2162     ( 1165)     NEXT ADS      1164                 FMASK1    TOS        2             EUBC

GROUP4            21360

GROUP3            36214

GROUP2           100000

GROUP1                0

GROUP0                0
MOP  131    CNT ACDF 2163     ( 1164)     NEXT ADS      1767                 NOOP      XUPF       P.007

GROUP4                0

GROUP3             3347

GROUP2            40000

GROUP1                0

GROUP0                0
MOP  133    CNT ADDR 3031     ( 1766)     NEXT ADS      1162                 MOVE7     16                       B

GROUP4               20

GROUP3             6212

GROUP2           141000

GROUP1                0

GROUP0                0
MOP  134    CNT ACDR 2165     ( 1162)     NEXT ADS      1767                 OPTM

GROUP4                0

GROUP3             3347
```

```
GROUP2         146400

GROUP1           4400

GROUP0         108022
MOP  135    CNT ADDR 3030     ( 1767)     NEXT ADS      1160     L.006      MOVE12     IR                        BA

GROUP4           6002

GROUP3           6210

GROUP2          40200

GROUP1              0

GROUP0          20000
MOP  136    CNT ADDR 2167     ( 1160)     NEXT ADS      1159                MOVE9      R12                       0

GROUP4              0

GROUP3           2207

GROUP2         120520

GROUP1              0

GROUP0             23
MOP  137    CNT ADDR 2170     ( 1159)     NEXT ADS      1277                NOOP

GROUP4              0

GROUP3           2375

GROUP2          40000

GROUP1              0

GROUP0              0
MOP  139    CNT ADDR 3003     ( 1788)     NEXT ADS      1156     DCA        PUSH

GROUP4              0

GROUP3          56204

GROUP2          40000

GROUP1              0

GROUP0              0
MOP  140    CNT ADDR 2173     ( 1156)     NEXT ADS      1263                CALL       EFTADR

GROUP4           2174

GROUP3          16357

GROUP2         100000

GROUP1              0

GROUP0              0
MOP  141    CNT ADDR 2174     ( 1155)     NEXT ADS      1154                MOVE2      R8                        BA
```

```
GROUP4             0

GROUP3          2202

GROUP2         40200

GROUP1             0

GROUP0         20030
MOP  142   CNT ADDR 2175    ( 1154)    NEXT ADS     1153                 MOVE9     R15                  0

GROUP4             0

GROUP3          2201

GROUP2        120520

GROUP1             0

GROUP0            20
MOP  143   CNT ADDR 2176    ( 1153)    NEXT ADS     1152                 NOOP

GROUP4             0

GROUP3          2200

GROUP2         40000

GROUP1             0

GROUP0             0
MOP  144   CNT ADDR 2177    ( 1152)    NEXT ADS     1151                 OPTM

GROUP4             0

GROUP3          2177

GROUP2        146400

GROUP1         11400

GROUP0        100020
MOP  145   CNT ADDR 2200    ( 1151)    NEXT ADS     1150                 MOVE12    IR                   BA

GROUP4          6002

GROUP3          6176

GROUP2         40200

GROUP1             0

GROUP0         20000
MOP  146   CNT ADDR 2201    ( 1150)    NEXT ADS     1149                 MOVE9     R12                  0

GROUP4             0

GROUP3          2175

GROUP2        120520

GROUP1             0
```

```
GROUP0            23
MOP  147    CNT ADDR 2202     ( 1149)     NEXT ADS      1277                NOOP

GROUP4             0

GROUP3          2375

GROUP2         40000

GROUP1             0

GROUP0             0
MOP  149    CNT ADDR 3004     ( 1787)     NEXT ADS      1146     JMS        PUSH

GROUP4             0

GROUP3         56172

GROUP2         40000

GROUP1             0

GROUP0             0
MOP  150    CNT ADDR 2205     ( 1146)     NEXT ADS      1263                CALL       EFTADR

GROUP4          2206

GROUP3         16357

GROUP2        100000

GROUP1             0

GROUP0             0
MOP  151    CNT ADDR 2206     ( 1145)     NEXT ADS      1144                MOVE2      R8                    8A

GROUP4             0

GROUP3          2170

GROUP2         40200

GROUP1             0

GROUP0         20030
MOP  152    CNT ADDR 2207     ( 1144)     NEXT ADS      1143                MOVE9      R13                   0

GROUP4             0

GROUP3          2167

GROUP2        120520

GROUP1             0

GROUP0            22
MOP  153    CNT ADDR 2210     ( 1143)     NEXT ADS      1142                NOOP

GROUP4             0

GROUP3          2166

GROUP2         40000
```

```
GROUP1           0

GROUP0           0
MOP  154    CNT ADDR 2211    ( 1142)    NEXT ADS     1141              MOVE7    16               B

GROUP4          20

GROUP3        6165

GROUP2      141000

GROUP1           0

GROUP0           0
MOP  155    CNT ADDR 2212    ( 1141)    NEXT ADS     1140              OPTM

GROUP4           0

GROUP3        2164

GROUP2      146400

GROUP1        4400

GROUP0      100030
MOP  156    CNT ADDR 2213    ( 1140)    NEXT ADS     1139              MOVE3    R8               D

GROUP4           0

GROUP3        2163

GROUP2      100400

GROUP1           0

GROUP0          30
MOP  157    CNT ADDR 2214    ( 1139)    NEXT ADS     1138              MOVE5    D                R13

GROUP4           0

GROUP3        2162

GROUP2      146000

GROUP1           0

GROUP0      100022
MOP  158    CNT ADDR 2215    ( 1138)    NEXT ADS     1137              MOVE12   IR               BA

GROUP4        6002

GROUP3        6161

GROUP2       40200

GROUP1           0

GROUP0       20000
MOP  159    CNT ADDR 2216    ( 1137)    NEXT ADS     1136              MOVE9    R12              D

GROUP4           0
```

```
GROUP3          2160

GROUP2        120520

GROUP1             0

GROUP0            23
MOP  160    CNT ADDR 2217    ( 1136)    NEXT ADS     1277              NOOP

GROUP4             0

GROUP3          2375

GROUP2         40000

GROUP1             0

GROUP0             0
MOP  162    CNT ADDR 3005    ( 1786)    NEXT ADS     1133    JMP       PUSH

GROUP4             0

GROUP3         56155

GROUP2         40000

GROUP1             0

GROUP0             0
MOP  163    CNT ADDR 2222    ( 1133)    NEXT ADS     1263              CALL      EFTADR

GROUP4          2223

GROUP3         16357

GROUP2        100000

GROUP1             0

GROUP0             0
MOP  164    CNT ADDR 2223    ( 1132)    NEXT ADS     1131              MOVE3     R8                  D

GROUP4             0

GROUP3          2153

GROUP2        100400

GROUP1             0

GROUP0            30
MOP  165    CNT ADDR 2224    ( 1131)    NEXT ADS     1130              MOVE5     D                   R13

GROUP4             0

GROUP3          2152

GROUP2        146000

GROUP1             0

GROUP0        100022
MOP  166    CNT ADDR 2225    ( 1130)    NEXT ADS     1129              MOVE12    IR                  BA
```

```
GROUP4          6002

GROUP3          6151

GROUP2         40200

GROUP1             0

GROUP0         20000
MOP  167    CNT ADDR 2226    ( 1129)    NEXT ADS     1128               MOVE9     R12               D

GROUP4             0

GROUP3          2150

GROUP2        120520

GROUP1             0

GROUP0            23
MOP  168    CNT ADDR 2227    ( 1128)    NEXT ADS     1277               NOOP

GROUP4             0

GROUP3          2375

GROUP2         40000

GROUP1             0

GROUP0             0
MOP  170    CNT ADDR 3006    ( 1785)    NEXT ADS     1125     IC        NOOP

GROUP4             0

GROUP3          2145

GROUP2         40000

GROUP1             0

GROUP0             0
MOP  171    CNT ADDR 2232    ( 1125)    NEXT ADS     1124               MOVE12    IR                BA

GROUP4          6002

GROUP3          6144

GROUP2         40200

GROUP1             0

GROUP0         20000
MOP  172    CNT ADDR 2233    ( 1124)    NEXT ADS     1123               MOVE9     R12               D

GROUP4             0

GROUP3          2143

GROUP2        120520

GROUP1             0
```

```
GROUP0              23
MOP  173    CNT ADDR 2234     ( 1123)     NEXT ADS      1277                 NOOP

GROUP4               0

GROUP3            2375

GROUP2           40000

GROUP1               0

GROUP0               0
MOP  175    CNT ADDR 3007     ( 1784)     NEXT ADS       255    OPT          NOOP1

GROUP4               0

GROUP3             377

GROUP2           40000

GROUP1               0

GROUP0               0
MOP  176    CNT ADDR 3001     ( 1790)     NEXT ADS      1791                 NOOP

GROUP4               0

GROUP3            3377

GROUP2           40000

GROUP1               0

GROUP0               0
MOP  177    CNT ADDR 3002     ( 1789)     NEXT ADS      1791                 NOOP

GROUP4               0

GROUP3            3377

GROUP2           40000

GROUP1               0

GROUP0               0
TOTAL SUM IS         153
```

APPENDIX E-6

PDP8 Benchmarks and Test Run

1) Benchmark 1 — PDPT2

```
Address                Code              Comment

 200              AND    215        /clear ACCM
 201              TAD    212
 202              DCA    213        /set counter to -64
 203              TAD    214
 204              DCA    10         /set adr(10) to 1777
 205              TAD    10
 206              DCA I  10         /used as an autoindex register
 207              ISZ    213         check counter
 210              JMP    .-3        /loop
 211              HALT

 212              7700
 213              0
 214              1777
 215              0
```

The output is as follows:

```
  Address          Contents
   2000              1777
   2001              2000
    .
    .                  .
    .                  .
                       .
   2076              2075
   2077              2076
```

```
.R MACRO
*,TT:=FDP8T2
.MAIN.  RT-11 MACRO VM02-12   25-MAY-78 01:26:34 PAGE 1


1              000000         .ASECT
2              000200         .=200
3 000200 000000         .WORD   0
4              004000         .=4000
5 004000 004320         .WORD   215*20
6              004020         .=4020
7 004020 024240         .WORD   1212*20
8              004040         .=4040
9 004040 064260         .WORD   3213*20
10             004060         .=4060
11 04060 024300         .WORD   1214*20
12             004100         .=4100
13 04100 060200         .WORD   3010*20
14             004120         .=4120
15 04120 020200         .WORD   1010*20
16             004140         .=4140
17 04140 070200         .WORD   3410*20
18             004160         .=4160
19 04160 044260         .WORD   2213*20
20             004200         .=4200
21 04200 124120         .WORD   5205*20
22             004220         .=4220
23 04220 170040         .WORD   7402*20
24             004240         .=4240
25 04240 176000         .WORD   7700*20
26             004260         .=4260
27 04260 000000         .WORD   0
28             004300         .=4300
29 04300 037760         .WORD   1777*20
30             004320         .=4320
31 04320 000000         .WORD   0
32             040000         .=40000
33             000001         .END
.MAIN.  RT-11 MACRO VM02-12   25-MAY-78 01:26:34 PAGE 1+
SYMBOL TABLE


. ABS.  040000     000
        000000     001
ERRORS DETECTED: 0
FREE CORE: 18439. WORDS

,TT:=FDP8T2

ERRORS DETECTED: 0
FREE CORE: 18439. WORDS

*
```

```
GET PDP012

.RU SMALL
040000: 037760
040020: 040000
040040: 040020
040060: 040040
040100: 040060
040120: 040100
040140: 040120
040160: 040140
040200: 040160
040220: 040200
040240: 040220
040260: 040240
040300: 040260
040320: 040300
040340: 040320
040360: 040340
040400: 040360
040420: 040400
040440: 040420
040460: 040440
040500: 040460
040520: 040500
040540: 040520
040560: 040540
040600: 040560
040620: 040600
040640: 040620
040660: 040640
040700: 040660
040720: 040700
040740: 040720
040760: 040740
041000: 040760
041020: 041000
041040: 041020
041060: 041040
041100: 041060
041120: 041100
041140: 041120
041160: 041140
041200: 041160
041220: 041200
041240: 041220
041260: 041240
041300: 041260
041320: 041300
041340: 041320
041360: 041340
041400: 041360
041420: 041400
041440: 041420
041460: 041440
041500: 041460
041520: 041500
041540: 041520
041560: 041540
041600: 041560
041620: 041600
041640: 041620
041660: 041640
041700: 041660
```

2) Benchmark 2 — PDPT3

```
    Address            Code              Comment

     200          AND        215      /clear ACCM
     201          TAD        212
     202          DCA        213      /set counter to -64
     203          TAD        214
     204          DCA        216      /set adr(216) to 2000
     205          DCA   I    216      /clear adr(2000) to zero
     206          ISZ        216      /increment adr(216)
     207          ISZ        213      /check counter
     210          JMP        ..3      /loop
     211          HALT

     212          7700
     213             0
     214          2000
     215             0
     216             0
```

The output is as follows:

```
     2000            0
      .
      .
      .
     2077            0
```

```
 ,TT:=PDP8T3
.MAIN.  RT-11 MACRO VM02-12   25-MAY-78 01:27:44 PAGE 1


 1         000000              .ASECT
 2         000200              .=200
 3 000200  000000              .WORD   0
 4         004000              .=4000
 5 004000  004320              .WORD   215*20
 6         004020              .=4020
 7 004020  024240              .WORD   1212*20
 8         004040              .=4040
 9 004040  064260              .WORD   3213*20
 10        004060              .=4060
 11 04060  024300              .WORD   1214*20
 12        004100              .=4100
 13 04100  064340              .WORD   3216*20
 14        004120              .=4120
 15 04120  074340              .WORD   3616*20
 16        004140              .=4140
 17 04140  044340              .WORD   2216*20
 18        004160              .=4160
 19 04160  044260              .WORD   2213*20
 20        004200              .=4200
 21 04200  124120              .WORD   5205*20
 22        004220              .=4220
 23 04220  170040              .WORD   7402*20
 24        004240              .=4240
 25 04240  176000              .WORD   7700*20
 26        004260              .=4260
 27 04260  000000              .WORD   0
 28        004300              .=4300
 29 04300  040000              .WORD   2000*20
 30        004320              .=4320
 31 04320  000000              .WORD   0
 32        004340              .=4340
 33 04340  000000              .WORD   0
 34        040000              .=40000
 35        000001'             .END
.MAIN.  RT-11 MACRO VM02-12   25-MAY-78 01:27:44 PAGE 1+
SYMBOL TABLE


. ABS.  040000     000
        000000     001
ERRORS DETECTED: 0
FREE CORE: 18439. WORDS

,TT:=PDP8T3

ERRORS DETECTED: 0
FREE CORE: 18439. WORDS

*
```

```
.RU SMALL
 040000: 000000
 040020: 000000
 040040: 000000
 040060: 000000
 040100: 000000
 040120: 000000
 040140: 000000
 040160: 000000
 040200: 000000
 040220: 000000
 040240: 000000
 040260: 000000
 040300: 000000
 040320: 000000
 040340: 000000
 040360: 000000
 040400: 000000
 040420: 000000
 040440: 000000
 040460: 000000
 040500: 000000
 040520: 000000
 040540: 000000
 040560: 000000
 040600: 000000
 040620: 000000
 040640: 000000
 040660: 000000
 040700: 000000
 040720: 000000
 040740: 000000
 040760: 000000
 041000: 000000
 041020: 000000
 041040: 000000
 041060: 000000
 041100: 000000
 041120: 000000
 041140: 000000
 041160: 000000
 041200: 000000
 041220: 000000
 041240: 000000
 041260: 000000
 041300: 000000
 041320: 000000
 041340: 000000
 041360: 000000
 041400: 000000
 041420: 000000
 041440: 000000
 041460: 000000
 041500: 000000
 041520: 000000
 041540: 000000
 041560: 000000
 041600: 000000
 041620: 000000
 041640: 000000
 041660: 000000
```

3) Benchmark 3 — PDPT5

```
Address          Code           Comment
  10              0
 200             JMS      250   /jump to subroutine
 201             TAD       10
 202             DCA  I    10   /used as autoindex register
 203             ISZ      213
 204             JMP      .-3   /loop
 205             HALT

 212             7760
 213                0
 214             1777
 215                0

 250                0
 251              AND     215   /clear ACCM
 252              TAD     212
 253              DCA     213   /set count to -16
 254              TAD     214
 255              DCA     010   /set adr(10) to 1777
 256              JMP  I  250   /return
```

The output is as follows:

```
Address          Contents
 2000              1777
 2001              2000
  .                 .
  .                 .
  .                 .
 2016              2015
 2017              2016
```

```
.MAIN.  RT-11 MACRO VM02-12   25-MAY-78 01:29:02 PAGE 1

 1          000000              .ASECT
 2          000200              .=10*20
 3 000200 000000                .WORD   0000*20
 4          004000              .=200*20
 5 004000 105200                .WORD   4250*20
 6          004020              .=201*20
 7 004020 020200                .WORD   1010*20
 8          004040              .=202*20
 9 004040 070200                .WORD   3410*20
10          004060              .=203*20
11  04060 044260                .WORD   2213*20
12          004100              .=204*20
13  04100 124020                .WORD   5201*20
14          004120              .=205*20
15  04120 170040                .WORD   7402*20
16          004240              .=212*20
17  04240 177400                .WORD   7760*20
18          004260              .=213*20
19  04260 000000                .WORD   0000*20
20          004300              .=214*20
21  04300 037760                .WORD   1777*20
22          004320              .=215*20
23  04320 000000                .WORD   0000*20
24          005200              .=250*20
25  05200 000000                .WORD   0000*20
26          005220              .=251*20
27  05220 004320                .WORD   0215*20
28          005240              .=252*20
29  05240 024240                .WORD   1212*20
30          005260              .=253*20
31  05260 064260                .WORD   3213*20
32          005300              .=254*20
33  05300 024300                .WORD   1214*20
34          005320              .=255*20
35  05320 060200                .WORD   3010*20
36          005340              .=256*20
37  05340 135200                .WORD   5650*20
38          000001              .END
.MAIN.  RT-11 MACRO VM02-12   25-MAY-78 01:29:02 PAGE 1+
SYMBOL TABLE


. ABS.  005342      000
        000000      001
ERRORS DETECTED: 0
FREE CORE: 18436. WORDS

,TT:=PDP8T5

ERRORS DETECTED: 0
FREE CORE: 18436. WORDS

*
```

```
. GET [illegible]

.RU SA\A\MALL
040000: 037760
040020: 040000
040040: 040020
040060: 040040
040100: 040060
040120: 040100
040140: 040120
040160: 040140
040200: 040160
040220: 040200
040240: 040220
040260: 040240
040300: 040260
040320: 040300
040340: 040320
040360: 040340
040400: 000000
040420: 000000
040440: 000000
040460: 000000
040500: 000000
040520: 000000
040540: 000000
040560: 000000
040600: 000000
040620: 000000
040640: 000000
040660: 000000
040700: 000000
040720: 000000
040740: 000000
040760: 000000
041000: 000000
041020: 000000
041040: 000000
041060: 000000
041100: 000000
041120: 000000
041140: 000000
041160: 000000
041200: 000000
041220: 000000
041240: 000000
041260: 000000
041300: 000000
041320: 000000
041340: 000000
041360: 000000
041400: 000000
041420: 000000
041440: 000000
041460: 000000
041500: 000000
041520: 000000
041540: 000000
041560: 000000
041600: 000000
041620: 000000
041640: 000000
041660: 000000
```