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

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1-1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1-1 Significance of the Research</td>
<td>4</td>
</tr>
<tr>
<td>1-3 Thesis Introduction</td>
<td>6</td>
</tr>
<tr>
<td>1-4 General Structure of the System</td>
<td>7</td>
</tr>
<tr>
<td>II THE FIELD DESCRIPTION MODEL</td>
<td>21</td>
</tr>
<tr>
<td>2-1 Introduction</td>
<td>21</td>
</tr>
<tr>
<td>2-2 Previous Research Review</td>
<td>22</td>
</tr>
<tr>
<td>2-2-1 Dasgupta Model</td>
<td>22</td>
</tr>
<tr>
<td>2-2-2 DeWitt Model</td>
<td>23</td>
</tr>
<tr>
<td>2-3 General Description of the Host Machine</td>
<td>25</td>
</tr>
<tr>
<td>2-3-1 Hardware Description</td>
<td>25</td>
</tr>
<tr>
<td>2-3-2 Software Description</td>
<td>27</td>
</tr>
<tr>
<td>2-4 Field Description Model</td>
<td>27</td>
</tr>
<tr>
<td>2-4-1 Definition of FDM</td>
<td>28</td>
</tr>
<tr>
<td>2-4-2 General Rule to Build the FDM</td>
<td>30</td>
</tr>
<tr>
<td>2-5 Discussion of the FDM</td>
<td>37</td>
</tr>
<tr>
<td>2-6 Conclusion</td>
<td>44</td>
</tr>
<tr>
<td>III PASS 1</td>
<td>46</td>
</tr>
<tr>
<td>3-1 Introduction</td>
<td>46</td>
</tr>
<tr>
<td>3-2 Problems Arising from the Differences Between Machines</td>
<td>47</td>
</tr>
<tr>
<td>3-3 Information Supported by the User</td>
<td>50</td>
</tr>
<tr>
<td>3-4 Pass 1</td>
<td>52</td>
</tr>
<tr>
<td>IV PASS 2</td>
<td>68</td>
</tr>
<tr>
<td>4-1 Introduction</td>
<td>68</td>
</tr>
<tr>
<td>4-2 Definition and Terminology</td>
<td>72</td>
</tr>
<tr>
<td>4-3 Register Allocation/Deallocation Scheme</td>
<td>79</td>
</tr>
<tr>
<td>4-3-1 Replacement Priority Assignment</td>
<td>80</td>
</tr>
<tr>
<td>4-3-2 RA/D Algorithm</td>
<td>83</td>
</tr>
<tr>
<td>4-3-3 Tuple5 Scheme</td>
<td>83</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>4-4 Problems Arising From the Control Flow Interface</td>
<td>88</td>
</tr>
<tr>
<td>4-5 Initial State of SLC</td>
<td>94</td>
</tr>
<tr>
<td>4-6 Final State of SLC</td>
<td>96</td>
</tr>
<tr>
<td>4-6-1 Final State of the Forward Branch SLC</td>
<td>101</td>
</tr>
<tr>
<td>4-6-2 Next Initial State of Sink SLC</td>
<td>102</td>
</tr>
<tr>
<td>4-6-3 Final State of the Backward Branch SLC</td>
<td>106</td>
</tr>
<tr>
<td>4-7 Conclusion</td>
<td>106</td>
</tr>
<tr>
<td>V PASS 3</td>
<td></td>
</tr>
<tr>
<td>5-1 Introduction</td>
<td>111</td>
</tr>
<tr>
<td>5-2 General Terminology</td>
<td>114</td>
</tr>
<tr>
<td>5-3 The Parallelism and Invertibility of MOP</td>
<td>115</td>
</tr>
<tr>
<td>5-3-1 I/O Resources</td>
<td>115</td>
</tr>
<tr>
<td>5-3-2 Timing Phase</td>
<td>118</td>
</tr>
<tr>
<td>5-3-3 Field Tuple</td>
<td>120</td>
</tr>
<tr>
<td>5-4 The Detection of Parallelism of MOPs</td>
<td>121</td>
</tr>
<tr>
<td>5-5 MOPs Allocation and Movement</td>
<td>126</td>
</tr>
<tr>
<td>5-5-1 Theoretical Constraints on Optimization</td>
<td>126</td>
</tr>
<tr>
<td>5-5-2 Linear Order Compaction Algorithm</td>
<td>129</td>
</tr>
<tr>
<td>VI EXAMPLE AND CONCLUSION</td>
<td>135</td>
</tr>
<tr>
<td>6-1 Example</td>
<td>135</td>
</tr>
<tr>
<td>6-2 Performance Evaluation of Passes</td>
<td>137</td>
</tr>
<tr>
<td>6-2-1 Pass 1</td>
<td>137</td>
</tr>
<tr>
<td>6-2-2 Pass 2</td>
<td>138</td>
</tr>
<tr>
<td>6-2-3 Pass 3</td>
<td>143</td>
</tr>
<tr>
<td>6-3 Conclusion</td>
<td>145</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>148</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>150</td>
</tr>
</tbody>
</table>
# LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>General Rule to Determine the FDM</td>
<td>32</td>
</tr>
<tr>
<td>2-2</td>
<td>Selection of the Tuple &quot;OP&quot; in FDM</td>
<td>34</td>
</tr>
<tr>
<td>2-3</td>
<td>Selection of the Tuple &quot;F&quot; in FDM</td>
<td>34</td>
</tr>
<tr>
<td>3-1</td>
<td>General Structure of Pass 1</td>
<td>54</td>
</tr>
<tr>
<td>4-1</td>
<td>General Structure of Pass 2</td>
<td>71</td>
</tr>
<tr>
<td>4-2</td>
<td>RA/D Scheme</td>
<td>84</td>
</tr>
<tr>
<td>4-3</td>
<td>Label Table Determination</td>
<td>92</td>
</tr>
<tr>
<td>4-4</td>
<td>Initial State of a SLC</td>
<td>97</td>
</tr>
<tr>
<td>4-5</td>
<td>Final State of a Forward Branch SLC</td>
<td>102</td>
</tr>
<tr>
<td>4-6</td>
<td>Next Initial State of a SLC</td>
<td>107</td>
</tr>
<tr>
<td>4-7</td>
<td>Branch Final State of Backward Branch SLC</td>
<td>108</td>
</tr>
<tr>
<td>4-8</td>
<td>Sequential Final State of Backward Branch SLC</td>
<td>110</td>
</tr>
<tr>
<td>5-1</td>
<td>General Structure of Pass 3</td>
<td>113</td>
</tr>
<tr>
<td>5-2</td>
<td>$O(mn)$ Compaction Algorithm</td>
<td>131</td>
</tr>
<tr>
<td>5-3</td>
<td>Subroutine Movement</td>
<td>133</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>A Multi-level Computer System</td>
<td>2</td>
</tr>
<tr>
<td>1-2</td>
<td>General Structure of the Translation System</td>
<td>8</td>
</tr>
<tr>
<td>1-3</td>
<td>Partial IISG of PDP8 Emulator</td>
<td>10</td>
</tr>
<tr>
<td>1-4</td>
<td>Partial IESG of PDP8 Emulator</td>
<td>12</td>
</tr>
<tr>
<td>2-1</td>
<td>Functional Flow Chart of the Generation of the FDM</td>
<td>31</td>
</tr>
<tr>
<td>2-2</td>
<td>Simple Diagram of PDP11/40E CPU</td>
<td>40</td>
</tr>
<tr>
<td>2-3</td>
<td>Use of the FDM</td>
<td>45</td>
</tr>
<tr>
<td>3-1</td>
<td>Functional Flow Chart of Pass 1</td>
<td>53</td>
</tr>
<tr>
<td>4-1</td>
<td>Simplified Flow Chart of Pass 2</td>
<td>70</td>
</tr>
<tr>
<td>4-2</td>
<td>Forward Branch and Backward Branch</td>
<td>89</td>
</tr>
<tr>
<td>4-3</td>
<td>Final States of Forward Branch SLC</td>
<td>99</td>
</tr>
<tr>
<td>4-4</td>
<td>Final States of Backward Branch SLC</td>
<td>100</td>
</tr>
<tr>
<td>5-1</td>
<td>Timing Conflicts in a Polyphase Micro-instruction</td>
<td>119</td>
</tr>
<tr>
<td>5-2</td>
<td>PDP11/40E Processor Clock</td>
<td>124</td>
</tr>
<tr>
<td>6-1</td>
<td>General Structure of Example 6-1</td>
<td>136</td>
</tr>
<tr>
<td>6-2</td>
<td>The Variation of the Number of Codes in the Whole System</td>
<td>138</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>56</td>
</tr>
<tr>
<td>Virtual Machine Information from IISG</td>
<td>56</td>
</tr>
<tr>
<td>3-2</td>
<td>75</td>
</tr>
<tr>
<td>TAGs of the Variable</td>
<td>75</td>
</tr>
<tr>
<td>4-1</td>
<td>75</td>
</tr>
<tr>
<td>Components of SR(I,J)</td>
<td>75</td>
</tr>
<tr>
<td>4-2</td>
<td>82</td>
</tr>
<tr>
<td>Replacement Priority Assignment</td>
<td>82</td>
</tr>
<tr>
<td>4-3</td>
<td>86</td>
</tr>
<tr>
<td>Undetermined Field of PDP11/40E FDM</td>
<td>86</td>
</tr>
<tr>
<td>4-4</td>
<td>91</td>
</tr>
<tr>
<td>Label Table</td>
<td>91</td>
</tr>
<tr>
<td>5-1</td>
<td>117</td>
</tr>
<tr>
<td>I/O Intersection</td>
<td>117</td>
</tr>
<tr>
<td>5-2</td>
<td>120</td>
</tr>
<tr>
<td>&lt;I,O,T&gt; Conflict Detection</td>
<td>120</td>
</tr>
<tr>
<td>5-3</td>
<td>127</td>
</tr>
<tr>
<td>Possible Positions of MOPs in the Allocation Problems</td>
<td>127</td>
</tr>
<tr>
<td>6-1</td>
<td>141</td>
</tr>
<tr>
<td>Evaluation of Pass 2 (Number of registers w.r.t. the length of code produced)</td>
<td>141</td>
</tr>
<tr>
<td>6-2</td>
<td>142</td>
</tr>
<tr>
<td>Testing O(mn) Algorithm on the Husson's Machine</td>
<td>142</td>
</tr>
<tr>
<td>6-3</td>
<td>144</td>
</tr>
<tr>
<td>Testing O(mn) Algorithm on the PDP11/40E Machine</td>
<td>144</td>
</tr>
</tbody>
</table>
## GLOSSARY

### Acronyms and symbols

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>arithmetic logic unit of a computer</td>
</tr>
<tr>
<td>CPU</td>
<td>central process unit of a computer</td>
</tr>
<tr>
<td>FDM</td>
<td>Field Description Model</td>
</tr>
<tr>
<td>FS</td>
<td>final state of a SLC</td>
</tr>
<tr>
<td>GPR</td>
<td>general purpose registers of the host machine</td>
</tr>
<tr>
<td>IESG</td>
<td>executable statement group of IML</td>
</tr>
<tr>
<td>IISG</td>
<td>information statement group of IML</td>
</tr>
<tr>
<td>IML</td>
<td>host machine independent intermediate language</td>
</tr>
<tr>
<td>IS</td>
<td>initial state of a SLC</td>
</tr>
<tr>
<td>MDIL</td>
<td>host machine dependent intermediate language</td>
</tr>
<tr>
<td>MET</td>
<td>Macro Expansion Table</td>
</tr>
<tr>
<td>MI</td>
<td>microinstruction</td>
</tr>
<tr>
<td>MOP</td>
<td>microoperation</td>
</tr>
<tr>
<td>NR</td>
<td>the number of general purpose registers in the host machine</td>
</tr>
<tr>
<td>RA/D scheme</td>
<td>register allocation and deallocation scheme</td>
</tr>
<tr>
<td>SLC</td>
<td>straight line code</td>
</tr>
</tbody>
</table>

### Symbols

- $M_i \beta M_j$ means $M_i$ is data independent of $M_j$
- $M_i / M_j$ means $M_i$ is parallel $M_j$
- $M_i \triangleright M_j$ 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 micro-instructions). 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. It is the translation of the IML described by Malik (13) that concerns this investigation.

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.

The major significance of this research, then, is
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
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 micro-
programming 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 be-
havior 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.
<table>
<thead>
<tr>
<th>IML (IISG Section)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00A PDP8</td>
<td>name of the emulator.</td>
</tr>
<tr>
<td>00D ...12</td>
<td>target machine has 12-bit words.</td>
</tr>
<tr>
<td>00E TWO</td>
<td>target machine is 2's complement.</td>
</tr>
<tr>
<td>221</td>
<td>target machine memory is 4096x12 bit words.</td>
</tr>
<tr>
<td>00G EFTADR</td>
<td>block name for effective address computation.</td>
</tr>
<tr>
<td>207 MEM</td>
<td>global variables used by the emulation to simulate the registers of the PDP8 target machine.</td>
</tr>
<tr>
<td>207 IR</td>
<td></td>
</tr>
<tr>
<td>207 PC</td>
<td></td>
</tr>
<tr>
<td>208 MAR</td>
<td></td>
</tr>
<tr>
<td>120 ADR,,7</td>
<td>local variables with 7.12, and 12 bit precision, respectively</td>
</tr>
<tr>
<td>120 PCTEMP,,12</td>
<td></td>
</tr>
<tr>
<td>120 MART,,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>IML Macro Table</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTR PGEADR IR +T.003</td>
<td>The first IML code of Figure 1-4. The following three codes are its macro expansion.</td>
</tr>
<tr>
<td>PUSH1 *1+IR TOS</td>
<td>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 &quot;1&quot;) that will be used later (denoted by the sign &quot;+&quot;).</td>
</tr>
</tbody>
</table>
IML *IESG Section)

OOG EFTADR

EXTR PGEADR IR +T.003

MOVE -T.003 ADR

CONDF .IR,7 TL.001

SUB PC c1 PCTEMP

EXTR CRNTPG PCTEMP +T.004

MOVE -T.004 PCTEMP

OR PCTEMP ADR MAR

Comments

Name of executable block for address calculation.

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

Copy to ADR. The "-" indicates 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.

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

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

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

Copy from temporary variable T.004 (made inactive ") into PCTEMP.

Inclusive OR PCTEMP with ADR and store into MAR.

Figure 1-4. Partial IESG of PDP8 Emulator
<table>
<thead>
<tr>
<th>IML Macro Table</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSMK TOS PGEADR D</td>
<td>Right-shift and mask the TOS word with PGEADR as a mask and store into host register D.</td>
</tr>
<tr>
<td>MOVE5 D *2+T.003</td>
<td>Move host register D to temporary variable T.003. Pass 1 tags T.003 as a local symbolic variable (indicated by the &quot;2&quot;) that will be used later.</td>
</tr>
</tbody>
</table>

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.
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.

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, $c_1=16$, 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.
MOVE 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 \( \langle OP, I, 0, F, P \rangle \).

- **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 \( \langle OP, I, 0 \rangle \).
- **F**: a set of fields in the host machine micro-instruction format used to execute the \( \langle OP, I, 0 \rangle \).

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.
- **0**: 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 SMB=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 micro-instruction.
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. \(\langle OP, I, O, F, P\rangle\), 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 micro-instructions. 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.

\[ m = (\text{OP}, \text{SC}, \text{SK}, U, V) \]

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:

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

\[ FDM = \{ M_i, 1 \leq i \leq n \} \]

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

\[ M_i = \{ OP, I, O, F, P \} \]
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:

\[ M_1 = \langle \text{ADD}, I, O, F, P \rangle \]

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.
collect all necessary "OPs" which are sufficient to describe the IESG of IML and denoted by $\text{OPN} = \{\text{OP}_i \mid 1 \leq i \leq n\}$

for each $\langle\text{OP}_i\rangle$, select the legal $\langle I_i, O_i\rangle$ such that there is no conflict in the execution of $\text{OP}_i, I_i, O_i$

from the control store cycle, find the timing period, denoted by $\langle P_i \rangle$, needed to execute the $\langle\text{OP}_i, I_i, O_i\rangle$

from the microinstruction format, find the fields used to store the $\langle\text{OP}_i, I_i, O_i, P_i\rangle$ while it is executing and denoted by $\langle F_i \rangle$

increment the index $i$

if $i > n$ then 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 PHASE(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 $\langle OP, I, O \rangle$. 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

$$R2 + B \rightarrow R3$$ ; 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 STM 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 \( \langle \text{OP, I, 0} \rangle \).

Each \( \langle \text{OP, I, 0} \rangle \) 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 \( \langle \text{OP, I, 0} \rangle \) 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.

\[ \uparrow \text{Example 2-4:} \]

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 micro-instruction 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 \( \langle OP, I, O \rangle \) 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 $<\text{OP}, I, O>$ 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 micro-address.

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: $400 \rightarrow D, P2$; 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 1 = 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.

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:

\[
\text{IF } (f(1,14)=1) \text{ and } (f(2,1) \text{ are set})
\]

THEN there is an RD bus conflict between M1 and M2

ELSE no conflict
where \( f(i,j) \) means field \( j \) in \( \text{MOP}_i \).

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.

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

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.

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

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:

<table>
<thead>
<tr>
<th>F1 and F3</th>
<th>F2</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(1,1)=f(3,1)=2</td>
<td>f(2,1)=3</td>
<td>f(4,1)=2</td>
</tr>
<tr>
<td>f(1,1)=f(3,2)=1</td>
<td>f(2,2)=1</td>
<td>f(4,2)=1</td>
</tr>
<tr>
<td>f(1,5)=f(3,5)=0</td>
<td>f(2,4)=2</td>
<td>f(4,4)=2</td>
</tr>
<tr>
<td>f(1,6)=f(3,6)=9</td>
<td>f(2,11)=3</td>
<td>f(4,11)=3</td>
</tr>
<tr>
<td>f(1,19)=f(3,29)=1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(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)

**Figure 2-3. Use of the FDM**

- **Pass 1**
  - Decode IESG
  - $\langle \text{OP, I, O} \rangle$
  - (machine dependent)

- **Pass 2**
  - Assign field and timing information to $\langle \text{OP, I, O} \rangle$
  - $\langle \text{OP, I, O, F.P} \rangle$

- **Pass 3**
  - (Based on the 5-tuple format, a concurrency detection rule is developed)
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-proven 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:

\[ \text{TADD SRC1 SRC2 Dest} \]

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
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.

Subroutine "WRDSIZE" is to simulate the word size difference problem.

Subroutine "OPRATOR" links the simple IML codes with the corresponding host machine codes.

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:

<table>
<thead>
<tr>
<th>Address</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>00A</td>
<td>PDP-8</td>
</tr>
<tr>
<td>00D</td>
<td>..,.12</td>
</tr>
<tr>
<td>00E</td>
<td>TWO</td>
</tr>
<tr>
<td>221</td>
<td>MEM,4096,12</td>
</tr>
<tr>
<td>220</td>
<td>ACCM</td>
</tr>
<tr>
<td>214</td>
<td>LNK,,1,1</td>
</tr>
<tr>
<td>005</td>
<td>OPCODE,,9,11,-9</td>
</tr>
<tr>
<td>00G</td>
<td>EFTADR</td>
</tr>
<tr>
<td>207</td>
<td>PC</td>
</tr>
<tr>
<td>120</td>
<td>ADR</td>
</tr>
<tr>
<td>120</td>
<td>MART</td>
</tr>
</tbody>
</table>
Table 3-1. Virtual Machine Information from IISG

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

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Host memory location</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCM</td>
<td>2000</td>
<td>PDP8 accumulator</td>
</tr>
<tr>
<td>PC</td>
<td>1000</td>
<td>PDP8 program counter</td>
</tr>
<tr>
<td>ADR</td>
<td>2003</td>
<td>PDP8 effective address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field variable</th>
<th>Value range</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>9,11,9</td>
<td>PDP8 opcode field</td>
</tr>
</tbody>
</table>

*** 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
COND T 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 \(2^4\times (8)\) into AB
```

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

\text{SHR, SHL, SLCT, and EXTR.}\textbf{44}

If the host machine \underline{does not provide} a facility to set flags, the problem of target-host mismatching must be \underline{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
```

<table>
<thead>
<tr>
<th>higher 16 bits</th>
<th>lower 16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBVAR</td>
<td>LBVAR</td>
</tr>
</tbody>
</table>

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

COND 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
COND 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
RSNK 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

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

*Code '1' 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/de-allocation 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) \( \text{OPND} = \{\text{OPND}_1, \text{OPND}_2\} \) is a set of operands, where \( \text{OPND}_1 \) is a set of machine unit names, and \( \text{OPND}_2 \) is a set of symbolic variables to be register allocated.

2) \( \text{GPR} = \{R(1), R(2), \ldots, R(NR)\} \) 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 \( J \)th register in the set of GPRs, where \( 1 \leq J \leq NR \).

3) \( \text{VML} \) 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 \( \text{BK} = \text{BK}_1, \text{BK}_2, \ldots, \text{BK}_{BNK} \) of blocks. Each block starts with a special code \( \text{BKS} \), 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:

\[ S(I) = \sum_{J=1}^{NR} SR(I,J) \]

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)

WMOVE 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)

<table>
<thead>
<tr>
<th>Status ST(J)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>The value of the variable in R(J) is different from the content of the same variable stored in VML.</td>
</tr>
<tr>
<td>Passive</td>
<td>The value of this variable is the same between the VML and the register.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position PT(J)</th>
<th>Source</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>This variable is used as source in the statement.</td>
</tr>
<tr>
<td></td>
<td>Dest</td>
<td>This variable is used as destination in the statement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type TY(J)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This global variable will be used later in this current block.</td>
</tr>
<tr>
<td>3</td>
<td>This global variable will not be used in the current block.</td>
</tr>
<tr>
<td>2</td>
<td>This local variable will be used later in this current block.</td>
</tr>
<tr>
<td>none</td>
<td>This local variable will not be used in the current block.</td>
</tr>
</tbody>
</table>

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:

\[
\text{RMOVE Mem PC IR ; IR} \leftarrow \text{Mem(PC)}
\]

This means the memory content of PC is read into a register IR (instruction register).

The corresponding DIL codes are:

\[
\begin{align*}
\text{(MOVE8 *1+PC BA ; copy the address of PC to Bus address register, set DATI, and then turn off processor clock)} \\
\text{(MOVE4 unibus *1-IR; copy the value of PC to IR)}
\end{align*}
\]

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:

\[
\text{WMOVE Mem \: MAR \: -T.001 \: ; Mem(MAR \leftrightarrow -T.001}
\]

The corresponding DIL codes are:

\[
\begin{align*}
\text{MOVE2} & \quad \ast 1 + \text{MAR} \quad \text{BA} \quad ; \text{copy the address of} \\
\text{MOVE9} & \quad \ast 2 - \text{T.001} \quad \text{D} \quad ; \text{copy the value of} \\
\text{NOOP} & \quad \text{; no operation}
\end{align*}
\]

\[
\text{This means the address of MAR is moved to} \\
\text{BA. Then the content of -T.001 is moved} \\
\text{to register D, and the machine stores the} \\
\text{content of D into the address which is} \\
\text{in BA.}
\]

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:

\[
\text{MEMREAD variable register} \quad ; \quad \text{register} \leftarrow \quad \text{Mem(variable)}
\]
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 \texttt{MEMWRITE} statement is:

\begin{verbatim}
MEMWRITE register variable ; Mem(register) ; variable
\end{verbatim}

and the corresponding PDP11/40E microcodes are:

\begin{verbatim}
MOVE12 variable BA ; copy the address of "variable" to register BA
MOVE9 register D ; copy the value of "register" to register D, set DAT0, and then turn off processor clock
NOOP ; no operation
\end{verbatim}

For the same reason, the reader may compare the difference between \texttt{WMOVE} in IML and \texttt{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 \texttt{MEMREAD} or \texttt{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 \texttt{MEMREAD} or \texttt{MEMWRITE} statements is kept as small as possible. The efficiency of the RA/D scheme is highly dependent on the priority assignment of variables.

\textbf{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 \texttt{MEMWRITE} statement is needed to store this variable in the host machine memory. However, an extra \texttt{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

\[
\text{ADD } *2-\text{AB } *1+\text{BC } *1+\text{BC } ; \text{AB}+\text{BC} \rightarrow \text{BC}
\]

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:

- \text{MEMREAD AB R1 } ; \text{R1} \leftarrow \text{Mem(AB)}
- \text{MEMREAD BC R1 } ; \text{R1} \leftarrow \text{Mem(BC)}
- \text{ADD R1 R1 R1 } ; \text{R1}+\text{R1} \rightarrow \text{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

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

*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 ≤ J ≤ 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 DETERMINE TY(J)
IF K=3 (This operand is destination)
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
ELSE BEGIN (This operand is source)
MEMREAD ODA(\\text{M},K). R(J)
(load the operand into R(J)
SA(J)=ODA(\\text{M},K)
ST(J)=\text{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)=\text{ACTIVE}
THEN "MEMWRITE R(J) SA(J)
IF ST(J)=\text{ACTIVE}
THEN "MEMWRITE R(J) SA(J)*
(store the content of R(J) into
memory)
GO TO FREE
END
END.
END.

Subroutine TYPE

BEGIN
SEPARATE ODA(\\text{M},K) INTO SY(1), SY(2), AND SY(4)
IF SY(3)=\text{"+" (ODA(\\text{M},K) will be used later in the block)
THEN TY(J)=SY(2)
ELSE BEGIN (ODA(\\text{M},K) will not be used any more)
IF SY(2)=\text{"2" (ODA(\\text{M},K) is a local variable)
THEN TY(J)=\text{NONE}
ELSE TY(J)=\text{"3"
END.
END.
Table 4.3. Undetermined Field of PDP11/40 FDM

<table>
<thead>
<tr>
<th>Case</th>
<th>Format in the FDM/Field value determination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OP SRC1(GPR) SRC2 Dest(*GPR)</td>
</tr>
<tr>
<td></td>
<td>Field(1)=function (the register used in the</td>
</tr>
<tr>
<td></td>
<td>operand *GPR)</td>
</tr>
<tr>
<td>2</td>
<td>OP *emit Dest</td>
</tr>
<tr>
<td></td>
<td>Field(18)=function (the constant used in *emit)</td>
</tr>
<tr>
<td>3</td>
<td>OP SRC1 $CT Dest</td>
</tr>
<tr>
<td></td>
<td>or OP B TOS,CT D</td>
</tr>
<tr>
<td></td>
<td>Field(15), Field(16), or Field(17) is a function of CT.</td>
</tr>
<tr>
<td>4</td>
<td>OP SRC1 $FF,LL,CT Dest</td>
</tr>
<tr>
<td></td>
<td>Field(15), Field(16), and Field(17) are a function of FF, LL, and CT.</td>
</tr>
<tr>
<td>5</td>
<td>OP variable Dest</td>
</tr>
<tr>
<td></td>
<td>Field(18)=function (address value of the variable)</td>
</tr>
<tr>
<td>6</td>
<td>Field(13)=function (next MOP address)</td>
</tr>
</tbody>
</table>
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 RS\text{\textit{MK}} in FDM is:
OP: RS\text{\textit{MK}}
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:
RS\text{\textit{MK}} 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(Iₖ), SLC(Iₘ), and SLC(Iₙ) 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(Iₚ) 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

<table>
<thead>
<tr>
<th>Components</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label vector</td>
<td>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(I) is a label name with index I in the label vector.</td>
</tr>
<tr>
<td>LBL</td>
<td>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.</td>
</tr>
<tr>
<td>SQ(I)</td>
<td>It is used to count the number of forward branch statements to this label.</td>
</tr>
<tr>
<td>SB(I)</td>
<td>It is a matrix which is used to record the indexes of SLCs which support the forward branch statement to this label.</td>
</tr>
<tr>
<td>BS(I,J)</td>
<td>It is used to count the number of backward branch statement to this label.</td>
</tr>
<tr>
<td>J=1 to</td>
<td>It is used to record the indexes of SLCs which support the backward branch statement to this label.</td>
</tr>
<tr>
<td>SB(I)</td>
<td>If the label name is a block name then it is used to record the block index.</td>
</tr>
<tr>
<td>BWL(I)</td>
<td>It is an index of the SLC which contains the label statement with label name LBL(I).</td>
</tr>
</tbody>
</table>
Algorithm 4-3.

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

(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 $\text{LBL}(M)$
    IF $\text{LBL}(M)$ IS A BLOCK NAME
    THEN $\text{BI}(M) =$ BLOCK INDEX
    ELSE $\text{BI}(M) = 0$
    $\text{SB}(M) = 0$ (set f.b. counter)
    $\text{SQ}(M) = 1$ (label name appears in the label position)
    $\text{BWL}(M) = 0$ (set b.b. counter)
    $\text{SLCD}(M) =$ 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 $\text{LBL}(J)$
    SET $\text{SQ}(J) = 0$, $\text{SB}(J) = 0$
    IF $\text{LBL}(J)$ IS A BLOCK NAME
    THEN $\text{BI}(J) =$ BLOCK INDEX
    ELSE $\text{BI}(J) = 0$
  END
ELSE BEGIN (it is from the branch stmt)
  IF $\text{SQ}(J) = 0$ (it is a forward branch)
    THEN BEGIN
      $\text{SB}(J) + 1$ (INC the f.b. counter)
      $\text{BS}(J, \text{SB}(J)) =$ CURRENT SLC INDEX
    END
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 \( I_1, I_2, \ldots, I_n \).

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\neq0, and SLC(I-1) is not an unconditional branch SLC then IS(I) can be expressed by \( IS(I)=f_1(FS(I_1) \ldots FS(I_n), FS(I-1)) \).

if SLC(I-1) is an unconditional branch SLC, then \( IS(I)=f_2(FS(I_1) \ldots FS(I_n)) \).
To simplify the description, we have
\[ IS(I) = f(FS(I_1), \ldots, FS(I_m)) \]  
where the number of \( m \) is \( n \) or \( n+1 \).

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

\[ IS(I) = \bigcup_{J=1}^{NR} ISR(I, J) \]
\[ FS(K) = \bigcup_{J=1}^{FSR(K, J)} \]

Where \( FSR(K, J) \) is the state of the \( j \)th 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:
\[ IS(I) = f(\bigcup_{k=1}^{m} FS(I_k)) \]
\[ ISR(I, J) = f_j(\bigcup_{k=1}^{m} FSR(I_k, J)) \]

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

4) Operator \( \exists \) is defined as:

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

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≤k≤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) = \( \exists A_i \) FST(I_k,J) (Determine the status of this variable)
THEN ITY(I,J) = FTY(I_k,J)
ELSE ITY(I,J) = FTY(I_k,J)
END
ELSE BEGIN (One of the variables in R(J) from SLCs, I_1, ..., I_1 is different from others)
   I\_SR(I,J) IS SET TO BE A NULL STATE
   FOR ALL k, 1 ≤ k ≤ 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 I\_SR(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.
1. Each circle means a statement.
2. $\text{SLC}(I)$ is a forward branch SLC.
3. $\text{SLC}(K)$ is a sink SLC to $\text{SLC}(I)$.
4. $\text{CS}(I)$, $\text{PS}(I)$, and $\text{S}(I)$ are defined in section 4-6.

Figure 4-3. Final State of Forward Branch SLC
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) = \( \sum_{J=1}^{NR} NSR(K,J) \). (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≤J≤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 sequential 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:

<table>
<thead>
<tr>
<th>IST(K,J)</th>
<th>NPT(K,J)</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>source</td>
<td>1</td>
</tr>
<tr>
<td>active</td>
<td>dest</td>
<td>2</td>
</tr>
<tr>
<td>passive</td>
<td>source</td>
<td>3</td>
</tr>
<tr>
<td>passive</td>
<td>dest</td>
<td>4</td>
</tr>
</tbody>
</table>

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 VM 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 destination)

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))
               (extra case 2)
               THEN BEGIN
                   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
    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))
       (extra case 2)
       THEN BEGIN
           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.
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) 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)=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 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.
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\(_1\)MOPs 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\(_1\)MOPs 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 FDP11/40E 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 \( \bar{M}_1, \bar{M}_2, \ldots \bar{M}_k \) from a compiler-generated sequence of microoperations, \( M_1, M_2, \ldots \ M_n \). The optimization algorithm which is used here to solve this problem is applied separately each SLU. 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 NEWREAD/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, \( M_1, M_2, \ldots, M_n \) are mapped into a sequence of micro-instructions, \( \bar{M}_1, \bar{M}_2, \ldots, \bar{M}_k, k \in \mathbb{N} \).

1) A SLC is the basic unit to be optimized and is represented by \( \text{SLC} = \{ M_1, M_2, \ldots, M_n \} \), 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 \( \{ \bar{M}_i \}, \{ \bar{M}_j \}, \{ \bar{M}_k \}, \ldots \), if the MOPs are executed in separate control store cycles. Two MOPs, \( M_i \) and \( M_j \), are executed concurrently, denoted by \( \{ \bar{M}_i, \bar{M}_j \} \), if they are executed in the same control store cycle.

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

4) \( M_i \) and \( M_j \) are said to be parallel, denoted by \( M_i \parallel M_j \), if for all inputs the sequential execution of \( \{ M_i \}, \{ M_j \} \), results in the same output as the concurrent execution of micro-instruction \( \bar{M}_k = \{ M_i, M_j \} \).

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 $I_i \cap O_j \neq \emptyset$, $I_j \cap O_i \neq \emptyset$, or $O_i \cap O_j \neq \emptyset$, 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 $M_1^i, M_2^i, \ldots M_k^i$, 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 $M_i < M_j$):

$M_i : \{O_{i_1}, I_i, O_i\}$

$M_j : \{O_{j_1}, I_j, O_j\}$

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 \( m_i \neq m_j \). This leads to a very important factor in the optimization problem. For example, consider the sequence of \( MOF_s, N_1, M_2, M_3 \), with \( M_1 \neq M_2 \) and additional properties that \( M_1 \neq M_2 \), \( M_2 \neq M_3 \) but \( M_1 \neq M_3 \). 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 \( MOF_s \) 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 \( M_1 \) leaving \( M_3 \) assigned to \( M_2 \) to give an optimized partition for \( r = 2 \). \( M_i, M_{i+1} \) are said to be invertible, denoted by \( M_i \neq M_{i+1} \), if \( M_i \neq M_{i+1} \).
<table>
<thead>
<tr>
<th>Row</th>
<th>Nonempty intersection</th>
<th>Sequential action</th>
<th>Parallel action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$I_i \cap I_j$</td>
<td>Data sharing from common resource</td>
<td>Same as sequential</td>
</tr>
<tr>
<td>2</td>
<td>$A = I_i \setminus O_j$</td>
<td>$I_i$ transfers to $O_i$ then $O_j$ modified $I_i$</td>
<td>*If $M_j$ is executed first, $O_j$ reset $I_i$ before $I_i$ transfers to $O_i$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Same as sequential</td>
</tr>
<tr>
<td>3</td>
<td>$B = O_i \cap I_j$</td>
<td>Data passes from $M_i$ to $M_j$</td>
<td>*$O_i$ has no chance to set $I_j$ if $M_j$ is executed first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Same as sequential</td>
</tr>
<tr>
<td>4</td>
<td>$C = O_i \setminus O_j$</td>
<td>$O_i$ is modified by $O_j$</td>
<td>*If $M_j$ is executed first, $O_j$ cannot modify $O_i$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Same as sequential</td>
</tr>
</tbody>
</table>

- $M_i, M$ 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 $M_i < M_j$ 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 $T_i = T_j$, $T_i \leq T_j$ or $T_i \geq T_j$

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 $M_k = \{M_i, M_j\}$ can produce the same output as the sequential execution of $M_k = M_i$, $M_{k+1} = M_j$, 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) \( T_i < T_j \)

\[ \overbrace{\quad T_i \quad}^{T_j} \overbrace{\quad T_i \quad} \]

CS cycle

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

B) \( T_i < T_j \)

\[ \overbrace{\quad T_i \quad}^{T_j} \overbrace{\quad T_i \quad} \]

CS cycle

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

C) \( T_i = T_j \)

\[ \overbrace{\quad T_i \quad}^{T_j} \overbrace{\quad T_i \quad} \]

CS cycle

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. \(<I,O,T>\) Conflict Detection

<table>
<thead>
<tr>
<th>Nonempty intersection</th>
<th>Parallel and sequential execution leave same result</th>
<th>Parallel and sequential execution leave different result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_i \cap I_j)</td>
<td>independent of timing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(I_i \cap 0_j) (T_i \leq T_j)</td>
<td>(T_i &gt; T_j)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0_i \cap I_j) (T_i &lt; T_j)</td>
<td>(T_i &gt; T_j)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0_i \cap 0_j) (T_i &lt; T_j)</td>
<td>(T_i &gt; T_j)</td>
</tr>
</tbody>
</table>

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.

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

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

\(M_i, M_j\) are in SLC. \(F_i, F_j\) are the field tuple to \(M_i, M_j\), respectively, and it is assumed:
If \( \forall f_i \in F_k \Rightarrow f_i \in F_A \) 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 \( \forall f_i \in F_k \Rightarrow f_i \in F_A \), 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 \( M_i \subseteq M_{i+1} \) then \( M_i \gg M_{i+1} \).
2) As \( P_i < P_j \).
   If \( F_i \cap F_j = 0 \) then \( M_i \triangleright M_j \).
3) As \( P_i \triangleleft P_j \).
   If \( (F_i \cap F_j = 0) \) and \( (M_i \triangleleft M_j \) then \( M_i \triangleright M_j \).

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: \( M_1 \): R2 \( \rightarrow \) D, P2 : copy R2 to register D
\( M_2 \): D \( \rightarrow \) R3, P3 : copy register D to R3
\( M_3 \): R3+B \( \rightarrow \) D, P2 : add R3 and register B to register D
\( M_4 \): D \( \rightarrow \) R4, P3 : copy register D to R4

\( M_2 \) and \( M_3 \) are examined to detect the parallelism.

From example 2-5, we know \( F_2 \cap F_j = 0 \), but \( M_2 \not\subseteq M_3 \). This implies \( M_2 \not\subseteq M_3 \). (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: $M_5$: emit → stack, $P_3$ ; copy content value "emit" to stack
$M_6$: $R_3$ → $D$, $P_2$ ; copy $R_3$ to register $D$

From the third rule, $(F_5 \cap F_6 = 0)$ and $M_5 \not\subseteq M_6$

imply $M_5 \neq M_6$.

Case 3: $M_7$: $R_3 + B$ → $D$, $P_2$ ; add $R_3$ and $B$ to register $D$
$M_8$: $D$ → $R_3$, $P_3$ ; copy register $D$ to $R_3$

The pulses used by $M_7$ and $M_8$ are $P_2$ and $P_3$, respectively. $F_7 \cap F_8 = 0$ implies $M_7 \neq M_8$ 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 $P_1$; b) CL2 cycle generates pulse $P_2$; and c) CL3 cycle generates pulse $P_2$ and pulse $P_3$.

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 $\rightarrow$ 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 $M_j \forall MI_k \neq M_j$. Also MOP $M_k$ is said to be invertible with MI, if $\forall M_j \exists MI_k \neq M_j$.

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 $MI_1, \ldots, MI_i$. As we allocated $M_k$, relationship between MOP and MI is: (refer to Table 5-3)

Case 1: $M_k$ not $\neq MI_i$, and $M_k$ not $\neq MI_i$

Case 2: $M_k$ not $\neq MI_i$, and $M_k$ $\neq MI_i$
Case 3: $M_k > M_i$, and $M_k$ not $/\!\!/ M_i$

Case 4: $M_k > M_i$, and $M_k /\!\!/ M_i$

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

<table>
<thead>
<tr>
<th>Possible position case number</th>
<th>$M_{i+1}$</th>
<th>$M_i$</th>
<th>$M_i \ldots M_{i-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case 2</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Case 3</td>
<td>X</td>
<td></td>
<td>$\Delta$</td>
</tr>
<tr>
<td>Case 4</td>
<td>X</td>
<td>X</td>
<td>$\Delta$</td>
</tr>
</tbody>
</table>

$X$: MOP can be in this position.

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

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

In Case 3 and 4 of Table 5-3, we have to check the MOP ahead of the current $M_i$. Again we face four cases. If $M_k$ is invertible with all MIs from $M_i$ back to $M_1$, there are $(i+1)$ possible positions for $M_k$: one position is ahead of $M_1$, one is after $M_i$. 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, $M_j$ 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 $M_3$.

\vdots

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

Totally, there are $\frac{n!}{i} \cdot k!$ 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 invertibility 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

\[
\frac{\sum_{j=1}^{h} |MI_{i-j}|}{h} \text{ is nearest to } m. 
\]

\( |MI_k| \) 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, ..., M_k, ..., M_n\}. Assume MOP \(M_k\) is to be determined for \(2^k \leq n\) and \(M_1, M_2, ..., M_{k-1}\) is partitioned into \(MI_1, MI_2, ..., MI_{j-1}\) 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 $M_{I_1} ... M_{I_j}$

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 compari-

sons 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)
$S=S+1_{MI_j}$
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 $M_{I_j}$
GO TO STRT
END

END
Algorithm 5-2 continued)

ELSE BEGIN

IF $M_k > M_{ij}$

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

ELSE GO TO A

END.

(B) IF $kk = 0$ ($M_k$ has never been parallel with any $M_{ik}$, where $kk < j$)

THEN BEGIN

ALLOCATE $M_k$ into $M_{ij+1}$

j = j + 1 (set the new MI index)

GO TO STRT

END

ELSE ALLOCATE $M_k$ INTO $M_{ik}$,

GO TO STRT

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 \( M_{i_j}, M_{i_{j-1}}, \ldots M_{i_q} \) (\( M_{i_q} \) is the MI contained the label MOP)

IF \( M_k \) IS INVERTIBLE WITH ALL THESE MIs
THEN BEGIN
  \( t=j \)
  WHILE \( t=q \) (change the index of MI)
  DO BEGIN
    \( M_{i_{t+1}} \leftarrow M_{i_t} \)
    \( t \leftarrow t-1 \)
  END
END
ALLOCATE \( M_k \) INTO \( M_{i_q} \) (\( M_k \) is moved out the branch region)
GO TO D
END
ELSE ALLOCATE \( M_k \) INTO \( M_{i_{j+1}} \) (\( M_k \) may not be used to generate \( NS(P) \) 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, \( M_{m+1},...M_n \), the total number of comparisons is \( T(n)=1+2+...+m+(n-m)m \). 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≤K (the details are in Chapter V). Pass 1 is one of the factors that influences the system's efficiency (with respect to the
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.

Rennem, 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)

<table>
<thead>
<tr>
<th>n</th>
<th>OP_n</th>
<th>f=OP_n-OP_9</th>
<th>f/OP_9</th>
<th>f2=OP_n-IP</th>
<th>f2/IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>267</td>
<td>76</td>
<td>39.8</td>
<td>93</td>
<td>53.4%</td>
</tr>
<tr>
<td>4</td>
<td>262</td>
<td>71</td>
<td>37.1%</td>
<td>88</td>
<td>50.6%</td>
</tr>
<tr>
<td>5</td>
<td>248</td>
<td>57</td>
<td>29.8%</td>
<td>74</td>
<td>42.5%</td>
</tr>
<tr>
<td>6</td>
<td>219</td>
<td>28</td>
<td>14.6%</td>
<td>45</td>
<td>25.8%</td>
</tr>
<tr>
<td>7</td>
<td>215</td>
<td>24</td>
<td>7.3%</td>
<td>41</td>
<td>23.6%</td>
</tr>
<tr>
<td>8</td>
<td>203</td>
<td>12</td>
<td>6.3%</td>
<td>29</td>
<td>16.7%</td>
</tr>
<tr>
<td>9</td>
<td>191</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>8.9%</td>
</tr>
<tr>
<td>10</td>
<td>191</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>8.9%</td>
</tr>
</tbody>
</table>

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:

\[
IR = \frac{(\# \ of \ OP_n - \# \ of \ IP)}{(\# \ of \ IP)}
\]

As shown, as the value of N decreases, the value of IR
Table 6-2. Testing $O(m)$ Algorithm on the Hussom's Machine

<table>
<thead>
<tr>
<th>$m$</th>
<th>$L$</th>
<th># of MIs</th>
<th>$m$</th>
<th>$L$</th>
<th># of MIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>2</td>
<td>30</td>
<td>**</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>31</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>**</td>
<td>3</td>
<td>21</td>
<td>**</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>**</td>
<td>4</td>
<td>18</td>
<td>**</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>18</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>**</td>
<td>5</td>
<td>18</td>
<td>**</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>18</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>**</td>
<td>**</td>
<td>18</td>
<td>**</td>
<td>**</td>
<td>3</td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th># of m</th>
<th># of MOPs reduced</th>
<th># of OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>38</td>
<td>153</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>153</td>
</tr>
<tr>
<td>5</td>
<td>38</td>
<td>153</td>
</tr>
<tr>
<td>6</td>
<td>38</td>
<td>153</td>
</tr>
</tbody>
</table>

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 \( m \leq 8 \).

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 programatically 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 \textit{extra case 1} and \textit{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.


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 $c \beta \gamma$ then:

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

$\gamma$ -
- $0$ 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:

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

OTHER INFORMATION

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.
153

$S_1$, $S_2$, $S_3$, $S_4$ 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 $\uparrow$, $\downarrow$, $\uparrow$, $\cdot$.

c) $2/2/1)9 - \alpha$

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

d) $2(2/1) - \beta$

The tag indicates a global flag declaration. $\beta$ 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:

  \[ \text{ID, NUMBER} \]

  where `NUMBER` refers to the bit of `ID` in question.

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

  \[ \text{ID}_1, \text{ID}_2 \]

  where `ID_1` and `ID_2` are identifier names.
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:

\[
\text{OPERATION SRC1 SRC2 DEST}
\]

which means:

- if operation is RMOVE
  \[
  \text{DEST } \leftarrow \text{SRC1 (SRC2)}
  \]
- else if operation is WMOVE
  \[
  \text{SRC1 (SRC2)} \leftarrow \text{DEST}
  \]

4 - This class deals with the various branch operations.

a. - \textit{COMP SRC1 SRC2}

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

\[
\text{SRC1 - SRC2}
\]

along with the flag settings.

b. - The direct branch statement is:

\[
\text{BRCH label}
\]

meaning go to the label.

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

\[
\text{OPERATION *FLAG LABEL}
\]

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.,

\[
\text{CONDF } Q \text{ ZETA}
\]

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  means  SRC1  SRC2  1
c. SET SRC1  means  SRC1  all 1's
d. CLR SRC1  means  SRC1  0
e. MOVE SRC1 DEST  means  DEST  SRC1
f. PUSH SRC1  means  Push SRC1 into STACK
g. POP DEST  means  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

  * 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): RIFE0t?1
FIELD(2): SRX[(47]
FIELD(3): SBAM[(17]
FIELD(4): SM[(14;15]
FIELD(5): SBM[(16][19]
FIELD(6): SALU[(24][28]
FIELD(7): SPS[(29][31]
FIELD(8): SA[(32][35]
FIELD(9): USE[(36][38]
FIELD(10): CB[(39]
FIELD(11): WR[(42][43]
FIELD(12): CLK[(46][47]
FIELD(13): XUPF+UPF[(48][59]
FIELD(14): EST+MSG[(59][63]
FIELD(15): LM[(64][67]
FIELD(16): RML[(68][71]
FIELD(17): SC[(72][75]
FIELD(18): EMD[(76][79]
FIELD(19): COF[(80][83]
FIELD(20): CB[(84][87]
FIELD(21): CKOFF[(88]
FIELD(22): PPE[(89][97]

MOP 1 ADD *GPR 3 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 ADOR
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 ADOR
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 ADOR
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 ADOR
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 ADOR
FIELD 14 = 1
FIELD 18 WILL BE DETERMINED BY EMIT
FIELD 19 = 1
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  TO3  O  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

MOP  27  RSMK  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 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 MOVER *GPR B A P1
FIELD 1 WILL BE DETERMINED 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 MOOP XUPF *LABEL
FIELD 12= 1
FIELD 13 WILL BE DETERMINED 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 DETERMINED 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 DETERMINED BY NEXT ADDR
FIELD 14= 7
FIELD 15= 0
FIELD 16= 15
FIELD 17 WILL BE DETERMINED 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 DETERMINED BY 15-CT
FIELD 16= 15
FIELD 17= 0
FIELD 19= 1
FIELD 22= 1
THE REST FIELDS ARE NOT USED
MOP 35 MOVEL9 *GPR D P2
FIELD 1 WILL BE DETERMINED 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  
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  
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  
FIELD 3 = 0  
FIELD 10 = 1  
FIELD 12 = 2  
FIELD 13 WILL BE DETERMINED BY NEXT ADDR  
FIELD 14 = 3  
FIELD 18 WILL BE DETERMINED BY VAR  
FIELD 22 = 1  
THE REST FIELDS ARE NOT USED  
MOP 39 CALL *LABEL  
FIELD 12 = 2  
FIELD 13 WILL BE DETERMINED BY NEXT ADDR  
FIELD 14 = 11  
FIELD 15 = 7  
FIELD 16 = 15  
FIELD 17 = 0  
FIELD 22 = 1  
THE REST FIELDS ARE NOT USED  
MOP 40 RETURN RETADDR EUBC P2  
FIELD 12 = 2  
FIELD 13 WILL BE DETERMINED BY NEXT ADDR  
FIELD 14 = 11  
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

**Simple IMLE Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>SRC1</th>
<th>SRC2</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>SRC1</td>
<td>SRC2</td>
<td>DEST</td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE1</td>
<td>SRC2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>SRC1</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>MOVES</td>
<td>D</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>FLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Simple IMLE Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>SRC1</th>
<th>SRC2</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>SRC1</td>
<td>SRC2</td>
<td>DEST</td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE1</td>
<td>SRC2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>SRC1</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>MOVES</td>
<td>D</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>FLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Simple IMLE Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>SRC1</th>
<th>SRC2</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>SRC1</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>SRC1</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>MOVES</td>
<td>D</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>FLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Simple IMLE Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>SRC1</th>
<th>SRC2</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB</td>
<td>SRC1</td>
<td>SRC2</td>
<td>DEST</td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE1</td>
<td>SRC2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>SRC1</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>MOVES</td>
<td>D</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>FLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Simple IMLE Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>SRC1</th>
<th>SRC2</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>SRC1</td>
<td>SRC2</td>
<td>DEST</td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE1</td>
<td>SRC2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>SRC1</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>MOVES</td>
<td>D</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>FLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Simple IMLE Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>SRC1</th>
<th>SRC2</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>SRC1</td>
<td>SRC2</td>
<td>DEST</td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE1</td>
<td>SRC2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>SRC1</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>MOVES</td>
<td>D</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>FLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Simple IMLE Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>SRC1</th>
<th>SRC2</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHR</td>
<td>SRC1</td>
<td>5,1</td>
<td>DEST</td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH2</td>
<td>6557E</td>
<td>TOS</td>
<td></td>
</tr>
<tr>
<td>SIMPLE IML CODE</td>
<td>SRC</td>
<td>DEST</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>SLH SRC 6,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH2 65535</td>
<td></td>
<td>TOS</td>
<td></td>
</tr>
<tr>
<td>LMASK TOS 6</td>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>PUSH1 SRC</td>
<td></td>
<td>TOS</td>
<td></td>
</tr>
<tr>
<td>ORLSM A TOS,6</td>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>MOVE5 D</td>
<td></td>
<td>DEST</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIMPLE IML CODE</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLH SRC 3,0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH1 SRC</td>
<td></td>
<td>TOS</td>
</tr>
<tr>
<td>LSMK TOS</td>
<td></td>
<td>NEWCHARA00D</td>
</tr>
<tr>
<td>MOVE5 D</td>
<td></td>
<td>DEST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIMPLE IML CODE</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLH SRC 4,0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH1 SRC</td>
<td></td>
<td>TOS</td>
</tr>
<tr>
<td>RLSM TOS</td>
<td></td>
<td>NEWCHARA010</td>
</tr>
<tr>
<td>MOVE5 D</td>
<td></td>
<td>DEST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIMPLE IML CODE</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMOVE MEM SRC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE3 SRC</td>
<td></td>
<td>BA</td>
</tr>
<tr>
<td>MOVE4 UNIBUS</td>
<td></td>
<td>DEST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIMPLE IML CODE</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMOVE MEM SRC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE2 SRC</td>
<td></td>
<td>BA</td>
</tr>
<tr>
<td>MOVE5 DEST</td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>MOVE9 DEST</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIMPLE IML CODE</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC SRC1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE CORRESPONDING MOPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC SRC1</td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>MOVE5 SRC1</td>
<td></td>
<td>SRC1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIMPLE IML CODE</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET SRC1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SIMPLE IML CODE
IN B SRC1
THE CORRESPONDING MOPS
IN B SRC1
MOVE5 D SRC1

SIMPLE IML CODE
CLR SRC1
CLR D SRC1
MOVE5 D SRC1

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

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

SIMPLE IML CODE
*COMP SRC1 DEST
THE CORRESPONDING MOPS
MOVE1 DEST 8
SUP SRC1 B D
FLAG

SIMPLE IML CODE
*COMP SRC1 8
THE CORRESPONDING MOPS
MOVE7 8 B
SUP SRC1 B D
FLAG

SIMPLE IML CODE
COND8 SRC,7 LABEL?
THE CORRESPONDING MOPS
PUSH1 SRC TOS
RMASK1 TOS 7 EU3C
WUP XUPF 831
BFC4 LABEL2 831 1
SIMPLE IML CODE

COND F IN

THE CORRESPONDING MOPS

PUSH3 PS TOS TOS
RMAK1 TOS 3 XUPF P.002
XUPF P.002 1

SIMPLE IML CODE

COND T C

THE CORRESPONDING MOPS

PUSH3 PS TOS TOS
RMAK1 TOS 0 XUPF P.003
XUPF P.003+1 1

SIMPLE IML CODE

COND .SR,8

THE CORRESPONDING MOPS

PUSH4 SR TOS TOS
RMAK1 TOS 8 XUPF P.004
XUPF P.004+1 1

SIMPLE IML CODE

BRCH LABEL

THE CORRESPONDING MOPS

BRCH LABEL

SIMPLE IML CODE

SLCT SRC1 C3

* SRC1 00 SSUBR1
* 01 SSUBR2
* 02 SSUBR3
*

THE CORRESPONDING MOPS

PUSH1 SRC1 TOS TOS
L4SK1 TOS 2 XUPF P.005
XUPF P.005+1 2
XUPF P.005+1 2

SIMPLE IML CODE

XEQ SUBR

* THE CORRESPONDING MOPS

PUSH CALL SUBR
SIMPLE INL CODE
RET
THE CORRESPONDING MCDS
RETURN RETAOR
NOOP1 XUPF 0
EU3C
APPENDIX D

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


**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.01X
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.01X
  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 ABO CD0
NOT AB1 CD1

THIS IML CODE IS BASED ON VM WORDSIZE
RMOVE MEM A3 CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
RMOVE MEM ABO CD0
RMOVE MEM AB1 CD1

THIS IML CODE IS BASED ON VM WORDSIZE
WMOVE MEM A3 CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
WMOVE MEM ABO 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 ABO
CLR AB1

THIS IML CODE IS BASED ON VM WORDSIZE
DEC AB
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
*DEC ABO
COND C L.0OG
INC AB1
L.0OG DEC AB1

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

THIS IML CODE IS BASED ON VM WORDSIZE
MOVE AB CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
MOVE ABO 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 WORCSIZE
MOVE C1234567 CD
THE FOLLOWING IML CODES ARE BASED ON HM WORCSIZE
MOVE C54919 CD0
MOVE C19 CD1

THIS IML CODE IS BASED ON VM WORCSIZE
INC AB
THE FOLLOWING IML CODES ARE BASED ON HM WORCSIZE
*INC AB6
CONDF C L.00G
INC AB1
L.00G (NEXT IML)

THIS IML CODE IS BASED ON VM WORCSIZE
*COMP AB CD
THE FOLLOWING IML CODES ARE BASED ON HM WORCSIZE
NOT C30 CD0
NOT C01 CD1
*INC C00
CONDF C L.00Z
INC C01
L.00Z *ADD A80 CD0 TEMPO
CONDF C L.00W
INC AB1
L.00W *ADD AB1 CD1 TEMP1

THIS IML CODE IS BASED ON VM WORCSIZE
CONDF AB4 LABEL2
THE FOLLOWING IML CODES ARE BASED ON HM WORCSIZE
CONDF A804 LABEL2

THIS IML CODE IS BASED ON VM WORCSIZE
CONDT AB23 LABEL1
THE FOLLOWING IML CODES ARE BASED ON HM WORCSIZE
CONDT A817 LABEL1

THIS IML CODE IS BASED ON VM WORCSIZE
SHR A8 19,1 CD
THE FOLLOWING IML CODES ARE BASED ON HM WORCSIZE
SHR A81 3,1 CD0
MOVE C65535 CD1

THIS IML CODE IS BASED ON VM WORCSIZE
SHR A8 13,0 CD
THE FOLLOWING IML CODES ARE BASED ON HM WORCSIZE
SHR A81 2,0 CD0
MOVE C0 CD1
THIS IML CODE IS BASED ON VM WORDSIZE
    SHR  A9  5,0  CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
    SHR  A90  5,0  CD0
    EXTR  CHAPA00  A81  CD1
    OR  C01  C00  CD0
    SHR  A81  5,0  CD1

THIS IML CODE IS BASED ON VM WORDSIZE
    SHR  A8  6,1  CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
    SHR  A90  6,0  CD0
    EXTR  CHAPA01  A81  CD1
    OR  C01  C00  CD0
    SHR  A81  6,1  CD1

THIS IML CODE IS BASED ON VM WORDSIZE
    SHL  A9  5,0  CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
    SHL  A81  5,0  CD1
    SHR  A90  11,0  CD0
    OR  C01  C00  CD1
    SHL  A90  5,0  CD0

THIS IML CODE IS BASED ON VM WORDSIZE
    SHL  A8  6,1  CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
    SHL  A81  6,0  CD1
    SHR  A90  10,0  CD0
    OR  C01  C00  CD1
    SHL  A90  5,1  CD0

THIS IML CODE IS BASED ON VM WORDSIZE
    SHL  A9  18,0  CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
    SHL  A8Q  2,0  CD1
    MOVE  C0  CD0

THIS IML CODE IS BASED ON VM WORDSIZE
    SHL  A9  19,1  CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
    SHL  A8Q  2,1  CD1
    MOVE  C65535  CD0

THIS IML CODE IS BASED ON VM WORDSIZE
    EXTR  CHAPA1  A8  CD
THE FOLLOWING IML CODES ARE BASED ON HM WORDSIZE
    EXTR  CHAPA02  A80  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 cd0

Move CO 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 a90 5,0 tempo

Extr chara06 ab1 tempo1

Op tempo1 tempo tempo0

Shr a91 5,0 tempo1

Extr chara08 tempo0 cd0

Extr chara09 tempo1 cd1

This IML code is based on VM wordsize

Extr char5 ab cd

The following IML codes are based on HM wordsize

Shl a91 7,0 tempo1

Shr a80 9,0 tempo0

Or tempo1 tempo tempo1

Shl a80 7,0 tempo0

Extr chara11 tempo1 cd1

Move CO 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 26 21 0
APPENDIX E-1

IISG of Emulator PDP8
00A POP 8
00C , , 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 DASTASH , 12
005 CODE , 9, 11, -9
005 CRNTPG , 7, 11, 0
005 PGEAOR , 0, 6, 0
005 ROTFLG , 1, 3, 1
005 DSC , 3, 8, 3
005 OSB , 0, 2, 0
006 0
00B PROGRAMSTART
00F INF
206 MEM
206 IR
206 PC
00F INSTDC
206 IR
296 OPCD
306 G
306 EFTADR
207 MEM
207 IR
207 PC
209 MAR
123 ADR , 7
123 PCTEMP , 12
120 MART , 12
00F M2R
206 MAR
206 EM
206 MDR
206 OPCD
406 EFTADR
00F AND
206 ACCM
206 MDR
00F IAD
206 ACCM
206 MDR
206 LNK
00F ISZ
206 ACCM
206 MAR
206 PC
00F DCA
206 EM
206 ACCM
206 MAR
00F JMS
206 EM
206 MAR
206 PC
00F JMP
206 PC
APPENDIX E-2

IESG of Emulator PDP8
<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th>Description</th>
<th>(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000H</td>
<td>XEQ EFTADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001H</td>
<td>RMOVE MEM</td>
<td>MAR +T.010</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE -T.010</td>
<td>MOR C3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLCT OPCD</td>
<td>C0 SAND</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C1 STAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C2 SISZ</td>
<td></td>
</tr>
<tr>
<td>002H</td>
<td>AND ACCM</td>
<td>MDR ACCM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRCH BINF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>003H</td>
<td>ADD ACCM</td>
<td>MDR ACCM C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRCH BINF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>004H</td>
<td>RMOVE MEM</td>
<td>MAR +T.011</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD -T.011</td>
<td>C1 +T.012</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WMOVE MEM</td>
<td>MAR -T.012</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RMOVE MEM</td>
<td>MAR +T.013</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*COMP -T.013</td>
<td>C0 Z</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDF Z</td>
<td>TL.006</td>
<td></td>
</tr>
<tr>
<td>005H</td>
<td>ING PC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>006H</td>
<td>BRCH EINF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>007H</td>
<td>WMOVE MEM</td>
<td>MAR ACCM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLK ACCM</td>
<td>BRCH BINF</td>
<td></td>
</tr>
<tr>
<td>008H</td>
<td>XEQ EFTADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>009H</td>
<td>WMOVE MEM</td>
<td>MAR PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INC MAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE MAR</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRCH BINF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00AH</td>
<td>XEQ EFTADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00BH</td>
<td>WMOVE MEM</td>
<td>MAR PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INC MAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE MAR</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRCH BINF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00CH</td>
<td>XEQ EFTADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00DH</td>
<td>MOVE MAR</td>
<td>PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRCH BINF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX E-3

Output of Pass 1
OUTPUT OF PASS1

BKS

INF

<table>
<thead>
<tr>
<th>MOVE</th>
<th>1+PC</th>
<th>BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>UNIBUS</td>
<td>2+T.001</td>
</tr>
<tr>
<td>MOVE</td>
<td>*2-T.001</td>
<td>D</td>
</tr>
<tr>
<td>MOVE</td>
<td>D</td>
<td>1-IR</td>
</tr>
<tr>
<td>MOVE</td>
<td>16</td>
<td>B</td>
</tr>
<tr>
<td>ADD</td>
<td>*1+PC</td>
<td>B</td>
</tr>
<tr>
<td>MOVE</td>
<td>D</td>
<td>1-PC</td>
</tr>
</tbody>
</table>

BKS

INSTDC

<table>
<thead>
<tr>
<th>PUSH1</th>
<th>1-IR</th>
<th>TOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSMK</td>
<td>TOS</td>
<td>OPC0DE</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+T.002</td>
</tr>
<tr>
<td>MOVE3</td>
<td>*2-T.002</td>
<td>D</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>1+OPCD</td>
</tr>
<tr>
<td>PUSH1</td>
<td>*1-OPCD</td>
<td>TOS</td>
</tr>
<tr>
<td>RSMK</td>
<td>TOS</td>
<td>NEWCHAR000</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+T.003</td>
</tr>
<tr>
<td>PUSH1</td>
<td>*2-T.00X</td>
<td>TOS</td>
</tr>
<tr>
<td>LMASK1</td>
<td>TOS</td>
<td>EUBC</td>
</tr>
<tr>
<td>NOOP</td>
<td>XUPF</td>
<td>P.001</td>
</tr>
<tr>
<td>UNJP</td>
<td>MRI</td>
<td>P.001+01</td>
</tr>
<tr>
<td>UNJP</td>
<td>MRI</td>
<td>P.001+02</td>
</tr>
<tr>
<td>UNJP</td>
<td>DCA</td>
<td>P.001+03</td>
</tr>
<tr>
<td>UNJP</td>
<td>JMS</td>
<td>P.001+04</td>
</tr>
<tr>
<td>UNJP</td>
<td>JMP</td>
<td>P.001+05</td>
</tr>
<tr>
<td>UNJP</td>
<td>IO</td>
<td>P.001+06</td>
</tr>
<tr>
<td>UNJP</td>
<td>OPT</td>
<td>P.001+07</td>
</tr>
</tbody>
</table>

BKS

EFTADR

<table>
<thead>
<tr>
<th>PUSH1</th>
<th>1+IR</th>
<th>TOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSMK</td>
<td>TOS</td>
<td>PGEADR</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+T.003</td>
</tr>
<tr>
<td>MOVE3</td>
<td>*2-T.003</td>
<td>D</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+ADR</td>
</tr>
<tr>
<td>PUSH1</td>
<td>*1-IR</td>
<td>TOS</td>
</tr>
<tr>
<td>RSMK</td>
<td>TOS</td>
<td>EUBC</td>
</tr>
<tr>
<td>NOOP</td>
<td>XUPF</td>
<td>P.002</td>
</tr>
<tr>
<td>BRCH</td>
<td>L.001</td>
<td>P.002+01</td>
</tr>
<tr>
<td>MOVE7</td>
<td>16</td>
<td>B</td>
</tr>
<tr>
<td>SUB</td>
<td>*1-PC</td>
<td>B</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+PCTEMP</td>
</tr>
<tr>
<td>PUSH1</td>
<td>*2-PCTEMP</td>
<td>TOS</td>
</tr>
<tr>
<td>RSMK</td>
<td>TOS</td>
<td>CRNPFG</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+T.004</td>
</tr>
<tr>
<td>MOVE3</td>
<td>*2-T.004</td>
<td>D</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+PCTEMP</td>
</tr>
<tr>
<td>MOVE1</td>
<td>*2+AD4</td>
<td>B</td>
</tr>
<tr>
<td>OR</td>
<td>*2-PCTEMP</td>
<td>B</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>1+MAR</td>
</tr>
<tr>
<td>BRCH</td>
<td>L.002</td>
<td>P.002</td>
</tr>
<tr>
<td>MOVE3</td>
<td>*2-ADR</td>
<td>D</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>1+MAR</td>
</tr>
<tr>
<td>L.002</td>
<td>PUSH1</td>
<td>1-IR</td>
</tr>
<tr>
<td>RSMK</td>
<td>TOS</td>
<td>EUBC</td>
</tr>
<tr>
<td>NOOP</td>
<td>XUPF</td>
<td>P.003</td>
</tr>
<tr>
<td>BRCH</td>
<td>L.003</td>
<td>P.003+01</td>
</tr>
<tr>
<td>MOVE8</td>
<td>*1+MAR</td>
<td>B</td>
</tr>
<tr>
<td>MOVE4</td>
<td>UNIBUS</td>
<td>2+T.005</td>
</tr>
<tr>
<td>MOVE3</td>
<td>*2-T.005</td>
<td>D</td>
</tr>
<tr>
<td>MOVE5</td>
<td>D</td>
<td>2+MART</td>
</tr>
<tr>
<td>MOVE7</td>
<td>128</td>
<td>B</td>
</tr>
</tbody>
</table>
SUB *1+MAR B D
FLAG
PUSH3 PS TOS 3 TOS
RMASK1 TOS 3 EUBC
NOOP XUFP P.005
BRCH L.005 P.005 1
MOVE7 16 B
ADD *2+MAR B D
MOVE5 D *2+T.007 D
MOVE3 *2+T.007 D
MOVE5 D *2+MAR BA
MOVE2 *1+MAR BA
MOVE9 *2-T.007 D
NOOP
L.005
NOOP
MOVE5 *2-MART D
MOVE3 *2-MART D
L.003 RETURN RETADR EUBC
NOOP1 XUFP 0

BKS

MRI

PUSH EFTADR
CALL EFTADR
MOVE1 *1-MDR 3A
MOVE4 UNIBUS *2+T.010
MOVE3 *2-T.010 D
MOVE5 D *1-MDR
PUSH1 *1-OPCO TOS
RMASK1 TOS NEWCHARA01
MOVE5 D *2+T.00X TOS
PUSH1 *2-T.00X TOS
LMASK1 TOS 2 EUBC
NOOP XUFP 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 INF
UNJP INF

BKS

TAO

MOVE1 *1-MDR B
ADD *1+ACCM B D
MOVE5 D *1-ACCM
FLAG
PUSH3 PS TOS
RMASK1 TOS NEWCHARA02
MOVE5 D *1+LNK
PUSH1 *1+LNK TOS
LSMK TOS NEWCHARA03
MOVE5 D *1-LNK
UNJP INF

BKS

ISZ

MOVE1 *1-MAR BA
MOVE4 UNIBUS *2+T.011
MOVE7 16 B
ADD *2-T.011 B
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
BRC 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
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 0 *1+MAR
MOVE3 *1-MAR D
MOVE5 0 *1-PC
UNJP INF
BKS JMP
PUSH CALL EFTADR
MOVE3 *1-MAR D
MOVE5 0 *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

<table>
<thead>
<tr>
<th>MOVE</th>
<th>PC</th>
<th>D</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 6</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD12</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD14</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD18</td>
<td>2048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD19</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE REST FIELDS ARE NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| FIELD 1 | 2   |    |    |
| FIELD 2 | 1   |    |    |
| FIELD 4 | 2   |    |    |
| FIELD11 | 3   |    |    |
| FIELD12 | 3   |    |    |
| THE REST FIELDS ARE NOT USED | | | |

<table>
<thead>
<tr>
<th>INF</th>
<th>MOVEA</th>
<th>R13</th>
<th>BA</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD10</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD12</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD21</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE REST FIELDS ARE NOT USED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOVE4</th>
<th>UNIBUS</th>
<th>R12</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD11</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD12</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE REST FIELDS ARE NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOVE3</th>
<th>R12</th>
<th>D</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 6</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD12</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD19</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE REST FIELDS ARE NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOVE5</th>
<th>D</th>
<th>R12</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD11</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE REST FIELDS ARE NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOVE7</th>
<th>16</th>
<th>B</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD14</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD18</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD20</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THE REST FIELDS ARE NOT USED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADD</th>
<th>R13</th>
<th>B</th>
<th>D</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD 6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIELD12</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
THE REST FIELDS ARE NOT USED

FIELD 1 = 2
FIELD 2 = 1
FIELD 4 = 2
FIELD11 = 3
FIELD12 = 3
THE REST FIELDS ARE NOT USED

INSTOC 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

FIELD 1 = 4
FIELD 2 = 1
FIELD 6 = 0
FIELD12 = 2
FIELD19 = 1
THE REST FIELDS ARE NOT USED

FIELD 1 = 4
FIELD 2 = 1
FIELD 4 = 2
FIELD11 = 3
FIELD12 = 3
THE REST FIELDS ARE NOT USED

FIELD 1 = 4
FIELD 2 = 1
FIELD 6 = 0
FIELD12 = 2
FIELD19 = 1
THE REST FIELDS ARE NOT USED

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 NEWCHARADOD P2
FIELD 6 = 0
FIELD12 = 2
FIELD15 = 11
FIELD16 = 15
FIELD17 = 4
FIELD12 = 2
FIELD14 = 6
FIELD15 = 10
FIELD16 = 11
FIELD17 = 0
FIELD19 = 1
FIELD22 = 1
THE REST FIELDS ARE NOT USED

FIELD 1 =  5
FIELD 2 =  1
FIELD 4 =  2
FIELD11 =  3
FIELD12 =  3
THE REST FIELDS ARE NOT USED

FIELD 1 =  5
FIELD 2 =  1
FIELD 6 =  0
FIELD12 =  2
FIELD19 =  1
THE REST FIELDS ARE NOT USED

FIELD 1 =  5
FIELD 2 =  1
FIELD 4 =  2
FIELD11 =  3
FIELD12 =  3
THE REST FIELDS ARE NOT USED

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

SUP R13 B D P2
FIELD 1 =  2
FIELD 2 =  1
FIELD 5 =  0
FIELD 6 =  6
<table>
<thead>
<tr>
<th>Field 1</th>
<th>Field 2</th>
<th>Field 3</th>
<th>Field 4</th>
<th>Field 5</th>
<th>Field 6</th>
<th>Field 7</th>
<th>Field 8</th>
<th>Field 9</th>
<th>Field 10</th>
<th>Field 11</th>
<th>Field 12</th>
<th>Field 13</th>
<th>Field 14</th>
<th>Field 15</th>
<th>Field 16</th>
<th>Field 17</th>
<th>Field 18</th>
<th>Field 19</th>
<th>Field 20</th>
<th>Field 21</th>
<th>Field 22</th>
<th>Field 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 P9 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

FIELD 1 = 10
FIELD 2 = 1
FIELD 6 = 0
FIELD12 = 2
FIELD19 = 1

THE REST FIELDS ARE NOT USED

MOVE3  R7  D  P2
FIELD 1 = 9
FIELD 2 = 1
FIELD 3 = 1
FIELD14 = 1
FIELD12 = 1

THE REST FIELDS ARE NOT USED

MOVE2  R8  BA  P1
FIELD 1 = 8
FIELD 2 = 1
FIELD 3 = 1
FIELD14 = 1
FIELD12 = 1

THE REST FIELDS ARE NOT USED

NOOP
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
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
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD 1</td>
<td>8</td>
</tr>
<tr>
<td>FIELD 2</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 3</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 9</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 10</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 21</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 1</td>
<td>9</td>
</tr>
<tr>
<td>FIELD 2</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 4</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 11</td>
<td>3</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 4</td>
<td>0</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>3</td>
</tr>
<tr>
<td>FIELD 14</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 18</td>
<td>0</td>
</tr>
<tr>
<td>FIELD 20</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 7</td>
<td>3</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 4</td>
<td>0</td>
</tr>
<tr>
<td>FIELD 7</td>
<td>6</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>8</td>
</tr>
<tr>
<td>FIELD 22</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>2</td>
</tr>
<tr>
<td>FIELD 14</td>
<td>7</td>
</tr>
<tr>
<td>FIELD 15</td>
<td>0</td>
</tr>
<tr>
<td>FIELD 16</td>
<td>15</td>
</tr>
<tr>
<td>FIELD 17</td>
<td>2</td>
</tr>
<tr>
<td>FIELD 22</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 12</td>
<td>1</td>
</tr>
<tr>
<td>FIELD 4</td>
<td>0</td>
</tr>
</tbody>
</table>
FIELD12= 3
FIELD14= 1
FIELD18= 16
FIELD20= 1
THE REST FIELDS ARE NOT USED
ADD P13 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= 3
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
FIELD12= 1
FIELD14= 1
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  F12  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= 1
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  F13  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
MOVE3   R8    D    P2
FIELD  1=  8
FIELD  2=  1
FIELD  5=  0
FIELD12=  2
FIELD19=  1
THE REST FIELDS ARE NOT USED
MOVE5   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  P1
FIELD12= 1
FIELD14= 11
THE REST FIELDS ARE NOT USED
CALL  P2
FIELD12= 2
FIELD14= 3
FIELD22= 1
THE REST FIELDS ARE NOT USED
MOVE3  P2
FIELD 1= 8
FIELD 2= 1
FIELD 6= 0
FIELD12= 2
FIELD19= 1
THE REST FIELDS ARE NOT USED
MOVE5  R13  P3
FIELD 1= 2
FIELD 2= 1
FIELD 4= 2
FIELD11= 3
FIELD12= 3
THE REST FIELDS ARE NOT USED
MOVE12  IR  P1
FIELD 3= 1
FIELD10= 1
FIELD12= 1
FIELD14= 1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
MOVE9  R12  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
ID  NOOP
FIELD12= 1
THE REST FIELDS ARE NOT USED
MOVE12  IR  P1
FIELD 3= 1
FIELD10= 1
FIELD12= 1
FIELD14= 1
FIELD18= 3074
THE REST FIELDS ARE NOT USED
MOVE9  R12  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
<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT ACCR</th>
<th>(1279)</th>
<th>NEXT ADS</th>
<th>1278</th>
<th>MOVE1G</th>
<th>PC</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP4</td>
<td>4000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>6376</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>CNT ACCR</td>
<td>(1279)</td>
<td>NEXT ADS</td>
<td>1278</td>
<td>MOVE1G</td>
<td>PC</td>
<td>D</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>--------</td>
<td>----</td>
<td>---</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2375</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>CNT ACCR</td>
<td>(1277)</td>
<td>NEXT ADS</td>
<td>1276</td>
<td>INF</td>
<td>MOVE5</td>
<td>D</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>-----</td>
<td>--------</td>
<td>----</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2374</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>60220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>CNT ACCR</td>
<td>(1277)</td>
<td>NEXT ADS</td>
<td>1276</td>
<td>INF</td>
<td>MOVE5</td>
<td>R13</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>-----</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2373</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>46000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>CNT ACCR</td>
<td>(1276)</td>
<td>NEXT ADS</td>
<td>1275</td>
<td>MOVE4</td>
<td>UNIEUS</td>
<td>R12</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>--------</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2372</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>CNT ACCR</td>
<td>(1275)</td>
<td>NEXT ADS</td>
<td>1274</td>
<td>CPTM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2372</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>CNT ACCR</td>
<td>(1274)</td>
<td>NEXT ADS</td>
<td>1273</td>
<td>MOVE7</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>--------</td>
<td>----</td>
<td>---</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>6371</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>141000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>MOP</td>
<td>Count</td>
<td>Address</td>
<td>Next Address</td>
<td>Year</td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-------</td>
<td>---------</td>
<td>--------------</td>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>7</td>
<td>1273</td>
<td>NEXT ADS 1272</td>
<td>CPTM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146400</td>
<td>1272</td>
<td>INSTOC</td>
<td>FUSH1 R12 TOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>42367</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>8</td>
<td>1271</td>
<td>NEXT ADS 1270</td>
<td>PSMK TOS OFGCOE D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>2466F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>32366</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>9</td>
<td>1271</td>
<td>NEXT ADS 1269</td>
<td>CPTM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2365</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>10</td>
<td>1269</td>
<td>NEXT ADS 1268</td>
<td>PUSII R11 TOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>42364</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>11</td>
<td>1269</td>
<td>NEXT ADS 1267</td>
<td>PSMK TOS NEWCHAR1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>22373</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>12</td>
<td>1269</td>
<td>NEXT ADS 1267</td>
<td>PSMK TOS NEWCHAR1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>22373</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GROUP3  323E3
GROUP2  100400
GROUP1  0

GROUP0 MOP  13  CNT ACDR  2014  ( 1267)  NEXT ADS  1266  MOVES  0  R10
GROUP4  0
GROUP3  2362
GROUP2  146000
GROUP1  0

GROUP0 MOP  14  CNT ACDR  2015  ( 1266)  NEXT ADS  1265  PUSH1  R10  TOS
GROUP4  20000
GROUP3  423E1
GROUP2  100000
GROUP1  0

GROUP0 MOP  15  CNT ACDR  2016  ( 1265)  NEXT ADS  1264  LMASK1  TOS  3  EUCC
GROUP4  203E2
GROUP3  363E0
GROUP2  100000
GROUP1  0

GROUP0 MOP  16  CNT ACDR  2017  ( 1264)  NEXT ADS  1791  NOOP  XUPF  P.001
GROUP4  0
GROUP3  3377
GROUP2  40000
GROUP1  0

GROUP0 MOP  25  CNT ACDR  2020  ( 1263)  NEXT ADS  1262  EFTADR  PUSH1  R12  TOS
GROUP4  20000
GROUP3  423E5
GROUP2  100000
GROUP1  0

GROUP0 MOP  26  CNT ACDR  2021  ( 1262)  NEXT ADS  1261  PSMK  TOS  PGEADR  9
<table>
<thead>
<tr>
<th>Group</th>
<th>MOP</th>
<th>CNT ACF</th>
<th>2022</th>
<th>NEXT ADS</th>
<th>1260</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP0</td>
<td>0</td>
<td>CPTM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>20272</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>32355</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td>NEXT ADS</td>
<td>1260</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2354</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>100025</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>27</td>
<td>CNT ACF</td>
<td>2023</td>
<td>NEXT ADS</td>
<td>1259</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>42353</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>29</td>
<td>CNT ACF</td>
<td>2024</td>
<td>NEXT ADS</td>
<td>1258</td>
</tr>
<tr>
<td>GROUP4</td>
<td>25760</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>36352</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td>NEXT ADS</td>
<td>1258</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>30</td>
<td>CNT ACF</td>
<td>2025</td>
<td>NEXT ADS</td>
<td>1783</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>3367</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td>NEXT ADS</td>
<td>1782</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>32</td>
<td>CNT ACF</td>
<td>3011</td>
<td>NEXT ADS</td>
<td>1256</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>6350</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>141000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>MOP</td>
<td>CNT</td>
<td>ACOR</td>
<td>NEXT ACS</td>
<td>12E5</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2347</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100410</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>3000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>22</td>
<td>CNT</td>
<td>ACOR</td>
<td>NEXT ACS</td>
<td>1254</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2346</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>100031</td>
<td></td>
<td>2030</td>
<td>(1254)</td>
<td>1253</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>42345</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>31</td>
<td>CNT</td>
<td>ACOR</td>
<td>NEXT ACS</td>
<td>1252</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20117</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>32344</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2343</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>100030</td>
<td></td>
<td>2034</td>
<td>(1254)</td>
<td>1251</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2342</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>100030</td>
<td></td>
<td>2034</td>
<td>(1251)</td>
<td>1250</td>
</tr>
<tr>
<td>GROUP0</td>
<td>100031</td>
<td>CNT ACCR 2035</td>
<td>(1250)</td>
<td>NEXT ADS 1249</td>
<td>MOVE1</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>---------------</td>
<td>--------</td>
<td>---------------</td>
<td>-------</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2341</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>25</td>
<td>CNT ACCR 2036</td>
<td>(1249)</td>
<td>NEXT ADS 1248</td>
<td>CR R9</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2340</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>17060</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>31</td>
<td>CNT ACCR 2037</td>
<td>(1248)</td>
<td>NEXT ADS 1247</td>
<td>MOV5</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2337</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>100030</td>
<td>CNT ACCR 2040</td>
<td>(1247)</td>
<td>NEXT ADS 1246</td>
<td>MOVE12</td>
</tr>
<tr>
<td>GROUP4</td>
<td>6007</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>6336</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>200600</td>
<td>CNT ACCR 2041</td>
<td>(1246)</td>
<td>NEXT ADS 1245</td>
<td>MOVE9</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2335</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>120520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>31</td>
<td>CNT ACCR 2042</td>
<td>(1245)</td>
<td>NEXT ADS 1241</td>
<td>NOOP</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>MOP</td>
<td>CNT ACCR</td>
<td>NEXT ADS</td>
<td>L.001</td>
<td>MOVE3</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>----------</td>
<td>----------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>0</td>
<td>3010</td>
<td>(1793)</td>
<td>1242</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>2045</td>
<td>(1242)</td>
<td>1241</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>48</td>
<td>2046</td>
<td>(1241)</td>
<td>1240</td>
<td>L.002</td>
</tr>
<tr>
<td>6</td>
<td>49</td>
<td>2047</td>
<td>(1240)</td>
<td>1239</td>
<td>Rmask1</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
<td>2050</td>
<td>(1239)</td>
<td>1781</td>
<td>NOOF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>MOP</th>
<th>CNT ACCR</th>
<th>NEXT ADS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>100400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2332</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>146300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>52</td>
<td>CNT ADDR</td>
<td>3013</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2325</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 2</td>
<td>60220</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 0</td>
<td></td>
<td>20070</td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>53</td>
<td>CNT ADDR</td>
<td>2052</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2324</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 2</td>
<td>46000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 0</td>
<td></td>
<td>40031</td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>54</td>
<td>CNT ADDR</td>
<td>2053</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2323</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 2</td>
<td>146400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 0</td>
<td></td>
<td>100031</td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>55</td>
<td>CNT ADDR</td>
<td>2054</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 3</td>
<td>6322</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 2</td>
<td>141000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>56</td>
<td>CNT ADDR</td>
<td>2055</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2321</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 2</td>
<td>100410</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 1</td>
<td>3000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 0</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP</td>
<td>57</td>
<td>CNT ADDR</td>
<td>2056</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP 2</td>
<td>40000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>MOP</td>
<td>CNT/ADR</td>
<td>NEXT ADS</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>GROUP0</td>
<td>60000</td>
<td>58</td>
<td>1231</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td>59</td>
<td>1231</td>
</tr>
<tr>
<td>GROUP3</td>
<td>42317</td>
<td>60</td>
<td>1779</td>
</tr>
<tr>
<td>GROUP2</td>
<td>140000</td>
<td>61</td>
<td>1779</td>
</tr>
<tr>
<td>GROUP1</td>
<td>140000</td>
<td>62</td>
<td>1228</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td>63</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP4</td>
<td>21760</td>
<td>64</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP3</td>
<td>36316</td>
<td>65</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td>66</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td>67</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td>68</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td>69</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP3</td>
<td>2313</td>
<td>70</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP2</td>
<td>100410</td>
<td>71</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP1</td>
<td>30000</td>
<td>72</td>
<td>1227</td>
</tr>
<tr>
<td>GROUP0</td>
<td>30</td>
<td>73</td>
<td>1226</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td>74</td>
<td>1226</td>
</tr>
<tr>
<td>GROUP3</td>
<td>2313</td>
<td>75</td>
<td>1226</td>
</tr>
<tr>
<td>Group</td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>60000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>FUSH3</th>
<th>PS</th>
<th>TOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>2065</td>
<td>(1226)</td>
<td>1225</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP4</td>
<td>20000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP3</td>
<td>42311</td>
</tr>
<tr>
<td>GROUP2</td>
<td>140000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>140000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>FMASK1</th>
<th>TOS</th>
<th>3</th>
<th>EUBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>2066</td>
<td>(1225)</td>
<td>1224</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP4</td>
<td>21760</td>
</tr>
<tr>
<td>GROUP3</td>
<td>36310</td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
</tr>
<tr>
<td>GROUP1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOOF</th>
<th>XUPF</th>
<th>P.005</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>2067</td>
<td>(1224)</td>
<td>1777</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP3</td>
<td>3361</td>
</tr>
<tr>
<td>GROUP2</td>
<td>46000</td>
</tr>
<tr>
<td>GROUP1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOVES</th>
<th>16</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>3017</td>
<td>(1776)</td>
<td>1222</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP4</td>
<td>20</td>
</tr>
<tr>
<td>GROUP3</td>
<td>6306</td>
</tr>
<tr>
<td>GROUP2</td>
<td>141000</td>
</tr>
<tr>
<td>GROUP1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOVES</th>
<th>16</th>
<th>8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>2071</td>
<td>(1222)</td>
<td>1221</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP3</td>
<td>2305</td>
</tr>
<tr>
<td>GROUP2</td>
<td>106400</td>
</tr>
<tr>
<td>GROUP1</td>
<td>4400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP0</td>
<td>71</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>OPMK</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>2072</td>
<td>(1221)</td>
<td>1220</td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2304</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>MOP 72</td>
<td>CNT ACCF 2073</td>
<td>1220</td>
<td>NEXT ADS</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2303</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>MOP 73</td>
<td>CNT ACCF 2074</td>
<td>1219</td>
<td>NEXT ADS</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2302</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>MOP 74</td>
<td>CNT ACCF 2075</td>
<td>1218</td>
<td>NEXT ADS</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2301</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>120520</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>MOP 75</td>
<td>CNT ACCF 2076</td>
<td>1217</td>
<td>NEXT ADS</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>3361</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>MOP 76</td>
<td>CNT ACCF 3016</td>
<td>1777</td>
<td>NEXT ADS</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>3362</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>MOP</td>
<td>CNT</td>
<td>ADDR</td>
<td>NEXT ACS</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>77</td>
<td>3015</td>
<td>1776</td>
<td>1214</td>
</tr>
<tr>
<td>1</td>
<td>2276</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>14600</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2275</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ACS</th>
<th>L.004</th>
<th>MOVE</th>
<th>R.004</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
<td>2101</td>
<td>1214</td>
<td>1213</td>
<td>L.004</td>
<td>MOVE</td>
<td>R8</td>
</tr>
<tr>
<td>1</td>
<td>2275</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4020</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ACS</th>
<th>L.004</th>
<th>MOVE</th>
<th>R.004</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>79</td>
<td>2102</td>
<td>1213</td>
<td>1212</td>
<td>L.004</td>
<td>MOVE</td>
<td>R9</td>
</tr>
<tr>
<td>1</td>
<td>2275</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12052</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ACS</th>
<th>L.004</th>
<th>MOVE</th>
<th>R.004</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>80</td>
<td>2103</td>
<td>1212</td>
<td>1211</td>
<td>L.004</td>
<td>MOVE</td>
<td>R9</td>
</tr>
<tr>
<td>1</td>
<td>2275</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12052</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ACS</th>
<th>L.004</th>
<th>MOVE</th>
<th>R.004</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>81</td>
<td>2104</td>
<td>1211</td>
<td>1781</td>
<td>L.004</td>
<td>MOVE</td>
<td>R9</td>
</tr>
<tr>
<td>1</td>
<td>2275</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12052</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ACS</th>
<th>L.004</th>
<th>MOVE</th>
<th>R.004</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>82</td>
<td>3012</td>
<td>1781</td>
<td>1209</td>
<td>L.003</td>
<td>MOVET</td>
<td>RETADR</td>
</tr>
<tr>
<td>1</td>
<td>20377</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>36271</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>10000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP  83</td>
<td>CNT ACIR 2106 (1209) NEXT ACS 255</td>
<td>NOOP1 XUPF 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>377</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP  84</td>
<td>CNT ACIR 3000 (1791) NEXT ACS 1207 MRT PUSH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>56267</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP  85</td>
<td>CNT ACIR 2110 (1207) NEXT ACS 1263 CALL EFTADR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>2111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>16357</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP  86</td>
<td>CNT ACIR 2111 (1206) NEXT ACS 1205 MOVE8 R8 BA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2265</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>60220</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP  87</td>
<td>CNT ACIR 2112 (1205) NEXT ACS 1204 MOVE4 UNIBUS R10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2264</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>46000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOP  88</td>
<td>CNT ACIR 2113 (1204) NEXT ACS 1203 CPC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP</td>
<td>MOP</td>
<td>CNT</td>
<td>ADDR</td>
<td>NEXT ADS</td>
<td>COUNT</td>
<td>FUSH1</td>
<td>R11</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>GROUP4</td>
<td>99</td>
<td>100025</td>
<td>2114</td>
<td>(1203)</td>
<td>1202</td>
<td>PUSH1</td>
<td>R11</td>
</tr>
<tr>
<td>GROUP3</td>
<td>42262</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>90</td>
<td>24</td>
<td>2115</td>
<td>(1202)</td>
<td>1201</td>
<td>RSMK</td>
<td>TOS</td>
</tr>
<tr>
<td>GROUP4</td>
<td>22373</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>32261</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>91</td>
<td>0</td>
<td>2116</td>
<td>(1201)</td>
<td>1200</td>
<td>MOVE5</td>
<td>D</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>2260</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>92</td>
<td>100031</td>
<td>2117</td>
<td>(1200)</td>
<td>1199</td>
<td>PUSH1</td>
<td>R9</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>42257</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>93</td>
<td>31</td>
<td>2120</td>
<td>(1199)</td>
<td>1198</td>
<td>LMASK1</td>
<td>TOS</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20361</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>36256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>94</td>
<td>0</td>
<td>2121</td>
<td>(1198)</td>
<td>1775</td>
<td>MOP</td>
<td>XUPF</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td>3357</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOVE1</th>
<th>R10</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>3020</td>
<td>(1775)</td>
<td>1196</td>
<td>AND</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP3</td>
<td>2254</td>
</tr>
<tr>
<td>GROUP2</td>
<td>41000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP0</th>
<th>25</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOVE12</th>
<th>IR</th>
<th>BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>2123</td>
<td>(1196)</td>
<td>1195</td>
<td>CPTM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP3</td>
<td>2253</td>
</tr>
<tr>
<td>GROUP2</td>
<td>146400</td>
</tr>
<tr>
<td>GROUP1</td>
<td>15400</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP0</th>
<th>10020</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOVE12</th>
<th>IR</th>
<th>BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2124</td>
<td>(1195)</td>
<td>1194</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP4</th>
<th>6002</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP3</td>
<td>6252</td>
</tr>
<tr>
<td>GROUP2</td>
<td>40200</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP0</th>
<th>20000</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOVE9</th>
<th>R12</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>2125</td>
<td>(1194)</td>
<td>1193</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP3</td>
<td>2251</td>
</tr>
<tr>
<td>GROUP2</td>
<td>120520</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP0</th>
<th>23</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MOP</th>
<th>CNT</th>
<th>ADDR</th>
<th>NEXT ADS</th>
<th>MOOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>2126</td>
<td>(1133)</td>
<td>1277</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GROUP4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP3</td>
<td>2375</td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>Group</td>
<td>MOP</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>GROUP 0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2246</td>
</tr>
<tr>
<td>GROUP 2</td>
<td>41000</td>
</tr>
<tr>
<td>GROUP 1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 0</td>
<td>25</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2245</td>
</tr>
<tr>
<td>GROUP 2</td>
<td>146400</td>
</tr>
<tr>
<td>GROUP 1</td>
<td>4460</td>
</tr>
<tr>
<td>GROUP 0</td>
<td>100020</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2244</td>
</tr>
<tr>
<td>GROUP 2</td>
<td>40000</td>
</tr>
<tr>
<td>GROUP 1</td>
<td>60000</td>
</tr>
<tr>
<td>GROUP 0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>20000</td>
</tr>
<tr>
<td>GROUP 3</td>
<td>42243</td>
</tr>
<tr>
<td>GROUP 2</td>
<td>140000</td>
</tr>
<tr>
<td>GROUP 1</td>
<td>140000</td>
</tr>
<tr>
<td>GROUP 0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>20360</td>
</tr>
<tr>
<td>GROUP 3</td>
<td>32242</td>
</tr>
<tr>
<td>GROUP 2</td>
<td>100400</td>
</tr>
<tr>
<td>GROUP 1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP 3</td>
<td>2241</td>
</tr>
<tr>
<td>GROUP</td>
<td>MOP</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>GROUP0</td>
<td>110</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>10000</td>
</tr>
<tr>
<td>GROUP3</td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>111</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>100400</td>
</tr>
<tr>
<td>GROUP3</td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>146000</td>
</tr>
<tr>
<td>GROUP3</td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>114000</td>
</tr>
<tr>
<td>GROUP3</td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td>112</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td></td>
</tr>
<tr>
<td>GROUP0</td>
<td></td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td></td>
</tr>
<tr>
<td>GROUP3</td>
<td></td>
</tr>
<tr>
<td>GROUP4</td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Group 3</td>
<td>2375</td>
</tr>
<tr>
<td>Group 2</td>
<td>40000</td>
</tr>
<tr>
<td>Group 1</td>
<td>0</td>
</tr>
<tr>
<td>Group 0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 117</td>
<td>CNT ACDP 3022</td>
</tr>
<tr>
<td>Group 4</td>
<td>0</td>
</tr>
<tr>
<td>Group 3</td>
<td>2231</td>
</tr>
<tr>
<td>Group 2</td>
<td>60220</td>
</tr>
<tr>
<td>Group 1</td>
<td>0</td>
</tr>
<tr>
<td>Group 0</td>
<td>20030</td>
</tr>
<tr>
<td>MOP 118</td>
<td>CNT ACDP 2146</td>
</tr>
<tr>
<td>Group 4</td>
<td>0</td>
</tr>
<tr>
<td>Group 3</td>
<td>2230</td>
</tr>
<tr>
<td>Group 2</td>
<td>46000</td>
</tr>
<tr>
<td>Group 1</td>
<td>0</td>
</tr>
<tr>
<td>Group 0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 119</td>
<td>CNT ACDP 2147</td>
</tr>
<tr>
<td>Group 4</td>
<td>20</td>
</tr>
<tr>
<td>Group 3</td>
<td>6227</td>
</tr>
<tr>
<td>Group 2</td>
<td>141000</td>
</tr>
<tr>
<td>Group 1</td>
<td>0</td>
</tr>
<tr>
<td>Group 0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 120</td>
<td>CNT ACDP 2150</td>
</tr>
<tr>
<td>Group 4</td>
<td>0</td>
</tr>
<tr>
<td>Group 3</td>
<td>2226</td>
</tr>
<tr>
<td>Group 2</td>
<td>146400</td>
</tr>
<tr>
<td>Group 1</td>
<td>4400</td>
</tr>
<tr>
<td>Group 0</td>
<td>100031</td>
</tr>
<tr>
<td>MOP 121</td>
<td>CNT ACDP 2151</td>
</tr>
<tr>
<td>Group 4</td>
<td>0</td>
</tr>
<tr>
<td>Group 3</td>
<td>2225</td>
</tr>
<tr>
<td>Group 2</td>
<td>40200</td>
</tr>
<tr>
<td>Group 1</td>
<td>0</td>
</tr>
<tr>
<td>Group 0</td>
<td>20030</td>
</tr>
<tr>
<td>GROUP0</td>
<td>3000</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>MOP 128</td>
<td>CNT ACCR 2160</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP3</td>
<td>2216</td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>60000</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 129</td>
<td>CNT ACCR 2161</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20000</td>
</tr>
<tr>
<td>GROUP3</td>
<td>42215</td>
</tr>
<tr>
<td>GROUP2</td>
<td>140000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>140000</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 130</td>
<td>CNT ACCR 2162</td>
</tr>
<tr>
<td>GROUP4</td>
<td>21360</td>
</tr>
<tr>
<td>GROUP3</td>
<td>36214</td>
</tr>
<tr>
<td>GROUP2</td>
<td>10000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 131</td>
<td>CNT ACCR 2163</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP3</td>
<td>3347</td>
</tr>
<tr>
<td>GROUP2</td>
<td>40000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 133</td>
<td>CNT ACCR 3031</td>
</tr>
<tr>
<td>GROUP4</td>
<td>20</td>
</tr>
<tr>
<td>GROUP3</td>
<td>6212</td>
</tr>
<tr>
<td>GROUP2</td>
<td>141000</td>
</tr>
<tr>
<td>GROUP1</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>MOP 134</td>
<td>CNT ACCR 2165</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP3</td>
<td>3347</td>
</tr>
<tr>
<td>Group</td>
<td>MOP</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>135</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>139</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>140</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>141</td>
</tr>
<tr>
<td>GROUP0</td>
<td>23</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>0</td>
</tr>
<tr>
<td>GROUP0</td>
<td>0</td>
</tr>
<tr>
<td>GROUP4</td>
<td>0</td>
</tr>
<tr>
<td>GROUP2</td>
<td>0</td>
</tr>
</tbody>
</table>
GROUP1

GROUP0
MOP 154  CNT ADDR 2211  (1142)  NEXT ADS  1141  MOVE7  16  3
GROUP4  20
GROUP3  6165

GROUP2  146400
GROUP1  0

GROUP0
MOP 155  CNT ADDR 2212  (1141)  NEXT ADS  1140  CPTM
GROUP4  0
GROUP3  2164
GROUP2  146400
GROUP1  4400

GROUP0
MOP 156  CNT ADDR 2213  (1140)  NEXT ADS  1139  MOVE3  R8  0
GROUP4  0
GROUP3  2163
GROUP2  100400
GROUP1  0

GROUP0
MOP 157  CNT ADDR 2214  (1139)  NEXT ADS  1138  MOVE5  0  R13
GROUP4  0
GROUP3  2162
GROUP2  146000
GROUP1  0

GROUP0
MOP 158  CNT ADDR 2215  (1138)  NEXT ADS  1137  MOVE12  IR  0A
GROUP4  6002
GROUP3  6161
GROUP2  40200
GROUP1  0

GROUP0
MOP 159  CNT ADDR 2216  (1137)  NEXT ADS  1136  MOVE9  R12  0
GROUP4  0
GROUP3  2160
GROUP2  120520
GROUP1  0
GROUP0  23
MOP  160  CNT ACOF  2217  (1136)  NEXT ADS  1277  MOOP

GROUP4  0
GROUP3  2375
GROUP2  40000
GROUP1  0
GROUP0  0
MOP  162  CNT ACOF  3005  (1736)  NEXT ADS  1133  JMP  FUSH

GROUP4  0
GROUP3  56155
GROUP2  40000
GROUP1  0
GROUP0  0
MOP  163  CNT ACOF  2222  (1133)  NEXT ADS  1263  CALL  EFTADR

GROUP4  2223
GROUP3  16357
GROUP2  100000
GROUP1  0
GROUP0  0
MOP  164  CNT ACOF  2223  (1132)  NEXT ADS  1131  MOVE3  R8  0

GROUP4  0
GROUP3  2153
GROUP2  100400
GROUP1  0
GROUP0  30
MOP  165  CNT ACOF  2224  (1131)  NEXT ADS  1130  MOVE5  D  R13

GROUP4  0
GROUP3  2152
GROUP2  146000
GROUP1  0
GROUP0  100022
MOP  166  CNT ACOF  2225  (1130)  NEXT ADS  1129  MOVE12  IR  BA
GROUP0
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 NOOP

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>AND 215</td>
<td>/ clear ACCM</td>
</tr>
<tr>
<td>201</td>
<td>TAD 212</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>DCA 213</td>
<td>/ set counter to -64</td>
</tr>
<tr>
<td>203</td>
<td>TAD 214</td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>DCA 10</td>
<td>/ set adr(10) to 1777</td>
</tr>
<tr>
<td>205</td>
<td>TAD 10</td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>DCA I 10</td>
<td>/ used as an autoindex register</td>
</tr>
<tr>
<td>207</td>
<td>ISZ 213</td>
<td>check counter</td>
</tr>
<tr>
<td>210</td>
<td>JMP -.3</td>
<td>/ loop</td>
</tr>
<tr>
<td>211</td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>7700</td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>1777</td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The output is as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1777</td>
</tr>
<tr>
<td>2001</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2076</td>
<td>2075</td>
</tr>
<tr>
<td>2077</td>
<td>2076</td>
</tr>
</tbody>
</table>
R. Hickli

*.

MAIN. RT-11 MACRO VM02-12 25-MAY-78 01:26:34 PAGE 1

1 000000
2 000200
3 000200
4 004000
5 004000
6 004020
7 004020
8 004040
9 004040
10 004060
11 004060
12 004100
13 004100
14 004120
15 004120
16 004140
17 004140
18 004160
19 004160
20 004200
21 004200
22 004220
23 004220
24 004240
25 004240
26 004260
27 004260
28 004300
29 004300
30 004320
31 004320
32 004000
33 000000

ERRORS DETECTED: 0
FREE CORE: 18439 WORDS

*
2) Benchmark 2 — PDPT3

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>AND</td>
<td>215</td>
</tr>
<tr>
<td>201</td>
<td>TAD</td>
<td>212</td>
</tr>
<tr>
<td>202</td>
<td>DCA</td>
<td>213</td>
</tr>
<tr>
<td>203</td>
<td>TAD</td>
<td>214</td>
</tr>
<tr>
<td>204</td>
<td>DCA</td>
<td>216</td>
</tr>
<tr>
<td>205</td>
<td>DCA I</td>
<td>216</td>
</tr>
<tr>
<td>206</td>
<td>ISZ</td>
<td>216</td>
</tr>
<tr>
<td>207</td>
<td>ISZ</td>
<td>213</td>
</tr>
<tr>
<td>210</td>
<td>JMP</td>
<td>..3</td>
</tr>
<tr>
<td>211</td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>212</td>
<td></td>
<td>7700</td>
</tr>
<tr>
<td>213</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>214</td>
<td></td>
<td>2000</td>
</tr>
<tr>
<td>215</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>216</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

The output is as follows:

```
2600 0
.
.
.
2077 0
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000000</td>
<td>ASEC</td>
</tr>
<tr>
<td>2</td>
<td>000200</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>000200</td>
<td>=200</td>
</tr>
<tr>
<td>4</td>
<td>004000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>004000</td>
<td>=40000</td>
</tr>
<tr>
<td>6</td>
<td>004000</td>
<td>=4020</td>
</tr>
<tr>
<td>7</td>
<td>004020</td>
<td>=40320</td>
</tr>
<tr>
<td>8</td>
<td>004040</td>
<td>=4040</td>
</tr>
<tr>
<td>9</td>
<td>004040</td>
<td>=4060</td>
</tr>
<tr>
<td>10</td>
<td>006060</td>
<td>=4100</td>
</tr>
<tr>
<td>11</td>
<td>006060</td>
<td>=4120</td>
</tr>
<tr>
<td>12</td>
<td>006060</td>
<td>=4140</td>
</tr>
<tr>
<td>13</td>
<td>006060</td>
<td>=4160</td>
</tr>
<tr>
<td>14</td>
<td>006060</td>
<td>=4180</td>
</tr>
<tr>
<td>15</td>
<td>006060</td>
<td>=4200</td>
</tr>
<tr>
<td>16</td>
<td>006060</td>
<td>=4220</td>
</tr>
<tr>
<td>17</td>
<td>006060</td>
<td>=4240</td>
</tr>
<tr>
<td>18</td>
<td>006060</td>
<td>=4260</td>
</tr>
<tr>
<td>19</td>
<td>006060</td>
<td>=4300</td>
</tr>
<tr>
<td>20</td>
<td>006060</td>
<td>=4320</td>
</tr>
<tr>
<td>21</td>
<td>006060</td>
<td>=4340</td>
</tr>
<tr>
<td>22</td>
<td>006060</td>
<td>=4000</td>
</tr>
<tr>
<td>23</td>
<td>006060</td>
<td>=40000</td>
</tr>
<tr>
<td>24</td>
<td>006060</td>
<td>=40000</td>
</tr>
<tr>
<td>25</td>
<td>006060</td>
<td>=40000</td>
</tr>
</tbody>
</table>

**SYMBOL TABLE**

- **ABS.** 040000 000
- **ERRORS DETECTED:** 0
- **FREE CORE:** 18439 WORDS

*TT=*-PDP8T3

**ERRORS DETECTED:** 0

**FREE CORE:** 18439 WORDS
<table>
<thead>
<tr>
<th>RU SMALL</th>
<th>040000</th>
<th>000000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>040020</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040040</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040060</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040100</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040120</td>
<td>000020</td>
</tr>
<tr>
<td></td>
<td>040140</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040160</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040200</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040220</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040240</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040260</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040300</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040320</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040340</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040360</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040400</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040420</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040440</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040460</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040500</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040520</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040540</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040560</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040600</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040620</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040640</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040660</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040700</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040720</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040740</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>040760</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041000</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041020</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041040</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041060</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041100</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041120</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041140</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041160</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041200</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041220</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041240</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041260</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041300</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041320</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041340</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041360</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041400</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041420</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041440</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041460</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041500</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041520</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041540</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041560</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041600</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041620</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041640</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>041660</td>
<td>000000</td>
</tr>
</tbody>
</table>
3) Benchmark 3 — PDPT5

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>JMS 250</td>
<td>/jump to subroutine</td>
</tr>
<tr>
<td>201</td>
<td>TAD 10</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>DCA I 10</td>
<td>/used as autoindex register</td>
</tr>
<tr>
<td>203</td>
<td>ISZ 213</td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>JMP .-3</td>
<td>/loop</td>
</tr>
<tr>
<td>205</td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>7760</td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>1777</td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>AND 215</td>
<td>/clear ACCM</td>
</tr>
<tr>
<td>252</td>
<td>TAD 212</td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>DCA 213</td>
<td>/set count to -16</td>
</tr>
<tr>
<td>254</td>
<td>TAD 214</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>DCA 010</td>
<td>/set adr(10) to 1777</td>
</tr>
<tr>
<td>256</td>
<td>JMP I 250</td>
<td>/return</td>
</tr>
</tbody>
</table>

The output is as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1777</td>
</tr>
<tr>
<td>2001</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>2015</td>
</tr>
<tr>
<td>2017</td>
<td>2016</td>
</tr>
</tbody>
</table>
ERRORS DETECTED: 0
FREE CORE: 18436, WORDS

ERRORS DETECTED: 0
FREE CORE: 18436, WORDS

*
| Time | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Data 8 | Data 9 | Data 10 | Data 11 | Data 12 | Data 13 | Data 14 | Data 15 | Data 16 | Data 17 | Data 18 | Data 19 | Data 20 | Data 21 | Data 22 | Data 23 | Data 24 | Data 25 | Data 26 | Data 27 | Data 28 | Data 29 | Data 30 | Data 31 | Data 32 | Data 33 | Data 34 | Data 35 | Data 36 | Data 37 | Data 38 | Data 39 | Data 40 | Data 41 | Data 42 | Data 43 | Data 44 | Data 45 | Data 46 | Data 47 | Data 48 | Data 49 | Data 50 | Data 51 | Data 52 | Data 53 | Data 54 | Data 55 | Data 56 | Data 57 | Data 58 | Data 59 | Data 60 | Data 61 | Data 62 | Data 63 | Data 64 | Data 65 | Data 66 | Data 67 | Data 68 | Data 69 | Data 70 | Data 71 | Data 72 | Data 73 | Data 74 | Data 75 | Data 76 | Data 77 | Data 78 | Data 79 | Data 80 | Data 81 | Data 82 | Data 83 | Data 84 | Data 85 | Data 86 | Data 87 | Data 88 | Data 89 | Data 90 | Data 91 | Data 92 | Data 93 | Data 94 | Data 95 | Data 96 | Data 97 | Data 98 | Data 99 | Data 100 |