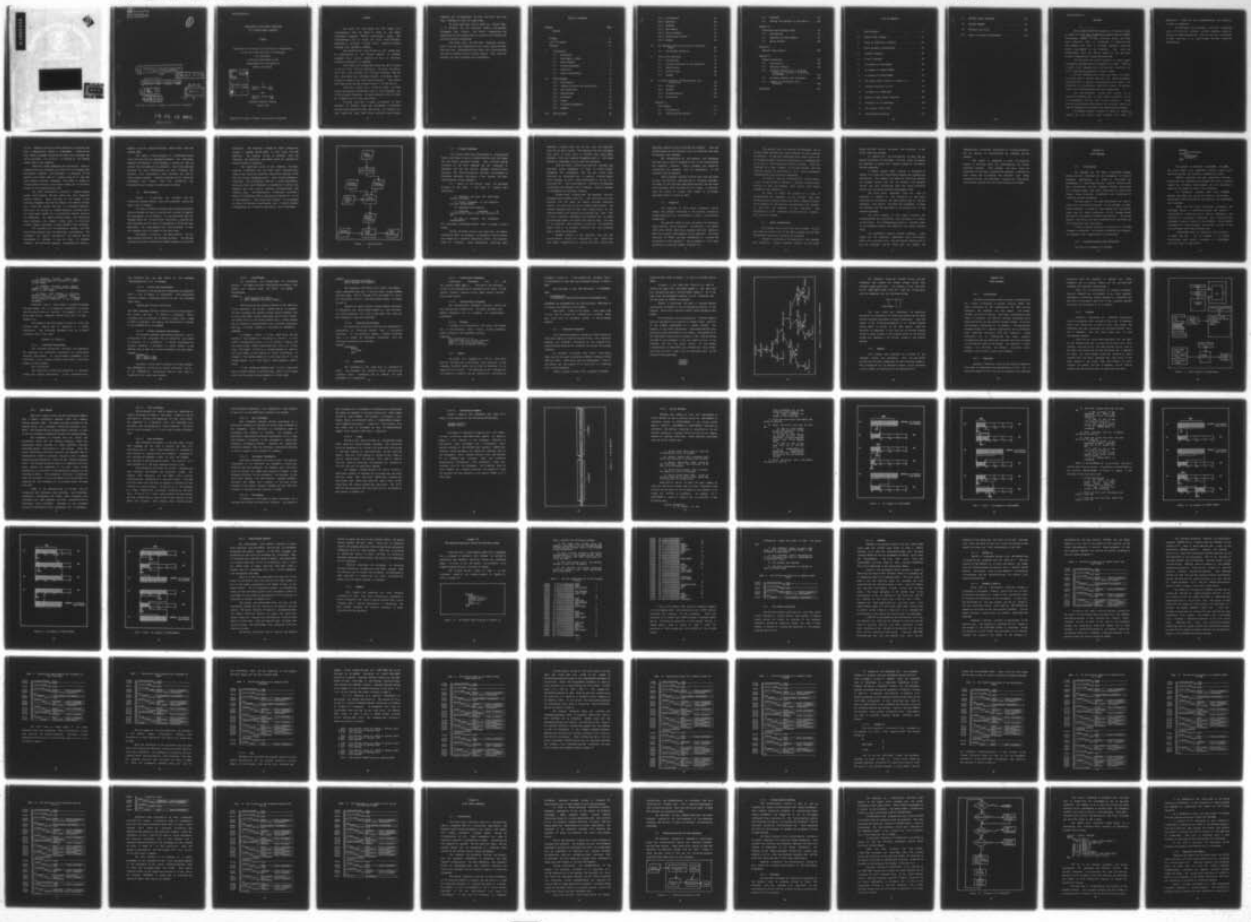
AD-A055 235 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
SIMULATION OF THE DIRECT EXECUTION OF A HIGHER ORDER LANGUAGE.(U)
MAR 78 D L AKIN, B C ALLEN

UNCLASSIFIED AFIT/GCS/EE/78-1. NL

1 OF 3
AD
A055 235



(14) AFIT/GCS/EE/78-1

(1)

(6) SIMULATION OF THE DIRECT EXECUTION OF A HIGHER ORDER LANGUAGE.

THESIS

(9) Master's Thesis,

AFIT/GCS/EE/78-1

(10) David L. /Akin 1st Lt USAF
Brian C. /Allen 2nd Lt USAF

(11) Mar 78

(12) 229 p.

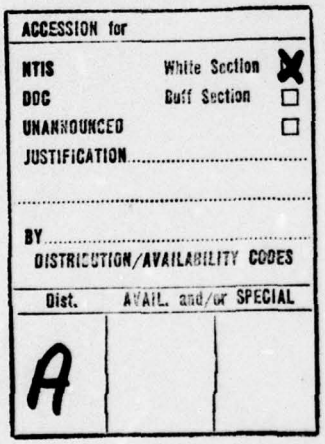
Approved for public release; distribution unlimited.

AFIT/GCS/EE/78-1

SIMULATION OF THE DIRECT EXECUTION OF A HIGHER ORDER LANGUAGE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science



by

David L. Akin, B.S.
1st Lt USAF

and

Brian C. Allen, B.S.
2nd Lt USAF

Graduate Computer Science

March 1978

Approved for public release; distribution unlimited.

Preface

The topic for this thesis was the result of a recommendation from the Phase II study of the Space Programming Language Machine Architecture Study. The recommendation was to recode the Clock Level Simulator developed in the Phase II study into a compiler oriented language (e.g. FORTRAN or JOVIAL).

This document has been written for the reader with an understanding of the various aspects of computer languages (e.g. syntax, compiler) as well as the basic hardware structures of a computer.

This thesis was accomplished using the DEC-10 system of the Air Force Avionics Laboratory. We found the DEC-10 to be very user oriented, and the quick response time and file structures were extremely helpful in working with a program as large as the Clock Level Simulator. This thesis was typed using the PUB document generator of the DEC-10.

Since this thesis was a follow-on study, we often needed and obtained information from Mr. Fred Gerbstadt, one of the original designers. We would therefore like to thank Mr. Gerbstadt for his help and advice.

We would also like to thank our sponsor Dr. Mark Michaels, Mr. Richard Sowar, and the members of Advanced Systems Group especially Capt. Dan Burton. Our readers, Dr. Gary Lamont and Capt. Pete Miller, provided many helpful

comments, and we appreciate the time that they took from their schedules to read the rough draft.

We would especially like to thank our advisor Capt. J. B. Peterson for his technical support and guidance throughout this project. His timely suggestions and enthusiasm for the project gave us incentive to overcome the various problems encountered.

This study could not have been completed, however, were it not for the cooperation of our wives, Nancy and Meg. They were very understanding when they were inconvenienced and were a constant source of moral support. Thus were are grateful for their patience and perseverance.

Table of Contents

List of Figures

Abstract

This thesis document lists the results of a follow-on study of Phase II of the Space Programming Language Machine (SPLM) Architecture Study (AD-A002 798). In this follow-on study, the Clock Level Simulator (CLS) was translated to JOVIAL/J73 and improvements were made to decrease execution times and memory requirements of the simulator. This work was accomplished using the DEC-10 system of the Air Force Avionics Laboratory at Wright-Patterson AFB.

CLS simulates the direct execution of SPLML (Space Programming Language Machine Language) on an SPLM. SPLML is a higher order language that was developed especially for avionic applications during the Phase I study.

CLS was developed as a design tool where the memory word size, memory address size, etc. are design parameters which represent a specific configuration of an SPLM. CLS, therefore, is a preliminary model for a family of machines (SPLM) with varying capabilities and complexities.

The initialization routine was written as an interactive program used to enter design parameters and is run separately from the rest of the simulator. A dump routine was developed which parses the simulated memory for ease of debugging the simulator and application programs.

An example program execution is given in detail in Chapter IV, and various other examples are listed in

Appendix D. There are also recommendations for follow-on studies in Chapter VI.

This document was developed to provide a starting point for follow-on studies, and the examples should be helpful in understanding CLS. The CLS code can be obtained in hard copy form or on tape through the AFIT School of Engineering.

SIMULATION OF THE DIRECT EXECUTION OF A HIGH ORDER LANGUAGE

Chapter I
Introduction

I.1 Background

Aerospace software has generally been programmed in assembly language, and the problems associated with coding, maintaining, and transferring programs between machines have been tremendous. Early in this decade, the Air Force became interested in the direct execution concept as a means of alleviating these problems. Direct execution of a higher order language means that a higher order language is used as the internal machine language of the computer.

In February 1971, the Air Force commissioned the System Integration Associates (SIA) to develop a computer architecture for a class of machines (Space Programming Language Machine) to execute directly the Space Programming Language (SPL). During this study (Phase I), SIA determined that SPL was not an appropriate language for direct execution and developed a new language (Space Programming Language Machine Language (SPLML)) and a meta-compiler. The meta-compiler accepts an SPLML syntax as input and outputs a

translator for the language. Thus when the language is changed, the new syntax is input to the meta-compiler and a new translator is generated.

During Phase I, SIA also developed a semantic simulator as an initial attempt to simulate SPLM. This semantic simulator was then developed into a clock level simulator (CLS) in Phase II. SIA envisioned CLS as a powerful design tool in the design of the Space Programming Language Machine (SPLM). CLS was thus developed as a parameterized statistical gathering mechanism which would provide the hardware engineer the capability of designing the most appropriate machine for a given application.

I.2 Advantages of SPLM

SPLM has several advantages in aerospace applications.

1) Compiler is eliminated.
2) Reduces debugging costs.
3) Reduces maintenance costs.
4) Reduces main memory.
5) Machine is smaller.
6) Software can be easily transferred between machines.

The following paragraphs explain these advantages in more detail.

Due to the direct execution implementation the compiler is replaced by a simple preprocessor (translator). Compilers have several disadvantages including the fact that they are costly to develop, costly to maintain, and costly

to run. Compiler errors are often difficult to pinpoint and have a demoralizing effect on programmers. These errors also reduce programmer productivity since the programmer may not be confident that an error is confined to his program rather than in the compiler.

SPLM will reduce debugging and maintenance costs of programs since the programs are written in the language that the machine executes. The programmer is debugging at the machine level rather than at a higher level as is the case with compiler oriented languages. This makes it easier to follow program execution since memory dumps can be directly related to the program source code.

The SPLM architecture provides a memory packing scheme which reduces the main memory and thus makes the machine smaller. Since the bulk of the weight of the aerospace computers has been the memory, this memory savings will reduce the weight and size of aerospace computers. This could be critical on weight sensitive missions where the added memory needed for a conventional computer would be too cumbersome. The memory packing scheme uses additional logic hardware, but present trends point to much more dramatic weight reductions in logic hardware than in memory.

The software can be transferred between any SPLM since these machines have the same basic architecture. The arithmetic and logical operators of the machine may be implemented in hardware or may be traps to software routines. The particular operator implementation, however,

affects only the program execution speed rather than the program code.

The Phase I study produced an architecture which could revolutionize the aerospace computer. The same basic computer architecture could navigate a missile as well as control the environment of an airplane. Since this approach provides for cost effectiveness of both hardware and software, this architecture could decrease the cost of future weapon systems. Software problems can be found and eliminated much faster; thus the reliability and maintainability of weapon systems would increase.

I.3 SPLM Software

Figure 1 illustrates the software and the interaction between modules that were developed during the Phase I and II studies.

The idea of deferred binding where binding decisions would be delayed as long as possible was followed throughout the development of the software. It was determined that the language would not be bound until application programs had been developed and the ease of programming in SPLML had been determined. The meta-compiler was thus developed so that the language could be changed as the need arose.

The syntax is input to the meta-compiler, and the meta-compiler generates the parsing routines. The parsing routines and driver routines are merged together to form the

translator. The translator accepts an SPLML program and outputs a program string which is then input into the simulator. The program string is executed with the simulator, and statistics and memory dumps are gathered at various points of execution.

The three main units of the software, the meta-compiler, translator, and the Clock Level Simulator were coded in APL since the excellent interactive features of APL made the initial debugging easier. This software package is a powerful aid in designing SPLM for a given application. Since the Clock Level Simulator is parametric in nature, the designer may test various hardware configurations with relative ease. The preprocessor (translator) provides a means of translating typical application programs to be run on the simulator. This allows the designer to investigate in detail the hardware requirements, and a minimal hardware configuration for a specific application can be developed.

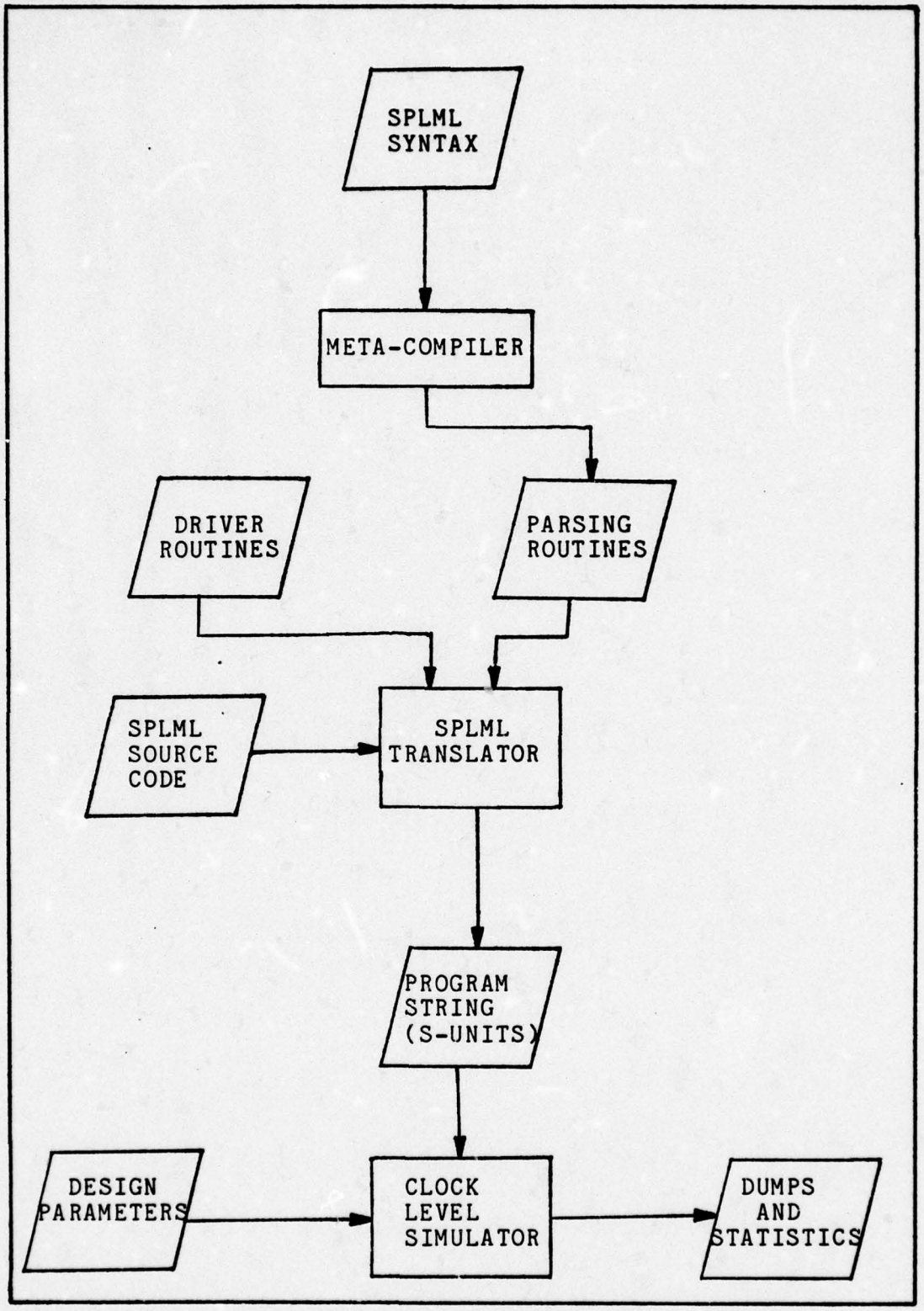


Figure 1. SPLM Software

I.4 Problem Statement

The initial effort was concentrated on studying the Phase I and Phase II work to become familiar with the design and the direct execution concept. This involved looking carefully at the SPLM language and the simulator to determine the way the various language constructs were executed. The SPLM architecture was then investigated to determine the basic structures of the machine and their representation in the simulator.

As a result of the initial study, the following problems of the Phase I and Phase II studies were identified:

1) Simulator was slow and used large amounts of memory.

2) Little debugging of the simulator had been accomplished.

3) Debugging was difficult.

4) Simulator contained an initialization routine that was difficult to use.

5) Lack of examples and inadequate documentation.

The following paragraphs explain these problems in more detail.

During the Phase II work, SIA found that APL memory limitations made it necessary to determine which routines were needed for a specific simulation, before the simulator could be executed. Large simulations requiring many

simulator routines could not be run, thus the simulator could not be fully tested. The simulator was also extremely slow since it took hours to execute even small SPLML programs. This also hampered debugging since it took large amounts of programmer time to test the simulator.

Thus the second problem was that the simulator had undergone little debugging. In order to enable further debugging, SIA recommended that the APL Clock Level Simulator be translated into a compiler oriented higher order language to allow large simulations to be run in a practical way. This investigation, therefore, encompassed translating the simulator into a compiler oriented language and debugging the simulator routines.

Another major problem with the simulator was that debugging was extremely difficult. This difficulty not only resulted from the speed and memory problems mentioned earlier, but also from a lack of a memory dump routine. The original designers did not feel that a dump routine for the simulated memory was feasible and discarded the idea. Thus in order to perform debugging, the designers had to output the simulated memory and reduce this data by hand. The lack of an automated data reduction routine for the simulated memory made it an extremely difficult and time consuming task to debug the simulator.

Another problem of the simulator was that the initialization routine was difficult to use. There were many inputs demanded by the routine and these inputs were

sometimes changed without notifying the designer. Thus the initialization of the simulator could be different from what the designer had intended.

The documentation of the simulator was inadequate and there was a lack of examples to aid in the understanding the simulator routines. Thus a document was needed to incorporate the examples with an explanation of the simulator and the machine.

These problems resulted from the fact that the Phase II work had not been completed due to financial problems. The Phase II work had been abandonned abruptly and the Clock Level Simulator (CLS) had been left in a state of disarray. Thus the result of 5 years work needed to be further developed before it could be used.

I.5 Objective

The objective of this study, therefore, was to remedy the problems mentioned in the previous section by redeveloping CLS into an economical, viable, and practical design tool.

The approach taken to solve the memory and execution speed problems was to translate the simulator into JOVIAL (J73) since JOVIAL offers many features that were needed to make the necessary improvements on the simulator. The table structure of JOVIAL was used extensively to increase the performance of the simulator, and an example of this table structure is contained Chapter V Section 2.3.

The approach used to continue the debugging was to develop SPLML programs that would exercise as many routines as possible. It was also determined that a dump routine for the simulated memory should be developed as a debugging aid to be used in this study and follow-on studies. Since the simulator is a large program, it was also decided that the simulator should be structured so as to avoid unnecessary recompilations and facilitate debugging.

In order to make the initialization routine easier to use, it was decided to develop a separate initialization routine so that the designer could easily enter design parameters (see Appendix A).

In order to remedy the problem of a lack of documentation and examples, it was decided that this document should include program execution examples (Chapter IV and Appendix D) and detailed explanations of the hardware of SPLM and the software of the simulator. Good documentation of the simulator routines was also needed to help follow-on studies.

I.6 Report Organization

The chapters that follow have been arranged to give the reader an overall view of the SPLM language and the SPLM architecture before the simulator is explained.

Chapter II provides an introduction to the language and translator. Various language features are discussed

along with the role of the syntax and translator in the direct execution concept.

In Chapter III, the architecture of SPLM and the machine operation are explained. The five major structures are examined in detail with emphasis placed on the memory packing mechanism.

A simple example SPLML program is presented in Chapter IV to show the relationship between the language and the machine. This example also shows the similarity between the program source code and the machine code (program string) and thus illustrates that the direct execution machine would decrease software development costs.

The Clock Level Simulator (CLS) for SPLM is described in Chapter V. The simulation of the major structures of the machine and improvements made to CLS are explained. This chapter discusses the advantages of using the simulator and the importance of this design tool to the hardware engineer.

The final chapter of this report contains the recommendations and conclusions that developed as a result of this study. The recommendations point to specific areas of the language, machine, and simulator that require further study.

The appendices contain program examples, a user manual for the simulator, departures from the original study, and a register list. The user manual explains how to use the simulator and the files that are needed. The

examples give a detailed description of program execution and are helpful in understanding the language and the machine.

This report is organized to give any follow-on studies a starting point for understanding the direct execution concept. This concept can only fully be understood by carefully studying the examples presented in this report and by studying the simulator code. These examples exercise many simulator routines and thus should provide good insight into the direct execution of SPLML.

Chapter II

SPLM Language

II.1 Introduction

The language for the Space Programming Language Machine has been developed for the programming of on-board aerospace applications. In order to facilitate the programming of these applications, several features have been incorporated into the language (SPLML): specialized I/O, variable length data items, operations on vectors and arrays, block structure, procedures, and declaration of interrupt invoked procedures.

The main goal of the SPLML development has been to provide for the direct execution of the language on SPLM. Direct execution in terms of the language and the machine means that all tokens (terminals) of the language except punctuation correspond to machine instructions. The actual machine instructions are called Semantic-Units (S-UNITS) and are incorporated into the syntax.

In the following sections, an overview of the SPLM language is presented. The syntax, translator and compactor are examined, and a simple parse is illustrated.

II.2 Program Structure and Definitions

The form of a program is a follows:

```
PROGRAM
     soft extensions
DEFAULT n;
     declarations
     statements
TERM
```

The program is delimited by PROGRAM and TERM, and since there is no supporting software (such as an operating system), the program is complete by itself.

DEFAULT separates the soft extensions from the declarations and statements and defines the default variable length for variables whose lengths are not declared (n specifies the number of bits in the variable).

An executable program must have PROGRAM, DEFAULT, and TERM; and any or all three of the program requirements (soft extensions, declarations and statements) may be omitted.

SPLML has two block structures: procedures and BEGIN/END blocks. A procedure provides the means of associating an identifier with a body of statements. The execution of a procedure identifier, with optional actual parameters, causes a branch to the procedure. The procedure statements are then executed, and an exit is made to the next statement after the procedure call.

A BEGIN/END block is a section of code delimited by BEGIN and END and is treated as an entity. Declarations made within a procedure or a BEGIN/END block are local to that block.

A simple SPLML program is given in Figure 2. This program executes some simple arithmetic and will be referenced throughout this report.

```
PROGRAM
   DEFAULT 4;
   INTEGER A BITS 5;
   INTEGER B[2,3];
   3>A;
   A+4>B[1,2]
TERM
```

Figure 2. Simple SPLML Program

II.3 Soft Extensions

Soft extensions to the processor are SPLML procedures used by the processor as traps for operators which are not fully implemented in hardware. When an operator which is not fully implemented in hardware is encountered (e.g. addition between two arrays), a jump is made to the soft extension and the operator is executed.

Soft extensions are used to save hardware cost at the expense of execution time. The hardware configuration is determined using the Clock Level Simulator as a design tool, and these configurations may vary widely between

different applications. Soft extensions, however, enable every SPLM to execute any SPLML program with only execution time variations.

II.4 Declarations

Any variable in an SPLML statement must have a preceding declaration, and declarations may occur in any block or body. Declarations create objects with particular attributes (i.e. type, length, and dimensions) and associate an identifier with that object. The identifier denotes the object complete with all of its declared attributes, and the object exists throughout the scope of its identifier. The scope of an identifier is the smallest block (i.e. proecdural or BEGIN/END) containing the declaration of the identifier.

II.4.1 Data Declarations

Data declarations define variables and constants of the following types: constant, boolean, cardinal, integer, and real. The default initial value after declaration is zero (false for boolean), and variables may receive new values by means of the assignment operator (→). Constant data can not be assigned new values but are given a value by the declaration. The four types of data objects (BOOLEAN, CARDINAL, INTEGER, REAL) are listed below along with their data ranges.

1) BOOLEAN- Variables which can have two values, true and false.

2) CARDINAL- Variable length data objects which range in [0,(2**n)-1], where n is the length.

3) INTEGER- Variable length signed integral values in [-((2**n)-1),(2**n)-1], where n is the length.

4) REAL- Data objects of the form A*(2**b) where A is variable length. A ranges in [-((2**n)-1),(2**n)-1]. b ranges in [-((2**k)-1),(2**k)-1]. k is a design parameter.

Dimension data is given within brackets following the data object, and if no dimension information is given, then the entire array is implied. For example, if A is an array then the the statement "A+4>A;" will add 4 to each element of A.

The length of the data object is given in bits, and multiple data objects may be declared in a single declaration. The following statement sets up 3 scalar variables, each 6 bits long.

INTEGER X,Y,Z BITS 6;

II.4.2 Reference Declaration

The reference declaration provides the capability for declaring an identifier equivalent to a previously declared data object. In the following statement, A is referenced to an already existing data object (the first element of the B array).

REF A TO B[0,0];

This construct provides the capability to reference arrays with simpler identifiers. In the statement above,

the reference has the same effect as the statement "EQUIVALENCE(A,B(1,1))" in FORTRAN.

II.4.3 Control Set Declaration

The control set declaration associates an identifier with a list of labels or procedures. This declaration provides a means of selecting control and has the following basic form:

```
CONTROL SET CCT(ECC1,ECC2,ECC3);
```

The SPLML statement "CCT[3];" selects the third data item in the control set list. If ECC3 is a procedure, then a procedure call is executed; if ECC3 is a label, then a jump to ECC3 is executed. The control set declaration is similar to the COMPUTED GO TO in FORTRAN.

II.4.4 Forward Procedure Declaration

The forward procedure declaration provides a way of specifying to the translator that an identifier will later be declared as a procedure. A forward declaration is required whenever a procedure is called before it was defined such as when two or more procedures call each other.

Example:

```
FORWARD Q;
PROC P BEGIN Q END;
PROC Q BEGIN P END;
```

Procedure P above calls Q before Q has been defined, thus "FORWARD Q;" is used as an initial identifier for Q. If the "FORWARD Q;" declaration had not been used, a translator error would have occured.

II.4.5 Input/Output

I/O operations are accomplished by allocating buffers within memory and then referencing the buffers. The buffers are defined using the INPUT and OUTPUT declarations.

Example:

```
1)  INPUT BOOLEAN A[5] PORT 0;
2)  INPUT CARDINAL B[2] BITS 6 PORT 1;
```

PORT specifies the physical address of the SPLM port on the interface, and the port number must be a constant. The port transfers bit streams (called stream elements) into or out of the memory buffer associated with the port number. The buffer size is one stream element and a stream element may be transferred simultaneously with processor execution. The type of stream element is restricted to BOOLEAN or CARDINAL.

In example 1 above, A is the input array and the length of each element in the array is one bit. A 5 bit buffer corresponding to port 0 is set aside in memory, and the input stream element associated with port 0 is 5 bits long. The number of bits in a stream element is equal to the total number of bits implied in a data declaration. In example 1, the stream element is 5 bits long; and in 2, the stream element is 12 bits long (2 array elements times 6 bits).

In the following example, port 3 is an input port and the stream element is 23 bits long. Port 4 is an output port, and the output stream element is 10 bits long.

Example:

```
INPUT BOOLEAN C[23] PORT 3;
OUTPUT BOOLEAN D[10] PORT 4;
```

The statement "C[6:15]→D[1:10];" fills the memory buffer associated with port 3 with an input stream element (23 bits long). Bits 6 through 15 of the buffer of port 3 fill the buffer of port 4, and the stream element is output to port 4.

Whenever an input (output) stream element identifier is referenced, the whole stream element is read (written). In a write operation, any bits in the buffer that have not been assigned a value are undefined.

II.4.6 Interrupt Declaration

The interrupt declaration provides the capability of associating an interrupt handler with a particular interrupt. In the following example, whenever interrupt line 3 is raised the statements associated with the interrupt declaration are executed.

Example:

```
INTERRUPT 3
      BEGIN
          A+B→C;
      END;
```

II.5 Statements

The statement is the basic unit of operation in SPLML. The statement may contain a block structure and procedure calls. Statements may be labeled and must terminate with a semicolon.

II.5.1 Conditional Statements

Conditional statements are of the "IF..formula..THEN..ELSE.." form where the execution of one of two alternatives is selected by the result of the boolean formula (result=1 follow THEN branch, result=0 follow ELSE branch).

II.5.2 Unconditional Statement

The unconditional statement provides control and data manipulation capabilities. The GOTO, DO-UNTIL, and RETURN statements are all examples of unconditional statements.

II.5.3 Formula

A formula is a computational rule which may contain any of the following: names, values, monadic operators, dyadic operators, and assignment operator.

Examples:

```
0→A; (value of A is 0)
ABS(X)→Y; (ABS is a monadic operator)
A+B→C; (+ is a dyadic operator)
A→B; (→ is the assignment operator)
```

II.6 Syntax

The syntax of a language is a set of rules which must be followed when constructing valid sentences of the language. The SPLML syntax can be found in Reference [2] on pages 2-2-13 to 2-2-25. The terminals of the language are surrounded by brackets, and the S-UNITS are surrounded by

triangles (▽ and △). A dot between two triangle (▽·△) is interpreted to mean that the preceding terminal is the S-UNIT.

The following is the BNF definition of STATEMENT-LIST:

```
STATEMENT-LIST
:=(STATEMENT) [SEMICOLON]▽CLEAN'UP△STATEMENT-LIST
```

STATEMENT and STATEMENT-LIST are non-terminals; SEMICOLON is a terminal and CLEAN'UP is its S-UNIT.

Emit lines (lines in the syntax which begin with ε) are used to govern the reordering of S-UNITS. These emit lines will be discussed further in the following section.

II.7 Translator/Compactor

The translator/compactor accepts an SPLML program as input and outputs an encoded Program String. The translator performs any necessary reordering of the program (e.g. Reverse Polish Notation for formulas) and checks the program validity.

As formulas are parsed, emit lines (lines which begin with ε) are used to generate S-UNITS in Reverse Polish Notation (RPN). These emit lines are supplemental lines in the syntax, and each symbol in an emit line is a function call in the translator.

Figure 3 shows a parse of the statement "A+B*C→D"

using the emit lines at points 1, 2, and 3 to reorder the S-UNITS.

At point 1, the emit line "Z←Z1,Z" is used to reorder the name B and the dyadic symbol +. The emit line at 2 reorders the name C and the dyadic symbol *. At 3, the name D and the assignment operator (→) are reordered, and the RPN parse of A+B*C→D is complete.

Figure 3 illustrates the ease of mapping machine instructions (S-UNITS) to the corresponding terminal in the program. This direct execution feature makes debugging much easier.

The translator also eliminates all terminal symbols that are punctuation and outputs an integer vector with all of the S-UNITS represented as a biased integer. For example, if the S-UNIT declaration CARD-CONST(LEN,VAL) were generated during translation, the integer representation of the CARD-CONST S-UNIT would be placed in the Program String followed by its parameters (LEN, the length in bits of the constant, and VAL, the actual value). If the integer representation of CARD-CONST were 10000024 and VAL=3 (LEN=2 since VAL is 2 bits long), then the following would be the Program String generated:

```
10000024
00000002
00000003
```

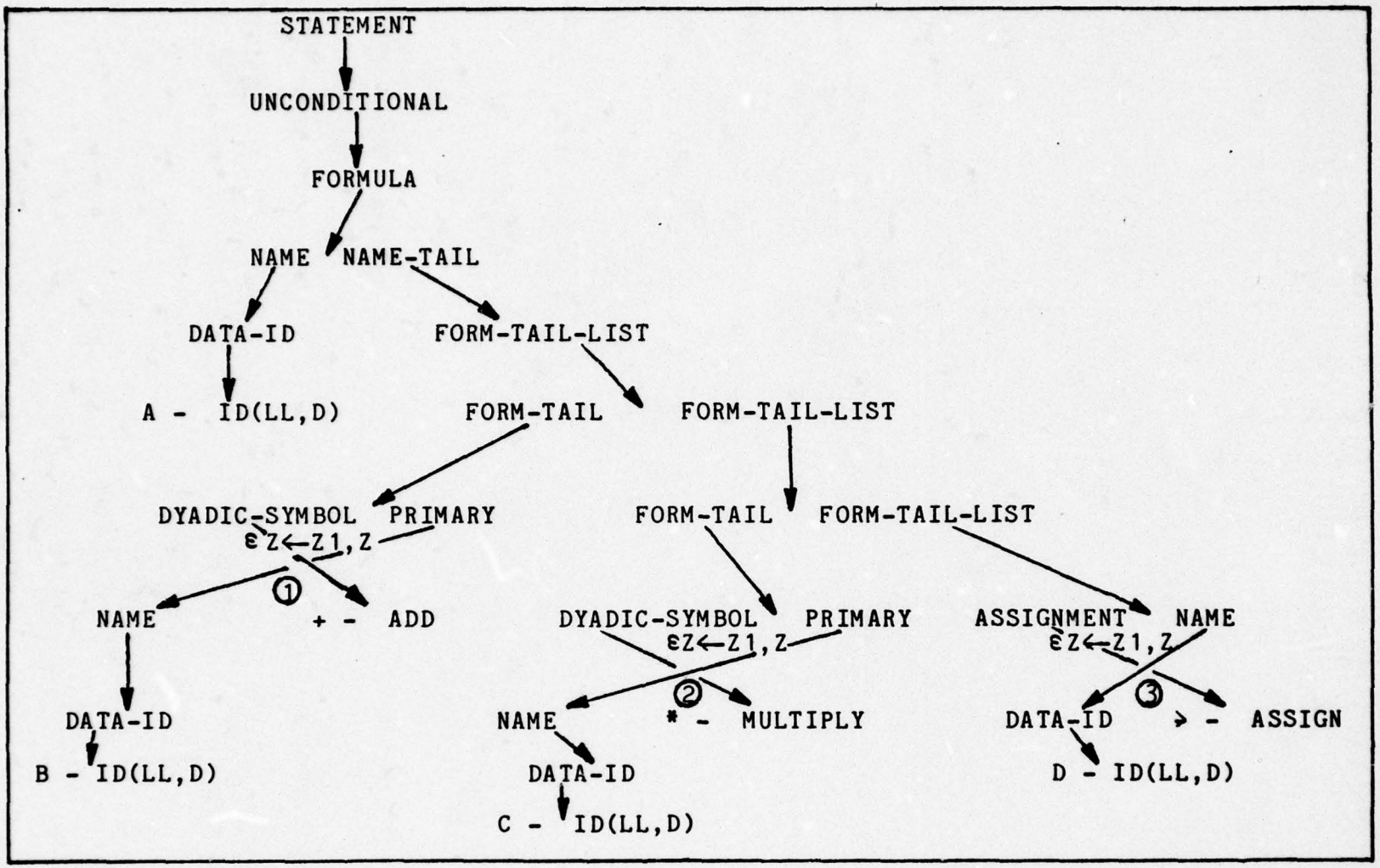
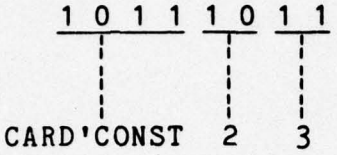


Figure 3. Parse of A+B*C→D to AB+C*D→

The compactor scans the Program String from the translator and encodes the biased integer vector into variable length code words. For example, if the code word of the CARD-CONST S-UNIT were 1011 then the string above would be compacted into the following string.



The code words are determined by gathering statistics on frequencies of S-UNIT utilization and by using Huffman encoding techniques on these frequencies. Thus the output of the Translator/Compactor is an encoded Program String which is stored in the SPLM memory. When the simulator is completed, it will also use the encoded Program String as input instead of the Program String in its present format (see Appendix A for current format of the Program String).

II.8 Summary

This chapter has presented an overview of the language, syntax, and translator. This overview should provide the necessary background for the following chapters. More information can be obtained on these topics presented in this chapter in Reference [2] and Reference [4].

Chapter III

SPLM Hardware

III.1 Introduction

The SPLM Architecture Study produced an architecture for a class of machines to execute directly SPLML. The major structures of the architecture for SPLM are the registers, the pushdown, the main memory, the barrel shifter, and the I/O section. This basic configuration (see Figure 4) reflects the deferred binding philosophy since these structures are the minimal hardware needed to provide the necessary operations of the machine. The appropriate hardware design for a specific application can be determined after examining statistics from the simulator. These statistics aid in determining the exact register configuration and any additional hardware that must be added to increase the speed of the design. The remainder of this chapter will describe the machine structures, their features, and the way in which they interact.

III.2 Registers

Appendix C contains a list of all possible registers that might be implemented and explanations of their use. A desirable subset of this list can be determined by examining

statistics from the simulator on register use. These registers are of varying length and are provided with increment and decrement capabilities.

There are no restrictions on register transfers, thus any register may be assigned to any other register. Assignment of differing length registers is performed with zero fill or truncation (zero fill if the assigned register is shorter and truncation if it is longer).

III.3 Pushdown

Arithmetic operations are performed exclusively within the pushdown which consists of the A and B registers (PDS'A and PDS'B) and a Last In First Out stack. The arithmetic and logic operations are performed between the A and B registers, with the stack being used to hold intermediate results.

There are two basic stack operators, POP and PUSH. The POP command moves the top word of the stack into the B register, sets the top of the stack pointer up one word in the stack, and clears the AF flag indicating the A register is empty. The PUSH command moves the contents of the B register onto the stack "pushing" the other words on the stack deeper in the stack. Then the contents of the A register are placed into the B register, the AF flag is cleared, and the stack pointer is moved down one word.

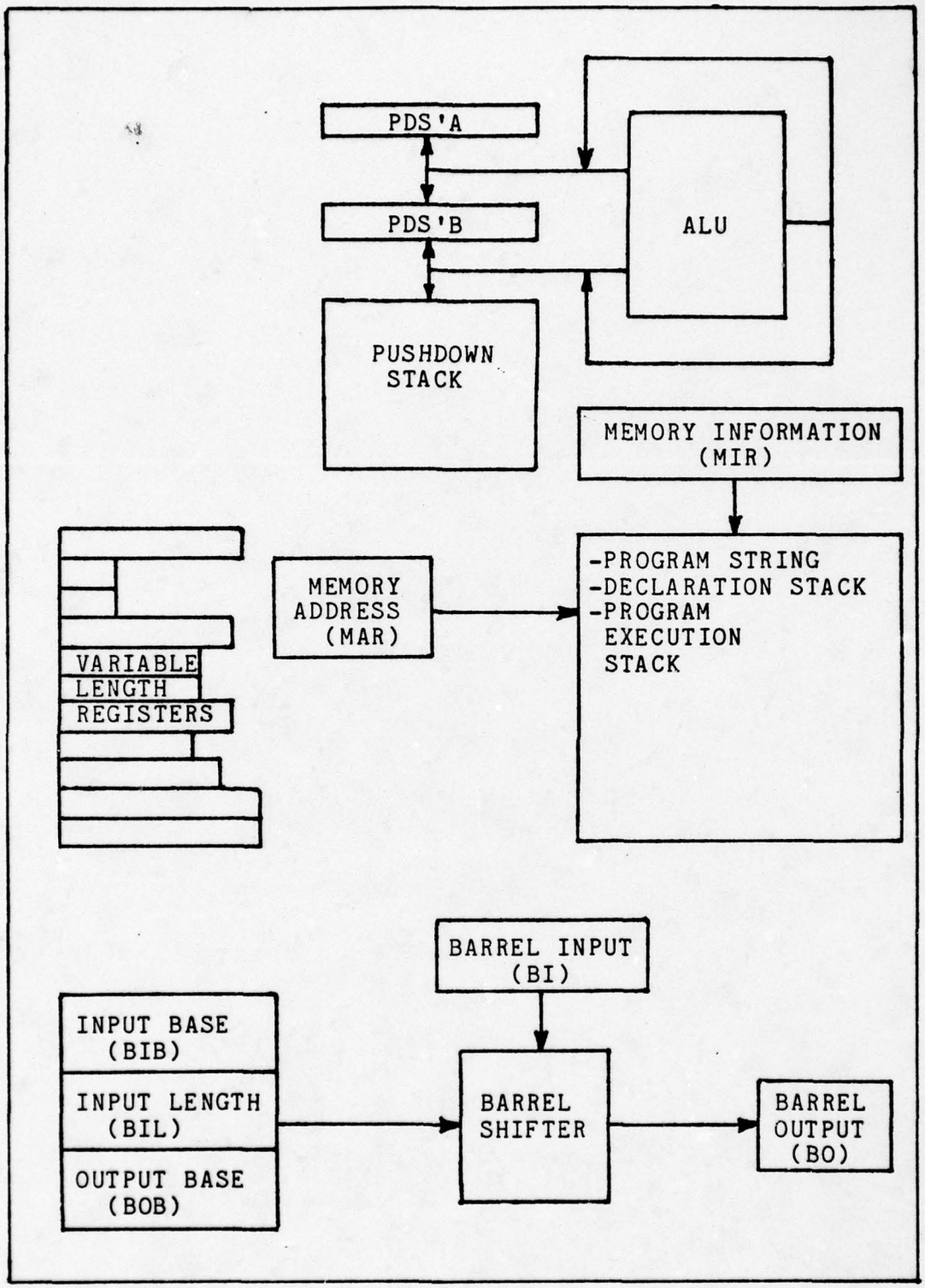


Figure 4. Basic Hardware Configuration

The arithmetic necessary for the statement "3+A>B[1,2];" (statement from simple program in Figure 2) is illustrated in Figure 5. Assume the initial stack conditions are as given with PDS'A containing a valid entry (AF=1). In order for the ADD to be accomplished, the operands must be pushed onto the stack (steps 2 and 3). Since the AF flag is set, the PUSH operation transfers the contents of PDS'A to PDS'B and transfers the contents of PDS'B to the pushdown. If AF had been clear (indicating no valid information in PDS'A), then the PUSH operation would transfer the contents of the register to PDS'A and set the AF flag; PDS'B and the pushdown would not be changed. Step 4 shows that the result of the ADD is left in PDS'B (all results of arithmetic and logic operations are contained in PDS'B) and AF is cleared. Steps 5 and 6 illustrate the operations involved in transferring a valid result from the pushdown to a register. As a result of the ADD, the AF flag is cleared; thus in step 5, the pushdown is popped and AF is set. In step 6, the contents of PDS'A are transferred to the appropriate register and AF is cleared. The contents of this register will then be stored in the B array (B[1,2]) in memory.

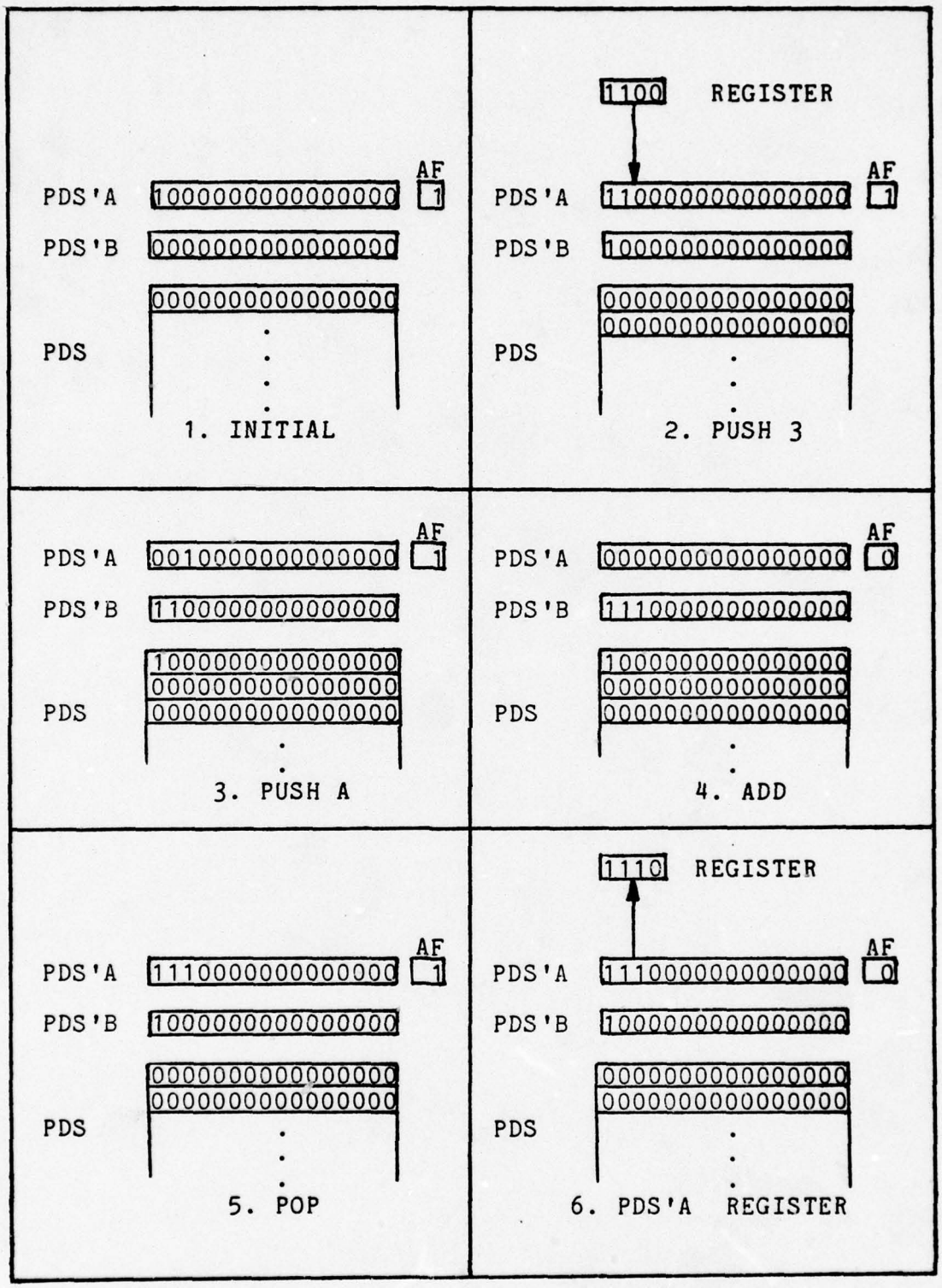


Figure 5. Pushdown Addition

III.4 Main Memory

SPLM has a random access and word addressable memory with a memory information register (MIR) and a memory address register (MAR). The memory provides storage for the Program String and a workspace during the execution of S-UNITS. The Program String occupies memory starting at word 0, and the workspace immediately follows the Program String.

The workspace is divided into two parts: the Declaration Stack and the Program Execution Stack. The Declaration Stack consists of items generated from the associated global declarations in the program. After the global declarations have been declared, the END-DECL (end of declarations) S-UNIT is executed and puts a mark in memory separating the Declaration Stack and the Program Execution Stack. The Program Execution Stack is a temporary work area where results are generated to be used later during execution or to be stored in the appropriate sections of the Declaration Stack. This temporary work area is destroyed after the execution of every statement thus saving memory by enabling the next statement to be executed using the same work area.

Items in the workspace can be divided into six categories: mark constants, type constants, syll constants, parameters, valuespaces, and links. Mark constants, syll constants, and type constants are also classifications of descriptor units (D-UNITS). Operands in the workspace consist of descriptors, their parameters, and a valuespace.

III.4.1 Mark Constants

Mark constants are used to signal the beginning of logical groupings of items on the stack. A mark is put on the stack to indicate the beginning of a new lexic level, the beginning of a subscript list, the beginning of a procedure, and the beginning of a soft extension. The first bit of a mark constant is a 1, and a mark constant follows a link.

III.4.2 Type Constants

Type constants correspond to the data types of the SPLM language and are used to specify the type of a declaration and id. When a type declaration is executed, a type constant is entered into the workspace. Execution of the declaration "INTEGER A BITS 5;" puts the INTEGER type constant on the Declaration Stack. The first bit of a type constant is 0, and type constants follow a link.

This binding of the first bit of mark constants and type constants facilitates the searching scheme. For example, when searching for the BEGIN mark constant (indicating the beginning of a new lexic level), only the first bit after the link need be scanned to determine if the constant is a mark constant. If the bit is 0, the constant is a type constant and the search continues at the next link. If the bit is 1, then a mark constant has been found, and then a comparison is made to see if the constant is the BEGIN mark constant. This binding saves searching time

since needless comparisons (i.e. comparing a type constant to see if it is the BEGIN mark constant) are avoided.

III.4.3 Syll Constants

Syll constants specify various attributes of a particular constant or variable and follow type constants in the workspace. Each attribute is information used in processing the constant or variable. For example, in executing a subscripted variable's declaration or id, a step syll constant is entered in the workspace to signal that dimension information follows. Likewise, the length syll constant indicates that the following information pertains to the bit length of the variable or constant.

III.4.4 Descriptor Parameters

Descriptor parameters are used to give the specific information of an attribute and parameters follow syll constants and mark constants. For example, when a new lexic level is entered, the BEGIN mark constant is placed in the workspace followed by two parameters: the contents of the lexic level register (LL) and dimension register (DIMREG). Similarly the LENGTH syll constant is followed by the contents of the length register (LAMBDA) and the STEP syll constant is followed by the contents of DIMREG.

III.4.5 Valuespace

A valuespace is the space in memory allocated for a variable and follows the VALUE syll constant. The number of

bits allocated for a valuespace is determined by multiplying the number of elements in the data item by the total length in bits of each element. The variable A in Figure 2 is a scalar, thus A is allocated 6 bits (1 element*total length, total length=5 bits long + 1 sign bit). The variable B is non scalar and is allocated 30 bits (6 elements*total length, total length=4 bits long + 1 sign bit).

III.4.6 Links

Moving up and down the stack is accomplished using links which are placed between operands. This gives the stack the appearance of a linked list where each link points to the next operand in both directions (up and down the stack). When the first operand is placed on the stack, a link is placed before the operand. This link is used to find the first bit following the operand, the location of the next link and its associated operand.

For example, during execution of the ID S-UNIT, a BEGIN mark constant is searched for by moving down the stack with the links. When the proper BEGIN mark constant has been found, then these same links are used to move up the stack until the proper operand has been found. The ID S-UNIT and the operations with the links will be discussed in more detail in Chapter IV.

III.4.7 Declaration Example

Figure 6 depicts with mnemonics, the stack as a result of the execution of the following declarations:

```
INTEGER A BITS 5;
INTEGER B[2,3];
```

The memory is depicted in Figure 6 as a bit vector, for this is the way that SPLM views memory. As shown in Figure 6, the operand in the workspace consists of descriptors, their parameters, and a valuespace. The A operand consists of the INTEGER type constant, LENGTH syll constant, length parameter (5), VALUE syll constant, and a 6 bit valuespace. The B operand follows a link and consists of an INTEGER type constant, STEP syll constants (subscripting information), step parameters, VALUE syll constant, and a 30 bit valuespace. The B operand does not have a LENGTH syll constant since the bit length of B was not declared and thus defaults to the default length (i.e. 4 bits long).

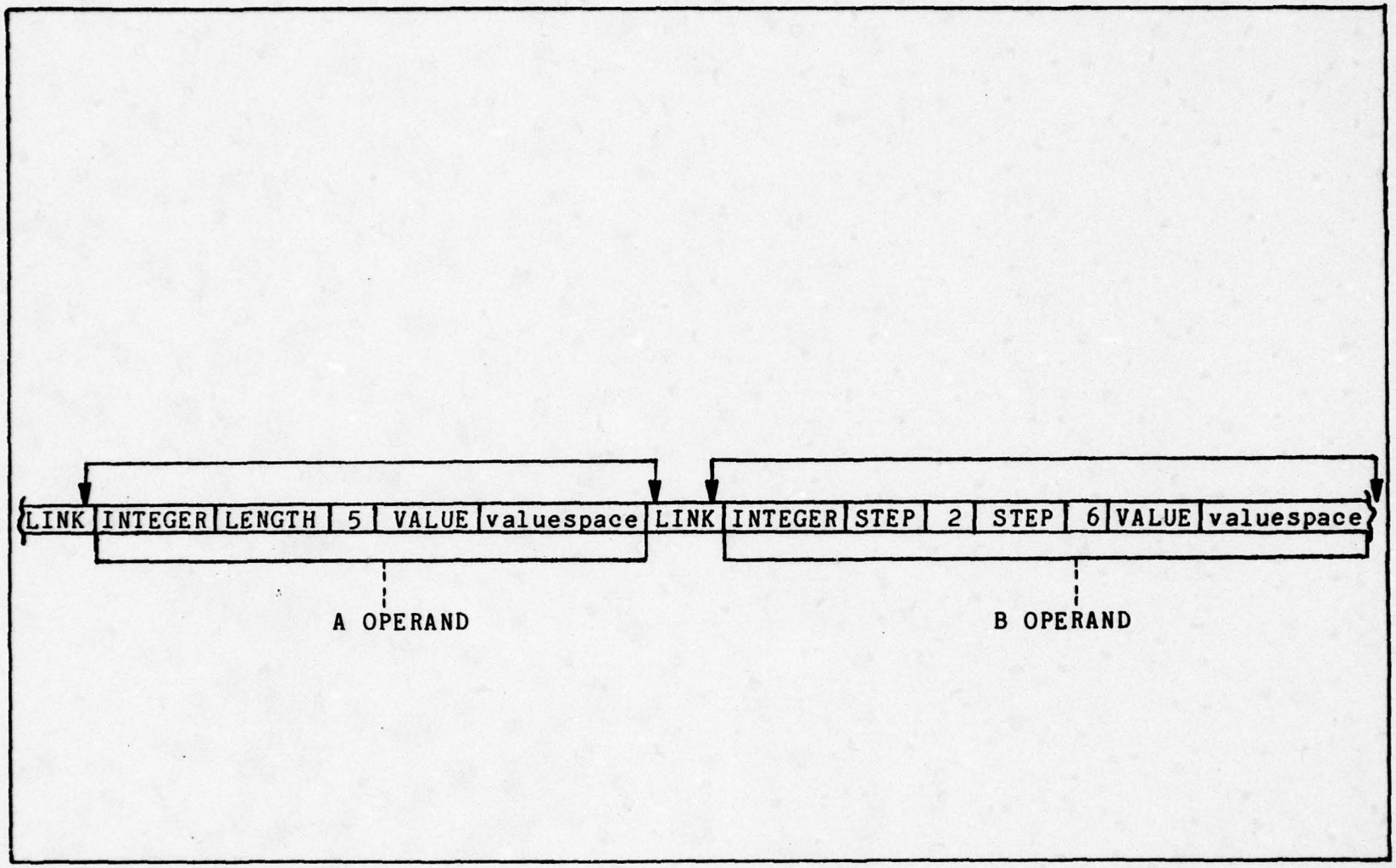


Figure 6. A and B Operands

III.5 Barrel Shifter

Because main memory is only word addressable, a barrel shifter is used to extract (known as LOAD'BARREL) or overwrite (known as WRITE'BARREL) a bit string within memory. Before a LOAD'BARREL or WRITE'BARREL operation is performed, the desired bitstring is specified by its bit address, placed in a register called MS, and its bit length, placed in a register called MSL. Other registers associated with the barrel shifter are:

1) Barrel Input (BI)- used to hold the bitstring that is to be shifted.

2) Barrel Output (BO)- contains field where the bitstring is shifted into from BI.

3) Barrel Input Base (BIB) points to the left hand bit of the bitstring in BI that is to be shifted.

4) Barrel Input Length (BIL)- contains the number of bits to be shifted.

5) Barrel Output Base (BOB)- points to the left hand bit of the BO field where the bitstring will be written into.

Registers BI and BO are both the same length, at least one memory word larger than all other registers, thus allowing enough space for the contents of any register to be loaded and shifted as necessary. An example of a LOAD'BARREL is given in Figure 7 and is discussed in the following text:

Initial Conditions:
a) MS points to the

first (leftmost) bit of the bitstring within memory.

b) MSL contains the length of the bitstring.

1) Fetch the first word from memory and place it into BI.

2) Shift the first bits from BI into BO.

a) BIB is first offset to the left most bit of the bitstring (bit 26).

b) BIL is then set to the length (3 bits) of the bitstring section contained within the first memory word.

c) BOB is set to the left most bit of BO (bit 31).

d) A BARREL'SHIFT is performed. BARREL'SHIFT shifts BIL bits starting at BIB from BI into BO starting at BOB.

3) Fetch the middle word from memory and place it into BI.

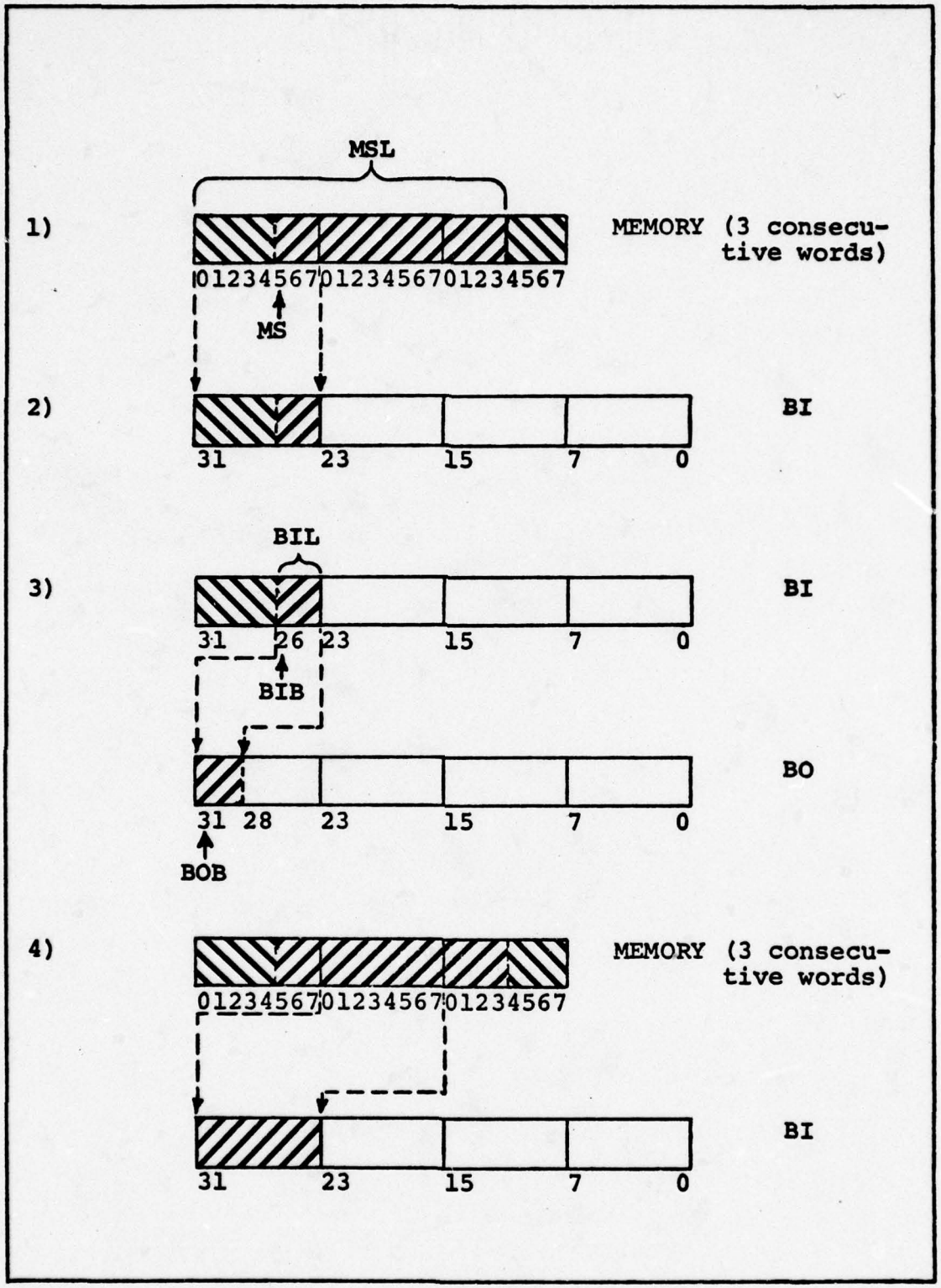


Figure 7. An example of LOAD'BARREL

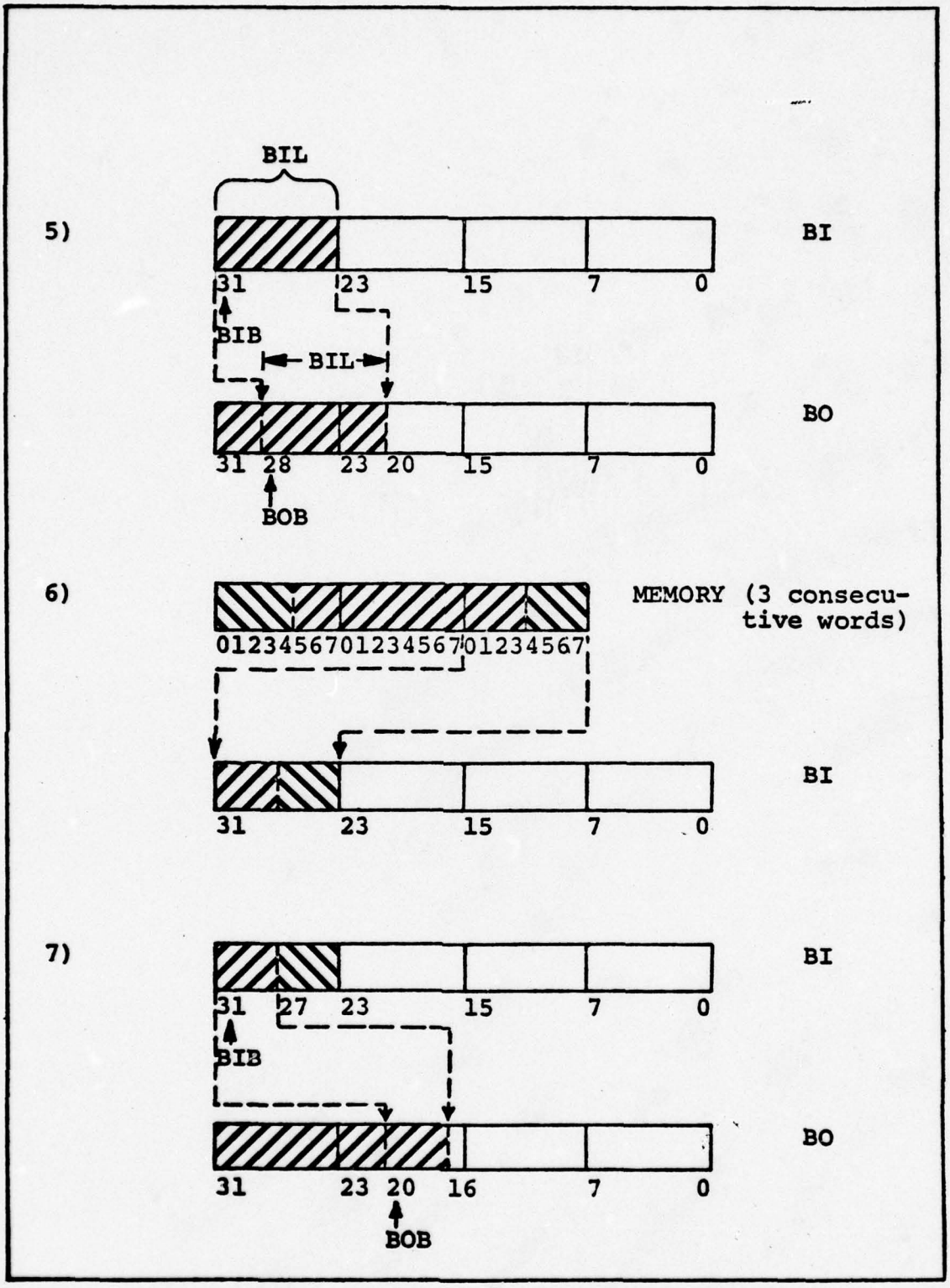


Fig 7. (Cont.) An example of LOAD'BARREL

4) Shift the middle word from BI into BO.

a) BOB is offset to the beginning of where the middle word is to be placed (bit 28).

b) BIB is set to the left most bit (bit 31).

c) BIL is set to the memory word length (8 bits).

d) A BARREL'SHIFT is performed.

5) Fetch the last word out of memory and place it into BI.

6) Shift the final bits from the last word from BI into BO.

a) BOB is offset to the beginning of where the last bits are to be placed (bit 20).

b) BIB is set to the left most bit (bit 31).

c) BIL is set to the length of the number of bits left (4 bits).

d) A BARREL'SHIFT is performed.

Before a WRITE'BARREL can be performed the barrel shifter must first be preset. Presetting is accomplished by the function PRESET'BARREL as shown in Figure 8 and discussed in the following text:

1) Initial Conditions:

a) MS points to the first bit of the space within memory where the bitstring is to be placed.

b) MSL contains the length of the bitstring.

2) Fetch the first word from memory and place it into BO.

3) Fetch the last word from memory and place it into BI.

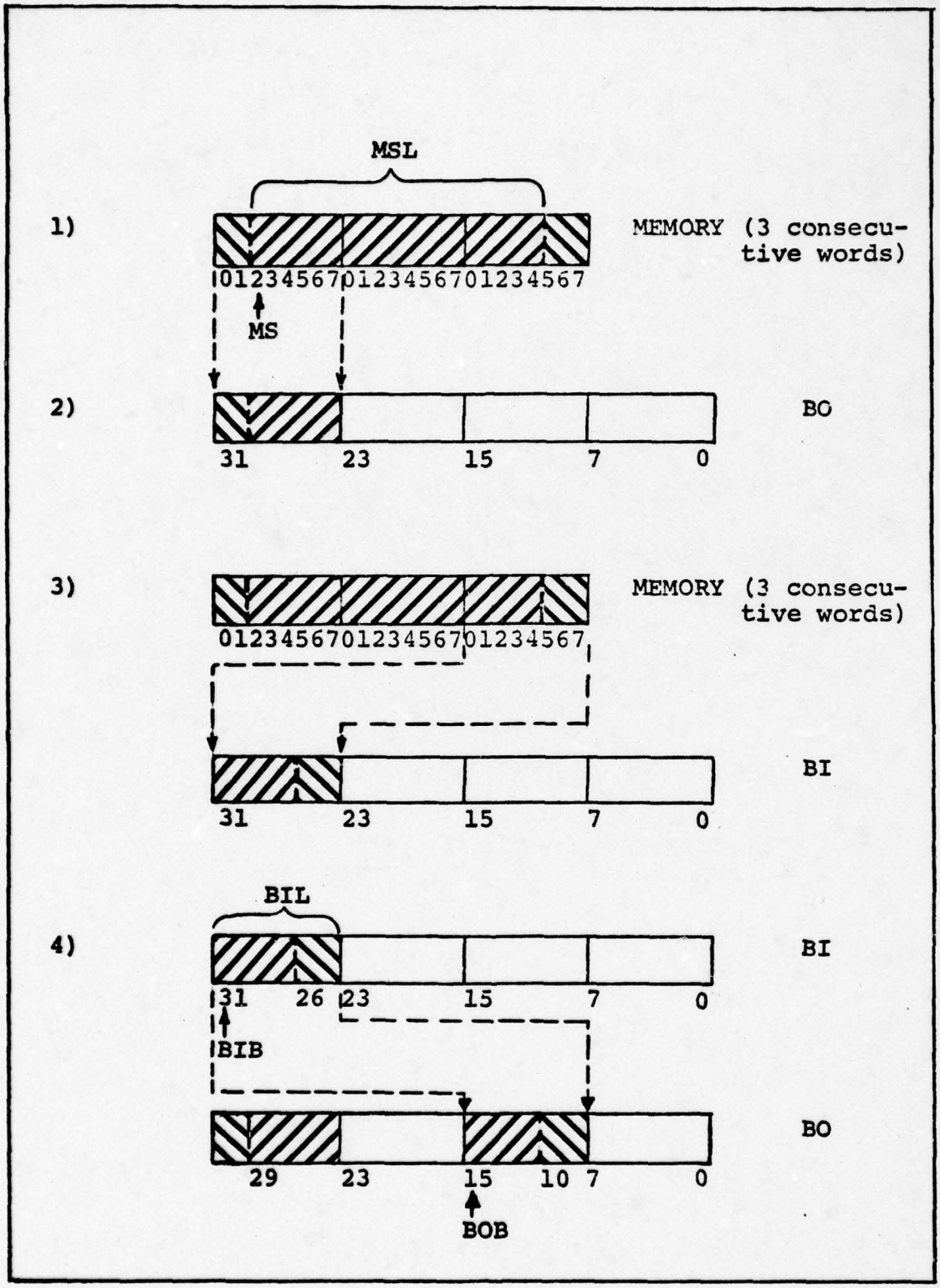


Figure 8. An example of PRESET'BARREL

4) Shift last word from BI into BO.
    a) First set BIB to the left most bit of BI (bit 31).
    b) Set BIL equal to the memory word length (8 bits).
    c) BOB is offset to the beginning of where the last word is to be placed in BO (bit 15).
    d) Perform a BARREL'SHIFT.

BO now is preset to allow the bitstring to be written into it. However, before this can be done, BI is first loaded with the bitstring. A WRITE'BARREL can now be performed as shown in Figure 9 and discussed in the following text:

1) Shift the contents of BI into BO.
    a) To accomplish this, BIL is set to MSL (19 bits).
    b) BIB is set to the left most bit (bit 31).
    c) BOB is offset to the the left most bit (bit 29) where the bitstring is to be placed in BO.
    d) And a BARREL'SHIFT is performed.

2) Move contents of BO into BI.

3) Write the first word from BO into memory.

4) Shift middle word from BI into BO.
    a) BIB is set to the beginning of the middle word within BI (bit 23).
    b) BOB is set to the left most bit (bit 31).
    c) BIL is set equal to the memory word length (8 bits).
    d) Perform a BARREL'SHIFT.

5) Write the middle word from BO into memory.

6) Shift last word from BI into BO.
    a) BIB is set to the beginning of the last word within BI.
    b) BOB is set to the left most bit (bit 31).
    c) BIL is set equal to the memory word length (8 bits).
    d) Perform a BARREL'SHIFT.

7) Write the last word from BO into memory.

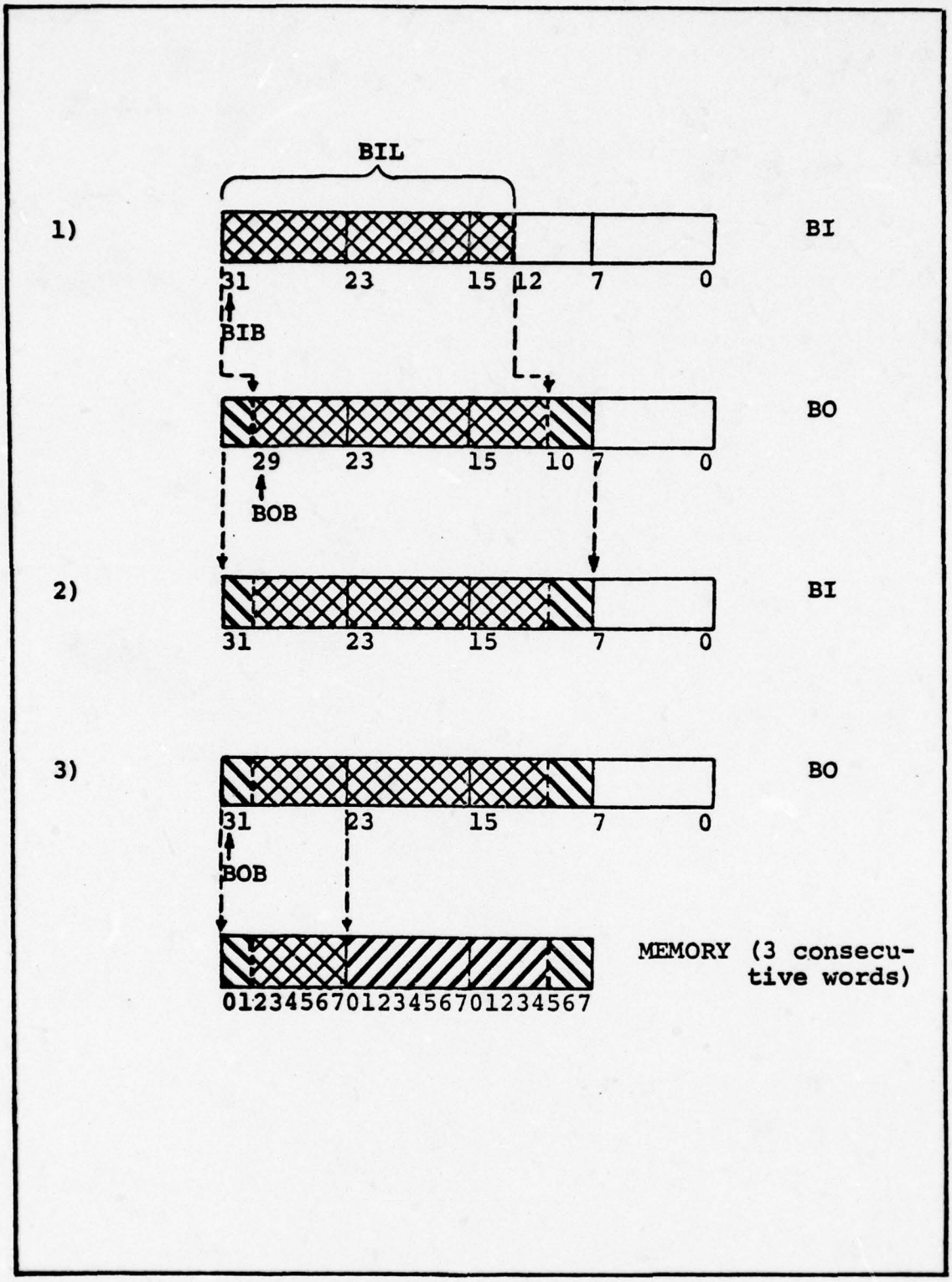


Figure 9. An example of WRITE'BARREL

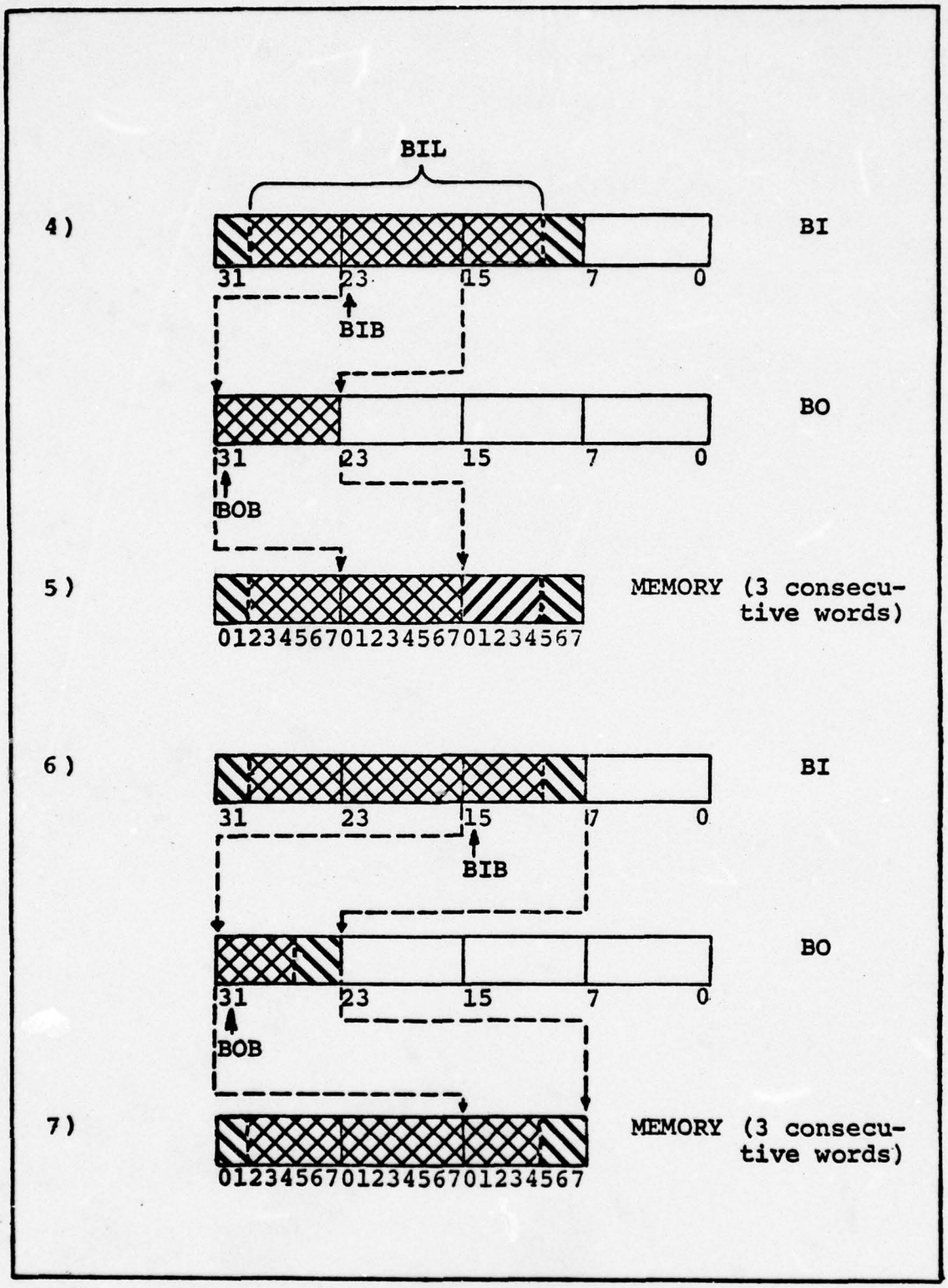


Fig 9. (Cont) An example of WRITE'BARREL

III.6 Input/Output Section

The input/output (I/O) section consists of ports which interface with peripheral devices and I/O variables associated with memory buffers. In the SPLM language, I/O transfers are set up as I/O variables are referenced. Each I/O variable has a specific block of memory associated with it to buffer data into or out of a particular port. Ports are either input or output ports and have data lines, a request line, an acknowledge line, and a forced completion line associated with them.

There is a data line associated with each bit of the buffer in the device, thus the width of the frame (number of bits in the frame) corresponds to the number of bits in the device buffer. An I/O operation consists of transferring frames to/from the memory buffer until the memory buffer is full/empty (a more detailed explanation of I/O operations is contained in Chapter II Section 4.5).

The request and acknowledge lines are used in the handshaking between the I/O section of the machine and the device. The serial transfer of frames is accomplished when the sender (SPLM I/O section for output and device for input) raises the request line notifying the receiver that data is on the lines. When the data has been accepted, the receiver raises the acknowledge line, signaling that more data may be sent.

The forced completion line is used by the sending

device to signal the end of data transfer before the entire stream element has been input. This line is raised to prevent the program execution from halting while waiting for completion of an I/O data transfer. This line is provided to enable the device to notify the SPLM I/O Section of abnormal conditions which prevent the device from completing the I/O operation.

External interrupts are processed by interrupt handlers written as procedures in the SPLML program. When an interrupt line is raised, the procedure corresponding to that interrupt is called, and all other interrupts are masked until an enable operator is executed.

III.7 Summary

This chapter has presented the basic hardware structures of SPLM. This basic architecture represents a class of machines from which a special purpose SPLM can be designed when a specific application is determined. The next chapter presents the hardware operation of these structures during execution.

Chapter IV

The Hardware Operation during the Execution Phase

Since the key to understanding SPLM lies in grasping how a program is executed, this chapter is devoted to explaining the hardware operation during the execution phase. In order to aid the reader, the execution of an example Program String is followed in detail.

The Program String, listed in Dump 1, is the translator output of the example program of Chapter II, shown in Figure 10.

```
PROGRAM
   DEFAULT 4;
   INTEGER A BITS 5;
   INTEGER B[2,3];
   3→A;
   A+4→B[1,2]
TERM
```

Figure 10. The Simple SPLML Program of Chapter II

Dump 1 contains the following columns:

1) The right most column gives the mnemonic description of the Program String S-UNITS and the decimal value of the Program String parameters.

2) Left of this column is the binary representation of the Program String as it appears in memory (the least significant bit is on the left).

3) The next column gives the decimal address of each word in memory.

4) And the far left column gives the octal bit address of the least significant bit of each word.

Dump 1. The Bit Representation of the Program String

```
00000    0 0011101000000000 PROG
00020    1 1111011000000000 DUMP              2
00040    2 0011011000000000 SET'DEFAULT
00060    3 0010000000000000                   4
00100    4 0001001000000000 NEW'INTEGER
00120    5 0100011000000000 CARD'CONST
00140    6 1100000000000000                   3
00160    7 1010000000000000                   5
00200    8 1111011000000000 DUMP              3
00220    9 0010011000000000 DYNA'LEN
00240   10 0001001000000000 NEW'INTEGER
00260   11 1111011000000000 DUMP              4
00300   12 0100011000000000 CARD'CONST
00320   13 0100000000000000                   2
00340   14 0100000000000000                   2
00360   15 1111011000000000 DUMP              5
00400   16 0000100000000000 DYNA'DIM
00420   17 1111011000000000 DUMP              6
00440   18 0100011000000000 CARD'CONST
00460   19 0100000000000000                   2
00500   20 1100000000000000                   3
00520   21 0000100000000000 DYNA'DIM
00540   22 1111011000000000 DUMP              7
00560   23 0110100000000000 END'DECL
00600   24 1111011000000000 DUMP              8
00620   25 0100011000000000 CARD'CONST
00640   26 0100000000000000                   2
00660   27 1100000000000000                   3
00700   28 1111011000000000 DUMP              9
```

```
00720    29 1110011000000000 ID
00740    30 0000000000000000                    0
00760    31 0000000000000000                    0
01000    32 1111011000000000 DUMP              10
01020    33 0111101000000000 ASSIGN
01040    34 1111011000000000 DUMP              11
01060    35 0110000000000000 CLEAN'UP
01100    36 1111011000000000 DUMP              12
01120    37 1110011000000000 ID
01140    38 0000000000000000                    0
01160    39 0000000000000000                    0
01200    40 0100011000000000 CARD'CONST
01220    41 1100000000000000                    3
01240    42 0010000000000000                    4
01260    43 1111011000000000 DUMP              13
01300    44 0000011000000000 S''ADD
01320    45 1111011000000000 DUMP              14
01340    46 0110101000000000 START'LIST
01360    47 0100011000000000 CARD'CONST
01400    48 1000000000000000                    1
01420    49 1000000000000000                    1
01440    50 1011101000000000 CANONICAL
01460    51 0100011000000000 CARD'CONST
01500    52 0100000000000000                    2
01520    53 0100000000000000                    2
01540    54 1110101000000000 SUBSCRIPT'MODE
01560    55 1111011000000000 DUMP              15
01600    56 1110011000000000 ID
01620    57 0000000000000000                    0
01640    58 1000000000000000                    1
01660    59 1111011000000000 DUMP              16
01700    60 0111101000000000 ASSIGN
01720    61 1111011000000000 DUMP              17
01740    62 0001101000000000 TERMINATE
```

One of the S-UNITS, DUMP, gives an expanded snapshot of the workspace after the execution of any S-UNIT without effecting the contents of the workspace. Since the workspace is in packed form, DUMP provides a readable memory map for following the execution of the Program String. A sample memory dump is shown in Dump 2 after the execution of PROG, always the first S-UNIT in the Program String.

Viewing the figure from right to left, the fields are:

1) The mnemonic label of each item (e.g. link, the mark constant BEGIN, or the parameter Displacement).

2) The expanded binary representation of the workspace (each item is seperated by a horizontal lines).

3) The decimal word address.

4) The least significant bit address of each individual item.

Dump 2. The Declaration Stack as it appears after PROG

```
01760   63 00000000000000   link
01776   63               00 link
        64 111111111000
02014   64             0000 link
        65 0000000000
02032   65            10000 BEGIN       (mark constant)
02037   65                0 Lexic Level (parameter)
        66 000
02043   66   000000         Displacement (parameter)
```

IV.1 The Example Execution

The previous chapter described the individual parts of the hardware in detail whereas this chapter is geared toward giving the reader an overview of the hardware operation during the execution phase. The rest of this chapter is devoted to viewing the execution of the example program line by line.

IV.1.1 PROGRAM

PROGRAM is translated directly into the PROG S-UNIT whose dump has already been shown in Dump 2. PROG's function is to mark the Declaration Stack with the mark constant BEGIN and the two parameters of BEGIN, Lexic Level and Displacement. The fact that Lexic Level and Displacement are both equal to zero, uniquely identifies this as the beginning of the Declaration Stack.

Links on the Declaration Stack provide a means of searching the stack either forwards (known as a stepping up) or backwards (known as a stepping down). The three links of Dump 2 are the starting point for building the link structure and are initially equal to zero. Link one remains zero and only its address is used in step ups and step downs. The value assigned to the second link is the exclusive-or of the bit address of the first link and the bit address of the third link (1760 ⊕ 2014 = 3774, remember the least significant bit is on the left). Link three remains equal to a zero until later when a fourth link is placed on the stack at which time it is assigned the exclusive-or of the second and fourth links' bit addresses.

A STEP'UP is performed by the exclusive-or of a link bit address (e.g. 1760, the bit address of link one) and the value of the next link up (e.g. 3774, the value of link two), resulting in the address of the second link up (e.g. 2014, the bit address of link three). Likewise, STEP'DOWN exclusive-ors the link bit address (e.g. 2014, the bit

address of link three) and the value of the next link down (e.g. 3774, the value of link two) to get the address of the second link down (e.g. 1760, the address of link one).

IV.1.2 DEFAULT 4;

DEFAULT 4; translates directly into SET'DEFAULT and a Program String parameter with the value of 4. The only function of SET'DEFAULT is to input the default bit length from the Program String and place it into a register called DEFAULT. The default value is used whenever a declaration is referenced and the declaration does not contain a bit length specification.

IV.1.3 INTEGER A BITS 5;

Since this is a declaration requiring storage, a valuespace is allocated. However, before this can by done, preliminary information is placed on the Declaration Stack to describe the attributes of the valuespace. In this case, the attributes are the type integer and the bit length 5. The type attribute results from executing NEW'INTEGER and appears on the stack as the type constant INTEGER. The bit length attribute results from two S-UNITS, CARD'CONST and DYNA'LEN.

Whenever a cardinal constant is encountered in the program (e.g. the constant 5 specifies the bit length of the integer A), the CARD'CONST S-UNIT is generated. During the execution of this S-UNIT, the attributes of the cardinal constant are placed on the stack. In the example, the

attributes are the type constant CARDINAL, the bit length equal to 3 (i.e. this is the number of bits required by the binary representation of the bit length parameter 5), the syll constant CONSTANT, and finally the address pointing to the bit length parameter 5.

Dump 3. Declaration Stack as it appears after the CARD'CONST 5

01760	63 00000000000000	link	
01776	63 00	link	
	64 111111111000		
02014	64 1110	link	
	65 1011111000		
02032	65 10000	BEGIN	(mark constant)
02037	65 0	Lexic Level	(parameter)
	66 000		
02043	66 000000	Displacement	(parameter)
02051	66 1110110	link	
	67 0000000		
02067	67 0010	INTEGER	(type constant)
02073	67 00000	link	
	68 000000000		
02111	68 0100	CARDINAL	(type constant)
02115	68 001	LEN	(syll constant)
	69 0		
02121	69 1100000	Bit Length	(parameter)
02130	69 0110	CONSTANT	(syll constant)
02134	69 1110	ADDR	(syll constant)
02140	70 00001110000000	Address	(parameter)

As seen in the Dump 3, the bit address pointing to the value 5 is 160 (octal) and is written into the CARDINAL declaration-operand at bit location 2140 (octal). (Note: 'Operand' is used to refer to any area in the workspace between two consecutive links and begins with a type constant. A declaration-operand is an operand within the Declaration Stack and, likewise, a statement-operand is an operand within the Program Execution Stack).

The workspace presently contains two declaration-operands, INTEGER at bit location 2067 and CARDINAL at bit location 2111, each of which describe only a part of the declaration INTEGER A BITS 5;. However, the desired result is to have each declaration-operand describe all of the attributes of one declaration. This is the function of the DYNA S-UNITS such as DYNA'LEN, the next S-UNIT in the Program String. DYNA'LEN uses the address in the CARDINAL declaration-operand to fetch the bit length of A from the cardinal constant declaration-operand. Next DYNA'LEN deletes the last link and the cardinal constant declaration-operand and adds the bit length attribute (i.e. LEN and the bit length parameter of 5) onto the INTEGER declaration-operand (refer to Dump 4, bits 2073 - 2105 octal).

The reason for creating the CARDINAL operand and not writing the LEN and bit length parameter directly is that the creation of the bit length attribute is sometimes more complicated than it was in this case. The syntax of SPLM allows the bit length of a declaration to be specified by a formula, for example INTEGER A BITS B+5; (note: the variable B would have to already have been declared for this declaration to be legal). The formula B+5 is evaluated first, creating a declaration-operand at the end of the stack. DYNA-LEN next scans this declaration-operand determining the bit length, deletes the declaration-operand, and adds the bit length attribute (LEN and the parameter bit length) to the INTEGER declaration-operand.

The allocation of the valuespace is next but actually does not take place until during the execution of the next S-UNIT as shown in Dump 4. (Note that the next S-UNIT is NEW'INTEGER which is the reason for type constant INTEGER at the end of Declaration Stack). The valuespace is marked with the syll constant VALUE and is allocated six bits, five for the value and one for the sign bit (left most bit).

Dump 4. Declaration Stack as it Appears after NEW'INTEGER

```
01760   63 000000000000000   link
01776   63              00   link
        64 111111111000
02014   64            1110   link
        65 1011111000
02032   65           10000   BEGIN          (mark constant)
02037   65               0   Lexic Level    (parameter)
        66 000
02043   66          000000   Displacement   (parameter)
02051   66         0011101   link
        67 0000000
02067   67            0010   INTEGER        (type constant)
02073   67            0010   LEN            (syll constant)
02077   67               1   Bit Length     (parameter)
        68 010000
02106   68            1010   VALUE          (syll constant)
02112   68          000000   Valuespace
02120   69 01100000011011   link
02136   69              00   INTEGER        (type constant)
        70 10
```

IV.1.4 **INTEGER B[2,3];**

Since the integer B is nonscalar, dimensions are used to describe the size of the array. Each dimension translates into the Program String as a CARD'CONST and

DYNA'DIM pair which when executed, places a step onto the Descriptor Stack. CARD'CONST sets up a CARDINAL declaration-operand as seen in Dump 5 which is used by DYNA'DIM to create the first step marked by the syll constant STEP as viewed in Dump 6.

Dump 5. Declaration Stack Showing the CARD'CONST 3 before DYNA'DIM

01760	63 00000000000000	link	
01776	63 00	link	
	64 111111111000		
02014	64 1110	link	
	65 1011111000		
02032	65 10000	BEGIN	(mark constant)
02037	65 0	Lexic Level	(parameter)
	66 000		
02043	66 000000	Displacement	(parameter)
02051	66 0011101	link	
	67 0000000		
02067	67 0010	INTEGER	(type constant)
02073	67 0010	LEN	(syll constant)
02077	67 1	Bit Length	(parameter)
	68 010000		
02106	68 1010	VALUE	(syll constant)
02112	68 000000	Valuespace	
02120	69 110100100000000	link	
02136	69 00	INTEGER	(type constant)
	70 10		
02142	70 00111000000000	link	
02160	71 0100	CARDINAL	(type constant)
02164	71 0010	LEN	(syll constant)
02170	71 0100000	Bit Length	(parameter)
02177	71 0	CONSTANT	(syll constant)
	72 110		
02203	72 1110	ADDR	(syll constant)
02207	72 000001110	Address	(parameter)
	73 00000		

Dump 6. Declaration Stack Showing the Placement of the First Step

```
01760   63 00000000000000   link
01776   63           00     link
        64 111111111000
02014   64         1110     link
        65 1011111000
02032   65        10000     BEGIN          (mark constant)
02037   65            0     Lexic Level    (parameter)
        66 000
02043   66   000000         Displacement   (parameter)
02051   66      0011101     link
        67 0000000
02067   67 0010             INTEGER        (type constant)
02073   67      0010        LEN            (syll constant)
02077   67         1        Bit Length     (parameter)
        68 010000
02106   68  1010            VALUE          (syll constant)
02112   68      000000      Valuespace
02120   69 11010010000000   link
02136   69           00     INTEGER        (type constant)
        70 10
02142   70  1100            STEP           (syll constant)
02146   70    010000        Multiplier     (parameter)
```

The first step is always equal to the first dimension while each successive step is the product of the last step and the present dimension. Therefore in this example,the first step is 2 and the next step is 2 x 3 or 6 as shown in Dump 7.

Dump 7. Declaration Stack Showing the Placement of Steps

```
01760   63 000000000000000   link
01776   63                00 link
        64 111111111000
02014   64              1110 link
        65 1011111000
02032   65           10000   BEGIN          (mark constant)
02037   65               0   Lexic Level    (parameter)
        66 000
02043   66    000000         Displacement   (parameter)
02051   66          0011101  link
        67 0000000
02067   67        0010       INTEGER        (type constant)
02073   67           0010    LEN            (syll constant)
02077   67               1   Bit Length     (parameter)
        68 010000
02106   68       1010        VALUE          (syll constant)
02112   68          000000   Valuespace
02120   69 101000100000000   link
02136   69               00  INTEGER        (type constant)
        70 10
02142   70   1100            STEP           (syll constant)
02146   70       010000      Multiplier     (parameter)
02154   70           1100    STEP           (syll constant)
02160   71 011000            Multiplier     (parameter)
```

The bit length of B is not specified and therefore is the default length. Consequently, whenever this declaration is later referenced, the default bit length will be used.

Again the allocation of the valuespace does not take place until during the execution of the next S-UNIT which in this case is END'DECL. The valuespace is marked by the syll constant VALUE and allocated 30 bits of storage (six four bit integers plus one sign bit each) as shown in Dump 8. After the valuespace, END'DECL marks the end of

the declaration stack and the beginning of the Program Execution Stack with the mark constant MARK.

Dump 8. Declaration Stack as it Appears after END'DECL

```
01760   63 000000000000000   link
01776   63                00 link
        64 111111111000
02014   64              1110 link
        65 1011111000
02032   65             10000 BEGIN          (mark constant)
02037   65                 0 Lexic Level    (parameter)
        66 000
02043   66    000000         Displacement   (parameter)
02051   66           0011101 link
        67 0000000
02067   67         0010       INTEGER      (type constant)
02073   67            0010    LEN          (syll constant)
02077   67                1   Bit Length   (parameter)
        68 010000
02106   68       1010         VALUE        (syll constant)
02112   68          000000    Valuespace
02120   69 10001101000000    link
02136   69                00 INTEGER       (type constant)
        70 10
02142   70   1100            STEP         (syll constant)
02146   70       010000      Multiplier   (parameter)
02154   70              1100 STEP         (syll constant)
02160   71 011000            Multiplier   (parameter)
02166   71       1010        VALUE        (syll constant)
02172   71          000000   Valuespace
        72 0000000000000000
        73 00000000
02230   73         01000000 link
        74 000000
02246   74       11000       MARK         (mark constant)
```

IV.1.5 3→A;

END'DECL also indicates the program contains no more global declarations and the program statement list has begun. In this example, 3→A; is the first statement and

appears in the Program String as a CARD'CONST and an ID followed by an ASSIGN. Executing the S-UNIT CARD'CONST results in a similar cardinal constant operand as was seen before in Dump 3 and Dump 5. The Program Execution Stack now contains the CARDINAL constant statement-operand having a bit length of 2 and an address pointing to the value of 3 (i.e. the bit address 660 octal) as shown in Dump 9.

ID is a reference to a specific declaration: in particular, the integer declaration A. The purpose of ID is to set up a second statement-operand containing an address of integer A's valuespace. To accomplish this task, ID first steps down starting at the last links' bit address 2253 (refer to Dump 9) until a BEGIN D-UNIT is found at bit address 2032 octal. The stepping down process is shown as follows (in octal):

```
  2253   link address (Check for BEGIN at address 2271)
@ 0373   value of link at address 2230
= 2120   link address

  2230   link address (Check for BEGIN at address 2246)
@ 0261   value of link at address 2120
= 2051   link address

  2120   link address (Check for BEGIN at address 2136)
@ 0134   value of link at address 2051
= 2014   link address

  2051   link address (Check for BEGIN at address 2067)
@ 3727   value of link at address 2014
= 1776   link address

  2014   link address (BEGIN found at address 2032)
```

Dump 9. Declaration Stack as it Appears after CARD'CONST of 3

01760	63	00000000000000	link	
01776	63	00	link	
	64	111111111000		
02014	64	1110	link	
	65	1011111000		
02032	65	10000	BEGIN	(mark constant)
02037	65	0	Lexic Level	(parameter)
	66	000		
02043	66	000000	Displacement	(parameter)
02051	66	0011101	link	
	67	0000000		
02067	67	0010	INTEGER	(type constant)
02073	67	0010	LEN	(syll constant)
02077	67	1	Bit Length	(parameter)
	68	010000		
02106	68	1010	VALUE	(syll constant)
02112	68	000000	Valuespace	
02120	69	10001101000000	link	
02136	69	00	INTEGER	(type constant)
	70	10		
02142	70	1100	STEP	(syll constant)
02146	70	010000	Multiplier	(parameter)
02154	70	1100	STEP	(syll constant)
02160	71	011000	Multiplier	(parameter)
02166	71	1010	VALUE	(syll constant)
02172	71	000000	Valuespace	
	72	0000000000000000		
	73	00000000		
02230	73	11011111	link	
	74	000000		
02246	74	11000	MARK	(mark constant)
02253	74	00000	link	
	75	000000000		
02271	75	0100	CARDINAL	(type constant)
02275	75	001	LEN	(syll constant)
	76	0		
02301	76	0100000	Bit Length	(parameter)
02310	76	0110	CONSTANT	(syll constant)
02314	76	1110	ADDR	(syll constant)
02320	77	00001101100000	Address	(parameter)

ID next checks to see if the Lexic Levels are the same, and since they are, steps up the number of declarations-operands equal to the displacement parameter of ID plus one. Since the displacement parameter is equal to zero, one step up is made leaving the link pointer at 2051 octal (i.e. 1776 ⊕ 3727 = 2051). The integer A's declaration-operand is then found at the link pointer plus the link size (i.e. equal to 14 decimal in this example) or bit address 2067. ID next scans this declaration-operand for information from which it builds the second statement-operand as shown in Dump 10.

The Program Execution Stack now contains two statement-operands, which is necessary before the next S-UNIT (ASSIGN) can be executed. ASSIGN scans the two statement-operands and obtains the address of the value in the Cardinal Constant 3 declaration-operand and the address of the A's valuespace in the integer statement-operand. ASSIGN then retrieves the cardinal constant 3 and writes it into the A's valuespace. After writing the value into the proper valuespace, ASSIGN deletes the two statement-operands and creates a new statement-operand containing the same value as what was assigned (refer to Dump 11).

Dump 10. Declaration Stack as it Appears after ID

```
01760   63 000000000000000   link
01776   63                00 link
        64 111111111000
02014   64              1110 link
        65 1011111000
02032   65             10000 BEGIN          (mark constant)
02037   65                 0 Lexic Level    (parameter)
        66 000
02043   66    000000         Displacement   (parameter)
02051   66           0011101 link
        67 0000000
02067   67         0010       INTEGER        (type constant)
02073   67              0010  LEN            (syll constant)
02077   67                 1  Bit Length     (parameter)
        68 010000
02106   68        1010        VALUE          (syll constant)
02112   68           000000   Valuespace
02120   69 10001101000000    link
02136   69                00 INTEGER        (type constant)
        70 10
02142   70   1100            STEP           (syll constant)
02146   70        010000     Multiplier     (parameter)
02154   70              1100 STEP           (syll constant)
02160   71 011000            Multiplier     (parameter)
02166   71        1010       VALUE          (syll constant)
02172   71           000000  Valuespace
        72 0000000000000000
        73 00000000
02230   73          11011111 link
        74 000000
02246   74       11000       MARK           (mark constant)
02253   74            01100  link
        75 010000000
02271   75          0100     CARDINAL       (type constant)
02275   75             001   LEN            (syll constant)
        76 0
02301   76  0100000          Bit Length     (parameter)
02310   76         0110      CONSTANT       (syll constant)
02314   76             1110  ADDR           (syll constant)
02320   77 00001101100000    Address        (parameter)
02336   77                00 link
        78 000000000000
02354   78             0010  INTEGER        (type constant)
02360   79 0010              LEN            (syll constant)
02364   79     1010000       Bit Length     (parameter)
02373   79            1110   ADDR           (syll constant)
02377   79                0  Address        (parameter)
        80 1010010001000
```

Dump 11. Declaration Stack as it Appears after ASSIGN

```
01760   63 00000000000000   link
01776   63           00     link
        64 111111111000
02014   64         1110     link
        65 1011111000
02032   65        10000     BEGIN          (mark constant)
02037   65            0     Lexic Level    (parameter)
        66 000
02043   66    000000        Displacement   (parameter)
02051   66       0011101    link
        67 0000000
02067   67      0010        INTEGER        (type constant)
02073   67         0010     LEN            (syll constant)
02077   67            1     Bit Length     (parameter)
        68 010000
02106   68      1010        VALUE          (syll constant)
02112   68         011000   Valuespace
02120   69 10001101000000   link
02136   69           00     INTEGER        (type constant)
        70 10
02142   70   1100           STEP           (syll constant)
02146   70      010000      Multiplier     (parameter)
02154   70         1100     STEP           (syll constant)
02160   71 011000           Multiplier     (parameter)
02166   71      1010        VALUE          (syll constant)
02172   71         000000   Valuespace
        72 0000000000000000
        73 00000000
02230   73       11011111   link
        74 000000
02246   74      11000       MARK           (mark constant)
02253   74         01100    link
        75 010000000
02271   75       0010       INTEGER        (type constant)
02275   75          001     LEN            (syll constant)
        76 0
02301   76  1010000          Bit Length     (parameter)
02310   76       1010        VALUE          (syll constant)
02314   76          0110     Valuespace
        77 00
```

If instead of the statement 3→A;, the statement 3→A→B[1,1]; existed, this new statement-operand would be used to assign A to B[1,1]. However since the statement 3→A; is terminated by a semicolon (appearing in the Program String as a CLEAN'UP S-UNIT), the new statement-operand is deleted during the execution of CLEAN'UP as seen is Dump 12. In general, the function of CLEAN'UP is to delete everything on the Program Execution Stack back to the first MARK encountered. In this particular case there is only one MARK D-UNIT in the workspace and therefore only the new statement-operand was deleted. However, multiple MARK's are possible (such as when procedures exist) where MARK's are used to seperate operands having different lexic levels).

IV.1.6 A+4→B[1,2]

The first operation occurring in this statement is the summing of A and 4. This appears within the Program String as:

```
ID
                    0
                    0
CARD'CONST
                    3
                    4
S''ADD
```

The ID and the CARD'CONST create two statement-operands as shown in Dump 13. S''ADD scans these two statement-operands' attributes to acquire the value of A and the value of the cardinal constant 4, then pushes the two

values onto the pushdown stack. These values are then added and the sum is used to create a resultant statement-operand.

Dump 12. The CLEAN'UP operation of the Declaration Stack

```
01760   63 00000000000000   link
01776   63                00 link
        64 111111111000
02014   64              1110 link
        65 1011111000
02032   65             10000 BEGIN          (mark constant)
02037   65                 0 Lexic Level    (parameter)
        66 000
02043   66    000000         Displacement   (parameter)
02051   66           0011101 link
        67 0000000
02067   67         0010       INTEGER      (type constant)
02073   67            0010    LEN          (syll constant)
02077   67               1    Bit Length   (parameter)
        68 010000
02106   68        1010        VALUE        (syll constant)
02112   68           011000   Valuespace
02120   69 10001101000000   link
02136   69                00 INTEGER       (type constant)
        70 10
02142   70   1100            STEP         (syll constant)
02146   70        010000     Multiplier   (parameter)
02154   70             1100  STEP         (syll constant)
02160   71 011000            Multiplier   (parameter)
02166   71        1010       VALUE        (syll constant)
02172   71           000000   Valuespace
        72 0000000000000000
        73 00000000
02230   73          11011111 link
        74 000000
02246   74        11000      MARK         (mark constant)
```

The resultant statement-operand is then written on the Program Execution Stack as soon as the two statement-operands of ID and CARD'CONST are deleted. The result of the addition is shown in Dump 14.

Dump 13. The Declaration Stack as it appears prior to S''ADD

```
01760   63 00000000000000   link
01776   63           00 link
        64 111111111000
02014   64         1110 link
        65 1011111000
02032   65        10000  BEGIN       (mark constant)
02037   65            0 Lexic Level (parameter)
        66 000
02043   66   000000      Displacement (parameter)
02051   66       0011101 link
        67 0000000
02067   67      0010     INTEGER     (type constant)
02073   67         0010  LEN         (syll constant)
02077   67            1  Bit Length  (parameter)
        68 010000
02106   68      1010     VALUE       (syll constant)
02112   68        011000 Valuespace
02120   69 10001101000000  link
02136   69           00 INTEGER     (type constant)
        70 10
02142   70  1100          STEP        (syll constant)
02146   70     010000     Multiplier  (parameter)
02154   70         1100   STEP        (syll constant)
02160   71 011000         Multiplier  (parameter)
02166   71      1010      VALUE       (syll constant)
02172   71        000000  Valuespace
        72 0000000000000000
        73 00000000
02230   73       11011111 link
        74 000000
02246   74      11000     MARK        (mark constant)
02253   74         01000  link
        75 010000000
02271   75        0010    INTEGER     (type constant)
02275   75           001  LEN         (syll constant)
        76 0
02301   76 1010000         Bit Length  (parameter)
02310   76       1110      ADDR        (syll constant)
02314   76          0101   Address     (parameter)
        77 0010001000
02332   77        000000   link
        78 00000000
02350   78       0100      CARDINAL    (type constant)
02354   78          0010   LEN         (syll constant)
02360   79 1100000         Bit Length  (parameter)
02367   79      0110       CONSTANT    (syll constant)
02373   79         1110    ADDR        (syll constant)
02377   79            0    Address     (parameter)
        80 0000101010000
```

Dump 14. The Declaration Stack as it appears after S''ADD

```
01760   63 00000000000000   link
01776   63               00 link
        64 111111111000
02014   64             1110 link
        65 1011111000
02032   65            10000 BEGIN        (mark constant)
02037   65                0 Lexic Level  (parameter)
        66 000
02043   66   000000         Displacement (parameter)
02051   66          0011101 link
        67 0000000
02067   67        0010      INTEGER      (type constant)
02073   67            0010  LEN          (syll constant)
02077   67               1  Bit Length   (parameter)
        68 010000
02106   68       1010       VALUE        (syll constant)
02112   68          011000  Valuespace
02120   69 10001101000000   link
02136   69              00  INTEGER      (type constant)
        70 10
02142   70  1100            STEP         (syll constant)
02146   70       010000     Multiplier   (parameter)
02154   70           1100   STEP         (syll constant)
02160   71 011000           Multiplier   (parameter)
02166   71       1010       VALUE        (syll constant)
02172   71          000000  Valuespace
        72 0000000000000000
        73 00000000
02230   73         11011111 link
        74 000000
02246   74       11000      MARK         (mark constant)
02253   74           01000  link
        75 010000000
02271   75          0010    INTEGER      (type constant)
02275   75             001  LEN          (syll constant)
        76 0
02301   76  1010000         Bit Length   (parameter)
02310   76         1010     VALUE        (syll constant)
02314   76            0111  Valuespace
        77 00
```

When an addition is performed with a value larger than what can fit in the pushdown register, the procedure is slightly more involved than in this simple example. In this case, S''ADD works with chunks the size of the pushdown taking the least significant bits of the values first. Corresponding chunks from each value are placed in the pushdown, a carry is retained to propagate through to the next successive chunk, and the intermediate results are stored at the end of the workspace. If one value is longer than the other, the other value is zero filled during the remaining additions. The intermediate results are placed in the workspace so that the final answer has consecutive bits starting from the least significant to the most significant. The answer, however, is in two's complement form (since the pushdown stack works in two's complement arithmetic) and is converted to sign magnitude form. Since there is no more use for the two statement-operands that remain on the Program Execution Stack, they are deleted and S''ADD creates the resultant statement-operand in their place.

Back to the original statement A+4→B[1,2], the S-UNITS that are associated with the subscript [1,2] are:

```
START'LIST
CARD'CONST
                  1
                  1
CANONICAL
CARD'CONST
                  2
                  2
SUBSCRIPT'MODE
```

The function of these S-UNITS is to specify which element within integer B's valuespace is to be used. As seen in Dump 15, the START'LIST S-UNIT places the mark constant LIST on the Declaration Stack indicating the start of the subscript [1,2]'s attribute list. As shown in Dump 15 the CARD'CONST, CARDINAL pair provide the first CANONICAL statement-operand with a parameter called Canonical Value having the value of one (corresponding to the first subscript). Similarily, the CARD'CONST, SUBSCRIPT'MODE pair provide the second CANONICAL statement-operand corresponding to the second subscript. CANONICAL and SUBSCRIPT'MODE are basically identical except SUBSCRIPT'MODE is used to terminate the subscript list and has the additional function of placing SPLM into the subscript mode.

Dump 15. The Creation of the Subscript List for B[1,2]

```
01760   63 00000000000000   link
01776   63           00     link
        64 1'1111111000
02014   64         1110     link
        65 1011111000
02032   65        10000     BEGIN        (mark constant)
02037   65            0     Lexic Level  (parameter)
        66 000
02043   66    000000        Displacement (parameter)
02051   66       0011101    link
        67 0000000
02067   67      0010        INTEGER      (type constant)
02073   67         0010     LEN          (syll constant)
02077   67            1     Bit Length   (parameter)
        68 010000
02106   68      1010        VALUE        (syll constant)
02112   68        011000    Valuespace
02120   69 10001101000000   link
02136   69           00     INTEGER      (type constant)
        70 10
02142   70   1100           STEP         (syll constant)
02146   70      010000      Multiplier   (parameter)
02154   70         1100     STEP         (syll constant)
02160   71 011000           Multiplier   (parameter)
02166   71      1010        VALUE        (syll constant)
02172   71        000000    Valuespace
        72 0000000000000000
        73 00000000
02230   73       11011111   link
        74 000000
02246   74      11000       MARK         (mark constant)
02253   74         01010    link
        75 010000000
02271   75        0010      INTEGER      (type constant)
02275   75           001    LEN          (syll constant)
        76 0
02301   76  1010000          Bit Length   (parameter)
02310   76       1010        VALUE        (syll constant)
02314   76          0111     Valuespace
        77 00
02322   77   01110010000000 link
02340   78 10100            LIST         (mark constant)
02345   78     00101011100  link
        79 000
02363   79    11011         CANONICAL    (mark constant)
02370   79       10000000   Canonical Value (parameter)
        80 000000
```

```
02406   80        0100001110 link
        81 0000
02424   81     11011         CANONICAL   (mark constant)
02431   81           0100000 Canonical Value (parameter)
        82 0000000
02447   82         111000001 link
        83 01011
02465   83      11100        END         (mark constant)
```

Subscript mode indicates to ID that a subscript attribute list exists. In subscript mode, ID proceeds to step down the Program Execution Stack until it finds a mark constant LIST, scans the subscript attributes, and determines the address of the particular element within the valuespace (in this case B[1,2]) that is being referenced. The subscript attribute list is then deleted and a statement-operand pointing to the valuespace is then created as shown in Dump 16. In this particular case, the address is pointing to the last element of B (i.e. B[1,2]) at bit 2223 octal in memory.

All that remains to be executed is a simple assignment of the valuespace of the first statement-operand to the valuespace of the second statement-operand. Dump 17 shows the workspace after the assign. Note that a CLEAN'UP S-UNIT is not needed here because it is the end of the program. TERMINATE is simply used to terminate the execution phase, thus ending the example.

Dump 16. The Creation of the Statement-Operand for B[1,2]

```
01760   63 00000000000000   link
01776   63           00     link
        64 111111111000
02014   64         1110     link
        65 1011111000
02032   65        10000     BEGIN        (mark constant)
02037   65            0     Lexic Level  (parameter)
        66 000
02043   66       000000     Displacement (parameter)
02051   66       0011101    link
        67 0000000
02067   67         0010     INTEGER      (type constant)
02073   67         0010     LEN          (syll constant)
02077   67            1     Bit Length   (parameter)
        68 010000
02106   68         1010     VALUE        (syll constant)
02112   68       011000     Valuespace
02120   69 10001101000000   link
02136   69           00     INTEGER      (type constant)
        70 10
02142   70         1100     STEP         (syll constant)
02146   70       010000     Multiplier   (parameter)
02154   70         1100     STEP         (syll constant)
02160   71 011000           Multiplier   (parameter)
02166   71         1010     VALUE        (syll constant)
02172   71       000000     Valuespace
        72 0000000000000000
        73 00000000
02230   73       11011111   link
        74 000000
02246   74       11000      MARK         (mark constant)
02253   74         01010    link
        75 010000000
02271   75         0010     INTEGER      (type constant)
02275   75          001     LEN          (syll constant)
        76 0
02301   76  1010000         Bit Length   (parameter)
02310   76        1010      VALUE        (syll constant)
02314   76         0111     Valuespace
        77 00
02322   77   01110010000000 link
02340   78 0010             INTEGER      (type constant)
02344   78     1110         ADDR         (syll constant)
02350   78       11001001   Address      (parameter)
        79 001000
```

Dump 17. The Workspace as it appears at the end of the Execution Phase

```
01760   63 00000000000000   link
01776   63           00     link
        64 111111111000
02014   64         1110     link
        65 1011111000
02032   65        10000     BEGIN          (mark constant)
02037   65            0     Lexic Level    (parameter)
        66 000
02043   66   000000         Displacement   (parameter)
02051   66       0011101    link
        67 0000000
02067   67     0010         INTEGER        (type constant)
02073   67         0010     LEN            (syll constant)
02077   67            1     Bit Length     (parameter)
        68 010000
02106   68     1010         VALUE          (syll constant)
02112   68        011000    Valuespace
02120   69 10001101000000   link
02136   69           00     INTEGER        (type constant)
        70 10
02142   70  1100            STEP           (syll constant)
02146   70    010000        Multiplier     (parameter)
02154   70        1100      STEP           (syll constant)
02160   71 011000           Multiplier     (parameter)
02166   71     1010         VALUE          (syll constant)
02172   71        000000    Valuespace
        72 0000000000000000
        73 00001110
02230   73      01011111    link
        74 000000
02246   74     11000        MARK           (mark constant)
02253   74         01010    link
        75 010000000
02271   75       0010       INTEGER        (type constant)
02275   75          101     VALUE          (syll constant)
        76 0
02301   76  01110           Valuespace
```

Chapter V

Clock Level Simulator

V.1 Introduction

The Clock Level Simulator (CLS) is a software tool used in designing SPLM for a particular application. The designer enters design parameters (e.g. memory size, memory word length, arithmetic register length) during the initialization of CLS. These parameters are used to simulate a specific configuration of SPLM. Statistics are then gathered on register use and execution times, and the optimal design can be determined by comparing these statistics for various sets of design parameters.

The register use statistics provide the designer with the necessary information to determine the best register configuration. The most frequently used registers should be implemented with fast hardware registers, but little used registers can be located in memory or eliminated altogether.

The timing statistics from CLS provide information on the execution speed of a design, and may point to a need for additional hardware to increase the speed of a design. For example, it may be necessary to add a barrel shifter dedicated to I/O for applications with heavy I/O requirements. It may also be necessary to implement

arithmetic operators between arrays in hardware for applications with a large number of array manipulations.

In order to obtain pertinent statistics, however, benchmark programs to be run on CLS should be carefully developed. These programs should contain typical instruction mixes for a given application and must exercise all of the intended machine functions to insure that the design meets the requirements of a given application. Any ommission in the benchmark programs will diminish the benefits of using the simulator and could cause design errors.

One of the benefits of using CLS is that hardware is designed from software. The designer can use the benchmark programs to analyze the functions that must be accomplished and the statistics to determine the minimal hardware configuration that will accomplish these functions. Thus CLS provides for cost effective designs since hardware is added only when needed to increase performance.

During construction and implementation of the design, the simulator continues as an important tool. During construction of the machine, the simulator can be an aid since the simulator routines can be easily converted into microcode. During implementation, the simulator can also be used to debug application programs. Workspace dumps should be extremely helpful in locating program errors, and these dumps should decrease debugging time.

Since the simulator can be used during the design,

construction, and implementation of a machine, CLS is a versatile and valuable tool. For a specific application, this versatility should lower the life-cycle costs of both the hardware and software of SPLM.

The remainder of this chapter describes the major software routines of CLS with emphasis on the simulator routines, and the way in which the hardware structures are simulated.

V.2 Software Structure of the Simulator

The simulator software is composed of three basic units: the initialization routine, the simulator routines, and the dump routine. These three units combine to make the simulator an important design tool for SPLM. Figure 11 shows the main units of CLS, and the following sections describe these basic units of the simulator.

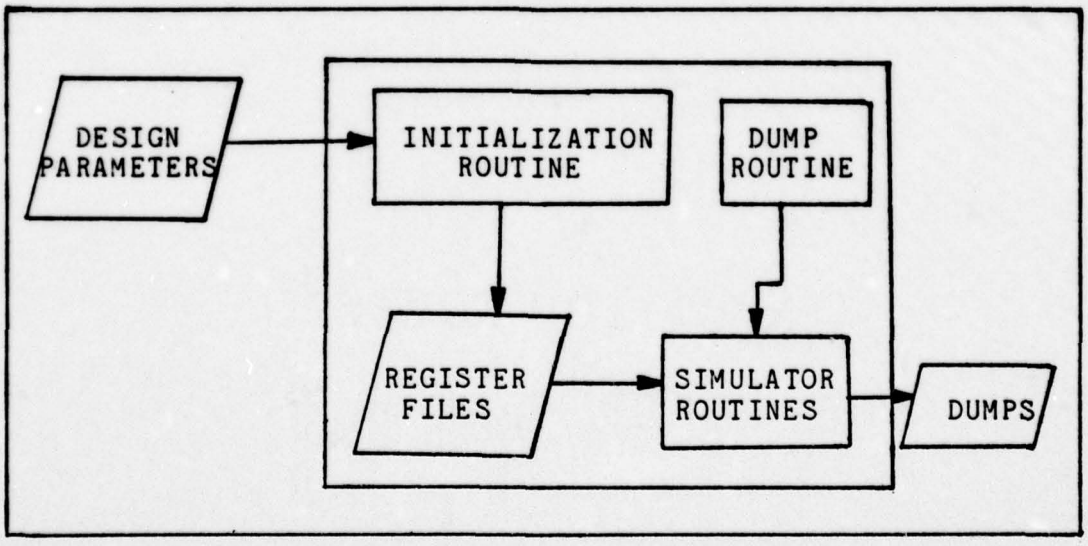


Figure 11. Software Structure of CLS

V.2.1 Initialization Routine

The initialization routine is used to set up register and memory configurations from design parameters. This program allows the designer to interactively enter design parameters and change one or all of the design parameters from a previous run. The initialization routine also provides the option of checking the parameter file at any point during the initialization run. This eliminates the need for the designer to remember the parameters he has already entered.

The basic function of the initialization routine is to output a register file using the design parameters. This register file contains the simulated registers and their bit lengths as determined by the design parameters. For example, if the design parameter "Memory Word Length" were entered in as 16, then the simulated registers MAR and MIR would be 4 bits long and 16 bits long respectively.

Appendix A explains the initialization routine in more detail and contains instructions on entering design parameters.

V.2.2 Simulator

The simulator provides for interpretive execution at the register level of programs written in SPLML. The simulator uses the register file generated by the initialization routine and the program string to perform the simulation of the program.

The simulator is a hierarchical structure with control in the higher level routines and the actual processing in the lower levels. Every S-UNIT corresponds to a routine in the simulator, and these S-UNIT routines are called by a main processing routine (P'SCAN'LOOP).

P'SCAN'LOOP is the program scan loop where S-UNITS are fetched from the program string and executed. This loop continues the fetch and execute of S-UNITS until a stop is encountered. The stop condition occurs upon execution of the TERMINATE S-UNIT or as a result of an error condition (e.g. array subscript out of range).

A functional flowchart of P'SCAN'LOOP is provided in Figure 12, and the following paragraphs explain these functions in more detail.

After the stop condition has been tested, P'SCAN'LOOP processes any interrupts that have occured during the execution of the previous S-UNIT. An interrupt is processed by calling the appropriate interrupt procedure as declared in the program (see Chapter II Section 4.6).

The next routine in P'SCAN'LOOP to be executed is the TEST'TRAP routine which checks the FAULT register for an occurance of a soft extension trap. If the FAULT register is set, then the previous operator ("faulty operator") is not fully implemented in hardware; and a trap to the appropriate routine in the soft extension code is then executed (these routines are listed in Reference [2], pages 2-4-45 to 2-4-49).

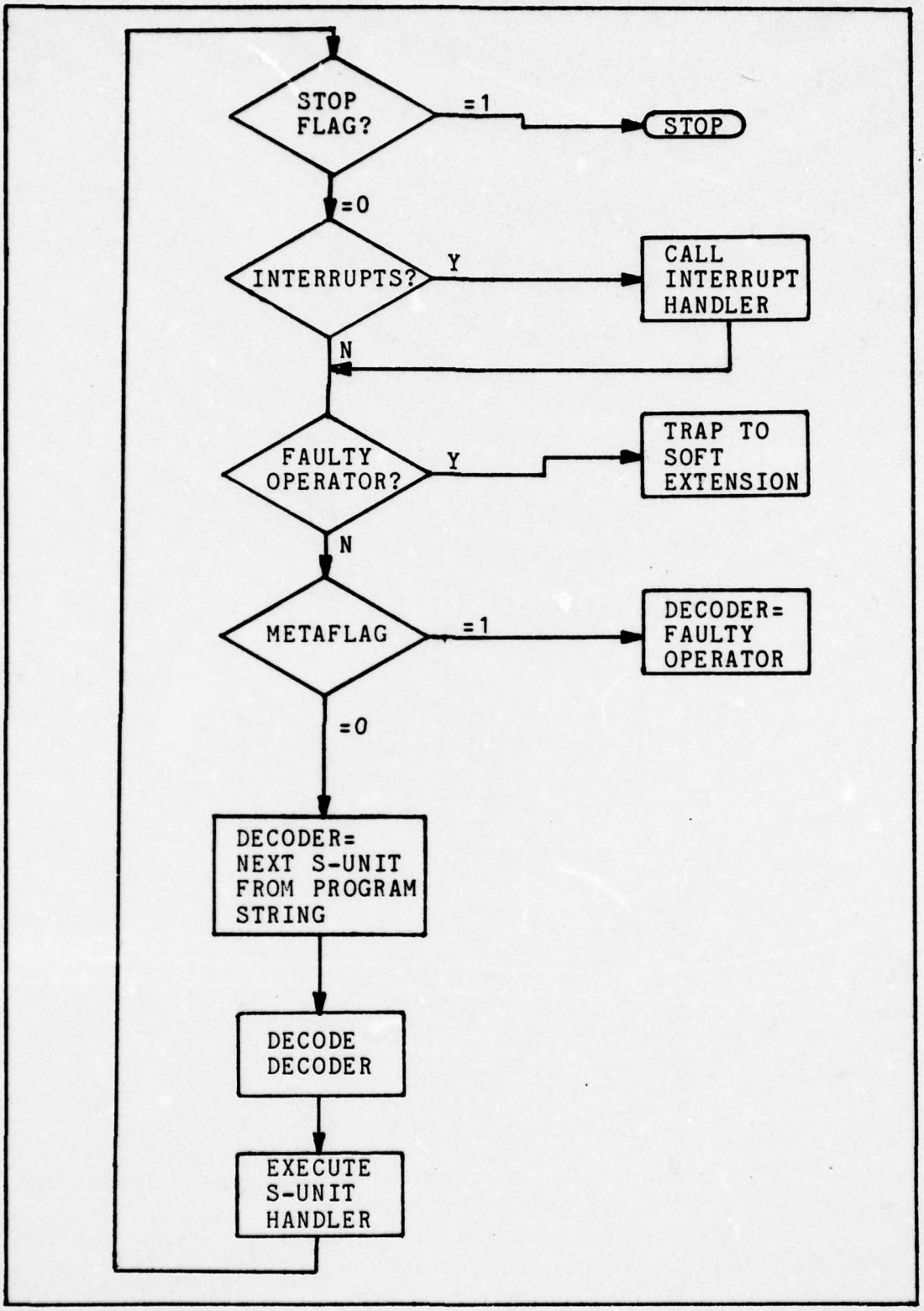


Figure 12. Flowchart of P'SCAN'LOOP

The trap is basically a procedure call, but this fact is hidden from the programmer by use of the soft extension. For example, if A, B, and C are arrays and array addition is not implemented in hardware then the statement "A+B→C;" will force a trap to a soft extension. This soft extension will perform the addition of the A and B arrays and store the result in the C array.

The dyadic soft extension listed below is a generalized soft extension which performs all operations between entire arrays.

```
OP(Y,X)  "DYADIC PATTERN"
BEGIN
     DECL R(X MIX Y,
       DIM(IF RANK X > RANK Y THEN X ELSE Y)
            (LEN Y MAX LEN X));
     REF A TO RAVEL X;
     REF B TO RAVEL Y;
     REF C TO RAVEL R;
     REF J TO NENT X;
     CARDINAL I BITS INDEX;
     IF   -(X COMMENSURATE Y) THEN ERROR ELSE
     DO  A[I] ? B[I]→C[I] UNTIL 1+→I=J;
END
```

The "?" is the operator parameter; and during translation, "?" generates the FAULT-OP S-UNIT. The operator parameter (?) eliminates the need for specific operators to be included in the soft extension and decreases the number of soft extensions required to extend all operators to arrays.

The next step in P'SCAN'LOOP is the loading of the DECODER register. This register contains the next S-UNIT to be executed and is loaded in one of the following two ways:

1) If METAFLAG is set (this flag is set during execution of FAULT-OP), a soft extension is being executed and DECODER is loaded with the S-UNIT for the "faulty operator".

2) If METAFLAG is clear, the next S-UNIT is fetched from the program string and loaded into DECODER.

A sequence of IF statements is then executed in P'SCAN'LOOP to determine the S-UNIT in the DECODER register. Once the S-UNIT has been decoded, the routine that handles the S-UNIT is executed and control returns to the beginning of P'SCAN'LOOP. This loop is continued until a stop condition is encountered which terminates the simulation.

In the next 4 sections, the simulation of the hardware structures of the machine will be discussed.

V.2.3 Registers and Memory

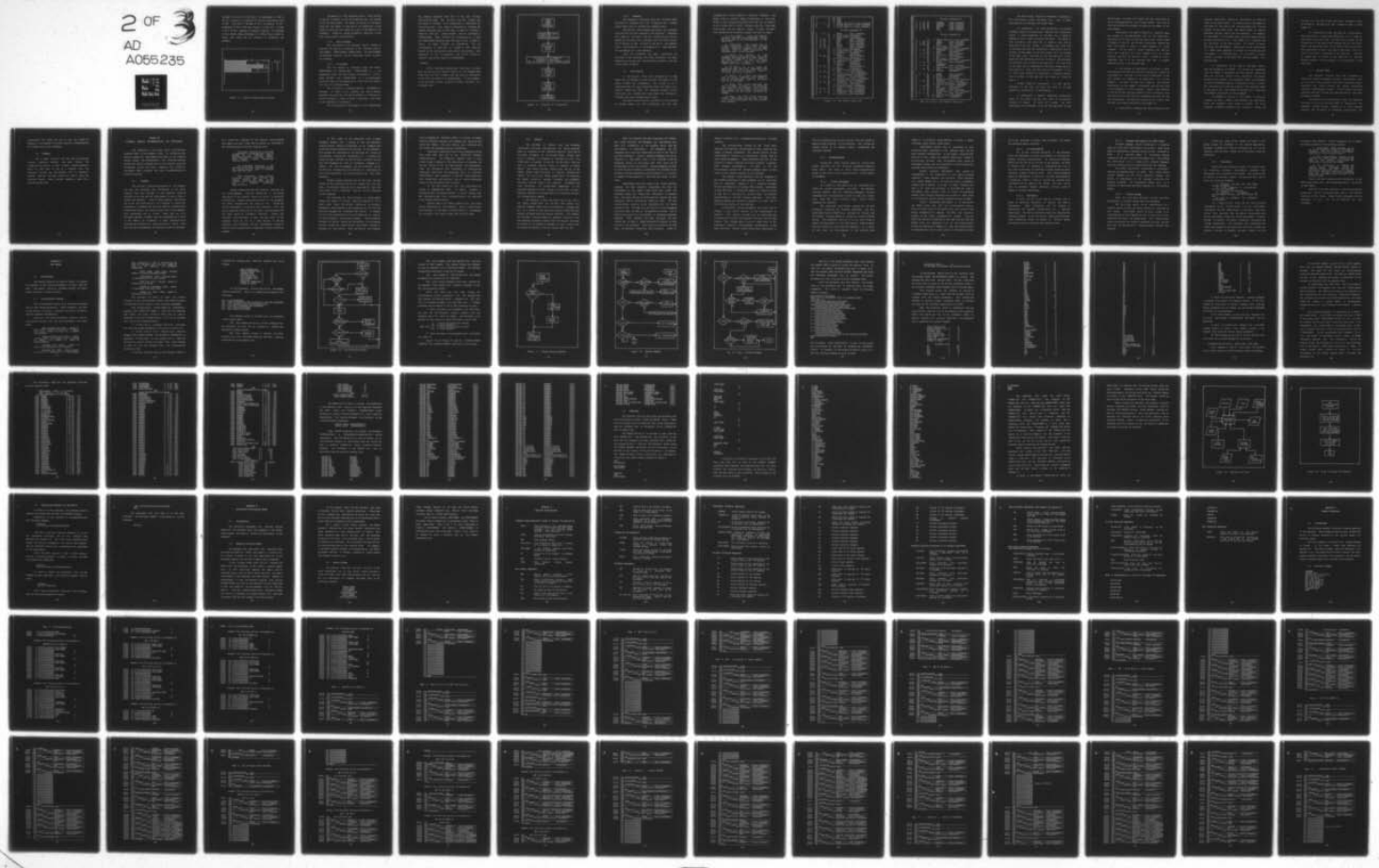
Registers and memory are simulated using the JOVIAL table structure, and their exact features are set up during the initialization run. Register transfers are simulated using the equal sign (=), thus the JOVIAL statement "LAMBDA=ALPHA;" simulates the register transfer between LAMBDA and ALPHA. The packed table structure of JOVIAL is used to take advantage of the JOVIAL features of zero fill and truncation of variables of differing lengths.

Figure 13 shows an example of a table containing simulated registers of varying lengths. The variable to simulate the MIR register has been declared to occupy bits 0

AD-A055 235 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
SIMULATION OF THE DIRECT EXECUTION OF A HIGHER ORDER LANGUAGE.(U)
MAR 78 D L AKIN, B C ALLEN

UNCLASSIFIED AFIT/GCS/EE/78-1 NL

2 OF 3
AD
A055235



through 15 of word 0 of the table. An assignment of MIR to OP1 will result in a truncation of the most significant bits of MIR. Only bits 2 through 16 will be assigned to OP1. Assignment of OP1 to MIR will result in a zero fill of bits 0 and 1 of MIR. Register to register transfer is simulated in this manner taking advantage of a JOVIAL feature which is faster than using the bit function to accomplish the transfer.

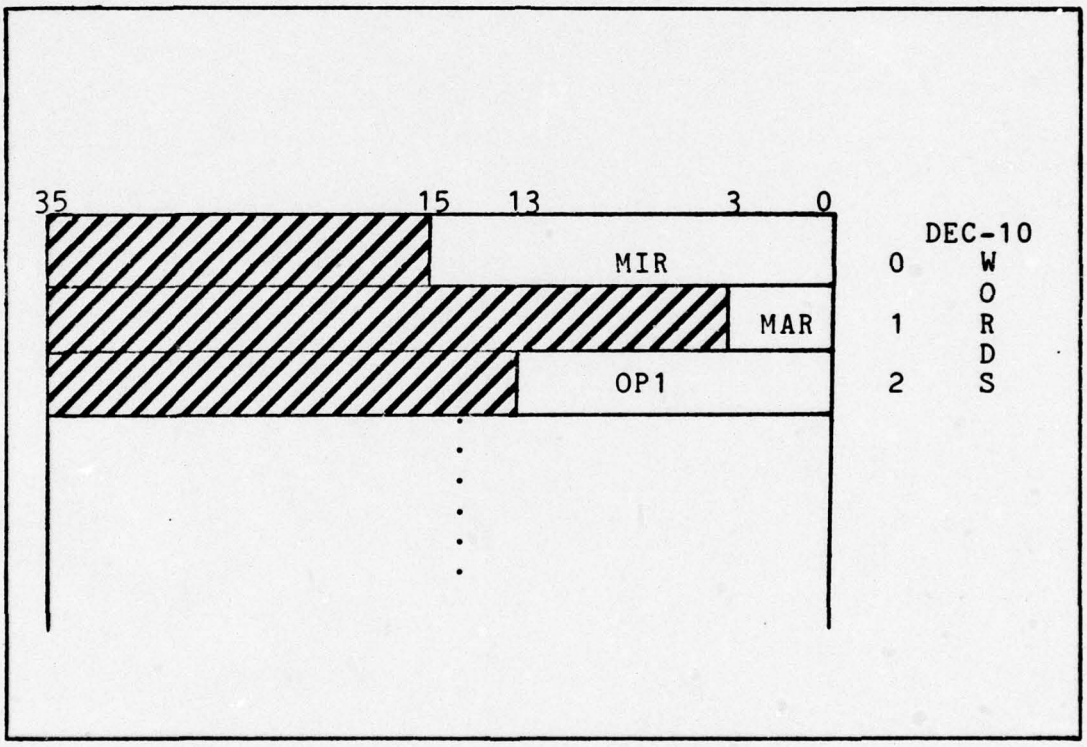


Figure 13. Register Packed Table Structure

The memory is also simulated using a table similar to the one in Figure 13 with the exception that all entries are of the same length. The number of entries in the memory table and the length of each entry are determined by the number of words and the length of a word in the memory to be simulated. These are design parameters specified by the user during system initialization.

V.2.4 Barrel Shifter

The operation of the simulated barrel shifter is basically the same as is discussed in the hardware section. LOAD'BARREL, PRESET'BARREL, BARREL'SHIFT, and WRITE'BARREL are simulator routines and are extensively called by other CLS routines.

V.2.5 I/O Section

The I/O section is simulated using two arrays: STREAM'ARRAY and STREAM'TIMES. STREAM'TIMES is a one dimensional array, and each element corresponds to a port's frame transfer time. STREAM'ARRAY is a two dimensional array with each row corresponding to a port and each element of a row is a simulated frame.

The I/O section is checked whenever LOAD'BARREL is executed. If there is I/O pending, the barrel shifter registers are saved and the barrel shifter is dedicated to the I/O operation. Figure 14 shows a functional flow chart of the simulated I/O operation.

The I/O operation is simulated by first determining

the elapsed simulated time (ET) of the last I/O scan (LAST'IO'SCAN'TIME). The I/O ports are then scanned, and the number of "frames" to be transferred is determined for each active port. This number is obtained by dividing the elapsed simulated time by the time it takes to transfer a frame for that port (frame transfer time is contained in STREAM'TIMES). These "frames" are transferred to or from the STREAM'ARRAY depending upon the type of port (output or input). As these "frames" are transferred, the row corresponding to that port is rotated so that each row element may be used several times during a simulation. Upon completion of the I/O operation, the barrel shifter is restored, and control returns to LOAD'BARREL.

Example:

If the simulated elapsed time since last I/O scan were .06 and the first element (PORT 0) of the STREAM'TIMES array were .02, then 3 frames (.06÷.02) would be transferred to the first row (PORT 0) of STREAM'ARRAY. In this example, .02 is the time of frame transfer for PORT 0 and PORT 0 is an output port.

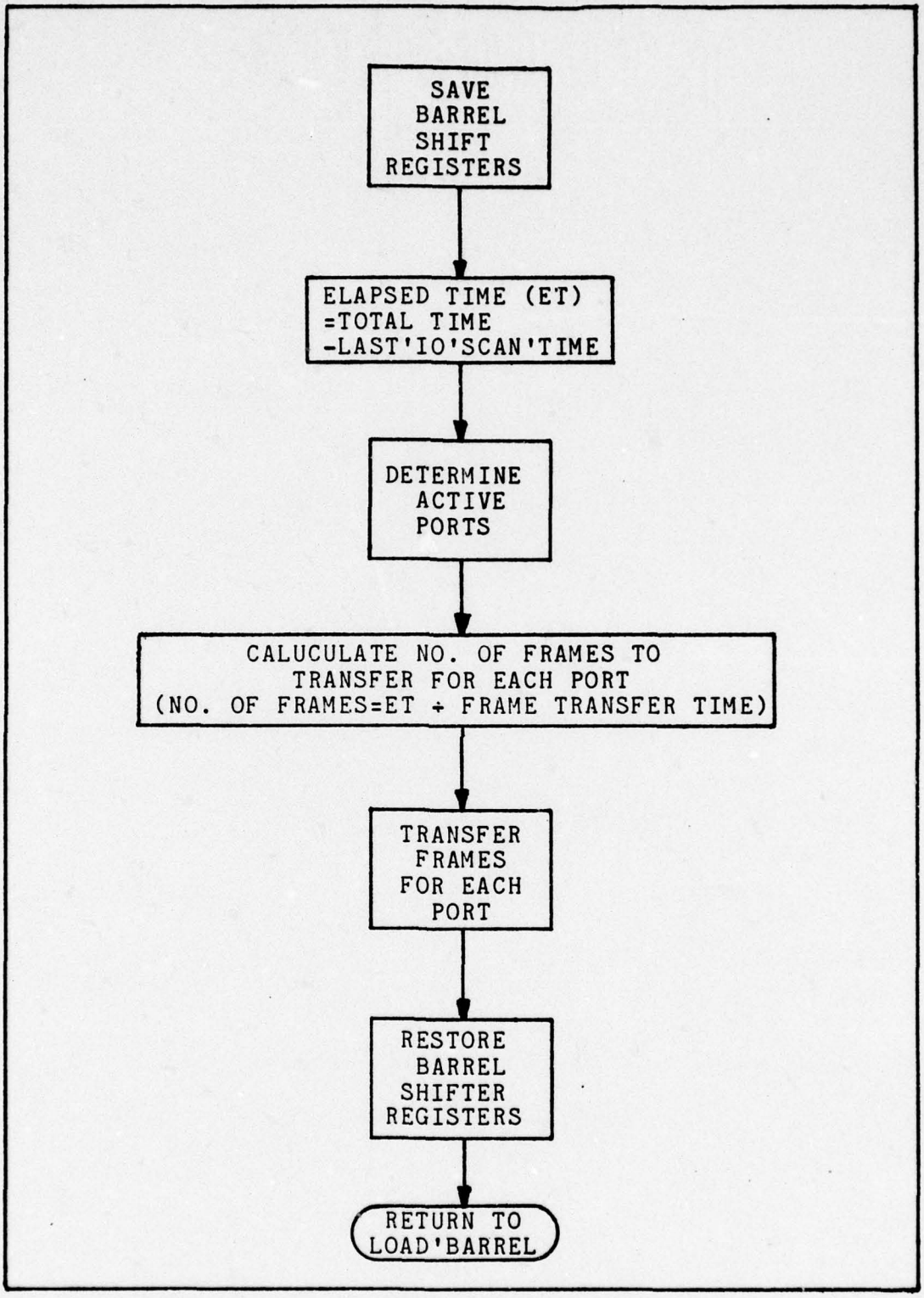


Figure 14. Flowchart of I/O Operation

V.2.6 Pushdown

The pushdown is simulated using the variables PDS'A and PDS'B to represent the A and B registers and a JOVIAL packed table (PDS) to simulate the pushdown stack.

The POP and PUSH pushdown operations are simulated using the POP and PUSH routines of the simulator. Registers are pushed on the stack with the V'PUSH routine and are popped off the stack into a register with the V'POP routine. The VT pointer is used to point to the top of the pushdown stack. The VT pointer is incremented for a PUSH operation and decremented for a POP operation.

Since all arithmetic and logic operations are performed on the pushdown, the JOVIAL arithmetic and logic operators are mostly used between PDS'A and PDS'B with the result stored in PDS'B.

V.3 Dump Routine

In the original study, the designers had to dump memory and go through by hand dividing the memory in the proper D-UNITS and valuespaces. This proved to be a cumbersome task for large simulations where the number of D-UNITS in memory was high. The designers pointed to this difficulty as a problem that would make the simulator a difficult and time consuming tool to use.

The dump routine provides a solution to this problem by parsing memory and thus eliminating the most time

consuming and tedious aspect of using the simulator. The method used in parsing memory incorporates a link list. (Note: In the following discussion the link list is referred to as a match list to avoid confusion with the links that exist within the CLS memory). Figure 15 shows the match list containing the following columns from left to right:

a) Line number - Lines are numbered starting with one and incremented by one. The first line is the starting point in the match list.

b) Instruction - This column has an integer value from 1 to 6 each of which provides different 'commands' to the dump routine. Each of these instructions is discussed later in this section.

c) Value - The parsing of the workspace involves reading a value from the workspace and comparing it to the number in the value column. If the two values compare, there is a 'match'; if they don't compare there is a 'no match'.

d) No Match - If a no match was determined from the value comparison, this number is used as the next line number of the match list to be 'executed'.

e) Match - If a match was determined from the value comparison, this number is used as the next line number of the match list to be 'executed'.

f) Size - This column specifies the size (in number of bits) of the comparison value read from the workspace. This is the only column in the match list requiring change should the SPLM design parameters be changed.

g) Name - The name column holds the mnemonic description of the type of item (i.e. D-UNIT or parameter) that is currently being checked.

```
 1   1                    2    14  link
 2   1                    3    14  link
 3   1                    4    14  link
 4   2    1    6          5     1  *check for mark or type constant*
 5   3                   10     5  *force match to a mark constant*
 6   3                   50     4  *force match to a type constant*
 7
 8                              ** mark constants **
 9
10   4    1   13          11        BEGIN        (mark constant)
11   5                    12     4  Lexic Level (parameter)
12   5                     3     6  Displacement (parameter)
13   4    3   14           3        MARK         (mark constant)
14   4    5   15           3        LIST         (mark constant)
15   4    7   16           3        END          (mark constant)
16   4    9   17           3        NULL         (mark constant)
17   4   11   18           3        COLON        (mark constant)
18   4   13   23          19        STATE        (mark constant)
19   5                    20     4  Lexic Level (parameter)
20   5                    21    14  Program Pointer (parameter)
21   5                    22     6  Dimension    (parameter)
22   5                     3     3  Mode         (parameter)
23   4   15   29          24        STATEOP      (mark constant)
24   5                    25     4  Lexic Level (parameter)
25   5                    26    14  Program Pointer (parameter)
26   5                    27     6  Dimension    (parameter)
27   5                    28     3  Mode         (parameter)
28   5                     3     7  Decoder      (parameter)
29   4   17   32          30        SET          (mark constant)
30   5                    31    14  Bound        (parameter)
31   5                     3    14  Stack Pointer (parameter)
32   4   19   35          33        PROC         (mark constant)
33   5                    34     4  Current Lexic Level(parameter)
34   5                     3    14  Program Pointer (parameter)
35   4   21   38          36        OPERATOR     (mark constant)
36   5                    37     4  Current Lexic Level (parameter)
37   5                     3    14  Program Pointer (parameter)
38   4   23   42          39        INTERRUPT    (mark constant)
39   5                    40     4  Lexic Level  (parameter)
40   5                    41    14  Program Pointer (parameter)
41   5                     3     4  Interrupt Number (parameter)
42   4   25   44          43        LABEL        (mark constant)
43   5                     3    14  Program Pointer (parameter)
44   4   27    3          45        CANONICAL    (mark constant)
45   5                     3    14  Canonical Value (parameter)
46
```

Figure 15. The 'Match' (Link) List

```
47
48                                   ** type constants **
49
50   4    0   51   60        BOOLEAN      (type constant)
51   4    2   52   60        CARDINAL     (type constant)
52   4    4   53   60        INTEGER      (type constant)
53   4    6   54   60        REAL         (type constant)
54   4    8    3    3        NULL         (type constant)
55
56
57                                   ** syll constant **
58
59
60   3              61    4  *force match to a syll constant*
61   4    1   62   64        INPUT        (syll constant)
62   4    2   63   64        OUTPUT       (syll constant)
63   4    8   66   64        PORT         (syll constant)
64   5              65    2  Port Number  (parameter)
65   3              66    4  *force match to a syll constant*
66   4    9   68   67        STEPBOUND    (syll constant)
67   5              69    6  Bound        (parameter)
68   4    3   70   69        STEP         (syll constant)
69   5              65    6  Multiplier   (parameter)
70   4    4   73   71        LEN          (syll constant)
71   5              72    7  Bit Length   (parameter)
72   3              73    4  *force match to a syll constant*
73   4    5   75   74        VALUE        (syll constant)
74   6               3        Valuespace
75   4    6   77   76        CONSTANT     (syll constant)
76   3              77    4  *force match to a syll constant*
77   4    7   80   78        ADDR         (syll constant)
78   5               3   14  Address      (parameter)
79
80   4   10    3   81        PATCH        (syll constant)
81   5              82    6  Offset       (parameter)
82   5              83    6  Bound 1      (parameter)
83   3              84    4  *force match to a syll constant*
84   4   11   87   85        SEGSPACE     (syll constant)
85   5              86    6  Physical Multiplier (parameter)
86   5              83    6  Logical Multiplier (parameter)
87   4    4   89   88        LEN          (syll constant)
88   5              90    7  Bit Length   (parameter)
89   4    6   91   90        CONSTANT     (syll constant)
90   3              91    4  *force match to a syll constant*
91   4   12    3   92        SADDR        (syll constant)
92   5              93   14  Stack Pointer (parameter)
93   5               3   14  Bound 2       (parameter)
```

Fig 15. (Cont.) The 'Match' (Link) List

The dump routine parses the workspace according to the instructions within the match list. Each of these instructions are discussed and listed below.

a) Instruction 1 is used exclusively for writing workspace links to the dump file. Whenever this instruction occurs, a value comparison is not made and a match is assumed to occur, known as forcing the match. The number of bits written to the dump file is equal to the size as specified in the size column. For example, the first line in the match list has the instruction 1 indicating a link of bitsize 14 (the number in the size column) is to be written to the dump file. This line is the starting point of the match list and therefore the first 14 bits of the workspace are the first bits written to the dump file. The next line in the match list is determined by the match column which in this case is 2. Thus the next line to be executed is line 2. Line 2 also contains an instruction 1; therefore another link is written to the dump file and a match is forced to line 3.

Another aspect of instruction 1 is that the bit addresses of the last two links are saved to be used whenever an instruction 6 is encountered.

b) Instruction 2 makes a comparison between the value in the match list and the value at the current location in memory. If there is a match, the next instruction to be executed is on the line specified by the

match column. If there is no match, the next instruction is on the line specified by the no match column. The size of the value within the workspace is given by the number in the size column.

Instruction 2 is used to check for a specific value and branch to the appropriate line in the match list. Instruction 2 is found in only one line of the match column of Figure 15 (line 4) and is used to determine whether the next item after a link is a mark constant or a type constant. If the item is a mark constant, the next bit after the link will be a one whereas if it is a type constant, the next bit will be a zero. A match in line 4 specifies line 5 to be executed next and a no match specifies line 6 to be executed next.

c) The purpose of instruction 3 is to read a value from memory to be used with the next instruction (always an instruction 4). The size of the value is specified in the size column. An example of an instruction 3 is in line number 5. In order to reach line number 5, the previous instruction, in line number 4, determined that the next item in the workspace is a mark constant. Since a mark constant is five bits long, the size column in line 5 contains a five and five bits are read from the valuespace and saved for the next instruction. Instruction 3 always forces a match, thus the next line number executed is line number 10.

d) Instruction 4 compares the value obtained in the

previous instruction (always an instruction 3) with the value in the value column. If a match exists, the number of bits equal to the size in the previous match list line is written to the dump file. The match column is used to determine the next match list line executed. For example, if line number 10 is executed next, the value saved from the previous instruction (i.e. line number 5) is compared with one, the value in the value column. If a match is found, the value and the mnemonic description in the name column (i.e. BEGIN (mark constant)) is written to the dump file and line number 11 is executed next. If a no match is found, nothing is written to the dump file and line number 13 is executed next.

e) Instruction 5s are used in the match list to dump the parameters of D-UNITS. If a D-UNIT has a parameter list, the number of parameters and their pattern remain the same and only the values of the parameters may change. Therefore the location of a parameter within the workspace is always predictable making it possible to force a match. Thus instruction 5 writes the portion of the workspace equal to the size in the size column to the dump file along with the mnemonic description of the parameter.

An example of this is in line number 11 which was branched to after a BEGIN mark constant has been found. Since the parameter Lexic Level is always the first parameter in BEGIN's parameter list, a match is forced, the

four bit (i.e. the size of the Lexic Level parameter) value is written to the dump file, and a branch is made to line number 12.

f) Instruction 6 uses the last two link pointers, saved previously by executing instruction 1s, in order to skip ahead to the next link. The next link is found by the exclusive-or of the first link's address and the last link's value, the same method used in the SPLM to step up one link. This instruction is used only once in the match list (i.e. line 74) to write a valuespace to the dump file. The area in memory that is written to the dump file is from the present location in the workspace to the beginning of the next link.

V.4 Current State

The simulator routines have been translated to JOVIAL and extensive debugging has been accomplished. Most of the lower level routines have been thoroughly tested, especially the barrel shifter and memory referencing routines. Approximately one-half of the S-UNITS have been exercised and most of these have been declaration and basic arithmetic S-UNITS.

The code for the initialization, dump and simulator routines is on file at the AFIT School of Engineering. The simulator routines contain comments as to their intended function and the amount of debugging accomplished. The

instructions for using CLS are in the user manual in Appendix A, and Chapter VI contains specific recommendations for further work on the simulator.

V.5 Summary

The 3 major units of CLS are the initilization routine, simulator routines, and dump routine. The initialization routine is used to enter design parameters which are then used to set up a register file. The simulator routines use the register file to simulate a specific SPLM and the dump routine is used to parse the simulated memory. These routines combine to make CLS a powerful design tool.

Chapter VI

Problems, Results, Recommendations, and Conclusions

The objective of this study was to solve several problems found of the original study. To accomplish this several changes and improvements were made to the original design, coding, and documentation. The following sections discuss the various problems found in the original study and encountered during this study, summarize the results in overcoming these problems, and make recommendations for follow-on studies.

VI.1 Problems

The two major problems encountered in the original CLS were the workspace size limitations of APL which required that only sections of the simulator could be executed at one time and the large amount of time it took to execute the simulator. Both of these problems were mainly due to the fact that each bit of a register or memory was simulated by a complete APL word and register transfers were accomplished using a bit function where bits of the register were transferred one at a time. Since each bit of a simulated register or memory word was represented by a word of APL memory, an arithmetic or logic operation first required the collection of these bits into a vector. Once this had been accomplished, the operation could be executed.

As a result the simulator of the original study required much memory and took a long time to execute as reflected in the following quotes from the original work:

> "The memory limitations of APL make it impossible to integrate more than 10 percent of the complete system. Exercising anything more than a few S-units becomes rapidly impractical (Ref [4]: 59)."

> "An experiment requiring a few declaration and some arithmetic S-units can require several hours of terminal time (Ref [4]: 56)."

> "Most experiments have involved execution of half dozen S-units. The largest sequence has been in the vicinity of 50 S-units (more than two days of terminal connection (Ref [4]: 59)."

Another problem was that the original simulator was not fully debugged. This was largely due to the abrupt termination of the original study as a result of financial difficulties. Another contributing factor to the debugging not being completed was the choice of APL. During the initial development of the Clock Level Simulator (CLS), APL was an excellent choice due to its high interactiveness and its large amount of arithmetic functions. However, the simulator grew enormously in size and only parts of the simulator could be run at a time because of the APL workspace size limitation and the unreasonable amount of time it took to execute even a moderately complex simulation program.

In this study it was especially hard to begin debugging because only a listing of the APL simulation program existed making it impossible to run examples that would have helped comprehend the workings of the simulator. This problem was compounded by the fact that the listing of the simulator program was incomplete (i.e. routines were called by other routines but did not have a listing) and the fact that debugging was done in a different language (i.e. JOVIAL). The task of debugging was also difficult due to the fact the documentation was difficult to understand and very little discussion was directed toward the Clock Level Simulator.

Another problem resulted from the fact that the APL version of the translator was not in a usable form for this study. The program string had to be developed by hand, and this proved to be a difficult task for all but the most trivial programs.

The simulator was also difficult to debug simply because the design of a direct execution machine is not a simple task. Since the original design was not formally documented, understanding of the design had to come from the partially debugged and incomplete APL listing. An example of one of the complexities in the CLS design can be seen by the way information was stored in memory. The information in memory is tightly packed and the ability to distinguish the bits within one item from the bits within another is necessary to test results. This was done in the original

work by dumping the simulator memory in a column of memory words and manually drawing lines between each seperate item within the memory. This was tedious, time consuming work and slowed the debugging process considerably.

Some major hindrances occured while debugging during this study that were due to the JOVIAL/J73 compiler characteristics. The JOVIAL/J73 compiler used in this effort allows a maximum of 256 subroutines in a program, but the APL version of CLS has approximately 400 subroutines. Thus, 150 subroutines were eliminated by replacing the subroutine calls with the subroutine code. It should be noted however that a majority of the 150 subroutines were only called once and therefore did not require duplicate copies of the subroutine code to exist.

In the APL version of CLS, the subroutines are listed in alphabetical order. In JOVIAL, however, a subroutine must be declared before it can be called; thus the subroutines had to be reordered before the simulator would compile without errors.

Several bugs in the JOVIAL compiler were also found during compilation of the simulator. Most of these errors were confined to the TRACE directive which is a debugging aid included in the JOVIAL system used for this study.

VI.2 Results

The problems of memory size and workspace limitations have been overcome with the JOVIAL simulator where an execution of the simulator requires about 40K 36 bit words of memory. In the current JOVIAL version, each bit of a register or of memory is simulated by a bit in a JOVIAL word as opposed to the APL version which used one word to simulate each bit within a register or within memory. This memory savings results from the use of the JOVIAL packed table structure to simulate registers and memory. The use of the JOVIAL packed table for registers and memory allows for register transfers and arithmetic functions without using a bit function. Arithmetic and logic operations are accomplished immediately on the simulated registers without having to first form a vector. Register transfers are accomplished with truncation and zero fill automatically handled.

The simulator is also much faster due to the use of the JOVIAL packed table and the fact that JOVIAL is a compiled language rather than an interpretive language like APL. As a result of this study the execution speed has been greatly increased using the present simulator. For example, the reference program example in Appendix D Section 1 took approximately 40 seconds connect time (33 seconds CPU time) to execute its 81 S-UNITS as opposed to 2 days connect time to execute 50 S-UNITS in the APL version (Ref [4]: 59).

After the simulator had been translated into JOVIAL, this study continued the debugging and improvements were made (e.g. readability of the program source code and improvements in algorithms). The debugging was not completed due to the time constraints as well as the lack of the SPLM translator. However, complete debugging was not an objective of this study since it was realized from the beginning review of this study that this could not be accomplished in the time alloted. Enough debugging was made to allow follow-on studies to understand the workings of the simulator through executing existing programs (i.e. as found in Chapter IV and Appendix D) which was the intent of this study.

Improvements have been made in the design whenever feasible. The most important improvement has been the development of the memory dump routine that parses the simulated memory and outputs the information in readable form. This dump routine has eliminated the tedious and time consuming task of parsing the memory by hand. The dump routine should prove to be an invaluable tool for future studies of CLS. The dump will not only aid in the debugging that still must be done on the simulator routines, but it should also prove to be very helpful in debugging benchmark programs. After hardware has been developed, the dump routine can be used to debug application programs as they are run on the simulator. Thus, CLS can continue to be used after the hardware design has been finalized. (Refer to

Chapter V Section 3 for a complete description of the dump routine).

The initialization routine of the Clock Level Simulator was improved and provides for easy insertion and change of design parameters. The current version of the initialization routine is an interactive program used to enter design parameters. These parameters are used by the routine to create a register file which reflects a specific SPLM hardware configuration. In order to run the initialization routine the hardware designer needs to know little of the internal workings of the simulator.

The original study contained very few examples of the hardware execution of SPLML programs, all of which were hard to comprehend. Chapter IV is a detailed, straight forward explanation of an example SPLML program. Admittedly it does take time to follow the example in Chapter IV step by step but it does give the serious reader an excellent understanding of the SPLM hardware operation. Four other example programs are included in Appendix D in order aid to any follow-on studies of the simulator.

A motivating concern throughout this study was to produce a tool which required as few compilations as possible. This concern has been met by separating the initialization routine, the simulator routines, and the dump routine. Thus changes made to one of these units do not necessarily require a corresponding recompilation in the other two units. Another feature which saves compilation is

that the Program String is read in from a file and stored in memory during execution of the simulator. This allows the Program String to be changed without recompiling the simulator routines.

VI.3 Recommendations

During this study, various needs for further work became apparent; and the following paragraphs summarize these needs. The order in which these recommendations appear is the order in which these follow-on studies should be attempted.

VI.3.1 Further Debugging

As a result of this study, CLS was translated into JOVIAL and various improvements were made. The simulator routines, however, were not completely debugged; and this debugging must be accomplished in order to make CLS a useful tool. Thus the first follow-on study should finish debugging the simulator.

The areas that need further testing are the soft extensions including the trapping mechanism, the more complicated dyadic S'UNITS (e.g. multiply and divide), and the Huffman routines needed to execute an encoded Program String. The translator will be needed in this study to enable the development of the wide range of SPLML programs that are required to fully test the simulator. As a result of this need, the meta-compiler of the original study

(Phase I) is currently being modified to output a JOVIAL translator under another study effort.

Improvements should also be considered in this follow-on study; especially in the pushdown stack and the basic register configuration. In the example programs that were run in this study, the pushdown stack never needed to be more than two deep. Thus the pushdown stack should be thoroughly studied, and the possibility of restructuring the pushdown stack should be considered.

Another possible improvement that should be investigated is the elimination of the general purpose registers. These registers are the Greek letter registers (e.g. ALPHA, LAMBDA, etc.), and each may contain information ranging from an address to a flag. As these registers are used, it is often very difficult to determine the type of information they contain. Thus it should be determined whether other existing registers can be used in their place or if additional special purpose registers should be added.

Another recommended improvement is to incorporate the MATCH (link) list into the initialization routine. Since the SIZE column of the MATCH (link) list varies as the design parameters are changed, the MATCH list should be changed as the design parameters are changed. Thus the MATCH list should be generated at the same time the ITEM and DEFINE files are generated (i.e. by the initialization routine as explained in Chapter V). This improvement should be attempted only after a wide variety of programs have been

run on the simulator to ensure that the MATCH list parses the simulated memory correctly.

VI.3.2 I/O Improvements

The I/O and interrupt structure of the simulator should be redesigned to enable the designer to interrupt the program execution interactively. An interactive terminal should also be used as an I/O port so the designer may send or receive data. In addition to redesigning the I/O, it will also be necessary to determine a timing mechanism which simulates program execution time. This timing mechanism should be implemented to allow for accurate approximations of execution times in order to provide the designer with information on the speed of the design. This data can be used to determine whether additional hardware might be needed to increase performance.

VI.3.3 Statistics

In order for CLS to be used as a design tool, a means of gathering statistics on register use must be implemented. These statistics will be helpful in determining the register configuration for a specific application. CLS must be carefully studied to determine the optimal placement of the statistical gathering mechanisms. These mechanisms should be placed to retrieve useful information without cluttering the simulator code.

VI.3.4 Language Improvements and SPLML Manual

The SPLM language should be examined in a detailed study for areas where improvements can be made. These areas might be determined by recoding various avionic programs in SPLML. As a possible means of increasing program readability and thus lowering maintenance cost, data typing and other Pascal (i.e. a higher order computer language) oriented features could be incorporated into the language.

This study should also concentrate on producing a detailed programming manual for SPLML. This manual should include comments on the direct execution concept, the syntax, an explanation of the language constructs, and detailed examples. The programming manual will be a necessity in developing benchmark programs for the hardware design.

VI.3.5 Hardware Design

After the previously mentioned studies have been accomplished, a hardware design should be attempted.

The first step should be the determination of a workable subset of SPLML and which S-UNITS are generated by this subset. This subset should be broad enough to be representative of the language but not so broad that it will complicate and delay the design phase. This simple design effort should give the designer experience in using CLS and also test the feasibility of designing SPLM hardware from software.

In the next phase of the design, the scope of the design should be broadened to the desired application. Benchmark programs can then be developed and run on the simulator, and the final design can be developed using the statistics.

VI.4 Conclusions

Since this was a follow-on study, the previous study was carefully reviewed, anticipated problems were determined, and an objective was reached. The objective was to develop an economic, viable, and practical tool and to solve the following problems:

1) Simulator was slow and used large amounts of memory.
2) Little debugging of the simulator had been accomplished.
3) Debugging was difficult.
4) Simulator contained an initialization routine that was difficult to use.
5) Lack of examples and inadequate documentation.

As a result of this study, the two major problems encountered in the original study (i.e. the large memory requirement and the large amount of execution time) were solved. Both execution time and memory requirements were greatly decreased, largely due to the development of better methods in register and memory transfers, operations and storage simulations. The simulator was translated into JOVIAL and debugged to the point that follow-on studies can execute a variety of examples and gain insight into the

workings of the simulator. Further debugging will be easier and faster because of the following factors:

1) Large sample SPLML programs can be run on the simulator without the problems of excessive time and memory requirements experienced in the original study.

2) The initialization routine of the simulator has been reworked increasing the ease of use without sacrificing the philosophy of parameterized design.

3) The difficulty in reading the tightly packed information within the simulator memory has been overcome by parsing the information before it is dumped. This will decrease considerably debugging time since the memory does not have to be parsed by hand.

Finally the much needed overall explanation of the SPLM design effort has been accomplished due to the result of this thesis.

Thus CLS is now the economical, viable and practical design tool set out to be developed both in the original study and in this study. Though further debugging will be necessary, it will not be as difficult and time consuming.

Appendix A

User Manual

A.1 Introduction

This manual explains the steps to follow in running the routines of CLS and gives examples of files that are used. This manual should be carefully studied to insure proper execution of these routines.

A.2 Initialization Routine

The initialization routine is an interactive program used to enter design parameters. These parameters are used by the routine to create a register file which reflects a specific hardware configuration.

There are nine design parameters that are used in creating the register file, and these parameters are listed below.

WORD ADDRESS SIZE (WAS)- Number of bits needed to address the bits of a memory word (i.e. number of bits of a memory word=2**WAS).

MEMORY ADDRESS SIZE (MAS)- Number of bits needed to address all of the words of memory (i.e. number of words of memory=2**MAS).

EXPONENT SIZE (EXS)- Number of bits in the exponent of real numbers.

MANTISSA SIZE (MNT)- Maximum number of bits in a mantissa of a real number.

This parameter is used in determining the size of registers that hold the number of bits in the mantissa for arithmetic operations.

LEXIC LEVEL SIZE (LLS)- Maximum number of nested BEGIN/END blocks.

DISPLACEMENT (DSP)- Maximum number of declarations in a lexic level.

STEP SIZE (STP)- Maximum number of elements in an array.

NUMBER OF INTERRUPTS (INT)- Number of simulated interrupt lines.

NUMBER OF PORTS (PRT)- Number of simulated I/O ports.

The flowchart in Figure 16 shows the various functions of the initialization routine, and these functions are described in more detail in the following paragraphs.

Then initialization routine begins by requesting a register file ("INPUT FILE NAME-"). After the file name has been given, the first record of this file is read to determine if the file already exists. If it is an existing file, one of the following occurs:

1) If the file is a register file from a previous run, then the design parameters do not have to be set to 0.

2) If the file is not a register file, then the message "FILE ALREADY EXISTS! DO YOU WISH TO OVERWRITE?" is generated. If the answer to this question is no, then the routine will ask for another file name. This file checking is done so that the designer does not inadvertently overwrite a file.

If the file does not exist or the designer wishes to

overwrite an existing file, then the register file is as follows.

```
WORD ADDRESS SIZE     =        0
MEMORY ADDRESS SIZE   =        0
EXPONENT SIZE         =        0
MANTISSA SIZE         =        0
LEXIC LEVEL SIZE      =        0
DISPLACEMENT SIZE     =        0
STEP SIZE             =        0
NUMBER OF INTERRUPTS  =        0
NUMBER OF PORTS       =        0
```

The initialization routine then prints the message "TYPE HLP FOR HELP". "HLP" is a command which generates the following:

```
LPR = LIST PARAMETERS
REG = COMPUTE REGISTER SIZES ACCORDING TO THE NEW PARAMETERS
LRG = LIST REGISTERS AND CORRESPONDING SIZES
LST = LIST BOTH PARAMETERS AND REGISTERS
END = END INITIALIZE ROUTINE
HLP = HELP (PRINT THIS TEXT)
```

The commands printed in the HELP file are explained in more detail below.

LPR- This command prints the current parameter file and eliminates the need for the designer to remember the parameters already entered.

REG- This command creates a register file that contains all of the simulated registers and their sizes as determined by the parameter list.

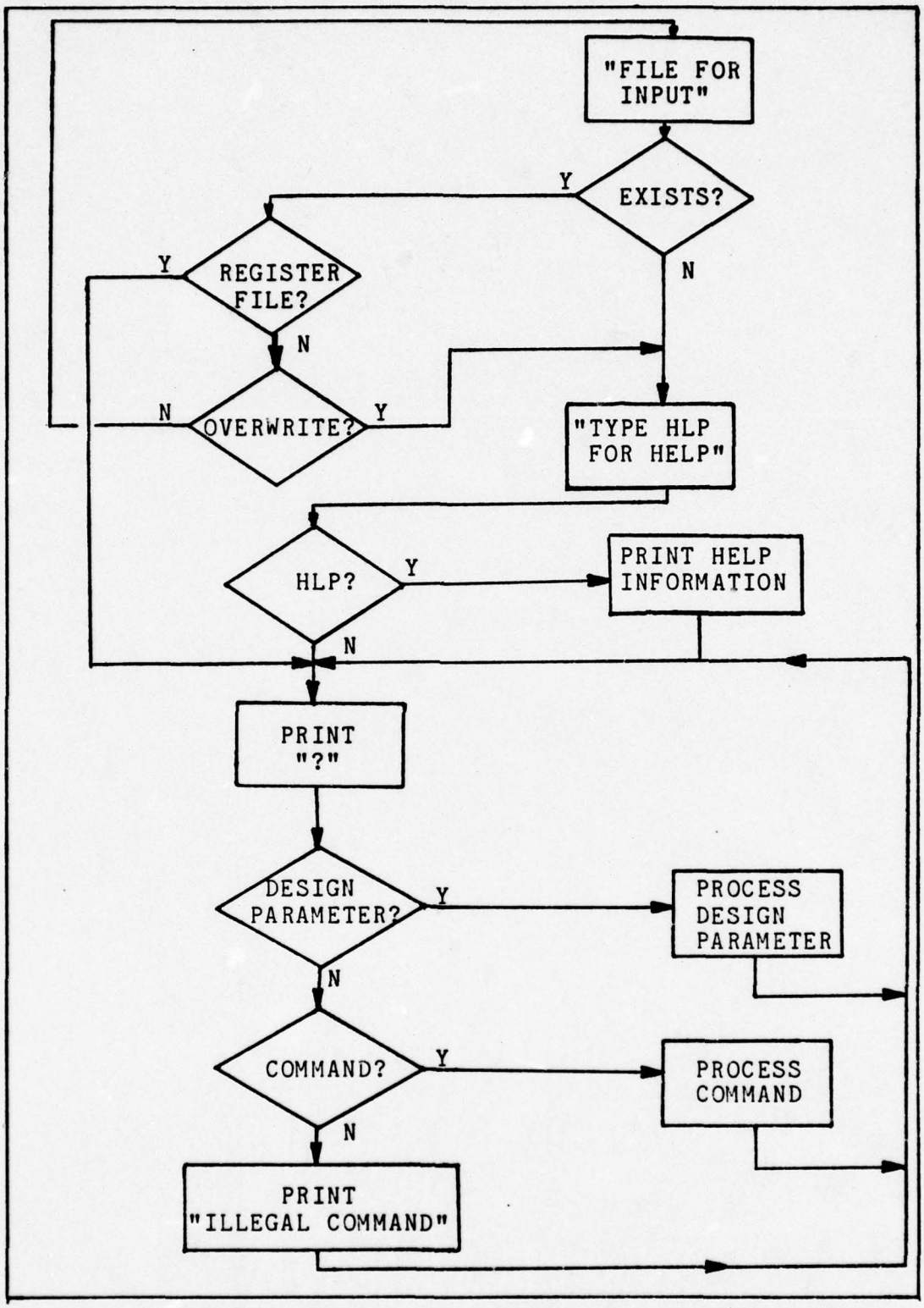


Figure 16. Initialization Routine

LRG- This command lists the register list that was created by "REG" command. This command enables the designer to scan the register file to determine whether the register configuration generated is the one he wanted.

LST- Same command as "LRG" except that the design parameters are listed with the registers.

END- After design parameters have been entered and the register file created, this command is entered to end the initialization routine.

After the HELP file has been listed, the initialization routine generates a "?". The designer may then respond by entering either a command (e.g. LST, LPR, etc.) or a design parameter (e.g. WAS, MAS, etc.). A design parameter may be entered in one of the following two ways.

1) Enter the name of the parameter (e.g. WAS, MAS), and when the initialization routine responds with the parameter and a "?"; enter the number. The following is an example of this process in which STEP SIZE is set to 200.

```
            ?   ----From initialization routine
          STP   ----From designer
STEP SIZE   ?   ----From initialization routine
          200   ----From designer
```

2) Enter the name of the parameter with the number (e.g "WAS 4").

Figure 17 and Figure 18 show the "PROCESS DESIGN PARAMETER" and "PROCESS COMMAND" functions in more detail.

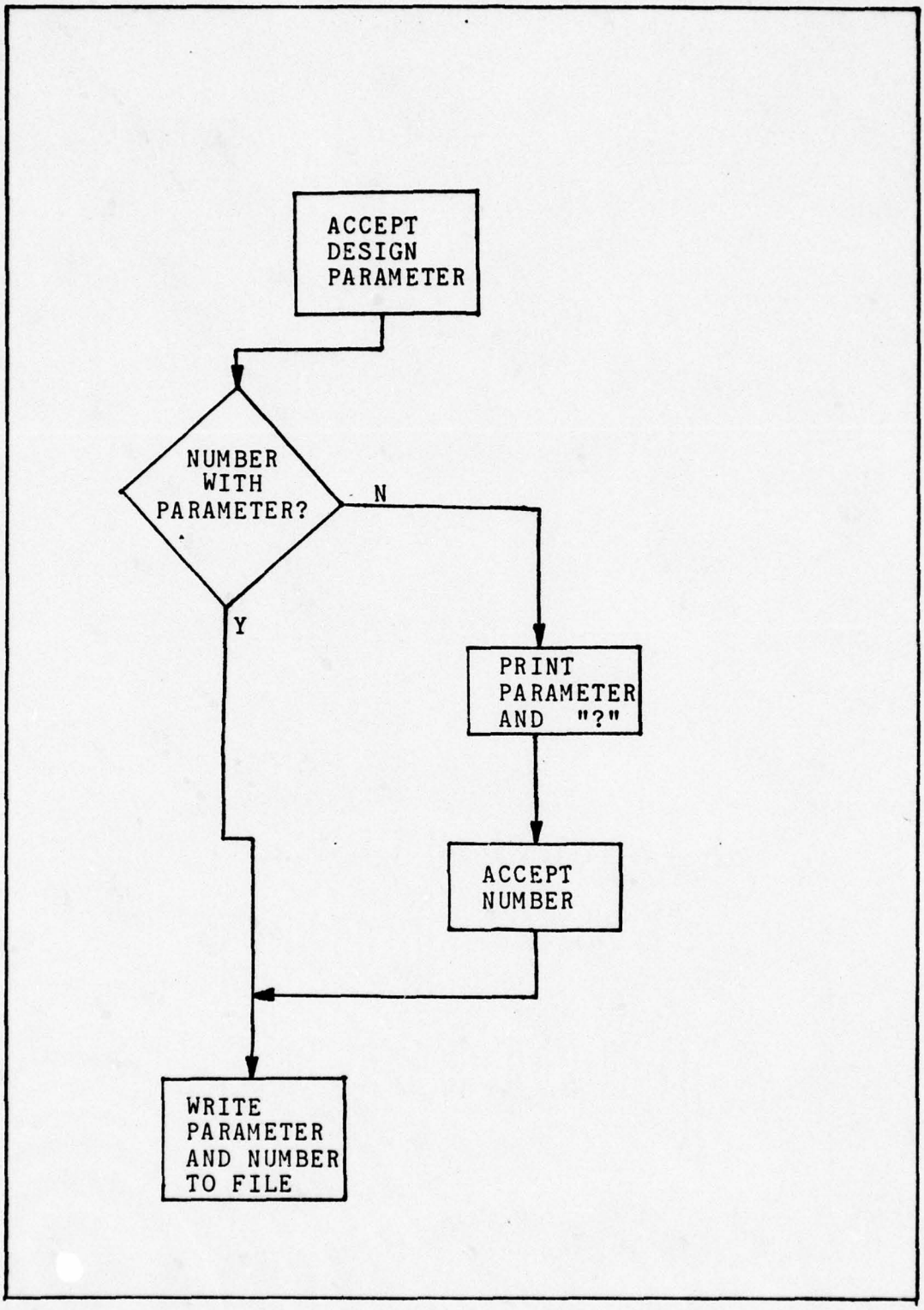


Figure 17. PROCESS DESIGN PARAMETER

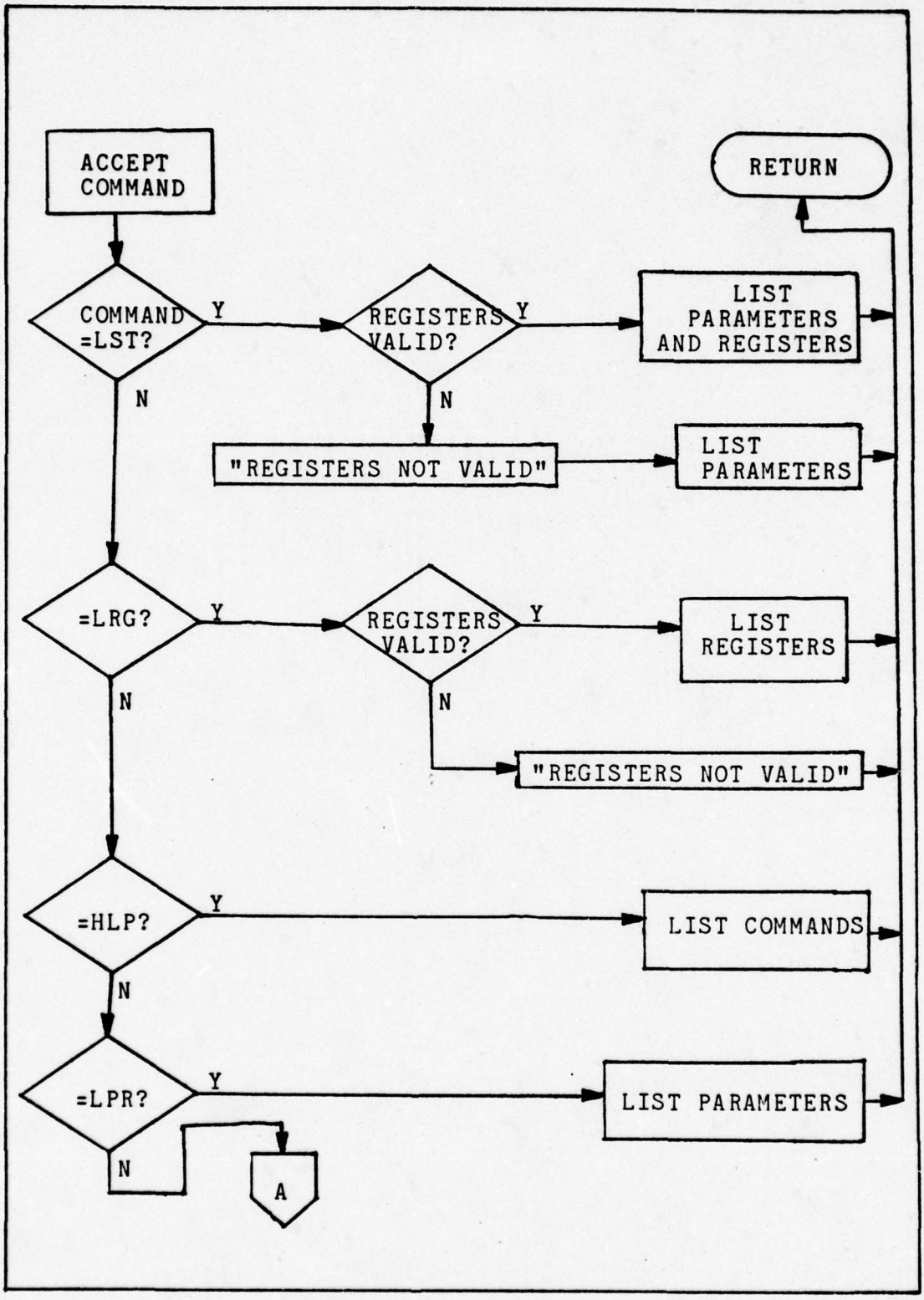


Figure 18. PROCESS COMMAND

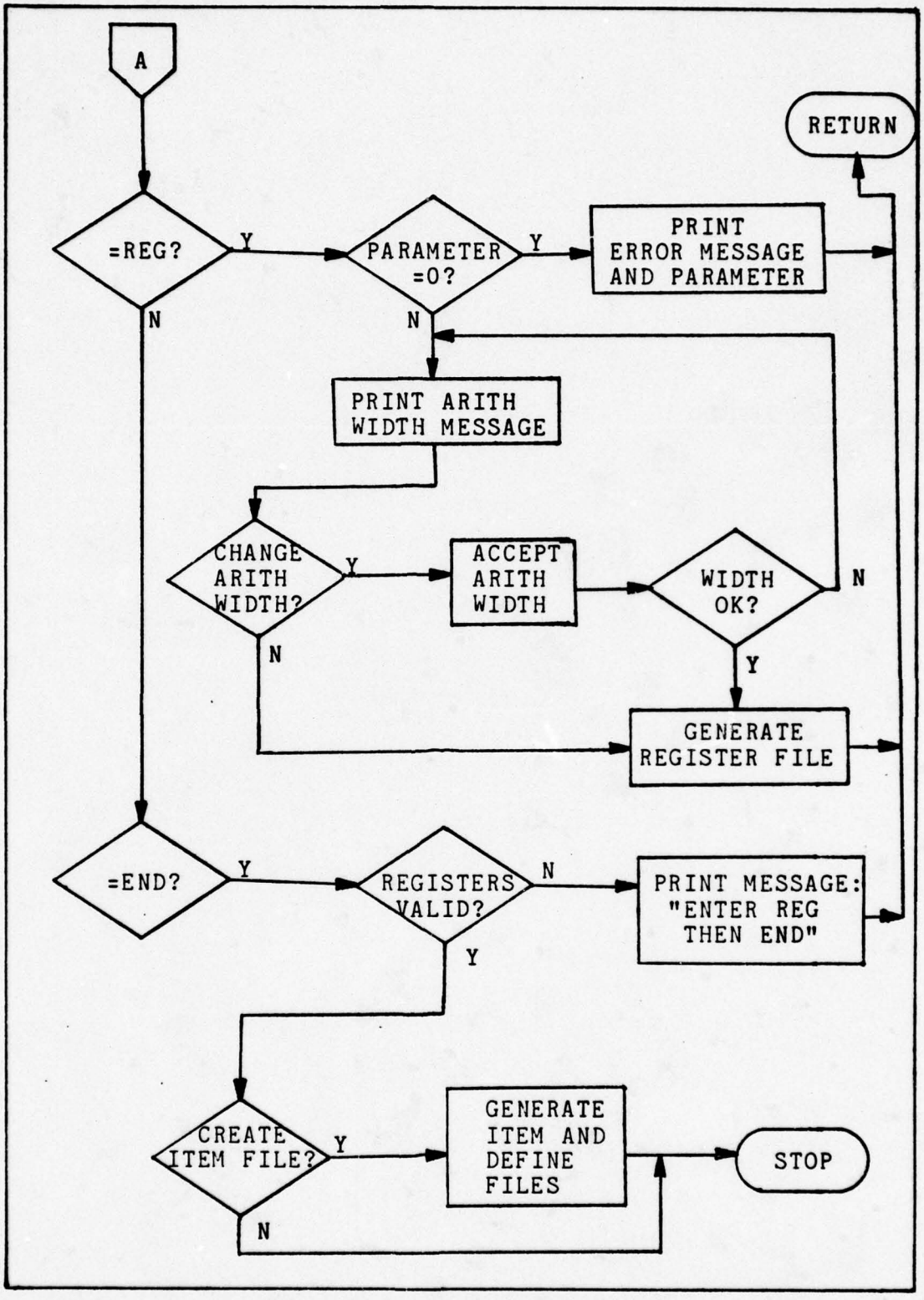


Fig 18. (Cont.) PROCESS COMMAND

When all of the design parameters have been entered the command "REG" is given to create the register file. If there are any design parameters less than or equal to 0, then the message "ONLY POSITIVE NONZERO PARAMETERS ARE VALID FOR COMPUTING REGISTERS" will be printed. The invalid parameter will also be given with the offending value.

After the parameters have been checked, the minimum width of the arithmetic unit is computed using the design parameters. This width is computed using the following JOVIAL procedure.

```
PROC SET'UP'ARITH'WIDTH;
BEGIN "SET UP ARITHMETIC WIDTH OF PUSHDOWN STACK"
ITEM ONE U =1;
WORD'SIZE=SHIFT(ONE,WORD'ADDR'SIZE);
    "WORD'SIZE=2**WORD'ADDR'SIZE"
MEMORY'SIZE=SHIFT(ONE,MEMORY'ADDR'SIZE);
    "MEMORY'SIZE=2**MEMORY'ADDR'SIZE"
ARITH'WIDTH=WORD'ADDR'SIZE+MEMORY'ADDR'SIZE;
IF ARITH'WIDTH<MANTISSA'SIZE;
     ARITH'WIDTH=MANTISSA'SIZE;
IF ARITH'WIDTH<LEXIC'LEVEL'SIZE;
     ARITH'WIDTH=LEXIC'LEVEL'SIZE;
IF ARITH'WIDTH<DISPLACEMENT'SIZE;
     ARITH'WIDTH=DISPLACEMENT'SIZE;
IF ARITH'WIDTH<EXPONENT'SIZE+1;
     ARITH'WIDTH=EXPONENT'SIZE+1;
IF ARITH'WIDTH<STEP'SIZE;
     ARITH'WIDTH=STEP'SIZE;
IF ARITH'WIDTH<WORD'SIZE;
     ARITH'WIDTH=WORD'SIZE;
IF ARITH'WIDTH<DECODER'SIZE;
     ARITH'WIDTH=DECODER'SIZE;
     "DECODER'SIZE IS THE SIZE OF THE DECODER REGISTER"
ARITH'WIDTH=ARITH'WIDTH+1;
END
```

This arithmetic width (ARITH'WIDTH) is then printed along with the message "DO YOU WANT TO INCREASE THE ARITHMETIC WIDTH?". For example, if the minimum arithmetic width is 17 then the following message would be printed.

```
ARITHMETIC WIDTH =        17
DO YOU WANT TO INCREASE THE ARITHMETIC WIDTH?
```

If the designer enters "yes" to the question, then the message "ENTER NEW ARITHMETIC WIDTH-" is printed. The designer may then enter the new width. The new width must be greater than or equal to the minimum arithmetic width, or the original arithmetic width message will be printed again.

This process allows the designer to specify an arithmetic width which is larger than the minimum width as computed from the design parameters. This enables the designer to specify a larger arithmetic width to increase the speed of the arithmetic operations of an SPLM.

When the arithmetic width is determined from the above process, then the size of the pushdown stack registers (PDS'A and PDS'B) are set to the arithmetic width. An example register list is given below with the parameters used to generate the register lengths.

```
WORD ADDRESS SIZE     =        4
MEMORY ADDRESS SIZE   =       10
EXPONENT SIZE         =        5
MANTISSA SIZE         =        7
LEXIC LEVEL SIZE      =        4
DISPLACEMENT SIZE     =        5
STEP SIZE             =        6
NUMBER OF INTERRUPTS  =        6
NUMBER OF PORTS       =        4

          REGISTER LIST-
BI                    =       32
BO                    =       32
BO1                   =       32
BIB                   =        5
BIL                   =        5
BOB                   =        5
```

DSCAN	=	14
DSCAN1	=	14
ALPHA	=	14
DEFAULT	=	14
DSTOP	=	14
BETA	=	14
BETAPRIME	=	14
IOTA	=	14
IOTAPRIME	=	14
PI	=	14
PIPRIME	=	14
BOS	=	14
TOS	=	14
OP1	=	14
OP2	=	14
LK1	=	14
LK2	=	14
CFG	=	1
CFG2	=	1
CVRG	=	14
CVRG2	=	14
IOTRG	=	3
IOTRG2	=	3
LRG	=	7
LRG2	=	7
MPTR	=	14
MPTR2	=	14
SFG	=	1
SFG2	=	1
SRG	=	4
SRG2	=	4
TRG	=	4
TRG2	=	4
ERG	=	5
ERG2	=	5
PNRG	=	3
PNRG2	=	3
CURRENT'LL	=	4
LL	=	4
DISPLACEMENT	=	5
DIMREG	=	6
DIMREG1	=	6
LAMBDA	=	7
D'DECODER	=	5
MODE	=	3
FAULT	=	8
LENFLAG	=	1
METAFLAG	=	1
PATCHGENFLAG	=	1
SCALARFLAG	=	1
CF	=	1
OVF	=	1
ASF	=	5
AL	=	5

```
BL                      =       5
CL                      =       5
EA                      =       5
EB                      =       5
EC                      =       5
ER                      =       5
E1                      =       5
E2                      =       5
LA                      =       7
LB                      =       7
LC                      =       7
LD                      =       7
LR                      =       7
L1                      =       7
L2                      =       7
L3                      =       7
L4                      =       7
L5                      =       7
L6                      =       7
SA                      =       1
SB                      =       1
SC                      =       1
SR                      =       1
S1                      =       1
S2                      =       1
SEA                     =       1
SEB                     =       1
SEC                     =       1
SER                     =       1
SE1                     =       1
SE2                     =       1
VA                      =      14
VB                      =      14
VC                      =      14
VD                      =      14
VR                      =      14
V1                      =      14
V2                      =      14
V3                      =      14
V4                      =      14
V5                      =      14
V6                      =      14
INT'NO'REG              =       4
INT1'NO'REG             =       4
INHIBITFLAG             =       1
INTERRUPTREG            =       1
MASK                    =       6
IO'HELD'BARREL'FF       =       1
IO'SELECT'REGISTER      =       2
MAR                     =      10
NOWD                    =      10
NOWD1                   =      10
MIR                     =      16
LOC'                    =      14
```

```
MD                  =      14
MDL                 =      14
MS                  =      14
MS1                 =      14
MSL                 =      14
MSL1                =      14
PSCAN               =      14
PSTOP               =       1
PDS'A               =      17
PDS'B               =      17
PDS'A'FULL          =       1
PDS'CARRY           =       1
VT                  =       3
COND                =       1
COPYVALUEFLAG       =       1
TIME                =      36
LAST'IO'SCAN'TIME   =      36
P'SIZE              =      14
```

It should be noted that whenever a design parameter is changed, the register file becomes invalid. If the "LST" command is given with an invalid register file, only the design parameters will be listed, and the message "REGISTERS NOT VALID" will be generated.

If an "LRG" command is given and the registers are not valid, the message "REGISTERS ARE NOT VALID" will be generated.

In order to validate the register file, the "REG" command should be given. This command creates a new register file using the current design parameters.

An "END" command given with an invalid register file will cause the following message to be printed:

REGISTERS ARE NOT VALID; ENTER "REG", THEN "END"

This message is used to insure that the register file is valid before the initialization routine terminates.

If the "END" command is given with a valid register file, then the message "CREATE ITEM AND DEFINE FILES?" is printed. The reply "Y" will cause the initialization routine to generate these files. The ITEM and DEFINE files are used by the simulator and should be generated if the designer is satisfied with the register file.

In generating the ITEM file, the initialization routine converts the register file into the JOVIAL packed table structure. This ITEM file is used by the simulator as the declarations for the variables (simulated registers). The variables are declared with ITEM declarations, and these ITEMs are nested in packed tables to automatically accomplish zero fill and truncation (see Chapter V Section 2.3).

Each simulated register is represented by an ITEM in the packed table. In the ITEM file below, the first 4 ITEMs of the packed table "REGISTERS1" are BI, BO, BO1, and BIB. BI occupies the first 32 bits of the first word (word 0) of "REGISTERS". The fourth ITEM of the packed table is BIB, and it occupies the first 5 bits of the fourth word (word 3) of REGISTERS1. In the declaration "ITEM REG$BO U 32 [0, 1];", the number 32 represents the bit length of REG$BO (simulated register BO). The information inside the brackets gives the starting bit (bit 0) of the particular word of the table (1 means second word since 0 index). An example packed table is given in Figure 13. For more information on the JOVIAL packed table structure see Reference [7].

The follwowing ITEM file was generated from the previous register file.

```
    TABLE MEM[0:  1023] 1 U 16 [20,0];
    TABLE PDS[0:7] 1 U 17 [0,0];
    TABLE REGISTERS1 [0] 50; BEGIN
ITEM  REG$BI                    U 32 [0,   0];
ITEM  REG$BC                    U 32 [0,   1];
ITEM  REG$BO1                   U 32 [0,   2];
ITEM  REG$BIB                   U  5 [0,   3];
ITEM  REG$BIL                   U  5 [0,   4];
ITEM  REG$BOB                   U  5 [0,   5];
ITEM  REG$DSCAN                 U 14 [0,   6];
ITEM  REG$DSCAN1                U 14 [0,   7];
ITEM  REG$ALPHA                 U 14 [0,   8];
ITEM  REG$DEFAULT               U 14 [0,   9];
ITEM  REG$DSTOP                 U 14 [0,  10];
ITEM  REG$BETA                  U 14 [0,  11];
ITEM  REG$BETAPRIME             U 14 [0,  12];
ITEM  REG$IOTA                  U 14 [0,  13];
ITEM  REG$IOTAPRIME             U 14 [0,  14];
ITEM  REG$PI                    U 14 [0,  15];
ITEM  REG$PIPRIME               U 14 [0,  16];
ITEM  REG$BOS                   U 14 [0,  17];
ITEM  REG$TOS                   U 14 [0,  18];
ITEM  REG$OP1                   U 14 [0,  19];
ITEM  REG$OP2                   U 14 [0,  20];
ITEM  REG$LK1                   U 14 [0,  21];
ITEM  REG$LK2                   U 14 [0,  22];
ITEM  REG$CFG                   U  1 [0,  23];
ITEM  REG$CFG2                  U  1 [0,  24];
ITEM  REG$CVRG                  U 14 [0,  25];
ITEM  REG$CVRG2                 U 14 [0,  26];
ITEM  REG$IOTRG                 U  3 [0,  27];
ITEM  REG$IOTRG2                U  3 [0,  28];
ITEM  REG$LRG                   U  7 [0,  29];
ITEM  REG$LRG2                  U  7 [0,  30];
ITEM  REG$MPTR                  U 14 [0,  31];
ITEM  REG$MPTR2                 U 14 [0,  32];
ITEM  REG$SFG                   U  1 [0,  33];
ITEM  REG$SFG2                  U  1 [0,  34];
ITEM  REG$SRG                   U  4 [0,  35];
ITEM  REG$SRG2                  U  4 [0,  36];
ITEM  REG$TRG                   U  4 [0,  37];
ITEM  REG$TRG2                  U  4 [0,  38];
ITEM  REG$ERG                   U  5 [0,  39];
ITEM  REG$ERG2                  U  5 [0,  40];
ITEM  REG$PNRG                  U  3 [0,  41];
ITEM  REG$PNRG2                 U  3 [0,  42];
ITEM  REG$CURRENT'LL            U  4 [0,  43];
ITEM  REG$LL                    U  4 [0,  44];
ITEM  REG$DISPLACEMENT          U  5 [0,  45];
```

```
ITEM   REG$DIMREG                    U  6 [0,   46];
ITEM   REG$DIMREG1                   U  6 [0,   47];
ITEM   REG$LAMBDA                    U  7 [0,   48];
ITEM   REG$D'DECODER                 U  5 [0,   49];
                        END
       TABLE REGISTERS2 [0] 50; BEGIN
ITEM   REG$MODE                      U  3 [0,    0];
ITEM   REG$FAULT                     U  8 [0,    1];
ITEM   REG$LENFLAG                   U  1 [0,    2];
ITEM   REG$METAFLAG                  U  1 [0,    3];
ITEM   REG$PATCHGENFLAG              U  1 [0,    4];
ITEM   REG$SCALARFLAG                U  1 [0,    5];
ITEM   REG$CF                        U  1 [0,    6];
ITEM   REG$OVF                       U  1 [0,    7];
ITEM   REG$ASF                       U  5 [0,    8];
ITEM   REG$AL                        U  5 [0,    9];
ITEM   REG$BL                        U  5 [0,   10];
ITEM   REG$CL                        U  5 [0,   11];
ITEM   REG$EA                        U  5 [0,   12];
ITEM   REG$EB                        U  5 [0,   13];
ITEM   REG$EC                        U  5 [0,   14];
ITEM   REG$ER                        U  5 [0,   15];
ITEM   REG$E1                        U  5 [0,   16];
ITEM   REG$E2                        U  5 [0,   17];
ITEM   REG$LA                        U  7 [0,   18];
ITEM   REG$LB                        U  7 [0,   19];
ITEM   REG$LC                        U  7 [0,   20];
ITEM   REG$LD                        U  7 [0,   21];
ITEM   REG$LR                        U  7 [0,   22];
ITEM   REG$L1                        U  7 [0,   23];
ITEM   REG$L2                        U  7 [0,   24];
ITEM   REG$L3                        U  7 [0,   25];
ITEM   REG$L4                        U  7 [0,   26];
ITEM   REG$L5                        U  7 [0,   27];
ITEM   REG$L6                        U  7 [0,   28];
ITEM   REG$SA                        U  1 [0,   29];
ITEM   REG$SB                        U  1 [0,   30];
ITEM   REG$SC                        U  1 [0,   31];
ITEM   REG$SR                        U  1 [0,   32];
ITEM   REG$S1                        U  1 [0,   33];
ITEM   REG$S2                        U  1 [0,   34];
ITEM   REG$SEA                       U  1 [0,   35];
ITEM   REG$SEB                       U  1 [0,   36];
ITEM   REG$SEC                       U  1 [0,   37];
ITEM   REG$SER                       U  1 [0,   38];
ITEM   REG$SE1                       U  1 [0,   39];
ITEM   REG$SE2                       U  1 [0,   40];
ITEM   REG$VA                        U 14 [0,   41];
ITEM   REG$VB                        U 14 [0,   42];
ITEM   REG$VC                        U 14 [0,   43];
ITEM   REG$VD                        U 14 [0,   44];
ITEM   REG$VR                        U 14 [0,   45];
ITEM   REG$V1                        U 14 [0,   46];
ITEM   REG$V2                        U 14 [0,   47];
```

```
ITEM  REG$V3                        U 14 [0,  48];
ITEM  REG$V4                        U 14 [0,  49];
                        END
      TABLE REGISTERS3 [0] 50; BEGIN
ITEM  REG$V5                        U 14 [0,   0];
ITEM  REG$V6                        U 14 [0,   1];
ITEM  REG$INT'NO'REG                U  4 [0,   2];
ITEM  REG$INT1'NO'REG               U  4 [0,   3];
ITEM  REG$INHIBITFLAG               U  1 [0,   4];
ITEM  REG$INTERRUPTREG              U  1 [0,   5];
ITEM  REG$MASK                      U  6 [0,   6];
ITEM  REG$IO'HELD'BARREL'FF         U  1 [0,   7];
ITEM  REG$IO'SELECT'REGISTER        U  2 [0,   8];
ITEM  REG$MAR                       U 10 [0,   9];
ITEM  REG$NOWD                      U 10 [0,  10];
ITEM  REG$NOWD1                     U 10 [0,  11];
ITEM  REG$MIR                       U 16 [0,  12];
ITEM  REG$LOC'                      U 14 [0,  13];
ITEM  REG$MD                        U 14 [0,  14];
ITEM  REG$MDL                       U 14 [0,  15];
ITEM  REG$MS                        U 14 [0,  16];
ITEM  REG$MS1                       U 14 [0,  17];
ITEM  REG$MSL                       U 14 [0,  18];
ITEM  REG$MSL1                      U 14 [0,  19];
ITEM  REG$PSCAN                     U 14 [0,  20];
ITEM  REG$PSTOP                     U  1 [0,  21];
ITEM  REG$PDS'A                     U 17 [0,  22];
ITEM  REG$PDS'B                     U 17 [0,  23];
ITEM  REG$PDS'A'FULL                U  1 [0,  24];
ITEM  REG$PDS'CARRY                 U  1 [0,  25];
ITEM  REG$VT                        U  3 [0,  26];
ITEM  REG$COND                      U  1 [0,  27];
ITEM  REG$COPYVALUEFLAG             U  1 [0,  28];
ITEM  REG$TIME                      U 36 [0,  29];
ITEM  REG$LAST'IO'SCAN'TIME         U 36 [0,  30];
ITEM  REG$P'SIZE                    U 14 [0,  31];
                        END
  ITEM WORD'ADDRESS'SIZE     U =        4;
  ITEM MEMORY'SIZE           U =     1023;
  ITEM CANON'SIZE            U =       14;
  ITEM LK'SIZE               U =       14;
  ITEM EXP'SIZE              U =        6;
  ITEM NO'OF'PORTS           U =        4;
  ITEM COPYVALUECODE         U =        0;
            ITEM FRAMES U;
TABLE IO$PORT$REGISTERS [1:       4]; BEGIN
        ITEM IN'OUT'FLAG           U;
        ITEM PORT'WIDTH            U;
        ITEM BUFFER'BASE           U;
        ITEM BUFFER'LEN            U;
        ITEM FRAME'CNT             U;
        ITEM FRAME'BUFFER          U;
        ITEM ACTIVE'FF             U;
        ITEM REQUEST'FF            U;
```

```
        ITEM ACK'FF                U;
        ITEM FORCED'FF             U;
        ITEM ENABLE'FF             U;
        ITEM FRAME'PTR             U;
        ITEM NO'FRAMES             U;
                    END
    TABLE STREAM'ARRAY [       4,4] U;
     TABLE STREAM'TIMES [       4] U;
```

The DEFINE file is used to increase the readability of the simulator code. Since all of the simulated registers are table items, the statement "LAMBDA=ALPHA;" would generate two compiler warning messages (i.e. table items not subscripted). The warning messages are eliminated by the following DEFINE statements:

```
DEFINE LAMBDA "REG$LAMBDA[0]";
DEFINE ALPHA  "REG$ALPHA[0]";
```

These DEFINE statements will change the statement "LAMBDA=ALPHA;" to "REG$LAMBDA[0]=REG$ALPHA[0];" during compilation. Thus the DEFINE file is used to change all of the simulated registers to subscripted items and allows the code to maintain readability without excessive compiler messages. The following is the DEFINE file that is generated using the previous register file.

```
DEFINE BI          "REG$BI          [0]";
DEFINE BO          "REG$BO          [0]";
DEFINE BO1         "REG$BO1         [0]";
DEFINE BIB         "REG$BIB         [0]";
DEFINE BIL         "REG$BIL         [0]";
DEFINE BOB         "REG$BOB         [0]";
DEFINE DSCAN       "REG$DSCAN       [0]";
DEFINE DSCAN1      "REG$DSCAN1      [0]";
DEFINE ALPHA       "REG$ALPHA       [0]";
DEFINE DEFAULT     "REG$DEFAULT     [0]";
DEFINE DSTOP       "REG$DSTOP       [0]";
DEFINE BETA        "REG$BETA        [0]";
```

```
DEFINE BETAPRIME          "REG$BETAPRIME          [0]";
DEFINE IOTA               "REG$IOTA               [0]";
DEFINE IOTAPRIME          "REG$IOTAPRIME          [0]";
DEFINE PI                 "REG$PI                 [0]";
DEFINE PIPRIME            "REG$PIPRIME            [0]";
DEFINE BOS                "REG$BOS                [0]";
DEFINE TOS                "REG$TOS                [0]";
DEFINE OP1                "REG$OP1                [0]";
DEFINE OP2                "REG$OP2                [0]";
DEFINE LK1                "REG$LK1                [0]";
DEFINE LK2                "REG$LK2                [0]";
DEFINE CFG                "REG$CFG                [0]";
DEFINE CFG2               "REG$CFG2               [0]";
DEFINE CVRG               "REG$CVRG               [0]";
DEFINE CVRG2              "REG$CVRG2              [0]";
DEFINE IOTRG              "REG$IOTRG              [0]";
DEFINE IOTRG2             "REG$IOTRG2             [0]";
DEFINE LRG                "REG$LRG                [0]";
DEFINE LRG2               "REG$LRG2               [0]";
DEFINE MPTR               "REG$MPTR               [0]";
DEFINE MPTR2              "REG$MPTR2              [0]";
DEFINE SFG                "REG$SFG                [0]";
DEFINE SFG2               "REG$SFG2               [0]";
DEFINE SRG                "REG$SRG                [0]";
DEFINE SRG2               "REG$SRG2               [0]";
DEFINE TRG                "REG$TRG                [0]";
DEFINE TRG2               "REG$TRG2               [0]";
DEFINE ERG                "REG$ERG                [0]";
DEFINE ERG2               "REG$ERG2               [0]";
DEFINE PNRG               "REG$PNRG               [0]";
DEFINE PNRG2              "REG$PNRG2              [0]";
DEFINE CURRENT'LL         "REG$CURRENT'LL         [0]";
DEFINE LL                 "REG$LL                 [0]";
DEFINE DISPLACEMENT       "REG$DISPLACEMENT       [0]";
DEFINE DIMREG             "REG$DIMREG             [0]";
DEFINE DIMREG1            "REG$DIMREG1            [0]";
DEFINE LAMBDA             "REG$LAMBDA             [0]";
DEFINE D'DECODER          "REG$D'DECODER          [0]";
DEFINE MODE               "REG$MODE               [0]";
DEFINE FAULT              "REG$FAULT              [0]";
DEFINE LENFLAG            "REG$LENFLAG            [0]";
DEFINE METAFLAG           "REG$METAFLAG           [0]";
DEFINE PATCHGENFLAG       "REG$PATCHGENFLAG       [0]";
DEFINE SCALARFLAG         "REG$SCALARFLAG         [0]";
DEFINE CF                 "REG$CF                 [0]";
DEFINE OVF                "REG$OVF                [0]";
DEFINE ASF                "REG$ASF                [0]";
DEFINE AL                 "REG$AL                 [0]";
DEFINE BL                 "REG$BL                 [0]";
DEFINE CL                 "REG$CL                 [0]";
DEFINE EA                 "REG$EA                 [0]";
DEFINE EB                 "REG$EB                 [0]";
DEFINE EC                 "REG$EC                 [0]";
DEFINE ER                 "REG$ER                 [0]";
```

```
DEFINE E1                    "REG$E1                    [0]";
DEFINE E2                    "REG$E2                    [0]";
DEFINE LA                    "REG$LA                    [0]";
DEFINE LB                    "REG$LB                    [0]";
DEFINE LC                    "REG$LC                    [0]";
DEFINE LD                    "REG$LD                    [0]";
DEFINE LR                    "REG$LR                    [0]";
DEFINE L1                    "REG$L1                    [0]";
DEFINE L2                    "REG$L2                    [0]";
DEFINE L3                    "REG$L3                    [0]";
DEFINE L4                    "REG$L4                    [0]";
DEFINE L5                    "REG$L5                    [0]";
DEFINE L6                    "REG$L6                    [0]";
DEFINE SA                    "REG$SA                    [0]";
DEFINE SB                    "REG$SB                    [0]";
DEFINE SC                    "REG$SC                    [0]";
DEFINE SR                    "REG$SR                    [0]";
DEFINE S1                    "REG$S1                    [0]";
DEFINE S2                    "REG$S2                    [0]";
DEFINE SEA                   "REG$SEA                   [0]";
DEFINE SEB                   "REG$SEB                   [0]";
DEFINE SEC                   "REG$SEC                   [0]";
DEFINE SER                   "REG$SER                   [0]";
DEFINE SE1                   "REG$SE1                   [0]";
DEFINE SE2                   "REG$SE2                   [0]";
DEFINE VA                    "REG$VA                    [0]";
DEFINE VB                    "REG$VB                    [0]";
DEFINE VC                    "REG$VC                    [0]";
DEFINE VD                    "REG$VD                    [0]";
DEFINE VR                    "REG$VR                    [0]";
DEFINE V1                    "REG$V1                    [0]";
DEFINE V2                    "REG$V2                    [0]";
DEFINE V3                    "REG$V3                    [0]";
DEFINE V4                    "REG$V4                    [0]";
DEFINE V5                    "REG$V5                    [0]";
DEFINE V6                    "REG$V6                    [0]";
DEFINE INT'NO'REG            "REG$INT'NO'REG            [0]";
DEFINE INT1'NO'REG           "REG$INT1'NO'REG           [0]";
DEFINE INHIBITFLAG           "REG$INHIBITFLAG           [0]";
DEFINE INTERRUPTREG          "REG$INTERRUPTREG          [0]";
DEFINE MASK                  "REG$MASK                  [0]";
DEFINE IO'HELD'BARREL'FF     "REG$IO'HELD'BARREL'FF     [0]";
DEFINE IO'SELECT'REGISTER    "REG$IO'SELECT'REGISTER    [0]";
DEFINE MAR                   "REG$MAR                   [0]";
DEFINE NOWD                  "REG$NOWD                  [0]";
DEFINE NOWD1                 "REG$NOWD1                 [0]";
DEFINE MIR                   "REG$MIR                   [0]";
DEFINE LOC'                  "REG$LOC'                  [0]";
DEFINE MD                    "REG$MD                    [0]";
DEFINE MDL                   "REG$MDL                   [0]";
DEFINE MS                    "REG$MS                    [0]";
DEFINE MS1                   "REG$MS1                   [0]";
DEFINE MSL                   "REG$MSL                   [0]";
DEFINE MSL1                  "REG$MSL1                  [0]";
```

```
DEFINE PSCAN                "REG$PSCAN                [0]";
DEFINE PSTOP                "REG$PSTOP                [0]";
DEFINE PDS'A                "REG$PDS'A                [0]";
DEFINE PDS'B                "REG$PDS'B                [0]";
DEFINE PDS'A'FULL           "REG$PDS'A'FULL           [0]";
DEFINE PDS'CARRY            "REG$PDS'CARRY            [0]";
DEFINE VT                   "REG$VT                   [0]";
DEFINE COND                 "REG$COND                 [0]";
DEFINE COPYVALUEFLAG        "REG$COPYVALUEFLAG        [0]";
DEFINE TIME                 "REG$TIME                 [0]";
DEFINE LAST'IO'SCAN'TIME    "REG$LAST'IO'SCAN'TIME    [0]";
DEFINE P'SIZE               "REG$P'SIZE               [0]";
```

A.3 Simulator

The simulator uses two files that are generated from the initialization routine (ITEM and DEFINE files). These files are copied into the simulator code during compilation, thus the simulator must be recompiled if any changes are made to these files.

The Program String to be executed is read from the file "PROGRM.J73". The S-UNITS are left justified in the file, and the constants are right justified with column 25. The Program String is read into the simulated memory using the FORTRAN "READER" subroutine, and this subroutine can be modified if the Program String format needs to be changed. The example Program String shown below was generated by translating the simple example program in Figure 2.

```
PROG
SET'DEFAULT
                        4
NEW'INTEGER
CARD'CONST
                        3
                        5
DYNA'LEN
DUMP
NEW'INTEGER
```

```
CARD'CONST
                    2
                    2
DYNA'DIM
CARD'CONST
                    2
                    3
DYNA'DIM
END'DECL
DUMP
CARD'CONST
                    2
                    3
ID
                    0
                    0
DUMP
ASSIGN
CLEAN'UP
ID
                    0
                    0
CARD'CONST
                    3
                    4
S''ADD
START'LIST
CARD'CONST
                    1
                    1
CANONICAL
CARD'CONST
                    2
                    2
SUBSCRIPT'MODE
DUMP
ID
                    0
                    1
ASSIGN
TERMINATE
```

A directory of S-UNITS is contained in the "DIR.J73" file, and this file is used by the FORTRAN "READER" subroutine when checking the Program String file for valid S-UNITS. The directory file contains an entry for each S-UNIT and each entry is left justified. The following is the directory file of S-UNITS.

```
S''ABS
ACTIVE
S''AND
APPLY'SUBS
ARRAY'CONST
CLEAN'UP
COMMENSURATE
S''CONC
S''CROSS'PRODUCT
S''DECODE
S''DECREMENT
DIMENSION
S''DIVIDE
DO'UNTIL
DYADIC'EXIT
DYNA'DIM
DYNA'TYPE
ELSE'
ENABLE'ALL
ENABLE'FORMULA
S''ENCODE
END'DECL
END'REF
ENTER'SCOPE
S''EQUAL
S''ERROR
S''EXCHANGE
EXIT'
FALSE
FAULT'OP
FORMAL'ID
GIVEN'UNTIL
S''GREATER
S''GREATER'EQ
S''INCREMENT
S''INNER'PROD
S''LESS
S''LESS'EQUAL
S''MAGNITUDE
S''MAX
S''MIN
S''MODULO
S''MULTIPLY
S''NEGATE
S''NOT
S''NOT'EQUAL
S''OR
S''ROUND
S''R'ADD
S''R'AND
S''R'MAX
S''R'MIN
S''R'MULT
S''R'NOT
```

```
S''R'OR
S''SHIFT
S''TRANSPOSE
S''TRUNCATE
S''TYPE
INHIBIT'ALL
INHIBIT'FORMULA
INTEGER'CONST
LEAVE'SCOPE
LENGTH
MIX'TYPE
MONADIC'EXIT
NEGATE
NENT
NEW'BOOLEAN
NEW'CARDINAL
NEW'COLON
NEW'INTEGER
NEW'INTERRUPT
NEW'LABEL
NEW'NULL
NEW'OPERATOR
NEW'PROC
NEW'REAL
NEW'SET
OUTPUT'PORT
RANK
RAVEL
REAL'CONST
SKIP'ARRAY
SKIP'CONST
START'LIST
SUBSCRIPT'MODE
TERMINATE
THEN'
TRUE
WHOOPS
PROG
CANONICAL
ASSIGN
COPY'VALUE
S''ADD
ALLOCATE
CARD'CONST
DUPLICATE'TOP
DYNA'LEN
END'ARRAY'CONST
FRAME'COUNT
ID
INDEX
INIT'PORT
INPUT'PORT
S''POWER
SET'DEFAULT
```

```
S''SUBTRACT
WAIT
DUMP
```

The simulator also reads two other files: "FRMTM.J73" and "FRAMES.J73". The elements of the "FRMTM.J73" file are read into the STREAM'TIMES array, and the elements of the "FRAMES.J73" file are read into STREAM'ARRAY. If there are n simulated ports, then the "FRMTM.J73" file should have n elements, and the "FRAMES.J73" file should have n*(column dimension of STREAM'ARRAY) elements. For example, if there are 5 simulated ports and STREAM'ARRAY is a 5x10 array, then "FRMTM.J73" should have 5 elements and "FRAMES.J73" should have 50 elements. Each element of the "FRMTM.J73" file should be a positive integer, and the elements of the "FRAMES.J73" file can be any integer. Thus these files are used to set up the I/O arrays and are read before the simulator begins executing the Program String.

While running the simulator, any dumps that are generated are written to the file "MEM.J73". The dump routine creates these dumps by parsing the simulated memory using a link list. The link list is contained on the "MATCH.J73" file, and the "GETLIST" FORTRAN subroutine is used to read this file. These dumps are powerful debugging aids, and the dump format is shown in the examples in Appendix D.

As shown in the example Program String above, the

DUMP S-UNIT is inserted into the Program String after any other S-UNIT. Execution of the DUMP S-UNIT parses the simulated memory, and during this parse the "parsed" memory is written to the "MEM;J73" file. The Program String is also written during execution of the first dump.

Before running the simulator, the simulator routines must be compiled and loaded with the relocatable files of the dump and FORTRAN routines (these FORTRAN routines are used in file manipulations). When the executable code is obtained, the simulator can be run if the necessary files have been created. Figure 19 shows the interaction of the simulator with the necessary files, and Figure 20 summarizes the steps in running the simulator.

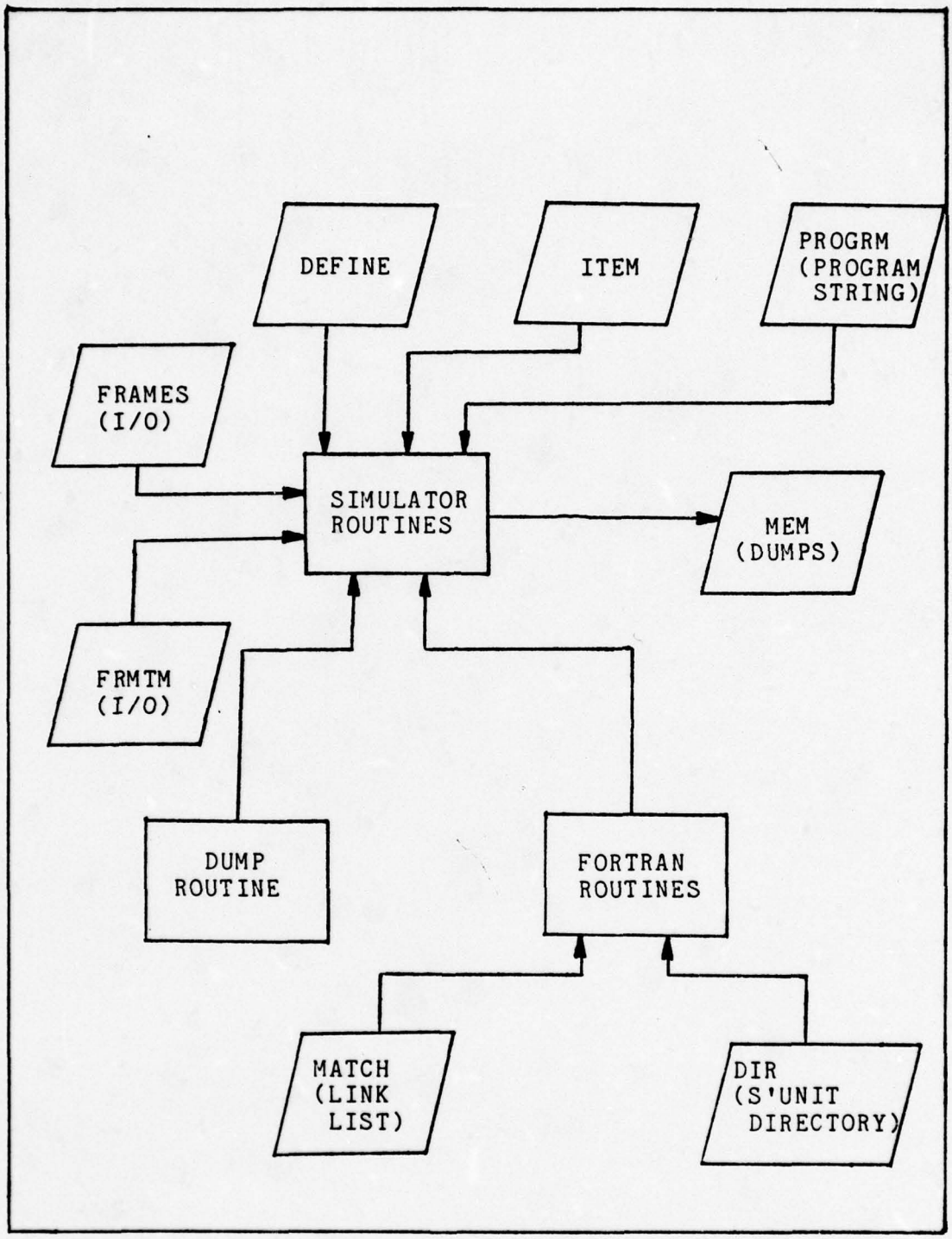


Figure 19. Simulator and Files

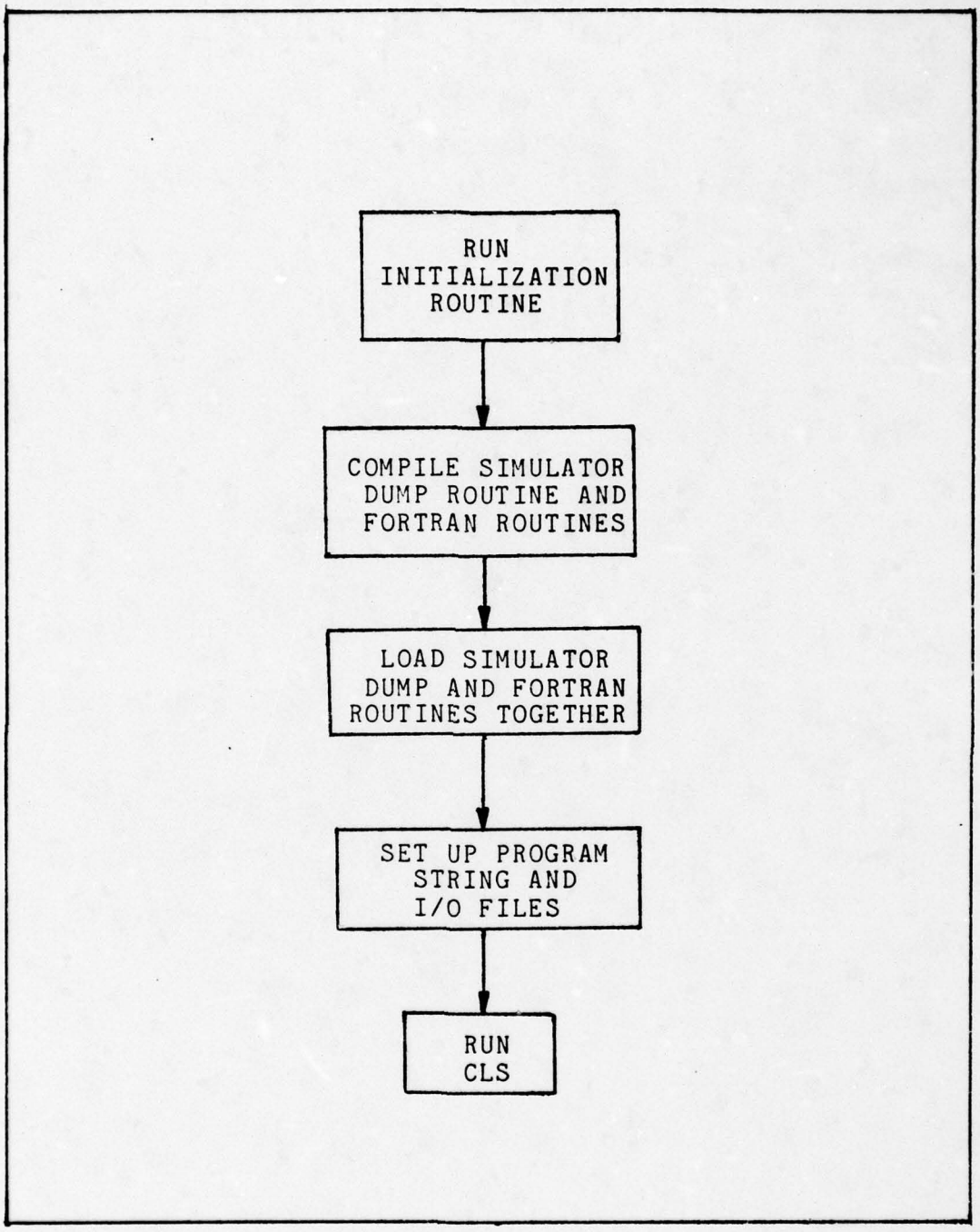


Figure 20. Steps in Running the Simulator

A.4 Running the Simulator on the DEC-10

In order to run the simulator, the simulator must be compiled and loaded with the DUMP and FORTRAN routines.

Compilation of the simulator is accomplished with the following commands.

```
R SYS:J73
CLS,TTY:=CLS.J73/NOS/NOP/NOI/IND
```

"CLS.J73" is the file that the simulator is on and the information following the "/" are switches (see Reference [7] Appendix C). The "TTY" in the command string causes the compiler messages to be sent to the terminal. The file "CLS.REL" contains the relocatable code generated by the compilation.

Since the DUMP routine is also a JOVIAL routine, then the following commands are given to obtain the DUMP relocatable code (DUMP.REL).

```
R SYS:J73
DUMP,TTY:=DUMP.J73/NOB/NOP/NOI/IND
```

In order to obtain the relocatable code for the FORTRAN routines (READ.REL), the following commands must be given.

```
R FORTRA
READ,TTY:=READ.J73
```

After these relocatable files have been obtained, then the following commands are given.

```
LOAD CLS,DUMP,READ,SYS:JLIB/LIBRARY
SAV
```

The executable code will then be on the file "CLS.EXE". The following command is then given to run the simulator.

```
RUN CLS
```

Appendix B

Deviations From Original Study

B.1 Introduction

The following paragraphs will describe various aspects of the simulator that were changed in this study. These changes were made to improve the performance of the simulator.

B.2 Registers and Main Memory

The registers and main memory were simulated using the table structure of JOVIAL (see Chapter V Section 2.3). As is shown in Figure 13, the most significant bit of the simulated register and the DEC-10 memory word correspond.

In the original study, each bit of a register and memory word was simulated in APL using a complete memory word. In this study each register and memory word is simulated using a word of DEC-10 memory and thus the memory requirements of the simulator have been reduced. However, a disadvantage is that the maximum register size and the maximum memory word size of the simulator are now restricted to the word length of the host machine (e.g. 36 bits for DEC-10). Thus the register/memory word simulation scheme will have to be changed if a register/memory word size must be greater than the word length of the host machine.

In the original study, the APL operators were used to simulate the low level machine operations. Since each bit of a register was simulated by an entire APL word, the bits of the simulated registers had to be transformed into a vector before an operation could be performed.

As a result of this study, however, the JOVIAL operators simulate the low level machine operations, and these operators are used between words of memory. This saves execution time due to the fact that the simulated registers do not have to be collected into a vector before an operation is performed. This eliminates the need for the bit manipulations used in the original study. For example, a simulated register transfer is accomplished by the JOVIAL assignment operator (=) between elements of packed tables (see Chapter V Section 2.3).

B.3 Barrel Shifter

The original study used bit zero to point to the least significant bit of the barrel shifter registers. Since the JOVIAL (J73) table specification uses bit zero as the most significant bit, changes have been made to the following routines:

BARREL'SHIFT
CLEAR'BARREL
LOAD'BARREL
LOAD'BARREL'LOOP
PRESET'BARREL
WRITE'BARREL
WRITE'BARREL'LOOP

These changes, however do not make the barrel shifter routines system dependent (e.g. DEC-10) since the table structure used is a JOVIAL specification.

Two of these routines, LOAD'BARREL and WRITE'BARREL have been further changed due to unnecessary steps found in their algorithms. Step 6 and 7 of both algorithms in Figure 5 (LOAD'BARREL) and Figure 6 (WRITE'BARREL) of Reference[4] have been eliminated. Figure 7 and Figure 9 of Chapter III gives a detailed look at the present algorithms used.

Appendix C

Register Descriptions

Pushdown Stack Registers (refer to Chapter III Section 3)

PDS'A	Input register to the pushdown stack through a V'PUSH operation and output register to the pushdown stack through a V'POP operation.
PDS'B	Used in conjunction with PDS'A during arithmetic operations.
PDS	FIFO pushdown stack.
PDS'A'FULL	Flag indicating PDS'A full if equal to 1 or empty if equal to 0.
PDS'CARRY	1 bit register holding carry from arithmetic operations.
VT	Pointer to first element in pushdown stack.
OVF	Overflow flag for stack addition.
COND	Flag checked during logical operations.

Main Memory Registers

MAR	Memory address register - word address used in accessing memory.
MIR	Memory information register - word sized register holding data read from or written to memory.
MS	Used to hold a bit address of memory.
MS1	MS backup during I/O operations.
MSL	Used to hold size of bit field to be moved to or from memory.
MSL1	MSL backup during I/O operations.

MD	Used to hold a bit address of memory.
MDL	Used to hold size of bit field to be moved to or from memory.
LOC'	Used to hold a bit address of memory.
CF	Flag indicating when a complement operation on a main memory operand remains to be performed.
ASF	Holds shift amount during floating-point mantissa add.

Program String Registers

DECODER	Holds current S-UNIT being fetched or executed within the Program String.
PSCAN	Holds bit address of S-UNIT being fetched or executed within the Program String.
P'SIZE	Holds the number of bits to be taken from the Program String starting at the address in PSCAN.
PSTOP	Flag indicating program termination (normal or error).

Workspace Registers

BOS	"Bottom of Stack" holds the address of first bit in workspace after Program String.
TOS	"Top of Stack" holds the address of the first bit of the last link in the workspace.
OP1	"Operand 1" holds address of first bit of last link in the workspace.
OP2	"Operand 2" holds address of first bit of second to last link in the workspace.
LK1 and LK2	Hold addresses of first bits of two consecutive links. Used to move up and down workspace.

Descriptor Parameter Registers

DEFAULT	Holds program default bit length.
CURRENT'LL	Holds the current lexic level of the S-UNIT presently being fetched or executed.
LL	Holds lexic level being searched for or desired (e.g. ID operations).
DISPLACEMENT	Holds displacement being searched for or desired (e.g. ID operations).
DIMREG/DIMREG1	Dimension registers holding the cumulative product of subscripts which when multiplied by LAMBDA equal the number of bits to be allocated for a valuespace.
CVRG/CVRG2	Hold canonical value of a subscript.
PORT'NUMBER	Holds current port number during I/O operations.

Variable Attribute Registers

AL	Holds number of bits remaining in the current chunk of the "A" operand.
BL	Holds number of bits remaining in the current chunk of the "B" operand.
CL	Holds number of bits remaining in the current chunk of the "C" operand.
EA	Holds exponent of "A" operand.
EB	Holds exponent of "B" operand.
EC	Holds exponent of "C" operand.
ER	Holds exponent of "Result" operand.
E1	Scratch exponent register.
E2	Scratch exponent register.
LA	Holds the total length of operand "A" (including all chunks).

LB	Holds the total length of operand "B" (including all chunks).
LC	Holds the total length of operand "C" (including all chunks).
LD	Holds the total length of operand "D" (including all chunks).
LR	Holds the total length of operand "Result" (including all chunks).
L1	Scratch exponent register.
L2	Scratch exponent register.
L3	Scratch exponent register.
L4	Scratch exponent register.
L5	Scratch exponent register.
L6	Scratch exponent register.
SA	Holds sign of "A" stack operand.
SB	Holds sign of "B" stack operand.
SC	Holds sign of "C" stack operand.
SR	Holds sign of "Result" stack operand.
S1	Scratch sign register.
S2	Scratch sign register.
SEA	Holds sign of exponent of "A" stack operand.
SEB	Holds sign of exponent of "B" stack operand.
SEC	Holds sign of exponent of "C" stack operand.
SER	Holds sign of exponent of "Result" stack operand.
SE1	Scratch exponent sign register.
SE2	Scratch exponent sign register.
VA	Pointer to "A" operand valuespace.

VB	Pointer to "B" operand valuespace.
VC	Pointer to "C" operand valuespace.
VD	Pointer to "D" operand valuespace.
VR	Pointer to "Result" operand valuespace.
V1	Scratch valuespace pointer.
V2	Scratch valuespace pointer.
V3	Scratch valuespace pointer.
V4	Scratch valuespace pointer.
V5	Scratch valuespace pointer.
V6	Scratch valuespace pointer.

Registers Holding Attributes of an Operand Descriptor

CFG/CFG2	Flag indicating constant encountered during a descriptor scan of a variable.
LRG/LRG2	Holds length value from descriptor operand of a variable.
MPTR/MPTR2	Holds mantissa from descriptor operand of a variable.
SFG/SFG2	Holds sign from descriptor operand of a variable.
SRG/SRG2	Holds syllables from descriptor operand of a variable.
ERG/ERG2	Holds exponent from descriptor operand of a variable.
TRG/TRG2	Holds type of the variable from descriptor operand of a variable.
IOTRG/IOTRG2	Holds I/O type (i.e. input or output) from descriptor operand of a variable.
PNRG/PNRG2	Holds I/O port number from descriptor operand of a variable.

Barrel Shifter Registers (see Chapter III Section 5)

BI	Barrel Input - Input register which holds source bit field to be shifted into BO.
BO	Barrel Output - Output register which holds destination field of bit field to be shifted from BI.
BO1	Backup for BO during I/O operations.
BIL	Barrel Input Length - Holds size of bit field to be shifted.
BOB	Holds starting bit of bit field in BO to be shifted.
BIB	Holds starting bit of bit field in BI to be shifted.

Descriptor Scanning Registers (see Reference[2] pages 5-2-17 to 5-2-19)

DSCAN/DSCAN1	Address registers used in descriptor scanning.
DSTOP	Stops descriptor scan when set to 1.
D'DECODER	Same as DECODER but used in descriptor scanning.
MODE	Holds the mode in which the descriptor is scanned.
LENFLAG	Used to signal that the elements of a raveled array are of the default length.
PATCHENFLAG	Used in raveling a referenced variable so that a simpler addressing scheme is used if the referenced array is contiguous.
SCALARFLAG	Inhibits the generation of dimension D-UNITS for a scalar.
ALPHA	Holds addresses.
BETA/BETAPRIME	Boundcheck registers for subscript values.

IOTA/IOTAPRIME Holds subscript canonical values.

PI/PIPRIME Holds intermediate products of the subscripts and their multipliers.

LAMBDA Holds bit length of elements of arrays.

I/O and Interrupt Registers

INT'NO'REG Holds number of interrupt to be processed.

INT'NO'REG1 Backup for INT'NO'REG.

INHIBITFLAG Inhibits all interrupts until an ENABLE S-UNIT is executed.

MASK Boolean vector which has a bit for every port and when a bit is set the corresponding port is masked.

IO'HELD'BARREL'FF When this flag is clear the I/O section is scanned before a LOAD'BARREL operation.

IO'SELECT'REGISTER Holds the number of the port during an I/O operation.

TIME Holds execution time.

LAST'IO'SCAN'TIME Holds the time that the I/O section was last scanned.

COPYVALUEFLAG Flag used in overlapping I/O operations with processing.

Refer to Reference[4] pp. 31-34 for Following I/O Registers

IN'OUT'FLAG

PORT'WIDTH

BUFFER'BASE

BUFFER'LEN

FRAME'CNT

FRAME'BUFFER

ACTIVE'FF

REQUEST'FF

ACK'FF

FORCED'FF

ENABLE'FF

Soft Extension Registers

FAULT	Holds the S-UNIT of the "Faulty Operator" during a soft extension.
METAFLAG	Flag used to inhibit the fetching of a new S-UNIT and is set during execution of the FAULT-OP S-UNIT.

Appendix D

Example Executions

D.4 Introduction

The following examples illustrate program execution on the simulator. These examples exercise many S-UNITS and should be studied carefully in the initial phase of a follow-on study.

The first example is basically the example program in Volume 1 of the Phase I study (see Reference [2] Appendix F). The second example shows the execution of a function (i.e. a procedure which returns a value), the third example contains a procedure with parameters, and the fourth example execution deals with a monadic soft extension.

D.5 Reference Example

```
PROGRAM
DEFAULT 32;
INTEGER A[3,4,5] BITS 4;
REF B TO A[,2:3,];
REF C TO B[2];
REF D TO C[1,2:4];
REF RA TO RAVEL A;
REF RB TO RAVEL B;
REF SV TO RB[2:3];
REF X TO SV[1];
11->SV[3]-20->X
TERM
```

Dump 1. The Program String

```
00000    0 0011101000000000 PROG
00020    1 0011011000000000 SET'DEFAULT
00040    2 0000010000000000                    32
```

Comment- The following section corresponds to

INTEGER A[3,4,5] BITS 4;

```
00060    3 0001001000000000 NEW'INTEGER
00100    4 0100011000000000 CARD'CONST
00120    5 0100000000000000                     2
00140    6 1100000000000000                     3
00160    7 0000100000000000 DYNA'DIM
00200    8 0100011000000000 CARD'CONST
00220    9 1100000000000000                     3
00240   10 0010000000000000                     4
00260   11 0000100000000000 DYNA'DIM
00300   12 0100011000000000 CARD'CONST
00320   13 0010000000000000                     4
00340   14 1010000000000000                     5
00360   15 0000100000000000 DYNA'DIM
00400   16 0100011000000000 CARD'CONST
00420   17 1100000000000000                     3
00440   18 0010000000000000                     4
00460   19 0010011000000000 DYNA'LEN
00500   20 1000011000000000 ALLOCATE
00520   21 1111011000000000 DUMP               2
```

Comment- The following section corresponds to

REF B TO A[,2:3,];

```
00540   22 0110101000000000 START'LIST
00560   23 1101001000000000 NEW'NULL
00600   24 1011101000000000 CANONICAL
00620   25 1110001000000000 NEW'COLON
00640   26 0100011000000000 CARD'CONST
00660   27 0100000000000000                     2
00700   28 0100000000000000                     2
00720   29 1011101000000000 CANONICAL
00740   30 0100011000000000 CARD'CONST
00760   31 0100000000000000                     2
01000   32 1100000000000000                     3
01020   33 1011101000000000 CANONICAL
01040   34 1101001000000000 NEW'NULL
01060   35 1110101000000000 SUBSCRIPT'MODE
01100   36 1111011000000000 DUMP               3
01120   37 1110011000000000 ID
01140   38 0000000000000000                     0
```

```
01160   39 0000000000000000                        0
01200   40 1000011000000000 ALLOCATE
01220   41 1111011000000000 DUMP                   4
```

Comment- The following section corresponds to

REF C TO B[2];

```
01240   42 0110101000000000 START'LIST
01260   43 0100011000000000 CARD'CONST
01300   44 0100000000000000                        2
01320   45 0100000000000000                        2
01340   46 1110101000000000 SUBSCRIPT'MODE
01360   47 1110011000000000 ID
01400   48 0000000000000000                        0
01420   49 1000000000000000                        1
01440   50 1000011000000000 ALLOCATE
```

Comment- The following section corresponds to

REF D TO C[1,2:4];

```
01460   51 0110101000000000 START'LIST
01500   52 0100011000000000 CARD'CONST
01520   53 1000000000000000                        1
01540   54 1000000000000000                        1
01560   55 1011101000000000 CANONICAL
01600   56 1110001000000000 NEW'COLON
01620   57 0100011000000000 CARD'CONST
01640   58 0100000000000000                        2
01660   59 0100000000000000                        2
01700   60 1011101000000000 CANONICAL
01720   61 0100011000000000 CARD'CONST
01740   62 1100000000000000                        3
01760   63 0010000000000000                        4
02000   64 1110101000000000 SUBSCRIPT'MODE
02020   65 1110011000000000 ID
02040   66 0000000000000000                        0
02060   67 0100000000000000                        2
02100   68 1000011000000000 ALLOCATE
```

Comment- The following section corresponds to

REF RA TO RAVEL A;

```
02120   69 1110011000000000 ID
02140   70 0000000000000000                        0
02160   71 0000000000000000                        0
02200   72 1111011000000000 DUMP                   5
02220   73 0100101000000000 RAVEL
02240   74 1000011000000000 ALLOCATE
```

```
02260   75 1111011000000000 DUMP                6

        Comment- The following section corresponds to

                        REF RB TO RAVEL B;

02300   76 1110011000000000 ID
02320   77 0000000000000000                      0
02340   78 1000000000000000                      1
02360   79 1111011000000000 DUMP                 7
02400   80 0100101000000000 RAVEL
02420   81 1111011000000000 DUMP                 8
02440   82 1000011000000000 ALLOCATE

        Comment- The following section corresponds to

                        REF SV TO RB[2:3];

02460   83 0110101000000000 START'LIST
02500   84 1110001000000000 NEW'COLON
02520   85 0100011000000000 CARD'CONST
02540   86 0100000000000000                      2
02560   87 1100000000000000                      3
02600   88 1011101000000000 CANONICAL
02620   89 0100011000000000 CARD'CONST
02640   90 1100000000000000                      3
02660   91 1110000000000000                      7
02700   92 1110101000000000 SUBSCRIPT'MODE
02720   93 1110011000000000 ID
02740   94 0000000000000000                      0
02760   95 1010000000000000                      5
03000   96 1000011000000000 ALLOCATE

        Comment- The following section corresponds to

                        REF X TO SV[1];

03020   97 0110101000000000 START'LIST
03040   98 0100011000000000 CARD'CONST
03060   99 1000000000000000                      1
03100  100 1000000000000000                      1
03120  101 1110101000000000 SUBSCRIPT'MODE
03140  102 1110011000000000 ID
03160  103 0000000000000000                      0
03200  104 0110000000000000                      6
03220  105 0110100000000000 END'DECL
```

Comment- The following section corresponds to

R→SV[3]-20→X

```
03240  106 1111011000000000 DUMP                9
03260  107 0100011000000000 CARD'CONST
03300  108 0010000000000000                      4
03320  109 1101000000000000                     11
03340  110 0110101000000000 START'LIST
03360  111 0100011000000000 CARD'CONST
03400  112 0100000000000000                      2
03420  113 1100000000000000                      3
03440  114 1110101000000000 SUBSCRIPT'MODE
03460  115 1110011000000000 ID
03500  116 0000000000000000                      0
03520  117 0110000000000000                      6
03540  118 1111011000000000 DUMP                10
03560  119 0111101000000000 ASSIGN
03600  120 0100011000000000 CARD'CONST
03620  121 1010000000000000                      5
03640  122 0010100000000000                     20
03660  123 1111011000000000 DUMP                11
03700  124 1011011000000000 S''SUBTRACT
03720  125 1110011000000000 ID
03740  126 0000000000000000                      0
03760  127 1110000000000000                      7
04000  128 1111011000000000 DUMP                12
04020  129 0111101000000000 ASSIGN
04040  130 1111011000000000 DUMP                13
04060  131 0001101000000000 TERMINATE
```

Dump 2. INTEGER[3,4,5] BITS 4;

```
04100  132 000000000000000   link
04116  132                00 link
       133 111000000000
04134  133              1110 link
       134 1100000000
04152  134             10000 BEGIN       (mark constant)
04157  134                 0 Lexic Level (parameter)
       135 000
04163  135    000000         Displacement (parameter)
04171  135           1010111 link
       136 1000000
04207  136         0010       INTEGER     (type constant)
04213  136             1100   STEP        (syll constant)
04217  136                1   Multiplier  (parameter)
       137 10000
04225  137      1100          STEP        (syll constant)
```

```
04231  137          001100  Multiplier  (parameter)
04237  137               1  STEP        (syll constant)
       138 100
04243  138    001111        Multiplier  (parameter)
04251  138          0010    LEN         (syll constant)
04255  138             001  Bit Length  (parameter)
       139 0000
04264  139     1010         VALUE       (syll constant)
04270  139         00000000 Valuespace
       140 0000000000000000
       141 0000000000000000
       142 0000000000000000
       143 0000000000000000
       144 0000000000000000
       145 0000000000000000
       146 0000000000000000
       147 0000000000000000
       148 0000000000000000
       149 0000000000000000
       150 0000000000000000
       151 0000000000000000
       152 0000000000000000
       153 0000000000000000
       154 0000000000000000
       155 0000000000000000
       156 0000000000000000
       157 0000000000000000
       158 0000
```

Dump 3. Dump of [,2:3,] in REF B TO A[,2:3,];

```
04100  132 00000000000000   link
04116  132              00  link
       133 111000000000
04134  133            1110  link
       134 1100000000
04152  134           10000  BEGIN       (mark constant)
04157  134               0  Lexic Level (parameter)
       135 000
04163  135    000000        Displacement (parameter)
04171  135         0001110  link
       136 1100000
04207  136        0010       INTEGER     (type constant)
04213  136           1100    STEP        (syll constant)
04217  136              1    Multiplier  (parameter)
       137 10000
04225  137      1100         STEP        (syll constant)
04231  137         001100    Multiplier  (parameter)
04237  137              1    STEP        (syll constant)
```

```
        138 100
04243   138    001111          Multiplier  (parameter)
04251   138          0010      LEN         (syll constant)
04255   138               001  Bit Length  (parameter)
        139 0000
04264   139     1010           VALUE       (syll constant)
04270   139          00000000  Valuespace
        140 0000000000000000
        141 0000000000000000
        142 0000000000000000
        143 0000000000000000
        144 0000000000000000
        145 0000000000000000
        146 0000000000000000
        147 0000000000000000
        148 0000000000000000
        149 0000000000000000
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
        154 0000000000000000
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
04744   158     011100011000  link
        159 00
04762   159   10100           LIST        (mark constant)
04767   159         101101111 link
        160 10000
05005   160       0001        NULL        (type constant)
05011   160           1101011 link
        161 1110000
05027   161         11010     COLON       (mark constant)
05034   161              0010 link
        162 1100000000
05052   162           11011   CANONICAL   (mark constant)
05057   162                0  Canonical Value (parameter)
        163 1000000000000
05075   163               010 link
        164 00010000000
05113   164            11011  CANONICAL   (mark constant)
05120   165 11000000000000    Canonical Value (parameter)
05136   165                10 link
        166 110010000000
05154   166              0001 NULL        (type constant)
05160   167 00000000000000    link
05176   167                11 END         (mark constant)
        168 100
```

Dump 4. REF B TO A[,2:3,];

```
04100  132 00000000000000   link
04116  132               00 link
       133 111000000000
04134  133             1110 link
       134 1100000000
04152  134            10000 BEGIN          (mark constant)
04157  134                0 Lexic Level    (parameter)
       135 000
04163  135   000000          Displacement   (parameter)
04171  135          0001110 link
       136 1100000
04207  136        0010        INTEGER      (type constant)
04213  136           1100     STEP         (syll constant)
04217  136              1     Multiplier   (parameter)
       137 10000
04225  137      1100          STEP         (syll constant)
04231  137         001100     Multiplier   (parameter)
04237  137              1     STEP         (syll constant)
       138 100
04243  138   001111           Multiplier   (parameter)
04251  138         0010       LEN          (syll constant)
04255  138            001     Bit Length   (parameter)
       139 0000
04264  139    1010            VALUE        (syll constant)
04270  139        00000000    Valuespace
       140 0000000000000000
       141 0000000000000000
       142 0000000000000000
       143 0000000000000000
       144 0000000000000000
       145 0000000000000000
       146 0000000000000000
       147 0000000000000000
       148 0000000000000000
       149 0000000000000000
       150 0000000000000000
       151 0000000000000000
       152 0000000000000000
       153 0000000000000000
       154 0000000000000000
       155 0000000000000000
       156 0000000000000000
       157 0000000000000000
       158 0000
04744  158    011100011000 link
       159 00
04762  159  0010           INTEGER      (type constant)
04766  159       1001      STEPBOUND    (syll constant)
04772  159          110000 Bound        (parameter)
05000  160 100000          Multiplier   (parameter)
```

```
05006  160        1001          STEPBOUND    (syll constant)
05012  160             010000   Bound        (parameter)
05020  161 110000              Multiplier   (parameter)
05026  161        1001          STEPBOUND    (syll constant)
05032  161             101000   Bound        (parameter)
05040  162 001100              Multiplier   (parameter)
05046  162        0010          LEN          (syll constant)
05052  162             001000   Bit Length   (parameter)
       163 0
05061  163  1110                ADDR         (syll constant)
05065  163       01101011000    Address      (parameter)
       164 100
```

Dump 5. REF RA TO RAVEL A; (before RAVEL)

```
04100  132 00000000000000     link
04116  132                00   link
       133 111000000000
04134  133              1110   link
       134 1100000000
04152  134             10000   BEGIN        (mark constant)
04157  134                 0   Lexic Level  (parameter)
       135 000
04163  135    000000           Displacement (parameter)
04171  135           0001110   link
       136 1100000
04207  136         0010        INTEGER      (type constant)
04213  136             1100    STEP         (syll constant)
04217  136                1    Multiplier   (parameter)
       137 10000
04225  137       1100          STEP         (syll constant)
04231  137          001100     Multiplier   (parameter)
04237  137                1    STEP         (syll constant)
       138 100
04243  138    001111           Multiplier   (parameter)
04251  138          0010       LEN          (syll constant)
04255  138               001   Bit Length   (parameter)
       139 0000
04264  139     1010            VALUE        (syll constant)
04270  139          00000000   Valuespace
       140 0000000000000000
       141 0000000000000000
       142 0000000000000000
       143 0000000000000000
       144 0000000000000000
       145 0000000000000000
       146 0000000000000000
       147 0000000000000000
       148 0000000000000000
```

```
        149 0000000000000000
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
        154 0000000000000000
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
04744   158      010111000100 link
        159 00
04762   159    0010           INTEGER     (type constant)
04766   159         1001      STEPBOUND   (syll constant)
04772   159              110000 Bound      (parameter)
05000   160 100000            Multiplier  (parameter)
05006   160         1001      STEPBOUND   (syll constant)
05012   160              010000 Bound      (parameter)
05020   161 110000            Multiplier  (parameter)
05026   161         1001      STEPBOUND   (syll constant)
05032   161              101000 Bound      (parameter)
05040   162 001100            Multiplier  (parameter)
05046   162         0010      LEN         (syll constant)
05052   162              001000 Bit Length (parameter)
        163 0
05061   163  1110            ADDR        (syll constant)
05065   163       01101011000 Address     (parameter)
        164 100
05103   164     0110111011000 link
        165 0
05121   165  0010            INTEGER     (type constant)
05125   165       1001       STEPBOUND   (syll constant)
05131   165            010000 Bound      (parameter)
05137   165                 1 Multiplier  (parameter)
        166 10000
05145   166       1001       STEPBOUND   (syll constant)
05151   166            101000 Bound      (parameter)
05157   166                 0 Multiplier  (parameter)
        167 01100
05165   167       0010       LEN         (syll constant)
05171   167            0010000 Bit Length (parameter)
05200   168 1110            ADDR        (syll constant)
05204   168      000001110001 Address     (parameter)
        169 00
05222   169    01001001000000 link
05240   170 0010            INTEGER     (type constant)
05244   170      1001       STEPBOUND   (syll constant)
05250   170           110000 Bound      (parameter)
05256   170                00 Multiplier  (parameter)
        171 1100
05264   171      0010       LEN         (syll constant)
05270   171           0010000 Bit Length (parameter)
05277   171                 1 ADDR        (syll constant)
        172 110
```

```
05303  172     1110011010010 Address     (parameter)
       173 0
05321  173  00011010001111  link
05337  173                0 INTEGER     (type constant)
       174 010
05343  174    1100          STEP        (syll constant)
05347  174        110000    Multiplier  (parameter)
05355  174              110 STEP        (syll constant)
       175 0
05361  175  001100          Multiplier  (parameter)
05367  175       1100       STEP        (syll constant)
05373  175            00111 Multiplier  (parameter)
       176 1
05401  176  0010            LEN         (syll constant)
05405  176      0010000     Bit Length  (parameter)
05414  176             1110 ADDR        (syll constant)
05420  177 00011101000100   Address     (parameter)
```

Dump 6. REF RA TO RAVEL A;

```
04100  132 00000000000000   link
04116  132               00 link
       133 111000000000
04134  133             1110 link
       134 1100000000
04152  134          10000   BEGIN       (mark constant)
04157  134              0   Lexic Level (parameter)
       135 000
04163  135    000000        Displacement (parameter)
04171  135         0001110  link
       136 1100000
04207  136       0010       INTEGER     (type constant)
04213  136          1100    STEP        (syll constant)
04217  136             1    Multiplier  (parameter)
       137 10000
04225  137      1100        STEP        (syll constant)
04231  137         001100   Multiplier  (parameter)
04237  137             1    STEP        (syll constant)
       138 100
04243  138    001111        Multiplier  (parameter)
04251  138         0010     LEN         (syll constant)
04255  138            001   Bit Length  (parameter)
       139 0000
04264  139     1010         VALUE       (syll constant)
04270  139        00000000  Valuespace
       140 0000000000000000
       141 0000000000000000
       142 0000000000000000
       143 0000000000000000
```

```
        144 0000000000000000
        145 0000000000000000
        146 0000000000000000
        147 0000000000000000
        148 0000000000000000
        149 0000000000000000
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
        154 0000000000000000
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
04744   158     010111000100 link
        159 00
04762   159   0010           INTEGER     (type constant)
04766   159        1001      STEPBOUND   (syll constant)
04772   159           110000 Bound       (parameter)
05000   160 100000           Multiplier  (parameter)
05006   160        1001      STEPBOUND   (syll constant)
05012   160           010000 Bound       (parameter)
05020   161 110000           Multiplier  (parameter)
05026   161        1001      STEPBOUND   (syll constant)
05032   161           101000 Bound       (parameter)
05040   162 001100           Multiplier  (parameter)
05046   162        0010      LEN         (syll constant)
05052   162           001000 Bit Length  (parameter)
        163 0
05061   163  1110           ADDR        (syll constant)
05065   163      01101011000 Address     (parameter)
        164 100
05103   164    0110111011000 link
        165 0
05121   165  0010           INTEGER     (type constant)
05125   165       1001      STEPBOUND   (syll constant)
05131   165          010000 Bound       (parameter)
05137   165               1 Multiplier  (parameter)
        166 10000
05145   166       1001      STEPBOUND   (syll constant)
05151   166          101000 Bound       (parameter)
05157   166               0 Multiplier  (parameter)
        167 01100
05165   167       0010      LEN         (syll constant)
05171   167          0010000 Bit Length (parameter)
05200   168 1110           ADDR        (syll constant)
05204   168     000001110001 Address     (parameter)
        169 00
05222   169   01001001000000 link
05240   170 0010           INTEGER     (type constant)
05244   170      1001      STEPBOUND   (syll constant)
05250   170         110000 Bound       (parameter)
05256   170             00 Multiplier  (parameter)
```

```
        171 1100
05264   171     0010            LEN          (syll constant)
05270   171          0010000    Bit Length   (parameter)
05277   171                1 ADDR           (syll constant)
        172 110
05303   172    1110011010010 Address        (parameter)
        173 0
05321   173  00110001100000  link
05337   173                0 INTEGER        (type constant)
        174 010
05343   174    1100          STEP           (syll constant)
05347   174        001111    Multiplier     (parameter)
05355   174             001 LEN            (syll constant)
        175 0
05361   175  0010000         Bit Length     (parameter)
05370   175         1110     ADDR           (syll constant)
05374   175            0001 Address        (parameter)
        176 1101000100
```

Dump 7. REF RB TO RAVEL B; (before RAVEL)

```
04100   132 00000000000000   link
04116   132              00 link
        133 111000000000
04134   133            1110 link
        134 1100000000
04152   134           10000 BEGIN          (mark constant)
04157   134               0 Lexic Level    (parameter)
        135 000
04163   135    000000       Displacement   (parameter)
04171   135          0001110 link
        136 1100000
04207   136        0010       INTEGER      (type constant)
04213   136           1100    STEP         (syll constant)
04217   136              1 Multiplier      (parameter)
        137 10000
04225   137      1100         STEP         (syll constant)
04231   137         001100    Multiplier   (parameter)
04237   137              1 STEP           (syll constant)
        138 100
04243   138    001111         Multiplier   (parameter)
04251   138         0010      LEN          (syll constant)
04255   138            001 Bit Length     (parameter)
        139 0000
04264   139     1010          VALUE        (syll constant)
04270   139         00000000 Valuespace
        140 0000000000000000
        141 0000000000000000
        142 0000000000000000
```

```
        143 0000000000000000
        144 0000000000000000
        145 0000000000000000
        146 0000000000000000
        147 0000000000000000
        148 0000000000000000
        149 0000000000000000
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
        154 0000000000000000
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
04744   158     010111000100 link
        159 00
04762   159   0010           INTEGER     (type constant)
04766   159        1001      STEPBOUND   (syll constant)
04772   159             110000 Bound      (parameter)
05000   160 100000           Multiplier  (parameter)
05006   160        1001      STEPBOUND   (syll constant)
05012   160             010000 Bound      (parameter)
05020   161 110000           Multiplier  (parameter)
05026   161        1001      STEPBOUND   (syll constant)
05032   161             101000 Bound      (parameter)
05040   162 001100           Multiplier  (parameter)
05046   162        0010      LEN         (syll constant)
05052   162             001000 Bit Length (parameter)
        163 0
05061   163  1110            ADDR        (syll constant)
05065   163      01101011000 Address     (parameter)
        164 100
05103   164    0110111011000 link
        165 0
05121   165  0010            INTEGER     (type constant)
05125   165      1001        STEPBOUND   (syll constant)
05131   165          010000  Bound       (parameter)
05137   165                1 Multiplier  (parameter)
        166 10000
05145   166      1001        STEPBOUND   (syll constant)
05151   166          101000  Bound       (parameter)
05157   166                0 Multiplier  (parameter)
        167 01100
05165   167      0010        LEN         (syll constant)
05171   167          0010000 Bit Length  (parameter)
05200   168 1110             ADDR        (syll constant)
05204   168     000001110001 Address     (parameter)
        169 00
05222   169   010010010000000 link
05240   170 0010             INTEGER     (type constant)
05244   170     1001         STEPBOUND   (syll constant)
05250   170         110000   Bound       (parameter)
```

```
05256  170                  00 Multiplier  (parameter)
       171 1100
05264  171     0010            LEN         (syll constant)
05270  171          0010000    Bit Length  (parameter)
05277  171                1 ADDR          (syll constant)
       172 110
05303  172    1110011010010 Address       (parameter)
       173 0
05321  173  000110011000000  link
05337  173                0 INTEGER       (type constant)
       174 010
05343  174    1100            STEP        (syll constant)
05347  174        001111      Multiplier  (parameter)
05355  174             001 LEN           (syll constant)
       175 0
05361  175  0010000           Bit Length  (parameter)
05370  175         1110       ADDR        (syll constant)
05374  175              0001 Address       (parameter)
       176 1101000100
05412  176           001110 link
       177 00011101
05430  177         0010       INTEGER     (type constant)
05434  177             1001 STEPBOUND     (syll constant)
05440  178 110000           Bound         (parameter)
05446  178       100000     Multiplier    (parameter)
05454  178             1001 STEPBOUND     (syll constant)
05460  179 010000           Bound         (parameter)
05466  179       110000     Multiplier    (parameter)
05474  179             1001 STEPBOUND     (syll constant)
05500  180 101000           Bound         (parameter)
05506  180       001100     Multiplier    (parameter)
05514  180             0010 LEN           (syll constant)
05520  181 0010000          Bit Length    (parameter)
05527  181         1110       ADDR        (syll constant)
05533  181           01101 Address       (parameter)
       182 011000100
```

Dump 8. REF RB TO RAVEL B;

```
04100  132 00000000000000   link
04116  132                00 link
       133 111000000000
04134  133             1110 link
       134 1100000000
04152  134           10000  BEGIN         (mark constant)
04157  134               0  Lexic Level   (parameter)
       135 000
04163  135    000000        Displacement  (parameter)
04171  135         0001110  link
```

```
        136 1100000
04207   136         0010        INTEGER     (type constant)
04213   136             1100    STEP        (syll constant)
04217   136                 1   Multiplier  (parameter)
        137 10000
04225   137       1100          STEP        (syll constant)
04231   137           001100    Multiplier  (parameter)
04237   137                 1   STEP        (syll constant)
        138 100
04243   138     001111          Multiplier  (parameter)
04251   138           0010      LEN         (syll constant)
04255   138               001   Bit Length  (parameter)
        139 0000
04264   139      1010           VALUE       (syll constant)
04270   139           00000000  Valuespace
        140 0000000000000000
        141 0000000000000000
        142 0000000000000000
        143 0000000000000000
        144 0000000000000000
        145 0000000000000000
        146 0000000000000000
        147 0000000000000000
        148 0000000000000000
        149 0000000000000000
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
        154 0000000000000000
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
04744   158      010111000100 link
        159 00
04762   159    0010           INTEGER     (type constant)
04766   159         1001      STEPBOUND   (syll constant)
04772   159             110000 Bound       (parameter)
05000   160 100000            Multiplier  (parameter)
05006   160         1001      STEPBOUND   (syll constant)
05012   160             010000 Bound       (parameter)
05020   161 110000            Multiplier  (parameter)
05026   161         1001      STEPBOUND   (syll constant)
05032   161             101000 Bound       (parameter)
05040   162 001100            Multiplier  (parameter)
05046   162         0010      LEN         (syll constant)
05052   162             001000 Bit Length  (parameter)
        163 0
05061   163   1110            ADDR        (syll constant)
05065   163       01101011000 Address     (parameter)
        164 100
05103   164     0110111011000 link
        165 0
```

```
05121  165  0010                INTEGER      (type constant)
05125  165       1001           STEPBOUND    (syll constant)
05131  165            010000    Bound        (parameter)
05137  165                   1  Multiplier   (parameter)
       166 10000
05145  166       1001           STEPBOUND    (syll constant)
05151  166            101000    Bound        (parameter)
05157  166                   0  Multiplier   (parameter)
       167 01100
05165  167       0010           LEN          (syll constant)
05171  167            0010000   Bit Length   (parameter)
05200  168 1110                ADDR         (syll constant)
05204  168      000001110001     Address      (parameter)
       169 00
05222  169    01001001000000    link
05240  170 0010                INTEGER      (type constant)
05244  170      1001           STEPBOUND    (syll constant)
05250  170           110000    Bound        (parameter)
05256  170                 00  Multiplier   (parameter)
       171 1100
05264  171      0010           LEN          (syll constant)
05270  171           0010000   Bit Length   (parameter)
05277  171                   1  ADDR         (syll constant)
       172 110
05303  172     1110011010010    Address      (parameter)
       173 0
05321  173   00011001100000     link
05337  173                   0  INTEGER      (type constant)
       174 010
05343  174     1100           STEP         (syll constant)
05347  174         001111       Multiplier   (parameter)
05355  174                 001  LEN          (syll constant)
       175 0
05361  175  0010000            Bit Length   (parameter)
05370  175          1110       ADDR         (syll constant)
05374  175              0001   Address      (parameter)
       176 1101000100
05412  176            000111   link
       177 01100000
05430  177          0010       INTEGER      (type constant)
05434  177               0101  PATCH        (syll constant)
05440  178 000000             Offset       (parameter)
05446  178        111111       Bound 1      (parameter)
05454  178               1101  SEGSPACE     (syll constant)
05460  179 110000             Physical Multiplier (parameter
05466  179        100000       Logical Multiplier (parameter)
05474  179               1101  SEGSPACE     (syll constant)
05500  180 011000             Physical Multiplier (parameter
05506  180        110000       Logical Multiplier (parameter)
05514  180               1101  SEGSPACE     (syll constant)
05520  181 011110             Physical Multiplier (parameter
05526  181        001100       Logical Multiplier (parameter)
05534  181               0010  LEN          (syll constant)
05540  182 0010000            Bit Length   (parameter)
```

```
05547  182          0011      SADDR        (syll constant)
05553  182               01101 Stack Pointer (parameter)
       183 011000100
05571  183            0111100 Bound 2       (parameter)
       184 0000000
```

Dump 9. The workspace after END'DECL

```
04100  132 000000000000000    link
04116  132                00 link
       133 111000000000
04134  133              1110 link
       134 1100000000
04152  134             10000 BEGIN          (mark constant)
04157  134                 0 Lexic Level    (parameter)
       135 000
04163  135    000000         Displacement   (parameter)
```

Comment- The following section corresponds to

INTEGER A[3,4,5] BITS 4;

```
04171  135           0001110 link
       136 1100000
04207  136         0010      INTEGER        (type constant)
04213  136             1100  STEP           (syll constant)
04217  136                1  Multiplier     (parameter)
       137 10000
04225  137       1100        STEP           (syll constant)
04231  137           001100  Multiplier     (parameter)
04237  137                1  STEP           (syll constant)
       138 100
04243  138    001111         Multiplier     (parameter)
04251  138           0010    LEN            (syll constant)
04255  138              001  Bit Length     (parameter)
       139 0000
04264  139     1010          VALUE          (syll constant)
04270  139          00000000 Valuespace
       140 0000000000000000
       141 0000000000000000
       142 0000000000000000
       143 0000000000000000
       144 0000000000000000
       145 0000000000000000
       146 0000000000000000
       147 0000000000000000
       148 0000000000000000
       149 0000000000000000
```

```
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
        154 0000000000000000
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
```

Comment- The following section corresponds to

REF B TO A[,2:3,];

```
04744   158      010111000100 link
        159 00
04762   159   0010            INTEGER     (type constant)
04766   159        1001       STEPBOUND   (syll constant)
04772   159             110000 Bound       (parameter)
05000   160 100000             Multiplier  (parameter)
05006   160        1001        STEPBOUND   (syll constant)
05012   160             010000 Bound       (parameter)
05020   161 110000             Multiplier  (parameter)
05026   161        1001        STEPBOUND   (syll constant)
05032   161             101000 Bound       (parameter)
05040   162 001100             Multiplier  (parameter)
05046   162      0010           LEN         (syll constant)
05052   162             001000 Bit Length  (parameter)
        163 0
05061   163  1110              ADDR        (syll constant)
05065   163       01101011000 Address     (parameter)
        164 100
```

Comment- The following section corresponds to

REF C TO B[2];

```
05103   164     0110111011000 link
        165 0
05121   165  0010              INTEGER     (type constant)
05125   165       1001         STEPBOUND   (syll constant)
05131   165            010000  Bound       (parameter)
05137   165                 1  Multiplier  (parameter)
        166 10000
05145   166       1001         STEPBOUND   (syll constant)
05151   166            101000  Bound       (parameter)
05157   166                 0  Multiplier  (parameter)
        167 01100
05165   167       0010         LEN         (syll constant)
05171   167            0010000 Bit Length  (parameter)
05200   168 1110               ADDR        (syll constant)
05204   168      000001110001 Address     (parameter)
```

```
                169 00
```

Comment- The following section corresponds to

REF D TO C[1,2:4];

```
05222   169   01001001000000 link
05240   170 0010             INTEGER     (type constant)
05244   170     1001         STEPBOUND   (syll constant)
05250   170         110000   Bound       (parameter)
05256   170               00 Multiplier  (parameter)
        171 1100
05264   171     0010         LEN         (syll constant)
05270   171         0010000  Bit Length  (parameter)
05277   171               1  ADDR        (syll constant)
        172 110
05303   172    1110011010010 Address     (parameter)
        173 0
```

Comment- The following section corresponds to

REF RA TO RAVEL A;

```
05321   173  00011001100000  link
05337   173               0  INTEGER     (type constant)
        174 010
05343   174    1100          STEP        (syll constant)
05347   174        001111    Multiplier  (parameter)
05355   174             001  LEN         (syll constant)
        175 0
05361   175  0010000         Bit Length  (parameter)
05370   175         1110      ADDR        (syll constant)
05374   175            0001   Address     (parameter)
        176 1101000100
```

Comment- The following section corresponds to

REF RB TO RAVEL B;

```
05412   176           011010 link
        177 10100000
05430   177         0010     INTEGER     (type constant)
05434   177            0101  PATCH       (syll constant)
05440   178 000000           Offset      (parameter)
05446   178       111111     Bound 1     (parameter)
05454   178            1101  SEGSPACE    (syll constant)
05460   179 110000           Physical Multiplier (parameter
05466   179       100000     Logical Multiplier (parameter)
05474   179            1101  SEGSPACE    (syll constant)
05500   180 011000           Physical Multiplier (parameter
05506   180       110000     Logical Multiplier (parameter)
```

```
05514  180                1101 SEGSPACE    (syll constant)
05520  181 011110             Physical Multiplier (parameter
05526  181       001100       Logical Multiplier (parameter)
05534  181                0010 LEN          (syll constant)
05540  182 0010000            Bit Length    (parameter)
05547  182         0011       SADDR         (syll constant)
05553  182              01101 Stack Pointer (parameter)
       183 011000100
05571  183           0111100 Bound 2       (parameter)
       184 0000000
```

Comment- The following section corresponds to

REF SV TO RB[2:3];

```
05607  184         011100001 link
       185 11000
05625  185       0010        INTEGER       (type constant)
05631  185           0101    PATCH         (syll constant)
05635  185               110 Offset        (parameter)
       186 000
05643  186     101000        Bound 1       (parameter)
05651  186           1101    SEGSPACE      (syll constant)
05655  186               110 Physical Multiplier (parameter
       187 000
05663  187     100000        Logical Multiplier (parameter)
05671  187           1101    SEGSPACE      (syll constant)
05675  187               011 Physical Multiplier (parameter
       188 000
05703  188     110000        Logical Multiplier (parameter)
05711  188           1101    SEGSPACE      (syll constant)
05715  188               011 Physical Multiplier (parameter
       189 110
05723  189     001100        Logical Multiplier (parameter)
05731  189           0010    LEN           (syll constant)
05735  189               001 Bit Length    (parameter)
       190 0000
05744  190      0011         SADDR         (syll constant)
05750  190           01101011 Stack Pointer (parameter)
       191 000100
05766  191        0111100000 Bound 2       (parameter)
       192 0000
```

Comment- The following section corresponds to

REF X TO SV[1];

```
06004  192      001011011110 link
       193 00
06022  193   0010            INTEGER       (type constant)
06026  193       0010        LEN           (syll constant)
06032  193           001000  Bit Length    (parameter)
```

```
        194 0
06041   194  1110              ADDR        (syll constant)
06045   194       01010111000 Address     (parameter)
        195 100
06063   195     0000000111010 link
        196 0
06101   196  11000             MARK        (mark constant)
```

Dump 10. 11→SV[3].... (before ASSIGN)

```
04100   132 00000000000000    link
04116   132                00 link
        133 111000000000
04134   133              1110 link
        134 1100000000
04152   134           10000   BEGIN       (mark constant)
04157   134               0   Lexic Level (parameter)
        135 000
04163   135    000000         Displacement (parameter)
04171   135          0001110  link
        136 1100000
04207   136        0010       INTEGER     (type constant)
04213   136           1100    STEP        (syll constant)
04217   136              1    Multiplier  (parameter)
        137 10000
04225   137      1100         STEP        (syll constant)
04231   137         001100    Multiplier  (parameter)
04237   137              1    STEP        (syll constant)
        138 100
04243   138    001111         Multiplier  (parameter)
04251   138         0010      LEN         (syll constant)
04255   138            001    Bit Length  (parameter)
        139 0000
04264   139     1010          VALUE       (syll constant)
04270   139        00000000   Valuespace
        140 0000000000000000
        141 0000000000000000
        142 0000000000000000
        143 0000000000000000
        144 0000000000000000
        145 0000000000000000
        146 0000000000000000
        147 0000000000000000
        148 0000000000000000
        149 0000000000000000
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
```

```
        154 0000000000000000
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
04744   158     010111000100 link
        159 00
04762   159   0010           INTEGER     (type constant)
04766   159        1001      STEPBOUND   (syll constant)
04772   159           110000 Bound       (parameter)
05000   160 100000           Multiplier  (parameter)
05006   160        1001      STEPBOUND   (syll constant)
05012   160           010000 Bound       (parameter)
05020   161 110000           Multiplier  (parameter)
05026   161        1001      STEPBOUND   (syll constant)
05032   161           101000 Bound       (parameter)
05040   162 001100           Multiplier  (parameter)
05046   162        0010      LEN         (syll constant)
05052   162           001000 Bit Length  (parameter)
        163 0
05061   163  1110            ADDR        (syll constant)
05065   163      01101011000 Address     (parameter)
        164 100
05103   164    0110111011000 link
        165 0
05121   165  0010            INTEGER     (type constant)
05125   165       1001       STEPBOUND   (syll constant)
05131   165          010000  Bound       (parameter)
05137   165                1 Multiplier  (parameter)
        166 10000
05145   166       1001       STEPBOUND   (syll constant)
05151   166          101000  Bound       (parameter)
05157   166                0 Multiplier  (parameter)
        167 01100
05165   167       0010       LEN         (syll constant)
05171   167          0010000 Bit Length  (parameter)
05200   168 1110             ADDR        (syll constant)
05204   168     000001110001 Address     (parameter)
        169 00
05222   169   01001001000000 link
05240   170 0010             INTEGER     (type constant)
05244   170     1001         STEPBOUND   (syll constant)
05250   170        110000    Bound       (parameter)
05256   170              00  Multiplier  (parameter)
        171 1100
05264   171     0010         LEN         (syll constant)
05270   171        0010000   Bit Length  (parameter)
05277   171                1 ADDR        (syll constant)
        172 110
05303   172    1110011010010 Address     (parameter)
        173 0
05321   173  00011001100000  link
05337   173                0 INTEGER     (type constant)
        174 010
```

```
05343  174     1100            STEP          (syll constant)
05347  174          001111     Multiplier    (parameter)
05355  174               001   LEN           (syll constant)
       175 0
05361  175  0010000            Bit Length    (parameter)
05370  175          1110       ADDR          (syll constant)
05374  175               0001  Address       (parameter)
       176 1101000100
05412  176            011010   link
       177 10100000
05430  177          0010       INTEGER       (type constant)
05434  177               0101  PATCH         (syll constant)
05440  178 000000             Offset        (parameter)
05446  178        111111       Bound 1       (parameter)
05454  178               1101  SEGSPACE      (syll constant)
05460  179 110000             Physical Multiplier (parameter
05466  179        100000       Logical Multiplier (parameter)
05474  179               1101  SEGSPACE      (syll constant)
05500  180 011000             Physical Multiplier (parameter
05506  180        110000       Logical Multiplier (parameter)
05514  180               1101  SEGSPACE      (syll constant)
05520  181 011110             Physical Multiplier (parameter
05526  181        001100       Logical Multiplier (parameter)
05534  181               0010  LEN           (syll constant)
05540  182 0010000            Bit Length    (parameter)
05547  182         0011        SADDR         (syll constant)
05553  182              01101  Stack Pointer (parameter)
       183 011000100
05571  183           0111100   Bound 2       (parameter)
       184 0000000
05607  184         011100001   link
       185 11000
05625  185       0010          INTEGER       (type constant)
05631  185            0101     PATCH         (syll constant)
05635  185                 110 Offset        (parameter)
       186 000
05643  186     101000          Bound 1       (parameter)
05651  186            1101     SEGSPACE      (syll constant)
05655  186                 110 Physical Multiplier (parameter
       187 000
05663  187     100000          Logical Multiplier (parameter)
05671  187            1101     SEGSPACE      (syll constant)
05675  187                 011 Physical Multiplier (parameter
       188 000
05703  188     110000          Logical Multiplier (parameter)
05711  188            1101     SEGSPACE      (syll constant)
05715  188                 011 Physical Multiplier (parameter
       189 110
05723  189     001100          Logical Multiplier (parameter)
05731  189            0010     LEN           (syll constant)
05735  189                 001 Bit Length    (parameter)
       190 0000
05744  190      0011           SADDR         (syll constant)
05750  190          01101011   Stack Pointer (parameter)
```

```
       191 000100
05766  191       0111100000 Bound 2      (parameter)
       192 0000
06004  192     001011011110 link
       193 00
06022  193   0010           INTEGER      (type constant)
06026  193       0010       LEN          (syll constant)
06032  193           001000 Bit Length   (parameter)
       194 0
06041  194  1110            ADDR         (syll constant)
06045  194      01010111000 Address      (parameter)
       195 100
06063  195    0100001000000 link
       196 0
06101  196  11000           MARK         (mark constant)
06106  196       0101001000 link
       197 0000
06124  197     0100          CARDINAL     (type constant)
06130  197         0010      LEN          (syll constant)
06134  197             0010  Bit Length   (parameter)
       198 000
06143  198    0110           CONSTANT     (syll constant)
06147  198        1110       ADDR         (syll constant)
06153  198           00001   Address      (parameter)
       199 011011000
06171  199          0101001  link
       200 1000000
06207  200        0010       INTEGER      (type constant)
06213  200            0010   LEN          (syll constant)
06217  200               0   Bit Length   (parameter)
       201 010000
06226  201        1110       ADDR         (syll constant)
06232  201          010010   Address      (parameter)
       202 00100100
```

Dump 11. ....SV[3]-20.... (before S''SUBTRACT)

```
04100  132 00000000000000   link
04116  132               00 link
       133 111000000000
04134  133             1110 link
       134 1100000000
04152  134          10000   BEGIN        (mark constant)
04157  134              0   Lexic Level  (parameter)
       135 000
04163  135    000000        Displacement (parameter)
04171  135         0001110  link
       136 1100000
04207  136        0010       INTEGER      (type constant)
```

```
04213  136              1100  STEP          (syll constant)
04217  136                 1  Multiplier    (parameter)
       137 10000
04225  137      1100          STEP          (syll constant)
04231  137           001100   Multiplier    (parameter)
04237  137                 1  STEP          (syll constant)
       138 100
04243  138    001111          Multiplier    (parameter)
04251  138          0010      LEN           (syll constant)
04255  138               001  Bit Length    (parameter)
       139 0000
04264  139     1010           VALUE         (syll constant)
04270  139         00000000   Valuespace
       140 0000000000000000
       141 0000000000000000
       142 0000000000000000
       143 0000000000000000
       144 0000000000000000
       145 0001101000000000   (Value of SV[3])
       146 0000000000000000
       147 0000000000000000
       148 0000000000000000
       149 0000000000000000
       150 0000000000000000
       151 0000000000000000
       152 0000000000000000
       153 0000000000000000
       154 0000000000000000
       155 0000000000000000
       156 0000000000000000
       157 0000000000000000
       158 0000
04744  158     010111000100   link
       159 00
04762  159   0010             INTEGER       (type constant)
04766  159        1001        STEPBOUND     (syll constant)
04772  159             110000 Bound         (parameter)
05000  160 100000            Multiplier    (parameter)
05006  160        1001        STEPBOUND     (syll constant)
05012  160             010000 Bound         (parameter)
05020  161 110000            Multiplier    (parameter)
05026  161        1001        STEPBOUND     (syll constant)
05032  161             101000 Bound         (parameter)
05040  162 001100            Multiplier    (parameter)
05046  162        0010        LEN           (syll constant)
05052  162             001000 Bit Length    (parameter)
       163 0
05061  163  1110              ADDR          (syll constant)
05065  163      01101011000   Address       (parameter)
       164 100
05103  164    0110111011000   link
       165 0
05121  165  0010              INTEGER       (type constant)
05125  165       1001         STEPBOUND     (syll constant)
```

```
05131  165          010000  Bound        (parameter)
05137  165               1  Multiplier   (parameter)
       166 10000
05145  166    1001          STEPBOUND    (syll constant)
05151  166         101000   Bound        (parameter)
05157  166              0   Multiplier   (parameter)
       167 01100
05165  167    0010          LEN          (syll constant)
05171  167         0010000  Bit Length   (parameter)
05200  168 1110             ADDR         (syll constant)
05204  168   000001110001   Address      (parameter)
       169 00
05222  169  01001001000000  link
05240  170 0010             INTEGER      (type constant)
05244  170    1001          STEPBOUND    (syll constant)
05250  170       110000     Bound        (parameter)
05256  170            00    Multiplier   (parameter)
       171 1100
05264  171    0010          LEN          (syll constant)
05270  171         0010000  Bit Length   (parameter)
05277  171               1  ADDR         (syll constant)
       172 110
05303  172   1110011010010  Address      (parameter)
       173 0
05321  173  00011001100000  link
05337  173              0   INTEGER      (type constant)
       174 010
05343  174   1100           STEP         (syll constant)
05347  174      001111      Multiplier   (parameter)
05355  174          001     LEN          (syll constant)
       175 0
05361  175  0010000         Bit Length   (parameter)
05370  175       1110       ADDR         (syll constant)
05374  175           0001   Address      (parameter)
       176 1101000100
05412  176         011010   link
       177 10100000
05430  177       0010       INTEGER      (type constant)
05434  177           0101   PATCH        (syll constant)
05440  178 000000           Offset       (parameter)
05446  178      111111      Bound 1      (parameter)
05454  178          1101    SEGSPACE     (syll constant)
05460  179 110000           Physical Multiplier (parameter
05466  179      100000      Logical Multiplier (parameter)
05474  179          1101    SEGSPACE     (syll constant)
05500  180 011000           Physical Multiplier (parameter
05506  180      110000      Logical Multiplier (parameter)
05514  180          1101    SEGSPACE     (syll constant)
05520  181 011110           Physical Multiplier (parameter
05526  181      001100      Logical Multiplier (parameter)
05534  181          0010    LEN          (syll constant)
05540  182 0010000          Bit Length   (parameter)
05547  182       0011       SADDR        (syll constant)
05553  182         01101    Stack Pointer (parameter)
```

```
        183 011000100
05571   183          0111100 Bound 2      (parameter)
        184 0000000
05607   184        011100001 link
        185 11000
05625   185      0010        INTEGER      (type constant)
05631   185           0101   PATCH        (syll constant)
05635   185                110 Offset      (parameter)
        186 000
05643   186    101000        Bound 1      (parameter)
05651   186           1101   SEGSPACE     (syll constant)
05655   186                110 Physical Multiplier (parameter
        187 000
05663   187    100000        Logical Multiplier (parameter)
05671   187           1101   SEGSPACE     (syll constant)
05675   187                011 Physical Multiplier (parameter
        188 000
05703   188    110000        Logical Multiplier (parameter)
05711   188           1101   SEGSPACE     (syll constant)
05715   188                011 Physical Multiplier (parameter
        189 110
05723   189    001100        Logical Multiplier (parameter)
05731   189           0010   LEN          (syll constant)
05735   189                001 Bit Length  (parameter)
        190 0000
05744   190     0011         SADDR        (syll constant)
05750   190         01101011 Stack Pointer (parameter)
        191 000100
05766   191       0111100000 Bound 2      (parameter)
        192 0000
06004   192     001011011110 link
        193 00
06022   193   0010           INTEGER      (type constant)
06026   193        0010      LEN          (syll constant)
06032   193             001000 Bit Length  (parameter)
        194 0
06041   194  1110            ADDR         (syll constant)
06045   194       01010111000 Address      (parameter)
        195 100
06063   195    0100001000000 link
        196 0
06101   196  11000           MARK         (mark constant)
06106   196       1111101000 link
        197 0000
06124   197     0010         INTEGER      (type constant)
06130   197          0010    LEN          (syll constant)
06134   197               0010 Bit Length  (parameter)
        198 000
06143   198    1010          VALUE        (syll constant)
06147   198         01101     Valuespace
06154   198               0001 link
        199 0110110000
06172   199             0100  CARDINAL     (type constant)
06176   199                00 LEN          (syll constant)
```

```
        200 10
06202   200   1010000          Bit Length   (parameter)
06211   200           0110     CONSTANT     (syll constant)
06215   200               111  ADDR         (syll constant)
        201 0
06221   201  00000101111000    Address      (parameter)
```

Dump 12. ....SV[3]-20→X (before ASSIGN)

```
04100   132 000000000000000    link
04116   132                00  link
        133 111000000000
04134   133             1110   link
        134 1100000000
04152   134           10000    BEGIN        (mark constant)
04157   134               0    Lexic Level  (parameter)
        135 000
04163   135    000000          Displacement (parameter)
04171   135          0001110   link
        136 1100000
04207   136        0010        INTEGER      (type constant)
04213   136           1100     STEP         (syll constant)
04217   136              1     Multiplier   (parameter)
        137 10000
04225   137      1100          STEP         (syll constant)
04231   137         001100     Multiplier   (parameter)
04237   137              1     STEP         (syll constant)
        138 100
04243   138    001111          Multiplier   (parameter)
04251   138         0010       LEN          (syll constant)
04255   138            001     Bit Length   (parameter)
        139 0000
04264   139     1010           VALUE        (syll constant)
04270   139        00000000    Valuespace
        140 0000000000000000
        141 0000000000000000
        142 0000000000000000
        143 0000000000000000
        144 0000000000000000
        145 0001101000000000   (Value of SV[3])
        146 0000000000000000
        147 0000000000000000
        148 0000000000000000
        149 0000000000000000
        150 0000000000000000
        151 0000000000000000
        152 0000000000000000
        153 0000000000000000
        154 0000000000000000
```

AD-A055 235 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
SIMULATION OF THE DIRECT EXECUTION OF A HIGHER ORDER LANGUAGE.(U)
MAR 78 D L AKIN, B C ALLEN

UNCLASSIFIED AFIT/GCS/EE/78-1 NL

3 OF 3
AD
A055235

END
DATE
FILMED
7-78
DDC

```
        155 0000000000000000
        156 0000000000000000
        157 0000000000000000
        158 0000
04744   158      010111000100 link
        159 00
04762   159    0010           INTEGER     (type constant)
04766   159         1001      STEPBOUND   (syll constant)
04772   159              110000 Bound      (parameter)
05000   160 100000             Multiplier  (parameter)
05006   160         1001       STEPBOUND   (syll constant)
05012   160              010000 Bound      (parameter)
05020   161 110000             Multiplier  (parameter)
05026   161         1001       STEPBOUND   (syll constant)
05032   161              101000 Bound      (parameter)
05040   162 001100             Multiplier  (parameter)
05046   162         0010       LEN         (syll constant)
05052   162              001000 Bit Length (parameter)
        163 0
05061   163  1110              ADDR        (syll constant)
05065   163       01101011000  Address     (parameter)
        164 100
05103   164     0110111011000 link
        165 0
05121   165  0010              INTEGER     (type constant)
05125   165       1001         STEPBOUND   (syll constant)
05131   165            010000  Bound       (parameter)
05137   165                 1  Multiplier  (parameter)
        166 10000
05145   166       1001         STEPBOUND   (syll constant)
05151   166            101000  Bound       (parameter)
05157   166                 0  Multiplier  (parameter)
        167 01100
05165   167       0010         LEN         (syll constant)
05171   167            0010000 Bit Length (parameter)
05200   168 1110               ADDR        (syll constant)
05204   168     000001110001   Address     (parameter)
        169 00
05222   169    0100100100000  link
05240   170 0010               INTEGER     (type constant)
05244   170      1001          STEPBOUND   (syll constant)
05250   170           110000   Bound       (parameter)
05256   170                 00 Multiplier  (parameter)
        171 1100
05264   171      0010          LEN         (syll constant)
05270   171           0010000  Bit Length (parameter)
05277   171                 1  ADDR        (syll constant)
        172 110
05303   172     1110011010010  Address     (parameter)
        173 0
05321   173  0001100110000     link
05337   173                 0  INTEGER     (type constant)
        174 010
05343   174     1100           STEP        (syll constant)
```

```
05347  174          001111    Multiplier  (parameter)
05355  174                001 LEN         (syll constant)
       175 0
05361  175  0010000           Bit Length  (parameter)
05370  175           1110     ADDR        (syll constant)
05374  175                0001 Address     (parameter)
       176 1101000100
05412  176             011010 link
       177 10100000
05430  177           0010     INTEGER     (type constant)
05434  177                0101 PATCH       (syll constant)
05440  178 000000             Offset      (parameter)
05446  178        111111       Bound 1     (parameter)
05454  178                1101 SEGSPACE    (syll constant)
05460  179 110000             Physical Multiplier (parameter
05466  179        100000       Logical Multiplier (parameter)
05474  179                1101 SEGSPACE    (syll constant)
05500  180 011000             Physical Multiplier (parameter
05506  180        110000       Logical Multiplier (parameter)
05514  180                1101 SEGSPACE    (syll constant)
05520  181 011110             Physical Multiplier (parameter
05526  181        001100       Logical Multiplier (parameter)
05534  181                0010 LEN         (syll constant)
05540  182 0010000            Bit Length  (parameter)
05547  182         0011        SADDR       (syll constant)
05553  182               01101 Stack Pointer (parameter)
       183 011000100
05571  183            0111100 Bound 2      (parameter)
       184 0000000
05607  184         011100001 link
       185 11000
05625  185       0010         INTEGER     (type constant)
05631  185            0101    PATCH       (syll constant)
05635  185                 110 Offset      (parameter)
       186 000
05643  186     101000         Bound 1     (parameter)
05651  186            1101    SEGSPACE    (syll constant)
05655  186                 110 Physical Multiplier (parameter
       187 000
05663  187     100000         Logical Multiplier (parameter)
05671  187            1101    SEGSPACE    (syll constant)
05675  187                 011 Physical Multiplier (parameter
       188 000
05703  188     110000         Logical Multiplier (parameter)
05711  188            1101    SEGSPACE    (syll constant)
05715  188                 011 Physical Multiplier (parameter
       189 110
05723  189     001100         Logical Multiplier (parameter)
05731  189            0010    LEN         (syll constant)
05735  189                 001 Bit Length  (parameter)
       190 0000
05744  190      0011          SADDR       (syll constant)
05750  190          01101011 Stack Pointer (parameter)
       191 000100
```

```
05766  191        0111100000 Bound 2       (parameter)
       192 0000
06004  192     001011011110 link
       193 00
06022  193   0010           INTEGER       (type constant)
06026  193        0010      LEN           (syll constant)
06032  193           001000 Bit Length    (parameter)
       194 0
06041  194  1110             ADDR          (syll constant)
06045  194       01010111000 Address       (parameter)
       195 100
06063  195    0100001000000 link
       196 0
06101  196  11000            MARK          (mark constant)
06106  196       0111101000 link
       197 0000
06124  197     0010          INTEGER       (type constant)
06130  197          0010     LEN           (syll constant)
06134  197             1010  Bit Length    (parameter)
       198 000
06143  198    1010           VALUE         (syll constant)
06147  198         110010    Valuespace
06155  198              001  link
       199 01101100000
06173  199            0010   INTEGER       (type constant)
06177  199               0   LEN           (syll constant)
       200 010
06203  200    0010000        Bit Length    (parameter)
06212  200          1110     ADDR          (syll constant)
06216  200            01     Address       (parameter)
       201 010111000100
```

Dump 13. 11→SV[3]-20→X

```
04100  132 00000000000000   link
04116  132              00   link
       133 111000000000
04134  133            1110   link
       134 1100000000
04152  134           10000   BEGIN         (mark constant)
04157  134               0   Lexic Level   (parameter)
       135 000
04163  135    000000         Displacement  (parameter)
04171  135         0001110   link
       136 1100000
04207  136        0010       INTEGER       (type constant)
04213  136           1100    STEP          (syll constant)
04217  136              1    Multiplier    (parameter)
       137 10000
```

```
04225  137        1100          STEP          (syll constant)
04231  137             001100   Multiplier    (parameter)
04237  137                  1   STEP          (syll constant)
       138 100
04243  138     001111           Multiplier    (parameter)
04251  138          0010        LEN           (syll constant)
04255  138              001     Bit Length    (parameter)
       139 0000
04264  139      1010            VALUE         (syll constant)
04270  139           00000000   Valuespace
       140 0000000000000000
       141 0000000000000000
       142 0000000000110010     (Value of X)
       143 0000000000000000
       144 0000000000000000
       145 0011010000000000     (Value of SV[3])
       146 0000000000000000
       147 0000000000000000
       148 0000000000000000
       149 0000000000000000
       150 0000000000000000
       151 0000000000000000
       152 0000000000000000
       153 0000000000000000
       154 0000000000000000
       155 0000000000000000
       156 0000000000000000
       157 0000000000000000
       158 0000
04744  158      010111000100    link
       159 00
04762  159    0010              INTEGER       (type constant)
04766  159         1001         STEPBOUND     (syll constant)
04772  159              110000  Bound         (parameter)
05000  160 100000               Multiplier    (parameter)
05006  160         1001         STEPBOUND     (syll constant)
05012  160              010000  Bound         (parameter)
05020  161 110000               Multiplier    (parameter)
05026  161         1001         STEPBOUND     (syll constant)
05032  161              101000  Bound         (parameter)
05040  162 001100               Multiplier    (parameter)
05046  162         0010         LEN           (syll constant)
05052  162              001000  Bit Length    (parameter)
       163 0
05061  163  1110                ADDR          (syll constant)
05065  163       01101011000    Address       (parameter)
       164 100
05103  164     0110111011000    link
       165 0
05121  165  0010                INTEGER       (type constant)
05125  165       1001           STEPBOUND     (syll constant)
05131  165            010000    Bound         (parameter)
05137  165                 1    Multiplier    (parameter)
       166 10000
```

```
05145  166       1001          STEPBOUND    (syll constant)
05151  166            101000   Bound        (parameter)
05157  166                 0   Multiplier   (parameter)
       167 01100
05165  167       0010          LEN          (syll constant)
05171  167            0010000  Bit Length   (parameter)
05200  168 1110               ADDR         (syll constant)
05204  168     000001110001    Address      (parameter)
       169 00
05222  169    01001001000000   link
05240  170 0010               INTEGER      (type constant)
05244  170      1001          STEPBOUND    (syll constant)
05250  170           110000   Bound        (parameter)
05256  170                 00  Multiplier   (parameter)
       171 1100
05264  171      0010          LEN          (syll constant)
05270  171           0010000  Bit Length   (parameter)
05277  171                 1   ADDR         (syll constant)
       172 110
05303  172     1110011010010   Address      (parameter)
       173 0
05321  173   00011001100000    link
05337  173                 0   INTEGER      (type constant)
       174 010
05343  174     1100            STEP         (syll constant)
05347  174        001111       Multiplier   (parameter)
05355  174              001    LEN          (syll constant)
       175 0
05361  175   0010000           Bit Length   (parameter)
05370  175          1110       ADDR         (syll constant)
05374  175              0001   Address      (parameter)
       176 1101000100
05412  176            011010   link
       177 10100000
05430  177          0010       INTEGER      (type constant)
05434  177              0101   PATCH        (syll constant)
05440  178 000000             Offset       (parameter)
05446  178       111111       Bound 1      (parameter)
05454  178              1101   SEGSPACE     (syll constant)
05460  179 110000             Physical Multiplier (parameter
05466  179       100000       Logical Multiplier (parameter)
05474  179              1101   SEGSPACE     (syll constant)
05500  180 011000             Physical Multiplier (parameter
05506  180       110000       Logical Multiplier (parameter)
05514  180              1101   SEGSPACE     (syll constant)
05520  181 011110             Physical Multiplier (parameter
05526  181       001100       Logical Multiplier (parameter)
05534  181              0010   LEN          (syll constant)
05540  182 0010000            Bit Length   (parameter)
05547  182        0011        SADDR        (syll constant)
05553  182           01101    Stack Pointer (parameter)
       183 011000100
05571  183         0111100    Bound 2      (parameter)
       184 0000000
```

```
05607  184          011100001 link
       185 11000
05625  185       0010         INTEGER     (type constant)
05631  185            0101    PATCH       (syll constant)
05635  185                 110 Offset      (parameter)
       186 000
05643  186     101000          Bound 1     (parameter)
05651  186            1101     SEGSPACE    (syll constant)
05655  186                 110 Physical Multiplier (parameter
       187 000
05663  187     100000          Logical Multiplier (parameter)
05671  187            1101     SEGSPACE    (syll constant)
05675  187                 011 Physical Multiplier (parameter
       188 000
05703  188     110000          Logical Multiplier (parameter)
05711  188            1101     SEGSPACE    (syll constant)
05715  188                 011 Physical Multiplier (parameter
       189 110
05723  189     001100          Logical Multiplier (parameter)
05731  189            0010     LEN         (syll constant)
05735  189                 001 Bit Length  (parameter)
       190 0000
05744  190       0011          SADDR       (syll constant)
05750  190           01101011  Stack Pointer (parameter)
       191 000100
05766  191         0111100000  Bound 2      (parameter)
       192 0000
06004  192      001011011110   link
       193 00
06022  193    0010             INTEGER     (type constant)
06026  193         0010        LEN         (syll constant)
06032  193              001000 Bit Length  (parameter)
       194 0
06041  194   1110               ADDR        (syll constant)
06045  194       01010111000    Address     (parameter)
       195 100
06063  195     0100001000000    link
       196 0
06101  196   11000              MARK        (mark constant)
06106  196         0111101000   link
       197 0000
06124  197      0010            INTEGER     (type constant)
06130  197           0010       LEN         (syll constant)
06134  197                0010  Bit Length  (parameter)
       198 000
06143  198     1010             VALUE       (syll constant)
06147  198         11001        Valuespace
```

D.6 Procedure Example with a Returned Value (i.e. similar to a Function in FORTRAN)

```
PROGRAM
DEFAULT 4;
INTEGER A;
CONSTANT B 4;
PROC C
   BEGIN
      B→A;
      RETURN A+9;
   END
C→A
TERM
```

Dump 14. The Program String

```
00000    0 0011101000000000 PROG
00020    1 0011011000000000 SET'DEFAULT
00040    2 0010000000000000                    4
```

Comment- The following section corresponds to

INTEGER A;

```
00060    3 0001001000000000 NEW'INTEGER
00100    4 1000011000000000 ALLOCATE
```

Comment- The following section corresponds to

CONSTANT B 4;

```
00120    5 0100011000000000 CARD'CONST
00140    6 1100000000000000                    3
00160    7 0010000000000000                    4
```

Comment- The following section corresponds to

PROC C

```
00200    8 1011001000000000 NEW'PROC
00220    9 0000001010000000                  320
00240   10 0001100000000000 ENTER'SCOPE
00260   11 0110100000000000 END'DECL
```

Comment- The following section corresponds to

B→A;

```
00300    12 1110011000000000 ID
00320    13 0000000000000000                    0
00340    14 1000000000000000                    1
00360    15 1110011000000000 ID
00400    16 0000000000000000                    0
00420    17 0000000000000000                    0
00440    18 0111101000000000 ASSIGN
00460    19 0110000000000000 CLEAN'UP
```

Comment- The following section corresponds to

RETURN A+9;

```
00500    20 1110011000000000 ID
00520    21 0000000000000000                    0
00540    22 0000000000000000                    0
00560    23 0100011000000000 CARD'CONST
00600    24 0010000000000000                    4
00620    25 1001000000000000                    9
00640    26 0000011000000000 S''ADD
00660    27 1111011000000000 DUMP              15
00700    28 0011100000000000 EXIT'
00720    29 0110100000000000 END'DECL
00740    30 1111011000000000 DUMP              16
```

Comment- The following section corresponds to

C→A

```
00760    31 0110101000000000 START'LIST
01000    32 1110011000000000 ID
01020    33 0000000000000000                    0
01040    34 0100000000000000                    2
01060    35 1111011000000000 DUMP              17
01100    36 1110011000000000 ID
01120    37 0000000000000000                    0
01140    38 0000000000000000                    0
01160    39 0111101000000000 ASSIGN
01200    40 1111011000000000 DUMP              18
01220    41 0001101000000000 TERMINATE
```

Dump 15. The Workspace as it appears before the Procedural Call in Statement C→A;

```
01240   42 00000000000000   link
01256   42                00 link
        43 111000000000
01274   43              1110 link
        44 1110000000
01312   44             10000 BEGIN        (mark constant)
01317   44                 0 Lexic Level  (parameter)
        45 000
01323   45    000000         Displacement (parameter)
01331   45           0001001 link
        46 0000000
01347   46         0010      INTEGER      (type constant)
01353   46            1010   VALUE        (syll constant)
01357   46                 0 Valuespace
        47 0000
01364   47     011111111000 link
        48 00
01402   48   0100            CARDINAL     (type constant)
01406   48        0010       LEN          (syll constant)
01412   48           110000  Bit Length   (parameter)
        49 0
01421   49  0110             CONSTANT     (syll constant)
01425   49       1110        ADDR         (syll constant)
01431   49          0000111  Address      (parameter)
        50 0000000
01447   50        000111011 link
        51 00000
01465   51       11001       PROC         (mark constant)
01472   51           0000    Current Lexic Level(parameter)
01476   51                00 Program Pointer (parameter)
        52 001001000000
01514   52              0000 link
        53 0000000000
01532   53           11000   MARK         (mark constant)
```

Dump 16. The Workspace as it appears after the Procedure has been Executed but before a Return has been made

```
01240   42 00000000000000   link
01256   42                00 link
        43 111000000000
01274   43              1110 link
        44 1110000000
```

```
01312   44              10000  BEGIN          (mark constant)
01317   44                  0  Lexic Level    (parameter)
        45 000
01323   45     000000          Displacement   (parameter)
01331   45            0001001  link
        46 0000000
01347   46          0010       INTEGER        (type constant)
01353   46               1010  VALUE          (syll constant)
01357   46                  0  Valuespace
        47 0010
01364   47      011111111000   link
        48 00
01402   48    0100             CARDINAL       (type constant)
01406   48         0010        LEN            (syll constant)
01412   48             110000  Bit Length     (parameter)
        49 0
01421   49   0110              CONSTANT       (syll constant)
01425   49       1110          ADDR           (syll constant)
01431   49            0000111  Address        (parameter)
        50 0000000
01447   50          000111011  link
        51 00000
01465   51       11001         PROC           (mark constant)
01472   51             0000    Current Lexic Level(parameter)
01476   51                 00  Program Pointer (parameter)
        52 001001000000
01514   52               0001  link
        53 1110000000
01532   53            11000    MARK           (mark constant)
01537   53                0    link
        54 1111100000000
01555   54              101    LIST           (mark constant)
        55 00
01562   55    11111111000000   link
01600   56 10110               STATE          (mark constant)
01605   56       1000          Lexic Level    (parameter)
01611   56           0000110   Program Pointer (parameter)
        57 0010000
01627   57         000000      Dimension      (parameter)
01635   57              100    Mode           (parameter)
01640   58 11110011000000      link
01656   58                10   BEGIN          (mark constant)
        59 000
01663   59     1000            Lexic Level    (parameter)
01667   59          000000     Displacement   (parameter)
01675   59              000    link
        60 01110000000
01713   60            11000    MARK           (mark constant)
01720   61 10010010000000      link
01736   61                00   INTEGER        (type constant)
        62 10
01742   62    1010             VALUE          (syll constant)
01746   62         01011       Valuespace
```

Dump 17. The Workspace as it appears after the Procedure has been Executed and after the Return has been made

```
01240   42 00000000000000   link
01256   42                00 link
        43 111000000000
01274   43              1110 link
        44 1110000000
01312   44             10000 BEGIN          (mark constant)
01317   44                 0 Lexic Level    (parameter)
        45 000
01323   45    000000         Displacement   (parameter)
01331   45           0001001 link
        46 0000000
01347   46         0010       INTEGER        (type constant)
01353   46             1010   VALUE          (syll constant)
01357   46                 0  Valuespace
        47 0010
01364   47     011111111000   link
        48 00
01402   48   0100             CARDINAL       (type constant)
01406   48       0010         LEN            (syll constant)
01412   48           110000   Bit Length     (parameter)
        49 0
01421   49  0110              CONSTANT       (syll constant)
01425   49      1110          ADDR           (syll constant)
01431   49          0000111   Address        (parameter)
        50 0000000
01447   50        000111011   link
        51 00000
01465   51      11001         PROC           (mark constant)
01472   51           0000     Current Lexic Level(parameter)
01476   51               00   Program Pointer (parameter)
        52 001001000000
01514   52             0001   link
        53 1110000000
01532   53           11000    MARK           (mark constant)
01537   53                1   link
        54 0010010000000
01555   54              001   INTEGER        (type constant)
        55 0
01561   55  1010             VALUE          (syll constant)
01565   55      01011        Valuespace
```

Dump 18. The Workspace as it appears after the Statement C→A; has been Executed

```
01240   42 00000000000000   link
01256   42               00 link
        43 111000000000
01274   43             1110 link
        44 1110000000
01312   44            10000 BEGIN        (mark constant)
01317   44                0 Lexic Level (parameter)
        45 000
01323   45   000000         Displacement (parameter)
01331   45          0001001 link
        46 0000000
01347   46        0010      INTEGER      (type constant)
01353   46           1010   VALUE        (syll constant)
01357   46                0 Valuespace
        47 1011
01364   47    011111111000 link
        48 00
01402   48  0100           CARDINAL     (type constant)
01406   48       0010      LEN          (syll constant)
01412   48          110000 Bit Length   (parameter)
        49 0
01421   49 0110            CONSTANT     (syll constant)
01425   49      1110       ADDR         (syll constant)
01431   49         0000111 Address      (parameter)
        50 0000000
01447   50        000111011 link
        51 00000
01465   51      11001       PROC         (mark constant)
01472   51          0000    Current Lexic Level(parameter)
01476   51               00 Program Pointer (parameter)
        52 001001000000
01514   52             0001 link
        53 1110000000
01532   53          11000   MARK         (mark constant)
01537   53                0 link
        54 1101100000000
01555   54              001 INTEGER      (type constant)
        55 0
01561   55 1010            VALUE        (syll constant)
01565   55      01011       Valuespace
```

D.7 Procedure Example with Parameters

```
PROGRAM
DEFAULT 4;
INTEGER A;
CONSTANT B 4;
PROC C(P1)
   BEGIN
      9+B→P1;
      2+P1→A;
   END
INTEGER D;
C(D)
TERM
```

Dump 19. The Program String

```
00000    0 0011101000000000 PROG
00020    1 0011011000000000 SET'DEFAULT
00040    2 0010000000000000                    4
```

Comment- The following section corresponds to

INTEGER A;

```
00060    3 0001001000000000 NEW'INTEGER
00100    4 1000011000000000 ALLOCATE
```

Comment- The following section corresponds to

CONSTANT B 4;

```
00120    5 0100011000000000 CARD'CONST
00140    6 1100000000000000                    3
00160    7 0010000000000000                    4
```

Comment- The following section corresponds to

PROC C(P1)

```
00200    8 1011001000000000 NEW'PROC
00220    9 0000001110000000                  448
00240   10 0001100000000000 ENTER'SCOPE
00260   11 0110100000000000 END'DECL
```

```
        Comment- The following section corresponds to

                                9+B>P1;

00300    12 0100011000000000 CARD'CONST
00320    13 0010000000000000                    4
00340    14 1001000000000000                    9
00360    15 1110011000000000 ID
00400    16 0000000000000000                    0
00420    17 1000000000000000                    1
00440    18 0000011000000000 S''ADD
00460    19 1111100000000000 FORMAL'ID
00500    20 1000000000000000                    1
00520    21 0000000000000000                    0
00540    22 0111101000000000 ASSIGN
00560    23 0110000000000000 CLEAN'UP

        Comment- The following section corresponds to

                                2+P1>A;

00600    24 0100011000000000 CARD'CONST
00620    25 0100000000000000                    2
00640    26 0100000000000000                    2
00660    27 1111100000000000 FORMAL'ID
00700    28 1000000000000000                    1
00720    29 0000000000000000                    0
00740    30 0000011000000000 S''ADD
00760    31 1110011000000000 ID
01000    32 0000000000000000                    0
01020    33 0000000000000000                    0
01040    34 0111101000000000 ASSIGN
01060    35 1111011000000000 DUMP              20
01100    36 0011100000000000 EXIT'

        Comment- The following section corresponds to

                                INTEGER D;

01120    37 0001001000000000 NEW'INTEGER
01140    38 0110100000000000 END'DECL
01160    39 1111011000000000 DUMP              21

        Comment- The following section corresponds to

                                C(D)

01200    40 0110101000000000 START'LIST
01220    41 1110011000000000 ID
01240    42 0000000000000000                    0
01260    43 1100000000000000                    3
```

```
01300    44 1101001000000000 NEW'NULL
01320    45 1111011000000000 DUMP              22
01340    46 1110011000000000 ID
01360    47 0000000000000000                    0
01400    48 0100000000000000                    2
01420    49 1111011000000000 DUMP              23
01440    50 0001101000000000 TERMINATE
```

Dump 20. The Workspace as it appears before the Procedural Call C(D)

```
01460    51 00000000000000   link
01476    51               00 link
         52 111110000000
01514    52             1110 link
         53 1010000000
01532    53          10000   BEGIN       (mark constant)
01537    53              0   Lexic Level (parameter)
         54 000
01543    54   000000         Displacement (parameter)
01551    54         0001001  link
         55 1000000
```

Comment- The following section corresponds to

INTEGER A;

```
01567    55        0010      INTEGER     (type constant)
01573    55           1010   VALUE       (syll constant)
01577    55              0   Valuespace
         56 0000
01604    56    011110110000 link
         57 00
```

Comment- The following section corresponds to

CONSTANT B 4;

```
01622    57   0100           CARDINAL    (type constant)
01626    57       0010       LEN         (syll constant)
01632    57          110000  Bit Length  (parameter)
         58 0
01641    58  0110            CONSTANT    (syll constant)
01645    58      1110        ADDR        (syll constant)
01651    58          0000111 Address     (parameter)
         59 0000000
01667    59        000110100 link
```

```
          60 00000
```

Comment- The following section corresponds to

PROC C(P1)

```
01705     60       11001        PROC          (mark constant)
01712     60            0000    Current Lexic Level(parameter)
01716     60                 00 Program Pointer (parameter)
          61 001001000000
01734     61               0000 link
          62 0010000000
```

Comment- The following section corresponds to

INTEGER D;

```
01752     62            0010    INTEGER       (type constant)
01756     62              10    VALUE         (syll constant)
          63 10
01762     63   00000            Valuespace
01767     63        000000000   link
          64 00000
02005     64       11000        MARK          (mark constant)
```

Dump 21. The Workspace with the Parameter D of the Procedural Call C(D) in the Program Execution Stack

```
01460     51 00000000000000    link
01476     51                00 link
          52 111110000000
01514     52              1110 link
          53 1010000000
01532     53           10000   BEGIN         (mark constant)
01537     53               0   Lexic Level   (parameter)
          54 000
01543     54    000000         Displacement  (parameter)
01551     54          0001001  link
          55 1000000
01567     55        0010       INTEGER       (type constant)
01573     55           1010    VALUE         (syll constant)
01577     55               0   Valuespace
          56 0000
01604     56     011110110000  link
          57 00
01622     57   0100             CARDINAL      (type constant)
01626     57       0010         LEN           (syll constant)
```

```
01632   57              110000 Bit Length  (parameter)
        58 0
01641   58  0110               CONSTANT    (syll constant)
01645   58      1110           ADDR        (syll constant)
01651   58           0000111   Address     (parameter)
        59 0000000
01667   59         000110100   link
        60 00000
01705   60       11001         PROC        (mark constant)
01712   60            0000     Current Lexic Level(parameter)
01716   60                00   Program Pointer (parameter)
        61 001001000000
01734   61              0000   link
        62 0010000000
01752   62            0010     INTEGER     (type constant)
01756   62                10   VALUE       (syll constant)
        63 10
01762   63   00000             Valuespace
01767   63         011010111   link
        64 11000
02005   64       11000         MARK        (mark constant)
02012   64            010101   link
        65 11111000
02030   65          10100      LIST        (mark constant)
02035   65              110    link
        66 10010000000
02053   66            0010     INTEGER     (type constant)
02057   66                1    ADDR        (syll constant)
        67 110
02063   67    0100111111000    Address     (parameter)
        68 0
02101   68  00000000000000     link
02117   68                0    NULL        (type constant)
        69 001
```

Dump 22. The Workspace as it appears after the Procedure has been Executed but before a Return is made

```
01460   51 00000000000000    link
01476   51                00 link
        52 111110000000
01514   52              1110 link
        53 1010000000
01532   53            10000  BEGIN       (mark constant)
01537   53                0  Lexic Level (parameter)
        54 000
01543   54    000000         Displacement (parameter)
01551   54           0001001 link
        55 1000000
```

```
01567   55          0010        INTEGER      (type constant)
01573   55              1010    VALUE        (syll constant)
01577   55                  0   Valuespace
        56 1111
01604   56      011110110000    link
        57 00
01622   57    0100              CARDINAL     (type constant)
01626   57         0010         LEN          (syll constant)
01632   57             110000   Bit Length   (parameter)
        58 0
01641   58  0110                CONSTANT     (syll constant)
01645   58       1110           ADDR         (syll constant)
01651   58            0000111   Address      (parameter)
        59 0000000
01667   59          000110100   link
        60 00000
01705   60       11001          PROC         (mark constant)
01712   60             0000     Current Lexic Level(parameter)
01716   60                 00   Program Pointer (parameter)
        61 001001000000
01734   61               0000   link
        62 0010000000
01752   62             0010     INTEGER      (type constant)
01756   62                10    VALUE        (syll constant)
        63 10
01762   63   01011              Valuespace
01767   63         011010111    link
        64 11000
02005   64       11000          MARK         (mark constant)
02012   64             010101   link
        65 11111000
02030   65          10100       LIST         (mark constant)
02035   65               110    link
        66 10010000000
02053   66             0010     INTEGER      (type constant)
02057   66                1     ADDR         (syll constant)
        67 110
02063   67     0100111111000    Address      (parameter)
        68 0
02101   68  01110010000000      link
02117   68                  0   NULL         (type constant)
        69 001
02123   69     0000001100000    link
        70 0
02141   70  10110               STATE        (mark constant)
02146   70        1000          Lexic Level  (parameter)
02152   70             000010   Program Pointer (parameter)
        71 00110000
02170   71          000000      Dimension    (parameter)
02176   71                10    Mode         (parameter)
        72 0
02201   72  101100011000000     link
02217   72                  1   BEGIN        (mark constant)
        73 0000
```

```
02224   73      1000            Lexic Level (parameter)
02230   73           000000     Displacement (parameter)
02236   73                 00 link
        74 001100000000
02254   74               1100 MARK          (mark constant)
        75 0
02261   75  01001010000000  link
02277   75                 0 INTEGER       (type constant)
        76 010
02303   76     1010          VALUE          (syll constant)
02307   76         01111     Valuespace
```

Dump 23. The Workspace as it appears after the Procedure has been Executed and after the Return has been made

```
01460   51 00000000000000   link
01476   51                00 link
        52 111110000000
01514   52              1110 link
        53 1010000000
01532   53             10000 BEGIN         (mark constant)
01537   53                 0 Lexic Level   (parameter)
        54 000
01543   54    000000         Displacement  (parameter)
01551   54           0001001 link
        55 1000000
01567   55         0010       INTEGER       (type constant)
01573   55             1010   VALUE         (syll constant)
01577   55                 0 Valuespace
        56 1111
01604   56     011110110000 link
        57 00
01622   57   0100             CARDINAL      (type constant)
01626   57        0010        LEN           (syll constant)
01632   57           110000   Bit Length    (parameter)
        58 0
01641   58  0110              CONSTANT      (syll constant)
01645   58      1110          ADDR          (syll constant)
01651   58          0000111   Address       (parameter)
        59 0000000
01667   59        000110100 link
        60 00000
01705   60       11001        PROC          (mark constant)
01712   60           0000     Current Lexic Level(parameter)
01716   60                00  Program Pointer (parameter)
        61 001001000000
01734   61              0000 link
        62 0010000000
```

```
01752   62            0010    INTEGER      (type constant)
01756   62                10  VALUE        (syll constant)
        63 10
01762   63   01011            Valuespace
01767   63         011010111  link
        64 11000
02005   64       11000        MARK         (mark constant)
02012   64             010010 link
        65 10000000
02030   65          0010      INTEGER      (type constant)
02034   65              1010  VALUE        (syll constant)
02040   66 01111              Valuespace
```

D.8 Example of the Monadic Soft Extension

This example shows the monadic soft extension as far as it has been debugged. The monadic soft extension is trapped to whenever a monadic function is used on a nonscalar.

```
PROGRAM PROCESSOR REF INDEX TO 10;
OP(X)   "MONADIC PATTERN"
BEGIN
        DECL R (TYPE X, DIM X, LEN X);
        REF A TO RAVEL X;
        REF B TO RAVEL R;
        REF J TO NENT X;
        CARDINAL I BITS INDEX;
        DO  ?A[I]→B[I] UNTIL 1+→I=J;
        R
END;
DEFAULT 4;
INTEGER A[3,4];
ABS(A)
TERM
```

Dump 24. The Program String

Comment- The following section corresponds to

PROGRAM PROCESSOR REF INDEX TO 10;

```
00000    0 0011101000000000 PROG
00020    1 0100011000000000 CARD'CONST
00040    2 0010000000000000                    4
00060    3 0101000000000000                   10
```

The S-UNITS in words 4 to 81 correspond to the Monadic Soft Extension

Comment- The following section corresponds to

OP(X) "MONADIC PATTERN"

```
00100    4 0011001000000000 NEW'OPERATOR
00120    5 0000001100100000                 1216
```

Comment- The following section corresponds to

DECL R (TYPE X, DIM X, LEN X);

```
00140    6 1000011000000000 ALLOCATE
00160    7 1111100000000000 FORMAL'ID
00200    8 1000000000000000                    1
00220    9 0000000000000000                    0
00240   10 1111011000000000 DUMP              28
00260   11 1101110000000000 S''TYPE
00300   12 1111011000000000 DUMP              29
00320   13 1000100000000000 DYNA'TYPE
00340   14 1111011000000000 DUMP              30
00360   15 1111100000000000 FORMAL'ID
00400   16 1000000000000000                    1
00420   17 0000000000000000                    0
00440   18 0011000000000000 DIMENSION
00460   19 0000100000000000 DYNA'DIM
00500   20 1111100000000000 FORMAL'ID
00520   21 1000000000000000                    1
00540   22 0000000000000000                    0
00560   23 0000001000000000 LENGTH
00600   24 0010011000000000 DYNA'LEN
```

```
        Comment- The following section corresponds to

                        REF A TO RAVEL X;

00620    25 1000011000000000 ALLOCATE
00640    26 1111100000000000 FORMAL'ID
00660    27 1000000000000000                    1
00700    28 0000000000000000                    0
00720    29 0100101000000000 RAVEL

        Comment- The following section corresponds to

                        REF B TO RAVEL R;

00740    30 1000011000000000 ALLOCATE
00760    31 1110011000000000 ID
01000    32 1000000000000000                    1
01020    33 0000000000000000                    0
01040    34 0100101000000000 RAVEL

        Comment- The following section corresponds to

                        REF J TO NENT X;

01060    35 1000011000000000 ALLOCATE
01100    36 1111100000000000 FORMAL'ID
01120    37 1000000000000000                    1
01140    38 0000000000000000                    0
01160    39 0010001000000000 NENT

        Comment- The following section corresponds to

                        CARDINAL I BITS INDEX;

01200    40 0110001000000000 NEW'CARDINAL
01220    41 1110011000000000 ID
01240    42 0000000000000000                    0
01260    43 0000000000000000                    0
01300    44 0010011000000000 DYNA'LEN
01320    45 0110100000000000 END'DECL

        Comment- The following section corresponds to

                    DO ?A[I]>B[I] UNTIL 1+>I=J;

01340    46 0110101000000000 START'LIST
01360    47 1110011000000000 ID
01400    48 1000000000000000                    1
01420    49 0010000000000000                    4
01440    50 1110101000000000 SUBSCRIPT'MODE
```

```
01460    51 1110011000000000 ID
01500    52 1000000000000000                    1
01520    53 1000000000000000                    1
01540    54 0111100000000000 FAULT'OP
01560    55 0110101000000000 START'LIST
01600    56 1110011000000000 ID
01620    57 1000000000000000                    1
01640    58 0010000000000000                    4
01660    59 1110101000000000 SUBSCRIPT'MODE
01700    60 1110011000000000 ID
01720    61 0100000000000000                    2
01740    62 0111101000000000 ASSIGN
01760    63 0100011000000000 CARD'CONST
02000    64 1000000000000000                    1
02020    65 1000000000000000                    1
02040    66 1110011000000000 ID
02060    67 1000000000000000                    1
02100    68 0010000000000000                    4
02120    69 1100010000000000 S''INCREMENT
02140    70 1110011000000000 ID
02160    71 1000000000000000                    1
02200    72 1100000000000000                    3
02220    73 1001100000000000 S''EQUAL
02240    74 0111000000000000 DO'UNTIL
02260    75 1111100000000000                   31
02300    76 0110000000000000 CLEAN'UP
```

Comment- The following section corresponds to

R

```
02320    77 1110011000000000 ID
02340    78 1000000000000000                    1
02360    79 0000000000000000                    0
02400    80 1111101000000000 COPY'VALUE
02420    81 0100001000000000 MONADIC'EXIT
02440    82 1111011000000000 DUMP              25
```

Comment- The following section corresponds to

DEFAULT 4;

```
02460    83 0011011000000000 SET'DEFAULT
02500    84 0010000000000000                    4
```

Comment- The following section corresponds to

INTEGER A[3,4];

```
02520   85 0001001000000000 NEW'INTEGER
02540   86 0100011000000000 CARD'CONST
02560   87 0100000000000000                    2
02600   88 1100000000000000                    3
02620   89 0000100000000000 DYNA'DIM
02640   90 0100011000000000 CARD'CONST
02660   91 1100000000000000                    3
02700   92 0010000000000000                    4
02720   93 0000100000000000 DYNA'DIM
02740   94 0110100000000000 END'DECL
02760   95 1111011000000000 DUMP              26
```

The following S-UNIT (i.e. START'LIST) is added to get the soft extension to work but is only a temporary fix.

```
03000   96 0110101000000000 START'LIST
```

Comment- The following section corresponds to

ABS(A)

```
03020   97 1110011000000000 ID
03040   98 0000000000000000                    0
03060   99 0100000000000000                    2
03100  100 1101001000000000 NEW'NULL
03120  101 1000000000000000 S''ABS
03140  102 1111011000000000 DUMP              31
03160  103 0001101000000000 TERMINATE
```

Dump 25. Dump showing the Monadic Soft Extension Declaration

```
03200  104 000000000000000   link
03216  104                00 link
       105 111000000000
03234  105              1110 link
       106 1100000000
03252  106             10000 BEGIN       (mark constant)
03257  106                 0 Lexic Level (parameter)
       107 000
03263  107    000000         Displacement (parameter)
03271  107           0000111 link
       108 0000000
```

```
03307  108          0100        CARDINAL     (type constant)
03313  108               0010   LEN          (syll constant)
03317  108                    0 Bit Length   (parameter)
       109 010000
03326  109         0110         CONSTANT     (syll constant)
03332  109              1110    ADDR         (syll constant)
03336  109                   00 Address      (parameter)
       110 001100000000
03354  110                 0000 link
       111 0000000000
03372  111              10101   OPERATOR     (mark constant)
03377  111                    0 Current Lexic Level (parameter
       112 000
03403  112     0000011000000   Program Pointer (parameter)
       113 0
```

Dump 26. Dump showing the Declarations on the Declarations Stack after the END'DECL at word 94 was reached

```
03200  104 00000000000000   link
03216  104                00 link
       105 111000000000
03234  105              1110 link
       106 1100000000
03252  106             10000 BEGIN        (mark constant)
03257  106                 0 Lexic Level  (parameter)
       107 000
03263  107    000000         Displacement (parameter)
03271  107           0000111 link
       108 0000000
03307  108         0100       CARDINAL     (type constant)
03313  108              0010  LEN          (syll constant)
03317  108                 0  Bit Length   (parameter)
       109 010000
03326  109         0110       CONSTANT     (syll constant)
03332  109              1110  ADDR         (syll constant)
03336  109                00  Address      (parameter)
       110 001100000000
03354  110                0001 link
       111 0101100000
03372  111             10101   OPERATOR     (mark constant)
03377  111                 0   Current Lexic Level (parameter
       112 000
03403  112     0000011000000   Program Pointer (parameter)
       113 0
03421  113  1101100110000000   link
03437  113                 0   INTEGER      (type constant)
       114 010
03443  114     1100           STEP         (syll constant)
```

```
03447  114          110000     Multiplier  (parameter)
03455  114                110 STEP        (syll constant)
       115 0
03461  115  001100             Multiplier  (parameter)
03467  115          1010       VALUE       (syll constant)
03473  115               00000 Valuespace
       116 0000000000000000
       117 0000000000000000
       118 0000000000000000
       119 0000000
03567  119          000000000 link
       120 00000
03605  120        11000        MARK        (mark constant)
```

Dump 27. The workspace as it appears after a trap is made to the Soft Extension. The trap occurs while executing ABS(A) because this is a reference to a nonscalar. The last part of the dump shows the result of executing the FORMAL'ID at word 7

```
03200  104 00000000000000    link
03216  104                00 link
       105 111000000000
03234  105              1110 link
       106 1100000000
03252  106           10000   BEGIN       (mark constant)
03257  106               0   Lexic Level (parameter)
       107 000
03263  107    000000         Displacement (parameter)
03271  107           0000111 link
       108 0000000
03307  108         0100       CARDINAL    (type constant)
03313  108            0010    LEN         (syll constant)
03317  108               0    Bit Length  (parameter)
       109 010000
03326  109        0110        CONSTANT    (syll constant)
03332  109           1110     ADDR        (syll constant)
03336  109              00    Address     (parameter)
       110 001100000000
03354  110            0001    link
       111 0101100000
03372  111           10101    OPERATOR    (mark constant)
03377  111               0    Current Lexic Level (parameter
       112 000
03403  112    0000011000000   Program Pointer (parameter)
       113 0
03421  113  11011001100000    link
03437  113               0    INTEGER     (type constant)
       114 010
```

```
03443  114     1100          STEP        (syll constant)
03447  114          110000   Multiplier  (parameter)
03455  114              110  STEP        (syll constant)
       115 0
03461  115  001100          Multiplier  (parameter)
03467  115        1010       VALUE       (syll constant)
03473  115             00000 Valuespace
       116 0000000000000000
       117 0000000000000000
       118 0000000000000000
       119 0000000
03567  119        110110010  link
       120 00000
03605  120      11000        MARK        (mark constant)
03612  120           010101  link
       121 11000000
```

The following LIST is necessary but was placed on the stack by the START'LIST at word 96. This is a temporary fix until the routine that is suppose to put LIST on the stack is determined and fixed.

```
03630  121         10100     LIST        (mark constant)
```

The following section is the result of executing the ID (word 97) corresponding to the variable A in the statement ABS(A).

```
03635  121              111  link
       122 11010000000
03653  122             0010  INTEGER     (type constant)
03657  122                1  STEP        (syll constant)
       123 100
03663  123    110000         Multiplier  (parameter)
03671  123          1100     STEP        (syll constant)
03675  123              001  Multiplier  (parameter)
       124 100
03703  124    1110           ADDR        (syll constant)
03707  124        110111001  Address     (parameter)
       125 11000
03725  125      01011110000  link
       126 000
03743  126    0001           NULL        (type constant)
03747  126        100100111  link
```

The following was the result of trapping to the monadic soft extension.

```
       127 11100
03765  127      11110        STATEOP     (mark constant)
03772  127           1000    Lexic Level (parameter)
```

```
03776  127                  00 Program Pointer (parameter)
       128 000110011000
04014  128               0000 Dimension   (parameter)
       129 00
04022  129   100             Mode        (parameter)
04025  129       1000000     Decoder     (parameter)
04034  129               0000 link
       130 0000000000
```

The following is the result of executing the FORMAL'ID at word 7 after trapping to the soft extension.

```
04052  130            0010   INTEGER     (type constant)
04056  130                11 STEP        (syll constant)
       131 00
04062  131   110000          Multiplier  (parameter)
04070  131          1100     STEP        (syll constant)
04074  131               0011 Multiplier  (parameter)
       132 00
04102  132   1110            ADDR        (syll constant)
04106  132        1101110011 Address     (parameter)
       133 1000
```

Dump 28. The workspace after executing S''TYPE at word 11. S''TYPE determines the type of the last statement operand (i.e. located at bit address 4072 octal in Dump 28)

```
03200  104 00000000000000   link
03216  104                00 link
       105 111000000000
03234  105              1110 link
       106 1100000000
03252  106           10000   BEGIN       (mark constant)
03257  106               0   Lexic Level (parameter)
       107 000
03263  107    000000         Displacement (parameter)
03271  107          0000111 link
       108 0000000
03307  108        0100       CARDINAL    (type constant)
03313  108             0010  LEN         (syll constant)
03317  108               0   Bit Length  (parameter)
       109 010000
03326  109       0110        CONSTANT    (syll constant)
03332  109           1110    ADDR        (syll constant)
03336  109              00   Address     (parameter)
       110 001100000000
03354  110            0001   link
       111 0101100000
03372  111           10101   OPERATOR    (mark constant)
```

```
03377  111                  0 Current Lexic Level (parameter
       112 000
03403  112     0000011000000 Program Pointer (parameter)
       113 0
03421  113   11011001100000  link
03437  113                  0 INTEGER      (type constant)
       114 010
03443  114     1100           STEP         (syll constant)
03447  114          110000    Multiplier   (parameter)
03455  114                110 STEP         (syll constant)
       115 0
03461  115   001100           Multiplier   (parameter)
03467  115         1010       VALUE        (syll constant)
03473  115              00000 Valuespace
       116 0000000000000000
       117 0000000000000000
       118 0000000000000000
       119 0000000
03567  119         110110010  link
       120 00000
03605  120       11000        MARK         (mark constant)
03612  120             010101 link
       121 11000000
03630  121          10100     LIST         (mark constant)
03635  121                111 link
       122 11010000000
03653  122              0010  INTEGER      (type constant)
03657  122                 1  STEP         (syll constant)
       123 100
03663  123     110000          Multiplier   (parameter)
03671  123           1100      STEP         (syll constant)
03675  123               001   Multiplier   (parameter)
       124 100
03703  124     1110            ADDR         (syll constant)
03707  124         110111001   Address      (parameter)
       125 11000
03725  125       01011110000   link
       126 000
03743  126     0001            NULL         (type constant)
03747  126         100100111   link
       127 11100
03765  127       11110         STATEOP      (mark constant)
03772  127            1000     Lexic Level  (parameter)
03776  127                00   Program Pointer (parameter)
       128 000110011000
04014  128              0000   Dimension    (parameter)
       129 00
04022  129    100              Mode         (parameter)
04025  129       1000000       Decoder      (parameter)
04034  129              0000   link
       130 0000000000
```

Comment- The following section corresponds to

the CARDINAL statement operand created by S''TYPE

```
04052  130            0100   CARDINAL     (type constant)
04056  130              00 LEN           (syll constant)
       131 10
04062  131   0111000         Bit Length   (parameter)
04071  131          1010     VALUE        (syll constant)
04075  131             110 Valuespace
       132 00000000000
```

Dump 29. The Workspace as it appears after Executing the S-UNIT DYNA'DIM at word 13

```
03200  104 00000000000000   link
03216  104               00 link
       105 111000000000
03234  105             1110 link
       106 1100000000
03252  106          10000   BEGIN        (mark constant)
03257  106              0 Lexic Level    (parameter)
       107 000
03263  107    000000        Displacement (parameter)
03271  107         0000111  link
       108 0000000
03307  108       0100       CARDINAL     (type constant)
03313  108          0010    LEN          (syll constant)
03317  108              0 Bit Length     (parameter)
       109 010000
03326  109      0110        CONSTANT     (syll constant)
03332  109          1110    ADDR         (syll constant)
03336  109              00 Address       (parameter)
       110 001100000000
03354  110              0001 link
       111 0101100000
03372  111          10101   OPERATOR     (mark constant)
03377  111              0 Current Lexic Level (parameter
       112 000
03403  112    0000011000000 Program Pointer (parameter)
       113 0
03421  113  11011001100000  link
03437  113              0 INTEGER        (type constant)
       114 010
03443  114    1100          STEP         (syll constant)
03447  114        110000    Multiplier   (parameter)
03455  114             110  STEP         (syll constant)
       115 0
03461  115  001100          Multiplier   (parameter)
```

```
03467  115          1010      VALUE        (syll constant)
03473  115               00000 Valuespace
       116 0000000000000000
       117 0000000000000000
       118 0000000000000000
       119 0000000
03567  119         110110010 link
       120 00000
03605  120       11000       MARK         (mark constant)
03612  120             010101 link
       121 11000000
03630  121           10100   LIST         (mark constant)
03635  121                111 link
       122 11010000000
03653  122              0010 INTEGER      (type constant)
03657  122                 1 STEP         (syll constant)
       123 100
03663  123     110000         Multiplier   (parameter)
03671  123           1100     STEP         (syll constant)
03675  123               001  Multiplier   (parameter)
       124 100
03703  124     1110           ADDR         (syll constant)
03707  124         110111001  Address      (parameter)
       125 11000
03725  125       01011110000  link
       126 000
03743  126     0001           NULL         (type constant)
03747  126         100100111  link
       127 11100
03765  127       11110        STATEOP      (mark constant)
03772  127            1000    Lexic Level  (parameter)
03776  127                00  Program Pointer (parameter)
       128 000110011000
04014  128              0000  Dimension    (parameter)
       129 00
04022  129    100             Mode         (parameter)
04025  129       1000000      Decoder      (parameter)
04034  129              0000  link
       130 0000000000
```

The following INTEGER results from executing DYNA'DIM which uses the CARDINAL operand bit address 4052 octal in the previous dump to determine the typeconstant to put on the stack.

```
04052  130           0010     INTEGER      (type constant)
```

References

1. Chu Y. "Concepts of High-Level-Language Computer Architecture," Proceedings of the 1975 ACM Conference: 6-13 (1975).

2. Grebert, Alain P., Frederick H. Gerbstadt, Paul Colen. Space Programming Language Machine Architecture Study, I: SAMSO TR 72-117. Los Angelos, California: Space and Missile System Organization, May 1972. (AD 743 014).

3. Grebert, Alain P., Frederick H. Gerbstadt, Paul Colen. Space Programming Language Machine Architecture Study, II: SAMSO TR 72-117. Los Angelos, California: Space and Missile System Organization, May 1972. (AD 743 015).

4. Grebert, Alain P., et al. Space Progaramming Language Machine (SPLM) Logic Simulator, I: SAMSO TR 74-26. Los Angelos, California: Space and Missle Systems Organization, March 1974. (AD/A 002 798).

5. Grebert, Alain P., et al. Space Progaramming Language Machine (SPLM) Logic Simulator, II: SAMSO TR 74-26. Los Angelos, California: Space and Missle Systems Organization, March 1974. (AD/A 002 799).

6. Grebert, Alain P., et al. Space Progaramming Language Machine (SPLM) Logic Simulator, III: SAMSO TR 74-26. Los Angelos, California: Space and Missle Systems Organization, March 1974. (AD/A 002 800).

7. Hara, Jeff and W. Lynn Trainer. JOVIAL J73/1 Computer Programming Manual. Wright-Patterson AFB, Ohio:Air Force Avionics Laboratory, January 1976.

8. Kerner, H. and L. Gellman, "Memory Reduction Through Higher Level Language Hardware," AIAA Journal, Vol 10 - No. 5 (November 1970).

9. Lee, Theodore M. P. Technical Review of "Space Programming Language Architecture Study". Speery Univac Defense Systems Division, St. Paul Minnesota, August 1972.

10. Nielsen, William C. Aerospace HOL Computer, I: AFAL-TR-72-292-Vol. I. Wright-Patterson AFB, Ohio: Air Force Avionics Laboratory, October 1972.

11. Nielsen, William C. and Steven A. Vere, Aerospace HOL Computer, Volume II (Direct Execution Tradeoffs): AFAL-TR-72-292-Vol. II. Wright-Patterson AFB, Ohio: Air Force Avionics Laboratory, October 1972.

12. Robinet, Bernard J. Architectural Design of a Directly Executed APL Processor. University of Maryland, August 1974. (NTIS PB-235 775)

13. Rosen, S. "Hardware Design Reflecting Software Requirements," Fall Joint Computer Conference, 1975 : 1443-1449 (1975).

VITA

David L. Akin was born on September 30,1952 in Englewood, Colorado. He graduated from Englewood High School in 1970 and attended Colorado State Univesity in Fort Collins, Colorado. In March 1975 he received a Bachelor of Science degree in Electrical Engineering and a commission in the United States Air Force through the ROTC program. Immediately following graduation he was employed by the Atmospheric Science Department at Colorado State University as an electronic technician until August 1976 when he enrolled in the Graduate Computer Science program at the Air Force Institute of Technology.

Permanent Address:
4530 S. Huron
Englewood, Colo. 80110

VITA

Brian C. Allen was born on May 3, 1954 in Atlanta, Georgia. He graduated from Good Counsel High School in Wheaton Maryland in 1972. He received a Bachelor of Science degree in Math/Computer Science in May 1976 from Tulane University and was a cum laude graduate. Upon graduation, he received a commission in the USAF through the ROTC program and was selected as a distinguished graduate. In August 1976 he enrolled in the Graduate Computer Science program at the Air Force Institute of Technology. He is a member of Eta Kappa Nu and Tau Beta Pi.

Permanent Address:
326 Fairway Dr.
New Orleans, La. 70124

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/78-1	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* SIMULATION OF THE DIRECT EXECUTION OF A HIGHER ORDER LANGUAGE		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David L. Akin 1st Lt USAF Brian C. Allen 2nd Lt USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory (AFAL) Wright-Patterson AFB, Ohio 45433		12. REPORT DATE March 1978 13. NUMBER OF PAGES 228
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

JERRAL F. GUESS, Captain, USAF
Director of Information

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Space Programming Language Machine
Direct Execution Machine
Computer
Simulation
Higher Order Language

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This document lists the results of a follow-on study of Phase II of the Space Programming Language Machine (SPLM) Architecture Study. In this follow-on study, the Clock Level Simulator (CLS) was translated to JOVIAL/J73 and improvements were made to decrease execution times and memory requirements of the simulator. This work was accomplished using the DEC-10 system of the Air Force Avionics Laboratory at Wright-Patterson AFB.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

CLS simulates the direct execution of SPLML (Space Programming Language Machine Language) on a SPLM. SPLML is a higher order language that was developed especially for avionic applications during the Phase I study.

The initialization routine was written as an interactive program used to enter design parameters and is run separately from the rest of the simulator. A dump routine was developed which parses the simulated memory for ease of debugging the simulator and application programs.

This document was developed to provide a starting point for follow-on studies, and the examples should be helpful in understanding CLS. The example program executions and recommendations for follow-on studies are given in this document. The CLS code can be obtained in hard copy form or on tape through the AFIT School of Engineering.