ON-LINE SPECTRE PROGRAMMING SYSTEM

A Thesis

Presented to

The Faculty of Graduate Studies and Research

The University of Manitoba

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in Computer Science

---

by

Ted Kasloff

October, 1971

ABSTRACT

This work describes the philosophy, implementation, and usage of the On-line SPECTRE Programming System (OSPS)--a conversational program which facilitates creation of a SPECTRE MAP program from a terminal, assembly of the SPECTRE MAP program, and execution of the SPECTRE machine program which has been generated.

ACKNOWLEDGEMENTS

I wish to express my thanks to Dr. C.I. Abraham, my supervisor, whose inspiration and guidance were essential to me in the preparation of this thesis.

In addition, I would like to thank Professor R.B. Pinkney and Professor J.M. Wells for their useful comments on, and criticisms of this manuscript.

PREFACE

The present work describes the philosophy, implementation, and the usage of a program which simulates the SPECTRE computer (1).

The hypothetical computer SPECTRE was first introduced at the University of Waterloo in 1965, and in 1966, major changes were made in its original design. The design of the SPECTRE computer is simple and also typical of most stored-program digital computers. For these reasons the study of SPECTRE is considered useful in the sense that it would introduce the student to the basic concepts of computer structure and techniques of programming.

To the author's knowledge, all simulation programs written for SPECTRE up to the present time have been restricted to operating in a batch environment. This seemed to hinder the speed and ease of learning SPECTRE because of the relatively slow turnaround time and inadequate debugging facilities provided. As a result, the student became rather discouraged with what could have otherwise been an easier job if overall improvements in these areas were made.

Apart from its use as a teaching aid, an on-line SPECTRE facility could be used as a programmable desk calculator due to the feature of on-line input-output.

Therefore, with these objectives in mind, it was considered desirable to develop the program OSPS (the On-line SPECTRE Programming System), which enables SPECTRE MAP programs (1) to be edited, assembled, and executed in an on-line environment under MUM (the Manitoba University Monitor (2)) on an IBM 360/65 computer.

In order to improve upon the facilities which were previously available to the batch SPECTRE MAP programmer, OSPS was designed with the following features in mind:

a. Free format for SPECTRE MAP statements.

b. The programmer could create and edit a SPECTRE MAP program from a terminal as though he were creating a card deck.

c. The programmer could submit his program for assembly from the terminal. If errors occurred he would be immediately informed, allowed to make corrections, and could submit his program for re-assembly.

d. After successful assembly, the programmer could submit his program for execution.

e. Interactive communication between programmer at the terminal and his program in execution - The programmer would have the option of specifying the number of statements to be executed before control was to be returned to him. However, OSPS would have

built in a feature to guard against unintentional (in the case of infinite loops) or intentional monopolization of the CPU. When the programmer received control he could monitor his program by requesting a memory dump or register display.

f. The programmer, when in control, could request re-execution of his program. In addition, it would not be necessary that the programmer submit his SPECTRE MAP program for re-assembly.

g. During program execution, if an error should occur the programmer would be immediately informed of the nature of the error. At this point he could either have his program re-executed without re-assembly, or make changes to his SPECTRE MAP program and submit it for re-assembly.

h. The programmer would have the facility of on-line input-output to his program while it is executing. The fixed format of input data which was required previously in a batch environment would be removed. The conversion of input data from external to internal representation would be performed by a data interpreter and would not be of concern to the programmer.

i. When the programmer has control he would have the

option of ending communication with OSPS.

In developing OSPS some modifications were made to the design of SPECTRE in order to reflect its on-line facilities. Also, the formal definition of the SPECTRE MAP assembly language was slightly modified in order to facilitate the free-format for SPECTRE MAP statements.

In Chapter 1 the design of the 'new' SPECTRE computer is discussed in terms of its architecture and instruction set.

In Chapter 2 the 'new' SPECTRE MAP assembler language is presented in terms of its specifications under OSPS.

In Chapter 3 the program logic of OSPS is explained for the benefit of maintenance programmers.

In Chapter 4 the conclusions drawn from the implementation of OSPS are summarized.

Appendix A is a user's guide to OSPS.

Appendix B contains examples of SPECTRE MAP programs which were run under OSPS.

TABLE OF CONTENTS

CHAPTER 1

The SPECTRE Computer

The objective of this chapter is to describe the SPECTRE computer from the point of view of the programmer using the On-line SPECTRE Programming System (OSPS). This description will differ in certain respects from descriptions presented by other authors due to changes which had to be made to reflect the on-line approach of OSPS.

1.1 Architecture of the SPECTRE Computer

The SPECTRE computer has four components:

1. Memory
2. Control unit
3. Arithmetic/Logic unit
4. Input/Output unit

1.1.1 Memory

SPECTRE memory consists of N units called "words".

The maximum number of words must be less than 1000 and is fixed according to the environment in which OSPS is implemented. Each word has a unique decimal address, 000 to N-1, and holds the equivalent of a ten-digit signed integer. The format of a word is:

Sddddddddddd

where "S" is either "+" or "-" and "d" is a decimal digit from 0 to 9. The contents of a word are interpreted as:

a. a machine language instruction.

b. a data item.

(a) Machine Language Instructions:

One machine language instruction occupies one word of memory. The format of an instruction is:

SXXXXXopAAA

where the digits "op" form the operation code of the instruction and the digits "AAA" form the address of the word being referenced by the instruction. The digits "XXXXX" and the sign "S" are ignored.

(b) Data Items:

There are three different types of data recognized by the SPECTRE computer:

1. Integers

2. Floating point numbers

3. Alphanumeric data

(1) Integers

An integer occupies one word of memory. Its format is:

Sddddddddd

where "ddddddddd" is the integer, and "S" is the sign of the integer.

Examples of integer data:

1) The integer 54 has as its internal representation

+0000000054

2) The negative integer -8567912345 has as its internal representation

-8567912345

The largest integer which can be stored in a word is such that its absolute value is less than ten to the power 10.

(2) Floating point numbers

A floating point number occupies one word of memory and its format is:

SccMMMMMMMM

where "S" is the sign of the floating point number and the digits "cc" represent its characteristic coded in excess-50 notation. The digits "MMMMMMMM" represent the mantissa of the floating point number. Floating point numbers, when interpreted, are assumed to be normalized; i.e. the high order digit of the mantissa is not zero. OSPS interprets a floating point number as "S.MMMMMMMM" times ten to the power (cc - 50). The floating point number 0.0 is internally represented as though it were the integer +0.

Examples of floating point number data:

1) The floating point number -.560 times ten to the power (33) is stored as

-8356000000

2) The floating point number .0052 is stored as

+4852000000

The range of a non-zero floating point number which can be stored in a word is such that .1 times ten to the

power (-50) is less than or equal to the absolute value of the floating point number, which is in turn less than ten to the power (+50).

(3) Alphanumeric data

When a word is interpreted as alphanumeric data each pair of digits, starting at the high order digit, is assumed to represent a particular alphanumeric character (see Table 1). The sign of the word is ignored. Therefore the capacity of a word is 5 alphanumeric characters.

Examples of alphanumeric data:

1) The alphanumeric string "ABCD2" is represented internally as:

S1718192002

2) The alphanumeric string "STUV " is represented internally as:

S5051525348

A Table of Alphanumerics and Their Numeric Equivalents

ALPHANUMERIC	2-DIGIT CODE
A	17
B	18
C	19
D	20
E	21
F	22
G	23
H	24
I	25
J	33
K	34
L	35
M	36
N	37
O	38
P	39
Q	40
R	41

TABLE 1

S	50
T	51
U	52
V	53
W	54
X	55
Y	56
Z	57
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
=	11
@	12
+	16
.	27
$	43

TABLE 1 [cont.]

*	44
BLANK	48
/	49
-	32

TABLE 1 [cont.]

1.1.2 Control Unit

The control unit is responsible for the interpretation and execution of SPECTRE machine language instructions. In order to do this, it uses two special registers:

1. IAR - Instruction Address Register
2. MAR - Memory Address Register

In the IAR, the control unit keeps the memory address of the instruction which is currently being executed, while in the MAR, the control unit stores the contents of the address part of the instruction. The MAR is used to make references to memory during execution of the instruction. Only the control unit can access these registers. When the instruction has been executed the contents of the IAR are updated to point to the next logical instruction.

In addition, the control unit has an interrupt mechanism which is triggered either by a programming error, or when a preset number of instructions have been executed and control is to be returned to the programmer. This feature is simulated to prevent monopolization of the facilities of the computer by any particular programmer at a terminal.

When an instruction is being interpreted, the following programming errors will cause interrupts:

1. illegal operation code (the operation code is not a legal SPECTRE machine operation code)
2. addressing exception (the address part of the machine language instruction is not a SPECTRE memory address)

Other programming errors will cause interrupts during the execution of an instruction. These errors are mentioned in 1.2 where the instruction repertoire of the SPECTRE computer is discussed.

1.1.3 Arithmetic/Logic Unit

The arithmetic/logic unit performs the unary and binary integer and floating point arithmetic, shifts, and tests which are made in conditional transfer instructions. In this regard, two special registers called the AC (accumulator) and MQ (multiplier/quotient) are used to hold operands as well as results. These registers have the same capacity as a word of memory.

1.1.4 Input/Output Unit

The input/output unit connects the SPECTRE computer to a remote terminal from which the programmer can provide input directly to his program as it is requested, or receive program output. In addition, the input/output unit has a data interpreter that translates input data into its internal representation.

1.1.4.1 Data Interpreter

The data interpreter can scan only the first 50 characters of the input string. If the length of the input string exceeds 50 characters the remaining characters are ignored. The data interpreter recognizes two types of input data:

a. numeric input

b. alphanumeric input

(a) Numeric input:

Numeric input may be:

1. integers

2. floating point numbers

(1) Integers: a sequence of at most ten decimal digits (excluding leading zeroes), which may be preceded by a sign.

(2) Floating point numbers: a sequence of at most eight decimal digits (excluding leading zeroes), which may be preceded by a sign, and must have a decimal point or be expressed in E-format. In addition, the value of a floating point number must be within the range stated in 1.1.1.

The presence of a decimal point or the E-notation distinguishes floating point numbers from integers. Either blanks or a comma can be used to delimit numbers. If only blanks appear in the input string, the items in the input list are assigned a value of zero.

Examples of numeric input:

1)	-4135	negative integer
2)	123456789	integer
3)	12345678912	invalid data (too many digits)
4)	-23.4	negative floating point

number

5) 18.456E-23 floating point number

6) 18.456E-63 invalid floating point number (value exceeds the range for floating point numbers)

(b) Alphanumeric input:

Alphanumeric input consists of alphanumeric strings. A string is a sequence of at most five non-blank alphanumeric characters listed in Table 1.

Blanks are used to delimit alphanumeric strings. If only blanks appear in the input string, the items in the input list are filled with blanks.

Examples of alphanumeric input:

1) "ACVB" alphanumeric string (interpreted as the string "ACVB ")

2) "ASDEWW" invalid alphanumeric string (too many characters)

1.2 SPECTRE Machine Instruction Set

The presentation of the SPECTRE machine instruction set includes the mnemonic and machine operation code of each instruction, followed by a description of its result along with any errors which may cause an interrupt during its execution. The SPECTRE instruction set is discussed under the following categories:

a. Load and Store

b. Transfer of Control

c. Integer Arithmetic

d. Floating Point Arithmetic

e. Unary Arithmetic

f. Shifting

g. Input/Output

In Table 2 is presented a list of all SPECTRE machine operation codes and their mnemonics.

1.2.1 Load and Store Instructions

There are five instructions which move information from either the AC or MQ to memory and vice-versa. There are no instructions which move information from one register to another, or from one memory location to another. The format

The SPECTRE Instruction Set

MNNEMONIC		NUMERIC OPERATION CODE
CLA	(clear and add AC)	10
STO	(store AC)	11
LDQ	(load MQ)	12
STQ	(store MQ)	13
STZ	(store zero)	15
STP	(stop)	40
TRA	(transfer)	50
TLE	(transfer less than equal zero)	51
TNZ	(transfer not zero)	52
TPL	(transfer positive)	53
TZE	(transfer zero)	54
TMI	(transfer negative)	55
TSL	(transfer subroutine linkage)	56
ADD	(add integer)	20
SUB	(subtract integer)	21
MPY	(multiply integer)	22
DIV	(divide integer)	23
FAD	(add floating point)	24
FSU	(subtract floating point)	25

TABLE 2

FMP	(multiply floating point)	26
FDV	(divide floating point)	27
SSP	(set sign positive)	16
CHS	(change sign)	17
ALS	(short left shift)	60
ARS	(short right shift)	61
LLS	(long left shift)	62
LRS	(long right shift)	63
INP	(input)	30
OUT	(output)	31
RNi (i = 1-5)	(read numeric)	71-75
RAi (i = 1-5)	(read alphanumeric)	76-80
PNi (i = 1-5)	(print numeric)	81-85
PAi (i = 1-5)	(print alphanumeric)	86-90

TABLE 2 [cont.]

of a load or store instruction is:

SXXXXXopAAA

where "op" is the operation code and "AAA" is the memory location either from which information is to be moved, or at which information is to be stored.

1. CLA (+XXXXX10AAA)

Result: The AC is loaded with the contents of the word at memory location "AAA".

2. STO (+XXXXX11AAA)

Result: The contents of the AC are stored in the word at location "AAA".

3. LDQ (+XXXXX12AAA)

Result: The MQ is loaded with the contents of the word at location "AAA".

4. STQ (+XXXXX13AAA)

Result: The contents of the MQ are stored in the word at location "AAA".

5. STZ (+XXXXX15AAA)

Result: +0000000000 is stored in the word at

location "AAA".

1.2.2 Transfer of Control Instructions

The transfer of control instructions determine which instruction is to be the next executed. By a "transfer of control to location AAA" is meant that the value "AAA" has been placed in the IAR by the control unit, implying that the next instruction to be executed is in the word at location "AAA". The format of a transfer of control instruction is:

SXXXXXopAAA

where "op" is the operation code and "AAA" is the address to which control is to be transferred.

1. STP (+XXXXX40XXX)

   Result: Program execution is terminated.

2. TRA (+XXXXX50AAA)

   Result: Control is transferred to location "AAA".

3. TLE (+XXXXX51AAA)

   Result: Control is transferred to location

"AAA" only if the number in the AC is less than or equal to zero.

4. TNZ (+XXXXX52AAA)

Result: Control is transferred to location "AAA" only if the number in the AC is not equal zero.

5. TPL (+XXXXX53AAA)

Result: Control is transferred to location "AAA" only if the number in the AC is positive.

6. TZE (+XXXXX54AAA)

Result: Control is transferred to location "AAA" only if the number in the AC is zero.

7. TMI (+XXXXX55AAA)

Result: Control is transferred to location "AAA" only if the number in the AC is negative.

8. TSL (+XXXXX56AAA)

Result: The contents of the IAR are incremented by 1 and stored in the address field of the instruction at location "AAA". Then control is transferred to location "AAA" + 1.

1.2.3 Integer Arithmetic Instructions

The format of an integer arithmetic instruction is:

SXXXXXopAAA

where "op" is the operation code and "AAA" is the memory location of the word in which one of the operands is stored. The other operand, and eventually the result, will be in either the AC, MQ or the (AC,MQ). (AC,MQ) denotes the concatenation of the AC and MQ. It is interpreted as a 20-digit integer whose sign is that of the MQ. The digits of the AC are interpreted as the high order ten digits of the integer while the digits of the MQ are interpreted as the low order ten digits of the integer.

Both operands are treated as integers and the result is stored as an integer.

1. ADD (+XXXXX20AAA)

Result: The integer at location "AAA" is added to the integer in the AC and the result is placed in the AC. The error "arithmetic overflow" can occur. This error is caused when the result of an integer arithmetic operation is such that its absolute value is greater than or equal to ten to the power 10.

2. SUB (+XXXXX21AAA)

Result: The integer at location "AAA" is subtracted from the integer in the AC and the result is placed in the AC. Arithmetic overflow can occur.

3. MPY (+XXXXX22AAA)

Result: The integer at location "AAA" is multiplied by the integer in the MQ and the result is placed as a 20-digit integer in the (AC,MQ). Both the AC and the MQ will have the sign of the product.

4. DIV (+XXXXX23AAA)

Result: The 20-digit integer in the (AC,MQ) is divided by the integer at location "AAA". The quotient is stored in the MQ while the remainder is stored in the AC. The AC will have the sign of the quotient. Both arithmetic overflow and a divide exception can occur. The error "divide exception" is caused by an attempt to divide a number by zero.

1.2.4 Floating Point Arithmetic Instructions

The format of a floating point arithmetic instruction is:

SXXXXXopAAA

where "op" is the operation code and "AAA" is the address of the word in which one of the operands is stored. The other operand is in either the AC or MQ. Both operands are treated as normalized floating point numbers. The eight most significant digits of the result will be stored as a normalized floating point number in the AC.

1. FAD (+XXXXX24AAA)

Result: The floating point number at location "AAA" is added to the floating point number in the AC and the result is placed in the AC. The errors "exponent overflow" and "exponent underflow" can occur. The error "exponent overflow" is caused when the result of a floating point arithmetic operation is such that its absolute value is greater than or equal ten to the power 50. "Exponent underflow" is caused when the result is not zero, but its absolute value is less than .1 times ten to the power (-50).

2. FSU (+XXXXX25AAA)

Result: The floating point number at location "AAA" is subtracted from the floating point number in the AC, and the result is placed in the AC. Exponent overflow and exponent underflow can occur.

3. FMP (+XXXXX26AAA)

Result: The floating point number at location "AAA" is multiplied by the floating point number in the MQ, and the product is placed in the AC. Both exponent overflow and exponent underflow can occur.

4. FDV (+XXXXX27AAA)

Result: The floating point number in the MQ is divided by the floating point number at location "AAA", and the result is placed in the AC. The errors exponent overflow, exponent underflow, and divide exception can occur.

1.2.5 Unary Arithmetic Instructions

Unary arithmetic is performed using the contents of the AC as the only operand. Due to the nature of the unary operations which can be performed, the type of number in the AC is of no significance. The format of a unary arithmetic instruction is:

SXXXXXopXXX

where "op" is the operation code. The address part of the instruction is not used. There are two unary arithmetic instructions:

1. SSP (+XXXXX16XXX)

Result: The sign of the AC is made positive.

2. CHS (+XXXXX17XXX)

Result: The sign of the AC is changed.

1.2.6 Shifting Instructions

Shifts are performed either on the digits of the AC--short shifts--or on the twenty digits of the (AC,MQ)--long shifts. The format of a shift instruction is:

SXXXXXopAAA

where "op" is the operation code, and "AAA" is the number of digit positions by which the number is to be shifted. Regardless of the number "AAA", in short shifts at most ten shifts will occur, and in long shifts at most twenty shifts will occur. There are four shift instructions:

1. ALS (+XXXXX60AAA)

Result: The number in the AC is shifted left by "AAA" digit positions. The high order "AAA" digits are lost, and the low order "AAA" digit positions are filled with zeroes. The sign of the number is unchanged.

2. ARS (+XXXXX61AAA)

Result: The number in the AC is shifted right by "AAA" digit positions. The low order "AAA" digits are lost, and the high order "AAA" digit positions are filled with zeroes. The sign of the number is unchanged.

3. LLS (+XXXXX62AAA)

Result: The sign of the AC is set the same as the sign of the MQ. The number in the (AC,MQ) is shifted left by "AAA" digit positions. The high order "AAA" digits are lost, and the low order "AAA" digit positions are filled with zeroes. The signs of the AC and the MQ remain unchanged after shifting.

4. LRS (+XXXXX63AAA)

Result: The sign of the AC is set the same as the sign of the MQ. The number in the (AC,MQ) is shifted right by "AAA" digit positions. The low order "AAA" digits are lost, and the high order "AAA" digit positions are filled with zeroes. The signs of the AC and the MQ remain unchanged after shifting.

1.2.7 Input/Output Instructions

The format of an input/output instruction is:

SXXXXXopAAA

where "op" is the operation code, and "AAA" is either the first location at which data is to be input, or the first location from which data is to be output.

In the mnemonic input/output instructions, the letter "i" represents an integer from 1 to 5.

1.2.7.1 Input Instructions

The execution of an input instruction consists of two phases:

a. Phase 1: A check is made to ensure that data items will not be stored beyond the last word of memory - in which case an addressing exception would occur. Otherwise, a message is displayed on the terminal requesting the programmer to input the type, and quantity of data currently required by his program. At this point, program execution is suspended.

b. Phase 2: When the programmer has submitted the data from the terminal phase 2 of the operation begins. The external representation of the data is checked for validity by the data interpreter (for

definitions of valid data see 2.2). Invalid data will cause an input error. Otherwise, the data is translated into its internal representation and stored at successive memory locations, starting at location "AAA".

1. RNi (+XXXXX71AAA,.....,+XXXXX75AAA)

Phase 1: The message "i NUM" is displayed on the terminal.

Phase 2: The data is read and the internal representation of each number is stored in "i" consecutive memory locations starting at location "AAA". If fewer than "i" numbers are input, the remainder are stored as zero. If more than "i" numbers are input the additional numbers are ignored.

The mnemonic instruction INP (30) can be used in place of RN1 (71).

2. RAi (+XXXXX76AAA,.....,+XXXXX80AAA)

Phase 1: The message "i STG" is displayed on the terminal.

Phase 2: The data is read and the internal representation of each alphanumeric string is stored in "i" consecutive memory locations, starting at location "AAA". If

fewer than "i" strings are input, the remainder are stored as blank strings. If more than "i" strings are input, the additional strings are ignored.

1.2.7.2 Output Instructions

The execution of an output instruction results in an immediate display, on the terminal, of the contents of the specified memory locations. After the display, program execution is suspended, and can continue only when requested to do so by the programmer.

3. PNi (+XXXXX81AAA,.....,+XXXXX85AAA)

Result: The contents of "i" consecutive words, starting at location "AAA", accompanied by their respective addresses, are displayed on the terminal. This display is in the form of a dump.

The mnemonic instruction OUT (31) can be used in place of PN1 (81).

4. PAi (+XXXXX86AAA,.....,+XXXXX90AAA)

Result: The alphanumeric equivalent of the contents of each word in succession, starting at location "AAA", are displayed on the terminal. A word which contains

a pair of digits for which there is no alphanumeric equivalent may be incorrectly displayed, but is not recognized as an error.

CHAPTER 2

The SPECTRE MAP Assembler Language

In this chapter, a description of the SPECTRE MAP assembler language implemented under OSPS is presented. Some of the specifications of the SPECTRE MAP assembler language implemented under OSPS differ slightly from the specifications of the language as used in a batch environment due to the on-line environment in which OSPS operates.

2.1 Symbols Used in SPECTRE MAP

There are three types of symbols used in the SPECTRE MAP assember language:

a. symbolic address

b. literal

c. location counter reference

(a) Symbolic address

A symbolic address is an alphanumeric string consisting of one to three non-blank characters. These

characters may be any of those appearing on the keyboard of the remote terminal being used. However, a symbolic address must not be the single character "*".

The value of a symbolic address is the SPECTRE memory address which was assigned to it during assembly.

(b) Literal

There are two types of literals in SPECTRE MAP:

1. numeric literals
2. alphanumeric literals

(1) NUMERIC LITERAL: A numeric literal consists of the character "=" followed immediately by an integer or floating point number. This number is the value of the numeric literal.

(2) ALPHANUMERIC LITERAL: An alphanumeric literal consists of the two characters "=A" followed immediately by an alphanumeric string. This string is the value of the alphanumeric literal.

The number/string which is indicated in the literal must conform to the specifications outlined in 1.1.4.1.

Examples of literals:

1. =12345 numeric literal (interpreted as the integer '12345')
2. =12345678912 invalid numeric literal (too many digits)
3. =12.4 numeric literal (interpreted as the floating point number '12.4')
4. =A12 alphanumeric literal (interpreted as the string "12 ")
5. =A123456 invalid alphanumeric literal (too many characters)

(c) Location counter reference

The location counter reference consists of the single character "*", and has the current value of the location counter, which during assembly contains the memory address at which the next machine instruction, or data item will be stored.

2.2 Format of a SPECTRE MAP Program

The SPECTRE MAP program can be thought of as a deck of 20-columned cards with each card containing one SPECTRE MAP statement. A statement has four fields which are (from left to right):

1. symbolic address field
2. operation field
3. operand field
4. modifier field

Other than the name field, which must begin in column one, it is not necessary for any one field to begin in any particular column. However, the fields must be in the same order (left to right) as they are listed. All fields are delimited by one or more blanks and, with the exception of the operation field, are variable in length.

(1) Symbolic address field

The symbolic address field must begin in column one of the card. A symbolic address may be entered into this field, in which case the symbolic address is "defined", or the field can be left blank. A particular symbolic address must be defined only once in a program.

(2) Operation field

The operation field must contain one of the mnemonic instructions listed in Table 2, or one of the assembler instructions discussed in 2.3.

(3) Operand field

Depending on the mnemonic instruction in the operation field, the operand field will contain one of the following:

a. SYMBOLIC ADDRESS: The symbolic address in the operand field must be defined in the program. A symbolic address must appear in the operand field of an "END" statement (see 2.3).

b. POSITIVE INTEGER: A positive integer must be used in the operand field of a statement which has one of the following mnemonics in its operation field: ALS, ARS,LLS, LRS, ORG, or RES. Only these statements have this type of operand. If the field is left blank it will be interpreted as the integer zero.

c. LITERAL: This type of operand must be used in a statement that has the mnemonic assembler instruction "CST" in its operation field.

d. LOCATION COUNTER REFERENCE: The location counter reference "*" may be used in all statements, with the exception of those statements in which positive integers must be used, and the "END" statement.

(4) Modifier field

The modifier field must contain a signed integer. This field can be used by statements, other than the "END" statement, which have a symbolic address or location-counter reference in the operand field. Otherwise the field is ignored.

Any additional columns which follow the modifier field, including the modifier field itself if it is missing, may be used for comments.

2.3 Assembler Instructions

SPECTRE MAP statements are classified into two groups:

a. Statements which are translated into machine instructions. Such statements will have one of the mnemonics of the SPECTRE instruction set (see Table 2) in its operation field.

b. Statements which contain directives to the assembler. These statements will have one of the following mnemonics in its operation field:

1. CST literal

This statement is used to define data in the program.

Result: The literal in the operand field is translated into its internal representation and stored at the memory location currently referenced by the location counter.

2. RES n

This statement is used to reserve storage in SPECTRE memory.

Result: The current value in the location-counter is incremented by "n", where "n" is the positive integer in the operand field.

3. ORG n

Result: The value of the location-counter is set to "n".

4. END address

Result: Assembly of the SPECTRE MAP program ends. The first instruction to be executed in the program is that instruction which is referenced by the symbolic address "address" in the operand field.

CHAPTER 3

The Program OSPS

OSPS is a conversational program written in IBM 360 ASSEMBLER to operate under the Manitoba University Monitor (MUM) on an IBM 360/65 computer. The purpose of OSPS is to provide users with the facility of creating and editing SPECTRE MAP programs, having them assembled, and executing them in a conversational mode.

This chapter presents a description of the program OSPS, including its I/O facilities, as well as the general program logic and algorithms developed to fulfill its capabilities.

3.1 Functions of OSPS

OSPS provides the user with three main facilities:

1. Program creation and editing.
2. Assembly of a previously created program.
3. Interactive program execution.

In relation to the program create/edit facility, OSPS will execute one of the following functions in response to the appropriate command from the user:

a. Adding statements to the logical end of the program.

b. Replacing statements in the program.

c. Inserting statements in the program.

d. Deleting statements.

e. Listing program statements.

f. Re-sequencing statements.

In relation to the facility of assembly of a previously created program, OSPS executes only one function, that function being the assembly of the SPECTRE MAP program which has been generated by the programmer.

In relation to the facility of interactive program execution, OSPS will execute one of the following functions in response to the appropriate command from the user:

a. Start/continue program execution.

b. Re-start program execution.

c. Memory dump and register display.

See Table 3 for a list of commands and the corresponding functions which are executed by OSPS.

OSPS Commands and Functions Executed

COMMAND	FUNCTION EXECUTED
$$C	adding program statements
$$R	replacing program statements
$$I	inserting program statements
$$D	deleting program statements
$$L	listing program statements
$$Q	resequencing program statements
$$T	program assembly
$$X	start/continue program execution
$$XR	re-start program execution
$$M	memory dump and register display
$$E	ending communication with OSPS

TABLE 3

3.2 Modes of Operation of OSPS

At any time the user has control, OSPS is said to be in either EDIT or EXECUTE mode. These modes reflect the current status of the user's program.

In EDIT mode the user's program is in one of two states:

1. EDIT/CREATE: At this stage the user is in the process of developing his SPECTRE MAP program.
2. POST-ASSEMBLY: The user's program has just been assembled but errors were found in his program.

In EXECUTE mode the user's program is in one of four states:

1. POST-ASSEMBLY: The user's program has just beeen successfully assembled and no errors were found. However, execution of his program has not yet been enabled.
2. SUSPENDED-EXECUTION: Program execution has been temporarily suspended, either to enable the user to monitor it, or because program input must be supplied.

3. POST-EXECUTION: Program execution has just been successfully completed.

4. PROGRAM-INTERRUPT: Program execution has been terminated because of a programming error.

OSPS will switch to EXECUTE mode from EDIT mode only when the user's program has just been successfully assembled. However, OSPS will switch to EDIT mode from EXECUTE mode as a result of the "$$C" command.

3.3 Input/Output

The purpose of this section is to describe the nature of input/output with respect to OSPS. Different types of I/O are discussed, along with their relationships to the EDIT and EXECUTE modes of OSPS.

3.3.1 Input/Output Types

3.3.1.1 Input

OSPS handles 2 types of input strings:

a. commands

b. card images/data

(a) Commands

A command is a request by the user for OSPS to execute one of the functions mentioned in 3.1. Commands are recognized by the presence of the characters "$$" in the first two positions of the input string. If the command indicated by the user is not listed in Table 3, then the command is invalid, and a diagnostic message is typed on the terminal.

In EDIT mode, only function requests relating to program creation/editing, or assembly, other than "$$E", are processed. Any requests relating to interactive program execution are flagged as invalid commands and an appropriate diagnostic message is displayed to the user on the terminal. Commands which may be processed in EDIT mode are called EDIT commands.

In EXECUTE mode, only function requests relating to interactive program execution, the request to list SPECTRE MAP program statements, the request to switch back into EDIT mode, or the request to end communication with OSPS are processed. All other function requests are flagged as invalid commands. Commands which may be processed in EXECUTE

mode are called EXECUTE commands.

(b) Card Images/Data

In EDIT mode, if the input string is not a command, OSPS treats the input string as a card image which is to replace an existing card image, to be inserted in the user's program, or added to the logical end of the user's program.

In EXECUTE mode, the input string may be treated as data which is to be supplied to the user's program. However, the input string will be treated as data only if the user has just been requested, by a message on the terminal, to input data to his program.

3.3.1.2 Output

An output string generated by OSPS is one of two types:

a. write/read

b. write-only

(a) Write/Read Output

The main characteristic of this type of output is that after the output string has been generated by OSPS and typed on the terminal, control is transferred to the user

and OSPS "waits" for the user to submit input before it will resume processing. Write/read output not only prompts subsequent input by the user, but also is the only means by which control is transferred to the user from OSPS.

In EDIT mode, write/read output consists of a sequence number. If the first two characters of the input string are "$$" the input string is treated as a command and checked for a valid EDIT command.

In EXECUTE mode, write/read output consists of a message which prompts the user to enter a valid EXECUTE command, or to enter data to be read by his program. The input string which follows a message prompting a command will be treated as a command and checked for a valid EXECUTE command. The input string which follows a message prompting program input will be interpreted as data unless the characters which appear in the first two positions of the input string are "$$", in which case the input string is treated as a command and checked for a valid EXECUTE command.

(b) Write-only Output

Immediately following the display of a string of write-only output, control returns directly to OSPS. The user "waits" for control to be passed to him while OSPS

remains in control.

In EDIT mode, write-only output appears in one of the following forms:

a) A card image preceded by its sequence number.

b) An assembler error diagnostic message which is generated as a result of an error which was encountered in the user's SPECTRE MAP program.

c) A command error diagnostic resulting from an invalid command or an invalid command parameter.

In EXECUTE mode, write-only output appears in one of the following forms:

a) The contents of a word of SPECTRE memory preceded by its 3-digit memory address.

b) The contents of the AC and MQ registers preceded by their names "AC" and "MQ".

c) Output which has been generated by the user's program.

d) An error diagnostic message as the result of a programming error which was encountered in the user's program.

e) A command error diagnostic.

In all cases, when control is returned to OSPS after generating a write-only output string, OSPS will immediately generate another string of write-only output, or a write/read output string. Thus, write-only output is used to display several consecutive output strings before returning control to the user.

3.3.2 Input/Output Subroutines

This section discusses the I/O subroutines used in OSPS in order to perform actual I/O operations, and to interpret the input string and transfer control to the appropriate routine to process it.

3.3.2.1 Subroutine MUMINT

(a) Purpose:

All I/O operations of OSPS are handled by MUM. Whenever an I/O operation is to be performed, control is passed to MUMINT, which passes I/O parameters to MUM. At this time OSPS is "rolled out" of memory, and is not "rolled in" until the operation has been completed. Thus, subroutine MUMINT acts as an interface between OSPS and MUM.

(b) Input parameters:

1. CODE - the code for the type of output string to be generated, ie. either write/read or write-only output.
2. INPUT - the address of the input string (in the case of write/read output).
3. LIN - the maximum length of the input string (in the case of write/read output).
4. OUTPUT - the address of the output string.
5. LOUT - the length of the output string.
6. RETURN - the address in OSPS, in base-displacement form, to which control is to be transferred when control is passed back to OSPS from MUM.
7. REG1,...,REGN - the address registers whose contents must be relocated upon return from MUM.

(c) Output parameters:

1. INPUT - the address of the input string.
2. LIN - the length of the input string.
3. REG1,....,REGN - address registers whose contents have been relocated.

3.3.2.2 Subroutine TESTIN

(a) Purpose:

The purpose of routine TESTIN is to interpret an input string and transfer control to the appropriate routine to process it.

(b) Input parameters:

1. VARBRAN - the address, in base-displacement form, of the routine to which control is to be transferred if the input string is card image/data. However, if the input string is a command, the command is interpreted and control is passed to the routine which processes it.

3.4 Program Creation/Editing

3.4.1 Card Image Storage and Referencing

Card images are stored in MAPAREA, a vector consisting of "MAXNO" elements, where "MAXNO" denotes the maximum number of card images which can be stored in MAPAREA. Each element in MAPAREA holds one "MAPRL"-columned card image. In addition, the lengths of card images are stored in LENGTH, a vector consisting of "MAXNO" elements, where LENGTH(I) is the length of the card image in MAPAREA(I). The "length" of a card image is the number of characters which were input by the user on the terminal when the card image was created. The index "I" in MAPAREA(I) will be referred to as the "position" in MAPAREA of a card image.

All positions in MAPAREA are kept in a stack called IMAGPOOL, with each entry in the stack holding a distinct position. In order to distinguish free positions in MAPAREA from those which are currently occupied, OSPS maintains a pointer, INDEX, to IMAGPOOL such that all available positions are in IMAGPOOL(J), where J is less than or equal to INDEX. If INDEX equals zero then all positions in MAPAREA are currently occupied. On the other hand, if INDEX equals "MAXNO" then all positions in MAPAREA are available for card

image storage.

For every card image stored in MAPAREA there exists a distinct sequence number which was assigned to it at the time it was created. All references are made to card images by their sequence numbers. OSPS stores the sequence numbers of card images which are in MAPAREA in a table called DISPCUR, consisting of "MAXNO" entries with one sequence number per entry. Sequence numbers are arranged in this table such that DISPCUR(I) will contain the sequence number of the card image in MAPAREA(I). Any sequence number which is in DISPCUR is said to "exist", and implies that there is a card image in MAPAREA with that particular sequence number. Elements of DISPCUR which do not contain sequence numbers are set to zero.

OSPS updates information regarding the logical order of card images in three areas:

1. FIRSTCD1 - the position of the first logical card image
2. CHAIN - the position of the last logical card image
3. DISPNEXT - a table of positions in MAPAREA, consisting of "MAXNO" entries. These positions are arranged such that DISPNEXT(J) equals K implies that the card

image in MAPAREA(K) logically follows the card image in MAPAREA(J). If J is not equal to zero, and K is zero, then the card image in MAPAREA(J) is the last logical card image.

3.4.2 Create/Edit Subroutines

The following subroutines were developed to be used by the create/edit function routines which are discussed in 3.4.3:

1. FINDPOS
2. FINDPREC
3. ADDIMAGE
4. DELIMAGE

3.4.2.1 Subroutine FINDPOS

(a) Purpose:

This routine finds the position of a card image when given its sequence number.

(b) Input parameters:

1. SEQ# - the sequence number of the card image whose position is to be found.

(c) Output parameters:

1. POSEQ - the position of the card image whose sequence number is SEQ#.

For an illustration of the procedure followed by FINDPOS see Figure 1.

Subroutine FINPOS

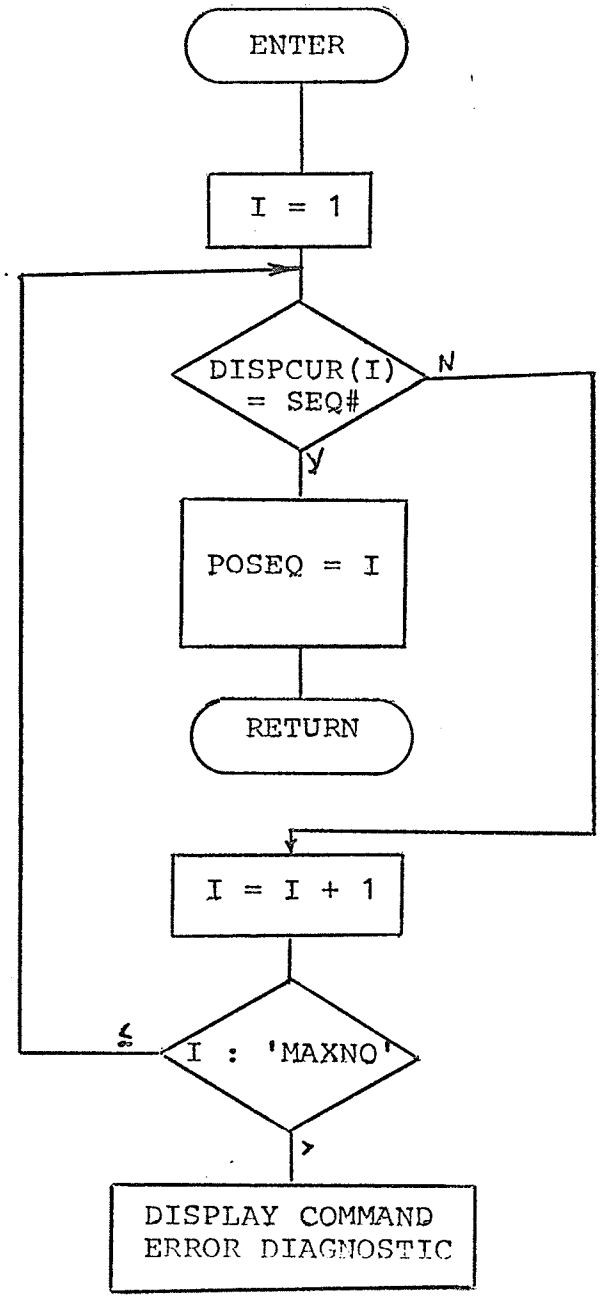

FIGURE 1

3.4.2.2 Subroutine FINDPREC

(a) Purpose:

Given the position "k" of a card image which is not the first logical card image, this routine finds the position of the card image which immediately precedes it.

(b) Input parameters:

1. POS - the position of the card image following that card image whose position is to be found.

(c) Output parameters:

1. POS1 - the position of the preceding card image.

For an illustration of the procedure followed by FINDPREC see Figure 2.

3.4.2.3 Subroutine ADDIMAGE

(a) Purpose:

The purpose of this routine is to store a card image in MAPAREA, as well as to update the information in DISPCUR and DISPNEXT.

(b) Input parameters:

Subroutine FINDPREC

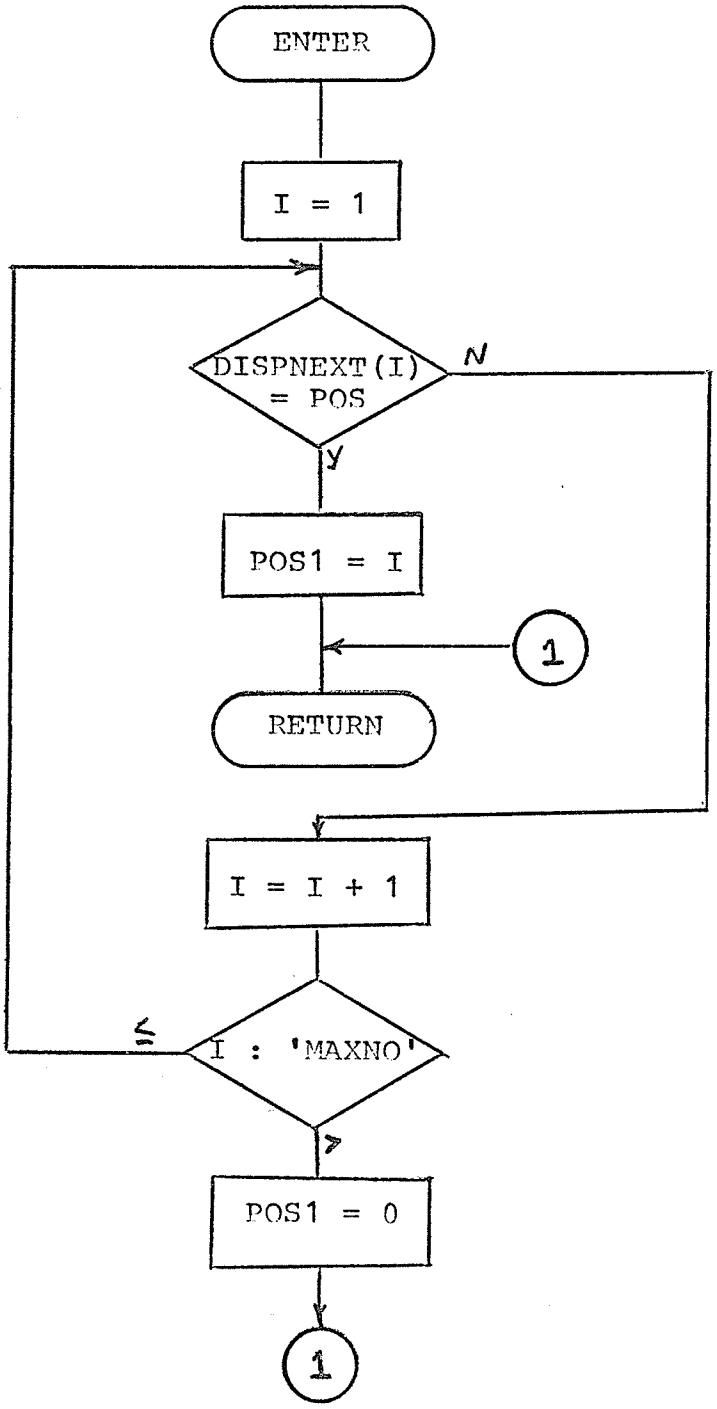

FIGURE 2

1. ASEQ# - the sequence number of the card image to be added to MAPAREA.
2. CHAIN1 - the position of the card image which logically precedes the card image to be added. If CHAIN1 equals zero, then the card image to be added is the first logical card image.

For an illustration of the procedure followed by ADDIMAGE see Figure 3.

3.4.2.4 Subroutine DELIMAGE

(a) Purpose:

The purpose of this routine is to delete a card image whose position is "k", and update the information in the DISPCUR and DISPNEXT tables.

(b) Input parameters:

1. POSD - the position "k" of the card image to be deleted.
2. POS1 - the position of the card image preceding the card image to be deleted.
3. POS2 - the position of the card image following the card image to be deleted.

Subroutine ADDIMAGE

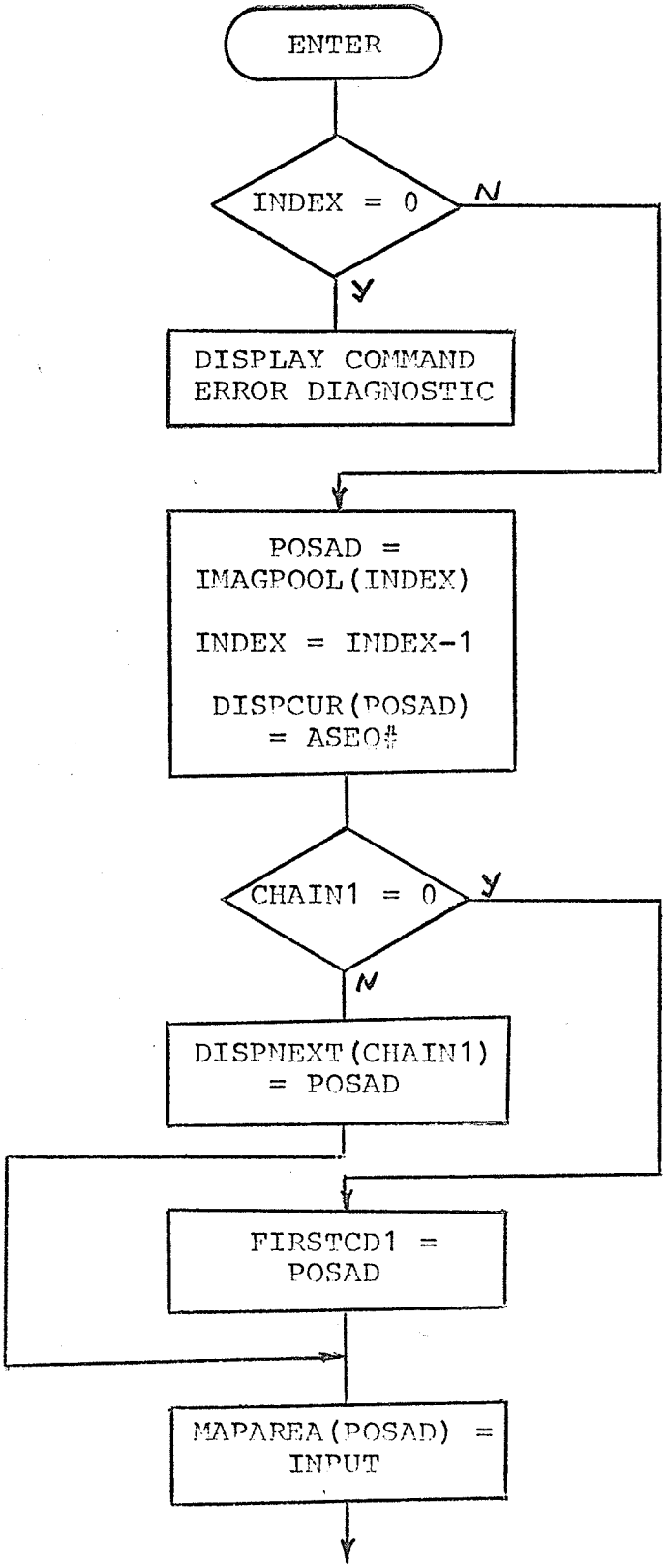

FIGURE 3

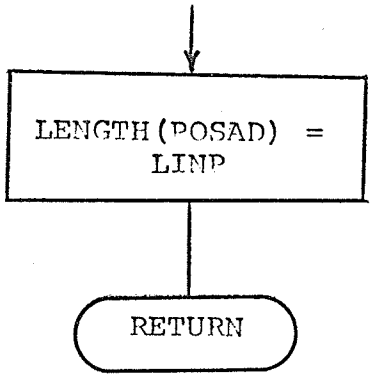

FIGURE 3 [cont.]

For an illustration of the procedure followed by DELIMAGE see Figure 4.

3.4.3 Create/Edit Function Routines

The following routines were developed to perform the create/edit functions of OSPS:

1. CREATE (adding statements to the logical end of the program)
2. DELETE (deleting statements from the program)
3. REPLACE (replacing statements in the program)
4. LIST (listing program statements)
5. INSERT (inserting statements into the program)
6. RESEQ (resequencing program statements)

3.4.3.1 Routine CREATE

Command: $$C

In response to the command "$$C", OSPS outputs the sequence number which is stored in LASTSEQ. This number will always be a multiple of ten, and will be the sequence number assigned to the last logical card in MAPAREA.

Figure 5 illustrates the procedure followed in

Subroutine DELIMAGE

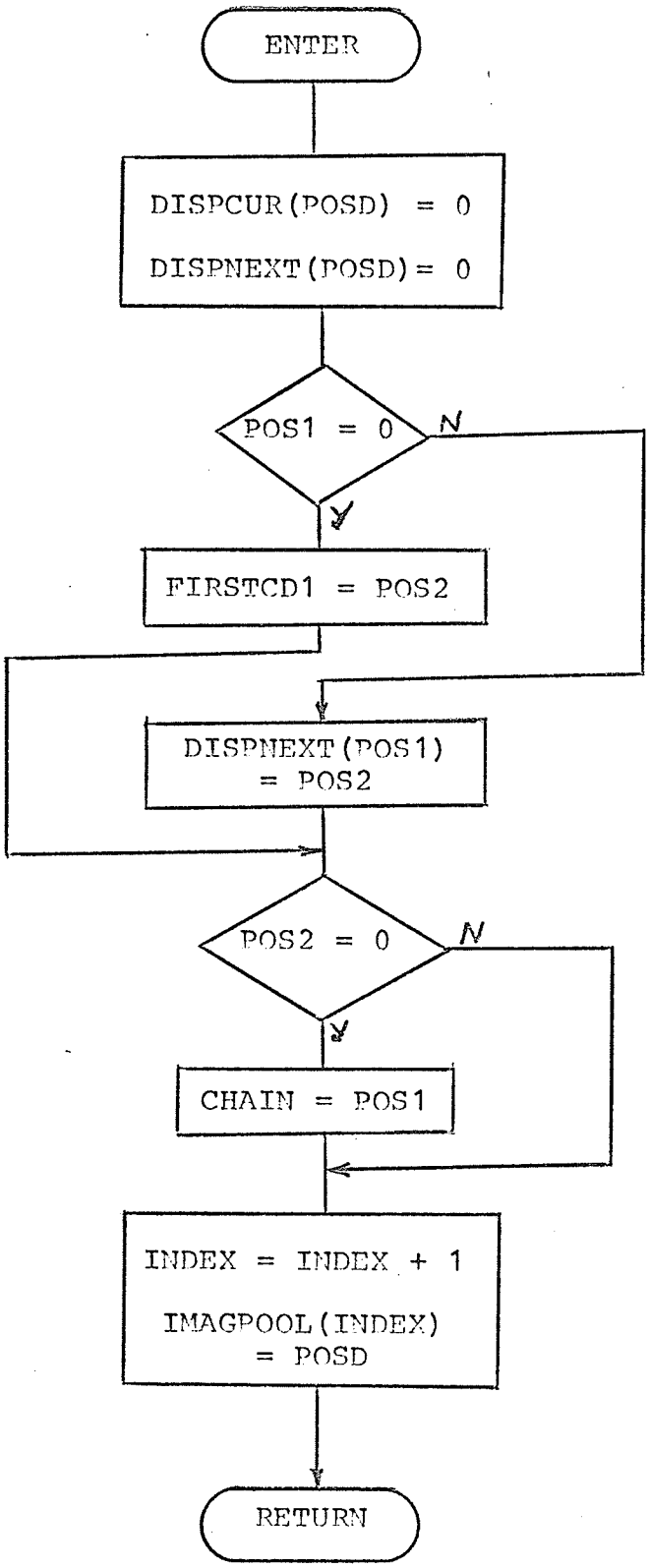

FIGURE 4

Routine CREATE

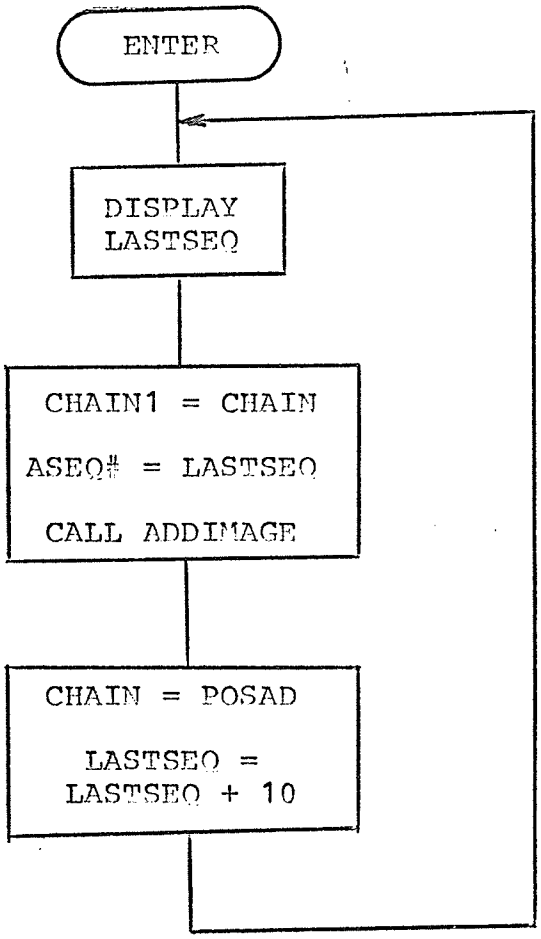

FIGURE 5

response to the "$$C" command.

3.4.3.2 Routine DELETE

Command: $$D m,n

In response to the "$$D m,n" command, OSPS deletes all card images whose sequence numbers are between "m" and "n" inclusive. Both "m" and "n" must be existing sequence numbers and "m" must be less than or equal to "n". If "n" is omitted then only the card image whose sequence number is "m" will be deleted.

Figure 6 illustrates the procedure which is followed by DELETE.

3.4.3.3 Routine REPLACE

Command: $$R m

In response to the "$$R m" command, OSPS displays the card image whose sequence number is "m", and retypes sequence number "m" on a new line. The card image which is submitted at this time will replace the card image which is displayed on the above line.

Figure 7 illustrates the procedure followed by

Routine DELETE

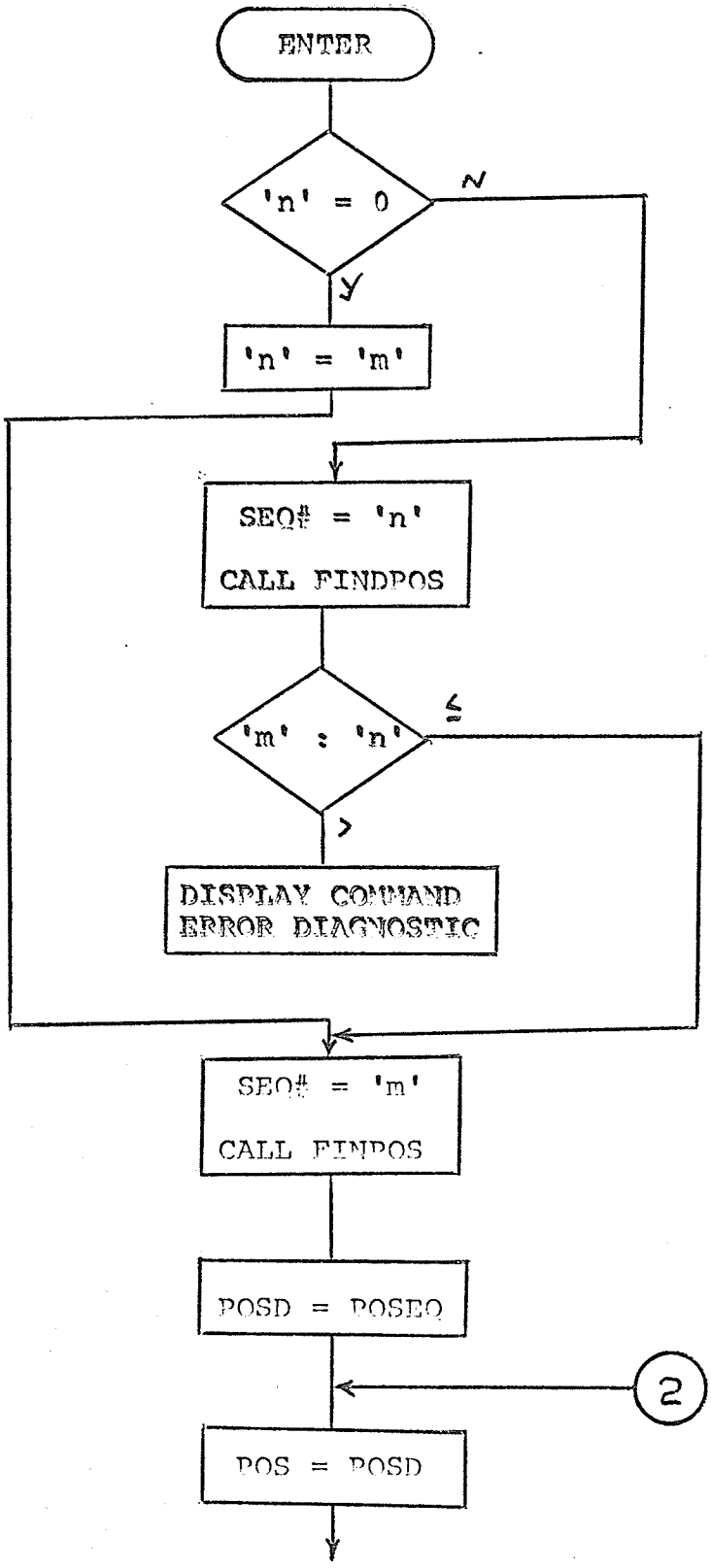

FIGURE 6

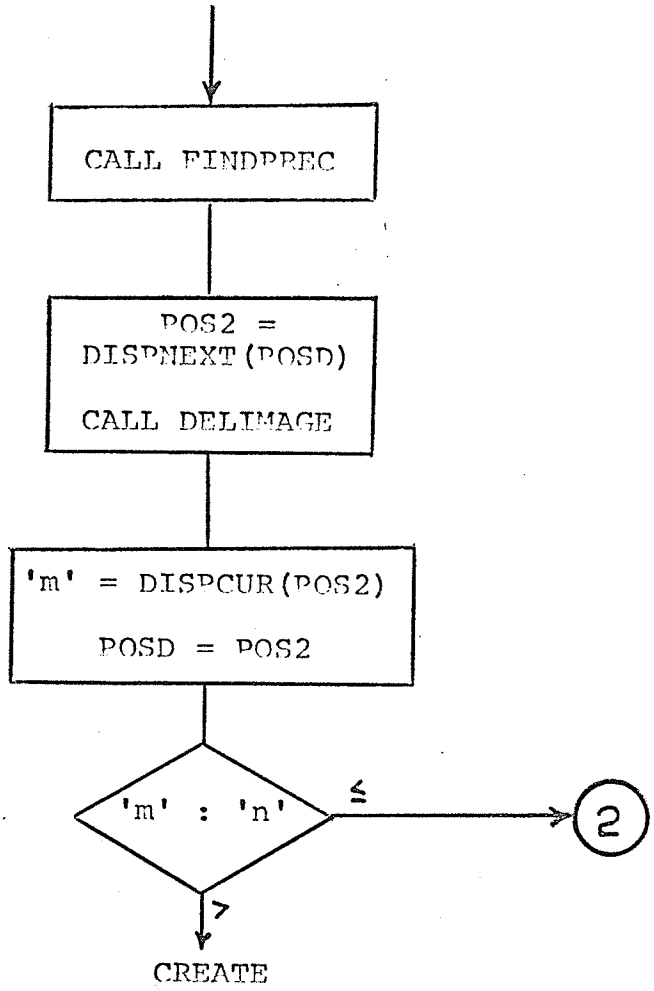

FIGURE 6 [cont.]

Routine REPLACE

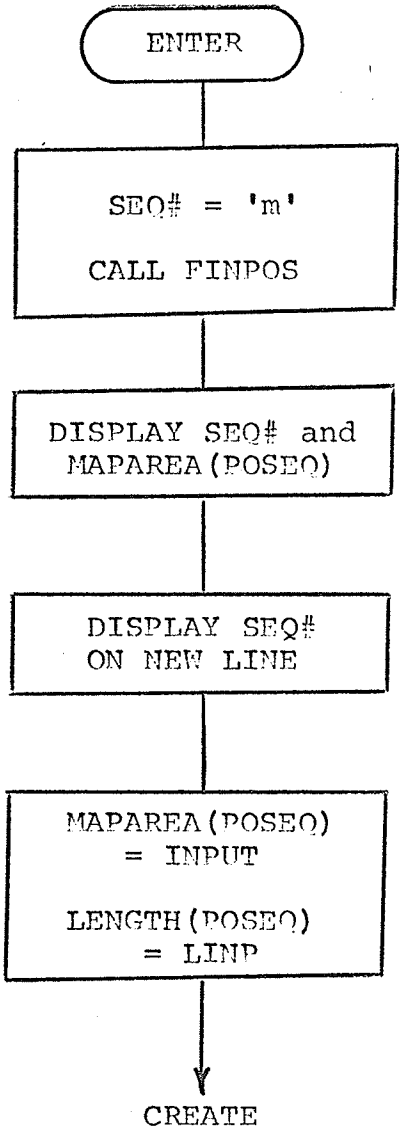

FIGURE 7

REPLACE.

3.4.3.4 Routine INSERT

Command: $$I m,k

In response to the insert command, OSPS permits the user to insert card images between the card image whose sequence number is "m" and the card image which immediately follows it. The integer "k" is the increment which is used to generate new sequence numbers. If "k" is either zero or omitted then an increment of one is used.

Figure 8 illustrates the procedure followed by this routine.

3.4.3.5 Routine LIST

Command: $$L m,k

In response to the "$$L m,k" command, OSPS displays the card image and sequence number for "k" consecutive card images, beginning with the card image whose sequence number is "m". If "m" is zero, listing begins with the first logical card image. If "k" is either zero or omitted then only one card image is listed.

Figure 9 illustrates the procedure followed by

Routine INSERT

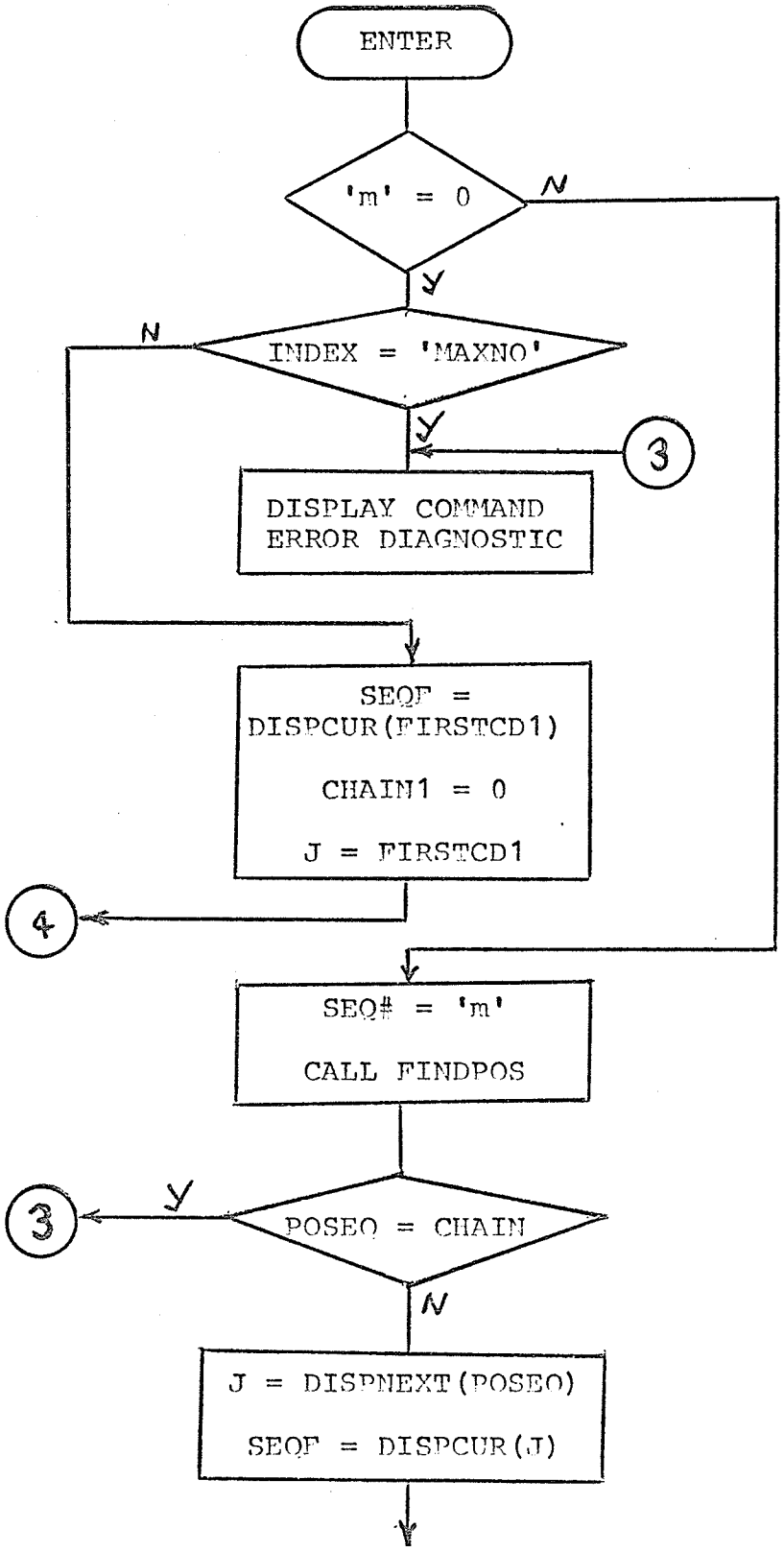

FIGURE 8

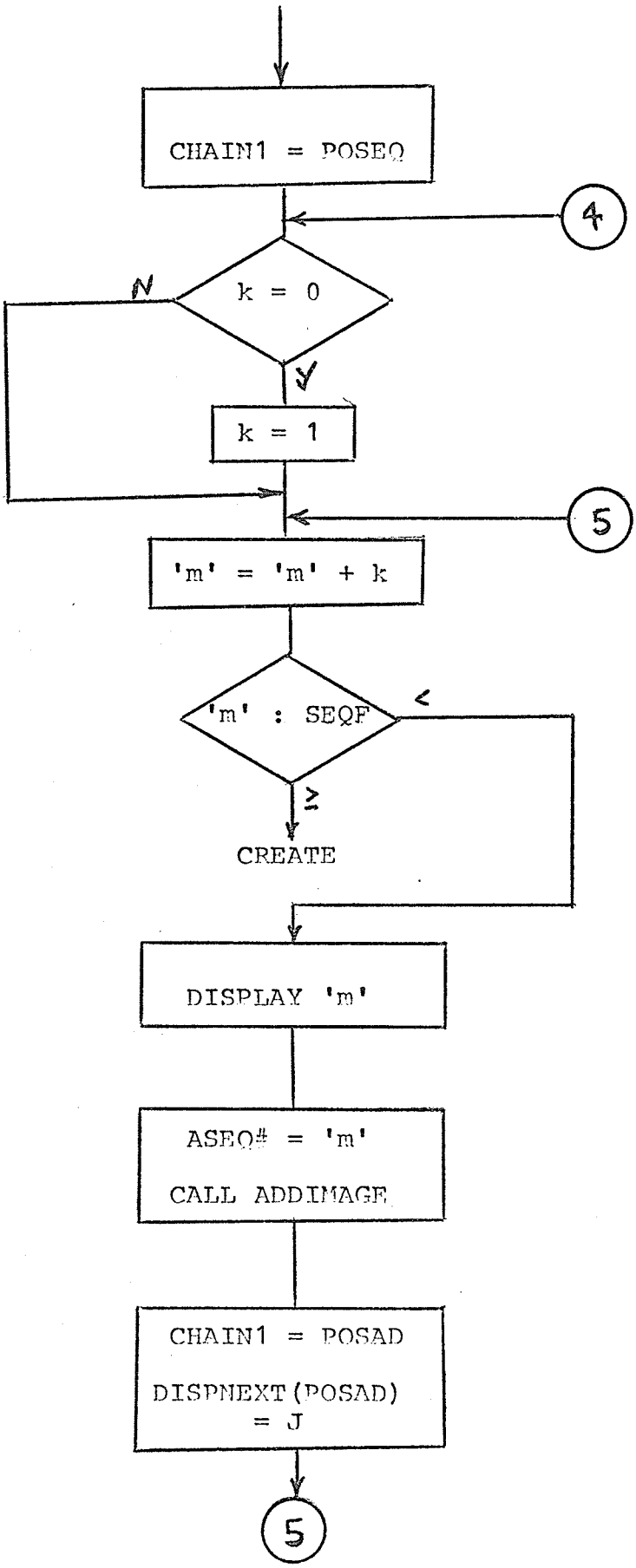

FIGURE 8 [cont.]

Routine LIST

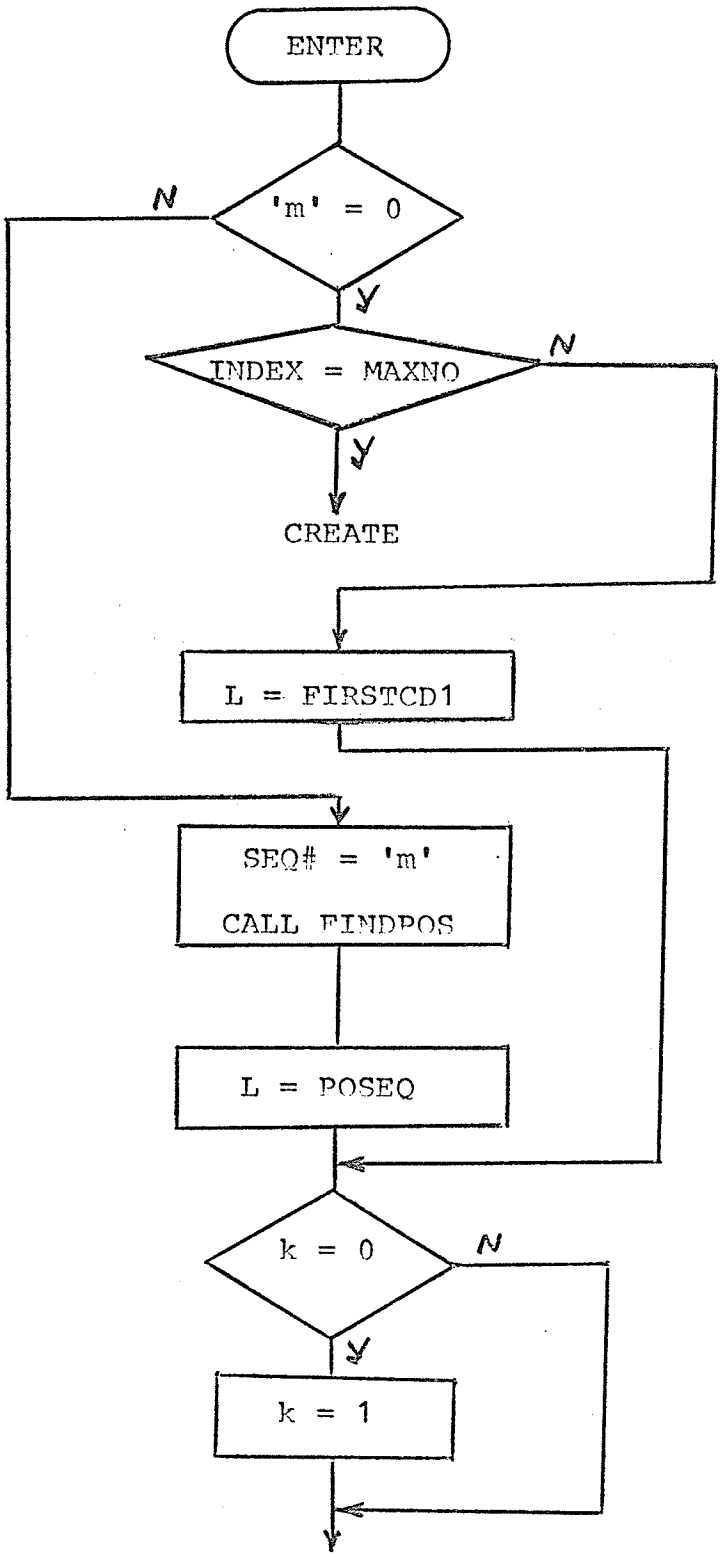

FIGURE 9

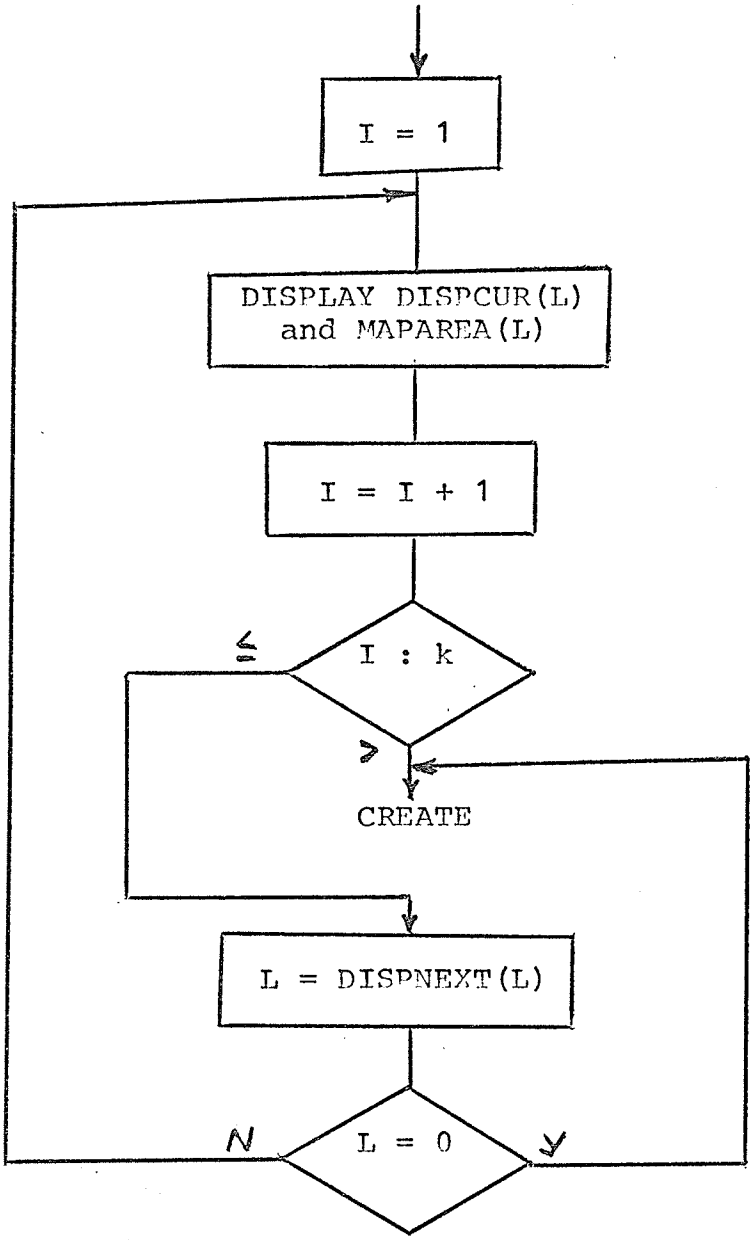

FIGURE 9 [cont.]

routine LIST.

3.4.3.6 Routine RESEQ

Command: $$Q

In response to the "$$Q" command, OSPS resequences all card images. The new sequence numbers are 00010 for the first logical card, 00020 for the second logical card, and so on for all card images in MAPAREA.

Figure 10 illustrates the procedure followed by routine RESEQ.

3.5 Program Assembly

This section describes the storage areas and subroutines used by the OSPS assembler, in addition to its actual operation.

3.5.1 Simulation of SPECTRE Memory

In OSPS, SPECTRE memory is represented by a vector called LOADAREA, consisting of "LITNO"+"MAXNO" elements, where each element is a 10-digit integer. "LITNO" is a parameter which varies according to the environment in which

Routine RESEQ

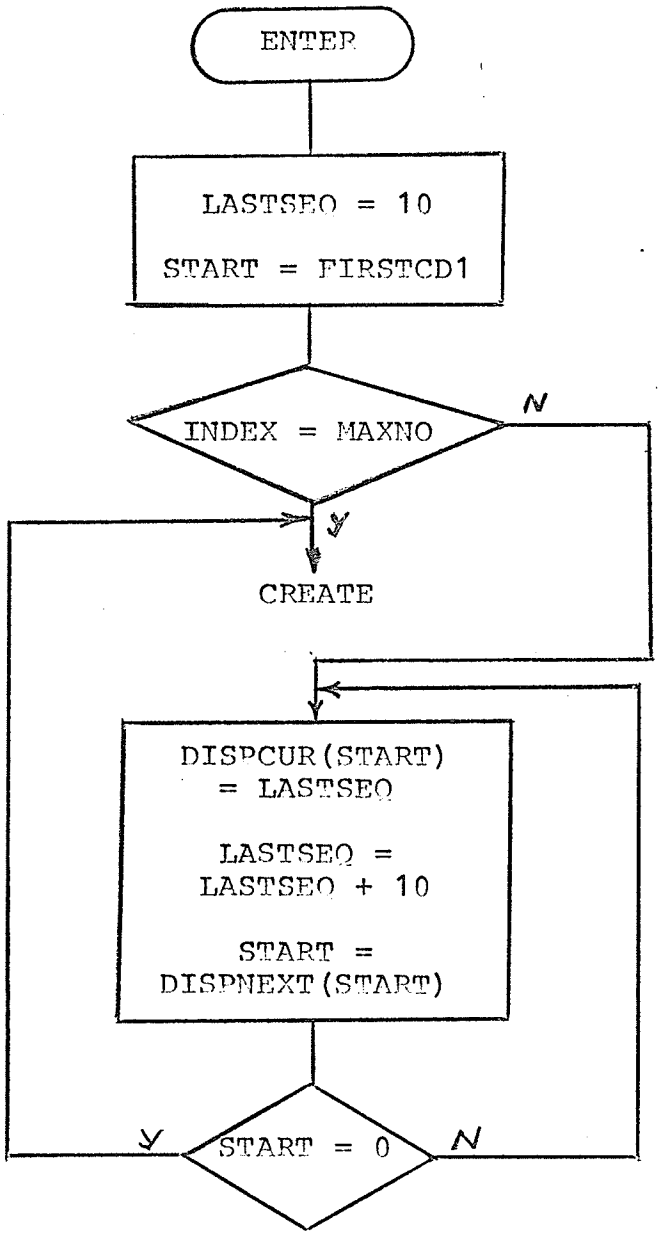

FIGURE 10

OSPS is implemented. The contents of LOADAREA(I) are interpreted as the contents of SPECTRE memory location I. In addition, the registers, AC and MQ, are represented by the 10-digit integers called AC and MQ.

3.5.2 Assembler Error Diagnostics

Assembler error diagnostic information is stored in two vectors, ERCODE and ERPOS. In ERCODE(I) is a pointer to an appropriate diagnostic message in a list of diagnostic messages. In ERPOS(I) the position of the card image in which the error occured is stored. During assembly, error diagnostic information is accumulated in ERCODE and ERPOS, and DPTER points to the next available entry in each of these vectors. When program assembly is completed, the information stored in ERCODE and ERPOS is used to provide a display of diagnostic messages to the user on the terminal.

The vectors ERCODE and ERPOS overlay the storage area reserved for LOADAREA and therefore do not occupy additional storage. In order to avoid over-writing diagnostic information, the assembler uses a switch, ASSERSW, which indicates if an assembler error has occurred, in which case SPECTRE machine instructions or data will not be generated into LOADAREA.

3.5.3 The Symbol Table and the Literal Pool

A. Symbol Table

The symbol table comprises two vectors, SYMAD and SYMLOC, with each vector having "MAXNO" entries. The element SYMLOC(I) is the SPECTRE memory address assigned to the symbolic address in SYMAD(I).

B. Literal Pool

The literal pool comprises two vectors, LITDATA and LITLOC, with each vector having "MAXNO" entries. The element in LITLOC(I) is the SPECTRE memory address assigned to the internal representation of the literal.

Throughout assembly, SPTER will point to the next available entry in the symbol table, while LPTER will point to the next available entry in the literal pool.

3.5.4 Assembly Subroutines

The following subroutines are called by the function routine ASSEMBLE. (see 3.5.5):

1. RECSCAN (scanning card images)
2. AED (storing assembler diagnostic

information)

3. SST (searching the symbol table)

4. SLP (searching the literal pool)

3.5.4.1 Subroutine RECSCAN

(a) Purpose:

The purpose of RECSCAN is to store the contents of each field in a SPECTRE MAP statement, as well as the length of each field, in certain parameter areas which can be accessed during assembly.

(b) Input parameters:

1. RECPOS - the position of the card image to be scanned.

(c) Output parameters:

1. MLABEL - the address of the symbolic address.
2. LMLABEL - the length of the symbolic address.
3. MOP - the address of the mnemonic operation code.
4. LMOP - the length of the mnemonic operation code.
5. MAD - the address of the operand.
6. LMAD - the length of the operand.
7. DISPSW - a switch indicating:

"1" a modifier was found.

Subroutine AED

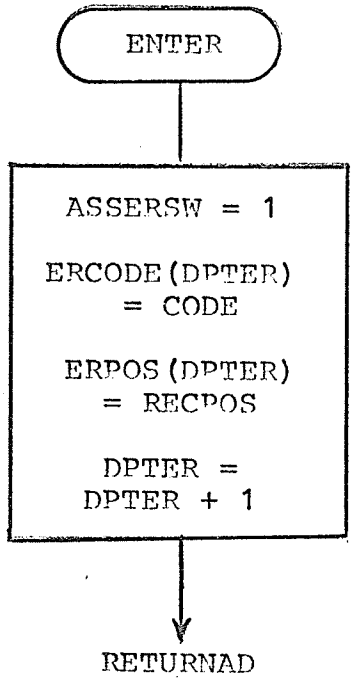

FIGURE 11

"0" a modifier was not found.

8. DISPAD - the address of the modifier.

3.5.4.2 Subroutine AED

(a) Purpose:

Subroutine AED stores diagnostic information in ERCODE and ERPOS, and updates DPTER.

(b) Input parameters:

1. CODE - the pointer to the appropriate diagnostic message in a table of assembler diagnostic messages.
2. RECPOS - the position of the card image in which the error was found.
3. RETURNAD - the point in routine ASSEMBLE to which control is to be returned once the diagnostic information has been stored.

The procedure followed by subroutine AED is illustrated in Figure 11.

3.5.4.3 Subroutines SST and SLP

(a) Purpose:

The purpose of these routines is to search the symbol table/literal pool for a symbolic address/literal data item.

(b) Input parameters:

1. NAME/LITERAL - the symbolic address/literal data for which the search is being conducted.

(c) Output parameters:

1. SENTRY/LENTRY - the entry in the symbol table/literal pool in which the NAME/LITERAL appears. If SENTRY/LENTRY equals SPTER/LPTER, this implies that NAME/LITERAL was not found in the symbol table/literal pool.

See Figure 12 for an illustration of the procedure followed by SST/SLP.

Subroutine SST/SLP

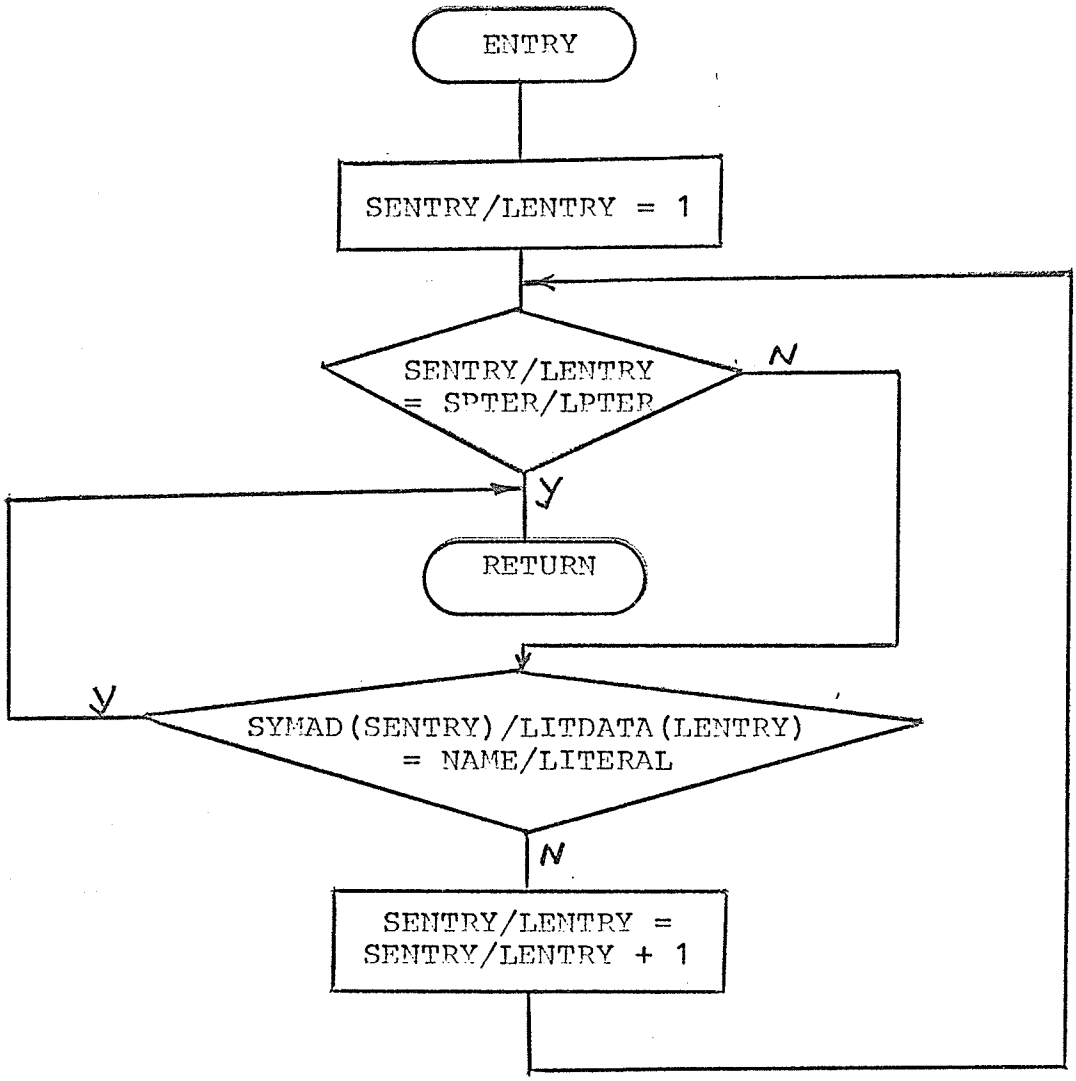

FIGURE 12

3.5.5 Routine ASSEMBLE

Command: $$T

In response to the command "$$T", control is passed to routine ASSEMBLE which assembles the user's SPECTRE MAP program and attempts to load the resulting SPECTRE machine language program into LOADAREA. If errors are encountered during assembly, then OSPS remains in EDIT mode. However, if no errors are found, OSPS will switch into EXECUTE mode.

The 2-pass method of assembly is used by the OSPS assembler. In the first pass the symbol table is constructed, and the numbers/strings encountered as literals, other than in the "CST" instruction, are stored in the LITDATA entries of the literal pool. At the end of the first pass, SPECTRE memory addresses are assigned to each literal and stored in the LITLOC entries of the literal pool.

During the second pass, in the case of executable statements, mnemonic operation codes are converted to their equivalent machine operation codes, address references are resolved into SPECTRE memory addresses, and the resulting machine language instructions are stored in LOADAREA. In the case of the "CST" instruction the literal is converted into

its internal representation and stored in LOADAREA. At the end of the second pass, the LITDATA entries in the literal pool are stored in LOADAREA at the locations indicated by their corresponding LITLOC entries.

3.6 Interactive Program Execution

3.6.1 Instruction Execution

For every SPECTRE machine operation code there exists a subroutine in OSPS which will process it. All routines which are responsible for the execution of instructions assume that ADDRESS holds the SPECTRE memory address found in the address part of the machine language instruction. If the instruction executed was a branch instruction, then the switch, BRANSW, is set to "1".

3.6.2 Interrupts

There are three types of interrupts which can occur during program execution:

1. program
2. input

3. programmer-enabled

3.6.2.1 Program Interrupts

When an instruction is being executed and a programming error is found, program execution is terminated and the status of memory and the AC and MQ is as though the instruction were not executed. When such interrupts occur, a code reflecting the nature of the interrupt is stored in CODE and the interrupt is handled by subroutine INTRUPT (see 3.6.3.1).

3.6.2.2 Input Interrupts

When an input instruction is being executed, program execution is temporarily suspended and control is passed to the programmer so that he can input the data requested by his program. Execution of the program will not resume until valid data has been submitted.

3.6.2.3 Programmer-enabled Interrupts

The user initiates program execution or enables

program execution to continue with the "$$X n" command. The integer "n" is used by OSPS to determine the maximum number of instructions to be executed before control is to be returned to the user. If "n" exceeds "DEFAULT" or equals zero then "DEFAULT" is used in place of "n". The size of the parameter "DEFAULT" is fixed according to the environment in which OSPS is implemented. The resulting number is stored in LIMIT. When "LIMIT" instructions have been executed, the contents of the the SPECTRE memory address from which the next instruction to be executed will be taken are displayed on the terminal preceded by the memory address. Program execution is temporarily suspended and control passes to the user. Program execution will not resume until the user inputs the "$$X n" command.

3.6.3 Interactive Program Execution Subroutines

The following subroutines were developed to be used by the interactive program execution function routines discussed in 3.6.4:

1. INTRUPT
2. CONTROL

3.6.3.1 Subroutine INTRUPT

(a) Purpose:

The purpose of subroutine INTRUPT is to display a diagnostic message to the user, informing him of the nature of a programming error, and of the instruction in which it occured. In addition, the switch DUMPSW is set to indicate to OSPS that program execution cannot be continued.

(b) Input parameters:

1. ICODE - the code of the interrupt.

For an illustration of the procedure followed by INTRUPT, see Figure 13.

3.6.3.2 Subroutine CONTROL

(a) Purpose:

The purpose of this routine is to control program execution. Each machine language instruction is interpreted and control is passed to the appropriate subroutine to process the instruction. When execution of an instruction has been completed, control returns to subroutine CONTROL. At this point the number of instructions which have been executed is checked to see if the maximum number of

Subroutine INTRUPT

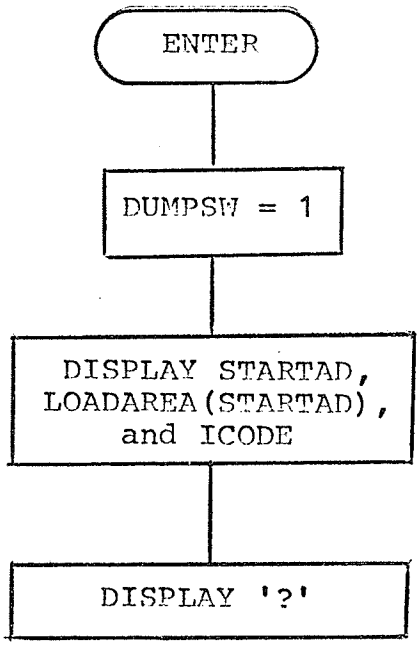

FIGURE 13

instructions "LIMIT" have been executed, in which case a programmer-enabled interrupt occurs. Otherwise, the next instruction is interpreted and the process repeats.

(b) Input parameters:

1. STARTAD - the address of the first instruction to be executed.
2. LIMIT - the maximum number of instructions to be executed before control is to be returned to the user.

For an illustration of the procedure followed by CONTROL, see Figure 14.

3.6.4 Interactive Program Execution Function Routines

The following routines are responsible for the execution of the interactive program execution functions of OSPS:

1. EXEC (start/continue program execution)
2. REXEC (restart program execution)
3. DUMP (memory dump and register display)

Subroutine CONTROL

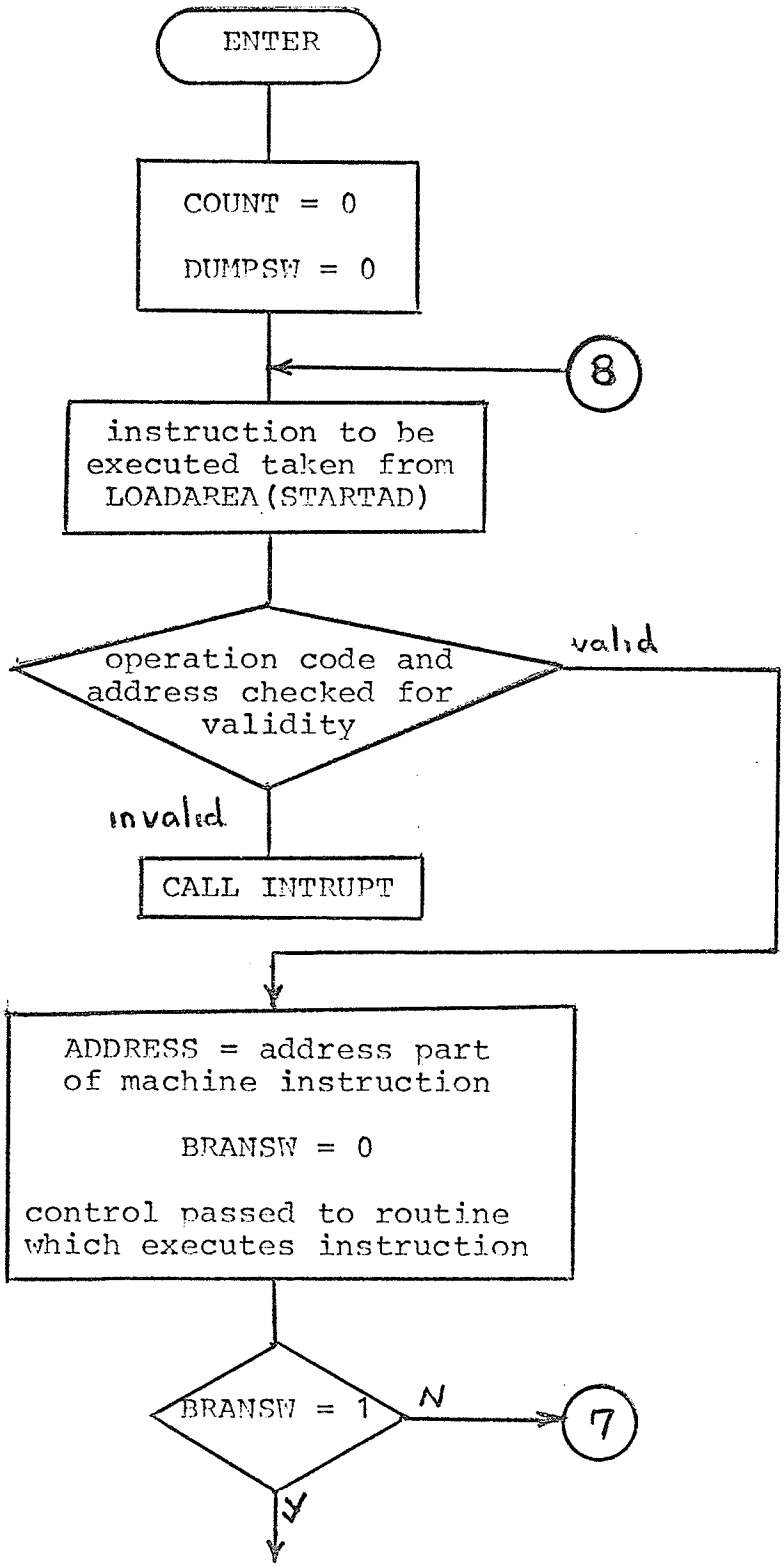

FIGURE 14

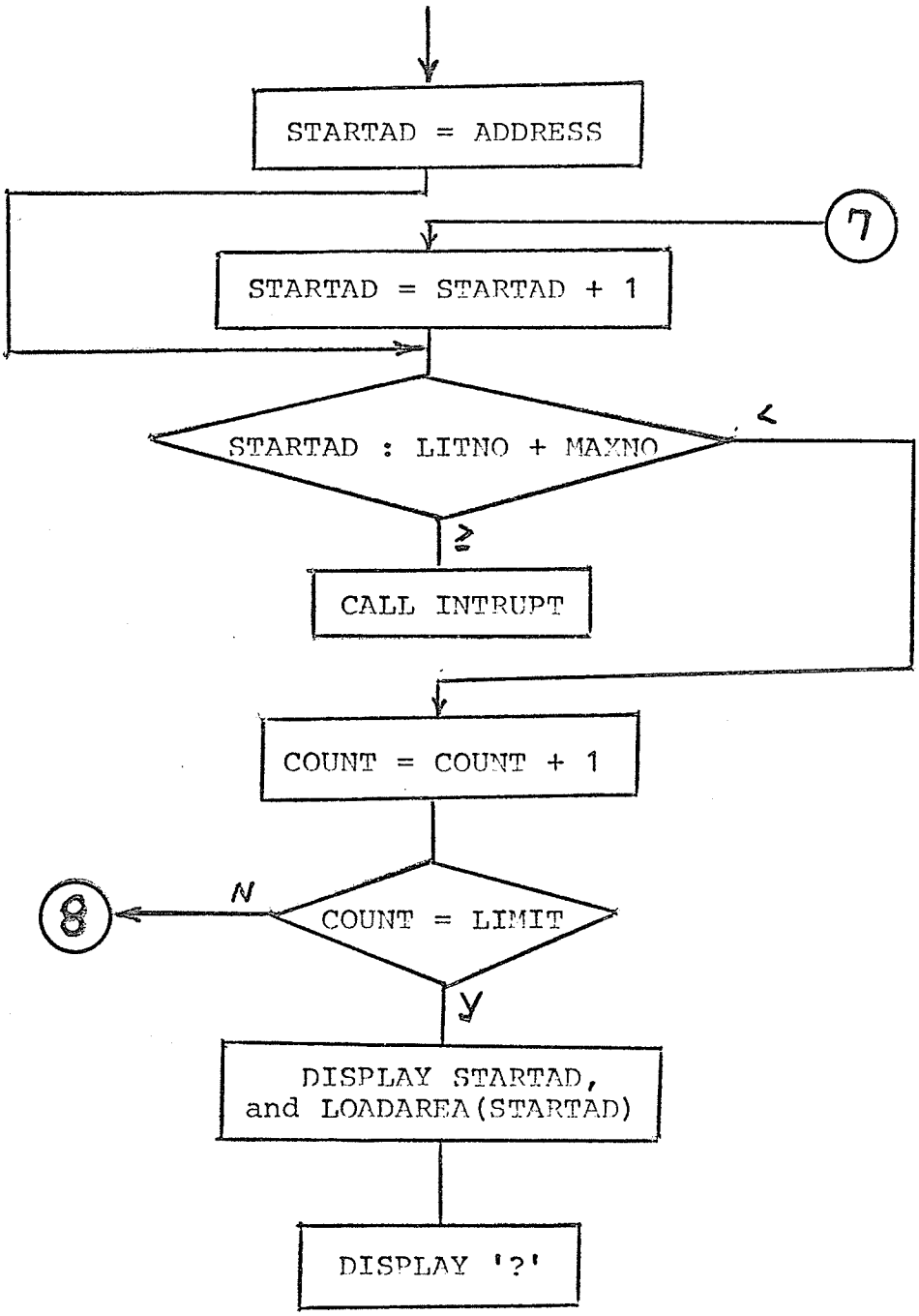

FIGURE 14 [cont.]

3.6.4.1 Routine EXEC

Command: $$X n

In response to the command "$$X n", OSPS starts or continues program execution depending on the current status of the user's program. The maximum number of instructions to be executed before control is returned to the user is stored in LIMIT (see 3.6.2.3), and control is passed to CONTROL.

3.6.4.2 Routine REXEC

Command: $$XR

In response to the command "$$XR", OSPS passes control to the second pass of routine ASSEMBLE. Thus, routine REXEC does not re-execute the user's program. Instead, REXEC re-initializes necessary pointers, switches, and data areas which may have been changed during previous execution of the program. When the next "$$X n" command is input by the user, program execution starts as though the program had just been assembled.

3.6.4.3 Routine DUMP

Command: $$M addr

In response to the command "$$M addr", OSPS displays

the contents of five consecutive SPECTRE memory locations, beginning at the memory location whose symbolic address is "addr". After a display of five words control returns to the user. At this point the user can request a display of the contents of the next five consecutive memory locations by pressing the return key without inputting characters. This process continues until the user either inputs an EXECUTE command, or until the last word in SPECTRE memory is displayed, in which case the contents of the AC and MQ are displayed.

CHAPTER 4

Summary

Most of the problems encountered in the development of OSPS were due to the limitation on its size, 7294 bytes of core storage. This restriction is the result of the fact that MUM stores Problem Oriented Programs (POP's) on separate tracks on the IBM 2314 disk pack which has a capacity of 7294 bytes per track.

As a result of this limitation, OSPS can hold a maximum of twenty card images at any one time. This number could have been increased if some features of OSPS had not been implemented, but since OSPS is to be used primarily as a teaching aid for students learning the SPECTRE computer, it was felt that OSPS should provide the user with these features instead of facilitating the execution of large programs.

If the maximum size of a POP were increased, the additional storage could be used solely for card image storage. For example, if the maximum size of a POP were increased from 7294 bytes to 10K bytes the card image

capacity of OSPS could be increased by sixty-six card images, resulting in a capacity of eighty-six card images. Therefore, OSPS could be used to execute much larger SPECTRE MAP programs.

However, core storage allocated to MAPAREA and LENGTH could have been saved. As an alternative to having information in MAPAREA and LENGTH core resident at all times, the entire MAPAREA and LENGTH vectors could be saved on a backing storage such as disk through the Backing Storage Management facility of MUM. As a result, the user could save his programs and at the same time have the benefits of writing larger SPECTRE MAP programs.

Approximately twenty-eight IBM/360 machine instructions are executed in order to simulate the functions performed by the SPECTRE control unit. The number of IBM/360 machine instructions required to simulate the execution of a SPECTRE machine instruction range from two IBM machine instructions for a load and store instruction to an average of one hundred IBM machine instructions for an input instruction.

APPENDIX A

User's Guide to OSPS

OSPS (the Online SPECTRE Programming System) is a problem oriented program developed to operate under the Manitoba University Monitor (MUM) through remote terminals such as TTY's, IBM selectric typewriters 2740, 2741, 1050, and CRT's. The purpose of OSPS is to provide users with the facility of generating and editing a SPECTRE MAP program on a terminal, having the program assembled, and executing the resulting SPECTRE machine language program in an interactive mode.

OSPS has two modes of operation, EDIT mode and EXECUTE mode. When OSPS is in EDIT mode the user can create and edit a SPECTRE MAP program. When OSPS is in EXECUTE mode the user may exercise a certain amount of control over the execution of the SPECTRE machine language program.

The user establishes communication with OSPS by typing in:

acct#.psswd OSPS,

where "acct#" is the user account number, "psswd" is the user password, and "OSPS" is the name of the program.

Once communication is established, OSPS is in EDIT mode and types out the sequence number 00010 on a new line. At this point, control passes to the user, and he may generate a SPECTRE MAP statement or issue one of the commands of OSPS.

EDIT MODE

In EDIT mode the user creates a SPECTRE MAP program by entering card images with one SPECTRE MAP statement per card image. In EDIT mode the user can:

a. add statements to his program.

b. replace statements in his program.

c. delete statements in his program.

d. insert statements in his program.

e. obtain a list of statements in his program.

f. request assembly of his program into SPECTRE machine code, and as a result switch OSPS into EXECUTE mode.

g. end communication with OSPS.

User Input in EDIT Mode

In EDIT mode the user is given control whenever a sequence number is typed out by OSPS on a new line. At that time the user has three options:

(1) He may type in an EDIT command (see EDIT Commands). OSPS recognizes a command only if the user types in at least three characters, and the first two characters are "$$". The character(s) which

immediately follows "$$" identifies the command. If the command is not in the list of EDIT commands then the diagnostic "ILLEG COMMAND" is typed by OSPS. For both EDIT and EXECUTE commands, the first parameter of the command must either immediately follow the identifier, or be separated from the identifier by blanks. The second parameter must be separated from the first parameter by either a comma and any number of blanks, or simply a sequence of one or more blanks. If parameters are omitted they are assigned a default value of zero.

(2) He may type in a card image. User input which is either fewer than three characters in length, or not immediately preceded by "$$", is recognized as a request to generate a card image. If "m" is the sequence number which was typed by OSPS, then the card image is stored by OSPS and assigned sequence number "m". Subsequent references to that card image may be made using sequence number "m". OSPS places a limit on the number of card images which can be stored. If the user attempts to enter a card image beyond this limit, the diagnostic "NO6ROOM" is typed on the terminal by OSPS.

(3) He may depress the return key without typing in characters. In this case no action is taken by OSPS, other than to type a sequence number on a new line.

EDIT Commands

This section presents the commands which can be entered in EDIT mode. The EDIT commands are listed with their full names followed by a description of the action taken by OSPS in response to the command.

When entering a command which uses parameters, only the capital letter(s) beginning the name can be used. For example, the command $$List n,k must be entered as "$$L n,k". On the other hand, the command $$Create could be entered as "$$CREATE" because it does not use parameters.

$$Create

OSPS types out a sequence number on a new line and control is passed to the user. If the user enters a card image, OSPS responds by adding the card with that sequence number to the logical end of his SPECTRE MAP program. The sequence number is then incremented by ten and typed on a new line. This process continues until:

1. the user enters a new command.
2. the user has entered the maximum number of card images.

This procedure is followed in three additional cases:

1. when the actions taken in response to other EDIT commands have been completed
2. when OSPS has typed out a diagnostic message on the terminal
3. when the user presses the return key without typing in characters

$$Replace m

This command is used to replace a card image in the SPECTRE MAP program.

In response to this command, OSPS types out the card image whose sequence number is "m". Immediately following this, sequence number "m" is typed on the next line. Control is then passed to the user. If the user enters a card image, then it will replace the card image which was typed on the above line. Otherwise, the card image displayed on the above line is not replaced.

POSSIBLE DIAGNOSTICS:

"ILLEG #" There is no card image with sequence number "m".

$$Delete m,n

This command is used to delete statements from the SPECTRE MAP program.

In response to this command, all card images from sequence number "m" to sequence number "n" inclusive are deleted. If only one card image is to be deleted then "n" can be omitted.

POSSIBLE DIAGNOSTICS:

"ILLEG #" Either there are no card images with sequence numbers "m" and "n", or sequence number "m" is greater than sequence number "n".

$$Insert m,k

This command is used to insert card images in the user's program between the card image whose sequence number is "m" and the next logical card image.

In response to this command, OSPS types out sequence

number "m+k" on a new line. If the user inputs a card image, OSPS inserts the card image with that sequence number into his program. Immediately following this, the sequence number is incremented by "k" and typed on a new line. This process continues until:

1. the user enters a new command.
2. the sequence number "m+yk", where "y" is an integer, is greater than or equal to the sequence number of the card image which logically followed the card image with sequence number "m" prior to insertion.
3. the user presses the return key without entering characters.

If "m" is zero then insertion occurs before the first logical card image in the user's program. If "k" equals zero then "k" is assigned a default value of one.

POSSIBLE DIAGNOSTICS:

"ILLEG #" Either:

a. "m" is a non-existing sequence number.
b. "m" is the sequence number of the last logical card image.
c. "k" is an invalid increment ("k" must be a positive integer).

$$List m,k

This command is used to list SPECTRE MAP program statements.

In response to this command, "k" card images are typed on separate lines, starting with sequence number "m". Listing of card images terminates either when "k" card images have been listed, or when the last logical card image has been typed.

If "m" equals zero then listing will start from the first logical card image. If "k" equals zero then "k" is assigned a default value of one.

POSSIBLE DIAGNOSTICS:

"ILLEG #" Either:

a. "m" is a non-existing sequence number.

b. "k" is invalid ("k" must be a positive integer).

$$Queue

This command is used to assign new sequence numbers to all card images in the user's program.

In response to this command, OSPS assigns sequence number 00010 to the first logical card image, 00020 to the second, and so on for all card images in the user's program.

$$Translate

This command is entered by the user when he wishes to have his SPECTRE MAP program assembled and prepared for execution in EXECUTE mode.

In response to this command, OSPS assembles the user's SPECTRE MAP program into SPECTRE machine code. If errors are encountered during assembly, a list of error diagnostics will be typed on the terminal (see Assembler Error Diagnostics), and OSPS remains in EDIT mode. On the other hand, if assembly errors are not found, OSPS switches into EXECUTE mode and types out "?" on the terminal.

$$End

This command is used to end communication with OSPS.

In response to this command, communication is terminated between OSPS and the user. The user's program is not retained and must be re-created if it is to be executed at some future time.

Assembler Error Diagnostics

This section presents a list of possible assembler diagnostics along with an explanation of each diagnostic.

Each diagnostic is typed on a separate line and, with the exception of the diagnostic "NO END CARD", is preceded by the sequence number of the statement in which the error occurred.

"INV OPCD" The operation code is not a valid SPECTRE MAP mnemonic instruction.

"INV LIT" The literal in the operand field is not a valid SPECTRE MAP literal (see 2.1).

"INV LAB" The symbolic address in the symbolic address field is not a valid SPECTRE MAP symbolic address (see 2.1).

"INV OPND" This error will occur in one of the following cases:

1) The operand in a "CST" instruction is not a literal.
2) The operand in a shift instruction, "ORG" instruction, or "RES" instruction is not a positive integer.
3) The modifier is not an integer.
4) The operand field in an "END" instruction is

empty.

"DUP NAME" The symbolic address in the symbolic address field has been defined in some other statement.

"UNDEF SYM" The symbolic address in the operand field is not defined in the program.

"CORE EXED" There is no space available in SPECTRE memory in which SPECTRE machine code can be loaded. This error occurs as the result of an attempt to store machine code beyond the limits of SPECTRE memory.

"NO END CARD" There is no "END" instruction in the program.

EXECUTE MODE

In EXECUTE mode, the user communicates with the SPECTRE machine language program which has been generated from his SPECTRE MAP program. In EXECUTE mode the user can:

a. start execution of his program.

b. provide a parameter which will cause OSPS to suspend execution of his program and return control to him.

c. request a dump of any part of SPECTRE memory, or the AC and MQ registers.

d. enable his program to continue execution.

e. provide data to his program when it is requested.

f. obtain a list of the statements in his SPECTRE MAP program.

g. switch back into EDIT mode.

h. end communication with OSPS.

User Input in EXECUTE Mode

In EXECUTE mode, the user is given control in two instances:

a. when OSPS types out "?", prompting the user to enter

an EXECUTE command.

b. when OSPS types out "i STG" or "i NUM", prompting the user to provide data to his program ("STG" indicates strings and "NUM" indicates numbers).

(a) In response to "?", the user can:

1. type in an EXECUTE command (see EXECUTE Commands).
2. press the return key without typing in any characters. With the exception of the $$Map EXECUTE command, OSPS responds by immediately re-typing "?" on a new line.

If the user types in characters which are not preceded by "$$", or if he types in fewer than three characters, then OSPS immediately re-types "?" on a new line.

(b) In response to "i STG" or "i NUM" the user can:

1. type in an EXECUTE command (see EXECUTE Commands).
2. enter data to his program. The letter "i" in the messages is an integer from one to five,

and indicates to the user the number of data items which are requested by his program. OSPS recognizes data input only if the user types in characters which are not preceded by "$$", or if the user types in fewer than three characters.

When the data has been entered, it is checked for validity (see 1.1.4.1). If the data is invalid, OSPS types out the message "INV DATA" and then the message "i STG" or "i NUM" on a new line. If the data is valid, the user's program continues execution. Therefore, if invalid data is entered, OSPS allows the user to re-enter the data. However, program execution will not continue until valid data is supplied by the user.

3. press the return key without typing in any characters. In response to this OSPS types the message "INV DATA" and the message "i STG" or "i NUM" on a new line.

EXECUTE Commands

$$Xecute n

This command is used to initiate the beginning or resumption of program execution.

As a result of this command, the user's program starts or continues execution, depending upon the previous state of his program. The parameter "n" is used by OSPS to determine the number of instructions which are to be executed before control is to be returned to the user. When control is returned after "n" instructions have been executed, OSPS types out the contents of the word in SPECTRE memory from which the next instruction will be taken. This is followed by a display of the contents of the AC and MQ registers, and finally "?" on a new line.

A default value of one hundred is assigned to "n" if "n" is zero or greater than one hundred.

POSSIBLE DIAGNOSTICS:

"ILLEG #" "n" is not a positive integer.

$$XR

This command is used to prepare the program for re-execution.

In response to this command, OSPS reloads the SPECTRE machine language program into SPECTRE memory and

types out "?" on a new line. At this time the status of the user's program is as though it had just been re-assembled and loaded into SPECTRE memory. The next "$$X n" command entered by the user will re-start program execution.

$$Map address

The user enters this command in order to obtain a dump of part of SPECTRE memory.

In response to this command, OSPS displays the contents of five consecutive SPECTRE words, beginning with the word whose symbolic address is "address". Following this, OSPS types out "?" on a new line. At this point, if the user presses the return key without entering any characters, OSPS displays the contents of the next five consecutive words of SPECTRE memory. This process continues until:

1. the last word in SPECTRE memory is displayed. In this case, the contents of the AC and the MQ are displayed on the next two lines, followed by "?" on a new line. At this point, if the user presses the return key without entering any characters, OSPS re-types "?" on a new line.
2. the user enters an EXECUTE command

If "address" is omitted, or if "address" is undefined in the user's SPECTRE MAP program, then OSPS displays only the contents of the AC and MQ.

$$Create

In response to this command, OSPS switches back to EDIT mode and then follows the same procedure as in EDIT mode (see EDIT Commands).

$$List m,k

In EXECUTE mode, the user enters this command in order to obtain a list of statements in his SPECTRE MAP program.

In response to this command, OSPS temporarily switches to EDIT mode and follows the same procedure used in EDIT mode (see EDIT Commands). However, when listing has completed, OSPS returns to EXECUTE mode and types out "?" on a new line.

$$End

Communication between OSPS and the user is

terminated.

Termination of Program Execution

The user's program will terminate execution in two cases:

a. the program has completed execution

b. a programming error has occurred

(a) When a program completes execution, OSPS types out the message "EX END" followed by "?" on a new line. If the user wishes to have his program re-executed, he must enter the "$$XR" command before entering the "$$X n" command. If the "$$X n" command is entered before entering the "$$XR" command, then OSPS types the diagnostic "ILLEG COMMAND" followed by "?" on a new line.

(b) When a programming error occurs, an error code explaining the nature of the error (see Programming Error Codes) and the contents of the word containing the instruction in which the error occured are displayed, along with the contents of the AC and MQ. Following this, "?" is typed on a new line. At this

point the use of the "$$X n" command is restricted as mentioned in (a) of this section.

Programming Error Codes

"OP" invalid operation code

"AE" addressing exception

"AO" arithmetic overflow

"EO" exponent overflow

"EU" exponent underflow

"DE" divide exception

"OC" out of core (the current instruction is beyond the limits of SPECTRE memory)

Summary of OSPS Commands

A. EDIT Commands

1.	$$C	(create)
2.	$$D m,n	(delete)
3.	$$I m,k	(insert)
4.	$$R m	(replace)
5.	$$L m,k	(list)
6.	$$Q	(resequence)
7.	$$T	(assemble)
8.	$$E	(end)

B. EXECUTE Commands

1.	$$X n	(execute)
2.	$$XR	(re-execute)
3.	$$M addr	(dump)
4.	$$C	(create)
5.	$$L m,k	(list)
6.	$$E	(end)

Implementation Restrictions of OSPS

In the current environment, the user may have at most twenty statements in his SPECTRE MAP program with a

maximum of twenty characters per card image. In addition, the capacity of SPECTRE memory is thirty words.

As a result of the twenty character capacity for card images it may be required that the length of a symbolic address in the symbolic address field of a SPECTRE MAP statement be decreased, or that the symbolic address in the symbolic address field be eliminated in order to allow the entire operand to be read as part of the statement.

APPENDIX B

Programs Run Under OSPS

In this section, three SPECTRE MAP programs which were run under OSPS are illustrated. Although SPECTRE MAP programs are currently limited to twenty statements, the examples show that OSPS can be used to perform simple tasks. In order to distinguish OSPS output from user input, output from OSPS will appear in upper case letters while user input will appear in lower case letters.

Example 1

This example illustrates a program which calculates square roots of floating point numbers using the Newton-Raphson method. The program request "1 NUM" is followed by the floating point number whose square root is to be calculated. Immediately following this is the internal floating point representation of the square root preceded by its SPECTRE word address.

```
ENTER ACCT #,PRG NAME
30.cia osps
00010      fin out x
00020      st  rn1 x +1
00030        ldq x +1
00040        fdv =2.0
00050      he sto x +2
00060        ldq x +1
00070        fdv x +2
00080        fad x +2
00090        sto x +3
00100        ldq x +3
00110        fdv =2.0
00120        sto x
00130        fsu x +2
00140        ssp
00150        fsu =1.0e-4
00160        tle fin
00170        cla x
00180        tra he
00190      x res 4
00200        end st
00210      $$t
?
```

```
$$x
1 NUM
5.0
018 +5122360679
1 NUM
7.0
018 +5126457513
1 NUM
9.0
018 +5130000000
1 NUM
10.0
018 +5131622776
1 NUM
11.0
018 +5133166247
1 NUM
16.0
018 +5140000000
1 NUM
$$e
```

Example 2

This example illustrates a program which functions like a desk calculator. It performs addition, subtraction, multiplication, and division on floating point numbers. The program request "2 NUM" is followed by the user input which consists of a floating point number and an integer code indicating the operation to be performed ("1" - addition, "2" - subtraction, "3" - multiplication, "4" - division). The first operand is assumed to be the floating point number in the AC register. Following each user input the internal representation of the floating point number currently in the AC is typed preceded by the SPECTRE word address in which it is stored.

Computing (((2.0 * .9E5) + 54.0) * 3.0)/ 18.4 in steps:

```
ENTER ACCT #,PRG NAME
30.cia osps
00010     aq sto ac
00020       out ac
00030     st rn2 n
00040       cla cod
00050       add =23
00060       als 3
```

```
00070         add =20
00080         sto ex
00090         cla ac
00100         ldq ac
00110      ex
00120         tra ag
00130         org 20
00140      n
00150      cod
00160      ac
00170         end st
00180      $$t
?
$$x
2 NUM
.9e5 1
022 +5590000000
2 NUM
2.0  3
022 +5618000000
2 NUM
54.0 1
022 +5618005400
2 NUM
```

```
3.0  3
022 +5654016200
2 NUM
18.4 4
022 +5529356630
2 NUM
$$e
```

Example 3

This example illustrates a program which requests input of a machine program that functions like an adder for integers. The first two input requests by the program are followed by integers which are assumed to be SPECTRE machine instructions. Following this, control passes to the machine program which was input, and subsequent requests for program input are followed by integers. After an integer has been input, the program types out the current total preceded by the SPECTRE word address in which it is stored.

```
ENTER ACCT #,PRG NAME
30.cia osps
00010        org 8
00020      s rn5 * -8
```

```
00030        rn1 * -4
00040        tra * -10
00050        end s
00060      $$t
?
$$x
5 NUM
10006   71007   20007   1006   81006
1 NUM
50000
1 NUM
1
006 +0000000001
1 NUM
2
006 +0000000003
1 NUM
3
006 +0000000006
1 NUM
4
006 +0000000010
1 NUM
5
```

```
006 +0000000015
1 NUM
6
006 +0000000021
1 NUM
7
006 +0000000028
1 NUM
$$e
```

References

1. Brillinger, P.C., B.L. Ehle, and J.W. Graham, "An Introduction to the SPECTRE Computer", Department of Applied Analysis and Computer Science, University of Waterloo, 1969.
2. Abraham, C.I., "Scientific Reports No. 1", Department of Computer Science, University of Manitoba, 1970.