RCA
ELECTRONIC
501

DATA PROCESSING SYSTEM

THE RCA 501
SCIENTIFIC INTERPRETER
MANUAL

RADIO CORPORATION OF AMERICA

Electronic Data
Processing Division
Camden, New Jersey

The data herein presented is subject to minor change without notice.

November 1960

CONTENTS

CHAPTER 1
INTRODUCTION

WHAT IS AN INTERPRETIVE SYSTEM?

All Electronic Computing Systems are designed to recognize and respond to a code which is unique to the system. The code that is recognized by a particular machine system is called its machine code. The machine code consists of a number of instructions which cause some internal operations to take place. Usually, the operation caused by any one machine instruction is very small in relation to the processing required for the solution of any problem.

A trained person, called a programmer, is generally required to effect the solution of a problem on an electronic computing system. The programmer must be thoroughly familiar with the machine code and the logical operations of the machine system with which he is working. The person desirous of a solution to some problem must, therefore, present his problem to a programmer. He must make the programmer thoroughly familiar with every detail of the problem before the programmer can write a "program" to effect a solution of the problem.

The program is a series of machine instructions, which when executed by the computer will perform the desired processing.

An Interpretive System is a program written in machine code. It causes the computer to operate upon a program written in some code other than machine code. This other code is called a pseudo-code. The Interpretive System is written to recognize a set of pseudo-code instructions, called pseudo-instructions, and to initiate a set of machine instructions, called a subroutine, for each pseudo-instruction in the pseudo-code program. The subroutine will cause the computer to perform the operation specified by the pseudo-instruction.

The RCA 501 Scientific Interpreter System is designed for the inexperienced programmer who desires to use the RCA 501 Data Processing System for the solution of engineering and mathematical problems. As such, its pseudo-code is directly related to the natural language of mathematics. The user of the RCA 501 Scientific Interpreter System need not become familiar with the internal workings of the machine system. He need only acquire some basic concepts of computer technique, must know enough mathematics to solve the problem and must understand the problem to be solved thoroughly.

Thus, with the use of the Scientific Interpreter, the scientist, the man with the problem, with little additional training, can write the program and achieve the solution directly on the RCA 501 Computing System.

The RCA 501 Scientific Interpreter System is convenient and simple to use, economical in its memory requirements, and performs at speeds which enable the user to employ it effectively.

In order for the user of the Scientific Interpreter to make better use of the system, he should have some general knowledge of the RCA 501 Electronic Data Processing System. The following pages are, therefore, a general description of the machine system.

INTRODUCTION TO THE RCA 501 ELECTRONIC DATA PROCESSING SYSTEM

The RCA 501 Electronic Data Processing System is a machine system designed to process information. It can perform mathematical and logical operations. Mathematical operations include the arithmetic functions of addition, subtraction, multiplication and division. Logical operations include comparisons, transfer of information, and determination of the next instruction to be performed.

The RCA 501 System receives information and performs the proper mathematical and logical operations upon it to produce the desired solutions. Since it is an electronic data processing system, it is capable of handling large volumes of information and performing long series of complex calculations with extreme accuracy.

All these functions are accomplished with lightning speed. The System is currently being used to great advantage in both business and scientific applications. Let us examine the components of the RCA Data Processing System and see how they combine to make a complete system.

THE RCA 501 COMPUTER

Specific instructions are written to control the operation of the Computer. These instructions are called the Program. Under the control of these programmed instructions, the Computer is the element which performs the mathematical and logical operations.

High Speed Memory

The High Speed Memory is a magnetic core device within which information is stored. The memory is available in increments of 16,384 character locations and may be expanded to a maximum of 262,144 locations. Each location is individually addressable and can store any one of sixty-four characters.

Program Control

The Program Control interprets the Program and thereby directs the Computer in the performance of the various mathematical and logical operations upon the data stored in the High Speed Memory. Simultaneously, automatic accuracy checks are executed.

The Console

The Console provides a means of monitoring the operations of the Computer. Automatic and manual operation, maintenance, program insertion and program testing are accomplished at the Console.

PERIPHERAL EQUIPMENT

The RCA 501 System has certain component parts that are referred to as peripheral gear. The peripheral gear that is directly controlled by the Computer is known as On-Line equipment. Those pieces of equipment that operate independently of the Computer are referred to as Off-Line equipment. Therefore as each component is described herein, it will be specified as On-Line or Off-Line equipment.

Tape Stations — On-Line (Input-Output Device)

The Tape Stations are the major input-output devices of the RCA 501 System. They are directly controlled by the Computer. Information is read into and written out of the Computer on magnetic tape through the Tape Stations. A Computer may have as few as two and as many as sixty-two Tape Stations that are directly connected to it. Tape Stations can magnetically read or write information on magnetic tape at the rate of 33,333 characters per second.

Paper Tape Reader — On-Line (Input Device)

Punched paper tape is read into the Computer through an input device called the Paper Tape Reader. This device reads information at the rate of 1,000 characters per second.

Tapewriter and Tapewriter Verifier — Off-Line (Input Device)

The Tapewriter and Tapewriter Verifier is a keyboard-operated, input device. Information is punched on paper tape and simultaneously printed on hard copy on the Tapewriter. In conjunction, the Tapewriter Verifier automatically checks punched information for accuracy by comparison with a previously prepared punched paper tape (prepared on a Tapewriter). Both devices function at a typing speed of up to 10 characters per second.

Monitor Printer — On-Line (Output Device)

The Monitor Printer is an output device that is directly controlled by the Computer. This is an on-line device that prints information received from the High Speed Memory as hard copy. When desired, information can be punched into paper tape and printed as hard copy at the same time.

The Monitor Printer prints at the rate of 10 characters per second, and its primary use is for printing low-volume outputs, operational program control and program testing.

Computer Paper Tape Punch — On-Line (Output Device)

The Computer Paper Tape Punch is an output device that is electrically connected to the Computer. It is capable of producing either 5-level punched paper tape or 7-level punched paper tape.

This device operates at the rate of 100 characters per second and a tape speed of up to 10 inches per second.

Also available is a High Speed Paper Tape Punch which operates at the information rate of 300 characters per second and a tape speed of up to 30 inches per second.

Card Transcriber — Off-Line (Input Device)

The Card Transcriber is an input conversion device. It accepts punched cards, translates the information from 80-column punched cards to RCA coded characters onto magnetic tape at the rate of up to 400 cards per minute. The magnetic tape is then used as an input device to the Computer.

The Card Transcriber consists of two units: the Card Reader and the Card Editor. The Card Reader may be used without the Editor. In this case, the Computer must provide the translation and editing.

In either case, each card is read through two card reading stations to provide an automatic accuracy check.

Transcribing Card Punch — Off-Line (Output Device)

The Transcribing Card Punch is an output conversion device. It accepts information, recorded in RCA code, on magnetic tape and translates it into 80-column card code, punching the information into cards. It punches cards at the rate of approximately 150 cards per minute.

The device reads the information from magnetic tape through two reading stations which provide for an automatic accuracy check. The Card Punch can be utilized without its editor in that the Computer must translate the information from RCA code into 80-column card code. It then arranges and selects the information under program control before writing onto tape from which the information is to be punched.

On-Line Printer (Output Device)

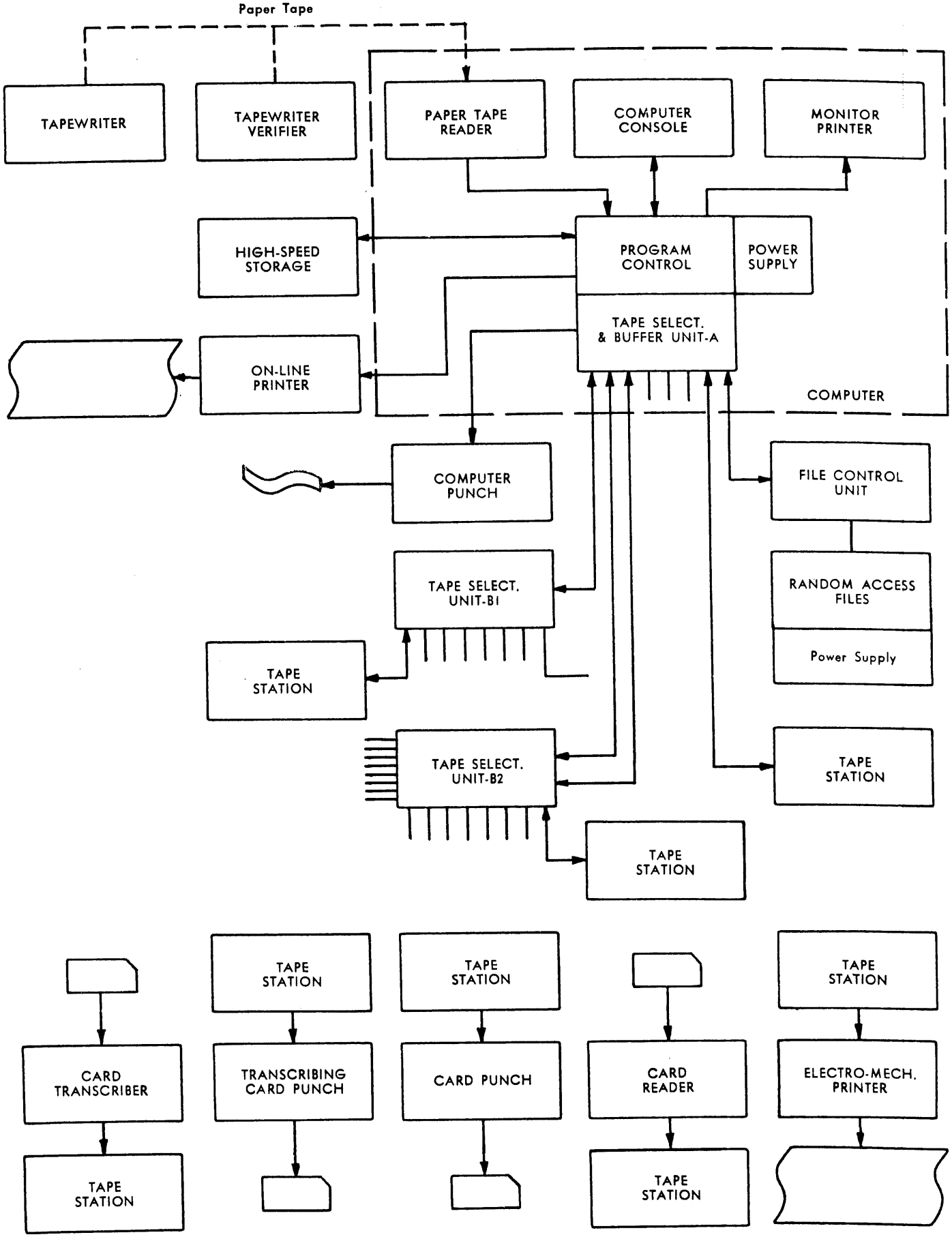
The On-Line Printer is a high speed printer that is electrically connected to and controlled by the Computer. Information is printed from the Computer High Speed Memory onto the output documents. The printer unit prints typographical characters on continuous edge-perforated (blank or pre-printed) paper as instructed by the Program. The On-Line Printer prints at the rate of 600 lines per minute, with as many as 120 characters per line.

Electro-Mechanical Printer — Off-Line (Output Device)

The Electro-Mechanical Printer is a High Speed Printer that operates independently of the Computer and has its own mechanical and electronic unit.

The Off-Line Printer operates under the control of a plugboard. Selection, arranging and editing of data is provided by manually wiring the plugboard (external programming) and inserting it into the printer unit. Information to be printed is read from a magnetic tape through the Off-Line Printer Tape Station.

The Off-Line Printer prints at the rate of 600 lines per minute. A line is composed of as many as 120 characters.



RCA 501 System—Example of Equipment Interconnection

CHAPTER 2 FEATURES

GENERAL

The RCA 501 Scientific Interpreter is designed to provide a facile programming tool for the inexperienced coder. Its pseudo-code consists, in the main, of a set of easily remembered instructions which permit the user to perform the usual arithmetic operations, obtain the elementary transcendental functions of a single variable, and express recursive processes in a form which is close to the basic notions of programming a digital computer. The sundry chores which are a part of most machine code programs are absorbed by the subroutines of the system, with which the user need not concern himself. In addition, the instructions are designed so as to approximate the manner in which one normally thinks of a computer operation.

The RCA Scientific Interpreter also includes a number of features designed for use by the experienced programmer. Some of these features are:

a. Segmentation.

b. The ability to write a program in parts.

c. The ability to use machine code if desired.

d. The ability to include interpretive subroutines as part of his current program.

INTERPRETER OPERATIONS

The interpretive pseudo-code provides for the following operations:

a. Floating-Point Arithmetic Operations.
   1. Add
   2. Subtract
   3. Multiply
   4. Negative Multiply
   5. Divide
   6. Negative Divide
   7. Vector Multiply
   8. Polynominal Multiply

b. The Logarithmic Functions of a Single Variable.
   1. Common Logarithm
   2. Natural Logarithm

c. The Trigometric Functions of a Single Variable.
   1. Sine
   2. Cosine
   3. Arctangent

d. Exponentiation.
   1. e^x
   2. 10^x

e. The Square Root of a Number.

f. Fixed Point Operations.

g. Logical Operations.
   1. A greater than test
   2. An equality test
   3. A less than test
   4. An exponent test
   5. Comparison of significance
   6. Unconditional transfer of control
   7. Transfer to Machine Code

h. Data Handling Control.

i. Execute Subroutine.

j. Address Modification.

k. Looping.

l. Stop.

m. Skip.

n. Overlay Operations.

INTERPRETIVE COMPUTER HARDWARE

The interpretive system is designed so that it will operate with only a minimum configuration of RCA 501 Computer hardware which includes the following:

a. One module of core storage.

b. Two tape stations.

c. On-line monitor printer.

d. On-line or off-line electromechanical printer.

e. Paper tape reader.

It should be emphasized that the hardware mentioned above is the **absolute minimum.** Additional hardware will, in general, enhance the operations of the interpreter system.

PROGRAM TESTING

To assist the user in checking and correcting any errors that might occur in programming, the following features are incorporated into the system:

a. A tracing routine which prints the complete result of the execution of each pseudo-instruction as it is executed.

b. A tracing routine which documents only those pseudo-instructions which break the normal sequence of pseudo-instruction execution.

c. Complete editing of the program.

d. A routine by which the user may make corrections to his problem with a minimum of effort.

MEMORY STRUCTURE

The high-speed memory of the RCA 501 Computer provides storage area for the program, the interpreter system and data. Information is represented in memory by the decimal digits 0 to 9, the letters A through Z, punctuation symbols, typewriter symbols, etc. The minimum configuration of computer hardware (one module of core storage) is capable of storing 16,384 characters, however, the interpreter system can make use of three modules of core storage (if available) which have the combined capacity of storing 49,152 characters. The available memory is considered to be divided into three specific areas:

a. Area occupied by the subroutines of the system (sequences of instructions used to implement the various interpreter functions).

b. Area occupied by the machine coding inserted into the program by the experienced programmer.

c. Area occupied by the interpretive memory.

Subroutine Space in the Memory

The space in machine memory occupied by the machine coding which constitutes the interpreter program is denoted by the symbol M0. The amount of memory space occupied by M0 varies from program to program, since certain subroutines of the interpreter are stored in machine memory only if they are required by the pseudo-code program.

Machine Coding Space in the Memory

The memory space occupied by any machine coding written to supplement the pseudo-coding is called M#. When M# exists in a program, it immediately follows M0.

Interpretive Memory Space

The space in the memory provided for the interpretive program is denoted by the symbol MP and begins wherever M# ends. The interpretive memory stores the following types of quantities:

a. Pseudo-code instructions.

b. Fixed and floating-point constants.

c. Floating-point data.

SUBROUTINES OF THE SYSTEM

The subroutines of the system divide into two parts: the mandatory subroutines which are automatically placed in a fixed sector of memory by the interpreter system; the optional subroutines which are inserted in the memory at the request of the programmer. The latter are placed in memory in such a manner that one follows the other without any waste of memory space. The entrance addresses are stored in a list of subroutine entries. As each optional subroutine is brought into memory, the subroutine name and its location in memory are printed by the Monitor Printer.

INTERPRETIVE MEMORY AND INTERPRETIVE ADDRESSING

Starting Point

The interpretive memory section begins at the first available machine address following the storage of the necessary interpreter subroutines and machine coding. The starting address of the interpretive memory is always called A00. The actual value of the beginning address of the interpretive memory, A00, is computed and stored for internal use by a pre-edit routine. For purpose of general information the actual value is printed in the course of the edit phase of the pre-edit.

Units of the Interpretive Memory

Interpretive memory is divided into twelve-character words. The words are arranged in banks, 100 words to the bank. Interpretive memory may have a maximum of 26 banks (lettered A through Z), in addition to M0 and M#, provided that machine memory has the capacity for the maximum number of words.

Addressing

Since the interpretive memory is thought of in terms of words and banks, all pseudo-code addressing is in terms of these units only. Reference to a word is made by means of three characters, the first being the bank designation and the last two the ordinal of the word in the bank. For example:

A17 signifies the 18th word in bank A (the 1st bank).

B33 signifies the 34th word in bank B (the 134th word of the interpretive memory).

D14 signifies the 15th word in bank D (the 315th word of the interpretive memory).

A00 is by definition the 12-character sector of memory following the last line of machine coding in memory.

Quantities Stored in Interpretive Memory

The types of quantities stored in the interpretive memory are: pseudo-code instructions, fixed and floating-point constants, and floating-point data. Each of these quantities is an interpretive word occupying 12 character locations. The floating-point constants and data are numbers written to conform to the format acceptable by the Floating-Decimal Arithmetic Routine. Fixed-point constants are strings of characters which are the operands (any one of the quantities entering into an operation) of the fixed add and subtract interpretive operations (See Fixed-Point Constants).

PSEUDO-CODE CHARACTERISTICS

General

A pseudo-code instruction is an interpretive word. The symbols assigned to each of the 12 characters of this word are as follows:

CHARACTER	1	2	3	4	5	6	7	8	9	10	11	12
SYMBOL	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}

The content of the various character positions vary from instruction to instruction; in general the following can be stated:

a. The content of C_1, denoted by (C_1) or p, is the patch number (first character position in the interpretive word).

b. The content of C_2, denoted by (C_2) or OPR, is the operation code (second character position in the interpretive word).

c. The content of C_3, denoted by (C_3) or i, is the pseudo-address modifier number (third character position in the interpretive word).

d. Characters $C_4C_5C_6$, $C_7C_8C_9$, and $C_{10}C_{11}C_{12}$ are treated as single units and usually represent the addresses of the operands of instructions. In some instances these characters are decimal numbers whose meaning is defined by the particular instruction. Whenever the triple-character unit refers to a pseudo-code address, then we denote:

the content of $C_4C_5C_6$, by **a**
the content of $C_7C_8C_9$, by **b**
the content of $C_{10}C_{11}C_{12}$, by **c**

Pseudo-Code Instruction Format

Each pseudo-code instruction consists of 12 RCA characters. The names attached to each character are shown in the following table:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3

THE PSEUDO-CODE INSTRUCTION CHARACTER SYMBOLS

The set of triples $a_1a_2a_3$, $b_1b_2b_3$, $c_1c_2c_3$, act as units of information and are referred to as **a**, **b**, **c** respectively.

A psuedo-code instruction is therefore the string of symbols represented by:

p OPR i **a b c**

The elements of the instruction code are further identified below.

THE PATCH NUMBER (p)

The first character position in a pseudo-code word is occupied by the **patch number** denoted by p. The use of the patch number is covered in detail in Chapter 6. It is used as a correction aid. Initially all instructions are written with p designated as an item separator symbol (ISS) •.

In Chapter 3, where the formats of the pseudo-instructions are given, an item separator symbol entered in the p position indicates that this instruction may have no other p number. A check symbol (✓) in this position means that some number may be entered there.

THE OPERATOR (OPR) CHARACTER

The operator character, the second character in the pseudo-instruction, is symbolically represented by OPR and represents the operation to be executed. The range of symbols assumed by the OPR is:

0 through 9
*
,
A through Z

THE ADDRESS MODIFIER (i) CHARACTER

The address modifier character, the third character in the pseudo-code instruction, is represented symbolically as i and represents the number of the pseudo-address modifier which is to be used to effect automatic address modification of the operand addresses in the pseudo-instruction prior to its execution. The instruction is executed as if the contents of the modifier were added to the addresses of the instructions. However, the instruction stored in interpretive memory remains unchanged.

The Intrepreter provides seven pseudo-address modifiers denoted by the decimal numbers 1 through 7; a zero character in the i position implies that no address modification is required prior to the execution of the instruction. A pseudo-address modifier occupies nine contiguous RCA characters in order that simulataneous modification of the three operand addresses may be made at once.

There are some exceptions to the meaning of the i characters. These will be covered as they arise. In Chapter 3, where the formats of pseudo-instructions are given, a zero entered in the i position indicates that this instruction may not make use of the automatic address modifier. A check ($\surd$) in this position means that the i character may be used.

THE A-ADDRESS (a) FIELD OF CHARACTERS

The fourth, fifth, and sixth character positions of the pseudo-instruction, denoted by $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, respectively, constitute the A-Address-field characters which are a pseudo-address (bank letter and two decimal numbers) or a three-digit decimal number. Usually the **a**-field contains the address of an operand; however, there are a number of exceptions and these are described in detail as these exceptions arise.

THE B-ADDRESS (b) FIELD OF CHARACTERS

The seventh, eighth, and ninth character positions of the pseudo-code instruction, denoted by $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$, respectively, constitute the B-Address-field of characters which are a pseudo-address (bank letter and two decimal numbers) or a three-digit decimal number. Usually the **b**-field is used to show an address of an operand; however, there are a number of exceptions and these are described in detail as the exceptions arise.

THE C-ADDRESS (c) FIELD OF CHARACTERS

The tenth, eleventh, and twelfth character positions of the pseudo-code instruction, denoted by $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, respectively, constitute the C-Address-field of characters which are a pseudo-address (bank letter and two decimal numbers) or a three-digit decimal number. Usually the **c**-field contains the address of an operand; however, there are a number of exceptions and these are described in detail as these exceptions arise.

CHAPTER 3 THE PSEUDO-CODE

The pseudo-code program is written on the RCA 501 Scientific Interpreter Program Record Sheet, Form RCA 1213, shown on page 13. In the column headed "HSM LOCATION", the programmer enters the address (bank letter and two decimal digits) of each instruction. Instructions or fixed-point constants are written on consecutive lines. Instructions are performed by the Interpreter System in sequential order, unless the sequence is interrupted by some instruction whose function it is to interrupt the sequence. When an interruption of a sequence occurs, it is said that a transfer of control has occurred. The next instruction to be executed, then, is not the next instruction in sequence, but some other instruction in the program. The address of this instruction is given in the sequence-interrupting instruction. When a transfer of control occurs, the Interpreter System continues sequential operations from that point in the program to which control was transferred until another sequence-interrupting instruction is encountered.

THE PSEUDO-INSTRUCTIONS

The pseudo-code is designed to provide the user with a convenient tool to perform the following operations:

a. Arithmetic Operations.
   1. Arithmetic Floating-Decimal Operations
   2. Transcendental Operations
   3. Fixed-Point Arithmetic Operations

b. Logical Operations.

c. Manipulation of Data Input/Output.

d. Specialized Operations.

e. Miscellaneous Operations.

A complete list of symbols used in the pseudo-code instructions is given in Appendix H.

FLOATING-DECIMAL ARITHMETIC OPERATIONS

The format of the arithmetic operations is as follows:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	√	A/B/C D/E/F G/H	√	**a** is a Pseudo-address			**b** is a Pseudo-address			**c** is a Pseudo-address		

These operations, without exception, are interpreted as follows:

The contents of location **a** and the contents of location **b** are the operands, and the result of the operation is stored in location **c**. Symbolically this is written as:

(**a**) OPR (**b**) → (**c**)

RCA 501 INTERPRETER PROGRAM RECORD

PSEUDO — CODE

TITLE

CODER

REMARKS

DATE

NAME

BLOCK NO.

SEGMENT NO.

					<	•					>					
FROM INSTRUCTION	HSM LOCATION	<•	P	OPR	i	A			B			C			REMARKS	CHART REF.
			1	2	3	4	5	6	7	8	9	10	11	12		

FORM NO. 1213 2.5M 4-60

Repertoire of Arithmetic Instructions

The following chart shows the repertoire of the arithmetic instructions indicating the type of operation, associated OPR symbol, and interpretation.

OPERATION	OPR SYMBOL	INTERPRETATION
Add	A	(a) + (b) → (c)
Subtract	B	(a) — (b) → (c)
Multiply	C	(a) × (b) → (c)
Divide	D	(a) ÷ (b) → (c)
Negative Multiply	E	— (a) × (b) → (c)
Negative Divide	F	— (a) ÷ (b) → (c)
Vector Multiply	G	(a) × (b) + (c) → (c)
Polynominal Multiply	H	(a) × (b) + (c) → (a)

Examples:

1. •C0 A10 A11 C00
   is interpreted to mean:
   (A10) × (A11) → (C00)
   where: the contents of the **a** and **b** operands are floating-point numbers, and the floating-point result will be stored in **c**.

2. •G0 A10 A11 C00
   is interpreted to mean:
   (A10) × (A11) + (C00) → (C00)
   where: the contents of all shown words are floating-point numbers.

TRANSCENDENTAL OPERATIONS

The pseudo-code provides a convenient means for obtaining the elementary transcendental functions of a single variable. Both function and argument are floating-point numbers expressed in appropriate units. The format of a transcendental operation is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	√	S/T/U/ V/W/X/ Y/Z/	√	**a** is a Pseudo-address			**b** is 0 0 0			**c** is a Pseudo-address		

Repertoire of Transcendental Instructions

The following chart shows the repertoire of the transcendental instructions indicating the type of operation, associated OPR symbol, and interpretation.

OPERATION	OPR SYMBOL	INTERPRETATION
Square Root	S	Square root **(a)** → **(c)**
Common Logarithm	T	Log_{10} **(a)** → **(c)**
Natural Logarithm	U	Log_e **(a)** → **(c)**
10 to a Power	V	$10^{(a)}$ → **(c)**
e to a Power	W	$e^{(a)}$ → **(c)**
Sine	X	Sin **(a)**[1] → **(c)**
Cosine	Y	Cos **(a)**[1] → **(c)**
Arctangent	Z	Arctan **(a)** → **(c)**[2]

[1] **(a)** is assumed to be a number in radians.

[2] **(c)** is an argument expressed in radians.

The **b** field remains unused in this set of operations; three zeros must be placed there.

FIXED-POINT OPERATIONS

Fixed-point arithmetic is made available for purposes of address modification only. Since the address portion of a pseudo-instruction occupies the **a, b, c** field, a fixed point constant is written in the form

•00 $a_1a_2a_3$ $b_1b_2b_3$ $c_1c_2c_3$

where: $a_1a_2a_3$, $b_1b_2b_3$, $c_1c_2c_3$ are either decimal numbers or pseudo-addresses.

Repertoire of Fixed-Point Instructions

The following chart shows the repertoire of the fixed point instructions indicating the operation, associated OPR symbol, and interpretation.

OPERATION	OPR SYMBOL	INTERPRETATION
Add	N	**(a)** + **(b)** → **(c)**
Subtract	O	**(a)** — **(b)** → **(c)**

Examples:

1. •N0 A10 B12 C13
   where: (A10) = •C0 F22 F23 G29
   (B12) = •00 001 002 093
   then (C13) = •C0 F23 F25 H22

2. •N0 B12 A10 C13
   for the same (A10) and (B12)
   (C13) = •00 F23 F25 H22

The **a**, **b**, and **c** fields of the operands are added separately to form three independent sums stored as a single word. Note that the leftmost three characters of the result are those which are found in the leftmost three characters of the operand specified in the **a**-address of the fixed-point arithmetic instruction.

It should be pointed out that decimal complementation cannot be performed. The fixed-point subtract must be used.

Care should be taken not to add, say,

E14 = 000 A14 B13 C59

to

D17 = 000 B93 A81 C59

because the sum is not likely to be of use to the programmer. As a practical rule the programmer should either add constants, or augment a constant employing bank designations by a constant which has no bank designations in it.

LOGICAL OPERATIONS

This set of pseudo-instructions is made up of instructions which can interrupt the sequence of program execution. These instructions are usually called logical instructions.

Repertoire of Logical Instructions

The following chart shows the repertoire of the logical instructions indicating the operation, associated symbol, and interpretation.

OPERATION	OPR SYMBOL	INTERPRETATION
Test if Greater	I	If (**a**) > (**b**) → transfer control to **c**
Test if Equal	J	If (**a**) = (**b**) → transfer control to **c**
Test if Smaller	K	If (**a**) < (**b**) → transfer control to **c**
Test Exponents	L	exp (**a**) ≤ exp (**b**) → transfer control to **c**
Compare Significance	7	significance ≤ **a** → transfer control to **c**
Compare to Z's	Q	**a** : Z's If = → transfer control to **b**. If ≠, to **c**
Unconditional Transfer	8	Transfer control to **c**

The first four test operations have the following format:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	✓	I/J/ K/L	✓	**a** is a Pseudo-address			**b** is a Pseudo-address			**c** is a Pseudo-address		

i.e., the contents of the **a** and **b** addresses are used for testing and the result is used as a criterion for transfer of control to **c**, according to the operator symbol used. Failure of the test results in the execution of the next instruction in sequence.

Compare Significance

Since the Floating-Decimal Arithmetic may be significance sensitive, a test of significance is incorporated into the set of simple logical instructions. The operator symbol is 7 and its format is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	✓	7	✓	**a** is a decimal number			**b** is blank 0 0 0			**c** is a Pseudo-address		

The instruction is read to mean:

If the number of significant digits of the previous arithmetic result is smaller than or equal to the decimal quantity **a** then transfer control to **c**. The number **a** is a decimal number from 001 to 008.

Sentinel Z Test

The Q instruction compares the contents of a word to a sentinel automatically supplied by the Read instruction (see page 19). It is used to test for the fact that the programmer has exhausted the data contained in a block. Its format is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	•	Q	✓	**a** is address of word to be tested			**b** is address of Pseudo-instruction			**c** is address of Pseudo-instruction		

The Q instruction compares the contents of the word specified by **a** to 000ZZZZZZZZZ. If equal, control will be transferred to the pseudo-instruction whose address is specified by **b**. If not equal, control will be transferred to the pseudo-instruction whose address is specified by **c**.

Unconditional Transfer of Control

The instruction which provides for an unconditional transfer of control is of this form:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	•	8	✓	**a** is a decimal number			**b** is blank 0 0 0			**c** is a Pseudo-address		

This instruction is interpreted as follows:

The decimal quantity **a** identifies a breakpoint button on the RCA 501 console. If **a** is 000, or if **a** designates a breakpoint switch that is not set, control is transferred to **c**.

The programmer can use the breakpoints 3, 4, 5, which are written in the pseudo-code as 003, 004, 005, respectively. If the breakpoint button indicated in the instruction is on, then the succeeding pseudo-instruction is executed and the control is not transferred to **c**.

DATA MANIPULATION AND INPUT/OUTPUT

There are five instructions designed to perform the commonly useful data handling operations. These divide into:

1. Data transfer instruction
2. Read/write instructions
3. Printing instructions

The Data Transfer Instruction

The data transfer instruction is designed to move an arbitrary set of contiguous words in memory from one position to another. The MOVE operation, denoted by the operator symbol M, is of the following format:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	✓	M	✓	**a** is a Pseudo-address			**b** is a Pseudo-address			**c** is a Pseudo-address		

It is interpreted to mean: Move the sector of memory which begins with the word stored in address **a** and terminates with the word stored in address **b** to the sector of memory beginning with the address **c**; e.g.,

•M0 A15 A30 C15

means transfer the sector of memory from the pseudo-address A15 through A30 inclusive to the sector of memory bounded by the pseudo-addresses C15 and C30.

Although **c** of the M instruction indicates the beginning address of the destination area, the actual transfer in the RCA 501 Computer takes place from right to left. Care must be taken, therefore, not to overlap sections of memory thereby destroying desired information. For example:

•M0 A85 A99 A84

operates in the following manner:

A99 moves to A98 overlapping the previous contents of A98.

A98, which has just been moved from A99, now moves to A97, etc.

When this instruction is completed the contents of A99 will have been transferred to every word from A84 to A98. This overlap technique is useful for initializing a sector of memory.

The Read-Write Operations

Floating-point data[1] on magnetic tape or paper tape is assumed to be in blocks of variable size; e.g., block 1 may contain 10 words of data, while block 2 may contain 27 words of data, etc. A block of data is, as provided for by the RCA 501 System, an arbitrary string of characters on magnetic tape separated by a gap from other units of information. To perform a READ or WRITE instruction the tape trunk number must be specified. Tape trunk numbers are written in pseudo-instructions as the decimal numbers 000 through 024. Usually 000 is used to represent the Paper Tape Reader in the Read Instruction and the Monitor Printer in the Write Instruction. The remaining numbers represent individual trunk numbers. The programmer assigns a unique pseudo-number to each of his input and output tapes, and uses these pseudo-numbers consistently throughout his program. At the time of running the program, actual trunk numbers are assigned to each pseudo-number used in the program through a simple operator action. The procedure for doing this is covered in Chapter 4.

The format for the READ instruction is as follows:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	✓	5	✓	Trunk Number			Number of blocks			Pseudo-address		

The READ instruction causes the number of blocks specified in **b** to be read from the tape station (or the Paper Tape Reader) specified in **a**, into consecutive memory locations starting from the address specified in **c**. The sentinel, 000ZZZZZZZZZ, is then placed in the word following the last word read in by this instruction. The actual machine address of the location into which the last character read from tape was placed, is inserted into machine memory location, $(000577)_8$.

The format of the WRITE instruction is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	✓	6	✓	Trunk Number			**b** is a Pseudo-address			**c** is a Pseudo-address		

Print Data Instructions

The pseudo-code provides for the following two printing operations:

OPERATION	OPR SYMBOL
Print on Monitor Printer	3
Print on On-Line Printer	4

The format of the instruction is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	√	3/4	√	**a** is a decimal number			**b** is a Pseudo-address			**c** is a Pseudo-address		

and is interpreted to mean:

Print on the Monitor or On-Line Printer the sector of memory bounded by the **b** and **c** addresses, **a** words per line of print, the range of **a** being 001 to 008. The information printed will be edited according to the assumption that the sector of memory specified contains **only** floating-point numbers.

SPECIALIZED OPERATIONS

Programming entails a set of operations of frequent occurrence which are not usually present in machine code. A set of such operations, constructed so as to yield maximum power, are incorporated into the pseudo-code.

Execute a Subroutine

The operator (OPR) symbol used is R and the format of the instruction is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	•	R	√	**a** is a Pseudo-address			**b** is a Pseudo-address			**c** is a Pseudo-address		

This instruction is interpreted to mean the following:

Transfer control to the instruction designated by **c** and execute the subsequent instructions until the one designated by **b** is reached. At this point, transfer control to **a**. It is assumed that the location specified by **b** is reserved by the programmer. Its previous contents will be overlaid by a transfer of control instruction.

Example:

PSEUDO-ADDRESS		PSEUDO-CODE
A01		
A02		•R0 A03 A13 A10
A03		•K0 B19 D27 A32
A04		•R0 A01 A13 A10
A05		
.		
.		
.		
Subroutine S	A10	
	A11	
	A12	
	A13	•80 000 000 A03 •80 000 000 A01

The instruction in A02 instructs the program to insert the instruction "Transfer Control to A03" in line A13, and to follow this by a transfer of control to A10 which initiates the execution of the subroutine. Following execution of the subroutine, control is transferred to A03. The instruction in A04 has a similar effect except that it places a "Transfer Control to A01" into A13. Thus control, after the execution of the subroutine, is now transferred to A01.

PSEUDO-ADDRESS MODIFICATION

The principal vehicle of address modification is the pseudo-address modifiers, which are fixed 9-character sectors of memory. There are seven of these, denoted by 1, 2, 3, 4, 5, 6, 7, and with each is associated an increment which is also a fixed, 9-character sector of

memory. Setting an address modifier involves the storage of fixed-point constants into the address modifier and its increment.[1]

Since address modification is designed to operate on interpretive memory addresses, and a pseudo-instruction contains three addresses, the content of the address modifier consists of a triplet of addresses and the process of incrementation is the same as that of fixed-point addition.

Set Address Modifier

Setting of address modifiers is effected by the instruction having the operator (OPR) symbol 1, and has the format:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	√	1	√	**a** is a decimal number			**b** is a Pseudo-address			**c** is a Pseudo-address		

a is the numeric tag (001, 002, ... 007) of the address modifier to be set, and the instruction is interpreted as follows:

Transfer the 9-character fixed-point constant located in **b** into the pseudo-address modifier **a**; transfer the 9-character fixed-point constant in **c** into the increment associated with address modifier **a**.

Example:

•10 002 A27 A28

where: (A27) = 000 000 000 000
(A28) = 000 001 000 010
is interpreted to mean:
Set address-modifier 2 to (0, 0, 0); i.e., 000 000 000
Set its increment to (1, 0, 10); i.e., 001 000 010

If this instruction is itself modified, it reads as follows: Set the content of pseudo-address modifier **a** to be the sum of the contents of **b** and pseudo-modifier i; set the increment of pseudo-address modifier **a** equal to the contents of **c**.

Example:

•13 004 D17 E19

where: (D17) = 000 A05 A00 B00
(E19) = 000 000 001 001
pseudo-address modifier 3 = 000 010 003.
is executed as follows:

Address-modifier 4 is set to the sum of (**b**) and modifier 3; namely:
A05 A00 B00 + 000 010 003 = A05 A10 B03
The increment to address modifier 4 is set to (**c**); namely:
000 001 001.

Note the unique function of address modification by pseudo-modifier i with this instruction; i.e., a fixed-point addition of the modifier i to the content of **b** only.

[1] The actual memory locations of the address modifiers and their increments are listed in Appendix C.

LOOP

A convenient means for iterating through a set of instructions is provided by the LOOP instruction; the operator symbol employed is 2 and its format is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	•	2	√	**a** is a decimal number			**b** is a Pseudo-address			**c** is a Pseudo-address		

This instruction is interpreted as follows: **a** denotes a pseudo-address modifier. The increment associated with modifier **a** is added to the content of **a** and the sum is stored in the location of modifier **a**. This new value of the modifier **a** is compared to the content of **b**. If **a** is less than **b** control is transferred to **c**, otherwise the next instruction in sequence is executed. (Note that the increment of address modifier **a** is not affected by this operation.)

Example: •20 007 B29 A13

where: (B29) = 000 100 000 100
address modifier 7 = 072 000 072
increment to modifier 7 = 009 000 009

is executed as follows:

Pseudo-address modifier 7 is replaced by the sum of 072 000 072 and 009 000 009, i.e.
081 000 081
This sum 081 000 081 is compared to 100 000 100, and since the sum is less than **b**, control is transferred to **c**.

If a loop instruction is modified, it reads as follows: Store the sum of (**a**) and its increment into (**a**). Compare this sum to the sum of the contents of **b** and address modifier i. If (**a**) is less than (**b**) plus (i) transfer control to **c**; otherwise execute the next instruction in sequence.

Example: •21 002 C13 G16

where: (C13) = •A5 C00 D00 E00
address modifier 1 = 000 005 015
address modifier 2 = A05 B06 B00
increment of modifier 2 = 001 001 001

Then this example is executed as follows:

Pseudo-modifier number 002 is replaced by the sum
A05 B06 B00 and 001 001 001
i.e.,
A06 B07 B01

The sum of C00 D00 E00 and 000 005 015 is computed as
C00 D05 E15.

Then A06 B07 B01 is compared against C00 D05 E15. Since the inequality A06 B07 B01 < C00 D05 E15 is satisfied, control is transferred to the instruction located at G16.

MISCELLANEOUS OPERATIONS

The STOP Instruction (,)

The STOP instruction is written:

/,0 000 000 000

This instruction will cause the computer to **stop** with the interpreter ready to execute the next instruction in sequence. If the **start** button is depressed, the program execution will continue.

The SKIP Instruction (0)

The SKIP instruction is written:

/00 000 000 000

This instruction will cause the interpreter to ignore this word. The **a**, **b**, and **c** addresses, however, may contain fixed-point constants to be used for address modification. It is these nine (9) character locations which are generally used in conjunction with the set, loop, fixed-point add, and fixed-point subtract instructions.

Overlay Instruction

— See Chapter 4

Transfer to Machine Code

— See Chapter 5

FLOATING-POINT CONSTANTS[1]

Since Floating Decimal Arithmetic instructions assume all operands to be in floating-point format, the programmer must write constants which are to enter floating-point arithmetic operations in proper format.

Floating-point constants are considered to be separate blocks of coding, therefore, the programmer can set aside a bank or more of memory for his constants. As the need for a constant arises, he writes the constant on a separate sheet in the following format:

< • XXXXXXXX ± EEE ± >[2]

where: < is a special RCA 501 symbol called the Start Message Symbol.
• is a special RCA 501 symbol called an Item Separator Symbol (ISS). The ISS represents the decimal point.
± is the sign associated with the constant.
EEE is a three digit exponent.
± is the sign of the exponent.
> is a special RCA 501 symbol called the End Message Symbol.

Each floating-point constant will occupy one word. They are addressed by the same pseudo-addresses as data; e.g., B10, C15, D96, etc.

[1] For floating-point data format see Appendix A.
[2] Note that this message contains 14 characters. However, this is converted to proper 12 character format during the pre-edit.

CHAPTER 4
PROGRAMMING AIDS

GENERAL

The interpreter system is designed to provide a number of features to facilitate the coding, debugging, and operational flexibility of the system. These features include the following:

a. An Edit of the Program
b. Means of tracing the programs via two modes
c. Flexibility of tape trunk designation
d. Automatic patching and correcting
e. Ability to insert subroutines written in pseudo-code into the body of a pseudo-program
f. Ability to write pseudo-code in parts
g. Ability to write overlays (program part), which replace a portion of a program segment

PROGRAM EDIT

In the course of the pre-edit, the Scientific Interpreter prints on the On-Line printer, or prepares a tape for printing on the Off-Line printer, a complete edit of the program. A sample of a portion of an edit is shown on the following page. The edit shows each instruction written by the programmer in two parts on a line. The left hand part is each pseudo-instruction as written by the programmer. The right hand part is the same pseudo-instruction with all pseudo-addresses replaced by their corresponding machine addresses.

The edit prints each block as an entity, giving the block number, segment number and starting location of each block.

TRACING A PROGRAM

Execution of the trace subroutine may be accomplished in two distinct modes, namely the normal mode and the logical mode. In the normal mode, execution of the trace subroutine implies that following the execution of each pseudo-instruction, information relevant to the indicated operation is printed on the On-Line printer or is written into the tape for Off-Line editing. In the logical mode, execution of the trace subroutine implies that following the execution of a pseudo-instruction which involves a break in the sequence of program step execution, the pseudo-address of the instruction executed and the one to which transfer of control is to be made, are printed on the Monitor Printer.

Initiation of either the normal or logical trace mode is controlled by the breakpoint function of the RCA 501 Computer. Whenever breakpoint 1 of the computer is set, the normal trace mode is in effect. When breakpoint 2 is set, the logical trace print is in effect. Following the execution of each pseudo-instruction, the instruction and pertinent results are printed as listed on subsequent pages.

RCA 501 SCIENTIFIC INTERPRETER PRE-EDIT

PROGRAM ENTRY A 00 013120

BLOCK 1 FROM 000 013060

LINE	CONTROLS	OP	A-ADDRESS	NN	B-ADDRESS	LINE	OP	A-ADDRESS	NN	B-ADDRESS
000	# 8 8	12	000010	40	770000	013060	12	000010	40	770000
001	# 8 8	71	001700	00	000000	013070	71	001700	00	000000
002	# 8 8	44	554301	56	450161	013100	44	554301	56	450161
003	# 8 8	64	557500	00	000000	013110	64	557500	00	000000

Tracing in either mode can be initiated anywhere in the program by use of a memory address stop (MASP). The programmer instructs the operator to set MASP to the machine address[1] of the pseudo-instruction at which the trace is to start. When the program stops at the MASP, the programmer then sets breakpoint 1 or 2, depending on which mode of tracing he desires.

The trace in normal mode prints a single line in the following order:

a. The word TRACE

b. The pseudo-address in which the pseudo-instruction is stored

c. The pseudo-instruction itself

d. The content of the pseudo-address modifier used in the instruction

e. The content of **a**

f. The content of **b**

g. The content of **c**

In all instances care is taken to print only relevant information in the input language so that the results printed are of use to the programmer in debugging. (A sample of part of a normal trace edit is shown on the next page.)

In summary we have the following features:

a. A normal trace mode controlled by breakpoint 1.

b. A logical trace mode controlled by breakpoint 2.

c. The beginning of either trace modes is controlled by memory address stop (MASP) and by setting breakpoints 1 and 2 during program execution.

d. The printouts of a logical trace mode are written on the monitor printer.

e. The printouts from a normal trace mode are written on the On-Line printer or onto tape for Off-Line editing.

TAPE TRUNK DESIGNATION

The user, in writing his program, employs symbolic designations for tape trunk numbers (000 through 023). Their meaning in terms of RCA 501 trunk numbers is determined by the contents of memory locations $(000600)_8$[2] through $(000627)_8$; where the contents of $(000600)_8$ determines the trunk number designated by 000, the contents of $(000601)_8$ determines the trunk number designated by 001, etc. Thus, actual trunk numbers can be assigned to the pseudo-numbers by the operator, by modifying the sequence of characters commencing in $(000600)_8$ according to the needs of the program. It is assumed that each installation maintains a permanent reference system, and that these modifications need only be made at the beginning of each interpretive session.

[1] The machine address will be the six digit number listed under "LINE" on the right side of the edit page.

[2] Memory locations are addressed by **octal** numbers. For a complete description of the properties of octal numbers, see the RCA 501 Electronic Data Processing System Programmers' Reference Manual (page 5).

SAMPLE OF A TRACE

TRACE	A00*	00	5	0	000	001	A50			0097 6241 6241	
TRACE	A01*	00	1	0	001	A22	A23			0000 0000 0000	0002 0002 0001
TRACE	A02*	00	M	1	A50	A50	A35	0000 0000 0000	.50000000 E000	.50000000 E00	.50000000 E000
TRACE	A03*	00	2	0	001	A24	A02			0006 0006 0003	0002 0002 0001
TRACE	A02*	00	M	1	A52	A52	A36	0002 0002 0001	.10000000 E001	.10000000 E001	.10000000 E001
TRACE	A03*	00	2	0	001	A24	A02			0006 0006 0003	0004 0004 0002
TRACE	A02*	00	M	1	A54	A54	A37	0004 0004 0002	.20000000 E001	.20000000 E001	.20000000 E001
TRACE	A03*	00	2	0	001	A24	A02			0006 0006 0003	0006 0006 0003
TRACE	A04*	00	Z	0	A35	000	A39		.50000000 E000		.46364761 E000
TRACE	A05*	00	C	0	A31	A39	A39		.30000000 E001	.46364761 E000	.13909428 E001
TRACE	A06*	00	M	0	A33	A33	A40		.10000000 E001	.10000000 E001	.10000000 E001
TRACE	A07*	00	G	0	A37	A37	A40		.20000000 E001	.20000000 E001	.50000000 E001
TRACE	A08*	00	C	0	A37	A40	A40		.20000000 E001	.50000000 E001	.10000000 E002
TRACE	A09*	00	C	0	A30	A40	A40		.40000000 E001	.10000000 E002	.40000000 E002
TRACE	A10*	00	D	0	A40	A39	A38		.40000000 E002	.13909428 E001	.28757472 E002
TRACE	A11*	00	C	0	A32	A36	A41		.70000000 E001	.10000000 E001	.70000000 E001
TRACE	A12*	00	S	0	A41	000	A41		.70000000 E001		.26457513 E001
TRACE	A13*	00	S	0	A41	000	A41		.26457513 E001		.16265766 E001
TRACE	A14*	00	B	0	A38	A41	A38		.28757472 E002	.16265766 E001	.27130896 E002

Initially, the Scientific Interpreter provides for the following tape trunk references:

TRUNK NUMBER	PSEUDO-NUMBER	MEMORY LOCATION
77	000	000600
10	001	000601
20	002	000602
30	003	000603
40	004	000604
50	005	000605
60	006	000606
70	007	000607

The remaining tape trunk designations 008-023 determined by locations $(000610)_8$-$(000627)_8$ are all assigned to trunk number 01.

INSERTION OF SUBROUTINES

Subroutines written in pseudo-code are inserted in a pseudo-program by call commands on paper tape input. The subroutines called for insertion must be on the ILT (Interpretive Library Tape) as a block of coding preceded by its identification message (see Appendix E, Pseudo-code Unload Routine). In addition the subroutine must possess the following properties:

a. No patch numbers can appear; in other words, a subroutine prior to being stored as a library routine, if necessary, must be changed so as not to require the patching machinery in the course of its execution.

b. The routine must be written in interpretive pseudo-code. Thus, if called for as a subroutine of another program, the program itself must be written to accommodate this subroutine in the interpretive memory area. For example, assume that BETA is a pseudo-program written to lie in pseudo-memory locations A00-A73; then if a program uses BETA as a subroutine, the program cannot occupy the interpretive memory locations A00 - A73.

ABILITY TO WRITE PSEUDO-CODE IN PARTS

A program may be written in several parts by different coders if care is taken that no memory locations are used for conflicting purposes in different parts of the program. When the Pre-edit processes a program written in parts, it associates a different block number with each part (the block number is used during program patching (see Patches and Corrections). Block numbers are assigned sequentially, starting with 1 for machine coding. If there is no machine coding, the number 1 is reserved, to permit future addition of machine code.

OVERLAYS

An overlay is a block or segment which is not brought into memory when the pseudo-program is initially loaded, but which is called in later by the pseudo-program to replace coding which is no longer needed. The OVERLAY operation facilitates this replacement by allowing the programmer to specify which part of the program on tape is to be used as an overlay. The positioning of each overlay block in memory is determined by the tag associated with the block (see Chapter 6 on format). The format of the OVERLAY operation is:

$$\bullet^{*}/ \quad a_1a_2a_3 \quad b_1b_2b_3 \quad c_1c_2c_3$$

where: **a** is the (decimal) segment number of the overlay part
b is the (decimal) block number of the overlay part
c is the pseudo-address of the instruction to be executed after the overlay is read into memory

If the segment specified is not the one currently in memory, **b** and **c** should be zero, and those parts of the segment designated as initial program parts will be read in.

CHAPTER 5
MACHINE CODING

GENERAL

To expand the flexibility of the interpretive system, machine code is made available to the experienced programmer for handling problems involving the expression of complete computing functions. Thus, the interpretive system is limited only by the ingenuity of the experienced programmer in using the system.

MACHINE CODE

Special coding sheets, form RCA 1212, are provided to the user to facilitate his writing of machine code. These coding sheets, shown on the following page, consist of eleven (11) columns per line of coding to express a machine instruction. The first column is used to insert the standard symbol #. This symbol, #, acts as a control and error check for the instruction. The next two character positions, denoted by **l** and **r** respectively are used to reference characters in a pseudo-word. For example, consider the instruction which refers to floating-point data designated E10. Assume that this E10 is converted during pre-edit to its octal address which is the leftmost character of E10. Machine instructions, however, require both right-hand and left-hand addresses of quantities, depending on the instructions in question. Also, references to character addresses are required in machine code. For this reason, the following convention is adopted: when writing machine code, the 12 characters of a pseudo-code word are referred to as A, B, C, D, E, F, G, H, I, J, K, and L, the enumeration progressing from the left to right. If the A-address of a machine code instruction requires a pseudo-address reference, the address of the character desired is stated in the l-position; if the B-address of a machine-code instruction requires a pseudo-address reference, the address of the character desired is stated in the **r**-position. For example, the machine instruction which transfers the mantissa of a floating-point constant stored in E10 to the rightmost end of the word E12 is:

```
#JL  21  442423  00  442425
```

The contents of E10 are of the form

A	B	C	D	E	F	G	H	I	J	K	L
p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
●	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	±	e	e

And the contents of E12 become

A	B	C	D	E	F	G	H	I	J	K	L
p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
✓	✓	●	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	±

RCA 501 INTERPRETER PROGRAM RECORD

MACHINE CODE

TITLE
CODER
REMARKS

DATE
NAME
BLOCK NO.
SEGMENT NO.

FROM INSTRUCTION LOCATION	HSM LOCATION	<•	#	|	r	OP	A			N	B			DECIMAL ADDRESS			REMARKS	CHART REF.
						0	1	2	3	4	5	6	7					
	0																	
	1																	
	2																	
	3																	
	4																	
	5																	
	6																	
	7																	
	0																	
	1																	
	2																	
	3																	
	4																	
	5																	
	6																	
	7																	
	0																	
	1																	
	2																	
	3																	
	4																	
	5																	
	6																	
	7																	

FORM NO. 1212 2.5M 4-60

The pseudo-addresses are written in the RCA octal equivalent, character by character as follows: E10 becomes 442423 and E12 becomes 442425. The presence of pseudo-addresses is indicated by these alphabetic characters in the **l** and **r** positions which instruct the pre-edit to effect the necessary address translation.

References within machine code is obtained by use of the automatic address modifiers. The coding is written **self-relative,** so that the first line of machine coding bears the pseudo-address 000. In such cases, the **l**-position or **r**-position must be 8 to indicate that the respective address is machine code. Each entry to machine code begins with the sequence of instructions:

```
73 AMi 00 400000;      Store the P-register in AMi
45 AMi 00 00667        Decrease AMi by (10)8
```

and henceforth AM_i is used to modify the internal address references. A typical instruction is:

```
<•  #88  25  000013  55  000573 >
```

if AM_5 was used to store the P-register. The relative addresses 000013 and 000573 are converted to absolute addresses by the automatic addition of (P) — 10_8 in the course of staticizing the instruction.

TRANSFER TO MACHINE CODE

The instruction which transfers control from pseudo-code to machine code is denoted by the operator (OPR) symbol 9. The format for this transfer is:

CHARACTER POSITION	1	2	3	4	5	6	7	8	9	10	11	12
CHARACTER NAME	p	OPR	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
FIELD CONTENT	•	9	0	**a** is a Pseudo-address			**b** is 0 0 0			**c** is a decimal number		

where: **a** is the address of the pseudo-instruction to be executed after the completion of the machine code.

c is the line number of the starting location of the machine code to be executed.

To effect a return of control to the address given in the **a**-address, the programmer writes a transfer of control to address $(001700)_8$, which returns control to the executive routine; i.e.,

```
<•  #88  71  001700  00  000000 >
```

CHAPTER 6
PREPARATION OF A PROGRAM

INTRODUCTION

After the user has written his program on the forms provided with the Interpreter, the program is then punched on paper tape for insertion into the system. This chapter covers the format and procedure by which this paper tape is prepared.

PAPER TAPE PREPARATION

A representation of the format in which the paper tape is prepared is shown on page 36.

The first message of this input enumerates the optional subroutines to be incorporated as a fixed part of the program. These subroutines are a function of the entire program and are requested only once. This message is of the form

<• XXX ... >

where: X's represent the operation codes of the subroutines to be called in; these need not be in order. The optional subroutines are as follows:

OPTIONAL SUBROUTINE OPERATIONS	DESIGNATION
Sine	X
Cosine	Y
Arctan	Z
Log_{10}	T
Log_e	U
10^x	V
e^x	W
Square Root	S
Print Data on Monitor Printer	3
Print Data on On-Line Printer	4
Read Data	5
Write Data	6
Overlay	*

If no optional subroutines are required this message is omitted.

Following the statement of the optional subroutines are the program segments. Each segment represents an independent running program part; the start of a segment is indicated by the control symbol EF. This EF must **always** be present as the first element of a segment.

Program Entrance

This message must always be present for every segment to identify the starting instruction of the segment. This message is in the following format:

< • SXX • FFF >

where: S identifies the assignment of the program entrance.
XX is the number of the segment.
FFF is the pseudo-address of the first instruction to be executed.

Calls for Pseduo-Code Subroutines

If the interpretive library tape, ILT, contains properly identified routines written in pseudo-code and if the programmer requests their use as part of the current program, he inserts the interpretive subroutine name in a message of the form

< • X/Y NAME >

where: Y is used if the subroutine is part of the program
but: X is used if the subroutine is used as an overlay (part of a program which replaces a portion of the original program).
and: NAME is a four (4) character identification of the subroutine on the ILT.

The interpretive subroutines, unlike the optional subroutines of the program, are considered parts of a segment and therefore each segment must request the subroutines to be incorporated into that program segment.

Main Body of Program

The main body of the program may contain three types of program blocks, that is, machine coded block, pseudo-code block, and floating-point constant block.

MACHINE-CODED BLOCK

Machine-coded messages are punched as the first set. These messages represent one block of coding only and do **not** require a block tag message.

PSEUDO-CODED BLOCK

The message of each pseudo-coded block is punched in sequence with each block preceded by a block tag message of the form

< • X/Y FFF >

where: Y is used if the block is an initial part of the program
but: X is used if the block is used as an overlay
and: FFF is the beginning pseudo-address of the block in question.

The pseudo-coded blocks are terminated by a sentinel of 12 Y's of the form

< • YYYYYYYYYYY >

FLOATING-POINT CONSTANT BLOCK

All floating-point constant messages are punched following this sentinel of Y's. Each block is again preceded by the program block tag, as

< • X/Y FFF >

The final program segment must be terminated with the control symbols

EF
ED

PAPER TAPE FORMAT OF PROGRAM

SEGMENT 1	OPTIONAL SUBROUTINES	< • xx . . . x >
	EF	EF
	PROGRAM ENTRANCE	< • S01 • FFF >
	CALLS FOR PSEUDO-	< • X/Y NAME >
	CODE SUBROUTINES	.
		.
		< • X/Y NAME >
	MACHINE CODE BLOCK	< • #lr OPR a n b >
		.
		.
		< • #lr OPR a n b >
	PSEUDO-CODE BLOCK	< • X/Y FFF >
		< •• OPR i a b c >
		.
		.
		< •• OPR i a b c >
		.
		.
	PSEUDO-CODE BLOCK	< • X/Y FFF >
		< •• OPR i a b c >
		.
		.
		< •• OPR i a b c >
	Y SENTINEL	< • YYYYYYYYYYY >
	FLOATING POINT CONSTANTS	< • X/Y FFF >
		< • XXXXXXXX±eee± >
		.
		.
		< • XXXXXXXX±eee± >
		< • X/Y FFF >
		< • XXXXXXXX±eee± >
		.
		.
		< • XXXXXXXX±eee± >
SEGMENT n		EF
		.
		.
		EF
		ED

CORRECTIONS AND PATCHES

Additions and changes to the original blocks of coding are classified as corrections; additions to the block reserved by the pre-edit as the "Patch Block" are called patches. The format of correction and patch messages is:

< • LLL • P/F/M • WORD >

where LLL = pseudo-address of pseudo-instruction or floating-point constant to be corrected; or
= decimal address of the machine code to be corrected.
P/F/M = P if WORD is a pseudo-instruction
= F if WORD is a floating-point constant
= M if WORD is a machine instruction.
WORD is either a pseudo-instruction, a machine instruction, or a floating-point constant.

Pseudo-Instructions

The Patch character, p, of the pseudo-instruction WORD differs from the original pseudo-instruction: this p character must be two decimal digits rather than an item separator symbol (see page 10). The maximum allowable value for this p character is the decimal number 58.

The purpose of this 2-digit p character is to effect a transfer of control to an instruction in the Patch Block whenever this p character is not zero (00). (Note from the sample edit preceding page 27 that all p characters are initially set to zero (00).)

Since it is not possible to correct a programming error or omission by inserting additional coding between existing instructions of a program, a means must be provided for effectively inserting additional coding. This is accomplished by a non-zero p character. The running procedure would be as follows:

Each instruction is executed sequentially under control of the program until a non-zero p character is encountered. The instruction accompanying this non-zero p character is executed **and then** a search is initiated in the Patch Block for the identical non-zero p character. When a match is found, the accompanying instruction in the Patch Block is executed. This instruction is followed sequentially by instructions in the Patch Block until a pseudo-instruction effects a transfer out of the patch.

In the following example, pseudo-instructions E13 — E15 would be executed following B32. The instructions would appear in memory as follows:

Example:

PSEUDO-ADDRESS	PSEUDO-CODE					
B32	01	A	0	A13	A21	A00
B33	00	C	0	A79	A93	B03
			.			
			.			
			.			
E13	01	D	0	A00	A10	A00
E14	00	1	0	004	A01	A02
E15	00	8	0	000	000	B33

The first pseudo-location of the Patch Block is assigned during the pre-edit, and this assignment is printed on the edit. (See edit of sample problem, Appendix B.) The first line of the Patch Block is determined by this rule: it is always the pseudo-address immediately succeeding the highest bank number used by the initial program. For example, if a program occupies pseudo-locations A00 - A49, A75 - B14, B25 - B53 and C00 - C14, the first patch instruction **must** be put in pseudo-address C15. All other patches must follow sequentially. Note that the first instruction of each patch in the Patch Block must have a non-zero p character. All succeeding instructions will have a zero p character unless it is desired to insert additional coding within the patches.

Floating-Point Constants

The message to place the floating-point number 2.5 into memory location C13 would appear as follows:

< • C13 • F • 25000000+001 + >

Machine Instructions

Patching of machine coding is accommplished through correction messages of the following form:

```
< • 032 • M•#88   71 000610 50 000000 >
< • 049 • M•#L8   72 412426 00 600000 >
< • 050 • M•#AL   26 402531 00 402531 >
< • 051 • M•#88   71 000410 50 000000 >
```

This example assumes that machine code in the program occupied the decimal memory locations 000 - 048. The above corrections effect a transfer of control from memory location 032 to location 049 where the contents of A26 are moved to B13, and then control returned to memory location 033. It is assumed that the machine location of pseudo-memory location 000 is stored in AM5 for relative coding.

Paper Tape Format of Corrections and Patches

All patches and corrections may be punched on a single patch tape; the individual message format is given above.

The format of this paper tape is as follows:

< • Sx_1x_1 • Bx_2x_2 >
< • LLL • P/F/M • WORD >
< • LLL • P/F/M • WORD >
•
•
•
< • Sx_1x_1 • Bx_2x_2 >
< • LLL • P/F/M • WORD >
•
•
•
ED

The effect of the messages

$$< \bullet Sx_1x_1 \bullet Bx_2x_2 >$$

is to identify the particular block and segment to which the patch messages pertain. The paper tape input for the preceding example would appear as follows, assuming the coding is to go into blocks 3 and 5 of segment 1:

```
                    < • S01 • B03 >
< • B32 • P • 01 A 0 A13 A21 A00 >
                    < • S01 • B05 >
< • E13 • P • 01 D 0 A00 A10 A00 >
< • E14 • P • 00 1 0 004 A01 A02 >
< • E15 • P • 00 8 0 000 000 B33 >
                    ED
```

When adding lines of coding to any block, they must be assigned sequential bank numbers immediately following the last one used in the respective block. It is not possible to leave gaps **within** any single blocks. A gap in the sequence of utilized pseudo-memory locations will cause an error stop[1].

No given instruction in the program may be corrected twice during any one patch run; i.e., no two messages may have identical LLL pseudo-addresses when patching.

[1] See Appendix G, Error Stops of the Program Patch Routine, $(005120)_8$.

APPENDIX A
INPUT DATA FORMAT

INPUT DATA

The Interpreter Read instruction is designed to bring into memory blocks of data stored on magnetic tape or paper tape. It is assumed that when this data is floating-point numbers it is in a form identical to that produced by the Pre-Edit pass of the Interpreter. The conversion of a block of numbers to proper floating-point form on magnetic tape may be accomplished by the SCALE routine of the Interpreter Library, where σσ (see 501 Matrix System publication) would be 08.

In lieu of using the SCALE routine, the tape format of input data would be as follows:

$$\bullet XXXXXXXX \pm ee$$

where: • represents the decimal point.
X is the 8-digit decimal mantissa.
ee is an octal exponent obtained by adding the octal equivalent of the absolute exponent of the number to the octal excess $(0150)_8$. This is the form required by FDA.

e.g. The number 3.141592 would be punched

$$\bullet 31415920 \_\_ J$$

where: $\_J$ is the RCA character representation of the exponent

$$(0150)_8 + (0001)_8 = (0151)_8.$$

The number $.186 \times 10^{12}$ would be punched

$$\bullet 18600000 \_\_ U$$

where: $\_U$ is the RCA character representation of the exponent

$$(0150)_8 + (0014)^*_8 = (0164)_8$$

With this format, the largest number expressible is $0.99999999 \times 10^{104}$; the smallest number expressible is $0.10000000 \times 10^{-104}$. [Notice that $(104)_{10} = (150)_8$.]

CONSOLE OPERATION

If data is prepared in accordance with this procedure, BRB (Block Read Bypass) must be set on the console to prevent decoding of these characters during the read.

* 14 is the octal equivalent of a decimal 12.

APPENDIX B
SAMPLE PROBLEM

To illustrate the use of the RCA 501 Scientific Interpreter in a practical application, a sample problem is shown in this Appendix. The sample problem is of the following form:

$$f(x, y, z) = \frac{4(z + z^3)}{3 \arctan x} - \sqrt[4]{7y}$$

with the following limits:

$x = 0.5$	$\Delta x = 0.1$
$y = 1.0$	$\Delta y = 1.0$
$z = 2.0$	$\Delta z = 0.5$

The function is evaluated for 25 values of x, y, and z.

RCA 501 INTERPRETER PROGRAM RECORD

PSEUDO — CODE

TITLE

CODER

REMARKS

DATE

NAME

BLOCK NO. 2

SEGMENT NO. 1

					<	•	Y	A	0	0	>					
FROM INSTRUCTION	HSM LOCATION	<•	P	OPR	i	A			B			C			REMARKS	CHART REF.
			1	2	3	4	5	6	7	8	9	10	11	12		
	A00	<•	•	5	0	0	0	0	0	0	1	A	5	0	READ X, Δ X, Y, Δ Y, Z, Δ Z	
	1	<•	•	1	0	0	0	1	A	2	2	A	2	3	SET AM1 = (0, 0, 0) Δ AM1 = (2, 2, 1)	
	2	<•	•	M	1	A	5	0	A	5	0	A	3	5		
	3	<•	•	2	0	0	0	1	A	2	4	A	0	2	LOOP ON AM1 < (6, 6, 3)	
	4	<•	•	Z	0	A	3	5	0	0	0	A	3	9	ARCTAN X	
	5	<•	•	C	0	A	3	1	A	3	9	A	3	9	3 ARCTAN X --> DENOMINATOR	
	6	<•	•	M	0	A	3	3	A	3	3	A	4	0	1 --> STORAGE	
	7	<•	•	G	0	A	3	7	A	3	7	A	4	0	$Z^2 + 1$	
	8	<•	•	C	0	A	3	7	A	4	0	A	4	0	$Z (Z^2 + 1)$	
	9	<•	•	C	0	A	3	0	A	4	0	A	4	0	$4 (Z^2 + 1)$	
	A10	<•	•	D	0	A	4	0	A	3	9	A	3	8	1^{st} TERM	
	11	<•	•	C	0	A	3	2	A	3	6	A	4	1	7 • Y	
	12	<•	•	S	0	A	4	1	0	0	0	A	4	1	$(7 \cdot Y)^{1/2}$	
	13	<•	•	S	0	A	4	1	0	0	0	A	4	1	$(7 \cdot Y)^{1/4}$	
	14	<•	•	B	0	A	3	8	A	4	1	A	3	8	Y	
	15	<•	•	4	0	0	0	4	A	3	5	A	3	8	PRINT RESULT	
	16	<•	•	1	0	0	0	2	A	2	2	A	2	5	AM2 = (0, 0, 0) Δ AM2 = (1, 2, 1)	
	17	<•	•	A	2	A	3	5	A	5	1	A	3	5	X + ∠ X, Y + ∠ Y, Z + ∠ Z	
	18	<•	•	2	0	0	0	2	A	2	6	A	1	7	LOOP ON AM2 < (3, 6, 3)	
	19	<•	•	I	0	A	3	5	A	3	4	A	2	1	is X ≥ X max YES STOP	
	A20	<•	•	8	0	0	0	0	0	0	0	A	0	4	RECYCLE	
	21	<•	•	,	0	0	0	0	0	0	0	0	0	0	STOP	
	22	<•	•	0	0	0	0	0	0	0	0	0	0	0		
	23	<•	•	0	0	0	0	2	0	0	2	0	0	1		
	24	<•	•	0	0	0	0	6	0	0	6	0	0	3		

FORM NO. 1213 2.5M 4-60

RCA 501 INTERPRETER PROGRAM RECORD

PSEUDO — CODE

TITLE

CODER

REMARKS

DATE

NAME

BLOCK NO. 2

SEGMENT NO. 1

					<	•					>					
FROM INSTRUCTION	HSM LOCATION	<	P	OPR	i	A			B			C			REMARKS	CHART REF.
			1	2	3	4	5	6	7	8	9	10	11	12		
	A25	<	•	0	0	0	0	1	0	0	2	0	0	1		
	26	<	•	0	0	0	0	3	0	0	6	0	0	3		

FORM NO. 1213 2.5M 4-60

RCA 501 INTERPRETER PROGRAM RECORD

PSEUDO — CODE

TITLE

CODER

REMARKS

DATE

NAME

BLOCK NO. 3

SEGMENT NO. 1

					<	•	Y	A	3	0	>					
FROM INSTRUCTION	HSM LOCATION	<•	P	OPR	i	A			B			C			REMARKS	CHART REF.
			1	2	3	4	5	6	7	8	9	10	11	12		
	A30	<•	4	0	0	0	0	0	0	0	+	0	0	1	+ > CONSTANT	
	1	<•	3	0	0	0	0	0	0	0	+	0	0	1	+ > "	
	2	<•	7	0	0	0	0	0	0	0	+	0	0	1	+ > "	
	3	<•	1	0	0	0	0	0	0	0	+	0	0	1	+ > "	
	4	<•	2	5	0	0	0	0	0	0	+	0	0	1	+ > "	
	A35														X	
	6														Y	
	7														Z	
	8														f(X • Y • Z)	
	9														DENOMINATOR	
	A40														NUMERATOR	
	1															
	2															
	3															
	4															
	A45															
	6															
	7															
	8															
	9															
	A50															

FORM NO. 1213 2.5M 4-60

RCA 501 INTERPRETER PROGRAM RECORD

PSEUDO — CODE

TITLE

CODER

REMARKS

DATE

NAME

BLOCK NO. 3

SEGMENT NO. 1

					<	•					>				
FROM INSTRUCTION	HSM LOCATION	<•	P	OPR	i	A			B			C			REMARKS	CHART REF.
			1	2	3	4	5	6	7	8	9	10	11	12		
	A25	<•	•	0	0	0	0	1	0	0	2	0	0	1		
	26	<•	•	0	0	0	0	3	0	0	6	0	0	3		

FORM NO. 1213 2.5M 4-60

RCA 501 SCIENTIFIC INTERPRETER PRE-EDIT

PROGRAM ENTRY A 00 017060

BLOCK 1 FROM A 00 017060

PAGE 01 OF SEGMENT 01

NO MACHINE CODE

BLOCK 2 FROM A 00 017060

LINE	PATCH	OP	INDEX	A-ADDRESS	B-ADDRESS	C-ADDRESS	LINE	PATCH	OP	INDEX	A-ADDRESS	B-ADDRESS	C-ADDRESS
A 00		5	0	000	001	A50	017060	00	5	0	000	000001	020210
A 01		1	0	001	A22	A23	017074	00	1	0	000001	017470	017504
A 02		M	1	A50	A50	A35	017110	00	M	1	020210	020210	017724
A 03		2	0	001	A24	A02	017124	00	2	0	000001	017520	017110
A 04		Z	0	A35	000	A39	017140	00	Z	0	017724	000000	020004
A 05		C	0	A31	A39	A39	017154	00	C	0	017644	020004	020004
A 06		M	0	A33	A33	A40	017170	00	M	0	017674	017674	020020
A 07		G	0	A37	A37	A40	017204	00	G	0	017754	017754	020020
A 08		C	0	A37	A40	A40	017220	00	C	0	017754	020020	020020
A 09		C	0	A30	A40	A40	017234	00	C	0	017630	020020	020020
A 10		D	0	A40	A39	A38	017250	00	D	0	020020	020004	017770
A 11		C	0	A32	A36	A41	017264	00	C	0	017660	017740	020034
A 12		S	0	A41	000	A41	017300	00	S	0	020034	000000	020034
A 13		S	0	A41	000	A41	017314	00	S	0	020034	000000	020034
A 14		B	0	A38	A41	A38	017330	00	B	0	017770	020034	017770
A 15		4	0	004	A35	A38	017344	00	4	0	000004	017724	017770
A 16		1	0	002	A22	A25	017360	00	1	0	000002	017470	017534
A 17		A	2	A35	A51	A35	017374	00	A	2	017724	020224	017724
A 18		2	0	002	A26	A17	017410	00	2	0	000002	017550	017374
A 19		1	0	A35	A34	A21	017424	00	1	0	017724	017710	017454
A 20		8	0	000	000	A04	017440	00	8	0	000000	000000	017140
A 21		,	0	000	000	000	017454	00	,	0	000000	000000	000000
A 22		0	0	000	000	000	017470	00	0	0	000000	000000	000000
A 23		0	0	002	002	001	017504	00	0	0	000030	000030	000014
A 24		0	0	006	006	003	017520	00	0	0	000110	000110	000044
A 25		0	0	001	002	001	017534	00	0	0	000014	000030	000014
A 26		0	0	003	006	003	017550	00	0	0	000044	000110	000044

BLOCK 3

NO PATCHES

YYYYYYYY

PAGE 03 OF SEGMENT 01

BLOCK 4 FROM A30 017630 PAGE 04 OF SEGMENT 01

LINE	CONSTANT	LINE	CONSTANT
A30	40000000 E001	017630	40000000 E151
A31	30000000 E001	017644	30000000 E151
A32	70000000 E001	017660	70000000 E151
A33	10000000 E001	017674	10000000 E151
A34	25000000 E001	017710	25000000 E151

PATCH BLOCK 3 FROM A35 017724

END OF PRE-EDIT

X	Y	Z	f(X,Y,Z)
50000000 000	10000000 001	20000000 001	27130896 002
60000000 000	20000000 001	25000000 001	42784008 002
70000000 000	30000000 001	30000000 001	63355132 002
80000000 000	40000000 001	35000000 001	89339784 002
90000000 000	50000000 001	40000000 001	12129150 003
10000000 001	60000000 001	45000000 001	15979232 003
11000000 001	70000000 001	50000000 001	20244216 003
12000000 001	80000000 001	55000000 001	25885288 003
13000000 001	90000000 001	60000000 001	32064434 003
14000000 001	10000000 002	65000000 001	39144190 003
15000000 001	11000000 002	70000000 001	47187459 003
16000000 001	12000000 002	75000000 001	56257396 003
17000000 001	13000000 002	80000000 001	66417336 003
18000000 001	14000000 002	85000000 001	77730716 003
19000000 001	15000000 002	90000000 001	90261070 003
20000000 001	16000000 002	95000000 001	10407198 004
21000000 001	17000000 002	10000000 002	11922706 004
22000000 001	18000000 002	10500000 002	13578998 004
23000000 001	19000000 002	11000000 002	15382438 004
24000000 001	20000000 002	11500000 002	17339389 004
25000000 001	21000000 002	12000000 002	19456228 004

APPENDIX C

HSM LOCATIONS OF PSEUDO-ADDRESS MODIFIERS

ADDRESS MODIFIER	HSM LOCATION	HSM LOCATION OF INCREMENT
1	000300–000310	000400–000410
2	000311–000321	000411–000421
3	000322–000332	000422–000432
4	000333–000343	000433–000443
5	000344–000354	000444–000454
6	000355–000365	000455–000465
7	000366–000376	000466–000476

APPENDIX D
INTERPRETER PATCH/EDIT ROUTINE

The RCA 501 Interpreter System may occasionally require updating. The Patch/Edit Routine is included in the system to facilitate this function. A new Interpreter Library Tape will be produced by this updating routine.

The paper input specifies the block to be updated followed by the patches to that block and terminated by an EF. A series of blocks may be updated with the restriction that the blocks appear in sequence on the paper tape. For example:

```
< • BBB  >
< MMSSCC  x----------x  >
              .
              .
              .
              .
              .
          (EF) Terminates Patches to a block
< • BBB  >
< MMSSCC  x----------x  >
              .
              .
              .
              .
              .
          (EF)
          (ED) Terminates Patch Routine
```

where xxx: Block Number to be updated

MMSSCC: Most Significant character location of line to be updated in Block BBB. The locations to be updated need not be ordered.

x----------x: 16 decimal-coded octal characters inserted in the line starting at location MMSSCC.

To Edit specific blocks of the Interpreter Library Tape requires, in addition to the paper tape input, the Interpreter Library Tape and a List Tape if off-line printing is requested.

The Paper Tape Input is:

```
< • MMDDYY >
< • BBB • Title  >
          .
          .
          .
          .
        ED
```

where MM: 2-decimal number denoting month.
DD: 2-decimal number denoting day.
YY: 2-decimal number denoting year.
BBB: Block Number to be edited.
Title: Name assigned (Alphanumeric) to the block for editing purposes. See the operating instructions for options and program insertion.

APPENDIX E
PSEUDO-CODE UNLOAD ROUTINE

The function of the Pseudo-Code Unload Routine is to copy the Interpreter Library Tape, inserting or deleting pseudo-code subroutines on the Interpreter Library Tape (ILT). A subroutine in the library consists of an identification message followed by a block of batched messages comprising the subroutine.

It is assumed that the subroutines to be placed in the Library have been prepared in accordance with the restrictions noted in the paragraph "Insertion of Subroutines" on page 29 of the Interpreter Manual. The subroutine to be inserted in the Library may come from either paper tape or a program tape generated by the Interpreter Pre-Edit.

OPTIONS

1. Delete subroutines from the Interpreter Library Tape. Delete requests must appear first on the paper call tape and in the same sequence as the subroutines (to be deleted) appear on the Library Tape.
2. Insert a subroutine from a program tape to the Interpreter Library Tape.
3. Insert a subroutine from a paper tape to the Interpreter Library Tape.
4. All of the above options may be requested via paper call tape in a given run subject to the restriction described in Option 1.

PAPER TAPE FORMATS

Option 1: < • NAME >

Option 2: < • NAME • M • X/Y FFF • Sxx • Bxx • F/P >

Option 3: < • NAME • P • X/Y FFF • F/P >

This option requests a subroutine from paper tape. The messages comprising the subroutine must follow the call message and terminate in an EF followed by an ED.

DEFINITIONS

Name: is the four letter identification assigned to the subroutine by the programmer.

M: indicates the subroutine is on the program tape.

P: indicates the subroutine is to come from paper tape.

X/Y: X or Y is chosen to match the corresponding character in the tag of the desired block on the program tape.

FFF: is the entrance (pseudo-address) of the subroutine.

Sxx: is the segment number in the program containing the subroutine.

Bxx: is the block number of the subroutine in segment xx.

F: indicates the block consists of floating-point constants.

P: indicates the block consists of pseudo-code instructions.

The paper tape input is terminated in an ED.

Option 4: A subroutine may be made up of one or two blocks. A one-block subroutine may consist either of pseudo-code instructions or floating-point constants. A subroutine consisting of two blocks may assume the same "Name" (see Name under definitions). If this is the case then one of the blocks must be pseudo-coding and the other floating-point constants. Furthermore, to insert the two blocks onto the ILT requires two unload call messages (see Option 2 or 3) with the same name. To delete a two (2) block subroutine will require two call messages (Option 1) with the same name.

APPENDIX F

OPERATING INSTRUCTIONS FOR THE RCA 501 INTERPRETER

The interpreter is a system designed to provide the user with facilities for constructing and operating a program easily. The actual production of an operative program may involve a number of functions. They are:

1. **Pre-Edit:** The purpose of which is to generate an "edited program" on tape.
2. **Data Generation:** To prepare floating-point constants in block format onto tape. This task is handled by the Scale routine.
3. **Program Insertion:** The loading of the "edited-program" into memory. This task is handled by the Program Tape Bootstrap.
4. **Tracing:** A debugging aid in which the Trace routine prints out information regarding each pseudo-instruction in the course of program execution.
5. **Patching/Correcting:** A technique is provided via the program Patch routine for updating the user's program in the course of debugging.

In addition to the above functions the following two services are included in the Interpreter System:

1. A program to add and/or delete Pseudo-Code Subroutines to the Interpreter Library System.
2. A program to update and edit selected blocks of the Interpreter Library System.

The operating procedures to call in these functions are described below.

PRE-EDIT PASS

A. Mount

1. Paper Tape Input
2. Interpreter Library Tape (with ring)
3. Output (Program) Tape (with ring)

B. Operating Procedures

1. Breakpoint 2
   a. ON: Rollback is incorporated in the edit program.
   b. OFF: Rollback is omitted from the edit program.
2. All other Breakpoints set to OFF.
3. Set Rollback Inhibit Switch OFF if Rollback is desired.
4. Set a 15 020000 00 $t_1t_1$0000 on the console to call in the first block of the Library Tape into location $(020000)_8$ of the memory (ILT Bootstrap).
   where t_1t_1 = trunk # of the Interpreter Library Tape.
   a. If the character in location $(020007)_8$ is an $(01)_8$ then the edited output will go to the on-line printer.

b. If the character in location $(020007)_8$ is a $(00)_8$ then the edited output will go to tape for off-line printing.

The interpreter is set for on-line printing. The user may make the desired setting manually at this point.

5. Set the P register to location $(020000)_8$ and hit the start button.
6. "SET-B-REGISTER-TO-REQUIRED-TRUNK-NUMBERS-THEN-HIT-START-BUTTON" will go to the monitor printer. The computer will then halt on a 76 instruction with 30, 40, 50 in the C1, C2, C3 positions of the B-register respectively.

   The C1 position denotes the Library trunk #

   The C2 position denotes the Output (Program) Trunk #

   The C3 position denotes the List Tape Trunk #, if editing off-line is requested.

   (If editing on-line then C2 and C3 should be the same.)

   At this point the operator may reset the trunk numbers in the B register as desired then hit the start button to continue with the Pre-Edit. Caution: DO NOT perform a general reset on the console. If so, restart by returning to step 5 above.
7. The Pre-Edit pass will then automatically run to completion and rewind all tapes to BTC. The computer will come to a halt on a 76 instruction with END-OF-PRE-EDIT on the monitor printer. The output tape will contain the edited program.

C. Restart

1. If conditions require a rerun, the procedure is simply to rewind the Interpreter Library Tape to BTC, reposition the paper tape input and return to B (Operating Procedures), step 4.

PROGRAM OPERATION

A. Mount

1. Program Tape (without ring)
2. Other Input/Output tapes required.

B. Program Insertion

1. Set Breakpoint options (see C of Program Operation).
2. Set Rollback Inhibit Switch OFF if Rollback is desired. This option is available only if requested during the Pre-Edit Pass. (See B, step 1, of Pre-Edit Pass.)
3. Set a 15 001000 00 $t_1t_1t_2t_2t_3t_3$ on the console to call the first block of the program tape into location $(001000)_8$ of the memory (Program Tape Bootstrap).

   where: t_1t_1 = Program Tape Trunk Number

   $t_2t_2 = t_3t_3 = (00)_8$, trace output goes to the on-line printer.

   $t_2t_2 = t_3t_3 =$ XX, trunk number of trace ouput for off-line printing.
4. The Trace routine may be moved to another module by setting location $(001006)_8$ to the module desired i.e., $(01)_8$ for module 1, $(02)_8$ for module 2 etc. The Bootstrap is set for module 1.
5. Set the P register to location $(001000)_8$ and hit the start button to initiate program insertion.

C. Breakpoint Options

1. Breakpoint 1
   a. ON: The Program Tape Bootstrap calls the trace routine into memory starting at location $(034630)_8$[1].
   b. OFF: The Trace routine will not be called into memory.
2. Breakpoint 0
   a. ON: The Bootstrap comes to a halt with the printout: SET-T-REGISTER-WITH-SEGMENT-# on the Monitor Printer. At this point the T register contains $(000001)_8$ to call in segment 1. Reset the T register to the desired segment # (in octal) to be called into memory. Hit the start button. The segment requested will be called into memory then halt on a 76 instruction with the A-address containing the location (in octal) of the first instruction to be executed. The program is now ready for execution.
   b. OFF: The Bootstrap automatically loads segment 1 into memory and when completed comes to a halt on a 76 instruction. At this point the A-address contains the location (in octal) of the first instruction to be executed. The program is now ready for execution.
3. Breakpoint 2
   a. ON: The Interpreter is set to perform computations in the significant arithmetic mode.
   b. OFF: The Interpreter is set to perform computations in the normalized arithmetic mode.

D. Program Execution

Following the halt which terminates the program insertion phase, Breakpoint 0, 1, 2 must be reset to select the following options. Then the start button must be hit to proceed with the program execution.

1. Breakpoint 0
   a. ON: During execution, the program in operation will halt on a 76 instruction following each program step. The A-address at this point will indicate the location (in octal) of the next intsruction to be executed.
   b. OFF: The execution of the program instructions will proceed continuously.
2. Breakpoint 1
   a. ON: The program will be executed in the normal trace mode.
   b. OFF: The normal trace mode is not requested.
3. Breakpoint 2
   a. ON: ıe program will be executed in the logical trace mode.
   b. OFF: ıe logical trace mode is not requested.
4. Memory ..ss Stop (MASP): To begin tracing from a specific point in the program ıng executed.
   a. Set Bı akpoints 1 and 2 to off.

[1] Subject to B (Program Insertion), step 4.

b. Set MASP address on console to the exact location (in octal) at which it is desired to begin tracing.

c. Set the MASP indicator on the console to ON.

d. Hit the start button to initiate program execution.

e. When the requested location is reached the computer will come to a halt on a 22 (OCT) instruction. The A-address will contain the location (in octal) of the instruction at which a stop was requested.

f. Set Breakpoints 1 and 2 as desired (see D, Program Execution, steps 2 and 3).

g. If the programmer wishes to terminate tracing at a particular point, reset the MASP address to that point, otherwise reset the MASP indicator to OFF.

h. Hit the start button.

i. The program will proceed in the trace mode and stop if a terminal MASP was set. To continue program execution without tracing reset the MASP indicator to OFF and Breakpoints 1 and 2 to OFF. Then hit the start button.

E. Restart

1. If conditions require program reloading: Rewind program tape to BTC, position other tapes as required and proceed from B, Program Insertion.
2. If the programmer desires to start at a particular point in the program without reloading, set memory and tape conditions as desired, place the exact starting location into $(000671\text{-}3)_8$ of the memory and set P to $(000670)_8$. Then hit the start button.

PATCHING (PROGRAM UPDATING)

A. Mount

1. Paper Tape Input
2. Program Tape (with ring)
3. Output Tape (with ring)
4. List Tape if required.

B. Operating Procedures

1. Set Breakpoint 3 ON
2. Set Rollback Inhibit Switch OFF if rollback is desired.
3. Set 15 001000 00 $t_1t_1t_2t_2t_3t_3$ on the console to call in the first block of the program tape into location $(001000)_8$ of memory (Program Tape Bootstrap).

   where: t_1t_1 = Program Tape Trunk Number.
   t_2t_2 = Output Tape Trunk Number.
   t_3t_3 = List Tape Trunk Number, required for off-line printing.
   $t_2t_2 = t_3t_3$ = output tape trunk number if editing on-line.
4. Location $(001007)_8$ contains an $(01)_8$ to indicate on-line editing. For off-line editing this character must be set to $(00)_8$.

5. Set the P register to $(001000)_8$ and hit the start button. The Patch Routine will be called into memory and halt on a 76 instruction. At this point Breakpoint 3 should be reset to off (see C, Breakpoint Options, step 1). To proceed hit the start button. The patch routine will call in the paper tape input to update the source language at which point the Pre-Edit pass will be called on and the updated program generated on the output tape.

C. Breakpoint Options

1. Breakpoint 0
   a. ON: The Patch routine will halt after the current paper tape input is processed (ED is sensed). At this point additional patch input may be mounted on the paper tape reader. Hit the start button. The computer will again come to a stop. When all paper tape input is exhausted reset Breakpoint 0 to OFF.
   b. OFF: The patch routine will proceed to update the source program (the paper tape input is preserved on the program tape) and then call in the pre-edit pass. On completion the output tape will then contain the updated version of the program tape.

D. Restart

1. If conditions require a rerun the procedure is simply rewind the Program tape to BTC, reposition the paper tape input and repeat the procedures starting with B, Operating Procedures, step 3.

SCALING (DATA GENERATION)

A. Mount

1. Paper Tape Input
2. Interpreter Library Tape or Program Tape
3. Output Tape (Data Tape).

B. Operating Procedure for Library Tape

1. Set Breakpoint to ON.
2. Set a 15 020000 00 $t_1t_1$0000 on the console to call in the first block of the Library Tape into location $(020000)_8$ of the memory (ILT Bootstrap).
   where: t_1t_1 = the trunk number of the Interpreter Library Tape.
3. Set the p-register to $(020000)_8$ and hit the start button.
4. "SET-B-REGISTER-TO-REQUIRED-TRUNK-NUMBERS-THEN-HIT-START-BUTTON" will go to the Monitor Printer. The computer will then halt on a 76 instruction with 30, 40, 50 in the C1, C2, C3 positions of the B register respectively.

   At this point the operator should set $t_1t_1$0000 into the B register.

   where: t_1t_1 = the trunk number of the Interpreter Library Tape.

   Then hit the start button to call in the SCALE routine and halt. Hit the start button to initiate SCALE routine. Caution: DO NOT perform a general reset on the console.

5. The SCALE routine will read in the parameter message and halt on a 76 instruction with t_1t_1 t_2t_2 t_3t_3 in the T register. (See SCALE description in the 501 Matrix Arithmetic System Publication). Set breakpoint 4 to OFF, and the operator may reset the trunk numbers in the T register as desired; then hit the start button to process data into block format on t_2t_2.
6. The SCALE routine finally prints onto the Monitor Printer the number of messages called in and the number of blocks written on the output tape, then comes to a halt on a 76 instruction.

C. Operating Procedure for Program Tape

1. Set Breakpoint 4 ON.
2. Set a 15 001000 00 $t_1t_1t_2t_2t_3t_3$ on the console to call in the first block of the program.
   where: t_1t_1 = Program Tape Trunk #
   $t_2t_2 = t_3t_3$ = Output Tape Trunk #
3. Set P to $(001000)_8$ and hit the start button.
4. Data from paper tape is processed (converted to floating-point format) in block format onto the output tape and the tapes then rewound to BTC.
5. The routine prints onto the Monitor Printer the number of messages called in and the number of blocks written on the output tape, then comes to a halt on a 76 instruction.

D. Restart

1. If conditions require a rerun the procedure is simply to rewind the Interpreter Library Tape or Program Tape to BTC and position the output tape as desired. The paper tape input is repositioned and the operating procedures repeated starting from B (Operating Procedure for Library Tape) step 3, or C (Operating Procedure for Program Tape) step 2.

PSEUDO-CODE UNLOAD ROUTINE

A. Mount

1. Paper Tape Input
2. Interpreter Library Tape, (without ring).
3. Output Tape, (with ring).
4. Program Tape (optional), (without ring).

B Operating Procedure

1. Set Breakpoint 5 ON.
2. Set Rollback Inhibit Switch OFF if Rollback is desired.
3. Set a 15 020000 00 $t_1t_1$0000 on the console to call in the first block of the Library Tape (ILT Bootstrap).
   where: t_1t_1 = Library Tape Trunk #

4. Set P to $(020000)_8$ and hit the start button.
5. "SET-B-REGISTER-TO-REQUIRED-TRUNK-NUMBERS-THEN-HIT-START-BUTTON" will go to the monitor printer. The computer will then halt on a 76 instruction with 30, 40, 50 appearing in the C1, C2, C3 positions of the B register respectively.

   C1 denotes the Library Tape Trunk #

   C2 denotes the Output Tape Trunk #

   C3 denotes the Program Tape Trunk # (when required, if not set C3 = C2).

   At this point the operator may reset the B register to the desired trunk numbers then hit the start button to continue.
6. The unload routine is called into memory and the unloading process initiated.
7. When the Routine is finished all tapes are rewound to BTC and the message: UPDATED ILT on Trunk XX is printed on the monitor printer. The routine then comes to a halt on a 76 instruction.

C. Restart

1. If conditions require a rerun the procedure is to rewind the Interpreter Library Tape to BTC, reposition the paper tape input, and return to the procedure starting with B (Operating Procedure) step 3.

INTERPRETER LIBRARY PATCH/EDIT ROUTINE

A. Mount

1. Paper Tape Input.
2. Interpreter Library Tape (without ring).
3. Output Tape (with ring) when patching.
4. List Tape (with ring) when editing off-line.

B. Operating Procedure

1. To call in Patch/Edit Routine
   a. Set Breakpoint 3 ON
   b. Set Rollback Inhibit Switch OFF if Rollback is desired.
   c. Set a 15 020000 00 $t_1t_1$0000 on the console to call in the first block of the Library Tape (ILT Bootstrap).

      where: t_1t_1 = Library Tape Trunk Number
   d. Set P to $(020000)_8$ and hit the start button.
   e. "SET-B-REGISTER-TO-REQUIRED-TRUNK-NUMBERS-THEN-HIT-START-BUTTON" will go to monitor printer. The computer will then halt on a 76 instruction with 30, 40, 50 appearing in the C1, C2, C3 positions of the B register respectively.

1. **For editing,** C1 = Library Tape Trunk Number
   For off-line editing, C2 = C3 = List Tape Trunk Number
   For on-line editing C2 = C3 = $(00)_8$

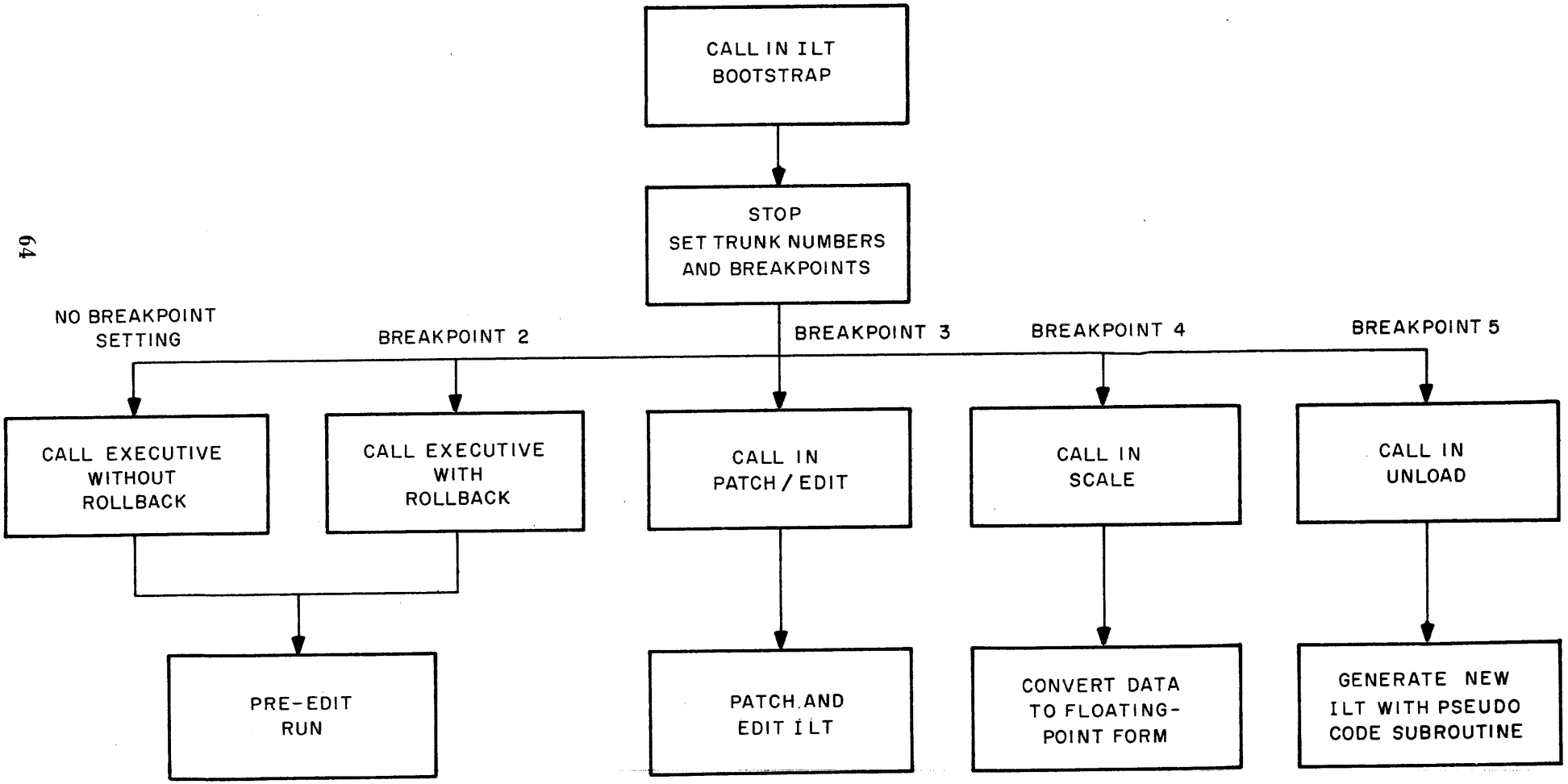
2. **For patching,** C1 = Library Tape Trunk Number
   C2 = C3 = Output Tape Trunk Number

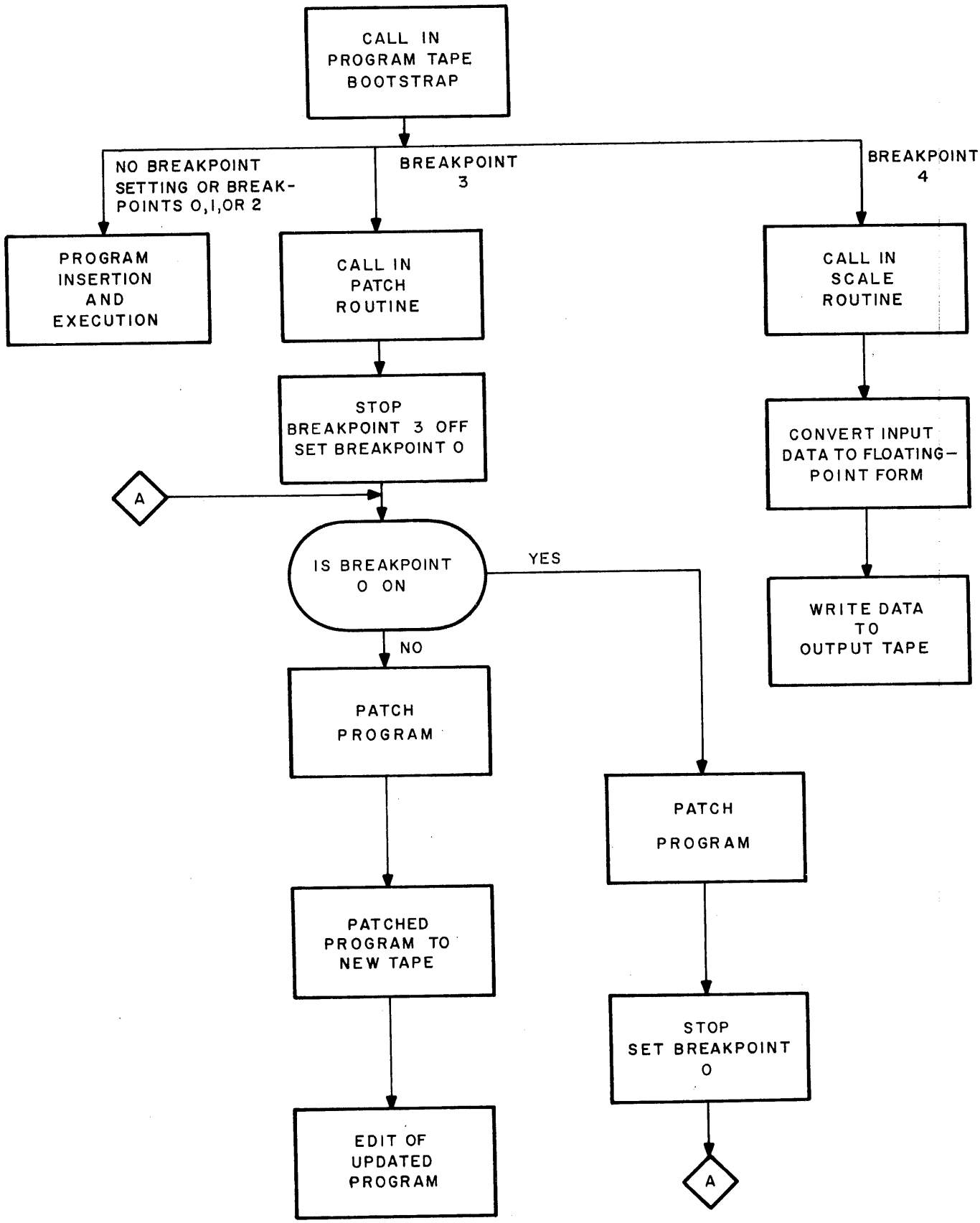
2. After the Patch/Edit routine is called in, the computer will halt on a 76 instruction. At this point Breakpoint Options must be selected.
3. Breakpoint Options
   a. Breakpoint 0 ON: Patches will be written to the Monitor Printer.
      OFF: No printing of patches.
   b. Breakpoint 3 ON: This requests the routine to operate in the edit mode only.
      OFF: This requests the routine to operate in the patch mode only.
   c. Breakpoint 4 ON: In conjunction with Breakpoint 3, editing will go to tape for off-line printing.
      OFF: Editing will go to the on-line printer.
4. Hit the start button to begin execution of the program.
5. On completion the computer will halt on a 76 instruction.

C. Restart

1. If conditions require reloading the routine; rewind the Interpreter Library Tape to BTC, reset all Breakpoints to OFF, reposition the paper tape input and return to the procedures starting with B (Operating Procedure) step 1.
2. If the program is still in memory, reposition the paper tape input and set P to $(001500)_8$, then hit the start button.

FUNCTIONS OF
INTERPRETER LIBRARY TAPE (ILT)





FUNCTIONS OF THE PROGRAM TAPE

APPENDIX G

ERROR STOPS AND RESTART PROCEDURES

This Appendix describes the error stops and restart procedures for the following:

a. ILT Bootstrap
b. Pre-Edit Run
c. Program Tape Bootstrap
d. Program Patch Routine
e. Interpreter Patch/Edit Routine
f. Pseudo-Code Unload Routine
g. Interpreter Program

ILT BOOTSTRAP

The following table describes how to correct, at the computer, format errors which are detected by the ILT Bootstrap before it loads the Pre-edit routine into memory. If a correct message is unknown, the programmer or operator may continue processing by substituting the pertinent **dummy message:**

	DUMMY MESSAGE
Machine Code:	< • #887600000000000000 >
Pseudo-Code:	< •• 00000000000 >
Floating-Point Constant:	< • 00000000+000+ >

ERROR STOPS AND RESTART PROCEDURES FOR ILT BOOTSTRAP

HSM STOP INSTRUCTION	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
01 021130 01 XXXXXX	None	First message of paper tape input must be an EF or optional subroutine selection. If first character is not EF or SM, this error stop occurs.	If paper tape not in error, reposition message in reader. Set P to $(021060)_8$, then hit start button. If paper tape in error, note position of input and read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(021100)_8$. Then hit start button.
01 021420 01 XXXXXX	Check machine code format: < Message-Read-In >	Format should be < • #lr Machine Code > This error stop occurs if the ISS is missing or the message is incorrect in length.	If input is in error, note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(021230)_8$. Then hit start button.
01 021560 01 XXXXXX	Specify block locator: < Message-Read-In >	Check for X/Y in Block Tag Message. Format: < • X/Y FFF >	If input is in error, note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(021450)_8$. Then hit start button.
01 021630 01 XXXXXX	Check block locator format: < Message-Read-In >	Incorrect Block Tag Message. Format: < • X/Y FFF > This error stop occurs if ISS is missing or message length incorrect.	If input is in error note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(021450)_8$. Then hit start buton.
01 022440 01 XXXXXX	None	This message should be a Block Tag. Format: < • X/Y FFF > This error occurs if the character after the ISS is a space.	If input is in error note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(021660)_8$. Then hit start button.

ERROR STOPS AND RESTART PROCEDURES FOR ILT BOOTSTRAP (Continued)

HSM STOP INSTRUCTION	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
01 022520 01 XXXXXX	Check Pseudo Code format: < Message-Read-In >	This message should be of the form: < ● ● OPR i **a b c** > **Note:** If a sentinal of Y's in the segment is missing, this error stop will also occur. In this case the paper tape input must be corrected and the pre-edit restarted.	See note in type of error. If input is in error note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(021660)_8$. Then hit start button.
01 022620 01 XXXXXX	Specify block locator: < Message-Read-In >	Check for X/Y in Block Tag Message. Format: < ● X/Y FFF >	If input is in error note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(022560)_8$. Then hit start button.
01 022670 01 XXXXXX	Check block locator format: < Message-Read-In >	Check Block Tag. Format: < ● X/Y FFF > This error stop occurs if ISS is missing or the message is incorrect in length.	If input is in error note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(022620)_8$. Then hit start button.
01 023030 01 XXXXXX	None	Check for X/Y in Block Tag. Format: < ● X/Y FFF >	If input is in error note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(022720)_8$. Then hit start button.
01 023110 01 XXXXXX	Check constant format: < Message-Read-In >	This error stop occurs if ISS is missing or message is incorrect in length.	If input is in error note position of paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(022720)_8$. Then hit start button.

ERROR STOPS AND RESTART PROCEDURES FOR ILT BOOTSTRAP (Continued)

HSM STOP INSTRUCTION	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
01 024020 01 XXXXXX	Not an EF: < Message-Read-In >	An EF should appear after the optional subroutine message. If missing this error stop occurs.	If the paper tape input is in error, i.e., if the EF has been omitted leave the paper tape in its current position. Set P to $(021150)_8$. Then hit start button.
01 024060 01 XXXXXX	Specify Entry Tag or ED < Message-Read-In >	Entry Tag Format: < • SXX • FFF > At this point either the segment tag or ED should appear. ED terminates the paper tape input.	If the input is in error note position of the paper tape. Read correct message into $(001000)_8$. Replace paper tape input at position noted and set P to $(021150)_8$. Then hit start button.

PRE-EDIT RUNS

The following error stops will occur only upon a malfunction of the computer during processing of the program paper tape input. Restart the Pre-Edit pass, rewinding all tapes.

01	003310	xx	xxxxxx
01	004620	xx	xxxxxx
01	004710	xx	xxxxxx
01	004720	xx	xxxxxx
01	006510	xx	xxxxxx
01	010740	xx	xxxxxx

ERROR STOPS AND RESTART PROCEDURES FOR PRE-EDIT RUNS

HSM STOP INSTRUCTION	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTION
01 005310 XX XXXXXX	Line $(XXX)_8$. Improper message format.	Machine code l-symbol is neither an 8 or an alphabetic.	Correct character in $(001653)_8$. Set P to $(005260)_8$ and hit start button.
01 005500 XX XXXXXX	Line $(XXX)_8$. Improper message format.	Machine code r-symbol is neither an 8 or an alphabetic.	Correct character in $(001654)_8$. Set P to (005450) and hit start button.
01 013550 XX XXXXXX	Line $(XXX)_8$. Improper message format.	Illegal OPR symbol.	Correct character in $(001653)_8$. Set P to (013530) and hit start button.

THE PROGRAM TAPE BOOTSTRAP

The following list of error stops indicate a machine or interpretive system malfunction. If there is nothing wrong with the hardware regenerate the pseudo-program by calling on the Pre-Edit pass.

01	001430	01	xxxxxx
01	001540	01	xxxxxx
01	001710	01	xxxxxx
01	002020	01	xxxxxx
01	002050	01	xxxxxx

PROGRAM PATCH ROUTINE

The following error stops indicate a machine or interpreter system malfunction. If there is nothing wrong with the hardware try a previous program tape or regenerate the program by calling on the Pre-Edit pass.

01	000600	01	xxxxxx
01	000720	01	xxxxxx
01	002410	01	xxxxxx
01	003400	01	xxxxxx
01	003560	01	xxxxxx
01	003750	01	xxxxxx
01	004250	01	xxxxxx
01	004310	01	xxxxxx
01	005500	01	xxxxxx
01	006600	01	xxxxxx

The following table lists the legitimate program error stops.

ERROR STOPS AND RESTART PROCEDURES FOR PROGRAM PATCH ROUTINE

HSM ERROR STOP	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
01 001150 01 XXXXXX	None	Format of the Block Selection Message incorrect: < • SXX • BXX >	If the input is in error note the position of the paper tape; read the correct message into $(003000)_8$ and replace the paper tape input at the position noted. Set P to $(000770)_8$, then hit start button.
01 001270 01 XXXXXX	None	This error stop occurs if the patch control symbol is not an M, F or P.	If the input is in error note the position of the paper tape; read the correct message into $(003000)_8$ and replace the paper tape input at the position noted. Set P to $(000770)_8$, then hit start button.
01 001410 01 XXXXXX	None	This error stop occurs if the machine code format is incorrect: < • LLL • M • Machine Code Instruction >	Note the position of the paper tape, read the correct message into $(003000)_8$ and replace the paper tape input at the position noted. Set P to $(000770)_8$, then hit start button.
01 001600 01 XXXXXX	None	This error stop occurs if the floating point message format is incorrect: < • LLL • F • Floating Point constant >	Note the position of the paper tape, read the correct message into $(003000)_8$ and replace the paper tape input at the position noted. Set P to $(000770)_8$, then hit start button.
01 001710 01 XXXXXX	None	This error stop occurs if the pseudo code instruction message format is incorrect: < • LLL • P • Pseudo-Code Instruction >	Note the position of the paper tape, read the correct message into $(003000)_8$ and replace the paper tape input at the position noted. Set P to $(000700)_8$, then hit start button.

ERROR STOPS AND RESTART PROCEDURES FOR PROGRAM PATCH ROUTINE (Continued)

HSM ERROR STOP	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
01 001750 01 XXXXXX	None	This error stop occurs if the patch number specified in the pseudo - code instruction is greater than $(58)_{10}$.	Note the position of the paper tape, read the correct message into $(003000)_8$ and replace the paper tape input at the position noted. Set P to $(000700)_8$, then hit start button.
01 005120 01 XXXXXX	None	This error stop occurs if the address requested does not equal the address of the counter. This implies the patches are not contiguous to the existing program block.[1]	Correct the paper tape and restart the Patch Routine.

[1] See explanation on page 39.

ERROR STOPS IN THE INTERPRETER PATCH/EDIT ROUTINE

The following list of error stops indicate a machine or interpreter system malfunction. If any of these stops occur and there is nothing wrong with the hardware rerun with a different Library Tape.

01	002330	01	xxxxxx
01	002430	01	xxxxxx
01	003070	01	xxxxxx
01	006300	01	xxxxxx
01	006380	01	xxxxxx
01	011230	01	xxxxxx

The following table lists the legitimate program error stops.

ERROR STOPS AND RESTART PROCEDURES FOR PATCH/EDIT ROUTINE

HSM ERROR STOP	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
01 001760 01 XXXXXX	None	Data Message format is: < • MMDDYY > This stop occurs if this message is missing, the ISS is missing or the message length is incorrect.	Note the position of the paper tape. Read the correct message into $(015000)_8$, replace the paper tape at the position noted and set P to $(001710)_8$. Then hit start button.
01 002250 01 XXXXXX	None	Block/Title Message format is: < • BBB • Title > This stop occurs if the ISS's are missing or mispositioned.	Note the position of the paper tape. Read the correct message into $(015000)_8$, replace the paper tape at the position noted and set P to $(002070)_8$ .Then hit start button.
01 002610 01 XXXXXX	None	Block Message format is: < • BBB > This stop occurs if the ISS is missing or the message is incorrect in length.	Note the position of the paper tape. Read the correct message into $(015000)_8$, replace the paper tape at the position noted and set P to $(002070)_8$. Then hit start button.
01 005510 01 XXXXXX	None	The format of the Patch Message is: < MMSSCC PATCH INSTR > This stop occurs if the length of the message is incorrect.	Note the position of the paper tape. Read the correct message into $(007300)_8$, replace the paper tape at the position noted and set P to $(005350)_8$. Then hit start button.

PSEUDO-CODE UNLOAD ROUTINE

All error stops in the Pseudo-Code Unload Routine are accompanied by a print-out on the Monitor Printer describing the source of the error. The error, its print-out and restart procedures are listed on the following page.

ERROR STOPS AND RESTART PROCEDURES FOR PSEUDO-CODE UNLOAD ROUTINE

LOCATION OF STOP	MESSAGE ON MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
001200	Subroutine to be deleted is not located on ILT.	1. Subroutine not on ILT. 2. Incorrect subroutine identification on paper call tape. 3. If deleting more than 1 subroutine at one time, the subroutine identifications on paper call tape are not in same sequence as they appear on ILT.	1. Mount ILT that has subroutine on ILT trunk, reposition paper call tape. Restart at location $(000300)_8$. 2. Repunch paper call tape with correct subroutine identification. Restart at location $(000300)_8$. 3. Repunch paper call tape using correct sequence and restart at location $(000300)_8$.
001200	Incorrect call tape format. Subroutine to be deleted is not in proper sequence.	1. A deletion message appears on paper call tape following an insertion message.	1. Repunch paper call tape using correct format and restart at location $(000300)_8$.
001200	Incorrect call tape format. Check number of ISS's in Call Message.	1. Call tape message contains less than 3 items, options 2 and 3 of paper tape format.	1. Repunch message in error using correct format, read corrected message into location $(014744)_8$, remount original paper tape at position after message in error. Restart at location $(000520)_8$.
001200	Incorrect call tape message. Check item 3 for X or Y.	1. Item 3 on call tape starts with a character other than X or Y, options 2 and 3 of paper tape format.	1. Repunch message in error using correct format. Read corrected message into location $(014744)_8$. Remount original paper tape at position after message in error. Restart at location $(000520)_8$.
001200	Incorrect call tape format. Check item 2 for M or P.	1. Item 2 on call tape starts with a character other than M or P, options 2 and 3 of paper tape format.	1. Repunch message in error using correct format. Read corrected message into location 014744. Remount original paper tape at position after message in error. Restart at location $(000520)_8$.

ERROR STOPS AND RESTART PROCEDURES FOR PSEUDO-CODE UNLOAD ROUTIN

LOCATION OF STOP	MESSAGE ON MONITOR PRINTER	TYPE OF ERROR	METHOD OF C
001200	Incorrect call tape format. Last subroutine read from Paper Tape not followed by EF.	1. ED preceding EF at end of paper call tape. 2. Call tape does not end in EF, ED.	1. and 2. Punch paper tap mount on paper ta at location (00212
003330	Tag for block XX segment XX: X/Y FFF, Tag on Call Tape X/Y FFF.	1. Using wrong program tape. 2. X/Y FFF punched incorrectly on paper tape.	1. Mount Program desired block a Program Tape paper call tape. F (000300)$_8$. 2. Repunch message rect X/Y FFF. R sage into locati mount original p tion after messag at location (0005
001200	Check if block number on Call Tape < 2.	1. Attempting to insert a machine code subroutine. 2. Block number incorrectly punched as 0 or 1.	1. Illegal procedure 2. Repunch messag correct format. R sage into locati mount original p tion after messag at location (0005
001200	Block XX of segment XX not located.	1. Using wrong Program Tape. 2. Segment being searched does not contain block number requested.	1. Mount Program rect block num number on Prog Reposition paper at location (000 2. Repunch paper c rect block numb message into loca mount original message in erro tion (000520)$_8$.

THE INTERPRETER PROGRAM

The following error stops indicate machine malfunctions. If there appears to be nothing wrong with the equipment re-insert the program from the Program Tape.

HSM LOCATION OF ERROR STOP	STP AT TIME OF STOP
006120	005500
006120	006000
006120	006220
006120	007050
006120	007600
000240	000370 + MSC of PRNT subroutine
002000 + MSC of EXPX subroutine	001230 + MSC of EXPX subroutine

ERROR STOPS AND RESTART PROCEDURES FOR INTERPRETER PROGRAM

HSM STOP INSTRUCTION	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
76 XXXXXX	EXCESS RANGE	The exponent of the result is less than zero or greater than twice the excess $(0150)_8$; i.e., $(0320)_8$.	None. The floating point operation may be continued from this point by pressing start. However, the subsequent results may be incorrect.
76 XXXXXX	ZERO DIVISOR	The contents of OP2 in a divide operation, i.e., the divisor, zero.	The floating-point operation may not be continued. However, program execution may be continued by setting P to $(XXXXXX)_8$ in the A-address of the stop instruction. This is the location of the instruction following the entrance into the divide subroutine.
01 000000 04 001660	None	Requested patch character not found in Patch Block.	Patch the program properly.
01 XXXXXX	None	P = $(006120)_8$ The sign of a STP = $(004500)_8$ floating-point operand is invalid.	The floating-point operation may not be continued. However, program execution may be continued by setting P to $(XXXXXX)_8$ in the A-address of the stop instruction. This is the location of the instruction following the entry into FDA.
01 XXXXXX	None	If P = $(006120)_8$ and STP = $(005040)_8$ or STP = $(005100)_8$, the two least significant digits of a floating point operand are not proper decimal numbers.	The floating point operation may not be continued. However, program execution may be continued by setting P to $(XXXXXX)_8$ in the A-address of the stop instruction. This is the location of the instruction following the entry into FDA.

ERROR STOPS AND RESTART PROCEDURES FOR INTERPRETER PROGRAM (Continued)

HSM STOP INSTRUCTION	MONITOR PRINTER	TYPE OF ERROR	METHOD OF CORRECTING
01 000250 + MSC of PRNT optional subroutine*	None	The A-Address of the print instruction specifies more than 8 words per line.	Correct character in HSM location $(000505)_8$ to the proper octal equivalent of the number of words per line. Set P to $(001400)_8$ and press start.
01 000450 + MSC of LOGX optional subroutine*	None	Argument is not greater than zero.	To continue program execution with the next pseudo-instruction, set P to $(000670)_8$ and hit start.
01 004050 + MSC of SQRT optional subroutine*	None	Argument is less than zero.	To continue program execution with the next pseudo-instruction, set P to $(000670)_8$ and hit start.
01 002000 + MSC of EXPX optional subroutine*	None	If STP = $(000500)_8$ + MSC, Exponent of argument is greater than $(1999)_{10}$.	To continue program execution with the next pseudo-instruction, set P to $(000670)_8$ and hit start.
01 XXXXXX	None	If P = $(002000)_8$ + MSC of EXPX and STP = $(000070)_8$ + MSC of EXPX Subroutine, the sign of argument x of e^x is invalid.	To continue program execution with the next pseudo-instruction, set P to $(000670)_8$ and hit start.

* See print on monitor printer for MSC's of optional subroutines.

REFERENCE TABLE OF INSTRUCTIONS

INSTR SYMBOL	INSTRUCTION NAME	INSTRUCTION						INTERPRETATION	PAGE REF.
		p	OPR	i	a	b	c		
*	Overlay	●	*	0	a_{10}	b_{10}	c	Read segment **a** block **b**	30
0	Skip	✓	0	0	a	b	c	Skip Instruction	24
1	Set Pseudo Address Modifier	✓	1	✓	a_{10}	b	c	$(AM_i) + (b) \rightarrow (AMa)$ $(c) \rightarrow (\Delta AMa)$	22
2	Loop	f.	2	✓	a_{10}	b_{10}	c	$(AMa) + (\Delta AMa) \rightarrow (AMa)$ If $(AMa) < (b) + (AMa)$ Then TC to **c**	23
3	Print on Monitor Printer	✓	3	✓	a_{10}	b	c	Write the sector of memory from **b** to **c** onto the on-line printer, **a** words per line.	20
4	Print On-Line	✓	4	✓	a_{10}	b	c	Write the sector of memory from **b** to **c** onto the monitor printer, **a** words per line.	20
5	Read Magnetic Tape	✓	5	✓	a_{10}	b_{10}	c	Read from tape **a**, **b** blocks of information; read into memory starting at location **c**.	19
6	Write to Magnetic Tape	✓	6	✓	a_{10}	b	c	Write the sector of memory from **b** to **c** onto tape trunk **a**.	19
7	Test Significance	✓	7	0	a_{10}	—	c	If significance of result < **a**, TC to **c**.	17
8	Transfer Control	●	8	✓	a_{10}	—	c	TC to **c** (subject to breakpoint setting **a**).	17
9	Transfer to Machine Code	●	9	✓	a	—	c_{10}	TC to line **c** of machine code. On return go to **a**.	33
,	Stop	●	,	✓	a	b	c	Stop	24

REFERENCE TABLE OF INSTRUCTIONS (Continued)

INSTR SYMBOL	INSTRUCTION NAME	INSTRUCTION						INTERPRETATION	PAGE REF.
		p	OPR	i	a	b	c		
A	Add	✓	A	✓	a	b	c	(a) + (b) → (c)	13
B	Subtract	✓	B	✓	a	b	c	(a) — (b) → (c)	13
C	Multiply	✓	C	✓	a	b	c	(a) × (b) → (c)	13
D	Divide	✓	D	✓	a	b	c	(a) ÷ (b) → (c)	13
E	Negative Multiply	✓	E	✓	a	b	c	— (a) × (b) → (c)	13
F	Negative Divide	✓	F	✓	a	b	c	— (a) ÷ (b) → (c)	13
G	Vector Multiply	✓	G	✓	a	b	c	(a) × (b) + (c) → (c)	13
H	Polynomial Multiply	✓	H	✓	a	b	c	(a) × (b) + (c) → (a)	13
I	If Greater	f[1]	I	✓	a	b	c	If (a) > (b) then TC to c	16
J	If Equal	f[1]	J	✓	a	b	c	If (a) = (b) then TC to c	16
K	If Smaller	f[1]	K	✓	a	b	c	If (a) < (b) then TC to c	16
L	If Exponent	f[1]	L	✓	a	b	c	If Exponent (a) < Exponent (b) then TC to c	16
M	Move	✓	M	✓	a	b	c	Move [a, b] to [c, c + (b — a)]	18
N	Add (fixed)	✓	N	✓	a	b	c	(a) + (b) → (c)	15
O	Subtract (fixed)	✓	O	✓	a	b	c	(a) — (b) → (c)	15
Q	Test Z's	•	Q	✓	a	b	c	(a) : Z's, If = TC to b, If = TC to c	17
R	Do Subroutine	•	R	✓	a	b	c	'TC to a' → (b) then TC to c	20
S	Square Root X	✓	S	✓	a	—	c	√(a) → (c)	15

[1] The jump to the patch block for a non-zero p character is effected only when the test fails and control would pass to the next instruction in sequence.

REFERENCE TABLE OF INSTRUCTIONS (Continued)

INSTR SYMBOL	INSTRUCTION NAME	INSTRUCTION						INTERPRETATION	PAGE REF.
		P	OPR	i	a	b	c		
T	Log_{10} X	✓	T	✓	a	—	c	Log_{10} (a) → (c)	15
U	Log_e X	✓	U	✓	a	—	c	Log_e (a) → (c)	15
V	10^x	✓	V	✓	a	—	c	$10^{(a)}$ → (c)	15
W	e^x	✓	W	✓	a	—	c	$e^{(a)}$ → (c)	15
X	Sin X	✓	X	✓	a	—	c	Sin (a) → (c)	15
Y	Cos X	✓	Y	✓	a	—	c	Cos (a) → (c)	15
Z	Arctan X	✓	Z	✓	a	—	c	Arctan (a) → (c)	15