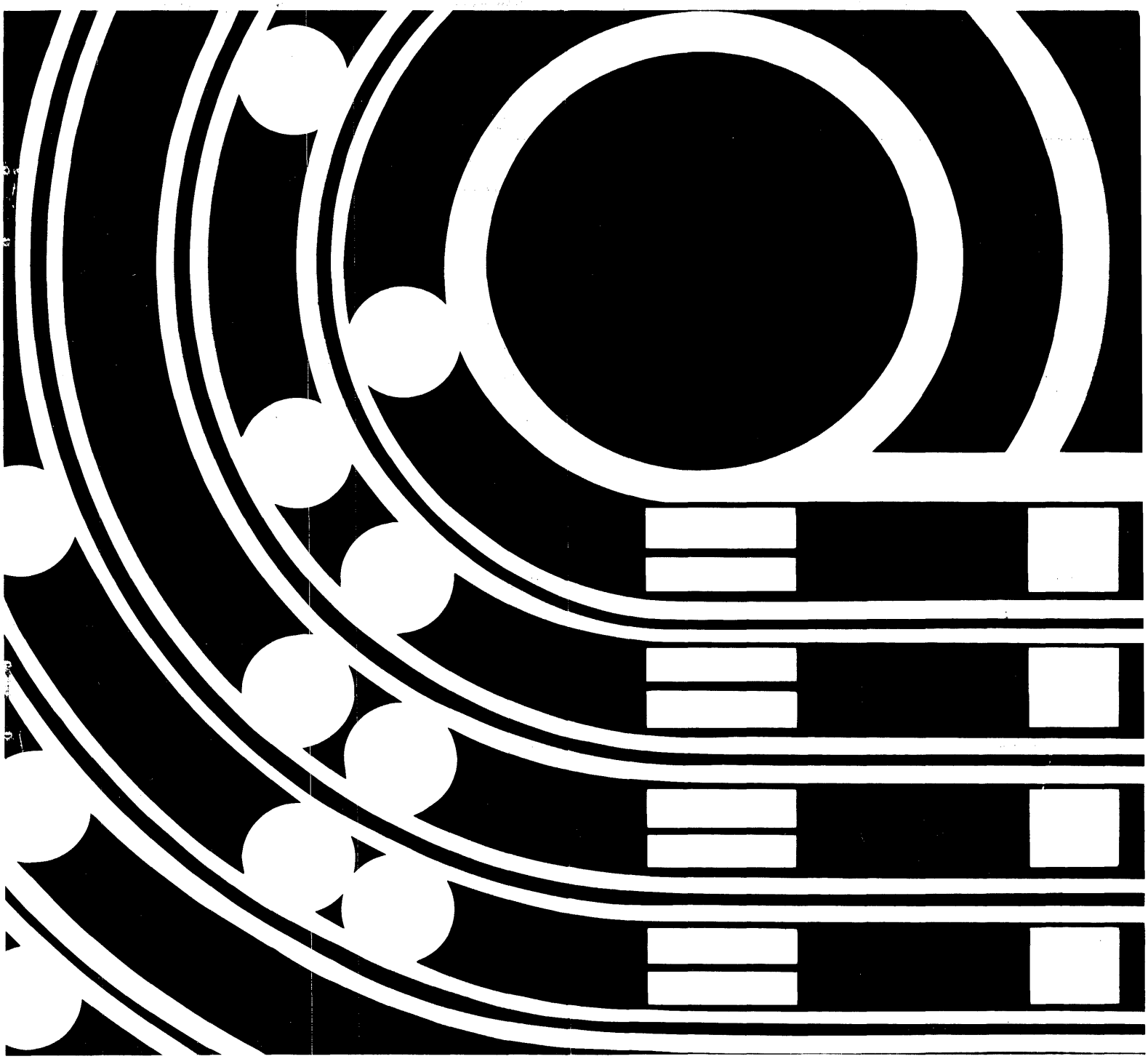


IM41-0059-00
NUTRAN USER AND PROGRAMMERS GUIDE

NUCLEAR DATA INC

NUCLEAR DATA, INC.
Post Office Box 451
Palatine, Illinois 60067

November, 1972

IM41-0059-00
NUTRAN USER AND PROGRAMMERS GUIDE

PREFACE

Most problems are presented in the form of information or data which requires some action or decision to produce a result. To solve such problems, the information or data input must be read and understood, the input must be properly manipulated or processed to produce the correct result, and the result must be distributed or read out in an intelligible form. The ND812 Computer, when properly programmed, can discern typed conversational language inputs, accurately process those inputs, and develop Teletype compatible output results.

The ND812 Computer, like other general purpose digital computers, is a complex electronic device, and normally, programming such a device would require the knowledge of a professional programmer as well as a thorough understanding of the ND812 electronics and instruction set. Fortunately, however, it is not necessary to know how the ND812 Computer operates, or to be familiar with complicated machine instructions in order to be able to develop and write simplified, but comprehensive programs. The NUTRAN Interpreter (41-0095) now provides a completely conversational language called NUTRAN, that operates similar to FORTRAN and allows any user to communicate directly with and totally exercise specific functions of the ND812 Computer.

The uses of NUTRAN are varied. Nuclear Data initially designed NUTRAN for scientific uses, and in particular, for stating mathematical and scientific problems in a language more closely associated with experimental requirements than with direct control of the ND812 Computer. NUTRAN, however, has also proven itself in many commercial and industrial applications. As specific user needs develop, any of the valid NUTRAN commands described in this manual may be implemented to further extend the practicality of NUTRAN.

TABLE OF CONTENTS

SECTION	TITLE	PAGE

APPENDICES

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I
INTRODUCTION

1.1 GENERAL

In addition to being a workhorse for Nuclear Data Systems, the ND812 Computer can also be used to solve problems using NUTRAN conversational language. This manual describes the use of the NUTRAN Interpreter (ND41-0059). Program tape loading and initialization procedures are given in Appendix A. A discussion of basic computer operation and a basic core map of the NUTRAN Interpreter are also included in this section.

1.2 EQUIPMENT REQUIRED FOR USING NUTRAN

Table 1-1 lists the required hardware for using NUTRAN.

Table 1-1. Equipment Required for NUTRAN

Minimum Hardware Requirements	Nuclear Data Part Number
ND812 Central Processor/8K Memory Model TC33ASR Teletype	88-0397/84-0097 86-0085

1.3 THE COMPUTER

Figure 1-1 shows a block diagram of a basic computer system. The main components of the system are the central processor and input/output devices. (The TC33ASR Teletype handles both input and output functions for NUTRAN.)

The central processor includes control logic to interpret, execute, and maintain the sequence of program instructions, memory storage to store directly accessible program and data information, arithmetic logic to perform mathematical calculations, and input/output logic to transfer data to/from the central processor. Refer to Principles of Programming The ND812 Computer (Nuclear Data Part Number ND IM41-0000) for a more detailed discussion of the ND812 Central Processor.

Computers require specified instructions organized in a logical sequence to solve a given problem. This sequence of instructions is called a program. Ths program directs the computer to operate on information or data from an external device (punched cards, magnetic tape, etc.) or on data developed during the execution of the program. Though the total number of instructions may be extensive, the programmer generally has at his command instructions to perform the following functions.

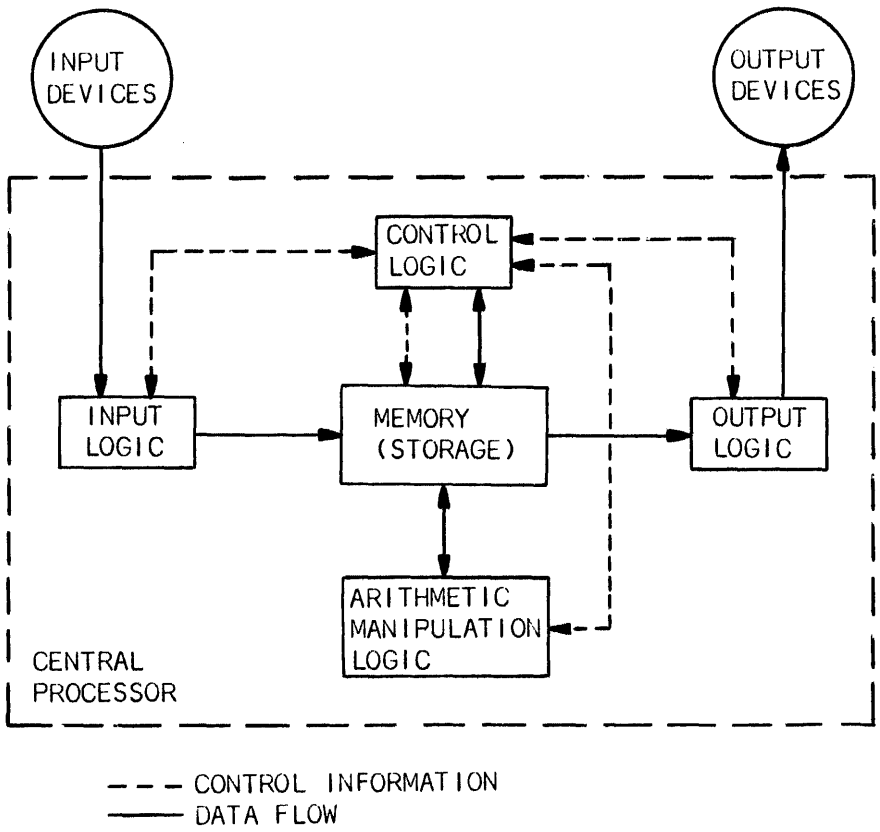


Figure 1-1. Basic Computer, Block Diagram

1) Arithmetic - permits desired mathematical calculations and manipulations.

2) Input/output - permits communication with external storage or readout devices (magnetic tapes, punched cards, etc.) to obtain specific data to be processed or to deliver resultant data to the output unit.

3) Decision making - permits comparisons of data to determine which of several possible operations is to be performed.

4) Control - permits selected operations to be performed or repeated a specified number of times and allows changes to be made in sequence of instruction execution.

1.4 CORE MAP

Figure 1-2 shows the basic core map for the NUTRAN Interpreter. The basic NUTRAN Interpreter program resides in locations ØØØ3 through 4Ø95 in memory field Ø and ØØØØ through Ø745 in memory field 1. (Each memory field contains 4096 (4K) directly accessible locations.) The text buffer, containing the user entered program, is variable in size and is built up in field 1 from location Ø745 as the program is entered via Teletype. The data base buffer, operand stack, and polish (index) stack are also variable in size and are built downward from location 3967 in field 1 when the program in the text buffer is executed. The data base buffer contains variables which are either entered via Teletype or generated by the program during execution. The polish stack contains indices to the symbols and arithmetic operators included in the symbol table of the basic NUTRAN Interpreter program. The operand stack contains the actual values assigned to the symbols indexed in the polish stack. The numbers in the operand stack appear in the same order as the respective symbols appear in the polish stack. (Refer to Appendix B for locating NUTRAN in memory fields other than Ø and 1.)

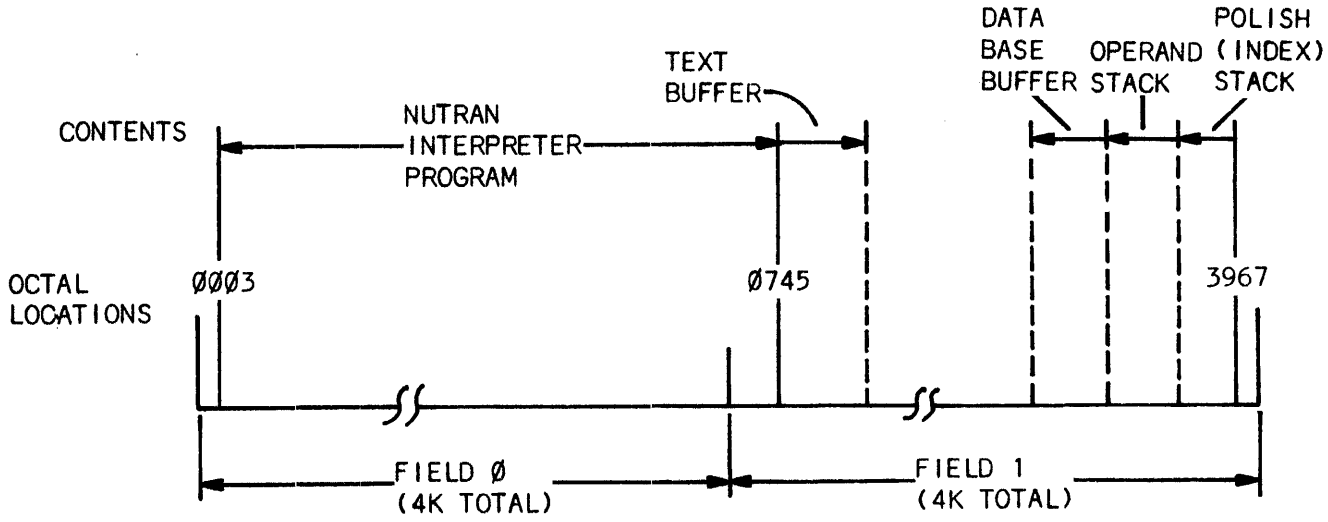


Figure 1-2. NUTRAN Interpreter, Core Map

SECTION II
FLOWCHARTING

2.1 GENERAL

Before a problem can be solved with a digital computer, it is necessary to thoroughly analyze the problem, decide on a procedure for solution, and to generate a set of step-by-step instructions to perform the procedure. This logical set of instructions is called a program, and can be represented graphically by a flowchart.

2.2 FLOWCHARTING FUNDAMENTALS

Flowcharts are basically a collection of boxes and lines. The boxes indicate what is to be done, and the lines indicate the sequence. The boxes are of various shapes which represent actions to be performed in the program. Figure 2-1 shows a set of symbols that can be used in constructing flowcharts.

For simple problems, programs can generally be written without the use of flowcharts. However, more complex problems require many steps, and writing programs for them often becomes involved and confusing. The flowchart is a map illustrating the logical steps required to solve a problem, the decisions to be reached, and the paths to be followed as a result of the decisions. Figure 2-2 shows an example flowchart describing the various steps involved in the selection and reading of a magazine.

The flowchart assumes that the person following the steps wants to read a magazine. A decision block (diamond) may have been inserted after the "START" oval to determine "DO YOU WANT TO READ A MAGAZINE?". A "YES" answer would have led the user normally to the "PICK UP A MAGAZINE" block. A "NO" answer would have led the user to the end of the program ("STOP" oval) or to another program, since the magazine selection flowchart would have been useless for him at that time.

The flowchart also assumes that it is bed-time. Another decision block (diamond) may have been inserted in place of the "GO TO SLEEP" processing block (rectangle) to determine the time of day. A branch to another decision block or perhaps to an entirely different flowchart may have then been initiated to determine the next activity to be pursued.

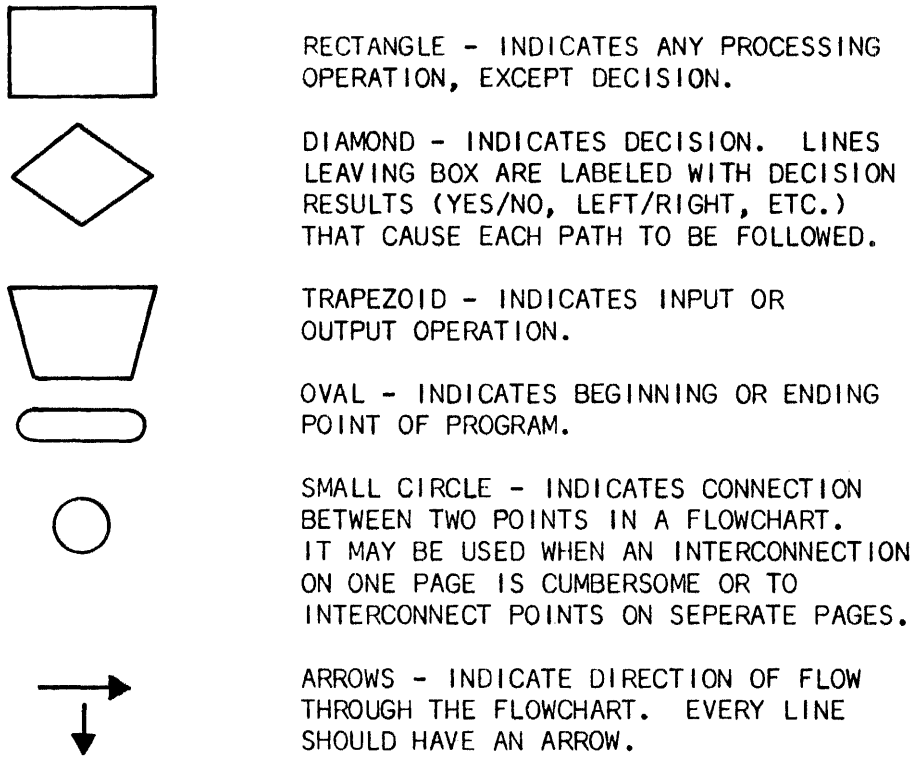


Figure 2-1. Flowchart Symbols

A flowchart may also be constructed for a mathematical problem. For example, a possible flowchart for the multiplication of two numbers (positive or negative) is shown in Figure 2-3. The first decision block (diamond) determines whether either of the inputs is negative and the second, if both are negative. If both are negative or positive, the absolute value of A ($|A|$) is multiplied by the absolute value of B and the result is printed. If one or the other is negative, a minus sign is printed and the product of the absolute values of the two entries is then calculated and printed.

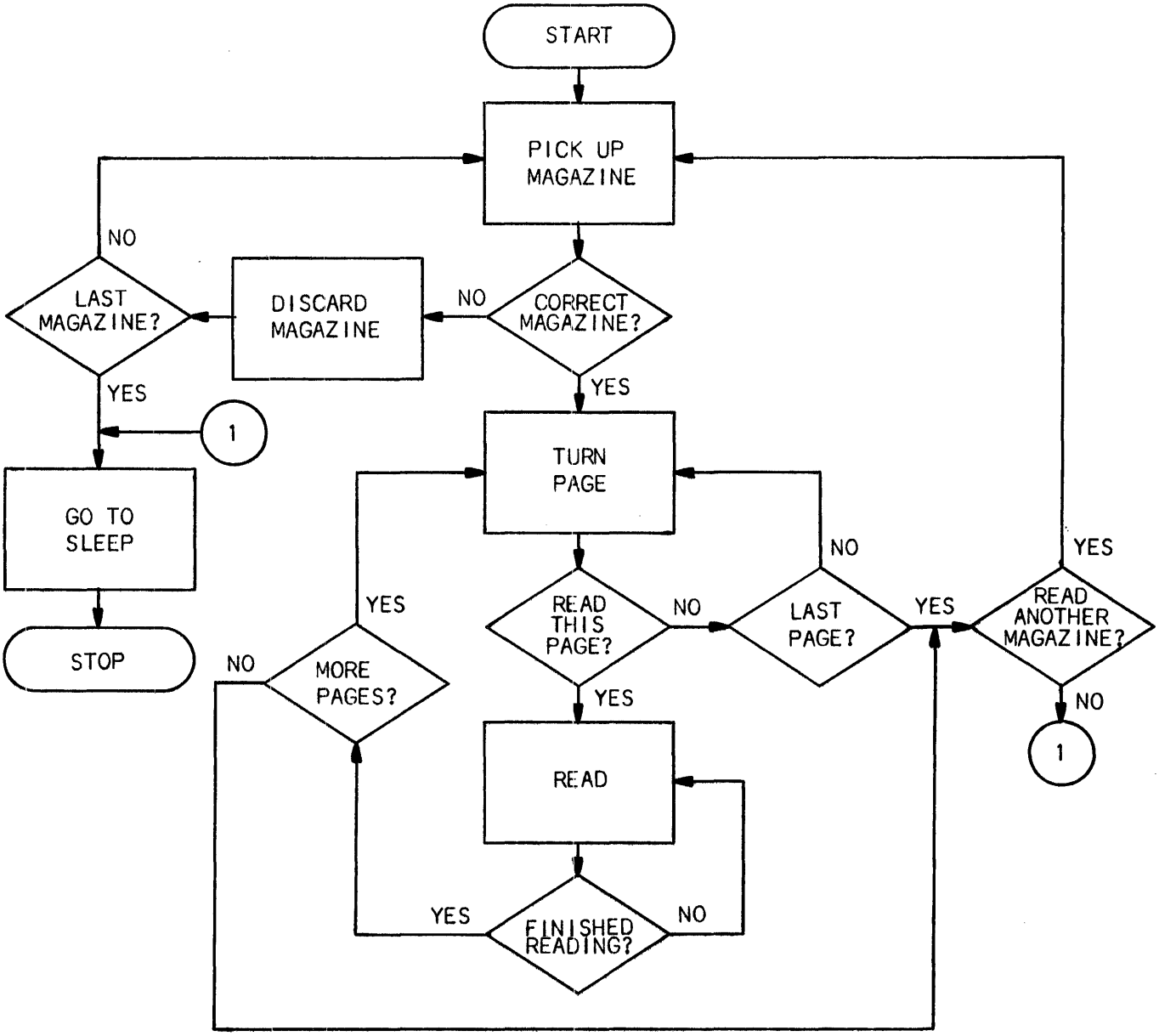


Figure 2-2. Example Flowchart For Selecting And Reading A Magazine

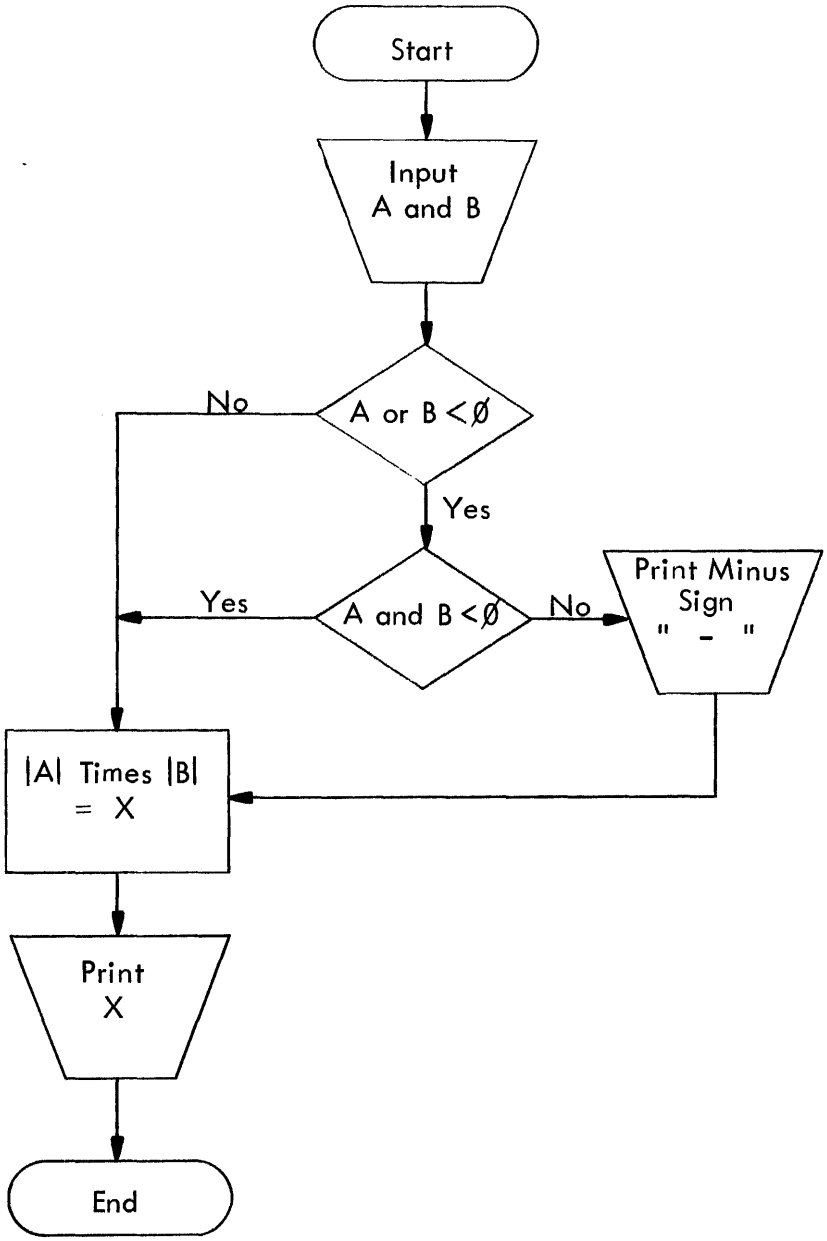


Figure 2-3. Example Flowchart for Multiplication Problem

SECTION III
NUTRAN LANGUAGE FUNDAMENTALS

3.1 GENERAL

The outstanding characteristic of NUTRAN is the continuing dialog between user and computer. NUTRAN statements are entered by the user at a remote device. When the program is executed, the statements are then automatically translated and verified as valid commands. If invalid statements are encountered in a program during execution, the interpreter responds by directing an error printout on Teletype. (Refer to Appendix C for error diagnostic information.) Also, if desired, as the program is being executed, literal messages and results of computations may be printed on Teletype. The features of NUTRAN conversational language are as follows.

1) The user has immediate and sustained access to the computer.

2) The user may selectively construct, execute, and edit statements or complete routines, change values of variables, and request information from the computer.

3) The user has diagnostic facilities to debug his NUTRAN program.

4) The user need not be concerned about integer and floating point data type formats.

This section describes the structure and specifies the contents of a NUTRAN statement.

3.2 NUTRAN STRUCTURE

The basic element in the NUTRAN language is the statement. A NUTRAN statement, like an English sentence, expresses a complete idea. A statement may be executable by specifying actions or procedures such as input/output routines or mathematical calculations. Or the statement may be non-executable by providing information such as format specifications to the Interpreter.

3.2.1 EXECUTABLE STATEMENTS

Two forms of executable statements may be written in NUTRAN. The first is written like an algebraic formula as follows.

$$A = B + C/3$$

This type of construction is called an arithmetic assignment statement. (Refer to paragraph 3.5.5 for a more detailed discussion of arithmetic assignment statements.) It indicates that the expression (B + C/3) is to be evaluated and the value calculated is to be assigned to the variable A.

The second form of executable statement consists of a word or words defined by the NUTRAN language. Such a word, when interpreted by the processor, always results in the same program action. These special words contain programmer-supplied parameters upon which or through which the action occurs. For example, in NUTRAN, there is a statement called the DO statement. The word DO tells the NUTRAN Interpreter that a group of statements is to be executed a number of times. The number of times the statements are to be executed is supplied by the programmer. Any variation in the execution of the total statement of which DO is a part is controlled by specific parameters supplied by the programmer. Refer to paragraph 5.9 for a detailed description of the DO statement.

3.2.2 NON-EXECUTABLE STATEMENTS

A NUTRAN statement may also be non-executable. Non-executable statements are directives to the NUTRAN Interpreter specifying, for example, format for print out of numerical data, or directing the Interpreter to enter a subroutine program. Format and subroutine statements are discussed in Section V.

3.3 NUTRAN CHARACTER SET

Any characters from the following character set may be used to write statements and comments in a NUTRAN program.

Alphabetics (Letters)	A through Z (written as capitals)
Numerics (digits)	0 through 9
NUTRAN Symbols	= , ( ) + - * / .
Blank Space	Denoted by a space

Up arrow (↑) which is equivalent to **, may also be used under Teletype control. ** indicates the number before the ** is raised to the power of the number following the **; i.e., X**2 (or X ↑ 2) indicates X^2.

3.4 NUTRAN STATEMENTS

NUTRAN statements may be made in one of three modes - Command mode (discussed in Section IV), Text mode (Section V), or Calculator mode (Section VI).

3.4.1 LINE NUMBERS

Each statement in Text mode must begin with a line number to identify the statement within the program and to specify the order in which statements are executed. The choice of line numbers is arbitrary but limited to six digits. The statements are executed in numerical order (although they need not be entered in numerical order). As the statements are entered, the NUTRAN program sorts and edits the program, putting the statements into the order specified by their respective line numbers.

In Command mode, a statement may or may not begin with a line number. In Calculator mode, statements begin by depressing SPACE bar and never contain line numbers.

3.4.1.1 Deleting or Replacing Line Number and NUTRAN statement

A line number and NUTRAN statement are deleted by entering the specific line number on the Teletype keyboard and depressing carriage return. A line is replaced by entering the line number and the new NUTRAN statement via Teletype keyboard and depressing carriage return.

3.4.2 SPACES

Spaces have no significance (except in literal printout character strings) and are used arbitrarily to make printout of statements more readable. In literal printout commands (paragraph 5.2.2), the space appears as specified in the command.

3.5 NUTRAN LANGUAGE STRUCTURE

3.5.1 CONSTANTS

Any literal, explicit number in a statement is called a constant. A quantity that is given a name is called a variable. Either real or integer constants may be entered, but all constants, whether real or integer are stored by NUTRAN as real (floating point) numbers. Representing numbers in floating point form is a method similar to scientific notation, in which a number is treated as a fraction (between 0.1 and 1.0), times a power of 10. The magnitude of the number so represented lies approximately between the limits of 10^{+300} and 10^{-300} or is zero.

All numerical constants may be preceded by a plus or minus sign to specify positive or negative values. Unsigned constants are always considered positive values by NUTRAN. The following are acceptable integer constants.

0	-300
+600	54321

The following are acceptable real constants.

0.0	-200
6.0	-.00123
6.	+15.25

The decimal point may appear at the beginning or end of a number or between two digits. A floating point constant may have any number of digits, but only seven decimal digits of significance are retained by the computer.

Constants may be followed by the letter D or E and a positive or negative power of ten by which the number is multiplied. This simplifies writing very large or very small numbers. Thus, the following are further examples of acceptable floating point constants.

Constant	Equivalent to
5.0E+6	5.0×10^{6}
6.25E8	6.25×10^{8}
2.89D2	2.89×10^{2}
-7.0E3	-7.0×10^{3}
4.33E-4	4.33×10^{-4}

3.5.2 VARIABLE IDENTIFIERS

A variable identifier is a name given to a designated quantity whose value may change during execution of a program. The name of a variable consists of one or more alphanumeric characters, the first of which must be alphabetic. Only the first two characters are interpreted as defining the variable name and the rest are ignored. Therefore, each variable name must be unique within the first two characters.

Variables are always stored as real values. The value range allowed a real variable is the same as that for a real constant (zero or any constant within the range of $10^{\pm 300}$). Examples of acceptable and unacceptable variable identifiers are as follows.

Acceptable	Unacceptable
KK	31
J2	2K
MK	$X

It is recommended that the programmer assign names that simplify recall of the meaning of the variable, but no such meaning is attached to the symbols by the NUTRAN Interpreter. Every combination of letters and digits constitutes a separate name. Thus, the name "AB" is not identical to the name "BA", and the names "A" and "B" are distinct.

3.5.3 ARRAYS

An array is a group of data with identical variable names, but with different subscripts. Each unit of data is called an element of the array and is specified by the subscript. The NUTRAN Interpreter allows for only one-dimensional arrays with a maximum of 4095 elements (subscripts).

3.5.3.1 Declaring An Array

In the NUTRAN Interpreter, it is not necessary to declare an array (unlike FORTRAN, where a dimension statement is required). Single-dimensional arrays are assigned only as needed and are referenced by name and specified subscript, allowing for dynamic allocation of storage during program execution.

3.5.3.2 Subscripts

The position in an array is identified by specifying the array name and the particular subscript. A subscript can be an expression containing one or more terms and must not exceed a value of 4095. The subscript must also be enclosed by parentheses. For example, an array A(I) may be defined in a program where A(I) is the value of I multiplied by 2, or A(I)=I*2. Then, I may be incremented from 1 to 3, such that an array is developed as follows.

A(1)=2
A(2)=4
A(3)=6

3.5.4 ARITHMETIC EXPRESSIONS

An arithmetic expression is classified as one or more terms separated by an operator, which, when evaluated, produce(s) a single value. In NUTRAN, a term can be a constant, a variable, a subscripted variable (an array element), or a function (intrinsic or user written-paragraph 3.5.7). A term in an expression is considered a reference to data (i.e., the current value of the term designated by a name is made available for processing during the execution of a given statement). The general form of an arithmetic expression is as follows.

term, operator, term, operator. . . . term

The simplest form of an expression is a single term. The following are examples of simple expressions.

Constants	5	3.14
Variables	IN	TAX
Array Elements	ON (1)	TX (4)
Function (intrinsic or user)	ABS (X)	SQRT (1)

3.5.5 ARITHMETIC OPERATORS

NUTRAN provides six basic arithmetic operators. An operator specifies that an action is to be performed on data. An arithmetic expression is formed by combining terms with any of the arithmetic operators shown in Table 3-1.

Table 3-1. Arithmetic Operators

Operator	Arithmetic Function
** or ↑	Exponentiation
-	Unary minus
/	Division
*	Multiplication
+	Addition
-	Subtraction

NOTE

The unary minus operator is used to sign a number or variable. An example of this is -A (or -5.3). Unsigned constants or variables are assumed to be positive.

If A and B represent two terms in an expression, then the following arithmetic expressions are valid;

1) A**B equals the value of A raised to the power B. (A must be positive. If not, an error indication is printed. Refer to Appendix A.)

2) A/B equals the value of A divided by the value B. (B must be non-zero. If not, an error indication is printed. Refer to Appendix A.)

3) A*B equals the value of A multiplied by the value of B.

4) A+B equals the value of A plus the value of B.

5) A-B equals the value of A minus the value of B.

The following arithmetic expressions are also valid.

1) 7.0

2) A

3) 3*1

4) 3*1+2

5) A+D*E-G/X

3.5.5.1 Hierarchy of Operators

When arithmetic expressions are evaluated, the arithmetic operations are performed according to the rules of precedence shown in Table 3-2.

Table 3-2. Arithmetic Operations Hierarchy

Operator	Arithmetic Operation
( )	Parentheses
**	Exponentiation
UNARY -	MINUS
* and /	Multiplication and division
+ and -	Addition and subtraction

Evaluation of an expression begins with the operator of highest precedence. If operators are of the same level, the execution is from left to right within the expression (including ** which in FORTRAN is evaluated from right to left). The following examples illustrate the evaluation of expressions.

2**3-4 = 4	Exponentiation is performed before subtraction (8-4 = 4).
3**2/3+5 = 8	Exponentiation is performed before division and division before addition (9/3+5 = 3+5 = 8).
5+4*2**3 = 37	(5+4*8 = 5+32 = 37).

3.5.5.2 Parentheses

There are instances in which the rules of precedence are insufficient to represent the desired sequence of operations in an expression. For example, the expression

$$\frac{A + B}{A - B}$$

cannot be expressed according to the rules of precedence. To write it as A + B / A - B is incorrect, because this represents the expression

$$A + \frac{B}{A} - B$$

Therefore, another method of defining A + B and A - B as elemental factors of the division operation is necessary. Parentheses do this by altering the heirarchy of operations (causing NUTRAN to evaluate the expression within the parentheses) before proceeding with the rest of the expression. Therefore, to write $\frac{A + B}{A - B}$ in NUTRAN, coding should be (A + B) / (A - B). Other examples of using parentheses to establish precedence are as follows.

5+3*2+4/2	equals	13, whereas
(5+3*2+4)/2	equals	7.5, and
((5+3)*(2+4))/2	equals	24.

When an expression has several levels of expressions enclosed within parentheses, the evaluation begins with the innermost pair of parentheses and proceeds to the outermost pair. Each parenthetical expression is treated as an elemental term. Parentheses may be used freely and as often as desired to simplify and clarify a NUTRAN statement.

3.5.6 ARITHMETIC ASSIGNMENT STATEMENTS

An arithmetic assignment statement assigns a single arithmetic value of an expression or variable to the name of a variable or subscripted variable. This assignment statement causes the arithmetic value of the expression to the right of the equal sign to replace any previous value for the variable or array element to the left of the equal sign. Thus, a statement such as N = N +1 indicates that 1 is added to the present value of variable N and that this result replaces the value in N. This effect is comparable to incrementing N by 1 and does not imply that N equals N + 1. The general form of the arithmetic statement is

variable identifier = expression

and is interpreted: the current value of the variable identifier is replaced by the evaluated expression.

3.5.7 SUBPROGRAM STATEMENTS

At various points in a program, operations may be required which are identical (or almost identical) to others. NUTRAN provides for a method of writing them once in a general form, and making them available as many times as necessary throughout a given program. Each such commonly used statement or group of statements is called a subprogram, and each has a unique reference name in a program. In NUTRAN, there are three basic kinds

of subprograms; intrinsic functions, subroutines, and user written functions.

3.5.7.1 Intrinsic Function Statements

NUTRAN intrinsic function statements are provided by the NUTRAN Interpreter and facilitate the use of common mathematical expressions, such as square root, logarithm, exponential, and absolute value. Every function has a preassigned name. For instance, the name for the exponential function is EXP. The valid NUTRAN mathematical functions are as follows.

Mathematical Function	NUTRAN Name
Square Root	SQRT
Exponential	EXP
Natural Logarithm	ALOG
Logarithm base 10	ALOG 10
Absolute Value	ABS

To make use of a mathematical function, it is necessary only to write the name of the function and to follow it with an expression enclosed in parentheses. This directs NUTRAN to compute the named function of the value represented by the expression in parentheses. For example, if the square root of a value is needed for part of a computation, the square root function in a statement assumes the form:

$$X = Y - SQRT(Z)$$

where SQRT is the function and Z is the argument. (Arguments for intrinsic functions must be enclosed in parentheses.) The value SQRT (Z) is computed and subtracted from the value of Y. The result replaces the value for X.

3.5.7.2 Subroutine Statements

A subroutine is a subprogram which the programmer must write and input to the main program. Arguments for subroutines may or may not be used, but if they are used, must be enclosed in parentheses. Refer to Paragraph 5.10 for a detailed description of the use of subroutines.

3.5.7.3 User Written Function Statements

User written function statements are subprograms that operate like NUTRAN intrinsic functions but are not provided by NUTRAN. Refer to Appendix D.

SECTION IV
COMMAND MODE

4.1 GENERAL

NUTRAN operates in either Command, Text, or Calculator mode. Command mode is described in this Section. Sections V and VI discuss Text and Calculator modes, respectively.

In Command mode, all inputs typed on Teletype request the NUTRAN Interpreter to perform some operation. To operate in Command mode, the user may or may not type a line number as desired, but a period, the desired Command code, and a terminating carriage return must be entered. The carriage return is a non-printing character represented by the letters CR. CR signals the Interpreter to process the information just typed. No command is executed until it is terminated by CR. After the command is executed by NUTRAN, the program enters Text mode, responds with a carriage return/line feed, and types > .

4.2 COMMAND DESCRIPTION

In Command mode, there are five one-letter commands to the NUTRAN Interpreter. These commands are as follows.

1) .L CR - Lists all variables and constants in the symbol table. The period must precede the "L" and carriage return (CR) must be depressed after typing "L". If the command is preceded by a statement line number, the listing includes only the variables and constants from that line to the end of the symbol table.

2) .W CR - Prints the entire program in the text buffer. The command may also be of the form,

   200.W CR

   where only the statement identified by line 200 (or any specified line number) is printed. In either form, the period must precede the "W" and the command must be followed by CR.

3) E CR - Erases the entire text buffer and the symbol table. The command may also be of the form,

200.E CR

where only the symbol table and statement 200 (or any specified line number) and larger in that text buffer are erased. In either form, the period must precede the "E" and the command must be followed by CR.

4) .N CR - Prints the next line in the text buffer. The period must precede the "N" and the command must be followed by CR.

5) .G CR - Executes the program in the text buffer. The command may also be of the form,

300.G CR

where execution of the program begins at line 300 (or any specified line number in the text buffer). In either form, the period must precede the "G" and the command must be followed by CR.

SECTION V
TEXT MODE – NUTRAN CONTROL STATEMENTS

5.1 GENERAL

In Text mode, all typed inputs (when terminated by carriage return CR) replace or are added to the contents of the program text, providing a means for the programmer to code, update and execute a program on-line. This eliminates the tedious task of compiling, executing and debugging a program off-line, and reduces the time necessary to develop new programs.

Adding/deleting statement lines in Text mode and Teletype keys having special functions in text mode are discussed in this section.

In order to actually write a program in Text mode, specific statements are required. This section describes NUTRAN control statements INPUT, PRINT, CONTINUE, STOP, FMT (format), LIST, and ERASE; transfer of control statements IF, GOTO, and DO; subprogram statements CALL, SUBROUTINE, and RETURN; and special statements FISET, GET, and PUT. Examples are given to illustrate the use of each statement. To understand the examples, it is recommended that the user be familiar with the language fundamentals given in Section III and Command mode given in Section IV. Also, in order to exercise the examples given, the NUTRAN Interpreter program (ND41-0059) must be loaded into the ND812 Computer as described in Appendix A.

5.1.1 ADDING/DELETING STATEMENT LINES

To add a statement to a program, the user must be in Text mode. The operator need only enter a line number and follow it with the new NUTRAN statement. At the end of the statement, the operator depresses CR and the processor enters the line into the program at the location specified by the line number. Valid statements are described in this section.

To delete a statement from a program, the operator must also be in Text mode. He need enter only the line number of the NUTRAN statement to be deleted and then depress the CR key. The processor deletes the indicated statement line from the program and returns control to the operator (Text mode) for further commands.

5.1.2 SPECIAL TELETYPE KEY COMMANDS

The following Teletype keys have special operating functions in Text mode.

1) CARRIAGE RETURN (CR - nonprinting) - Depressing the CR key causes the line of text (NUTRAN statement) preceding it to be entered into the text buffer where the program is stored. A typed line does not become part of the buffer until it is terminated by CR.

2) BACK ARROW (←) - The back arrow is used for error recoveries and cancels everything to the left of itself back to the line number. The user continues typing on the same line.

3) RUB-OUT (\) - Rub-out is also used in error recovery. Typing a rub-out prints " \ " and deletes the last typed character.

4) ALT (non-printing) - ALT key is used to ignore the line being entered (assuming CR has not been depressed for that line). A new line number must then be entered to begin a new statement.

5.2 INPUT/PRINT CONTROL STATEMENTS

Solution of a problem usually requires that an input be supplied before a result is obtained. Therefore, the programmer must have some means of supplying an input for execution of his program. Also, when the program has been executed, the results must be returned to the programmer in some intelligible form. NUTRAN conversational language provides the INPUT/PRINT control statements for transfer of information into and out of the ND812 Computer via Teletype. The INPUT control statement provides manual data entry to program via Teletype keyboard, and the PRINT control statement provides hardcopy printout of results. The following is a discussion of the INPUT and PRINT statements.

5.2.1 INPUT STATEMENT

In a conversational language, data is supplied to the program by the operator through a Teletype. This is particularly true when one person writes the program and another supplies the data. The NUTRAN program receives data directly from the operator via the Teletype by use of the INPUT statement. An example of the INPUT statement format is,

```
INPUT  A
```

The form of the data being entered for an INPUT statement must be either an integer or real constant. For example, if the person writing the program wants the user to supply a value for A and B in the program, he might use the statement,

```
INPUT A, B
```

When the program encounters this statement, a colon is printed at the Teletype. The user must then enter a value for A and enter a terminating character (carriage return key, space bar, or any character other than a number or decimal point). Another colon is printed and a value for B must be typed. After again entering a terminating character, the program continues until another INPUT statement is encountered or until the end of program is reached.

5.2.2 PRINT STATEMENT

In a conversational language, it may be desirable for a program to communicate with the operator or user in the form of literal character strings. A NUTRAN program does this through Teletype by using PRINT statement. The formats for the PRINT statement are,

PRINT 'message'

PRINT ' message', Variable Identifier

PRINT Variable Identifier 1, Variable Identifier 2,....Variable Identifier N

For example, PRINT 'X+Y=', Z causes "X+Y=" to be printed literally followed by the calculated value for the variable identifier "Z". The PRINT statement may be used to,

1) Skip a line and print a message on the following line using a slash as follows: PRINT /, 'message'.

2) Print out the results of calculations, as desired.

3) Print out messages included in the program.

4) Perform combinations of 2 and 3.

Examples of PRINT statements are as follows.

1) 12 PRINT X, Y, Z (2)
   NUTRAN prints the value of X and then the value of Y. The second element of array Z is printed to the right of Y.

2) 14 PRINT 'HEADED LINE'
   Line 14 prints the statement HEADED LINE literally. Groups of information can be printed out across the page by enclosing information in single quotes and separating items within the group with commas.

3) 16 PRINT 'SQUARE OF', X, 'IS', Y...
   In this example, messages and variables are combined in one PRINT statement. The message SQUARE OF is printed literally, and is followed by the value

of X. The word IS is printed literally and followed by the value of Y, where Y was previously calculated as X*X. At the execution of the statement, this printout might be as follows.

SQUARE OF 4.0 IS 16.0

5.3 STOP STATEMENT

Execution of the STOP statement causes termination of the program. This statement is interpreted as the logical end of the program rather than the physical end. More than one STOP statement may appear within a program. For example, a program could have a STOP statement at the end of each logical program path.

5.3.1 EXAMPLE PROGRAM USING INPUT, PRINT AND STOP STATEMENTS

As an exercise, the following example program may be entered into the ND812 Computer via keyboard. The printout shown below is obtained by typing .W after the program is entered.

NOTE

Every statement must be preceded by a line number.

```
1 PRINT 'INPUT VALUES FOR X AND Y'
2 INPUT X,Y
3 Z=X+Y
4 PRINT 'X+Y= ',Z
5 STOP
```

Now enter 1.G and depress carriage return at Teletype keyboard. The program prints line 1 literally and then types a colon (:) requesting an input for the variable X. After the value for X is entered and carriage return is depressed, another colon is typed for Y. When Y is entered and carriage return depressed, the program types the requested printout. The following printout shows a calculation for X = 3 and Y = 2 using the above program. NUTRAN then returns to Text mode and types > .

```
>1.G
INPUT VALUES FOR X AND Y
:3
:2
X+Y=     .5000000E  1

>
```

The values for X and Y in the above example may also be terminated by depressing space bar. Using the same inputs for X and Y, the following printout would then result.

```
>1.G
INPUT VALUES FOR X AND Y
:3 :2 X+Y=     .5000000E  1

>
```

By using space bar to terminate inputs, columns may be generated to clarify entries. For example the above program may be rewritten as follows. The slash, "/", is included in line 5 to assure that a line is skipped after the last entry.

```
1 PRINT 'INPUT VALUES FOR X AND Y'
2 PRINT ' X  Y'
3 INPUT X,Y
4 Z=X+Y
5 PRINT /,'X+Y= ',Z
6 STOP
```

Then, by again entering 3 and 2 for X and Y and terminating those entries with space bar, the following printout results.

```
>1.G
INPUT VALUES FOR X AND Y
 X  Y
:3 :2
X+Y=     .5000000E  1

>
```

5.4 (FMT) FORMAT STATEMENT

The format (FMT) statement allows the user to specify the manner in which the output from the PRINT statement is arranged. There are three different ways to format the data.

1) Integer (I) - the data is printed as an integer, i.e., without decimal point or fraction portion. For example,

   2.69 is printed as 3

2) Floating (F) - the data is printed as a floating number, with decimal and fraction. For example,

   2.69 is printed as 2.69

3) Exponential (E) - the data is printed as a fraction and is followed by a signed power of ten. For example,

   2.69 is printed as .269E1

5.4.1 INTEGER (I) FORMAT STATEMENT

Integer format statement is of the general form FMT (In), where "I" specifies integer format and "n" is the field width. Field width specifies that the right justified number (with leading blanks) is printed. For example,

```
FMT (I 10)
PRINT  2, -1234
```

prints the following.

```
          n=10
     ┌─────────┐
bbbbbbbbb2 bbbbb-1234
└────────┘
    n=10
```

where "b" indicates "space".

5.4.2 FLOATING (F) FORMAT STATEMENT

Floating format statement is of the general form FMT (Fw,d) where "F" specifies floating format, "w" is the field width, including sign and decimal point, and "d" is the number of digits to the right of the decimal point. The "w" must be $\geq$ d + 2 to allow for the decimal point and at least one digit. For example,

```
FMT (F8,3)
PRINT  22.355
```

prints the following.

```
      w=8
  ┌──────┐
  bb22.355
      └─┘
       d=3
```

where "b" indicates "space".

NOTE

If a number to be printed exceeds the number of digits specified in the I or F format statement, an error printout results. The error indication includes an asterisk (*) for each field digit position specified in the I or F format statement. For example,

```
FMT (I6)
PRINT  1234567
```

results in the following indication.

******

5.4.3 EXPONENTIAL (E) FORMAT STATEMENT

Exponential format statement is of the general form FMT (Ew,d) where "E" specifies exponential format, "w" is the field width, including fraction sign, decimal point, E, exponent sign and exponent, and "d" is the number of digits in the fraction. The "w" must be $\geq$ d+5. If fraction and exponent signs are not indicated, positive is assumed. For example,

```
FMT (E10,3)
PRINT 25
```

prints the following.

```
     w=10
  ┌──────┐
  bb.250Ebb2
     └─┘
      d=3
```

where "b" indicates "space".

NOTE

Integer, floating, and exponential format entries may be made in the same program regardless of format statement specifications.

5.4.4 DEFAULT FORMAT

If the format is not specified in a program, FMT (E 15,7) is assumed. (Paragraph 5.4.3 describes E format.) Also, only one format may be specified in a single statement line. That format is then used for each succeeding output until another format statement (on another line) is encountered.

5.4.5 ROUNDING

The NUTRAN rounding procedure depends upon three variables; (1) format type, (2) decimal exponent of data, and (3) in the case of F and E formats, the number of digits to be printed to the right of the decimal point. A fourth factor may also be important and should be clearly understood to avoid confusion. The fourth factor is the number of accurate decimal digits expectable from the word size used in ND812 NUTRAN. Normally, one can expect seven decimal digits of accuracy. Since machine representation is often an approximation of the true decimal number, large numbers or numbers where more than seven digits are to be printed, might not produce the expected result. Inspection, however, shows that the error, if any, is in the seventh or, more likely, the eighth digit.

The rounding method generally adds 5 to the number at the first decimal digit to the right of the rightmost digit being outputed. For example, in integer format,

```
FMT (I5)
PRINT   12345.9
```

is rounded as follows,

```
 12345.9
   + .5
-------
 12346.4
```

and prints this result.

```
12346
```

In the following floating format example, the number is rounded and printed as shown.

```
FMT (F 12,3)
PRINT     3.1234
```

rounding,

```
 3.1234
+.0005
------
 3.1239
```

and the result is

```
3.123
```

However, using the same format statement, but a larger number, the following occurs.

```
FMT (12,3)
PRINT   1234.9999
```

The number theoretically is rounded as follows.

```
 1234.9999
   +.0005
---------
 1235.0004
```

and prints

```
1235.000
```

But word size could result in lost accuracy in the eighth digit and the result could be,

```
1234.999
```

Using exponential format, the following rounding occurs.

```
FMT (E 10,3)
PRINT  .004326
```

rounding,

```
  .004326
+ .000005
 -------
  .004331
```

and the result is,

```
.433E-2
```

5.4.6 EXAMPLE PROGRAM USING FMT STATEMENT

The following is an example program for addition, subtraction, multiplication, and division which may be entered at the Teletype keyboard. A format statement FMT (F7,2)is used as shown at line 122. The printout was generated by typing .W after the program was entered.

```
100 ERASE
105 PRINT 'ADDITION EXAMPLE'
110 PRINT /,'CALCULATE A+B, A+B+C, AND A+B+C+D',/
115 PRINT 'INPUT VALUES FOR A, B, C, AND D'; INPUT A,B,C,D
120 X=A+B; Y=X+C; Z=Y+D
122 FMT(F7,2)
125 PRINT 'A+B IS ',X
130 PRINT 'A+B+C IS ',Y
135 PRINT 'A+B+C+D IS ',Z
140 PRINT 'END OF ADDITION EXAMPLE PROGRAM',/,/,/,/
150 ERASE
155 PRINT 'SUBTRACTION EXAMPLE'
160 PRINT /,'CALCULATE A-B, B-C, AND C-A',/
165 PRINT 'INPUT VALUES FOR A, B, AND C'; INPUT A,B,C
170 X=A-B; Y=B-C; Z=C-A
175 PRINT 'A-B IS ',X
180 PRINT 'B-C IS ',Y
185 PRINT 'C-A IS ',Z
190 PRINT 'END OF SUBTRACTION EXAMPLE PROGRAM',/,/,/,/
198 ERASE
200 PRINT 'MULTIPLICATION EXAMPLE AND USE OF PARENTHESES'
205 PRINT /,'CALCULATE A*B, A*B*C, AND A*(B+C)',/
210 PRINT 'INPUT VALUES FOR A, B, AND C'; INPUT A,B,C
215 X=A*B; Y=X*C; Z=A*(B+C); T=A*B+C
220 PRINT 'A*B IS ',X
225 PRINT 'A*B*C IS ',Y
230 PRINT 'A*(B+C) IS ',Z,/,'WHILE A*B+C IS ',T
```

```
235 PRINT 'SHOWING HIERARCHY OF OPERATIONS WITHIN PARENTHESES'
240 PRINT 'END OF MULTIPLICATION EXAMPLE PROGRAM',/,/,/,/
250 ERASE
255 PRINT 'DIVISION EXAMPLE'
260 PRINT /,'CALCULATE A/B, A+B/C+D, AND (A+B)/(C+D)',/
265 PRINT 'INPUT VALUES FOR A, B, C, AND D'; INPUT A,B,C,D
270 X=A/B; Y=A+B/C+D; Z=(A+B)/(C+D)
275 PRINT 'A/B IS ',X
280 PRINT 'A+B/C+D IS ',Y
285 PRINT '(A+B)/(C+D) IS ',Z
290 PRINT 'END OF DIVISION EXAMPLE '
300 STOP
```

The above program is executed by typing 100.G and depressing carriage return. The following printout shows an example execution of the program. Each colon (:) shown in the example is followed by a number entered via Teletype. Otherwise, all other dialog and printout is program initiated.

```
>100.G
ADDITION EXAMPLE

CALCULATE A+B, A+B+C, AND A+B+C+D

INPUT VALUES FOR A, B, C, AND D
:2
:3
:4
:5
A+B IS     5.00
A+B+C IS     9.00
A+B+C+D IS    14.00
END OF ADDITION EXAMPLE PROGRAM



SUBTRACTION EXAMPLE

CALCULATE A-B, B-C, AND C-A

INPUT VALUES FOR A, B, AND C
:2
:4
:9
A-B IS    -2.00
B-C IS    -5.00
C-A IS     7.00
END OF SUBTRACTION EXAMPLE PROGRAM
```

```
MULTIPLICATION EXAMPLE AND USE OF PARENTHESES

CALCULATE A*B, A*B*C, AND A*(B+C)

INPUT VALUES FOR A, B, AND C
:3
:6
:5
A*B IS    18.00
A*B*C IS    90.00
A*(B+C) IS    33.00
WHILE A*B+C IS    23.00
SHOWING HIERARCHY OF OPERATIONS WITHIN PARENTHESES
END OF MULTIPLICATION EXAMPLE PROGRAM



DIVISION EXAMPLE

CALCULATE A/B, A+B/C+D, AND (A+B)/(C+D)

INPUT VALUES FOR A, B, C, AND D
:11
:7
:34
:2
A/B IS     1.57
A+B/C+D IS    13.20
(A+B)/(C+D) IS      .50
END OF DIVISION EXAMPLE

>
```

5.5 LIST AND ERASE STATEMENTS

The contents of the symbol table may be printed by entering a LIST command. To erase the contents of the symbol table, an ERASE command may be entered. For example, the program in Paragraph 5.3.1 may be rewritten as follows to erase the symbol table before entering variables and to then list the symbol table to assure that the new variables are entered correctly.

```
1 ERASE
2 PRINT 'INPUT VALUES FOR X AND Y'
3 INPUT X,Y
4 Z=X+Y
5 PRINT 'X+Y= ',Z
6 LIST
7 STOP
```

The program is then executed normally with the following result.

```
>1.G
INPUT VALUES FOR X AND Y
:3
:2
X+Y=     .5000000E  1

Z (   0)     .5000000E  1
Y (   0)     .2000000E  1
X (   0)     .3000000E  1

>
```

5.6 CONTINUE STATEMENT

CONTINUE is a dummy statement which causes no action when executed within a program. Its function merely satisfies the rule that the last statement in the range of a DO must not be one which can cause transfer of control. (Refer to paragraph 5.9 for a discussion of the DO statement.) It is also used to provide a statement to which an IF can transfer when the computations in the range of a DO have been completed. This is necessary because a transfer within the range of a DO is normally not permitted to return to the DO itself. Transfer is legal, however, if it is actually desired to repeat execution of the DO loop. (Refer to paragraph 5.7 for a discussion of the IF statement.) An example use of the CONTINUE statement is included in the DO statement example in paragraph 5.9.3.

5.7 IF STATEMENT

The arithmetic IF provides a three-way test on the value of an arithmetic expression. An arithmetic IF contains an arithmetic expression which is evaluated as less than, equal to, or greater than zero. The specific evaluated result dictates where program control is to be transferred. The general form of the arithmetic IF is,

IF (arithmetic expression) expression 1, expression 2, expression 3

If the value of the arithmetic expression is negative, a branch is executed to the statement specified by expression $_1$; if it is zero, a branch to statement specified by expression $_2$ is executed; and if it is positive, a branch to the statement specified by expression $_3$ is executed. The following examples illustrate the use of the IF statement.

```
  5 IF (A - 5) 10, 15, 20
100 IF (B*SQRT(Q) - 2), X, X+3, 17
```

In the first IF example, assuming statement lines exist with line numbers 10, 15, and 20,

If A - 4, control is transferred to statement 10.
If A = 5, control is transferred to statement 15.
If A - 6, control is transferred to statement 20.

In the second IF example, assuming statement lines exist with line numbers X, X+3, and 17,

If (B*SQRT (Q) - 2) < 0, control is transferred to statement X.
If (B*SQRT (Q) - 2) = 0, control is transferred to statement X+3.
If (B*SQRT (Q) - 2) > 0, control is transferred to statement 17.

NOTE

If a statement number specified by a branch from an IF statement does not exist, an error indication is printed. Refer to Appendix C for error diagnostics.

5.7.1 EXAMPLE PROGRAM USING IF STATEMENT

This paragraph provides a method for computing the sum of squares of 5 numbers which are entered into the ND812 Computer via the INPUT statement. The following is the mathematical notation for this problem.

$$SUMSQR = \sum_{i=1}^{5} Xi^2$$

A flowchart of a procedure for planning the sequence of operations for writing a program for computing the above problem is given in Figure 5-1.

The following is a sample program which may be entered at the Teletype keyboard for computing sum of squares using the IF statement.

```
400 ERASE
402 PRINT 'SUM OF SQUARES EXAMPLE'
403 PRINT 'COMPUTES SUM OF X(I) SQUARED, WHILE I GOES FROM 1 TO 5'
410 SUMSQR=0
420 I=1
430 PRINT 'INPUT A VALUE FOR X';INPUT X
440 I=I+1
450 SUMSQR=SUMSQR+X**2
455 FMT(F6,2)
460 IF(I-5)430,430,470
470 PRINT 'SUM OF SQUARES = ',SUMSQR
480 PRINT 'END OF COMPUTATION'
490 STOP
```

Now enter 400.G and depress carriage return at Teletype keyboard. The result is as follows.

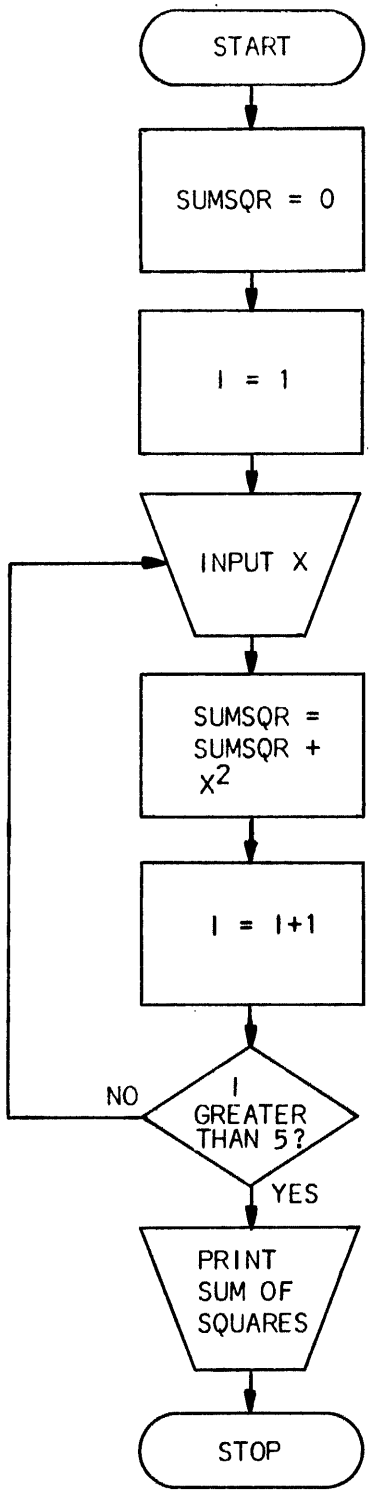


Figure 5-1. Flowchart For Computing Sum of Squares

```
>400.G
SUM OF SQUARES EXAMPLE
COMPUTES SUM OF X(I) SQUARED, WHILE I GOES FROM 1 TO 5
INPUT A VALUE FOR X
:2
INPUT A VALUE FOR X
:8
INPUT A VALUE FOR X
:4
INPUT A VALUE FOR X
:5
INPUT A VALUE FOR X
:1
SUM OF SQUARES = 110.00
END OF COMPUTATION

>
```

5.8 GOTO STATEMENT

The GOTO statement provides a means of transferring control to some statement other than the next in sequence and is of the form GOTO n, where n is either a statement number or variable name that has been assigned a value equal to a statement number in the program. When such a statement is encountered, the next statement executed is the one specified by the statement number. The unconditional GOTO statement is one in which n is a statement number which cannot change during a program run. The conditional GOTO is one in which n is a variable name which can change during a program run with an arithmetic assignment statement. The statement number transferred to is the one last assigned the variable name. The following are examples of GOTO statements.

GOTO 99 (unconditional)
GOTO X (conditional assuming a statement line number exists for value of X)
GOTO X+3*Y (conditional assuming a statement line number exists for value of X+3*Y)

5.8.1 EXAMPLE PROGRAM USING GOTO STATEMENT

The following is a sample program for solution of the quadratic equation using the GOTO statement. As an exercise, this program can be entered into the ND812 Computer via the Teletype keyboard in the following format. This printout is obtained by typing .W after the program is entered.

```
 5 ERASE
10 PRINT 'SOLUTION TO QUADRATIC EQUATION',/
12 FMT(F15,2)
15 PRINT 'ROOTS X1 AND X2 DEFINED AS FOLLOWS:'
20 PRINT '      X1 = (-B + SQRT(B*B - 4AC))/2A'
25 PRINT '      X2 = (-B - SQRT(B*B - 4AC))/2A'
30 PRINT /,'INPUT VALUES FOR A, B, AND C'; INPUT A,B,C
35 D=B*B-4*A*C; F=-B/(2*A)
40 IF(D+0)45,65,65
45 PRINT 'X1 = ',F,'+ IMAGINARY'
50 PRINT 'X2 = ',F,'- IMAGINARY'
55 PRINT 'END OF COMPUTATION'
60 GOTO80
65 G=SQRT(D)/(2*A)
70 R=F+G; S=F-G
75 PRINT 'X1 = ',R,/,'X2 = ',S,/,'END OF COMPUTATION'
80 CONTINUE
85 STOP
```

Now enter 5.G command and carriage return at the Teletype keyboard. The following printout shows the result for A = 2, B = -3, and C = 4.

```
>5.G
SOLUTION TO QUADRATIC EQUATION

ROOTS X1 AND X2 DEFINED AS FOLLOWS:
     X1 = (-B + SQRT(B*B - 4AC))/2A
     X2 = (-B - SQRT(B*B - 4AC))/2A

INPUT VALUES FOR A, B, AND C
:2
:-3
:4
X1 =              .75+ IMAGINARY
X2 =              .75- IMAGINARY
END OF COMPUTATION

>
```

In the following two examples, the quadratic equation program is used to calculate for E format entries. The first example uses E to indicate E format and the second uses D. E and D are interchangeable in E format entries as shown.

```
>5.G
SOLUTION TO QUADRATIC EQUATION

ROOTS X1 AND X2 DEFINED AS FOLLOWS:
     X1 = (-B + SQRT(B*B - 4AC))/2A
     X2 = (-B - SQRT(B*B - 4AC))/2A

INPUT VALUES FOR A, B, AND C
:2E-6
:4E3
:5E2
X1 =            -64.00
X2 =  -1999999809.26
END OF COMPUTATION

>5.G
SOLUTION TO QUADRATIC EQUATION

ROOTS X1 AND X2 DEFINED AS FOLLOWS:
     X1 = (-B + SQRT(B*B - 4AC))/2A
     X2 = (-B - SQRT(B*B - 4AC))/2A

INPUT VALUES FOR A, B, AND C
:2D-6
:4D3
:5D2
X1 =            -64.00
X2 =  -1999999809.26
END OF COMPUTATION

>
```

5.9 DO STATEMENT

The DO statement is one of the most powerful features of the NUTRAN language, for through it, repeated initiation and control of the execution of a section of program (with changes in the value of a variable between repetitions) are possible. The DO statement allows the programmer to specify the necessary parameters for constructing a program loop. The general format of the DO statement is: DO label i = parameter 1, parameter 2, parameter 3, where the label and parameters are defined as follows.

1) The label is the statement number of the last or terminal statement in a group of statements to be repeated.

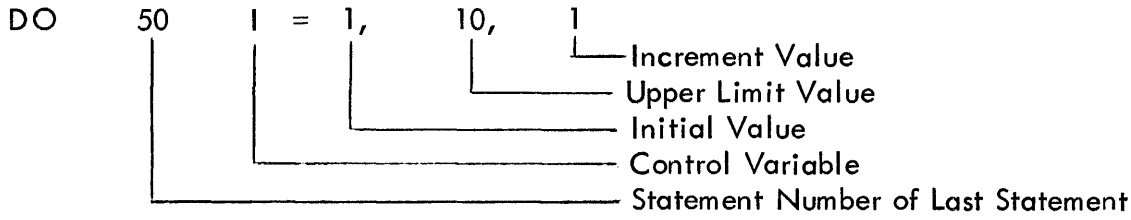
2) The "i" is a control variable which represents a loop counter assigned the initial value of parameter 1.

3) Parameter 1 is an initial value (any number, variable identifier, or arithmetic expression) for the loop counter.

4) Parameter 2 is an upper limit or maximum value (any number, variable identifier, or arithmetic expression) which is not exceeded by the loop counter.

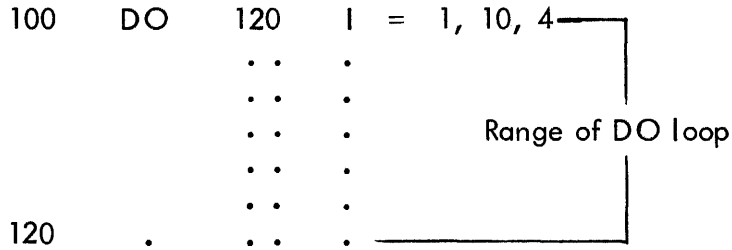
5) Parameter 3 is an increment value (any number, variable identifier, or arithmetic expression) by which the loop counter is to be modified after each execution. This parameter is optional. If it is not included, a value of one is assumed.

A typical DO statement is,



5.9.1 DO LOOPS

The DO statement is followed by a group of statements which are repeatedly executed until the value of the control variable exceeds the upper limit value. A DO loop is the group of statements (including the DO). The last statement in the DO loop must be a labeled, executable statement and must not be a control statement - STOP, IF, GOTO, CALL RETURN, or DO. An example DO loop is shown below.



The statement, DO 120 I = 1, 10, 4 specifies that the control variable representing the loop counter is initially set to 1. The upper limit of the loop counter is 10, and the quantity by which the loop counter is to be incremented after each execution is 4.

During the first execution of the loop, the control variable has an initial value of 1. After each execution, the control variable is incremented by 4. During the second execution, the control variable has a value of 5, and during the third execution it has a value of 9. A fourth incrementation causes the contents of the loop counter to exceed the designated upper limit value of 10. Therefore, the DO loop is terminated after the third execution.

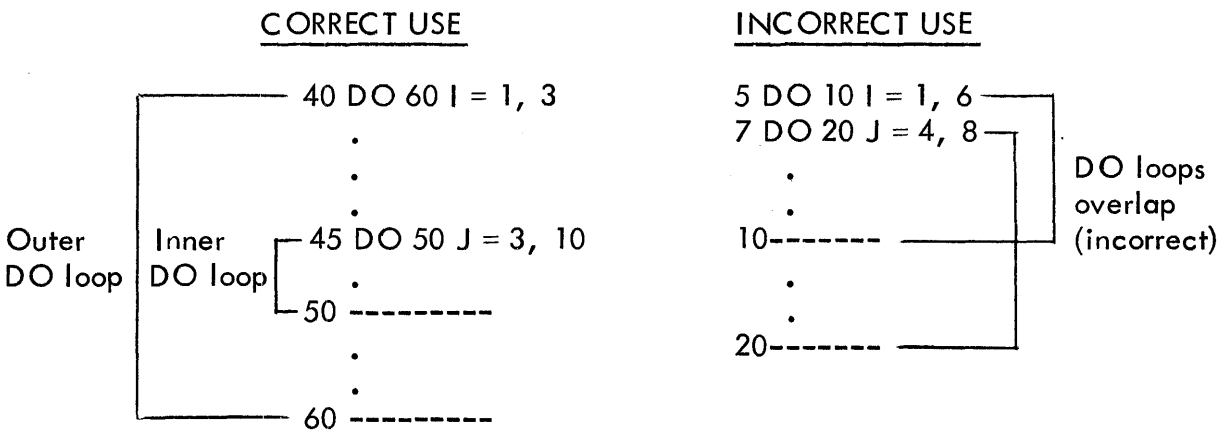
The terminal statement in the range of a DO loop must not be GOTO, CALL, RETURN, IF, DO or STOP statements. However, if it is necessary to use one of these (except STOP) as the last statement in a DO loop, the statement should be followed by CONTINUE.

5.9.2 NESTED DO LOOPS

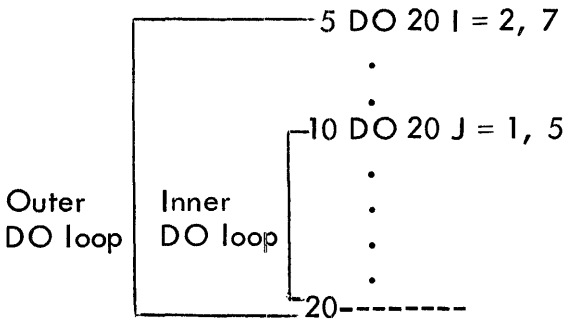
The rules for writing single DO loops have been described. However, it is quite useful to write DO loops which contain other DO loops. DO loops may be nested within the range of other DO loops if prescribed rules are followed.

1) All statements in the range of the inner DO must be completely contained within the range of the outer DO.

2) Control variables and parameters of outer DO loops must not be altered by inner DO loops. For example,



3) Nested DO loops can specify a common terminal statement. For example,



4) It is not recommended that transfer of control by IF or GOTO statements be permitted into the range of any DO statement from outside its range. Such transfer might not allow the DO loop control variable to be properly initialized.

5.9.3 EXAMPLE PROGRAM USING DO STATEMENT

The following is a sample program for computing the sum of squares of five numbers using the DO statement rather than the IF statement given in paragraph 5.7.1. As an exercise, this program may be entered into the ND812 Computer via Teletype keyboard as shown.

The printout is obtained by typing .W and depressing carriage return after the program is entered.

```
500 ERASE
502 PRINT 'SUM OF SQUARES EXAMPLE'
503 PRINT 'COMPUTES SUM OF X(I) SQUARED, WHILE I GOES FROM 1 TO 5'
510 SUMSQR=0
515 FMT (F6,2)
520 DO 575 I=1,5,1
530 PRINT 'INPUT A VALUE FOR X';INPUT X
550 SUMSQR=SUMSQR+X**2
570 PRINT 'SUM OF SQUARES = ',SUMSQR
575 CONTINUE
580 PRINT 'END OF COMPUTATION'
590 STOP
```

Now enter 500.G command and depress carriage return at Teletype keyboard. The result using the indicated entries is as follows. In this program the sum of squares is typed after each entry until a total of 5 entries is made.

```
>500.G
SUM OF SQUARES EXAMPLE
COMPUTES SUM OF X(I) SQUARED, WHILE I GOES FROM 1 TO 5
INPUT A VALUE FOR X
:2
SUM OF SQUARES =   4.00
INPUT A VALUE FOR X
:8
SUM OF SQUARES =  68.00
INPUT A VALUE FOR X
:4
SUM OF SQUARES =  84.00
INPUT A VALUE FOR X
:5
SUM OF SQUARES = 109.00
INPUT A VALUE FOR X
:1
SUM OF SQUARES = 110.00
END OF COMPUTATION

>
```

5.9.3.1 Example Program With Error

The following is an example of the program given in paragraph 5.9.3 with an error in line 520.

```
500 ERASE
502 PRINT 'SUM OF SQUARES EXAMPLE'
503 PRINT 'COMPUTES SUM OF X(I) SQUARED, WHILE I GOES FROM 1 TO 5'
510 SUMSQR=0
515 FMT (F6,2)
520 DO I=1,5,1
530 PRINT 'INPUT A VALUE FOR X';INPUT X
550 SUMSQR=SUMSQR+X**2
570 PRINT 'SUM OF SQUARES = ',SUMSQR
575 CONTINUE
580 PRINT 'END OF COMPUTATION'
590 STOP
```

The program is then executed as shown below by typing 500.G and depressing carriage return. The program proceeds as normal until the error is encountered and prints the error message as shown.

```
>500.G
SUM OF SQUARES EXAMPLE
COMPUTES SUM OF X(I) SQUARED, WHILE I GOES FROM 1 TO 5

ER@    520
>
```

Line 520 is then rewritten as shown below to show the correct terminal statement for the DO loop.

```
>520 DO 575 I=1,5,1
```

The program is again executed by typing 500.G and depressing carriage return. Using the same entries (in reverse order) as in section 5.9.3, the result is as follows.

```
>500.G
SUM OF SQUARES EXAMPLE
COMPUTES SUM OF X(I) SQUARED, WHILE I GOES FROM 1 TO 5
INPUT A VALUE FOR X
:1
SUM OF SQUARES =   1.00
INPUT A VALUE FOR X
:5
SUM OF SQUARES =  26.00
INPUT A VALUE FOR X
:4
SUM OF SQUARES =  42.00
INPUT A VALUE FOR X
:8
SUM OF SQUARES = 106.00
INPUT A VALUE FOR X
:2
SUM OF SQUARES = 110.00
END OF COMPUTATION

>
```

5.9.4 EXAMPLE PROGRAM USING DO, GOTO, IF AND FMT STATEMENTS

The following program increments from the number 1 to the number 5 and computes the square for each of the incremented numbers. The incrementation/squaring procedure is performed three times with a different printout format specified each time as shown. Also, the contents of the symbol table is listed after each format change.

```
 10 FMT(F10,3)
 20 J=0
 30 DO 100 I=1,5
 40 A(I)=I**2
100 PRINT I,A(I)
110 LIST
120 J=J+1
130 IF(J-2)140,150,160
140 FMT(E10,3)
145 GOTO30
150 FMT(I10)
155 GOTO30
160 STOP
```

When the program is executed by typing 10.G and depressing carriage return, the following printout results.

```
>10.G
      1.000      1.000
      2.000      4.000
      3.000      9.000
      4.000     16.000
      5.000     25.000

A (    5)      25.000
A (    4)      16.000
A (    3)       9.000
A (    2)       4.000
A (    1)       1.000
> (    0)       2.000
A (    0)        .000
> (    0)       5.000
> (    0)       1.000
I (    0)       6.000
> (    0)       0.000
J (    0)       0.000
> (    0)       3.000
> (    0)      10.000
  .100E  1   .100E  1
  .200E  1   .400E  1
  .300E  1   .900E  1
  .400E  1   .160E  2
  .500E  1   .250E  2
```

```
> (    0)    .300E  2
> (    0)    .140E  3
A (    5)    .250E  2
A (    4)    .160E  2
A (    3)    .900E  1
A (    2)    .400E  1
A (    1)    .100E  1
> (    0)    .200E  1
A (    0)    .278E-08
> (    0)    .500E  1
> (    0)    .100E  1
I (    0)    .600E  1
> (    0)    .000E  1
J (    0)    .100E  1
> (    0)    .300E  1
> (    0)    .100E  2
          1          1
          2          4
          3          9
          4         16
          5         25

> (    0)        150
> (    0)         30
> (    0)        140
A (    5)         25
A (    4)         16
A (    3)          9
A (    2)          4
A (    1)          1
> (    0)          2
A (    0)          0
> (    0)          5
> (    0)          1
I (    0)          6
> (    0)          0
J (    0)          2
> (    0)          3
> (    0)         10

>
```

5.10 CALL, SUBROUTINE AND RETURN STATEMENTS

Transfer of control to a subroutine is made by a CALL statement, specifying the name of the subroutine. The general forms of the CALL statement are,

CALL name

CALL name (arg_1, arg_2, -----arg_n)

The first statement of a subroutine procedure must be the procedure definition statement SUBROUTINE, followed by the name of the subroutine and an argument list (if one is required). The general forms of the SUBROUTINE statement are,

SUBROUTINE name

SUBROUTINE name (arg_1, arg_2, -----arg_n)

The last executed statement in any subprogram is normally a RETURN statement which need not be at the physical end of the subroutine. More than one RETURN statement can appear in a subroutine. The RETURN statement causes return from the subroutine to the statement immediately following the CALL statement for that subroutine.

5.10.1 EXAMPLE PROGRAM USING CALL, SUBROUTINE, AND RETURN STATEMENTS

Refer to the example program solving for the quadratic equation in paragraph 5.8.1. In that example, any unreal (imaginary) solutions are detected, but ignored, and the statement "IMAGINARY" is typed to indicate that the numbers entered for A, B, and C have imaginary roots. However, the imaginary portions of the roots may be solved by using a subroutine program. The program in paragraph 5.8.1 may be rewritten as follows to include CALL and SUBROUTINE statements.

```
 10 PRINT 'SOLUTION TO QUADRATIC EQUATION',/
 12 FMT(F15,2)
 15 PRINT 'ROOTS X1 AND X2 DEFINED AS FOLLOWS:'
 20 PRINT '      X1 = (-B + SQRT(B*B - 4AC))/2A'
 25 PRINT '      X2 = (-B - SQRT(B*B - 4AC))/2A'
 30 PRINT /,'INPUT VALUES FOR A, B, AND C'; INPUT A,B,C
 35 D=B*B-4*A*C; F=-B/(2*A)
 40 IF(D+0)43,65,65
 43 CALL UNREAL(A,B,C)
 45 PRINT 'X1 = ',F,'+ IMAGINARY',E
 50 PRINT 'X2 = ',F,'- IMAGINARY',E
 55 PRINT 'END OF COMPUTATION'
 60 GOTO80
 65 G=SQRT(D)/(2*A)
 70 R=F+G; S=F-G
 75 PRINT 'X1 = ',R,/,'X2 = ',S,/,'END OF COMPUTATION'
 80 CONTINUE
 85 STOP
700 SUBROUTINE UNREAL(X,Y,Z)
710 N=-1*((Y*Y)-(4*X*Z))
720 E=SQRT(N)/(2*X)
730 RETURN
```

In this program, a negative square root detected by the IF statement in line 40 causes a branch to the "CALL UNREAL (A, B, C)" statement and the statement "SUBROUTINE UNREAL (X, Y, Z)" is immediately accessed. The subroutine program then calculates the value of the imaginary root and returns control to the line number following the CALL statement in the main program. The calculated answer is then printed and the program returns to text mode. The program is executed by typing 10.G and depressing carriage return as shown below. The same values for A, B, and C (2, -3 and 4) are then entered as were entered in the first example in paragraph 5.8.1.

```
>10.G
SOLUTION TO QUADRATIC EQUATION

ROOTS X1 AND X2 DEFINED AS FOLLOWS:
     X1 = (-B + SQRT(B*B - 4AC))/2A
     X2 = (-B - SQRT(B*B - 4AC))/2A

INPUT VALUES FOR A, B, AND C
:2
:-3
:4
X1 =              .75+ IMAGINARY          1.19
X2 =              .75- IMAGINARY          1.19
END OF COMPUTATION

>
```

If values for A, B, and C are entered that result in real roots, the program stays in the main routine and calculates the roots. For example, the values for A, B, and C entered in the second example of paragraph 5.8.1 (2E-6, 4E3, and 5E2) are entered into this program as shown below. The program is executed by typing 10.G and depressing carriage return as shown.

```
>10.G
SOLUTION TO QUADRATIC EQUATION

ROOTS X1 AND X2 DEFINED AS FOLLOWS:
     X1 = (-B + SQRT(B*B - 4AC))/2A
     X2 = (-B - SQRT(B*B - 4AC))/2A

INPUT VALUES FOR A, B, AND C
:2E-6
:4E3
:5E2
X1 =           -64.00
X2 =  -1999999809.26
END OF COMPUTATION

>
```

5.11 FISET, PUT, AND GET STATEMENTS

If more than the required 8K of memory is available, the remainder over 8K (Memory Fields 2 and larger) may contain data rather than specific program information. Each word of data in these memory locations may be stored and retrieved via the FISET, PUT, and GET statements.

Data storage may be in either one-word or two-word form. If storage is in one-word form, each word of data is stored (most significant bit first) in one memory location. For two-word data storage, the least significant word (first word) of data is stored in one memory location, and the most significant word (second word) is stored in the next successive location. The most significant bit of each word still appears first in each word.

The FISET statement is used to specify one-word or two-word formats and is of the general form FISET (n), where "n" may be either 1 or 2, specifying one- or two-word format respectively. If FISET statement is not used, default value is 1.

The PUT statement specifies the storage location for the one or two words of data as specified by the FISET statement. The PUT statement is of the general form PUT (A,X), where "A" is the expression to be stored, and "X" specifies storage position in Memory Field 2. The "X" may be an expression that is evaluated in the program or may be a specific number. In either case, the actual storage location for the data word (or for the first word of two-word data) is the position specified by "X" minus 1 in Memory Field 2. For example, if "X" is 1, storage is in location Ø of Memory Field 2. (If two-word data is specified, the first word is stored in location Ø and the second in location 1 of Memory Field 2.) Positions greater than 4096 specify locations in Memory Field 3. For example, position 4100 specifies location 4 in Memory Field 3.

The GET function facilitates retrieval of the data in Memory Field 2 and is of the general form GET (x), where "x" specifies a position in Memory Field 2. The "x" in the GET statement follows the same rules as the "x" in the PUT statement as described above.

5.11.1 EXAMPLE PROGRAM USING FISET, PUT, AND GET STATEMENTS

The following program shows a simple PUT and GET program. The value "5" is first stored in position 1 (location Ø of Memory Field 2). The contents are then retrieved from position 1 and printed.

```
1 PUT(5,1)
2 PRINT GET(1)
3 STOP
```

The program is executed by typing 1.G with the following result.

```
>1.G
   .5000000E  1

>
```

If desired, FISET may be used to specify one- or two-word storage as described in paragraph 5.11. FISET may be inserted in the above program as follows to specify two-word storage. (The second word, however, in this example would be unused.) Execution of the program results in the same printout as above.

```
1 FISET(2)
2 PUT(5,1)
3 PRINT GET(1)
4 STOP
```

SECTION VI
CALCULATOR MODE

6.1 GENERAL

The calculator mode allows separate execution of a statement outside a program, without transferring control to the program. A space precedes the statement, rather than a statement line number. The calculator mode may be used to perform the following.

a. Spot check values of variables during the execution of a program. Use a STOP statement in the program and then input:

PRINT V_1, V_2, V_3

The values of the specified variables are printed without changing the value or destroying program execution sequence.

b. Reassign values for variables. For example,

A = 7.6

assigns a new value, 7.6, for variable A.

c. Quick computations. The following printouts show example calculations using calculator mode. No line number is needed as shown and after computation is made and the required printout is accomplished, NUTRAN automatically returns to text mode without deleting or modifying any text buffer program.

```
> PRINT 2+45
    .470000E  2

> PRINT SQRT(ABS(-49))
    .700000E  1

>
```

d. Erase symbol table only. For example,

```
E
```

erases the contents of the symbol table. (Using .E also erases text or specified portions of text as well as the symbol table.)

e. List symbol table. For example,

```
L
```

causes the contents of the symbol table to be listed on Teletype.

APPENDIX A
LOADING AND INITIALIZATION

In order to communicate with the ND812 Computer in NUTRAN conversational language, the NUTRAN Interpreter program (ND41-0059) must be loaded into the ND812 Computer. The program may be loaded via Teletype or optional High Speed Reader or Tape Cassette. The following paragraphs provide loading and initializing procedures for the NUTRAN Interpreter using the above peripheral devices.

LOADING AND INITIALIZATION USING TELETYPE

The following procedures describe loading and initializing for the NUTRAN Interpreter via Teletype. It is assumed that the ND812 Computer and the Teletype are turned on and operating properly before the procedures are perfomed.

a. Depress ND812 STOP switch.

b. Set Teletype START/FREE/STOP switch to FREE.

c. Load NUTRAN Interpreter Program (41-0059) tape, Part I into Teletype reader. Advance the tape so that the leader (level 8 punched) is over the read head.

d. Set Teletype START/FREE/STOP switch to START.

e. Simultaneously depress ND812 LOAD AR and NEXT WORD switches. The tape is automatically fed through the Teletype reader. When tape motion stops, set ND812 SELECT REGISTER switch to J. All ND812 front-panel SELECTED REGISTER lamps should be off. If any lamps are on, repeat steps a through e.

f. If all lamps are off, set Teletype START/FREE/STOP switch to FREE and remove tape. Load NUTRAN Interpreter Tape, Part II, as described in step c.

g. Repeat step d.

h. Repeat step e. If any lamps are on, repeat steps f through h.

i. If all lamps are off, set ND812 SWITCH REGISTER switches to $\emptyset 1\emptyset\emptyset_8$ (switch 5 up, all others down). Then, in sequence, depress ND812 LOAD AR and START switches. The Teletype responds by typing >. Any valid command described in Section IV, V, or VI may be entered. If > is not typed, repeat all steps.

LOADING AND INITIALIZATION USING HIGH SPEED READER OR TAPE CASSETTE

The NUTRAN Interpreter Program (ND41-0059) is loaded via High Speed Reader or Tape Cassette using the Binary Loader Program (ND41-0005). The Binary Loader Program is loaded via Teletype. It is assumed that the ND812 Computer, the Teletype, and the high-speed reader (or cassette unit) are turned on and operating properly before the following procedures are performed.

a. Depress ND812 STOP switch.

b. Set ND812 MEMORY FIELD switch $\emptyset$ down and 1 up.

c. Set Teletype START/FREE/STOP switch to FREE.

d. Load Binary Loader Program (41-0005) tape into Teletype reader. Advance the tape so that the leader (level 8 punched) is over the read head.

e. Set Teletype START/FREE/STOP switch to START.

f. Simultaneously depress ND812 LOAD AR and NEXT WORD switches. The tape is automatically fed through the Teletype reader. When tape motion stops, set ND812 SELECT REGISTER switch to J. All ND812 front-panel SELECTED REGISTER lamps should be off. If any lamps are on, repeat steps a through e.

g. If all lamps are off, set Teletype START/FREE/STOP switch to FREE and remove tape.

h. Set ND812 SWITCH REGISTER switches to $77\emptyset\emptyset_8$ (switches $\emptyset$ - 5 up, 6 - 11 down).

i. Depress ND812 LOAD AR key.

j. Perform one of the two following procedures for either high-speed reader or tape cassette loading.

1. HIGH-SPEED READER

a) Load NUTRAN Interpreter Program (41-0059) Tape, Part I into

high-speed reader. Be sure leader (level 8 punched) is over the read head.

b) Set ND812 SWITCH REGISTER switches to $37ØØ_8$ (switches 1 - 5 up, Ø and 6 - 11 down).

c) Depress ND812 START switch. The tape is automatically fed through the high-speed reader. When tape motion stops, set ND812 SELECT REGISTER switch to K. All ND812 front-panel SELECTED REGISTER lamps should be off. If any lamps are on, repeat steps a) through c).

d) If all lamps are off, load NUTRAN Interpreter Tape, Part II, into high-speed reader as described in step a).

e) Depress ND812 CONT or START switch. After tape motion stops, all SELECTED REGISTER lamps should be off. If any lamps are on, repeat steps d) and e).

f) If all lamps are off, proceed to step k.

2. TAPE CASSETTE

a) Set SWITCH REGISTER switches Ø and 1 down, and 2, 3, and 4 to the desired cassette number as follows.

Cassette 1: 2 and 3 down, 4 up (ØØ1)
Cassette 2: 2 and 4 down, 3 up (Ø1Ø)
Cassette 3: 3 and 4 down, 2 up (1ØØ)

b) Select tagword by setting SWITCH REGISTER switches 5 through 11 to desired number ($ØØØØ_8$ through $Ø177_8$).

c) Depress ND812 START switch. The tape is automatically read into the ND812. When tape stops, set ND812 SELECT REGISTER switch to K. All ND812 front-panel SELECTED REGISTER lamps should be off. If any lamps are on, repeat steps a) through c).

d) If all lamps are off, proceed to step k.

k. Set ND812 SWITCH REGISTER switches to $Ø1ØØ_8$ (switch 5 up, all others down) and ND812 MEMORY FIELD switches Ø and 1 down. Then, in sequence, depress ND812 LOAD AR and START switches. The Teletype responds by typing > . Any valid command described in Section IV, V, of VI may be entered. If > is not typed, repeat all steps.

APPENDIX B
LOCATING NUTRAN IN MEMORY FIELDS OTHER THAN Φ OR 1

If the user has an ND812 Computer with a memory size greater than 8K, it may be desirable to make use of more than two memory fields for NUTRAN. Two methods exist to accomplish this. In the first method, after loading NUTRAN, Parts 1 and 2, load the appropriate overlay to modify NUTRAN to end in Memory Field 2 (NUTRAN, Part 3) or Memory Field 3 (NUTRAN, Part 4). (Loading procedures are identical to those for Parts 1 and 2 shown in Appendix A.)

In the second method, a reassembly is necessary, requiring only one change in the first block of the source. The statement

```
FLFLD = F1  /END OF PROCESSOR MEMORY FIELD
```

should be changed to equate FLFLD to the last field desired (F2 or F3).

Also, if it is necessary to locate NUTRAN in two fields other than Memory Field zero (FØ = Ø - 4K) and Memory Field 1 (F1 = 4K - 8K), the two directives preceding the above FLFLD directive, namely,

```
FØFLD = FØ
F1FLD = F1
```

should be changed to the new fields as desired.

APPENDIX C
ERROR DIAGNOSTICS

During execution of a NUTRAN program, two possible errors may occur - logical or syntactical. A logical error may be an overflow of a buffer. A syntax error may be an illegal command, improper coding, or improper placement of operators in an arithmetic expression. When any of these errors are encountered during execution of NUTRAN, the Teletype prints the following.

ER@ line number

The line number indicates where the error occured. Verification of the listing at that point (using .W command for that line number) should be sufficient to detect syntax errors. A dump of the data tables (using .L command) should be useful in detecting logic errors.

NUTRAN returns to Text mode after the error message is typed. Ordinarily, after correcting the error, the user must restart his program from the beginning (using .G command) since the return to Text mode erases DO loop buffers and subroutine argument buffers.

APPENDIX D
USER WRITTEN SUBPROGRAM FUNCTIONS

ADDING FUNCTIONS TO NUTRAN

Functions coded by the user can be a powerful tool in using the NUTRAN Interpreter. These "user written" functions can be named any six-character name, with the first character being alphabetic and the rest alphanumeric, and can be used in the same manner as the "intrinsic" functions SQRT, ABS, etc. For example, a function MAX (A, B, C, D, E) which finds the maximum of the five arguments can be coded and used in an expression such as,

```
A = 41.63*MAX(3.7,C,100,X,Y)
```

To append a function to NUTRAN, two tables in the NUTRAN program must be modified. These are the operator table, labeled FTBB, and the function table, labeled FNTB. Table FTBB contains the instructions to transfer control to each function. Any new two-word instruction must then be added to the end of the table as shown in the following example. The name of the function (in ND-packed characters, 2 characters per word) is added at the end of table FNTB. The last word of table FNTB must have the label ENDP as shown below.

To facilitate adding functions to NUTRAN, only the revised tables, appropriate pointers, and the new function programs have to be assembled as shown. The binary is then an overlay to the regular NUTRAN, Parts 1 and 2.

RETRIEVING ARGUMENTS FOR FUNCTIONS

Function arguments are retrieved from the operand stack reversed from the order in which they are listed in the source. Therefore in the example MAX(A, B, C, D, E), the arguments will be returned as E, D, C, B, A. The routine in NUTRAN used to get arguments is called MOVE and requires one argument as shown below. This argument is the address of a two-word load or store instruction that points to the first word where the argument is to go. In the example, the first argument off the list (i.e., the last argument on the list) will be placed in a five-word field starting at location DATARA in field F1FLD. The general format for the five-word argument is as follows.

WORD	CONTENT
0	Variable Name
1	Subscript
2	Floating Point Value
3	
4	

Therefore, if the argument is to be used for its data value only, the recipient address+2 (or in the example, DATARA+2) will point to the data itself.

FLOATING POINT ROUTINES IN USER WRITTEN FUNCTIONS

When using floating point routines in user-written functions, the left operand address for binary operators (plus, minus, multiply, and divide) is loaded into the K register, and the right operand address into the J register. For unary operators, the operand address is loaded into the J register. For either the binary or unary operation, the result of the operation is then stored in the right operand address (address contained in the J register).

In order to perform floating point operations, the operands must first be transferred to either location EXPFK+2 or EXP1A+2 in Memory Field Ø. These locations are the first word of the three-word floating point value included in the five-word argument shown above.

PLACING RESULTS ONTO OPERAND STACK

At the conclusion of a function operation, the resultant operation must be placed on the operand stack. (There is no legal NUTRAN statement which does not expect the result on the stack.) The NUTRAN routine that performs this task is called REPL. REPL takes a two-word argument, namely a two-word load (or store), including field control bits, which points to the five words to be moved. (As shown in the example, these 5 words include variable identifier and subscript, as well as value.)

The example also illustrates why the pointer to a two-word pointer was used in subroutine MOVE. Since NUTRAN is multifield, field bits must be included in any address. The conjunctive use of MOVE and REPL with their associated arguments, can thus save time and space in a program. Also, at the conclusion of a function operation and after placement of the operand onto operand stack using REPL, control must be returned to the polish stack processor as shown.

EXAMPLE USER-WRITTEN SOURCE LISTING

In order to properly use a user-written function, certain locations in the basic NUTRAN program must be known. The statements for these locations are listed in the example and the associated locations are provided with each NUTRAN tape.

```
FØFLD = FØ
F1FLD = F1

/ EQUATE STATEMENTS - LOCATIONS PROVIDED WITH NUTRAN TAPE
/ LOCATIONS OF FLOATING POINT ROUTINES ARE ALSO PROVIDED

        FTBB    =
        MOVE    =
        REPL    =
        EAPS1   =
        PNTB    =
        EFTB    =
        BTBF    =
        ETBF    =
        EXPFK   =
        EXP1A   =

 [FIELD 1               /FIELD DESIGNATED BY F1FLD
* FTBB + 46
        TWJMP
        MAX

FNTB,   6361            / "SQUARE ROOT"
        6264
        ØØØØ
        4142            /"ABSOLUTE VALUE"
        63ØØ
        ØØØØ
        457Ø            /"EXP"
        6ØØØ
        ØØØØ
        4154            /"ALOG"
        5747
        ØØØØ
        4154            /"ALOG 1Ø"
        5747
        2120
        4745            /"GET"
        64ØØ
        ØØØØ
        5541            /"MAX"
        7ØØØ
ENDP,   ØØØØ

MAX,    TWJPS FØFLD         /REMOVE LAST ARGUMENT IN LIST
        MOVE
```

```
          POINT I
             .
             .
             .
             .
          TWJPS FØFLD        /PLACE RESULT ON STACK
          REPL
          TWLDJ F1FLD
 POINT I, DATARA
          TWJMP FØFLD        /RETURN TO NUTRAN POLISH STACK PROCESSOR
          EAPS1
DATARA,     Ø                /VARIABLE NAME
            Ø                /SUBSCRIPT
            Ø                /SIGN AND EXPONENT WORD
            Ø                /HIGH ORDER WORD
            Ø                /LOW ORDER WORD

  TBFB = .                   /START OF TEXT BUFFER; ALL ASSEMBLER
                             LANGUAGE CODING MUST PRECEDE TBFB

  [FIELD Ø                   /FIELD DESIGNATED BY FØFLD

 *PNTB
  TWLDJ  F1FLD
  FNTB
 *EFTB
  TWLDK  F1FLD
  ENDP + 1
 *BTBF
  TWLDJ  F1FLD
  TBFB - 1
 *ETBF
  TWLDS  F1FLD
  TBFB
```

NUCLEAR DATA INC.

Nuclear Data Inc.
P. O. Box 451
100 West Golf Road
Palatine, Illinois 60067
Tel: (312) 529-4600

Nuclear Data Inc.
103 Pincushion Road
Framingham, Massachusetts 01701
Tel: (617) 899-4927

Nuclear Data Inc.
P. O. Box 2192
14278 Wicks Boulevard
San Leandro, California 94577
Tel: (415) 483-9200

Nuclear Data Inc.
2335 Brannen Road, S.E.
Atlanta, Georgia 30316
Tel: (404) 241-3220

Nuclear Data, GmbH
Mainzerlandstrasse 29
6 Frankfurt/M, Germany
Tel: 23 11 44

Nuclear Data Inc. (U.K.)
Rose Industrial Estate
Cores End Road
Bourne End, Bucks., England
Tel: 22733

Nuclear Data (Ireland) Ltd.
Kinsale Road, Ballycurreen
P. O. Box #23
Cork, Ireland
Tel: 22137

Nuclear Data (Scandinavia)
Division of Selektronik A/S
Hammervej 3
2970 Hørsholm, Denmark
Tel: (01) 86 30 00