File No. 1620-25
Form C26-5594-1

IBM Systems Reference Library

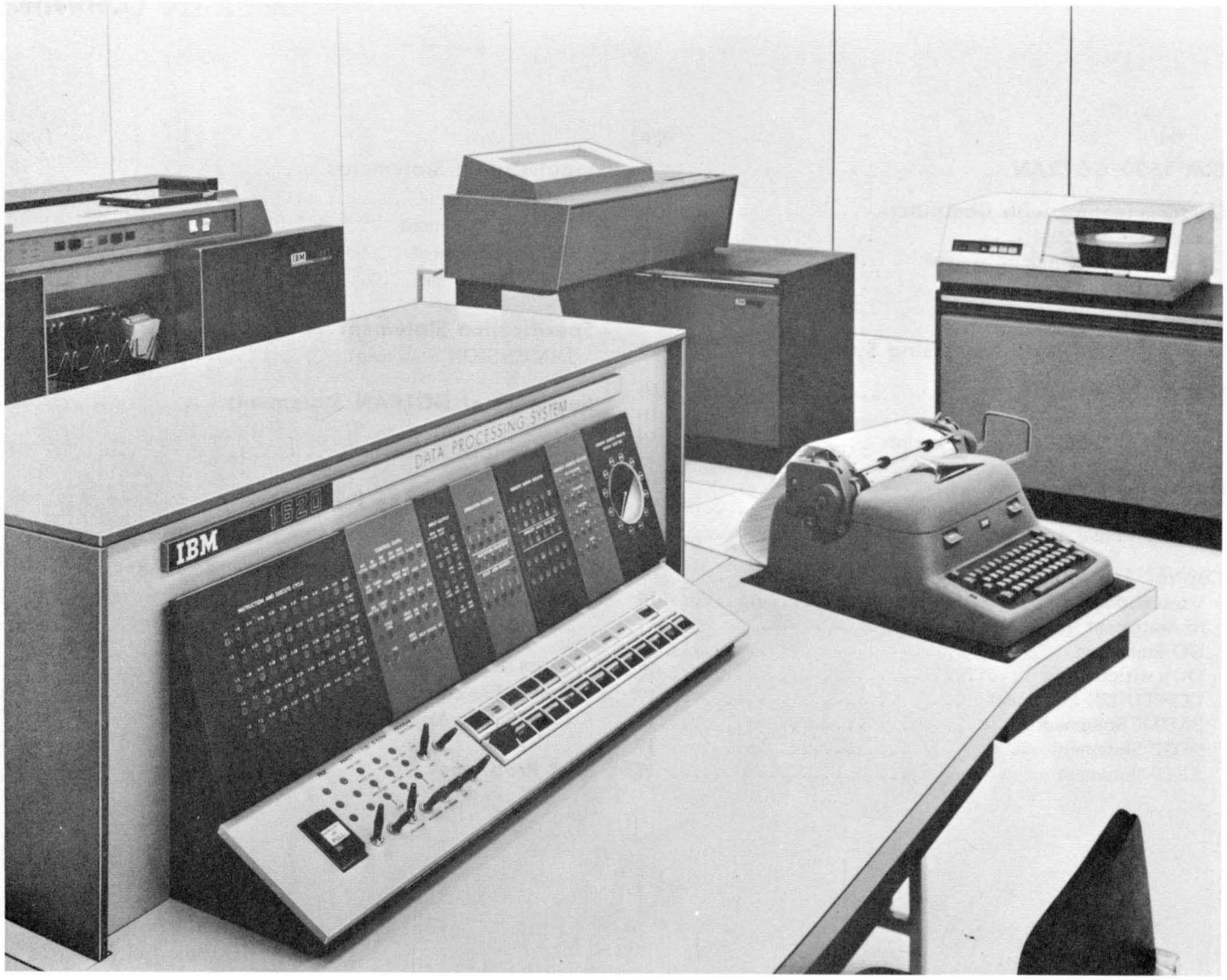
IBM 1620 GOTRAN Interpretive Programming System

This reference manual explains the GOTRAN system, a subset of 1620 FORTRAN language. The text progresses step by step from basic fundamentals to complete sample problems that facilitate the writing of GOTRAN programs. Some familiarity with the 1620 computer, mathematics, and data processing is assumed.

This manual is simply a reprint of the GOTRAN S Reference Manual, same title and same form number. Information from Technical Newsletter N26-0004 (dated 12-28-61) is included.

Copies of this and other IBM publications can be obtained through IBM Branch Offices. Comments concerning the contents of this publication may be addressed to: IBM, Product Publications Department, San Jose, California.

Contents



IBM 1620 Data Processing System

IBM 1620 GOTRAN

Electronic computers are already being widely used in the fields of science and engineering and their use is growing rapidly. Because the application of these machines to problems requiring precise and accurate calculations in large volume relieves the engineer of a tremendous burden of paper work, the digital computer has become firmly established as one of his important tools.

There exist, however, many problems for whose solution the computer is not being utilized. In these instances, it is usually because the problem is not considered sufficiently complex or voluminous to warrant the application of a computer, or because the engineer is not thoroughly familiar with computing techniques and is reluctant to program his problem for the machine. In short, the engineer finds communication with the computer a barrier.

IBM has developed a programming system for the IBM 1620 Data Processing System, that enables the engineer to surmount the communication barrier between man and computer with a minimum of effort. This system, which is known as the 1620 GOTRAN, uses a language that is a subset of the 1620 FORTRAN language. It executes programs directly, that is, without the need to convert GOTRAN statements into machine language instructions first. For this reason, the user is relieved of any consideration of the actual characteristics of the computer and thus is permitted to concentrate on the problem itself.

This manual gives the specifications of the GOTRAN system, to allow those who have had little computer background to learn the language and to use the system profitably. A section containing a general description of programming and a brief description of the 1620 is included.

Communication with Computers

Computers, despite their well-deserved reputation for speed and power of calculation, cannot think nor solve any and all problems by themselves. To solve a particular problem, the computer must be supplied with a step-by-step plan of operation. This plan is called a *program.*

Machine Language

All computers can perform to some degree such operations as add, multiply, compare, etc. Many of the operations it performs are quite elementary; however, it is the combination of a large number of elementary operations arranged in logical sequence that makes it possible for the computer to carry out meaningful processing tasks. This sequence of operations constitutes the program.

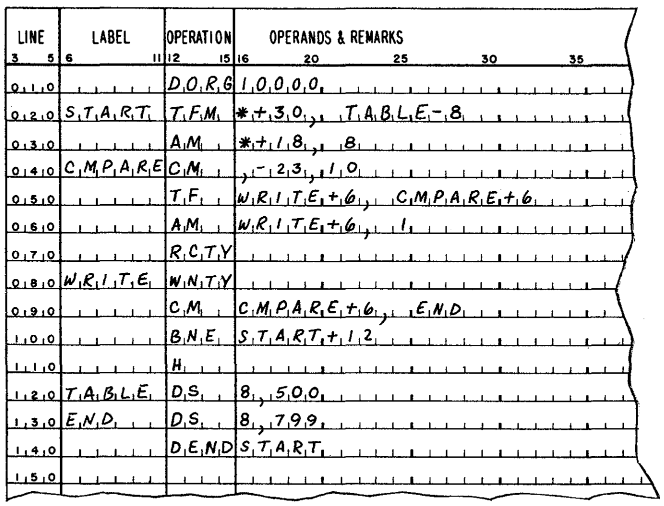
Each operation defines a precise function that the machine is to perform; therefore, the procedure must be detailed to a degree not required by manual methods. For example, in assigning a problem to someone, one might instruct him to "solve this equation." To accomplish this with a computer, it is necessary to define each single step that it must perform to solve the equation and then write the program accordingly.

After the computer is loaded with the necessary program, it can be directed to execute the instructions composing the program. Normally, the program is executed in a sequential manner. The computer starts with the first instruction and progresses serially through the program, analyzing and executing each instruction. The sequence of operations can be altered if desired, by the use of instructions that direct the computer to an instruction located at other than the next sequential position. The last instruction of a program may be used to return control to the first instruction, and to begin the sequence anew for a new set of data.

In addition to writing the program precisely, another requirement is that it be written in code. Whereas instructions to a human might be given in ordinary language, as:

Multiply A by B
Divide the product by C
Compare the result with D
Etc.,

the computer must be addressed in a much more concise numerical language. The instruction "Multiply A by B" must be presented to the computer in a form such as 23 08125 16501 called *machine language* in which the number represents the information needed to perform the operation. Expressing problems in terms of the code of the computer (machine language) can be a deterrent to the user.



LINE	LABEL	OPERATION	OPERANDS & REMARKS
3 5	6 11	12 15	16 20 25 30 35
010		DORG	10000
020	START	TFM	*+30, TABLE-8
030		AM	*+18, 8
040	CMPARE	CM	, -23, 10
050		TF	WRITE+6, CMPARE+6
060		AM	WRITE+6, 1
070		RCTY	
080	WRITE	WNTY	
090		CM	CMPARE+6, END
100		BNE	START+12
110		H	
120	TABLE	DS	8, 500
130	END	DS	8, 799
140		DEND	START
150			

```
10000
10000    16    10030  0̄0492
10012    11    10030  0̄0008
10024    14    00000  0002̄3̄
10036    26    10078  10030
10048    11    10078  0̄0001
10060    34    00000  00102
10072    38    00000  00100
10084    14    10030  0̄0799
10096    47    10012  01200
10108    48    00000  00000
00500          00008
00799          00008
10000
```

Figure 1. Symbolic Program and Resulting Machine Language Program

Symbolic Programming Systems

To allow programmers to express their problems in a language more easily comprehensible than machine language, a considerable effort has been made to develop Symbolic Programming Systems. These programming systems employ symbols, mnemonic operation codes, and English words or abbreviations in developing the program. Associated with the programming system is a machine language program, called a *processor*. The processor accepts the program written in symbolic language which is termed the *source program*, and converts it to a machine language, or *object*, program that can be loaded into the machine and executed to accomplish the assigned task. Figure 1 illustrates a short program written in 1620 symbolic language and the resulting machine language program.

FORTRAN

Although symbolic programming systems greatly simplify programming, they are essentially machine-oriented; that is, they require the programmer to think in terms of the command structure of the particular machine. With the advent of compilers such as FORTRAN, a highly significant step was taken in the development of programming systems. FORTRAN made it possible, for the first time, for coding systems to be designed in terms of the language of the problem to be coded, instead of in terms of a specific data processing system. The FORTRAN language is now used to formulate mathematical problems for several data processing systems including the 1620. Source programs written in FORTRAN language (a language that resembles the ordinary language of mathematics) must be converted into a machine language program before they become useful. The program that accomplishes this translation is called a FORTRAN *compiler*.

The programming systems described in the preceding paragraphs enable a programmer to write a program in a language more familiar to him than machine language. They are converted into machine language by a processor program, and the output of the processor program, which is the object program, is loaded back into the machine to perform useful work. The machine language program can be loaded into the machine at the discretion of the user, and can be re-executed at any future time simply by reloading it. This technique is particularly effective for large programs that must be used time and time again with varying data and for which it is most economical to process the source program just once.

IBM GOTRAN

Many problems require only one run or possibly a few runs on the computer, after which they are supplemented by a different problem that requires not only new data but a new program. Here, it is desirable to find a means to solve these, singly or in batches, without the need to create a different object program each time (perhaps to gather dust after its first use). This would enable the powers of the computer to be effectively harnessed to a host of new problems still being attacked by slower, less efficient methods.

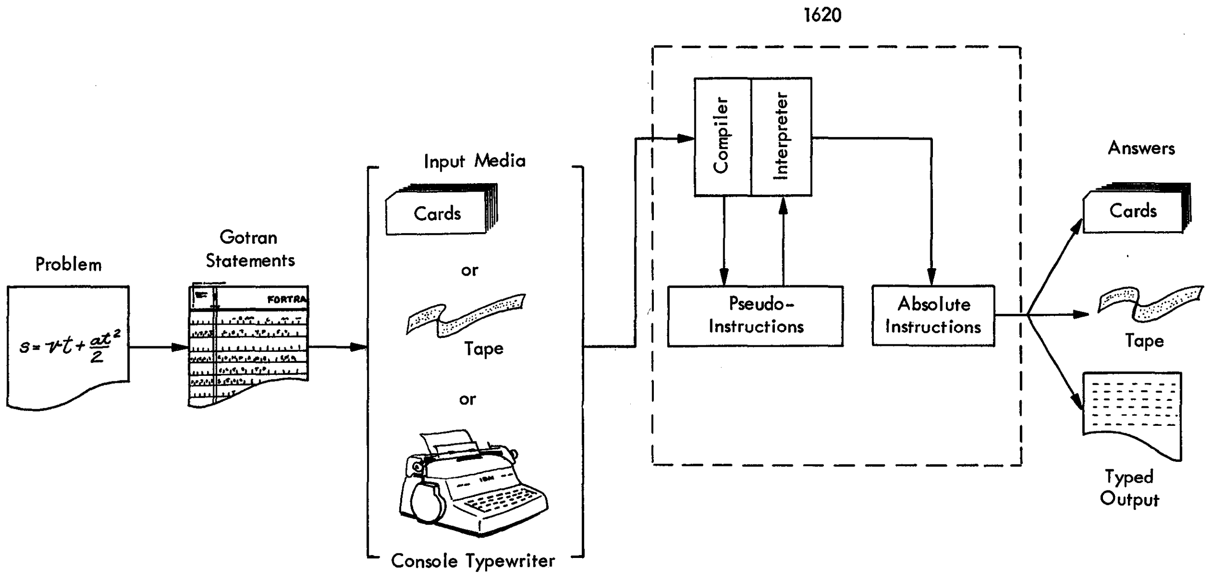


Figure 2. Structure of the GOTRAN System

The IBM 1620 GOTRAN System has been developed to fill this need. It compiles the problem-oriented source statements into pseudo-instructions within the machine; the computer then interprets and executes them without recourse to the conventional machine language output program. Such a system is called an *interpretive* system and the pseudo-instructions (which the programmer never sees) are referred to as *interpretive language.*

Figure 2 illustrates the basic structure of the GOTRAN System.

One great advantage of the GOTRAN System is that the compiler, once loaded, remains in the machine; it is effectively a part of the computer until removed. Therefore any number of different problems may be presented to the computer, singly or in batches, with solutions forthcoming at once. The time-consuming tedium of loading the system, creating the object deck or tape, loading the object program, etc., for each job, is eliminated. All that the user need be concerned with is the GOTRAN language because conversion to the interpretive language and subsequent execution of the instructions is entirely automatic. There is, naturally, a price exacted for this: that price is a somewhat reduced speed of operation as well as sacrifice of the portion of core storage that must be occupied by the compiler. The user must decide, on the basis of the individual job, whether GOTRAN is the system which is best for him.

The 1620 GOTRAN System language falls within the framework of the IBM 1620 FORTRAN System.*

*GOTRAN derives its name from this relationship as well as from the fact that the source program is executed in a load-and-go fashion.

The IBM 1620 Data Processing System

Any data processing system may be subdivided functionally into five parts (Figure 3).

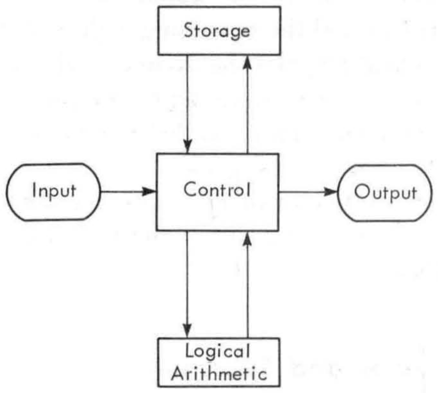


Figure 3. Functional Parts of a Computer

The Input Section

The *input* section accepts external information and converts it into a form in which it can be manipulated and stored internally. In the 1620, input information may be inserted by means of punched cards, punched paper tape, or the console typewriter. Figure 4 shows an input/output component of the 1620, the IBM 1622 Card Read Punch.

Storage Section

The *storage* section of a computer contains both the coded instructions that specify the actions to be performed (the program) and the data that is to be manipulated (added, multiplied, compared, etc.) by the instructions.

The 1620 utilizes magnetic core storage; its capacity ranges from 20,000 to 60,000 digits.

Logical-Arithmetic Section

The *logical-arithmetic* section is the place where data manipulation is actually performed. The *control* section decodes and interprets the instructions in storage and monitors the progression of the program through its planned sequence.

Output Section

The *output* section translates the internal information to some convenient external form. In the 1620, this external form may be punched cards, punched paper tape, or a printout at the console typewriter (Figure 5).

Although the speeds of the input/output devices are very rapid when compared to manual methods, they are still generally quite slow in comparison with the internal speeds of data handling.

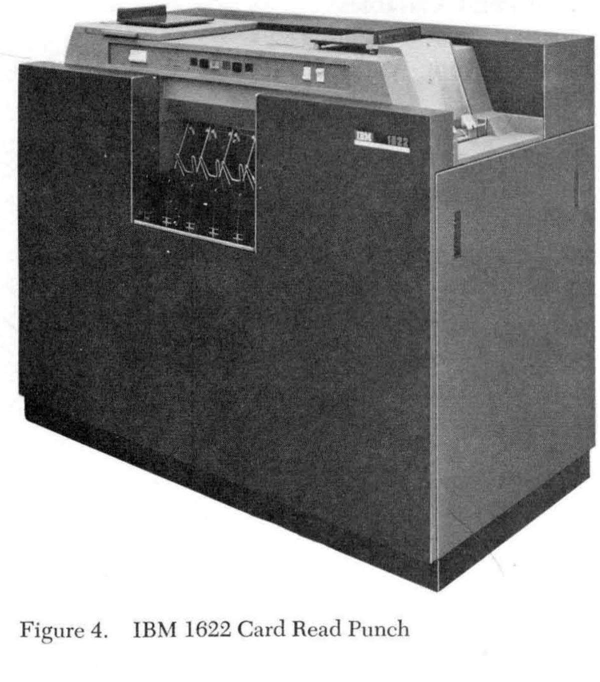


Figure 4. IBM 1622 Card Read Punch

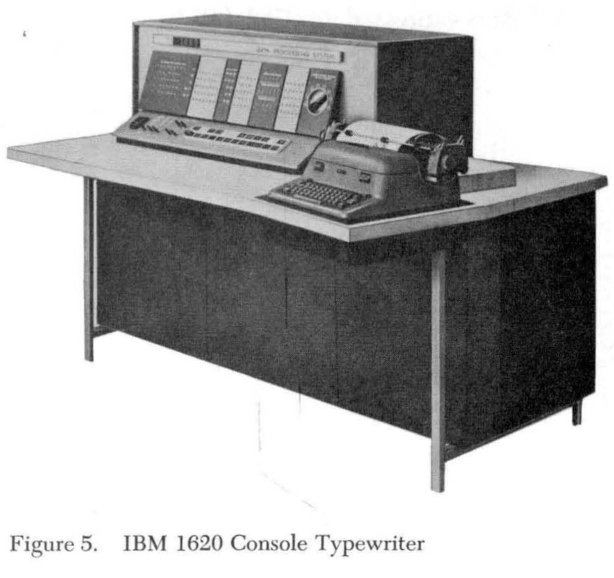


Figure 5. IBM 1620 Console Typewriter

The GOTRAN Language

The GOTRAN language consists of 13 different kinds of statements which may be grouped into four classifications: arithmetic statements, control statements, input/output statements, and a specification statement.

Before explaining the GOTRAN language, there are some other considerations bearing on the statements that need to be explored. These are: (1) Mode of arithmetic and (2) Kinds of numbers, i.e., constants, variables, and subscripted variables.

Mode of Arithmetic

Generally speaking, arithmetic in computers is done in either a *fixed-point* or *floating-point* mode. These two terms indicate the form in which the numbers are expressed. Numbers expressed as integers are considered fixed point. Thus, the integers 3, 52, and 724 are fixed-point numbers.

Floating-point arithmetic is a technique used to eliminate the complex programming required for correct placement of a decimal point in arithmetic operations. This complex programming results from the fact that computers do not generally recognize the decimal point present in any quantity during calculations, and therefore cannot recognize the magnitude of the result. Floating point is a method in which information regarding the magnitude of all numbers accompanies the quantities in the calculation. This is accomplished by expressing all numbers as decimal fractions times a power of ten. For example:

427.93456 is expressed as $.42793456 \times 10^3$
.0009762 is expressed as $.9762 \times 10^{-3}$.

Internally these numbers are carried as 5342793456 and 4797620000 where the first two digits are the characteristic (i.e., the power of ten with 50 added to avoid a negative characteristic) and the remaining eight digits are the mantissa. In floating point the decimal is always moved to the left of the first *nonzero* digit, with proper adjustment to the exponent. This is called *normalizing* the number.

In GOTRAN either fixed-point or floating-point numbers can be used, subject to the rules described under ARITHMETIC STATEMENTS.

Constants, Variables and Subscripts

Within the framework of fixed-point and floating-point arithmetic, the system must provide for expressing constants and variables.

Whereas in ordinary computations these quantities are expressed routinely and with considerable freedom, very definite rules must be observed in GOTRAN statements, as the computer is incapable of exercising judgment in interpreting the wishes of the programmer. Also, a *floating-point* variable may be subscripted so that it becomes one element of a one-dimensional array.

The GOTRAN programmer will encounter these four kinds of numbers in writing arithmetic statements.

1. Fixed-Point Constants
2. Fixed-Point Variables
3. Floating-Point Constants
4. Floating-Point Variables (these may be subscripted)

Arithmetic Statements

Since the arithmetic statements define the numerical calculations that a program is to execute, they will, no doubt, constitute the heart of most GOTRAN programs. An arithmetic statement *resembles,* but is not, in fact, a simple statement of equality. It consists of a variable to be computed, followed by an equal sign, followed by an expression.

The mode of arithmetic used in an arithmetic statement can be either fixed point or floating point, but cannot be mixed. The mode is determined by the conventions used to write the statement.

By its nature, fixed-point notation is somewhat restricted. Thus it is expected that most statements will be in floating-point form. The following descriptions are divided into sections on fixed-point and floating-point statements.

Fixed-Point Constants

Fixed-point constants may consist of one to three decimal digits. A preceding plus or minus sign is optional

Examples: +7
29
—438

Fixed-Point Variables

Fixed-point variables must be represented by one of the alphabetic characters I, J, K, L, M, or N.

Examples: I
M
K

Fixed-Point Arithmetic Statements

The expression to the right of the equal sign in a fixed-point arithmetic statement may contain a constant and a variable and one operation symbol. The operation symbols in fixed-point statements are restricted to + and —. Fixed-point multiplication and division are not allowed. Thus, a fixed-point statement *must* be in one of the following forms:

V=C
V=V+C
V=V—C

where *v* represents a variable and *c* a constant.

Examples: J=765
M=J+42
L=N—372

The results of a fixed-point arithmetic operation are restricted to a maximum of three digits. If a result greater than three digits is generated, only the three low-order (rightmost) digits are carried as the result.

The equal sign in any arithmetic statement means *is to be replaced by* rather than *is equal to.* A statement such as N=N—40 is therefore valid. The quantities represented by a fixed-point variable must, of course, be fixed point. Fixed-point variables may not be subscripted.

Floating-Point Constants

Floating-point constants may consist of any number of decimal digits (up to 20) with a decimal point at the beginning, at the end, or between two digits. A preceding plus or minus sign is optional.

Examples: 562.
—0.0583
3.1415926

An alternative way of expressing floating-point constants is to affix a one-digit or two-digit decimal exponent preceded by an E (for exponent). The E must immediately follow the last digit of the number.

Examples: 14.2E3=14.2 x 10^3 or 14200.
5243.7E—5=5243.7 x 10^{-5} or .052437
3.0E11=3.0 x 10^{11} or 300000000000.

In GOTRAN, floating-point constants are expressed in "external" form by the user (i.e., in either of the two forms shown as examples), and are converted automatically to the form in which they are carried internally (see MODE OF ARITHMETIC). Internally, floating-point constants become ten-digit quantities in the form:

$$\overbrace{\text{XX}}^{c}\ \overbrace{\text{XXXXXXXX}}^{m}$$

where *c* is the characteristic, *m* the mantissa, and the original number is $m \times 10^{c-50}$.

The magnitude of the number thus expressed must be zero, or must lie between 10^{-50} and 10^{49}.

Floating-point numbers that contain more than eight digits (up to 20 digits) have their magnitude properly maintained by correct adjustment to the characteristic. Only the eight high-order (leftmost) significant digits, however, are carried as the mantissa.

Floating-Point Variables

Floating-point variables are expressed as one to four alphabetic or numerical characters (no special characters). The first of these characters must be alphabetic, but may not be I, J, K, L, M, or N, since these are reserved for fixed-point variables.

Examples: B
HIT2
PRES

SUBSCRIPTED FLOATING-POINT VARIABLES

A floating-point variable may be subscripted by appending to it one *fixed-point variable* enclosed in parentheses. The floating-point variable can thus be made to represent any member of a one-dimensional array. The subscripts are fixed-point variables with values that determine which member of the array is meant.

Examples: C (I)
TEMP (N)

The use of a fixed-point subscript with a floating-point variable does *not* violate the rule of mixing modes in an arithmetic statement.

Functions

A GOTRAN floating-point statement may contain any one of six functions and its argument. The functions are:

LOG (X)	meaning LOG_e of X
EXP (X)	meaning e^x
SQR (X)	meaning $\sqrt{X}$
SIN (X)	meaning SINE of X
COS (X)	meaning COSINE of X
ATN (X)	meaning ARC TANGENT of X

The name of the function to be evaluated must be written with the three characters shown, followed by the argument in parentheses. The argument *must* be a floating-point variable; it may or may not be subscripted.

Examples: SIN (B)
LOG (Q (N))
SQR (RAD)

Floating-Point Arithmetic Statements

Floating-point constants, variables, and functions have been defined, but before examining some floating-point arithmetic statements, it is necessary to state a few general rules:

The variable to the left of the equal sign must be a floating-point variable (subscripted or not subscripted).

The expression to the right of the equal sign may contain floating-point constants, floating-point variables (subscripted or not), and functions.

The expression to the right of the equal sign must contain no more than one operation symbol. This symbol may be:

+	for addition
—	for subtraction
*	for multiplication
/	for division
* *	for involution (raising to a power)

Examples:
1. TORQ=—7.5
2. SUM=AREA+3.14E—2.
3. RAD=W*T
4. PER=1.0/F
5. ANG2=Y**3.
6. A=B
7. C=ATN (Y)
8. A (I)=B (J)+C (K)

Example No. 2 follows the rule of only one operation symbol per statement although it appears to have two signs; the minus sign merely denotes a negative exponent. Statements such as C=—D+E and A=—B are invalid. The latter statement must be written A=0.0—B to be acceptable in the GOTRAN language. The former, C=—D+E, must be broken into two statements, each containing one operation symbol, thus:

C=0.0—D
C=C+E

for solution to be effected by GOTRAN.

As stated previously, in GOTRAN language, the equal sign means "is to be replaced by" and *not* "is equal to." Therefore, the result of the first statement is that C is replaced by (—D). The result of the second statement is that C (which is now (—D)) is replaced by C+E, which is —D+E. The solution is complete.

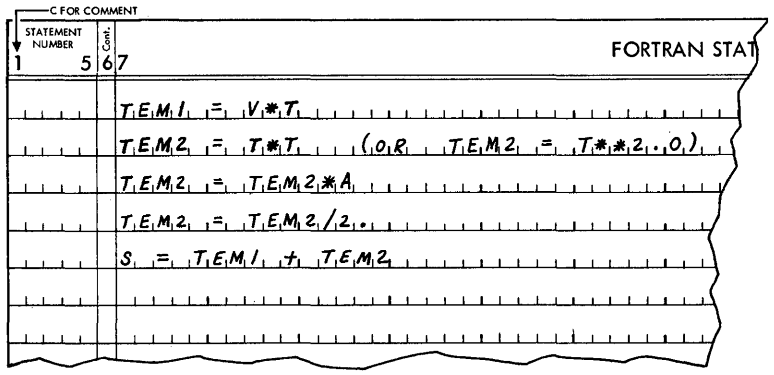
Coding a Typical Formula

As a further illustration of the manner in which arithmetic formulas containing more than one operation symbol must be broken down into several statements, consider the formula:

$$s=vt+\frac{at^2}{2}$$

CODING STATEMENTS

Figure 6 shows the program coded on the standard IBM FORTRAN Coding Sheet, Form X28-7327-2. This form is also used in 1620 GOTRAN. Each GOTRAN statement is entered on a separate line of the coding sheet. A statement cannot exceed 72 characters in length, including blanks and the end-of-line character used to terminate it. Any number of blanks may be included in the statement, with one minor exception which is noted under COMMENTS.


C FOR COMMENT
STATEMENT NUMBER 1 5 | Cont. 6 | 7 FORTRAN STAT

```
TEM1 = V*T
TEM2 = T*T   (OR  TEM2 = T**2.0)
TEM2 = TEM2*A
TEM2 = TEM2/2.
S = TEM1 + TEM2
```

Figure 6. Breakdown of the Equation $s = vt + \frac{at^2}{2}$

COMMENTS

An entry beginning with a C followed by two blanks is considered to be a comment. It is not processed. If the first character of a statement to be *processed* is C, only one blank may follow it. Comments may be used at any point in a program; they are typed out at the console during compilation.

The third and fourth statements of Figure 6 again illustrate the fact that an equal sign in an arithmetic statement means *is to be replaced by* rather than *is equal to*. This concept may be better understood if we consider for a moment the internal functioning of the machine.

The basic IBM 1620 has 20,000 positions of core storage, each of which is capable of holding a single digit of information. Each core storage position has a unique 5-digit address; addresses run sequentially from 00000 to 19999. Thus, each single digit of information is addressable. When several digits are to be treated as a single quantity (e.g., a 10-digit floating-point number), it too is addressable by a unique address that references the low-order position of the field. The high-order position is identified by a *flag bit*. Therefore, the machine language instruction 22 15000 19000 is a command to subtract (22 is the machine operation code for subtract) the quantity stored at address 19000 from the quantity stored at address 15000. The result of the operation is stored at address 15000; the quantity at 19000 remains unaltered.

In GOTRAN, the letters or names used for variables are merely convenient methods of noting machine addresses. The GOTRAN instruction A=B—C can be translated literally as "subtract the quantity stored at C from that stored at B and place the result in location A." In this operation, the previous quantity stored at A is replaced by the quantity B—C; quantities B and C remain the same.

In Figure 6, then, the second statement sets up a temporary storage location (TEM2) in which the value of t^2 is stored. The next statement multiplies this quantity by a, and stores the value of at^2 back in the same storage location. The fourth statement divides the result of the previous statement by 2, and the value of at^2 is now replaced by $at^2/2$.

Another important point is illustrated in Figure 6. The divisor in the fourth statement and the exponent in the alternate second statement are constants. Normally they would be written as integers. In this example, however, the statements are written in floating-point form and therefore the integer (2), which is considered a fixed-point notation in GOTRAN, must abide by floating-point conventions and contain a decimal point.

Control Statements

The second class of GOTRAN statements comprises seven control statements. These are so named because they permit the programmer to control the flow of his program. Their format and use are explained in this section.

Unconditional GO TO Statement

GENERAL FORM: GO TO *n*

Ordinarily, the computer executes instructions sequentially; if it is desired to vary this pattern, the GO TO statement is used. It causes the program to branch to some statement (*n*), other than the next sequential statement, where *n* represents the statement number. Any unsigned integer from 1 to 999 may be used. Statement numbers are used merely for cross-referencing within a program; they need not be assigned in numerical sequence, nor do all statements require statement numbers.

The GO TO is unconditional in that execution of this instruction *always* causes a branch to statement *n* and the program proceeds from that point.

A blank must separate the words GO and TO.

IF Statement

GENERAL FORM: IF $(a)\ n_1, n_2, n_3$ where a is a floating-point variable (subscripted or not) and n_1, n_2, n_3 are statement numbers. A blank must follow the word IF.

The IF statement says: if $(a) < 0$, go to n_1
$(a) = 0$, go to n_2
$(a) > 0$, go to n_3

Thus, the IF statement provides a means of conditionally branching on the basis of the value of the floating-point variable (a). For example, the statement

IF (A (I)) 5, 10, 12

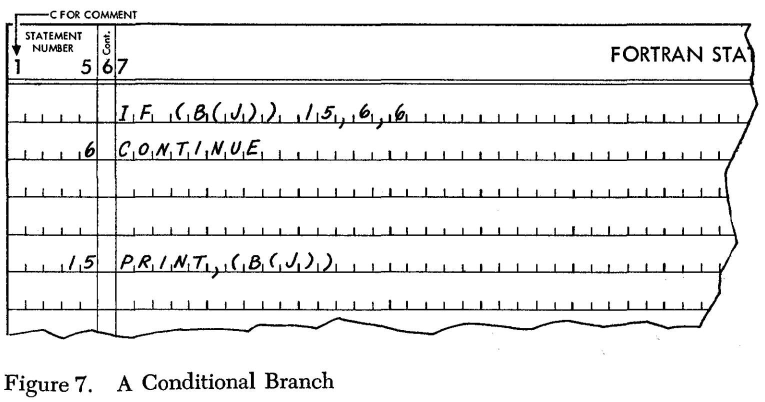
causes the subscripted variable $A\ (I)$ to be tested. If it is negative, the program branches to statement 5; if it is zero, to statement 10; if it is positive, to statement 12.

Branching out of the main line sequence of a program when a number is negative can be done by using an IF statement as shown in Figure 7. Statement 15 could be the first of a series of instructions treating the negative number is a special way. At the end of this sequence, it might be desired to branch back to the main line sequence of the program. Thus, the last instruction of the sequence beginning with statement 15 might be the statement GO TO 6. These techniques of branching conditionally and unconditionally are shown to illustrate examples of the use of the IF and GO TO statements. Other uses will suggest themselves in the course of using GOTRAN.

DO Statement

GENERAL FORM: DO $\overbrace{n}^{\text{range}}\quad \overbrace{i = m_1,}^{\text{index}}\quad \overbrace{m_2,}^{\text{limit}}\quad \overbrace{m_3}^{\text{increment}}$

where n is a statement number,
i is a fixed-point variable and
m_1, m_2, m_3 are unsigned, fixed-point constants.
m_3 may be omitted, in which case it is understood to be 1.
m_1, m_2, m_3, must not exceed 3, 3, and 2 digits, respectively. A blank must follow the word DO.


C FOR COMMENT

STATEMENT NUMBER 1 5	Cont. 6	7 FORTRAN STA
		IF (B(J)) 15,6,6
6		CONTINUE
15		PRINT, (B(J))

Figure 7. A Conditional Branch

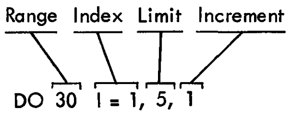
The DO statement is perhaps the most powerful of all in the GOTRAN language. It is a command to execute repeatedly a group of succeeding statements (a loop). *n* specifies the *range* of the DO (or DO loop). The range is the sequence of consecutive statements immediately following the DO statement, up to and including the statement numbered *n*.

The *index* of the DO is the fixed-point variable *i* which is set at the value m_1 for the first execution of the loop and is increased each time by the *increment*(m_3) until it is about to exceed the *limit* (m_2).

As previously stated, if the increment (m_3) is not specified in a DO statement, it is assumed to be 1. Thus, the first time the range is executed, $i = m_1$, then $i = m_1 + 1$, and so on until $i = m_2$.

Throughout the range, *i* is available for computation, either as an ordinary fixed-point variable or as a subscript (see also INDEX VALUES).

The DO statement says: Execute the following statements down through *n*. Start with $i = m_1$. Each time statement *n* is executed, increase the value of *i* by m_3 and begin again with the first statement following the DO. Continue in this way so long as *i* does not exceed m_2. When a further increase in *i* will cause it to exceed m_2, proceed to the next statement following statement *n*. Figure 8 illustrates three typical DO statements.



DO 6 I = 2, 10, 2

DO 12 I = 1, 20

Figure 8. Typical DO Statements

The Index as a Subscript

For an example of the use of the index of a DO statement as a subscript, consider the statements shown in Figure 9.

In this program it is desired to calculate A = B + C for twenty values of B and C. Mathematically, this is expressed: $A_i = B_i + C_i$, $i = 1, 2, \ldots 20$. Assume the twenty values of *B* and *C* (arrays *B* and *C*, respectively) are already in storage and control has reached statement 10. The range of the DO is statement 11, and the index is 1. The DO sets *I* equal to 1 and control passes into the range. $B(1) + C(1)$ is computed and stored in $A(1)$. Now, because statement 11 is the last statement in the range of the DO and the DO is unsatisfied, *I* is increased to 2 and control returns to the beginning of the range, statement 11. $B(2) + C(2)$ is computed and stored in $A(2)$. This continues until statement 11 has been executed with $I = 20$, thus completing the formation of array *A*. Since the DO is satisfied, the program proceeds to statement 12.

The following rules must be observed with regard to DO loops:

Rule 1. No two DO statements in the same program may have the same value of *n*.

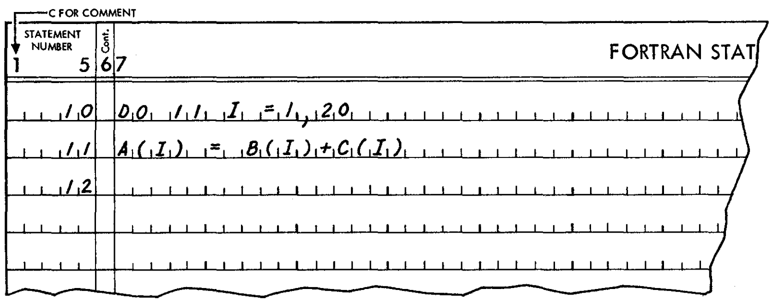
Rule 2. The program must not branch to the last statement (*n*) of a DO loop.

Rules 1 and 2 may be summarized as follows: Once a statement number appears in a DO, another DO, IF, or GO TO must not refer to this same statement.

Rule 3. The last statement in the range of a DO must not be a branch, either conditional or unconditional. In other words, it must not be an IF or GO TO statement.

DO's Within DO's

One or more DO statements may be included within the range of another DO statement, provided a fourth rule is observed.


C FOR COMMENT
STATEMENT NUMBER
Cont.
1
5
6
7
FORTRAN STAT

```
   10  DO 11 I = 1, 20
   11  A(I) = B(I)+C(I)
   12
```

Figure 9. DO Loop for an Array

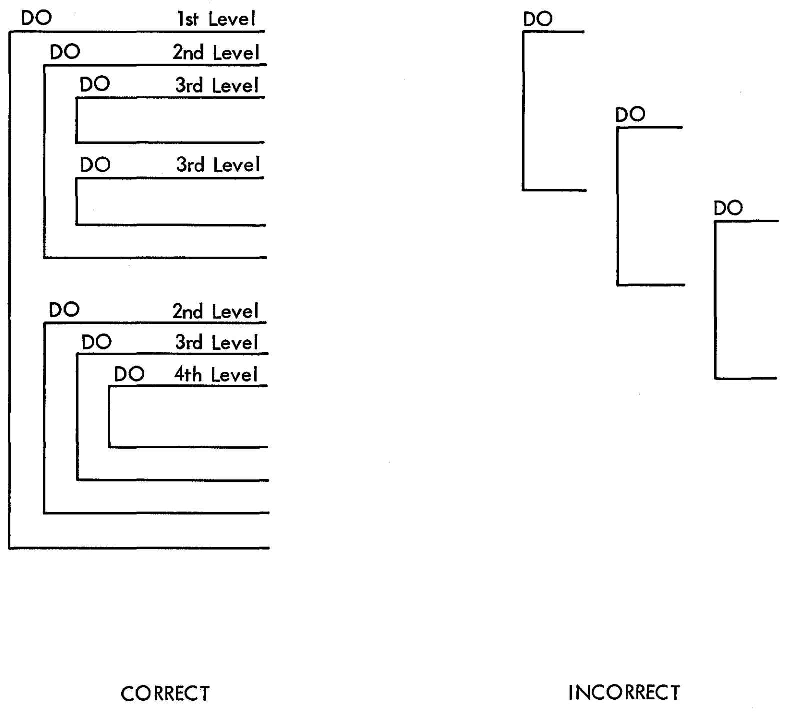


Figure 10. Nested DO Loops

Rule 4. If the range of a DO statement includes another DO statement, all statements in the range of the included statement must also be within the range of the encompassing statement. Figure 10 illustrates this rule.

A set of DO's satisfying this rule is called a *nest* of DO's. As can be seen from Figure 10, more than one *level* of DO loops is possible. In GOTRAN, as many as *six levels* of DO loops in one nest are permitted. Any number of loops may exist within each level. All DO loops, whether nested or single, must obey rules 1, 2, and 3. Rule 4 applies only to nested DO's.

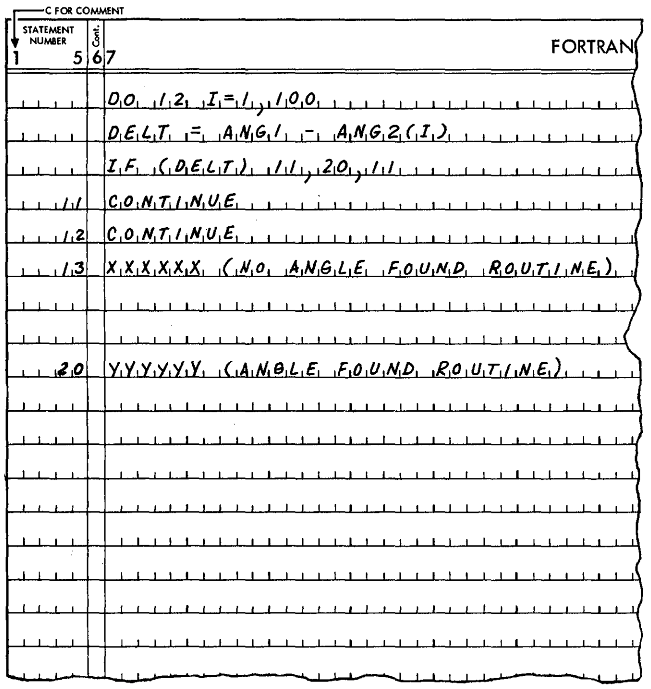
Index Values

A significant aspect of DO loops is the use and preservation of index values. As previously indicated, the index is a fixed-point variable and hence must be represented by one of the six characters reserved for this purpose: *I, J, K, L, M,* or *N*. In a nest of DO's, each DO loop level begins before the loop of the previous level has been completed. Therefore, the index of the inner DO loop must not be the same as the succeeding higher levels of loops. For this reason the levels of DO loops in any one nest is restricted to six. Within each level, however, a particular index may be used as often as desired. For example, assume the first-level DO in Figure 10 has an index of *J*; no other DO *within* this nest can use *J* as an index. This leaves the variables *I, K, L, M, N* for indexes of the second level. Assume both DO's in this level use the index *M*. This is permissible because the index value of the first loop is completely used before the second loop is begun. Any one or all of the letters *I, K, L, N* can be used for the third level.

Program control may leave the DO loop in one of two ways: either as a result of the DO becoming satisfied, or as a transfer out of the DO range by an IF or GO TO statement (see Figure 11). With the DO becoming satisfied and control passing to the next statement after the range, the exit is said to be a *normal* exit. After a normal exit from a DO occurs, the value of the index controlled by that DO is the value of m_2 or the highest value in the sequence that does not exceed m_2. In this case and in the case in which the exit occurs by a transfer out of the DO range, the current value of the index remains available for any subsequent use. If the exit occurs by a transfer that is in the range of several DO's, the current value of all the indexes controlled by these DO's are preserved for any subsequent use.

In IBM 1620 GOTRAN it is possible to transfer into the range of any DO statement from outside its range by using either IF or GO TO statements. The value of the index is, however, not reset by such a transfer. The current value is therefore used, or the value of the index may be defined prior to the branch.

Statements within the range of a DO can be used to redefine the value of the index. Naturally, changing the


C FOR COMMENT
STATEMENT NUMBER
Cont.
1 5 6 7
FORTRAN

```
      DO 12 I=1,100
      DELT = ANG1 - ANG2(I)
      IF (DELT) 11,20,11
   11 CONTINUE
   12 CONTINUE
   13 XXXXXX (NO ANGLE FOUND ROUTINE)


   20 YYYYYY (ANGLE FOUND ROUTINE)
```

Figure 11. Table Search

index during the execution of the DO loop can lead to unexpected results unless due caution is exercised.

CONTINUE Statement

This is a nonexecutable dummy statement. It consists merely of the word CONTINUE and is analogous to the instruction NO OP which is found in nearly all computer command structures.

CONTINUE is frequently used in the range of a DO to fill the requirement that the last statement in the DO loop cannot be a branch statement. As an example of a program that requires CONTINUE statements, consider the table search program shown in Figure 11.

This program examines the 100-entry angle table until it finds an entry that equals ANG1, whereupon it will exit to statement 20, with the successful value of *I* available for fixed-point use: if no entry in the array equals ANG1, a normal exit to statement 13 ultimately occurs. The same program without the two CONTINUES would not work because

A) the IF statement (a conditional branch) must not refer the program to the last statement of the DO loop, and

B) the last statement in the range of a DO is a branch.

Another use of the CONTINUE statement is in a nest of DO loops. A rule of DO nesting is that no two DO statements can end in the same statement number. Figure 12 illustrates the manner in which CONTINUES are used to achieve the effect of having a common statement terminating several DO loops, without violating the rule of two DO's ending with the same statement.

PAUSE Statement

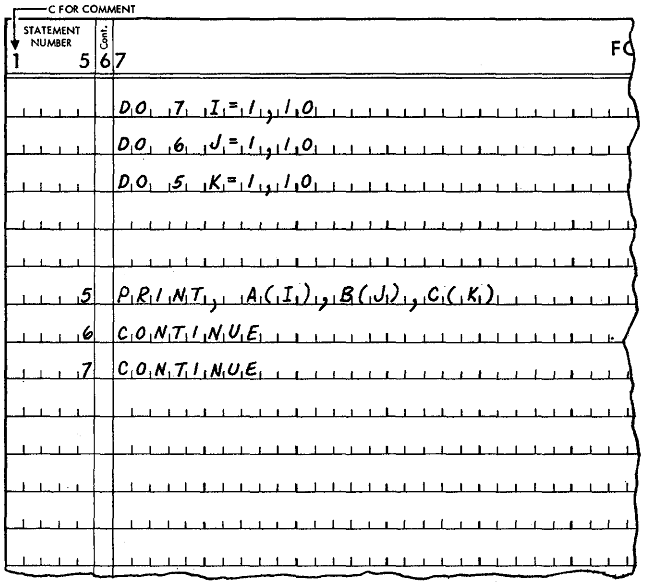
This statement consists of the word PAUSE. The program will halt upon execution of the PAUSE statement. Pressing the Start key causes the program to resume operation with the next GOTRAN statement. Inserting PAUSE statements into the program gives the programmer the opportunity to check its performance at critical points.

STOP Statement

This statement consists of the word STOP. Execution of the statement causes the program to halt. Prior to halting, the machine will print out: STOP END OF PROGRAM. Depressing the Start key conditions the machine to accept another GOTRAN program. Stop statements may be inserted in the program to indicate various conditions beyond which the programmer does not want the program to progress.

END Statement

This statement consists of the word END. It *must* appear as the last statement of the GOTRAN program. The END statement informs the GOTRAN compiler that there are no more source statements to be compiled. As a result of the END statement, the phrase END OF PROGRAM is compiled and the program halts. Pressing the Start key now causes the program to be executed. END OF PROGRAM will print when execution is completed.


C FOR COMMENT
STATEMENT NUMBER
Cont.
1 5 6 7
F

```
      DO 7 I=1,10
      DO 6 J=1,10
      DO 5 K=1,10


    5 PRINT, A(I), B(J), C(K)
    6 CONTINUE
    7 CONTINUE
```

Figure 12. Use of CONTINUES with DO Loops

Input/Output Statements

In order to execute the GOTRAN program, the machine must be provided with values corresponding to the variables used in the statements. This *input data* may be brought into core storage via punched cards, paper tape, or the console typewriter. Answers to problems and other intelligence from the computer must be furnished to the user. This *output data* may take the form of punched cards, punched paper tape or a printout from the console typewriter.

There are four GOTRAN statements concerned with the transmission of data between core storage and the various input/output devices of the 1620, i.e., the 1620 Paper Tape Reader, the 1622 Card Read Punch, the 1624 Tape Punch, and the Console Typewriter. Three of these statements are the conventional Read, Punch, and Print commands. The fourth is a powerful new statement for plotting curves.

READ Statement

A READ statement causes the program to read data into core storage from punched cards (a 1620 equipped with a 1622 Card Read Punch) or from punched paper tape (a 1620 equipped with a 1621 Paper Tape Reader). The same statement used in conjunction with a console switch causes either system to accept data entered manually at the console typewriter. It must be borne in mind that the program is calling into the computer variable data, not GOTRAN statements, and that this is occurring during the execution phase. The manner in which GOTRAN statements (including the READ statement itself) are loaded into the machine is described under GOTRAN OPERATING PROCEDURES.

GENERAL FORM: READ, List of Variables*
where the list consists of from 1 to 5 fixed-point or floating-point variables (they may be intermingled), separated by commas.

The READ statement causes input data records (each value is one record) to be brought into core storage until the complete list has been read in. When each variable in the list (up to five) has received a value, the list is said to be *satisfied.*

*READ, PUNCH, or PRINT statements may also contain a number *n* following the verb, i.e., READ *n*, list. It represents format and is ignored in GOTRAN, but is mentioned here so that it will be understood.

Examples: READ, A, B, L, M, TEMP
READ, D, J, X (K)
(Note that the last item in the list is not followed by a comma.)

The records (card, tape, or console typewriter) are brought into the machine, converted to internal notation if floating point, and stored in the locations specified in the list of the READ statement.

Card Input

Each card is one record; each card may therefore contain the value assigned to one variable. To satisfy a list of 5 variables requires 5 cards.

The value to be assigned to a variable is punched in the card in any columns from 1 through 72. Each floating-point number can contain up to 20 digits, of which the eight high-order significant digits will be used. The internal characteristic will be adjusted to reflect the magnitude of the number.

For example, the number
00161473902.75
is brought into the machine as
161473902.75
and converted to
5916147390

If the number is beyond the range of values that can be expressed with 20 digits, it is necessary to express it in *exponential* form (see FLOATING-POINT CONSTANTS).

Neither an end-of-line character nor a record mark is required on the card. The order of the input data records (cards) must be the same as the order of the list associated with the READ statement that brings the data in. For example, the list of the statement
READ, A, B, L, M, TEMP
calls for two floating-point numbers, two fixed-point numbers, and one floating-point number, in that order (see Figure 13).

Paper Tape Input

The same rules that apply to card input apply generally to paper tape input. Each input value must appear as a separate record. With paper tape, however, an end-of-line (EOL) character must immediately follow the last digit. Each record is limited to 72 characters including blanks and the EOL character. As with card input, each floating-point number can contain up to 20

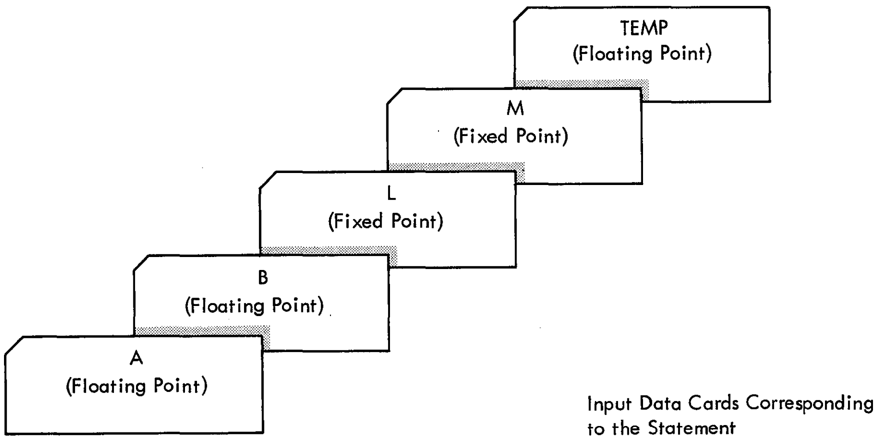


Figure 13. Input Data Records

digits, of which only the eight high-order significant digits will be used. When expressed in exponential form, the *E* must follow immediately after the last digit of the number. The plus or minus sign may separate the exponent from the *E*.

Console Typewriter Input

If Program Switch 1 is set to OFF during execution, and a READ statement is executed, the program selects the console typewriter as an input device and halts. It is up to the operator to enter the input data manually, in the same order and in the same mode as called for in the list of the READ statement responsible for the halt. An EOL character must be typed at the end of each input value in the list.

PUNCH Statement

GENERAL FORM: PUNCH, List of Variables
where the list consists of from 1 to 5 fixed-point or floating-point variables separated by commas. The PUNCH statement causes the program to punch data into cards or paper tape until the complete list has been satisfied.

Examples: PUNCH, A, C, P (J)
PUNCH, B, D, L, M, X

A maximum of five variables can be punched out by any one PUNCH statement.

Each variable is punched as a separate record; that is, one variable is punched per card if the 1622 Card Read Punch is used. If the output is produced on the 1624 Tape Punch, the last character of each variable punched is followed by an EOL.

PRINT Statement

GENERAL FORM: PRINT, List of Variables
where the list consists of from 1 to 5 fixed-point or floating-point variables separated by commas. The PRINT statement causes the GOTRAN program to type output data on the console typewriter.

Examples: PRINT, A, C, P (J)
PRINT, B, D, L, M, X

One line of data is typed for each PRINT statement. A maximum of five variables is typed per line.

The typewriter carriage returns before the value of the first variable listed in the PRINT statement is typed. Tabulation occurs between the typing of each variable. The tab stops must be set by the operator. Typing continues until all variables specified in the list have been printed.

Output Data Format

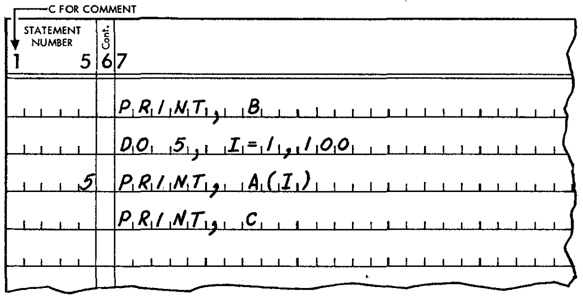
The numbers punched or printed are in the same mode and same order as the names of the corresponding variables that represent them in the source program statement. Fixed-point numbers are in the same form as input fixed-point constants (see FIXED-POINT CONSTANTS); the value is signed only when it is negative. Floating-point numbers are in one of the following two forms:

1. Eight digits with a decimal point at the beginning, end, or between two digits, preceded by a – sign, when applicable.

For example:

3.0000000
–17.437986
.79340000
0.0*

*Any number that contains all zeros in the mantissa appears as 0.0 during output.


C FOR COMMENT
STATEMENT NUMBER
Cont.
1 5 6 7

```
      PRINT, B
      DO 5, I=1,100
    5 PRINT, A(I)
      PRINT, C
```

Figure 14. Output DO Loop

2. Eight digits with a decimal point between the first two, followed by a decimal exponent consisting of a signed (if negative) two-digit fixed-point number preceded by an E. The number is preceded by a — sign if it is negative.

   Examples: 4.1000000E09
   1.7986437E—12
   —1.6147390E03

The selection of either one of the preceding forms for floating-point numbers being printed or punched is under the control of the GOTRAN system. Whenever possible, output values are in the form first shown. If the output quanity is beyond the range of values covered by this form, the number is printed or punched in the exponential form. Hence, the output values always contain all the significant digits computed.

Use of Input/Output DO Loops

No indexing, (e.g., READ, *A (I) I*=1, 5) is allowed within a list. Thus, every item in a list corresponds to one and only one quantity read into or out of the machine when the associated READ, PUNCH, or PRINT statement is executed. Arrays with members that are so numerous that itemizing them requires a large number of input (or output) statements may be handled by using one input/output statement and including it in the range of an indexed DO statement. For example, suppose quantities *B, A,* and *C* are to be printed, in that order, and *A* represents a one-dimensional array of 100 elements. The statement PRINT, *B,* (*A*(*I*) *I*=1,100), *C* is not valid because no indexing is allowed within a list. Specifying each element of the 100-element array requires twenty PRINT statements, thus rendering this method cumbersome and impractical. The solution employed is shown in Figure 14. Note that the statement to print the array is included within the range of a DO statement.

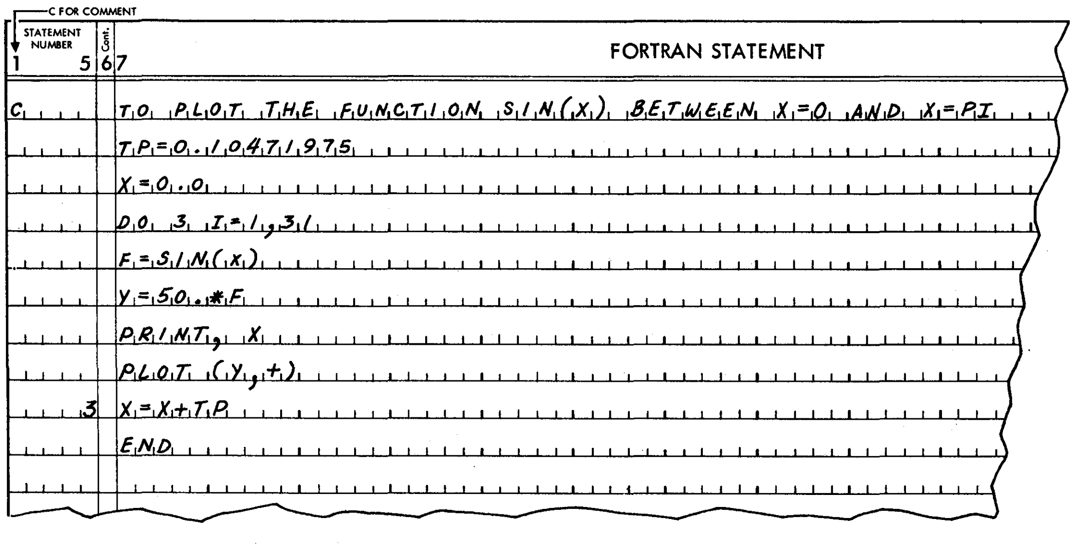
PLOT Statement

With this statement it is now possible to plot curves on the console typewriter of the 1620.

GENERAL FORM: PLOT (*v*, *c*)

where *v* is a floating-point variable and *c* is the character to be used in the plot.

The PLOT statement spaces the carriage laterally the number of spaces that is equal to the integral part of the variable *v* and prints the character specified as *c*. This character may be any 1620 character, except the record mark.


C FOR COMMENT
STATEMENT NUMBER
Cont.
1 5 6 7
FORTRAN STATEMENT

```
C     TO PLOT THE FUNCTION SIN(X), BETWEEN X=0 AND X=PI
      TP=0.10471975
      X=0.0
      DO 3 I=1,31
      F=SIN(X)
      Y=50.*F
      PRINT, X
      PLOT (Y,+)
    3 X=X+TP
      END
```

Figure 15. A Program to Plot SIN (X)

Examples: PLOT (X, *)
PLOT (BETA, —)

Only a PRINT statement returns the carriage, thus functions of more than one variable may be plotted by suitable programming and scaling.

If the standard margin and tab settings* are used, the variable v should be scaled between 0 and 66.

Since only one point is plotted for each PLOT statement, the statement should be made part of a DO loop to plot the entire curve. Such a DO loop is illustrated in Figure 15. This program plots the function sin (x) for 31 values of x ranging from 0 to π.

The value chosen for TP is approximately $\pi/30$, selected so that 30 points on the curve (31 including zero) are plotted.

The statement $y=50 * F$ is needed to scale the function to a realistic value for representation as spaces of the typewriter carriage. The actual curve produced at the console typewriter, along with the corresponding values of x, is shown in Figure 16.

NOTE: The PLOT statement is not compatible with IBM 1620 FORTRAN.

*See MARGIN AND TAB SETTINGS.

```
C   TO PLOT THE FUNCTION SIN(X) BETWEEN X=0 AND X=PI
      TP = 0.10471975
      X = 0.0
      DO 3  I = 1, 31
      F = SIN (X)
      Y = 50. * F
      PRINT , X
      PLOT (Y, +)
  3   X = X + TP
      END
```

NOTE: Final Value of X is Slightly Less Than Correct Value of π to 7 Decimal Places (3.1415926) because Point Values Chosen are Truncated to 8 Places.

```
0.0          +
.10471975         +
.20943950              +
.31415925                   +
.41887900                        +
.52359875                             +
.62831850                                 +
.73303825                                     +
.83775800                                         +
.94247775                                            +
1.0471975                                               +
1.1519172                                                 +
1.2566370                                                   +
1.3613567                                                    +
1.4660765                                                     +
1.5707962                                                      +
1.6755160                                                     +
1.7802357                                                    +
1.8849555                                                   +
1.9896752                                                 +
2.0943950                                               +
2.1991147                                            +
2.3038345                                         +
2.4085542                                     +
2.5132740                                 +
2.6179937                             +
2.7227135                        +
2.8274332                   +
2.9321530              +
3.0368727         +
3.1415925    +
END OF PROGRAM
```

Figure 16. Machine Plot of SIN (X)

Specification Statement

The final class of statements in the GOTRAN language is the specification statement of which there is but one: DIMENSION. This is a nonexecutable statement that supplies the GOTRAN compiler with necessary information regarding arrays.

DIMENSION Statement

GENERAL FORM: DIMENSION v_1 (n_1), v_2 (n_2)
where each v is a subscripted floating-point variable and each n represents a fixed-point constant (the subscript). A blank must separate DIMENSION from the first variable.

The DIMENSION statement provides the information necessary to allocate storage for arrays. Each floating-point variable that appears in subscripted form in a program must appear in a DIMENSION statement of that program; the DIMENSION statement must precede the first appearance of that variable. It lists the maximum number of elements in the arrays; references to these arrays must never exceed the specified dimension.

In the following example, *E* is specified as a 10-element array, *V* as a 75-element array, and *R* as a 15-element array. Only one-dimensional arrays may be specified.

```
DIMENSION E (10), V (75), R (15)
```

A single DIMENSION statement may specify the dimension of any number of arrays as long as the entire statement does not exceed 72 characters.

A DIMENSION statement cannot be used in conjunction with an input/output statement in an attempt to transmit an entire array. For example,

```
DIMENSION X (60)
READ, X
```

cannot be written with the expectation that the 60 elements of array *X* will be read in. To accomplish this, the READ statement within the range of an indexed DO must be used, in a manner similar to the use of the PRINT statement in Figure 14.

Summary of GOTRAN Statements

The following summary, Table 1, presents the thirteen GOTRAN statements in condensed form. Note that all of the arithmetic statements, both ordinary and function, are counted as one statement in the total.

Table 1. SUMMARY OF GOTRAN STATEMENTS

Class	General Form	Remarks
Arithmetic		
Ordinary	A=B+C	Addition
	A=B–C	Subtraction
	A=B * C	Multiplication
	A=B/C	Division
	A=B * * C	Involution
Function	A=LOG (X)	Log_e (X)
	A=EXP (X)	e^x
	A=SQR (X)	$\sqrt{X}$
	A=SIN (X)	Sine (X)
	A=COS (X)	Cosine (X)
	A=ATN (X)	Arc Tangent (X)
Control	GO TO n	Blank must follow "GO"
	IF (A) n_1, n_2, n_3	Blank must follow "IF"
	DO n i = m_1, m_2, m_3	$m_3 = 1$ if Omitted
	CONTINUE	
	PAUSE	
	STOP	
	END	Must be last statement
Input/Output	READ, A, B, C, D, E	
	PUNCH, A, B, C, D, E	
	PRINT, A, B, C, D, E	
	PLOT (v, c)	c Any character but ‡
Specification	DIMENSION	Specifies size of arrays

An Example Problem

This section presents a problem illustrating some of the thirteen different GOTRAN statements that have been explained and examines, in detail, a program using many of them.

Problem: Compute the square root of each of the numbers from 1 to 15.

The solution is based upon successive iterations of the formula

$$RT=\frac{\frac{B}{R}+R}{2}$$

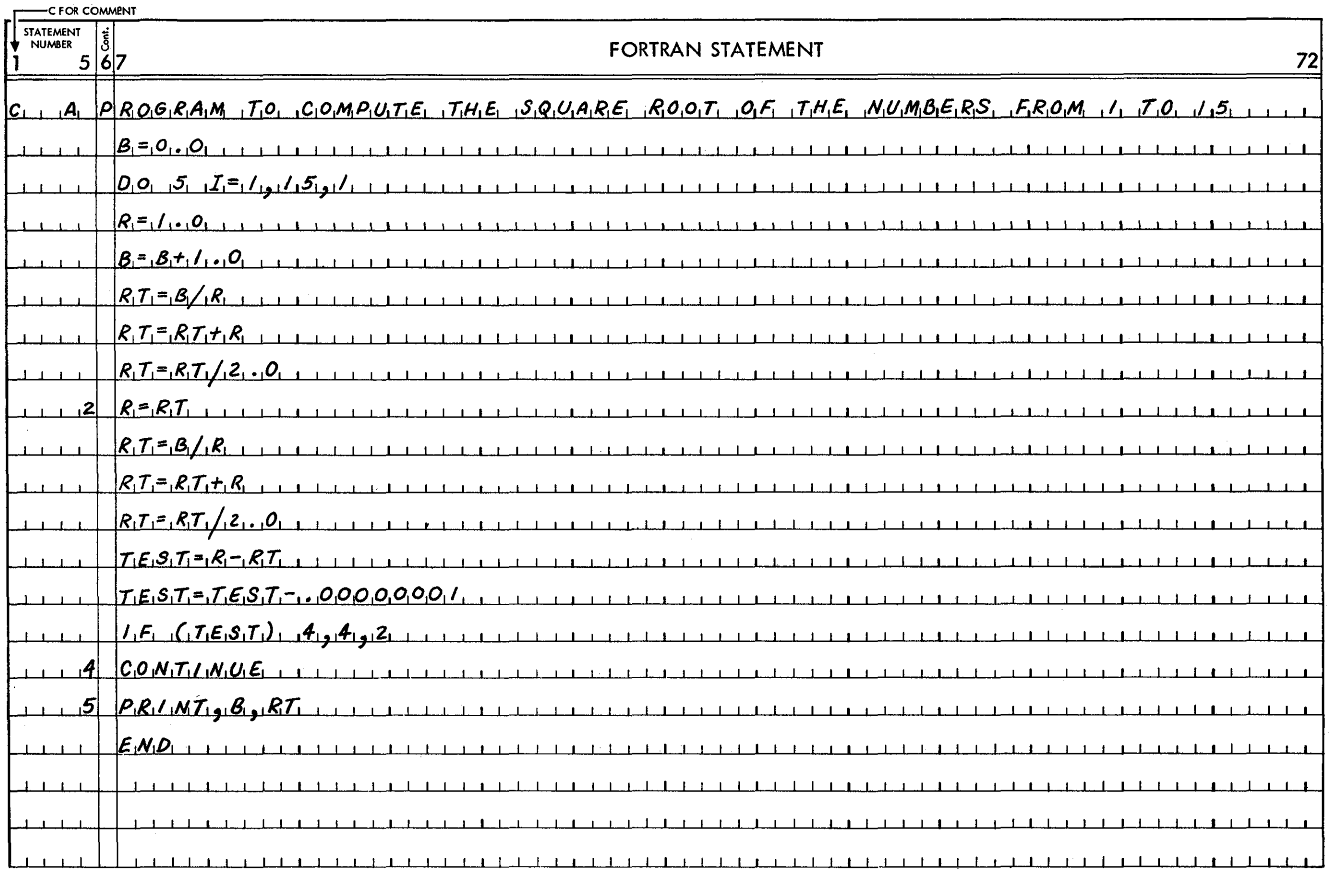
where RT = the square root to be found

B = the number whose square root is to be found (the argument)

R = the successive approximations of RT.

One solution to the problem is presented, along with a comment elaborating on the purpose of each statement used. Figure 17 shows this program.

STATEMENT	PURPOSE
$B = 0.0$	The argument (B) is initially set to zero. It is a floating-point number. All variables that appear on the right side of the equal sign must be previously defined, i.e., they must have appeared on the left side of an equation or they must have been part of an input list. This statement defines the variable *B*.


C FOR COMMENT
STATEMENT NUMBER
Cont.
FORTRAN STATEMENT
1
5
6
7
72

```
C  A P ROGRAM TO COMPUTE THE SQUARE ROOT OF THE NUMBERS FROM 1 TO 15
      B=0.0
      DO 5 I=1,15,1
      R=1.0
      B=B+1.0
      RT=B/R
      RT=RT+R
      RT=RT/2.0
    2 R=RT
      RT=B/R
      RT=RT+R
      RT=RT/2.0
      TEST=R-RT
      TEST=TEST-.00000001
      IF (TEST) 4,4,2
    4 CONTINUE
    5 PRINT,B,RT
      END
```

Figure 17. A Program to Compute a Series of Square Roots

STATEMENT	PURPOSE
DO 5 $I = 1, 15, 1$	A DO loop is set up to accomplish 15 passes through the succeeding statements, down through statement No. 5. The final 1 in the statement can be omitted.
$R = 1.0$	R is set to 1.0 as an initial approximation.
$B = B + 1.0$	The value of B is increased by 1. This new value of B then *replaces* the old, in accordance with the special meaning of the equal sign in GOTRAN.
$RT = B/R$	The first actual operation of the formula. B is divided by R and the result replaces RT.
$RT = RT + R$	R is added to RT and the result replaces RT. RT now consists of $B/R + R$.
$RT = RT/2.0$	This statement completes one iteration of the formula. RT now has been evaluated once.
2 $R = RT$	The value of R is replaced by the computed value of RT. The next iteration uses this new value of R. This statement is given a number because it is referred to in a subsequent statement. Certain other statements have been assigned numbers for the same reason.
$RT = B/R$ $RT = RT + R$ $RT = RT/2.0$	Another iteration using the previously computed value of RT for the new value of R.
TEST $= R - RT$	The value of RT as found in the latest iteration is subtracted from the previous value (R) and the result replaces TEST. The statement is used to seek convergence – the point at which two successive iterations are nearly equal.
TEST = TEST –.00000001	The value of TEST is adjusted downward by .00000001. If the disparity between the last two iterations is no greater than this amount, the root has been found to the desired accuracy.
IF (TEST) 4, 4, 2	If TEST is zero or negative, no more iterations using the current argument are necessary. Proceed with the program. If TEST is positive, one or more additional iterations are required. Go back to Statement 2.
4 CONTINUE	This dummy statement is inserted to fulfill the requirement that a branch (conditional or unconditional) must not send the program to the last statement of a DO loop. Thus the preceding IF statement references the CONTINUE statement.
5 PRINT, B, RT	The final statement of the DO loop. B is printed along with its square root, RT. The program returns to the first statement following the DO statement, increments I and proceeds to compute the square root of $B + 1.0$. The process of computing, testing, and printing continues through the time that I (and B) equal 15, after which the program proceeds to END.
END	The last statement required for any GOTRAN program. It signals the compiler that there are no further statements to be compiled.

The answers, typed out as a result of Statement 5, are shown in Figure 18.

```
001                                  1.0000000
002                                  1.4142135
003                                  1.7320508
004                                  2.0000000
005                                  2.2360679
006                                  2.4494897
007                                  2.6457513
008                                  2.8284271
009                                  3.0000000
010                                  3.1622776
011                                  3.3166247
012                                  3.4641016
013                                  3.6055512
014                                  3.7416573
015                                  3.8729833
END OF PROGRAM
```

Figure 18. Printout of Square Root Problem

GOTRAN Operating Procedures

The operating procedures for the IBM 1620 GOTRAN system can be divided into three phases (Figure 19):

1. Loading the GOTRAN compiler/interpreter program.
2. Loading and compiling the source program.
3. Executing the interpretive program.

Phase I: Loading the GOTRAN Compiler/Interpreter Program

Loading the GOTRAN system into the 1620 varies slightly depending upon whether the input is via the 1621 Paper Tape Reader or the 1622 Card Read Punch.

Paper Tape Input

To initiate the loading of the GOTRAN system tape, a Read instruction must be manually entered from the console. The procedure is as follows:

1. Depress the Reset key
2. Depress the Insert key
3. Type the Instruction: 36 00000 00300
4. Depress the Release key
5. Depress the Start key

After the GOTRAN compiler/interpreter program has been loaded into storage, the 1620 will halt and await entry of the source statements.

Punched Card Input

Console Program Switch #1 must be set *before* the system deck is loaded. If the switch is OFF, *all* subroutines are loaded. If the switch is ON, SIN, COS and ATN are omitted.

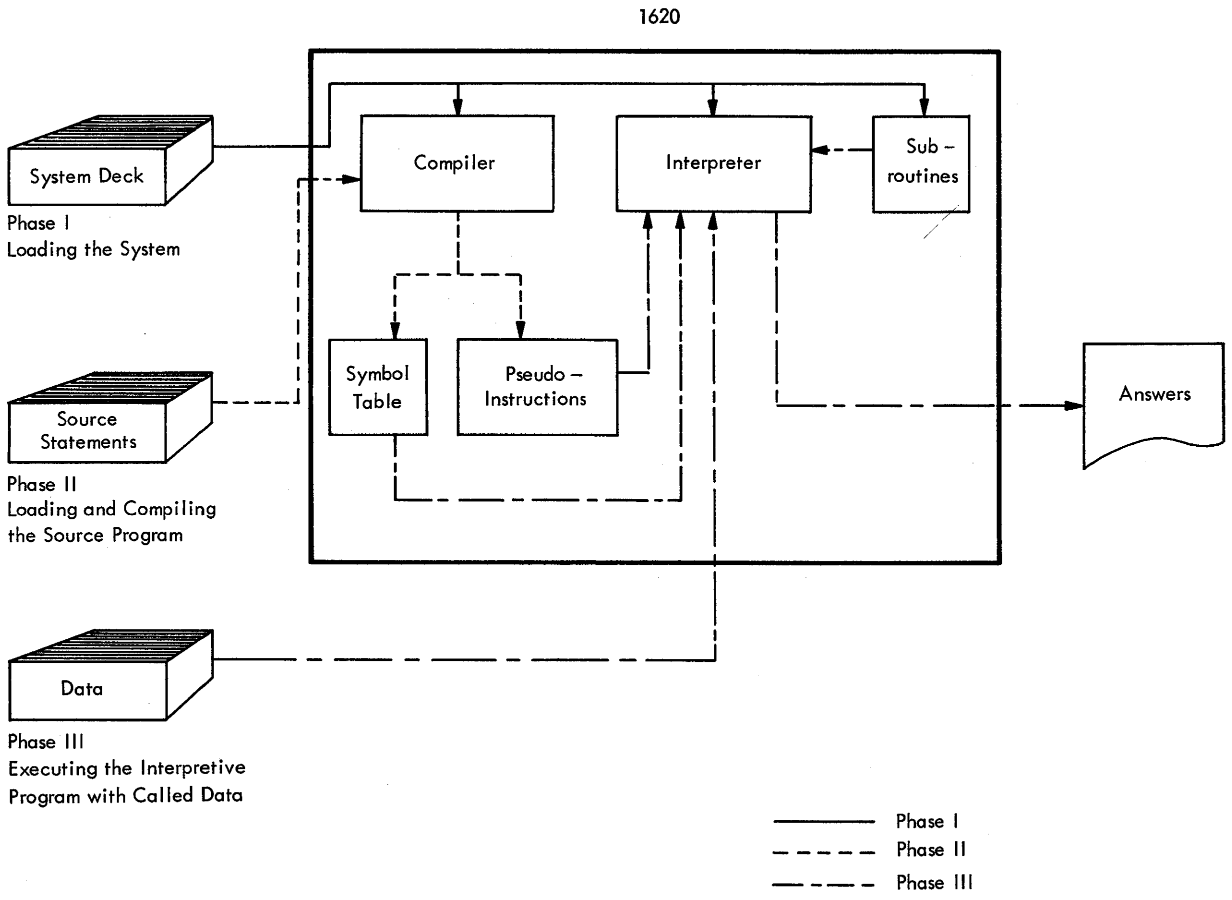


Figure 19. The Three Phases of GOTRAN

Therefore if the trigonometric functions are not needed, they may be left out and the storage saved may be utilized for a larger program.

The procedure for loading the system deck is:

1. Set console switch #1
2. Depress the Reset key
3. Place system deck in Read hopper
4. Depress the Load key

The system deck will now feed through the 1622, causing the compiler/interpreter program to be loaded into 1620 core storage. When loading of the program has been completed, the machine will come to a halt.

Phase II: Loading and Compiling the Source Program

Switch Settings

During the halt that follows the loading of the compiler/interpreter program, the console program switches must be set for the loading and compilation of the source program. The settings of the switches during this operation designate the following:

SWITCH	ON	OFF
1	Source program is being entered on paper tape (tape system) or punched cards (card system).	Source program is being entered via console typewriter (either system).
2	Source statements are punched on paper tape (tape system) or into cards (card system) as they are entered.	Source statements are not punched (either system).
3	Source statements are printed at console (either system).	Source statements are not printed (either system).
4	This switch is used only when an error is made while entering source statements at the console typewriter. It is normally OFF. When a typing error is made, it is turned ON, the Release and Start keys are depressed, and it is turned OFF. The operator may now retype the statement that was in error.	

All Console Check switches are set to PROGRAM.

After the program switches are set according to the method of entering the source program, the appropriate action is taken to initiate loading.

Source Program Entered on Paper Tape

1. Load the tape containing source statements onto the 1621 Paper Tape Reader.
2. Depress Start key on the console.

The source statements will now be loaded into core storage and compiled. After compilation the program will halt to permit the operator to prepare the machine for the next phase of GOTRAN.

Source Program Entered on Punched Cards

1. Place deck containing source statements into 1622 Read hopper.
2. Depress Start key on 1622.
3. Depress Start key on console.

The source statements load and the program halts, as before.

Source Program Entered by Console Typewriter

Since Program Switch 1 is off, the typewriter will be selected as the input device by the program.

1. Depress Start key on console
2. Type first source statement
3. Depress Release key
4. Depress Start key
5. Type next source statement, etc.

When the last source statement has been entered, and the Release key depressed, the program halts as before. The switches may now be set for the execution phase.

Error Messages

The following error messages may be typed out during the compilation phase of GOTRAN:

Message	*Meaning*
Error C1	Symbol table has been exceeded
Error C2	Unacceptable function name
Error C3	Improper operation symbol in arithmetic statements
Error A1	MAR check or any other indicator turned on

After such an error message appears, the program halts, preventing the statement that caused the message from being compiled. The operator may then enter a corrected version of the statement on the console typewriter. The procedure is as follows:

1. If Console Switch 1 is ON, set it OFF.
2. Depress the Start key on the console. The program will select the console and await the entry of the typed statement.
3. Enter the corrected statement.
4. Depress the Release and Start keys (if switch 1 was ON originally to accept input from cards or paper tape, it must be restored to this position after entry of the corrected statement).

Phase III: Program Execution

Switch Settings

During the halt following the compilation phase of the GOTRAN program, the console switches must be set for the execution phase. During this phase the console program switches are interpreted as follows:

SWITCH	ON	OFF
1	Input data to be entered on paper tape (tape system) or punched cards (card system).	Input data to be entered via console typewriter.
2	Result of each arithmetic statement is printed (this is known as the trace mode).	Not used.
3	Not used except in restart procedures.	
4	This switch is used during the execution phase for a typing error on input data. It is used similarly during the compilation phase for a typing error on a source statement (see LOADING AND COMPILING THE SOURCE PROGRAM). Normally, switch 4 is OFF. When a typing error is made, it is turned ON, the Release and Start keys are depressed, and it is turned OFF. The operator may now retype the data that was in error.	

All Console Check switches are set to PROGRAM.

Margin and Tab Settings

During the execution phase, the left margin on the console typewriter should be set at 10 and the right margin at 95. Tab settings should be at 28, 46, 63, 80, and 94.

Tape Data Input

1. Load data tape on 1621
2. Depress Start key on console

The compiled GOTRAN program will now be executed, utilizing the data furnished and calling it in as needed under program control.

Card Data Input

1. Place data cards in Read hopper of 1622
2. Depress Start key on 1622 to run in cards
3. Depress Start key on console

The compiled GOTRAN program will now be executed, reading data cards under program control.

Typewriter Data Input

1. Depress Start key

   Program execution proceeds until such time as the program calls for data input. The console typewriter is selected and the program awaits the manual entry of data.
2. Type in data
3. Depress Release key
4. Depress Start key

Table 2. ERROR MESSAGES

Message	*Operation*	*Meaning*	*Result in Storage*
Error E1	Floating Point, Add or Subtract	Overflow	All nines
Error E2	Floating Point, Add or Subtract	Underflow	All zeros
Error E3	Floating Point, Multiply	Overflow	All nines
Error E4	Floating Point, Multiply	Underflow	All zeros
Error E5	Floating Point, Divide	Overflow	All nines
Error E6	Floating Point, Divide	Underflow	All zeros
Error E7	Floating Point, Divide	Attempt to divide by zeros	All nines
Error F1	SIN (X) or COS (X)	Loss of all significance in function	All nines
Error F2	LOG (X)	$X=0$	All zeros
Error F3	LOG (X)	$X<0$	Log $\|X\|$
Error F4	EXP (X)	Overflow	All nines
Error F5	EXP (X)	Underflow	All zeros
Error F6	A**B	$A<0$	$\|A\|^B$
Error A1	Any Operation	MBR-E, MBR-O, RD CHK, or WR CHK indicators turned on.	Random

Error Messages

Coded error messages of various meanings may be typed out during program execution. They are explained in Table 2.

Restart Procedures

During either the compilation or execution phases it is possible for the machine to halt on an error condition or to go into an uncontrolled loop. The operator then has a choice between two options: He may

1. Reinitialize the GOTRAN compiler to begin processing anew the program in storage, or
2. Reinitialize the GOTRAN compiler to accept a new source program.

Console Program Switch #3 is interrogated for this purpose by the following procedure:

1. Depress Stop key (if machine is in a loop)
2. Depress Reset key
3. Depress Insert key
4. Depress Release key
5. Depress Start key

If Switch #3 is ON, the compiler will initialize itself and come to the halt which precedes the entering of a GOTRAN source program. A new source program may now be entered.

If Switch #3 is OFF, the compiler will begin again the execution of the compiled program currently in core storage.

Obviously, if the error occurs in the loading and compilation phase, the operator should turn switch #3 on and reload the source program, as there is no point in executing an incorrect program.

If the error occurs in the execution phase, he may wish to repeat the program at least once, with different data, before entering a new source program.

Core Storage Arrangement

The GOTRAN system utilizes the 1620 core storage as shown in Figure 20.

The compiled program can make use of any storage not used by the symbol table. Thus, any storage economies effected in this area permit larger programs to be written.

The user can determine whether the program exceeds the capacity of the machine by the following:

If $14\,(N_s)+10(N_y+N_c)\leq 4953$

where N_s = number of statements used,

N_y = number of symbols used*,

N_c = number of constants used,

then the program is within the capacity of a 1620 whose core storage is 20,000 characters. The figures used are averages. However, they do provide the basis for a reasonable estimate of the size of program that can be handled.

*N_y must not exceed 500 (tape system) or 490 (card system).

00000 to 09165	09166 to 14469	14470 to 14759	14760 to 19999
Compiler/Interpreter	Subroutines	Compiled Program	Symbol Table and Data Storage 490 Symbols (Card System) 500 Symbols (Tape System)

Figure 20. GOTRAN Core Storage Layout

Test Problems

A group of seven test problems is presented in this section. They illustrate the use of GOTRAN programs to perform various operations as described throughout this manual. The actual printed output of the console typewriter is shown in each case. The list of statements as typed during compilation is followed by the answers typed out during the execution phase.

Note that problems 1 and 2 are executed in the trace mode (Switch #2 on), causing the result of each statement to be printed (fixed-point statements of the form $I=n$ are not traced).

To improve his understanding of GOTRAN, the reader should establish to his own satisfaction the purpose of each statement used in the following programs (see AN EXAMPLE PROBLEM).

```
C          TEST PROBLEM 1
C    SWITCH TWO MUST BE ON DURING EXECUTION
 12     I = 1
 13     J = 2
 14     J = J + 1
 15     J = J - 1
 16     I = I + 1
 17     I = I - 1
 18     K= J + 2
 19     L= I - 5
 20     PRINT , I, J, K, L
 21     END

 003
 002
 002
 001
 004
 -004
 001          002          004          -004
 END OF PROGRAM
```

```
C          TEST PROBLEM 2
C   SWITCH TWO MUST BE ON DURING EXECUTION
        DIMENSION ROT (4), RRT (4), RC (4), B (4), A (4), RY (4)
  1     CONTINUE
        K= 1
        ROT (K) = 6.0
        RRT (K) = 5.0
        K= 2
        ROT (K) = 7.0
        RRT (K) = 5.0
        K= 3
        ROT (K) = 5.0
        RRT (K) = 3.0
        K= 4
        ROT (K) = 4.0
        RRT (K) = 2.0
        PSEC = 0.0
  2     DO 3  J = 1,4
  3     RC 0.0
        END


6.0000000
5.0000000
7.0000000
5.0000000
5.0000000
3.0000000
4.0000000
2.0000000
0.0
0.0
0.0
0.0
0.0
END OF PROGRAM
```

```
C          TEST PROBLEM 3

           DIMENSION A (25), B (25), C (25), D (25)
           I = 1
           A(I) = 0.0
           PRINT , I, A(I)
           DO 15  I  = 2,25
           J = I - 1
           A(I) = A(J) + 1.0
    15     PRINT , I, A(I)
           DO 2  J = 1,25
           B(J) = LOG(A(J))
     2     PRINT , J, A(J), B(J)
     5     CONTINUE
           DO 3  K = 1,25
           D(K) = A(K) ** 2.0
           C(K) = SQR(A(K))
     3     PRINT , K, A(K), D(K), C(K)
           END

  001              0.0
  002              1.0000000
  003              2.0000000
  004              3.0000000
  005              4.0000000
  006              5.0000000
  007              6.0000000
  008              7.0000000
  009              8.0000000
  010              9.0000000
  011              10.000000
  012              11.000000
  013              12.000000
  014              13.000000
  015              14.000000
  016              15.000000
  017              16.000000
  018              17.000000
  019              18.000000
  020              19.000000
  021              20.000000
  022              21.000000
  023              22.000000
  024              23.000000
  025              24.000000
ERROR F2
  001              0.0              0.0
  002              1.0000000        0.0
  003              2.0000000        .69314718
  004              3.0000000        1.0986122
  005              4.0000000        1.3862943
  006              5.0000000        1.6094379
  007              6.0000000        1.7917594
  008              7.0000000        1.9459101
  009              8.0000000        2.0794415
  010              9.0000000        2.1972245
  011              10.000000        2.3025850
  012              11.000000        2.3978952
  013              12.000000        2.4849066
  014              13.000000        2.5649493
  015              14.000000        2.6390573
  016              15.000000        2.7080502
  017              16.000000        2.7725887
  018              17.000000        2.8332133
  019              18.000000        2.8903717
  020              19.000000        2.9444389
  021              20.000000        2.9957322
  022              21.000000        3.0445224
  023              22.000000        3.0910424
  024              23.000000        3.1354942
  025              24.000000        3.1780538
```

```
TEST PROBLEM 3 (CONT)

001            0.0              0.0              0.0
002            1.0000000        1.0000000        1.0000000
003            2.0000000        4.0000000        1.4142135
004            3.0000000        8.9999983        1.7320507
005            4.0000000        15.999997        1.9999999
006            5.0000000        24.999998        2.2360679
007            6.0000000        35.999994        2.4494896
008            7.0000000        48.999994        2.6457511
009            8.0000000        63.999993        2.8284271
010            9.0000000        80.999987        2.9999999
011            10.000000        99.999978        3.1622774
012            11.000000        120.99997        3.3166246
013            12.000000        143.99998        3.4641015
014            13.000000        168.99997        3.6055513
015            14.000000        195.99998        3.7416574
016            15.000000        224.99999        3.8729832
017            16.000000        255.99998        4.0000000
018            17.000000        288.99996        4.1231057
019            18.000000        323.99996        4.2426407
020            19.000000        360.99993        4.3588989
021            20.000000        399.99993        4.4721357
022            21.000000        440.99996        4.5825755
023            22.000000        483.99994        4.6904155
024            23.000000        528.99997        4.7958314
025            24.000000        575.99995        4.8989793
END OF PROGRAM
```

```
 C     TEST PROBLEM 4
       A = .5
       B = .5
       DO 4 I = 1, 5
       DO 5 J = 1, 5
       AXB = A ** B
       PRINT , I, J, A, B, AXB
 5     A = A + .5
       A = .5
 4     B = B + .5
       END

 001          001        .50000000      .50000000      .70710681
 001          002        1.0000000      .50000000      1.0000000
 001          003        1.5000000      .50000000      1.2247448
 001          004        2.0000000      .50000000      1.4142135
 001          005        2.5000000      .50000000      1.5811388
 002          001        .50000000      1.0000000      .50000002
 002          002        1.0000000      1.0000000      1.0000000
 002          003        1.5000000      1.0000000      1.4999999
 002          004        2.0000000      1.0000000      1.9999999
 002          005        2.5000000      1.0000000      2.4999999
 003          001        .50000000      1.5000000      .35355339
 003          002        1.0000000      1.5000000      1.0000000
 003          003        1.5000000      1.5000000      1.8371172
 003          004        2.0000000      1.5000000      2.8284271
 003          005        2.5000000      1.5000000      3.9528470
 004          001        .50000000      2.0000000      .25000001
 004          002        1.0000000      2.0000000      1.0000000
 004          003        1.5000000      2.0000000      2.2499999
 004          004        2.0000000      2.0000000      4.0000000
 004          005        2.5000000      2.0000000      6.2500001
 005          001        .50000000      2.5000000      .17677670
 005          002        1.0000000      2.5000000      1.0000000
 005          003        1.5000000      2.5000000      2.7556760
 005          004        2.0000000      2.5000000      5.6568545
 005          005        2.5000000      2.5000000      9.8821182
END OF PROGRAM
```

```
C       TEST PROBLEM 5
        X = -20.0
        DO 5 I = 1, 41
        XP = X * X
        XP = 400.0 - XP
        Y = XP / 10.0
        PRINT , X
        PLOT (Y, *)
  5     X = X + 1.0
        END

-20.000000      *
-19.000000         *
-18.000000             *
-17.000000                 *
-16.000000                    *
-15.000000                       *
-14.000000                          *
-13.000000                             *
-12.000000                               *
-11.000000                                 *
-10.000000                                    *
-9.0000000                                     *
-8.0000000                                       *
-7.0000000                                         *
-6.0000000                                          *
-5.0000000                                           *
-4.0000000                                            *
-3.0000000                                             *
-2.0000000                                             *
-1.0000000                                             *
 0.0                                                    *
 1.0000000                                             *
 2.0000000                                             *
 3.0000000                                             *
 4.0000000                                            *
 5.0000000                                           *
 6.0000000                                          *
 7.0000000                                         *
 8.0000000                                       *
 9.0000000                                     *
 10.000000                                    *
 11.000000                                 *
 12.000000                               *
 13.000000                             *
 14.000000                          *
 15.000000                       *
 16.000000                    *
 17.000000                 *
 18.000000             *
 19.000000         *
 20.000000      *
END OF PROGRAM
```

```
C       TEST PROBLEM 7
        DO 3 J = 1, 10
  1     READ, I, A
  2     PRINT, I, A
  3     PUNCH, I, A
        PAUSE
        DO 25 K = 1, 10
        READ, I, A
 25     PRINT, I, A
        END


 001             1.0000000
 002             2.0000000
 003             3.0000000
 004             4.0000000
 005             5.0000000
 006             6.0000000
 007             7.0000000
 008             8.0000000
 009             9.0000000
 010             10.000000
 001             1.0000000
 002             2.0000000
 003             3.0000000
 004             4.0000000
 005             5.0000000
 006             6.0000000
 007             7.0000000
 008             8.0000000
 009             9.0000000
 010             10.000000
END OF PROGRAM
```

```
C       TEST PROBLEM 6
C       TEST OF SUBROUTINES -- SIN, COS, ATN
          TEST = 0.0
          DO 3  J = 0, 12
          A = SIN(TEST)
          B = COS (TEST)
          C = ATN (TEST)
          PRINT, J, TEST, A, B, C
  3       TEST = TEST + .10
          END


 000          0.0            0.0              1.0000000       0.0
 001          .10000000      9.9833414E-2     .99500416       9.9668652E-2
 002          .20000000      .19866932        .98006658       .19739555
 003          .30000000      .29552020        .95533649       .29145680
 004          .40000000      .38941834        .92106100       .38050637
 005          .50000000      .47942554        .87758256       .46364761
 006          .60000000      .56464247        .82533561       .54041950
 007          .70000000      .64421769        .76484218       .61072595
 008          .80000000      .71735608        .69670671       .67474093
 009          .90000000      .78332690        .62160997       .73281509
 010          1.0000000      .84147098        .54030231       .78539816
 011          1.1000000      .89120736        .45359612       .83298127
 012          1.2000000      .93203909        .36235776       .87605805
END OF PROGRAM
```

Index

S046503MFW

C26-5594-1

IBM 1620 Printed in U.S.A. C26-5594-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York

J3