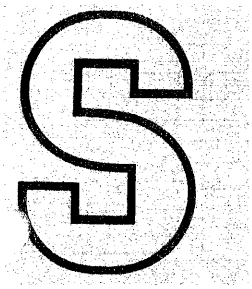


IBM

Reference Manual

650 FORTRAN - Automatic Coding System

for the IBM 650 Data Processing System

IBM

Reference Manual
650 FORTRAN - Automatic Coding System
for the IBM 650 Data Processing System

TABLE OF CONTENTS

INTRODUCTION

Purpose of FORTRAN

FORTRAN, which was originally developed for use on the IBM 704, is a language closely resembling the language of mathematics and is designed primarily for scientific and engineering computation. One of the main purposes of FORTRAN is to provide the engineer or scientist with an efficient means of writing programs requiring a relatively short period of instruction and no detailed knowledge of the computer itself.

650 FORTRAN

650 FORTRAN, an automatic coding system, has been developed to reduce the number of machine passes required to transform FORTRAN statements to IBM 650 machine language, while retaining the optimizing features of a SOAP Assembly Program. This system may be substituted for FOR TRANSIT II (S). However, FORTRAN statements that have been written for the FOR TRANSIT II (S) system may have to be rewritten to conform to the 650 FORTRAN system. (See "Requirements for Changing a Source Program from FOR TRANSIT II (S) to 650 FORTRAN," page 5.)

The 650 FORTRAN system consists of two programs: The compiler - 650 FORTRAN - accepts FORTRAN statements (the source program) and compiles 650 instructions in symbolic (SOAP II*) language.

The assembler - SOAP-PACKAGE - is a modified version of SOAP II which includes certain built-in subroutines. It functions in the following manner:

1. Punches out in five instructions-per-card format, the routine for loading the object program and all the built-in subroutines of the SOAP-PACKAGE deck.

2. Assigns optimum locations to the symbolic instructions generated by the compiler and punches out, in machine language, the object program in five instructions-per-card format.

As illustrated in the overall schematic representation of the 650 FORTRAN system (see Figure 1), two passes on the IBM 650 are required to process a source program from FORTRAN statements to machine language.

---

*SOAP II, Symbolic Optimal Assembly Program for the IBM 650 Data Processing System (see SOAP II Programmer's Reference Manual, form C28-4000).

Machine Requirements

For the compilation and assembly phases of the 650 FORTRAN system, the following equipment is required:

Basic IBM 650
Index Registers
Alphabetic Device
Special Character Device, Group II

In addition, the Floating Point Arithmetic Device is required to run the object program.

Organization of Manual

Programming the IBM 650 using the 650 FORTRAN system requires a knowledge of the FORTRAN language, but little detailed knowledge of 650 machine operations. Accordingly, the first and largest part of this manual is devoted to a description of the FORTRAN language and the rules governing its use in the 650 FORTRAN system. Subsequent sections of the manual deal with the usage of subroutines, operating instructions for the compilation and assembly phases, and information about running the object program.

650 FORTRAN Statements

A source program consists of a sequence of FORTRAN statements. As each statement is read into the 650, the 650 FORTRAN system will analyze and interpret it. Based on the configuration of characters making up the statement, instructions are compiled and assembled into an object program.

The FORTRAN statements which are permissible in writing 650 FORTRAN are:

a = b (Arithmetic Statement)
GO TO n
GO TO $(n_1, n_2, \ldots, n_m)$, i
IF (a) n_1, n_2, n_3
PAUSE or PAUSE n
STOP or STOP n
DO n i = m_1, m_2
DO n i = m_1, m_2, m_3
CONTINUE
READ n, list
PUNCH n, list
DIMENSION $v_1, v_2, v_3, \ldots v_n$
END

As an example of the general appearance and some of the properties of a 650 FORTRAN program, the following brief program is illustrated and explained.

650 FORTRAN SYSTEM

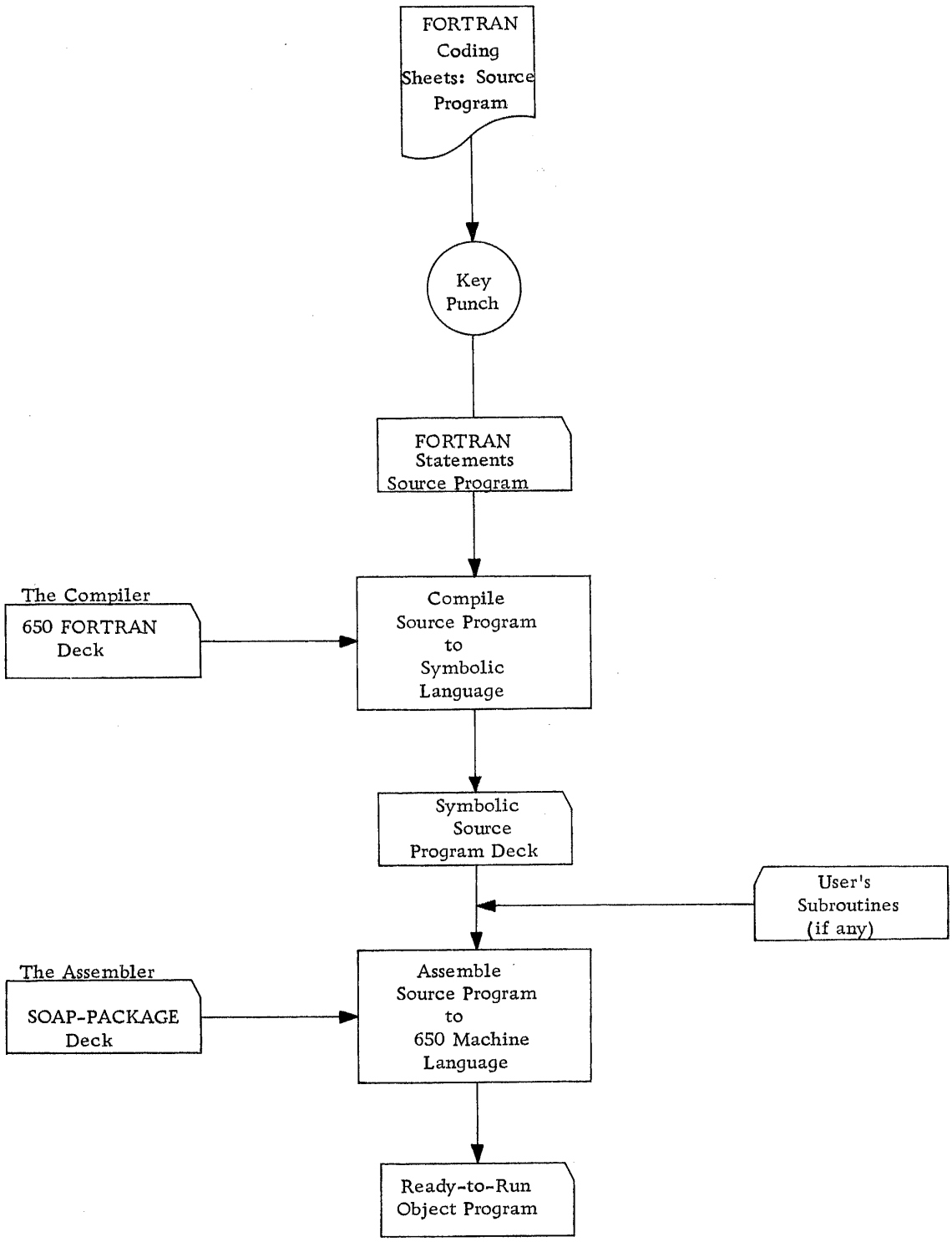


Figure 1

Example of a 650 FORTRAN Program

C ← FOR COMMENT	STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
C	0000	0	PROGRAM FOR FINDING THE LARGEST VALUE
C	0000	0	ATTAINED BY A SET OF NUMBERS
0	0000	0	DIMENSION A (10)
0	0000	0	BIGA = A(1)
0	0000	0	DO 20 I = 2, 10
0	0000	0	IF (BIGA - A(I) 10, 20, 20
0	0010	0	BIGA = A (I)
0	0020	0	CONTINUE
			Note: Card columns 1-6 must not be left blank.
			Punch zeros to fill out these columns.

This program examines the set of n numbers a_i (i = 1,...10) and stores the largest value attained in BIGA. It begins (after a comment describing the program) by replacing BIGA by a_1. Next the DO statement causes the succeeding statements to and including statement 20 to be carried out repeatedly, first with i = 2, then with i = 3, etc., and finally with i = 10. During each repetition of this loop the IF statement compares BIGA with a_i; if BIGA is less than a_i, statement 10, which replaces BIGA by a_i, is executed before continuing.

Compatibility of 650 FORTRAN with 704 FORTRAN

As stated, the FORTRAN system was originally designed for a larger machine than the 650. As a result, only twelve of the thirty-two statements found in the IBM 704 FORTRAN Reference Manual, C28-6003, are present in the 650 FORTRAN system. In addition, certain other restrictions to the FORTRAN language have been added. However, none of these restrictions make a source program written in 650 FORTRAN incompatible with the 704 FORTRAN system. For example, 704 FORTRAN variables can be made up of from one to six characters whereas 650 FORTRAN requires that variables be made up of from one to five characters.

A statement number of all zeros (interpreted by the 650 FORTRAN system as a blank statement number) will be identified by the 704 FORTRAN system as a unique statement number. Therefore, if 650 FORTRAN statements are to be processed on the 704, the statements containing all zeros in columns 2-5 must be repunched with these columns either left blank or assigned a significant unique statement number.

It should be noted, that in a few instances FORTRAN restrictions have been relaxed to take advantage of certain features of the 650; specific information regarding such modifications is included at the applicable places in the following pages for the benefit of users concerned with compatibility.

Additional information concerning the writing of FORTRAN statements may be obtained by referring to the IBM General Information Manual, "Programmer's Primer for FORTRAN, Automatic Coding System for the IBM 704 Data Processing System," F28-6019.

Requirements for Changing a Source Program from FOR TRANSIT II (S) to 650 FORTRAN

In order to make a FORTRAN program written for the FOR TRANSIT II (S) system compatible with the 650 FORTRAN system, the following modifications must be made.

(1) Delete all EQUIVALENCE statements.

(2) Delete all Function Title cards.

(3) Statement cards must not be blank in columns 1-6.

(4) Expressions to the right of the = sign can contain only one mode, fixed or floating point.

(5) The instruction compiled for a STOP statement in FOR TRANSIT II (S) allows continuation of the program following the halt by depressing the Program Start key. In 650 FORTRAN, however, the instruction compiled does not permit continuation of the program and, therefore, it may be necessary to substitute PAUSE statements for STOP statements in FOR TRANSIT II (S) source programs.

(6) The conditional PUNCH statement in FOR TRANSIT II (S), is not recognized as such by 650 FORTRAN. That is, a PUNCH statement which is not numbered will be compiled in 650 FORTRAN as a normal punch statement and subsequent punching in the object program will not be controlled by the sign switch on the console. Therefore, it may be necessary to remove the punch statements that were intended originally to be used only as a diagnostic aid.

Compatibility of 650 FORTRAN with 7070 FORTRAN

When a source program written in 650 FORTRAN is to be processed on an IBM 7070 system using 7070 FORTRAN, certain changes will be required in the input and output statements of the source program.

One of these changes is the addition of FORMAT statements which are required to define input and output data fields.

The other change, which is directly associated with the use of the FORMAT statements, is that the READ and PUNCH statements must specify the FORMAT number to be used with the particular input or output statement.

The use of FORMAT statements in 650 FORTRAN to be processed on the IBM 650 is not allowed.

Program Decks

Requests for program decks for the 650 FORTRAN system should be addressed to:

IBM 650 Program Librarian
International Business Machines Corporation
590 Madison Avenue
New York 22, New York

CHAPTER I — WRITING THE SOURCE PROGRAM

Writing FORTRAN statements requires a knowledge of the following:

1. The components of a FORTRAN statement — i. e., the symbols and characters which may appear within a FORTRAN statement, and the rules that must be followed in using each of these components.

2. The twelve 650 FORTRAN statements — the purpose and function of each FORTRAN statement in the 650 FORTRAN system.

This chapter treats each of the above in detail.

As an example, consider the algebraic formula,

$$ROOT = \left[-B + \sqrt{B^2 \quad -4AC} \right] / 2A$$

As an arithmetic FORTRAN statement, the above algebraic formula would appear as

```
ROOT = (-B+ SQRTF (B**2 - 4. 0*A*C)) / (2. 0*A)
```

The entire arithmetic FORTRAN statement above means, "evaluate the expression on the right side of the equal sign and make this the value of the variable on the left. "

Components of a FORTRAN Statement

By coding each letter, number and symbol in the sample FORTRAN statement, each component can be defined by the corresponding code number as shown in Figure 2.

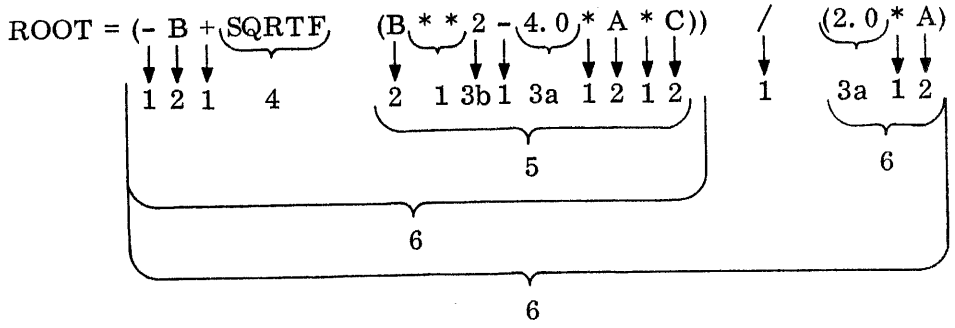
The component parts that make up FORTRAN statements are explained in two groups:

1. Constants, variables and subscripts
2. Functions and expressions.

Constants, Variables and Subscripts

Constants

Two types of constants are permissible: fixed point (numbers written without a decimal point) and floating point (numbers written with a decimal point).



Key	Name of Component
1	OPERATION SYMBOLS: * Denotes Multiplication ** Denotes Exponentiation; e.g., A**3 means A^3 + Denotes Add - Denotes Subtract / Denotes Divide
2	VARIABLES (Floating Point)
3	CONSTANTS: a. Floating Point b. Fixed Point
4	FUNCTION NAME denotes a subroutine which computes the square root of the argument enclosed in parentheses. This particular routine must be incorporated by the user.
5	ARGUMENT of the Function SQRTF
6	EXPRESSION

NOTE: Variables may also be in fixed point even though none appear in this example.

Figure 2

Fixed Point Constants

GENERAL FORM	EXAMPLES
1 to 10 decimal digits. The plus is optional. However, a minus sign must be stated if the constant is to be negative.	3 +1 -28987

NOTE: The magnitude of fixed point constants in the 704 FORTRAN system must be less than 32768. 650 FORTRAN users must comply with this restriction if compatibility is desired.

Floating Point Constants

GENERAL FORM	EXAMPLES
1 to 8 significant digits with a decimal point at the beginning, at the end, or between two digits. A preceding plus is optional. However, a minus sign must be stated if the constant is to be negative. A decimal exponent (a one or two digit fixed point constant) preceded by an E may follow. The exponent may be signed.	17. 5.0 -.0003 5.0E3 (=5.0 x 10^3) 5.0E+3 (=5.0 x 10^3) 5.0E-7 (=5.0 x 10^{-7}) 5.0E13 (=5.0 x 10^{13})

The floating point number will appear in the object program as a normalized single-precision number in the form .XXXXXXXXPP, where PP is a power of 10 with 50 added. The decimal point is assumed to be at the left of the high-order non-zero digit of the number. Thus a floating point number can assume any value from $\pm$ .10000000x10^{-50} to 99999999x10^{49}. The values in the examples above would appear in the machine as follows:

Input Value	Machine Word
17	1700000052
5.0	5000000051
-.0003	-30000000047
5.0E3	5000000054
5.0E+3	5000000054
5.0E-7	5000000044
5.0E13	5000000064

NOTE: The magnitude of floating point constants in the FORTRAN system for the 704 must lie between the limits of 10^{-38} and 10^{38}. 650 FORTRAN users must comply with this restriction if compatibility is desired.

Variables

Two types of variables are also permissible: fixed point (restricted to integral values) and floating point. Fixed point variables are distinguished by the fact that their first character is I, J, K, L, M, or N.

Fixed Point Variables

GENERAL FORM	EXAMPLES
1 to 5 alphabetic or numerical characters (not special characters) of which the first is I, J, K, L, M, or N.	I M2 JOBNO

Floating Point Variables

GENERAL FORM	EXAMPLES
1 to 5 alphabetic or numerical characters (not special characters) of which the first is alphabetic but not I, J, K, L, M, or N.	A B7 DELTA

NOTE: For compatibility with 704 FORTRAN, the name of a variable must not be the same as the name of any function used in the program after the terminal F of the function name has been removed.

Subscripts and Subscripted Variables

A variable can be made to represent any member of a one- or two-dimensional array of quantities by appending to it one or two subscripts; the variable is then a subscripted variable. The subscripts are fixed point quantities whose values determine which member of the array is being referenced.

Subscripts

GENERAL FORM	EXAMPLES
Let v represent any fixed point variable (I in example) and c (or c') any unsigned fixed point constant (numbers 2 and 3, respectively, in example). Then a subscript is an expression of one of the forms v c v + c or v - c c*v c*v+c' or c*v-c'	I 3 I + 2 I - 2 3 * I 3 * I + 2 3 * I - 2

The variable v must not itself be subscripted.

Subscripted Variables

GENERAL FORM	EXAMPLES
A fixed or floating point variable followed by parentheses enclosing one or two subscripts separated by commas.	A (I) K(3) BETA (5*J-2, K+2)

NOTE: 1. Because a variable ending with the letter F to the immediate left of an open parenthesis is identified as a function, no subscripted variable may end in F.

2. A maximum of 20 subscripted variables may be used in any one program. However, there is no limit on the number of non-subscripted variables that may be used.

For each variable that appears in subscripted form, the size of the array, i.e., the maximum values which its subscripts can attain, must be stated in a DIMENSION statement (page 25) preceding the first appearance of the variable.

The minimum value which a subscript may assume in the object program is +1.

NOTE: A two-dimensional array A will, in the object program, be stored sequentially in the order $A_{1,1}$, $A_{2,1}$, . . . , $A_{m,1}$, $A_{1,2}$, $A_{2,2}$, . . . , $A_{m,2}$, . . . , $A_{m,n}$. Thus it is stored "columnwise," with the first of its subscripts varying more rapidly. One-dimensional arrays are of course stored sequentially.

Functions and Expressions

Of the twelve 650 FORTRAN statements, it is the arithmetic formula which defines a numerical calculation that the object program is to do. A FORTRAN arithmetic formula resembles a conventional arithmetic formula. It consists of the variable to be computed, followed by an = sign, followed by an arithmetic expression.

For example, the arithmetic formula,

```
Y = A - SINF (B-C)
```

means "replace the value of y by the value of a-sin(b-c)."

Functions

As in the above example, a FORTRAN expression may include the name of a function (e.g., the sine function SINF), provided the routine for evaluating the function is available to the 650 FORTRAN system. These routines can be either SOAP-PACKAGE (built-in) subroutines or subroutines added by the user.

GENERAL FORM	EXAMPLES
The name of the function is 4 or 5 alphabetic or numerical characters (not special characters), of which the last must be F and the first must be X if and only if the value of the function is to be fixed point. The name of the function is followed by parentheses enclosing the arguments (which may be expressions).	SINF (A+B) SQRTF (SINF(A)) XABSF (3.*X)

Expressions

An expression is any sequence of constants, variables (subscripted or not subscripted), and functions, separated by operation symbols, commas, and parentheses so as to form a meaningful mathematical expression.

However, one special restriction does exist. A FORTRAN expression may be either a fixed or a floating point expression, but it must not be a mixed expression. This does not means that a floating point quantity cannot appear in a fixed point expression, or vice versa, but rather that a quantity of one mode can appear in an expression of the other mode only in certain ways. Briefly, a floating point quantity can appear in a fixed point expression only as an argument of a function; a fixed point quantity can appear in a floating point expression only as an argument of a function, as a subscript, or as an exponent.

Rules for Forming Expressions

By repeated use of the following rules, all permissible expressions may be derived.

1. Any fixed point (floating point) constant, variable, or subscripted variable is an expression of the same mode. Thus 3 and I are fixed point expressions, and ALPHA and A(I, J) are floating point expressions.

2. If SOMEF is some function of n variables, and if E, F, . . . , H are a set of n expressions of the correct modes for SOMEF, then SOMEF (E, F, . . . , H) is an expression of the same mode as SOMEF.

3. Two operation symbols may not appear in sequence. If E is an expression, and if its first character is not + or -, then +E and -E are expressions of the same mode as E. Thus -A is an expression, but +-A is not.

4. If E is an expression, then (E) is an expression of the same mode as E. Thus (A), ((A)), (((A))), etc. are expressions.

5. If E and F are expressions of the same mode, and if the first character of F is not + or -, then

   E + F
   E - F
   E * F
   E / F

   are expressions of the same mode. Thus A-+B and A/+B are not expressions.

6. If E and F are expressions, and F is not floating point unless E is too, and the first character of F is not + or -, and neither E nor F is of the form A **B, then

   E ** F

   is an expression of the same mode as E. Thus A**(B**C) is an expression, but I**(B**C) and A**B**C is not.

   Similarly in the case of consecutive divisions, the order of operations must be specified by appropriate use of parentheses.

   Thus A/B/C must be written as (A/B) /C or A/(B/C) whichever is intended.

Hierarchy of Operations

When the hierarchy of operations in an expression is not explicitly specified by the use of parentheses, it is processed by FORTRAN in the following order (moving from innermost operations to outermost).

Exponentiation, then;
Multiplication and Division, then;
Addition and Subtraction.

For example, the expression

A+B/C+D**E*F-G

will be taken to mean

$A+(B/C)+(D^{E}*F)$ -G

When the sequence of consecutive operations of the same hierarchal level (e.g., consecutive multiplications) is not completely specified by parentheses, the order of operations is assumed to be from left to right.

Verification of Correct Use of Parentheses

The following procedure is suggested for checking that the parentheses in a complicated expression correctly express the desired operations.

Label the first open parenthesis "1"; thereafter, working from left to right, increase the label by 1 for each open parenthesis and

decrease it by 1 for each closed parenthesis. The label of the last parenthesis should be 0; the mate of an open parenthesis labeled n will be the next parenthesis labeled n-1.

The maximum number of pairs of parentheses that may appear in any one arithmetic expression is 25.

Preparation of a FORTRAN Statement

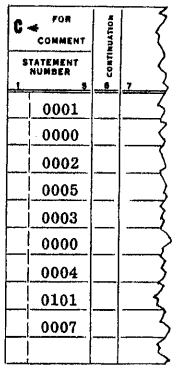
Certain limitations and precautions must be observed in preparing a FORTRAN statement.

Because 650 FORTRAN contains no program error-detection tests, special care must be taken in writing the statements in the form the user wishes the program to function. However, there are utility programs available such as FORSCAN*, which edit each source program statement and determine whether it has been correctly written. The FORSCAN program may be obtained from the IBM 650 Program Librarian (see page 6 for address).

Rules for Statements and Statement Numbers

The 650 FORTRAN program will accept valid statements of up to a maximum of 125 characters exclusive of blanks. Statements need not be in any numerical order nor do all statements need statement numbers. However, cross-referencing within a program is accomplished by giving statement numbers to those statements referred to by other statements.

Statement numbers can be any unique unsigned fixed point constants, from 0001 to 9999 and may appear in any sequence. Thus the numbering of statements as shown below would be acceptable in a program.



*Contributed by Messrs. C. A. Irvine & M. A. Smith, Continental Oil Company, Ponca City, Oklahoma.

Arithmetic Formula

GENERAL FORM	EXAMPLES
"a=b" where a is a variable (subscripted or non-subscripted) and b is an expression.	A(I) = B(I) + SINF(C(I))

The = sign in an arithmetic formula has the meaning "is to be replaced by." An arithmetic formula is therefore a command to compute the value of the right-hand side and to store that value in the storage location designated by the left-hand side.

The result will be stored in fixed or floating point form according as the variable on the left-hand side is a fixed or floating point variable.

If the variable on the left is fixed point and the expression on the right is floating point, the result will first be computed in floating point and then truncated and converted to a fixed point integer. Thus, if the result is $\pm$ 3.569, the fixed point number stored will be $\pm$ 3, not $\pm$ 4.

No arithmetic statement may contain more than nine different constants. Two floating point numbers having the same equivalent value are not considered "different." For example, 4.0 and .4E1 have the same equivalent value and thus would not be considered different. Signs are not considered in determining difference.

Examples of Arithmetic Formulas

FORMULA	MEANING
A = B	Store the value of B in A.
I = B	Truncate B to an integer, convert to fixed point, and store in I.
A = I	Convert I to floating point and store in A.
I = I + 1	Add 1 to I and store in I. This example illustrates the point that an arithmetic formula is not an equation but a command to replace a value.
A = 3.0*B	Replace A by 3B.

Control Statements

The second class of FORTRAN statements is the set of seven control statements, which enable the programmer to state the flow of his program.

Unconditional GO TO

GENERAL FORM	EXAMPLES
"GO TO n" where n is a statement number.	GO TO 3

This statement causes transfer of control to the statement with statement number n.

Computed GO TO

GENERAL FORM	EXAMPLES
"GO TO $(n_1, n_2, \ldots, n_m)$, i" where $n_1, n_2, \ldots, n_m$ are statement numbers and i is a non-subscripted fixed point variable.	GO TO (30, 40, 50, 60), I

If at the time of execution the value of the variable i is j, then control is transferred to the statement with statement number n_j. Thus, in the example, if I has the value 3 at the time of execution, a transfer to statement 50 will occur.

This statement is used to obtain a computed many-way fork. A maximum of 25 branches may be used in any one of these statements.

IF

GENERAL FORM	EXAMPLES
"IF (a) n_1, n_2, n_3" where a is any expression and n_1, n_2, n_3 are statement numbers.	IF (A(J, K)-B)10, 20, 30

Control is transferred to the statement with statement number n_1, n_2, or n_3 according as the value of the expression a is less than, equal to, or greater than zero. In the example, control will be transferred to statement 10, 20 or 30 according as the value of the expression, (A(J,K)-B), is less than, equal to, or greater than zero.

PAUSE

GENERAL FORM	EXAMPLES
"PAUSE" or "PAUSE n" where n is any unsigned fixed point constant less than or equal to 1999.	PAUSE PAUSE 1234

A PAUSE statement compiles as a stop command. During execution of the object program, the machine will halt with the number n shown in the console address lights. (If n is not stated, it is taken to be zero.) A subsequent depression of the Program Start key causes the program to resume at the point in the object program corresponding to the next FORTRAN statement.

STOP

GENERAL FORM	EXAMPLES
"STOP" or "STOP n" where n is any unsigned fixed point constant less than or equal to 9999.	STOP STOP 1234

A STOP statement compiles as a stop command. During execution of the object program, the machine will halt in such a way that pressing the Program Start key will have no effect. Therefore, in contrast to the PAUSE, it is used where a get-off-the-machine stop, rather than a temporary stop, is desired. The number n is shown in the address field of the console display lights. (If n is not stated, it is taken to be zero.)

DO

GENERAL FORM	EXAMPLES
"DO n i = m_1, m_2" or "DO n i = m_1, m_2, m_3" where n is a statement number, i is a non-subscripted fixed point variable, and m_1, m_2, m_3 are each either an unsigned fixed point constant or a non-subscripted fixed point variable. If m_3 is not stated it is taken to be 1. NOTE: If m_1, m_2, m_3 are constants, they may be no more than four digits long.	DO 30 I = 1, 10 DO 30 I = 1, M, 3

The DO statement is a command to execute repeatedly the statements which follow, up to and including the statement with statement number n. The first time the statements are executed with i = m_1. For each succeeding execution i is increased by m_3. After they have been executed with i equal to the highest of this sequence of values which does not exceed m_2, control passes to the statement following the last statement in the range of the DO.

Example of DO

Suppose, for example, that control has reached statement 10 of the program

10	DO 11 I = 1, 10
11	A (I) = I*N(I)
12	

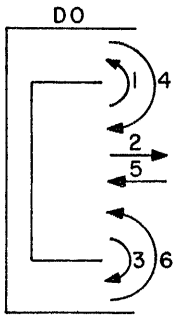
The range of the DO is statement 11, and the index is I. The DO sets I to 1 and control passes into the range. 1N(1) is computed, converted to floating point, and stored in A(1). Now, since statement 11 is the last statement in the range of the DO and the DO is unsatisfied, I is increased to 2 and control returns to the beginning of the range, statement 11. 2N(2) is computed and stored in A(2). This continues until statement 11 has been executed with I = 10. Since the DO is satisfied, control now passes to statement 12.

DOs within DOs — Among the statements in the range of a DO may be other DO statements. When this is so, the following rule must be observed:

Rule: If the range of a DO includes another DO, then all of the statements in the range of the latter must also be in the range of the former. A set of DOs satisfying this rule is called a nest of DOs. A nest must not exceed a depth of four DOs.

Transfer of Control and DOs — Transfers of control by IF-type or GO TO-type statements are subject to the following rule:

Rule: No transfer is permitted into the range of any DO from outside its range. Thus, in the configuration below 1, 2 and 3 are permitted transfers, but 4, 5 and 6 are not.



EXCEPTION — There is one situation in which control can be transferred into the range of a DO from outside its range. Suppose control is somewhere in the range of one or more DOs, and that it is transferred to a section of a program, completely outside the nest to which the DOs belong, which makes no change in any of the indices or indexing parameters (m's) in the nest. Then after the execution of this section of program, control can be transferred back to the "same part of the nest" from which it originally came. (By "same part of the nest" is meant that no DO, and no statement which is the last statement in the range of a DO, shall lie between the exit point and re-entry point.) This provision makes it possible to exit temporarily from the range of a DO to execute a subroutine.

Restriction on Calculations in the Range of a DO — Only one type of statement is not permitted within the range of a DO loop, namely any statement which redefines the value of the index or of any of the indexing parameters (m's). In other words, the indexing of a DO loop must be completely set before the range is entered.

The first statement in the range of a DO must be executable.

CONTINUE

GENERAL FORM	EXAMPLES
"CONTINUE"	CONTINUE

CONTINUE is a dummy statement and provides no instructions in the object program. A frequent use of it is as the last statement in the range of a DO to provide a transfer address for IF and GO TO statements. As an example of a program requiring a CONTINUE statement, consider the table search program:

```
10    DO 12 I = 1, 100
11    IF (ARG - VALUE(I)) 12, 20, 12
12    CONTINUE
13
```

This program will examine the 100-entry VALUE table until it finds an entry equal to ARG. As long as an equal entry is not found, statement 11 (IF) will transfer to statement 12 (CONTINUE). Statement 12 will in turn cause the DO loop to be repeated. When the equal entry is found, the program will exit to statement 20 with the successful value of I available for fixed point use. However, if no entry in the table equals ARG, an exit to statement 13 will occur. The program

```
10    DO 11 I = 1, 100
11    IF(ARG - VALUE(I)) 11, 20, 11
12
```

would not work since, as stated in the next section, DO sequencing does not occur if the last statement in the range of a DO is a transfer.

Summary of FORTRAN Sequencing — The precise laws which govern the order in which the statements of a FORTRAN program will be executed, and which have been left unstated up to this point, may be stated as follows:

1. Control begins at the first executable statement.

2. If control is at statement S, then control will next go to the statement dictated by the normal sequencing properties of S.

3. EXCEPTION. If, however, S is the last statement in the range of one or more DOs which are not yet satisfied, and if S is not a transfer (IF or GO TO statement), then the normal sequencing of S is ignored and DO-sequencing occurs, i.e., control will next go to the first statement of the range of the nearest of the unsatisfied DOs, and the index of that DO will be raised.

4. The statement DIMENSION, which is discussed in this chapter is a non-executable statement, and in any question of sequencing is simply to be ignored.

Input-Output Statements

The 650 FORTRAN system provides for input and output of data by means of punched cards using the FORTRAN statements READ and PUNCH.

From one to seven ten-digit words can be read or punched on a single card starting at column 1 and ending with column 70. Reading or punching of data will begin at the left and continue under the control of the READ or PUNCH statement until all the seven words of data have been processed.

As many cards as necessary can be read or punched providing there is no break in the data being processed.

The last data card can contain from one to seven ten-digit words. If less than seven words are required, the remaining word(s) may be left blank.

NOTE: A READ statement calling for only five words of data will ignore the remaining last two words at the right side of the card.

READ

GENERAL FORM	EXAMPLES
"READ, LIST" or "READ n, LIST" where n may be a 1-4 digit fixed point constant and LIST is as described below. NOTE: The comma placed after "READ" and "READ n" is absolutely necessary for the operation of the program and must never be omitted.	READ, A, B, C READ 1, A, B, C READ 52, X, Y

The READ statement causes the object program to read card after card until the entire list has been brought in and stored. The n portion of the READ statement is optional but must be included if compatibility with the FORTRAN systems for the 704 and 7070 is desired.

PUNCH

GENERAL FORM	EXAMPLES
"PUNCH, LIST" or "PUNCH n, LIST" where n may be a 1-4 digit fixed point constant and LIST is as described below. NOTE: The comma placed after "PUNCH" and "PUNCH n" is absolutely necessary for the operation of the program and must never be omitted.	PUNCH, ROOT1, ROOT2 PUNCH 1, ROOT PUNCH 32,ARRAY PUNCH 1, ELMNT (2, 5)

The PUNCH statement causes the object program to punch card after card until the entire list has been punched. The n portion of the PUNCH statement is optional, but must be included if compatibility with the FORTRAN systems for the 704 and 7070 is required.

LIST — Both the READ and PUNCH statements call for the transmission of information and include a list of the quantities to be transmitted. The list is ordered, and its order must be the same as the order in which the words of information exist (for input), or will exist (for output), in the cards. Below are the various forms a list may take and the types of variables that may be placed in a list. (Only variables, and not constants, may be listed.)

VARIABLE	EXPLANATION
	Non-subscripted Variables
READ, A	The name of a non-subscripted variable.
	Entire Arrays
PUNCH, B	An entire array specified by giving only the name of the array.

VARIABLE	EXPLANATION
PUNCH, B (Con't)	(The size of the array previously must have been given in a DIMENSION statement.) The array will be punched (or read) column-wise in its natural order.
	Single Elements of Arrays
READ, C(2)	A single element of a one-dimensional array, with the subscript in absolute form.
PUNCH, D(1, 3)	A single element of a two-dimensional array, with both subscripts in absolute form.
READ, E(I)	A single element of a one-dimensional array, with the subscript in variable form.
PUNCH, F(I, 5)	A single element of a two-dimensional array, with one subscript in variable form and the other subscript in absolute form.
PUNCH, H(I, K)	A single element of a two-dimensional array, with both subscripts in variable form.

Groups of Elements of Arrays Using Indexing

READ, $(P(I), I = m_1, m_2, m_3)$

A group of elements of a one-dimensional array using a variable subscript with a set of values to indicate the particular elements desired. This specifies every m_3th element of the P array beginning with m_1 and not exceeding m_2.

READ, $(Q(I, 10), I = m_1, m_2, m_3)$ or
READ, $(R(2, J), J = m_1, m_2, m_3)$

A group of elements of a two-dimensional array with one subscript in absolute form and the other in variable form with a set of values. The absolute subscript indicates

the column (or row) of the elements; the variable subscript, with its set of values, specifies the elements within the column (or row).

PUNCH, (S(K, L), K=m_1, m_2, m_3) or
READ, (T(I, L), L=m_1, m_2, m_3)

A group of elements of a two-dimensional array with both subscripts in variable form and a set of values for one subscript. The variable without a set of values indicates the column (or row) of the elements; the subscript with the set of values specifies the elements within the column (or row).

PUNCH, ((V(J, K), J=m_1, m_2, m_3), K=m_1, m_2, m_3)

A group of elements of a two-dimensional array with both subscripts in variable form and a set of values for each subscript.

NOTE: In "Groups of Elements of Arrays Using Indexing," m_1 is the starting value of a subscript, m_2 is the highest value of the subscript, and m_3 is the amount m_1 is increased each time until m_2 is reached. m_1, m_2, m_3 are each either an unsigned fixed point constant or a non-subscripted fixed point variable. If m_3 is not stated, it is taken to be 1.

There is no limit as to the number of variables that may be in a list, but adjacent variables in the list must be separated by a comma.

More than one variable per list may be indexed; however, each indexed variable must have its own indexing parameters (m's).

Specification Statements

DIMENSION

GENERAL FORM	EXAMPLES
"DIMENSION v_1, v_2, v_3, ... v_n" where each v_n is a variable subscripted with 1 or 2 unsigned fixed point constants. Any number of v_n's may be given.	DIMENSION A(10), B(5, 15), C(3, 4)

The DIMENSION statement provides the information necessary to allocate storage in the object program for arrays of quantities.

Every variable which appears in a source program in subscripted form must appear in a DIMENSION statement, and the DIMENSION statement must precede the first appearance of the variable. In the DIMENSION statement are given the dimensions of the array; in the executed program any subscripted variable referring to the array must never take on values larger than those dimensions.

Thus the example states that B is a two-dimensional array and that the subscripts of B will never exceed 5 and 15; it causes 75 words of storage to be set aside for the B array.

A single DIMENSION statement may be used to dimension any number of arrays. However, no one dimension given in the statement may be greater than three digits. For example, D(3001) is not permitted because the dimension is more than three digits.

WARNING: No error checking is done for incorrectly written statements. The foregoing rules must be observed exactly. Punctuation marks, parentheses, etc. must never be omitted.

END

GENERAL FORM	EXAMPLES
"END"	END

An END statement is required as the last statement of a source program.

An end of job halt 01 0000 8000 is always compiled from the END statement, and therefore END should never be the last statement in a DO loop.

Transfers to the END statement should be made only if it is to be an end-of-job indication.

Summary of Limitations for Writing Source Programs

Limitations have been given throughout this chapter and are listed below for review and reference.

Condition	Limits
Statement length (characters)	125 exclusive of blanks
Maximum number of decimal digits in a fixed point constant	10
Maximum number of decimal digits in a floating point constant	8
Maximum number of fixed and floating point constants in any one statement	9
Maximum number of characters for a fixed point variable	5 alphabetic or numerical (not special) characters of which the first is I, J, K, L, M or N
Maximum number of characters for a floating point variable	5 alphabetic or numerical (not special) characters of which the first is alphabetic but is not I, J, K, L, M or N
Maximum number of subscripted variables in one program	20 (none may end in F)
Minimum value of a subscript	+1
Mode	No mixed modes (see page 13)
Maximum number of branches in a computed GO TO statement	25
DO statements	Indexing parameters (m's) may not exceed four digits if they are constants
Number of DOs within DOs	Nests may not exceed depth of four DOs
DIMENSION statement	No one dimension may be greater than three digits
Maximum number of parentheses in one statement	25 pairs
Number of arguments for a built-in subroutine	1

CHAPTER II — USAGE OF SUBROUTINES

650 FORTRAN Subroutines

The Assembly phase of the 650 FORTRAN system contains several built-in routines for evaluating functions. These subroutines may be called on by the user's program. Listed below are the eight built-in subroutines which may be used:

Subrt. Name	Purpose of the Subroutines	Mode Argument	Mode Function
ABSF	Taking the absolute value of a floating point number	Floating	Floating
XABSF	Taking the absolute value of a fixed point number	Fixed	Fixed
XFIXF	Fixing floating point numbers	Floating	Fixed
FLOTF	Floating fixed point numbers	Fixed	Floating
LOGF	$Log_{10}x$	Floating	Floating
EXPF	Raising 10 to a floating point power (Antilogarithm)	Floating	Floating
LOGEF	$Log_{e}x$	Floating	Floating
EXPEF	Raising e to a floating point power (Exponential)	Floating	Floating

WARNING: The user must not use any of the built-in subroutine names for his own subroutines.

Because of non-standard entry conditions, the following built-in subroutines are for internal use only and are not available to the user:

XPOWF	POW2F	PNCHF
POWIF	READF	

Removing 650 FORTRAN Subroutines

By using a SOAP II "BLA" block availability card, locations of some built-in subroutines that are not used by the object program (see table on the following page) can be made available during the assembly phase of 650 FORTRAN. The procedure required to delete these unused built-in subroutines is as follows: (1) prepare a BLA card with the locations used by the subroutine(s) to be

dropped (see table below), (2) place the BLA card(s) in front of first user's subroutine, or if no subroutines have been added by the user, in front of the compiled program.

Explanation	Subrt. Name	Consecutive Locations Used
No exponentiation FLOAT ^{FIX}	POW2F	0040 - 0056
No exponentiation FIX ^{FIX}	XPOWF	0057 - 0071
Neither FLOAT ^{FIX} nor FIX ^{FIX}	POW2F XPOWF	0040 - 0087
No $Log_e x$	LOGEF	1996 - 1998
No e^x	EXPEF	1993 - 1995
No exponentiation FLOAT ^{FLOAT}	POW1F	1831 - 1839
No { Exponentiation FLOAT ^{FLOAT} $Log_{10} x$ $Log_e x$	POW1F LOGF LOGEF	1831 - 1890
No { Exponentiation FLOAT ^{FLOAT} 10^x e^x	POW1F EXPF EXPEF	1831 - 1839
No { Exponentiation FLOAT ^{FLOAT} $Log_{10} x$ $Log_e x$ 10^x e^x	POW1F LOGF LOGEF EXPF EXPEF	1831 - 1945

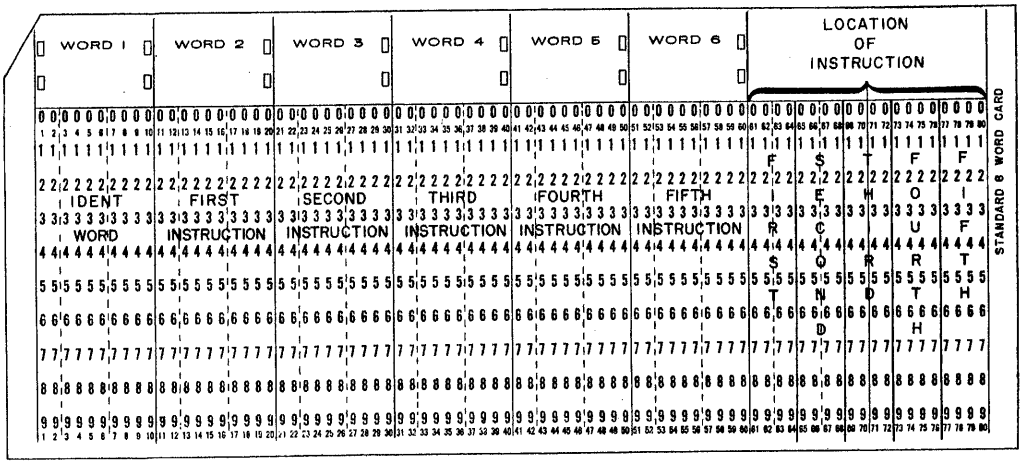
Adding Function Subroutines

Any subroutines required for evaluating functions must be incorporated into the system by the user. Any number of function subroutines (limited only by storage capacity) may be used in any one program.

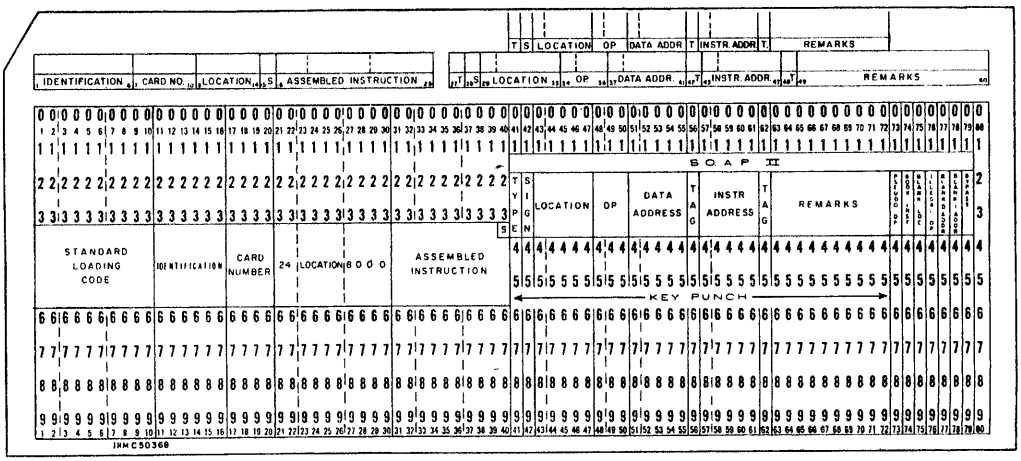
Subroutines added to the system by the user for the purpose of evaluating functions operate in a manner similar to that of the built-in subroutines. When a function name is encountered in a FORTRAN program, a symbolic entry to the subroutine is compiled and the arguments of the function are stored in locations allocated for temporary storage.

The function subroutines must be prepared in either

Five-instructions-per-card Absolute:



or SOAP II Symbolic:



and must always be entered into the system during the assembly phase of the operation (i. e. , on the second pass). A symbolic subroutine name, encountered more than once during the assembly phase, will always be assigned the same absolute drum address.

Subroutines in Absolute Five-instructions-per-card Format

If the five-instructions-per-card format is to be used, a SOAP II synonym card (equating the symbolic name of the subroutine with its starting drum address) must be prepared for each subroutine to be added. The synonym cards should follow the last five-instructions-per-card subroutine when entering the subroutines into the system.

The subroutines themselves may be located on the drum immediately following the area used for the table of subscripted variables (see page 11). To obtain the starting address of the first subroutine, count the number of locations reserved for subscripted variables in the FORTRAN program, and then add 101 to this number. For example, suppose that in a FORTRAN program, there is a DIMENSION statement containing these subscripted variables: A(10), B(5,15), C(3,4). In this example, the subscripted variable table will be 97 locations long. Adding 101 to this figure, a total of 198 is obtained. The first subroutine may start in this location (0198). The second subroutine may start in the location immediately following the last instruction of the first subroutine, etc. See "Determining Available Drum Locations" below, for information regarding reservation of locations by the compiler.

The five-instructions-per-card subroutines are read into the system as load cards and require a "12" punch in column 2.

Subroutines in SOAP II Format

If the SOAP II format is to be used, no synonym cards are needed for the subroutines. The assembly phase will assign available locations.

Subroutines incorporated in the 650 FORTRAN system in symbolic (SOAP II) format require a "12" punch in card column 5 for correct read-in of the cards. Card columns 7-36 and columns 73-75 must be left blank.

Determining Available Drum Locations

The compiling phase of the 650 FORTRAN system sets up a table of subscripted variables starting at location 0101 and continuing for as many locations as necessary. A second table for computed GO TO statements starts with location 1710, and sets aside, in descending sequence, as many locations as required. The final output of the compiler is a block reservation (BLR) card for each table making all these locations unavailable to the assembly program. The remaining locations between these two tables are available for all function subroutines added by the user. The subroutines may be in five-instructions-per-card or symbolic format.

NOTE: Overlapping of the subscripted variable and the computed GO TO tables will not cause the program to stop compiling. However, each of the two BLR cards for the tables will show an overlapping of the locations reserved. Subsequent assembly would result in a packed drum error halt.

When adding symbolic format subroutines to the system, block reservation cards must be placed in front of the added symbolic subroutines.

It is possible that the FORTRAN program will have no computed GO TO statements or subscripted variables, and therefore, no block reservation cards. Obviously, drum locations not used by the subscripted variable or computed GO TO statement tables are available to the assembly program.

NOTE: When added subroutines are in both five-instructions-per-card and SOAP formats, the following sequence will prevail:

1. SOAP-PACKAGE assembly deck
2. Five-instructions-per-card format subroutines
3. SOAP symbolic format subroutines.

Determining the Input Parameters of Subroutines

To determine the input parameters of a subroutine, let the subroutine be a function of "k" variables in the order: V_1, V_2, ... V_k; where V_n is an expression, variable or constant.

The entry conditions are as follows:

1. V_1 is stored by the compiler in symbolic location @1; V_2 is stored by the compiler in @2; etc.

2. The exit instruction is in the distributor.

The exit conditions are as follows:

1. If the subroutine is in fixed point, the result must be placed in the lower accumulator.

2. If the subroutine is in floating point, the result must be placed in the upper accumulator.

NOTE: (1) @1 has been assigned the address 0039. @2, @3, etc., do not have specific addresses. Subroutines added by the user in SOAP II format need only refer to these storage locations by their symbolic names. However, the absolute addresses of these locations must be used in function subroutines added in five-instructions-per-card absolute format. In this case, the user must assign available addresses to these "@" locations by preparing a synonym card (with the symbolic name of the location and the proper absolute address) for each. Locations following those used for the last added function subroutine and before the starting location of the computed GO TO statement table are available for the "@" locations.

(2) When adding function subroutines in five-instructions-per-card format, the last card may contain fewer than five instructions. The user should punch zeros in the unused instruction fields. The corresponding address fields should be punched so as to load these fields into the read band or any temporary storage location. (Location 0000 is suggested for this purpose.)

User's Subroutines Requiring Index Registers

The user may wish to add to the 650 FORTRAN system function subroutines which use index registers. In such a case, provision must be made (at the beginning of the user's subroutine) to save the existing information in the index registers and to restore the original contents after the completion of the subroutine. The following built-in subroutines may be called in to accomplish this:

@4001 - stores the contents of index registers A, B and C in temporary storage locations @3003, @3004 and @3005 respectively.

@4002 - resets index registers A, B and C from temporary storage locations @3003, @3004 and @3005 respectively.

Instructions in the user's subroutine

1. The first instruction should store the exit instruction (which is in the distributor) in symbolic location @3001.

2. The second instruction should load the distributor with the next instruction and then transfer to symbolic location @4001.

3. The exit instruction from the user's subroutine should transfer to symbolic location @4002. (Subroutine @4002 will reset the index registers and then transfer to location @3001 which was set to the return instruction to the main program.)

Available Temporary Storage and Constants

The following temporary storage locations are available for user's subroutines:

Symbolic Address	Actual Address
@3001	0000
@3002	1950
@3003	1990
@3004	1991
@3005	1992
@3006	1989
@3007	1976
@3008	1987
@3009	1988

WARNING: If index registers are needed in the user's subroutine, symbolic locations @3003, @3004, @3005, and @3006 are not available.

The constants listed below are used in the 650 FORTRAN built-in subroutines, and are available for the user's function subroutines.

Symbolic Address	Contents	Absolute Address
@2001	10 0000 0051	0089
@2002	50 0000 0000	0090
@2003	00 0000 0060	0091
@2004	10 0000 0000	0092
@2005	51 0000 0000	0093
@2006	52 0000 0000	0094
@2007	00 0001 0000	0095
@2008	43 4294 4850	0096
@2009	99 9999 9999	0097
@2010	00 0000 0051	0098
@2011	00 0000 0001	0099

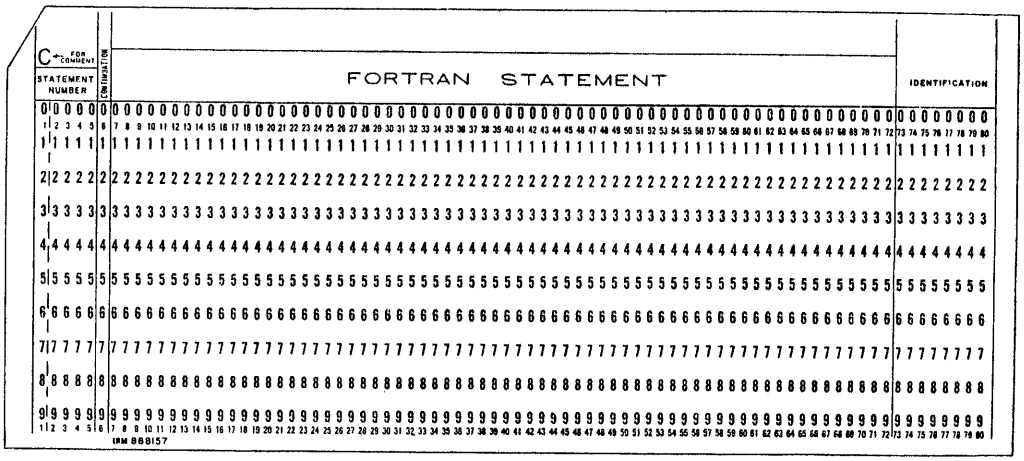
CHAPTER III — PROCESSING THE SOURCE PROGRAM

This chapter includes the necessary information and instructions for processing a 650 FORTRAN source program to obtain a 650 machine language object program. The first section of the chapter deals with the preparation of statement cards, and subsequent sections constitute operator's instructions and notes for each of the two phases of the 650 FORTRAN system.

Source programs stated in the FORTRAN language may be written on standard FORTRAN coding sheets, IBM form X28-7327, as illustrated on page 4. The use of the coding forms is encouraged to avoid programming errors and to facilitate the transcription of the FORTRAN statements to cards. These forms may be obtained through local IBM sales representatives.

Preparing the Statement Cards

650 FORTRAN statements are punched in FORTRAN statement cards (IBM 888157), as shown below, using an IBM Card Punch.



Each FORTRAN statement is punched on a separate card using the FORTRAN characters, as follows:

TABLE OF FORTRAN CHARACTERS

Char	Punch	650	Char	Punch	650	Char	Punch	650	Char	Punch	650
1	1	91	A	12-1	61	J	11-1	71	/	0-1	31
2	2	92	B	12-2	62	K	11-2	72	S	0-2	82
3	3	93	C	12-3	63	L	11-3	73	T	0-3	83
4	4	94	D	12-4	64	M	11-4	74	U	0-4	84
5	5	95	E	12-5	65	N	11-5	75	V	0-5	85
6	6	96	F	12-6	66	O	11-6	76	W	0-6	86
7	7	97	G	12-7	67	P	11-7	77	X	0-7	87
8	8	98	H	12-8	68	Q	11-8	78	Y	0-8	88
9	9	99	I	12-9	69	R	11-9	79	Z	0-9	89
Blank		00	+	12	20	-	11	30	0	0	90
=	8-3	48	.	12-3-8	18	-	4-8	49	,	0-3-8	38
			)	12-4-8	19	*	11-4-8	29	(	0-4-8	39

NOTE: On the 24 and 26 Card Punch Machines equipped for special character punching, the character ⌑ is the equivalent of the character) ; % is the equivalent of (; & is the equivalent of + ; and # is the equivalent of =. If desired, the 24 and 26 machines may be modified on an RPQ basis (Request Price Quotation) to include the "FORTRAN key tops and printing code plate." This includes @ equivalent to "—".

If a statement is too long to fit in the statement field of a single card, it may be continued over as many additional (continuation) cards as necessary until the maximum statement length of 125 characters, (exclusive of blanks) is reached.

When continuation cards are used, the statement number must be carried forward in columns 2-5. A digit 1-9 must be placed in column 6 of each continuation card following the first card of the statement. This digit can be used in numbering the continuation cards of the statement. For example, if the formula

ROOT=(-B+SQRTF(B**2-4. 0*A*C)) /(2. 0*A)

were to use continuation cards, it could appear in three cards in the following manner:

C ← FOR COMMENT	STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
0	0 1 0	0	ROOT =
0	0 1 0	1	(- B + SQRTF (B**2-4. 0*A*C))
0	0 1 0	2	/ (2. 0*A)

Numerical, alphabetic and special characters, and blanks may be included in the FORTRAN statement.

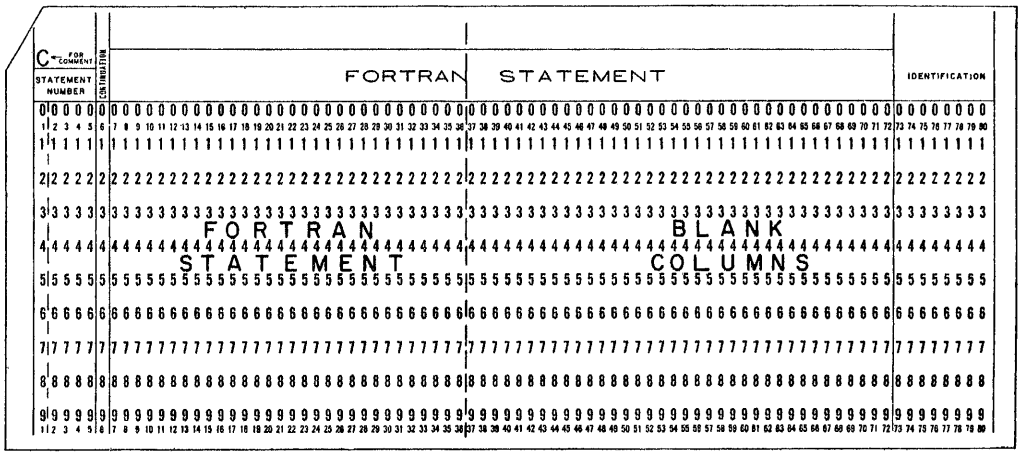
Blanks in the statement field are ignored by the FORTRAN system and the programmer may use them freely to improve the readability of the source program listing.

Statement Numbers

As noted previously, fixed point constants from 0001 to 9999 may be used as statement numbers. The statement number field in the cards may not be left blank; if a statement is not given a number, zeros must be punched in the statement number field (columns 2-5). If a statement number does not require all four card columns, the remaining unused columns must be filled with zeros.

Statement Card Format

The card format for 650 FORTRAN statement cards is shown below. Each field of the card is described in the following table.



Card Columns	Description
1	An alphabetic C (or any non-zero punch) in this column indicates a comments card, which will be ignored during processing. A zero indicates a statement to be processed. This column must have a punch in it.
2-5	Statement number field may be any number from 0000 to 9999. This field must contain numerical punches.
6	Used to indicate continuation cards. A zero indicates first card of a statement regardless of whether the statement uses one or more cards. A non-zero punch from 1-9 indicates a continuation card. This field must contain a numerical punch. Comments cards must contain a zero punch in this column.
7-36	The statement. Numerical, alphabetic, and special characters, and blank columns are all acceptable in this field.
37-80	Blank columns.

Operating Instructions: 650 FORTRAN (Compilation) Phase

Console Settings

Storage Entry: 70 1952 9999 (or 00 0000 0000 if 650 FORTRAN deck is already loaded).

Switches:		
	Programmed	STOP
	Half Cycle	RUN
	Control	RUN
	Display	UPPER
	Overflow	SENSE
	Error	STOP

Operation

1. Ready 650 Console with proper settings; insert 650 FORTRAN control panel into 533; feed blank cards in the punch hopper.

2. Ready read hopper with

   a. 650 FORTRAN Compiler deck
   b. FORTRAN statements cards

3. Depress Computer Reset key; Program Start key; and, when the 533 read hopper empties, End-of-File key.

The 650 will load the 650 FORTRAN Compiler deck and will automatically start reading the FORTRAN statement cards. Each FORTRAN statement card read will immediately be punched out as a comments card. Behind the punched comments card will be all the SOAP symbolic instructions compiled from that FORTRAN statement. This process will be repeated each time until every FORTRAN statement has been read.

Error Procedure for the Compilation Phase

The 650 FORTRAN deck is sequentially numbered in columns 7-10 and is checked for correct sequence while being loaded. If a card is missing or out of order, a sequence error halt will appear on the console:

01 0000 ABBB

where A will be either the digit 1 or 2. This digit corresponds to the phase being loaded (1 for compilation phase, 2 for the assembly phase). The BBB number (3 low-order digits) of the halt indicates the last card in correct order.

If the error occurs in the compilation phase, clear the read hopper, correct the sequence, and reload, starting at the beginning of the deck.

Other than the sequence error stop, no programmed stops are included in the compilation phase. Therefore, it is extremely important that the user follow exactly the rules for writing FORTRAN statements.

It is possible for the machine to stop because of (1) a read error (blank or illegal punch) in columns 1-6 of a statement card, or (2) an incorrectly punched FORTRAN statement causing the compiler to attempt an illegal operation (Branch Distributor operation on other than 8 or 9, or entering a loop causing an illegal address and a storage selection light, etc.). If a machine stop does occur, remove the cards from the read hopper and stacker, and run out the cards still in the read unit. If corrections can be made immediately, (1) reload the read hopper with the corrected card and all of the remaining cards of the program, and (2) transfer to location 1999 and depress the Program Start key.

NOTE: Under certain circumstances, such as a continuation card in a lengthy arithmetic statement, part of the statement containing the error may have been compiled and punched out before the error was encountered. In this case, clear the cards from the punch hopper and remove all cards up to and including the last comments card(s).

If the error occurred on a continuation card, reload the read hopper starting at the first card of the statement, i.e., reprocess the entire statement rather than restarting at the error card.

Completion of the Compilation Phase

The last FORTRAN statement to be compiled must be an END statement card. Immediately after the END statement has been processed, the machine will punch a block reservation card with "BLR" in the operation code columns, and blanks in both the data and instruction address columns. This card is required for the SOAP-PACKAGE assembly phase and must remain in the exact position as punched out in the output deck. The purpose of the blank BLR card is for punching out constants contained in the FORTRAN source program. One or two additional block reservation cards may also be punched depending on whether subscripted variable and/or GO TO tables have been established. See "Determining Available Drum Locations," page 31.

Rearranging Output Deck

1. Run all cards out of the punch feed and discard the first and last card. The remaining cards, in order, are: (1) each FORTRAN statement (comment card format) and the compiled SOAP symbolic instructions for that FORTRAN statement, (2) the blank BLR card, and (3) the block reservation card(s), if any.

2. Rearrange the card order so that the subscripted variable and/or GO TO table(s) block reservation card(s), if any, are now in front of the deck.

NOTE: The assembly phase of the 650 FORTRAN system performs part of the compilation. Accordingly, the output from the compiling phase may contain cards which are blank except for numerical punches in the comments field (columns 63-72) or special characters in the operation field (columns 48-51) or data address field (columns 51-56). These cards are a necessary part of the system and must not be discarded under any circumstances.

Operating Instructions: SOAP-PACKAGE (Assembly) Phase

Console Settings

Storage Entry: 70 1952 9999

Switches: Same as for 650 FORTRAN (Compilation) Phase.

Operation

1. Ready the 650 Console with proper settings; insert the 650 FORTRAN control panel into the 533; feed blank cards in the punch hopper.

2. Ready 533 read hopper with

   a. SOAP-PACKAGE Assembly deck.

   b. Function subroutines in five-instructions-per-card absolute format, if any.

   c. Entry point synonym cards for subroutines in absolute format, if any.

   d. Block reservation cards, if any.

   e. Function subroutines in SOAP II symbolic format, if any.

   f. Compiler output in SOAP symbolic format.

   g. One blank card, if it is desired to punch out the availability table after assembling. (The availability table can also be obtained by manually transferring control to location 1900 at completion of assembly.)

3. Depress Computer Reset key, Program Start key and, when the 533 read hopper empties, the End-of-File key.

4. Run cards out of the punch feed. Discard the first and last cards. The remaining cards, all in five-instructions-per-card format, are

   a. Object program load routine.

   b. Package of built-in subroutines.

   c. Subroutines entered in five-instructions-per-card format.

   d. Subroutines entered in SOAP II symbolic format.

e. Object program with the last instruction a transfer to the starting instruction of the object program.

f. Availability table, if specified. (These table cards must be removed before running the object program. They are identified by a "12" punch in column 41.)

Programmed Stops

As stated in the "Error Procedure for the Compilation Phase" page 39, the error halt

01 0000 2BBB

indicates that a card is missing or out of order while the SOAP-PACKAGE assembly deck is being loaded. The BBB number (3 low-order digits) of the halt indicates the last card in correct order.

If the error stop occurs, clear the 533 punch hopper and discard the output. Then clear cards from 533 read hopper, correct sequence of deck, and reload, starting at the beginning of the deck.

Other programmed stops in this phase are identified by Console address lights, as follows:

Address Lights	Reason for Stop
0111	Symbol table full.
0222	Drum packed.
0333	Illegal SOAP II symbolic card has been encountered in user's subroutine. Depress Program Start key to continue assembly.
0999	Load card has been encountered during assembly of compiled instructions.

Error Procedure

Programmed stop 0111 indicates the symbol table capacity of 300 has been reached. This stop should rarely occur without first having obtained a programmed stop 0222 — packed drum. However, if this stop is encountered, the following correction procedure is available.

If, in the source program, the user has assigned statement numbers to statements other than those which are referenced by other statements, the statement numbers can be deleted from the non-referenced statements. Statements that do not have numbers will not be included in the symbol table.

If the above procedure does not eliminate the stop, the source program must be rewritten and divided into smaller programs.

Programmed stop 0222 indicates drum capacity has been reached and requires that the source program be rewritten, and divided into smaller programs.

Programmed stop 0333 indicates an illegal operation code or location address in user's subroutine and will insert blanks in the instruction and its respective address. The output card will not be punched as a load card. The instruction can be corrected after the program has been assembled by making use of the availability table provided by the SOAP-PACKAGE assembly phase.

Programmed stop 0999 indicates that placement of the user's five-instructions-per-card absolute subroutine is out of sequence. See "Operating Instructions — SOAP-PACKAGE (Assembly) Phase, Operation 2."

Assembling more than one Program

If more than one program is to be assembled, the SOAP-PACKAGE assembly program must be reloaded each time. Reloading is necessary because the object program loading routine and built-in subroutines are punched directly from the assembly program deck while it is being loaded. The procedure to be followed in reloading the assembly program is the same as the initial loading above.

CHAPTER IV — USING THE OBJECT PROGRAM

This chapter contains the information and instructions necessary for utilizing an object program produced by the 650 FORTRAN system. The first section deals with the preparation of data cards, and the second section consists of operator's instructions and notes for running the object program.

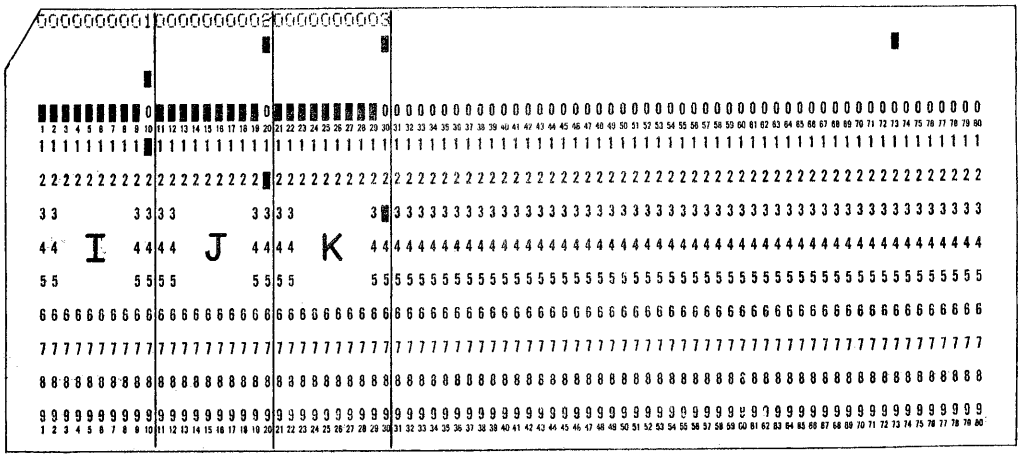
Preparing Data Cards

As indicated in Chapter II, a READ or PUNCH statement in the source program will cause the object program to read or punch data cards until the complete List has been processed. The reading and punching of data is accomplished by built-in subroutines provided in the assembly phase.

Data Cards

Data cards are identified by a "12" punch over card column 73. One to seven ten-digit words of data may be punched on one card.

For example, a data card for a READ, I, J, K statement with a value of -1 for I, +2 for J, and +3 for K would be punched as follows:



NOTE: The sign of the word is punched in the units position of the 10-digit field.

If the List requires more than one card, i.e., more than seven words of data, additional cards are read or punched.

Data is punched in the first seven fields of the card (each field is ten card columns). There must be a separate group of data

cards for each READ statement in a FORTRAN program. All seven fields on each card must be filled, except for the last card in the group. Because the last card may have less than seven fields, the remaining (unused) fields may be left blank. For example, the following statements might occur in a FORTRAN program:

```
DIMENSION A(4, 5), B(2, 3), N(4, 1), M(3, 4), L(7, 2)
READ 1, A, B
READ 2, N, M, L
```

There must be two separate groups of data cards, one for A, B and a second for N, M, L. Twenty-six words of data are required for A and B. Therefore, the first three cards in this group will each contain seven words of data and the fourth card will contain five words. The second READ statement requires thirty words of data. Four data cards will have all seven fields filled, and a fifth card will have two fields. It is important to remember that the order of data punched in the cards is controlled by the READ or PUNCH statement List. The data in a List will be read or punched from left to right with arrays in column sequence.

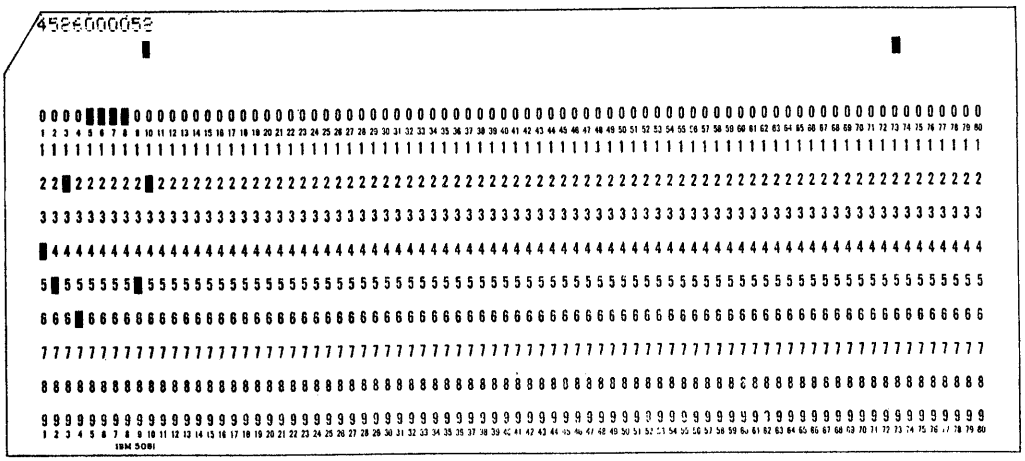
Output cards will be sequentially numbered by the object program. Word eight of each data card will contain this number. For input cards, this word may be used for identification purposes if desired.

Negative values are indicated by an "11" punch over the units positions of the respective field.

Form of Data

Data representing values of floating point variables are punched in data cards as floating point numbers of the form .xxxxxxxxPP, where PP is the power of 10 with 50 added, to avoid negative exponents.

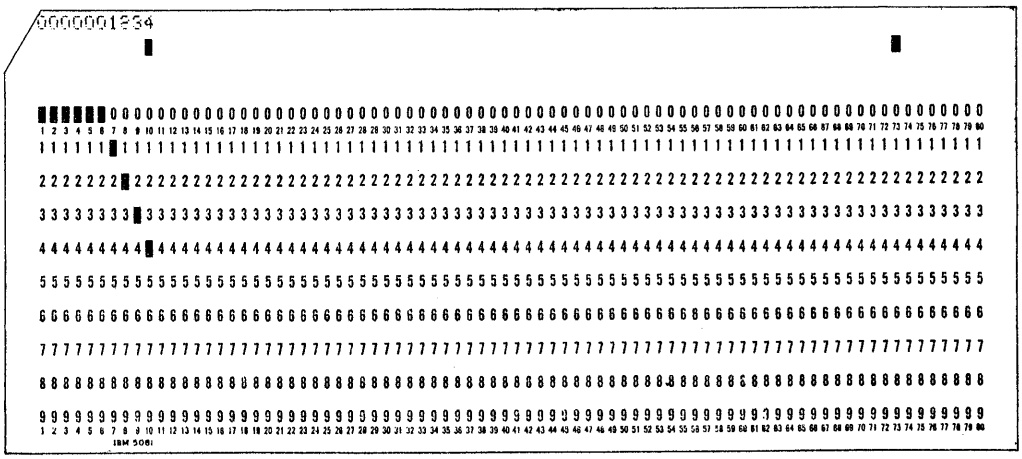
For example, the floating point value, +45. 26, is punched as a ten-digit word starting at card columns 1-10 and appears as follows:—



The sign of the value (a "12" punch for plus, or an "11" punch for minus) is punched in the units position of the word. The "12" punch in column 73 designates the card as a data card. Up to seven 10-digit words may be punched into each data card.

Data representing values of fixed point variables are handled as integers and are punched in cards as ten-digit words. Any unused (high order) positions must be punched with zeros.

For example, the value, +1234, is punched as a ten-digit word starting at card columns 1-10 and appears as follows:



The sign of the value (a "12" punch for plus, or an "11" punch for minus) is punched in the units position of the word. The "12" punch in column 73 designates the card as a data card. Up to seven 10-digit words may be punched into each data card.

Operating Instructions: Object Program

Console Settings

Storage Entry: 70 1952 9999 (or 00 0000 0000 if object program is already loaded).

Switches: Same as for 650 FORTRAN (Compilation) Phase.

Operation

1. Ready the 650 Console with proper settings; insert the 650 FORTRAN control panel into the 533; feed blank cards in the punch hopper.

2. Ready read hopper with entire output of the assembly phase and data cards (if required by the program).

3. Depress Computer Reset key; Program Start key; and, when read hopper empties, End-of-File key.

When the object program has been loaded, the last card of the object program deck will transfer control to the first instruction of the object program, which is always location 1999.

Programmed Stops Established in the Object Program by the SOAP-PACKAGE Subroutines

Address Lights	Error Condition	Built-in Subrt. Involved
0001	Negative or zero argument	LOGF, LOGEF
0002	Floating point result $> 9.9999999 \times 10^{48}$	EXPF, EXPEF
0003	Error in floating point exponentiation	POW1F
0011	Floating point argument of zero with negative exponent	POW2F
0049	Floating point result $> 9.9999999 \times 10^{48}$	POW2F
0100	Floating point overflow or underflow in an arithmetic statement	Any subroutine using floating point
0501	Floating point number to be fixed $\geq 10^{10}$	XFIXF

Error Procedure

The various error conditions listed above may result from such causes as logical errors or scaling problems inherent in the source program, errors in preparing data cards, etc. Depressing the Program Start key will cause the 650 to perform the instruction contained in the distributor, which will be the subroutine exit instruction. The instruction in the distributor should be noted as an aid in finding the point in the object program where the error was encountered.

SUMMARY OF OPERATING PROCEDURE

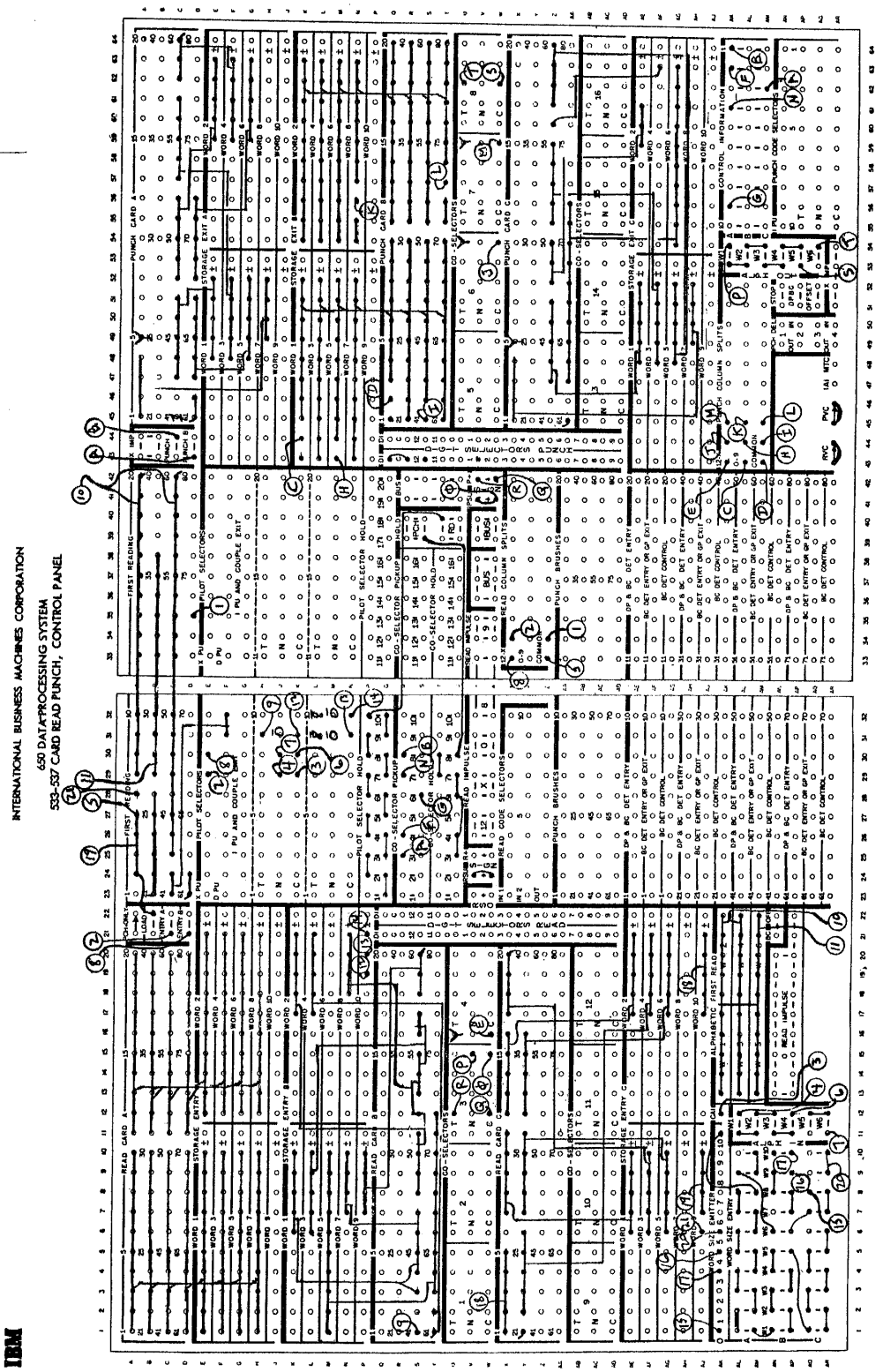
CONSOLE SETTINGS

Storage Entry:		70 1952 9999
Switches:	Programmed	STOP
	Half Cycle	RUN
	Control	RUN
	Display	UPPER
	Overflow	SENSE
	Error	STOP

OPERATION	COMPILATION PHASE	ASSEMBLY PHASE	OBJECT PROGRAM
533 Control Panel	650 FORTRAN	650 FORTRAN	650 FORTRAN
Ready Punch Hopper	Blank cards	Blank cards	Blank cards
Ready Read Hopper	650 FORTRAN deck; FORTRAN statement cards. Note: If 650 FORTRAN deck is already loaded, set Storage Entry switches to 00 0000 0000 or transfer control to location 0000 by means of Address Selection switches.	SOAP-PACKAGE deck; User's subroutines in five-per-card absolute; User's entry point synonym cards for absolute subroutines; Block Reservation cards, if any; User's subroutines in SOAP II format; Compiled program (in SOAP format); "BLR" blank card. (2)	Entire output deck of the assembly phase; Object program data cards. Note: If object program is already loaded set Storage Entry switches to 00 0000 0000 or transfer control to location 0000 by means of Address Selection switches.
Depress keys	Computer Reset Program Start End-of-File (when read hopper empties)	Computer Reset Program Start End-of-File (when read hopper empties)	Computer Reset Program Start End-of-File (when read hopper empties)
Output - after discarding the first and last cards.	Compiled program in SOAP format; "BLR" blank card; Subscripted Variable and/or GO TO Table; Block Reservation cards. (1)	Object program load routine; 650 FORTRAN built-in subroutines; User's subroutines; Object program. (3)	Output from the object program

1. Rearrange output cards to give the order shown as input to assembly phase.
2. Included with these Block Reservation cards, can be BLA cards for removing SOAP-PACKAGE built-in subroutines.
3. All output is in five-instructions-per-card format.

533 Control Panel Wiring Diagram



APPENDIX III

SAMPLE PROBLEM: MATRIX MULTIPLICATION

Listing of FORTRAN Source Program Statement Cards

```
C     0000          RECTANGULAR MATRIX
C     0000          MULTIPLICATION
                    DIMENSION  A(4,5) ,B(5,3)
                    READ 1 , A,B
                    READ 1 , N,M,L
          7         DO 4 J= 1,N
          1         DO 4 I= 1,M
          6         SUM = 0.0
          2         DO 3 K= 1,L
          3         SUM=SUM+A(I,K)*(K,J)
          4         PUNCH 1, SUM, I, J
          8         END
```

Note: This sample problem provides a test case for the system. It is recommended that the source program be processed through each phase of the system and the output for each step compared with the appropriate listings in this appendix.

Listing of Output from Compilation Phase of 650 FORTRAN: The Program in SOAP II Symbolic Form

```
                BLR     0101     0135        0000000000
1                       RECTANGULAR MATRIX
1                         MULTIPLICATION
1                       DIMENSION  A(4,5),B(5,3)
1                       READ 1 ,A,B
      $1        LDD               READF       0000000000
                )                             0040050100
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                )                             0050030120
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                LDD     $2        -           0000000000
1                       READ 1 , N,M,L
      $2        LDD               READF       0000000000
                )                             7500000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                )                             7400000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                )                             7300000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                LDD     +7        -           0000000000
1                       DO 4 J= 1,N
      +7        RAA       0001    $3          0000000000
1                       DO 4 I= 1,M
      +1        RAB       0001    $4          0000000000
1                       SUM = 0.0
      +6        RAU     *                     0000000000
                STU     SUM       +2          0000000000
1                       DO 3 K= 1,L
      +2        RAC       0001    $5          0000000000
1                        SUM=SUM+A(I,K)*B(K,J)
      +3        RAL       8007                0000000000
                STL     -1                    0000000000
                RAU       8005                0000000000
                MPY     *                     0000000005
                ALO     -1                    0000000000
                SLT       0004                0000000000
                ALO                 8002      0000000000
                RAU       0115                0000000000
                STU     )1                    0000000000
                RAL       8006                0000000000
                STL     -1                    0000000000
                RAU       8007                0000000000
                MPY     *                     0000000004
                ALO     -1                    0000000000
                SLT       0004                0000000000
                ALO                 8002      0000000000
                RAU       0096                0000000000
                STU     )2                    0000000000
                RAU     )2                    0000000000
                FMP     )1                    0000000000
                FAD     SUM                   0000000000
                STU     SUM                   0000000000
                AXC       0001    $5          0000000000
      $5        RSL       8007                0000000000
                STD     K                     0000000000
                ALO     L                     0000000000
                BMI     +4        +3          0000000000
1                       PUNCH 1, SUM, I, J
      +4        LDD               PNCHF       0000000000
                )                             8284740000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                )                             6900000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                )                             7100000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                                              0000000000
                *                             0000000000
                LDD                 -         0000000000
                AXB       0001    $4          0000000000
      $4        RSL       8006                0000000000
                STD     I                     0000000000
                ALO     M                     0000000000
                BMI                 +6        0000000000
                AXA       0001    $3          0000000000
      $3        RSL       8005                0000000000
                STD     J                     0000000000
                ALO     N                     0000000000
                BMI     +8        +1          0000000000
1                       END
      +8        HLT       0000      8000      0000000000
                BLR                           0000000000
```

Listing of Output from Assembly Phase of 650 FORTRAN: The Object Program in Five-per-Card Format

```
100000083  6901521820  6501550158  6901611822  6501640167  6901701822  1999015201  5801610167
100000084  6901731823  6901761820  6502290182  6901851822  6501880141  0170017301  7601820185
100000085  6901441822  6501970150  6901531822  6901561823  8000010162  0141014401  5001530156
100000086  8200010160  6001000205  2102100163  8800010169  6580070211  0154020402  0501630254
100000087  2000390142  6080050149  1902020172  1500390143  3500040203  0211014201  4901720143
100000088  1502068002  6001150219  2101740177  6580060235  2000390192  0203020602  1901770235
100000089  6080070199  1902520222  1500390193  3500040253  1502568002  0192019902  2201930253
100000090  6000960151  2103060159  6003060261  3901740224  3202100137  0256015101  5902610224
100000091  2102100213  5800010169  6680070227  2401800183  1501470201  0137021301  6902270183
100000092  4603040254  6901571821  6502600263  6901661822  6503190272  0201030401  5702630166
100000093  6901751822  6502280181  6901841822  6901871823  5200010160  0272017501  8101840187
100000094  6680060217  2402690322  1501380243  4601460204  5000010162  0160021703  2202430146
100000095  6680050369  2401780231  1501790233  4601360154  0100008000  0162036902  3102330136
100000096  0040050100  0050030120  0000000210  0000000004  0000000005  0155016402  6002520202
100000097  0000000147  0000000138  0000000178  0000000179  0000000269  0197018802  2802290319
100000098  0000000147  0000000138  0000000178  0000000179  0000001999  0000000000  0000001830
```

Note: The above listing does not include the SOAP-PACKAGE cards produced in the Assembly Phase. Accordingly, the card serial numbers (word 1, columns 8-10) begin at 083.

Actual Problem and Answer Matrix (Input and Output Data)

$$\begin{bmatrix} 16 & -13 & -7 & 2 & 5 \\ 2 & 1 & 8 & -10 & -12 \\ 1 & -6 & 15 & 11 & 18 \\ 14 & 17 & 3 & -2 & 9 \end{bmatrix} \times \begin{bmatrix} -3 & 5 & 7 \\ -8 & 13 & -4 \\ 6 & -4 & 10 \\ 12 & 3 & -5 \\ 2 & -9 & 11 \end{bmatrix} = \begin{bmatrix} 48 & -100 & 139 \\ -110 & 69 & 8 \\ 303 & -262 & 324 \\ -166 & 192 & 169 \end{bmatrix}$$

Listing of Input Data Cards for Object Program

```
1600000052  2000000051  1000000051  1400000052  1300000052-  1000000051  6000000051-  0000000002
1700000052  7000000051-  8000000051  1500000052  3000000051  2000000051  1000000052-  0000000003
1100000052  2000000051-  5000000051  1200000052-  1800000052  9000000051  3000000051-  0000000004
8000000051-  6000000051  1200000052  2000000051  5000000051  1300000052  4000000051-  0000000005
3000000051  9000000051-  7000000051  4000000051-  1000000052  5000000051-  1100000052  0000000006
  00000003  0000000004  0000000005                                                    0000000001
```

Listing of Output (Answer) Cards from Object Program

```
4800000052  0000000001  0000000001  0000000000  0000000000  0000000000  0000000000  0000010000
1100000053-  0000000002  0000000001  0000000000  0000000000  0000000000  0000000000  0000020000
3030000053  0000000003  0000000001  0000000000  0000000000  0000000000  0000000000  0000030000
1660000053-  0000000004  0000000001  0000000000  0000000000  0000000000  0000000000  0000040000
1000000053-  0000000001  0000000002  0000000000  0000000000  0000000000  0000000000  0000050000
6900000052  0000000002  0000000002  0000000000  0000000000  0000000000  0000000000  0000060000
2620000053-  0000000003  0000000002  0000000000  0000000000  0000000000  0000000000  0000070000
1920000053  0000000004  0000000002  0000000000  0000000000  0000000000  0000000000  0000080000
1390000053  0000000001  0000000003  0000000000  0000000000  0000000000  0000000000  0000090000
8000000051  0000000002  0000000003  0000000000  0000000000  0000000000  0000000000  0000100000
3240000053  0000000003  0000000003  0000000000  0000000000  0000000000  0000000000  0000110000
1690000053  0000000004  0000000003  0000000000  0000000000  0000000000  0000000000  0000120000
```

GLOSSARY

650 FORTRAN System — An automatic coding system for the IBM 650 which uses a subset of the original FORTRAN language for its source programs and gives optimized 650 machine language programs as output.

Assemble — Assign actual machine language addresses and operation codes to symbolic addresses and operation codes.

Compile — The generation of a series of machine language instructions to execute the operation indicated by the source program statements.

FORTRAN Language — Statements closely resembling the language of mathematics which are acceptable to a computer as a source program.

FORTRAN Program — A source program written in the symbolic language of FORTRAN.

FORTRAN System — An automatic coding system originally designed for the IBM 704, intended primarily for scientific computation. In addition to the 704, this system has been adapted to the following IBM Data Processing Systems: 650, 705, 709, 1620, 7070, 7080, 7090.

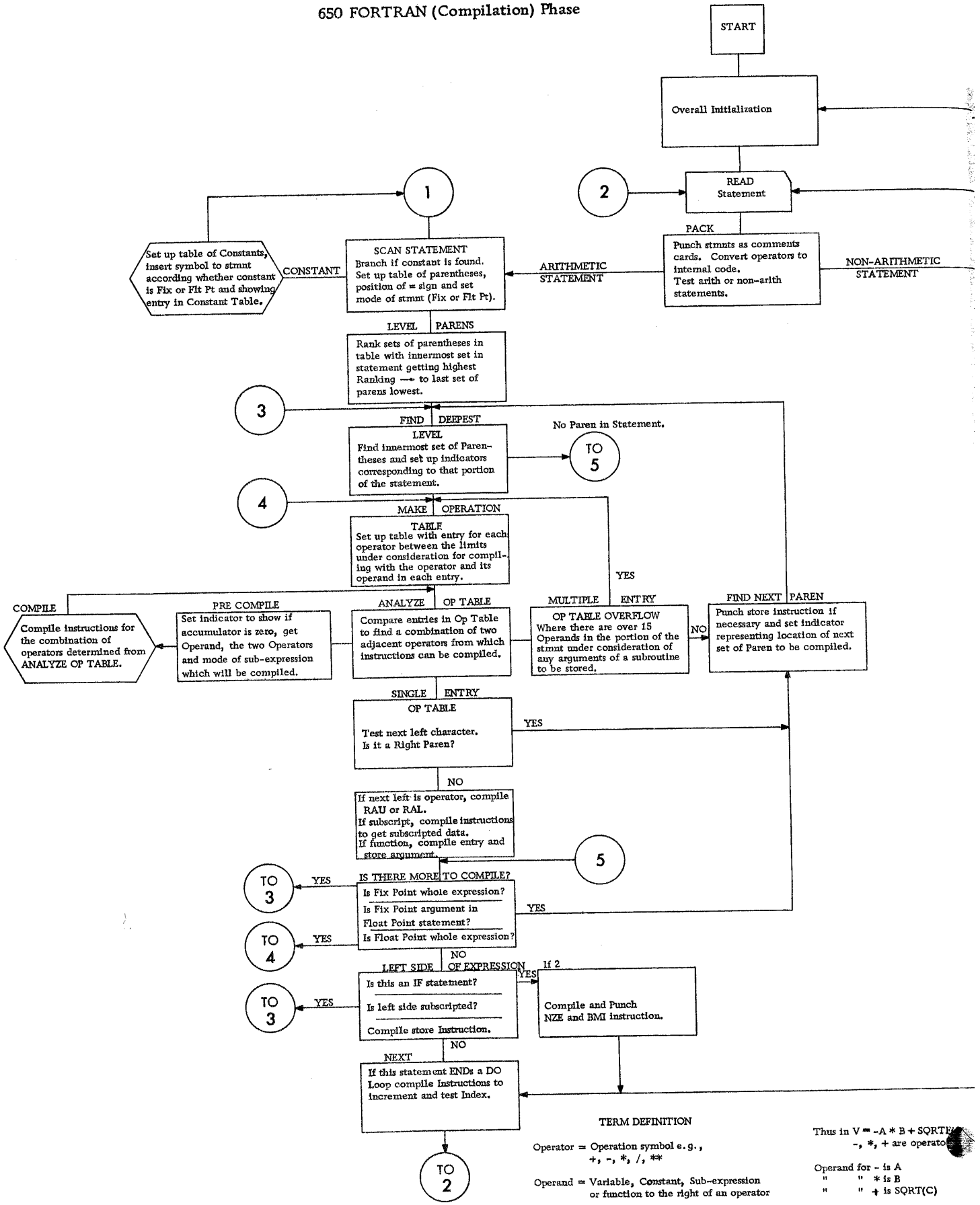
Object Program — The machine language program which is the final output of an automatic coding system.

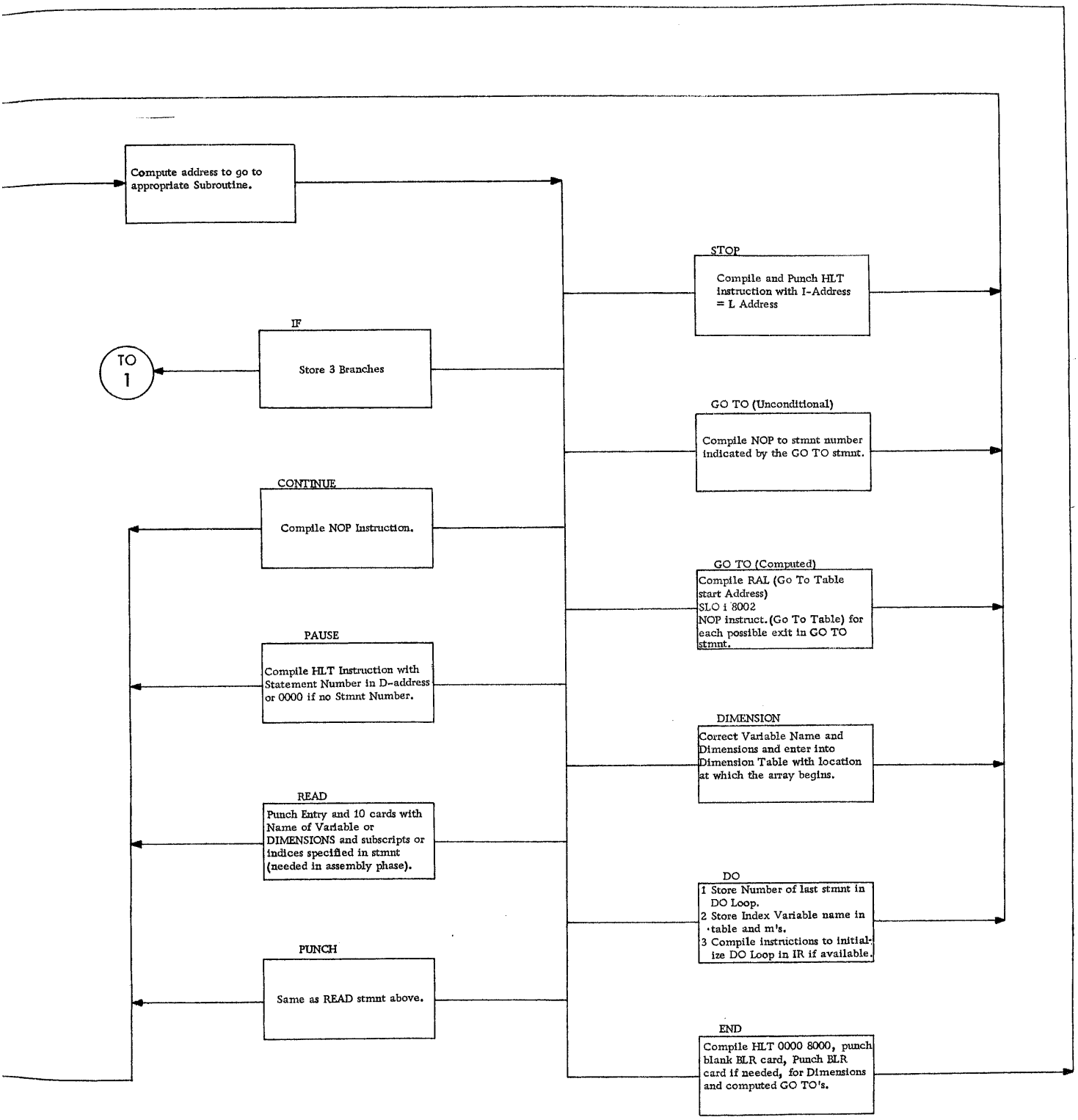
Optimize — To select the proper memory location so as to have the minimum amount of access time between each instruction.

Source Program — The input to an automatic coding system. In the 650 FORTRAN system the source program consists of FORTRAN statements.

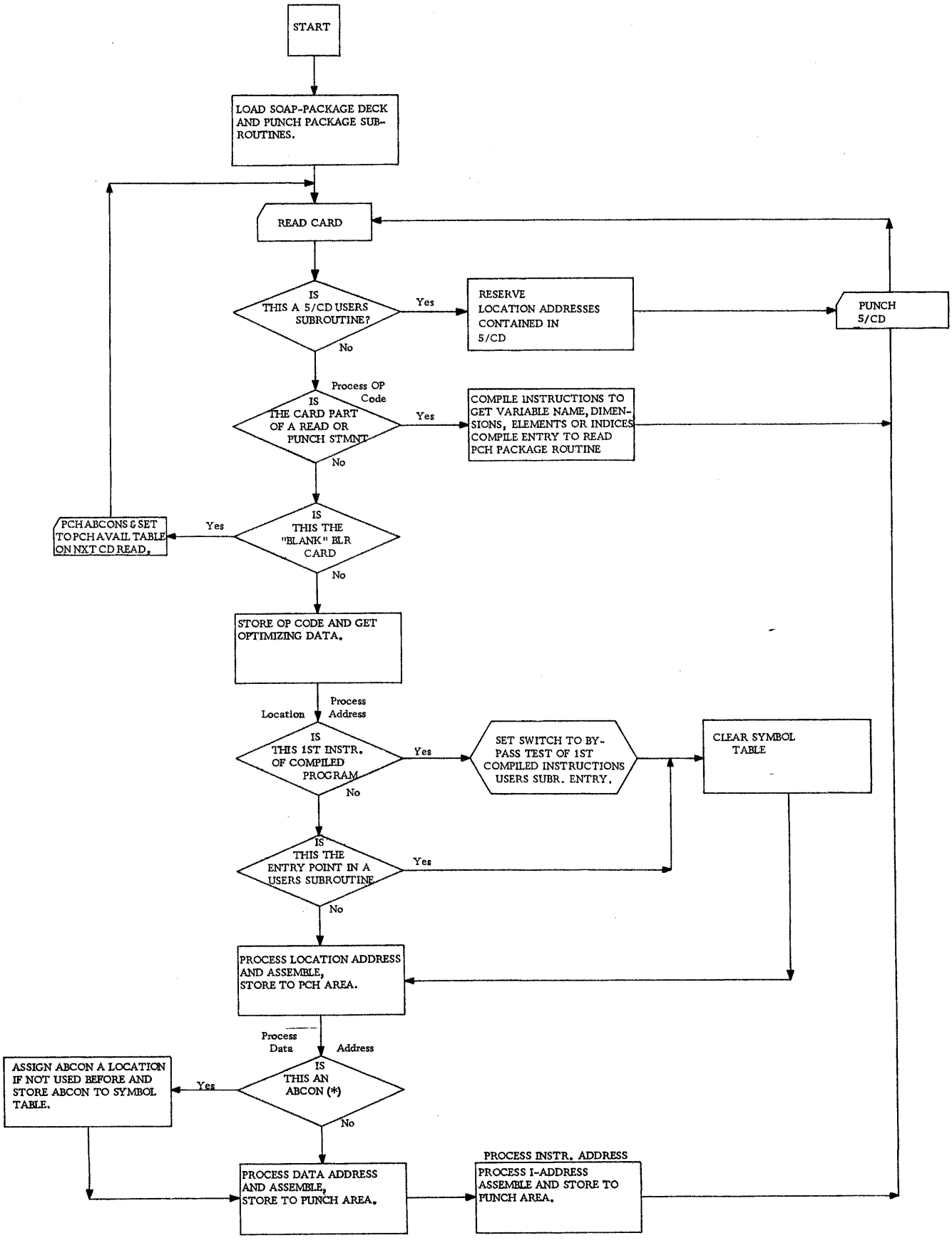
APPENDIX V

650 FORTRAN (Compilation) Phase




Compute address to go to appropriate Subroutine.
STOP
Compile and Punch HLT instruction with I-Address = L Address
IF
Store 3 Branches
TO 1
GO TO (Unconditional)
Compile NOP to stmnt number indicated by the GO TO stmnt.
CONTINUE
Compile NOP Instruction.
GO TO (Computed)
Compile RAL (Go To Table start Address)
SLO i 8002
NOP instruct. (Go To Table) for each possible exit in GO TO stmnt.
PAUSE
Compile HLT Instruction with Statement Number in D-address or 0000 if no Stmnt Number.
DIMENSION
Correct Variable Name and Dimensions and enter into Dimension Table with location at which the array begins.
READ
Punch Entry and 10 cards with Name of Variable or DIMENSIONS and subscripts or indices specified in stmnt (needed in assembly phase).
DO
1 Store Number of last stmnt in DO Loop.
2 Store Index Variable name in table and m's.
3 Compile instrictions to initialize DO Loop in IR if available.
PUNCH
Same as READ stmnt above.
END
Compile HLT 0000 8000, punch blank BLR card, Punch BLR card if needed, for Dimensions and computed GO TO's.

SOAP-PACKAGE (Assembly) Phase



IBM 650 PUBLICATIONS

The following IBM 650 Systems manuals have been published as of the date of this manual:

Form Number	Title
GENERAL INFORMATION MANUALS	
F28-4016	Planning for IBM 650 Card Systems
F28-4033	Program Testing IBM 650 Data Processing System
REFERENCE MANUALS	
C28-4000	SOAP II for the IBM 650 Data Processing System
C28-4004	IBM 650 Tape Merging Program Merge II
C28-4022	650 Tape Sorting Program Sort III
C28-4024	Floating-Decimal Interpretive System for the IBM 650
C28-4028	FOR TRANSIT Automatic Coding System for the IBM 650 Data Processing System
C28-4046	RAMAC 650 Programs: Utility — Scheduling — Chaining
X21-7643	Physical Planning Installation Manual 650 System

IBM

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

Printed in U.S.A. C29-4047