COMPUTING CENTER HANDBOOK

Third Edition

Edited by Charlotte Rubashkin

Computing Center

Newark College of Engineering

Newark, New Jersey

June 1966

NCE COMPUTING CENTER HANDBOOK

CONTENTS

INTRODUCTION

The NCE Computing Center Handbook is intended to provide users of the Computing Center with general information about the various 1620 programming procedures. The Handbook discusses in detail the two versions of Fortran in common use at the Center: Load and Go, (called L and G) for the Model I 1620, and Kingston Fortran, (called KFII) for the Model II 1620. Some information is also included on other available programming systems.

Section II lists definitions for those elements of the Fortran language which are common to the Fortran systems discussed in the Handbook.

Section III lists those Fortran language specifications which apply specifically to the Load and Go system. Section III also describes in detail operating instruction for using the NCE Load and Go compiler. The compiler, punched on cards, is available in the Computer room. The card deck containing the compiler will usually be found in a box on top of the 1620 Model I card reader.

Section IV lists those Fortran language specifications which apply specifically to the Kingston Fortran II system. A Kingston Fortran II compiler and an IBM Fortran II Compiler are available on the 1311 disk of the 1620 Model II. Section IV notes operating instructions for using the KFII compiler. Section V notes operating instructions for using the IBM Fortran II compiler. Section V also lists those IBM Fortran II (called IBM FII) language instructions which provide for operations not permitted by the KFII compiler. Additional IBM Fortran II language specifications are discussed fully in the IBM 1620 Fortran Manual (#C26-5619).

Section VI briefly describes other systems which may be used on either the 1620 Model I or II. Section VI also lists the library programs available at the Center for use with either the 1620 Model I or II. The programs are stored either on the 1311 disk, on punch cards, or on paper tape.

Section VII consists of instructions for using the other equipment in the Center. Included are instructions for using the card punches, the reproducer, the printer and the sorter.

The first section of the Handbook describes the equipment available at the Center and the rules for using the equipment. Users please note the procedures for applying for problem numbers (Section I.2), for using the IN-OUT boxes (Section I.4c) and for signing the Time Sheet when operating the computer (Section I.4).

The Handbook can be used in a loose-leaf binder and is designed so that pages may be removed for use at the machines. The Handbook will also be kept up to date with the printing of revised pages. The revised pages will be numbered by section and page and should be inserted at the appropriate location.

I NCE COMPUTING CENTER

I.1 Equipment available

The Computing Center at NCE has available for use two analog computers and two digital computers. The analog computers are PACE TR-10 and PACE TR-15 computers manufactured by Electronic Associates, Inc. They may be connected together for use as one larger computer. An associated plotter and oscilloscope are available. They are best adapted to solving small systems of ordinary differential equations. The IBM 1620 computer is a general-purpose computer capable of solving a variety of numerical and logical problems. The 1620 Model I has a 20,000 digit core storage capacity. The 1620 Model II has a 40,000 digit core storage. Model II users may use two 1311 disks which provide auxiliary storage for 4 million digits.

Input to either 1620 is via punched cards or console typewriter. Output from the 1620 Model I is punched into cards (which can be printed on an off-line device) or typed by the console typewriter. Output from the 1620 Model II may be printed by a 1443 on-line printer at the rate of 240 lines a minute, or punched into cards (which can be printed on an off-line device) or typed on the console typewriter. Card punches are available at the Center. Card punches and other card-handling off-line machines are discussed in Section VI. Paper tape is also available on the 1620 Model II for input-output in special applications.

This Handbook deals primarily with the 1620. Information about the analog computers may be obtained at the Center. The Center library has several copies of the Handbook of Analog Computation published by Electronic Associates, Inc., Long Branch, New Jersey. (Publ. No. L 800 0001 OA.)

I.2 Applying for use of the 1620 computer

The computers and their adjunct equipment at the Center are available at NCE for use by classes, student projects, student theses, and student and staff research.

The equipment is available for unsponsored and sponsored research at the charges discussed below. A staff member wishing to use the computer for research should consult with the Data Processing Manager, Mr. Alexander Altieri. If the research is not funded, then the staff member should first see Dean Bedrosian to find out if college funds are available. If they are not, it still may be possible to obtain some free time. Instructors planning to use the computer in their classes should let the Computing Center know at the beginning of the semester how many computer hours they plan to use and what specific times they would like to reserve. Forms for this are obtainable at the Center.

A certain amount of time is also available for use by other educational institutions and for commercial applications. Those wishing to use the services of the Center should call the Computing Center MA 4-2424, Ext. 217, and consult with the Data Processing Manager or the Secretary. I.2

All individuals wishing to use a computer must first fill out an application form, available at the Center, and receive a problem number. Problem numbers must also be assigned to each class section wishing to use the Computing Center. Problem numbers are assigned by the Secretary at the Center, Mrs. de la Vega, upon receipt of the application form. A sample application form is shown on page 4.

I.3 Charges for computer time (all figures quoted are subject to change) I.3

Computer time is provided free of charge for class use and for unsponsored research on both the undergraduate and graduate levels. Unsponsored research, either by students or faculty, must be approved by Dean Bedrosian unless the work is in connection with a student project or thesis. The general laboratory fee paid by all students includes the use of the computer for approved projects. The fee paid by special students for courses using the computer also covers the computer charge.

N.C.E. COMPUTING CENTER

SCHEDULE OF CHARGES

(Effective June 1966)

1620 Model I with 20 K Memory	Per Hour
Commercial or Industrial Users	$30.00
Sponsored Research	15.00
Other Educational Institutions	15.00/Negotiated
1620 Model II with 40 K Memory, two Disk Drives (2 million digits each) and 1443 On-Line Printer	
Commercial or Industrial Users	65.00
Sponsored Research	35.00
Other Educational Institutions	35.00/Negotiated

	Per Hour	
407 Printer	7.00	I.3
519 Reproducing Machine with Mark Sensing	3.00	
082 Sorter	2.00	
Services		
Keypunching (excluding cards which are $1/thousand).	4.00	
Computer Operation	5.00	
Programming	10.00	
Problem Analysis	15.00	

Minimum Monthly Charges = $10.00

I.4 Scheduling computer time I.4

The Computing Center is normally open 8 A.M. - 10 P.M. weekdays. Special arrangements can be made with the Computing Center Secretary for computing time during the evening (if the time is not pre-empted by class use) or on Saturdays.

Staff members who have obtained problem numbers and wish to operate the computer themselves may schedule time with the Computing Center Secretary. A schedule of computer time for the week will be posted on the bulletin board in the Center; it should be consulted by all users.

It is required that all individuals operating a computer sign in and out on a time sheet, available at the computer.

I.4a Class use of computer time I.4a

Instructors wishing to schedule time on the computer for their classes should consult the Computing Center Staff at the beginning of the semester. Each class must be assigned a problem number. See Section I.2 Applying for the use of the computer.

I.4b Student operation of the 1620 Model I I.4b

Students may operate the 1620 Model I during periods scheduled for student use. Runs of up to five minutes duration only can be run if others are waiting. Otherwise up to 15 minute runs will be permitted. A student

COMPUTING CENTER

NEWARK COLLEGE OF ENGINEERING

Application for Problem Number

Name ______________________________

Address ______________________________

Telephone Number ______________

Date ______________

NCE USERS

Staff Member — Department ______________ Telephone Ext. ______

Type of Use:

- [ ] Class Use — Number of Students ______
  - Time Blocks ______________
- [ ] Unsponsored Research
- [ ] Sponsored Research — Account Number ______
- [ ] Other ______________

Student — Year ______ Department ______

Type of Use:

- [ ] Thesis — Subject ______
- [ ] Computer Education
- Other ______________

OUTSIDE USERS

Affiliation ______________________________

Problem or program title ______________________________

Problem Number Assigned ______________ Approved by ______________ may sign up for this time by writing his name on the bottom of the list provided daily on the computing center bulletin board. After running he should cross his name off, but he can sign up again immediately (on the bottom!) of the list. No problem number is required for this computer time. I.4

Another block of student time will be scheduled at 6 - 7 P.M. every day for the benefit of evening students who will have priority at this time.

I.5 IN-OUT boxes I.5

Students may not operate the 1620 Model II unless an instructor or Center Staff member is present at the machine. Center staff members will process source decks prepared by students and staff members for the 1620 Model II. NCE staff members who would like their programs batch processed by Computing Center staff members should see Mr. Altieri at the Center office. Students who have prepared source decks for Model II batch processing should leave their decks in a file box marked "IN-Model II" located in the preparation room. The face of the first card should be clearly marked with the name of the user and the initials "F.C." The back of the last card should be marked "L.C." The programs will be processed on a first come first served basis and the source decks and all printed output will be placed in the "OUT-Model II" file. The decks should include all necessary control cards. (See Appendix "Model II-batch processing")

I.6 Punched cards I.6

Programmers using the Center must do their own card punching. Blank cards for punching are available at the bookstore at a cost of $2.00 a box.

Programmers may also consider using mark sensing which is a method for marking the face of the cards with a special pencil. Fortran cards printed for mark sensing are also available at the bookstore. The marked cards are made into punched cards by processing them through the 519 Reproducing Punch. A Computing Center staff member will process mark sensed decks which are placed in the box marked IN on the 519. See Appendix "Use of Mark Sense Cards" for details on staff processing of mark sense cards. Card punching and mark sensing are also discussed in Section VII.

Room for card storage at the Center is limited, but cardboard boxes for storing programs will be provided for active research projects, and for storing programs for any class section using the computer.

Individuals prone to carry small program decks about with them should be very careful not to dent the card edges; cards with very minor dents often will not be read by the equipment. The Center provides stiff protecting covers which should always be used to protect program decks that are not stored in boxes or card storage drawers.

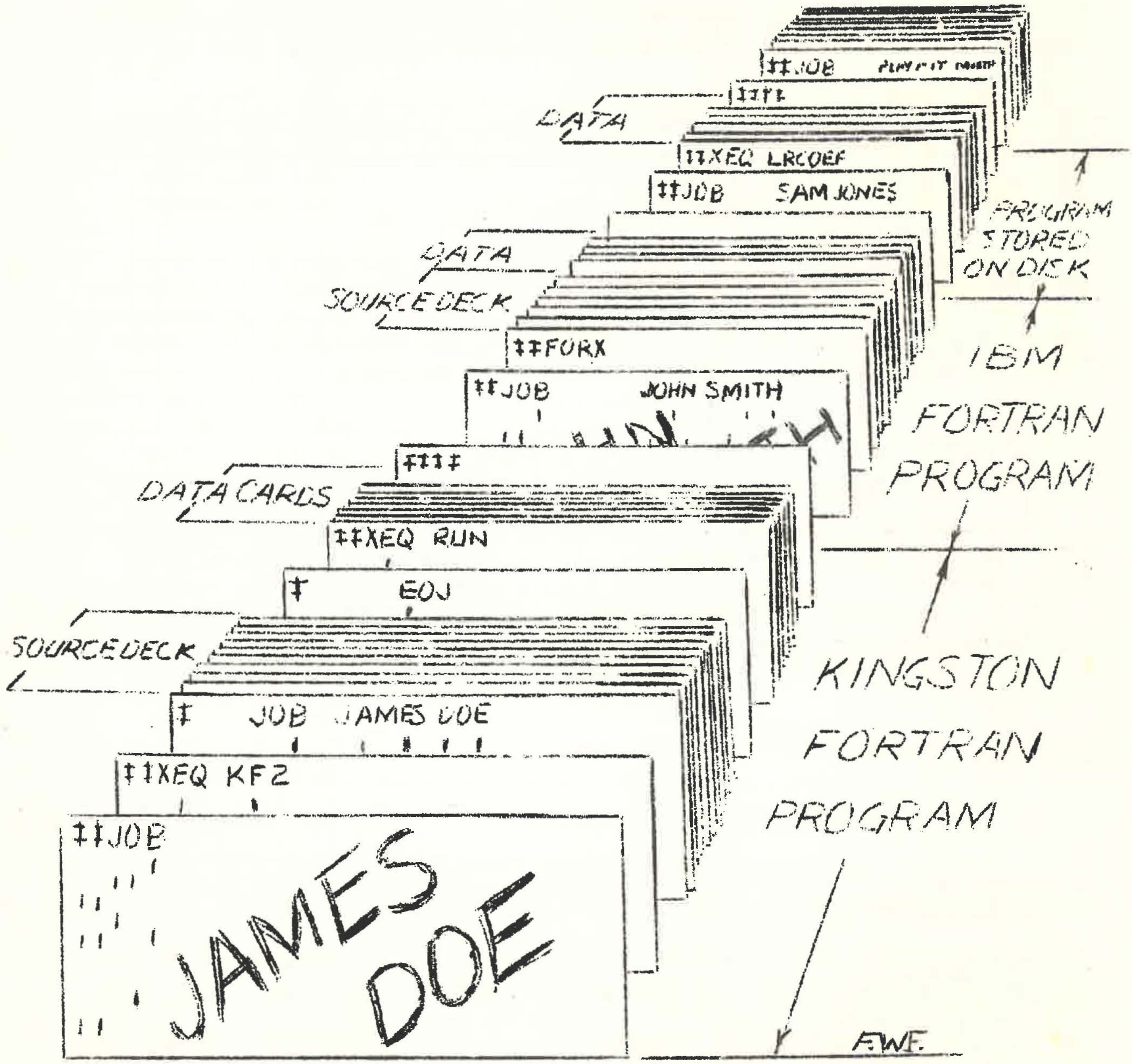


Fig. 1.

Source decks prepared for batch processing on the 1620 Model II.

I.7 References, and manuals I.7

The following IBM manuals, which are available at the NCE bookstore, are particularly helpful in programming the 1620 Model I and II:

IBM 1620 Data Processing System .... No. A26-4500
IBM 1620 1710 Symbolic Programming Systems .... No. C26-6500
IBM 1620 FORTRAN .... No. C26-5619
IBM 1620 Monitor II System .... No. C26-5774-0
IBM 1620 Central Processing Unit, Model 2 .... No. A26-5781-0

A small collection of books on programming, periodicals on data processing, and a set of 1620 library programs are available at the Center for use as reference materials. The library programs may be borrowed for short periods such as overnight or weekends. The main library has many books on programming, computers, and numerical analysis.

I.8 Staff I.8

Director	Dr. Frederick G. Lehman
Associate Director..................	Dr. Phyllis Fox
Data Processing Manager	Mr. Alexander Altieri
Graduate Assistants	Mr. Young D. Kim Mr. Hubbard Seward
Undergraduate Assistants............	(Listed on the Center bulletin board)
Computing Center Secretary..........	Mrs. Hortensia de la Vega
Programmer/Systems Analyst.........	Mr. Larry Arakaki

I.9 Courses in programming I.9

In addition to the courses related to computing and data processing in the regular curriculum, the Computing Center provides a non-credit six-week course in programming which is held each semester starting about three weeks after the start of the semester. There is a $15 charge to help cover the cost of the manuals which are provided and the time used on the computer. The course involves about an hour of lecture a week supplemented by individual practice in programming and operating the computer.

There is also a short course in Fortran Programming given under the direction of the Special Courses and Continuing Education Division which is held on the evenings during the fall, spring, and summer sessions. For further information, please contact Mr. Paul Burns, Ext. 330.

II BASIC ELEMENTS OF THE FORTRAN LANGUAGE

II

II.1 Source program and object program

II.1

A source program is a series of statements written in the Fortran language. The source statements are analyzed by the Fortran compiler, or processor, which then generates machine language instructions. The machine language instructions, produced by the Fortran compiler, comprise the object program. During execution of the object program, the computer uses data supplied by the programmer to execute the arithmetic and logical operations required by the problem.

Fortran source statements can be grouped into 5 categories:

1. Arithmetic statements which define the calculations to be performed. Arithmetic statements include operators, variables, constants, paranthesis and functions.

   Examples:

   ```
   A = 4. - B + 6.*C*(D+E)
   ROOT = (-B + (B*B - 4.*A*C)**.5)/(2.*LOG(A))
   ```

2. Replacement statements which cause the item to the left of the equal sign to be given the same value as the item to the right. All arithmetic statements are also replacement statements.

   Examples:

   ```
   A = B
   A = 4.2
   ```

3. Control statements which determine the sequence of execution of the object program instructions.

4. Input/output statements which transmit information between the computer and the input/output devices such as the console typewriters, the card read-punch, the paper tape device.

5. Specification statements which supply information required by the processor to allocate locations in storage for certain variables and/or arrays. They may also enable the user to control the allocation of storage.

Fortran compilers may also provide for various types of subprograms.

(See L and G: III.5a)*
(See KFII: IV .8)

---

*Reference to relevant material in other sections will be indicated in parentheses of this sort. This one indicates for example that Section III.5a contains more on subprograms in Load and Go.

Example of a source program:

Problem to be solved: Sum the integers from 1 to 1000.

Statement Number	Source Statement	Comment: Type of Statement
	SUM = 0.0	Replacement statement
	A = 1.0	Replacement statement
3	SUM = A + SUM	Arithmetic statement
	A = A + 1.	Arithmetic statement
	IF (A-1000.) 3,3,6	Control statement
6	PUNCH 10, SUM	Input/output statement
10	FORMAT (F8.2)	Input/output statement
	STOP	Control statement
	END	Control statement

Instruction for punching a source deck are given in the following sections:

(L and G: III.1)
(KFII: IV .1)

II.2 Arithmetic statements II.2

II.2a Constants II.2a

Integer and real constants may be used in a source program written for either the Load and Go Compiler or the Kingston Fortran II Compiler. Hollerith constants may only be used in a source program written for the KFII compiler.

(See KFII: IV.2a)

Integer constants - An integer constant is a number written without a decimal point.

Example:

```
I = 678    (678 is a valid integer constant)
K = I - 23   (23 is a valid integer constant)
```

The magnitude of integer constants depends on the compiler.

(See L and G: III.2a)
(See KFII: IV .2a)

Real constants - A real constant is number written with a decimal point and consisting of 1-8 significant decimal digits.

Examples:

Valid real constant	Invalid real constant
1.0099999	234
234.	

The real constant 123456789.1 is accepted by the Load and Go and KFII Fortran processor but it is stored as 12345678. (See Appendix "Storage of integer and real numbers")

A real constant may be followed by a decimal exponent written as the letter E followed by a one-or two-digit integer constant indicating the appropriate power of 10.

Example:

```
A = 4.2 + B*(5.E2-D)      (5.E2 is a valid  real  constant)

500.0 may be written:     .5  may be written:
    500.0                     .5
    5.E2                      5.0E-1
    5.0E2                     5.E-1
    5.0E02                    .05E1
    5.0E+02
   50.0E01
```

The magnitude of a nonzero real constant must be such that:

$$1.0*10^{-51} \leq \text{constant} \leq 9.9999999*10^{+48}$$

IBM Fortran II allows for variation in the number of significant digits permitted for real and integer constants.

(See IBM FII: V.1)

II.2b Variables

A Fortran variable is a symbol which represents a quantity that may assume different values. The value of a variable may change either for different executions of a program or at different stages within the program.

Example:

C = 5.0 + D   C and D are variables. The value of D must be determined by some previous statement and may change from time to time. The value of C varies whenever this computation is performed with a new value for D.

Variable names:

The number of alphameric (numeric and alphabetic) characters allowed in a variable name depends on which Fortran Compiler is used.

(See L and G: III.2b)
(See KFII: IV.2b)

The first character in a name must be alphabetic. Special characters are not permitted.

Variable types:

A variable may represent either an integer or a real number (i.e. a number containing a decimal point). See the definition above in Section II.2a

The type of a variable, that is, whether it is integer or real, can be specified implicitly as follows:

1. If the first character of the variable name is I, J, K, L, M, or N then the variable is an integer variable.

2. If the first character of the name is not I, J, K, L, M, or N, then the variable is a real variable.

Example:

```
INT = LEMMA + NUM      (These variables represent integers)
X = Y - ALPHA          (These variables represent real numbers)
```

Explicit specification of variable types is only allowed in KFII.

(See KFII: IV.2b)

Subscripted variables:

A subscripted variable consists of a variable name followed by a pair of parentheses enclosing subscripts separated by commas. The number of subscripts allowed depends on the Fortran Compiler.

(See L and G: III.2b)
(See KFII: IV.2b)

The subscripts specify the position of the variable in an array. An array is a group of quantities arranged in order.

Example:

Let A be an array consisting of the quantities, 700.4, .34, 532.99, then

```
A(1) = 700.4
A(2) = .34
A(3) = 532.99
```

An array may be multi-dimensional. The number of dimensions allowed in the array depends on the number of subscripts permitted by the compiler.

Example:

A two-dimensional array M may be a 2 by 3 table of integers with the following:

```
M(1,1) = 31          M(2,1) = -4
M(1,2) =  6          M(2,2) = 89
M(1,3) = 17          M(2,3) = -11
```

II.2c Arithmetic operation symbols II.2c

The arithmetic operation symbols: +, -, *, /, ** denote addition, subtraction, multiplication, division, and exponentiation, respectively.

II.2d Arithmetic expressions II.2d

An arithmetic expression is usually a combination of constants, subscripted or nonsubscripted variables, function names, (see subprograms) and arithmetic operation symbols.

General Rules for Forming Expressions:

(See L and G: III.2c)
(See KFII: IV.2c)
(for specific rules)

1. The variables and constants in an arithmetic expression must be of the same type with the exception that in exponentiation a real variable or constant may have an integer exponent.

2. Any expression may be enclosed in parentheses.

3. All operation symbols must be explicitly present.

4. No two operators may appear in sequence. (Note exception: Load and Go use of the minus sign. See L and G: III.2c)

5. Hierarchy of operations - Parentheses may be used in expressions, as in algebra, to specify the order in which operations are to be computed. Where parentheses are omitted, the order is understood to be as follows:

   a) Subscript evaluation

   b) Subscripting

   c) Argument evaluation, Function evaluation

   d) Exponentiation (**)

   e) Multiplication and Division (* and /)

   f) Addition and Subtraction (+ and -)

   Example:

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

6. An expression is scanned from left to right, and no operation is completed if there is a possibility of one of higher hierarchy first.

7. If operations fall within the same hierarchy rank, and parentheses are not used to indicate which operations are performed first, the following rules apply: II.2d

a) A/B/C is always compiled (A/B)/C

b) A*B/D*C is always compiled as (A*(B/D)*C)

c) A**B**C is not acceptable, parentheses must be provided.

d) A**-B is always compiled as $(A)^{-B}$
E-B**C is always compiled as $E-(B^{C})$
E = -A**B, the order of compilation depends upon the compiler.

(See L and G: III.2c)
(See KFII: IV.2c)

Note that the order of compilation discussed above can be important from the point of view of round-off error. (A/B)/C may give a different answer then A/(B/C), especially, if either B or C are small.

II.2e Replacement statements II.2e

In evaluating an arithmetic statement the value of the expression to the right of the equal sign is determined and that value is assigned to the variable to the left of the equal sign. If the result of the expression evaluation is not of the same type as the variable name to the left, the result is converted before assigned.

Example:

I = B + 4. - (C-D)**E	Evaluate B + 4.-(C-D)**E. Truncate the result to convert it to an integer. Assign it to I.
X = NUM + L - 1	Evaluate NUM+L-1. Convert the result to a real (decimal) number. Assign it to X.
A = A + 1.	A valid replacement statement. The value assigned to A is increased by 1.
I = J	A simple replacement statement. Value of J assigned to I.

Computations involving real variables are truncated (not rounded) to 8 significant digits. However, a subroutine which will round arithmetic computations is available to users of the IBM FII compiler. II.2e

(See IBM FII: V.2)

II.3 Control statements II.3

Normally Fortran statements are executed sequentially. However, it is often undesirable to proceed with each statement in this manner. Control statements alter the sequence of execution of the object program instructions.

A statement number must be assigned to each statement referenced in a program. Statement numbers must be entirely numeric. The number of digits allowed in a statement number depends upon the compiler.

(See L and G: III.3a)
(See KFII: IV.3a)

KFII also provides for address variables which can be used to reference statements.

(See KFII: IV.3b)

II.3a Unconditional GO TO statement II.3a

General form:

GO TO xxxx

Where: xxxx is a statement number

Examples:

GO TO 25 Control is transferred to the statement with number 25.

II.3b Computed GO TO statement II.3b

General form:

GO TO (x_1, x_2, x_3, ... x_n), i

Where: x_1, x_2, x_3, ... x_n are statement numbers, i is an integer constant or integer expression depending on the compiler used.

(See L and G: III.3b)
(See KFII: IV.3c)

This statement causes control to be transferred to statement x_1, x_2, or x_n depending on whether the current value of i is 1, 2, or n.

Example: II.3b

GO TO (10, 40, 50), I

When I = 2 control is transferred to statement number 40.

II.3c Arithmetic IF statement II.3c

General form:

IF (a) n_1, n_2, n_3

Where: a is an arithmetic expression and n_1, n_2, n_3, are statement numbers.

This statement causes control to be transferred to statement n_1, n_2, or n_3, if the value of the expression (a) is less than, equal to, or greater than zero, respectively.

II.3d IF (SENSE SWITCH i) n_1, n_2

Where: i is an integer constant or integer expression depending on which compiler is used and n_1, n_2, are statement numbers.

(See L and G: III.3c)
(See KFII: IV.3d)

This statement causes control to be transferred to the statement n_1 if the sense switch is on, or to n_2 if the sense switch is off. i determines which machine indicator is to be interrogated. Any of the machine indicators can be interrogated by the IF (SENSE SWITCH) statement. However not all machine indicators are relevant to the computations performed by the object program.

II.3e DO statement II.3e

General form:	End of Range	index		initial value	test value	increment
DO	n	i	=	m_1,	m_2,	m_3

The form (integer, integer expression) that the index (i), initial value (m_1), test value (m_2), and increment (m_3), may take depends on which compiler is used.

(See L and G: III.3d)
(See KFII: IV.3e)

The DO statement is a command to execute repeatedly the statements that follow, up to and including the statement n. The first time the statements are executed, i has the value m_1, and each succeeding time i is incremented by the value of m_3. After the statements have been executed with i equal to the highest value that does not exceed m_2 in the direction of incrementation, control passes to the statement following statement number n. This is called a normal exit from the DO statement. II.3e

The range is the series of statements to be executed repeatedly. It consists of all statements following the DO, up to and including statement n. The range can consist of any number of statements.

The index is an integer variable that is incremented by the value m_3 for each execution of the range of statements. Throughout the range of the DO, the index is available for use either as a subscript or as an ordinary integer variable. However, the index should not be changed by a statement within the range of the DO. (See KFII IV for exception). Upon completion of the DO, the index must be redefined before being used again. When transferring out of the range of a DO, the index is available for use and is equal to the last value it attained.

The initial value is the value of the index for the first execution of the range.

The test value is the value that the index must not exceed in the direction of incrementation. After the range has been executed with the highest value of the index that does not exceed the test value, the DO is completed and the program continues with the first executable statement following the range.

The increment is the amount by which the value of the index will be changed after each execution of the range. The increment may be omitted, in which case, it is assumed to be 1.

Example:

The statement `DO 10 I = 1,5,2`

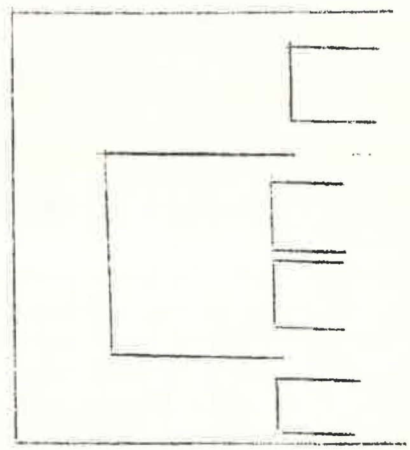
will cause the range of the DO to be executed with I taking on the successive values 1, 3, and 5.

Restrictions on statements in the range of a DO

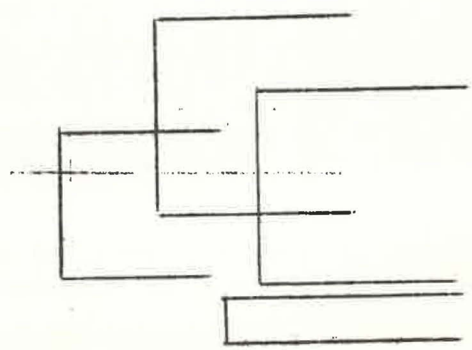
The restrictions on statements in the range of a DO are as follows:

(1) Within the range of a DO may be other DOs. When this is so, all statements in the range of the inner DO must be in the range of the outer DO.

A set of DOs satisfying this rule is called a nest of DOs. For example, the following configuration is permitted (brackets are used to indicate the range of the DOs):



The following configuration is not permitted:



(2) Transfer of control from inside the range of a DO to outside its range is permitted at any time. However, a transfer is not permitted into the range of a DO statement from outside its range.

(3) The range of a DO cannot end with a GO TO, IF, FORMAT, STOP, RETURN, or another DO statement.

II.3f Continue statement II.3f

General form:

CONTINUE

CONTINUE is a dummy statement that does not produce any executable instructions. It is used to furnish a reference point which must be assigned a statement number. It is required if the last statement of a DO would otherwise be a transfer statement.

II.3g Pause statement II.3g

General form:

PAUSE

The PAUSE statement causes the program to halt. Pushing START causes the program to resume execution starting at the next statement following the PAUSE statement.

II.3h STOP, END, and CALL EXIT statements II.3h

The use of the STOP, END and CALL EXIT statement depends on which compiler is used.

(See L and G: III.3e)
(See KFII: IV.3g)

II.4 Input/output statements II.4

II.4a General form for input statements for Fortran without Format II.4a

READ, list	(punched card input)
ACCEPT, list	(console typewriter)
ACCEPT TAPE, list	(paper tape)

The list specifies the number of items to be read and the locations into which the items are to be placed.

Examples:

READ, A, CAT, DOG	read from a punched card three numerical values to be assigned to the variables A, CAT and DOG respectively
ACCEPT, X	Expect a numerical value to be typed in and become the value for X.

II.4b <u>General form for input statements for Fortran with Format</u> II.4b

READ n, list (punched card input)

ACCEPT n, list (console typewriter)

ACCEPT TAPE n, list (paper tape)

n references the Format statement number

(See KFII: IV.4a)
(Fortran with Format)

II.4c <u>General form for output statements for Fortran without Format</u> II.4c

PUNCH, list (punched card output)

The list specifies what items are to be outputted on cards. When the item in a list is a variable name, the last value assigned to the variable name will be punched on the card. All items in the list must be variable names, not numeric values.

<u>Example</u>:

PUNCH, X, Y, N If the machine assigned X = 3.4, Y = 100.7, and N=2, the numbers 3.4, 100.7 and 2 will be outputted on a card.

TYPE, list or PRINT, list may be used to put output on the console typewriter, depending on the compiler used.

(See L and G: III.4)
(See KFII: IV.6)

II.4d <u>General form for output statement for Fortran with Format</u> II.4d

PUNCH n, list (card output)

TYPE n, list (output on console typewriter)

PRINT n, list (output on 1443 printer) (See KFII: IV.4c)
(Fortran with Format)

II.5 <u>Specification statements</u> II.5

II.5a DIMENSION statement II.5a

General form:

DIMENSION VA(i_1), VB(i_2)

Where: VA, VB, are variable names and i_1, i_2, may be one or more unsigned integer constants separated by commas.

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

Example:

DIMENSION A (10), B(5,15)

Space will be set aside for 10 values of A, and 75 values of B.

Dimension statements must be written for each subscripted variable (unless using other specification statements permitted by KFII) and must appear before the subscripted variable is mentioned.

II.5b Other specification statements II.5b

Additional Specification statements permitted in KFII are discussed in Section IV.7.

Introduction

The NCE Load and Go compiler, written by George Rumrill, Bruce Fowler, and Hubbard Seward and revised by them in January 1965, allows 100 to 200 card FORTRAN source programs to be both compiled and executed on the 1620 Model I in a single continuous run. Furthermore, since the compiler stays in the memory, a sequence of different programs can be run, one after the other. This method of processing is well suited to processing student programs. The system cannot be damaged by errors in programs. Many error checks are made, both during compilation and execution. No format specifications are used. Output format is determined by the type and range of the variable; input format is free form. Features include; arithmetic and flow trace, double subscripting, computed GO TO statements, limited Hollerith listing, and undefined variable detection.

Two versions of the Load and Go processor are available for student and staff use. Version Three is designed for rapid batch processing and eliminates the need for operator intervention during the processing of source programs. All input and output is punched on cards. Statements which require operator intervention, IF(SENSE SWITCH n) and PAUSE, are not recognized by the processor. Version Three, batch processing, should prove adequate for most student programs and its use should save considerable machine time during class laboratory hours. The Computing Center will also schedule hours when a staff member will batch process Load and Go programs placed in the IN box on top of the Model I. The source decks and a 407 listing of all card output will be placed in the OUT box at the end of the scheduled hour.

Version Two, which is designed for individual user console operation, will also be available during hours scheduled for "hands-on" individual user console operation. Input and output may be punched on cards or typed on the console typewriter, the source program may include STOP, PAUSE, and IF(SENSE SWITCH n) statements, and the user may exercise options which increase the amount of core storage available for the user's program. (See operator instructions, section III.7b.)

The Load and Go system is a variation of Fortran. Only those specifications, which differ from the basic Fortran definitions discussed in Section II, are listed in Section III.

III. NCE LOAD AND GO FORTRAN LANGUAGE SPECIFICATIONS III

III.1 Punching input source programs III.1

1. The statements of the source program can be punched anywhere on the first 72 columns of the card. Punching is free form; all blanks are ignored (except in input data as described below). If a statement number precedes the statement it can be punched starting at any column; spaces between the statement number and the statement are not necessary. Columns 73-80 are reserved for sequence numbers or other identification.

2. Comment cards are allowed and require a C in Column 1 followed by two blanks.

3. Continuation of a statement to another card is not permitted.

III.2 Arithmetic statements III.2

III.2a Constants III.2a

Integer Constants - The magnitude of an integer constant must not be greater than 9999.

Example:

Valid constants	Invalid constants
9999	10991
0	0.0
-356	-356.0
2	

III.2b Variables III.2b

Variable names

The name of a variable may have no more than five characters.

Example:

NUMB, DAT15, INT, X, Y, FARAD

Subscripted variables

1. A variable may have one or two subscripts. Thus one and two dimensional arrays are permissible.

   Example:

   A(I), B(1,2)

2. Subscripts may consist of an integer constant, an integer variable, or an integer variable plus or minus an integer constant.

Forms of subscripts:

i
i + c'
i - c'
c

Where: i is an integer variable and c is an integer constant.

3. The additive integer constant, c' must not be larger than 49.

4. Subscripts whose value during execution becomes negative or zero, or whose value exceeds the size of the DIMENSION statement, will result in ERROR 70. (See error messages, III.9)

Examples:

Valid Subscripting	Invalid Subscripting
A(IMAX-37)	A(IMAX-50)
I(J,MATH+17)	B(X,IMAX)
Q(124, LM-3)	C(MATH+51,J)

III.2c Arithmetic expressions

Rules for forming arithmetic expressions

1. Minus sign: The operation symbols, (,*,/,**, may be followed by a minus sign and will be correctly compiled.

   Example:

   A-B*-C, A+B/-C, A-B**-C are valid arithmetic expressions.

2. Exponentiation: A=-B**I will be compiled as $A=(-B)^I$ and calculated as the product (-B) (-B) (-B) ....

   Example: When B= 2, I=3

   $A= (-2)^3$ or (-2)(-2)(-2), A= -8

   A = -B**C will be compiled as A=EXP(C*LOG (-B))

   When (-B) is negative ERROR 62 will result. (See error messages, III.9)
   When (-B) is negative the value of A is calculated as A= EXP(C*LOG(ABS(-B))) during user processing.

   Example: When B=2, C=3
   A=EXP(3*LOG(ABS(-2))), A=8
   ERROR 62 will be indicated

   When B= -2, C=3
   A= EXP(3*LOG(-(-2))), A = 8

   During batch processing all execution errors result in the termination of the program in error. Thus Error 62 will result in termination of program execution during batch processing.

A= X-B**C will be compiled as A= X-(B**C) and calculated as A= X-EXP (C*LOG(B)).

3. Plus sign: A plus sign may not immediately follow an equal sign, left parenthesis, or any arithmetic operator.

   Example:

   A=+B, IF(+2.1-A) 12,20,10 are invalid arithmetic expressions.

4. An integer variable or constant may never be given an exponent.

III.3 Control statements III.3

III.3a Statement numbers

Statement numbers may be any one, two, three, or four digit integer.

III.3b Computed GO TO statement III.3b

GO TO (J_1, J_2, ...J_n), i

Where: J_1, J_2, ...J_n are statement numbers and i is a nonsubscripted integer variable.

The comma before the i is required punctuation.

The statement may contain any number of statement numbers.

During execution, the index of a computed GO TO statement will be checked to see if it is defined and if it is less than or equal to the number of statement numbers listed in the source statement.

Sample problem 5 at the end of Section III demonstrates how a computed GO TO statement may be used as the return statement in a subprogram.

III.3c IF (SENSE SWITCH n) statement III.3c

IF (SENSE SWITCH n) j_1, j_2

Where: n is any one or two digit integer constant

j_1, j_2 are statement numbers. Control is transferred to j_1 if sense switch n is on; control is transferred to j_2 if sense switch n is off.

The statement may be used to test the condition of any of the machine indicators. Care should be exercised in using the IF(SENSE SWITCH) statement to test the status of the arithmetic indicators since the operating subroutines may leave the indicators in a position which does not correspond to the result of the arithmetic calculation.

The IF(SENSE SWITCH) statement may be used to test the status of the program switches 1, 2 and 3, and the last card indicator, switch 9. The program switches are manually operated and should be set by the programmer prior to the execution of the IF(SENSE SWITCH) statement. Insertion of a PAUSE statement before the IF(SENSE SWITCH) statement will give the operator time to set the program switch. Switch 9, will automatically be set to the ON position when the last data card has been read. The processor will turn switch 9 off at the beginning of the execution of each program.

Example:

The last card indicator may be interrogated as follows:

IF(SENSE SWITCH 9) n_1, n_2

Control will be transferred to statement n_1, if the last card has been read.

The IF(SENSE SWITCH n) statement should not be included in a source deck prepared for Version Three, batch processing. (See Section III.6a, Preparing the source deck for batch processing)

III.3d DO statement III.3d

DO n i = m_1, m_2, m_3

Where: n is a statement number and i is a nonsubscripted integer (fixed-point) variable.

m_1, m_2, m_3, are either integer constants or nonsubscripted integer variables. If m_3 is not stated, it is taken to be 1. m_1, m_2, m_3, may be negative, zero, or positive but care should be taken when negative or zero values are assigned to m_1, m_2, or m_3. (Note that if during execution the value of a subscript becomes negative or zero ERROR 70 will result, and execution will be terminated.)

No more than seven levels of nested DO-loops are permitted.

Sample problem 2 uses nested DO-loops for data input and output. (See Section III.10)

III.3e STOP and END statements

The END statement signals the processor to terminate compilation. Thus it must always appear as the last statement in a source program. The STOP statement need not appear in a program.

The END statement is an executable statement and may have a statement number.

The control exercised by the execution of the STOP and END statements depends on whether the source program is compiled by Version two, individual operator processing or Version three, batch processing. The execution of the STOP or END statements results in transfer of control to the processor. During batch processing, execution of the STOP or END statements results in automatic compilation of the next source program in the batch. During user processing, execution of the END statement results in a machine halt. A new source program may then be compiled by pressing START. (See operator instructions, user processing, Section III.7b) During user processing execution of the STOP statement also results in a machine halt. The user's program may then be re-executed by pressing START.

III.4 Input-output statements III.4

III.4a General rules for input-output statements without format III.4a

1. Card input and card output must be used for Version three, batch processing. ACCEPT and PRINT statements will not be accepted by Version three, batch processing; use READ and PUNCH.

2. A format statement number in an input-output statement is ignored. A comma must be present if a list follows the input/output command. Thus:

```
PRINT 3, A, B
PRINT, A, B
PRINT
```

   are valid. Format statements may be included in the source program but will be ignored.

3. "TYPE" is not a valid statement (Use PRINT).

4. Input-output statements without lists (with or without format statement numbers) have the following effect:

   a) PUNCH produces a blank card during execution
   b) PRINT causes a carriage return during execution
   c) READ and ACCEPT are compiled, but ignored during execution

   Note: Omit comma after PUNCH and PRINT if there is no list.

III.4b Rules for input data III.4b

1. To obtain a line or a card of alphameric output, a card or a typed input record containing a T or P as a first character may be inserted in the appropriate place or places in the input data. A READ statement, calling for data, will type or punch, respectively the contents of the rest of the T or P card (or record) before reading in the data from the next card. All other data records must be entirely numeric. See sample problem 1, Section III.10.

2. Input data can follow a free format on a card with spaces separating each piece of data. All columns (1-80) will be read as data.

3. Commas can not be used to separate numbers.

4. Regardless of the mode specified in the input list, data may be in any of the following forms:

   2.0   2.   +0.2E1   2   .02E+02   +20000E-4

   If no decimal is punched it is assumed to lie at the right-hand end of the number. If a real number is entered when an integer number has been called for, the four digits immediately before the decimal point will be converted to an integer, and there will be no error indication. Thus

1.	Converts to 1
-325.E2	Converts to -2500

5. Regardless of the number of input and output statements that are executed, input data will be taken from one record (e.g. one card) until:
   a) That record is exhausted
   b) A record mark is encountered
   c) A change in the input device is required
   d) The program is reinitialized

6. Blank records intermixed with the data or source statements are ignored. This if the programmer wishes to read three numbers punched on three cards the read statement can be written as:

Read, A,B,C

and the cards punched as:

Card 1. 234.5
Card 2. 0045.678
Card 3. 4567.9

7. To input (or output) a one or two dimensional array a DO statement must be used. I/O statements with impiled DO statement are not allowed. See sample problems 2 and 3 at the end of Section III.

8. Input data outside the allowed range (e.g. larger than 10^{49}) will be read incorrectly. No error indication is given.

III.4c Rules for output data III.4c

1. Output is put out five (or fewer, as required) items per typed line or punched card. See output for sample problem 2.

2. Integer numbers will be in the format I5,11X. (where X indicates space)

3. Real numbers whose magnitude falls in the range (.1) to (99,999,999) will be output in the format F16.d. (where d indicates a variable number of decimals). Numbers whose magnitude falls outside this range will be in the format E16.7.

4. Sequence numbers will not be punched on output cards.

5. For alphameric output see 1. under Section III.4b.

III.5 Subprograms III.5

III.5a Library subroutines III.5a

1. The following library subroutines are allowed with real arguments. Note that the subroutines are to be considered as real functions regardless of their letter.

Example:

Y = A + LOG (B) is a valid arithmetic statement.

2. The absolute value function is the only library subroutine which may also use an integer argument.

The arithmetic expression containing an absolute value function with an integer argument must be in the integer mode.

Example:

M = J + ABS(K*L)

Subroutine	Operation	*Symbolic Name
Natural Logarithm	log A	LOG
Exponential	e^{A}	EXP
Square root	$\sqrt{A}$	SQRT
Sine	sin A	SIN
Cosine	cos A	COS
Arc tangent	$\tan^{-1}A$	ATAN
Absolute value	/A/	ABS

The argument of the trigonometric functions (SIN,COS,ATAN) must be expressed in radian measure.

*A terminal F may be added to function symbolic names. (LOG, EXPF, SQRTF etc.)

3. When a subroutine is given an impossible argument (e.g. SQRT of a negative number) an error message is printed out but the computation procedes. (See Section III.8 errors 61, 62, 65). The values used for the functions in these cases are the following:

$\frac{0.}{0.} = \frac{X}{0.}$ or $\frac{0.0}{0.0} = \frac{X}{0.0}$ ----> 9.9999999 E+48

0.0**X ---> 0.0

LOG (0.0) ----> 0.0

LOG (X where X < 0) ------> LOG (/X/)

SQRT (X where X < 0) ------> SQRT (/X/)

SIN (X where X ≥ 1.E9 radians) -----> 1.0

COS (X where X ≥ 1.E9 radians) ----> 1.0

III.5b Subprograms written by the programmer III.5b

The Load and Go processor does not provide special statements to call subprograms. The programmer who wishes to use a subprogram must provide for the entry to the subprogram, the transfer of the argument to the subprogram, and the return to the main program. Sample problem 5 indicates how the programmer can use the computed GO TO statement to return to the main program.

III.6 Batch processing instructions - Version Three III.6

III.6a Preparing the source deck for batch processing III.6a

1. The following Load and Go control card must be used to identify the users source deck and to signal the processor that a new source deck is submitted for batch processing. The Load and Go control card must precede the first source statement.

```
Card col. 1 2 3 ..........40 ........... 50 .........................72
            $   Name of user  Problem number Optional user identification
```

Where: The $ is punched in column 1, the users name in columns 3-35, users problem number in columns 40-48, and optional user identification in columns 50-72.

2. The following statements will not be accepted by Version Three, batch processing:

   PAUSE
   ACCEPT, list
   PRINT, list
   IF (SENSE SWITCH n) j_1, j_2

3. Either the END or the STOP statement should be the last executable statements in a program. Execution of either the END or STOP statements will cause the processor to compile the next program in the read hopper.

III.6b Operator instructions-batch processing III.6b

1. Load the processor, Version Three: (IF in memory go to 2.)

   a. Put Load and Go processor - Version Three in read hopper followed by the source decks to be run.

Press	INSTANT STOP	(on 1620)
	RESET	(on 1620)
	LOAD	(Yellow button on card reader)

   b. Put blank cards in punch hopper

Press	PUNCH START	(on 1622)

c. When the processor has been loaded the typewriter should type the following message:

READY FOR BATCH PROCESSING

d. Remove the processor deck from reader stacker and put away.

e. The "READER NO FEED" light remains on when two cards are left in the read hopper.

Press READER START

2. <u>If the processor is in memory:</u>

a. Put source decks in the read hopper and blank cards in the punch hopper.

Press	READER START	(red bottom on card reader)
	PUNCH START	(green button on card reader)

b. The "READER NO FEED" light remains on when two cards are left in the card reader.

Press READER START

During batch processing there should be no program halts, no need to press START, no need to reinitialize or reload the processor.

NOTE: If the control card is incorrectly punched, the source deck following the incorrect control card will not be compiled. Also note that any execution error will cause termination of the program in error.

III.6c <u>Punched card output - batch processing</u>

All output including error messages is punched on cards. The first card output contains the information punched on the users control card and will identify all subsequant card output resulting from the compilation and execution of the users program. Error messages will be punched as follows:

```
Card cols. 1 2 3 4 5 6 7 .............................................72
           X X X X + X X ERROR XX
```

Where: The 4 digit integer in columns 1-4 indicates the last statement number encountered before the error. If the error is a compilation error the 2 digit integer in columns 6-7 indicates a count of the number of additional cards from the indicated statement number up to the source statements containing the error. The card count includes comment cards but not blank source records.

If the error is an execution error the 2 digit integer in columns 6-7 indicates a count of the number of additional statements executed from the indicated statement number up to the statement containing the error. Note that the sequence in which statements are executed may be very different from the sequence in which they are written in the source program. Thus if the numbered statement is part of a loop the executed statements may not be those listed in the source program as directly below the numbered statement. Also note that during execution the count includes only executable statements. Thus blank records, dimension, continue and comment statements are not counted.

ERROR XX is the error code. Tables indicating the error code and the appropriate error message will be found at the end of the chapter. (See Sections III.8 and III.9)

The trace feature is not available during batch processing. The user should insert PUNCH statements in the source program so that he will have the information necessary to de-bug his program when program execution does not result in satisfactory output.

Note that if the PUNCH statement is placed within a DO loop, output will include each value calculated during the loop. If the PUNCH statement is placed outside the DO loop, output will include only the last value calculated. (See sample problem 4, III.10)

The last card output signals the IBM 407 Printer to start a new page (for the next program output) and is blank except for a Z punched in card column 80.*

III.7a Preparing the source deck for user processing III.7a

1. A Load and Go control card must not be used.

2. The following statements will be accepted by Version Two.

```
ACCEPT, list
PRINT, list
IF (SENSE SWITCH n) J1, j2
PAUSE
```

3. Execution of the END statement results in a machine halt. A new source program may be compiled by pressing START. Execution of the STOP statement also results in a machine halt. The user's program may be re-executed by pressing START.

--------------- ******** ---------------

*NOTE SWITCH 3 on the IBM 407 must be set to the ON position when printing output produced during batch processing on the Model I.

III.7b Operator instructions - user processing

Step 1. Loading the processor

If the Load and Go processor is in memory, go to 2.
If Load and Go processor is not in memory proceed as follows:

a) Put Load and Go processor in card reader

Press	INSTANT STOP	(on 1620)
	RESET	(on 1620)
	LOAD	(Yellow button on card reader)

b) After the deck has been read press START (1620). Follow instructions typed on console typewriter as follows:

set program switches--normally all off. Special options:

Switch 1 ON — to omit Trig Functions

Switch 2 ON — to eliminate Flow Trace Feature

If the TRIG functions are omitted the area available for storage of program and data will be enlarged by 1,040 digits.

If the FLOW TRACE is omitted, the stored program will be shortened by 4 digits for each statement number. However, when the FLOW TRACE is omitted there is no statement identification of execution error messages.

Step 2. Compiling the source statements

a) Set switches for compilation--normally all off. Special options:

Switch 1 ON — for typed input

Switch 2 ON — will type out source program

Switch 4 — to correct typed-in statements as indicated below

b) Entering the source statements punched on cards:

Press START
START

The console typewriter will type the message COMPILATION

Press READER START
Press READER START again to read last two cards

If the console typewriter does not type the message COMPILATION the processor must be reinitialized before a new source program will be compiled. (See f below, reinitialization.)

c) Entering the source statements from the typewriter:

Enter source statements on the typewriter, each statement must be terminated by pressing R/S Key. No record mark is required.

d) To correct typed input: (steps may be also followed to correct typed data input during program execution.)

If R/S has already been pressed, it is too late. Otherwise:

Turn Switch 4 to alternate position
Depress R/S
Return Switch 4 to original position
Retype entire item
Depress R/S

The position of Switch 1 may be changed so that part of the source program may be entered on cards and part on the typewriter.

e) Error messages are typed on the console typewriter and follow the form indicated in Section III.6c.

f) Reinitialization - only needed when the typewriter did not type COMPILATION after START was pressed twice.

Set Switch 3 OFF for compilation of a new program, ON for reexecution of previous program.

Press: INST. STOP
RESET
INSERT
RELEASE
START
START

Sense Switches 1 or 2 should be set to control input devices and listing (See 2a, "Compiling the source statements")

Step 3. Program execution

a) When the END statement is compiled, if no Error Messages have been typed, program execution will begin.

b) Switches 1, 2 and 3 are available for use during program execution. If the program requires that Switch 1, 2 or 3 be reset, it is advisable to use a PAUSE statement before the first executable statement in the program. The PAUSE will give the user time to reset the switches.

c) Switch 4 is set ON to trace. Switch 4 is set ON to correct errors in typed input as described in step 2b above.

d) The program execution may be stopped at any time by pressing INSTANT STOP.

III.7c Trace feature - User processing III.7c

The object program may be traced at any time by turning Switch 4 ON and running the program. The result of each arithmetic statement will then be typed preceded by the work "TRACE". Normal output will not be inhibited.

Note that tracing is time-consuming and should be used sparingly in pursuit of an elusive bug. The user should consider inserting PUNCH statements in the source program so that he will have the information necessary to de-bug his program. (See sample problem 4, III.10)

Switch 4 may be turned ON or OFF at any time during the running of the program, to cause only selected parts of the program to be traced. However, unless care is exercised, it will be difficult to tell what part of the program is being traced.

1.8 ERROR MESSAGES DURING COMPILATION

TABLE 1

COMPILATION ERROR CODES FOR N.C.E. LOAD GO. (REVISED)

ERROR 11	UNRECOGNIZABLE STATEMENT, INCORRECT SPELLING, ETC.
ERROR 12	MIXED MODE, OR MISSING RIGHT PARENTHESIS IN IF STATEMENT
ERROR 13	MISPLACED OR MISSING COMMA OR EQUAL SIGN
ERROR 14	MISPLACED OR MISSING PARENTHESIS
ERROR 15	MISPLACED OR MISSING VARIABLE OR OPERAND, OR ILLEGAL SPECIAL CHARACTER SEQUENCE
ERROR 20	STATEMENT NUMBER USED MORE THAN ONCE
ERROR 21	UNDEFINED STATEMENT NUMBER CALLED .....
ERROR 22	WRONG NUMBER OF PARAMETERS ON A TRANSFER STATEMENT, OR INVALID INDEX ON COMPUTED GO TO STATEMENT.
ERROR 23	CONSTANT IN UNACCEPTABLE POSITION, OR STATEMENT NO. IS ZERO
ERROR 24	MISSING OR UNACCEPTABLE STATEMENT NUMBER OR ARRAY SIZE, OR CONSTANT IS IN UNACCEPTABLE FORM.
ERROR 30	DIMENSIONED VARIABLE USED WITHOUT SUBSCRIPT,OR SUBSCRIPTED VARIABLE HAS NOT BEEN DIMENSIONED, OR INCORRECT FUNCTION NAME, OR TRIG FUNCTIONS WERE ELIMINATED.
ERROR 31	SUBSCRIPT IN INCORRECT FORM OR MISSING OPERATOR
ERROR 32	FUNCTION NAME USED IN NON-ARITHMETIC STATEMENT
ERROR 40	DO STATEMENT IS INCORRECTLY FORMED
ERROR 41	DO LOOPS INCORRECTLY NESTED, OR ERROR IN LAST STATEMENT OF LOC
ERROR 42	DO LOOP ENDS WITH AN IF,GO TO, COMPUTED GO TO, DO, STOP, OR END STATEMENT
ERROR 43	MORE THAN SEVEN LEVELS OF DO LOOP NESTING ENCOUNTERED
ERROR 50	VARIABLE NAME CONSISTS OF MORE THAN FIVE CHARACTERS
ERROR 51	RAISING FIXED POINT QUANTITY TO A POWER
ERROR 52	A**B**C. PARENTHESES MUST BE ADDED TO INDICATE ORDER
ERROR 53	SOURCE PROGRAM IS TOO LARGE TO COMPILE

NOTE:

ERRORS 21 and 41 can only be detected at the end of the compilation process. Thus when ERROR 41 occurs no statement number will be typed out.

When ERROR 21 occurs the undefined statement number is typed out. No indication is given of the statement in which the undefined statement number is referenced.

TABLE 2

EXECUTION ERROR CODES FOR N.C.E. LOAD AND GO (REVISED)

```
    ERROR 60  DIVISION BY ZERO
                   RESULT  ---  9.9999999E+48, OR 9999
    ERROR 61  LOG(A)            ------------     WITH A = 0
                   RESULT  ---  0.0000000
    ERROR 62  A**B, LOG(A), OR SQRT(A)  ----     WITH A NEGATIVE
                   RESULT  ---   FUNCTION OF ABS(A)
    ERROR 63  CALCULATED EXPONENT GREATER THAN +49
                   RESULT  ---  9.9999999E+48
    ERROR 64  CALCULATED EXPONENT LESS THAN -50
                   RESULT  ---  0.0000000
    ERROR 65  SIN(A), COS(A)   ----    WITH A GREATER THAN 1.E9 RADIANS
                   RESULT  ---  1.0000000

*   ERROR 70  SUBSCRIPT ON VARIABLE EXCEEDS SIZE OF DIMENSIONED ARRAY,
                   OR IS NOT POSITIVE NUMBER, OR THE INDEX OF A
                   COMPUTED GO TO IS OUT OF RANGE
*   ERROR 71  UNDEFINED VARIABLE
*   ERROR 72  UNACCEPTABLE NUMBER IN INPUT DATA
**  ERROR 73  EXECUTION HAS TAKEN TOO LONG,PROGRAM MAY BE IN A LOOP
**  ERROR 74  PROGRAM CALLS FOR MORE DATA THAN INCLUDED WITH CARD DECK,OR
                   END STATEMENT IS MISSING, OR MORE THAN ONE CONTROL CARD
```

* IF ERRORS 70, 71, OR 72 OCCUR DURING USER PROCESSING, THE COMPUTER WILL HALT. SET SWITCH 3 OFF AND PUSH START TO COMPILE A NEW SOURCE PROGRAM. PUSHING START WITH SWITCH 3 ON WILL CAUSE THE REEXECUTION OF THE PROGRAM WHICH CAUSED THE ERROR.

** ERRORS 73 AND 74 ARE INDICATED ONLY DURING BATCH PROCESSING.

A CHECK STOP MAY OCCUR IF THE SOURCE CARDS ARE PUNCHED WITH INVALID CHARACTERS. REINITIALIZATION WILL BE NECESSARY.

III.10 SAMPLE PROBLEMS

THE FOLLOWING SOURCE DECKS ARE PREPARED FOR BATCH PROCESSING AND WERE PROCESSED USING VERSION THREE OF THE PROCESSOR

```
$ SAMPLE PROBLEM 1

C  PROGRAM ILLUSTRATES HOW ALPHAMERIC DATA MAY BE USED AS OUTPUT
READ,A,B,C
PUNCH,A,B,C
END
P          THIS IS A TEST PROGRAM
P      A                   B                   C
2.0
4.0
6.999
```

```
$ SAMPLE PROBLEM 2

C  TO READ IN AND PUNCH OUT A ONE DIMENSIONAL ARRAY THE FOLLOWING
C  SOURCE STATEMENTS MAY BE USED
C  N, SIZE OF ARRAY, IS PUNCHED ON FIRST DATA CARD
DIMENSION A(10)
50 READ,N
C  THE LAST DATA CARD IS PUNCHED 9999
IF (N-9999) 20,30,20
20 DO 10 I=1,N
READ,A(I)
10 PUNCH,A(I)
GO TO 50
30 END
P        OUTPUT
P
3
4.5
-3.2
2156
6
15 .009 9456 -36.5 999.99 +15
9999
```

```
$ SAMPLE PROBLEM 3
EXECUTION

        OUTPUT

  ROW                COL.              NUMBER
  0001               0001              5.1000000
  0001               0002              5.2000000
  0001               0003              5.3000000
  0002               0001              6.1000000
  0002               0002              7.2000000
  0002               0003              60.400000
  0003               0001              300.00000
  0003               0002              22.600000
  0003               0003             -34.400000
  0004               0001              51.000000
  0004               0002              6.3600000
  0004               0003              .22000000
  0001               0001              5.3600000
  0001               0002              9.1100000
  0001               0003              10.200000
  0001               0004              11.300000
  0001               0005              11.500000
  0002               0001              6.2000000
  0002               0002              8.3000000
  0002               0003              9.3000000
  0002               0004              6.5600000
  0002               0005              700.82000
  0001               0001             -834.22678
  0001               0002              111.50000
  0002               0001             -6.8000000E-03
  0002               0002              5.4000000
$ SAMPLE PROBLEM 4
EXECUTION
        OUTPUT          SAMPLE PROBLEM 4

 GROUP NUMBER   GROUP SUM
   0001           40.000000
   0001           70.000000
   0001           94.000000
   0001           119.00000
   0002           26.000000
   0002           53.000000
   0002           93.000000
   0002           130.00000
   0003           25.000000
   0003           48.000000
   0003           70.000000
   0003           120.00000
   0002           130.00000
```

THE FOLLOWING SOURCE DECKS ARE PREPARED FOR USER PROCESSING AND WERE PROCESSED USING VERSION TWO OF THE PROCESSOR.

SAMPLE PROBLEM 5

```
C  PROGRAM CALCULATES THE FORMULA C(N,R)=N /R (N-R)
C  FOR GIVEN VALUES OF N,R
C  N IS ANY INTEGER GREATER THAN I AND LESS THAN 50
C  R IS ANY INTEGER LESS THAN N
C  VALUES OF N,R PUNCHED ON SAME CARD
C  LAST DATA CARD PUNCHED 50 IN COL.1,2
01 I=0
READ,XN,R
C  CHECKS TO SEE THAT THE DATA IS WITHIN THE ALLOWABLE RANGE
IF (XN-50.)10,98,99
10 IF(XN-1.)99,99,05
05 XNR=XN-R
IF (XNR)99,99,11
C  SIMPLE TRANSFER STATEMENTS CARRY THE ARGUMENT TO THE SUB-ROUTINE
11 L=XN
20 I=I+1
GO TO 30
15 XNF=FAC
L=R
GO TO 20
16RF=FAC
L=XNR
GO TO 20
17XNRF=FAC
CRN=XNF/(RF*XNRF)
97 PUNCH,XN,R,CRN
GO TO 01
C  SUB-PROGRAM COMPUTES THE FACTORIALS
30FAC=1.
DO 50 J=1,L
Y=J
50 FAC=FAC*Y
GO TO (15,16,17),I
99 STOP
98 END
P
5 3
3 1
19 10
50 32
```

```
   5.0000000       3.0000000       10.000000
   3.0000000       1.0000000       3.0000000
   19.000000       10.000000       92377.987
```

THE FOLLOWING SOURCE DECKS ARE PREPARED FOR USER PROCESSING AND WERE PROCESSED USING VERSION TWO OF THE PROCESSOR.

SAMPLE PROBLEM 6

```
C   SOLUTION OF SIMULTANEOUS EQUATIONS
C   BY GAUSS-JORDAN ELIMINATION
C      MODIFICATION OF USER S GROUP LIBRARY PROGRAM  5.0.007          0003
C   THE PAUSE AT THE BEGINNING OF THE PROGRAM ALLOWS THE OPERATOR
C   TO SET SENSE SWITCH 1
C       SWITCH 1    ON.PUNCH          OFF,  PRINT                       0004
C       SWITCH 2    ON.STOP ON TOLERANCE CHECK        OFF,  CONTINUE     0005
C      ENTER TOLERANCE AND SIZE OF MATRIX AS FIRST PIECE OF DATA        0006
C   17 EQUATIONS AND TRIG FUNCTIONS NEED NOT BE ELIMINATED.             0008
C   WITH THIS SIZE DIMENSION THE PROGRAM WILL HANDLE
       DIMENSION A(17,18)
       PAUSE
10      READ  ,TOLR,N2                                                  0010
       N1=N2+1                                                          0011
       DO 2 I=1,N2                                                      0012
       DO 2 J=1,N1                                                      0013
2      READ  ,A(I,J)
       DO 14 I=1,N2                                                     0016
       DIAG=A(I,I)
       IF (DIAG)  4, 20, 4                                              0020
4      IF(ABS(DIAG)-TOLR) 19,19,5                                       0021
5      DO 6 J=I,N1                                                      0022
6      A(I,J)= A(I,J)/DIAG
       K=1                                                              0025
9      IF (K-I) 11,13,11                                                0026
11     FCTR=A(K,I)
       DO 12 J=I,N1                                                     0029
12     A(K,J)=A(K,J) -FCTR*A(I,J)
13     K=K+1                                                            0033
       IF (K-N2) 9,9,14                                                 0034
14     CONTINUE                                                         0035
       J=N1
       IF(SENSE SWITCH 1) 15,17                                         0037
15     DO 16 I=1,N2
16     PUNCH ,A(I,J)
       GO TO 10                                                         0039
17     DO 18 I=1,N2
18     PRINT ,A(I,J)
       GO TO 10                                                         0041
                                                                        0042
C       STATEMENT  19 IS TOLERANCE STOP                                 0043
19     PAUSE                                                            0044
       IF (SENSE SWITCH 2) 10,5                                         0045
                                                                        0046
C       STATEMENT 20 IS ERROR STOP                                      0047
20     STOP                                                             0048
       END                                                              0057
```

IV KINGSTON FORTRAN LANGUAGE SPECIFICATIONS IV

The Kingston Fortran II system, composed of a language and compiler for the IBM 1620, was written in July 1964 and revised in November 1964, by J.A.A. Field[1], D.A. Jardine[2], E.S. Lee[1], J.A.N. Lee[3] and D.G. Robinson[2]. The KFII compiler allows the following operations and specifications, which are not discussed in Section 2 and not permitted by other Fortran compilers:

- Use of stored Hollerith constants.
- Explicit specification of variable types.
- Use of up to 13 subscripts.
- Use of integer expressions in indexing Computed GO TO statements, and DO statements.
- Use of DATA specifications.
- Optional free format.

A complete listing of library subroutines may be found in Section IV.9a. The KFII compiler includes the following library subroutines, which are in addition to the standard library subroutines provided by other Fortran systems:

- MAX which chooses the largest value in a group of values.
- MIN which chooses the smallest value in a group of values.
- PLOT and PLOTP which plot the values of real variables on punch cards or the console typewriter.
- SORT which sorts elements in an array using the Shell Method.

The KFII compiler, stored on the 1311 disk of the 1620 Model II and used with appropriate IBM Monitor System control cards, permits convenient and quick one pass compilation and execution of a source deck written in KFII language. The discussion of the KFII language which follows includes only those language specifications which apply specifically to KFII, and assumes that the reader is familiar with the language discussed in Section II.

---

1. Dept. of Electrical Engineering, University of Toronto, Toronto, Ontario.

2. Du Pont of Canada Ltd., Research Centre, Kingston, Ontario.

3. Computing Centre, Queen's University, Kingston, Ontario.

IV.1 Punching input source programs IV.1

The statements of a Kingston Fortran II source program may be punched in columns 7-72 of a source program card. If the statement is too long for one card, it may be continued on the following cards. These continuation cards must have a non-zero number in column 6. The first card of a statement must have column 6 either blank or zero. Blanks in columns 7-72 are ignored except for Hollerith specifications, discussed below.

Columns 1-5 on the first card of a statement are used for the statement number, if any.

Columns 73-80 are not used by the compiler and may be used for program identification, sequencing, or any other purpose.

Comments to explain the program may be punched in columns 2-80 of a card, if the letter C is placed in column 1.

IV.2 The arithmetic statement IV.2

IV.2a Constants IV.2a

Integer constants - The magnitude of an integer constant must not be greater than 99999.

Hollerith constants - A Hollerith constant consists of any characters, $1 \leq n \leq 5$, including blanks and special characters. It is written with the integer n, followed by the letter H, followed by exactly n characters.

Example:

Valid	Invalid
1HA	6HABCDEF
4H(*/-	OH
3HbbC	4HbbbbD

Note: The b indicates blanks.

Hollerith constants or variables whose values are Hollerith constants, carry symbolic rather than numeric information. They may be used in the following statements:

1. Simple replacement statements may define a Hollerith constant

Example:

```
I = 3HYXA
A = 4H1234
```

2. Data statements may define a Hollerith constant.

(See Section IV. 7d)

3. Input, output statements, using A-format may define a Hollerith constant.

Example:

```
      READ 2, (TABLE (J),J = 1,5)
    2 FORMAT (5A5)
```

The above statements will read the 5 table values, SMITH, JONES, TERRY, ALPHA, DATUM, into machine core storage labeled TABLE (1), TABLE (2), TABLE (3), TABLE (4), and TABLE (5).

(See Section IV.5b)

4. Arithmetic statements may include Hollerith constants providing the only arithmetic operations are integer subtraction or addition. Due to the machine representation of the blank as zero, the following equations are correct and may be used in a source program to form Hollerith constants.

Example:

```
      5HABCDE = 5Hbbbbb + 5HABCDE
         2H1A = 2H1b + 2HbA
         2HbA = 2H1A - 2H1b
```

5. IF statements may compare two Hollerith constants, or variables whose values are Hollerith constants, for identity.

Example:

```
      If (LIST (1)-ITEM(2)) 10,20,10
      Where LIST and ITEM contain symbolic information.
```

6. CALL statements may include Hollerith constants as function arguments.

(See Section IV.8c)

7. DO statements may use Hollerith constants as the initial and increment values of the index. Due to the machine representation of an alphabetic character, the following DO statement causes the Index I to assume all the possible letters of the alphabet in order.

Example:

```
      Do  n  I = 1HA, 1HZ, 1HB - 1HA
```

Thus the user can sort cards alphabetically.

IV.2b Variables

Variable names- A variable name consists of 1-6 alphameric (numeric or alphabetic) characters.

Variable types- The type of a variable, integer or real, can be specified in two ways: implicitly or explicitly. Implicit specification is discussed in Section II. Explicit specification of a variable type requires using an INTEGER or REAL statement.

Example:

```
INTEGER DEV, ITA
REAL ITEM, FAN, LIST
```

The INTEGER and REAL statements are used to overide the implicit type assignment associated with a variable name. The INTEGER statement declares all variables in its list to be of type integer regardless of their initial letters. Similarly, the REAL statement declares all variables in its list to be of type REAL, regardless of their initial letters. Variables listed in an INTEGER or REAL statement remain that type throughout the program; the type can not be changed. The INTEGER or REAL statement is a type specification statement and must precede the first executable statement of the program, but must follow COMMON, DIMENSION and EQUIVALENCE statement. (See "Order of appearance of specification statements," Section IV.7e)

Subscripted variables- KFII allows up to thirteen subscripts. Subscripts may take the form of any arithmetic expression whatsoever provided that the result of evaluation of the subscript be an integer number. In particular, the subscript may itself contain subscripted variables, whose subscripts, in turn, may be expressions involving subscripted variables.

Example:

Valid subscripts	Valid subscripted variables
(I)	A(I)
(3)	K(3)
(2+MU)	ALPHA(I,J, 2+MU)
(MU+2)	RUN(MU*5+M, 4*J(K(2)-L+M),K(N(M))
(J*5+M)	
(5*J)	
(6*J-K+2-10/L+M)	
(4*J(K+2-L+M)+K(M(N+2))/3)	
*(FIXF(A*B*2.0**C)*L/2)	

Invalid subscripted variables

```
X(A,I)
A(I,J*2.5)
```

*(See library function FIXF,IV.9a)

IV.2c Arithmetic expression IV.2c

Rules for forming expressions: (See Section II. for general Fortran rules for forming arithmetic expressions.)

1. E = -A**B is compiled, $E = -(A^B)$

2. Within the same priority, the addition, subtraction, and multiplication operations are performed from right to left. For example, A+B+C+D will be performed as (D+C+B+A); whereas (A+B)+(C+D) will be performed as the sum of (D+C) and (B+A). However A*B/C*D will be calculated as: A*(B/C)*D.

IV.3 Control statements IV.3

IV.3a Statement numbers IV.3a

Statement numbers may be any integer n, $0 < n \leq 99999$.

IV.3b Address variables IV.3b

An address variable is a variable which has been made equivalent to a statement number. The ASSIGN statement, assigns the statement number to the address variable.

General form of ASSIGN statement:

ASSIGN i to n

where: i is a statement number or address variable.
n is an address variable.

An address variable must be defined ultimately in terms of a statement number. Thus if I is an address variable, it must at the time of execution of an ASSIGN statement, have previously been defined in terms of a statement number or another address variable which was defined in terms of a statement number. The contents of storage assigned to the address variable is not the statement itself, but rather the object time representation of the statement number. Address variables may be subscripted, if desired. If an address variable is assigned to another variable it must be enclosed in parenthesis.

Example:

Valid ASSIGN statements

```
ASSIGN  12  to K
ASSIGN  (K) to J(L)
ASSIGN  13  to A(M(N))
ASSIGN  (A(M(N))) to K
```

Rules for using address variables

1. Address variables may appear in the following control statements in place of a statement number:

Statement	Example
UNCONDITIONAL GO TO	GO TO K
COMPUTED GO TO	GO TO (10,K, 30, L(M), 15,35), ITEM(J)
ARITHMETIC IF	IF(A(J,K)-B)10,4,L
IF SENSE SWITCH	IF(SENSE SWITCH 1) N,A

2. Address variables may not appear in a DO statement.

3. Address variables may be used as the FORMAT designator in an input/output statement. Address variables may be reassigned within a program.

(See Section IV.4)

4. Address variables may be transmitted to a subprogram in a CALL statement. Thus address variables may be used to provide multiple returns from a subroutine, each to a different point.

(See Section IV.8c)

5. Address variables may be defined in a DATA specification statement. They may not be used in any other type of specification statement.

6. Arithmetic may not be performed on Address variables. They may not be used in an arithmetic statement.

IV.3c Computed GO TO statement

GO TO (x_1, x_2, x_3, ...x_n), i

where: x_1, x_2, x_3, ...x_n are statement numbers or address variables.

i is an integer expression of any complexity whose value is greater than or equal to 1 and less than or equal to the number of statement numbers or address variables within the parentheses. The comma preceding i is optional.

Example: GO TO (10,K, 30, L(M), 15, 35), ITEM(J)

If the value of ITEM(J) is 3 at the time of execution, a transfer occurs to the statement whose number is the third in the series. If the value of ITEM(J) is 4, a transfer occurs to the statement whose number is fourth in the series, address variable L(M).

IV.3d **IF SENSE SWITCH statement** IV.3d

IF (SENSE SWITCH i) n_1, n_2

where: i is an integer constant or arithmetic expression
n_1, n_2 are statement numbers or address variables.

The last two digits of the integer constant or expression i are used to determine which machine indicator is to be interrogated. Control is transferred to statement n_1 if the machine indicator is on. Any of the machine indicators can be interrogated by the IF (SENSE SWITCH) statement. However, not all machine indicators are relevant to the computations performed by the object program. Care should be exercised in using the IF (SENSE SWITCH) statement since the operating system subroutines may leave the indicators in a position which does not correspond to the result of the arithmetic calculation.

IV.3e **DO statement** IV.3e

	End of Range	Index		Initial Value	Test Value	Increment
DO	n	i	=	m_1,	m_2,	m_3

where: n is any statement number, but NOT an address variable.

i is a subscripted or nonsubscripted integer variable.

m_1, m_2, m_3 are signed or unsigned integer constants, subscripted or non-subscripted or integer expressions of any desired complexity.

m_3 is optional; if it is omitted, its value is assumed to be 1. In this case, the preceding comma must also be omitted.

Rules for using the DO statement

1. The range is the series of statement to be executed repeatedly. The range can consist of any number of statements.

2. The values of the index, test value, or increment, (i, m_1 , m_2 , m_3 ,) may be changed within the DO loop if and only if they are simple variables. The DO will then be continued with the new values, and normal incrementing will occur until an exit from the range of the DO takes place.

3. The initial value, m_1, and test value, m_2, may be positive, negative, or zero. The normal algebraic sign convention is applied for incrementing and testing.

4. The increment, m_3, may be positive or negative, but not zero.

Example: IV.3e

The statement DO 20 I = 5, -4, -3

will cause the range of the DO to be executed with I taking on the successive values 5, 2, -1, -4.

Example:

The statement DO 100 I(J) = L*M+2, 6*K, N(K)

will cause the range to be executed with I(J) taking on values starting at the value of L*M+2, and continuing with increments of the value of N(K) until the value of 6*K is exceeded in the direction of incrementation.

5. Transfer into the range of a DO statement is permitted if a previous transfer has left the range of the DO and it is desired to return to the range of the DO.

IV.3f PAUSE statement IV.3f

PAUSE n

where: n is an unsigned integer constant, or an integer variable or expression, n is optional.

The PAUSE statement causes the program to halt. PAUSE n is typed on the console typewriter. If n is omitted, the program is halted and there is no typewriter output. Pushing START causes the program to resume execution, starting at the next statement after the PAUSE statement.

IV.3g STOP statement, CALL EXIT, CALL SKIP IV.3g

STOP n

Where: n is an unsigned integer constant, or an integer variable or expression, n is optional.

The STOP statement causes the program to print STOP 0000 on the typewriter if n is not specified. If n is specified, STOP n is printed. In either case, the execution of the program is terminated, and may not be resumed.

CALL EXIT

The CALL EXIT statement halts the object program and returns control to the supervisor so that another source program may be compiled. The

CALL EXIT statement must be the last executable statement in a program written for the IN-OUT box unless a CALL SKIP is used as specified below. IV.3g

CALL SKIP

The CALL SKIP statement causes interruption of the normal program. The CALL SKIP will usually be employed to stop calculation on a block of data because of an abnormal situation (e.g. failure to converge on an iteration, bad data) which has occurred in the block of data. In such a case, CALL SKIP will cause that particular calculation to be abandoned, and a new set of data to be presented to the program. The data must be sectioned by end of file cards.

General form: End of file card

$$$
where: $ signs are punched in card cols, 1-3.

The end of file card indicates the beginning of a new block of data. If the data abandoned is the last block of data, a normal exit to a new program will result. This is the only instance where a CALL EXIT is not the last executable statement in a program.

IV.3h END statement IV.3h

END

The END statement defines the end of a program or subprogram for the compiler. Physically it must be the last statement of each program or subprogram. When it is encountered in the flow of the source program, compilation halts and any source program cards following the END card are not compiled.

The END statement is not executable. The last executable statement before the END statement must be a transfer statement (IF, GO TO, STOP, CALL, or RETURN). An END statement may not have a statement number.

IV.4 INPUT/OUTPUT statements IV.4

IV.4a Input statements IV.4a

READ n, list	Cards
ACCEPT TAPE n, list	Paper tape
ACCEPT n, list	Console typewriter
REREAD n, list	Causes the last record read (regardless of input device) to be read again.

where: n is a statement number, address variable, or the name of an array containing the format in the form of Hollerith constants, representing the FORMAT statement describing the type of data conversion. n is optional. If omitted, the system will supply a standard format.

(See section IV.6 KFII without Format)

List is a list of variable names, separated by commas, representing the input data.

IV.4b Array input IV.4b

1. When an array name appears in an I/O list in non-subscripted form, all of the quantities in the array are transmitted. If the list item is a multi-dimension array, it is transmitted columnwise, with the first subscript varying most rapidly, and the last subscript least rapidly.

Example:

```
Dimension C(10)
Read 40,C
```

The above statements will cause all of the quantities C(1)....C(10) to be read into storage.

Given D is a 3 x 5 x 5 array

Example:

```
Dimension D(3,5,5)
Read 40,D
```

The above statements will cause all of the quantities, D_{111}....D_{211}.... D_{215}....D_{311}....D_{351}....D_{355} to be read into storage in the following order:

D_{111}, D_{211}, D_{311}, D_{121}, D_{221}, D_{321}, D_{131}, etc.

2. Indexing I/O lists - Variables within an I/O list may be indexed and incremented in the same manner as with a DO statement. For example, suppose it is desired to read data into the first five positions of the array A. This may be accomplished by using an indexed list, as follows:

```
READ 50, (A(I),I = 1,5)
```

This is equivalent to the following:

```
READ 50,A(1),A(2),A(3),A(4),A(5)
```

As with DO statements, a third indexing parameter may be used to specify the amount by which the index is to be incremented at each iteration. Thus: IV.4b

```
READ 50,(A(I),I = 1,10,2)
```

causes transmission of values for A(1),A(3),A(5),A(7), and A(9).

Furthermore, this notation may be nested. For example, the list:

```
((C(I,J),D(I,J),J = 1,5),I = 1,4)
```

would transmit data in the following order:

```
C(1,1),D(1,1),C(1,2),D(1,2),...,C(1,5),D(1,5)
C(2,1),D(2,1),C(2,2),D(2,2),...,C(2,5),D(2,5)
C(3,1),D(3,1),C(3,2),D(3,2),...,C(3,5),D(3,5)
C(4,1),D(4,1),C(4,2),D(4,2),...,C(4,5),D(4,5)
```

The notation for the implied DO statement in an I/O list may be of the same complexity as that described earlier for the DO statement proper. In particular, the indexing variable may itself be subscripted, and the limits may be integer expressions. For example, the following are permitted:

```
READ 10,( (A(I,J), I = K,L), J = M,N)
READ 10,( (A(I(K1),J(M1)),K1 = K-JOB*2,L+5,-J6), M1 = M*8-MM9,N,3*N18)
```

Restriction

In an input list, the items may be only subscripted or nonsubscripted variables or array names. All variables in an implied DO statement must be in the DO loop. Thus the following example is invalid:

```
READ 50, DOG,(A(I),I = 1, 10, 2).
```

3. Sample problem 7 (Section IV.14) illustrates array input and output.

IV.4c Output statements IV.4c

PUNCH n, list	Cards
PUNCH TAPE n, list	Paper tape
TYPE n, list	Console typewriter
PRINT n, list	1443 On-line printer

where: n is a statement number, address variable, or the name of an array containing the FORMAT statement in the form of Hollerith constants. If n is omitted, the system will supply a standard format.

(See section IV.6: KFII without format)

Where: Iw, Fw.d, Ew.d, wX, Iw, Aw, represent data conversion codes separated by commas. IV.5
/ represents the beginning of a new record.

IV.5a Numeric conversion codes IV.5a

I-conversion- is used to input or output an integer quantity as follows:

Iw

Where: w represents the number of spaces that are scanned on input or reserved for the number on output.

1. If the number to be output is greater than w spaces, the excess is lost and an error indication results.

2. If the number to be output has less than w digits, the left-most spaces are filled with blanks. Blanks in input data are regarded as zeros.

3. A positive sign need not be punched on input. Space need not be left for a positive sign on output. However, the space preceding the left-most digit must be reserved for sign, if the quantity to be output is negative.

4. If a real number is output under I-conversion, the integer part is punched without rounding. Sufficient width must be allowed for the resulting integer number.

Example:

specification I3 will punch the internal values as follows:

Internal value	Punched
721	721
-721	error message
-12	-12
68114	error message
4336.15	error message
-43.72	-43

F-conversion- is used to input or output a number with decimal.

General form of F-conversion code:

Fw.d

where: w is the total field reserved on output or scanned on input.
d is the number of places to the right of the decimal.

1. Numbers for E-conversion input need not be punched with four spaces devoted to the exponent field. The start of the exponent field may be marked by an E, or by a plus or minus (not a blank--all blanks in fields are read as zeros).

   Example:
   .3E2, .3E+2, .3+2, .3+02 are all valid input data.

2. The total field width, reserved for output must include a space for sign if the number is negative, a space for the decimal point, and four spaces for the exponent.

3. The decimal portion is rounded if insufficient spaces are reserved on output.

4. If an integer number is handled with E-conversion, the integer number is changed to the corresponding real number before E-conversion takes place.

   Example:
   Specification E10.3 punches the internal values as follows:

Internal Value	Punched
238.	bb.238E+03
-.002	b-.200E-02
.0000000004	bb.400E-9
-21.0057	b-.210E+02

N-conversion- is used for input which is punched "free form" and will supply a standard format on output. N-conversion neither permits nor allows width or decimal point location specification.

General form of N-conversion code:

(xN)

where: x is the number of variables in the input or output list

1. Input data may be any type; integer, real, or E, punched with one blank separating each number. The internal form of the number is entirely determined by the Type of the variables in the input list.

   Example:

```
READ 10, X, I, N, Y
10 FORMAT (4 N)
```

The card is punched:		The numbers are read as follows:
Card Cols.	Contents	
1-3	462	X = 462
4	b	I = 2
5-6	b2	N = -398
7	b	Y = 539.3218
8-11	-398	
12	b	
13-20	539.3218	

2. On output, N-conversion is equivalent to 1PE14.7, 1X for real numbers.

(See "Scale factors" below)

and I6,1X for integer numbers.

Example:

```
    PUNCH 10, Y, I
 10 FORMAT (2N) will produce
    the following output:
    5.6300000E+02bbbb-21
```

Internal Value

Y = 563
I = -21

Scale factors

To permit more general use of E-, and F-conversion, a scale factor followed by the letter P may precede the specification. The magnitude of the scale factor must be between -49 and +49 inclusive. The scale factor is defined for input as follows:

$$10^{\text{scale factor}} \times \text{external quantity} = \text{internal quantity}$$

The scale factor is defined for output as follows:

$$\text{external quantity} = \text{internal quantity} \times 10^{\text{scale factor}}$$

For input, scale factors have effect only on F-conversion. For example, if input data are in the form xx.xxxx and it is desired to use it internally in the form .xxxxxx, then the FORMAT specification to effect this change is 2PF7.4 For output, scale factors may be used with E-, and F-conversion.

For example, the statement FORMAT (I2,3F11.3) might give the following printed line:

27bbbb-93.209bbbbb-0.008bbbbb0.554

but the statement FORMAT (I2,1P3F11.3), used with the same data, would give the following line:

27bbb-932.094bbbbb-0.076bbbbbb5.536 IV.5a

Whereas, the statement FORMAT (I2,-1P3F11.3) would give the following line:

27bbbbb-9.321bbbb-0.001bbbbbb0.055

A positive scale factor used for output with E-conversion increases the number and decreases the exponent. Thus, with the same data, FORMAT (I2,1P3E12.4) would produce the following line:

27b-9.3209Eb01b-7.5804E-03bb5.5536E-01

The scale factor is assumed to be zero if no other value has been given. However, once a value has been given, it will hold for all E-, and F-conversions following the scale factor within the same FORMAT statement. This applies to both single-record formats and multiple-record formats. Once the scale factor has been given, a subsequent scale factor of zero in the same FORMAT statement must be specified by OP. Scale factors have no effect on I-conversion or N-conversion.

IV.5b Alphameric conversion codes IV.5b

There are two specifications available for input/output of alphameric information: H-Specification and A-Conversion. H-Specification is used for alphameric data which are not going to be changed by the object program (e.g. page headings); A-Conversion may be used for alphameric data in storage which are to be operated on by the program (e.g. modifying a line to be printed).

H-type FORMAT specification

H-type specification is written within the FORMAT statement and is preceded by nH where n is the number of characters in the specification. For example:

```
25  FORMAT  (15HbTHISbISbH-TYPE)
```

The effect of this statement depends on whether it is used with an input or output statement. A comma separating the H-type specification from a succeeding specification, is optional.

Output: All characters (including blanks) within the specification are written as part of the output record. Thus, the statements:

```
5 FORMAT  (27HbTHISbISbISbALPHAMERICbDATA)
  .
  .
  .
  PRINT   5
```

would cause the following record to be written on the printer: IV.5b

THIS IS ALPHAMERIC DATA

Input: A number of characters, equal to the number, n, of characters specified, are read from the designated input record and replace, in storage, the characters within the H-Specification. For example, the statements:

```
5    FORMAT  (8HHEADINGS)
     .
     .
     .
     READ  5
```

would cause the first eight characters to be read from the next input card and these characters would replace the characters HEADINGS in the FORMAT statement.

Restriction: The number of characters in a single H-Specification must not be greater than 99.

Note: If a Hollerith specification extends beyond the end of the source statement card on which it was started, it may be completed on a continuation card. In this case, the first card is considered to end at column 72.

A-Conversion

The specification Aw is used to read or write alphameric data. w must be 1,2,3,4, or 5. It causes the w characters to be read into, or written from, the area of storage specified in the I/O list. For example, if a data card having the characters ABCD in columns 1-4 were read under control of the following statements

```
10   FORMAT (A4)
     .
     .
     .
     READ 10, SAM
```

the four alphameric characters ABCD would be read from the card and placed into the field in storage named SAM.

The following statements:

```
15   FORMAT (3HXY=,F9.3,A4/)
     .
     .
     .
     PUNCH 15,A,SAM,B,SAM
```

would produce the following lines: IV.5b

XY = 5976.214ABCD
XY = 6173.928ABCD

Characters transmitted under A-conversion are stored in memory as Hollerith constants. Conversely, a Hollerith constant, or a variable whose value is a Hollerith constant, may be output using A-conversion.

IV.5c Specifying blank fields IV.5c

X-conversion provides for blank characters in an output record, and skipping of characters in an input record.

General Form of X-conversion code:

wX

Where: w characters are skipped on an input record, or w blanks provided in an output record.

1. X-conversion must be carefully distinguished from H-specification with blank characters. Reading an input record under X-conversion causes the appropriate part of the record under X-conversion to be ignored completely.

   Example:

Card is punched:		
Col. 1-5	543.2	Read 5, A, I
6-8	423	5 FORMAT (F5.1, 3XI4)
9-12	3233	will result in:
		A=543.2
		I=3233

IV.5d Repeating specifications IV.5d

A specification may be repeated as many times as desired (within the limits of the output device) by preceding the specification with an unsigned integer constant.
Thus:
2F10.4
is equivalent to:
F10.4, F10.4

Parenthetical expressions are permitted to enable repetition of data fields according to certain format specifications within a longer FORMAT statement.

Thus: IV.5d

10 FORMAT (2(F10.6, E10.2),I4)

is equivalent to:

10 FORMAT (F10.6, E10.2, F10.6, E10.2, I4)

Five levels of nested parentheses, in addition to the parentheses, required by the FORMAT statement, are permitted.

If there are more items in the list than there are specifications in the FORMAT statement, control transfers to the immediately preceding left parenthesis of the FORMAT statemnt. A new card (or line) is punched with the specifications used again for the next item in the list.

Example:

The following statements:

```
10 FORMAT (F10.3, E12.4, F12.2)
   .
   .
   .
   PUNCH 10, A, B, C, D, E, F, G
```

cause the data to be transmitted in the following order:

Data Transmitted	Specification	
A)	F10.3)	
B)	E12.4)	First card
C)	F12.2)	
D)	F10.3)	
E)	E12.4)	Second card
F)	F12.2)	
G	F10.3	Third card

IV.5e The use of the slash (/) IV.5e

1. The / may be used to denote the end of a record. On input the / calls for the reading of the next card.

Example:

```
5 FORMAT (F5.2, /F10.2)
READ 5, A, B,
```

A is read from the first card, B from the second card.

On output the / calls for the punching (or typing) of a new card.

2. The / may also be used to provide blank lines between output records or records skipped for input records.

For example, if the statement FORMAT (I2,E12.4////F12.3) is used for printed output, three blank lines will be inserted between the data specified by I2,E12.4 and the data specified by F12.3. However if the dashes are placed at the beginning or end of the FORMAT specification an additional blank line (or second skipped) is provided. For example FORMAT (////I6) provides for the insertion of 4 blank lines.

IV.5f <u>Printer carriage control</u>

A Printer carriage control Hollerith character must be included in each Format statement used with a PRINT statement to designate the desired space or skip operation for each printed line. The printer-oriented Format statement must begin with 1H followed by a control character which specifies the desired operation. The control characters and their effects are:

blank	single space before printing
0	double space before printing
1	skip to a new page

The control character itself does not become part of the printed output.

<u>Example</u>:

```
      PRINT 2, A,B,J
    2 FORMAT (1HO, F8.2, F8.2, I8)
```

This specification will provide a double space between the line being printed and the previous printed line.

The control carriage specification is applicable to the first line of print only. If more than one line is called for, the user must be sure that the carriage control specifications precede the normal specifications for each line of print.

<u>Example</u>:

```
      PRINT 2, A, B
    2 FORMAT (28H1SMITH, OUTPUT FOR PROBLEM 3/1H ,2F8.2)
```

The line SMITH, OUTPUT FOR PROBLEM 3 will be printed on a new page. The value for A, and the value for B will be printed on the next line.

IV.6 <u>KFII without FORMAT</u> IV.6

The FORMAT statement and the corresponding statement number or address variable in an I/O statement are optional in KFII and may therefore be omitted entirely. If no FORMAT statement is specified, the system will supply FORMAT (5N).

IV.7 <u>Specification statements</u> IV.7

The specification statements are nonexecutable, because they do not cause the generation of instructions in the object program. Instead they provide the processor with information about the nature of the variables used in the program. In addition, they supply the information required to allocate locations in storage for certain variables and/or arrays. Specification statements must appear at the beginning of the source program. The order in which they must appear is specified at the end of this section. (Section IV.7e)

IV.7a COMMON statement IV.7a

General form:

COMMON a, b, c, ...

Where: a, b, ... are variables that may contain dimension information as in the DIMENSION statement.

Variables, including array names, appearing in a COMMON statement, are assigned locations at the upper end of the memory. This COMMON area permits variables to be shared by a program and its subprograms without transmitting arguments.

1. If the variables appearing in a COMMON statement require dimension information, they must appear in the COMMON statement in the same form as they would in a DIMENSION statement; they must not then appear in a DIMENSION statement.

   Example:

   COMMON A, B, C(10, 20, 2)

   where C is a three-dimensional array 10 x 20 x 2.

2. The locations in the COMMON area are assigned in the sequence in which the variables appear in the COMMON statement, beginning with the first COMMON statement of the program.

3. Two variables in COMMON may not be made equivalent to each other.

IV.7b EQUIVALENCE statement IV.7b

General form:

EQUIVALENCE (a,b,c,......), (d, e, f,.....)

Where: a, b, c, d, e, f,..... are variables that may be multiple subscripted; the subscripts must be integer constants.

Each pair of parentheses in the statement list encloses the names of two or more variables that are to be stored in the same location during execution of the object program; any number of equivalences (i.e., sets of parentheses) may be given. IV.7b

Example:

```
DIMENSION B(5), C(10,10), D(5, 10, 15)

EQUIVALENCE (A, B(1), C(5,5)), (D(1, 2, 5), E)
```

The EQUIVALENCE statement indicates that A, and the B and C arrays are to be assigned storage locations so that the elements A, B(1), and C(5,5) are to occupy the same location. In addition, it also specifies that D(1,2,5) and E are to share the same location.

IV.7c Type statements IV.7c

The type statements INTEGER and REAL, are discussed in Section IV.2b, Variables types.

IV.7d DATA statements IV.7d

General form:

```
DATA V1/C1/, V2/C2/, .....Vn/Cn/
```

Where: Vi is a variable name, an element of an array, or an array name.
Ci is a list of constants (separated by commas).

The address of the variable v, which for an array is the first element of the array, is initialized with the first constant of the list c. If the list c has more than one element, these subsequent constants are stored in order in the memory locations which follow the position of the variable v. If the variable v is an array the constants will be stored in the array in the same sequence that data would be stored in the array by the appearance of the array name in an input list.

If there are more constants in the list c than there are elements in the variable v, difficulties may be encountered at object time. No check is made for this error. Moreover, no check is made to see that the variable and its constant(s) are the same type.

Integer and real constants may be preceded by a minus sign. A plus sign preceding a constant is not permitted, but is implied by the absence of a minus sign.

An address constant is used for pre-assigning a statement number to an address variable. Address constants may not appear anywhere in the program but in the DATA statement.

The DATA statement may appear at any position after the specification statements.

Examples:

```
        DIMENSION       X(10,5),B(2,2),I3,TRANS(10)
        .
        .
        .
        DATA            R/3.0/,HOLL/5HABCDE/,JOB/-1/
        DATA            NUM/23S/
        DATA            F/-3.6/,PIE/.31415927E+1/
        DATA            TRANS/3S,4S,4S,3*7S,4*3S/
        .
        .
        .
```

variable	is initialized with:
R	3.0
HOLL	ABCDE
JOB	-1
NUM	the object time representation of statement No. 23
F	-3.6
PIE	3.1415927
TRANS(1)	statement no. 3(obj. time representation)
TRANS(2),TRANS(3)	" " 4 " " "
TRANS(4),(5),(6)	" " 7 " " "
TRANS(7),(8),(9),(10)	" " 3 " " "

IV.7e Order of appearance of specification statements IV.7e

The specification statements must be the first statements of the program. The order of these must be (excluding comments cards) as follows:

COMMON	if any
DIMENSION	if any
EQUIVALENCE	if any
REAL	if any
INTEGER	if any

REAL and INTEGER are considered equivalent, and may be interchanged in the above list.

The DATA statement may appear anywhere after the above list.

IV.8 Subprograms IV.8

The programmer preparing a KFII source program may find that he uses an algebraic function, or a series of source statements, many times in the same program. For example, the program may call for calculating the log of several variables, or the standard deviation of several sets of variables, or the inverse matrix of several sets of matrices. If the source statements that are to be repeated are defined in a subprogram, the programmer need write the source statements only once. These source statements comprise the subprogram definition.

The KFII language provides for four (4) types of subprogram definitions; Arithmetic Statement Functions, Function Subprograms, Subroutine Subprograms, and Library Functions. The Library Function Subprograms are written into the KFII compiler. A list of Library Functions is included in Section IV.9. All other subprograms must be written by the programmer. The Arithmetic Statement Function is expressed in a statement, all other subprograms may include any number of statements.

For each subprogram definition, the KFII language also provides source statements which perform the following operations:

1. Transfer control to the subprogram (Call the subprogram) at each point in the program where the calculations are needed.

2. Transfer the variables to be used in the subprogram calculations (function or subroutine arguments) to the subprogram.

3. Return the values of the calculated variables to the main program.

4. Return control to the main program.

Function subprograms differ from the subroutine subprograms in that functions always return a single value to the main program, whereas a Subroutine Subprogram can return more than one value to the calling program.

IV.8a Arithmetic statement function IV.8a

The Arithmetic Statement Function is analogous to an algebraic function. It is defined by a single arithmetic statement within the program in which it appears.

General definition:

name (a, b,,,,n) expression

Where: name is the name of the Arithmetic Statement Function and a, b,...n are the function arguments represented by distinct non-subscripted variables.

Expression is any arithmetic expression defining the type of computation to be performed when the function is used in an arithmetic statement.

1. The user, naming an Arithmetic Statement Function, must follow the rules for naming a variable. The name must consist of 1-6 alphameric character, the first of which must be alphabetic (special characters may not be used). The name must correspond to the type of arithmetic expression (integer or real); it may be explicitly defined in a Type statement, or implicitly defined by the first letter of the name.

2. Any number of variables appearing in the expression may be used as arguments of the function. Those variables in the expression that are not stated as arguments, are treated as parameters and take the current value of these variables when the Arithmetic Function Statement is called. Parameters may not appear in an equivalence statement.

3. An Arithmetic Statement Function may appear within the expression of another Arithmetic Statement Function provided it has been defined previously.

4. All the Arithmetic Statement Function definitions to be used in a program must precede the first executable statement of the program, and follow the last specification statement.

5. Control is transferred to an Arithmetic Statement Function definition when its name appear in the arithmetic expression. The arguments in the Arithmetic Statement Function definition are set equal to the value of the variables in the calling arithmetic expression. The computations indicated by the function definition are then performed. The resulting quantity replaces the function reference in the expression.

Example:

```
Definition     AVG (A,B,C,D) = (A+B+C+D)/4

Calling        AAGE = X + AVG(E,F,G,H)
```

The calling statement is evaluated by first substituting the argument values in the Arithmetic Statement Function definition: IV.8a

```
A = E
B = F
C = G
D = H
```

The Arithmetic Statement Function, AVG, is then evaluated with the substituted variables. The resulting value is added to X, and then assigned to AAGE.

Example:

Valid Arithmetic Statement Function Definitions

```
SUM (A,B,C,D) = A+B+C+D
FUNC (A)      = A+X*Y*Z(J)
```

Invalid Arithmetic Statement Function Definitions

```
SUBPRG (3,J,K)   = 3*I+J**2
SOMEF  (A(I),B)  = A(I)/B+3
SUBPRFN(A,B)     = A**2+B**2
3 FUNC (D)       = 3.14*D
IDEN   (X,Y,Z)   = X/Y + Y/Z (valid if a real specification
                              statement is included in the
                              program: REAL IDEN)
```

6. It is not permissible to give the same name to an Arithmetic Statement Function and to a Library Function, subprogram or subroutine subprogram when they are used in the same program.

IV.8b FUNCTION subprogram IV.8b

The FUNCTION subprogram is a FORTRAN subprogram consisting of any number of statements. It is an independently written program that is executed wherever its name appears in another program.

General Definition:

```
FUNCTION name (a1, a2, a3, ...an)
     .
     .
     .
RETURN
END
```

where: name is the name of the FORTRAN function. IV.8b

$a_1, a_2, a_3, \ldots, a_n$ are nonsubscripted real or integer variable names, array names, dummy names of Library Subprograms or address variables.

1. The FUNCTION subprogram may contain any FORTRAN statement except a SUBROUTINE statement, another FUNCTION statement, or an input/output statement.

2. Ths user, naming a FUNCTION subprogram must follow the rules for naming a variable as follows: The name must be alphabetic (special characters may not be used). The first letter must be alphabetic. The name must correspond to the type of the result of the FUNCTION subprogram. It may be implicitly defined by the first letter of the name or explicitly defined by using the designator REAL FUNCTION or INTEGER FUNCTION.

Example:

```
REAL FUNCTION MATRIX (A,I,B)
  .
  .
MATRIX = A(I,J) + B(I,J)
RETURN
END

FUNCTION COUNT (I,J,A)
DIMENSION I(10), J(10)
  .
  .
  .
COUNT = I(J+1) + L(J+2)
RETURN
END
```

3. At execution time the arguments of the Function subprogram are replaced by the variables in the calling statement. The current value of the variables is used to perform the calculations. Thus the arguments of the FUNCTION subprogram may be considered to be dummy variable names.

Example:

Program statements	Comments
REAL NUT X = NUT(A)	Calling statement, transfers control to function subprogram NUT.

Subprogram

```
REAL FUNCTION NUT(C)        Before evaluating NUT, C
                            is set equal to current
                            value of A.
D = 48.2
NUT = C/D
RETURN
END
```

4. The variable appearing as the function argument in the calling statement should not be re-defined in the subprogram. Thus, in the above example, A should not be re-defined in NUT.

5. When a dummy argument is an array name, an appropriate array specification in a COMMON or DIMENSION statement must appear in the FUNCTION subprogram. The DIMENSION specification of an argument of a subprogram need not be the same as the DIMENSION specification in the calling program. Any subscripts will refer to the dimensions of the array as declared in the subprogram.

6. The value calculated by the Function subprogram is returned to the calling program by placing the name of the function at least once as the variable name on the left side of the arithmetic statement in the subprogram.

Example:

Program statements	Comments
N = MAX (I,J,K,L)	Calling statement arguments are I,J,K,L, Control transferred to MAX.
Subprogram	
FUNCTION MAX (M,L,MM,NN)	M=I, L=J, MM=K, NN=L
!	MAX is returned to calling program.
MAX = MM	
RETURN	
END	

7. The FUNCTION subprogram must return control to the calling program with a RETURN statement. There may be more than one RETURN statement in a subprogram. The FUNCTION subprogram must also contain an END statement which specifies, for the processor, the last instruction of the subprogram.

Example:

Program statements	Comments
A = ROOTS1 + CALC(Y,X,I)	Calling statement transfers control to CALC
Sub-program	

FUNCTION CALC (A,B,J)	A=Y, B=X, J=I
.	CALC is calculated and if
.	positive or negative the
.	value is returned to the
CALC = A/B+B**J	calling program.
If (CALC) 10,20,10	
10 RETURN	
.	If CALC is zero it is
.	calculated again.
20 CALC = A**J	
RETURN	
END	

IV.8c SUBROUTINE subprogram

The SUBROUTINE subprogram is a set of commonly used operations, it does not restrict itself to a single value for the result, as does the FUNCTION subprogram. A SUBROUTINE subprogram can be used for almost any operation with as many results as desired. Since the SUBROUTINE is a separate subprogram, the variables and statement labels do not relate to any other program, except arguments (including address variables) which are used to carry calculations back to the calling program.

General Definition:

SUBROUTINE name (a_1, a_2, a_3, ...a_n)
.
.
.
RETURN
END

where: name is the subroutine name

a_1, a_2, a_3, ...a_n, are arguments. There need not be any. Each argument used must be a nonsubscripted variable name, array name, or address variable.

1. The user, naming a Subroutine, must note the following rules: The name must consist of 1-6 alphameric characters, the first of which must be alphabetic (special characters may not be used). The name does not have to correspond to any Real or Integer type variable.

2. SUBROUTINE subprograms may contain any Fortran statement except FUNCTION or SUBROUTINE definitions. The DIMENSION specification of an argument of a subroutine need not be the same as the DIMENSION specification in the calling program.

3. The arguments may be considered dummy variable names that are replaced at the time of execution by the actual arguments supplied in the CALL statement. (See below) The actual arguments must correspond in number, order, and type to the dummy arguments. None of the dummy arguments may appear in an EQUIVALENCE statement in a SUBROUTINE subprogram.

4. The SUBROUTINE subprogram is called by a special FORTRAN statement: the CALL statement, which consists of the word CALL followed by the name of the subprogram and its parenthesized arguments, if any.

General form of CALL statement:

CALL name (a_1, a_2, ...a_m)

where: name is the symbolic name of a SUBROUTINE subprogram.

a_1, a_2, ...a_m are the actual arguments (if any) that are being supplied to the SUBROUTINE subprogram.

Example:

Program statements	Comments
CALL MATRIX (X,Y,L,M)	Transfers control to subroutine matrix.
SUBROUTINE MATRIX (A,B,I,J)	A=X, B=Y, I=L, J=M
DIMENSION A(20,20), B(20,20)	
.	MATRIX A is calculated and
.	
10 A(K,M)=A(K,M)+B(K,M)	returned to the main program.
RETURN	Control is returned to
END	the main program.

5. The RETURN statement returns control to the calling program. Multiple returns from a subroutine, each to a different point, can be effected by using address variables as arguments.

If an address variable is carried into a subprogram as an argument, and a transfer to the dummy address variable of the subprogram is executed, control will transfer back to the main subroutine, each to a different point.

Examples:

```
ASSIGN 173 to J    )          main
CALL BOMB(J)       )          program
   .
   .
   .
SUBROUTINE BOMB (ZIP)  )      Subroutine. The GO TO
   .                   )      will transfer control to
   .                   )      statement 173 of the
   .                   )      Main Program
GO TO ZIP
```

6. The SUBROUTINE Subprogram must follow the main program. IV.8c

IV.9 Subprograms provided by FORTRAN IV.9

KINGSTON FORTRAN II includes several commonly used routines that are available to the programmer. The mathematical routines that are provided are defined as FUNCTION subprograms.

IV.9a Mathematical subroutines IV.9a

The names and types (integer or real) of all of these subprograms are automatically assigned by the compiler; therefore, they must not appear in Type statements. Variables used as arguments of mathematical routines must be typed, either explicitly or implicitly, to agree with the type of the arguments of the function reference in which they appear. The mathematical routines are listed in Table 4. In several cases the same routine may be called by more than one name.

TABLE 4

Table of Library Functions

FUNCTION	DEFINITION	NO.OF ARGS.	NAME OR NAMES	TYPE OF ARGUMENT	TYPE OF FUNCTION
Exponential	e^{Arg}	1	EXP EXPF	Real	Real
Natural logarithm	$\log_e(Arg)$	1	LOG LOGF ALOG	Real	Real
Arctangent	arctan(Arg) in range $-\frac{\pi}{2}$ to $\frac{\pi}{2}$	1	ATAN	Real	Real
Arctangent	$\arctan(Arg_1/Arg_2)$ in range $-\pi$ to π	2	ATAN	Real	Real
Trig.Sine	sin(Arg)	1	SIN SINF	Real	Real
Trig.Cosine	cos(Arg)	1	COS COSF	Real	Real
Square Root	$(Arg)^{\frac{1}{2}}$	1	SQRT SQRTF	Real	Real
Absolute value	$\|Arg\|$	1	ABS ABSF IABS	Real Integer	Real Integer

Choosing largest value	Max(Arg_1,Arg_2,---)	$\geq 2 \leq 9$	MAX AMAX	Integer Real	Integer Real
Choosing smallest value	Min(Arg_1,Arg_2,---)	$\geq 2 \leq 9$	MIN AMIN	Integer Real	Integer Real
Float	Conversion from integer to real	1	FLOAT FLOATF	Integer	Real
FIX	Conversion from real to integer	1	INT IFIX	Real	Integer
Transfer of sign	Magnitude of Arg_1 with sign of Arg_2	2	SIGN	Real	Real
Ideal Relay Function	Arg/ABS(Arg) for Arg $\neq$ 0 and 0 otherwise	1	SIGN	Real	Real
Plot	See below (1)	10	PLOT PLOT P	Real 0 arg 80	Real
Rand	See below (2)	1	RAND	Real	Real
Sort	See below (3)	2	SORT	Real	Real

The Library Subroutines are CALLED when they are named in an arithmetic statement.

<u>Example</u>:

Y = SQRT (A)

The square root of A is computed and assigned to Y.

1. The PLOT subroutine may be called with a CALL statement.

<u>General form</u>:

CALL PLOT (Z_1, Z_2, ...Z_m)

where: Z_1, Z_2, ...Z_m are real variables whose range is 0 to 80.
The integer value of any argument to be plotted must be scaled to lie within the range 0 arg 80; values outside this range are considered erroneous. Each CALL PLOT statement causes a single card to be punched. The plot is obtained by listing those output cards on the 407 tabulator.

Up to ten quantities can be plotted. The plotted value is truncated, not rounded. The plot of the first argument listed (Z_1) is printed with a "1", the second with a "2" etc. If there is a tenth quantity, it is given a "0". For example, if Z_5 = 32.59 then a 5 is punched in Col. 32 of the Card. If two arguments have the

<u>General form:</u>

CALL SOLVE (A,N,M,DET)

where: A is a matrix with N rows and N+1 columns.
The N coefficients of the N unknowns of the first equation are in A(1,N) The N unknowns of the first equation are in A(2,N,) and so on.
The constant vector must be in column N+1 of A.
N is the number of equations to be solved and
M is the first number in the DIMENSION statement.

DET is the value of the determinent of the N Rows by N columns of Matrix A, and is defined after the execution of SOLVE. The parimeter DET may be left out of the call statement if the value of the determinent is not wanted.

The answers are left in column N+1 of A.

(See sample problem 10
Section IV.14)

6. Resolv will re-solve the original coefficient matrix when a new constant vector has been put in column N+1 of matrix A. This resolution routine is called by the following statemnt:

CALL RESOLV (A,N,M)

where: A is the A matrix as defined in the solutioh matrix SOLVE.
N is the number of equations to be solved.
M is the first subscript in the DIMENSION statement

7. The determinant subroutine may be used to evaluate the determinent of a square matrix. It is called by the following statement:

Dummy = DETER (B,N,M)

where: B is the square array containing N rows.
N is the first subscript in the DIMENSION statement

(See sample problem 10
Section IV.14)

IV.10 Operating instructions, control cards IV.10

The use of the control cards described in the section Required control cards results in one-pass compilation and execution of a source program written for the KFII compiler. The cards listed under Required control cards should be included in all source decks submitted to the IN-OUT box. Operators of the Model II should note that a COLD START card must be loaded before the Monitor control cards if the compiler is not in memory.

COLD START card: 3400032007013600032007024902402511963611300102

The numbers are punched consecutively starting in card col. 1.

Processing the control cards mentioned in the sector Optional control cards may involve considerable machine time. Users should exercise discretion when including them in a program.

IV.10a Required control cards IV.10a

The compiler is loaded from disk memory into machine core storage when the following two Monitor control cards are read. The cards must be punched in the card columns noted below:

```
Card Col. 1 2 3 4 5 6 7 8 9
          # # J O B
          # # X E Q   K F 2
```

NOTE: The # is a multiple punch 028.

The above cards must precede the first source statement. A KFII system control card must follow the Monitor cards.

```
Card Col. 1 2 3 4 5 6 7 8 9 10.........................45........................
          #         J O B N A M E ................... OPTIONAL USER IDENTIFICATION
```

NOTE: NAME is the programmers name, OPTIONAL USER IDENTIFICATION may include users problem number or class section. The # JOB card will be printed on the on-line printer and will identify all printed program output including source program listing, error messages and problem solutions.

An EOJ card must follow the last source statement. The EOJ card informs the compiler that the last source statement of the program has been reached.

```
Card Col. 1 2 3 4 5 6 7 8 9
          #         E O J (a $ may be substituted for the # when punching the #
                          JOBNAME card or the # EOJ card)
```

A program preceded by a JOB card will result in the output of machine language instructions, stored in the disk work area, ready to be loaded into machine core memory. The following control card must be used to load the object program from the disk work area to machine core storage:

Card Col. 1 2 3 4 5 6 7 8 9 IV.10a
X E Q   R U N

This card must follow the EOJ card and immediately precede the data.

A #### card (record marks punched in Co. 1-4) must follow the last data card. Even if no data cards are used, the #### card should be used to signal the end of the program execution. A source program submitted for batch processing will not be run if the #### is omitted from the source deck.

Sample: Card Deck prepared for program execution, consisting of source deck, data for execution of the program, and monitor control cards.

Monitor Control Cards	##JOB	
	##XEQ KF2	
Internal Control Card	#.....JOB JOE SMITH	MATH 90
Source statements	....	
	....	
	....	
Internal Control Card	# EOJ	
	##XEQ RUN	
Data	....	
	....	
	....	
Monitor Control Card	####	

IV.10b Optional control cards IV.10b

A PRESCAN card may be used in place of the JOB card. A program preceded by a PRESCAN card will produce no machine instructions. However, source language errors will be detected in the normal way. The purpose of this is to allow rapid detection of source language errors. The PRESCAN card must be followed by the ##XEQ KF2 card.

Sample: Card Deck prepared for rapid detection of source language errors

Monitor Control Cards	##XEQ KF2	(Card Col. 2-6 must be blank)
	# PRESCAN	(Card Col. 2-6 must be blank)
Source statements	....	
	....	
	....	
Monitor Control Card	# EOJ	

OBDECK Card IV.10b

An object deck may be obtained during compilation by punching the letter D in col. 32 and/or 33 of the ##XEQ KF2 card. If an bject deck was not obtained during compilation, it is possible to punch an object deck from the program currently in the work areas of the disk. This is done using the following Monitor Control cards:

```
##JOB                                   (Col. 6 must be left blank)
##XEQ OBDECK
```

If the object program is in the form of a card deck, the object deck may be loaded by preceding it by the Monitor Control Cards:

```
##JOB                                   (Col. 6 must be left blank)
##XEQ RUNDK
```

The BEGIN TRACE card causes arithmetic trace instructions to be compiled for each arithmetic and IF statement, beginning with the statement following the control card. No additional instructions are generated for arithmetic statements; one additional instruction is generated for each IF statement. An arithmetic trace halts when an END statement is compiled or an END TRACE card is read.

```
BEGIN TRACE card:  #     BEGIN TRACE     (Card Col. 2-6 are blank
                                          Card Col. 11-80 may contain any valid
END   TRACE card:  #     END   TRACE                             characters.)
```

To execute arithmetic and FLOW TRACE instructions, console switch 4 must be on during program execution. The result of arithmetical FLOW TRACING will be punched 5 per card. TRACE output for arithmetic statements is in modified E15.8 notation preceded by the object time address of the variable on the left-hand side of the arithmetic statement.

The FLOW TRACE card causes instructions to trace the flow of the object program to be generated, beginning with the next executable statement labeled with a statement number. FLOW TRACE generates one additional instruction (12 digits) per statement number traced. The Output of FLOW TRACING is the statement number in order of execution, preceded by the letter NO. (See sample problem 11, Section IV.14).

```
FLOW TRACE card:        #    FLOW TRACE     (see above)

END FLOW TRACE card:    #    END FLOW TRACE
```

A symbol table will be punched for the portion of the program following an INDEX card, unless a STOP INDEX card is read. The symbol table will contain five different types of items, with their names and appropriate addresses. If the variable or subprogram name is undefined, an asterik will appear before the name.

1) Simple variables IV.10b
2) Dimensioned variables
3) Statement Numbers
4) Subprogram Names
5) Constants

INDEX card: # INDEX (Card columns punched as described above)

STOP INDEX card: # STOP INDEX

IV.11 Operating instructions, automatic printer output IV.11

During compilation all control cards will be printed on the 1443 on-line printer. The # JOBNAME card will identify all printed output. The source statements and any compilation error messages will also be listed on the printer. The compilation error codes are identified in table 5 "Kingston Fortran II ERROR MESSAGES".

Any variables, statement numbers, or function, which are undefined in a subprogram, will be typed or printed during compilation. The name of the undefined quantity will be preceded by an asterisk. If the undefined quantity is a statement number, the letter S will appear to the right of the number.

The compiler will accept a program containing undefined variables as O.K.; however, the program may not run, depending on the nature of the particular situation. If no source program errors have been detected, the message O.K. will be printed and the program will be executed. If a source statement error or an undefined statement number has been found, the message NO GO will be printed when the EOJ card is read and the object deck will not load.

IV.12 Operating instructions, error messages during compilation IV.12

If a source statement contains an error, an error message will be printed.Output of machine language instructions is then suppressed for the duration of the job, but the remainder of the compilation will continue until an EOJ card is read, so that any additional errors will be detected.

Source statement errors are printed in the following form:

CC NNNNN MMMM

Where: CC is two digit error code
NNNNN is the last encountered statement number in the subprogram or main program.
MMMM is the number of statements after statement numbered NNNNN in which error occured (comment cards, monitor control cards, and continuation cards are not counted). (See Table 5, KINGSTON FORTRAN II ERROR MESSAGES).

IV.13 Operating instructions, error messages during object program execution IV.13

During the object program execution, errors are noted by inserting digits in a table stored in memory. The error table is printed when a CALL EXIT statement is encountered. The error codes will be printed between the words EXIT XXXXXXXXXXXXXXXXX CHECK. A 0 indicates no error. A digit indicates an error; errors are identified by numbers and position in the EXIT line (See Table 6). Sample problem (Section IV.14) describes the form of printed execution errors. If there are no execution errors the message EXIT CHECK will be printed at the conclusion of program execution.

In addition, certain input-output errors are detected at object time. Table 7 outlines these errors and the action taken in each case. (See Table 6 OBJECT TIME ERRORS, and Table 7, I/O ERRORS AT OBJECT TIME).

TABLE 5

KINGSTON FORTRAN II ERROR MESSAGES

A1	STATEMENTS CONTAINING EXPRESSIONS	ILLEGAL SYNTAX IN AN EXPRESSION
A2	..	(1) ILLEGAL SYNTAX IN AN ARITHMETIC STATEMENT (2) AN EXPRESSION OR INTEGER CONSTANT ON THE LHS OF AN ARITHMETIC STATEMENT (3) A SUBSCRIPTED VARIABLE NOT MENTIONED IN A DIMENSION STATEMENT
A3	..	MIXED MODE IN AN EXPRESSION
A4	..	WRONG NUMBER OF SUBSCRIPTS IN A DIMENSIONED VARIABLE
A5	..	SUBSCRIPT IS A REAL QUANTITY
A6	..	NAME OF A NON-EXTERNAL FUNCTION USED AS A VARIABLE
A7	..	THE CHARACTERS - OR $ APPEAR AS OPERATOR IN AN EXPRESSION
A8	..	ONE OF THE TABLES USED BY THE COMPILER HAS OVERFLOWN (STATEMENT IS TOO LONG OR COMPLEX)
C1	COMMON	(1) SYMBOL IS ALREADY IN THE SYMBOL TABLE
	DIMENSION	(1) SYMBOL IS ALREADY IN THE SYMBOL TABLE AND IS NOT A FORMAL PARAMETER (2) NO DIMENSIONS GIVEN FOR VARIABLE
C2	COMMON OR DIMENSION	ARRAY SIZE IS GREATER THAN 9999 ELEMENTS
C3	COMMON OR DIMENSION	MORE THAN 13 DIMENSIONS SPECIFIED
C4	COMMON	(1) INVALID CHARACTER, MOST LIKELY CAUSED BY A MISSING COMMA OR CLOSING PARENTHESIS (2) CONSTANTS WHERE VARIABLE NAMES SHOULD BE
	REAL, INTEGER, EXTERNA FUNCTION, SUBROUTINE, ARITHMETIC STATEMENT FUNCTIONS	CONSTANTS WHERE VARIABLE NAMES SHOULD BE

C5	COMMON, DIMENSION, EQUALENCE, INTEGER, REAL, EXTERNAL	STATEMENTS ARE NOT IN THE SPECIFIED SEQUENCE
C6	COMMON, EQUIVALENCE	ILLEGAL EXPANSION OF COMMON IN A SUBPROGRAM
D1	DO, I/O DO	(1)VARIABLE OR EXPRESSION IS REAL (FLOATING POINT) MODE RATHER THAN INTEGER (2) THE INDEX OF THE DO IS AN EXPRESSION
D2	DO, I/O DO	SYNTAX ERROR. TOO MANY OR TOO FEW TERMS, OR COMMA OR RIGHT PARENTHESIS MISPLACED
D3	DO	THE STATEMENT NUMBER SPECIFYING THE RANGE OF THE DO HAS ALREADY BEEN DEFINED
D4	TERMINATION OF A DO OR I/O DO	PREVIOUS ERRORS HAVE MESSED UP THE DO TERMINATIONS. WHEN ALL DO STATEMENTS AND THEIR RANGES ARE CORRECT THIS ERROR CANNOT OCCUR
E1	EQUIVALENCE	TRYING TO EQUIVALENCE A DEFINED VARIABLE TO SOMETHING ELSE
E2	EQUIVALENCE	TRYING TO EQUIVALENCE TWO ARRAYS IN SUCH A WAY THAT THEY HAVE NO COMMON ELEMENTS
E3	EQUIVALENCE	(1) STATEMENT SAID EQUIVALENCE (V1),---- AND DID NOT SPECIFY THE SECOND ITEM (2) INVALID EXPRESSION (ARITHMETIC) IN EQUIVALENCE (3) STATEMENT IS INCOMPLETE
	DATA	STATEMENT IS INCOMPLETE
F1	FORMAT	(1) A LEFT PARENTHESIS HAS BEEN FOUND BEFORE THE REPEATING SECTION NN(......) HAS BEEN COMPLETED (2) A MINUS SIGN THAT IS NOT PART OF A HOLLERITH FIELD OR A -NNP TERM (3) MORE THAN 5 LEVELS OF NN(.....)IN A NEST (4) INVALID LETTER IN WHAT LOOKS LIKE A NUMERIC SPECIFICATION
F2	FORMAT	FINAL CLOSING PARENTHESIS IS MISSING. MAY BE DUE TO A HOLLERITH STATEMENT WITH TOO BIG A CHARACTER COUNT
	DATA	INCOMPLETE STATEMENT. MAY BE MISSING A /
	CALL	NO SUBROUTINE NAME
	OTHERS	INCOMPLETE OR GARBLED STATEMENT

F3	FORMAT	(1) THE W SPECIFICATION IS MISSING IN A AW OR IW TERM (2) THE W OR D OR DECIMAL POINT IS MISSING IN A EW.D OR FW.D SPECIFICATION (3) AN AW SPECIFICATION HAS A W GREATER THAN 50 (4) UNINTELLIGIBLE
F4	FORMAT	(1) SPECIFICATION EW.D OR FW.D HAS W-D GREATER THAN 45 (2) SPECIFICATION EW.D OR FW.D HAS D GREATER THAN W (3) SPECIFICATION IW, FW.D, OR EW.D HAS W GREATER THAN 80 (4) SPECIFICATION AW HAS W=0
F5	FORMAT	(1) SPECIFICATION =NNP OR NNP HAS NN GREATER THAN 49 (2) SPECIFICATION NNH, NNX, NN(, NNE, NNF, NNI, NNA HAS NN=0
L1	STATEMENTS WITH, OR CONTAINING STATEMENT NUMBERS	WHAT SHOULD BE A STATEMENT NUMBER OR ADDRESS IS EITHER AN ARITHMETIC EXPRESSION REAL (FLOATING POINT) MODE, NEGATIVE, OR ZERO
N1		(1) WHATSHOULD BE A NAME OR NUMBER BEGINS WITH ONE OF(, $ + - * /) (2) DIMENSIONING INFORMATION IS NOT AN INTEGER CONSTANT
N3		A SYMBOL HAS MORE THAN SIX CHARACTERS IN IT
N4		REAL (FLOATING POINT) CONSTANT IS GREATER THAN 0.0 BUT LESS THAN 1.E-51
N5		REAL (FLOATING POINT) CONSTANT IS EQUAL TO OR GREATER THAN 1.E+49
N6	FORMAT	SOME CONSTANT IN THE STATEMENT CONTAINS MORE THAN 2 DIGITS
	OTHER THAN FORMAT	AN INTEGER (FIXED POINT) CONSTANT OR STATEMENT NUMBER (USED IN THE STATEMENT) HAS MORE THAN 5 DIGITS
	SIZE	THE SIZE CONTAINS MORE THAN 5 DIGITS
	ORIGIN	THE ORIGIN CONTAINS MORE THAN 5 DIGITS
N7		A REAL (FLOATING POINT) CONSTANT CONTAINS DECIMAL POINTS

N8	NON-FORMAT	A HOLLERITH CONSTANT CONTAINS MORE THAN FIVE LETTERS
P2	ASSIGN	IN THE EXPRESSION ASSIGN I TO J (1) I IS A VARIABLE BUT DOES NOT HAVE BRACKETS AROUND IT (2) THE TO J IS MISSING (3) J IS NOT AN INTEGER VARIABLE (4) THE J IS NOT THE END OF THE STATEMEN (5) I IS NOT AN INTEGER VARIABLE
P3	IF	(1) THE ARGUMENT OF THE IF STATEMENT IS AN INTEGER CONSTANT (2) THE ARGUMENT IS NOT PROPERLY ENCLOSE WITHIN PARENTHESIS
	IF(SENSE SWITCH I)	THE SENSE SWITCH I IS NOT PROPERLY ENCLOSED WITHIN PARENTHESIS
P4	GO TO	(1) THERE IS AN UNDESIRABLE) IN GO TO N (2) THERE IS AN = SIGN IN A COMPUTED GO TO (3) THE INDEX IN THE COMPUTED GO TO IS NOT THE LAST THING IN THE STATEMENT (4) THE INDEX OF THE COMPUTED GO TO IS A REAL VARIABLE
P7	CALL	INCOMPLETE STATEMENT
	DATA	INVALID DELIMITER
Q5	IF	(1) THERE ARE TOO MANY OR TOO FEW STATEMENT NUMBERS OR LABELS AFTER AN IF (2) THE LIST OF STATEMENT NUMBERS AND LABLES HAS A MISPLACED RIGHT PARENTHESIS
Q7		UNRECOGNIZABLE STATEMENT
Q8	STOP N, PAUSE N	THE N IS NOT AN INTEGER EXPRESSION
Q9		DOUBLY DEFINED STATEMENT NUMBER
R1	I/O	INCORRECT I/O STATEMENT. PARENTHESIS, COMMAS, AND EQUAL SIGNS (DO S) ARE MISSING OR MISPLACED.
	DO	INCORRECT DO STATEMENT. PARENTHESIS, COMMAS, AND EQUAL SIGNS (DO S) ARE MISSING OR MISPLACED
	COMPUTED GO TO	INCORRECT COMPUTED GO TO. PARENTHESIS, OR COMMAS ARE MISPLACED OR MISSING.

R2	I/O	(1) EXPRESSION IN AN INPUT STATEMENT. (2) INVALID SYNTAX. MAY BE A MISPLACED CLOSING PARENTHESIS
R3	I/O	(1) THE FORMAT NUMBER IS FOLLOWED BY A RIGHT PARENTHESIS. (2) SYNTAX ERROR. PROBABLY SOME OTHER DELIMITER WHERE A COMMA SHOULD BE
R4	I/O	FUNCTION SUBPROGRAM HAS AN INPUT STATEMENT
S1	FUNCTION	(1) NOT FIRST STATEMENT IN A FUNCTION SUBPROGRAM (2) DOES NOT HAVE ARGUMENTS (3) SUBPROGRAM NAME OR OTHER INVALID ARGUMENT (4) INVALID SYNTAX. PROBABLY MISSING COMMA OR RIGHT PARENTHESIS
	SUBROUTINE	(1) NOT FIRST STATEMENT IN A SUBROUTINE SUBPROGRAM (2) SUBPROGRAM NAME OR OTHER INVALID ARGUMENT (3) INVALID SYNTAX. PROBABLY MISSING COMMA OR RIGHT PARENTHESIS
	ARITHMETIC STATEMENT F	(1) STATEMENTS NOT IN PROPER SEQUENCE (2) DOES NOT HAVE ARGUMENTS (3) SUBPROGRAM NAME OR OTHER INVALID ARGUMENT (4) INVALID SYNTAX. PROBABLY MISSING COMMA OR RIGHT PARENTHESIS
	ARITHMETIC STATEMENT	SUBSCRIPTED VARIABLE ON LEFT HAND SIDE FOR WHICH NO DIMENSION STATEMENT EXISTED
S2	FUNCTION, SUBROUTINE, ARITHMETIC STATEMENT-F	NAME OF FUNCTION OR SUBROUTINE IS DEFINE TWICE OR IS THE SAME AS A LIBRARY PROGRA
S3	RETURN	PROGRAM IS NOT A SUBPRGORAM
S4	CALL	STATEMENT HAS INVALID SYNTAX. PROBABLY A COMMA OR RIGHT PARENTHESIS IS OUT OF PLACE
T1	INTEGER, REAL	ATTEMPTED TO CHANGE THE MODE OF ALREADY DEFINED FUNCTION

T2	EXTERNAL	TRYING TO MAKE A VARIABLE INTO A FUNCTION
T3	INTEGER, REAL, EXTERNA EQUIVALENCE	INVALID CHARACTER IN STATEMENT WHERE A COMMA SHOULD BE
X1		HAVE DESTROYED DIMENSION TABLES AND MAYBE PART OF SYMBOL TABLE. JOB ABANDONED
X2		PUNCH CHECK PERSISTS FOR TWO TRIES
X3		SYMBOL TABLE FULL. JOB ABANDONED
X4		PROGRAM TOO BIG FOR MEMORY AVAILABLE. JOB ABANDONED
X5	DIMENSION, COMMON	WORK AREA FULL. STATEMENT TOO LONG TO PROCESS
X6		FUNCTION CARD(S) ARE UNINTELLIGIBLE
X7		DISK ERROR PERSISTED FOR TEN ATTEMPTS
X8		INVALID CONTROL CARD. JOB ABANDONED
Z1		STATEMENT IS A MEANINGLESS COLLECTION OF ZERO TO THREE CHARACTERS
Z2		THERE IS AN UNPAIRED CLOSING PARENTHESIS
Z3		HOLLERITH FIELD WAS INCOMPLETE AT END OF STATEMENT
Z4		THE EXPRESSION NNNNNH (WHERE N IS A DIGIT) HAS OCCURRED. THIS IS TOO MANY DIGITS FOR A VALID HOLLERITH, AND ALSO TOO MANY TO BE PART OF A SYMBOL
Z5		A STATEMENT WHICH IS NOT AN ARITHMETIC STATEMENT IS NOT COMPLETE
Z6		STATEMENT NUMBER IS GARBLED SOMEHOW
	SIZE	THE SIZE SPECIFICATION IS MISSING OR GARBLED
	ORIGIN	THE ORIGIN SPECIFICATION IS MISSING OR GARBLED
Z7	EOJ	(1) EOJ CARD NOT PRECEDED BY AN END CARD (2) NO MAIN PROGRAM IN A NON@RELOCATABLE JOB

Z8	END	(1) END STATEMENT HAD A STATEMENT NUMBER (2) LAST EXECUTABLE STATEMENT WAS NOT A TRANSFER OR CALL STATEMENT (3) THE PROGRAM CONTAINS NO EXECUTABLE STATEMENTS (4) NO RETURN STATEMENT IN A FUNCTION SUBPROGRAM (5) TWO MAIN PROGRAMS IN A JOB. (6) MAIN PROGRAM IN A RELOCATABLE COMPILATION (7) IN A FUNCTION SUBPROGRAM, THE FUNCTION HAS NOT BEEN EVALUATED BEFORE THE END STATEMENT IS ENCOUNTERED.
Z9	FORMAT	DOES NOT HAVE STATEMENT NUMBER.
	COMMON DIMENSION EQUIVALENCE EXTERNAL FUNCTION INTEGER REAL DATA SUBROUTINE ARITH.STATE.FUNCTION	HAS A STATEMENT NUMBER.
	GO TO IF IF(SENSE SWITCH) ARITH.STATEMENT ASSIGN ACCEPT PUNCH TAPE ACCEPT TAPE PAUSE PRINT PUNCH READ REREAD TYPE CALL STOP CONTINUE RETURN DO GO TO (N1, N2,---),I	FOLLOWS A TRANSFER STATEMENT AND DOES NOT HAVE A STATEMENT NUMBER

RETURN STOP GO TO GO TO (N1, N2,---), I IF IF(SENSE SWITCH) DO	LAST STATEMENT IN THE RANGE OF A DO.
BAD CARD	THERE HAS BEEN A READ CHECK. RELOAD THE CARD AND PUSH START.
PCH CHK	THERE HAS BEEN A PUNCH CHECK WHEN PUNCHING THE LOADER ROUTINE. CLEAR THE PUNCH AND PUSH START TO TRY AGAIN.
JOB ABANDONED	(1) OCCURS AFTER ERRORS X1 AND X3. (2) INVALID CONTROL CARD.

Table 6

OBJECT TIME ERRORS

Position in Error Field	Digit	Meaning	Action Taken (FAC = Accumulator = Result Field)
1st digit	1	Floating Point Underflow	FAC = 0000000000
2nd	2	Floating Point Overflow	FAC = ±9999999999
3rd	3	Floating Point Divided by Zero	FAC = ±9999999999
4th	4	Fixed Point Divided by Zero	FAC is unchanged, i.e. J/O ≡ J
5th	5	Square Root of Neg.Number	Square root of absolute value of arg.
6th	6	Log of zero or Neg.Number	Log(0) ≡ -9999999999; otherwise log of abs. value of arg.
7th	7	Sin or cos, arg. > 10^8	CALL EXIT
8th	8	Exp(x) out of range	FAC = +9999999999 or zero
9th	9	Input number too small	The number entered memory as 0000000000
10th	$\bar{1}$	Input number too big arg <0 or >80	The number entered memory as ±9999999999
11th	$\bar{2}$	Plot, -1 ≥ arg >80	Point not plotted
12	$\bar{3}$	I/O error 2. (Table 1.3.6.2)	Number ignored. ER2 inserted in output
13	$\bar{4}$	I/O error 1. (Table 1.3.6.2)	Number ignored. ER1 inserted in output
14	$\bar{5}$	Unused. Available for user-defined relocatable subprograms	
15	$\bar{6}$		
16	$\bar{7}$		
17	$\bar{8}$		

Table 7

I/O ERRORS AT OBJECT TIME

I/O Error	Reason	Result
0	Input record from T/W or paper tape over 150 characters long	CALL EXIT
1	Non-alphabetic data on A-type output	Number ignored *
2	Field width too small on I, E, F output; may be an undefined variable	Number " *
3	Invalid character on input data on I, E, F, or N Format	CALL SKIP
4	An input word has more than 88 significant digits	CALL SKIP
5	Input-output list with no numeric specifications between last opening-closing parenthesis pair in Format statement	CALL EXIT
6	Format requires more than 150 characteris in a record	CALL EXIT
7	Write-check occurred 3 times when attempting to punch output or trace card	CALL EXIT
20	Disc error persisted 10 times	CALL EXIT
Errors in Variable Format		
1F	- Format too long or complex for available work area - Comma or right parenthesis before completion of repeating format	CALL SKIP
2F	- Minue sign which is not part of a H- or P-specification - Incorrect length of H-specification - No closing parenthesis - Statement incomplete - Non-permissible character	CALL SKIP
3F	- More than 5 levels of repeated, parenthesized Format - Repeated Format with more than 49 repeats - Field width missing in I, A, E, or F, specification - Field witdth greater than 50 in A-specification - d or decimal point missing in Ew.d or Fw.d - non-permissible character - d>w in Ew.d or Fw.d - (w-d)>45 - Field width, w, >50 in Ew.d, Iw, or Fw.d - Specification Aw has w = 0	CALL SKIP

Table 7 Continued:

READ 1	Read check on T/W	Computer halts. When start is pressed, the machine will attempte to read the record again.
READ 2	" " " paper tape	
READ 5	" " " cards	

* Alphabetic field is replaced by ER1. Last item in record may be lost.

** Numeric field is replaced by ER2. Last item in record may be lost.

*IV.14 SAMPLE PROBLEMS

SAMPLE PROBLEM 7

```
##JOB 5
##XEQ KF2
$      JOB JOE SMITH
C   PROGRAM READS DATA IN BY COLS,PUNCHES DATA OUT BY ROWS
       DIMENSION C(5,5)
       READ 10,((C(I,J),I=1,4),J=1,5)
    10 FORMAT (5F8.2)
       PUNCH 10,((C(I,J),J=1,5),I=1,4)
       CALL EXIT
       END
$      EOJ
##XEQ RUN
00001.1000002.1000003.1000004.1000001.20
00002.2000003.2000004.2000001.3000002.30
00003.3000004.3000001.4000002.4000003.40
00004.4000001.5000002.5000003.5000004.50
####

     1.10     1.20     1.30     1.40     1.50
     2.10     2.20     2.30     2.40     2.50
     3.10     3.20     3.30     3.40     3.50
     4.10     4.20     4.30     4.40     4.50
```

SAMPLE PROBLEM 8

```
##JOB 5
##XEQ KF2
$      JOB JOE SMITH
C   PROBLEM OUTPUTS A MATRIX WITH A
C   VARIABLE NUMBER OF COLUMNS AND ROWS
C   PROBLEM,TO GENERATE A MATRIX A(I,J)=1/(I+J+1)
C   I AND J ARE INTEGERS BETWEEN 1 AND 30
       DIMENSION A(30,30)
    10 READ 11,I,J
    11 FORMAT(2I2)
       IF(I-99)112,99,112
   112 IF(I-30)111,111,10
   111 IF (J-30)12,12,10
    12 PUNCH 13,I,J
    13 FORMAT(32HTHE MATRIX IS AS FOLLOWS WITH I=,I2,3H J=,I2)
```

NOTE Sample problems 7-12 call for punch card output. An on-line 1443 printer will become available to users of the Model II during the Fall'66 semester. A Computing Center memorandum which will include sample problems illustrating print output will be issued in the Fall.

```
      DO 14 K=1,I
      DO 14L=1,J
      B=K
      C=L
   14 A(K,L)=1./(B+C+1.)
C  OUTPUT ROUTINE
      DO 21 K=1,I
   21 PUNCH 22,K,(A(K,L),L=1,J)
   22 FORMAT(4HROW=,I4//(6F11.8,1X))
      GO TO 10
   99 CALL EXIT
      END
$     EOJ
##XEQ RUN
6123
1214
0510
####
```

```
THE MATRIX IS AS FOLLOWS WITH I=12 J=14
ROW=    1

  .33333333  .25000000  .20000000  .16666667  .14285714  .12500000
  .11111111  .10000000  .09090909  .08333333  .07692308  .07142857
  .06666667  .06250000
ROW=    2

  .25000000  .20000000  .16666667  .14285714  .12500000  .11111111
  .10000000  .09090909  .08333333  .07692308  .07142857  .06666667
  .06250000  .05882353
ROW=    3

  .20000000  .1666667   .14285714  .12500000  .11111111  .10000000
  .09090909  .08333333  .07692308  .07142857  .06666667  .06250000
  .05882353  .05555556
ROW=    4

  .16666667  .14285714  .12500000  .11111111  .10000000  .09090909
  .08333333  .07692308  .07142857  .06666667  .06250000  .05882353
  .05555556  .05263158
ROW=    5

  .14285714  .12500000  .11111111  .10000000  .09090909  .08333333
  .07692308  .07142857  .06666667  .06250000  .05882353  .05555556
  .05263158  .05000000
ROW=    6

  .12500000  .11111111  .10000000  .09090909  .08333333  .07692308
  .07142857  .06666667  .06250000  .05882353  .05555556  .05263158
  .05000000  .04761905
ROW=    7
```

```
 .11111111   .10000000   .09090909   .08333333   .07692308   .07142857
 .06666667   .06250000   .05882353   .05555556   .05263158   .05000000
 .04761905   .04545455
ROW=   8

 .10000000   .09090909   .08333333   .07692308   .07142857   .06666667
 .06250000   .05882353   .05555556   .05263158   .05000000   .04761905
 .04545455   .04347826
ROW=   9

 .09090909   .08333333   .07692308   .07142857   .06666667   .06250000
 .05882353   .05555556   .05263158   .05000000   .04761905   .04545455
 .04347826   .04166667
ROW=  10

 .08333333   .07692308   .07142857   .06666667   .06250000   .05882353
 .05555556   .05263158   .05000000   .04761905   .04545455   .04347826
 .04166667   .04000000
ROW=  11

 .07692308   .07142857   .06666667   .06250000   .05882353   .05555556
 .05263158   .05000000   .04761905   .04545455   .04347826   .04166667
 .04000000   .03846154
ROW=  12

 .07142857   .06666667   .06250000   .05882353   .05555556   .05263158
 .05000000   .04761905   .04545455   .04347826   .04166667   .04000000
 .03846154   .03703704
THE MATRIX IS AS FOLLOWS WITH I= 5 J=10
ROW=   1

 .33333333   .25000000   .20000000   .16666667   .14285714   .12500000
 .11111111   .10000000   .09090909   .08333333
ROW=   2

 .25000000   .20000000   .16666667   .14285714   .12500000   .11111111
 .10000000   .09090909   .08333333   .07692308
ROW=   3

 .20000000   .16666667   .14285714   .12500000   .11111111   .10000000
 .09090909   .08333333   .07692308   .07142857
ROW=   4

 .16666667   .14285714   .12500000   .11111111   .10000000   .09090909
 .08333333   .07692308   .07142857   .06666667
ROW=   5

 .14285714   .12500000   .11111111   .10000000   .09090909   .08333333
 .07692308   .07142857   .06666667   .06250000
```

SAMPLE PROBLEM 9

```
##JOB 5
##XEQ KF2
$     JOBJOE SMITH
C   ILLUSTRATION OF PLOT SUBROUTINE
C   ESTIMATION OF SQUARE ROOTS WITH PLOT OF SUCCESSIVE ESTIMATES
C   FORMULA FOR ESTIMATION OF SQUARE ROOTS,
C   EX(N+1)=.5(EX(N)+Y/EX(N))
C   WHERE EX IS THE ESTIMATED SQUARE ROOT OF Y
      PUNCH 60
   60 FORMAT(//20X37HAPPROXIMATION OF THE SQUARE ROOT OF Y//)
   12 READ 14,Y
   14 FORMAT (F8.2)
      IF (Y) 16,18,20
   20 IF(Y-999999.99) 22,24,24
   22 N=0
      IF(Y-1.0)26,28,28
   28 PUNCH 80,Y
   80 FORMAT(27HPLOT OF APPROX. SQ.ROOTS OF,F8.2)
      EX=1.
   30 EX=.5*(EX+Y/EX)
      Z=EX
C   SUB-ROUTINE PLOT DOES NOT PLOT ARGUMENTS
C   LESS THAN ONE OR GREATER THAN 79
      IF(Z-80.)85,82,82
   85 IF(Z-1.)82,81,81
   82 PUNCH83,Z
   83 FORMAT(24HEST.SQ.ROOT NOT PLOTTED,2X2HZ=,F9.3)
      GO TO 31
   81 CALL PLOT(Z)
   31 AE=ABSF((EX**2)-Y)
      RE=AE/Y
      N=N+1
      IF(RE-.0001)50,50,30
   50 PUNCH 34
   34 FORMAT (4X1HY,7X10HABS. ERROR,4X10HREL. ERROR,7X,
     11HN,12X13HEST. SQ. ROOT)
      PUNCH 36,Y,AE,RE,N,EX
   36 FORMAT(F8.2,3XF12.8,3XF10.8,3XI5,15XF9.3//)
      GO TO 12
   26 EX=.1
      GO TO 30
   16 PUNCH 38,Y
   38 FORMAT (11HY NEGATIVE=,F8.2,2X,
     132HEST. SQ. ROOT OF ABS. VALUE OF Y)
      Y=ABSF(Y)
      GO TO 70
   70 PUNCH 71
   71 FORMAT (//)
      GO TO 22
   18 PUNCH 40
   40 FORMAT (3HY=0)
```

```
      GO TO 12
   24 CALL EXIT
      END
$     EOJ
##XEQ  RUN
+ 100.10
+    .25
+1608.01
99999999
####
```

```
                    APPROXIMATION OF THE SQUARE ROOT OF Y


PLOT OF APPROX. SQ.ROOTS OF 1608.01
EST.SQ.ROOT NOT PLOTTED,  Z=  804.505
EST.SQ.ROOT NOT PLOTTED,  Z=  403.252
EST.SQ.ROOT NOT PLOTTED,  Z=  203.620
EST.SQ.ROOT NOT PLOTTED,  Z=  105.758
                                                        1
                                            1
                                       1
                                       1
   Y         ABS. ERROR    REL. ERROR        N              EST. SQ. ROOT
 1608.01      .01830000     .00001138        8                     40.100


1
EST.SQ.ROOT NOT PLOTTED,  Z=      .746
EST.SQ.ROOT NOT PLOTTED,  Z=      .541
EST.SQ.ROOT NOT PLOTTED,  Z=      .502
EST.SQ.ROOT NOT PLOTTED,  Z=      .500
   Y         ABS. ERROR    REL. ERROR        N              EST. SQ. ROOT
    .25       .00000230     .00000920        5                       .500

PLOT OF APPROX. SQ.ROOTS OF  100.10
                                                    1
                        1
             1
     1
     1
     1

   Y         ABS. ERROR    REL. ERROR        N              EST. SQ. ROOT
  100.10      .00107000     .00001069        6                     10.005
```

SAMPLE PROBLEM 10

```
##JOB 5
##XEQ KF2
$       JOBJOE SMITH
C   SOLUTION OF SIMULTANEOUS LINEAR EQUATIONS USING THE
C   SOLV AND RESOLV SUB-ROUTINES STORED ON THE DISK
C   DIMENSION STATEMENT PERMITS USE OF 10 EQUATIONS
        DIMENSION A(10,11)
        READ4,N
      4 FORMAT(I2)
        L=N+1
        READ 10,((A(I,J),J=1,L),I=1,N)
     10 FORMAT (4F6.2)
C   HEADER CARD
        PUNCH 5,N
      5 FORMAT(16HTHE ANSWERS FOR ,I2,32H LINEAR EQUATIONS ARE AS FOLLOWS)
        M=10
        CALL SOLVE (A,N,M)
        PUNCH 20,(A(I,L),I=1,N)
     20 FORMAT (3F6.2)
        READ 20,(A(I,L),I=1,N)
        CALL RESOLV (A,N,M)
        PUNCH 20,(A(I,N+1),I=1,N)
C   SINCE THERE IS NO INSTRUCTION TO SIGNAL THAT THE
C   LAST DATA CARD HAS BEEN READ
C   THE END OF DATA CARD IS READ AS THE LAST DATA CARD
C   ADDITIONAL DATA CARDS COULD BE ADDED WITH AN INSTRUCTION
C   SIGNALLING THE COMPUTER TO RETURN TO THE FIRST READ
C   INSTRUCTION
        CALL EXIT
        END
$       EOJ
03
+08.00-05.00+07.00+29.00
-01.00+09.00-06.00+01.00
+00.00+02.00-02.00-02.00
+27.00+12.12+03.02
THE ANSWERS FOR  3 LINEAR EQUATIONS ARE AS FOLLOWS
  2.00  3.00  4.00
  4.10  2.39   .88
```

NOTE: ##XEQ RUN should precede the $ EOJ card.
      #### should follow data cards.

SAMPLE PROBLEM 11

```
##JOB
##XEQ KF2
$      JOE JOE SMITH
C  ILLUSTRATION OF FLOW TRACE
C  TO SUM THE ODD NUMBERS FROM 1TO 99
*      BEGIN TRACE
*      FLOW TRACE
       SUM=0.0
       DO 30 I=1,100,2
       X=I
   30  SUM=SUM+X
       PUNCH 40,SUM
   40  FORMAT(4HSUM=,F12.2/)
       CALL EXIT
       END

$      EOJ
##XEQ RUN
####  (2)              (3)        (4)         (5)          (6)    (7)                 (8)
1)399491000000001         NO 00030  399991000000001399493000000001         NO 0003
 399994000000001399495000000001         NO 00030  399999000000001399497000000
      NO 00030  399991600000002399499000000001         NO 00030  39999250000
 399491100000002         NO 00030  399993600000002399491300000002         NO 0003
 399994900000002399491500000002         NO 00030  399996400000002399491700000
      NO 00030  399998100000002399491900000002         NO 00030  39999100000
 399492100000002         NO 00030  399991210000003399492300000002         NO 0003
 399991440000003399492500000002         NO 00030  399991690000003399492700000
      NO 00030  399991960000003399492900000002         NO 00030  39999225000
 399493100000002         NO 00030  399992560000003399493300000002         NO 0003
 399992890000003399493500000002         NO 00030  399993240000003399493700000
      NO 00030  399993610000003399493900000002         NO 00030  39999400000
 399494100000002         NO 00030  399994410000003399494300000002         NO 0003
 399994840000003399494500000002         NO 00030  399995290000003399494700000
      NO 00030  399995760000003399494900000002         NO 00030  39999625000
 399495100000002         NO 00030  399996760000003399495300000002         NO 0003
 399997290000003399495500000002         NO 00030  399997840000003399495700000
      NO 00030  399998410000003399495900000002         NO 00030  39999900000
 399496100000002         NO 00030  399999610000003399496300000002         NO 0003
 399991024000004399496500000002         NO 00030  399991089000004399496700000
      NO 00030  399991156000004399496900000002         NO 00030  39999122500
 399497100000002         NO 00030  399991296000004399497300000002         NO 0003
 399991369000004399497500000002         NO 00030  399991444000004399497700000
      NO 00030  399991521000004399497900000002         NO 00030  39999160000
 399498100000002         NO 00030  399991681000004399498300000002         NO 0003
 399991764000004399498500000002         NO 00030  399991849000004399498700000
      NO 00030  399991936000004399498900000002         NO 00030  39999202500
 399499100000002         NO 00030  399992116000004399499300000002         NO 0003
 399992209000004399499500000002         NO 00030  399992304000004399499700000
      NO 00030  399992401000004399499900000002         NO 00030  39999250000
SUM=     2500.00
```

1. ADDRESS OF X 2. INITIAL VALUE OF X 3. STATEMENT NUMBER
4. ADDRESS OF SUM 5. INITIAL VALUE OF SUM 6. ADDRESS OF X
7. SECOND VALUE OF X 8. STATEMENT NUMBER

SAMPLE PROBLEM 12

```
##JOB 5
##XEQ KF2
$       JOBJOE SMITH
C   ILLUSTRATION OF EXECUTION ERROR
C   TO SUM THE ODD NUMBERS FROM 1 TO 99
        SUM=0.0
        Y=0.0
        DO 30 I=1,100,2
        X=I
   30   SUM=SUM+X
        SUM=SUM/Y
        PUNCH 40,SUM
   40   FORMAT(F12.2)
        CALL EXIT
        END
$       EOJ
        ##XEQ RUN
        ####
             ER2
```

THE CONSOLE TYPEWRITER TYPED THE FOLLOWING LINE WHEN THE CALL EXIT STATEMENT WAS EXECUTED,

EXIT 003000000003000000 CHECK

THE FIRST 3 INDICATED ERROR 3,FLOATING POINT DIVIDE BY ZERO
THE SECOND FLAGGED 3 INDICATES I/O ERROR 2

V. IBM FORTRAN II V.

This section describes those IBM FII instructions which are not yet available to users of the KFII system. Included in the section are instructions for using the disk to store data and program segments, instructions for varying the number of significant digits stored in the machine, and instructions for using the subroutine, ROUND.

The IBM Monitor II System Reference Manual (C26-5774-0) includes detailed operating instructions for using the Monitor II system, and detailed instructions for programming using the IBM FII compiler. The user should note that there are many differences in the language required by the KFII system and the IBM FII system. The IBM manual should be consulted when writing for the IBM FII compiler.

V.1 Varying the word length (number of significant digits stored in the machine) V.1

The IBM FII compiler will automatically convert numbers either to floating point form in which the number is specified as a decimal fraction times a power of 10, or to fixed-point form in which the number represents an integer. The number of decimal digits, x, and the number of integer digits, y may be varied by using a FANDK control card.

General form:

*FANDKxxyy

where: * is punched in col. 1.
FANDK is punched in col. 2-6.
xx is the number of decimal digits (that is, the length of the mantissa) punched in col. 7-8.
yy is the digits used to represent an integer, punched in col. 9-10.

The range of x is 2 through 28; the range of y is 4 through 10.

Example:

*FANDK1505

Fifteen decimal digits plus 2 exponent digits will be stored for each real (floating point) number, and five digits will be stored for each integer (fixed point) number.

The FANDK card must follow the ##FORX control card, and must precede the source deck. (See section V.4). If a FANDK card is not used the compiler will automatically convert floating point numbers to 8 decimal digits plus 2 exponent digits, and fixed point numbers to 4 digits.

V.2 Library functions added to the IBM FII compiler V.2

The function ROUND will round an arithmetic expression to a specified number of significant digits.

General form

CALL ROUND (A,B)

The arithmetic expression A will be rounded to B significant digits.

V.3 Instructions which use the 1311 disk for storage of data V.3

The following instructions, DEFINE DISK, FIND, RECORD are used to store data on the 1311 disk. The data must be in machine core storage. The FETCH statement is used to read data from the disk into machine core storage.

V.3a DEFINE DISK statement V.3a

The DEFINE DISK statement specifies to the FORTRAN processor the size and quantity of data records that will be used with a particular program and its associated subprograms. The statement must appear in the main program when Disk I/O statements appear in any part of the program or subprograms, and may appear only once in that program. Thus, all subprograms used by that main program must use the same size record defined in the statement.

The DEFINE DISK statement must not be used in subprograms.

General form

DEFINE DISK (N_1, N_2)

where: N_1 is a fixed point (integer) constant which specifies the number of variables contained in a record of data.

N_2 is a fixed-point constant which specifies the number of data records that will be used by the main program and its associated subprograms.

The program may use either one-sector records or two sector records for data storage.

A record of data may contain up to 100 digits if the programmer chooses to use a one-sector record or up to 200 digits if the programmer chooses to use a two-sector record. The user should estimate the number of digits he wishes to store on the disk for each record and then choose either one or two sector records, whichever makes the most efficient use of disk storage. V.3a

Example:

DEFINE DISK (10, 3)

Stores a vector containing 13 variables on the disk. Given that the word length is 10, 8 for the mantissa and 2 for the exponent the total number of digits to be stored is 13 x 10. These 130 digits are stored on 3 one-sector records. Space for 7 additional variables is reserved on every third record. This space is not used by the program.

Example:

DEFINE DISK (16, 50)

Stores a 20 x 40 matrix on the disk given that the word length is 12 digits, 10 for the mantissa and 2 for the exponent, the total number of digits to be stored is 800 x 12. These 9,600 digits can be stored on 50 data records if 192 digits are stored on each two sector record. Thus 16 variables may be stored on each data record. Space for 8 digits is reserved on each record. This space is not used by the program.

Note: If two sector records are specified the two sectors are treated as a unit. Thus the digits of a variable may be split between the sectors of a two sector record. (See sample problem 13)

V.3b Assigning numbers to disk sectors V.3b

All disk records are referenced by an integer (fixed-point) variable name. The current value assigned to the variable name references the first sector in a record when the variable name is used in a RECORD or FETCH statement. The programmer must assign, in a simple initialization statement, a value to the first disk sector before the first RECORD statement is executed.

Example:

IREC = 1

During execution of a RECORD or FETCH statement the Monitor Control system will assign numbers to every sector if one sector records are specified, to every second sector if two sector records are specified. The numbers assigned to the sectors will start with the value assigned to the first sector (one (1) if it is the first RECORD statement in a program, or one greater than the number assigned to the last sector during execution of the preceding RECORD or FETCH statement) and will be incremented by one for each sector if one sector records are specified. If two sector records are specified the number will be incremented by one for every second sector.

V.3b

If more than one RECORD statement is included in a program, the programmer must determine the numbers assigned to the first sector of each record. Before execution of a FETCH statement the programmer must set the variable name assigned to the RECORD equal to the number assigned to the first sector of the RECORD when the RECORD statement was executed. (See sample problem 14).

V.3c RECORD statement V.3c

The Record statement is used to write data from core storage into the 1311 Disk storage drive.

General form

RECORD (IREC) List

where: IREC is a non-subscripted or subscripted fixed-point variable, assigned by the programmer to represent the record. The same variable name is used when referring to the record in either a FIND, or FETCH statement. IREC must be set equal to 1, in the program before the first RECORD statement in the program is read. This assigns the number one to the first sector of the record. All records are referenced by the number assigned to the first sector of the record.

LIST is as described in input/output statements.

The data designated by the list are written as the total record represented by (IREC). If the list specifies more items than can be contained in one or two disk sectors, the value of (IREC) is incremented by one, by the Monitor Control system, and writing proceeds to the next sequential seztor. This procedure continues until either all items in the list have been written or until the end of the area specified by N_2 in the DEFINE DISK statement has been reached.

V.3d FETCH statement V.3d

The FETCH statement is used to read date from the 1311 disk into machine core storage.

General form

FETCH (IREC) List

where: IREC is the variable name that was assigned by the programmer to the Record in the RECORD statement. Before the FETCH statement is executed IREC must be set equal to the number assigned to the first sector of the record. (See sample problem 14).

LIST is as described in input/output statements. V.3d

The data designated by the list is read from the record specified by IREC. If the list specifies more items than can be obtained from one record, than the value of IREC is incremented by one and reading proceeds from the next sequential record. This procedure continues until either the list has been "satisfied" or until the end of the area specified by N_2 in the DEFINE DISK statement has been reached. At the conclusion of a read operation, the value of IREC is one greater than the number of the last record read.

V.3e FIND statement V.3e

This statement may be used before a RECORD statement or a FETCH statement to cause the disk access arm to be positioned over a cylinder which will subsequently be read from or written on.

General form

FIND (IREC)

where: IREC is a nonsubscripted or subscripted fixed-point variable name which references the disk to be read from or written on.

V.4 Operating instructions, Monitor Control cards V.4

The following monitor control cards must precede a source deck written for the IBM FII compiler

```
Card Col.     1 2 3 4 5 6 7 8 9....32...........62........................80
              # # J O B            USER'S NAME    OPTIONAL USER IDENTIFICATION
              # # F O R X
```

V.5 SAMPLE PROBLEMS

SAMPLE PROBLEM 13

```
WWJOB 5
WWFORX
*FANDK0506
C  PROGRAM STORES 25 VARIABLES, EACH VARIABLE CONSISTING OF
C  8 DIGITS ON THE DISK
      DIMENSION A(25)
C  DEFINE A TWO SECTOR RECORD CONTAINING 200 DIGITS
      DIMENSION A(25)
      DEFINE DISK(25,1)
      DO 10 K=1,25
   10 A(K)=K
      PUNCH 20,(A(K),K=1,25)
   20 FORMAT(5F14.2)
C  THE NAME OF THE FIRST RECORD MUST BE ASSIGNED
C  THE INTEGER VALUE 1
      IA=1
      RECORD(IA)(A(K),K=1,25)
      DO 30 K=1,25
      B=K*K
   30 A(K)=B
      PUNCH 20,(A(K),K=1,25)
C  THE NAME OF THE FIRST RECORD MUST BE ASSIGNED THE INTEGER
C  VALUE 1
      IA=1
      FIND (IA)(A(K),K=1,25)
      FETCH (IA)(A(K),K=1,25)
      DO 40 K=1,25
   40 A(K)=A(K)+1.
      PUNCH 20,(A(K),K=1,25)
      CALL EXIT
      END
```

1.00	2.00	3.00	4.00	5.00
6.00	7.00	8.00	9.00	10.00
11.00	12.00	13.00	14.00	15.00
16.00	17.00	18.00	19.00	20.00
21.00	22.00	23.00	24.00	25.00
1.00	4.00	9.00	16.00	25.00
36.00	49.00	64.00	81.00	100.00
121.00	144.00	169.00	196.00	225.00
256.00	289.00	324.00	361.00	400.00
441.00	484.00	529.00	576.00	625.00
2.00	3.00	4.00	5.00	6.00
7.00	8.00	9.00	10.00	11.00
12.00	13.00	14.00	15.00	16.00
17.00	18.00	19.00	20.00	21.00
22.00	23.00	24.00	25.00	26.00

NOTE: The W's punched in the Monitor control cards are record marks, multi-punch 028.

SAMPLE PROBLEM 14

```
##JOB 5
##FORX
C  PROGRAM STORES 25 VARIABLES,EACH VARIABLE CONSISTING OF 10 DIGITS ON
      DIMENSION A(25)                                           THE DISK
C  DEFINE 6 ONE SECTOR RECORDS
      DEFINE DISK (10,6)
      DO 10 K=1,25
   10 A(K)=K
      PUNCH 20,(A(K),K=1,25)
   20 FORMAT(5F14.2)
C  THE NAME OF THE RECORD MUST BE INITIALIZED TO EQUAL 1
      IA=1
      RECORD(IA)(A(K),K=1,25)
      DO 30 K=1,25
      B=K*K
   30 A(K)=B
C  THE NAME OF THE RECORD MUST BE REINITIALIZED TO EQUAL 1
      PUNCH 20,(A(K),K=1,25)
      IA=1
      FIND (IA)(A(K),K=1,25)
      FETCH(IA)(A(K),K=1,25)
      DO 40 K=1,25
   40 A(K)=A(K)+1.
      PUNCH 20,(A(K),K=1,25)
      RECORD(IA)(A(K),K=1,25)
      DO 50 K=1,25
   50 A(K)=A(K)+2.
      PUNCH 20,(A(K),K=1,25)
C  THE NAME OF THE RECORD MUST BE REINITIALIZED TO EQUAL 4
      IA=4
      FIND (IA)(A(K),K=1,25)
      FETCH(IA)(A(K),K=1,25)
      DO 60 K=1,25
   60 A(K)=A(K)+1.
      PUNCH 20,(A(K),K=1,25)
      CALL EXIT
      END
```

```
          1.00          2.00          3.00          4.00          5.00
          6.00          7.00          8.00          9.00         10.00
         11.00         12.00         13.00         14.00         15.00
         16.00         17.00         18.00         19.00         20.00
         21.00         22.00         23.00         24.00         25.00
          1.00          4.00          9.00         16.00         25.00
         36.00         49.00         64.00         81.00        100.00
        121.00        144.00        169.00        196.00        225.00
        256.00        289.00        324.00        361.00        400.00
        441.00        484.00        529.00        576.00        625.00
```

2.00	3.00	4.00	5.00	6.00
7.00	8.00	9.00	10.00	11.00
12.00	13.00	14.00	15.00	16.00
17.00	18.00	19.00	20.00	21.00
22.00	23.00	24.00	25.00	26.00
4.00	5.00	6.00	7.00	8.00
9.00	10.00	11.00	12.00	13.00
14.00	15.00	16.00	17.00	18.00
19.00	20.00	21.00	22.00	23.00
24.00	25.00	26.00	27.00	28.00
3.00	4.00	5.00	6.00	7.00
8.00	9.00	10.00	11.00	12.00
13.00	14.00	15.00	16.00	17.00
18.00	19.00	20.00	21.00	22.00
23.00	24.00	25.00	26.00	27.00

VI OTHER 1620 PROGRAMS AVAILABLE VI

VI.I AFIT Fortran VI.1

The AFIT system is a Fortran system designed for use on the 1620 Model I. Programs which do not fit into core storage when the Load and Go system is used may fit when the AFIT system is used. The Load and Go compiler allows approximately 4,200 digits for the compiled program. The AFIT compiler allows approximately 14,000 digits for the compiled program.

The AFIT system consists of an AFIT compiler, punched on cards, and language specifications. The language specifications are modifications of the basic Fortran language discussed in Section II. Fortran language specifications which apply specifically to the AFIT system can be found in a manual, AFIT Fortran, available in the Center office.

Unlike a Load and Go system, the AFIT compilation proceeds in two stages. There is also a precompilation stage that checks the program for clerical errors, punctuation, and spelling. During the first stage of actual compilation, the AFIT processor reads the source deck and produces another deck known as the object deck. The second stage is the execution stage during which the new object deck is read in and run. The manual, AFIT Fortran, also lists operating instructions for using the AFIT compiler. A subroutine which finds the arcsine of a number has been added to the library subroutines. The card deck containing the AFIT compiler may be found on top of the 1620 Model I card reader.

VI.2 SPS (Symbolic Programming System) VI.2

For those who wish to write in a language more intimately associated with computer operation, we provide the programming language described in IBM's Reference Manual for the IBM 1620/1710 Symbolic Programming System #C26-6500. A copy of the SPS assembler, punched on cards for use on the 1620 Model I, is available at the Center office.

The 1620 Model II Monitor System includes a SPS II-D assembler stored on the disk. Instructions for using the assembler may be found in IBM 1620 Monitor II System Reference Manual C26-5774-0.

The following subroutines have been added to the SPS II-D subroutine set 02 by NCE Computing Center staff members:

1. OUTC, output conversion, see write-up for LIB. 1.6.053

2. INC, input conversion, see write-up for LIB. 1.6.053

3. FC, Floating compare, see write-up for LIB. 7.0.050

VI.3 Programs written by the NCE Computing Center staff and stored on the 1311 disk VI.3

The following programs were written by the Center staff for general use and stored on the 1311 disk. They are ready to be used with the appropriate Monitor Control cards.

VI.3a Butler VI.3a

The program Butler will accept as data a Fortran or SPS source deck, and will repunch the deck as follows:

1. The Fortran deck will have the statement numbers in columns 2-5, column 6 will be blank, the statement itself will start in column 7. A sequence number will be punched at the end of the card.

2. Continuation cards are not produced from a long Fortran statement.

3. The SPS deck will have new page/line numbers punched in column 1-5 starting with the number entered from the typewriter.

Before entering Fortran or SPS source deck, set the console switches as follows:

1 ON for SPS
1 OFF for Fortran
2 and 3 are not used

The following Monitor Control cards are placed in front of the Fortran or SPS source deck:

Card col:	1	2	3	4	5	6	7	8	9	10	11	12.....32	
Monitor Control card	#	#	J	O	B								USER'S NAME
Monitor Control card	#	#	X	E	Q			U	T	L	E	R	

Note: # is a multiple punch 028
The card must be punched as indicated with the USER'S NAME punched in card columns 32-60

VI.3b Equivalence table description VI.3b

A description of the program is available at the Center. The Monitor Control card is punched as follows:

Card col:	1	2	3	4	5	6	7	8	9	10	11	12
Control card	#	#	X	E	Q		E	Q	T	B	L	E

A ## JOB 5 card precedes the ##XEQ card as shown above.

VI.3c Programs written by the NCE Computing Center staff and stored on punched cards VI.3c

The following programs, written for general use are stored on punched cards. A descriptive write-up is available at the Center.

Number	Program name
EEPD 1	Muller's method for finding the roots (real or complex) of an algebraic equation with real coefficients
EEPD 2	Transient response evaluation: Time function obtained from Laplace transform
EEPD 3	Frequency response analysis

VI.4 Library of 1620 programs VI.4

A set of 1620 library programs containing descriptive write-ups of programs available for general use may be found in the Computing Center library. Listed below are the programs which are stored either on the 1311 disk, ready for use on the 1620 Model II, or on cards or tape ready for use on the 1620 Model I or II. A descriptive write-up of each program will be found under the appropriate library program reference number. The descriptive write-up will specify input format and indicate the output format.

VI.4a Programs stored on the 1311 disk VI.4a

A program stored on the 1311 disk is executed when the appropriate Monitor Control card is read by the 1620 Model II. The Monitor Control card signals the system to read the program off the disk into machine core storage. The Monitor Control card also informs the Monitor system that the program is to be executed with the data following the control card. The data should be in the form specified by the program write-up which will be found under the appropriate Library program number. The data should be followed by a card punched with record marks in col. 1-4 (####).

The following list includes a brief description of each program and indicates the Monitor Control card and the Library program number for each program. Monitor control cards are punched with ## in col. 1-2, XEQ in col. 3-5 and the name of the program in col. 7-12. The # is a multiple punch 028.

General form of cards used when executing a program stored on the 1311 disk

Card col	1	2	3	4	5	6	7	8	9	10	11	12.......32
Monitor Control card	#	#	J	O	B							User's Name
Monitor Control card	#	#	X	E	Q	S	.	A	M	E	O	F
Data cards	-	-	-	-	-	-	-	-	-	-	-	-
(specified by	-	-	-	-	-	-	-	-	-	-	-	-
Library program)	-	-	-	-	-	-	-	-	-	-	-	-
End of data card	#	#	#	#								

1. Computation of Bessel functions, first kind, integral order, for arguments in the range greater than 0.001 to less than 200.0.

LIB. No. 3.0.005
Monitor Control card ##XEQSBESSEL

2. Solution for initial value problems involving n first order differential equations by Runge-Kutta-Gill and Hamming's method

LIB. No. 4.0.001
Monitor Control card ##XEQSDIFEQS

3. Solution of simultaneous linear equations. The maximum number of equations is 25.

LIB. No. 5.0.007
Monitor Control Card ##XEQSSIMEQS

4. Calculation of eigenvalues and eigenvectors of real symmetric matrics.

LIB. No. 5.5.016
Monitor Control card ##XEQSEIGENV

5. Computation of the sum, mean, standard deviation, error of estimate, sum of square deviations, and coefficient of variation, for each variable, and t-ratio and degree of freedom, between all pairs of variables, for up to 50 variables.

LIB. No. 6.0.039
Monitor Control card ##XEQSSTATIS

6. Linear regression analysis for all combination of variables. The program selects variables to be included in a complete multiple linear regression analyses.

LIB. No. 6.0.057
Monitor Control card ##XEQSLRAVAR

7. Linear regression of two variables by least squares fit.

LIB. No. 6.0.067
Monitor Control card ##XEQSLR2VAR

8. Electric circuit analysis program

LIB. No. ECAP 1620-EE-02X
Monitor Control card ##XEQSECA

Note: The JOB card for the above program must be punched with an 01 in col. 8-9 as follows:

Card col:	1	2	3	4	5	6	7	8	9
	#	#	J	O	B		5	0	1

9. Finite Fourier analysis including coefficients determination and plot-back program.

   LIB. No. 6.0.056
   Monitor Control card ##XEQSFORIER

VI.4b How to clear memory (MODEL 1)

Various programs may require that the memory be cleared (set to zeros) before they are run. To clear memory:

1. Set all check switches to PROGRAM
2. Depress INSTANT STOP and RESET
3. Depress INSERT
4. Type 160001000000 (12 digits , no spaces)
5. Depress RELEASE and START (or the R/S key)
6. After the MAR lights have cycled through memory at least once, depress INSTANT STOP.
7. Depress RESET

VI.5 PROGRAMS STORED ON PUNCH CARDS OR TAPE

```
      1.1.002 ADDITIONAL INSTRUCTION MACRO SUBROUTINE(CARD)
      1.1.005 MULTIPROCESSING FORTRAN               (TAPE)
      1.1.006 MULTI@PURPOSE SPS CARD OUTPUT COMPRESSO(
1185X01.1.009 LOAD + GO FORTRAN FOR CARD OPERATION  (CARD)5 D
1185-01.1.010 AFIT IMPROVED FORTRAN                 (CARD)6 D
      1.1.011 AN INTERPRETIVE LANG ASSEMBLER IBM 1620
     01.1.012 OSAP ASSEMBLY SYSTEM (CONDENSED DECKS) (CARD)8 D
1185 01.1.012 OSAP ASSEMBLY SYSTEM (SYMBOLIC DECKS)  (CARD)8 D
1185-01.1.014                                       (CRD)12 D
1185-01.1.019           PROGRAM WRITEUP             (CRD)14 D 1620
1185-01.1.019           PROGRAM DECK                (CRD)14 D 1620
1185-01.1.020           PROGRAM WRITEUP             (CRD)14 D 1620
1185-01.1.020           PROGRAM DECK                (CRD)14 D 1620
1185-01.1.020           PROGRAM DECK                (CRD)14 D 1620
1185 01.1.023             DOCUMENTATION             (CRD)21 D 1620
1185-01.1.024           PROGRAM DECK                (CRD)23 D 1620
1185 01.1.024             DOCUMENTATION             (CRD)23 D 1620
1185 01.1.026             DOCUMENTATION             (CRD)24 D 1620
1185-01.1.026           PROGRAM DECK                (CRD)24 D 1620
1185-01.2.003 PROGRAM CONDENSER AND LOADER          (CARD)3 D
```

01.2.005	RELOCATOR PROGRAM	(CARD)3 D
01.2.006	DUMP TO RELOAD	(CARD)5 D
01.2.007	FORTRAN COMPRESSOR-LOADER	(CARD)5 D
01.2.008	A FLEXIBLE CARD READ ROUTINE	(CARD)6 D
1.2.009	FORMAT FORTRAN OBJECT DECK COMPRESSOR	
01.3.003	1620 GENERAL PURPOSE CARD COMPRESSOR	(CARD)2 D
01.3.005	SQUEEZ	(CARD)3 D
01.3.006	SQUISHER	(CARD)6 C
01.3.008	1620 FORTRAN COMPRESSOR AND 75 COL.DUMP	(CARD)6 C
1.4.001	SELECTIVE TRACE	
01.4.002	TRACE PROGRAM FOR CARD I/O	(CARD)1 C
01.4.003	1620 MULTI-TRACE	(CARD)1 C
01.4.004	STROBIC	(CARD)1 D
01.4.005	TRACE AND 1A SIMULATOR	(TAPE)1 C
01.4.006	1620 MULTITRACE	(TAPE)1 C
01.4.007	1620-402 MULTI-TRACE	(CARD)2 D
01.4.008	DYNAMIC TRACE PROG FOR IBM 1620 COMP	(CARD)2 D
01.4.010	GENERAL TRACE ROUTINE	(CARD)3 D
01.5.001	FORTRAN SOURCE TAPE CORR	(TAPE)1 D
01.5.004	POST MORTEM DUMP	(CARD)1 C
01.5.005	UNIVERSAL TAPE DUPLICATOR	(TAPE)
01.5.008	ALPHANUMERIC TAPE DUP. AND CORRECTOR	(TAPE)
01.6.001	REGRESSLON ANALYSIS DATA PREPARATION	(TAPE)1 D
01.6.003	1620 AUTOPLOTTER	(TAPE)1 C
01.6.004	1620 AUTOPLOTTER	(CARD)1 C
01.6.008	FORTRAN I/O ROUTINE FORMAT CONTROL	(CARD)1 D
01.6.015	DYNAMIC DUMP	(CARD)1 C
01.6.017	FORMAT CONTRL SUBROUT FOR CARD FORTRAN	(CARD)1 D
1185-01.6.019	FORTRAN II DIAGNOSTICIAN	(CARD)1 C
01.6.020	402 CORES DUMP	(CARD)2 D
01.6.021	SYMBOL TABLE ANALYZER	(CARD)2 D
01.6.022	ANL MNEMONICS DUMP	(CARD)2 D
01.6.024	IMPROVED HASH TOTAL PROGRAM	(TAPE)
01.6.028	I/O SUBROUT FOR USE IN SYM PROG	(CARD)2 D
01.6.029	PROGRAM INTERPRUPT	(CARD)2 D
01.6.030	CARD DUMP AND LOAD	(CARD)2 D
01.6.031	CARD HASH TOTAL	(CARD)2 D
01.6.032	HASH TOTAL FOR CARDS	(CARD)3 D
01.6.033	FLOATIPG POINT OUTPUT SUBROUTINE	(TAPE)
01.6.042	SBRS. FOR PRESET PREC. F.P. ARITHMETIC	(CARD)4 D
01.6.043	LOGGING PROGRAM	(CARD)4 D
01.6.044	GENERAL COMPRESSOR	(CARD)5 D
01.6.045	FORTRAN COMPRESSOR + MULTI-PROGRAMMER	(CARD)5 D
01.6.047	DBD (DAYS BETWEEN DATES) SUBROUTINE	(CARD)6 D
01.6.049	L106 FLT PT TO FIXED PT CD.O/P ROUT.SPS	(CARD)6 C
1185X01.6.055	PLOT SUBROUTINE FOR FORTRAN	(CRD)10 D
1185 01.6.056	PLOT SUBROUTINE FOR FORTRAN W FORMAT	(CRD)10 D
01.6.058	TRANSLATOR OF ALPHANUMER.TO EXCESS 50	(TPE)10 D
01.6.060	SPS OBJECT DECK ANALYZER	( )
85 01.6.061	1620 RECORD DUMP	(TPE)10 D
02.0.003	INTERPRTIVE SYS PERFRM OPER COMPLEX NO.	(TAPE)1 D
02.0.006	INTERPRETIVE ROUTINE	(TAPE)
1185-02.0.008	FORGO (LOAD AND GO FORTRAN)	(CARD)2 D
1185-02.0.009	FOR-TO-GO (2 PASS FORGO)	(CARD)2 D
02.0.011	INT. SYS. FOR PERFORM OPS COMPLEX NOS.	(CARD)3 D
02.0.012	NOVATRDN I	(TAPE)
03.0.001	VARIABLE FIELD SQUARE ROOT SUBR.	(CARD)1 D

	03.0.002	1620 FIXED POINT SQR (CLOSED) SUBRTN	(CARD)	
	03.0.003	ORTHOGONAL POLYNOMIAL COEFFICIENTS	(CARD)2	D
	03.0.005	COMP OF BESSEL FUNCT. OF INTGRAL ORDER	(CARD)3	D
	04.0.001	SOL.FOR INI.VAL.PROB.N FIRST ORD.D.EQ.	(CARD)3	D
	04.0.002	SOL.FOR INI.VAL.PROB.1 FIRST ORD.D.EQ.	(CARD)3	D
	04.0.003	SOL.FOR INI.VAL.PROB.1 FIRST ORD.D.EQ.	(CARD)3	D
1185	05.0.002	SIMULTANEOUS EQUATION SOLUTION	(CARD)	
	05.0.003	EIGENVALUES OF REAL SYMMETRIC MATRICES	(CARD)1	C
	05.0.004	EIGENVALUES OF REAL SYMMETRIC MATRICES	(TAPE)	
	05.0.005	EVALUATION OF DETERMUNANTS (CARD)		
*1185-	05.0.007	SOLUTION OF SIMULTS LINEAR EQUATIONS	(CARD)1	C
	05.0.008	SIMULTANEOUS EQUALTIONS A LA KING	(CARD)2	C
1185X	05.0.009	CAL. EIGENVALS+EIGENVECTOR OF HY =LDY	(CARD)3	D
	05.0.013	MATRIX INVERSION SUBROUTINE	(CARD)3	D
	05.0.014	SIMULTANEOUS EQUATIONS	(CARD)3	D
	05.0.015	MADAME	(CARD)4	D
1185-	05.0.016	CAL.OF EIGENVALUES + VECTORS OF ZV=LAV	(CARD)5	D
	05.0.017	FLT.PT.MACRO INST FOR SOL OF LIN.SYS.EQ	(CARD)6	D
1185-	05.0.019	EVALUATION OF A DETERMINANT	(CRD)11	D
	6.0.002	MULTIPLE LINEAR NON@REGRESS ANALYSIS		
	06.0.003	SCRAP	(CARD)1	C
	06.0.004	STRAP	(TAPE)1	C
1185-	06.0.007	STEPWISE MULT. LIN. REGRESSION ANALYSIS	(CARD)1	C
	06.0.009	CORRELATING PROGRAM- UP TO 30 VARI	(CARD)	
	06.0.010	ANALYSUS OF VARIANCE	(CARD)	
	06.0.011	NON LINEAR PATCH FOR MLR PROG OF WILDER	(CARD)2	D
*	06.0.012	DISTRIBUTION STATISTICS	(CARD)2	D
	06.0.013	SIMPLE LINEAR CORRELATION	(CARD)2	D
	06.0.014	GENERAL ANOV	(CARD)2	D
	06.0.015	40-40 CORRELATION	(CARD)2	D
	06.0.016	FREQUENCY DISTRIBUTIONS-SINGLEPDBLE COL	(CARD)2	D
	06.0.017	MULTIPLE LINEAR REGRESSION	(CARD)2	D
	06.0.018	MULTIPOE LINEAR REGRESSION	(TAPE)	
	06.0.019	MANN-WHITNEY TEST	(CARD)2	D
	06.0.020	SCATTERGRAM GENERATOR	(CARD)2	D
	06.0.021	CORRELATION COEFFICIENTS (UCRBL 0024)	(CARD)2	D
	06.0.022	PRODUCT MOMENT CORRELATIONS(UCRBL 0004)	(CARD)2	D
	06.0.023	ANALYSIS OF COVARIANCE (UCRBL 0007)	(CARD)2	D
	06.0.024	ANALYSIS OF COVARIANCE (UCRBL 0009)	(CARD)2	D
	06.0.026	ANALYSIS OF VARIANCE (UCRBL 0013)	(CARD)2	D
	06.0.027	ANALYSIS OF VARIANCE (UCRBL 0014)	(CARD)2	D
	06.0.028	ANALYSIS OF VARIANCE (UCRBL 0015)	(CARD)2	D
	06.0.029	ANALYSIS OF VARIANCE (UCRBL 0016)	(CARD)2	D
	06.0.030	ANALYSIS OF VARIANCE (UCRBL 0019)	(CARD)2	D
	06.0.031	STEPWISE REGRESSION (UCRBL 0018)	(CARD)2	D
	06.0.032	ANALYSIS OF COVARIANCE (UCRBL 0025)	(CARD)2	D
	06.0.033	ANALYSIS OF VARIANCE (UCRBL 0026)	(CARD)2	D
	06.0.034	NORMALITY (UCRBL 0027)	(CARD)2	D
	06.0.035	HOMOGENEITY OF VARIANCE (UCRBL 0032)	(CARD)2	D
	06.0.036	MULT RANGE TEST OF MEAN DIF (UCRBL0034)	(CARD)2	D
	06.0.037	MAT INVERSION-SIMULT EQ (UCRBL0052)	(CARD)2	D
	06.0.038	LINEAR CORRELATION COEFFICIENT	(CARD)2	D
	06.0.039	STATISTICS 1	(CARD)2	D
1185X	06.0.041	FACTORIAL ANALYSIS OF VARIANCE	(CARD)3	D
	06.0.042	CONT. FOREST INV.STATISTICAL CHECK PROG	(CARD)3	D
	06.0.043	MULTIPLE REGRESSION PACK FOR CARD 1620	(CARD)3	D

*NOTE: Programs have been modified for Load and Go. A print-out of the modified programs and card source decks are available in the Center office.

	06.0.044	ANAL.OF 2-LEVEL FACTOR EX.FOR CARD 1620	(CARD)3 D
	06.0.045	CORRELATION FOR THE IBM 1620	(CARD)3 D
	06.0.046	CORRELATION FOR THE IBM 1620	(TAPE)3 D
	06.0.049	PROGRAM TO PLOT CONTOURS OF CONST.RESP.	(CARD)4 D
1185-	06.0.050	FISHERS EXACT PROBABILITY FOR 2X2 TABLE	(CARD)4 D
	06.0.056	FINITE FOURIER ANALYSIS	(CARD)5 D
	06.0.057	LIN.REG.ANAL.OF ALL COMB. OF VARIABLES	(CARD)5 C
	06.0.058	MANN WHITNEY U TEST	(TAPE)5 D
	06.0.059	PROBIT ANALYSIS	(CARD)5 C
1185-	06.0.063	FISHERS EXACT METHOD	(CARD)6 D
	06.0.066	STRAP@A STEPWISE REGRESSION ANALYSIS P	(CARD)
	06.0.067	LINEAR REGSS (2 VARIABLES) LEAST SQ FIT	(CARD)
	06.0.075	ROUNDING SUBROUTINE	(CARD)
1185-	06.0.077	RIDGE ANAL. HIGHLY CORRELATED DATA	(TAPE)7 D
	06.0.078	WEIBULL ANALYSIS PACKAGE	(TAPE)
1185	06.0.111	RANDOM EXPONENTIAL NO. GEN. SUBPROGRAM	(CRD)11 D
	07.0.001	POLYNOMIAL CURVE FITTING	(TAPE)1 C
	07.0.002	POLYNOMIAL CURVE FITTING	(CARD)1 C
	07.0.003	1620 FIX POINT SQUARE ROOT	(CARD)1 D
	07.0.005	SQUARE ROOT SUBROUTINE	(CARD)2 D
M1	07.0.006	ARCTANGENT SUBROUTINE	(CARD)2 D
	07.0.007	PLYNOMAL CURVE FIT LEGNDR PLYNOMAL	(CARD)2 D
	07.0.008	SINE-COSINE SUBROUTINE	(CARD)2 D
	07.0.009	1620 RNDOM NUM SUB FOR FORTRAN W/FORMAT	(TAPE)2 D
	07.0.010	DBLE PRECISION FLOATPT ARITH SUBROUT	(TAPE)2 D
	07.0.012	POLY.REGRESSION PROGRAM FOR IBM 1620	(CARD)3 D
	07.0.014	REAL AND COMPLEX RTS OF POLYNOMIALS	(TAPE)4 D
	07.0.015	FORTRAM SYS REL.SUB.FOR GEN.RAND.NOS.	(TAPE)5 D
	07.0.016	INTERPOL.BY NEWTONS METHOD OF 3RD DIFF.	(TAPE)6 D
	07.0.017	CAL. OF REAL ROOTS OF REAL POLY. EQUA.	(CARD)6 D
	07.0.020	EMPIRICAL EQUAT BY METH LEAST SQUARES	(CARD)
	07.0.021	RANDOM NUMBER SUBROUTINE FOR FORTRAN F	(CARD)
	07.0.022	RANDOM NO. SUBR.-FORTRAN W/ FORMAT	(TAPE)7 C
	07.0.023	POLYNOMIAL CURVE FIT.@NEWTONS DIV DIF F	(CARD)
	07.0.024	FOURIER CURVE FITTING @ 1 PASS	(CARD)
	07.0.025	GAMMA SUBROUTINE FOR FORMAT FORTRAN	(CARD)
	07.0.026	ERROR FUNCTION@1620 FORTRAN SUBROUTINE	(CARD)
1185X	07.0.027	POLY. CURVE FITTING AND EVALUATION	(CARD)8 C
1185X	07.0.029	CAL.OF THE ROOTS OF A COMPLEX POLY.EQ.	(CARD)8 D
1185-	07.0.030	LAGRANGE INTERPOLATION	(CARD)8 D
1185X	07.0.031	FOURIER CURVE FITTING - 2 PASS	(CARD)8 C
1185X	07.0.032	CAL. OF THE ROOTS OF A REAL POLY. EQUA.	(CARD)8 D
	07.0.033	CAL.REAL ROOTS-REAL NON-LIN.EQ.IN 1 VAR	(CARD)8 D
1185X	07.0.036	ROOTS OF POLYNOMIAL EQUATION	(CARD)9 D
1185	07.0.037	BESSEL FUNCTION SUB. FORTRAN W FORMAT	(TPE)10 D
1185	07.0.039	BIVARIATE CURVE FIT	(CRD)11 C
1185	07.0.041	CAL. OF THE REAL ROOTS OF A SYS. OF K	(CRD)11 D
	08.2.001	A ONE DIMENSIONAL FEW GRP DIFF. CODE	(CARD)5 D
	08.2.002	CAPTURE GAMMA SHIELDING PROGRAM	(CARD)5 D
	08.2.004	SPEK (SPEEDY KATE)	(CARD)
+	08.2.005	RSNT	(CARD)
	09.1.001	ARDC MODEL ATMOSPHERE SUBROUTINE	(CARD)2 C
	09.2.001	1620 SUBDIVISION PROGRAM	(TAPE)1 C

	09.2.002	CUT AND FILL	(TAPE)1 C
	09.2.003	CUT+FILL	(CARD)1 C
	09.2.004	WATERWAY COMPULATIONS	(TAPE)1 D
	09.2.006	TRAVERSE ANALYSIS	(CARD)1 C
	09.2.007	TRAVERSE ANALYSIS	(TAPE)1 C
	09.2.008	WATERWAY COMPUTATIONS PROG	(TAPE)2 D
	09.2.009	SKEWED BRIDGE ELEVATIONS	(TAPE)2 D
	09.2.012	RELOCATION OFFSETS	(CARD)2 D
	09.2.013	RECANGULAR CONCRETE COLUMN ANALYSIS	(TAPE)2 D
	09.2.014	GEN.VIRTUAL WORK ANALYSIS OF STRUCTURES	(TAPE)3 D
	09.2.016	DTM DESIGN SYSTEM PROGRAM	(CARD)4 D
	09.2.019	DTM DESIGN SYSTEM 40K	(CARD)5 D
	09.2.020	SLOPE STABILITY ANALYSIS	(TAPE)5 D
	09.2.022	COL.ANAL.UNDER AXIAL LD.+2 WAY BENDING	(CARD)5 C
	09.2.024	BEAM CAMBER CALCULATIONS	(CARD)6 D
	09.2.028	BACKWATER CURVE PROGRAM	(CARD)8 D
1185	09.2.032	VEHICLE SIM. AND OPER. COST SYSTEM	(CRD)11 D
	09.3.001	GAS NETWORK ANALYSIS	(TAPE)1 C
	09.3.002	SHORTCUT DISTILLATION	(TAPE)1 C
	09.3.003	GAS NETWORKS ANALYSIS-PUBLIC UT. DEPT.	(CARD)1 C
	09.3.006	ASTM-TO-TBP+TBP-TO-ASTM.DISTILL.CONVER	(TAPE)3 D
1185-	09.3.007	FLASH VAPORIZATION CALCULATIONS	(CARD)6 C
	09.3.008	MULTI.DISTIL.TOWER DESIGN SHT/CUT METH.	(CARD)6 C
	09.3.009	UNIT OPERTS. SIMULATOR VAPOR@LIQD. SYST	(CARD)
1185-	09.3.010	PLATE@TO@PLATE DISTILLATION PROGRAM	(CARD)
	09.4.001	ELECTRIC LOAD FLOW PROGRAM	(TAPE)1 C
	09.4.002	LOCA OF SHUNT CAPACITOR ON RADIAL LINES	(TAPE)1 C
	09.4.003	ELECTRIC LOAD FLOW PROGRAM	(CARD)1 C
	09.4.004	SELECTION OF ECONOMIC CONDUCTOR SIZE	(CARD)1 D
	09.4.005	ECOM CONDCTR SIZE SELEC BY KELVIN LAW	(TAPE)1 C
	09.4.006	SHORTCIRCUIT ANALYSIS	(CARD)1 D
	09.4.007	SHORT CIRCUIT CALCULATION	(CARD)1 D
	09.4.008	TRANSMISSION LOSSESPPENALTY FACTORS	(CARD)1 D
	09.4.009	CURVE FIT-SIMUL PLANT RECORD(NEES-38)	(CARD)1 D
	09.4.010	ECONOMIC DISPATCH DETERMINATION	(TAPE)2 D
	09.4.012	TRANFORMR RATING FOR NORMAL+EMERG OPER	(CARD)2 D
	09.4.013	TRANSIENT STABILITY FOR TEN MACHINES	(CARD)2 D
	09.4.014	TRANSFORMER SHORT TIME LOADING CURVES	(CARD)2 D
	09.4.015	SIMULT WQ SOLPMAT INVERSION/COMPLEX VAL	(CARD)2 D
	09.4.016	NETWORK REDUCTION PROGRAM	(TAPE)3 D
	09.4.020	RAD.3 PHASE LINE DROP CAL.IN 2 PASSES	(CARD)5 D
	09.4.021	BATCH LOAD FLOW	(TAPE)5 D
	09.4.022	ECON GENERATION DISPATCH PROGRAM	(CARD)5 C
1185-	09.4.023	SHORT CIRCUIT ANALYSIS BY MATRIX METHOD	(CARD)6 D
	09.4.025	LOAD ANALYSIS OF A COMMUNICATIONS NETWK	(CARD)
	09.6.001	STRAIN GAGE DATA REDUCTION	(CARD)1 D
	09.6.002	STRAIN GAGE DATA REDUCTION	(TAPE)1 D
1185	09.7.001	DIST OF WATER FLOW IN A PIPE NETWORK	(TAPE)1 C
	09.7.002	GENERALIZED PLOTTER II	(CARD)1 C
	09.7.003	GENERALIZED PLOTTER	(CARD)1 C
	09.7.004	S-100 STRESS ANAL FLNGE WITH TAPED HB	(CARD)1 D
	9.7.006	HYDRAULIC ANAL.OF FLOW IN PIPE NETWORK	(CARD)4 D
	09.7.007	STEAM + WATER PROP. OF EFFICIENCY PROG.	(CARD)5 D
1185-	09.7.008	WATER FLOW IN A PIPE NET. BY H.C.SOL.	(CARD)5 D

1185	09.7.009	MULTICURVE PLOTTING PROGRAM	(CRD)10 C
1185	09.7.010	BLIVIT	(CRD)10 D
	10.1.001	LINEAR PROGRAMMING FOR THE 1620	(TAPE)1 C
	10.1.002	LI. PROGRAMMING CODE FOR THE IBM 1620	(CARD)1 C
	10.1.004	MXV PROGRAM FOR L.P. MATRIX PREPARATION	(CARD)1 D
	10.1.005	TRANSPORTATION PROGRAM FOR THE IBM 1620	(CARD)1 D
1185	10.1.006	LINEAR PROG CODE CRD PUNCH C OPTION FIN OUTP	
	10.1.008	LINEAR PROGRAMMING II	(CARD)6 D
	10.2.001	INVENTORY MANAGEMENT SIMULATOR	(CARD)1 C
	10.2.003	AN INVENTORY MANAGEMENT SIMULATOR	(CARD)
	10.2.004	SALES FORECASTING SIMULATOR	(CARD)
	10.2.005	BOSTON COLLEGE DECISION-MAKING EXERCISE	(CARD)3 D
	10.2.006	MANAGEMENT DECISION MAKING	(CARD)4 D
	10.2.007	EXPONENTIAL SMOOTHING PROGRAM	(CARD)4 D
	10.3.001	LEAST-COST ESTIMATING AND SCHEDULING	(TAPE)
	10.3.002	LEAST COST ESTIMATING AND SCHEDULING	(TAPE)
	10.3.003	LESS	(CARD)1 C
	10.3.004	LESS II	(TAPE)1 C
	10.3.005	CRITICAL PATH SCHEDULING	(CARD)1 C
	10.3.006	1620 PERT	(CARD)2 C
	10.3.007	A PROG FOR ANALYZING THE INVSTMT OF CAP	(CARD)2 C
	10.3.008	1620 NODE NUMBERING	(CZRD)
	10.3.009	1620 PERT	(TAPE)3 D
	10.3.010	ECON.ANALYSIS OF PLANS OUTOUT 1 + 2	(CARD)5 D
	10.3.011	MISS LESS	(CARD)6 C
	10.3.012	MISS LESS	(TAPE)6 C
	10.3.013	MAN@SCHEDULING PROGRAM FOR 1620 IBM	(CARD)
1185	10.3.016	KWIC	(CRD)11 D
1185	10.3.017	STUDENT SCHEDULING	(CRD)12 C
	11.0.001	THE CHINESE BARPRING PUZZLE	(CARD)1 C
	11.0.002	1620 SIMULATION OF A ONE-ARMED BANDIT	(TAPE)1 C
	11.0.003	CHINESE BAR AND RING PUZZLE	(TAPE)1 C
	11.0.004	THE EXECUTIVE GAME	(TAPE)1 D
	11.0.006	BLACKJACK DEMONSTRATION	(CARD)1 C
1185	11.0.007	BBC-VIC BASEBALL DEMONSTRATOR	(CARD)1 C
	11.0.008	BBC-VIC BASEBALL DEMONSTRATOR	(TAPE)1 C
	11.0.009	RANDOM WALK	(TAPE)1 D
	11.0.010	SELF DEMONSTRATOR	(TAPE)1 C
	11.0.012	TIC-TAC-TOE- DEMONSTRATION	(CARD)2 D
	11.0.013	TIC-TAC-TOE-A LEARNING PROGRAM	(CARD)3 D
	11.0.014	TIC TAC TOE-3 DIMENSIONAL	(CARD)4 C
	11.0.015	TIC TAC TOE-3 DIMENSIONAL	(TAPE)4 C
	11.0.016	RQNDOM WALK DEMO	(TAPE)5 C
	11.0.017	RANDOM WALK DEMO	(CARD)5 C
1185-	11.3.030	PERFECT NUMB DEMONS PROG	(CRD)
1185	12.3.001	UNIVERSAL OUTPUT SUBROUTINE	(CRD)10 D
1185-	12.4.001	1710 APP.TO STEAM GEN UNIT-SYSTEM DEMO.	(CARD)6 C
	13.0.001	PLOT SUBROUTINE	(CARD)4 D
	13.0.002	LINK SUBROUTINE	
1185-	13.0.003	NORTHEASTERN UNIV. TEST SCORING PROGRAM	(CARD)8 D

VII PUNCHED-CARD EQUIPMENT

VII.1 Card punch (Model 026)

Three card (or key) punches are available at the Center. During operation of the 1620 Model II, the card punch next to the computer is reserved for short corrections. A fourth card punch is available at Tiernan Hall, 240 High Street.

Single-card punching

The ON - OFF switch is a toggle switch on the card stacker on the upper left-hand corner of the punch.

Slide the blank card in the punching station (the right-hand station) and press the key on the keyboard marked REG (register). The card should engage; if it does not, make sure the right edge of card is under hook.

With the card engaged you can punch any combination of numbers, letters, or special characters present on the keyboard. The keyboard works much as does a typewriter keyboard: the punch is normally in alphabetic shift; to punch numbers, or characters on the upper level of the keys, press the numeric shift, a key marked "NUM". This must be held down while you are punching the numbers or characters. To punch more than one number or sign, in the same column hold down MULP PCH while punching. When you have finished the card, press REL to release the card. Then push REG, REL, REG to get the card through the reading station (the left-hand station) and up into the card stacker.

Single-Card duplicating and correction

Slide the card to be duplicated and/or corrected into the left-hand (reading station), and a blank card into the right-hand (punching station), and push REG. For duplication push DUP, and note that a pointer in the window at the center of the punch indicates which column of the card is passing by the stations. To insert corrections, stop duplicating at the appropriate column and type in the corrections. Duplication of the rest of the card can then be done.

Multiple-card punching

Put your blank deck of cards in the feed hopper on top of the punch at the right between the spring-loaded follower and the front of the punch. With the AUTO FEED toggle above the keyboard off, you must push FEED and REG to get a card to the punching station. With the AUTO FEED toggle on, pushing REL will accomplish the same thing. (You must start the process by pushing REL twice)

Note (-) The minus signs on the card punch is the one on the right (on the key which says SKIP). It gives an "eleven" punch in either mode. The dash on the = key gives and "8" and "4" punch.

Note (O) Be careful not to use the letter O (alphabetic mode) for the number (zero), 0, (numeric mode).

Control cards

When you are punching a deck of cards it often saves time to set up a control card to control the punching in the various columns of the card. The control can set the punch in the alphabetic mode or the numeric mode, determine which fields are to be duplicated, skipped, etc.

The control unit visible through a small window in the cover, consists of a cylinder around which a card is wrapped. The cover swings back to permit access. Once the small V-switch located below the cover window is turned off (down to the right). The cylinder can be removed from the spindle and the card changed. To put a control card on the cylinder, hold the cylinder with the vertical chrome strip towards you and the lever at the top. Turn the lever to the extreme left. Slip the right-hand end of the control card under the left-hand side of the chrome strip and push the bottom of the card down against the ledge at the bottom of the cylinder. Check that the card is straight by looking at the two small holes in the chrome strip. The cylinder should not show at the edge of the card through these holes. To clamp this end of the card, move the locking lever at the top around to its center position. Then wrap the control card around the cylinder and slip the free end under the other side of the chrome strip, making sure the card is wrapped snugly around the cylinder. Lock the card in place by moving the top lever to the right. Gently replace the cylinder, being sure the bottom pin is seated, and close the cover. Turn the V-switch to the left, turn on the AUTO FEED, AUTO SKIP, Turn the V-switch to the left, turn on the AUTO FEED, AUTO SKIP, AUTO DUP, and PRINT switches, and the punch is ready to operate.

A control card may be removed from the control cylinder by reversing the sequence given above.

The format for a cylinder control card consists of variations of 4 punches:

\+ continues whatever operation was done in the previous column

\- causes the punch to skip that column

0 casues automatic duplication (numeric) of the same

1 activates the alphabetic shift

blank activates the numeric shift

An example of a control card is the following:

```
column
123........                                                            80

-++++1AAAAA1AAA1AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA0+b+++
```

which causes an automatic skip to column 6, then a shift to alphabetic mode (an "A" is a combination of a "+" and a "1" punch) and continues the shift. The reason for the 1 in columns 12 and 16 is to break the field definition so that pushing the "SKIP" key will advance the card to the next field from wherever it was positioned in the previous field. Columns 75 and 76 will cause automatically duplication from the card at the reading station into the card at the punching station. The last four columns (column 77 is b for a blank, i.e., no punch) permit numeric card numbering to be punched.

VII.2 Printer (IBM 407 Accounting Machine) VII.2

Control Panels

The format of the printing or "listing" from cards is controlled by the control panel and switches on the right-hand end of the machine. The control panel is held in a drawer that tilts outward from the printer so that the panel can be slid in or out. Never try to run the machine unless a control panel is securely in the holder and the drawer is closed.

Ordinarily the "Reproduce 80 - 80" control panel is used to print cards exactly as they are punched. Other control panels can be wired to distribute the information on the cards across a printed line in various formats. Wiring diagrams are included in the 407 manual.

Operation

Turn on the ON-OFF switch on the left-hand side of the machine. Put the cards 9-edge leading, face down, into the card hopper to the left. Hold down the START button for three cycles to start the cards feeding through the machine; hold it down again to get the last cards through and out.

If the CARD FEED STOP light comes on, the bottom card of the feed deck is probably bent. Duplicate this card and replace it, press the STOP then START to continue the listing. If one of your cards is missing, you will have to get help from a staff member in extricating it from the machine.

VII.3 Reproducer (Model 519) VII.3

The reproducer's major use is in duplicating decks and in punching regular punched cards from mark sensed cards (See Below VII.3). It can also compare two decks, rearrange the columns of a card, and punch sequence identification numbers.

There are two card feeds on the top of the reproducer. The left-hand one is for the deck to be read and the right-hand one is for the deck to be punched. The control panel holder is below the card feeds.

Reproducing

Use the panel marked "80 - 80 REPRODUCING". Put the deck to be reproduced in the read feed (left-hand hopper) face down, 12 or top edge to the right, and put blank cards in the punch feed in the same relative position. Turn on the switch on the right side of the machine. Hold the START button down for three cycles, and then the machine should go by itself. Hold the START button down again to get the last card to go through. The old deck will come out in the left stacker; the new duplicate deck will come out in the right stacker.

Error stops on reproducer

If the machine stops with the red light labeled "COMP" an error in the duplicating process has been detected. In the window low down on the front of the machine metal pointers indicate the columns containing errors.

Remove the cards not yet processed from the hoppers, and run the cards in the machine out by holding down the "START" button. Take the top three cards off both piles. Put the three from the old deck back in the left hopper. Throw out the three from the new deck. Put the remaining unprocessed cards back in their respective hoppers.

Pull up on the lever beside the compare light until the light goes off. Press START as in the beginning.

Save the portion of the new deck already punched -- it is valid. When the machine has finished processing, combine the parts of the new deck.

Sequence numbering of cards

To punch sequence numbers into columns 76-80 of a deck of cards use the control panel labeled "SEQUENCE NUMBERING". Put a card containing zeros in columns 76-80 on top of the deck to be numbered and place the deck face down, top edge to the right, in the right-hand hopper. Certain adjustments can be made inside the machine to permit the sequence numbers to be printed on the cards. See a member of the staff for more details.

Partial reproducing and gang punching

It is possible to wire a control panel to reproduce only certain columns on the card (possibly rearranging them), or to insert the same information in the same columns of several cards. (The latter is called "gang-punching".) Consult a member of the Computing Center Staff if you want help.

Mark Sensing

Mark sensing is based on the principle that a special pencil mark with a high graphite content can conduct electricity. The mark-sensing device on the reproducer reads the pencil marks on the cards and punches corresponding holes. Each mark-sensing column covers three punching columns so that up to 27 columns of data can be marked on a card. For our purposes the first mark-sensing column will be converted to the first punched column, etc., so that the 27 columns spanning the marked card become holes in the first 27 columns of the punched card.

We plan to try to lighten the load on the card punches and also make card preparation more convenient for our users by providing the special cards for mark sensing and the device on the reproducer to convert them to punched cards. The special pencils required can be purchased at the bookstore. The allowance of 27 columns should be adequate for almost any Fortran statement. (Also see Appendix "Use of Mark Sense Cards")

Marked to punched-card conversion

Place the deck of marked cards in the right hopper of the reproducer. Use the "Mark Sensing" board. The marked cards will be punched.

VII.4 Sorter (Model 082) VII.4

Place the cards carefully (this is a fussy device) in the hopper at the right end of the machine with 9 (bottom) edge toward the machine, face down.

Set the column on which you wish to sort by moving the crank until the pointer rests on the right number. You can move it longer distances without cranking by pushing down on the round release button on the side of the pointer. The "ON" switch is a round series of suppression switches used mainly in alphabetic sorting. Consult the manual if you wish to use this option. When sorting, always sort on the least significant digit first or, in a field, start on the right-hand column.

VII.5 Character Coding on Cards VII.5

Punched or marked cards have one character per column. The rows of the column are called, from top to bottom:

12 (or +)
11 (or -)
0
1
2
3
4
5
6
7
8
9

Thus, the number 3 has a punch in row 3 and so on. Letters and special characters have more than one punch per column. A "J" for example has an 11 and a 1 punch. The complete set of character codes are:

Alphameric Character	Card
(Blank)	(Blank)
. (Period)	12, 3, 8
)	12, 4, 8

Alphameric Character	Card
+	12
$	11, 3, 8
*	11, 4, 8
- (Minus)	11
/	0, 1
, (Comma)	0, 3, 8
(	0, 4, 8
=	3, 8
—— (Dash, not a minus)	4, 8
A	12, 1
B	12, 2
C	12, 3
D	12, 4
E	12, 5
F	12, 6
G	12, 7
H	12, 8
I	12, 9
J	11, 1
K	11, 2
L	11, 3
M	11, 4
N	11, 5
O	11, 6
P	11, 7
Q	11, 8
R	11, 9
S	0, 2
T	0, 3
U	0, 4
V	0, 5
W	0, 6
X	0, 7
Y	0, 8
Z	0, 9
Numbers 0, 1, ..., 9	0, 1, ..., 9 respectively

APPENDIX

A. NOTES ON THE STORAGE OF INTEGER AND REAL NUMBERS

A real number may be expressed as a decimal times a power of 10. All real numbers are stored in computer memory as decimals; the appropriate power of 10 is stored with the decimal as follows:

.XXXXXXXX   XX
MANTISSA   EXPONENT

Where: The mantissa is the decimal portion of the number. The first digit of the mantissa (after the decimal point) is always non-zero.

The mantissa consists of eight digits. (IBMFII-D does permit the user to specify an alternate mantissa length. See Section V.1 "Varying the word length.")

The exponent is a two digit integer. The mantissa multiplied by ten to the exponent is equal to the real number as expressed in the source program or data. The range of the exponent is discussed in Section II.2a "Real Constants." Note that the exponent is sometimes referred to as a characteristic.

Example:

A = (B+500.22*C) / (.005*D+4.)

500.22 is stored as 03 (Exponent) .50022 (Mantissa)

$.50022*10^3 = 500.22$

.005 is stored as -02 (Exponent) .5 (Mantissa)

$.5 * 10^{-2} = .005$

4. is stored as 01 (Exponent) .4 (Mantissa)

$.4 * 10^1 = 4$

Integer(fixed point) numbers are stored in digit form as follows:

XXXX   L. & G.
XXXXX   K F II

Arithmetic operations may be directly performed on integer numbers. If the result is more digits than permitted by the compiler the high order digits are truncated.

Example:

I= 9986 + 24

I is stored as 0010 when the statement is processed using the Load and Go processor.

When performing arithmetic operations on real numbers the computer must be instructed to handle the mantissa and exponent parts of the number. The 1620 Model II has built in components capable of carrying out real (floating point) arithmetic. The 1620 Model I does not have this capability and the processor must include subroutines which carry out the real (floating point) arithmetic. (For a discussion of floating-point arithmetic see Kuo, S. Shan, "Numerical Methods and Computers," pp 28-29.) Note that when arithmetic operations develop a mantissa with more than 8 digits the low order digits are truncated.

B. NOTES ON THE USE OF MARK SENSE CARDS

Use of Fortran Mark Sense cards will enable programmers to prepare their source decks without losing time waiting for a free key punch machine. The mark sense cards and the IBM graphite pencil may be purchased at the bookstore. Instructions for marking the cards are listed on pages 124, 125 of the Handbook. A Computing Center staff member will process mark sensed decks on the IBM 519 reproducer during the hours listed on the Computing Center bulletin board. The decks should be placed in the box marked IN on the 519. A punched deck and 407 print-out of the deck will be returned to the OUT box. To assure the return of the proper deck to the proper programmer the programmer is required to place the following two cards at the top of the mark sensed deck.

Card cols. 12 ....20.........................80

Card 1: Z

Card 2: NAME PROBLEM NUMBER

Where: The first card is blank except for a Z in column 80.

NAME is the name of the user.

PROBLEM NUMBER is the problem number assigned to the user.
Mark the face of Card 1 with the name of the user and the initials "F.C." (Use a magic marker type pencil) Mark the back of the last card of the source deck with the initials "L.C."

The Z in column 80 punched on the first card signals the IBM 407 to start printing the following source deck on a new page. Switch 3 (on the 407) must be set to the ON position when batch processing print-outs of mark sensed decks.

The above two cards can be key punched at the beginning of the semester and re-used for each mark sensed deck.

C. NOTES ON MODEL II-BATCH PROCESSING

1. Computing Center batch processing schedule

A Computing Center staff member will process Fortran source decks prepared for batch processing on the Model II during the hours listed on the Computing Center bulletin board. Source decks should be left in the IN box next to the Model II. Printed Output and the source deck will be placed in the OUT box. Please note that all output should be printed. All decks must include correctly punched control cards. Mark the first card with your name and the last card "L.C."

2. Listing of required control cards - KFII

The following cards should be punched at the beginning of the semester and can then be used throughout the semester for each Kingston Fortran deck submitted for batch processing:

```
Card columns:  123456789..........20 ......................50

Card 1.        ##JOB

Card 2.        ##XEQ KF2

Card 3.        $     JOBNAME OF PROGRAMMER    OPTIONAL USER IDENTIFICATION

Card 4.        #     EOJ

Card 5.        ##XEQ RUN

Card 6.        ####
```

Where: The #'s (record marks) are multiple punch 028
NAME OF PROGRAMMER is punched in columns 10-43.
Any additional user identification may be punched in card columns 45-80.

See Section IV.10 "Operating instructions, control cards" for instructions on placing the control cards in the source deck.

It is suggested that the user use the Kingston Fortran processor for speed in compiling and executing a Fortran program. However programs written in Fortran II-D may be batch processed with Kingston programs if the correct control cards are included with the source deck.

3. Listing of required control cards - IBM FII-D

Card columns:	123456789 32	................60.62	80
Card 1.	##JOB	NAME OF PROGRAMMER	OPTIONAL USER IDENTIFICATION
Card 2.	##FORX		

Where: The #'s (record marks) are multiple punch 028, NAME OF PROGRAMMER is punched in card columns 32-60. Any additional user identification may be punched in card columns 62-80.

4. Listing of required control cards - disk stored programs.

Card columns:	123456789..........32	62	80
Card 1.	##JOB	NAME OF PROGRAMMER	OPTIONAL USER IDENTIFICATION
Card 2.	##XEQ NAME OF PROGRAM.		

Where: The #'s (record marks) are multiple punch 028, NAME OF PROGRAMMER is punched in columns 32-60, card columns 62-80 may be punched with any optional user identification, NAME OF PROGRAM is the name of the program currently stored on the disk. (See Section VI.4a for a listing of programs available for disk storage) The program name is punched starting in Card column 7.

INDEX TO ACCEPTABLE FORTRAN STATEMENTS

NOTE:

G same as general form

FII permitted in IBM FII-D, See FII

KF permitted in KFII, See KFII

LG permitted in L. and G., See L. and G.

MS See 1620 Monitor I System Reference Manual, Fortran II-D

N.P. Not permitted

CALL SKIP	KF (See NOTE p.1)	N.P.	53	N.P.
CALL SOLVE	KF	N.P.	78	N.P.
CALL SORT	KF	N.P.	78	N.P.
COMMON	KF	N.P.	66,69	MS
COMPUTED GO TO	15	25	50	MS
CONTINUE	19	G	G	MS
DATA	KF	N.P.	67	N.P.
DIMENSION	21	G	G,69	MS
DO	16	26	51	MS
END	19	27	53	MS
EQUIVALENCE	KF	N.P.	66,69	MS
FORMAT	20	27	56	MS
FETCH	FII	N.P.	N.P.	106
FIND	FII	N.P.	N.P.	107
FUNCTION name	KF	N.P.	71	MS
IF(SENSE SWITCH)	16	25	51	MS
PAUSE	19	G	52	MS
PRINT	LG	27,28	55,65	MS
PUNCH	20	28	55	MS
PUNCH TAPE	KF	N.P.	55	MS
READ	19,20	28	53	MS
RECORD	FII	N.P.	N.P.	106
REREAD	KF	N.P.	53	N.P.
RETURN	KF	N.P.	75	MS
STOP	19	27	52	MS
SUBROUTINE name	KF	N.P.	74	MS
TYPE	20	27	55	MS
Unconditional GO TO	15	G	G	MS