# NCE COMPUTING CENTER HANDBOOK

## CONTENTS

### INTRODUCTION

<table>
<thead>
<tr>
<th>I</th>
<th>NCE COMPUTING CENTER</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.1</td>
<td>Equipment available</td>
<td>1</td>
</tr>
<tr>
<td>I.2</td>
<td>Applying for use of the 1620 computer</td>
<td>1</td>
</tr>
<tr>
<td>I.3</td>
<td>Charges for computer time</td>
<td>2</td>
</tr>
<tr>
<td>I.4</td>
<td>Scheduling computer time</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>a. Class use of computer time</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>b. Student operation of the 1620 Mod I computer</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>c. Scheduled Load and Go periods</td>
<td>3</td>
</tr>
<tr>
<td>I.5</td>
<td>In-Out boxes</td>
<td>5</td>
</tr>
<tr>
<td>I.6</td>
<td>Punched cards</td>
<td>5</td>
</tr>
<tr>
<td>I.7</td>
<td>References and manuals</td>
<td>7</td>
</tr>
<tr>
<td>I.8</td>
<td>Staff</td>
<td>7</td>
</tr>
<tr>
<td>I.9</td>
<td>Courses in programming</td>
<td>7</td>
</tr>
</tbody>
</table>

### II BASIC ELEMENTS OF THE FORTRAN LANGUAGE

<table>
<thead>
<tr>
<th>II</th>
<th>BASIC ELEMENTS OF THE FORTRAN LANGUAGE</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.1</td>
<td>Source program and object program</td>
<td>9</td>
</tr>
<tr>
<td>II.2</td>
<td>Arithmetic statements</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>a. Constants</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>b. Variables</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>c. Arithmetic operation symbols</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>d. Arithmetic expressions</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>e. Replacement statements</td>
<td>14</td>
</tr>
<tr>
<td>II.3 Control statements</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>a. Unconditional GO TO statement</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>b. Computed GO TO statement</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>c. Arithmetic IF statement</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>d. IF (SENSE SWITCH) statement</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>e. DO statement</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>f. CONTINUE statement</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>g. PAUSE statement</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>h. STOP, END, and CALL EXIT statements</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>II.4 Input/Output statements</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. General form for input statements for Fortran without Format</td>
<td>19</td>
</tr>
<tr>
<td>b. General form for input statements for Fortran with Format</td>
<td>20</td>
</tr>
<tr>
<td>c. General form for output statements for Fortran without Format</td>
<td>20</td>
</tr>
<tr>
<td>d. General form for output statements for Fortran with Format</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>II.5 Specification statements</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. DIMENSION statement</td>
<td>21</td>
</tr>
<tr>
<td>b. Other specification statements</td>
<td>21</td>
</tr>
</tbody>
</table>

| III NCE LOAD AND GO | 22 |

| III.1 Punching input source programs | 23 |
| III.2 Arithmetic statements | 23 |
| a. Constants | 23 |
| b. Variables | 23 |
c. Arithmetic expressions .................................................. 24

III.3 Control statements .................................................... 25
  a. Statement numbers ................................................... 25
  b. Computed GO TO statement ......................................... 25
  c. IF (SENSE SWITCH) statement ..................................... 25
  d. DO statement ................................................................ 26
  e. STOP and END statements ............................................ 27

III.4 Input-Output statements ................................................ 27
  a. General rules for input-output statements without format .... 27
  b. Rules for input data ................................................... 28
  c. Rules for output data .................................................. 29

III.5 Subprograms ................................................................. 29
  a. Library subroutines ................................................... 29
  b. Subprograms written by the programmer ......................... 30

III.6 Batch processing instructions - Version Three .................. 31
  a. Preparing the source deck for batch processing ................. 31
  b. Operator instructions batch processing .......................... 31
  c. Punched card output-batch processing ............................ 32

III.7 User processing - Version Two ...................................... 33
  a. Preparing the source deck for user processing ................. 33
  b. Operator instructions - user processing ......................... 34
  c. Trace feature - user processing .................................... 36

III.8 Error messages during compilation ................................ 37
  TABLE 1 ........................................................................ 37

III.9 Error messages during execution ................................... 38
  Table 2 ........................................................................ 38

III.10 Sample problems .......................................................... 39
Sample problem 6

IV. KINGSTON FORTRAN LANGUAGE SPECIFICATIONS

IV.1 Punching input source program

IV.2 The arithmetic statement
   a. Constants
   b. Variables
   c. Arithmetic expression

IV.3 Control statements
   a. Statement numbers
   b. Address variables
   c. Computed GO TO statement
   d. IF (SENSE SWITCH) statement
   e. DO statement
   f. PAUSE statement
   g. STOP statement, CALL EXIT, CALL SKIP
   h. END statement

IV.4 INPUT/OUTPUT statements
   a. Input statements
   b. Array input
   c. Output statements
   d. Array output
   e. Maximum size of output records

IV.5 The Format statement
   a. Numeric conversion codes
   b. Alphabetic conversion codes
c. Specifying blank fields ........................................ 63

d. Repeating specifications ......................................... 63

e. The use of the slash (/) ............................................ 64
   TABLE 3 .......................................................... 65

IV.6 KFI1 without FORMAT ........................................... 65

IV.7 Specification statements ......................................... 65
   a. COMMON statement ............................................... 66
   b. EQUIVALENCE statement ......................................... 66
   c. Type statements ................................................ 67
   d. DATA statements ................................................ 67

IV.8 Subprograms ..................................................... 69
   a. Arithmetic statement function .................................. 70
   b. FUNCTION subprogram ........................................... 71
   c. SUBROUTINE subprogram ......................................... 74

IV.9 Subprograms provided by FORTRAN ............................ 76
   a. Mathematical subroutines ....................................... 76
      TABLE 4 ....................................................... 76

IV.10 Operating instructions, control cards ....................... 80
   a. Required control cards ......................................... 80
   b. Optional control cards .......................................... 81

IV.11 Operating instructions, automatic console typewriter
      output during program compilation .............................. 83

IV.12 Operating instructions, error messages during compilation
      TABLE 5 ........................................................ 85

IV.13 Operating instructions, error messages during object
      program execution ............................................... 84
      TABLE 6 ........................................................ 93
      TABLE 7 ........................................................ 94
V

Sample problems

Sample problem 7
Sample problem 8
Sample problem 9
Sample problem 10
Sample problem 11
Sample problem 12

IBM FORTRAN II

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

V.2 Library functions added to the IBM FII compiler

V.3 Instructions which use the 1311 disk for storage of data
   a. DEFINE DISK statement
   b. Assigning numbers to disk sectors
   c. RECORD statement
   d. FETCH statement
   e. FIND statement

V.4 Operating instructions, Monitor Control cards

V.5 Sample problems
   Sample problem 13
   Sample problem 14

VI

OTHER 1620 PROGRAMS AVAILABLE

VI.1 AFIT Fortran

VI.2 SPS (Symbolic Programming System)
VI.3 Programs written by the NCE Computing Center staff and stored on the 1311 disk
   a. Butler
   b. Equivalence table description
   c. Programs written by the NCE Computing Center staff and stored on punched cards

VI.4 Library of 1620 programs
   a. Programs stored on the 1311 disk
   b. How to clear memory (Model I)
   c. Programs stored on punched cards or tape

VII PUNCHED CARD EQUIPMENT
   VII.1 Card punch (Model 026)
   VII.2 Printer (IBM Accounting Machine)
   VII.3 Reproducer (Model 519)
   VII.4 Sorter (Model 082)
   VII.5 Character coding on cards

APPENDIX
   A. Notes on the storage of integer and real numbers
   B. Notes on the use of mark sense cards
   C. Notes on Model II-batch processing

INDEX TO ACCEPTABLE FORTRAN STATEMENTS
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 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.

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)

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. CO-PUTING CENTER**

**SCHEDULE OF CHARGES**

*(Effective June 1966)*

<table>
<thead>
<tr>
<th>1620 Model I with 20 K Memory</th>
<th>Per Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial or Industrial Users</td>
<td>$30.00</td>
</tr>
<tr>
<td>Sponsored Research</td>
<td>15.00</td>
</tr>
<tr>
<td>Other Educational Institutions</td>
<td>15.00/Negotiated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1620 Model II with 40 K Memory, two Disk Drives (2 million digits each) and 1443 On-Line Printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial or Industrial Users</td>
</tr>
<tr>
<td>Sponsored Research</td>
</tr>
<tr>
<td>Other Educational Institutions</td>
</tr>
</tbody>
</table>
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

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

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

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 ________________________________

NC: USERS

Staff Member __________________________ Department __________________________ Telephone Ext. ______

Type of Use: □ Class Use Number of Students ______

Time Blocks ________________________________

□ Unsponsored Research

□ Sponsored Research Account Number ______

□ Other ________________________________

Student __________________________ 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.

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

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

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

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

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

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.1 Source program and object program

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, parenthesis and functions.

   Examples:
   \[
   A = 4. - B + 6. * C * (D + E) \\
   \text{ROOT} = (-B + (B*B - 4.*A*C)**.5)/(2.*LOG(A))
   \]

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.

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

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.2a Constants

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:

\[ I = 678 \quad (678 \text{ is a valid integer constant}) \]
\[ K = I - 23 \quad (23 \text{ is a valid integer constant}) \]

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:

<table>
<thead>
<tr>
<th>Valid real constant</th>
<th>Invalid real constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00999999</td>
<td>234</td>
</tr>
</tbody>
</table>

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:**

\[ A = 4.2 + B \times (5.E2-D) \quad (5.E2 \text{ is a valid real constant}) \]

500.0 may be written:  
500.0  
5.E2  
5.0E2  
5.OEO2  
5.0E+02  
50.0EO1

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

\[ 1.0 \times 10^{-51} \leq \text{constant} \leq 9.9999999 \times 10^{+48} \]

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:**

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

**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:

\[ \text{INT} = \text{LEMMA} + \text{NUM} \quad \text{(These variables represent integers)} \]
\[ X = Y - \text{ALPHA} \quad \text{(These variables represent real numbers)} \]

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 \( \text{A} \) be an array consisting of the quantities, 700.4, 3.4, 532.99, then

\[ A(1) = 700.4 \]
\[ A(2) = 3.4 \]
\[ A(3) = 532.99 \]

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 \( \text{M} \) may be a 2 by 3 table of integers with the following:

\[ M(1,1) = 31 \]
\[ M(1,2) = 6 \]
\[ M(1,3) = 17 \]
\[ M(2,1) = -4 \]
\[ M(2,2) = 89 \]
\[ M(2,3) = -11 \]
II.2c Arithmetic operation symbols

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

II.2d Arithmetic expressions

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:

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

a) \( \frac{A}{B/C} \) is always compiled as \( \frac{(A/B)}{C} \)

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

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 \( E-(BC) \)

\( E = -A**B, \) the order of compilation depends upon the compiler.

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

II.2e Replacement statements

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 \( I \).

\[ X = \text{NUM} + L - 1 \]
Evaluate \( \text{NUM}+L-1 \).
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.

(See IBM FII: V.2)

II.3 Control statements

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

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

General form:
GO TO (x1, x2, x3, ..., xn), i

Where: x1, x2, x3, ..., xn 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 x1, x2, or xn depending on whether the current value of i is 1, 2, or n.
Example:  

GO TO (10, 40, 50), I  

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

II.3c Arithmetic IF statement

General form:

IF (a) n₁, n₂, n₃

Where: a is an arithmetic expression and n₁, n₂, n₃, are statement numbers.

This statement causes control to be transferred to statement n₁, n₂, or n₃, if the value of the expression (a) is less than, equal to, or greater than zero, respectively.

II.3d IF (SENSE SWITCH i) n₁, n₂

Where: i is an integer constant or integer expression depending on which compiler is used and n₁, n₂, are statement numbers.

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

This statement causes control to be transferred to the statement n₁ if the sense switch is on, or to n₂ 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

General form:  

End of Range  index  initial  test  increment  value  value

DO  

n i = m₁, m₂, m₃

The form (integer, integer expression) that the index (i), initial value (m₁), test value (m₂), and increment (m₃), 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 m₁, and each succeeding time i is incremented by the value of m₃. After the statements have been executed with i equal to the highest value that does not exceed m₂ in the direction of incrementation, control passes to the statement following statement number n. This is called a normal exit from the DO statement.

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 m₃ 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 KFI II 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):

\begin{verbatim}
[r] [c] [r]
[l]
[l]
\end{verbatim}

The following configuration is not permitted:

\begin{verbatim}
[r] [c] [r]
[l]
[l]
\end{verbatim}

(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**

**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**

**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**

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

READ, list  
( )

ACCEPT, list  
( )

ACCEPT TAPE, list  
( )

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 General form for input statements for Fortran with Format

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 General form for output statements for Fortran without Format

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.

Example:

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 General form for output statement for Fortran with Format

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 Specification statements
II.5a **DIMENSION statement**

General form:

```
DIMENSION VA(i1), VB(i2)
```

Where: VA, VB, are variable names and i1, i2, 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**

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.1 Punching input source programs

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.2a Constants

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

Example:  

<table>
<thead>
<tr>
<th>Valid constants</th>
<th>Invalid constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999</td>
<td>10991</td>
</tr>
<tr>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>-356</td>
<td>-356.0</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

III.2b Variables

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:

\[
\begin{align*}
&i \\
&i + c' \\
&i - c' \\
&c
\end{align*}
\]

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:

<table>
<thead>
<tr>
<th>Valid Subscripting</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A(\text{IMAX}-37))</td>
</tr>
<tr>
<td>(I(J,\text{MATH}+17))</td>
</tr>
<tr>
<td>(Q(124, L4-3))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Invalid Subscripting</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A(\text{IMAX}-50))</td>
</tr>
<tr>
<td>(B(X,\text{IMAX}))</td>
</tr>
<tr>
<td>(C(\text{MATH}+51,J))</td>
</tr>
</tbody>
</table>

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: \(A=\text{-B**I}\) will be compiled as \(A=(\text{-B})^I\) and calculated as the product \((\text{-B})(\text{-B})(\text{-B})\) .........

Example: When \(B=2, I=3\)

\[A = (-2)^3\] or \((-2)(-2)(-2), A = -8\]

\[A = -B**C\] will be compiled as \(A=\text{EXP}(C*\text{LOG}(-B))\)

When \((-B)\) is negative ERROR 62 will result. (See error messages, III.9)

When \((-B)\) is negative the value of \(A\) is calculated as \(A= \text{EXP}(C*\text{LOG}(\text{ABS}(-B)))\) during user processing.

Example: When \(B=2, C=3\)

\[A=\text{EXP}(3*\text{LOG}(\text{ABS}(-2))), A=8\]

ERROR 62 will be indicated

When \(B= -2, C=3\)

\[A = \text{EXP}(3*\text{LOG}((-2))), A = 8\]

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.3a Statement numbers

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

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

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

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

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

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

1. To obtain a line or a card of alphabetic 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 -32500

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

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 (0.1) 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.5a Library subroutines

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:

\[ M = J + \text{ABS}(K \times L) \]

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Operation</th>
<th>*Symbolic Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural Logarithm</td>
<td>( \log A )</td>
<td>LOG</td>
</tr>
<tr>
<td>Exponential</td>
<td>( e^x )</td>
<td>EXP</td>
</tr>
<tr>
<td>Square root</td>
<td>( \sqrt{A} )</td>
<td>SQRT</td>
</tr>
<tr>
<td>Sine</td>
<td>( \sin A )</td>
<td>SIN</td>
</tr>
<tr>
<td>Cosine</td>
<td>( \cos A )</td>
<td>COS</td>
</tr>
<tr>
<td>Arc tangent</td>
<td>( \tan^{-1} A )</td>
<td>ATAN</td>
</tr>
<tr>
<td>Absolute value</td>
<td>(</td>
<td>A</td>
</tr>
</tbody>
</table>

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. (LOGF, 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 proceeds. (See Section III.8 errors 61, 62, 65). The values used for the functions in these cases are the following:

- \( 0.0 = X \) or \( 0.0 = \frac{X}{0.0} = 0.0 \) \( \rightarrow \) \( 9.9999999 \times 10^{48} \)
- \( 0.0 \times \times X \) \( \rightarrow \) \( 0.0 \)
- \( \text{LOG}(0.0) \) \( \rightarrow \) \( 0.0 \)
- \( \text{LOG}(X \text{ where } X < 0) \) \( \rightarrow \) \( \text{LOG}(\|X\|) \)
- \( \text{SQRT}(X \text{ where } X < 0) \) \( \rightarrow \) \( \text{SQRT}(\|X\|) \)
- \( \text{SIN}(X \text{ where } X \geq 1.0 \text{ radians}) \) \( \rightarrow \) \( 1.0 \)
- \( \text{COS}(X \text{ where } X \geq 1.0 \text{ radians}) \) \( \rightarrow \) \( 1.0 \)

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

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.6a Preparing the source deck for batch processing

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) J1, J2

3. Either the END or the STOP statement should be the last executable state­
ments 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

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. If the processor is in memory:

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

Press READER START

PUNCH START

(red bottom on card reader)

(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 Punched card output - batch processing

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 subsequent 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 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

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 type-
writer.

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**

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.
### TABLE 1

**Compilation Error Codes for NCE Load Go, (Revised)**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Unrecognizable statement, incorrect spelling, etc.</td>
</tr>
<tr>
<td>12</td>
<td>Mixed mode, or missing right-parenthesis in if statement</td>
</tr>
<tr>
<td>13</td>
<td>Misplaced or missing comma or equal sign</td>
</tr>
<tr>
<td>14</td>
<td>Misplaced or missing parenthesis</td>
</tr>
<tr>
<td>15</td>
<td>Misplaced or missing variable or operand, or illegal special character sequence</td>
</tr>
<tr>
<td>20</td>
<td>Statement number used more than once</td>
</tr>
<tr>
<td>21</td>
<td>Undefined statement number called</td>
</tr>
<tr>
<td>22</td>
<td>Wrong number of parameters on a transfer statement, or invalid index on computed go to statement</td>
</tr>
<tr>
<td>23</td>
<td>Constant in unacceptable position, or statement number is zero</td>
</tr>
<tr>
<td>24</td>
<td>Missing or unacceptable statement number or array size, or constant is in unacceptable form</td>
</tr>
<tr>
<td>30</td>
<td>Dimensioned variable used without subscript, or subscripted variable has not been dimensioned, or incorrect function name, or trig functions were eliminated</td>
</tr>
<tr>
<td>31</td>
<td>Subscript in incorrect form or missing operator</td>
</tr>
<tr>
<td>32</td>
<td>Function name used in non-arithmetic statement</td>
</tr>
<tr>
<td>40</td>
<td>Do statement is incorrectly formed</td>
</tr>
<tr>
<td>41</td>
<td>Do loops incorrectly nested, or error in last statement of loop</td>
</tr>
<tr>
<td>42</td>
<td>Do loop ends with an if, go to, computed go to, do, stop, or end statement</td>
</tr>
<tr>
<td>43</td>
<td>More than seven levels of do loop nesting encountered</td>
</tr>
<tr>
<td>50</td>
<td>Variable name consists of more than five characters</td>
</tr>
<tr>
<td>51</td>
<td>Raising fixed point quantity to a power</td>
</tr>
<tr>
<td>52</td>
<td>A<strong>B</strong>C, parentheses must be added to indicate order</td>
</tr>
<tr>
<td>53</td>
<td>Source program is too large to compile</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>ERROR 60</th>
<th>DIVISION BY ZERO</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td><strong>9.9999999E+48</strong>, OR <strong>9999</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERROR 61</th>
<th>LOG(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td><strong>0.0000000</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERROR 62</th>
<th>A<strong>B</strong>, LOG(A), OR SQRT(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>FUNCTION OF ABS(A)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERROR 63</th>
<th>CALCULATED EXPONENT GREATER THAN +49</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td><strong>9.9999999E+48</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERROR 64</th>
<th>CALCULATED EXPONENT LESS THAN -50</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td><strong>0.0000000</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ERROR 65</th>
<th>SIN(A), COS(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULT</td>
<td>WITH A GREATER THAN 1E9 RADIANS</td>
</tr>
</tbody>
</table>

* 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
1F (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
3
4.5
-3.2
2156
6
15.009 9456 -36.5 999.99 +15
9999
## Sample Problem 3

### Execution

**Output**

<table>
<thead>
<tr>
<th>ROW</th>
<th>COL</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0001</td>
<td>1.000000</td>
</tr>
<tr>
<td>0001</td>
<td>0002</td>
<td>2.000000</td>
</tr>
<tr>
<td>0001</td>
<td>0003</td>
<td>3.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0001</td>
<td>4.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0002</td>
<td>5.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0003</td>
<td>6.000000</td>
</tr>
<tr>
<td>0003</td>
<td>0001</td>
<td>7.000000</td>
</tr>
<tr>
<td>0003</td>
<td>0002</td>
<td>8.000000</td>
</tr>
<tr>
<td>0003</td>
<td>0003</td>
<td>9.000000</td>
</tr>
<tr>
<td>0004</td>
<td>0001</td>
<td>10.000000</td>
</tr>
<tr>
<td>0004</td>
<td>0002</td>
<td>11.000000</td>
</tr>
<tr>
<td>0004</td>
<td>0003</td>
<td>12.000000</td>
</tr>
<tr>
<td>0001</td>
<td>0002</td>
<td>13.000000</td>
</tr>
<tr>
<td>0001</td>
<td>0003</td>
<td>14.000000</td>
</tr>
<tr>
<td>0001</td>
<td>0004</td>
<td>15.000000</td>
</tr>
<tr>
<td>0001</td>
<td>0005</td>
<td>16.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0001</td>
<td>17.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0002</td>
<td>18.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0003</td>
<td>19.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0004</td>
<td>20.000000</td>
</tr>
<tr>
<td>0002</td>
<td>0005</td>
<td>21.000000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
<td>-83.22678</td>
</tr>
<tr>
<td>0001</td>
<td>0002</td>
<td>11.500000</td>
</tr>
<tr>
<td>0002</td>
<td>0001</td>
<td>-6.8000000E-03</td>
</tr>
<tr>
<td>0002</td>
<td>0002</td>
<td>5.4000000</td>
</tr>
</tbody>
</table>

## Sample Problem 4

### Execution

**Output**

<table>
<thead>
<tr>
<th>GROUP NUMBER</th>
<th>GROUP SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>40.000000</td>
</tr>
<tr>
<td>0001</td>
<td>70.000000</td>
</tr>
<tr>
<td>0001</td>
<td>94.000000</td>
</tr>
<tr>
<td>0001</td>
<td>119.000000</td>
</tr>
<tr>
<td>0002</td>
<td>26.000000</td>
</tr>
<tr>
<td>0002</td>
<td>53.000000</td>
</tr>
<tr>
<td>0002</td>
<td>93.000000</td>
</tr>
<tr>
<td>0002</td>
<td>130.000000</td>
</tr>
<tr>
<td>0003</td>
<td>25.000000</td>
</tr>
<tr>
<td>0003</td>
<td>48.000000</td>
</tr>
<tr>
<td>0003</td>
<td>70.000000</td>
</tr>
<tr>
<td>0003</td>
<td>120.000000</td>
</tr>
<tr>
<td>0002</td>
<td>130.000000</td>
</tr>
</tbody>
</table>
THE FOLLOWING SOURCE DECKS ARE PREPARED FOR USER PROCESSING AND WERE PROCESSED USING VERSION TWO OF THE PROCESSOR.

SAMPLE PROBLEM 5

PROGRAM CALCULATES THE FORMULA \( C(N,R) = \frac{N}{R} C(N-R) \)

FOR GIVEN VALUES OF \( N,R \)

\( N \) IS ANY INTEGER GREATER THAN 1 AND LESS THAN 50

\( R \) IS ANY INTEGER LESS THAN \( N \)

VALUES OF \( N,R \) PUNCHED ON SAME CARD

LAST DATA CARD PUNCHED 50 IN COL. 1:2

READ XN,R

CHECKS TO SEE THAT THE DATA IS WITHIN THE ALLOWABLE RANGE

IF \( (XN-50.0) > 99.99 \) 10 IF \( (XN-1.0) < 99.99 \) 05 XNR=XN-R

IF \( (XNR) > 99.99 \) 11 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

16 RF=FAC

L=XNR

GO TO 20

17 XNRF=FAC

CRN=XNF/(RF*XNRF)

97 PUNCH XN,R,CRN

GO TO 01

SUB-RPGM COMPUTES THE FACTORIALS

30 FAC=1.

DO 50 J=1,L

Y=J

50 FAC=FAC*Y

GO TO (15,16,17,1)

99 STOP

98 END

5 3

3 1

19 10

50 32

5.0000000 3.00000000 10.0000000

3.0000000 1.00000000 3.00000000

19.0000000 10.0000000 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
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
C SWITCH 2 ON, STOP ON TOLERANCE CHECK OFF, CONTINUE
C ENTER TOLERANCE AND SIZE OF MATRIX AS FIRST PIECE OF DATA
C 17 EQUATIONS AND TRIG FUNCTIONS NEED NOT BE ELIMINATED.
C WITH THIS SIZE DIMENSION, THE PROGRAM WILL HANDLE
DIMENSION A(17,18)
PAUSE
10 READ *, TOLR, N2
   N1=N2+1
   DO 2 I=1,N2
       DO 2 J=I,N1
2      READ *, A(I,J)
2      END
      DO 14 I=1,N2
14     A(I,A(I,J))/DIAG
      IF (DIAG) 4., 20, 4
5      IF (ABS(DIAG)-TOLR) 19., 19, 5
6      DO 6 J=I,N1
6      A(I,J)=A(I,J)/DIAG
5      K=1
9      IF (K-I) 11., 13, 11
11     FCTR=A(K,J)
      DO 12 J=I,N1
12     A(K,J)=A(K,J)-FCTR*A(I,J)
13     K=K+1
14     CONTINUE
      J=N1
      IF (SENSE SWITCH 1) 15., 17
15     DO 16 I=1,N2
16     PUNCH *, A(I,J)
      GO TO 10
17     DO 18 I=1,N2
18     PRINT *, A(I,J)
      GO TO 10
19     STATEMENT 19 IS TOLERANCE STOP
19     PAUSE
       IF (SENSE SWITCH 2) 10., 5
20     STATEMENT 20 IS ERROR STOP
20     STOP
      END
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, D.A. Jardine, E.S. Lee, J.A.N. Lee, and D.G. Robinson. 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 PLOTIP 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.
3. Computing Centre, Queen's University, Kingston, Ontario.
IV.1 Punching input source programs

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.2a Constants

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:

<table>
<thead>
<tr>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1HA</td>
<td>6HABcdef</td>
</tr>
<tr>
<td>4H(*/-</td>
<td>OH</td>
</tr>
<tr>
<td>3HbbC</td>
<td>4HbbbbD</td>
</tr>
</tbody>
</table>

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:

\[
I = \text{3HYXA}
\]
\[
A = \text{4H1234}
\]

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.2a)

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 = 5Hbbbb + 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.2d)

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 alphanemic (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 override 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- KF11 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, i*j*(K(2)-L+M),K(N(M))
 (j*5+M)                                      ...
 (5*j)                                        ...
 (6*j-K+2-10/L+M)                              ...
 (I+J*(K+2-L+M)+K(M(N+2))/3)                   ...
 (* (FIXF(A*B*2.0*0)*)L/2)                    ...
```

Invalid subscripted variables

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

*(See library function FIXF, IV.9a)*
IV.2c Arithmetic expression

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

1. \( E = -A \times B \) 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 \( A \times B / C \times D \) will be calculated as: \( A \times (B/C) \times D \).

IV.3 Control statements

IV.3a Statement numbers

Statement numbers may be any integer \( n \), \( 0 \leq n \leq 99999 \).

IV.3b Address variables

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:

\[
\text{ASSIGN } i \text{ to } n
\]

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

\[
\begin{align*}
\text{ASSIGN } 12 & \text{ to } K \\
\text{ASSIGN } (K) & \text{ to } J(L) \\
\text{ASSIGN } 13 & \text{ to } A(M(N)) \\
\text{ASSIGN } (A(M(N))) & \text{ to } K
\end{align*}
\]
Rules for using address variables

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNCONDITIONAL GO TO</td>
<td>GO TO K</td>
</tr>
<tr>
<td>COMPUTED GO TO</td>
<td>GO TO (10,K, 30, L(M), 15, 35), ITEM(J)</td>
</tr>
<tr>
<td>ARITHMETIC IF</td>
<td>IF(A(J,K)-B) &gt; 10, L</td>
</tr>
<tr>
<td>IF SENSE SWITCH</td>
<td>IF(SENS SWITCH 1) N,A</td>
</tr>
</tbody>
</table>

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.6c)

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 (x₁, x₂, x₃, ....xₙ), i

where: x₁, x₂, x₃, ....xₙ 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

IF (SENSE SWITCH i) n₁, n₂

where: i is an integer constant or arithmetic expression
n₁, n₂ 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 n₁ 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

<table>
<thead>
<tr>
<th>End of Range</th>
<th>Index i</th>
<th>Initial m₁, m₂, m₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO n</td>
<td>i =</td>
<td>m₁, m₂, m₃</td>
</tr>
</tbody>
</table>

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

i is a subscripted or nonsubscripted integer variable.
m₁, m₂, m₃ are signed or unsigned integer constants, subscripted or
non-subscripted or integer expressions of any desired complexity.
m₃ 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, m₁, m₂, m₃),
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 in-
crementing will occur until an exit from the range of the DO takes place.

3. The initial value, m₁, and test value, m₂, may be positive, negative,
or zero. The normal algebraic sign convention is applied for increment-
ing and testing.

4. The increment, m₃, may be positive or negative, but not zero.
Example: 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

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

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.

**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**

```
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.4a Input statements**

- `READ n, list`
- `ACCEPT TAPE n, list`
- `ACCEPT n, list`
- `REREAD n, list`

Cards

Paper tape

Console typewriter

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

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 \( C(1) \ldots C(10) \) to be read into storage.

   Given \( D \) is a \( 3 \times 5 \times 5 \) array

   **Example:**
   
   ```
   Dimension D(3,5,5)
   Read 40,D
   ```

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

   \( D_{111}, D_{211}, D_{311}, D_{121}, D_{221}, D_{321}, D_{131}, \ldots \)

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:

```plaintext
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:

```plaintext
((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:

```plaintext
READ 10, ( (A(I,J), I = K,L), J = M,N)
```

```plaintext
READ 10, ( (A(I(KL), J(ML)), KL = K-JOB*2, L+5,-J6), ML = 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:

```plaintext
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**

- **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. / represents the beginning of a new record.

IV.5a Numeric conversion codes

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:

<table>
<thead>
<tr>
<th>Internal value</th>
<th>Punched</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>721</td>
</tr>
<tr>
<td>-721</td>
<td>error message</td>
</tr>
<tr>
<td>-12</td>
<td>-12</td>
</tr>
<tr>
<td>68114</td>
<td>error message</td>
</tr>
<tr>
<td>4336.15</td>
<td>error message</td>
</tr>
<tr>
<td>-43.72</td>
<td>-43</td>
</tr>
</tbody>
</table>

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 EL0.3 punches the internal values as follows:

<table>
<thead>
<tr>
<th>Internal Value</th>
<th>Punched</th>
</tr>
</thead>
<tbody>
<tr>
<td>238.</td>
<td>bb.238E+03</td>
</tr>
<tr>
<td>-.002</td>
<td>b-.200E-02</td>
</tr>
<tr>
<td>.0000000004</td>
<td>bbb.400E-9</td>
</tr>
<tr>
<td>-21.0057</td>
<td>b-2100E+02</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Card Cols.</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>462</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
</tr>
<tr>
<td>5-6</td>
<td>b2</td>
</tr>
<tr>
<td>7</td>
<td>b</td>
</tr>
<tr>
<td>8-11</td>
<td>-398</td>
</tr>
<tr>
<td>12</td>
<td>b</td>
</tr>
<tr>
<td>13-20</td>
<td>539.3218</td>
</tr>
</tbody>
</table>

The numbers are read as follows:

\[ X = 462 \]
\[ I = 2 \]
\[ N = -398 \]
\[ Y = 539.3218 \]

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

and 1E6,1X for integer numbers.

(See "Scale factors" below)

Example:

```
PUNCH 10, Y, I
10 FORMAT (2N) will produce
the following output:
5.63000000E+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 `2FF7.4`. 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-0.008bbbb0.554
```

but the statement `FORMAT (I2,1P3F11.3)`, used with the same data, would give the following line:
Whereas, the statement \texttt{FORMAT (I2, -1P3Fl1, 3)} would give the following line:

\begin{verbatim}
27b932.09b0bbbbb-0.076bbbbb5.536
\end{verbatim}

A positive scale factor used for output with E-conversion increases the number and decreases the exponent. Thus, with the same data, \texttt{FORMAT (I2, 1P3El2, 4)} would produce the following line:

\begin{verbatim}
27b-9.3209E01b-7.5804E-03bb5.5536E-01
\end{verbatim}

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.

\section*{Alphameric conversion codes}

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).

\subsection*{H-type FORMAT specification}

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

\begin{verbatim}
25 FORMAT (15HbTHISbISbH-TYPE)
\end{verbatim}

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:

\begin{verbatim}
5 FORMAT (27HbTHISbISbISbALPHAMERICbDATA) 
5 PRINT 5
\end{verbatim}
would cause the following record to be written on the printer:

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 (3HX=,F9.3,A4/)
•
•
•
PUNCH 15,A,SAM,B,SAM
```
would produce the following lines:

\[
XY = 5976.214\text{ABCD} \\
XY = 6173.928\text{ABCD}
\]

Characters transmitted under A-throw 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-throw.

**IV.5c Specifying blank fields**

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

**General Form of X-throw code:**

\[wX\]

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

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

**Example:**

Card is punched:

<table>
<thead>
<tr>
<th>Col. 1-5</th>
<th>543.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-8</td>
<td>423</td>
</tr>
<tr>
<td>9-12</td>
<td>3233</td>
</tr>
</tbody>
</table>

Read 5, A, I

5 FORMAT (F5.1, 3XI4)

will result in:

\[A=543.2\]

\[I=3233\]

**IV.5d Repeating specifications**

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:

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 statement. 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)

cause the data to be transmitted in the following order:

<table>
<thead>
<tr>
<th>Data Transmitted</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>A)</td>
<td>F10.3</td>
</tr>
<tr>
<td>B)</td>
<td>E12.4</td>
</tr>
<tr>
<td>C)</td>
<td>F12.2</td>
</tr>
<tr>
<td>D)</td>
<td>F10.3</td>
</tr>
<tr>
<td>E)</td>
<td>E12.4</td>
</tr>
<tr>
<td>F)</td>
<td>F12.2</td>
</tr>
<tr>
<td>G)</td>
<td>F10.3</td>
</tr>
</tbody>
</table>

The use of the slash (/)

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 Printer carriage control

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.

Example:

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.

Example:

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 KFII without FORMAT

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 Specification statements

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

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

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.

Example:

\[
\text{DIMENSION B(5), C(10,10), D(5, 10, 15)}
\]

\[
\text{EQUIVALENCE (A, B(1), C(5,5)), (D(1, 2, 5), E)}
\]

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

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

IV.7d DATA statements

General form:

\[
\text{DATA V1/C1/, V2/C2/, \ldots../Vn/Cn/}
\]

Where: \(V_i\) is a variable name, an element of an array, or an array name.

\(C_i\) 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:

\[
\begin{align*}
\text{DIMENSION} & \quad X(10,5),B(2,2),I3,\text{TRANS}(10)  \\
\text{DATA} & \quad R/3.0/,\text{HOLL}/5\text{ABCDE}/,\text{JOB}/-1/  \\
\text{DATA} & \quad \text{NUM}/23/  \\
\text{DATA} & \quad F/-3.6/,\text{PIE}/.31415927E+1/  \\
\text{DATA} & \quad \text{TRANS}/3S,4S,4S,3*7S,4*3S/  \\
\end{align*}
\]

\[\text{variable} \quad \text{is initialized with:}\]

\[
\begin{align*}
R & \quad 3.0  \\
\text{HOLL} & \quad \text{ABCDE}  \\
\text{JOB} & \quad -1  \\
\text{NUM} & \quad \text{the object time representation of statement No. 23}  \\
\text{F} & \quad -3.6  \\
\text{PIE} & \quad 3.1415927  \\
\text{TRANS}(1) & \quad \text{statement no. 3 (obj. time representation)}  \\
\text{TRANS}(2),\text{TRANS}(3) & \quad " \quad 4 \quad " \quad " \quad "  \\
\text{TRANS}(4),(5),(6) & \quad " \quad 7 \quad " \quad " \quad "  \\
\text{TRANS}(7),(8),(9),(10) & \quad " \quad 3 \quad " \quad " \quad "
\end{align*}
\]

The specification statements must be the first statements of the program. The order of these must be (excluding comments cards) as follows:
COMON if any
DIMENSION if any
EQUIVALENCE if any
REAL if any
INTEGER if any

REAL and INTEGER are considered equivalent, and may be inter-
changed in the above list.

The DATA statement may appear anywhere after the above list.

IV.8 Subprograms

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 state-
ments 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 defin-
itions; 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**

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:**

```plaintext
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 alphabetic 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:**

<table>
<thead>
<tr>
<th>Definition</th>
<th>AVG (A,B,C,D) = (A+B+C+D)/4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calling</td>
<td>AAGE = X + AVG(E,F,G,H)</td>
</tr>
</tbody>
</table>
The calling statement is evaluated by first substituting the argument values in the Arithmetic Statement Function definition:

\[ A = E \\
B = F \\
C = G \\
D = H \]

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

\[
\begin{align*}
\text{SUM} (A,B,C,D) &= A+B+C+D \\
\text{FUNC} (A) &= A+X*Y*Z(J)
\end{align*}
\]

Invalid Arithmetic Statement Function Definitions

\[
\begin{align*}
\text{SUBPRG} (3,J,K) &= 3*I+J**2 \\
\text{SOMEF} (A(I),B) &= A(I)/B+3 \\
\text{SUBPRFM}(A,B) &= A**2+B**2 \\
3 \text{ FUNC} (D) &= 3.14*D \\
\text{IDEN} (X,Y,Z) &= X/Y + Y/Z \text{ (valid if a real specification statement is included in the program: REAL IDEN)}
\end{align*}
\]

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

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:

\[
\begin{align*}
\text{FUNCTION} \text{ name} (a_1, a_2, a_3, \ldots a_n) \\
&: \\
&: \\
\text{RETURN} \end{align*}
\]

END
name is the name of the FORTRAN function.

\[ a_1, a_2, a_3, \ldots, a_n \] 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. The 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:

<table>
<thead>
<tr>
<th>Program statements</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL NUT</td>
<td>Calling statement, transfers</td>
</tr>
<tr>
<td>X = NUT(A)</td>
<td>control to function subprogram NUT.</td>
</tr>
</tbody>
</table>
REAL FUNCTION NUT(C)  

Before evaluating NUT, C is set equal to current value of A.

D = 18.2
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:

<table>
<thead>
<tr>
<th>Program statements</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>N = MAX (I,J,K,L)</td>
<td>Calling statement arguments are I,J,K,L, Control transferred to MAX.</td>
</tr>
</tbody>
</table>

Subprogram

FUNCTION MAX (M,L,MM,NN)  
!  
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:

<table>
<thead>
<tr>
<th>Program statements</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = ROOTSL + CALC(Y,X,I)</td>
<td>Calling statement transfers control to CALC</td>
</tr>
</tbody>
</table>

Sub-program
FUNCTION CALC (A,B,J)

CALC = A/B+B**J

If (CALC) 10,20,10

10 RETURN

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 alphabetic 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)

10 A(K,M)=A(K,M)+B(K,M)
RETURN
END
```

**Example:**

```
ASSIGN 173 to J                      main program
CALL BOMB(J)

SUBROUTINE BOMB (ZIP)               Subroutine. The GO TO
                                     statement 173 of the
                                     Main Program

GO TO ZIP
```

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 program
CALL BOMB(J)

SUBROUTINE BOMB (ZIP)               Subroutine. The GO TO
                                     statement 173 of the
                                     Main Program

GO TO ZIP
6. The SUBROUTINE Subprogram must follow the main program.

IV.9 Subprograms provided by FORTRAN

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

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

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>DEFINITION</th>
<th>NO. OF ARGS.</th>
<th>NAME OR TYPE OF NAME</th>
<th>ARGUMENT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>$e^{\text{Arg}}$</td>
<td>1</td>
<td>EXP</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Natural logarithm</td>
<td>$\log_{e}(\text{Arg})$</td>
<td>1</td>
<td>LOG</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LOGF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALOG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arctangent</td>
<td>$\text{arctan}($Arg$)$</td>
<td>1</td>
<td>ATAN</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>in range $-\frac{\pi}{2}$ to $\frac{\pi}{2}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arctangent</td>
<td>$\text{arctan}(\text{Arg}_1/\text{Arg}_2)$</td>
<td>2</td>
<td>ATAN</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>in range $-\pi$ to $\pi$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trig. Sine</td>
<td>$\sin($Arg$)$</td>
<td>1</td>
<td>SIN</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SINF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trig. Cosine</td>
<td>$\cos($Arg$)$</td>
<td>1</td>
<td>COS</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>COSF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Square Root</td>
<td>$(\text{Arg})^{\frac{1}{2}}$</td>
<td>1</td>
<td>SQRT</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SQRTF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Absolute value</td>
<td>$</td>
<td>\text{Arg}</td>
<td>$</td>
<td>1</td>
<td>ABS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ABSF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IABS</td>
<td>Integer</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Choosing largest value

<table>
<thead>
<tr>
<th>Max(Arg₁,Arg₂,---) ≥ 2 ≤ 9</th>
<th>MAX</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AMAX</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Choosing smallest value

<table>
<thead>
<tr>
<th>Min(Arg₁,Arg₂,---) ≥ 2 ≤ 9</th>
<th>MIN</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AMIN</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Float

<table>
<thead>
<tr>
<th>Conversion from integer to real</th>
<th>1</th>
<th>FLOAT</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FLOATF</td>
<td>Real</td>
</tr>
</tbody>
</table>

FIX

<table>
<thead>
<tr>
<th>Conversion from real to integer</th>
<th>1</th>
<th>INT</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>IFIX</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Transfer of sign

<table>
<thead>
<tr>
<th>Magnitude of Arg₁ with sign of Arg₂</th>
<th>2</th>
<th>SIGN</th>
<th>Real</th>
</tr>
</thead>
</table>

Ideal Relay Function

<table>
<thead>
<tr>
<th>Arg/ABS(Arg) for Arg ≧ 0 and 0 otherwise</th>
<th>1</th>
<th>SIGN</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>IFIX</td>
<td>Real</td>
</tr>
</tbody>
</table>

Plot

<table>
<thead>
<tr>
<th>See below (1)</th>
<th>10</th>
<th>PLOT</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PLOT P</td>
<td>arg 80</td>
</tr>
</tbody>
</table>

Rand

<table>
<thead>
<tr>
<th>See below (2)</th>
<th>1</th>
<th>RAND</th>
<th>Real</th>
</tr>
</thead>
</table>

Sort

<table>
<thead>
<tr>
<th>See below (3)</th>
<th>2</th>
<th>SORT</th>
<th>Real</th>
</tr>
</thead>
</table>

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

Example:

\[ Y = \text{SQRT} \ (A) \]

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

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

General form:

CALL PLOT \((Z₁, Z₂, \ldots Zₘ)\)

where: \(Z₁, Z₂, \ldots Zₘ\) 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 to 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 \((Z₁)\) 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 \(Z₅ = 32.59\) then a 5 is punched in Col. 32 of the Card. If two arguments have the
**General form:**

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 determinant of the N Rows by N columns of Matrix A, and is defined after the execution of SOLVE. The parameter DET may be left out of the call statement if the value of the determinant 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 statement:

CALL RESOLV (A,N,M)

where: A is the A matrix as defined in the solution 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 determinant 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)
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 section Optional control cards may involve considerable machine time. Users should exercise discretion when including them in a program.

IV.10a Required control cards

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
## JOB
## X EQ 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..........................
# JOBNAME

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
# EOJ (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:
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

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

An object deck may be obtained during compilation by punching the letter D in col. 32 and/or 33 of the ##XEQ KP2 card. If an object 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
##XEQ OBDECK

(Col. 6 must be left blank)

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
##XEQ RUNDK

(Col. 6 must be left blank)

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
(See above)

END TRACE card: # END TRACE

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 EL5.8 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 asterisk will appear before the name.
1) Simple variables
2) Dimensioned variables
3) Statement Numbers
4) Subprogram Names
5) Constants

INDEX card:  #  INDEX  (Comment columns punched as described above)
STOP INDEX card:  #  STOP INDEX

IV.11 Operating instructions, automatic printer output

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

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.
MMMMM is the number of statements after statement numbered NNNNN in which error occurred (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

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**

<table>
<thead>
<tr>
<th>A1</th>
<th>STATEMENTS CONTAINING ILLEGAL SYNTAX IN AN EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>(1) ILLEGAL SYNTAX IN AN ARITHMETIC STATEMENT</td>
</tr>
<tr>
<td></td>
<td>(2) AN EXPRESSION OR INTEGER CONSTANT ON THE LHS OF AN ARITHMETIC STATEMENT</td>
</tr>
<tr>
<td></td>
<td>(3) A SUBSCRIPTED VARIABLE NOT MENTIONED IN A DIMENSION STATEMENT</td>
</tr>
<tr>
<td>A3</td>
<td>MIXED MODE IN AN EXPRESSION</td>
</tr>
<tr>
<td>A4</td>
<td>WRONG NUMBER OF SUBSCRIPTS IN A DIMENSIONED VARIABLE</td>
</tr>
<tr>
<td>A5</td>
<td>SUBSCRIPT IS A REAL QUANTITY</td>
</tr>
<tr>
<td>A6</td>
<td>NAME OF A NON-EXTERNAL FUNCTION USED AS A VARIABLE</td>
</tr>
<tr>
<td>A7</td>
<td>THE CHARACTERS &quot;- OR $&quot; APPEAR AS OPERATOR IN AN EXPRESSION</td>
</tr>
<tr>
<td>A8</td>
<td>ONE OF THE TABLES USED BY THE COMPILER HAS OVERFLOWED (STATEMENT IS TOO LONG OR COMPLEX)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>COMMON DIMENSION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1) SYMBOL IS ALREADY IN THE SYMBOL TABLE AND IS NOT A FORMAL PARAMETER</td>
</tr>
<tr>
<td></td>
<td>(2) NO DIMENSIONS GIVEN FOR VARIABLE</td>
</tr>
<tr>
<td>C2</td>
<td>COMMON OR DIMENSION</td>
</tr>
<tr>
<td></td>
<td>ARRAY SIZE IS GREATER THAN 9999 ELEMENTS</td>
</tr>
<tr>
<td>C3</td>
<td>COMMON OR DIMENSION</td>
</tr>
<tr>
<td></td>
<td>MORE THAN 13 DIMENSIONS SPECIFIED</td>
</tr>
<tr>
<td>C4</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td>(1) INVALID CHARACTER, MOST LIKELY CAUSED BY A MISSING COMMA OR CLOSING PARENTHESIS</td>
</tr>
<tr>
<td></td>
<td>(2) CONSTANTS WHERE VARIABLE NAMES SHOULD BE REAL, INTEGER, EXTERNAL FUNCTION, SUBROUTINE, ARITHMETIC STATEMENT FUNCTIONS</td>
</tr>
</tbody>
</table>

**TABLE 5**

<table>
<thead>
<tr>
<th>KI MGSTON FORTRANII ERROR MESSAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
</tr>
<tr>
<td>A2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>A3</td>
</tr>
<tr>
<td>A4</td>
</tr>
<tr>
<td>A5</td>
</tr>
<tr>
<td>A6</td>
</tr>
<tr>
<td>A7</td>
</tr>
<tr>
<td>A8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C1</th>
<th>COMMON DIMENSION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1) SYMBOL IS ALREADY IN THE SYMBOL TABLE</td>
</tr>
<tr>
<td></td>
<td>(2) NO DIMENSIONS GIVEN FOR VARIABLE</td>
</tr>
<tr>
<td>C2</td>
<td>COMMON OR DIMENSION</td>
</tr>
<tr>
<td></td>
<td>ARRAY SIZE IS GREATER THAN 9999 ELEMENTS</td>
</tr>
<tr>
<td>C3</td>
<td>COMMON OR DIMENSION</td>
</tr>
<tr>
<td></td>
<td>MORE THAN 13 DIMENSIONS SPECIFIED</td>
</tr>
<tr>
<td>C4</td>
<td>COMMON</td>
</tr>
<tr>
<td></td>
<td>(1) INVALID CHARACTER, MOST LIKELY CAUSED BY A MISSING COMMA OR CLOSING PARENTHESIS</td>
</tr>
<tr>
<td></td>
<td>(2) CONSTANTS WHERE VARIABLE NAMES SHOULD BE REAL, INTEGER, EXTERNAL FUNCTION, SUBROUTINE, ARITHMETIC STATEMENT FUNCTIONS</td>
</tr>
</tbody>
</table>
C5 COMMON, DIMENSION, EQUIVALENCE, INTEGER, REAL, SEQUENCE
STATEMENTS ARE NOT IN THE SPECIFIED SEQUENCE
EXTERNAL

C6 COMMON, EQUIVALENCE
ILLEGAL EXPANSION OF COMMON IN A SUBPROGRAM

D1 DO, I/O DO
(1) VARIABLE OR EXPRESSION IS REAL
(2) 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 PARENTHESES 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
STATEMENT IS INCOMPLETE

F1 FORMAT
(1) A LEFT PARENTHESES 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 -MNIP 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 PARENTHESES 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
(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

(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 IV, FW.D, or EW.D has
W greater than 80
(4) specification aw has W=0

(1) specification =MNP or NNP has NN
greater than 49
(2) specification NNH, NNX, N(, NNE,
NNF, NNI, NNA has NN=0

what should be a statement number or ad-
dress is either an arithmetic expression
real (floating point) mode, negative,
or zero

(1) whatshould be a name or number begin-
with one of ( $ + - * / )
(2) dimensioning information is not an
integer constant

A symbol has more than six characters in
it

Real (floating point) constant is greater
than 0.0 but less than 1.E-51

Real (floating point) constant is equal
to or greater than 1.E+49

Some constant in the statement contains
more than 2 digits
An integer (fixed point) constant or
statement number (used in the statement)
has more than 5 digits
The size contains more than 5 digits
The origin contains more than 5 digits

A real (floating point) constant contains
decimal points
A Hollerith constant contains more than five letters.

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 statement.
(5) I is not an integer variable.

IF (SENSE SWITCH I)

If the sense switch I is not properly enclosed within parenthesis.

IF

(1) The argument of the IF statement is an integer constant.
(2) The argument is not properly enclosed within parenthesis.

The sense switch I is not properly enclosed within parenthesis.

GO TO

(1) There is an undesirable ) in GO TO.
(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.

INCOMPLETE STATEMENT

INCOMPLETE STATEMENT

INCOMPLETE STATEMENT

UNRECOGNIZABLE STATEMENT

THE N IS NOT AN INTEGER EXPRESSION

DOUBLY DEFINED STATEMENT NUMBER

INCORRECT I/O STATEMENT, PARENTHESIS, commas, and equal signs (DO S) are missing or misplaced.

INCORRECT DO STATEMENT, PARENTHESIS, commas, and equal signs (DO S) are missing or misplaced.

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 SUBROUTINE

(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

(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 PROGRAM

S3  RETURN

PROGRAM IS NOT A SUBPROGRAM

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, EXTERNAL    INVALID CHARACTER IN STATEMENT WHERE A
    EQUIVALENCE    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 NNNNNNH (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
     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
(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.
DOES NOT HAVE STATEMENT NUMBER.
HAS A STATEMENT NUMBER.

FOLLOW S A TRANSFER STATEMENT AND DOES NOT HAVE A STATEMENT NUMBER.
RETURN
STOP
GO TO
GO TO (N1, N2,---), I
IF
IF(SENSE SWITCH)
DO
BAD CARD
PCH CHK
JOB ABANDONED

LAST STATEMENT IN THE RANGE OF A DO.

THERE HAS BEEN A READ CHECK.
RELOAD THE CARD AND PUSH START.

THERE HAS BEEN A PUNCH CHECK WHEN
PUNCHING THE LOADER ROUTINE, CLEAR
THE PUNCH AND PUSH START TO TRY AGAIN.

(1) OCCURS AFTER ERRORS X1 AND X3.
(2) INVALID CONTROL CARD.
Table 6

<table>
<thead>
<tr>
<th>Position in Error Field</th>
<th>Digit</th>
<th>Meaning</th>
<th>Action Taken (FAC = Accumulator = Result Field)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st digit</td>
<td>1</td>
<td>Floating Point Underflow</td>
<td>FAC = 00000000000</td>
</tr>
<tr>
<td>2nd</td>
<td>2</td>
<td>Floating Point Overflow</td>
<td>FAC = +99999999999</td>
</tr>
<tr>
<td>3rd</td>
<td>3</td>
<td>Floating Point Divided by Zero</td>
<td>FAC = ±99999999999</td>
</tr>
<tr>
<td>4th</td>
<td>4</td>
<td>Fixed Point Divided by Zero</td>
<td>FAC is unchanged, i.e. J/O ≠ J</td>
</tr>
<tr>
<td>5th</td>
<td>5</td>
<td>Square Root of Neg. Number</td>
<td>Square root of absolute value of arg.</td>
</tr>
<tr>
<td>6th</td>
<td>6</td>
<td>Log of zero or Neg. Number</td>
<td>Log(0) = -99999999999; otherwise log of abs. value of arg.</td>
</tr>
<tr>
<td>7th</td>
<td>7</td>
<td>Sin or cos, arg. &gt; 10^8</td>
<td>CALL EXIT</td>
</tr>
<tr>
<td>8th</td>
<td>8</td>
<td>Exp(x) out of range</td>
<td>FAC = ±99999999999 or zero</td>
</tr>
<tr>
<td>9th</td>
<td>9</td>
<td>Input number too small</td>
<td>The number entered memory as 000000000</td>
</tr>
<tr>
<td>10th</td>
<td>1</td>
<td>Input number too big</td>
<td>The number entered memory as ±9999999999</td>
</tr>
<tr>
<td>11th</td>
<td>2</td>
<td>Plot, -1 &gt; arg &gt; 80</td>
<td>Point not plotted</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>I/O error 2. (Table 1.3.6.2)</td>
<td>Number ignored. ER2 inserted in output</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>I/O error 1. (Table 1.3.6.2)</td>
<td>Number ignored. LR1 inserted in output</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>Unused. Available for user-defined relocatable subprograms</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O Error</td>
<td>Reason</td>
<td>Result</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Input record from T/1 or paper tape over 150 characters long</td>
<td>CALL EXIT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Non-alphabetic data on A-type output</td>
<td>Number ignored</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Field width too small on I, E, F output; may be an undefined variable</td>
<td>Number</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Invalid character on input data on I, E, F, or N format</td>
<td>CALL SKIP</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>An input word has more than 88 significant digits</td>
<td>CALL EXIT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Input-output list with no numeric specifications between last opening-</td>
<td>CALL EXIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>closing parenthesis pair in Format statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Format requires more than 150 characters in a record</td>
<td>CALL EXIT</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Write-check occurred 3 times when attempting to punch output or trace</td>
<td>CALL EXIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>card</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Disc error persisted 10 times</td>
<td>CALL EXIT</td>
<td></td>
</tr>
<tr>
<td>Errors in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Format</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1F</td>
<td>- Format too long or complex for available work area</td>
<td>CALL SKIP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Comma or right parenthesis before completion of repeating format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2F</td>
<td>- Minus sign which is not part of a H- or P-specification</td>
<td>CALL SKIP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Incorrect length of H-specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- No closing parenthesis</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Statement incomplete</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Non-permissible character</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3F</td>
<td>- More than 5 levels of repeated, parenthesized Format</td>
<td>CALL SKIP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Repeated Format with more than 49 repeats</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Field width missing in I, A, E, or F, specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Field width greater than 50 in A-specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- d or decimal point missing ng in Iw.d or Fw.d</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Non-permissible character</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- d&gt;w in Iw.d or Fw.d</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- (w-d)&gt;45</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Field width, w, &gt;50 in Iw.d, Iw, or Fw.d</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Specification Aw has w = 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 7 Continued:

| READ 1 | Read check on T/H |
| READ 2 | " " " paper tape |
| READ 5 | " " " cards |

Computer halts when start is pressed, the machine will attempt to read the record again.

* Alphabetic field is replaced by R1. Last item in record may be lost.
** Numeric field is replaced by R2. Last item in record may be lost.
SAMPLE PROBLEM 7

C PROGRAM READS DATA IN BY COLS, PUNCHES DATA OUT BY ROWS

DIMENSION C(5,5)
READ 10,((C(1,J),I=1,4),J=1,5)
10 FORMAT (5F8.2)
PUNCH 10,((C(1,J),J=1,5),I=1,4)
CALL EXIT
END

SAMPLE PROBLEM 8

C PROBLEM OUTPUTS A MATRIX WITH A 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 11,1,J
11 FORMAT (2I2)
112 IF (I=99) 112,99,112
112 IF (I=30) 111,111,10
111 IF (J=30) 12,12,10
12 PUNCH 13,1,J
13 FORMAT (32H THE MATRIX IS AS FOLLOWS WITH I=,12,3H J=,12)

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.
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 .16666667 .14285714 .12500000 .11111111 .10000000
.09090909 .08333333 .07692308 .07142857 .06666667 .05882353
.05555556 .05263158

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
<table>
<thead>
<tr>
<th>ROW</th>
<th>.11111111</th>
<th>.06666667</th>
<th>.04761905</th>
<th>.10000000</th>
<th>.09090909</th>
<th>.08333333</th>
<th>.07692308</th>
<th>.07142857</th>
<th>.06666667</th>
<th>.06250000</th>
<th>.05882353</th>
<th>.05555556</th>
<th>.05263158</th>
<th>.05000000</th>
<th>.04761905</th>
<th>.04545455</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.11111111</td>
<td>.06666667</td>
<td>.04761905</td>
<td>.10000000</td>
<td>.09090909</td>
<td>.08333333</td>
<td>.07692308</td>
<td>.07142857</td>
<td>.06666667</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
</tr>
<tr>
<td>2</td>
<td>.10000000</td>
<td>.09090909</td>
<td>.08333333</td>
<td>.07692308</td>
<td>.07142857</td>
<td>.06666667</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.09090909</td>
<td>.08333333</td>
<td>.07692308</td>
<td>.07142857</td>
<td>.06666667</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.08333333</td>
<td>.07692308</td>
<td>.07142857</td>
<td>.06666667</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.07692308</td>
<td>.07142857</td>
<td>.06666667</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.07142857</td>
<td>.06666667</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.06666667</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.06250000</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>.05882353</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>.05555556</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>.05263158</td>
<td>.05000000</td>
<td>.04761905</td>
<td>.04545455</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE MATRIX IS AS FOLLOWS WITH I= 5 J=10
SAMPLE PROBLEM 9

###JOB 5
###XEQ KF2

$ JOB JOE 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)

20 IF (Y) 16,18,20

16 N=0

18 IF(Y-999999.99) 22,24,24

22 PUNCH 60,Y

20 FORMAT(Z7HPL0T OF APPROX. SQ. ROOTS OF,F8.2)

24 EX=1.

26 EX=.5*(EX+Y/EX)

30 SUB-ROUTINE PLOT DOES NOT PLOT ARGUMENTS

C LESS THAN ONE OR GREATER THAN 79

1 IF(Z<00.)85,62,82

85 IF(Z=1.)82,81,81

82 PUNCH O3, Z

83 FORMAT(24HEST. SQ. ROOT NOT PLOTTED,2X2HZ=,F9.3)

GO TO 31

31 CALL PLOT(Z).

31 AE=ABS(F((EX**2)-Y)

34 RE=AE/Y

36 FORMAT(F8.2,3XF12.8,3XF10.8,3X15,15XF9.3//)

40 FORMAT(3HY=O)

40 FORMAT(3HY=O)
APPROXIMATION OF THE SQUARE ROOT OF Y

PLOT OF APPROX. SQ. ROOTS OF 1608.01
EST. SQ. ROOT NOT PLOTTED, Z = 604.505
EST. SQ. ROOT NOT PLOTTED, Z = 403.252
EST. SQ. ROOT NOT PLOTTED, Z = 203.620
EST. SQ. ROOT NOT PLOTTED, Z = 105.758

Y   ABS. ERROR   REL. ERROR   N  EST. SQ. ROOT

1608.01   .01830000   .00001138   8  40.100

.25   .000000230   .00000920   5  .500

PLOT OF APPROX. SQ. ROOTS OF 100.10

Y   ABS. ERROR   REL. ERROR   N  EST. SQ. ROOT

100.10   .00107000   .00001069   6  10.005
SAMPLE PROBLEM 10

###JOB 5
###XEQ KF2

$ JOE 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(12)
L=I+1
READ 10,(A(I,J),J=1,L),I=1,N)

10 FORMAT(4F6.2)

C HEADER CARD

PUNCH 5,N

5 FORMAT(16THE ANSWERS FOR 12,32H 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,H)

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

$ E0J

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 .08

NOTE: ###XEQ RUN should precede the $ E0J 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 1 TO 99

* BEGIN TRACE
* FLOW TRACE
SUM=0.0
DO 30 I=1,100,2
X=1
30 SUM=SUM+X
PUNCH 40,SUM
40 FORMAT(4HSUM=,F12.2/)
CALL EXIT
END

$ EOJ
# XEQ RUN

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
$ JOB JOB 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=1
30 SUM=SUM+X
SUM=SUM/Y
PUNCH 40 SUM
40 FORMAT(F12.2)
CALL EXIT
END
$
EOJ
##XEQ RUN
#####
THE CONSOLE TYPEWRITER TYPED THE FOLLOWING LINE WHEN THE CALL
EXIT STATEMENT WAS EXECUTED,
EXIT 00300000000300000 CHECK
THE FIRST 3 INDICATED ERROR 3, FLOATING POINT DIVIDE BY ZERO
THE SECOND FLAGGED 3 INDICATES I/O ERROR 2
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)

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:

*FANDKxxxyy

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

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

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

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 (N1, N2)

where: N1 is a fixed point (integer) constant which specifies the number of variables contained in a record of data.

N2 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.

**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 \(13 \times 10\). 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 \(800 \times 12\). 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**

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.
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).

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

General form

\textbf{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 sector. 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.

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

General form

\textbf{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.

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, then 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 N2 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

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

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
# # JOB USER'S NAME OPTIONAL USER IDENTIFICATION
# # FOR X
SAMPLE PROBLEM 13

XXJOB 5
XXFORX
*XANDK0506
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(5E14.2)
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
      A(K)=A(K)+1
   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
  81.00 121.00 144.00 169.00 196.00 225.00
 225.00 289.00 324.00 361.00 400.00
 625.00 648.00 629.00 576.00 625.00
  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 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

A(K) = A(K) + 1

40 PUNCH 20, (A(K), K = 1, 25)

RECORD (IA) (A(K), K = 1, 25)

DO 50 K = 1, 25

A(K) = A(K) + 2

50 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

A(K) = A(K) + 1

60 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
<table>
<thead>
<tr>
<th>2.00</th>
<th>3.00</th>
<th>4.00</th>
<th>5.00</th>
<th>6.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.00</td>
<td>8.00</td>
<td>9.00</td>
<td>10.00</td>
<td>11.00</td>
</tr>
<tr>
<td>12.00</td>
<td>13.00</td>
<td>14.00</td>
<td>15.00</td>
<td>16.00</td>
</tr>
<tr>
<td>17.00</td>
<td>18.00</td>
<td>19.00</td>
<td>20.00</td>
<td>21.00</td>
</tr>
<tr>
<td>22.00</td>
<td>23.00</td>
<td>24.00</td>
<td>25.00</td>
<td>26.00</td>
</tr>
<tr>
<td>4.00</td>
<td>5.00</td>
<td>6.00</td>
<td>7.00</td>
<td>8.00</td>
</tr>
<tr>
<td>9.00</td>
<td>10.00</td>
<td>11.00</td>
<td>12.00</td>
<td>13.00</td>
</tr>
<tr>
<td>14.00</td>
<td>15.00</td>
<td>16.00</td>
<td>17.00</td>
<td>18.00</td>
</tr>
<tr>
<td>19.00</td>
<td>20.00</td>
<td>21.00</td>
<td>22.00</td>
<td>23.00</td>
</tr>
<tr>
<td>24.00</td>
<td>25.00</td>
<td>26.00</td>
<td>27.00</td>
<td>28.00</td>
</tr>
<tr>
<td>3.00</td>
<td>4.00</td>
<td>5.00</td>
<td>6.00</td>
<td>7.00</td>
</tr>
<tr>
<td>8.00</td>
<td>9.00</td>
<td>10.00</td>
<td>11.00</td>
<td>12.00</td>
</tr>
<tr>
<td>13.00</td>
<td>14.00</td>
<td>15.00</td>
<td>16.00</td>
<td>17.00</td>
</tr>
<tr>
<td>18.00</td>
<td>19.00</td>
<td>20.00</td>
<td>21.00</td>
<td>22.00</td>
</tr>
<tr>
<td>23.00</td>
<td>24.00</td>
<td>25.00</td>
<td>26.00</td>
<td>27.00</td>
</tr>
</tbody>
</table>
VI. AFIT Fortran

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)

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

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

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
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

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

The following programs, written for general use, are stored on punched cards. A descriptive write-up is available at the Center.
VI.4 Library of 1620 programs

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

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
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.
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 matrices.

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 ##XBQSMLRVAR

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 0 B 5 0 1
9. Finite Fourier analysis including coefficients determination and plot-back program.

LIB. No. 6.0.056
Monitor Control card ##ESEQFORIER

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 MULTIPURPOSE SPS CARD OUTPUT COMPRESSO (CARD)
1185-01.1.009 LOAD + GO FORTRAN FOR CARD OPERATION (CARD) 5 D
1185-01.1.010 AFIT IMPROVED FORTRAN (CARD) 6 D
1185-01.1.011 AN INTERPRETIVE LANG ASSEMBLER IBM 1620
1.1.012 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
01.2.006 DUMP TO RELOAD
01.2.007 FORTRAN COMPRESSOR-LOADER
01.2.008 A FLEXIBLE CARD READ ROUTINE
01.2.009 FORMAT FORTRAN OBJECT DECK COMPRESSOR
01.3.003 1620 GENERAL PURPOSE CARD COMPRESSOR
01.3.005 SQUEEZ
01.3.006 SQUISHER
01.3.008 1620 FORTRAN COMPRESSOR AND 75 COL. DUMP
01.4.001 SELECTIVE TRACE
01.4.002 TRACE PROGRAM FOR CARD I/O
01.4.003 1620 MULTI-TRACE
01.4.004 STROBIC
01.4.005 TRACE AND 1A SIMULATOR
01.4.006 1620 MULTITRACE
01.4.007 1620-402 MULTI-TRACE
01.4.008 DYNAMIC TRACE PRG FOR IBM 1620 COMP
01.4.010 GENERAL TRACE ROUTINE
01.5.001 FORTRAN SOURCE TAPE CORR
01.5.004 POST MORTEM DUMP
01.5.005 UNIVERSAL TAPE DUPLICATOR
01.5.008 ALPHANUMERIC TAPE DUP. AND CORRECTOR
01.6.001 REGRESSION ANALYSIS DATA PREPARATION
01.6.003 1620 AUTOPL OTTER
01.6.004 1620 AUTOPLOTTER
01.6.008 FORTRAN I/O ROUTINE FORMAT CONTROL
01.6.015 DYNAMIC DUMP
01.6.017 FORMAT CTRL SUBROUT FOR CARD FORTRAN
01.6.020 402 CORES DUMP
01.6.021 SYMBOL TABLE ANALYZER
01.6.022 ANL MNEMONICS DUMP
01.6.024 IMPROVED HASH TOTAL PROGRAM
01.6.028 I/O SUBROUT FOR USE IN SYM PROG
01.6.029 PROGRAM INTERRUPT
01.6.030 CARD DUMP AND LOAD
01.6.031 CARD HASH TOTAL
01.6.032 HASH TOTAL FOR CARDS
01.6.033 FLOATING POINT OUTPUT SUBROUTINE
01.6.042 SBRS. FOR PRESET PREC. F.P. ARITHMETIC
01.6.043 LOGGING PROGRAM
01.6.044 GENERAL COMPRESSOR
01.6.045 FORTRAN COMPRESSOR + MULTI-PROGRAMMER
01.6.047 DBD (DAYS BETWEEN DATES) SUBROUTINE
01.6.049 L106 FLT PT TO FIXED PT C.D.,O/P ROUT.SPS
01.6.055 PLOT SUBROUTINE FOR FORTRAN
01.6.056 PLOT SUBROUTINE FOR FORTRAN W FORMAT
01.6.058 TRANSLATOR OF ALPHANUMERIC TO EXCESS 50
01.6.060 SPS OBJECT DECK ANALYZER
01.6.061 1620 RECORD DUMP
02.0.003 INTERPRETIVE SYS PERFRM OPER COMPLEX NO.
02.0.006 INTERPRETIVE ROUTINE
02.0.008 FORGO (LOAD AND GO FORTRAN)
02.0.011 INT. SYS. FOR PERFORM OPS COMPLEX NOS.
03.0.001 VARIABLE FIELD SQUARE ROOT SUBR.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03.0.002</td>
<td>1620 FIXED POINT SQR (CLOSED) SUBRTN (CARD)</td>
</tr>
<tr>
<td>03.0.003</td>
<td>ORTHOGONAL POLYNOMIAL COEFFICIENTS (CARD)</td>
</tr>
<tr>
<td>03.0.005</td>
<td>COMP OF BESSEL FUNCT. OF INTEGRAL ORDER (CARD)</td>
</tr>
<tr>
<td>04.0.001</td>
<td>SOL FOR INT. VAL. PROB. 1 FIRST ORD. D.EQ. (CARD)</td>
</tr>
<tr>
<td>04.0.002</td>
<td>SOL FOR INT. VAL. PROB. 1 FIRST ORD. D.EQ. (CARD)</td>
</tr>
<tr>
<td>04.0.003</td>
<td>SOL FOR INT. VAL. PROB. 1 FIRST ORD. D.EQ. (CARD)</td>
</tr>
<tr>
<td>1185</td>
<td>SIMULTANEOUS EQUATION SOLUTION (CARD)</td>
</tr>
<tr>
<td>05.0.003</td>
<td>EIGENVALUES OF REAL SYMMETRIC MATRICES (CARD)</td>
</tr>
<tr>
<td>05.0.004</td>
<td>EIGENVALUES OF REAL SYMMETRIC MATRICES (TAPE)</td>
</tr>
<tr>
<td>05.0.005</td>
<td>EVALUATION OF DETERMINANTS (CARD)</td>
</tr>
<tr>
<td>05.0.007</td>
<td>SOLUTION OF SIMULTS LINEAR EQUATIONS (CARD)</td>
</tr>
<tr>
<td>05.0.009</td>
<td>CAL OF EIGENVALUES + EIGENVECTOR OF HY =LDY (CARD)</td>
</tr>
<tr>
<td>05.0.013</td>
<td>MATRIX INVERSION SUBROUTINE (CARD)</td>
</tr>
<tr>
<td>05.0.014</td>
<td>SIMULTANEOUS EQUATIONS (CARD)</td>
</tr>
<tr>
<td>05.0.015</td>
<td>MADAME (CARD)</td>
</tr>
<tr>
<td>1185</td>
<td>SIMULTANEOUS EQUATIONS A LA KING (CARD)</td>
</tr>
<tr>
<td>05.0.017</td>
<td>FLT. PT. MACRO INST. FOR SOL. OF LIN. SYSEQ (CARD)</td>
</tr>
<tr>
<td>06.0.002</td>
<td>MULTIPLE LINEAR NON@REGRESS ANALYSIS</td>
</tr>
<tr>
<td>06.0.003</td>
<td>SCRAP (CARD)</td>
</tr>
<tr>
<td>06.0.004</td>
<td>STRAP (TAPE)</td>
</tr>
<tr>
<td>06.0.007</td>
<td>STEPWISE MULT. LIN. REGRESSION ANALYSIS (CARD)</td>
</tr>
<tr>
<td>06.0.009</td>
<td>CORRELATING PROGRAM- UP TO 30 VARI (CARD)</td>
</tr>
<tr>
<td>06.0.010</td>
<td>ANALYSUS OF VARIANCE (CARD)</td>
</tr>
<tr>
<td>06.0.011</td>
<td>NON LINEAR PATCH FOR MLR PROG OF WILDER (CARD)</td>
</tr>
<tr>
<td>06.0.012</td>
<td>DISTRIBUTION STATISTICS (CARD)</td>
</tr>
<tr>
<td>06.0.013</td>
<td>SIMPLE LINEAR CORRELATION (CARD)</td>
</tr>
<tr>
<td>06.0.014</td>
<td>GENERAL ANOV (CARD)</td>
</tr>
<tr>
<td>06.0.015</td>
<td>40-40 CORRELATION (CARD)</td>
</tr>
<tr>
<td>06.0.016</td>
<td>FREQUENCY DISTRIBUTIONS- SINGLEPDBLE COL (CARD)</td>
</tr>
<tr>
<td>06.0.017</td>
<td>MULTIPLE LINEAR REGRESSION (CARD)</td>
</tr>
<tr>
<td>06.0.018</td>
<td>MULTIPOR LINEAR REGRESSION (TAPE)</td>
</tr>
<tr>
<td>06.0.019</td>
<td>MANN-WHITNEY TEST (CARD)</td>
</tr>
<tr>
<td>06.0.020</td>
<td>SCATTERGRAM GENERATOR (CARD)</td>
</tr>
<tr>
<td>06.0.021</td>
<td>CORRELATION COEFFICIENTS (UCRBL 0024) (CARD)</td>
</tr>
<tr>
<td>06.0.022</td>
<td>PRODUCT MOMENT CORRELATIONS(UCRBL 0004) (CARD)</td>
</tr>
<tr>
<td>06.0.023</td>
<td>ANALYSIS OF COVARIANCE (UCRBL 0007)</td>
</tr>
<tr>
<td>06.0.024</td>
<td>ANALYSIS OF COVARIANCE (UCRBL 0009)</td>
</tr>
<tr>
<td>06.0.026</td>
<td>ANALYSIS OF VARIANCE (UCRBL 0013)</td>
</tr>
<tr>
<td>06.0.027</td>
<td>ANALYSIS OF VARIANCE (UCRBL 0014)</td>
</tr>
<tr>
<td>06.0.028</td>
<td>ANALYSIS OF VARIANCE (UCRBL 0015)</td>
</tr>
<tr>
<td>06.0.029</td>
<td>ANALYSIS OF VARIANCE (UCRBL 0016)</td>
</tr>
<tr>
<td>06.0.030</td>
<td>ANALYSIS OF VARIANCE (UCRBL 0019)</td>
</tr>
<tr>
<td>06.0.031</td>
<td>STEPWISE REGRESSION (UCRBL 0018)</td>
</tr>
<tr>
<td>06.0.032</td>
<td>ANALYSIS OF COVARIANCE (UCRBL 0025)</td>
</tr>
<tr>
<td>06.0.033</td>
<td>ANALYSIS OF VARIANCE (UCRBL 0026)</td>
</tr>
<tr>
<td>06.0.034</td>
<td>HOMOGENEITY (UCRBL 0027)</td>
</tr>
<tr>
<td>06.0.035</td>
<td>HOMOGENEITY OF VARIANCE (UCRBL 0032)</td>
</tr>
<tr>
<td>06.0.036</td>
<td>MUL RANGE TEST OF MEAN DIF (UCRBL0034) (CARD)</td>
</tr>
<tr>
<td>06.0.037</td>
<td>MAT INVERSION-SIMULT EQ (UCRBL0052) (CARD)</td>
</tr>
<tr>
<td>06.0.038</td>
<td>LINEAR CORRELATION COEFFICIENT (CARD)</td>
</tr>
<tr>
<td>06.0.039</td>
<td>STATISTICS 1</td>
</tr>
<tr>
<td>1185</td>
<td>FACTORIAL ANALYSIS OF VARIANCE (CARD)</td>
</tr>
<tr>
<td>06.0.042</td>
<td>CONT. FOREST INV., STATISTICAL CHECK PROG (CARD)</td>
</tr>
<tr>
<td>06.0.043</td>
<td>MULTIPLE REGRESSION PACK FOR CARD 1620 (CARD)</td>
</tr>
</tbody>
</table>

*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
06.0.045 CORRELATION FOR THE IBM 1620
06.0.046 CORRELATION FOR THE IBM 1620
06.0.049 PROGRAM TO PLOT CONTOURS OF CONST. RESP. CARD 1620
1185-06.0.050 FISHERS EXACT PROBABILITY FOR 2X2 TABLE
06.0.051FINITE FOURIER ANALYSIS
06.0.055 MANN WHITNEY U TEST
06.0.059 PROBIT ANALYSIS
1185-06.0.062 FISHERS EXACT METHOD
06.0.066 STRAP@A STEPWISE REGRESSION ANALYSIS P
06.0.067 LINEAR REGSS (2 VARIABLES) LEAST SQ FIT
06.0.075 ROUNDED SUBROUTINE
1185-06.0.077 RIDGE ANAL. HIGHLY CORRELATED DATA
06.0.078 WEIBULL ANALYSIS PACKAGE
1185 06.0.111 RANDOM EXPONENTIAL NO. GEN. SUBPROGRAM
07.0.001 POLYNOMIAL CURVE FITTING
07.0.002 POLYNOMIAL CURVE FITTING
07.0.003 1620 FIX POINT SQUARE ROOT
07.0.005 SQUARE ROOT SUBROUTINE
1185-06.0.006 ARCTANGENT SUBROUTINE
07.0.007 POLYNOMAL CURVE FIT LEGENDR POLYNOMAL
07.0.008 SINE-COSINE SUBROUTINE
07.0.009 1620 RANDOM NUN SUB FOR FORTRAN W/FORMAT
07.0.010 DBLE PRECISION FLOATPT ARITH SUBROUT
07.0.012 POLY REGRESSION PROGRAM FOR IBM 1620
07.0.014 REAL AND COMPLEX RTS OF POLYNOMIALS
07.0.015 FORT RAN SYS REL.SUB.FOR GEN.RAND. NOS.
07.0.016 INTERPOL. BY NEWTONS METHOD OF 3RD DIFF.
07.0.017 CAL. OF REAL ROOTS OF REAL POLY. EQUA.
07.0.020 EMPIRICAL EQUAT BY M ETH LEAST SQUARES
07.0.021 RANDOM NUMBER SUBROUTINE FOR FORTRAN F
07.0.022 RANDOM NO. SUBR. FORTRAN W/FORMAT
07.0.023 POLYNOMIAL CURVE FITTING @ NEWTONS DIV DIFF
07.0.024 FOURIER CURVE FITTING @ 1 PASS
07.0.025 GAMMA SUBROUTINE FOR FORMAT FORTRAN
07.0.026 ERROR FUNCTION@1620 FORTRAN SUBROUTINE
1185 07.0.027 POLY. CURVE FITTING AND EVALUATION
1185 07.0.029 CAL. OF THE ROOTS OF A COMPLEX POLY. EQ.
1185-07.0.30 LAGRANGE INTERPOLATION
1185 07.0.31 FOURIER CURVE FITTING - 2 PASS
1185 07.0.32 CAL. OF THE ROOTS OF A REAL POLY. EQUA.
07.0.033 REAL ROOTS-REAL NON-LIN. EQ. IN 1 VAR
1185 07.0.036 ROOTS OF POLYNOMIAL EQUATION
1185 07.0.037 BESSEL FUNCTION SUB. FORTRAN W FORMAT
1185 07.0.039 BIVARIATE CURVE FIT
1185 07.0.041 CAL. OF THE REAL ROOTS OF A SYS. OF K
08.2.001 A ONE DIMENSIONAL FEW GRP DIFF. CODE
08.2.002 CAPTURE GAMMA SHIELDING PROGRAM
08.2.004 SPEK (SPEEDY KATE)
08.2.005 RSNT
09.1.001 ARDC MODEL ATMOSPHERE SUBROUTINE
09.2.001 1620 SUBDIVISION PROGRAM
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Medium</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.2.002</td>
<td>CUT AND FILL</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>09.2.003</td>
<td>CUT+FILL</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.2.004</td>
<td>WATERWAY COMPUTATIONS</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.2.006</td>
<td>TRAVERSE ANALYSIS</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.2.007</td>
<td>TRAVERSE ANALYSIS</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>09.2.008</td>
<td>WATERWAY COMPUTATIONS PROG</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.2.009</td>
<td>SKewed BRIDGE ELEVATIONS</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.2.012</td>
<td>RELOCATION OFFSETS</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.2.013</td>
<td>RECTANGULAR CONCRETE COLUMN ANALYSIS</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.2.014</td>
<td>GEN.,VIRTUAL WORK ANALYSIS OF STRUCTURES</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.2.016</td>
<td>DTM DESIGN SYSTEM PROGRAM</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.2.019</td>
<td>DTM DESIGN SYSTEM 4OK</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.2.020</td>
<td>SLOPE STABILITY ANALYSIS</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.2.022</td>
<td>COL. ANAL. UNDER AXIAL Lg. 2 WAY BENDING</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.2.024</td>
<td>BEAM CAMBER CALCULATIONS</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.2.028</td>
<td>BACKWATER CURVE PROGRAM</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>1185</td>
<td>09.2.032</td>
<td>VEHICLE SIM. AND OPER. COST SYSTEM</td>
<td>CARD</td>
</tr>
<tr>
<td>09.3.001</td>
<td>GAS NETWORK ANALYSIS</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>09.3.002</td>
<td>SHORTCUT DISTILLATION</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>09.3.003</td>
<td>GAS NETWORKS ANALYSIS—PUBLIC UT. DEPT.</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.3.006</td>
<td>ASTM-TO-TBP+TBP-TO-ASTM, DISTILL, CONVER</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>1185-09.3.007</td>
<td>FLASH VAPORIZATION CALCULATIONS</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.3.008</td>
<td>MULTI. DISTIL. TOWER DESIGN SHT/CUT M ETH.</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.3.009</td>
<td>UNIT OPIERTS. SIMULATOR VAPOR@LIQD. SYST</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>1185-09.3.010</td>
<td>PLATE@TO-PLATE DISTILLATION PROGRAM</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.4.001</td>
<td>ELECTRIC LOAD FLOW PROGRAM</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>09.4.002</td>
<td>LOCA OF SHUNT CAPACITOR ON RADIAL LINES</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>09.4.003</td>
<td>ELECTRIC LOAD FLOW PROGRAM</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.4.004</td>
<td>SELECTION OF ECONOMIC CONDUCTOR SIZE</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.005</td>
<td>ECOM CONDCTR SIZE SELEC BY KELVIN LAW</td>
<td>TAPE</td>
<td>C</td>
</tr>
<tr>
<td>09.4.006</td>
<td>SHORTCIRCUIT ANALYSIS</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.007</td>
<td>SHORT CIRCUIT CALCULATION</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.008</td>
<td>TRANSMISSION LOSSES PENALTY FACTORS</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.009</td>
<td>CURVE FIT-SIMUL PLANT RECORD (NEES-36)</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.010</td>
<td>ECONOMIC DISPATCH DETERMINATION</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.4.012</td>
<td>TRANSFORMR RATING FOR NORMAL+EMERG OPER</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.013</td>
<td>TRANSIENT STABILITY FOR TEN MACHINES</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.014</td>
<td>TRANSFORMER SHORT TIME LOADING CURVES</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.015</td>
<td>SIMUL WQ SOLPMAT INVERSION/COMPLEX VAL</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.016</td>
<td>NETWORK REDUCTION PROGRAM</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.4.020</td>
<td>RAD. 3 PHASE LINE DROP CAL. IN 2 PASSES</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.021</td>
<td>BATCH LOAD FLOW</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>09.4.022</td>
<td>ECON GENERATION DISPATCH PROGRAM</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>1185-09.4.023</td>
<td>SHORT CIRCUIT ANALYSIS BY MATRIX METHOD</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.4.025</td>
<td>LOAD ANALYSIS OF A COMMUNICATIONS NET</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.6.001</td>
<td>STRAIN GAGE DATA REDUCTION</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.6.002</td>
<td>STRAIN GAGE DATA REDUCTION</td>
<td>TAPE</td>
<td>D</td>
</tr>
<tr>
<td>1185</td>
<td>09.7.001</td>
<td>DIST OF WATER FLOW IN A PIPE NETWORK</td>
<td>TAPE</td>
</tr>
<tr>
<td>09.7.002</td>
<td>GENERALIZED PLOTTER II</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.7.003</td>
<td>GENERALIZED PLOTTER</td>
<td>CARD</td>
<td>C</td>
</tr>
<tr>
<td>09.7.004</td>
<td>S-100 STRESS ANAL FLNGE WITH TAPED HB</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.7.006</td>
<td>HYDRAULIC ANAL. OF FLOW IN PIPE NETWORK</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>09.7.007</td>
<td>STEAM + WATER PROP. OF EFFICIENCY PROG.</td>
<td>CARD</td>
<td>D</td>
</tr>
<tr>
<td>1185-09.7.008</td>
<td>WATER FLOW IN A PIPE NET. BY H.C., SOL.</td>
<td>CARD</td>
<td>D</td>
</tr>
</tbody>
</table>
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 LINEAR 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
10.2.002 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) 12 C
10.3.007 A PROG FOR ANALYZING THE INVESTMT OF CAP (CARD) 12 C
10.3.008 1620 NODE NUMBERING (CARD)
10.3.009 1620 PERT (TAPE) 3 D
10.3.010 ECON ANALYSIS OF PLANS OUT 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 (CARD) 11 D
1185 10.3.017 STUDENT SCHEDULING (CRD) 12 C
11.0.001 THE CHINESE BAR PRING 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 RANDOM 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. 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 (0) 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 causes 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........
-+++IAAAAAALaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab+++
which causes an automatic skip to column 6, then a shift to alphabetic mode (an "A" is a combination of a "4" 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)

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)

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)**

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**

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:

<table>
<thead>
<tr>
<th>Alphameric Character</th>
<th>Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Blank)</td>
<td>(Blank)</td>
</tr>
<tr>
<td>. (Period)</td>
<td>12, 3, 8</td>
</tr>
<tr>
<td>)</td>
<td>12, 4, 8</td>
</tr>
<tr>
<td>Alphabetic Character</td>
<td>Card</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
</tr>
<tr>
<td>+</td>
<td>12</td>
</tr>
<tr>
<td>$</td>
<td>11, 3, 8</td>
</tr>
<tr>
<td>*</td>
<td>11, 4, 8</td>
</tr>
<tr>
<td>(Minus)</td>
<td>11</td>
</tr>
<tr>
<td>/</td>
<td>0, 1</td>
</tr>
<tr>
<td>(Comma)</td>
<td>0, 3, 8</td>
</tr>
<tr>
<td>(</td>
<td>0, 4, 8</td>
</tr>
<tr>
<td>(Dash, not a minus)</td>
<td>3, 6</td>
</tr>
<tr>
<td>A</td>
<td>12, 8</td>
</tr>
<tr>
<td>B</td>
<td>12, 7</td>
</tr>
<tr>
<td>C</td>
<td>12, 6</td>
</tr>
<tr>
<td>D</td>
<td>12, 5</td>
</tr>
<tr>
<td>E</td>
<td>12, 4</td>
</tr>
<tr>
<td>F</td>
<td>12, 3</td>
</tr>
<tr>
<td>G</td>
<td>12, 2</td>
</tr>
<tr>
<td>H</td>
<td>12, 1</td>
</tr>
<tr>
<td>I</td>
<td>11, 9</td>
</tr>
<tr>
<td>J</td>
<td>11, 8</td>
</tr>
<tr>
<td>K</td>
<td>11, 7</td>
</tr>
<tr>
<td>L</td>
<td>11, 6</td>
</tr>
<tr>
<td>M</td>
<td>11, 5</td>
</tr>
<tr>
<td>N</td>
<td>11, 4</td>
</tr>
<tr>
<td>O</td>
<td>11, 3</td>
</tr>
<tr>
<td>P</td>
<td>11, 2</td>
</tr>
<tr>
<td>Q</td>
<td>11, 1</td>
</tr>
<tr>
<td>R</td>
<td>10, 9</td>
</tr>
<tr>
<td>S</td>
<td>10, 8</td>
</tr>
<tr>
<td>T</td>
<td>10, 7</td>
</tr>
<tr>
<td>U</td>
<td>10, 6</td>
</tr>
<tr>
<td>V</td>
<td>10, 5</td>
</tr>
<tr>
<td>W</td>
<td>10, 4</td>
</tr>
<tr>
<td>X</td>
<td>10, 3</td>
</tr>
<tr>
<td>Y</td>
<td>10, 2</td>
</tr>
<tr>
<td>Z</td>
<td>9, 9</td>
</tr>
</tbody>
</table>

Numbers 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. (IBM®FII-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 = (3+500.22%C) / (.005%D+4.)
500.22 is stored as 03 .50022
       Exponent  Mantissa
.50022*10^3 = 500.22
.005 is stored as -02 .5
        Exponent  Mantissa
.5 * 10^-2 = .005
4. is stored as 01 .4
        Exponent  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 III-D

Card columns: 123456789 ..........32.............60.62.............80

Card 1.     ##JOB NAME OF PROGRAMMER OPTIONAL USER IDENTIFICATION
Card 2.     ##FORX

Where: The #1'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 #1'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.14a for a listing of programs available for disk storage) The program name is punched starting in Card column 7.
## INDEX TO ACCEPTABLE FORTRAN STATEMENTS

<table>
<thead>
<tr>
<th>Statement</th>
<th>General Form</th>
<th>L. &amp; G.</th>
<th>KFII</th>
<th>FII</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>19, 20</td>
<td>28</td>
<td>53</td>
<td>MS</td>
</tr>
<tr>
<td>ACCEPT TAPE</td>
<td>19, 20</td>
<td>N.P. (See NOTE)</td>
<td>53</td>
<td>MS</td>
</tr>
<tr>
<td>ARITHMETIC IF</td>
<td>16</td>
<td>G</td>
<td>G</td>
<td>MS</td>
</tr>
<tr>
<td>ARITHMETIC Statement Function</td>
<td>KF (See NOTE below)</td>
<td>N.P.</td>
<td>70</td>
<td>MS</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>KF</td>
<td>N.P.</td>
<td>49</td>
<td>N.P</td>
</tr>
<tr>
<td>CALL name</td>
<td>KF</td>
<td>N.P.</td>
<td>75</td>
<td>MS</td>
</tr>
<tr>
<td>CALL EXIT</td>
<td>19</td>
<td>N.P.</td>
<td>52</td>
<td>MS</td>
</tr>
<tr>
<td>CALL PLOT</td>
<td>KF</td>
<td>N.P.</td>
<td>77</td>
<td>N.P</td>
</tr>
<tr>
<td>CALL RAND</td>
<td>KF</td>
<td>N.P.</td>
<td>78</td>
<td>N.P</td>
</tr>
<tr>
<td>CALL RESOLV</td>
<td>KF</td>
<td>N.P.</td>
<td>79</td>
<td>N.P</td>
</tr>
<tr>
<td>CALL ROUND</td>
<td>FII</td>
<td>N.P.</td>
<td>N.P.</td>
<td>104</td>
</tr>
</tbody>
</table>

**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

CALL SOLVE

CALL SORT

COMMON

COMPUTED GO TO

CONTINUE

DATA

DIMENSION

DO

END

EQUIVALENCE

FORMAT

FETCH

FIND

FUNCTION name

IF(SENSE SWITCH)

PAUSE

PRINT

PUNCH

PUNCH TAPE

READ

RECORD

REREAD

RETURN

STOP

SUBROUTINE name

TYPE

Unconditional GO TO