DOCUMENT RESUME

ED 053 546 EM 009 125

AUTHOR	Hill, Edward, Jr.
TITLE	The On Line Modeling System. Part One, General Discussion. Part Two and Part Three, Programmers' Reference Manual.
INSTITUTION	National Institutes of Health, Bethesda, Md. Div. of Computer Research and Technology.
PUB DATE	Apr 71
NOTE	211p.
EDRS PRICE	EDRS Price MF-$0.65 HC-$9.87
DESCRIPTORS	*Biology, Computer Graphics, *Computer Programs, *Manuals, *Medicine, Models, *On Line Systems, Programing Languages
IDENTIFIERS	OLMS, *On Line Modeling System

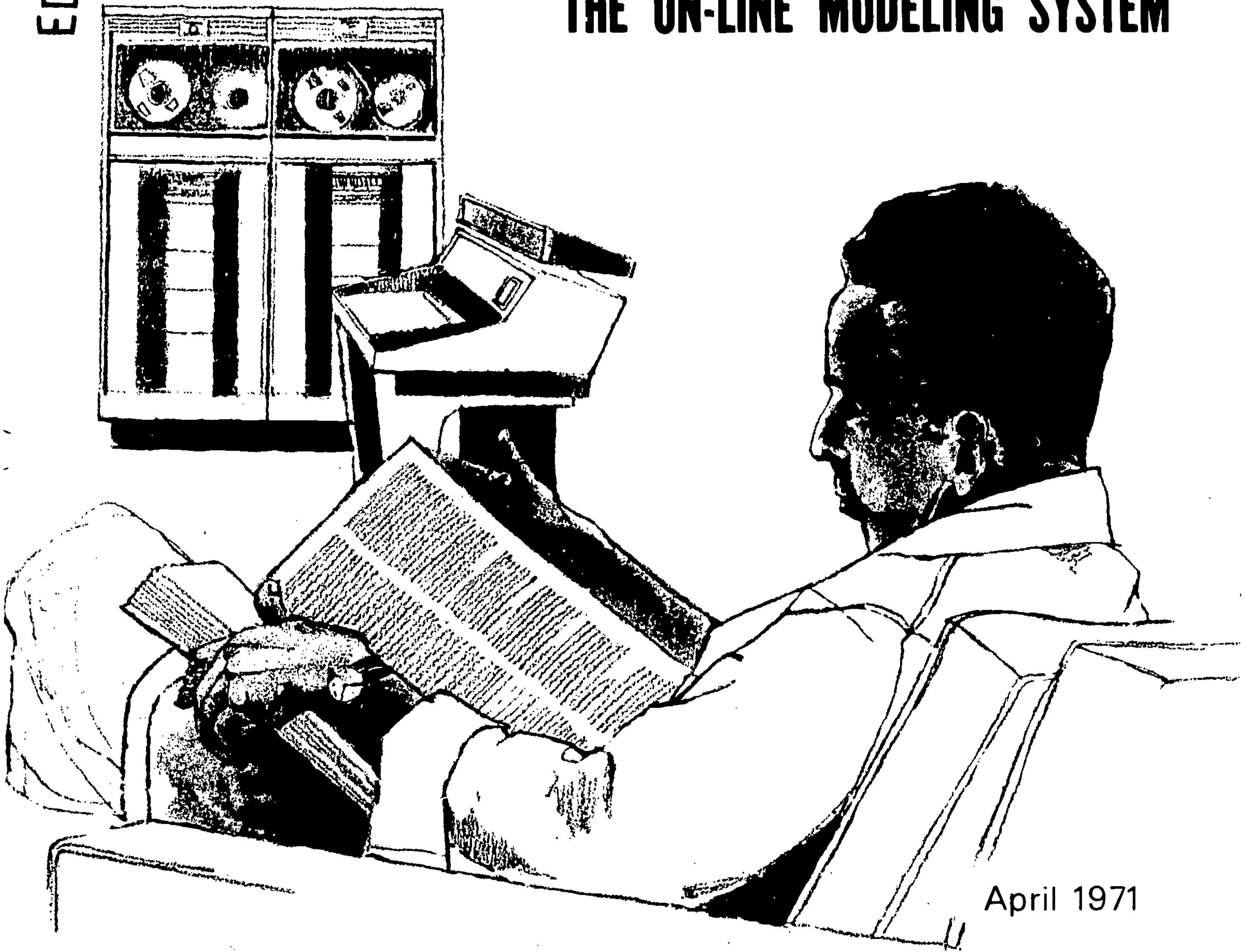
ABSTRACT

An On-Line Modeling System (OLMS) which should be particularly useful to the biomedical community is described which utilizes a process called "overlaying" to simplify loading and executing programs. OLMS has been constructed and implemented as an interpreter; that is, a program that translates and then executes each source statement in sequence where these two operations follow each other in close time proximity. It has been written so that it can be run under the IBM S/360 and the PDP-10 operating systems, but has been implemented only on the PDP-10. OLMS operates from a command language that is executed interpretively through a set of closed subroutines. The user may run a job, save data, get data, and display data. While his program is running, he can access the whole memory. This report consists of a description of OLMS and programmer's reference manuals for both the OLMS graphical system and OLMS. (JY)

TECHNICAL REPORT NO. 6

PART I

THE ON-LINE MODELING SYSTEM



April 1971

U.S. DEPARTMENT OF HEALTH, EDUCATION, AND WELFARE
Public Health Service National Institutes of Health

The Division of Computer Research and Technology, NIH, will issue on an irregular basis technical documents which we believe will be of particular interest to the biomedical community. These reports will include detailed descriptions of relevant computer programs and instructions in their use (as well as some theoretical background), in hopes that interested scientists will be encouraged to gain first-hand experience in applying them. In some cases, such reports may serve as foci around which DCRT will structure training courses to expand the knowledge and experience of NIH staff in applying computer science to problems of research and management. Circulation of these reports within the biomedical community broadly is, of course, encouraged.

A. W. Pratt, M.D.

A. W. Pratt, M.D., Director, DCRT

THE ON LINE MODELING SYSTEM

Part I. General Discussion

by

Edward Hill, Jr.

Laboratory of Applied Studies

Division of Computer Research and Technology
National Institutes of Health, Public Health Service
Department of Health Education and Welfare, Bethesda, Maryland 20014

PART I

THE ON LINE MODELING SYSTEM

OLMS

Table of Contents

PREFACE

This report is intended to introduce a simple modeling system, that is easy to use, flexible and general enough to allow a scientist or programmer to extend it using Fortran IV.

There are three parts to this report;

Part I : The On Line Modeling System ,

Part II : A Programmers' Reference Manual For The On Line Modeling Graphical System, and

Part III : A Programmers' Reference Manual For The On Line Modeling System.

Before using this system a user should read all of Part I and at least the miscellaneous Section in Part III.

SECTION I

1.1 INTRODUCTION

Access to computers is limited by many constraints; namely, memory size, memory cost, computer structure, computer language, internal operating systems and monitors, etc. The design and implementation of any program that a user wishes to interact with is directly influenced by the above factors and perhaps others that are not listed. Most computers are designed for general purpose usage and as such the generality given any particular user is a compromise between the speed of loading and executing programs. Mathematical models of complex processes require complicated programs which tend to grow in size as the problem solving capability increases, and new knowledge of the system being modeled is incorporated. This concept is clear if we think of modeling a physiological structure. Independent unit structures may be modeled, and at some point, connected together as a model for a complete structure. An example is given in Section IV showing how the input output relations of the Hodgkin-Huxley Nerve Equations can be studied.

Since most models grow as the investigator proceeds, the total storage requirements for instructions in a given program may exceed the available storage capacity of the machine. It then becomes necessary to use repeatedly the same blocks of the computer's internal storage in order to execute the total program. The process whereby data or subprograms replace other data or subprograms which are no longer needed, is called overlaying. This overlaying process provides a more efficient utilization of memory capacity and increases the flexibility of the internal programming structure. The On-Line Modeling System discussed here utilizes this overlay concept to simplify the process of loading and executing programs.

A program that translates and then executes each source statement (e.g., an individual Fortran statement such as x = y) in sequence, where these two operations follow each other in close time proximity is called an interpreter. The use of an interpreter is advantageous if rapid response to modifications of the source statements is required. For this reason The On-Line Modeling System has been constructed and implemented as an interpreter.

Present day operating systems and monitors are influenced by the needs of many people; however, they may not completely fulfill the needs of any particular user. Furthermore, many generalized operating systems are designed in such a way that they cannot normally allow the user to run programs, save data, get data, and display data by pushing buttons on a keyboard. The On-Line Modeling System described in this report has been written so that it can be run under the existing (S/360) and (PDP-10) Operating Systems. However, it has been implemented only on the PDP-10.

The OLMS operates from a command language that is executed interpretively, through a set of closed subroutines. This command language has eight modes of operation which are discussed later in this report. In one of these, the Function Key Mode, the system can be operated by a push button keyboard, allowing a user to manipulate the OLMS without having to know the details of the command language.

In summary, the OLMS is a system that allows a user to run a job,save data, get data, generate data, and display data. When the user's program is running he can access the whole memory (i.e., the OLMS is not in memory when the user's program is running). This is accomplished by using the system overlay loader to overlay the OLMS. The interpreter for the OLMS command language gives the user flexibility in using the various modes of the system. In the Function Key Mode a user can interact with the system by pushing buttons and answering questions. As the user answers the questions a syntactic command is generated and passed to the interpreter to test its acceptance. The advantages of such a system are as follows:

1) A user can use the system without any display programming or any prior knowledge of how the display works.
2) A user can run a program and see immediate relationships among the variables (i.e., graphs can be generated by pushing buttons).
3) When the user's program is running he has the capability of using the central processor, fast memory, and all I/O devices.

Finally, the handling of programs that are too large to fit in memory is possible since the files are each organized and the user can interact with the total organization. Therefore, the OLMS provides investigators who must use large programs and large amounts of experimental data with a flexible and responsive computer facility which allows them to explore all the characteristics of their data. Furthermore, the individual user may modify the OLMS with some prior knowledge of Fortran.

1.2 Example Problem

This example shows how the OLMS can be used to study the input and output relations of a simple quadratic equation of the form $Y = AX^2 + BX + C$. The algorithm listed below shows how this equation can be implemented on the OLMS.

Algorithm

The system buffer is defined as a named common block; namely, COMMON/COMBUF/SYSBUF(384).

```
A1.  [Input coefficients by teletype]

     A,B,C←---INPUT

A2.  [Input the number of points to generate by teletype].

     NPTS←---INPUT
     if (NPTS>128) then go to A9

A3.  [Initialize I].

     I←---1

A4.  [Generate points].
     if (I>NPTS) then go to A8.

A5.  [Generate X value].

     SYSBUF(I)←---I
```

```
A6.  [Generate Y value].
     SYSBUF(128+I) = A*(I**2) + B*I + C

A7  [Increment I].
    I←---I+1
    Go to A4

A8.  [Write SYSBUF on disk].
     CALL WRR(1)

A9.  [Load Interpreter].
     CALL INTERP(1)

A10. Terminate
```

In this algorithm the routine WRR(1) writes the content of the system buffer on the disk. The routine INTERP(1) is a call to the Loader to load the OLMS interpreter.

A Fortran IV program is given to show an implementation of this algorithm.

FORTRAN IV PROGRAM

```
      COMMON/COMBUF/SYSBUF(384)
      TYPE 12
      ACCEPT 1,A,B,C
  1   FORMAT(3F)
      TYPE 13
      ACCEPT 2,NPTS
  2   FORMAT(I)
      IF(NPTS.GT.128)TYPE 11
      IF(NPTS.GT.128)GO TO 9
      I = 1
  4   IF(I.GT.NPTS)GO TO 8
      SYSBUF(I)+I
      SYSBUF(128+I)=A*FLOAT(I)**2+B*FLOAT(I)+C
      I = I+1
      GO TO 4
  8   CALL WRR(1)
  9   CALL INTERP(1)
 11   FORMAT(' ERROR NPTS IS, GREATER THAN 128'/'  ')
 12   FORMAT(' INPUT A,B,C'/'  ')
 13   FORMAT(' INPUT NPTS'/'  ')
      END
```

Steps used to get a job ready for the OLMS.

(1) Place cards on disk

```
R PIP
DSK:NAME ← CDR:
```

(2) Compile job

```
COM NAME
```

(3) Load job

```
LOAD NAME,INTOP
```

(4) Save job

```
SA DSK:NAME
```

The information needed to operate the OLMS can be acquired by reading The Function Key Mode in Section III, Section V, and the following semantics in Section II of this report:

4.1 (Save Statement)

5.1 (Get Statement)

6.1 (Delete Statement)

9.1 (Clear Statement)

10.1 (Rename Statement)

12.1 (Plant Statement)

16.1 (Start Statement)

18.1 (Kill Statement)

19.1 (Enter Statement)

20.1 (Display Statement)

22.1 (Dout Statement)

23.1 (Top Statement)

25.1 (Transfer Statement).

Now you are ready to use the OLMS; therefore, branch to the computer, otherwise, proceed to add depth to your understanding.

1.3 Generalized Statement of Problem

Given that a program is too large to fit in the existing computer memory, we may divide it into C_j parts (where j = 1,2,...,n). The subdivision is made so that the user may interact with any c_j.

Each C_j is loaded for execution with an overlay structure. Suppose that the part C_1 has been loaded and executed. Depending on the output from C_1 the user may want to execute C_1 again or one of the other $C_2,C_3,\ldots,C_n$ parts of the program. To perform this task, it is necessary to keep all program parts organized for execution. This is done by using the proper push buttons on the Function Key Box (see Section 3 of Part II).

To simplify the loading and executing of program parts, we use a command language that is executed interpretively, and which keeps the C_j's and their associated files organized for manipulation. To accomplish this kind of organization, a storage management scheme must be available that is general and easy to implement. The OLMS uses an algorithm called the T-Algorithm to accomplish this management scheme. This algorithm requires that the computer system being used to implement the interpreter have an overlay structure. The T-Algorithm is outlined below:

T-Algorithm

T1: Create a simple overlay program that calls the interpreter for the command language.

T2: The program created in step T1, will be a part of every C_j that we wish to interact with at execution time. Note, a call to the program created in step T1 may appear anywhere in C_j.

T3: To start the system, make a call to the program created in step T1.

The T-Algorithm functions so that any task executed from the interpreter will overlay the interpreter. This algorithm allows a user to develop a good simulation program under most existing operating systems. Finally, this algorithm generates flexibility for interaction with the Rand Tablet and Function Keys.

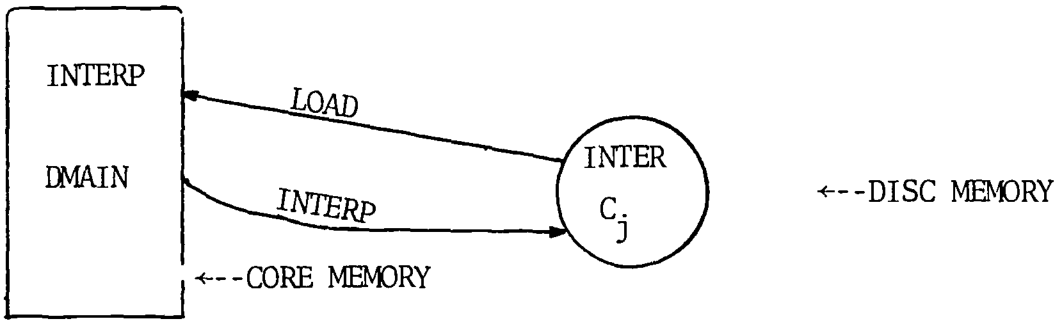
The sections of this report that follow are an implementation of the T-Algorithm. To understand the details of how the T-Algorithm works, refer to the following discussion and diagrams.

An Example of the T-Algorithm

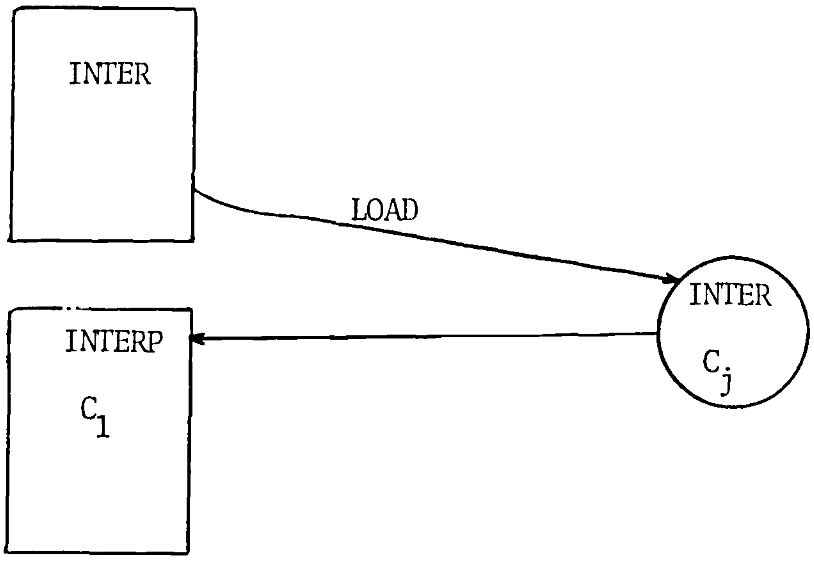
Let INTERP be the name of a routine that calls an interpreter (called INTER).

Let DMAIN be the name of a routine that calls INTERP to start the system.

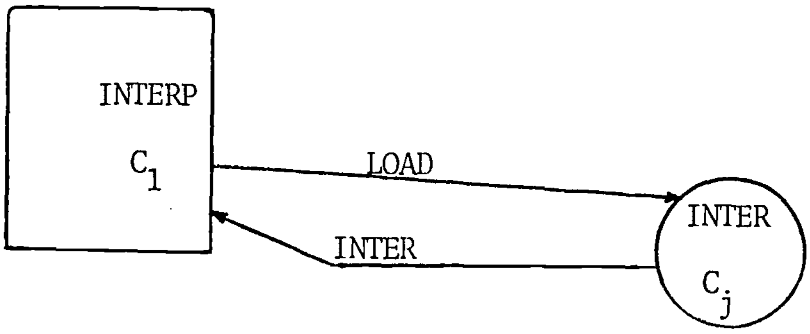
Let C_1 be a part of a program that displays data. Assume that INTER and the C_j's are on disk. The diagrams below show how the memory would look under the T-Algorithm.

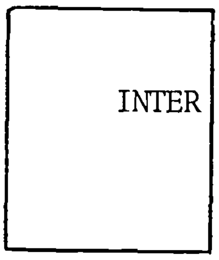


The interpreter is now in memory and we can interact with it in any desired way. From the interpreter we will now allocate the C_j's.



The program C_1 to display data is now in memory. When we wish to return to our interpreter, a call to INTERP is executed.





The interpreter is now in memory and our algorithm may continue.

SECTION II

A Syntactic Description of A Modeling Command Language

Language Description

The language in which a language may be defined is termed a metalanguage and must be uniquely distinguishable from the language being described. To formalize the definition in the metalanguage, each definition is given the form of a statement or construct, which is analogous to a formula. However, to accomplish some unique features of such a specification, the operators define a mode of construction, or concatenation. The following symbols are employed in this metalanguage:

X	the object named X
:=	can be formed from
|	or (the exclusive or)
$\{Z\}_i^j$	Z is to be repeated at least i times but not more than j times. When i is omitted, its value is to be assumed to be 1, and when j is absent, its value is assumed to be infinity.

The syntactic part of this paper is divided into sections. Each section begins with a section number. Within each section there are subsections. All subsections are numbered with subsection numbers. An example of the structure of the syntactic part of this paper is given below.

1.   Section

1.1   Subsection

1.1.1   Subsection

2.   Section

The last subsection of every section contains the general semantics of the section.

1. Letter, Special Character, Null, Blank, Digits, Sign Numbers

1.1. Letter

< Letter >: = A|B|C|D-------|Z

1.1.1. Semantics

Letters do not have individual meaning. They are used for forming variables.

1.2. Special Character

<special character>: = + |-|,

1.3. Null

<null>: = Λ

1.3.1. Semantics

Lambda is a string of length zero.

1.4. Blank

<blank>: =

1.4.1. Semantics

Blank is a "special character" of length one.

1.5. Digits

<digit>: = 0|1|2|3|4|5|6|7|8|9

1.5.1. Semantics

Digits are used for forming numbers and variables.

1.6. Sign

<sign>: = <null> |+|-

1.6.1. Semantics

A null sign is an implicity plus sign.

1.7. Exponent

<exponent>: = E<sign>{<digit>}$_1^2$

1.8. Numbers

1.8.1. Integer

<integer>: = {<digit>}$_1^{10}$

1.8.2. Decimal Number

<decimal number>: = {<digit>}$_1^{n \le 10}$ {<digit>}$^{10-n}$

1.8.3. Fraction

<fraction>: = {.{<digit>}$_1^{10}$ }

1.8.4. Number

<Number>: = <fraction>|<decimal number>|

<integer>|{< decimal number >|<fraction>}$_1^1$

{<exponent>}$_1^1$

1.1.0. Signed Number

<Signed number>:=sign><number>

2. Variable

<Variable> : =<Name>

3. Name

<Name>: = <letter>}<letter>|<digit>}$_0^4$

4. Save statement

<Save statement>: = SAVE{<Blank>|,}$_1^1$

{<Directory statement>}:{<Blank>|,^{1}}$_1^1$

{<File>}$_1^1$

4.1. Semantics

Files may be saved on magnetic tape, Dectape, and Disk. All files are saved from the system buffer. The system buffer is a named common block called COMBUF. The buffer size is three hundred and eighty four words.

5. Get statement

<Get statement>:= GETF{<Blank>},}$_1^1$

{<Directory statement>}$_1^1$ {<Blank>|^1, }$_1^1${File>}$_1^1$

5.1. Semantics

Files are read from magnetic tape, Dectape, and Disk into the system buffer.

6. Delete statement

<Delete statement> : = DELE{<Blank>|,}$_1^1$

{<Directory statement>}$_1^1$ {<Blank>|,}$_1^1$ {<File>}$_1^1$

6.1. Semantics

Files may be deleted from magnetic tape, Dectape and Disk. Any deleted can never be recovered.

7. System Directory statement

<System Directory statement>: = NEWD

{<Blank>|,}$_1^1$ <Device Name>

7.1 Semantics

The system directories can be written on magnetic tape, Dectape, and Disk. The file and library directory are called the system directories.

8. Directory Input statement

<Directory input statement>: = DIRM

{<Blank>|,}$_1^1$ {<Device Name>}$_1^1$ {{<Blank>|,}$_1^1$ {<Directory statement>}$_1^1$}$_0^1$

8.1 Semantics

If the device name is magnetic tape the magnetic tape directory will be read. A device name for Dectape and Disk will read the system directories. When the system directories are read the old contents of the directory tables are destroyed. For magnetic tape the <Directory statement> must be omitted.

9. Clear statement

<Clear statement>: = CLEA {<Blank>|,}$_1^1$

{<Device name >}$_1^1$

9.1. Semantics

The Dectape and magnetic tape directories can be cleared. The disk directory cannot be cleared.

10. Rename statement

<Rename statement>: = RENA {<Blank>|,}$^1_1$ {<Directory statement>}$^1_1$

{<Blank>|,}$^1_1$ {<File>}$^1_1$ {<Blank>|,}$^1_1$

TO {<Blank>|,}$^1_1$ {<File>}$^1_1$

10.1. Semantics

Files may be renamed on magnetic tape, Dectape and Disk.

11. Description statement

<Description statement>: = DSCR

{<Blank>|,}$^1_1$ {<Directory statement>}$^1_1$

{<Blank>|,}$^1_1$ {<File>}$^1_1$

11.1. Semantics

File descriptions of fifty characters may be printed in the system directories.

12. Plant statement

<Plant statement>: = PLAN {<Blank>|,}$^1_1$

{<Directory statement>}$^1_1$

{<Blank>|,}$^1_1${<File>}$^1_1$

12.1. Semantics

File names may be planted in the system directories.

13. Assign statement

<Assign statement>: = ASSN {<Blank>|,}$^1_1$

{<Device name>|{CHAN{<Blank>|,}$^1_1$

<Device name>}$^1_1$ |{OLMS{<Blank>},}$^1_1$

<Device name>}$^1_1$}$^1_1$

13.1. Semantics

All devices are assigned at execution time. These assignments are only methods to update system device tables. CHAN in the commands tells the system where user chain files are stored. OLMS in the commands tells the system where the On Line Modeling System is stored.

14. Deassign statement

<Deassign statement>: = DEAS{<Blank>|,}$_1^1$

{<Device name>|{CHAN{<Blank>|,}$_1^1$<Device name}$_1^1$

|{OLMS{<Blank>|,}$_1^1$ <Device name>}$_1^1$}$_1^1$

14.1. Semantics

Devices are deassigned at execution time. All deassignments are local, that is, they update system tables. CHAN in the command deassigns all CHAIN Files. OLMS in the command deassigns the system.

15. Model statement

<Model statement>: = {<Blank>|,}$_1^1$MODEL{<Blank>|,}$_1^1$

<Name>{<Blank>|,}$_1^1$ {A|S|G}$_1^1$

15.1. Semantics

The A refers to an analysis model. S and G refers to a simulation and generation model respectively.

16. Start statement

<Start statement>: = START{<Model statement>}$_1^1${<Core statement>}$_0^1$

16.1. Semantics

If the core statement is omitted the model will be run in 20k including the root segment and Block Data. The core size can be determined when the CHAIN file is generated. The loader returns the CHAIN size when a file is generated. If this size is greater than 20k the core statement should be used.

17. Proccss statement

<Process statement>: = PROCE{<Blank>|,}$_1^1$

{<Model statement>}$_1^1$ {<Core statement>}$_0^1$

17.1. Semantics

The Process statement will process the output from a simulation model. The core rules are the same as those in 16.1. Remember these syntactic rules are defined on tables which are updated at execution time; therefore, a simulation model or generation model can be run using this statement if the name has been given with the A property in the ENTER statement.

18. Kill statement

<Kill statement>: = KILLM <Model statement>

18.1. Semantics

KILL is local in that it removes the model name from the model name table.

19. Enter statement

<Enter statement>: = ENTER{<Model statement>}$_1^1$

19.1. Semantics

The ENTER statement is used to put model names in the model name tables.

20. Display statement

<Disp statement>: = DISP

20.1. Semantics

This command will bring in the OLMGS. The OLMGS is operated interactively using the RAND Tablet. All the capabilities of the OLMGS is utilized to fit any users needs.

When the OLMGS is loaded a user can use it interactively as described in the flow diagram in Figure 2. The flow diagram is designed to show the appearance of the CRT at various stages.

The commands available in this program are listed below.

<DOUT statement>: = DOUT

<Top statement>: = T

<Yes-No statement>: = {Y|N}$_1^1$

A carriage return is used to proceed in multipicture displays.

To understand the capabilities of the OLMGS read reference [12]. When multipictures are displayed the system block transfer function is used to move blocks of data. A diagram in figure 3 shows how the data is moved from one buffer to another to accomplish the multipicture display task. Consider the example given in Figure 1.

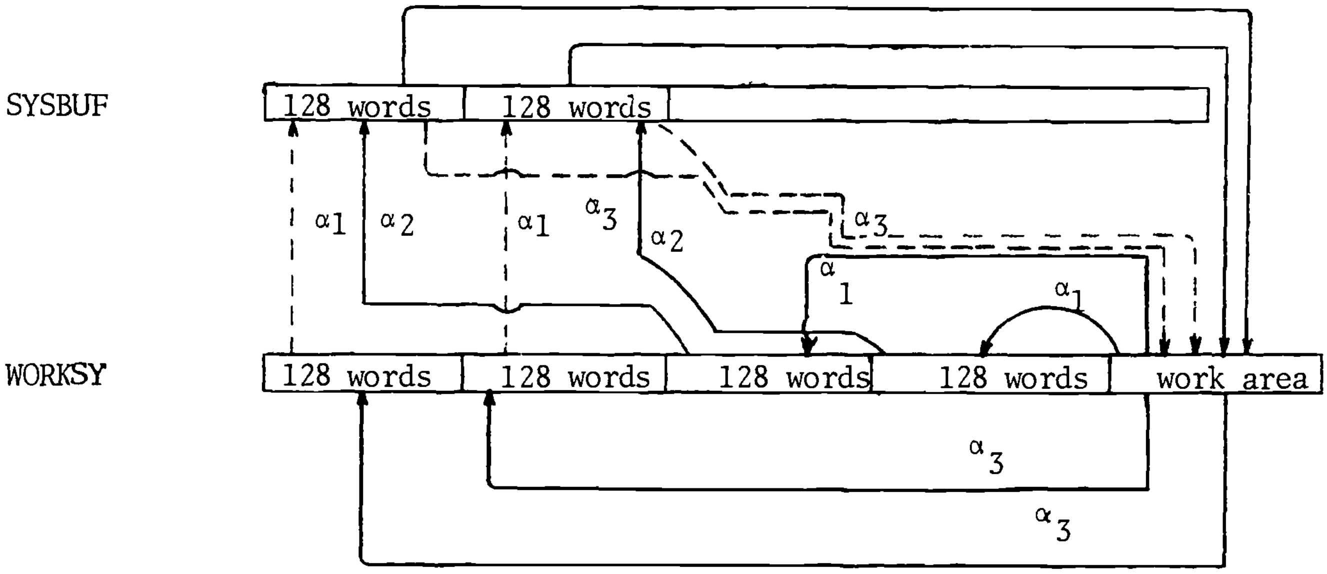


Figure 3

The first picture displayed is the one in SYSBUF. The dotted lines show what happens when the second picture is displayed. The lines show what happens when the third picture is displayed.

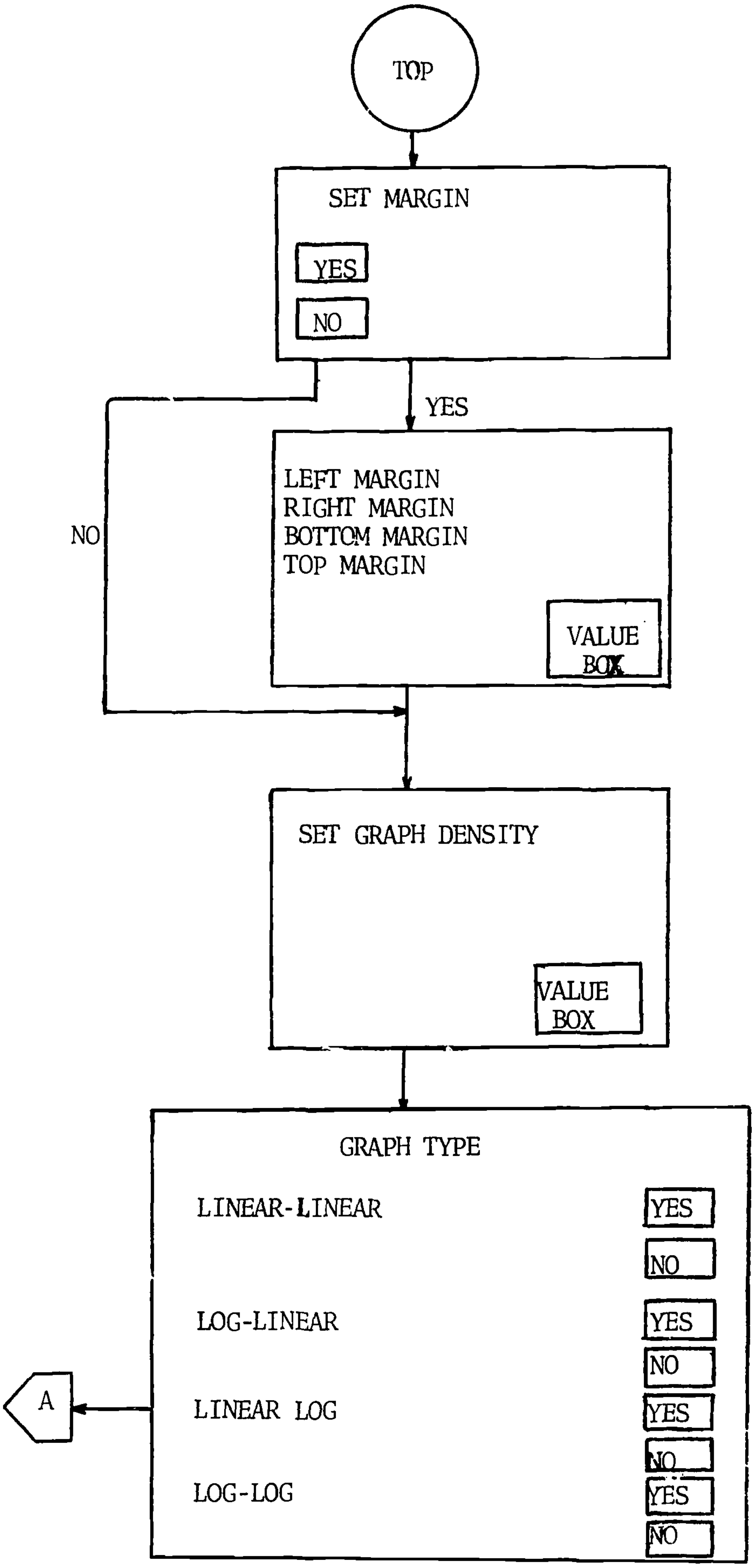
In Figure 2 the user flags the boxes that appear on the CRT on the Rand Tablet to accomplished certain yes-no task. A flag is defined as a close of the stylus switch followed by a release. To input parameters when they are asked for by the system via the Rand Tablet a user should place the pen on the Rand Tablet depress it and move it to the right horizontally watching the CRT for the value box. When the value box is set to the desired value then raise the pen.

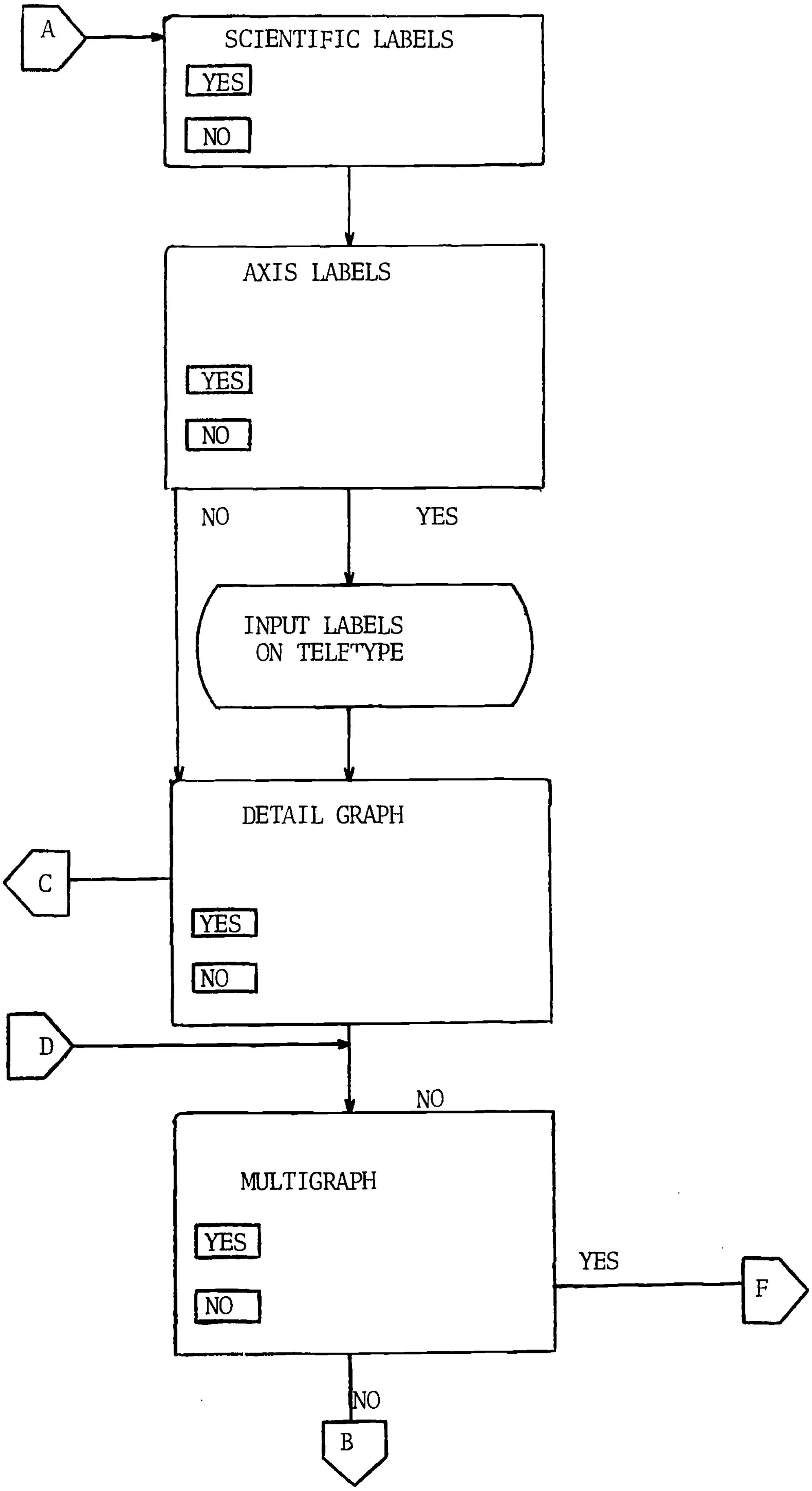
21. Create statement

<Create statement>: = CREA

21.1. Semantics

This command must be given before the Disp command in 20. When CREA is realized certain flags are set to prevent the user from doing anything but DISP. This is done to let the user think a fraction of a second before he gives the DISP command. The CREATE flags can be turned off without giving the DISP command.


TOP
SET MARGIN
YES
NO
YES
NO
LEFT MARGIN
RIGHT MARGIN
BOTTOM MARGIN
TOP MARGIN
VALUE BOX
SET GRAPH DENSITY
VALUE BOX
GRAPH TYPE
LINEAR-LINEAR
YES
NO
LOG-LINEAR
YES
NO
LINEAR LOG
YES
NO
LOG-LOG
YES
NO
A


A
SCIENTIFIC LABELS
YES
NO
AXIS LABELS
YES
NO
NO
YES
INPUT LABELS ON TELE"YPE
DETAIL GRAPH
C
YES
NO
D
NO
MULTIGRAPH
YES
NO
YES
F
NO
B

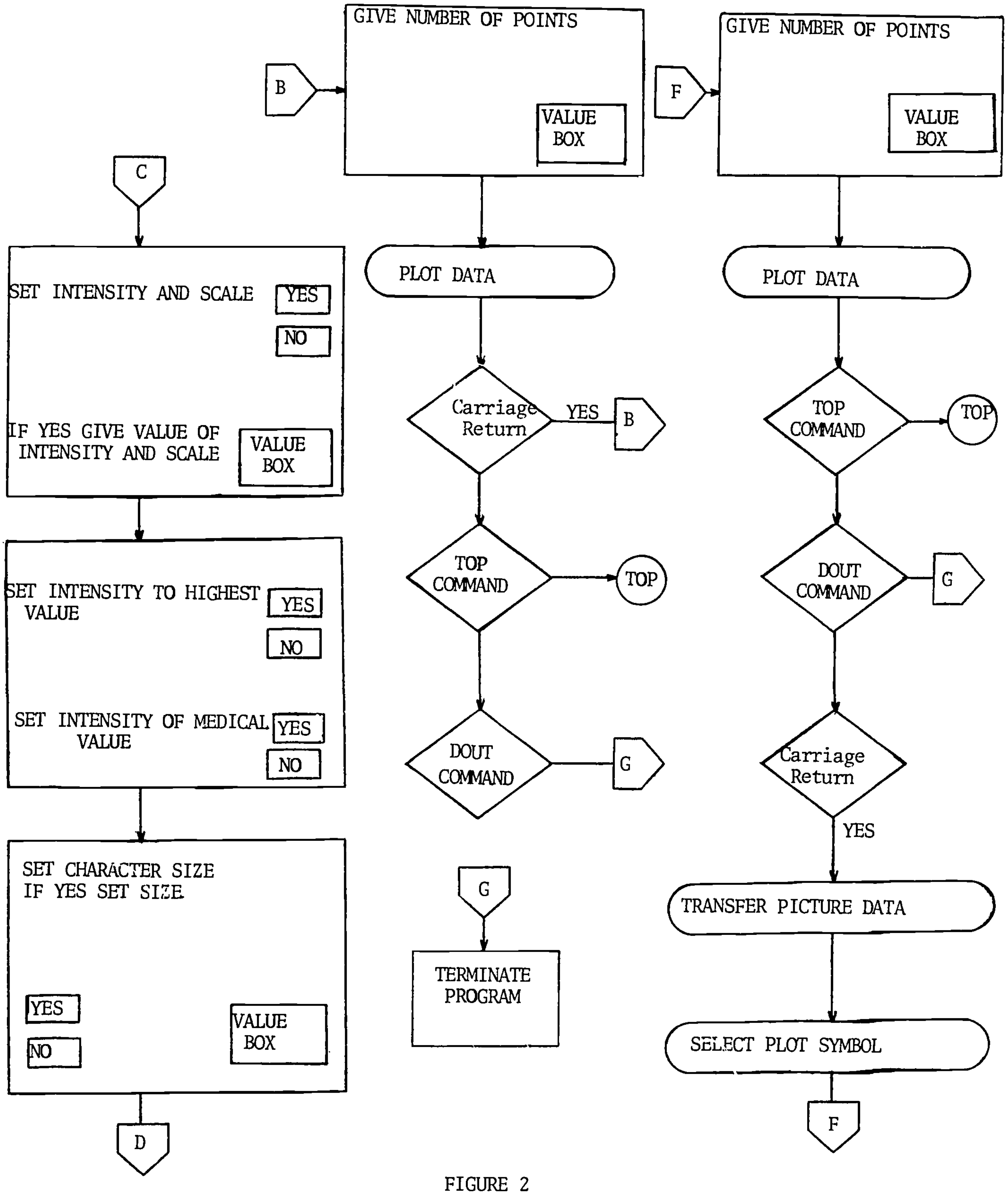

B
GIVE NUMBER OF POINTS
VALUE BOX
PLOT DATA
Carriage Return
YES
B
TOP COMMAND
TOP
DOUT COMMAND
G
G
TERMINATE PROGRAM
F
GIVE NUMBER OF POINTS
VALUE BOX
PLOT DATA
TOP COMMAND
TOP
DOUT COMMAND
G
Carriage Return
YES
TRANSFER PICTURE DATA
SELECT PLOT SYMBOL
F
C
SET INTENSITY AND SCALE
YES
NO
IF YES GIVE VALUE OF INTENSITY AND SCALE
VALUE BOX
SET INTENSITY TO HIGHEST VALUE
YES
NO
SET INTENSITY OF MEDICAL VALUE
YES
NO
SET CHARACTER SIZE
IF YES SET SIZE
YES
NO
VALUE BOX
D

FIGURE 2

22. DOUT statement

<DOUT statement>: = DOUT

22.1. Semantics

The DOUT statement is used to return to the OLMS from the OLMGS. A DOUT given at the level of the OLMS will turn off the CREATE flags.

23. Top statement

<Top statement>: = T

23.1. Semantics

This statement allows a return to the beginning of an interactive process.

24. Core statement

<Core statement>: = 1

24.1. Semantics

The core statement tells the OLMS to allocate the maximum amount of core available. If no core is available the appropriate message will be typed and the system will return for another command.

25. Transfer statement

<Transfer statement>: = TRANS $\{\langle\text{Blank}\rangle|,\}_1^1$

$\{1|2\}_1^1$

25.1. Semantics

The transfer statement is defined on the system buffer and a work area. The work area is a named common block called WORKSY which is seven hundred words long. The OLMGS uses this area when multiple pictures are being displayed on one graph. If multiple pictures are to be displayed by the OLMGS, it is necessary to place the picture data in the buffers at the OLMS level. The OLMGS is defined on the first 256 words of system buffer. The first 128 words are the X-values and the second 128 words are the Y-values. The buffers in figure 1 shows how the transfer command works.

Suppose we have done a GETF of a file of data. Now the file of data is in the system buffer. Let us call this file α_1. Give a transfer to put α_1 in the work buffer.

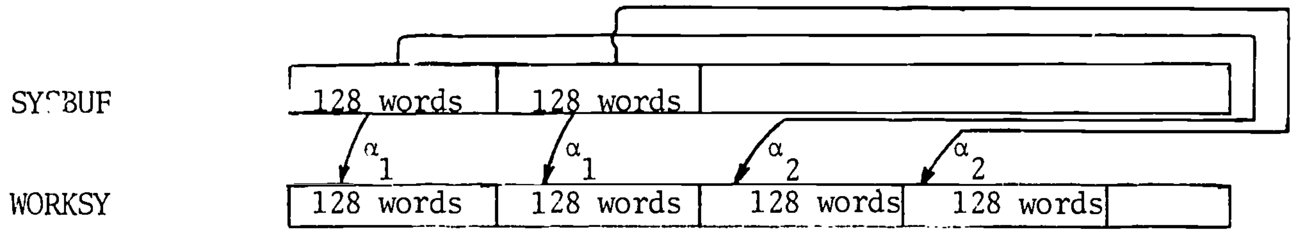


Figure 1

TRANS, 1

Now do a GETF of a file called α_2. Give another transfer.

TRANS, 2

We have two pictures data sets stored in WORKSY. Give another GETF and call the file data α_3. We have α_3, α_1, α_2 as the first, second and third picture sets respectively.

26. Generation statement

<Generation statement>: = GENET {<Model statement>}$_1^1$

26.1. Semantics

Every model will require a specific kind of input. To meet this need the generation statement has been implemented to operate as the START and PROCESS statements. This command is discussed in detail in section three of this paper.

27. File

<File>: = <Name>{{. <EXT>}$_0^1$ {<Blank>|,}$_1^1$

{<Device name>}$_1^1$}$_1^1$

27.1. Semantics

The extension may be omitted.

28. Device Name

<Device Name>: = MTA{<Unit num>}$_1^1$|DTA

{<Unit number>}$_1^1$ |{DSK{<Blank>}$_1^1$}$_1^1$

28.1. Semantics

These are the only devices on most systems that users may access.

29. Property statement

<Property statement>: = MTA|DTA|DSK

29.1. Semantics

The properties MTA, DTA, DSK are magnetic tape, Dectape and Disk respectively.

30. Directory statement

<Directory statement>: = $\{L|F\}_1^1$

31.1. Semantics

L and F refers to the system library directory respectively.

31. File Name

<File Name>: = <Name>$\{.\text{<EXT>}\}_0^1\{\text{<Blank>}|,\}_1^1$

<Property statement>$\}_1^1$

31.1. Semantics

The extension may be omitted.

32. List statement

<List statement>: = LIST$\{\text{<Blank>}|,\}_1^1\{A|S|M|D|T|G|LA|FA\}_1^1|\{L|F\}_1^1\{\text{<Blank>}|,\}_1^1$

<File Name>$\}_0^1$

32.1. Semantics

This command will type out the system tables.

The terminals are defined below.

A	List analysis models
S	List simulation models
M	List system mode
D	List devices assigned to system
T	List magnetic tape directory
G	List generation names
LA	List the whole system library directory
FA	List the whole system file directory
L	List a file name, extension, property and description from the system library directory
F	List a file name, extension, property and description from the system file directory

33. Mode statement

<Mode statement>: = MODE{<Analysis statement>|<Simulation statement>|

<Function Key-light pen statement>|<Expand statement>|<Edit statement>

<Null mode>|<Build statement>|<Generate statement>}$_1^1$

33.1. Semantics

While designing the OLMS command language considerable thought was given to forcing the user to do what he wanted to do, but at the same time giving him considerable time to think and change his mind. To accomplish this task the language is designed with eight operation modes. In our implementation any mode can be entered from another mode.

34. Analysis statement

<Analysis statement>: = {<Blank>|,}$_1^1$ ANAL

34.1. Semantics

The commands available in this mode are listed below.

<Mode statement>: = MODE{<Analysis statement>

|<Simulation statement>|<Function Key-light pen statement>|<Expand statement>

|<Edit statement>|<Null mode>|<Build statement>|<Generate statement>}$_1^1$

<List statement>: = LIST{<Blank>|,}$_1^1$

{A|S|M|D|T|G|LA|FA}$_1^1$|{<Blank>|,}$_1^1$

{<File name>}$_1^1$}$_0^1$

<Save statement>: = SAVE{<Blank>|,}$_1^1$

{<Directory statement>}$_1^1${<Blank>|,}$_1^1$

{<File>}$_1^1$

<Get statement>: = GETF{<Blank>|,}$_1^1$

{<Directory statement>}$_1^1${<Blank|,}$_1^1$

{<File>}$_1^1$

<Delete statement>: = DELE {<Blank>|,}$_1^1$

{<Directory statement>}$_1^1${<Blank>|,}$_1^1$

{<File>}$_1^1$

<Rename statement>:=RENA{<Blank>|,}$^{1}_{1}$

{<Directory statement>}$^{1}_{1}${<Blank>|,}$^{1}_{1}$

{<File>}$^{1}_{1}${<Blank>|,}$^{1}_{1}$TO{<Blank>|,}$^{1}_{1}${<Directory statement>}$^{1}_{1}${<Blank>|,}$^{1}_{1}$

{<File>}$^{1}_{1}$

<Description statement>:=DSCR

{<Blank>|,}$^{1}_{1}${<Directory statement>}$^{1}_{1}$

{<Blank>|,}$^{1}_{1}$ {<File>}$^{1}_{1}$

<Help statement>:=HELP

<Disp statement>:=DISP

<Create statement>:=CREA

<DOUT statement>:=DOUT

<Transfer statement>:=TRANS{<Blank>|,}$^{1}_{1}$ {1|2}$^{1}_{1}$

<Process statement>:=PROCE{<Blank>|,}$^{1}_{1}$

{<Model statement>}$^{1}_{1}${<Core statement>}$^{1}_{0}$

35. Simulation statement

<Simulation statement>:={<Blank>|,}$^{1}_{1}$SIMM

35.1. Semantics

The commands available in this mode are listed below.

<Start statement>:=START{<Model statement>}$^{1}_{1}$

{<Core statement>}$^{1}_{0}$

<Mode statement>:=MODE{<Analysis statement>|

<Simulation statement>|<Function Key-light pen statement>|<Expand statement>|

<Edit statement>|<Null mode>|<Build statement>|<Generate statement>}$^{1}_{1}$

<List statement>:=LIST{<Blank>|,}$^{1}_{1}${A|S|M|D|T|G|LA|FA}$^{1}_{1}${<Blank>|,}$^{1}_{1}$

{<File Name>}$^{1}_{1}$}$^{1}_{0}$

36. Build statement

<Build statement>:={<Blank>|,}$^{1}_{1}$BUIL

36.1. Semantics

When the build made is entered the system initilization flag is set to initialize the system in null mode when it is reloaded. The system exits to the computer monitor.

37. Null Mode

<Null mode>: = {<Blank>|,}$_1^1$ Null

37.1. Semantics

The commands available in this mode are listed below.

<Mode statement>: = <MODE{<Analysis statement>|<Simulation statement>|<

Function Key-light pen statement>|<Expand statement>|<Edit statement>

|<Null Mode>|<Build statement>|<Generate statement>}$_1^1$

<List statement>: = LIST{<Blank>|,}$_1^1${A|S|M|D|T|G|LA|FA}$_1^1$|{L|F}$_1^1${<Blank>|,}$_1^1$

{<File name>}$_1^1$}$_0^1$

<System Directory statement>: = NEWD{<Blank>|,}$_1^1${<Device name>}$_1^1$

<Directory input statement>: = DIRM

{<Blank>|,}$_1^1${<Device name>}$_1^1${{<Blank>|,}$_1^1$

{<Directory statement>}$_1^1$}$_0^1$

<Clear statement>: = CLEA{<Blank>|,}$_1^1$

{<Device name>}$_1^1$

<Plant statement>: = PLAN{<Blank>|,}$_1^1$

{<Directory statement>}$_1^1$

{<Blank>|,}$_1^1${<File>}$_1^1$

<Assign statement>: = ASSN{<Blank>|,}$_1^1$

{<Device name>|{CHAN{<Blank>|,}$_1^1$

<Device name>}$_1^1$|{OLMS{<Blank>|,}$_1^1$

<Device name>}$_1^1$}$_1^1$

<Help statement>: = HELP

<End statement>: = ENDS

38. Expand Statement

<Expand statement>: = {<Blank>|,}$_1^1$EXPA

38.1. Semantics

The commands available in this mode are listed below.

<Help statement>: = HELP

<Mode statement>: = MODE{<Analysis statement>|<Simulation statement>|<

Function Key-light pen statement>|<Expand statement>|<Edit statement>

|<Null mode>|<Build statement>|<Generate statement>}$_1^1$

<List statement>: = LIST{<Blank>|,}$_1^1$

{A|S|M|D|T|G|LA|FA}$_1^1$|{L|F}$_1^1${<Blank>|,}$_1^1$

{<File name>}$_1^1$}$_0^1$

<Enter statement>: = ENTER{<Model statement>}$_1^1$

<Kill statement>: =KILLM {<Model statement>}$_1^1$

39. Edit statement

<Edit statement>: = {<Blank>|,}$_1^1$EDIT

39.1. Semantics

Network models will be editted in this mode. Any variable in Block Data which is the memory of the OLMS can be changed.

40. Generate statement

<Generate statement>: = {<Blank>|,}$_1^1$GENN

40.1. Semantics

The commands available in this mode are listed below.

<Help statement>: = HELP

<LIST statement>: = LIST{<Blank>|,}$_1^1${A|S|M|D|T|G|LA

|FA}$_1^1$|{L|F}$_1^1${<Blank>|,}$_1^1${<File name>}$_1^1$}$_0^1$

<Mode statement>: = MODE{<Analysis statement>|<Simulation statement>

|<Function Key-light pen statement>|<Expand statement>|<Edit statement>

|<Null mode>|<Build statement>|<Generate statement>}$_1^1$

<Generation statement>:=GENET{<Model statement>}$_1^1$

41. Function Key-light Pen statement

<Function Key-light pen statement>:={<Blank>|,}$_1^1$FKLP

41.1. Semantics

In this mode certain commands in the command language are assigned to function keys. This mode is discussed in Section III.

42. Unit Number

<Unit Number>:= 0|1|2|3|4|5|6|7

43. Unit Num

<Unit Num>:= 0|1|2

44. EXT (File extension)

<EXT>:=<Null>|<Letter>{<Letter>|<digit>}$_0^2$

45. Help statement

<Help statement>:=HELP

45.1. Semantics

Help is a program that teaches a user how to use the OLMS. The commands available in this program are listed below.

<Top statement>:=T

<Yes-No statement>:={Y|N}$_1^1$

<Dout statement>:=DOUT

46. END statement

<END statement>:=ENDS

46.1. Semantics

This command terminates the system.

47. Yes-No statement

<Yes-No statement>:={Y|N}$_1^1$

SECTION III

GENERATION OF INPUT

The Rand Tablet and Function Keys are excellent input devices for interactive work with interactive systems. To use these devices economically it is necessary to have a flexible system. This flexibility is obtained in the OLMS by allowing any type of INPUT program using the Rand Tablet and Function Keys.

The OLMS has a standard graphical input using the Rand Tablet. The flow chart in figure 1, shows the sequence for inputting a graph and collecting the graphical data input. The large blocks show how the CRT looks at certain levels in the program.

Commands available are listed below.

<Top command>:=T

<Dout command>:=DOUT

<Transfer command>:=TRANS{<Blank>|,}$_1^1${1|2}$_1^1$

A carriage return will return to the data bounds input level.

This implementation allows a user to collect 128 X-values and 128 Y-values.

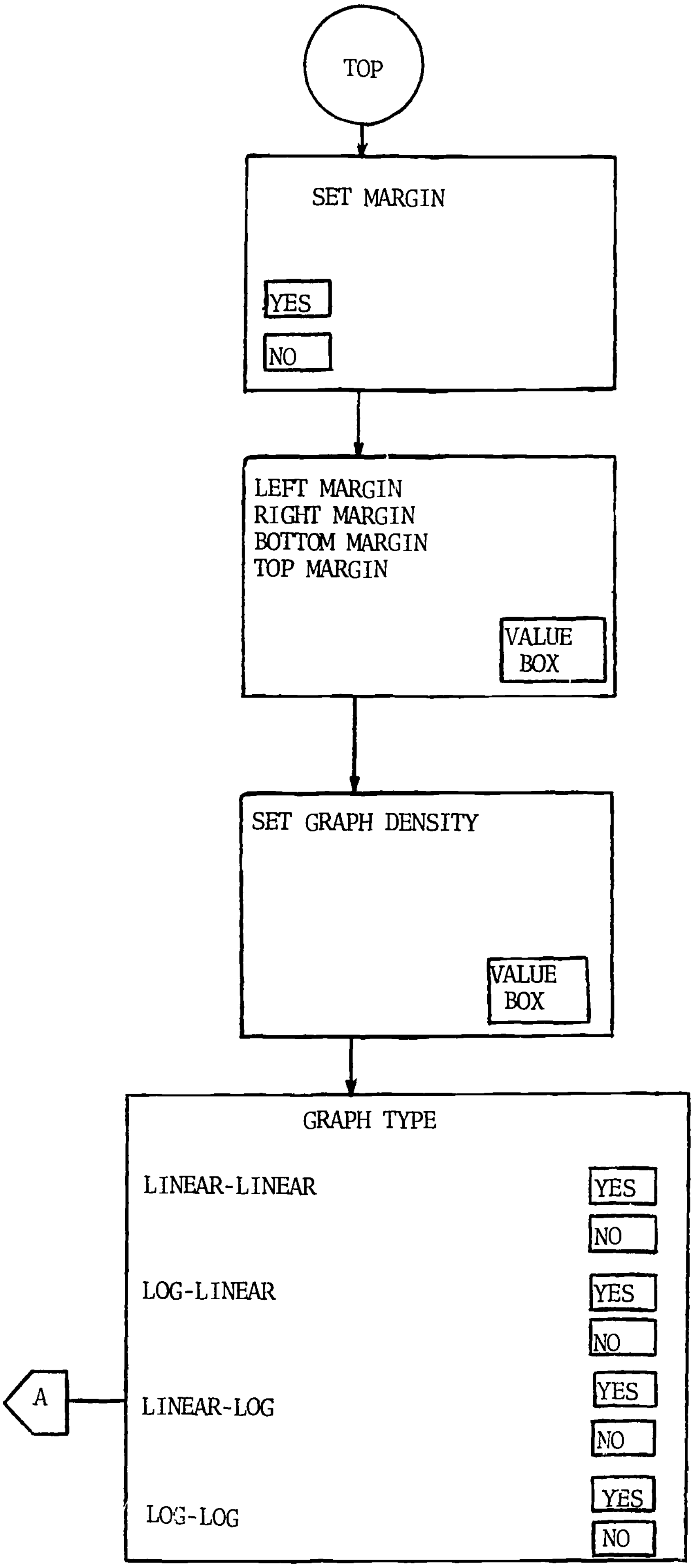
FUNCTION KEY LIGHT PEN MODE

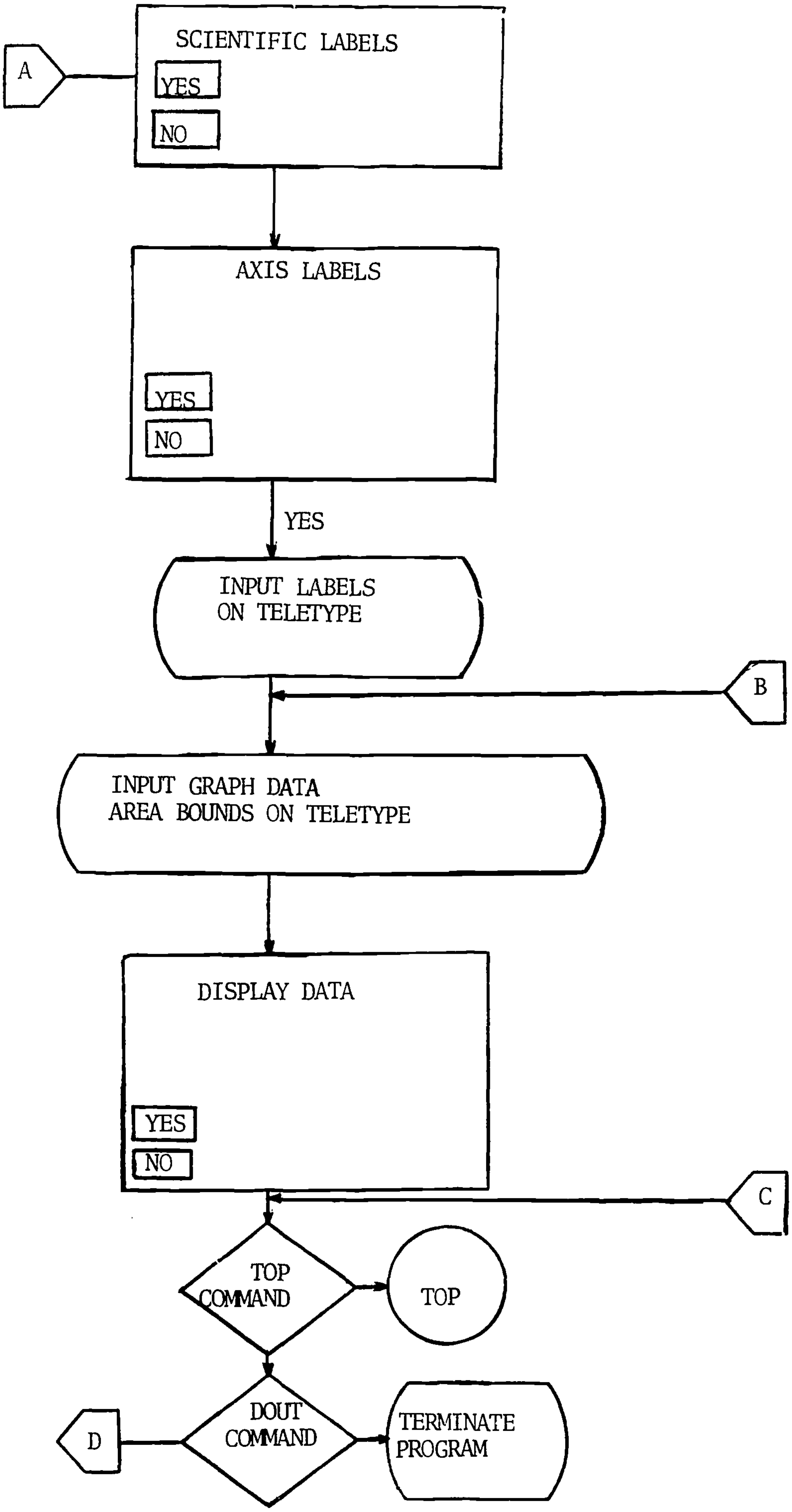
In this mode various commands of the Command Language are assigned to Function Keys. The assignments are listed below.

COMMAND	KEY ASSIGNMENT
SAVE	1
MODE	2
DISP	3
GETF	4
DELE	5
TRANS	6
START	7
CLEA	8
RENA	9

COMMAND	KEY ASSIGNMENT
DSCR	10
PLAN	11
ASSN	12
DEAS	13
PROCE	14
GENET	15
DOUT	16
KILLM	16
ENTER	16

This mode is operated by depressing a key associated with a particular command and following the program sequence. Whenever a decision is needed a small box will appear adjacent to the statement on the CRT. The Rand Tablet is used to flag these boxes. If data input is requested on the teletype the CONTINUE will be typed before the system is ready for input. The program sequence is listed in figure 2. The large blocks show the appearance of the CRT at a particular level in the program.


TOP
SET MARGIN
YES
NO
LEFT MARGIN
RIGHT MARGIN
BOTTOM MARGIN
TOP MARGIN
VALUE BOX
SET GRAPH DENSITY
VALUE BOX
GRAPH TYPE
LINEAR-LINEAR
YES
NO
LOG-LINEAR
YES
NO
A
LINEAR-LOG
YES
NO
LOG-LOG
YES
NO


A
SCIENTIFIC LABELS
YES
NO
AXIS LABELS
YES
NO
YES
INPUT LABELS
ON TELETYPE
B
INPUT GRAPH DATA
AREA BOUNDS ON TELETYPE
DISPLAY DATA
YES
NO
C
TOP
COMMAND
TOP
D
DOUT
COMMAND
TERMINATE
PROGRAM

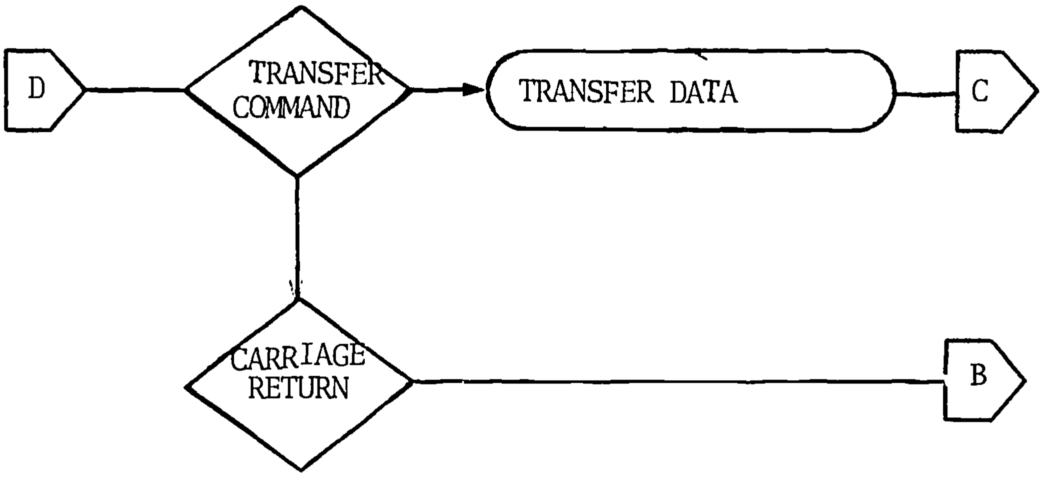
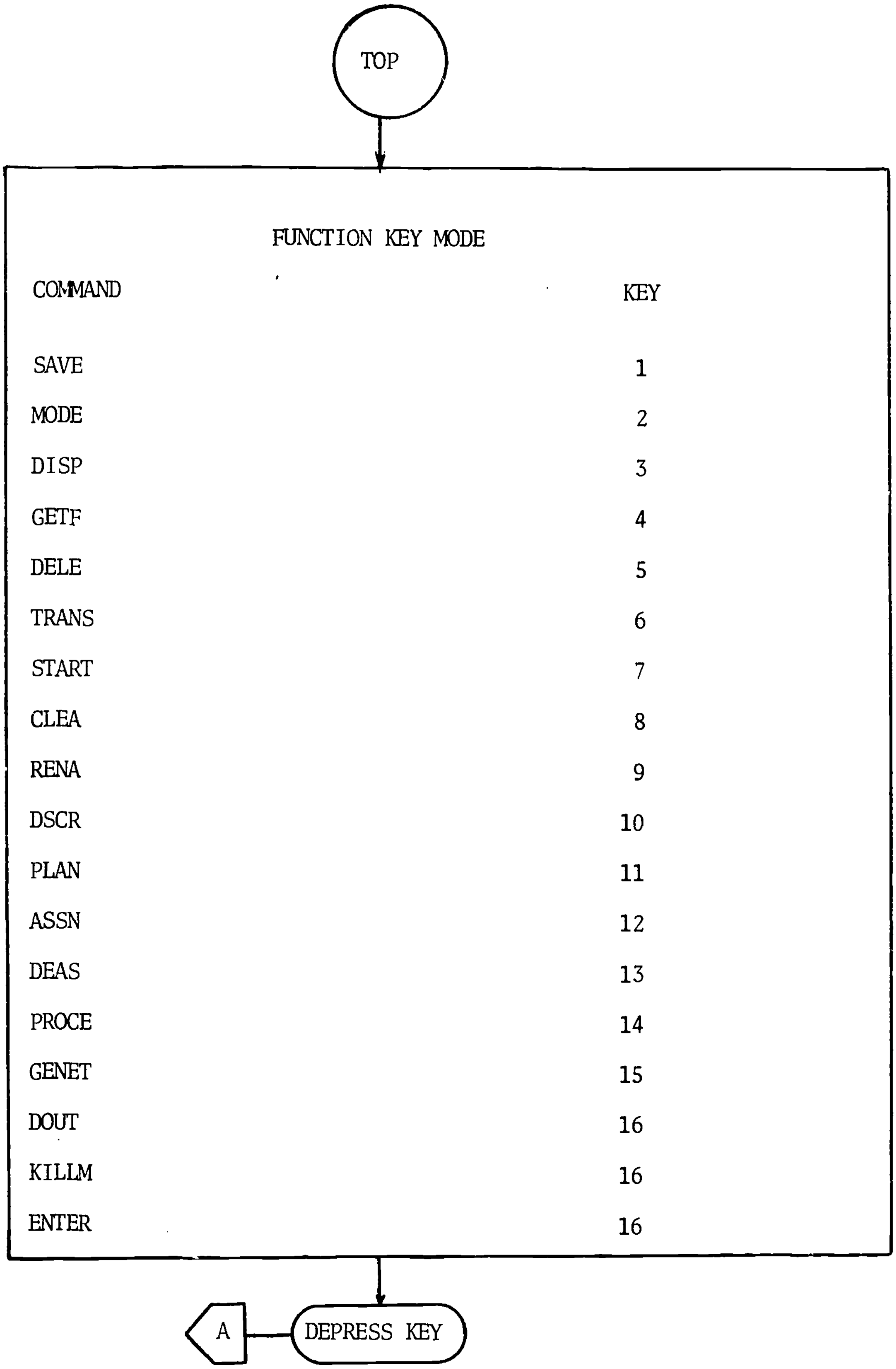


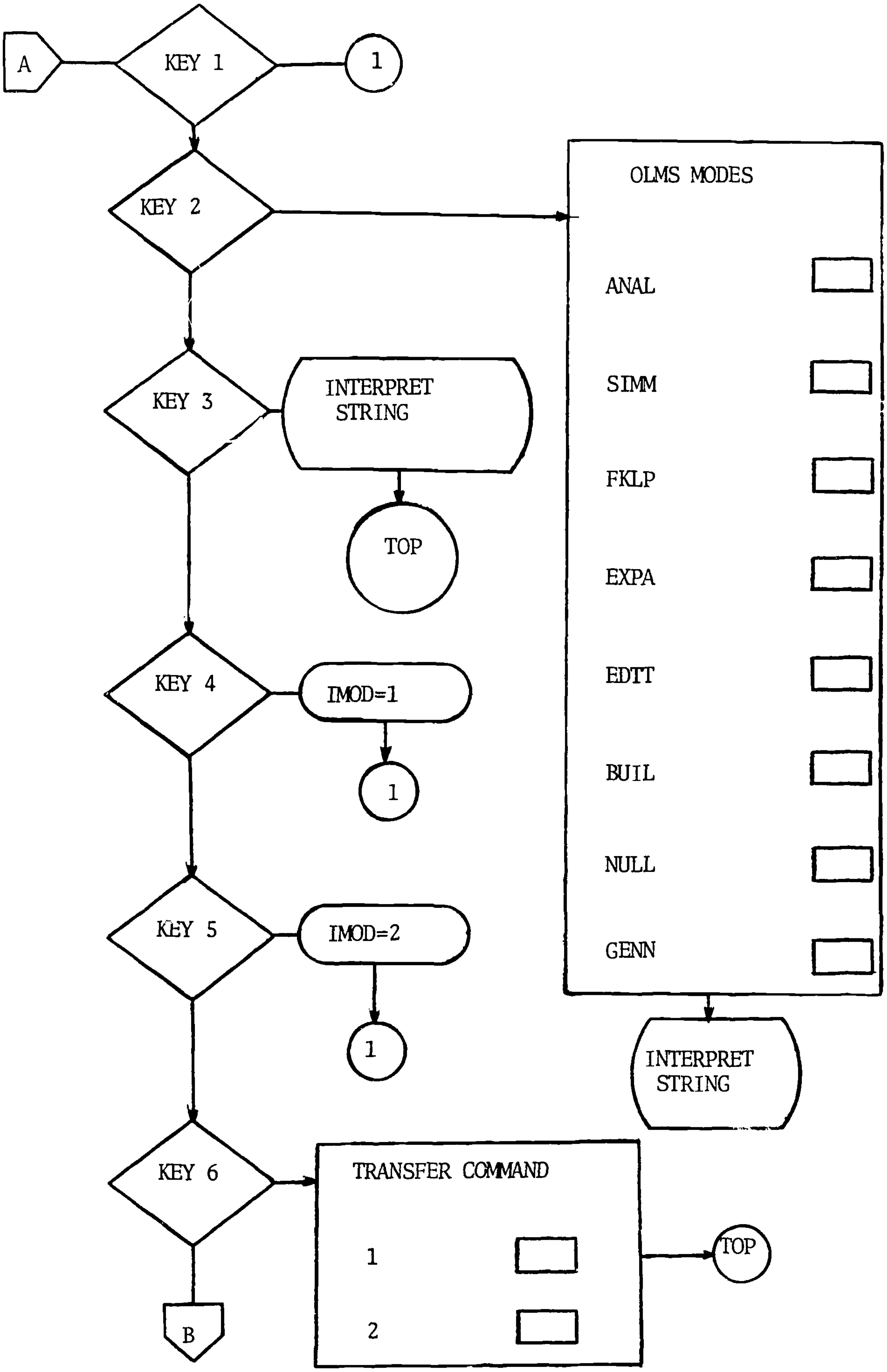
FIGURE 1

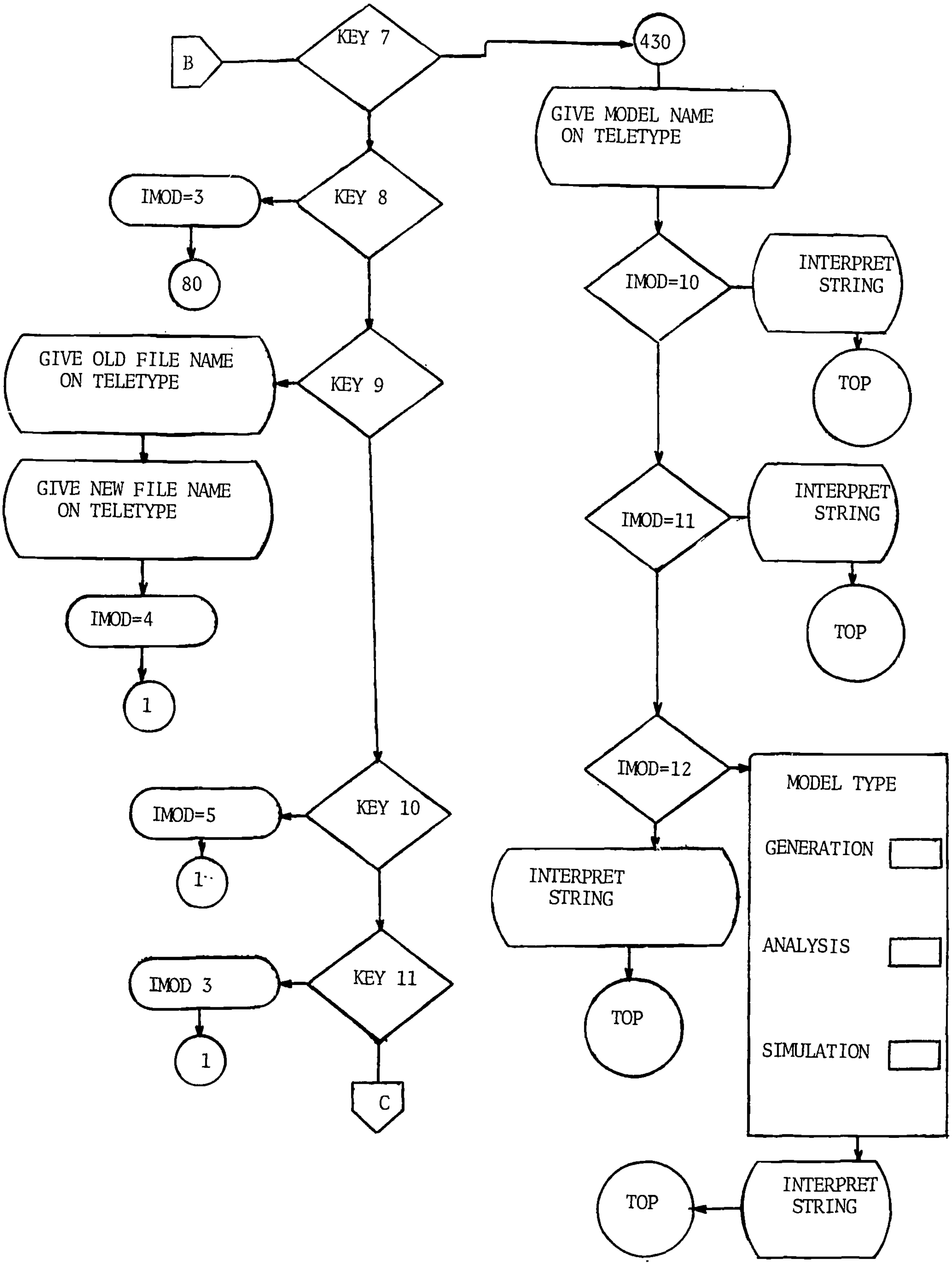


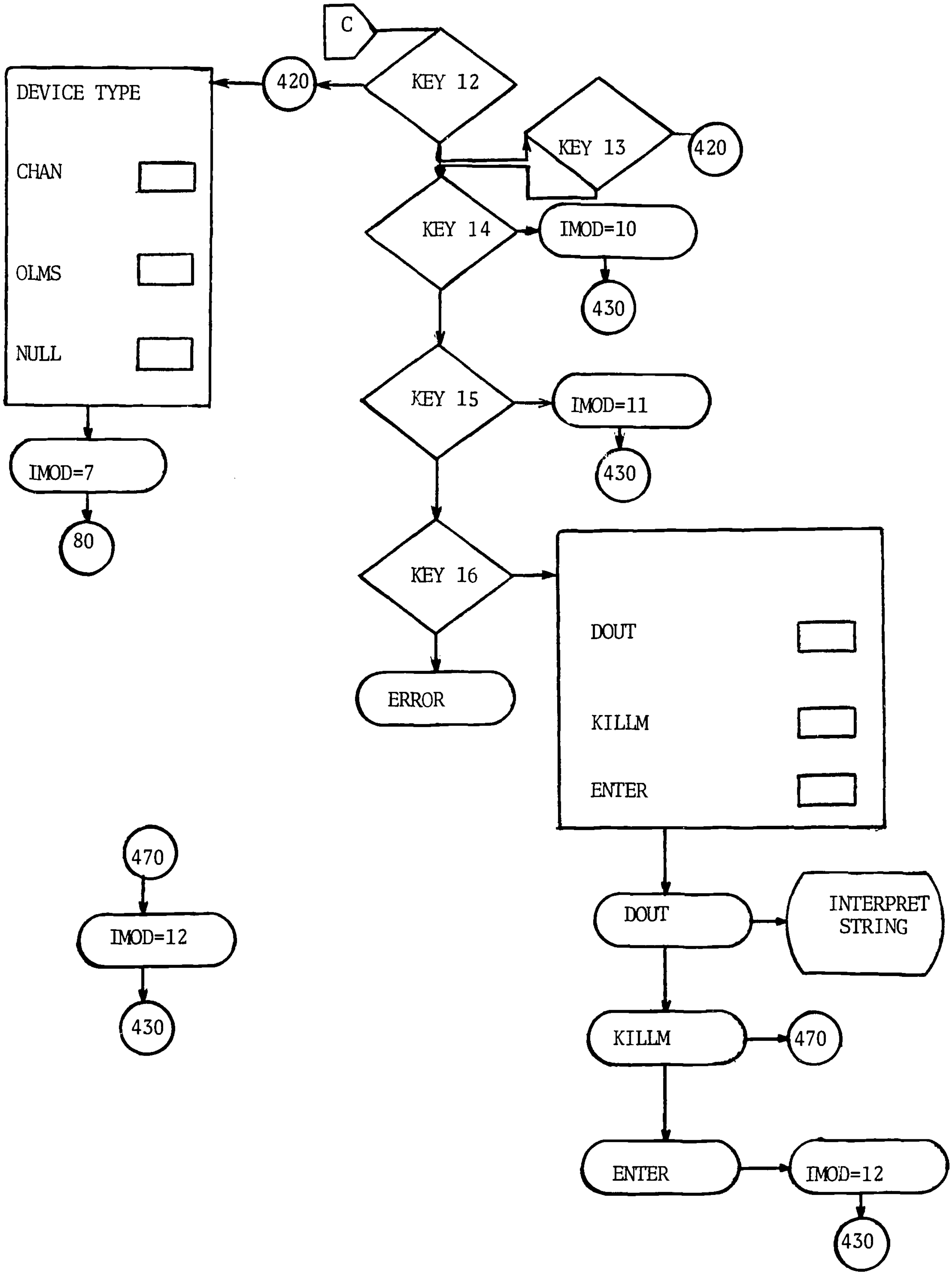
TOP

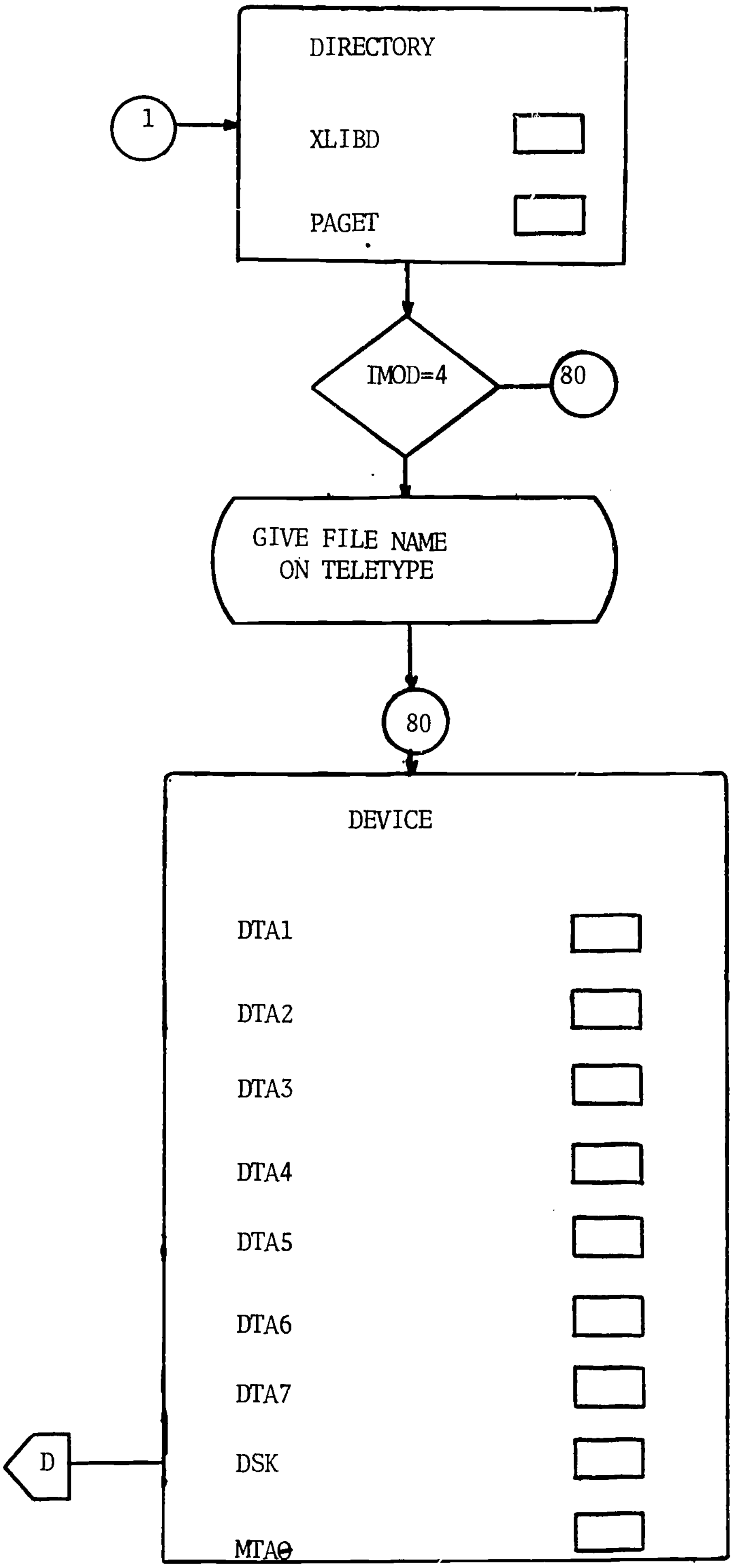
FUNCTION KEY MODE

COMMAND	KEY
SAVE	1
MODE	2
DISP	3
GETF	4
DELE	5
TRANS	6
START	7
CLEA	8
RENA	9
DSCR	10
PLAN	11
ASSN	12
DEAS	13
PROCE	14
GENET	15
DOUT	16
KILLM	16
ENTER	16


A
KEY 1
1
KEY 2
OLMS MODES
ANAL
SIMM
FKLP
EXPA
EDTT
BUIL
NULL
GENN
KEY 3
INTERPRET STRING
TOP
KEY 4
IMOD=1
1
KEY 5
IMOD=2
1
INTERPRET STRING
KEY 6
TRANSFER COMMAND
1
2
TOP
B


B
KEY 7
430
GIVE MODEL NAME ON TELETYPE
IMOD=3
KEY 8
80
IMOD=10
INTERPRET STRING
TOP
GIVE OLD FILE NAME ON TELETYPE
KEY 9
GIVE NEW FILE NAME ON TELETYPE
IMOD=11
INTERPRET STRING
TOP
IMOD=4
1
IMOD=12
MODEL TYPE
GENERATION □
ANALYSIS □
SIMULATION □
IMOD=5
KEY 10
1
INTERPRET STRING
IMOD 3
KEY 11
TOP
1
C
TOP
INTERPRET STRING


C
KEY 12
420
DEVICE TYPE
CHAN
OLMS
NULL
IMOD=7
80
KEY 13
420
KEY 14
IMOD=10
430
KEY 15
IMOD=11
430
KEY 16
ERROR
DOUT
KILLM
ENTER
470
IMOD=12
430
DOUT
INTERPRET STRING
KILLM
470
ENTER
IMOD=12
430


DIRECTORY
1
XLIBD
PAGET
IMOD=4
80
GIVE FILE NAME ON TELETYPE
80
DEVICE
DTA1
DTA2
DTA3
DTA4
DTA5
DTA6
DTA7
D
DSK
MTAØ

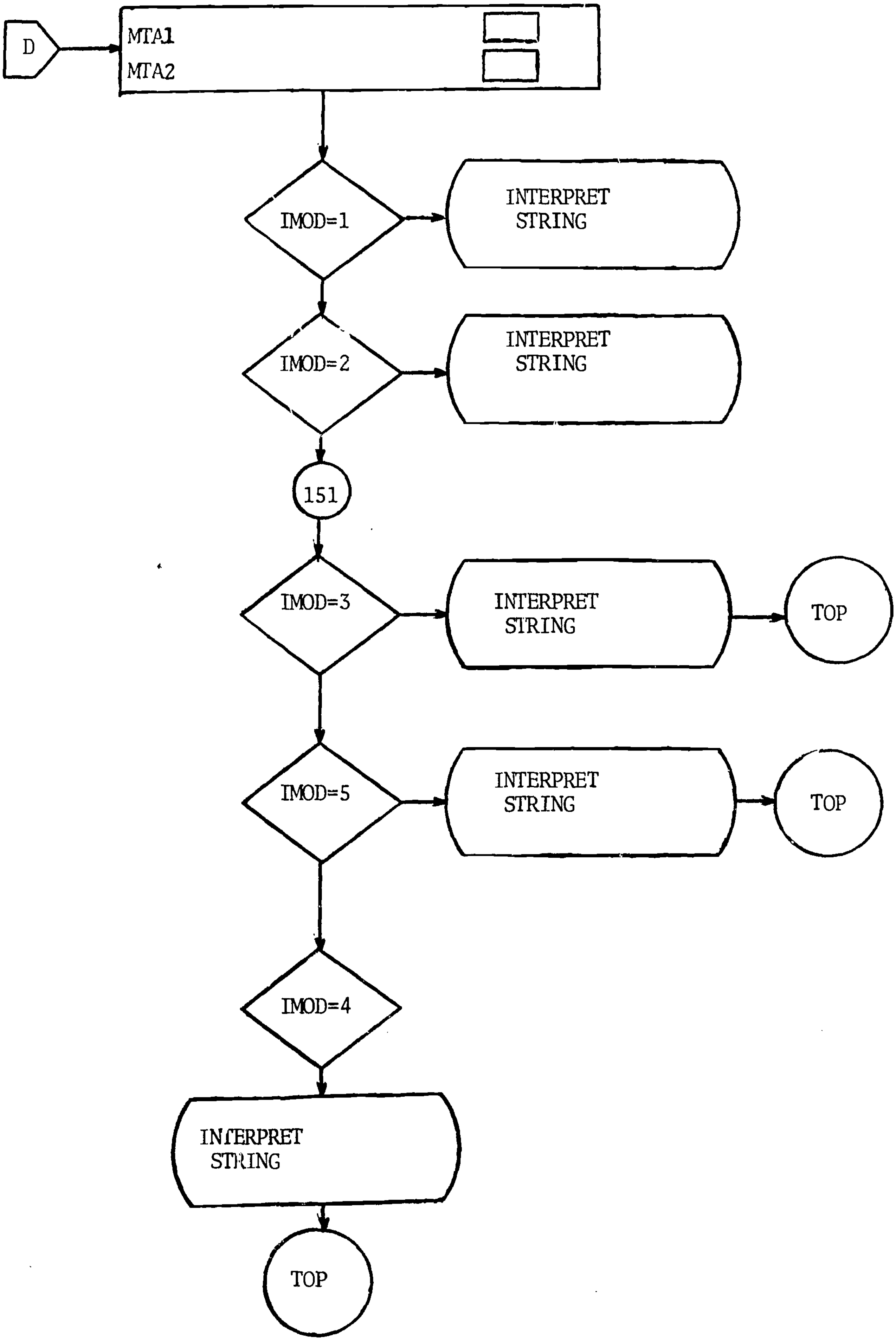

D
MTA1
MTA2
IMOD=1
INTERPRET STRING
IMOD=2
INTERPRET STRING
151
IMOD=3
INTERPRET STRING
TOP
IMOD=5
INTERPRET STRING
TOP
IMOD=4
INTERPRET STRING
TOP

FIGURE 2

SECTION IV

4.1 Example

This example shows how the OLMS can be used to study the input output relations of the Hodgkin-Huxley Nerve Equations. The differential equations are solved using the Predictor-Corrector Method with the Runge-Kutta Method as a starter.

The equations are given below under the heading Hodgkin-Huxley Equations. Any details about these equations can be found in references [29] and [28].

Hodgkin-Huxley Equations

(1) $I=C\dot{V}+\bar{g}_{Na}m^3h(V+115)+\bar{g}_K n^4(V-12)+\bar{g}_L(V+10.5989)$

(2) $\dot{m}=\phi[(1-m)\alpha_m(V)-m\beta_m(V)]$,

(3) $\dot{h}=\phi[(1-h)\alpha_h(V)-h\beta_h(V)]$,

(4) $\dot{n}=\phi[(1-n)\alpha_n(V)-n\beta_n(V)]$.

where

$\alpha_m(V)=0.1(V+25)[\exp(\frac{V+25}{10})-1]^{-1}$,

$\beta_m(V)=4\exp(V/18)$,

$\alpha_h(V)=0.07\exp(V/20)$,

$\beta_h(V)=[\exp(\frac{V+30}{10})+1]^{-1}$,

$\alpha_n(V)=0.01(V+10)[\exp(\frac{V+10}{10})-1]^{-1}$,

$\beta_n(V)=0.125\exp(V/80)$, and

$\phi=3^{(T-6.3)/10}$

The Predictor-Corrector used is given as follows:

Predictor

Given m points

$x_{n-m+1}.......,x_{n-1},x_n$

$f_{n-m+1}.......,f_{n-1},f_n$ and Newton's Integrating Newton's backward interpolation formula we have:

$$y_{n+1} = y_n+h \sum_{j=0}^{m-1} \beta_j \Delta^j f_{n-j}$$

where

$\beta_0=1,\beta_j=(-1)^j \int_0^1 ({-z \atop j})dz,$

z variable at x_n is $z=\frac{x-x_n}{h}$ and

$j=1,2,.......,m-1.$

Corrector

Given:

$x_{n-m+1},.......,x_n,x_{n+1}$

$f_{n-m+1},.......,f_n,f_{n+1}$ and Newton's Integrating Newton's backward interpolation formula we have:

$$y_{n+1}=y_n+h \sum_{j=0}^{m-1} \beta_j^* \Delta^j f_{n+1-j}$$

where

$\beta_0^*=1,\beta_j^*=(-1)^j \int_{-1}^0 ({-z \atop j})dz,$

z variable at x_n is $z=\frac{x-x_n}{h}$ and

$j=1,2,......,m-1$

This model is given in algorithmic form and a program listing is given for the solution of the model. This model can be extended by modifying the algorithm and using two dimensional arrays to solve the partical differential equations in reference [28]. Someone may claim the algorithm uses too much storage. An algorithm can be developed that checks for vector overflow. If the vectors are full the data in the vectors from the beginning up to the total length minus three can be outputted. The three data points can be moved to the beginning of the vectors. By adjusting the appropriate pointers the algorithm will think it is starting. It is not necessary to move the data if linked data structures are used. The stepsize can be changed by creating an algorithm that starts the Runge-Kutta when some condition occurs, perhaps when the Predictor-Corrector fails to converge. This can be done by backing up one step and starting the Runge-Kutta with the new stepsize for some number of points greater than or equal to three in our case. Adjust the appropriate pointers and give control to the Predictor-Corrector.

The Model in Algorithmic Form

In thealgorithms that follow it is assumed that the relationship of functions and subroutines exist. There are two classes of operators called Operator and Parameter and Transfer Operator. The Operator and Parameter class is used to give input to the algorithms. The Transfer Operator is used to transfer data to the OLMS system buffer. All operators are defined below:

Operator and Parameter Class.

CAP - membrane capacitance

UCON - turn off all variables held constant

DX - distance stepsize

COLL - data collection stepsize used to transfer data to the OLMS system buffer.

RUN - start algorithm

TOWM - T_m

TOWH - T_h

TOWN - T_n

TEMP - temperature (^{o}C.)

M - sodium activation

H - sodium inactivation

N - potassium activation

T - initial time

V - initial voltage

CUR - stimulating current pulse

DUR - duration of current pulse

SZ - integration stepsize

EPS - convergence term

STAR - number of starter values

FT - number of solution points desired

MITE - maximum number of corrector iterations per step

MC - hold sodium activation constant

HC - hold sodium inactivation constant

NC - hold potassium activation constant

Transfer Operator Class

TV - time .vs. voltage

TM - time .vs. sodium activation

TH - time .vs. sodium inactivation

TN - time .vs. potassium activation

VM - voltage .vs. sodium activation

VH - voltage .vs. sodium inactivation

VN - voltage .vs. potassium activation

DI - distance .vs. current

TI - time .vs. current

The global variables in the algorithms are defined as follows:

H - integration stepsize

EPS - convergence term

TEMP - temperature

NC - total number of solution points desired

N - number of starter values

MAX - maximum number of corrector iterations per step

PHI - ϕ for sodium activation

PHH - ϕ for sodium inactivation

PHN - ϕ for potassium activation

ICO - counter

DX - distance stepsize

ARAD - radius of axon

RR - specific resistance of axoplasm

SIS - amplitude of the stimulating current

CURR - current

DIST - distance along axon

DURRAT - duration of current pulse

CAP - membrane capacitance

CHAR - control vector

YHH - vector os sodium inactivation

YMM - vector of sodium activation

YNN - vector of potassium activation

X - vector of time

SYSBUF - OLMS system buffer

The R-Algorithm is used to input data and transfer data to the OLMS system buffer. The A-Algorithm is the main algorithm that uses the V,FUM,FUH,FUN,P,C, and FN algorithms to integrate the equations.

R-Algorithm

Subroutine RUNMOD

```
R1. [Give Operator and Parameter]
     VAR ←INPUT
     VALUE ←INPUT
R2. [What is the Operator?]
     if VAR.EQ.'CAP' then CAP ←VALUE, go to R1,
     Otherwise
     if VAR.EQ.'UCON' then CHAR(22) ← 0,CHAR(23) ← 0,CHAR(24) ← 0,
     go to R1,
     Otherwise
     if VAR.EQ.'DX' then DX ← VALUE, go to R1,
     Otherwise
     if VAR.EQ.'COLL' then VALCOL ← VALUE, go to R1.
     Otherwise
     KCT ← 0  (initialize control count)
     LOOP:  KG ← 1 to 20  (loop to set KCT)
            KCT ← KCT+1
            if VAR.EQ.CHAR(KCT) then go to LO,
            Otherwise go to LOOP
            Operator Error, go to R1.
       LO:  if KCT.EQ.1 then go to R3,
            Otherwise
            if KCT.EQ.2 then TOWM ← VALUE
            if KCT.EQ.3 then TOWH ← VALUE
```

A-Algorithm

Subroutine ADAMS

```
A1. [Generate Starting Points]
    N ← The number of starting values
    MAX ← The maximum number of iterations per step
    EPS ← The convergence region
    H ← Stepsize
    NC ← Maximum number of steps
    NCMN ← NC-N  The number of steps to use the predictor-corrector
    I ← 0  Initialize the predictor-corrector step counter
    CALL RUNFIT  Generate N starting points
          ICO ← 0
    LOOP: ICO ← ICO+1
          if ICO ≤ N then go to LOP
          Otherwise  ICO ← ICO-1, go to LO.
     LOP: F(ICO) ← VOLT(Y(ICO))
          FMM(ICO) ← FUM(YMM(ICO))
          FHH(ICO) ← FUH(YHH(ICO))
          FNN(ICO) ← FUN(YNN(ICO))
          X(ICO+1)=X(ICO)+H
      LO: if ICO < 3 then ERROR.
A2. [Start the predictor-corrector]
    ICO ← N+I
    Initialize convergence switches
    IVSW ← 0,IMSW ← 0,IHSW ← 0,INSW ← 0
```

R7. [Transfer Time.VS.Sodium Activation]

Same as R6 after replacing Y(I) in A2 by YMM(I) and LOOP1 by LOOP2.

R8. [Transfer Time.VS.Sodium Inactivation]

Same as R6 after replacing Y(I) in A2 by YHH(I) and LOOP1 by LOOP3.

R9. [Transfer Time.VS.Potassium Activation]

Same as R6 after replacing Y(I) in A2 by YNN(I) and LOOP1 by LOOP4.

R10. [Transfer Voltage.VS.Sodium Activation]

Same as R6 after replacing X(I) in A1 by Y(I), Y(I) in A2 by YMM(I) and LOOP1 by LOOP5.

R11. [Transfer Voltage.VS.Sodium Inactivation]

Same as R6 after replacing X(I) in A1 by Y(I), Y(I) in A2 by YHH(I) and LOOP1 by LOOP6.

R12. [Transfer Voltage.VS.Potassium Activation]

Same as R6 after replacing X(I) in A1 by Y(I), Y(I) in A2 by YNN(I) and LOOP1 by LOOP7.

R13. [Transfer Distance.VS.Current]

Same as R6 after replacing X(I) in A1 by DIST(I), Y(I) in A2 by CURR(I) and LOOP1 by LOOP8.

R14. [Transfer Time.VS.Current]

Same as R6 after replacing Y(I) by CURR(I) and LOOP1 by LOOP9.

R15. [Return to the OLMS]

```
R4. [Transfer data to the OLMS system buffer]
     VAR ← INPUT
R5. [Check Transfer Operator]
     VALTEM ← VALCOL
     J ← 1
     if VAR.EQ.'TV' then go to R6
     if VAR.EQ.'TM' then go to R7
     if VAR.EQ.'TH' then go to R8
     if VAR.EQ.'TN' then go to R9
     if VAR.EQ.'VM' then go to R10
     if VAR.EQ.'VH' then go to R11
     if VAR.EQ.'VN' then go to R12
     if VAR.EQ.'DI' then go to R13
     if VAR.EQ.'TI' then go to R14
     Otherwise
     Transfer Operator Error, go to R4
R6. [Transfer Time.VS.Voltage]
     LOOP1: Increment I by 1, if I=500 then finished otherwise,
             if J.GT.128 then go to LOOP1
             if X(I).NE.VALTEM then go to LOOP1

             Otherwise
         A1. SYSBUF(J) ← X(I)
         A2. SYSBUF(J+128) ← Y(I)
             J ← J+1
             VALTEM ← VALTEM+VALCOL
             go to LOOP1, go to R15 (after the loop is complete)
```

```
              if KCT.EQ.4 then TOWN ← VALUE
              if KCT.EQ.5 then YMM(1) ← VALUE
              if KCT.EQ.6 then YHH(1) ← VALUE
              if KCT.EQ.7 then YHH(1) ← VALUE
              if KCT.EQ.8 then YNN(1) ← VALUE
              if KCT.EQ.9 then X(1) ← VALUE
              if KCT.EQ.10 then Y(1) ← VALUE
              if KCT.EQ.11 then SIS ← VALUE
              if KCT.EQ.12 then DURRAT ← VALUE
              if KCT.EQ.13 then H ← VALUE
              if KCT.EQ.14 then EPS ← VALUE
              if KCT.EQ.15 then N ← VALUE
              if KCT.EQ.16 then NC ← VALUE
              if KCT.EQ.17 then MAX ← VALUE
              if KCT.EQ.18 then CHAR(22) ← CHAR(21)
              if KCT.EQ.19 then CHAR(23) ← CHAR(21)
              if KCT.EQ.20 then CHAR(24) ← CHAR(21)
                go to R1.
R3. [Start Model]
      PHI ← DEXP(.1098612289D0*(TEMP-6.3D0))
      PHH ← PHI
      PHN ← PHI
      if TOWM.NE.1 then PHI ← PHI/TOWM
      if TOWH.NE.1 then PHH ← PHH/TOWH
      if TOWN.NE.1 then PHN ← PHN/TOWN
        CALL ADAMS  (start integration)
```

```
    Check for constant conditions
      if CHAR(22).NE.0.0 then IMSW ← 1.
      if CHAR(23).NE.0.0 then IHMS ← 1
      if CHAR(24).NE.0.0 then INSW ← 1
    Initialize corrector iteration counter
            M ← 0
            NPI ← N+I
A3. [Compute $f_{N+I} = f(x_{N+I}, Y_{N+I})$, etc.]
    F(N+I) ← VOLT(Y(N+I))
    Cehck for constant conditions
    if CHAR(22).NE.0.0 then go to B
    Otherwise, ICO ← ICO-1
       FMM(N+I) ← FUM(YMM(N+I))
       ICO ← ICO+1
    Check for constant conditions
    O: if CHAR(23).NE.0.0 then go to G
    Otherwise, ICO ← ICO-1
       FHH(N+I) ← FUH(YHH(N+I))
       ICO ← ICO+1
    Check for constant conditions
    L: if CHAR(24).NE.0.0 then go to L1
       ICO ← ICO-1
       FNN(N+I) ← FUN(YNN(N+I))
       ICO ← ICO+1
```

```
A4. [Compute predicted value for Step I]
      if IVSW.NE.1 then Y(NPI+1) ← PREDIC(Y,NPI,H,F)
      if IMSW.NE.1 then YMM(NPI+1) ← PREDIC(YMM,NPI,H,FMM)
      if IHSW.NE.1 then YHH(NPI+1) ← PREDIC(YHH,NPI,H,FHH)
      if INSW.NE.1 then YNN(NPI+1) ← PREDIC(YNN,NPI,H,FUNN)
   T: if IVSW.EW.0 then go to E
      Otherwise,
A5. [Set up for corrector]
      if IMSW.EW.0 then FMM(NPI+1) ← FUM(YMM(NPI+1))
      if IHSW.EQ.0 then FHH(NPI+1) ← FUH(YHH(NPI+1))
      if INSW.EQ.0 then FNN(NPI+1) ← FUN(YNN(NPI+1))
A6. [Compute corrected solution value for step I, iteration M]
      if IVSW.EQ.0 then YVC ← CORREC(Y,NPI,H,F)
      if IMSW.EQ.0 then YC ← CORREC(YMM,NPI,H,FMM)
      if IHSW.EQ.0 then YCH ← CORREC(YHH,NPI,H,FHH)
      if INSW.EQ.0 then YCN ← CORREC(YNN,NPI,H,FNN)
A7. [Test for convergence]
      DELV ← |YVC-Y(NPI+1)|
      DELY ← |YC-YMM(NPI+1)|
      DELH ← |YCH-YHH(NPI+1)|
      DELN ← |YCN-YNN(NPI+1)|
      if IVSW.EQ.1 then go to 01
      if DELV-EPS ≤ 0 then go to 02
      if DELV-EPS > 0 then go to 01
```

```
02: Y(NPI+1) ← YVC    ; good value converges
    IVSW ← 1
01: if IMSW.EQ.1 then go to 03
    if DELY-EPS ≤ 0 then go to B1
    if DELY-EPS > 0 go to B2
B1: YMM(NPI+1) ← YC    ; good value converges
    IMSW ← 1
B2: if IHSW.EQ.1 then go to G1
    if DELH-EPS ≤ 0 then go to G2
    if DELH-EPS > 0 then go to G1
G2: YHH(NPI+1) ← YCH    ; good value converges
    IHSW ← 1
G1: if INSW.EQ.1 then go to T1
    if DELN-EPS ≤ 0 then go to T2
    if DELN-EPS > 0 then go to T1
T2: YNN(NPI+1) ← YCN    ; good value converges
    INSW ← 1
T1: if M-MAX ≤ 0 then go to E1
    if M-MAX > 0 then go to E2
E1: if IVSW.NE.1 then Y(NPI+1) ← YVC
    if IMSW.NE.1 then YMM(NPI+1) ← YC
    if IHSW.NE.1 then YHH(NPI+1) ← YCH
    if INSW.NE.1 then YNN(NPI+1) ← YCN
    if IMSW+IHSW+INSW+IVSW.EQ.4 then go to S
    Otherwise,
    M ← M+1
    go to T
```

```
E:  ICO ← ICO+1
    IMSW ← 0
    IHSW ← 0
    INSW ← 0
    F(NPI+1) ← VOLT(V(NPI+1))
      ICO ← ICO-1
      go to A5
B:  [Constant YMM]
     YMM(ICO+1) ← YMM(ICO)
     FMM(ICO+1) ← FMM(ICO)
       go to 0
S:   if I-NCMN ≤ 0 then go to S1
     if I-NCMN > 0 then go to S2
S1:  I ← I+1
     X(NPI+1)=X(NPI)+H
       go to A2
G:  [Constant YHH]
     FHH(ICO+1) ← FHH(ICO)
     YHH(ICO+1) ← YHH(ICO)
       go to L
L1: [Constant YNN]
     FMM(ICO+1) ← FNN(ICO)
     YNN(ICO+1) ← YNN(ICO)
       go to S
S2:  Terminate algorithm.
E2:  NC ← I

     Fails to converge.
```

FN-Algorithm

Function FN(X)

```
FN1. [Initialize BCOEFF vector]
       BCOEFF(1) ← -.083333333
       BCOEFF(2) ←  .0166666667
       BCOEFF(3) ←  .0238095238
       BCOEFF(4) ←  .025
       BCOEFF(5) ←  .025252525
       BCOEFF(6) ←  .02531136
       BCOEFF(7) ←  .02533
FN2.   if |X|-1. < 0 then go to FN5
       if |X|-1. ≥ 0 then go to FN3
FN3.FN ← X/(EXP(X)-1.)
FN4.Terminate
FN5.X2 ← -X*X
     IOFLO ← OVERFL(IOFLO)
     FN ← 1.
     A ← -.5*X
     IOFLO ← OVERFL(IOFLO)
     if IOFLO.EQ.1 then terminate
FN6.FN ← 1.+A
FN7.[LOOP]
     LOOP: I=1 to 7
            A ← BCOEFF(I)*X2*A
            IOFLO ← OVERFL(IOFLO)
```

```
if IOFLO.EQ.1 then terminate
FN ← FN+A
go to LOOP
terminate  (after loop is complete)
```

This is not the only way to implement this model using the OLMS; however, it is one of the simplest.

V-Algorithm

Function Volt (Y)

```
V1.TOLT ← -120.*YMM(ICO)**3*YHH(ICO)*(Y-115.)
   -36.*YNN(ICO)**4*(Y+12.)-.3*(Y-10.598921)+CURR(ICO)
V2. if CAP.NE.1. then TOLT ← TOLT/CAP
V3. VOLT ← TOLT
V4. Terminate
```

FUH-Algorithm

Function FUH(YHH)

```
FUH1.ICO ← ICO+1
FUH2.FUH ← PHH*((1.-YHH)*.07*EXP(-.05*Y(ICO))-YHH/(1.+EXP(3.-.1*Y(ICO))))
FUH3.ICO ← ICO-1
FUH4.Terminate
```

FUM-Algorithm

```
                    Function FUM(YMM)
FUM1.ICO ← ICO+1
FUM2.FUM ← PHI*((1.-YMM)*FN(2.5-.1*Y(ICO))-YMM*4.*EXP(-Y(ICO)/18.))
FUM3.ICO ← ICO-1
FUM4.Terminate
```

FUN-Algorithm

```
                    Function FUN(YNN)
FUN1.ICO ← ICO+1
FUN2.FUN ← PHN*((1.-YNN)*.1*FN(1.-.1Y(ICO))-YNN*.125*EXP(-.0125*Y(ICO)))
FUN3.ICO ← ICO-1
FUN4.Terminate
```

P-Algorithm

```
                    Function PREDIC(Y,NPI,H,F)
P1.PREDIC ← Y(NPI)+(H/24.)*(55.*F(NPI)-59.*F(NPI-1)+37.*
            F(NPI-2)-9.*F(NPI-3))
P2. Terminate
```

C-Algorithm

```
          Function CORREC(Y,NPI,H,F)
C1.CORREC ← Y(NPI)+(H/24.)*(9.*F(NPI+1)+19.*F(NPI)-5.*F
          (NPI-1)+F(NPI-2))
C2.Terminate
```

FN-Algorithm

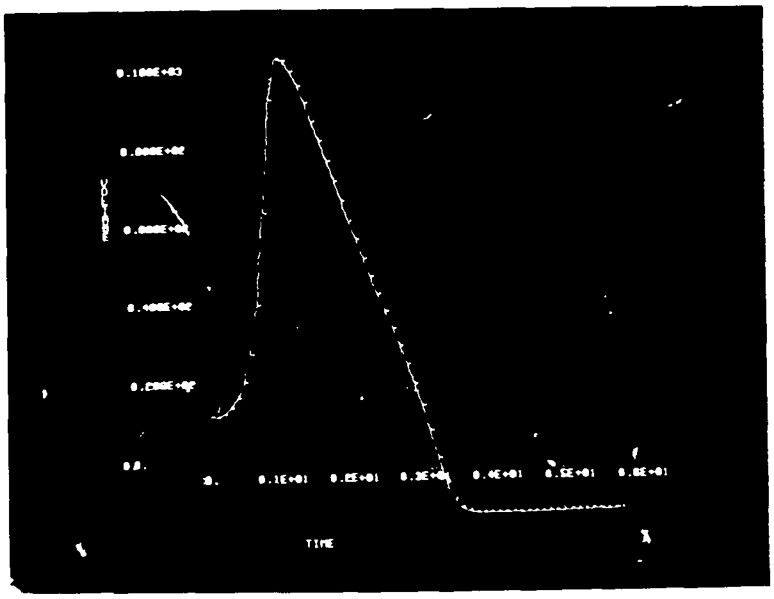
Function FN(X)

```
FN1.
FN2. if |X|-1. < 0 then go to FN3
     if |X|-.1 ≥ 0 then go to FN5
FN3.FN ← X/(EXP(X)-1.)
FN4.Terminate
FN6.X2 ← -X*X
    IOFLO ← OVERFL(IOFLO)
FN7.FN ← 1.
FN8.A ← -.5*X
```

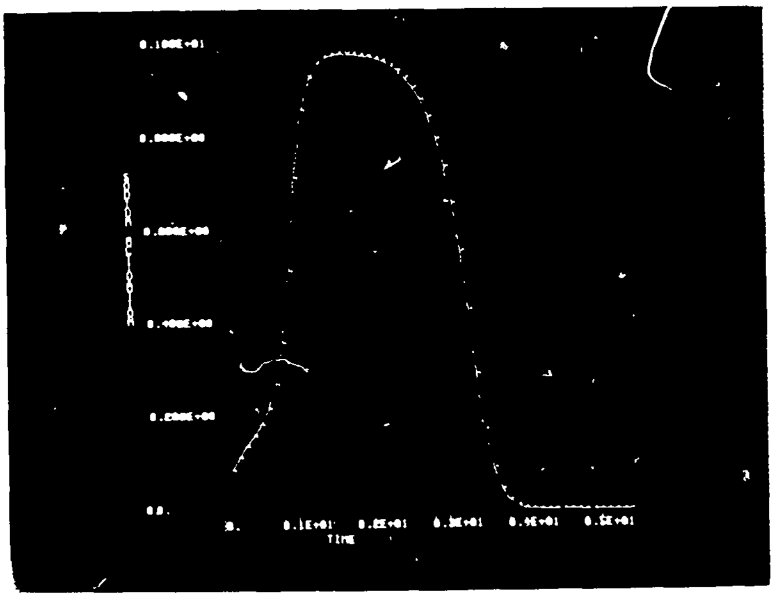
This is not the only way to implement this model using the OLMS; however, it is one of the simplest. Several graphs are given to show the input output relationships of this model.

RESULTS OF SIMULATION

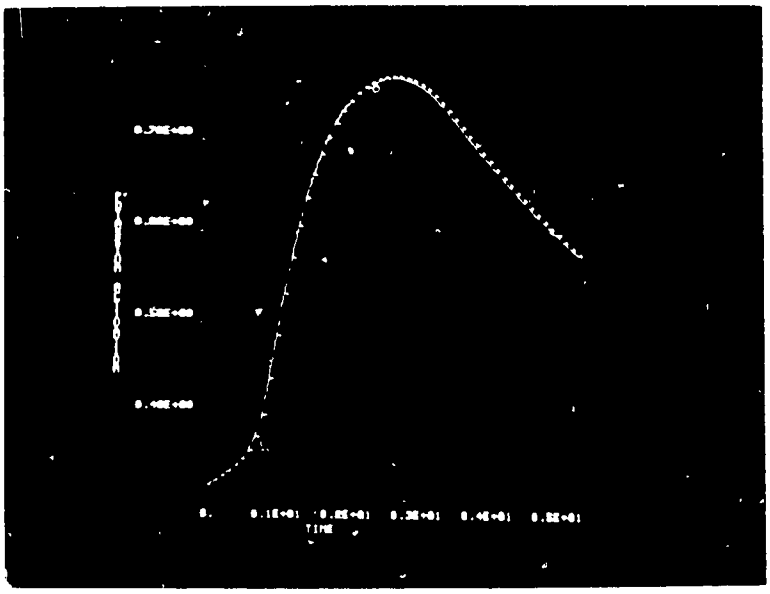
These graphs are given to show how the results of a model can be displayed using the OLMS.



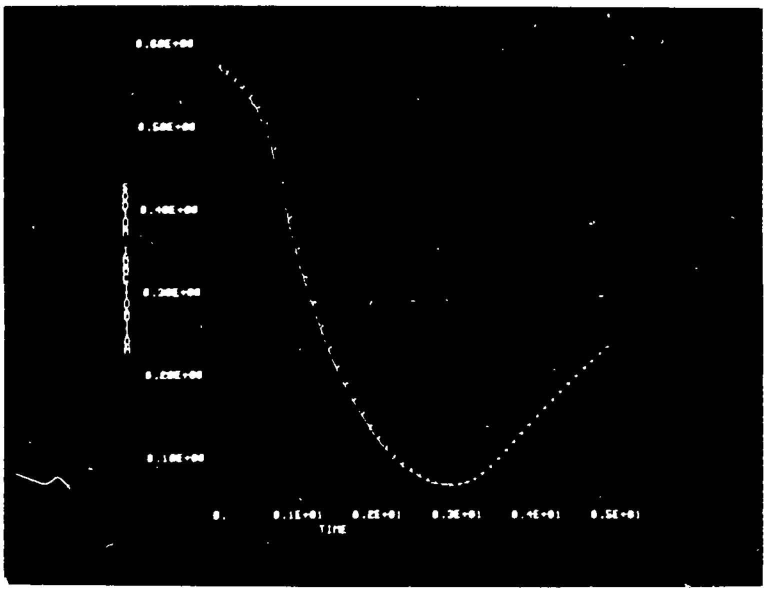
Action Potential



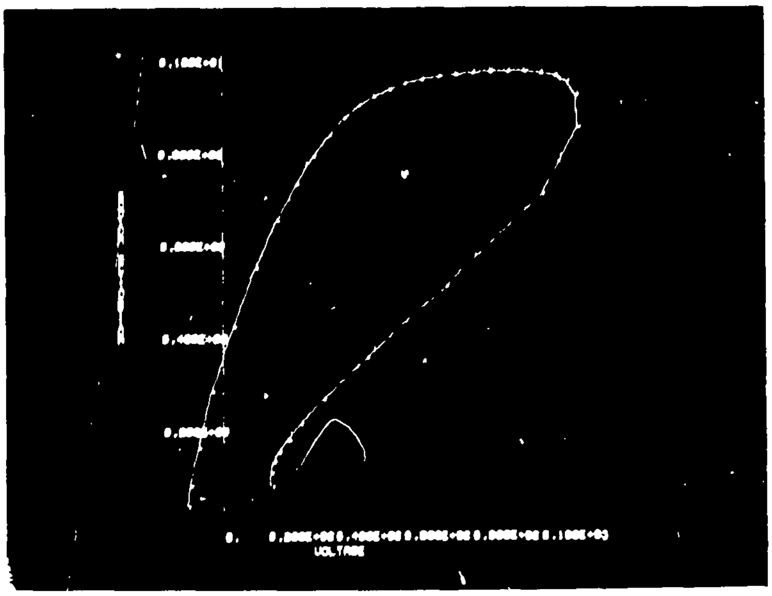
Sodium Activation



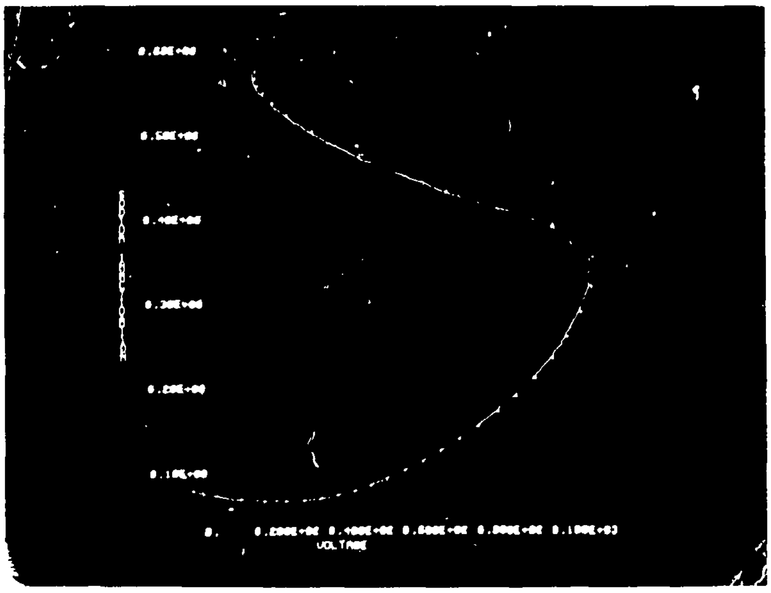
Potassium Activation



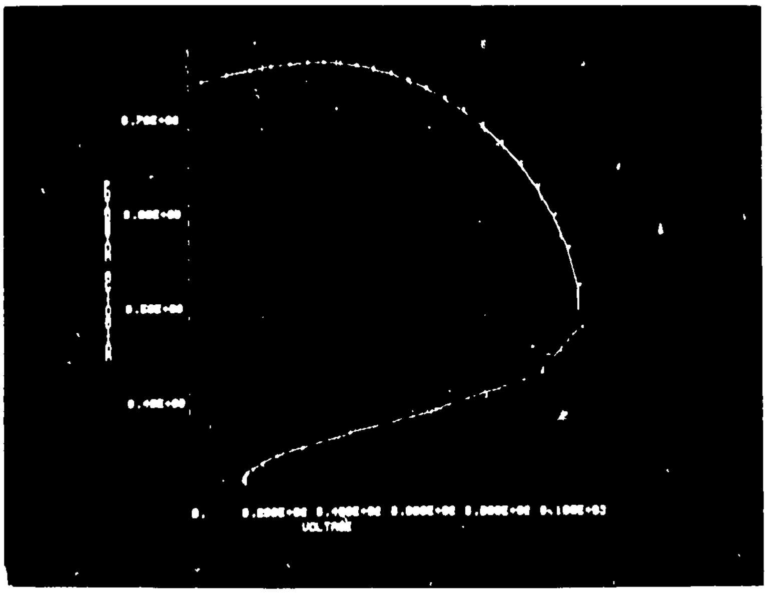
Sodium Inactivation



Sodium Activation v.s. Voltage



Sodium Inactivation v.s. Voltage



Potassium Activation v.s. Voltage

References

1. DEC-10-LOVA-D, CHAIN, Digital Equipment Corporation, Maynard, Massachusetts, February 8, 1968.

2. Bruce, M. C., PDP-10 Equipment, NIH Internal Memorandum, CCB, October 9, 1968.

3. Perkel, D. H., A Digital Computer Model of Nerve-Cell Functioning, Memorandum RM-4132-NIH, The Rand. Corp. Santa Monica, Calif., June 1964.

4. Lockemann, P. C. and Knutsen, W. D., Phase 2 User's Guide, Programming Report No. 4, Booth Computing Center, Calif. Inst. of Tech., Pasadena, Calif., June 1967.

5. Gwynn, J. S., Lockemann, P. C., Knutsen, W. D., Advanced Programming Support User's Guide, Programming Report No. 5, Booth Computing Center, Calif., Inst. of Tech., Pasadena, Calif., September 1967.

6. Randall, D. L., Richer, J., Dill, J. C., Plexus User's Guide, Programming Report No. 6, Booth Computing Center, Calif. Inst. Of Tech., Pasadena, Calif., December 1967.

7. Rosen, Saul, Programming Systems and Languages, McGraw-Hill, New York, 1967.

8. Lee, J.A.N., The Anatomy of a Compiler, Reinhold Publishing Corporation, New York, 1967.

9. Dill, J. C., Randall, D. L., Richer, I., Plexus An On-Line System for Modeling Neural Networks, Communications of the ACM, Vol. 11, pp. 623-629; September 1968.

10. Knuth, D. E., The Art of Computer Programming, Addison-Wesley Publishing Company, Rading Massachusetts, 1968.

11. Wegner, P., Introduction to System Programming, The Automatic Programming Information Center, England, 1964.

12. Mize, J. H., Cox, J. G., Essentials of Simulation, Prentice-Hall, New Jersey, 1968.

13. Stromberg-Carlson, Programmers' Reference Manual, Data Products, San Diego, Calif., October 1964.

14. Lewis, Harry, Fortran-Lisp Display Routines, NIH, Division of Computer Research and Technology, CCB, Bethesda, Maryland April 1969.

15. Adler, C., Sanford, 340 Display Programming Manual, Decus No. 7-13, New York University's Department of Industrial Engineering and Operations Research, Bronx, New York,

16. Lewis, Harry, Assembly Language For DEC 340 Display, NIH, Division of Computer Research and Technology, CCB, Bethesda, Maryland, April 1969.

17. Adler, C., Sanford, 340 Display Programming Manual, Decus No. 7-13, New York University's Department of Industrial Engineering and Operations Research, Bronx, New York.

18. DEC - 10 - MTEO - D, Time-Sharing Monitors; Multi-programming Monitor (10/50) Swapping Monitor (10/50), Digital Equipment Corporation, Maynard, Massachusetts, November 1968.

19. DEC - 10 - MTEO - D, Time-Sharing Monitors; Multi-programming Monitor (10/50) Swapping Monitor (10/50), Digital Equipment Corporation, Maynard, Massachusetts, November 1968.

20. Feldmann, R. J. Rand Tablet Service Routine, NIH, Division of Computer Research and Technology, CCB, Bethesda, Maryland, August 1969.

21. Stromberg-Carlson, Information Manual, Data Products, San Diego, Calif., October 1966.

22. Freedman, S.R., Filer, M.I.T./L.N.S., Cambridge, Massachusetts, February 12, 1968.

23. Vreenegoor, H., Lewis, H., Function Box Service Routine, NIH, Division of Computer Research and Technology, CCB, Bethesda, Maryland, September 1969.

24. DEC-10-NGCA-D, PDP-10 System User's Guide, Digital Equipment Corporation, Maynard, Massachusetts, 1967.

25. DEC-10-PPCO-D, Peripheral Interchange Program, Maynard, Massachusetts, 1968.

26. DEC-10-ETEB-D, Text Editor and Corrector Program, Maynard, Massachusetts, 1968.

27. Feldmann, R. J. An Extension to CCL (Concise Command Language), NIH, Division of Computer Research and Technology, CCB, Bethesda, Maryland,

28. Cooley, J. W. and Dodge, F. A., Digital Computer Solutions for Excitation and Propogation of the Nerve Impulse, Biophysical Journal, Volume 6, 1966.

29. Fitzhugh, R. Thresholds and Plateaus In The Hodgkin-Huxley Nerve Equations, The Journal of General Physiology, Volume 43, Number 5, pp. 867-896, May, 1960.

EM

TECHNICAL REPORT NO. 6

PART II

THE ON-LINE MODELING SYSTEM

April 1971

U.S. DEPARTMENT OF HEALTH, EDUCATION, AND WELFARE
Public Health Service National Institutes of Health

The Division of Computer Research and Technology, NIH, will issue on an irregular basis technical documents which we believe will be of particular interest to the biomedical community. These reports will include detailed descriptions of relevant computer programs and instructions in their use (as well as some theoretical background), in hopes that interested scientists will be encouraged to gain first-hand experience in applying them. In some cases, such reports may serve as foci around which DCRT will structure training courses to expand the knowledge and experience of NIH staff in applying computer science to problems of research and management. Circulation of these reports within the biomedical community broadly is, of course, encouraged.

A. W. Pratt, M.D.

A. W. Pratt, M.D., Director, DCRT

THE ON LINE MODELING SYSTEM

Part II. Programmers' Reference Manual

by

Edward Hill, Jr.

Laboratory of Applied Studies

Division of Computer Research and Technology
National Institutes of Health, Public Health Service
Department of Health Education and Welfare, Bethesda, Maryland 20014

PROGRAMMERS' REFERENCE MANUAL

Table of Contents

Section Page

V
(cont)

SECTION I

INTRODUCTION

To model anything it is necessary to analyze the results given by the model. In order to model it is important to have flexibility in both the modeling system and the modeling graphical subsystem. During the development of an On-Line Modeling System (Part I), I found it desirable to develop an On-Line Modeling Graphical System (OLMGS). This OLMGS is completely flexible with respect to graphical display. This flexibility will become obvious as this report is read.

The most obvious application of the OLMGS is the rapid production of labeled graphs. Results in graphical form are usually much easier to analyze then are the same results in printed tabular form. An option that is available is the production of a picture that combines tabular data with a plot of the related curve. Still another use of the OLMGS techniques is the creation of diagrams and line drawings. A camera may be operated with this system under program control.

At first glance, the list of the OLMGS routines on the following pages may seem formidable indeed, but the user should not feel baffled by the great volume of details. Some routines were developed for special purposes and have limited usage. Many are used principally as lower modules for more general subroutines, and will seldom be called directly by the programmer.

This Programmer's Reference Manual has been organized in major sections. These are arranged according to the function of the routines described therein. The major sections are: Introduction, Basic OLMGS Programming on the DEC 340 Display, Control Routines, Grid Drawing Routines, Scaling Routines, Point-Plotting and Line Drawing Routines, Titling and Labeling Routines, and a Higher Level Printing routine. Information about the interpretation of the system subroutines has been included in the control section.

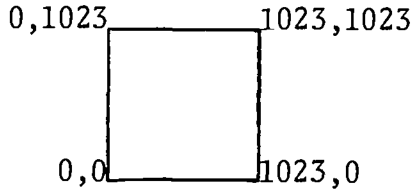
The OLMGS is a set of Fortran IV and Macro-10 Assembly Language routines that drive display premitives which were written by Mr. Harry Lewis. The Light Pen and Rand Tablet can be used in the manner described in Mr. Lewis' Reports, (see reference [3] and [5]). It is possible to implement this system on any computer if the rasters on the CRT are the same as the ones used in this report. If the rasters are different a few constants and uppe limits for loops can be changed to make the system compatible.

SECTION II

BASIC OLMGS PROGRAMMING

ON THE DEC 340 DISPLAY

The DEC 340 Display plots images on a grid 1024- horizontal by 1024- vertical points. There are therefore, over 1 million addressable positions. The center of any character may be plotted at any of these positions. The origin (0,0) is in the lower left corner, with X and Y increasing to the right and upward, respectively as shown in the figure below. The electron beam does not scan the face of the tube; its position is determined by the contents of the X and Y registers of the display. (If a data word specifies a point outside of the raster an edge violation occurs and stops the display).



SCALE

The scale setting determines the number of positions each succeeding spot is moved before it is intensified. It effects both the size and appearance of lines or symbols drawn in the vector, vector continue, increment, or character modes. At scale setting 11$_2$, each point can be clearly distinguished. At scale setting 00$_2$, lines and symbols appear to be continuous. The point spacing is illustrated in the following table.

Scale	Point Spacing	Itensity
00	● ● ● ● ● ● ● ● ●	Every
01	● ○ ● ○ ● ○ ● ○ ●	2nd
10	● ○ ○ ○ ● ○ ○ ○ ●	4th
11	● ○ ○ ○ ○ ○ ○ ○ ●	8th

INTENSITY

There are eight intensity levels available on the display, ranging from 00$_2$, which is barely visible, to 111$_2$, which is very bright. Note that scale and intensity settings are interrelated. For example, if characters are drawn (with the character generator) at the lowest scale setting, and too high an intensity is used, they will be badly blurred. On the other hand, if many characters are to be displayed simultaneously or if the light pen is to be used, it is best to use as high an intensity level as possible.

MODE

The mode register is a 3-bit register whose contents determine the way in which the next data word will be interpreted. The eight different data word formats (modes) will be discussed in the following paragraphs of this section.

DATA WORD FORMATS

PARAMETER MODE (000_2)

		Mode			Light Pen		Stop				Scale			Intensity			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Parameter mode is the control mode of the display. A parameter word can be used to change the mode, scale, intensity, light pen and/or interrupt parameters of the display

POINT MODE (001_2)

	H=0 V=1	Mode			Light Pen		INT	Horizontal or Vertical Address									
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Point mode is used to specify an X or a Y coordinate location on the display. It can change the mode, light pen, and intensify parameters. A data word is interpreted by the display as point mode if the mode register was set equal to $001_2$by the previously interpreted word.

RASTER MODE (010_2)

Esc		Ret	Int			Int			Int			Int			Int		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

CHARACTER MODE (011_2)

1st Character						2nd Character						3rd Character					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

In character mode the display interprets each word as containing three alphanumeric characters. Each character is specified by a 6-bit modified ASCII code. The display remains in the character mode until an escape code is encountered, then the display returns to parameter mode. A data word is interpreted by the display as character mode if the mode register was set equal to 011_2 by the previously interpreted word of if the previous data word was in character mode and did not contain the escape character. Bits 0-5 are interpreted as the second character and bits 12-17 are interpreted as the third character.

VECTOR MODE (100_2)

Esc	Int	← ΔY →								← ΔX →							
0	1	2 ±	3	4	5	6	7	8	9	10 ±	11	12	13	14	15	16	17

In vector mode the display interprets each word as containing vector size and direction, intensify, and escape information. The display remains in vector mode until the escape bit is set, at which time it returns to parameter mode. If the edge of the raster is violated, a flag is set which causes a computer interrupt. A data word is interpreted by the display as vector mode if the mode register was set equal to 100_2 by the previously interpreted word or if the previous word was in vector mode with the escape bit equal to 0.

VECTOR CONTINUE MODE (101_2)

Esc	Int	← ΔY →								← ΔX →							
0	1	2 ±	3	4	5	6	7	8	9	10 ±	11	12	13	14	15	16	17

In vector continue mode the display interprets the word as containing vector directions, and intensify information. The vector is drawn from the starting point to the edge of the raster. When the vector violates the edge of the raster, the display returns to parameter mode. A data word is interpreted by the display as vector continue mode if the mode register was set equal to 101_2 by the previously interpreted word.

INCREMENT MODE (110_2)

Esc	Int	1st Point				2nd Point				3rd Point				4th Point			
0	1	2 Move X	3 Right Left	4 Move Y	5 Up Down	6 M_X	7 R-L	8 M_Y	9 U-D	10 M_X	11 R-L	12 M_Y	13 U-D	14 M_X	15 R-L	16 M_Y	17 U-D

When in increment mode, the display interprets each succeeding word as containing information to plot four successive spots; each adjacent to the preceding one. A spot can be placed into any one of the eight adjacent locations at each movement. A data word is interpreted by the display as increment mode if the mode register was set equal to 110_2 by the previously-interpreted word or if the previous word was in increment mode and the escape bit was not equal to 1. The display remains in increment mode until the escape bit is set equal to 1 or it moves a spot past the edge of the raster.

SUBROUTINE MODE (111_2)

Op Code		Mode			Address												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

When in this mode, the display interprets the next word as a jump instruction to some location in memory. The subroutine word sets the mode of the next word to be interpreted and allows the display of data from nonconsecutive memory locations.

SYMBOL CONVENTIONS

In the subprogram descriptions, floating point variable names have been assigned in accordance with the FORTRAN IV conventions. If integer variables are required, names beginning with I, J, K, L, M, or N are used. The actual floating point number (or integer) may be used in the argument lists in place of a floating point (or integer) variable name, if the argument represents input to the subprogram. Constants should never be substituted for argument names that represent output from the subprogram.

Details about anything in this section can be found in references [6], [5], [3], and [7].

SECTION III

CONTROL ROUTINES

SETTING UP ERROR POINTERS: SETSEV

At the outset of every job, this routine should be called to set up system error pointers.

CALL SETSEV

CHAIN FILE COMPATABILITY

The OLMGS has a named common block in which it communicates with itself and other systems. This common block is designed for use by CHAIN. A description of the common block is in figure 3-1.

INTENSITY SELECTION: BRITEV,FAINTV,ITSCV

GRID1V ensures that the bright intensity mode is on. Normally, this intensity mode should be left on, since experience has shown that it produces the best results. If the programmer wants to change this setting to the faint mode, he can use the following statement:

CALL FAINTV

The faint mode will always be a setting of 4. Then, to restore the bright intensity mode, he can use the statement:

CALL BRITEV

Bright mode will always be a setting of 7. To set the intensity and scale to any value use the following statement:

CALL ITSCV (I,J)

where I = intensity $0 \leqq I \leqq 7$

J = scale 1,2,4,8

```
COMMON/DLIST/LIST,ZDDPTA,DXXSYY,XXXXDD,YYYYDD,ACDD,AYDD,BXDD,BYDD,DHIGHD,
DHIGH1,DLOWDD,DLOWD1,MLDD,MRDD,MBDD,MTDD,WIDEDD,HIGHDD,DDDDYY,DXXDYY,
DDDSDD,HOLDDD,CAMVDD,YTOPDD,YREGDD,DO,D1,FRMCNT,SCFL,NOCOMP
COMMON/INSCAL/INTENS,ISCALE,ICHASZ,LENGTH
```

FIGURE 3-1

SCALE SIZE CONTROL: BIGV, SMALLV, CHARV

The scale may be selected by the programmer. The program statement.

CALL BIGV

will set the scale to its maximum setting; and the statement,

CALL SMALLV

reduced the scale to its median setting. Character size can be set by calling CHARV. The calling sequence is:

CALL CHARV (k)

where k is the character size.

SECTION IV

GRAPHS

GRAPHING DATA: KWKPLT

The purpose of this routine is to provide the programmer with a quick look at the relationship between two variables. KWKPLT will automatically provide the programmer with a series of linearly connected points on a scaled linear grid with identification printing.

It is not necessary to arrange the coordinates in an increasing or decreasing order of magnitude. If the table of X-coordinates are not in ascending order, KWKPLT will rearrange them in ascending order within the table. The Y-coordinates will be arranged accordingly.

The calling sequence with identification printing is:

```
CALL KWKPLT(X,Y,N,LH,LV,ID)
```

where

X = starting location of a forward stored array of floating point numbers representing the X-coordinates.

Y = starting location of a forward stored array of floating point numbers representing the Y-coordinates.

N = number of points to be plotted.

LH = 18 character identification for the X-coordinates.

LV = 18 character identification for the Y-coordinates.

ID = 1 label axis.
= 0 no label on axis.

The printing routine assumes a full 18 characters including blanks.

PDP340

This routine provides the programmer with a quick look at the relationship between two variables. PDP340 will automatically provide the programmer with a series of linearly or non-linearly connected points on a scaled linear-linear, log-linear, linear-log, or log-log grid with identification printing. The calling sequence is:

```
CALL PDP340(NPLOT,MODE,NCHAR,NPTS,X,Y,XMIN,XMAX,YMIN,YMAX,XLABEL,
            YLABEL,GLABEL,IERR)
```

where

NPLOT	= =	1 eject and start new graph 2 use same graph
MODE	= = = =	1 Linear-Linear 2 Log-Linear 3 Linear-Log 4 Log-Log
NCHAR	=	Character Number
NPTS	=	Character Number
X	=	NPTS of X-coordinates
Y	=	NPTS of Y-coordinates
XMIN- XMAX	=	Minimum and Maximum of X-coordinates
YMIN- YMAX	=	Minimum and Maximum of Y-coordinates
XLABEL	=	72 character X axis label
YLABEL	=	72 character Y axis label
GLABEL	=	72 character heading
IERR	= =	1 Normal 2 or 3 Error

SECTION V

GRIDS

GENERATING A GRID: GRID1V

In many ways, plotting on the OLMGS is very much like plotting on a sheet of graph paper, but there are also distinct differences. For one thing, the programmer must create the grid; the picture is completely blank to start with.

Although every line of the grid must be specified on the OLMGS, there are advantages to this situation. A hand-plotted graph must be adapted to some preprinted form; more frequently than not, this means that some plotting area must be sacrificed in order to use the most convenient scale.

On the OLMGS, the programmer can select a scale that will be easy to read and that will accommodate the entire range of data. The number of light grid lines, the number of emphasized grid lines, and the spacing between lines can be chosen to suit the plot. The programmer is not restricted to the use of a single form for a variety of plots. For each graph, a new grid can be tailored to the data.

The easiest way to create a grid is to call the GRID1V subprogram. At the outset, GRID1V makes certain that the Bright Intensity Mode is on.

GRID1V will produce a grid which has some lines emphasized and some lines labeled. Margin space (which may be used for titles) will be reserved at the top, left side, and bottom of the grid. Normally, the title margin spaces are 24 raster counts wide.

Upon completion of GRID1V, scale factors will have been established and made available (internally) for the conversion requirements of other subprograms; i.e., the conversion of floating point coordinates into raster coordinates.

The call statement for GRID1V appears below, with a description of the arguments.

```
          CALL GRID1V (L, XL, XR, YB, YT, DX, DY, ±N, ±M, ±I, ±J,

                    ±NX, ±NY)
```

L — This integer argument contrcls the label margins computations.

L ≠ 1 No label margins will be computed.

L = 1 Compute label margins.

XL,XR — Floating point values of X for the left-most and right-most limits of the grid.

YB,YT — Floating point values for the bottom limit and the top limit of the grid.

After margin space for titles and labels has been reserved, the limits of the remaining space are assigned the data values given for XL, XR, YB, and YT. Scale factors are then computed; they will remain in effect until another GRID1V statement is made (or until other action is taken to compute new scale factors).

DX,DY — Floating point data increment at which vertical (specified by DX) and horizontal (specified by DY) grid lines will be displayed. If 0.0, no lines will be shown.

Positions are stepped off in DX increments in the positive and negative directions from X = 0, and in DY increments in the positive and negative directions from Y = 0.

N,M — Fixed point integers that cause every Nth vertical grid line and every Mth horizontal grid line to be retraced for emphasis. If N (or M) is zero, no vertical (or horizontal) lines will be emphasized.

To force the grid to be square, a negative sign should be used on N and/or M. (If either N or M is zero, the negative sign should go on both N and M.)

I, J — Fixed point integers which cause every Ith vertical line and every Jth horizontal line to be labeled. If I (or J) is zero, no vertical (or horizontal) lines will be labeled.

If I and J are positive, the line labels will lie along the X = 0 and Y = 0 lines, provided these lines are within the grid limits. If X = 0 (or Y = 0) does not fall within the grid limits, labels will be placed in a space reserved at the left (or at the bottom) of the grid.

Negative signs can be used on I and/or J to force labels outside the grid. Label space is reserved at the left if I is negative, or at the bottom if J is negative, and labels will be placed in these reserved spaces. Note that label margin space is in addition to the margin reserved for titles.

NX, NY — Fixed point integers indicating the number of characters to be displayed in the labels of vertical and horizontal lines.

+NX, +NY — The labels will be in a decimal format similar to the F-type format. In specifying +NX and +NY, a decimal point must be counted as one of the NX or NY characters, but the sign is not counted. The largest number of digits permitted is 6 (or 7 if one character is a decimal point).

-NX, -NY — The labels will be in scientific notation. (Example: $1.25x10^{+02}$.) NX indicates the number of significant figures in the labels of vertical grid lines, and NY indicates the same for the labels of horizontal lines. The sign, decimal point, and exponent will be displayed in addition to NX (or NY) characters. NX (or NY) must not be greater than 6.

Examples of GRID1V Usage

Figures 5-1 through 5-9 are examples of the effect of the various parameters in the

GRID1V call statement. These examples are reproduced from DEC 340 output. The call statement to produce each graph is printed in the figure with the graph.

Figure 5-1 is a simple grid with the x=0, y=0 lines crossing in the middle of the grid. The numeric labels have been placed along the x=0, y=0 lines. For simplicity in the illustration, constants were used in the parameter list. In actual usage, variable names may be substituted for any parameter.

Figure 5-2 is similar to Figure 5-1 except that the XL, XR and YB, YT have been reversed to show that the scaling routines have no difficulty handling data which decreases from left to right and bottom to top.

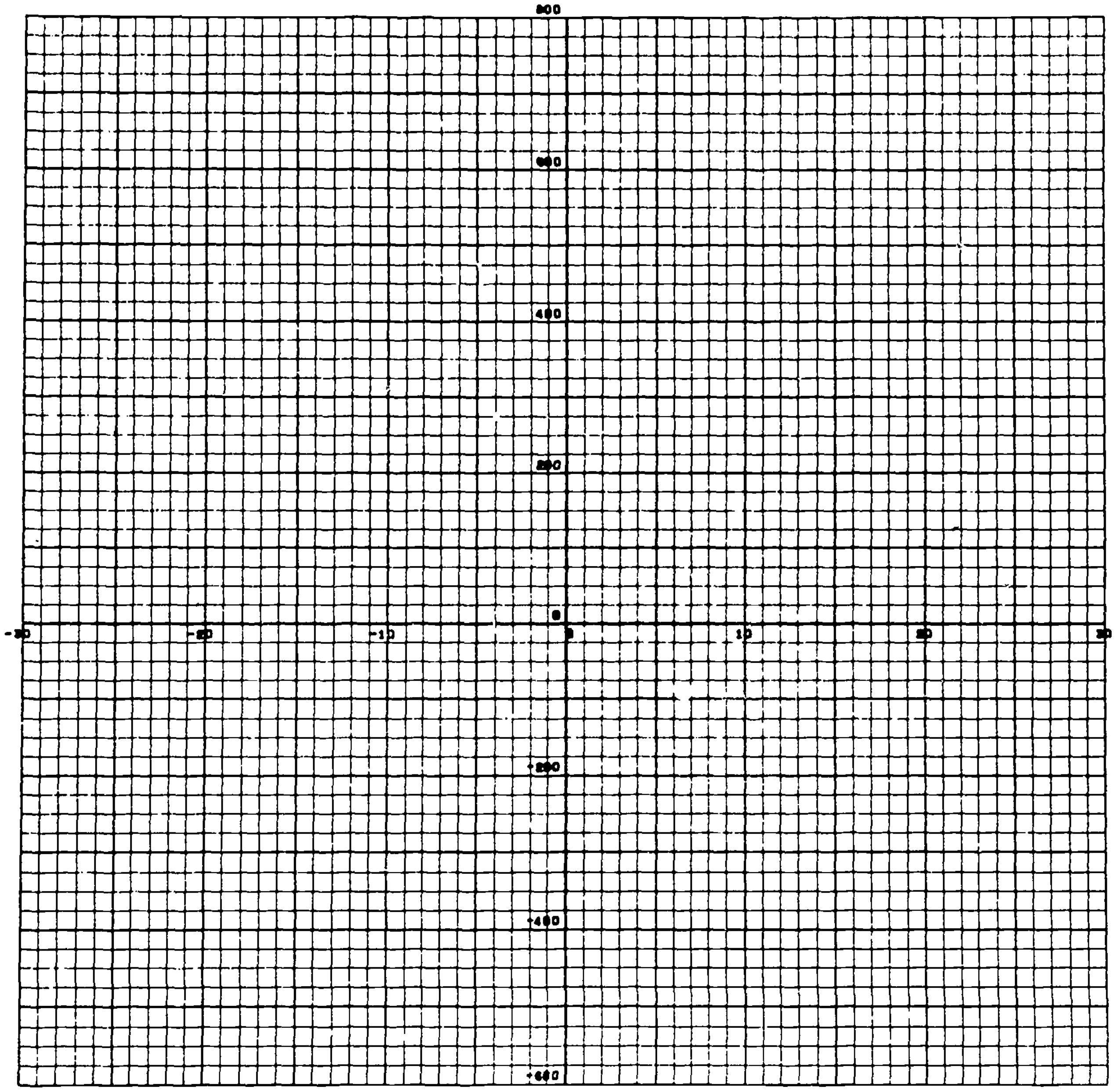
Figure 5-3 illustrates the effect of negative values for I, J in the parameter list. Note that the numeric labels are outside the grid and that the margins have been increased to accommodate the labels.

Figures 5-4 and 5-5 show the same grid with labels in integer notation and scientific notation.

Figure 5-6 has been double exposed to show the effect of negative arguments at N or M. The outer frame was produced by the first call statement with the positive argument for N and M. The grid utilizes the maximum available space in both directions and is taller than it is wide. The second call statement with negative N and M forced the frame to be shorter in the Y direction in order to be square. It is important to have a square grid when representing geometric figures such as a circle or a square. Note that the first parameter of the second call statement is a 2 which inhibits the margin calculation.

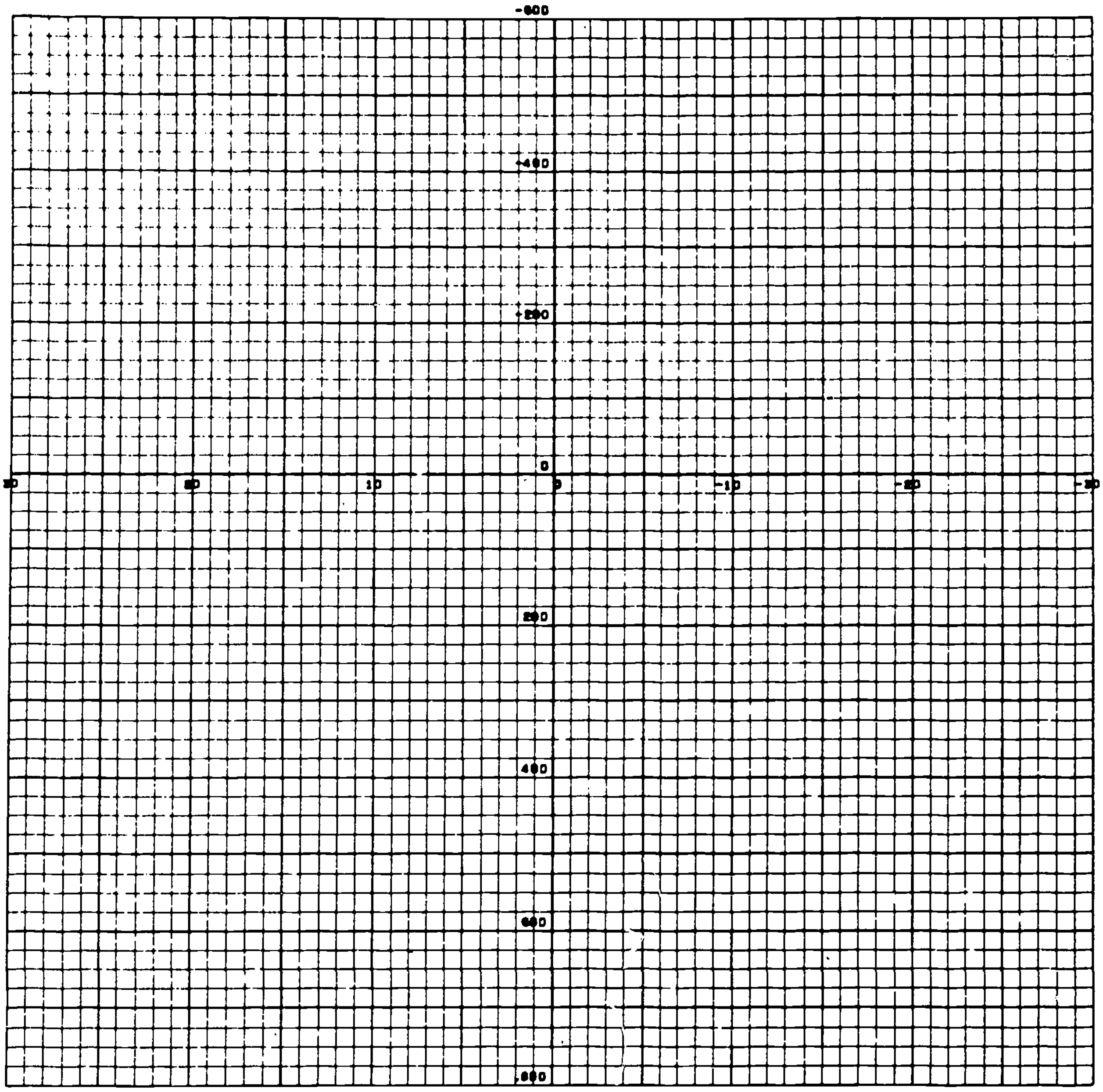
Figure 5-7 illustrates the use of the routine DXDYV to compute some of the values for the GRID1V parameter statement. DXDYV is explained on page 5-13.

Figures 5-8 and 5-9 show the influence upon the grid of the density factor used by DXDYV. For the case of Figure 5-8, a density factor of 8.0 was specified as the 8th argument of DXDYV. A larger factor, 20.0, caused DXDYV to derive values of DX and DY such that the grid in Figure 5-9 is less dense.



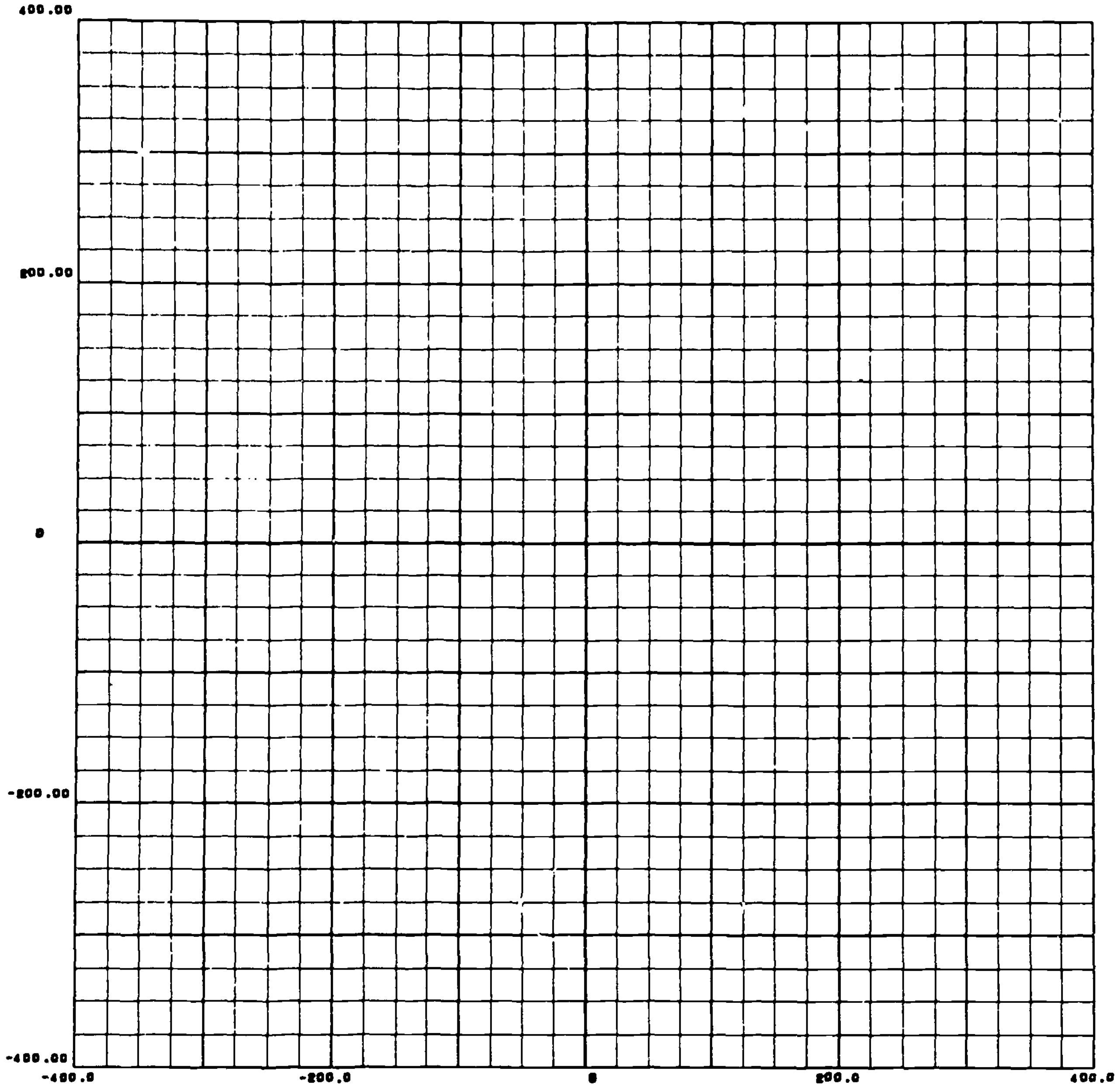
```
CALL GRIDIV (1,-30.0, 30.0,-600.0, 800.0, 1.0, 25.0, 5, 4, 10, 6, 2, 3 )
```

Figure 5-1



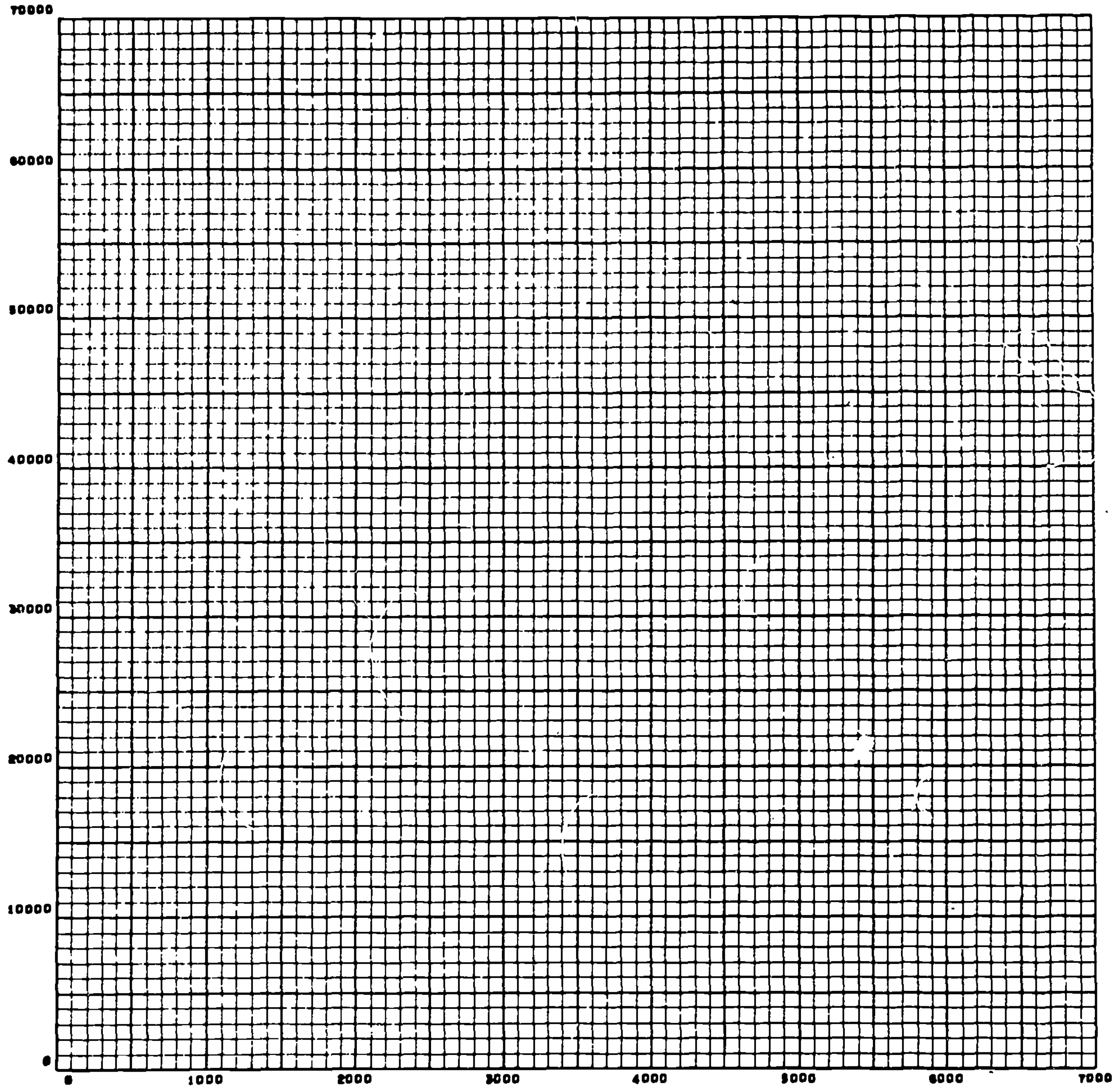
```
CALL GRIDIV (1, 30.0,-30.0, 800.0,-600.0, 1.0, 25.0, 5, 4, 10, 8, 2, 3 )
```

Figure 5-2



```
CALL GRIDIV (1,-400.0, 400.0,-400.0, 400.0, 25.0, 25.0, 4, 4,-8,-8, 5, 8 )
```

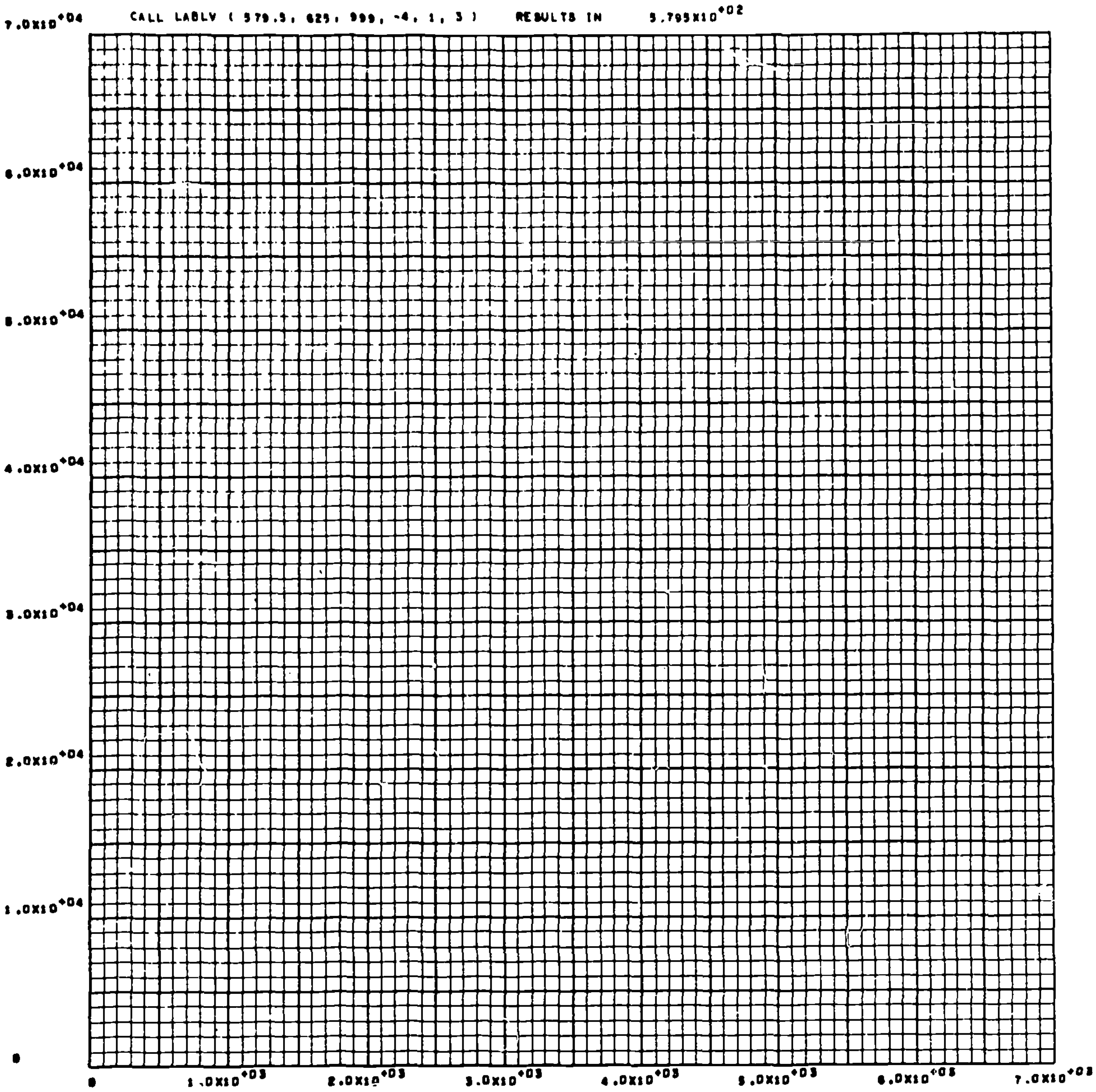
Figure 5-3



```
CALL GRID1V ( 1, 0.0, 7000.0, 0.0, 70000.0, 100.0, 1000.0, 5, 5, 10, 10, 4, 5 )
```

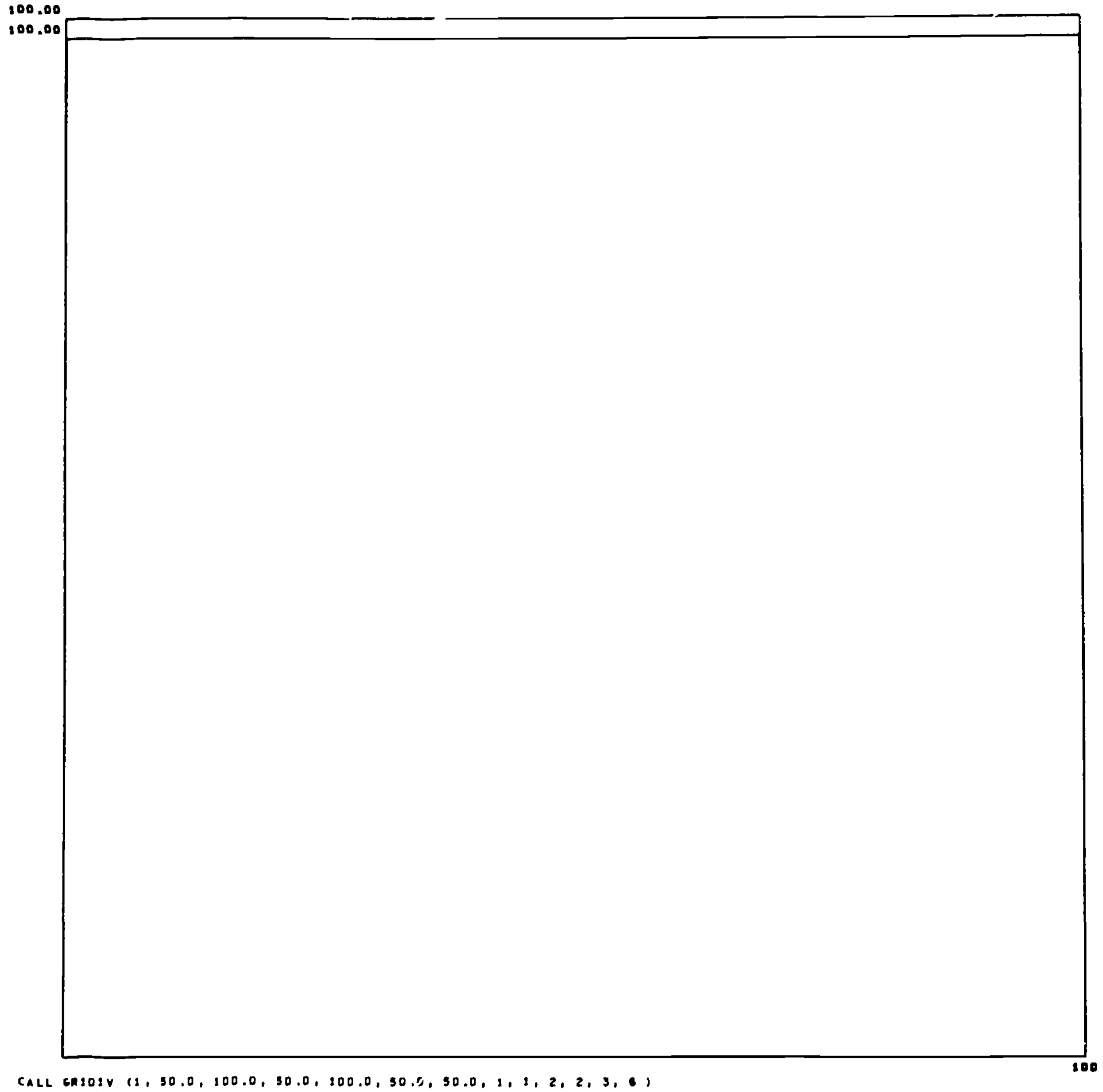
Figure 5-4

```
CALL LABLV ( 579.5, 625, 1023, 5, 1, 3 )     RESULTS IN     579.5
CALL LABLV ( 579.5, 625, 999, -4, 1, 3 )     RESULTS IN     5.795X10+02
```



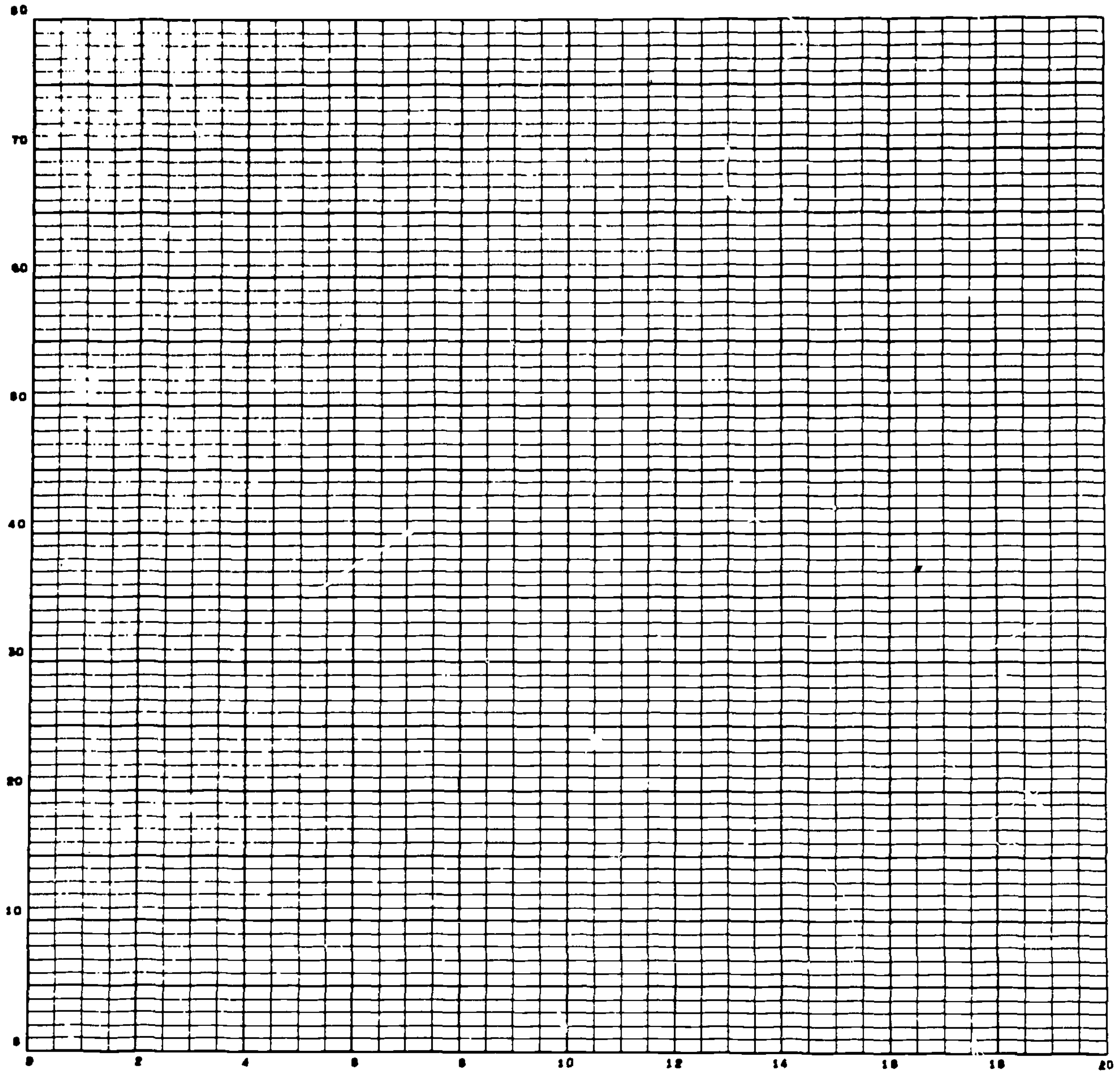
```
CALL GRID1V ( 1, 0.0, 7000.0, 0.0, 70000.0, 100.0, 1000.0, 5, 5, 10, 10, -2, -2 )
```

Figure 5-5



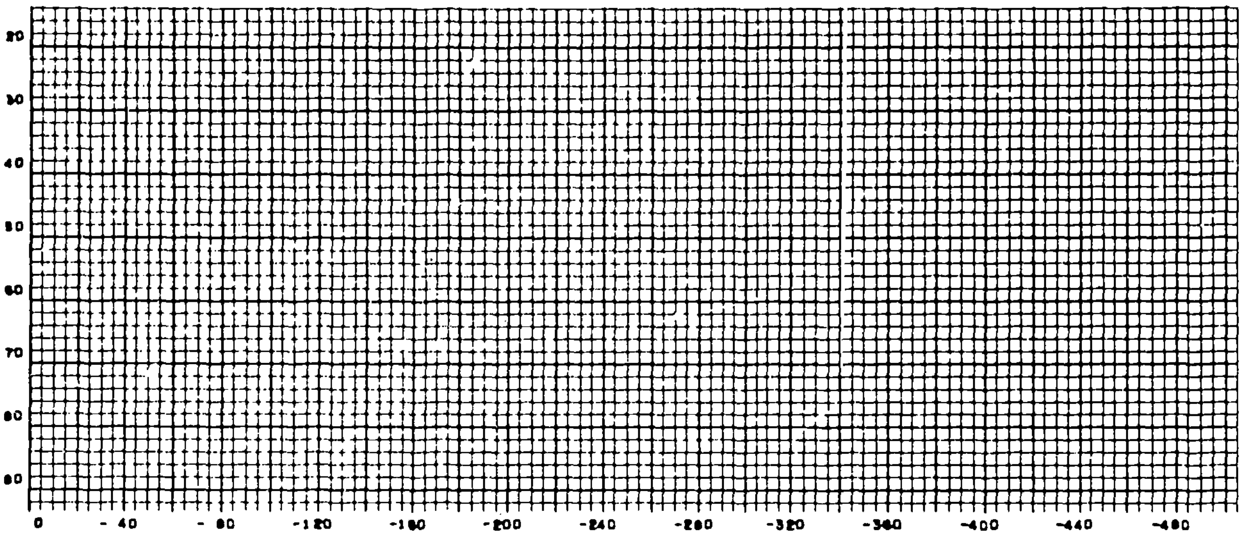
```
CALL GRIDIV (1, 50.0, 100.0, 50.0, 100.0, 50.0, 50.0, 1, 1, 2, 2, 3, 6 )
CALL GRIDIV (2, 50.0, 100.0, 50.0, 100.0, 50.0, 50.0,-1,-1, 2, 2, 3, 6 )
```

Figure 5-6



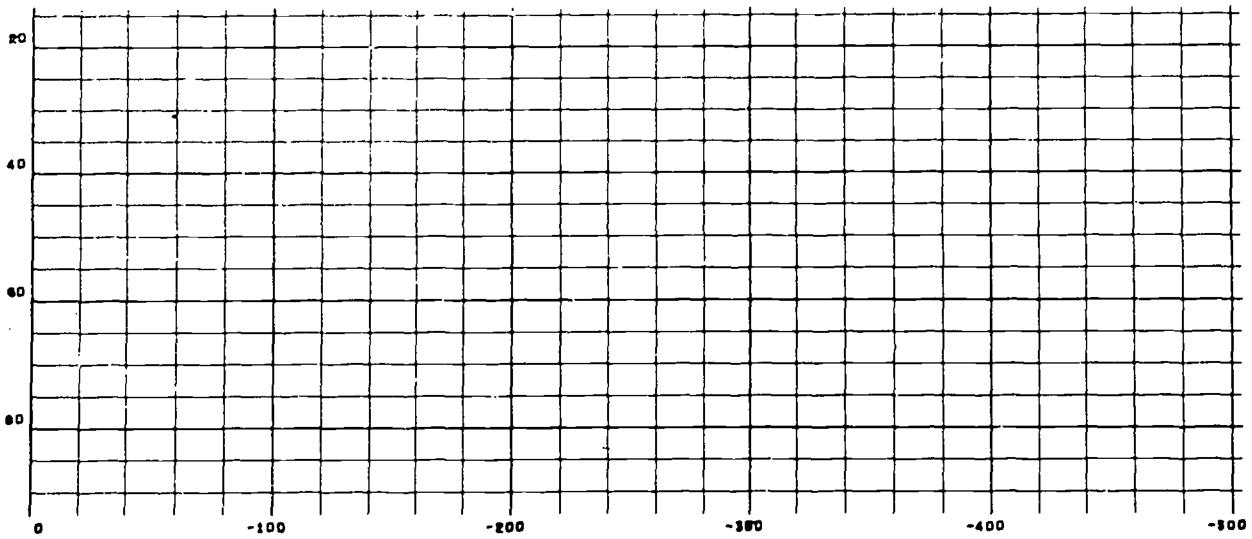
```
CALL DXDYV (1, 0.0, 20.0, DX, N, I, NX, 10.0, IERR1 )
CALL DXDYV (2, 0.0, 80.0, DY, M, J, NY, 10.0, IERR2 )
CALL GRID1V (1, 0.0, 20.0, 0.0, 80.0, DX, DY, N, M, I, J, NX, NY )
```

Figure 5-7



```
CALL DXDYV ( 1, 0.0, -505.0, DX, M, I, NX, 8.0, IERR )
CALL DXDYV ( 2, 93.5, 14.0, DY, M, J, NY, 8.0, IERR  )
CALL GRIDIV ( 1, 0.0, -505.0, 93.5, 14.0, DX, DY, M, M, I, J, NX, NY )
```

Figure 5-8



```
CALL DXDYV ( 1, 0.0, -505.0, DX, N, I, NX, 20.0, IERR )
CALL DXDYV ( 2, 93.5, 14.0, DY, M, J, NY, 20.0, IERR )
CALL GRIDIV ( 2, 0.0, -505.0, 93.5, 14.0, DX, DY, N, M, I, J, NX, NY )
```

Figure 5-9

COMPUTATION OF GRID1V ARGUMENTS: DXDYV

It frequently happens that the programmer does not have sufficient advance information about the range of data his program will encounter to be able to assign practical values to all arguments of GRID1V. In this case, a series of FORTRAN statements can be used to determine the upper and lower X and Y bounds. For example, the values of XL and XR for a block of data, X, can be computed as follows:

```
      XL = X(1)
      XR = X(1)
      DO 10 J = 2, NPTS       where NPTS is the number
                              of points in the X block of
                              data
      XL = MIN1F(XL, X(J))
   10 XR = MAX1F(XR, X(J))
```

A similar group of statements can be used to compute YB and YT for the Y block of data.

Once XL and XR (or YB and YT) are known, the routine DXDYV is available to compute arguments for line spacing, line emphasizing, and line labeling. Two call statements are available, one for the X direction and one for the Y direction. They are:

```
      CALL DXDYV(1, XL, XR, DX, N, I, NX, DC, IERR)

      CALL DXDYV(2, YB, YT, DY, M, J, NY, DC, IERR)
```

On each entry to DXDYV, four arguments are furnished by the programmer:

> The first argument is a 1 or a 2, to indicate whether DXDYV is being applied in the X direction or in the Y direction.
>
> XL and XR (or YB and YT) are defined as in the summary of GRID1V arguments.
>
> DC represents a floating point quantity which limits the density of the grid. The grid lines drawn by GRID1V, using arguments furnished by DXDYV, will be no closer than DC raster positions. DC should never have a value less than 3.0; values of 8.0 to 20.0 are recommended.

The remainder of the arguments are variables to which DXDYV will assign values. Any value previously assigned these variables will be destroyed during execution of the subroutine. NEVER USE CONSTANTS FOR THESE ARGUMENTS.

IERR is an error indicator. It is set to zero if a reasonable grid can be drawn, and to one if the parameters given would result in an impossible grid. After execution of DXDYV, IERR should always be tested before proceeding to draw the graph.

In using DXDYV, it should be noted that no provision has been made for generating labels in scientific notation. If this is desired, it is necessary to assure that there is sufficient space for the longer labels and also to change the sign of NX and NY to be negative.

GRID1V CONTROLS

Certain features of the basic linear GRID1V can be altered by subprograms that control its internal operation. The subprograms can be classified as "set" and "retrieve" routines since they permit information to be set by the programmer and retrieved during execution of GRID1V.

The routines that furnish values different from those normally employed by GRID1V are:

SETMIV, which allows the programmer to make nonstandard margin assignments. The companion routine called by GRID1V to retrieve margin values is SETMOV.

SETCIV, which makes it possible to provide extra space for grid line labels. The companion routine is SETCOV.

Routines that furnish indicators recognized by GRID1V as signals to execute alternate branches are:

HOLDIV, which assists in holding margins from graph to graph. HOLDOV is called to retrieve the indicators.

SMXYV, which enables the programmer to select a non-linear mode of operation. The companion routine is MSXYV. These two routines are described under "Log and Semi-log Plotting."

Grid Margin Variation: SETMIV, SETMOV

As discussed in an earlier section, GRID1V normally reserves a strip, 24 raster counts in width, at the top, left, and bottom of the grid, for the display of titles. For the many applications which require special margin widths, the subprogram SETMIV can be called to change the basic specifications.

One obvious application of SETMIV is to provide margin space for multiple lines of printed titles and headings. In addition, and perhaps even more important, SETMIV makes it possible to display more than one graph on a frame, or to display a graph with its accompanying text.

The standard GRID1V margin specifications can be altered by the statement:

```
CALL SETMIV (MTL, MTR, MTB, MTT)
```

Each argument is an integer which specifies, in raster counts, the width of one area to be reserved for a margin.

MTL	Width of area for left margin.
MTR	Width of area for right margin.
MTB	Width of area for bottom margin.
MTT	Width of area for top margin.

GRID1V does not necessarily use these exact values for the upper and lower limits of X and Y. It guarantees that the reserved space will not be overlapped, assigning additional space if required for label margins. After the total margin space has been reserved, the remaining area will be used for the grid.

If SETMIV is never called, GRID1V will use the values 24, 0, 24, 24 as MTL, MTR, MTB, and MTT, respectively. To return to a standard grid after the margins have been altered, restore the standard margin values by

```
CALL SETMIV (24, 0, 24, 24)
```

The current values of MTL, MTR, MTB, and MTT can be retrieved by using the statement

```
CALL SETMOV (MTLL, MTRL, MTBL, MTTL)
```

where the arguments are variables (never constants) to which SETMOV is to assign the current margin values. SETMOV was designed for use by GRID1V to retrieve current margin values; the programmer will rarely have reason to call it.

Examples. Figure 5-10 shows three grids with the SETMIV and GRID1V call statements used to produce them. The grid at the bottom was the first one displayed; a 1 was used as the first argument of the first GRID1V statement executed, in order to change the film frame. The other two GRID1V statements include a 2 as the first argument, to inhibit margin calculation.

Note particularly the variation in the raster locations assigned by GRID1V to XL in each of the grids. This effect is caused primarily by the differences in the specification of NY (the last argument), which gives the number of characters to be displayed in the labels of horizontal lines. In each case, NY has been assigned a value just large enough to satisfy the needs of the grid. For the bottom grid, NY = 1; for the middle grid, NY = 3; and for the top grid, NY = 5. Since margin space was reserved for labels of different lengths, the positions of the left limits, and of the corresponding values of X, vary noticeably. Such a nonalignment is often of no importance, but if it does matter, the programmer may have to make special provisions to force alignment.

PROVIDING FOR SPECIAL LABEL CHARACTERS: SETCIV, SETCOV

GRID1V computes the starting location of each label, taking into consideration the size of the characters used. If the labels are to be placed outside the grid, GRID1V assigns space for them, again taking the character size into consideration. Normally, labeling is done in CHARACTRON characters (via LABLV). If the programmer substitutes a non-system labeling routine for LABLV, it may be necessary to furnish adjusted character dimensions to GRID1V.

To state the dimensions of nonstandard label characters, use

CALL SETCIV (IW, IH)

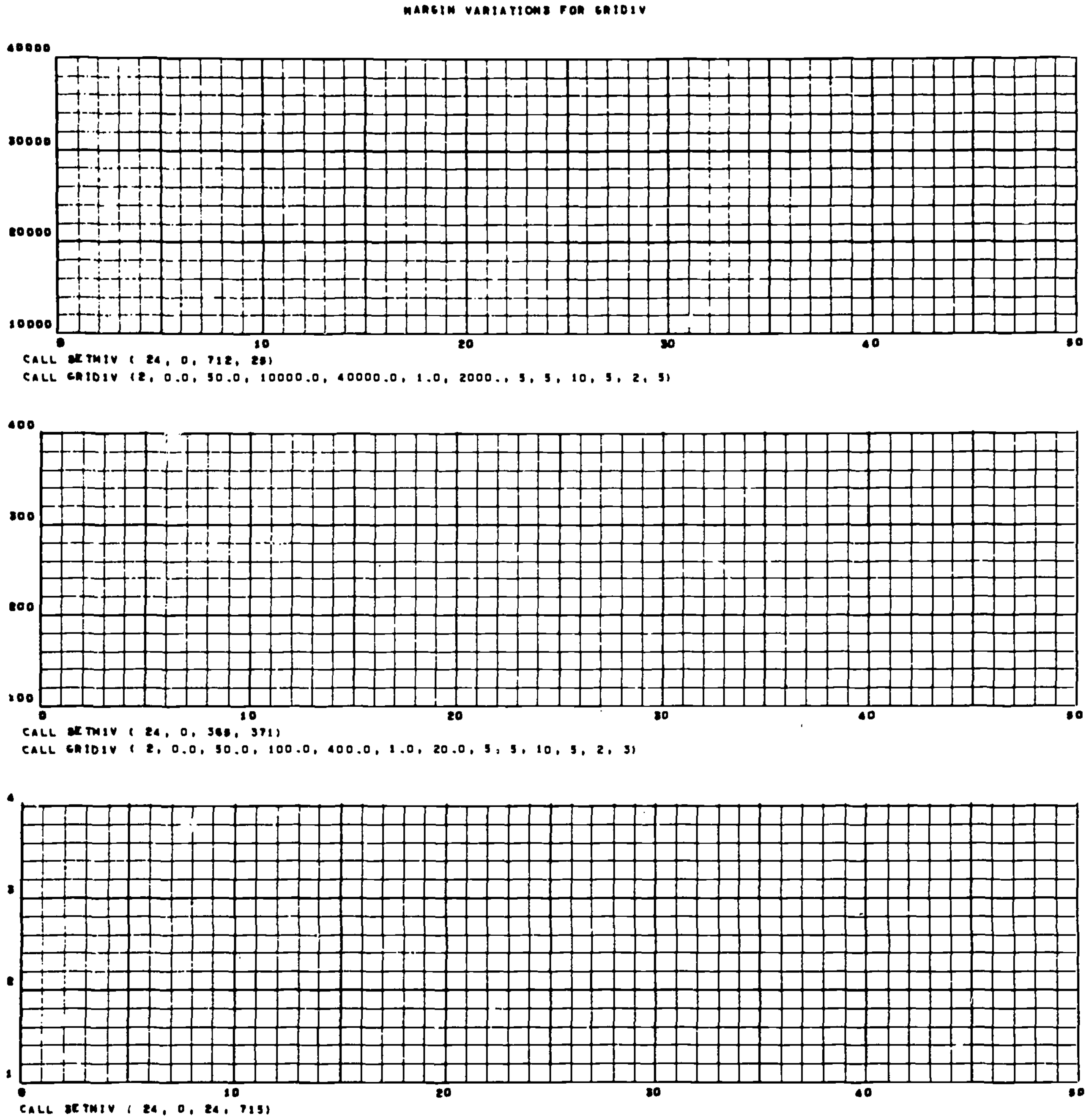
IW An integer which specifies, in raster counts, the allowance needed for the width of each label character.

IH An integer which specifies, in raster counts, the allowance for the height of a label character.

If SETCIV is never called, the indicator table contains IW = 8 and IH = 10. Obviously, if it is called, the arguments must be compatible with the size of the characters employed by the LABLV subprogram used.

GRID1V retrieves the values of the indicators by using:

CALL SETCOV (IWL, IHL)



```
CALL SETMIV ( 24, 0, 712, 28)
CALL GRIDIV (2, 0.0, 50.0, 10000.0, 40000.0, 1.0, 2000., 5, 5, 10, 5, 2, 5)
```

```
CALL SETMIV ( 24, 0, 368, 371)
CALL GRIDIV ( 2, 0.0, 50.0, 100.0, 400.0, 1.0, 20.0, 5; 5, 10, 5, 2, 3)
```

```
CALL SETMIV ( 24, 0, 24, 715)
CALL GRIDIV ( 1, 0.0, 50.0, 1.0, 4.0, 1.0, 0.2, 5, 5, 10, 5, 2, 1)
```

Figure 5-10

The width will be retrieved from the table and stored in the fixed point variable location IWL, and the height will be similarly stored in IHL. (The arguments must not be constants.) Note that GRID1V uses this information to control the space that will be reserved for labels; it does not control the size of the label characters themselves in any way.

HOLDING MARGINS FROM GRAPH TO GRAPH: HOLDIV, HOLDOV

For a large graph, it may be necessary for the programmer to display segments of the graph in separate frames, and join the segments "tile fashion" to form the complete plot. If the graphs are to have the same scale, certain equalities should exist.

1. The range of X in each segment should be equal, and the range of Y should be equal. In other words, the quantity (XR - XL) should be the same in each segment, and, similarly, the quantity (YT - YB).
2. The dimensions of the scaled area should be the same from segment to segment; that is, the dimensions of the space between margins should be equal.

The programmer can easily provide for equality in the ranges of X and Y, but he cannot so readily ensure equality in the scaled areas. Since GRID1V computes label margins (and, therefore, total margins) to suit the needs of each graph, the dimensions of the scaled area may vary.

One method that will usually give equality of scaled areas is to specify the option that forces labels to be placed outside the grid, and to always request the same number of label characters (NX, NY) for each segment. If this is not practical, a "holding" feature is provided.

GRID1V can be instructed to hold the label margin spaces used for the preceding grid and use them in computing total margins for the next grid. The statement to be used is

```
CALL HOLDIV (NH)
```

If NH $\neq 0$, the label margins from the preceding grid will be used again. If NH = 0 (as is the case if HOLDIV is never called), label margins will be computed in the normal manner.

The status of this indicator is tested in GRID1V by using

CALL HOLDOV (NHL)

The value of the indicator will be retrieved and stored in the location named in the argument.

The "hold" may be released by executing the statement

CALL HOLDIV (0)

Figure 5-11 is similar to Figure 5-10 except that NY = 5 on all three grids, permitting the left limits to be in line. If XL, NX, and NY are equal from grid to grid, the desired alignment will usually be achieved.

Figure 5-12 shows four graphs on a single frame. The SETMIV statements used to produce the margins for each grid are shown. The programmer must remember to set the first argument of the GRID1V statement to 2 so as not to calculate margins.

Figures 5-13, 5-14, and 5-15 show additional examples of special effects which can be obtained with GRID1V when the routine HOLDIV is used to retain grid margins from one grid to another. The labeling is self explanatory.

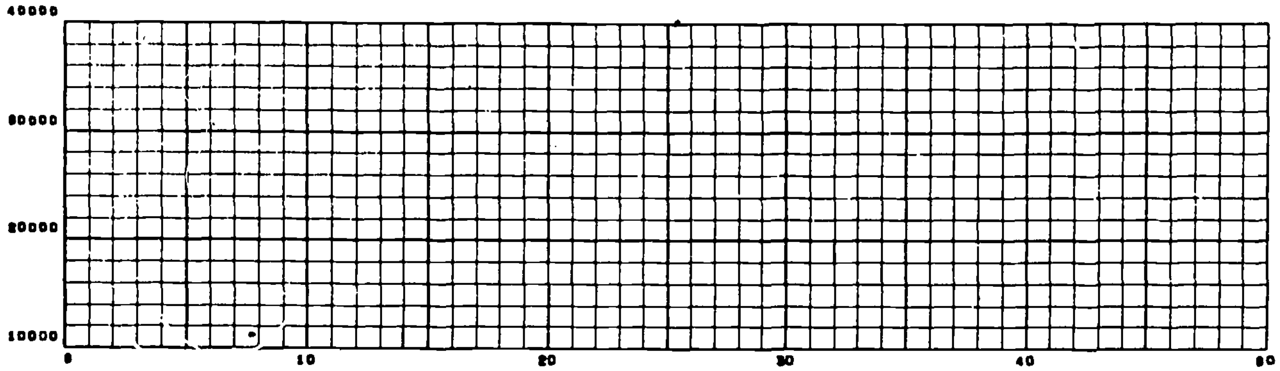
OPERATIONAL DETAILS OF GRID1V

GRID1V is, in many respects, an executive routine. It examines the information furnished by the argument list and by certain external subprograms, makes decisions based on this information, and then calls other subprograms to calculate margins, compute scale factors, generate the grid, etc.

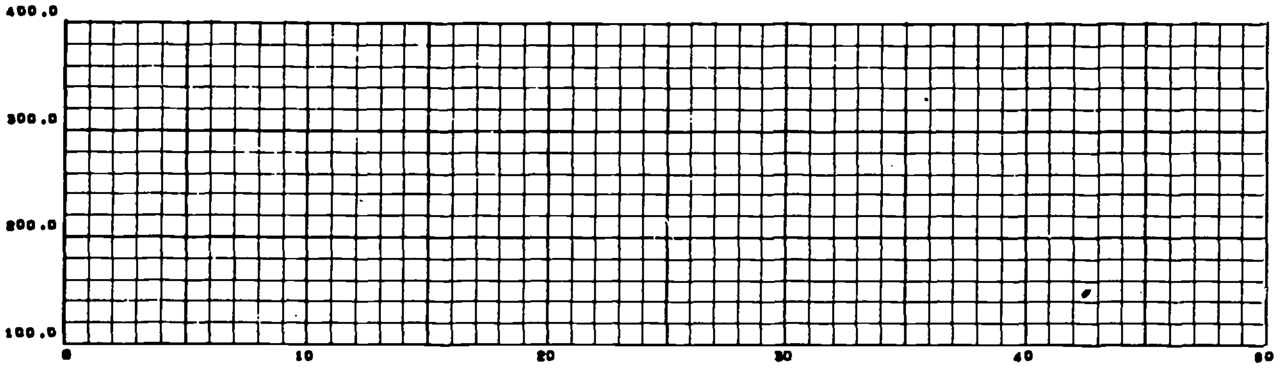
Initially, GRID1V uses BRITEV to ensure that the bright intensity mode is "on."

GRID1V then checks certain internal locations to obtain basic information, by calling the following subprograms:

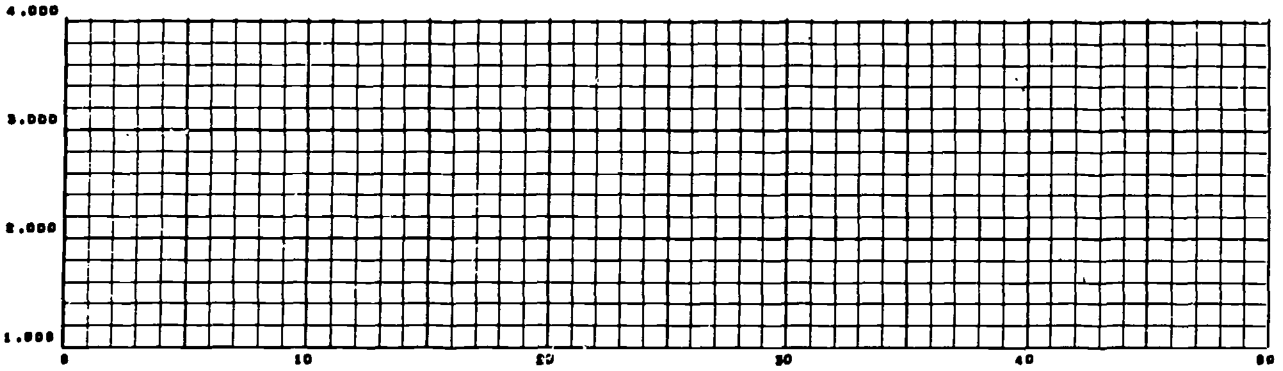
SETMOV	Retrieves margin assignments. The programmer may have changed the standard specifications by a CALL SETMIV.
SETCOV	Retrieves character size specifications that programmer may have changed by a CALL SETCIV.



```
CALL SETMIV ( 24. 0, 712, 28)
CALL GRIDIV (2, 0.0, 50.0, 10000.0, 40000.0, 1.0, 2000., 5, 5, 10, 5, 2, 5 )
```



```
CALL SETMIV ( 24, 0, 388, 371)
CALL GRIDIV ( 2, 0.0, 50.0, 100.0, 400.0, 1.0, 20.0, 5, 5, 10, 5, 2, 5)
```



```
CALL SETMIV ( 24, 0, 24, 715)
CALL GRIDIV ( 1, 0.0, 50.0, 1.0, 4.0, 1.0, 0.2, 5, 5, 10, 5, 2, 5)
```

Figure 5-11

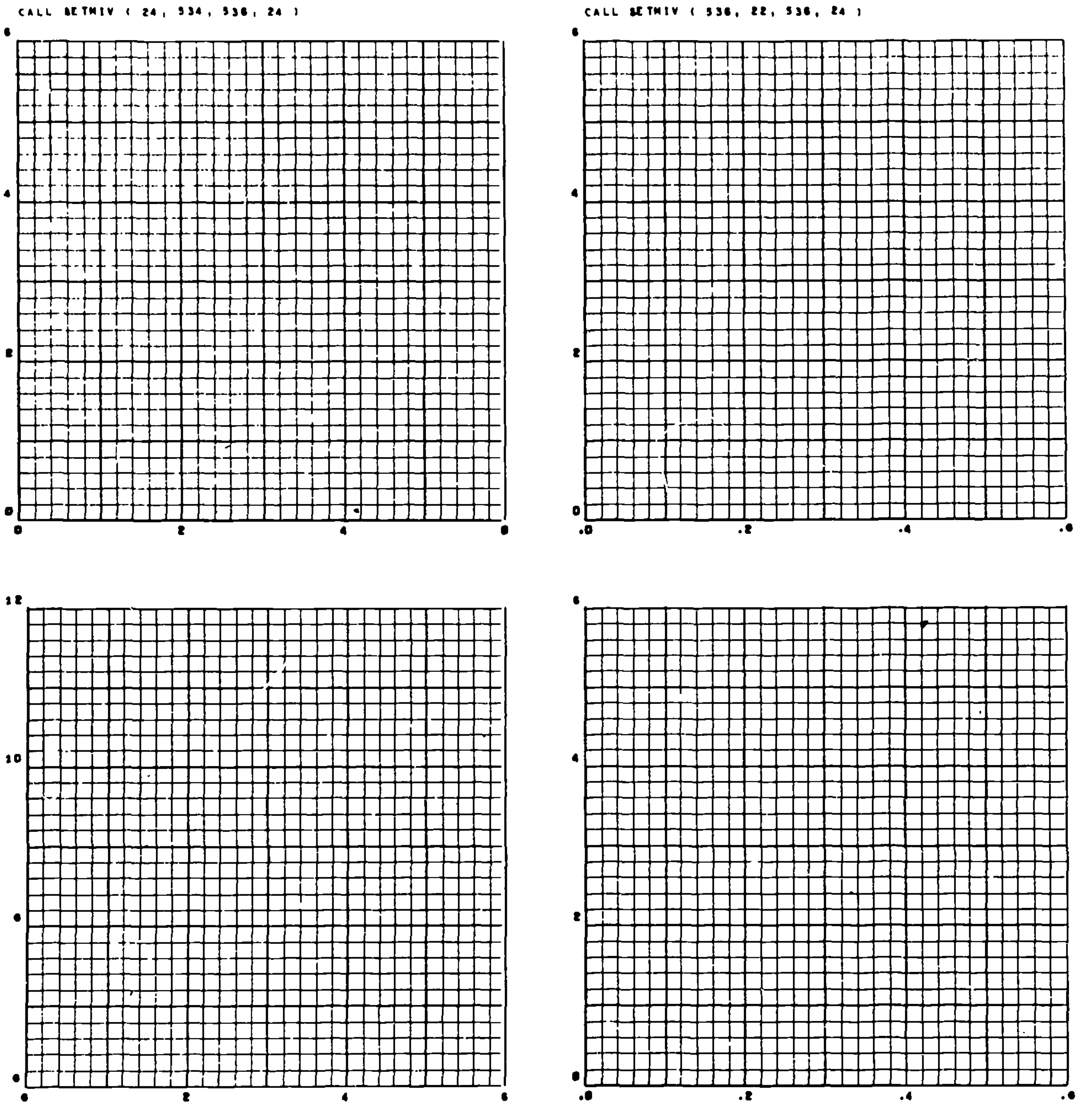
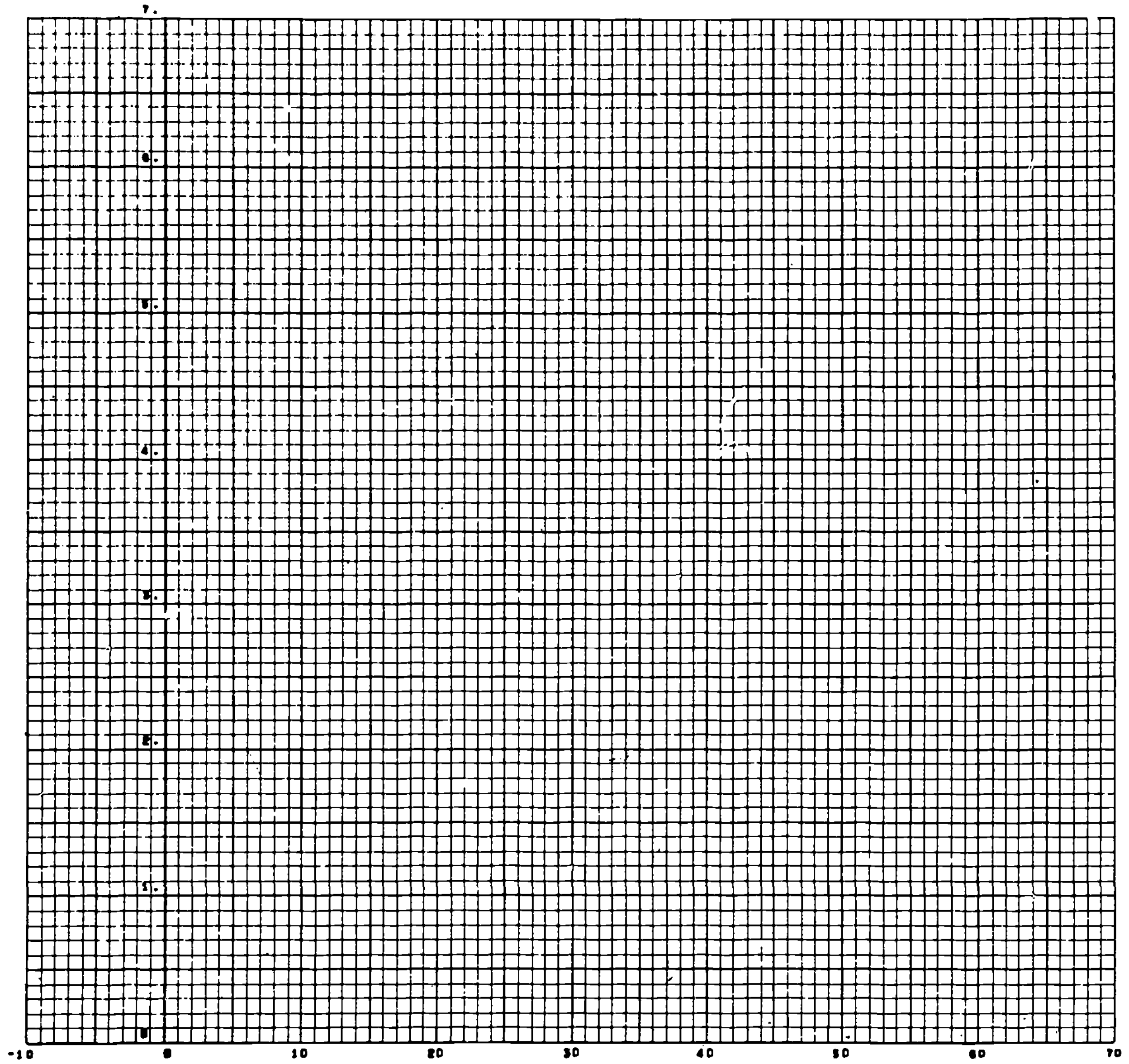

CALL SETMIV (24, 534, 536, 24)
6
4
2
0
0
2
4
6
CALL SETMIV (536, 22, 536, 24)
6
4
2
0
.0
.2
.4
.6
12
10
8
6
0
2
4
6
CALL SETMIV (24, 534, 24, 536)
6
4
2
0
.0
.2
.4
.6
CALL SETMIV (536, 22, 24, 536)

Figure 5-12

SPECIAL GRIDIV EXAMPLES

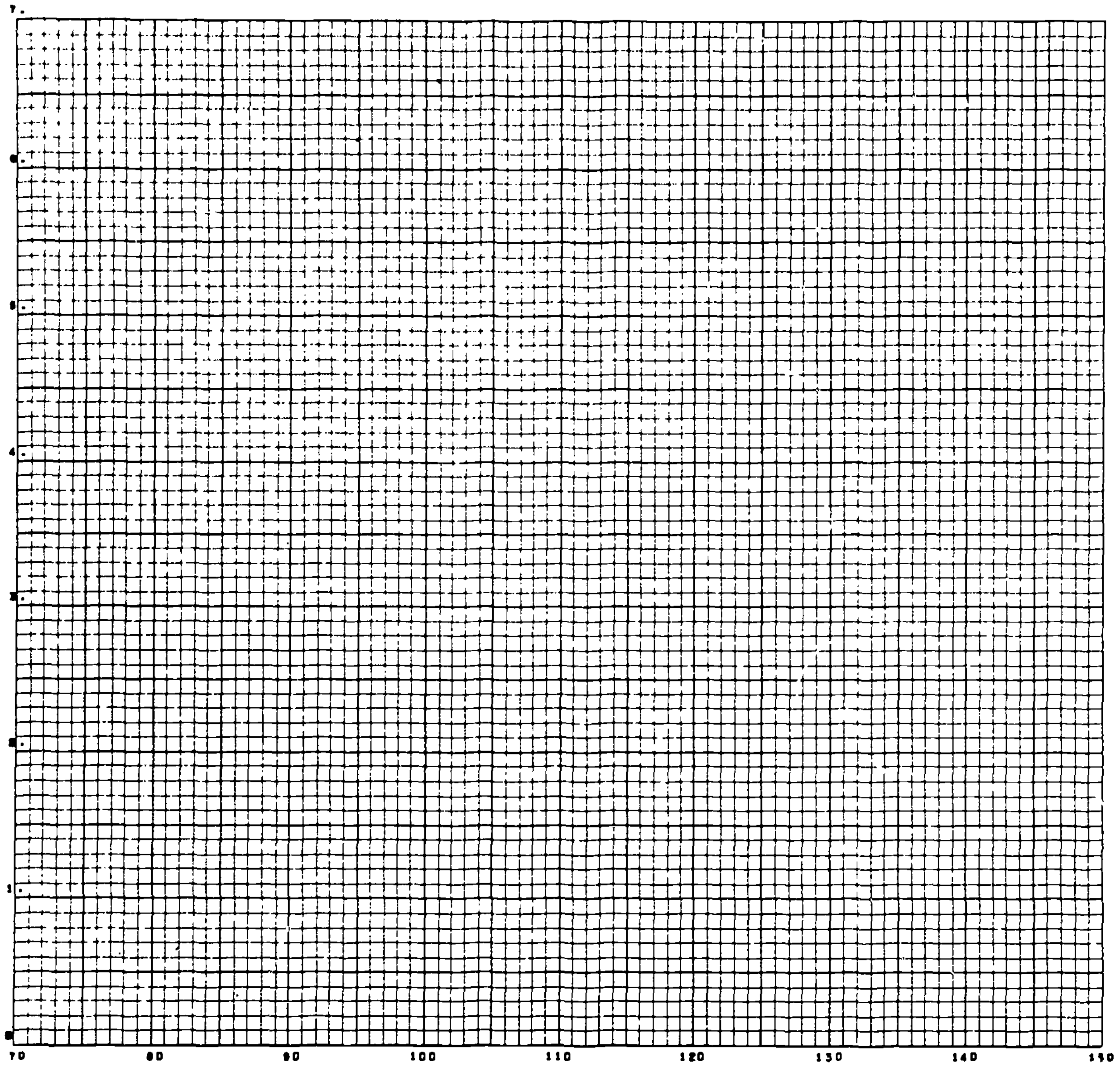
HOLDIV AS AN AID IN MAINTAINING THE SAME SCALE FROM ONE GRID TO ANOTHER



```
CALL GRIDIV ( 1, -10.0, 70.0, 0.0, 7.0, 1.0, 0.1, 5, 5, 10, 10, 2, 2 )
CALL HOLDIV(1)
```

THE HOLD STATEMENT WILL CAUSE LABEL MARGINS TO BE HELD FOR SUBSEQUENT GRIDS
UNTIL CALL HOLDIV(0) RELEASES THE HOLD

Figure 5-13



CALL GRIDIV (1., 70.0, 150.0, 0.0, 7.0, 1.0, 0.1, 5, 5, 10, 10, 5, 2)
NOTE...NOT ONLY HAS HOLDIV BEEN USED BUT ALSO XL,XR,YB,YT HAVE BEEN SPECIFIED SUCH THAT
XR-XL AND YT-YB OF ONE GRID EQUAL THEIR COUNTERPART IN THE OTHER GRID

Figure 5-14

SPECIAL GRIDIV EXAMPLES

USE OF HOLDIV WITHIN A FRAME

```
CALL SETMIV ( 24,599, 50, 573 )
CALL GRIDIV ( 1, -10.0, 90.0, 0.0, 9.0, 5.0, 1.0, 4, 2, -4, -2, 2, 1 )
CALL HOLDIV (1)
CALL SETMIV ( 524, 99, 50, 573 )
CALL GRIDIV ( 2, 90.0, 190.0, 0.0, 9.0, 5.0, 1.0, 4, 2, -4, -2, 3,1 )
```

NOTE... NOT ONLY MUST XR-XL AND YB-YT OF ONE GRID EQUAL THE CORRESPONDING QUANTITY IN THE OTHER GRID, BUT ALSO THE ARGUMENTS OF SETMIV MUST HAVE A SPECIFIC RELATIONSHIP........SAY THE ARGUMENT LIST IS (MTL,MTR,MTB,MTT) ,THEN THE VALUES 1023-(MTL+MTR) AND 1023-(MTB+MTT) OF ONE GRID MUST EQUAL THE CORRESPONDING QUANTITY IN THE OTHER GRID

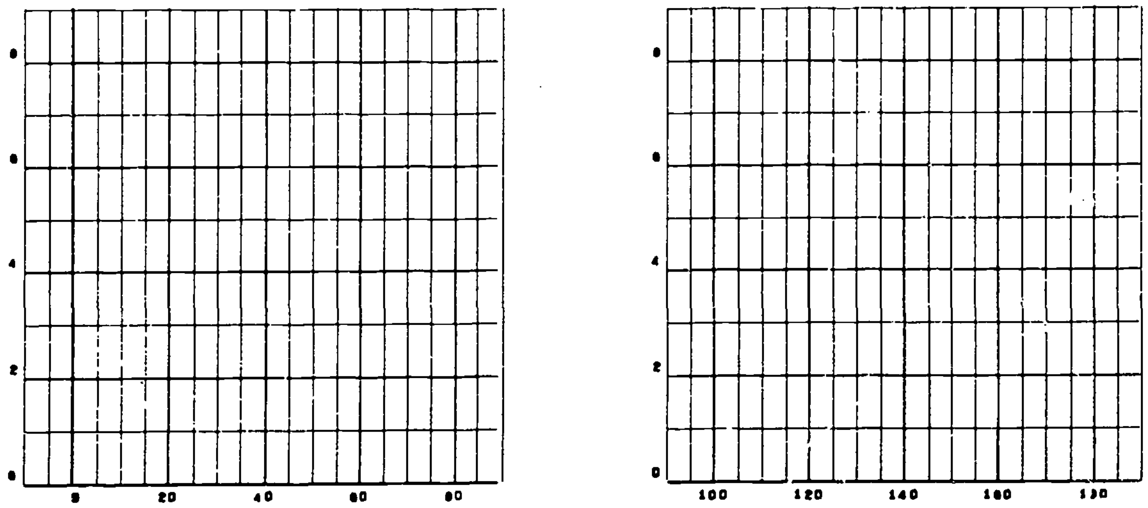


Figure 5-15

MSXYV	Determines linear or nonlinear mode. Indicators may have been set to nonlinear by a CALL SMXYV.
HOLDOV	Determines whether margin specifications were held over from preceding graph. Indicator may have been specified by a CALL HOLDIV.

Computations are made to determine the raster positions to be used as grid boundaries. ERRLNV and/or ERRNLV are called to check the boundaries and the data limits to see if a grid can be produced from this information. Finally, XSCALV and YSCALV are called to compute scale factors, and LINRV (in the linear mode) and/or NONLNV (in the logarithmic mode) are called to generate the grid.

DETERMINING GRID BOUNDARY POSITIONS

GRID1V sets aside the margin space specified by a prior call to SETMIV (or the standard margin space if SETMIV has not been called). With the exception of the small area in the upper right corner used for frame identification, GRID1V does no writing in these basic margins.

Then GRID1V tests to see if labels are to be placed (or may extend) outside the grid area. Space required for such labels is computed, taking into consideration the type of label (fixed point or scientific), the number of label characters specified in GRID1V arguments plus space for a sign, and the height and width of the label characters.

If any space is needed for labels at the left, right, bottom, or top of the grid, it is added to the basic margins specified by SETMIV table, to produce the total margins. (HOLDIV can alter this procedure by causing label spaces "held" from a previous grid to be added when computing the total margins).

If the option for a square grid is specified, the right, or top, total margin will be adjusted so the remaining area will be square.

If the total margins and the grid limits (XL, XR, YB, and YT) meet certain error tests, they are used as arguments of XSCALV and YSCALV in computing the scale factors required for generating the grid and plotting on it.

Since so many items influence the margin assignments, the grid boundaries will rarely fall at the exact raster positions that the programmer might have estimated. If the precise raster positions of the boundaries are required in a program, they should be derived by converting the limits into raster counts after GRID1V has been called. For example:

IXL = NXV (XL)

IXR = NXV (XR) (The right total margin is 1023 - IXR)

IYB = NYV (YB)

IYT = NXV (YT) (The top total margin is 1023 - IYT)

GRID1V ERROR PROCEDURE

Even if bad input data is used, GRID1V will attempt to produce a grid. The philosophy is that some useful information may be revealed, even if the grid is inaccurate.

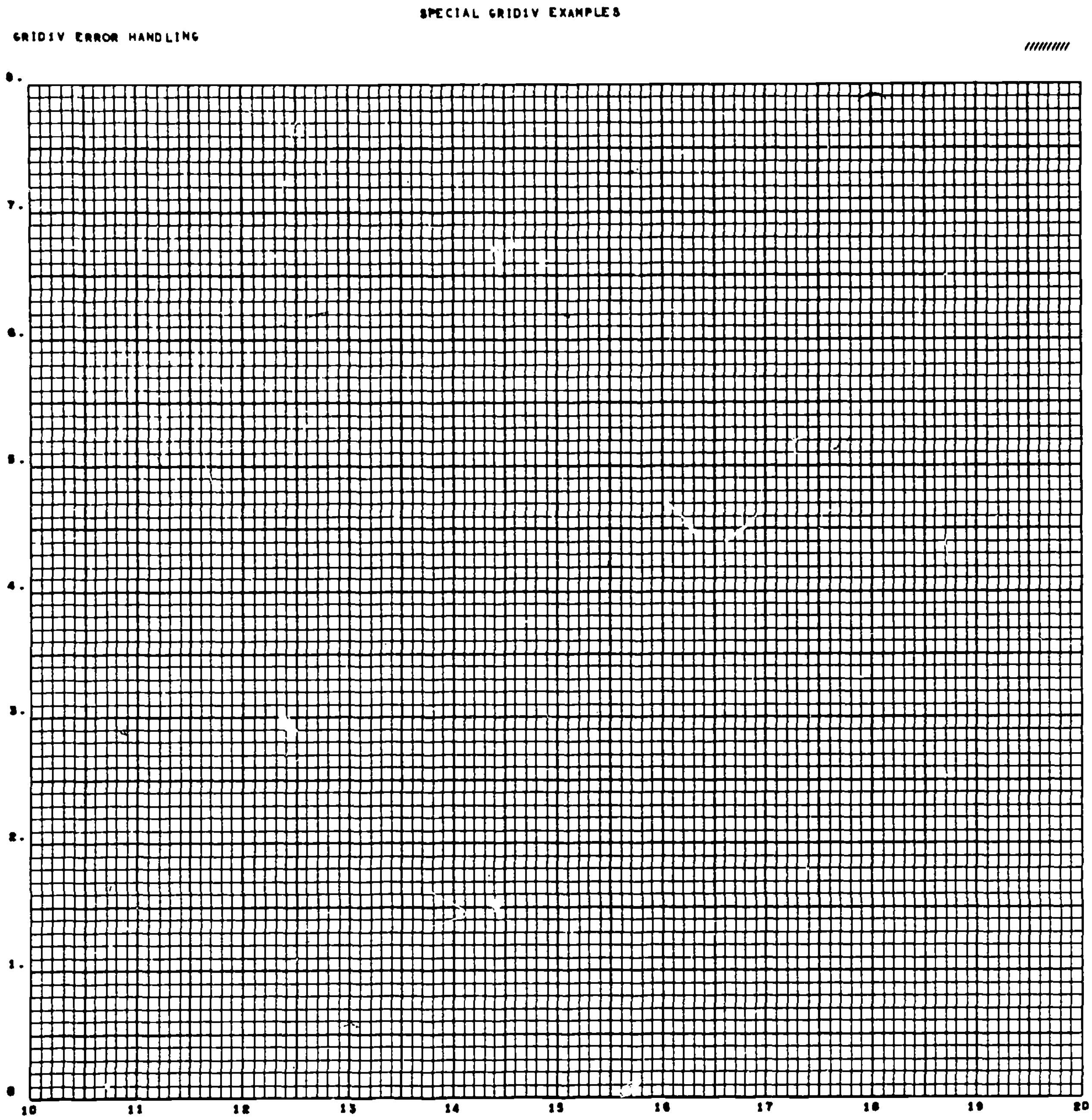
The grid limits and total margins are tested by lower-level subprograms, ERRLNV (for linear mode) and/or ERRNLV (for log mode). If these tests show that a grid cannot be produced from the given information, some data values are manufactured so that the program can continue. The manufactured quantities are used only internally; data values in the main program will not be affected. (Since ERRLNV and ERRNLV are meant to be used only by GRID1V, the call statements are not given.)

When artificial quantities are used, an error mark (/////) is placed in the upper right corner of the frame by ERMRKV. Although ERMRKV was designed for use by GRID1V, the programmer can place this mark on the frame if he uses the statement

```
CALL ERMRKV
```

An error mark from GRID1V may indicate that one or more of the following errors has been found.

1. Equal values have been specified for grid limits; that is, XL = XR and/or YB = YT. See Figure 5-16.
2. The specified margins are too wide. In other words, MR + ML - 1023, and/or MB + MT - 1023. See Figure 5-17.
3. In a linear grid, the specified values of DX and/or DY would result in grid lines spaced closer than 3 raster counts.
4. In a log grid the value of one or more of the limits is either zero or negative.
5. In the log mode, there are more than 10 cycles in either the X or Y direction.

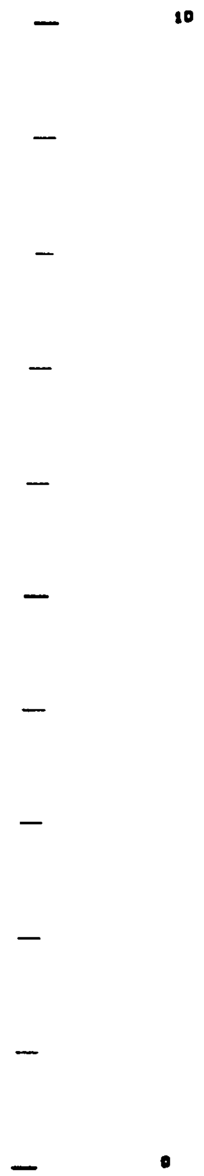


```
CALL GRIDIV ( 1, 10.0, 10.0, 0.0, 8.0, 0.1, 0.1, 5, 5, 10, 10, 2, 2 )
```

Figure 5-16

GRIDIV ERROR HANDLING

SPECIAL GRIDIV EXAMPLES



```
CALL SETMIV ( 600, 600, 48, 48 )
CALL GRIDIV ( 1, 0.0, 10.0, 0.0, 10.0, 1.0, 1.0, 5, 5, 10, 10, 2, 2 )
```

Figure 5-17

Examples of GRID1V Characteristics

Certain special characteristics of the grid generated by GRID1V are shown in the following examples.

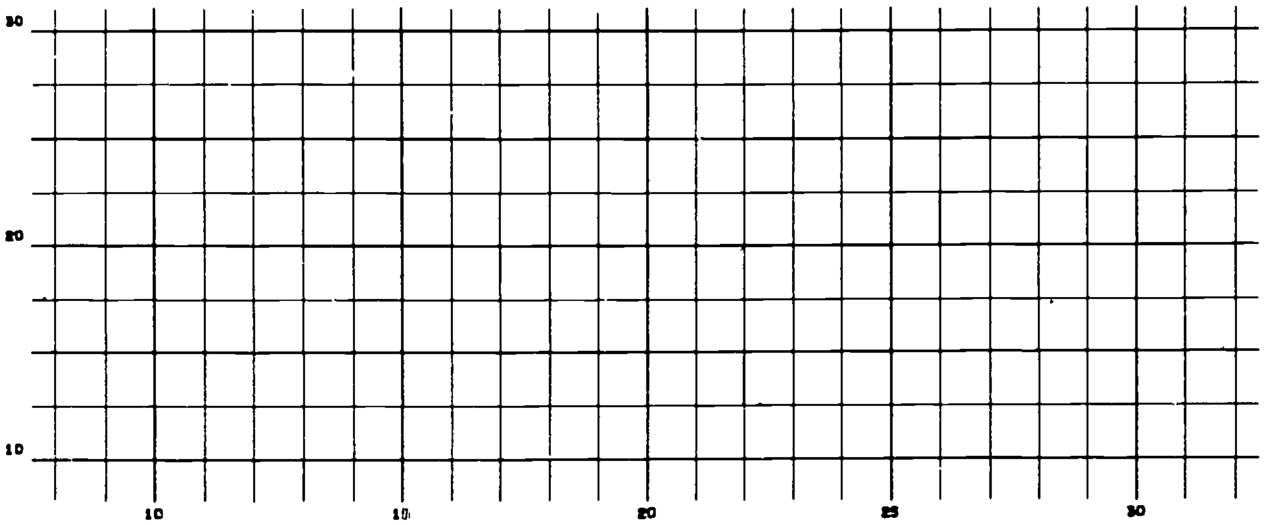
Example 1. During construction of a linear grid, vertical lines are displayed at intervals "stepped off" from X = 0 in DX increments until the maximum limit of the grid is exceeded. Then they are "stepped off" from X = 0 in the negative direction until the minimum limit of the grid is passed. Similarly, horizontal lines are "stepped off" from Y = 0 in DY increments. Because of this method of construction, lines will be generated at the grid limits only if XL and XR are integer multiples of DX, and if YB and YT are integer multiples of DY. Figures 5-18 and 5-19 illustrate the effect of changing DX and DY. Figures 5-20 and 5-21 illustrate a similar effect in the negative range of values.

Example 2. Vertical lines to be emphasized are "stepped off" from X = 0 by increments of N·DX, and vertical lines to be labeled are "stepped off" by increments of I·DX. Similarly, horizontal lines to be emphasized are "stepped off" from Y = 0 by increments of M·DY, and those to be labeled by increments of J·DY. This procedure means that labels and/or emphasized lines will not necessarily occur at the grid limits. Lines at XL and XR will be emphasized only if XL and XR are integer multiples of N·DX, and labeled only if XL and XR are integer multiples of I·DX. In the same way, lines at YB and YT are integer multiples of M·DY, and labeled only if they are integer multiples of J·DY.

This method of determining the positions of line labels is responsible for the absence of labels on the limit lines in Figures 5-19 and 5-21. Notice in the example that the labels for vertical grid lines are adequate; there is no real need to provide labels on the lines at XL and XR. However, the labels for the horizontal lines in this illustration are not adequate; they demonstrate another factor to be considered in planning for line labels. To be meaningful, labels should appear on at least two lines. In the example, the use of J = 5, instead of J = 10, would cause labeling of the lines of Y = 25.0, 30.0, and 35.0, giving a scale that can be read easily.

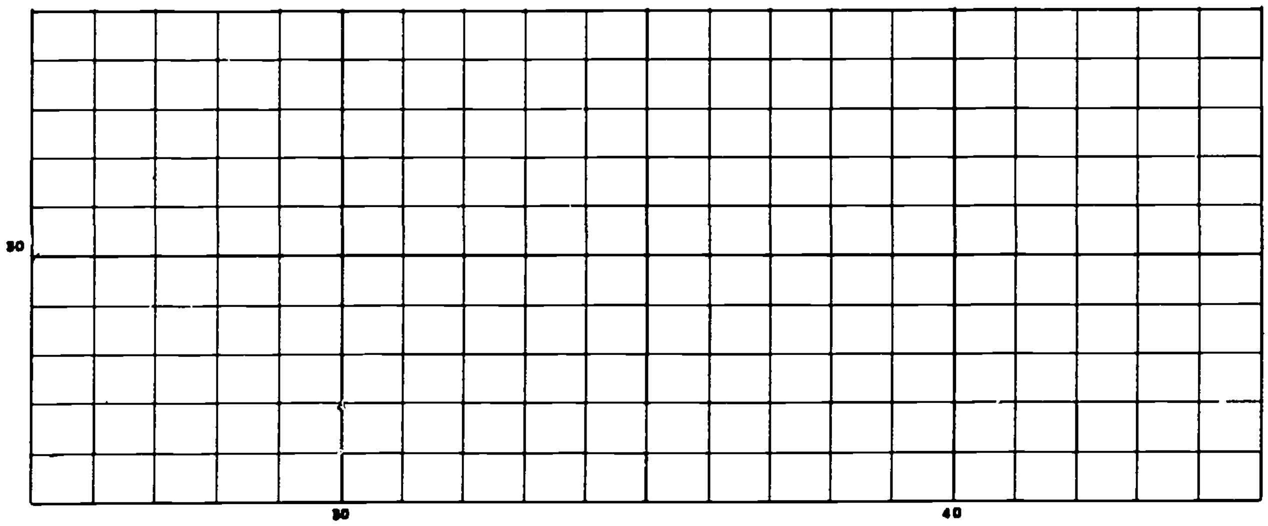
SPECIAL GRIDIV EXAMPLES

EFFECTS OF STEPPING OFF DELTA INCREMENTS FROM ZERO



```
CALL GRIDIV ( 1, 7.5, 32.5, 8.0, 31.0, 1.0, 2.5, 5, 2, 5, 4, 2, 2 )
```

Figure 5-18

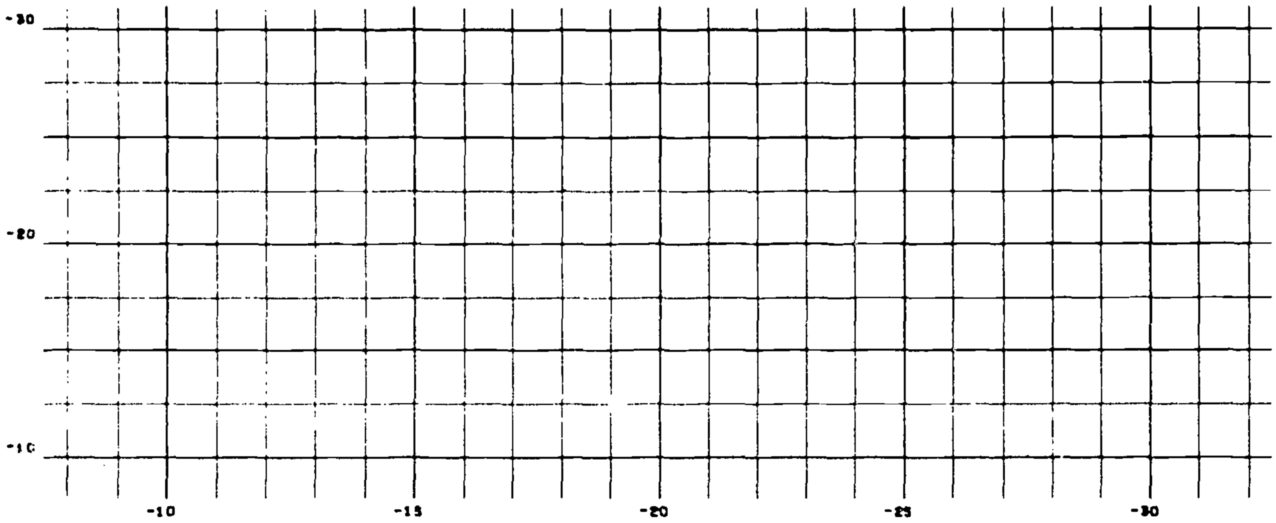


```
CALL GRIDIV ( 2, 25.0, 45.0, 25.0, 35.0, 1.0, 1.0, 5, 5, 10, 10, 2, 2 )
```

Figure 5-19

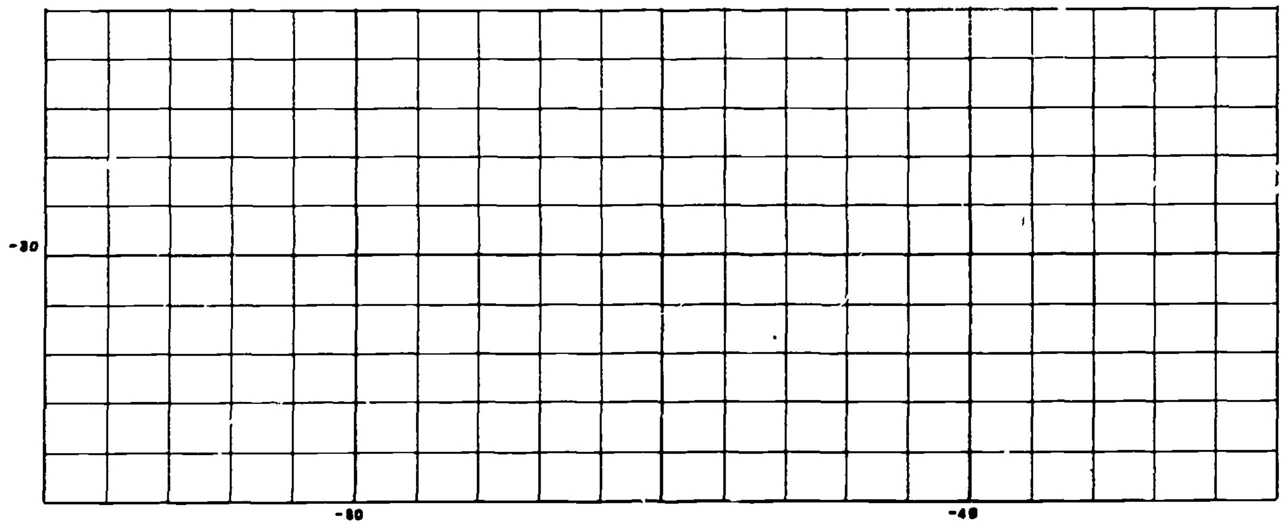
SPECIAL GRIDIV EXAMPLES

THE NEGATIVE RANGE OF VALUES IS HANDLED IN A SIMILAR WAY



```
CALL GRIDIV ( 1, -7.5, -32.5, -8.0, -31.0, 1.0, 2.5, 5, 2, 5, 4, 2, 2 )
```

Figure 5-20



```
CALL GRIDIV ( 2, -25.0, -45.0, -25.0, -35.0, 1.0, 1.0, 5, 5, 10, 10, 2, 2 )
```

Figure 5-21

Example 3. Under certain conditions, GRID1V purposely omits labels. Figure 5-22 shows how this can happen. The limits of X used for the grid were -99.9999 and 99.9999, and the limits of Y were 0.0 and 9.9999. During scaling, these limits were rounded, causing them to be treated as -100.0, 100.0 and 0.0, 10.0.

This process caused the decimal scales of the grid limits to be larger than the initial limits, i.e., the values of NX and NY specified by the programmer were not compatible with the new limits. To avoid erroneous labeling, label values will not be displayed if the rounding process causes the decimal scale to be larger than the initial value.

The system does not adequately provide for all problems of this type. Such complications can be avoided if the programmer considers the rounded values of XL and XR, and YB and YT, and when specifying NX and NY.

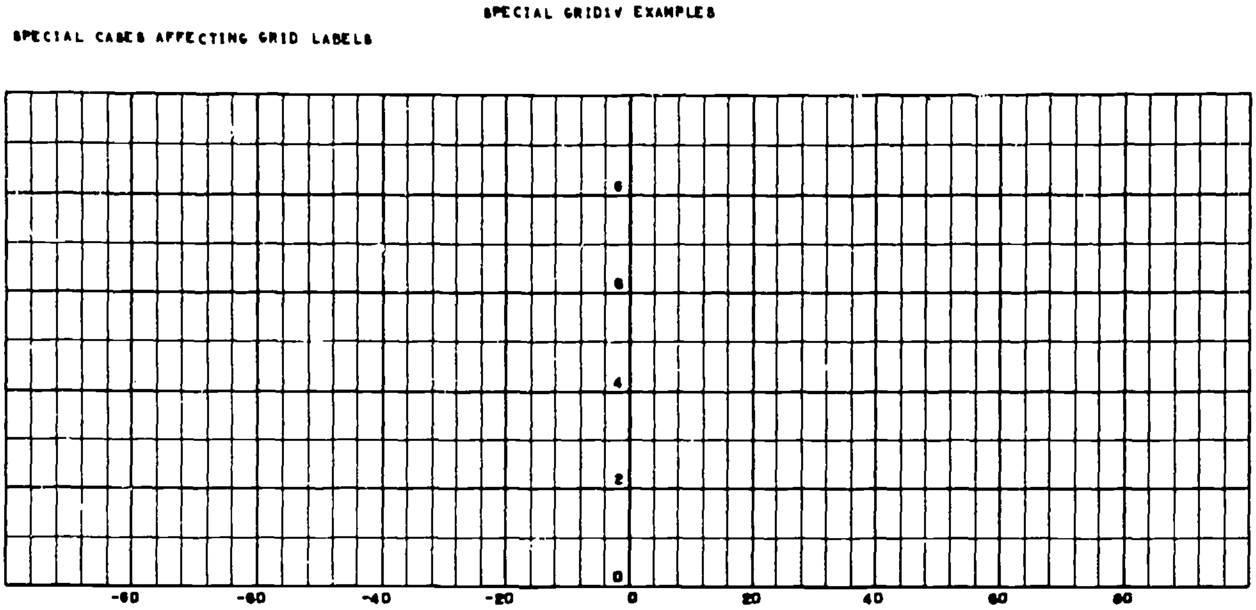
Example 4. When X = 0 (and/or Y = 0) lies within the grid limits, and I (and/or J) is positive, GRID1V places labels along the X = 0 (or Y = 0) line. If there isn't enough space for the label between X = 0 and the left limit of the grid (or between Y = 0 and the bottom of the grid), GRID1V will provide space outside the grid area for labels, just as if negative values of I (and/or J) had been specified. An illustration is shown in Figure 5-23.

Example 5. The number of label characters, NX (or NY), specified for fixed point labels should satisfy the largest and smallest label values to be displayed. Normally, the quantity should be the sum of: (1) the decimal scale of the largest value of X (or Y), (2) the number of fractional positions required in the smallest value, and (3) one more position to provide for the decimal point, if required. The total quantity may not exceed 6 (or 7 if the decimal point is included).

Figure 5-24 illustrates how trouble can occur when NX and/or NY are not specified properly. Note that the values of NX = 1 and NY = 1 are adequate for the largest values of X and Y, but do not allow for the fractional values required in some of the labels. In this example, 3 should have been used for the values of NX and NY, in the top graph.

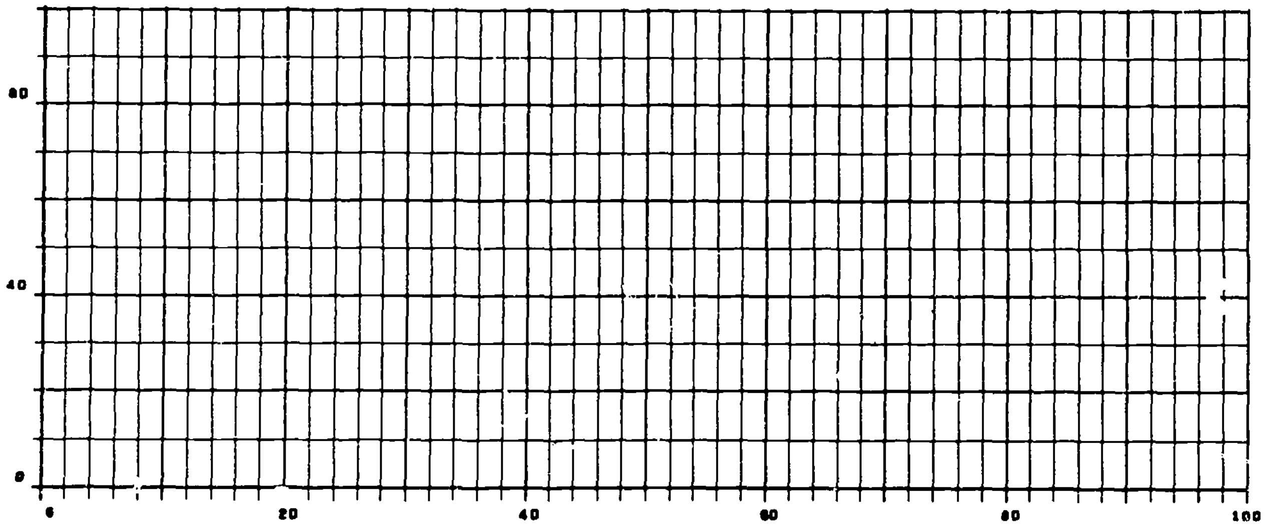
For some applications, it may be advantageous to specify values of NX and/or NY less than the decimal scale of the largest label value. In the middle grid of Figure 5-24, the two low-order positions in the labels have been dropped intentionally, by using values of NX and NY that are less than the decimal scales of the largest values of X and Y. In this case, the resulting graph has been effectively rescaled.

This procedure must be used with care. In the bottom grid, Figure 5-24, the values of NX and NY are so small that the increment between labeled lines is not reflected in the labels that are displayed.



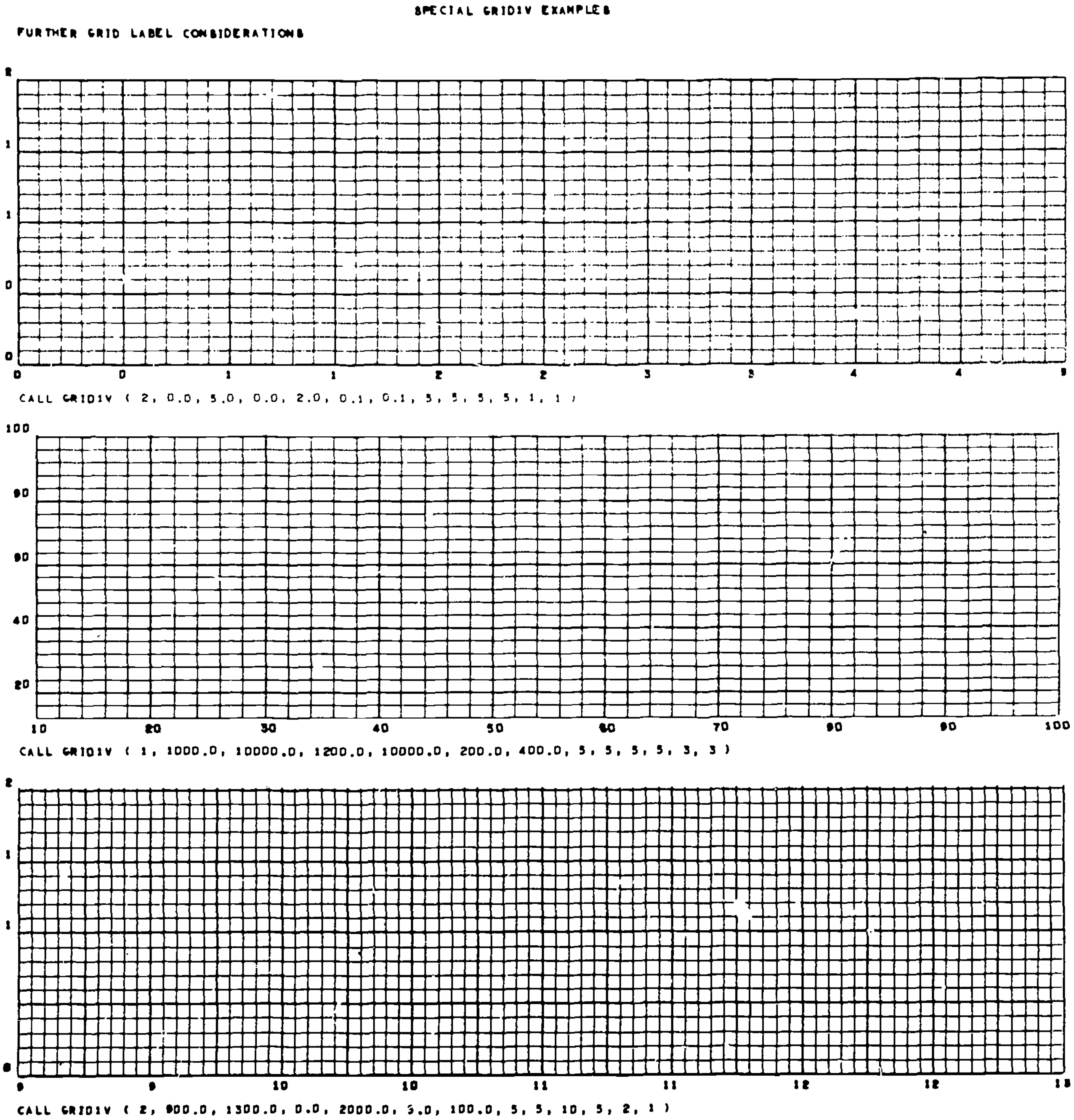
```
CALL GRIDIV ( 1, -99.9999 , 99.9999 , 0.0, 9.9999, 4.0, 1.0, 5, 2, 5, 2, 2, 1 )
```

Figure 5-22



```
CALL GRIDIV ( 2, -0.7, 100.0, -3.0, 100.0, 2.0, 10.0, 5, 2, 10, 4, 3, 3 )
```

Figure 5-23



SPECIAL GRIDIV EXAMPLES

FURTHER GRID LABEL CONSIDERATIONS

```
CALL GRIDIV ( 2, 0.0, 5.0, 0.0, 2.0, 0.1, 0.1, 5, 5, 5, 5, 1, 1 )
```

```
CALL GRIDIV ( 1, 1000.0, 10000.0, 1200.0, 10000.0, 200.0, 400.0, 5, 5, 5, 5, 3, 3 )
```

```
CALL GRIDIV ( 2, 900.0, 1300.0, 0.0, 2000.0, 5.0, 100.0, 5, 5, 10, 5, 2, 1 )
```

Figure 5-24

LOG AND SEMI-LOG GRIDS

The earlier examples used only linear scaling and conversion. The modal subroutine SMXYV can be used to alter the scaling mode such that scaling and conversions will be made in the logarithmic mode.

The call statement for establishing the logarithmic mode is:

```
CALL SMXYV (MX, MY)
```

where MX and MY are scale mode indicators that designate whether the logarithmic or linear mode is to be used:

If $MX \neq 0$, $MY \neq 0$ Log in X, log in Y

$MX \neq 0$, $MY \neq 0$ Log in X, linear in Y

$MX = 0$, $MY \neq 0$ Linear in X, log in Y

$MX = 0$, $MY = 0$ Linear in both X and Y
(to restore linear mode)

At the beginning of each job, MX and MY are zero, so that linear scaling and conversion result if SMXYV is never called.

If the programmer wants to generate a log or semi-log plot, he must call SMXYV to set the logarithmic mode before using any OLMGS subroutines that involve scaling or conversion.

Once the log-log or semi-log mode has been set by SMXYV and the scale factors have been established, the function statements NXV and NYV can be used to convert data coordinates into raster coordinates, in the same manner as shown for linear scaling.

The contents of the scale mode indicators may be retrieved by using the following statement:

```
CALL MSXYV (MXL, MYL)
```

The indicator for the X scale mode will be stored in MXL, and for the Y scale mode in MYL.

GRID1V uses this statement to determine what scale mode has been selected by the programmer.

RESTRICTIONS ON LOGARITHMIC MODE

In general, once SMXYV has been called, the programmer can use any of the routines to generate a logarithmic display. (One exception: DXDYV should not be used to generate arguments for GRID1V in the direction in which logarithmic scaling is being used.)

However, some of the GRID1V arguments are restricted in the logarithmic mode. Following is a list of the 13 arguments, with notations as to the arguments affected if the mode is logarithmic in the direction affected by each.

```
CALL GRID1V (L, XL, XR, YB, YT, DX, DY,
             ±N, ±M, ±I, ±J, NX, NY)
```

L Controls margin calculations.

XL, XR Left and right limits of the grid. May not be negative or zero.

YB, YT Bottom and top limits of the grid. May not be negative or zero.

DX, DY Should be set to 1.0. If less, only the cycle lines will be displayed. If greater, no lines will be drawn.

N, M Will be ignored; however, an argument must be present. A negative sign on N or M will force the grid to the square.

I, J Will be ignored; however, an argument must be present.

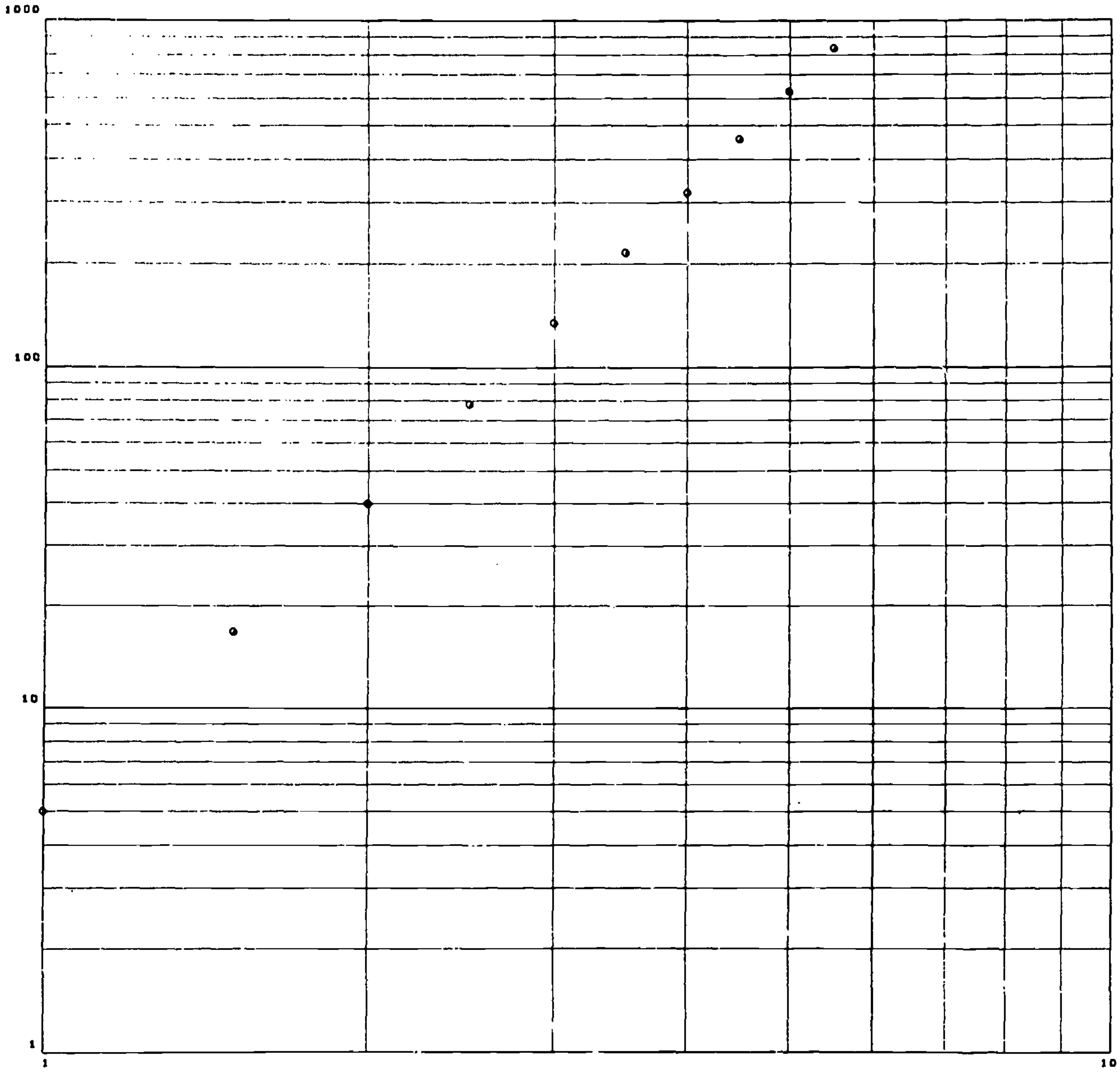
NX, NY Number of characters to be displayed in the labels.

Generally, line labels will be placed only at the cycle lines. If, however, the grid spans less than one complete cycle, each grid line will be labeled.

No more than 10 log cycles are permitted in the GRID1V system.

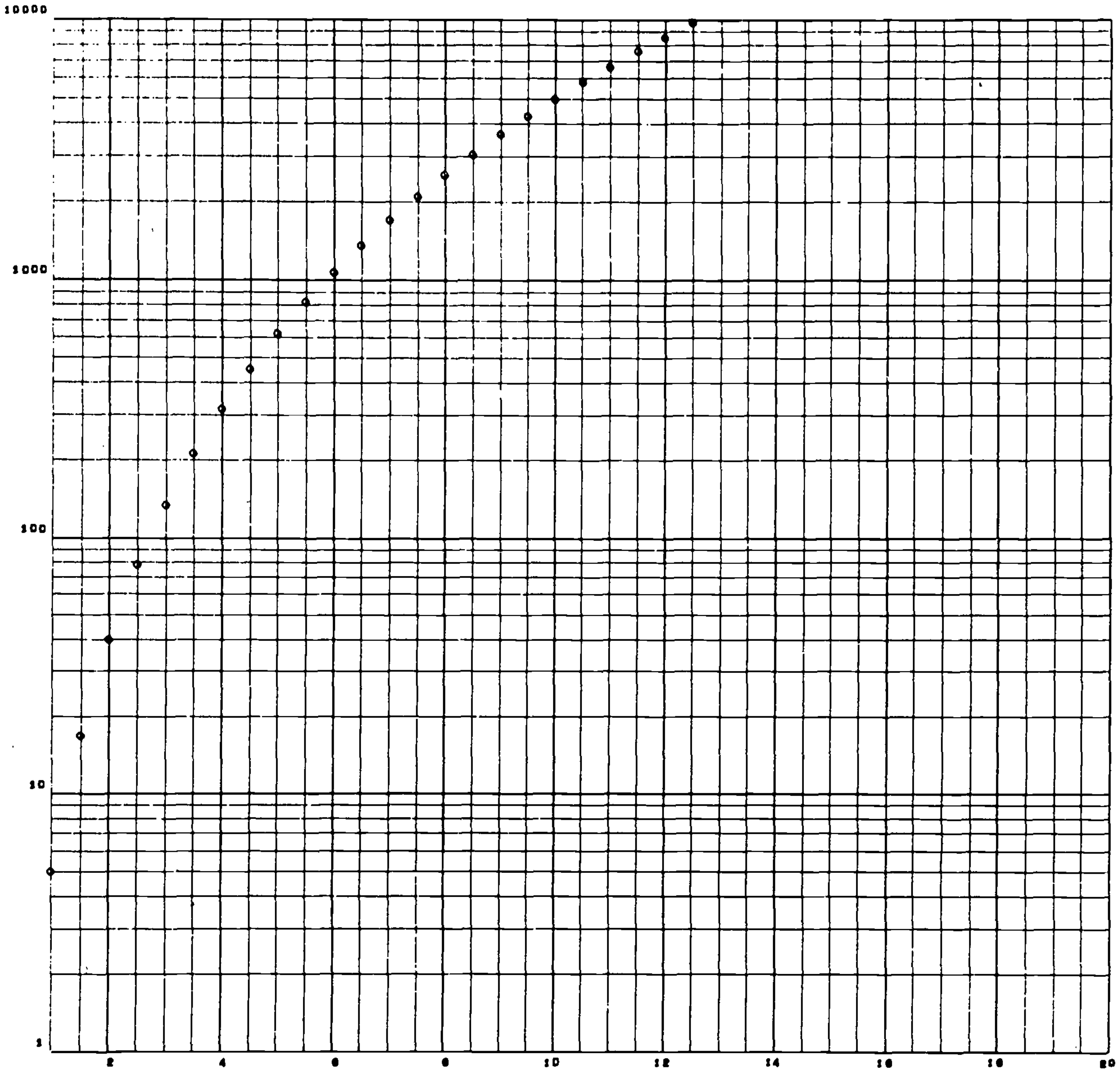
Examples. Figure 5-25 illustrates a plot that is logarithmic in both the X and Y directions. Note the arguments for SMXYV and GRID1V in the coding.

Figure 5-26 illustrates a semi-log grid, with plotting of the data used in Figure 5-25.



```
CALL SMXYV ( 1, 1 )
CALL GRIDIV ( 1, 1.0, 10.0, 1.0, 1000.0, 1.0, 1.0, 1, 1, 1, 1, 2, 4 )
          POINTS PLOTTED BY USING CALL POINTV ( X,Y, 1 )  IN A LOOP
```

Figure 5-25



```
CALL SMXYV ( 0, 1 )
CALL DXDYV ( 1, 1.0, 20.0, DX, N, I, NX, 12.0, IERR )
CALL GRID1V ( 1, 1.0, 20.0, 1.0, 10000.0, DX, 1.0, N, 1, I, 1, NX, 5 )
```

Figure 5-26

BUILDING SPECIAL GRIDS: LINRV, NONLNV

Two of the lower-level modules employed by GRID1V are useful as building blocks for building special grids. LINRV may be used to generate only the vertical portion or only the horizontal portion of a linear grid. NONLNV can generate only the vertical or only the horizontal portion of a log grid. See examples in Figure 5-27 and Figure 5-28.

In addition to generating the vertical and horizontal portions of a grid in separate operations, these modules offer other special capabilities:

1. The programmer can control the length of the grid lines. For example, the grid lines can be as short as "time tics."

2. The programmer can exert greater control over the position of line labels by specifying a label reference location.

3. Selected grid lines can be emphasized and/or labeled in the same way as with GRID1V.

4. DXDYV can be used with LINRV in much the same way it is used with GRID1V.

Certain subprograms must be executed prior to the use of LINRV or NONLNV. The frame must have been advanced, and XSCALV and/or YSCALV must have established scale factors. If a change in scale mode is required, SMXYV must have been called.

The call statement for using LINRV to generate a vertical grid is

```
CALL LINRV (1, LYREFR, IYMIN, IYMAX, XL, XR,
           DX, N, I, ±NX, IW)
```

For a horizontal grid, the statement is

```
CALL LINRV (2, LXREFR, IXMIN, IXMAX, YB, YT,
           DY, M, J, ±NY, IH)
```

LYREFR
LXREFR — Label reference locations.
For a vertical grid, LYREFR is the Y raster coordinate that will be used to position the labels of the vertical grid lines. The X raster coordinate will vary, and will be computed by the subprogram.

For a horizontal grid, LXREFR is the X raster coordinate that will be used to position the labels of the horizontal grid lines. The Y raster coordinates will vary, and will be computed by the subprogram.

The character dimensions, IW and IH (below), should be considered in assigning LYREFR and LXREFR. For example, LYREFR should be chosen to allow at least one character height (IH) below the origin of vertical grid lines (IYMIN). LXREFR should be a raster position that is at least IW* (NY + 1) raster counts to the left of the origin of the horizontal grid lines (IXMIN). In both instances, some additional space should be added to prevent overlapping.

If scientific labels are selected (-NX and/or -NY), provision must be made for 7 additional label characters in the computation of LXREFR. About 5 raster counts extra must be added to LYREFR to allow for the raised exponent.

NOTE: IW* (NY + 1) allows space for a sign μ.

IYMIN, IYMAX
IXMIN, IXMAX — Raster positions that determine the origin and end of each line. For a vertical grid, these integers are raster counts in the Y direction. For a horizontal grid, these integers are raster counts in the X direction.

IW, IH The dimension to be allowed for each label character: IW for the width, and IH for the height. Since LINRV uses LABLV to display the labels, and since the system LABLV uses PDP-10 characters, the dimensions should be IW = 8 and IH =12 . (If the standard values have not been altered, these values can be obtained by calling SETCOV).

The remainder of the arguments have the same definitions given for these terms in the discussion of GRID1V.

The vertical portion of a log grid can be generated by the statement

```
CALL NONLNV (1, LYREFR, IYMIN, IYMAX, XL, XR, DX,
             N, I, NX, IW)
```

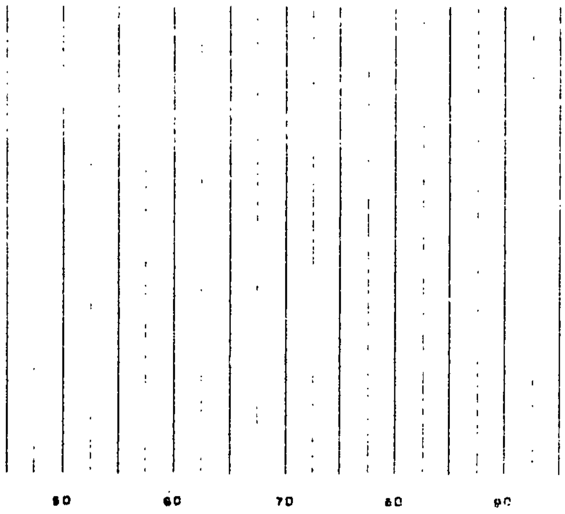
For a horizontal log grid, the statement is

```
CALL NONLNV (2, LXREFR, IXMIN, IXMAX, YB, YT, DY,
             M, J, NY, IH)
```

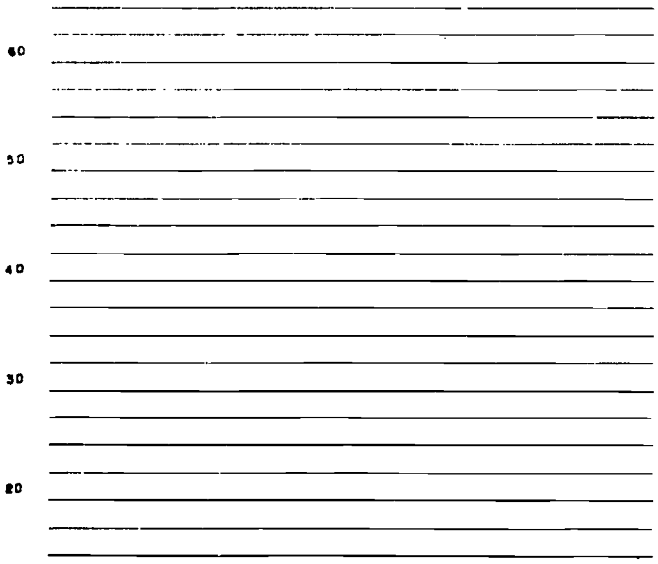
The arguments XL, SR, DX, N, I, NX, and YB, YT, DY, M, J, NY have the same definitions (and restrictions) given for these terms in the discussion of GRID1V for the log mode. The remaining arguments have the definitions given for LINRV.

Figures 5-27 and 5-28 show examples of portions of grids created by LINRV and NONLNV, respectively.

EXAMPLES OF USE OF LINRV



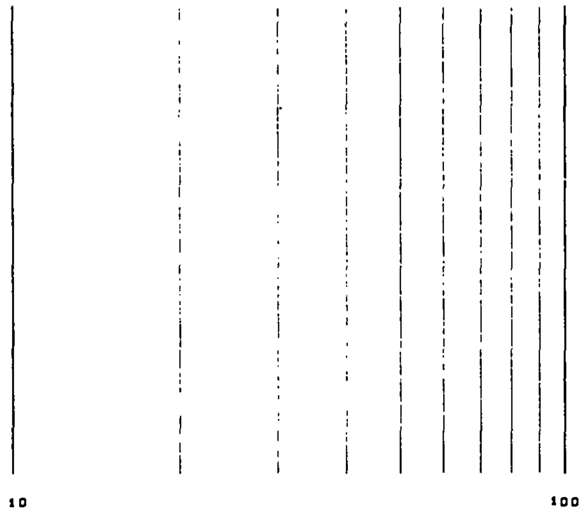
```
CALL XSCALV ( 45.0, 95.0, 50, 544 )
CALL LINRV ( 1, 515, 535, 900, 45.0, 95.0, 2.5, 2, 4, 2, 8 )
```



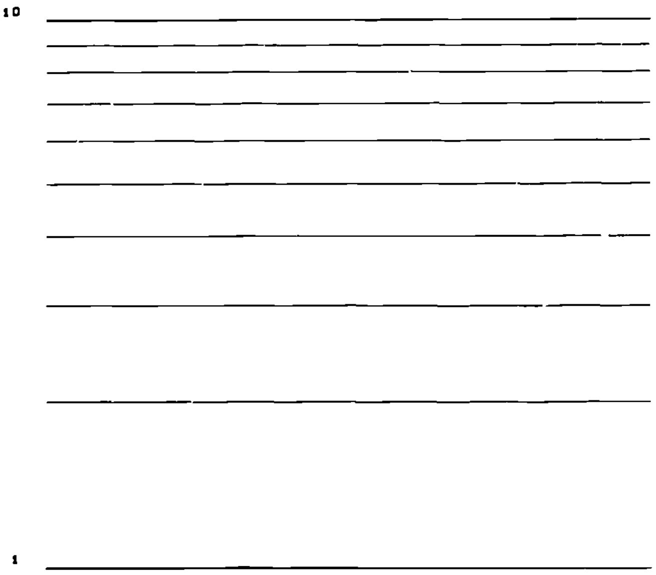
```
CALL YSCALV ( 15.0, 65.0, 20, 575 )
CALL LINRV ( 2, 520, 550, 1023, 15.0, 65.0, 2.5, 2, 4, 2, 12 )
```

Figure 5-27

EXAMPLES OF USE OF NONLNV



```
CALL BNXYV ( 1, 0)
CALL XBCALV ( 10.0, 100.0, 50, 544 )
CALL NONLNV ( 1, 515, 535, 900, 10.0, 100.0, 1.0, 1, 1, 3, 8 )
```



```
CALL BNXYV ( 0, 1)
CALL YBCALV ( 1.0, 10.0, 20, 575 )
CALL NONLNV ( 2, 520, 550, 1023, 1.0, 10.0, 1.0, 1, 1, 2, 8 )
```

Figure 5-28

AXIS LINES: XAXISV, YAXISV

Axis lines XAXISV, YAXISV enable the programmer to use the axis line feature of the DEC 340 to generate horizontal or vertical lines. Horizontal axis lines started at a specified raster position will be swept to the right to the specified X stop point in raster count. Vertical axis lines started at a specified raster position will be swept upwards to the specified Y stop point in raster count. The axis generator will not sweep lines down or to the left; therefore, the program automatically interchanges the coordinates if it is required in order not to stop the DEC 340 . If no stop point is given in the parameter list, the axis will be swept to the right margin or the top margin.

The call statement for sweeping a horizontal line is:

```
    CALL XAXISV (IX, IY)

or  CALL XAXISV (IX, IY, NSTPT)
```

IX, IY The fixed point raster coordinates of the origin of the line. IX may have a value from 0 to 959, while IY may have a value from 0 to 959.

NSTPT The fixed point X coordinate of the stop point.

For a vertical line, the statement is:

```
    CALL YAXISV (IX, IY)

or  CALL YAXISV (IX, IY, NSTPT)
```

IX, IY The fixed point raster coordinates of the origin of the line. IX may have a value from 0 to 959, while IY may have a value from 0 to 959.

NSTPT The fixed point Y coordinate of the stop point.

Section VI

SCALING AND CONVERSION

In earlier sections, scaling and conversion problems have been left to the routine GRID1V. However, much of the actual computation is done in lower-level modules. This section describes these and some associated modules that provide additional tools for some scaling and conversion problems.

The descriptions may also be useful in clarifying the operation of higher-level routines. For example, GRID1V uses XSCALV, YSCALV as a lower-level routine to do scaling. Consequently, the comments in this section concerning scaling and conversion equations, scale factors, and retrieving and resetting scale factors also apply when the programmer uses GRID1V to control scaling.

Methods of operation in both the linear and nonlinear modes are discussed. The nonlinear mode built into the system is the logarithmic mode, but the possibility of substitution of other nonlinear modes is mentioned.

BASIC SCALING SUBPROGRAMS: XSCALV, YSCALV

XSCALV, YSCALV will compute the scale factors for a specified display and store them in an internal table for later use by those functions which convert data. The calling statements are:

```
CALL XSCALV (XL, XR, ML, MR)

CALL YSCALV (YB, YT, MB, MT)
```

XL, XR — Floating point values of X for the leftmost and rightmost limits of the scaled plotting area.

ML, MR — The amount of margin space to be reserved to the left and right of the scaled area, expressed in raster counts (fixed point integers).

YB, YT Floating point values of Y for the bottom and top limits of the scaled plotting area.

MB, MT The amount of margin space to be reserved below and above the scaled area, expressed in raster counts (fixed point integers).

XSCALV, YSCALV contains a test for nonlinear mode. If this mode is indicated, XL, XR and/or YB, YT will be transformed before the scale factors are computed by the basic scaling equations.

Example

Figure 6-1 illustrates the relationship of the arguments. The margin specifications are: ML = 170, MR = 192, MB = 340, MT = 128.

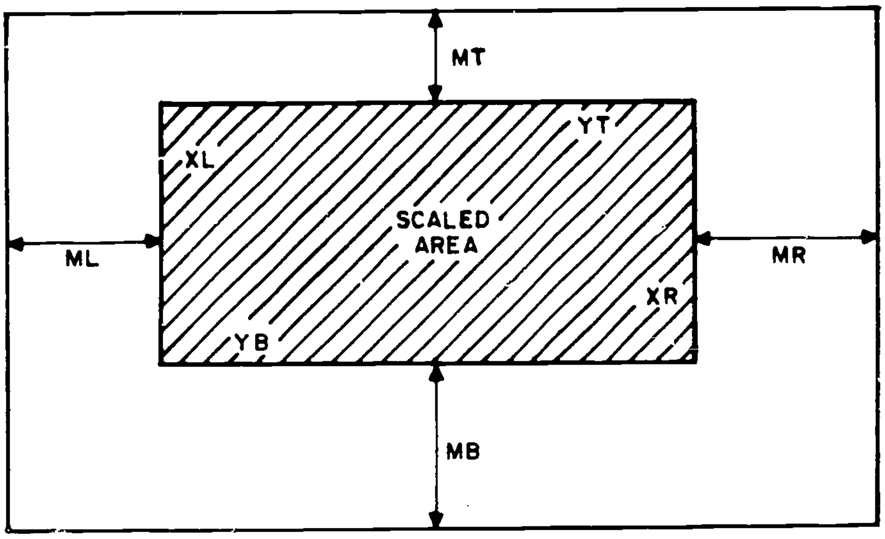


Figure 6-1

XSCALV will assign XL to raster location IX = 170, and XR to raster location IX = 831 (i.e., 1023 - 192). YSCALV will assign YB to raster location IY = 340, and YT to raster location IY = 895 (i.e., 1023 - 128). The scaled area will then be the rectangle from IX = 170 to IX = 831, and from IY = 340 to IY = 895.

BASIC SCALING EQUATIONS

In the following equations, "A" and "B" represent the scale factors computed and stored by XSCALV, and "C" and "D" are the factors computed and stored by YSCALV. (Since the computation is done in floating point arithmetic, the floating point variable names FML, FMR, FMB, and FMT are used to represent the floating point equivalents of the margin values ML, MR, MB, and MT.)

$$A = \frac{(1023.- \mathrm{FMR}) - \mathrm{FML}}{\mathrm{XR} - \mathrm{XL}} \qquad \mathrm{I}$$

$$B = \mathrm{FML} - A*\mathrm{XL} \qquad \mathrm{II}$$

$$C = \frac{(1023. - \mathrm{FMT}) - \mathrm{FMB}}{\mathrm{YT} - \mathrm{YB}} \qquad \mathrm{III}$$

$$D = \mathrm{FMB} - C*\mathrm{YB} \qquad \mathrm{IV}$$

CONVERSION OF DATA: NXV, NYV, IXV, IYV

Four function subprograms, NXV, NYV, IXV, and IYV, are provided to convert data coordinates into raster coordinates. The argument for each of the functions must be a floating point quantity; the result will be an integer quantity.

The following FORTRAN statements show how these functions may be used to convert data coordinates X (or Y) into raster coordinates IX (or IY):

```
IX = NXV(X)

IY = NYV(Y)

IX = IXV(X)

IY = IYV(Y)
```

These four functions are similar in that they all convert data by means of the basic equations for data conversion discussed below. They are dissimilar in the way they handle off-scale data (that is, data which falls outside the limits XL, XR or YB, YT.)

The functions NXV and NYV check for off-scale data values. The result IX (or IY) will be set to zero if the argument X (or Y) is outside the limits that were used to establish the scale. In addition, an error indication is set, as discussed under Off-Scale Error Detection.

The functions IXV and IYV do not test for off-scale data values. The resulting position can be outside the plotting area, or even outside the frame, but the value will be properly scaled relative to the plotting area. (However, no test is made for the possibility that the result is greater than 131,071; integer bits above the 17th will be lost.)

Figure 6-2 illustrates how error testing of the results of NXV, NYV can be used to by-pass plotting of points that are off-scale. NXV and NYV were used to convert the points along the curve into raster positions, and LINEV was employed to connect the points. Since the results of NXV and NYV were tested for zeros, and plotting was by-passed whenever a point was off-scale, the curve stopped at the top and right limits of the scaled area (outlined).

In Figure 6-3 two sine curves are shown, one plotted after using NXV, NYV to do the conversion, and the other after IXV, IYV were used. As in Figure 6-2, LINEV was used to connect the points. NO ERROR TESTS WERE MADE.

The lower curve shows the line going to zero when off-scale values were encountered by NXV, NYV. Note that this curve drops to the bottom of the frame (IY = 0) when values that were off-scale in Y were encountered. Also note that the off-scale initial and last values of X caused the curve to start and end at the left edge of the frame (IX = 0).

The higher curve was drawn after IXV, IYV were used to convert the points. The curve continued past the boundaries of the scaled area when off-scale values were encountered. (When IXV, IYV are used, the programmer must decide what action should be taken when the result of IXV and/or IYV is < 0 or > 1023.)

All four functions test for nonlinear mode. If indicated, X (or Y) will be transformed before it is converted by one of the basic conversion equations.

BASIC CONVERSION EQUATIONS

The following equations show how the conversion functions convert data coordinates X and Y into raster coordinates IX and IY:

$$IX = A*X + B$$

$$IY = C*Y + D$$

The scale factors A, B, C, and D are those derived from the equations I, II, III, and IV.

Generally speaking, the programmer should use the conversion functions rather than writing statements of his own containing these equations. The functions offer the

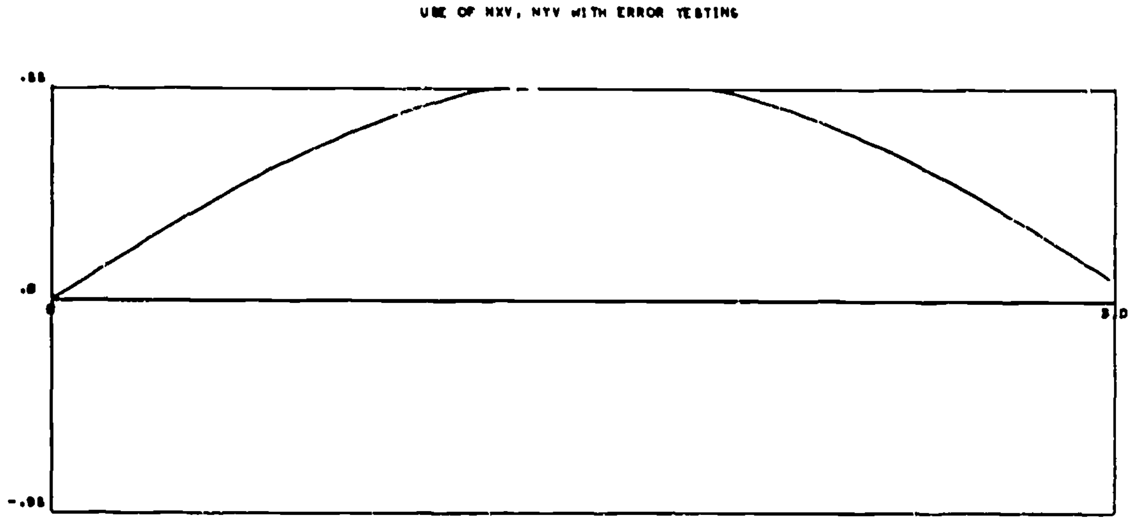


Figure 6-2

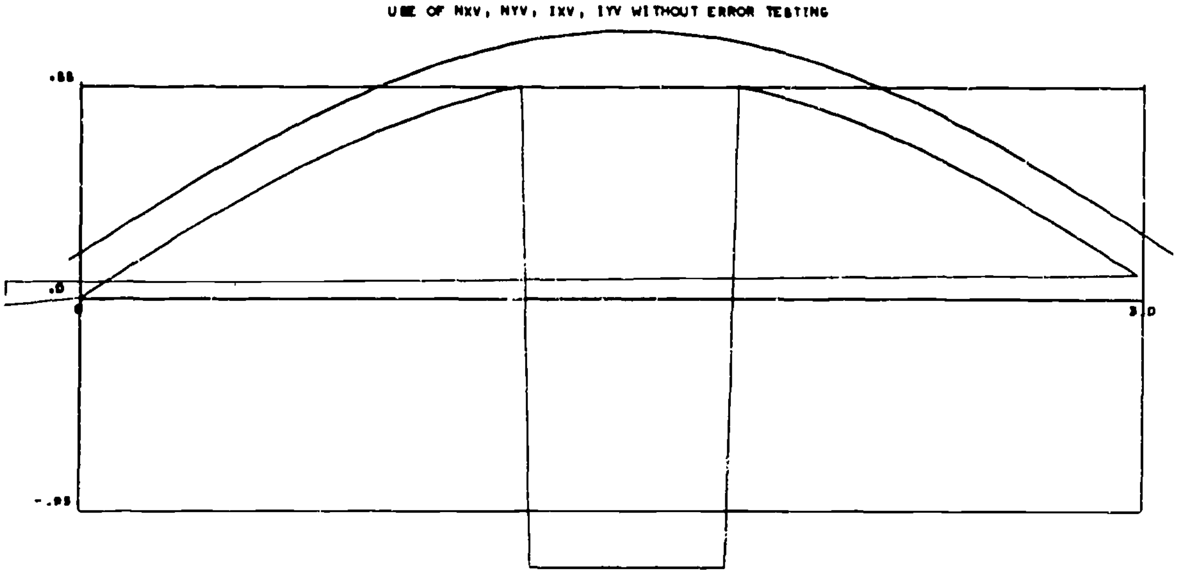


Figure 6-3

following advantages:

a.. They have direct access to the internal table in which the scale factors A, B, C, and D are stored.

b. They check the scale mode, and use a nonlinear conversion if that mode is indicated.

c. NXV, NYV contain a test for off-scale points.

If it is ever necessary to use these equations directly, the programmer can retrieve the scale factors (A, B, C, D) by employing the routine SCLSAV.

INVERSE CONVERSION UXV, UYV

The functions UXV, UYV allow the programmer to obtain the coordinates of a specified raster location in terms of his data. The following statements show how these functions may be used:

```
X = UXV (IX)

Y = UYV (IY)
```

Although UXV, UYV represent the inverse of IX = IXV (X) and IY = IYV (Y), the results are approximate because truncation occurs in the IXV and IYV functions).

NOTE

UXV, UYV CANNOT BE USED
IN THE NONLINEAR MODE.

The equations used by UXV, UYV are the inverse of the equations for data conversion:

$$X = \frac{IX - B}{A}$$

$$Y = \frac{IY - D}{C}$$

(The computation is performed in floating point arithmetic.)

RETRIEVAL OF SCALING INFORMATION: SCLSAV

The subroutine SCLSAV will retrieve scale factors and other scaling information from an internal table and store them in an array named by the programmer. Although SCLSAV makes scaling information available for special-purpose conversions, certain limit tests, etc., its principal value is that it permits saving scaling information from one program link to another. When a new link is entered, another routine RESCLV, can be called to restore the scaling information in the internal table, wherε it is accessible to NXV, NYV, IXV, or IYV.

The calling statement to retrieve the scaling information is:

```
CALL SCLSAV (R)
```

R The name of a ten-cell array, named and dimensioned by the programmer.

The storage locations in the block of cells, R, are assigned as follows:

R(10)	Minimum IY	
R(9)	Minimum IX	
R(8)	Maximum IY	
R(7)	Maximum IX	
R(6)	D	
R(5)	B	Scale factors
R(4)	C	
R(3)	A	
R(2)	Scale mode indicator for Y	
R(1)	Scale mode indicator for X	

RESETTING SCALING INFORMATION: RESCLV

If the scaling information has been stored in COMMON by SCLSAV, it can be reset into the internal table when a new chain link is entered. The statement is:

```
CALL RESCLV(R)
```

where R is the ten-cell array described under SCLSAV.

NONLINEAR SCALING AND CONVERSION

The only nonlinear capability built into the system is logarithmic scale mode, used in connection with log grids. However, the system design allows the programmer to incorporate some special nonlinear scale mode, by the substitution of one module of his own. For this reason, the more inclusive term, "nonlinear," is used in the following discussion, instead of "logarithmic."

LINEAR→NONLINEAR SCALE MODE INDICATORS: SMXYV, MSXYV

The subprogram SMXYV allows the programmer to set scale mode indicators. It must be called if nonlinear scaling is desired. These indicators are tested within other subprograms (GRID1V, IXV, IYV, NXV, NYV, XSCALV, YSCALV). If nonlinear mode is indicated, additional steps will be taken to handle the selected mode. The programmer may also retrieve these indicators, using the subprogram MSXYV.

The statement which sets the scale mode indicators is:

```
CALL SMXYV (MX, MY)
```

$MX \neq 0$, $MY \neq 0$	X nonlinear, Y nonlinear
$MX \neq 0$, $MY \neq 0$	X nonlinear, Y linear
$MX = 0$, $MY \neq 0$	X linear, Y nonlinear
$MX = 0$, $MY = 0$	X linear, Y linear

To reset the indicators for linear-linear, use CALL SMXYV (0,0). If SMXYV is never called, the scale mode will be linear in X and in Y, and the internal scale mode indicators will be set as if CALL SMXYV (0, 0) had been executed.

For chain jobs, note that the values of MX and MY do not carry over from link to link. If a scale mode other than linear-linear is desired, it will be necessary to restate the SMXYV statement in each chain link.

The following statement will retrieve the scale mode indicators for testing:

CALL MSXYV (MXL, MYL)

MXL, MYL Locations in which the quantities furnished as arguments in the last SMXYV statement will be stored.

NONLINEAR TRANSFORMATION: XMODV, YMODV

The scaling and conversion equations shown under number conversion apply not only to the linear mode, but also to the nonlinear mode if transformed arguments are used. The functions XMODV, YMODV are provided to perform such transformation; they can be used in statements of the following type:

XPRIME = XMODV(X)

YPRIME = YMODV(Y).

The scaling and conversion subprograms (XSCALV, YSCALV, IXV, IYV, NXV, NYV) test the scale mode indicators to see if they have been set to nonlinear mode by a call SMXYV. If nonlinear mode is indicated, each of these subprograms will use XMODV (or YMODV) to perform a nonlinear transformation on X (orY) before the remainder of the scaling or conversion takes place. Since the XMODV, YMODV functions in the system compute the log of X or Y, the system nonlinear mode is synonymous with log mode.

For a problem requiring special nonlinear transformation, the programmer can substitute subprograms of his own named XMODV and YMODV, with one argument and one result. Since the system XMODV and YMODV are physically contained in one subprogram, both must be replaced if a substitution is made for either. Obviously, such substitute functions must meet the nonlinear scaling and conversion requirements of the entire program (or chain link), since they will replace the system functions.

System subprograms that use XMODV and YMODV (either directly or indirectly) are: GRID1V, IXV, IYV, LINRV, NONLNV, NXV, NYV, XSCALV, YSCALV, APLOTV, POINTV.

> WARNING: GRID1V WAS DESIGNED ONLY FOR THE LINEAR, LOG, AND SEMI-LOG OPTIONS. SUBSTITUTION OF A SPECIAL TRANSFORMATION FUNCTION MAY CAUSE UNEXPECTED DIFFICULTIES IF USED BY GRID1V.

OFF-SCALE ERROR DETECTION

Whenever there is a possibility that off-scale data points might be encountered, error tests should be made by the programmer. For example, APLOTV sets an error indicator which should be tested. A zero result from NXV or NYV nearly always indicates an error; a test should be made for this condition.

There are additional situations which demand special error detection procedures. For one thing, if no left and/or bottom margin space is reserved, the conversion of XL and/or YB can produce a legitimate zero result from NXV, NYV. More important, the programmer may be using NXV or NYV indirectly, via other modules, and thus be unable to test the results. For these reasons, additional subprograms are provided for detailed analysis of conversion errors resulting from off-scale points.

Keep in mind that the special procedures which follow are designed for unusual situations, in which normal error testing does not suffice.

SET CONVERSION ERROR INDICATORS: SCERRV

Two internal cells are used by NXV and NYV to store indications of successful or unsuccessful data conversion. The subroutine SCERRV allows the programmer to assign two cells which NXV, NYV will use in place of the internal error cells. In this way, the programmer can name error indicator locations that are accessible to his program. The call statement is:

```
CALL SCERRV (KX, KY)
```

When NXV converts a quantity successfully, it will place a "0" in KX; when unsuccessful, it will store a "1" in KX. NYV will use the cell KY in the same way. The cells named will be used in each subsequent execution of NXV, NYV (including execution via other subprograms) until new cells are named by another call to SCERRV (or the internal cells are reset at the beginning of a new link of a chain job). If possible, tests of these error cells should be made as soon after execution of NXV, NYV as possible, to avoid any possibility that they might be altered by subsequent executions of NXV, NYV.

The following practical examples show how SCERRV can provide error indicator cells in connection with POINTV. Since POINTV uses NXV and NYV only once, the named error cells will still contain indications of off-scale errors when control is returned from POINTV to the calling program. (The examples assume that the scale factors have already been established.)

Example 1

```
           .

           .

           .

           CALL SCERRV (KX, KY)

           DO 300 I = 1, N

           CALL POINTV (X, Y, NS)

           IF (KX*KY) 700, 300, 700      Test for non-zero KX and/or KY

       300 CONTINUE

           .

           .

           .

       700 CALL DUMP                     Dump when an off-scale point is
                                         encountered
```

Example 2

```
          .

          .

          .

          CALL SCERRV (KX, KY)

          M1 = 0

          M2 = 0

          DO 300 I = 1, N

          CALL POINTV (X, Y, NS)

          M1 = M1 + KX

    300   M2 = M2 + KY

          .

          .

          .
```

In Example 2, M1 and M2 will contain the total number of points that are off-scale. The contents of these locations may be printed out at the end of the job or after all points have been plotted on each frame.

SAVING AND RESETTING ERROR INDICATOR CELLS: SERSAV, SERREV

As has been pointed out, the locations assigned by SCERRV will be used as error indicator cells until new ones are named (or a new link of a chain job is loaded). As a result, subsequent executions of NXV, NYV, whether on the same level or on a lower or higher one, will alter the contents of the error cells (setting them to 1 for an unsucessful conversion, to 0 for a successful conversion).

Two problems can occur. The obvious one is that indications of error may be masked by a subsequent execution of NXV, NYV. This can usually be avoided if the programmer completes error tests before there is any possibility that the contents of the error cells might be changed.

The less obvious problem is that the error cells named in one level of a program will not be accessible for error testing in other levels unless special action is taken. One way to solve this problem is to carry the error cell names in the call statements of the subprograms, not always a desirable solution.

The modules SERSAV and SERREV, used in conjunction with SCERRV, offer a convenient means for avoiding these problems in multi-level jobs. The call statements are:

> CALL SERSAV (LOCX, LOCY)
>
> This subprogram saves, in LOCX and LOCY, the locations of the cells that are currently being used for off-scale indicators.
>
> CALL SERREV (LOCX, LOCY)
>
> This subroutine resets the off-scale error cell locations that were saved by SERSAV.

The arguments, LOCX and LOCY, must be variable names (either fixed or floating point) which are not used for any other purpose between the execution of SERSAV and of SERREV.

Section VII

PLOTTING

PLOTTING DATA

The 64 PDP-10 characters are shown in Figure 7-1 along with the selection code for each.

Reduced to fundamentals, printing or plotting of one PDP-10 character involves the selection and display of that character at a specified position on the raster. (A basic subprogram, PLOTV, can be used to select and display one character at a time.) The higher-level subprograms, however, contain features that make each one suitable for a specialized purpose: plotting, printing, or labeling. These specialized features of the higher-level routines make them appear to be distinctly different from each other.

POINT PLOTTING SUBPROGRAMS

Since point plotting usually involves the scaled representation of a physical phenomonon, most point plotting subprograms accept physical data coordinates for position information. During execution of these subprograms, the data coordinates are converted into raster coordinates. Scale factors must have been established for the plotting routines to use in making these conversions; this requirement can be satisfied by a prior entry to GRID1V.

Scale factors are not normally saved from link to link in a chain job. Consequently, plotting should be done within the link in which scale factors are computed.

PLOTTING AN ARRAY: APLOTV

APLOTV was designed for situations in which a large number of X values are stored in one array and and the corresponding Y values are stored in another array. It is possible to plot the entire set of data with one entry to APLOTV. If desired, only a portion of the data can be plotted.

PDP-10 STANDARD CHARACTERS

1. Space
2. !
3. "
4. #
5. $
6. %
7. &
8. '
9. (
10.)
11. *
12. +
13. ,
14. -
15. .
16. /
17. 0
18. 1
19. 2
20. 3
21. 4
22. 5
23. 6
24. 7
25. 8
26. 9
27. :
28. ;
29. <
30. =
31. >
32. ?
33. @
34. A
35. B
36. C
37. D
38. E
39. F
40. G
41. H
42. I
43. J
44. K
45. L
46. M
47. N
48. O
49. P
50. Q
51. R
52. S
53. T
54. U
55. V
56. W
57. X
58. Y
59. Z
60. [
61.
62.]
63. ↑
64. ←

SUGGESTED PLOTTING CHARACTERS

1. .
2. O
3. 0
4. +
5. X
6. *
7. C
8. H
9. U

Figure 7-1

It is also possible to use diffcrent PDP-10 characters as symbols to identify curves. The programmer furnishes a table (of one or more PDP-10 characters) that will be selected cyclically.

APLOTV also keeps a tally of the number of off-scale points encountered.

The calling statement is:

CALL APLOTV (±N, XARRAY, YARRAY, JX, JY, ±NC, MARKPT, IERR)

N — Controls the number of points to be plotted. N is actually the number of points if all the data points in the arrays are to be plotted in succession. The value of N may be computed by letting K in the following formulas equal the number of points to be plotted:

N = K*JX or N = K*JY, whichever is larger.

Usually, N should be positive, to indicate that the data arrays are in normal FORTRAN order of storage. However, if the arrays are in increasing order of storage, N should be negative.

XARRAY
YARRAY — Normally, the names of the arrays of floating point data to be plotted. Since these arguments must name the locations of the data coordinates of the first point to be plotted, subscripts may be necessary. Both arrays must be in the same order of storage.

JX, JY — Fixed point positive integers giving the increments to be added to the subscripts of XARRAY and YARRAY as each point is plotted.

NC — The number of characters in the array MRKPT to be used as plotting symbols (usually the size of the array MRKPT). Positive NC indicates that MRKPT is a normal FORTRAN array. Negative NC shows that MRKPT is stored in increasing locations in core.

MRKPT — The array that contains the plotting character(s) to be used. These will be used cyclically, with the first one being used again after the NCth one has been used.

To set up this information when only one character is to be used, the argument can be one of the following:

a. A Hollerith argument for one FORTRAN character, as "1HO."

b. The name of a location containing an "0", as "MRKPT."

a. An array read in by the A format, one character to a location. Such an array is not restricted to integer-type names; it could be called PTMRK, for example, if the contents are not used in a computation.

IERR The name of an error location supplied by the programmer. Any point that falls outside the grid drawn by GRID1V will not be plotted. Instead, a count of such points will be stored in IERR.

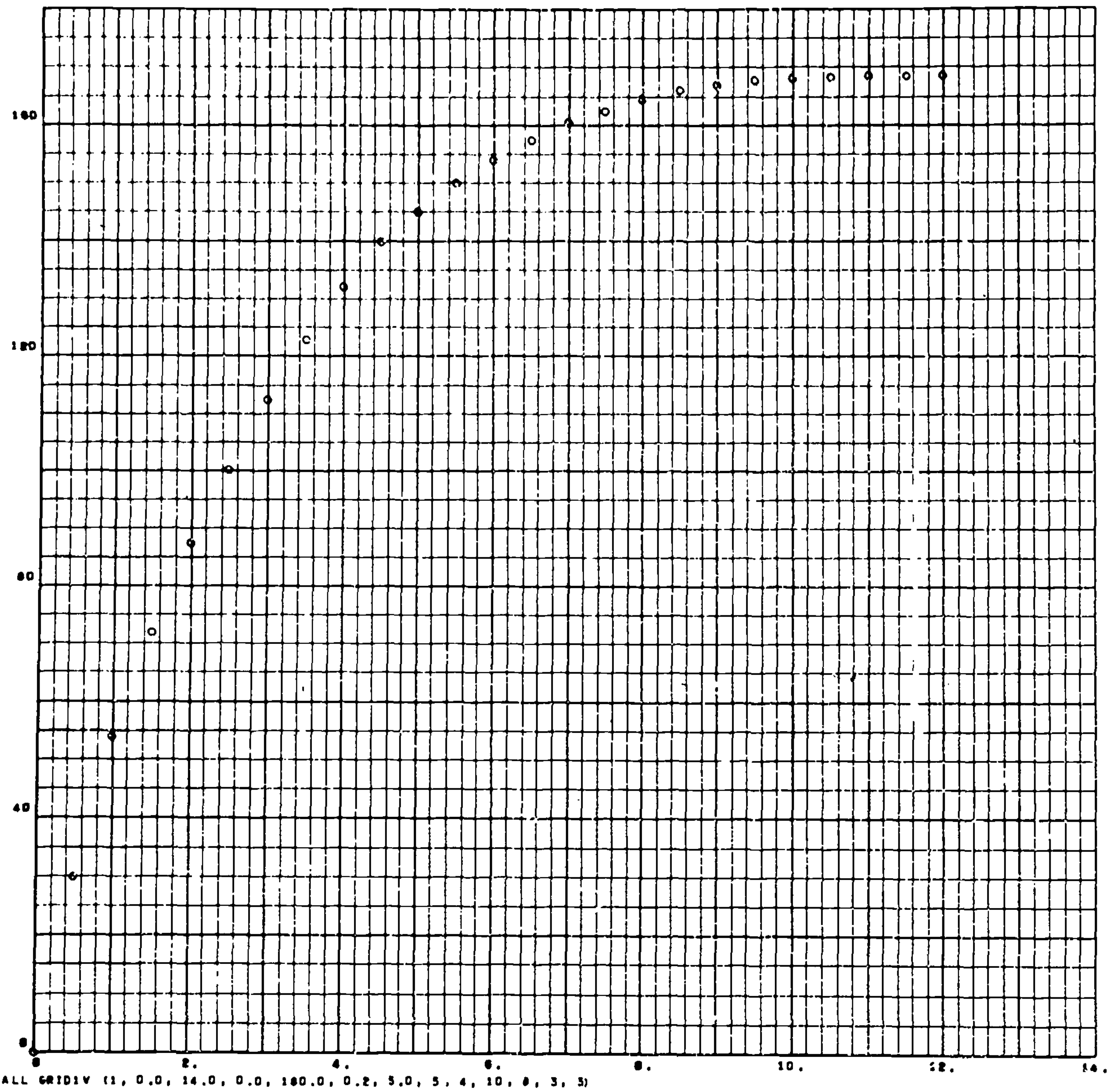
APLOTV plots only one character per point. If a center dot is desired, APLOTV may be repeated, using the plotting dot as the symbol. The use of APLOTV is illustrated by the Figures 7-2 and 7-3.

In Figure 7-2, it is assumed that the data are stored in two arrays: X(1), X(2), ..., X(25), and Y(1), Y(2), ..., Y(25). A prior entry was made to GRID1V, using the statement:

```
CALL GRID1V (1, 0.0, 14.0, 0.0, 180.0, 0.2, 5.0, 5, 4, 10, 8, 3, 3)
```

Note that the X and Y limits cover the range of the data to be plotted. Also, since GRID1V was entered first, the necessary scale factors have been established for APLOTV. The calling statement of APLOTV for this example is the following:

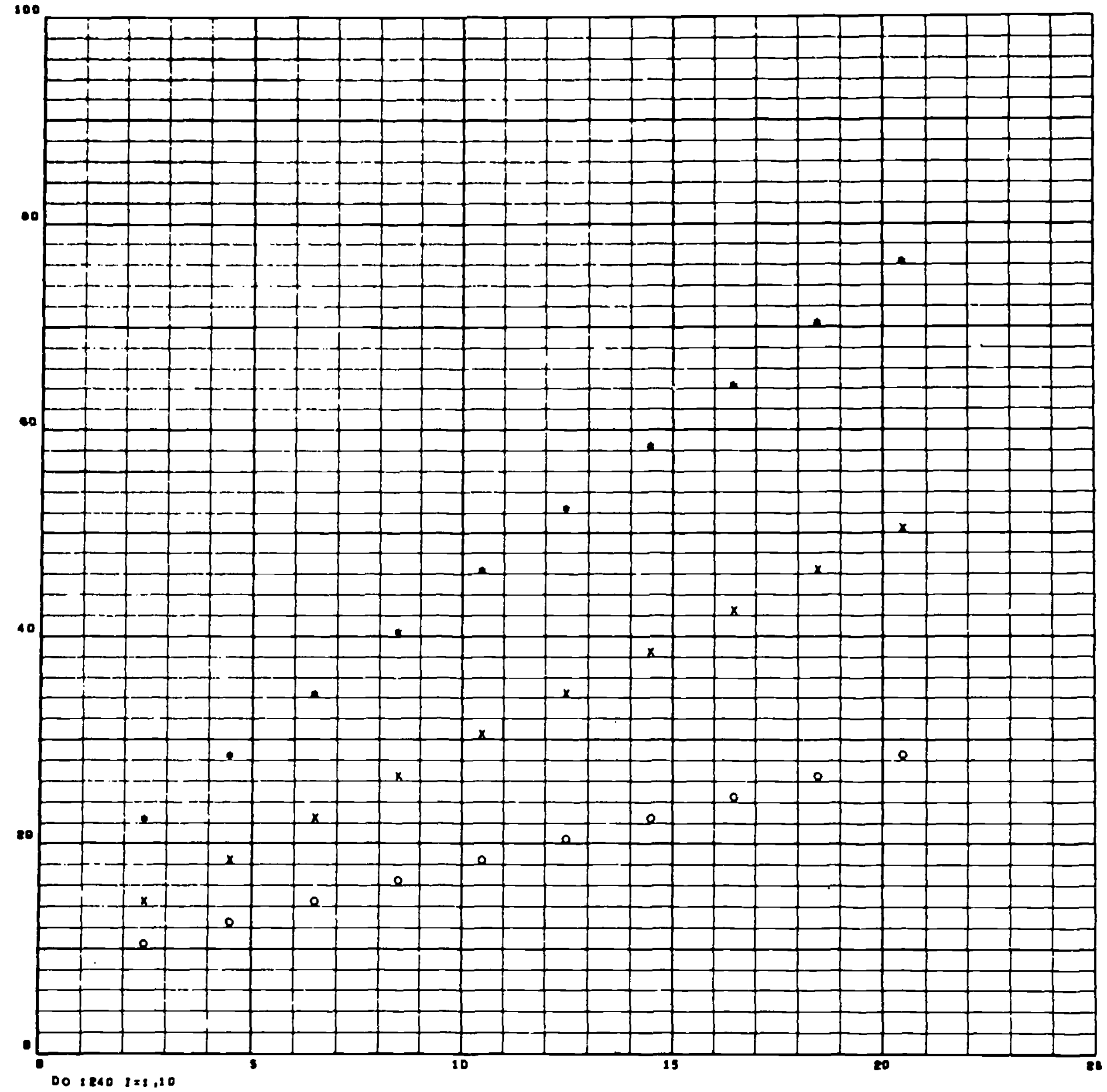
```
DATA IS STORED X1,X2,...X25 AND Y1,Y2,...Y25
```



```
CALL GRIDIV (1, 0.0, 14.0, 0.0, 160.0, 0.2, 5.0, 5, 4, 10, 8, 3, 3)
CALL APLOTY (25, X, Y, 1, 1, 1, 38, IERR )   OR   CALL APLOTV (25, X, Y, 1, 1, 1, 1HO, IERR )
```

Figure 7-2

```
    XA ARRAY IS STORED X1,X2,...X10    YA ARRAY IS STORED A1,A2,...A10 B1,B2,...B10 C1,C2,...C10
 '  PTMRK CONTAINS O,X,AND * LOADED UNDER CONTROL OF FORMAT (3A1)
```



```
    DO 1240 I=1,10
1240 CALL APLOTV ( 30, XA(I), YA(I), 0, 10, 3, PTMRK, IERR)
```

Figure 7-3

```
   CALL APLOTV (25, X, Y, 1, 1, 1, XT, IERR)
or CALL APLOTV (25, X, Y, 1, 1, 1, 1HO, IERR)
```

N = 25 The array size, 25 in this case, is equivalent to the number of points to be plotted. This can only be true when all points in each array are to be plotted.

X, Y Names of the X and Y arrays (properly dimensioned).

JX = 1
JY = 1 Since every X versus Y is to be plotted, the fourth and fifth arguments are set to 1.

NC = 1 Only one plotting symbol is to be used; hence the sixth argument is set to 1.

MARKPT In the two calling sequences shown, one specifies the plotting symbol by a symbol ; the other uses a Hollerith argument.

Figure 7-3 shows one way that APLOTV might be used to plot a family of curves. The example assumes that the XA and YA arrays are properly dimensioned. The data is stored XA(1), XA(2), . . . , XA(10); corresponding Y's for the three curves are stored in the array YA in the order A(1), A(2), . . . , A(10), B(1), B(2), . . . , B(10), C(1), C(2), . . . , C(10). PTMRK is the name of a three-word array which was loaded as the BCD equivalents (read in by the A format) of the characters O, X, and *.

In the example, A(I), B(I), and C(I) were plotted versus X(I) each time APLOTV was entered. A DO loop was used to proceed to the next value of X, so that a total of 10 entries were made to APLOTV. The coding was:

```
DO 1240 I = 1,10

1240 CALL APLOTV (30, XA(I), YA(I), 0, 10, 3, PTMRK, IERR)
```

Note that the plotting symbols are used cyclically, returning to the first one when the array PTMRK is exhausted. If desired, a center plotting dot can be superimposed upon the plotting symbol by repeating the entries to APLOTV with the plotting dot used for PTMRK.

PLOTTING INDIVIDUAL DATA POINTS: POINTV

For each entry to POINTV, one symbol is plotted (with or without a center dot). The coordinates may be specified as floating point data, which POINTV will convert into raster coordinates. Scale factors must have been established; this can be accomplished by a prior entry to GRID1V.

The calling statement is:

```
CALL POINTV(X,Y,±NS,ANY)
```

WHERE X,Y are coordinates of the point to be plotted, stated as floating point data values.

±NS If NS is minus one, there will be no center dot plotted. If NS is positive one, a center dot will appear.

ANY any character to be displayed at the point pointed at by POINTV.

Points outside the scaled area will not be plotted. Until the programmer learns how to detect off-scale points, he should be sure that the data coordinates will fall within the limit of the scaled area.

An <u>alternate</u> version of this subprogram allows the programmer to specify position information in raster coordinates. This is particularly useful when the programmer wants to construct a legend in the margin, showing the symbols used and their meaning. The alternate call statement is:

```
CALL POINTV(IX,IY,±NS,ANY)
```

Where IX, IY are raster coordinates of the point to be plotted; fixed point.

±NS If NS is a minus two, there will be no center dot plotted and the routine will accept IX and IY as raster coordinates. If NS is a positive two a center dot will appear and IX and IY will be accepted as raster coordinates.

ANY Any character to be displayed at the point pointed at by POINTV.

BASIC PLOT-PRINT SUBPROGRAM: PLOTV

Any of 64 PDP-10 characters can be displayed at a specified raster position by using PLOTV, the basic subprogram used as a lower-level module of other routines.

However, the programmer may find it useful when other plotting or printing subprograms are not suitable. The call statement is:

```
CALL PLOTV(IX,IY,ANY)
```

Where IX,IY are fixed point raster coordinates at which the character will be displayed.

ANY is any desired character. (See Figure 7-1).

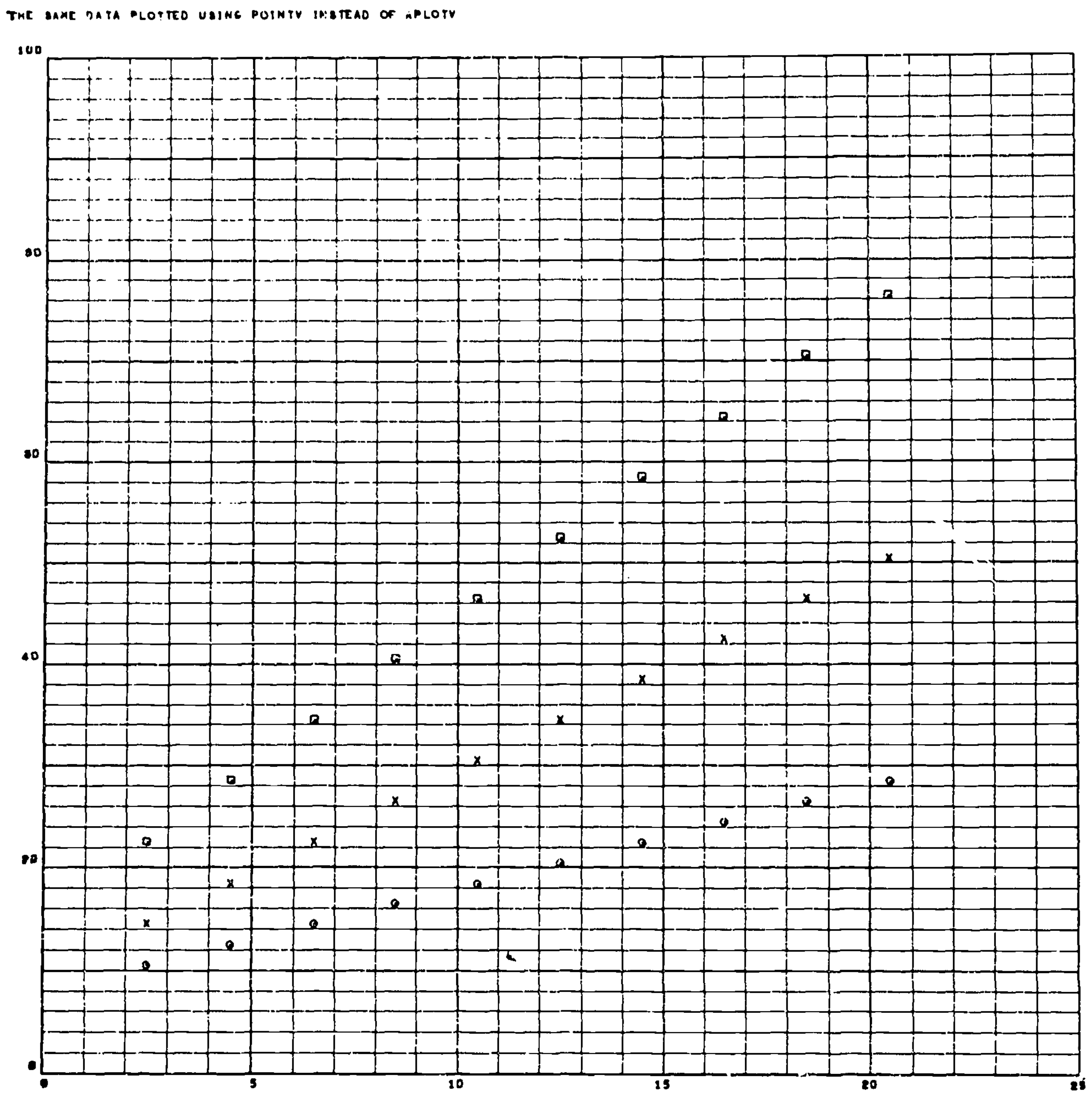

THE SAME DATA PLOTTED USING POINTV INSTEAD OF APLOTV
100
80
60
40
20
0
0
5
10
15
20
25

Figure 7-5

LINE GENERATION: LINEV

LINEV connects two points by a straight line composed of vectors, joined end-to-end. The arguments for LINEV, which specify the points to be connected, must be given in raster counts. As described above, the programmer may connect two data points by a line if he first uses the functions NXV and NYV to convert the data coordinates into raster coordinates. (If there is a possibility that the data points being converted may be off-scale, the conversion results should be tested for errors before LINEV is executed.) The calling statement is:

```
CALL LINEV (IX1, IY1, IX2, IY2)
```

IX1, IY1 Raster coordinates of one end point.

IX2, IY2 Raster coordinates of the other end point.

Figure 7-6 contains an illustration of the use of LINEV.

LINE2V is used to draw a line from a fixed point in some direction specified by DX and DY. The calling sequence is:

```
CALL LINE2V (IX1, IY1, IDX, IDY)
```

IX1, IY1 Raster coordinates of starting point.

IDX, IDY Number of raster points that the line is to be extended in the X and Y directions. DX and DY will be handled modulo 64.

In either LINEV or LINE2V, a floating point data value may be utilized,if scaling has been established,by utilizing the function subprograms NXV, NYV as follows:

```
CALL LINEV (NXV(X1), NYV(Y1), NXV(X2), NYV(Y2))
```

Figure 7-7 is an example of the use of LINE2V. Each line in the figure is produced by incrementing IDX or IDY.

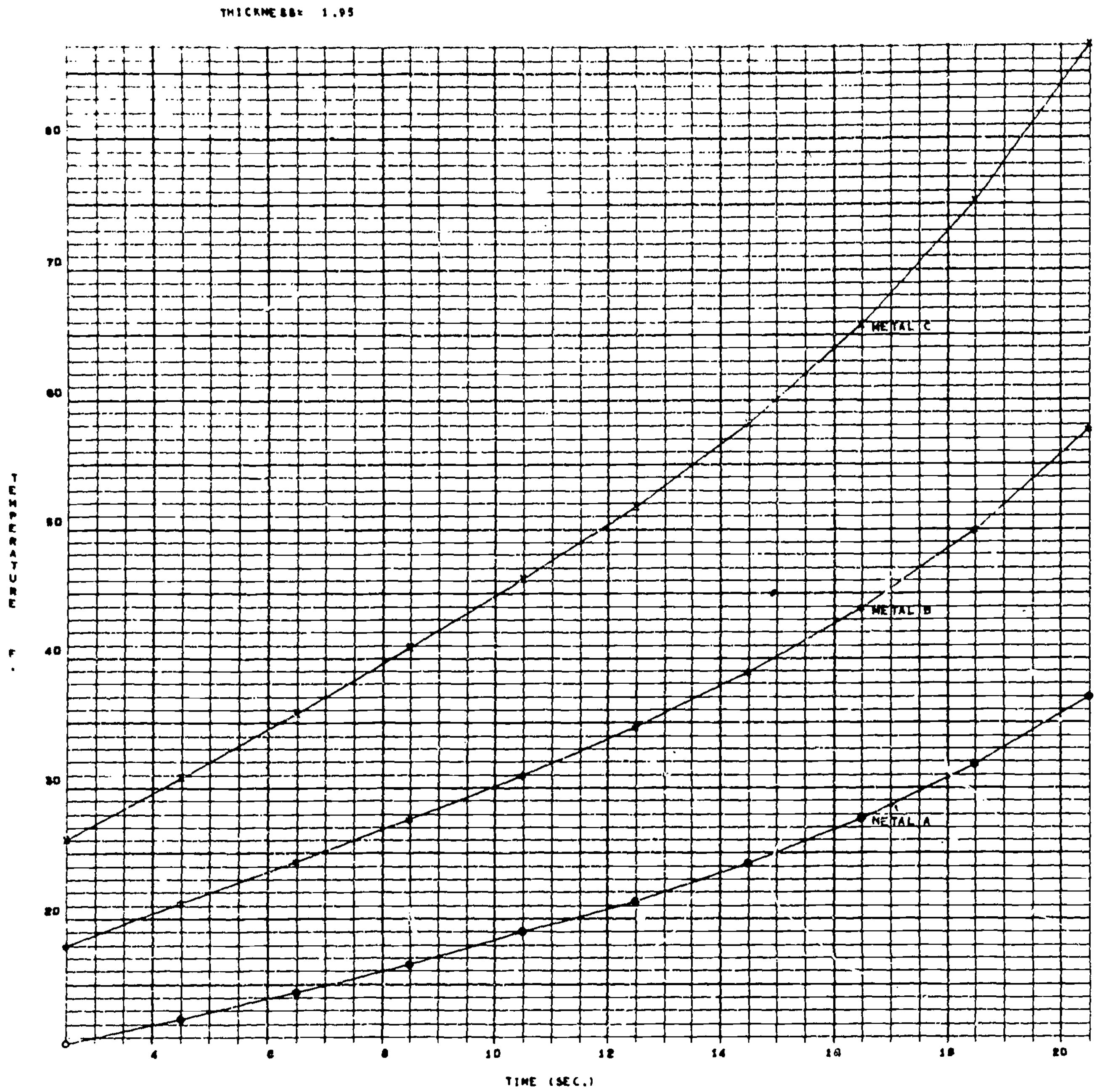

THICKNESS= 1.95
80
70
60
50
40
30
20
TEMPERATURE F.
4
6
8
10
12
14
16
18
20
TIME (SEC.)
METAL C
METAL B
METAL A

Figure 7-6

```
      CALL XSCALV (-511.0, 512.0, 0, 0)
      CALL YSCALV (-511.0, 512.0, 0, 0)
      Z0 = 0.0
      Z1 = 4.0
      Z2 = 4.0
      CALL POINTV (Z0, Z0, -16)
      DO 5 I = 1, 63
      INC = I
      JNC = -INC
      CALL LINE2V (NXV (Z1), NYV (Z0), 0, INC)
      CALL LINE2V (NXV (Z2), NYV (Z0), 0, JNC)
      CALL LINE2Y (NXV (Z0), NYV (Z1), JNC, 0)
      CALL LINE2V (NXV (Z0), NYV (Z2), INC, 0)
      Z1 = Z1 + 3.0
5     Z2 = Z2 - 3.0
```

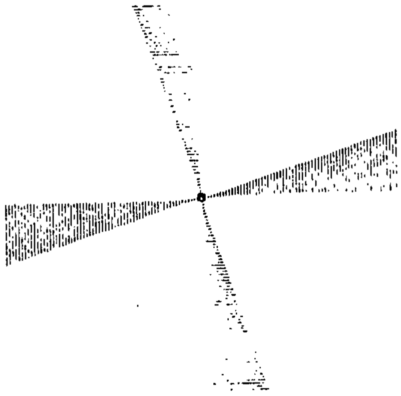


Figure 7-7

Section VII

TITLING AND LABELING

TITLING AND LABELING SUBPROGRAMS

The printing and labeling subprograms enable the programmer to affix titles and other identifying information to a picture. Three subprograms of this type will be introduced: PRINTV, APRNTV, and LABLV. Other means for printing and labeling will be given in the section on printing.

For many applications, the positions of titles and labels must be independent of the scale. Therefore, printing and labeling subprograms accept position information in raster coordinates. This contrasts with the plotting routines, which includes facilities for the conversion of data into raster counts. Titles or labels may be positioned relative to data. The conversion functions discussed later can be employed to find raster coordinates from floating point location data.

HORIZONTAL TITLES: PRINTV

This subprogram allows the programmer to display horizontal titles composed of characters. The call statement provides for printing characters read in by A- type format. The call statement is:

```
CALL  PRINTV(N,ASCTEX,IX,IY)
```

N The number of characters to be printed.

ASCTEX An array containing the (A- type format) text to be printed.

IX, IY   The raster coordinates for the center of the first character.

VERTICAL TITLES: APRINTV

This subprogram can be used to display vertical titles composed of PDP-10 characters. Each individual character will be upright. The call statement provides for printing characters read in by the A format or characters stored as a Hollerith argument. Since APRINTV prints text or character, the spacing of the characters is controlled by arguments specified by the programmer.

The call statement for APRINTV is:

```
CALL APRINTV (INCRX, INCRY, N, ASCTXT, IX, IY)
```

INCRX
INCRY   Increments used to space the characters in the X or Y, direction, given in raster counts. For vertical titles, INCRX will be zero and INCRY should have a negative value. (It is suggested that INCRY fall in the range between -12 and -18 for vertical titles in most applications).

The remaining arguments are as specified under PRINTV.

FIXED POINT LABELS: LABLV

LABLV was developed for GRID1V to employ in labeling grid lines, but will prove helpful when the value of computed quantities must be printed. The routine performs one chief task: it displays a floating point number at the raster coordinates specified.

The calling statement of LABLV is:

```
CALL LABLV (D, IX, IY, NCHAR, NT, NDMAX)
```

D The floating point quantity to be printed.

IX, IY The raster coordinates which will position the first character of the label. Note that this first character may be a leading blank. If the quantity to be displayed is negative, the minus sign will be displayed one character space to the left of IX, IY.

NCHAR Number of characters to be displayed, including leading blanks and the decimal point, if any. NCHAR is limited to 6 (or 7 if one of the characters is the decimal point).

NT The number of times each character is to be displayed (number of over-strikes). Normally this should be 1, but 2 or more may be chosen if a darker label is desired.

NDMAX Maximum decimal scale; i.e., maximum number of characters to be displayed to the left of the decimal point.

An integer quantity may be displayed by first changing it to floating point form and then using LA3LV.

An alternate form of LABLV may be used to display labels in scientific notation. The call statement shown is used with the following changes.

NCHAR Number of significant figures to be displayed. NCHAR may be less than or equal to 6. The negative sign will result in the use of scientific notation.

NDMAX May be any fixed point quantity. Since the right adjustment of these labels will not be necessary, the value of NDMAX will be ignored.

NCHAR will affect the format in the following ways:

NCHAR	FORMAT
1	Y1x10±YY
2	Y.Yx10±YY
3	Y.YYx10±YY
4	Y.YYYx10±YY
5	Y.YYYYx10±YY
6	Y.YYYYYx10±YY

Since the space required for these labels will be greater than that required for the fixed point format, the programmer should allow NCHAR + 7 character spaces in width and 1-1/2 spaces in height as a minimum; it may be necessary to allow even more to avoid overlapping other images.

SECTION IX

PRINTING

OLMGS 340 PRINTING: TX340

The purpose of this routine is to provide the programmer with a quick method of printing on the CRT.

The calling sequence is:

```
CALL TX340  (A,NCHAR,NLINE,NCOL,IER)
```

where A = the array of character to be printed

NCHAR = Number of characters.

NLINE = Number of lines.

NCOL = Number of columns

IER = 1 normal

2 Error

SECTION X

MISCELLANEOUS ROUTINES

These routines are classified because they are never used directly by the programmer; however, many may be used in other programs.

UNPACKING CHARACTERS: UNPACK

This subprogram is used as a low-level subprogram to unpack characters. Characters in one vector are unpacked into a second vector one character per word (A-type format compatible). The call statement for UNPACK is:

```
CALL UNPACK(VEC,UVEC)
```

where:

VEC is a vector of characters (any length).

UVEC is a vector of length five, which will contain the unpacked characters.

SETTING SYSTEM FLAGS: FLAGS

System error indicators may be set by the programmer while scaling data, by calling FLAGS. Note it is not necessary for the programmer to scale any data in this system; however this facility is available if the programmer uses this option. The calling statement is:

```
CALL FLAGS(I,N)
```

where:

I is the value of the indicator.

N=0 set X indicator to I.

≠0 set Y indicator to I.

DRAWING VECTORS: PLTW5

This subprogram is used to draw a vector from a fixed point in some direction specified by IDELX and IDELY. The calling sequence is:

```
CALL PLTW5(IX1,IY1,IDELX,IDELY)
```

IX1,IY1 raster coordinates of starting point.

IDELX,IDELY number of raster points that the line is to be extended in the X and Y directions.

DISPLAY PREMITIVES

These routines are described in reference [3].

```
        PGEN(IX,IY,INTENSITY)
        VGEN(IX,IY,INTENSITY)
        TEXTP
        PARMS(ISCALE,INTENSITY)
```

INDEX OF SUBROUTINES

References

1. DEC - 10- LOVA-D, CHAIN, Digital Equipment Corporation, Maynard Massachusetts, Feburary 8, 1969.

2. Stromberg- Carlson, Programmers' Reference Manual, Data Products, San Diego, Calif., October 1964.

3. Lewis, Harry, Fortran-Lisp Display Routines,NIH, Division of Computer Research and Technology, Bethesda, Maryland, April 1969.

4. Hill, Edward, A Proposed On-Line System For Modeling Networks, N. I. H. Division of Computer Research and Technology, Bethesda, Maryland, Feburary 1969.

5. Lewis, Harry, Assembly Language For DEC 340 Display,NIH, Division of Computer Research and Technology, Bethesda, Maryland, April 1969.

6. Adler, C., Sanford, 340 Display Programming Manual, Decus No. 7-13, New York University's Department of Industrial Engineering and Operations Research, Bronx, New York,

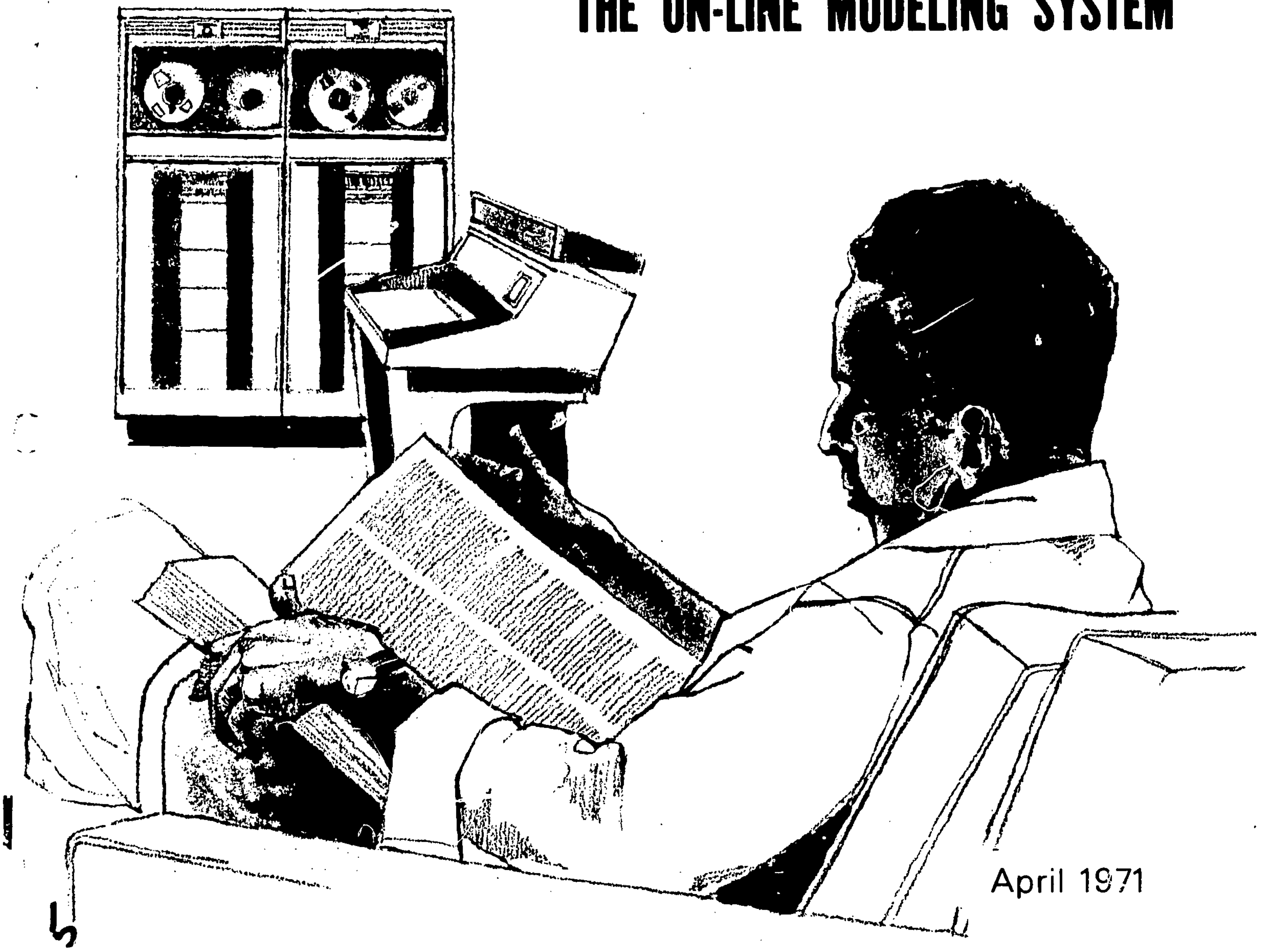
7. DEC - 10 - MTEO - D, Time-Sharing Monitors; Multi-programming Monitor (10/50) Swapping Monitor (10/50), Digital Equipment Corporation, Maynard, Massachusetts, November 1968.

**Portions of this manual were printed with permission of the Stromberg DatagraphiX, Inc.

TECHNICAL REPORT NO. 6

PART III

THE ON-LINE MODELING SYSTEM



April 1971

U.S. DEPARTMENT OF HEALTH, EDUCATION, AND WELFARE
Public Health Service
National Institutes of Health

The Division of Computer Research and Technology, NIH, will issue on an irregular basis technical documents which we believe will be of particular interest to the biomedical community. These reports will include detailed descriptions of relevant computer programs and instructions in their use (as well as some theoretical background), in hopes that interested scientists will be encouraged to gain first-hand experience in applying them. In some cases, such reports may serve as foci around which DCRT will structure training courses to expand the knowledge and experience of NIH staff in applying computer science to problems of research and management. Circulation of these reports within the biomedical community broadly is, of course, encouraged.

A. W. Pratt, M.D.

A. W. Pratt, M.D., Director, DCRT

THE ON LINE MODELING SYSTEM

Part III. Programmers' Reference Manual

by

Edward Hill, Jr.

Laboratory of Applied Studies

Division of Computer Research and Technology
National Institutes of Health, Public Health Service
Department of Health Education and Welfare, Bethesda, Maryland 20014

Table of Contents

SECTION PAGE

IV Input-Output

SECTION I

INTRODUCTION

The basic philosophy of the OLMS is the minimization of storage used during the time when display is INITED, while at the same time keeping the system flexibility under the control of the user.

This part is divided into four sections. The contents of the sections are listed below:

SECTION II : A discussion of the routines used by the interpreter for the command language.

SECTION III: The routines used to generalize the OLMGS under INTER are discussed. Routines which input data and graphical functions using the RAND TABLET and Function Key routines are discussed.

SECTION IV : The system I/O is discussed in this section. Reasons for writing I/O are:

(1) To detect as many errors as possible before an EXIT to the monitor.

(2) A magnetic tape directory system was needed.

(3) The magnetic tape directory system which existed, called FILER, had no user level error returns that could be handled before an EXIT to the monitor.

DIRECTORIES

The OLMS has two directories called PAGET and SLIBD. PAGET and XLIBD are the system file and library directories respectively. In the present configuration PAGET and XLIBD are set up for fifty files in each directory. Each file has an associated file description of fifty characters. Each directory is a total of 700 words. An example of the directories is given in figure 1.

NAME	EXT	PROP	DESCRIPTION 50 CHARACTERS

PAGET

NAME	EXT	PROP	DESCRIPTION 50 CHARACTERS

XLIBD

FIGURE 1

SYSTEM GENERATION

A subsystem of this system can be generated with a few minor changes in some of the routines associated with the interpreter. For example, a user could get along with twenty or less files and no file description. This saves 1360 words of storage. Clearly, this is a compromise between generality, storage and processing time. Many subsystems can be developed to fit the individual user's needs.

SECTION II

INTERPRETER

INTERPRETING COMMANDS: INTER

This routine interprets the command language. The string to be interpreted must be in a common block called STRBUF.

The calling sequence is:

```
CALL INTER
```

No arguments are present since this routine is defined on a common block.

STORAGE ALLOCATION: GETCOR

Storage is allocated by the interpreter by calling GETCOR.

The calling statement is:

```
CALL GETCOR(N,IER)
```

N = The amount of storage to allocate.

IER = 1 The amount of storage asked for is not available.

= 0 The storage is allocated.

Storage is allocated in 1024-word blocks. Any details about how storage is allocated can be found in reference (1).

TASK ALLOCATION: INTERP

After each task is complete in the OLMS this routine should be called to return control to the OLMS. This call should be made at the point where the user wishes to exit or return from his routine.

The calling statement is:

CALL INTERP(JCHDEV)

JCHDEV is the name for the device where the chains are stored. All modeling systems running under the OLMS must make this call in the, instruction sequence. JCHDEV is a variable in a common block called SYSDDD.

LIST STRING INTERPRETER: LISTT

The list commands are interpreted by a subinterpreter of INTER. LISTT is used to give flexibility. A part of the interpreter may be a chain file. This minimizes storage in case a user wants a system with a resident interpreter.

The calling statement is:

CALL LISTT

This routine assumes that the string is in a common block called STRBUF.

FILE NAME DELETIONS FROM SYSTEM DIRECTORIES: DELEFL

File names are deleted from the system directories using DELEFL. The directory is determined by the state of the ITAMAS switch located in the common block called COMSWT. The file name deleted is the name equal to XNAME, the extension equal to EXT and the property equal to PROP located in the common block called COMFIL. LOOK must be called before DELEFL to set the IFOUND pointer.

The calling statement is:

CALL DELEFL

ITAMAS = 0 Before call delete from PAGET.

ITAMAS = 1 Before call delete from XLIBD.

BLANK VECTOR: BLANKS

This routine blanks a buffer of five words. This buffer is used to send characters to a name syntax checker. The buffer or vector is called XPNAM and is located in a common block called PNAME.

The calling sequence is:

```
CALL BLANKS
```

LISTING MODE: MMODE

The OLMS has eight modes. This routine is defined on MMMODE. MMMODE is the system mode indicator. The mode indicator is located in the common block called MODTAB.

The calling statement is:

```
CALL MMODE.
```

DEVICE CHARACTERISTICS: CHARD

This routine determines the physical characteristics associated with a logical device. The DEVCHR UUO is used in this routine.

The calling sequence is:

```
CALL CHARD(DEVIC,IACL,IACR)
```

DEVIC A sixbit device name.

IACL The content of the left-half of the accumulator.

IACR The content of the right-half of the accumulator.

Details about this routine can be found in reference (1).

FIND ADDRESS: GETADR

This routine returns the address of the variable in the first argument.

The calling sequence is:

CALL GETADR(VAR,IRVAR)

VAR Find the address of this variable.

IRVAR The address of the variable.

BLOCK TRANSFER DATA: TRANSF

This routine will block transfer data from the place pointed at by the first argument to the place pointed at by the second argument.

The calling sequence is:

CALL TRANSF(IF,IT,N)

IF A link variable pointing to the place where the data will be moved from in the transfer.

IT A link variable pointing to the place where the data will be moved to in the transfer.

N One less than the number of words to be moved.

The pointers can be found by using GETADR. Data·may be transfered in the opposite direction by exchanging the link variables.

CONVERT TO SIXBIT: CONVER

This routine converts ASCII characters to SIXBIT characters.

The calling sequence is:

CALL CONVER(XNAME)

XNAME Contains the ASCII characters when the routine is called and SIXBIT when the routine returns.

LOOK FOR NAME: LOOK

This routine looks at the content of the system directories to determine if a file is present. The directory is determined by the state of the ITAMAS switch located in the common block called COMSWT. The file name looked for is the name equal to XNAME, the extension equal to EXT and the property equal to PROP located in the common block called COMFIL. After the search is made for the file the switch LOCATE in COMSWT is set to one if the file name is found; otherwise, LOCATE is zero. If the file is found a pointer called IFOUND is set to point to the location in the directory where the file name is located. IFOUND is a pointer located in the common block called COMPNT.

The calling sequence is:

CALL LOOK

ITAMAS=0 Before call will look at PAGET.

ITAMAS=1 Before call will look at XLIBD.

LOCATE=0 After the return indicates the absence of the file name.

LOCATE=1 After the return indicates the presence of the file name.

IFOUND Pointer to the file name location.

PLANTING FILE NAMES: PLANT

This routine is used to plant file names in the system directories. The directory is determined by the state of the ITAMAS switch located in the common block called COMSWT. The file name planted is the name XNAME, the extension EXT and the property PROP located in the common block called COMFIL. This routine calls LOOK and TOP. If an error occurs an error message will be written before the return.

The calling statement is:

CALL PLANT

ITAMAS=0 Before call plant in PAGET.

ITAMAS=1 Before call plant in XLIBD.

PLANTING FILE DESCRIPTIONS: PDESCR

This routine is used to plant file descriptions in the system directories. The directory is determined by the state of the ITAMAS switch located in the common block called COMSWT. A call to LOOK must be made before this routine is called to set the IFOUND pointer.

The calling sequence is:

CALL PDESCR

ITAMAS=0 Before call plant in PAGET.

ITAMAS=1 Before call plant in XLIBD.

Any description to be planted must be in the DESVEC vector before the call to the routine. DESVEC is a vector in the common block called COMDIR. A description of fifty characters may be planted.

FINDING SPACE: TOP

This routine is used to find space in the system directories. The directory is determined by the state of the ITAMAS switch located in the common block called COMSWT. A check is made to see if the upper limits of the directories are exceeded. The upper limits of the directories are IPAEND and ILDEND located in the common block called COMPNT. If the upper limit is not exceeded then ISPACE is set to point to the first empty location.

The calling statement is:

```
CALL TOP
```

ITAMAS=0 Before call find space in PAGET.

ITAMAS=1 Before call find space in XLBD.

PLANTING NAMES: PCHNAM

This routine is similar to PLANT. Only the file name is planted.

The calling sequence is:

```
CALL PCHNAM
```

PLANTING EXTENSIONS: PCHEXT

This routine is similar to PLANT. Only the file extension is planted.

The calling sequence is:

```
CALL PCHEXT
```

PLANTING PROPERTIES: PCHPRO

This routine is similar to PLANT. Only the file property is planted.

The calling sequence is:

```
CALL PCHPRO
```

CHECK FILE NAME SYNTAX: CHECKN

This routine checks the syntax of a five or less than five character name. The name must be in A1 type format. A switch called ITWITC is set to indicate an error or correct name. LOCATT is set to the number of characters in the name. ITWITC and LOCATT are located in a common block called COMSWT.

The calling sequence is:

```
CALL CHECKN
```

ITWITC=0	After return name error.
ITWITC≠0	Correct name.
LOCATT	The number of characters in the name.

BLANKS should be called before the name is placed in the XPNAM buffer. The name must be in XPNAM before calling CHECKN.

CONTINUATION INDICATOR: FINISH

This routine types CONTINUE after a command has been executed.

The calling sequence is:

```
CALL FINISH
```

SYSTEM DIRECTORY LISTER: LISTER

This routine lists the system directories. The directory is determined by the state of the ITAMAS switch located in the common block called COMSWT. To list a single file name LOOK must be called to set the IFOUND pointer, IPASS must be set to IFOUND, LOCATT must be set to IFOUND before calling LISTER. To list the whole directory IPASS must be set to one and LOCATT must be set to zero before calling LISTER. IPASS is a variable in the common block called COMPAS.

The calling sequence is:

```
CALL LISTER
```

ITAMAS=0 Before call list PAGET.

ITAMAS=1 Before call list XLIBD.

SIXBIT DEVICE NAMES: CONDTA,CONMTA

These routines return SIXBIT device names.

The calling sequence for CONDTA is:

```
CALL CONDTA(N,DEVICE)
```

N $0\leq N\leq 6$. Is one less than a Dec tape device number.

DEVICE The returned SIXBIT device name.

The calling sequence for CONMTA is:

CALL CONMTA(N,DEVICE)

N 0≤ N≤ 3. Is one less than a Magnetic tape device number. If N=3 the device is DSK.

DEVICE The returned SIXBIT device name.

PACKING CHARACTERS: PACK

PACK is used to pack characters from one in a word to five in a word. The calling sequence is:

CALL PACK(TEST,XSTRIN,N)

TEST The word where the packed characters are stored.

XSTRIN A vector of the characters to be packed.

N 0< N≤ 5. Is the number of characters to pack in TEST.

SECTION III

DISPLAY

RAND TABLET SERVICE ROUTINE: GNF

This is a routine that allows a user to INIT the RAND TABLET. Details about this routine are found in reference (2).

GENERALIZED DISPLAY: GENDIS

This program operates the OLMGS in a general way using the RAND TABLET. Details about this program can be found in Part I and Part II.

GENERALIZED RAND TABLET INPUT: GENPUT

This program uses the OLMGS and routines GRDVAL, VALUEY, VALUEX, TPAGE, YESNO, RECTAN, XDATA and FINHIT to allow a generalized input graph using the RAND TABLET. Details about this routine can be found in Part I and Part II.

CREATING A GENERAL GRID: GRDVAL

This routine creates a general grid from the data limits XL, XR, YB and YT given as arguments.

The calling sequence is:

```
CALL GRDVAL(XL,XR,YB,YT,DC,ITEK,ISLAB,XLABEL,YLABEL)
```

XL,XR	Left and right limits of the data area.
YB,YT	Bottom and top limits of the data area.
DC	The data area grid density. DC should never be a value less than 3.0; values of 8.0 to 20.0 are recommended.
ITEK=0	No labels on axis.
≠0	Label axis.

ISCLAB=0 Standard labels.

≠0 Scientific labels.

XLABEL A vector of fifty or less characters to be placed on the X-axis in A5 format.

YLABEL A vector of fifty or less characters to be placed on the Y-axis in A5 format.

VALUE GENERATION FROM SCALED AREA: VALUEX,VALUEY

These routines use an inverse mapping on the grid scale area to obtain the data value of a raster coordinate. The coordinates are left in a buffer by the RAND TABLET service routine. Details about the RAND TABLET service routine can be found in reference (5).

The calling sequence for VALUEX is:

```
CALL VALUEX(XL,XR,ICOR,VALUE)
```

XL,XR Left and right limits of the data area.

ICOR X-axis coordinate.

VALUE The value of the scaled data area associated with the X-axis at this coordinate.

The calling sequence for VALUEY is:

```
CALL VALUEY(YB,YT,ICOR,VALUE)
```

YB,YT Bottom and top limits of the data area.

ICOR Y-axis coordinate.

VALUE The value of the scaled data area associated with the Y-axis at this coordinate.

DISPLAYING CHARACTER STRINGS: TPAGE

This routine is used to display character strings.

The calling sequence is:

CALL TPAGE (IX,IY,SAVF,N)

IX,IY — The raster coordinate of the point where the string will begin.

SAVF — A vector of characters.

N — $0 < N \leq 15$. The number of words of characters.

DECISION BOX: YESNO,RECTAN

These routines generate a decision box. YESNO displays YES,NO in a box generated by RECTAN.

The calling sequence for YESNO is:

CALL YESNO(IX,IY)

IX,IY — The coordinate of the YES. NO will be located at IX,IY-50.

A box will be drawn by YESNO by calling RECTAN.

The calling sequence for RECTAN is:

CALL RECTAN(IX,IY)

IX,IY — The coordinates of the center of the box.

GENERAL DATA DISPLAY: PDP342

This routine uses the OLMGS to do generalized data display. Any details may be found in Part I.

The calling sequence is:

```
CALL PDP342(NPLOT,DC,NCHAR,NPTS,X,Y,ISCLAB,XLABEL,YLABEL,IDF,IERR)
```

NPLOT=1 Use new graph.

NPLOT=2 Use same graph.

DC Limits the density of the grid. DC should <u>never</u> have a value less than 3.0; values of 8.0 to 20.0 are recommended.

NCHAR $0 < \text{NCHAR} \le 5$. The plot character selected.

NCHAR	PLOT CHARACTER
1	.
2	*
3	X
4	0
5	0

NPTS The number of points to plot.

X X-axis values.

Y Y-axis values.

ISCLAB=0 Standard labels.

ISCLAB≠0 Scientific labels.

XLABEL A vector of fifty or less characters in A5 format. These characters will be placed on the X-axis.

YLABEL A vector of fifty or less characters in A5 format. These characters will be placed on the Y-axis.

IDF=0 No axis labels.

IDF≠0 Axis labels.

IERR≠1 After return an error occurred.

PARAMETER INPUT ON THE RAND TABLET: XDATA

This routine allows a user to input data via the RAND TABLET. IRE2DI should be set to some maximum data value. Details on the use of this routine can be found in Part I.

The calling sequence is.

```
CALL XDATA
```

IRE3DI	The value of the input data.

IRE2DI and IRE3DI are located in the common block called REGPAS.

HITTING A DECISION BOX: FINHIT

This routine is used to flag a decision box. Details about how this routine works can be found in Part I. Before a call to this routine a call to GFNINS an entry point in GNF must be made.

The calling statement is:

```
CALL FINHIT
```

IREGDI	The X-coordinate of the hit.
IRE1DI	The Y-coordinate of the hit.

IREGDI and IRE1DI are located in the common block called REGPAS.

FUNCTION KEY INIT: FUNINI

This routine is used to INIT the function keys.

The calling sequence is:

```
CALL FUNINI(ICHANN,IER)
```

ICHANN	$0 \leq$ ICHANN ≤ 15. The user channel number
IER=1	INIT error

FINDING A FUNCTION KEY DEPRESSION: KEYNUM

This routine returns the number of the function key depressed.

The calling sequence is:

CALL KEYNUM(ICHANN,NUM)

ICHANN $0 \leq \text{ICHANN} \leq 15$. The user channel number.

NUM   The number of the function key depressed.

TURNING OFF FUNCTION KEY LIGHTS: LIGHTS

This routine turns off the function key light and returns the light number turned off.

The calling sequence is:

CALL LIGHTS (ICHANN,LIGHT)

ICHANN $0 \leq \text{ICHANN} \leq 15$. The user channel number

LIGHT   The light number turned off.

CLOSING FUNCTION KEY CHANNEL: CLOFBX

This routine closes the function key channel.

The calling sequence is:

CALL CLOFBX(ICHANN)

ICHANN $0 \leq \text{ICHANN} \leq 15$. The user channel number.

RELEASING FUNCTION KEY CHANNELS: RELFBX

This routine releases function key channels.

CALL RELFBX(ICHANN)

ICHANN $0 \leq \text{ICHANN} \leq 15$. The user channel number.

Any details about the Function Key Monitor Service Routine may be found in reference (7).

FKLP MODE PROGRAM: FKLP

This program assigns the OLMS commands to the function keys. In this mode the system is operated by the function keys.

SECTION IV

INPUT-OUTPUT

CLEAR DEC TAPE DIRECTORY: CLEAR,CCLEAR

This routine uses the UTPCLR program operator to clear the directory.

The calling sequence for CLEAR is:

```
CALL CLEAR(ICHANN)
```

ICHANN $0 \leq ICHANN \leq 15$. A user channel.

The program CCLEAR is a CHAIN file that INITS a device and clears the directory by calling CLEAR.

WRITING AND READING SYSTEM DIRECTORIES: NTAD,DIRD

These programs are CHAIN files used to write and read the system directories using Dec tape and Disk.

The use of NTAD is:

NTAD writes a system directory on Dec tape or Disk.

The use of DIRD is:

DIRD reads a system directory from Dec tape or Disk.

The device and directory is determined by the command given to INTER.

READING AND WRITING DATA ON DEC TAPE OR DISK: GETDD,SADA

These programs are CHAIN files used to read and write data.

The use of GETDD is:

GETDD reads data from Dec tape or Disk into the system buffer.

The use of SADA is:

SADA writes data in the system buffer on Dec tape or Disk.

The device is determined by the command given to INTER. The system buffer is

SYSBUF located in the common block called COMBUF.

DELETING FILES FROM DEC TAPE OR DISK: CDEDAS

CDEDAS is a CHAIN file that deletes files from DEC tape and DISK.

The CDEDAS program calls RENAME to delete file names. Any details about this program can be found in reference (1).

RENAMEING FILES: RENAME, CRENA

The RENAME routine is used to rename file.

The calling sequence for RENAME is:

```
CALL RENAME(XNAME,EXT,XNAME1,EXT1,IER,DEVICE,ICHANN)
```

XNAME,EXT — The name and extension of the file to be renamed.

XNAME1,EXT1 — The new name and extension of the file.

IER=1 — Init error
=2 — LOOKUP error
=3 — RENAME error

DEVICE — The SIXBIT device name

ICHANN — $0 \leq \text{ICHANN} \leq 15$. The user channel number.

The program CRENA calls RENAME to rename files. For details see reference (1).

ENTERING FILE NAMES IN DEC TAPE AND DISK DIRECTORIES: ENTER

This routine enters file names in the Dec tape and Disk directories.

The calling sequence is:

```
CALL ENTER(XNAME,EXT,IER,ICHANN)
```

XNAME,EXT — The file name and extension to be entered.

IER=4 — ENTER error

ICHANN — $0 \leq \text{ICHANN} \leq 15$. The user channel number

CLOSING CHANNELS: CLOSEC

This routine closes a channel and resets the JOBFF pointer.

The calling sequence is:

```
CALL CLOSEC(ICHANN,JOB,IER)
```

ICHANN	$0 \leq \text{ICHANN} \leq 15$. The user channel.
JOB	The contents of the old JOBFF pointer.
IER=1	Closing error.

JOB and ICHANN are in the common block called INOUT. JOB is set by a call to BUFFER. For details see reference (1).

CHECKING FOR A FILE IN THE DEC TAPE AND DISK DIRECTORIES: CHECKF

This routine does a LOOKUP on a file name.

The calling sequence is:

```
CALL CHECKF(XNAME,EXT,ICHANN,IER)
```

XNAME,EXT	The file name and extension to LOOKUP.
ICHANN	$0 \leq \text{ICHANN} \leq 15$. The user channel number.
IER=2	The file was not found.

For details see reference (1).

WRITING AND READING MAGNETIC TAPE DIRECTORIES:MTODIR,MTIDIR

These routines are used to write and read magnetic tape directories. The magnetic tape directory is DIRECT. DIRECT is located in the common block called MTAPE.

The calling sequences are:

```
CALL MTODIR
```

Write the tape directory.

```
CALL MTIDIR
```

Read the tape directory.

POINTER ROUTINES: CDRBUF,NUMWDS

These routines are used to manipulate pointers and plant data.

The calling sequence for CDRBUF is:

```
CALL CDRBUF(IBUF)
```

IBUF — A link variable. The contents of the right half of the location pointed at by IBUF is returned.

The calling sequence for NUMWDS is:

```
CALL NUMWDS(IBUF,WORDS)
```

IBUF — A link variable.

WORDS — The number to be stored. The contents of WORDS are stored in the right half of the location pointed at by IBUF.

ALLOCATING BUFFERS: BUFFER

Input or output buffers are allocated by this routine.

The calling sequence is:

```
CALL BUFFER(IND,IER,DEVICE,ICHANN,IBUF,MODE,NUMBUF,JOB)
```

IND=0 — Allocate output buffers.

IND≠0 — Allocate input buffers.

IER=1 — After return INIT error.

DEVICE — SIXBIT device name.

ICHANN — $0 \leq$ ICHANN ≤ 15. The user channel.

IBUF — A link variable that points to the buffer header set by the routine.

MODE — The device mode. For details see reference (1).

NUMBUF — The number of buffers.

JOB — The contents of the old JOBFF.

OUTPUT: OUTI,OUT

These routines are used to output data.

The calling sequence for OUTI is:

```
CALL OUTI(ICHANN,IER)
```

ICHANN	$0 \le \text{ICHANN} \le 15$. The user channel number.
IER=3	Bit transfer error.

Before output on a channel, OUTI must be called to initialize the channel.

The calling sequence for OUT is:

```
CALL OUT(ICHANN,IER)
```

ICHANN	$0 \le \text{ICHANN} \le 15$. The user channel number.
IER=4	Bit transfer error.

OUT should be called one time for each buffer. Data may be transfered to the buffers using the address in the buffer header pointed at by IBUF. The routine called TRANSF can be used to block transfer a block of data to the output area. For details see reference (1).

INPUT: INI,GET

These routines are used to input data.

The calling sequence for INI is:

```
CALL INI(ICHANN,IER)
```

ICHANN	$0 \le \text{ICHANN} \le 15$. The user channel number.
IER=6	Initialization error.
IER=3	Bit transfer error.

Before an input on a channel INI must be called to initialize the channel.

The calling sequence for GET is:

```
CALL GET(ICHANN,IER)
```

ICHANN	$0 \leq$ ICHANN ≤ 15. The user channel number.
IER=7	Input error.
IER=5	Bit transfer error.
IER=8	End of file was encountered.

GET should be called one time for each buffer. Data may be transfered from buffers using the address in the buffer header pointed at by IBUF. The routine called TRANSF can be used in conjunction with GETADR to block transfer a whole buffer of data to the user area. For details see reference (1).

MAGNETIC TAPE OPERATIONS: TAPE

This routine can do all of the magnetic tape MTAPE functions.

The calling sequence is:

```
CALL TAPE(ICHANN,FUNCTION,IER)
```

ICHANN	$0 \leq$ ICHANN ≤ 15. The user channel.
FUNCTION	See reference (1).
IER=1	Function error.

TESTING FILE STATUS BITS: STATUS

This routine is used to test file status bits.

The calling sequence is:

```
CALL STATUS(ICHANN,MASS,IER)
```

ICHANN	$0 \leq$ ICHANN ≤ 15. The user channel number.
MASS	An integer representing the bits to test.
IER=1	An error occurred associated with the MASS.

The bit information can be found in reference (1).

GENERAL INPUT,OUTPUT: IFILE,OFILE,IFILED,OFILED

These routines read and write data on magnetic tape, Dec tape and Disk.

The calling sequence for IFILE is:

CALL IFILE

Before calling IFILE a pointer where the data will be stored and a pointer to the buffer ring must be given. ITBUF is a link variable that must point to the area where the data will be stored. IFBUF is a link variable that must point to the buffer ring. NUMBUF must be equal to the number of buffers.

The calling sequence for OFILE is:

CALL OFILE

Before the call ITBUF must point to the buffer ring. IFBUF must point to the area where the data will be moved from during the output. NUMBUF must be equal to the number of buffers.

ITBUF,IFBUF and NUMBUF are located in the common block called INOUT.

OFILED and IFILED are defined on the same pointers as OFILE and IFILE. The difference is in the buffer size for DEC tapes. These routines are used for DEC tape.

RELEASING CHANNELS: RELEAS

This routine releases channels.

The calling sequence is:

CALL RELEAS(ICHANN,JOB)

ICHANN	$0 \leq \text{ICHANN} \leq 15$. The user channel number.
JOB	The contents of the old JOBFF.

WRITING AND READING SYSTEM DIRECTORIES ON MAGNETIC TAPE: WRTDIR,REDDIR

These routines write and read the system directories on magnetic tape.

The calling sequence for WRTDIR is:

CALL WRTDIR.

The calling sequence for REDDIR is:

CALL REDDIR.

INPUTTING AND OUTPUTTING DATA ON MAGNETIC TAPE: GETF,SAVEM

These routines input and output data on magnetic tape by calling IFILE and OFILE.

The calling sequence for GETF is:

CALL GETF.

This call reads data.

The calling sequence for SAVEM is:

CALL SAVEM

This call writes data.

MAGNETIC TAPE CHAIN FILES: CGETF,CDIR,CNTAP,CSAVEM,CDESCR,CDELE

These CHAIN programs are used for INPUT-OUTPUT on magnetic tape.

CGETF	Reads input data into the system buffer.
CSAVEM	Writes data on magnetic tape from the system buffer.
CDIR	Reads the magnetic tape directory into DIRECT.
CNTAP	Writes the magnetic tape and system directories on magnetic tape.
CDESCR	Writes file descriptions.
CDELE	Deletes file name from the magnetic tape directory. This program is used to rename files on magnetic tape.

SECTION V

MISCELLANEOUS

SLEEPING A JOB: RESTIN

This routine uses the SLEEP program operator to stop a job and continue automatically after an elapsed real time of ITIME.

The calling sequence is:

CALL RESTIN(ITIME)

ITIME $0 \leq \text{ITIME} \leq 68$. The number of seconds to sleep.

LOGIN AND TAPE ASSIGNMENTS

LOG

PROJECT NUMBER $\{,|/\}_1^1$PROGRAMMER NUMBER

Assign or Deassign Devices

$\{\text{AS}|\text{DEAS}\}_1^1\{\text{<Blank>}\}_1^1\{\text{<Device Name>}\}_1^1$

Putting Source Deck on Disk:

R PIP

DSK:$\{\text{<Name>}\}_1^1 \leftarrow$ CDR:

FORTRAN COMPILE

COMPILE$\{\text{<Blank>}\}_1^1\{\text{<Name>}\}_1^1$

PUTTING THE OLMS ON DISK

R PIP

DSK:(XB) $\leftarrow$ $\{\text{<Device Name>}\}_1^1$:$\{\text{<Name>}\}_1^1$

BUILDING A CHAIN FILE

LOAD $\{<\text{Blank}>\}_1^1$INTOP,$\{<\text{NAME}>\}_1^1$

SAVE $\{<\text{Blank}>\}_1^1\{<\text{Device Name}>\}_1^1\{<\text{Blank}>\}_1^1\{<\text{Name}>\}_1^1$

RUNNING THE OLMS

RUN $\{<\text{Blank}>\}_1^1\{<\text{Device Name}>\}_1^1\{<\text{Blank}>\}_1^1$OLMS

For any details see references (1), (14).

INDEX OF ROUTINES

REFERENCES

1. DEC - 10 - MTEO - D, Time-Sharing Monitors; Multi-programming Monitor (10/50) Swapping Monitor (10/50), Digital Equipment Corporation, Maynard, Massachusetts, November 1968.

2. Feldmann, R. J., Rand Tablet Service Routine, NIH, Division of Computer Research and Technology, Bethesda, Maryland, August 1969.

3. Hill, Edward, The On-Line Modeling System, NIH, Division of Computer Research and Technology, Bethesda, Maryland, September 1969.

4. Hill, Edward, Programmers' Reference Manual for the On-Line Modeling Graphical System, NIH, Division of Computer Research and Technology, Bethesda, Maryland, June 1969.

5. Lewis, Harry, Fortran-Lisp Display Routines, NIH, Division of Computer Research and Technology, Bethesda, Maryland, April 1969.

6. Freedman, S. R., Filer, M.I.T./L.N.S., Cambridge, Massachusetts, February 12, 1968.

7. Vreenegoor, H., Lewis, H., Function Box Service Routine, NIH, Division of Computer Research and Technology, Bethesda, Maryland, September 1969.

8. Lewis, Harry, Assembly Language for DEC 340 Display, NIH, Division of Computer Research and Technology, Bethesda, Maryland, April 1969.

9. Adler, C., Sanford, 340 Display Programming Manual, Decus No. 7-13, New York University's Department of Industrial Engineering and Operations Research, Bronx, New York.

10. Stromberg-Carlson, Programmers' Reference Manual, Data Products, San Diego, Calif., October 1964.

11. Hill, Edward, A Proposed On-Line System for Modeling Networks, NIH, Division of Computer Research and Technology, Bethesda, Maryland, February 1969.

12. DEC-10-LOVA-D, CHAIN, Digital Equipment Corporation, Maynard, Massachusetts, February 8, 1968.

13. Bruce, M. C., PDP-10 Equipment, NIH Internal Memorandum, October 9, 1968.

14. DEC-10-NGCA-D, PDP-10 System User's Guide, Digital Equipment Corporation, Maynard, Massachusetts, 1967.

15. Wegner, P., Introduction to System Programming, The Automatic Programming Information Center, England, 1964.

16. Knuth, D. E., The Art of Computer Programming, Addison-Wesley Publishing Company, Rading Massachusetts, 1968.

17. Rosen, Saul, Programming Systems and Languages, McGraw-Hill, New York, 1967.

18. Lee, J.A.N., The Anatomy of a Compiler, Reinhold Publishing Corporation, New York, 1967.

19. DEC-10-PPCO-D, Peripheral Interchange Program, Maynard, Massachusetts, 1968.

20. DEC-10-ETEB-D, Text Editor and Corrector Program, Maynard, Massachusetts, 1968.