NASA

Technical Memorandum 79584

The LHEA PDP 11/70 Graphics Processing Facility User's Guide

Data Management and Programming Office
Code 664

JULY 1978
SECOND EDITION

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

TM 79584

THE LHEA PDP 11/70 GRAPHICS PROCESSING FACILITY USER'S GUIDE

Data Management and
Programming Office
Code 664

July 1978
SECOND EDITION

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

CONTENTS

1. The PDP 11/70

It is the intention of this document to put in one place, those pieces of information that the new user of the LHEA PDP 11/70 facilities needs in order to most quickly phase in. It is also intended to act as a reference guide for the occasional user.

All of the information contained in the document is available or has been available to all users of this facility. However, it is spread throughout six volumes of DEC manuals and, in some cases, is information other users have gathered and is not documented elsewhere.

1.1 Philosophy

The Lab for High Energy Astrophysics purchased the DEC PDP 11/70 system in November, 1975 to serve as a graphics data processing facility. It is not expected that the 11/70 will serve as a general purpose computer facility in the manner of the SACC 360/75 and 360/91. The justification for the use of the LHEA PDP 11/70 has always been, and remains, to provide an interactive graphics capability to be used in support of the scientific data analysis activities of Lab personnel.

1.2 Operations Management

This section will present some of the more important aspects of management of the facility which affect the user community. Some items will be presented via memos issued to the user community in the past. These memos follow this section.

1.2.1 Assignment of the UIC

All users sign-on the 11/70 via the use of the Users Identification Code. This code is assigned to the user by the systems manager and has the form [xxx,yyy] where xxx is the group code and yyy is the user code within the group.

In the LHEA facility, the following convention for group codes is used:

Group Code (octal)	Group
100	Gamma Ray Studies
200	Cosmic Ray Studies
300	X-Ray Studies

The user code is assigned in numerical order starting with $100_{(8)}$.

1.2.2 Disk Space Management
[See Memo dated 2/28/78, following this section].

The policy stated in the memo has been changed. The preserve is done every Tuesday and Friday and the scratch (i.e. purge) is done every other Tuesday preceeding the preserve.

1.2.3 Vector General Daily Use (SD) Schedule
[See Memo dated 3/31/77, following this section].

This schedule is currently undergoing revision and may change. If it is changed, that information will be communicated to all users and recipients of the document via memo.

1.2.4 Special Use of the Facilities

So that we might give maximum cooperation to users who wish to use the facility for demonstrations, special requirements for publication, etc, the system manager should be contacted well in advance concerning their needs. It is necessary to do this so we can properly schedule these requirements relative to hardware repairs and maintenance, and installation of new equipment or software, etc.

1.2.5 Preventive and Remedial Maintenance

Preventive maintenance is performed on the PDP 11/70 by DEC the last Tuesday of every month, from 0830 to 1200. Remedial maintenance is scheduled by the system manager on an as-needed basis. Any user who discovers a need for remedial maintenance should notify the systems manager.

1.2.6 Communication to the Users

[See the Memo dated 3/8/77 following this section].

Users will be notified about system related items in two ways. The user emergency notice area is discussed in the referenced memo. Information will also be made available to the user via the notices printed to the terminal when he signs on (the HEL command).

1.2.7 Terminal Use

[See the Memo dated January 23, 1978]

1.3 PDP 11/70 Hardware and Software Overview

1.3.1 PDP 11/70 Hardware

The Lab's PDP 11/70 configuration is shown in Figure 1. Remarks about the capabilities of some of these hardware components and the software system follow:

THE UNIBUS – The UNIBUS is used to move data between devices in a synchronous manner. The maximum transfer rate is one million bytes per second.

THE FLOATING POINT PROCESSOR (FPP) – The FPP has its own set of 6 64-bit accumulators used to do both single precision (32 bits) and double precision (64 bits) arithmetic. The operation of the FPP is overlapped with the CPU operation in order to increase throughput. Two double-precision floating point numbus can be multiplied in 9 microseconds.

CACHE MEMORY – The cache is a high-speed solid-state memory with 2,048 byte capacity used as a buffer between the CPU registers and main memory. The cycle time of the cache is 240 nanoseconds. When a memory read request is issued by the CPU, cache is checked to see if the desired information is present. If it is present, no memory read is required. When a memory write is initiated, the information is written to the cache and memory to insure both have the most up to date data.

THE RP04 DISK – The single RP04 disk has a capacity of 88 million bytes and a single controller can support 8 drives. Data can be transferred in blocks of 2 to 130,712 bytes at a rate of 806,000 bytes-per-second.

THE TU16 TAPE DRIVES – These drives are industry compatible, 9 track, dual density – NRZI 800 bpi and Phase Encoded 1600 bpi – with a speed of 45 ips to attain a maximum transfer rate of 72,000 characters-per-second.

CR11 CARD READER – The CR11 reads cards at a rate of up to 300 cards per minute. This unit does not have a punch capability.

HAZELTINE TERMINALS – The Hazeltine 2000 Video terminals are used as remote terminals to the 11/70. The screen size is 74 characters by 27 lines.

The RX-11 Floppy Disk System

This is a dual floppy disk drive system using single sided, single density, preformatted diskettes with a storage capacity of 256,256 bytes, or about 450 blocks.

THE VERSATEC – The VERSATEC (Model 1200A) is an electrostatic printer/plotter. On the 11/70 hardware configuration it is used as a hard copy device to the Vector General. However, it can act as a printer when necessary.

1.3.2 Vector General Graphics Display Hardware

The VG used with the Labs 11/70 is a model 2D3 with the following hardware features:

- direct memory access from the CPU
- 21" rectangular; 13" × 14" CRT tube
- .020" spot size
- 32 intensity levels
- 30" × 30" dynamic plot range
- 4096 × 4096 addressable locations
- 2% positioning accuracy
- KB1 Alphanumeric Keyboard with 72 keys
- FS2 Function Keyboard with 32 momentary, 1 interrupt keys
- LP3 light pen with a 3μ sec response time
- Character set of 96 ASCII and 96 special characters

Section 6 of this guide gives the reader a detailed description of the use of the VG.

1.3.3 11/70 Software Overview

The operating system used on the LHEA 11/70 is the most current version (6.2) of RSX-11D. This is a multi-programmed, real-time operating system designed for a wide range of applications. It incorporates the memory management system thereby allowing a program to be loaded anywhere in memory without modification.

The basic program unit executing under RSX-11D is the task, which consists of a program module or a set of program modules. The task is limited to a size of 32K words.

The RSX-11D File System provides support of files on disk volumes. Four levels of file protection are supported via four levels of access. Both sequential and random access are supported for block structured and fixed length record files. Variable length record files can only be processed sequentially. All types of files can be expanded dynamically.

Console operations for RSX-11D are supported by tasks called the Monitor Console Routines (MCR). Some operations include:

- Log on and log off
- Mount or dismount peripheral volumes
- Initialize peripheral volumes
- Run and schedule tasks
- Assign a logical unit number
- List LUN assignments for an indicated task

The program development and utility functions provided by the system include:

- Peripheral Interchange Program, PIP
- File Transfer Utility, FLX
- File Compare Utility, CMP
- File Dump Batch, DMP
- Character Oriented Text Editor, EDI
- Task Builder Overlay Linker, TKB
- Single Stream Batch, BATCH
- Fortran IV Plus, FOR

Also, a scientific subroutines package is available.

1.4 Terminal Control Conventions

The average user should be familiar with the following special key functions:

(1) TAB Key.

The TAB key automatically moves the cursor to the next tab stop, if any, whether vertical, horizontal, or diagonal. Tab stops are set automatically when the display contains both background and foreground fields. The first foreground character following a background field is a tab stop. If there are no tab stops set, the tab key will cause the cursor to move to the lower right hand corner of the display.

(2) Escape (ESC) Key.

Use of this key is optional, depending on communication software being used. ESC generally is used to generate a program interrupt signal.

(3) Control (CTRL) Key.

The control key is used in conjunction with other character keys to generate non-printing characters for a number of reasons including security, function codes, etc. When CTRL is used, it should be depressed and held while the other required character key is depressed. CTRL sends special codes by altering the code pattern of the other key used with it.

(4) Line Feed (LF) Key.

The LF key causes the line feed character to be transmitted when operating in standard full or half duplex mode, but has no other effect. It does not move the cursor down the screen and does not cause the line feed character to be stored.

(5) Carriage Return (CR) Key.

The CR key moves the cursor back and down to the beginning of the next lower line, thereby accomplishing both carriage return and line feed.

(6) Number Key Cluster.

These keys transmit the same code as the numeric keys across the top of the keyboard and are provided in an adding machine cluster to facilitate entry of numeric data.

(7) Cursor Control Keys.

The HOME key moves the cursor immediately to the home position (first character position in the top line). The cluster of four arrow keys are cursor-stepping keys. Each moves the cursor one space in the indicated direction.

These keys may be pressed in conjunction with the REPEAT key for rapid cursor movement. Cursor positioning with these keys is completely non-destructive; it will not alter any characters in the display.

KEYS	FUNCTION
CTRL B	Start paper tape input. Signals computer to start reading the tape.
CTRL T	Terminate paper tape input.
CTRL V	Deletes all of the type-ahead buffer.
CTRL C	Causes MCR to be activated.
CTRL Z	Logical End-of-File.
RETURN	Terminates the current line and causes the system to print the prompt for the next command.
RUBOUT	Causes the most recently typed character to be deleted and the cursor if left where it was before the character was typed.
ALT or ESC	Terminates MCR. Used when requesting a program that is to interact with the operator.
CTRL I	Causes a horizontal tab. Tab stops are set by the software at every eight character position (9, 17, 25, 33).
CTRL K	Causes a vertical tab of one line.
CTRL L	Causes a form feed to the top of the next page.
CTRL O	Interrupts system output to the terminal. Successive pressings cause start and stop.

CTRL R	Causes the system to print the current terminal line.
CTRL U	Cancels the current input line.
CTRL Q (XON)	Starts output to terminal.
CTRL S (XOFF)	Stops output until CTRL Q is typed.

UNITED STATES GOVERNMENT

Memorandum

TO : DISTRIBUTION **DATE:** March 8, 1977

FROM : G.A. Muckel
Mary Ann Esfandiari

SUBJECT: 11/70 Update

We would like to inform you of two items:

(1) If any user requires 11/70 support (time on the machine, software help, etc.) to support his efforts in preparing for a paper, presentation, demonstration, etc., please coordinate this request with either of us as soon as it becomes known to you. This will enable us to give you maximum cooperation and support.

(2) We have installed a "user emergency" notice area on the bulletin board at the computer room entrance. This is the quickest way we can notify the user community of items that need their immediate attention (like a corrupted disk, etc.). If, when you enter the computer room, the red arrow is displayed, read the posted items.

Thanks very much.

Gerald A. Muckel

Mary Ann Esfandiari

GAM:gee

GSA FPMR (41 CFR) 101-11.6

UNITED STATES GOVERNMENT

Memorandum

TO : DISTRIBUTION

DATE: March 31, 1977

FROM : G.A. Muckel/M.A. Esfandiari

SUBJECT: New Daily Use Schedule for V.G.

The following schedule for the use of the V.G. will be put into effect beginning 4/4/77. It will remain in effect on a trail basis until mid-July unless we find it causes a major problem to some group. We will attempt to schedule 664's use of the V.G. for system items between 11:00 and 1:00.

	SAS	COSMIC	X-RAY	SYSTEM	FREE
8:00- 9:00	X				
9:00-10:00	X				
10:00-11:00			X		
11:00-12:00	X			X	
12:00- 1:00				X	X
1:00- 2:00		X			
2:00- 3:00			X		
3:00- 4:00			X		
4:00- 5:00	X				
5:00- 6:00			X		
After 6:00					X

DISTRIBUTION:
P. Serlemitsos/X-ray
D. Thompson/Gamma-ray
T. von Rosenvinge/Cosmic Ray

GAM:gee

Buy U.S. Savings Bonds Regularly on the Payroll Savings Plan

January 23, 1978

TO: All 11/70 Users

FROM: G.A. Muckel
M.A. Esfandiari

SUBJECT: Terminals on the 11/70

Effective Wednesday, January 25, the following policy is adopted for terminal use on the PDP-11/70:

(1) The Hazeltines are to be allocated to the research groups as follows:

2 - X-ray
1 - Cosmic Ray
1 - Gamma Ray/Code 664

(2) Anyone can use any terminal, but can be bumped by a member of the "owner" group with 5 minutes' warning.

(3) One additional Hazeltine will be added to the system in the near future (within 60 days) to be used as a "free" one on a first-come-first-serve basis. If any "owned" Hazeltine goes down, the "free" one will be allocated to the owner group until all are back in service.

(4) You are reminded that all VG runs are to be submitted from the LA36.

This terminal policy will be constantly evaluated and will be adjusted as needed.

Gerald A. Muckel, Head
Data Management & Programming Office

Mary Ann Esfandiari

February 28, 1978

TO: F.B. McDonald, Cosmic Ray Group
C.E. Fichtel, Gamma Ray Group
E. Boldt, X-ray Group

FROM: G.A. Muckel
M.A. Esfandiari

SUBJECT: PDP 11/70 Disk Space Management

In order to ease the problem of disk space management on the PDP 11/70, the following action will be taken:

(1) The heads of each research group will name one person to manage the portion of disk allocated to that group.

(2) The allocation of space is:

Group	% of Available Space	Total # Blocks
X-ray	50	50000
Cosmic Ray	25	25000
Gamma Ray	25	25000

(3) Whenever a group goes over their allocation, the person named as 11/70 liaison from that group will be notified. He then has the immediate responsibility of getting his group's use back within limits. The report of disk use he gets will include the number of blocks being used by each group member, as well as the number of blocks used in excess of group allocation.

This policy will become effective as soon as group representative names are received by us.

2. Starting the PDP 11/70

2.1 Booting.

The steps for cold-starting the PDP 11/70 are as follows:

(a) Push the START/STOP button on the disk drive, which will begin to activate.

(b) Press HALT/ENABLE toggle switch on the CPU console down to HALT mode. Turn the key on the console to the POWER ON position.

(c) Toggle in the octal number 17765000 on the console. That is:

● ●●● ●●● ●●0 ●0● 000 000 000

where the dark toggles are up, white are down. Depress the LOAD ADDRESS toggle and release.

(d) Next, toggle in the disk device number octal number $70_{(8)}$ on the console. This is:

0 000 000 000 000 000 ●●● 000.

Lift the HALT/ENABLE toggle up to the ENABLE position. Depress the START toggle and release.

(e) A prompt for the current time is issued on the DEC writer by the PDP. Input is as follows: DAY (2 digits), Dash (–), MONTH (3 letters), Dash, Year (2 digits), Space, Hours (2 digits in the 24-hour clock style), Colon (:), Minutes (2 digits), Colon, Seconds (2 digits) and Carriage Return:

TIM>05-JUL-76 08:30:0 0(CR)

(f) The graphics package must now be installed by entering the following commands, turning VG screen on first with the ON/OFF button on lower left of the VG unit:

MCR>VGI(CR).

NOTE: If disk and power are already on, press HALT toggle and start at Step C.

2.2 Turning I/O Devices On

If the various I/O devices are off, press the START buttons on the Hazeltines and the ON/OFF button for the line printer. Leave the card reader off until use is desired.

2.3 Signing On

To sign on the Hazeltines, press CTRL and Key C simultaneously to get an MCR> response. Then type in HEL [X Y] where [X Y] is your UIC (User Identification Code). The computer will respond with another MCR>, unless a typing mistake occurred. In this case, start the procedure again.

2.4 Powering Down

To power down the PDP, follow this procedure:

1. After making sure that no one is presently using the computer, turn all Hazeltine CRT's off by depressing the POWER switch. Turn the Vector General screen off by pressing the switch to the OFF position.

2. Stop the computer by depressing the HALT toggle on the CPU and then pressing the START/STOP button on the disk drive.

3. Turn the CPU off by turning the key to the POWER OFF position.

NOTE: Line printer, DEC writer and card reader power switches should not be touched as these are controlled by the power key on the CPU console.

3. Files on the PDP 11/70

3.1 Maintenance using PIP

The PDP 11/70 provides each user with their own UIC- [ggg,nnn]. The first number, ggg, in your UIC indicates the group in which you work. The present scheme is: 100 – gamma-ray, 200 – cosmic ray, and 300 – X-ray astronomy. The second number is your identification within that group. These numbers are assigned in sequence beginning with the number $100_{(8)}$.

Use of wild cards in the output file specifiers is restricted. For the following PIP functions, the output file specifier may not have any wildcards:

(a) Copying a single file from device to device
(b) Concatenating files to a specified file
(c) Appending to an existing file
(d) Updating (rewriting) an existing file
(e) Listing a directory

When a list of files is to be copied, the output file specifier must be *.*;* or default.

In all cases in which wildcards are allowed in the output file specifier, the wildcard UIC form [*,*] is used to indicate that the output UIC is to be the same as the input UIC.

3.1.1 Purging

Although there are no restrictions on the number and size of files that users can maintain under their UIC, all users should perform periodic clean-up operations. At the beginning of each EDIT session and upon execution of the EDI command TOF a new version is created of a pre-existing source file. A new version is also created each time a compilation or task build is performed. A command to PIP such as the following gets rid of all but the latest version of all files

(1) PIP *.*/PU

Variations on this include purging all but the latest two versions

(2) MCR>PIP *.*/PU:2

and purging all but the latest in data (.DAT) files

(3) MCR>PIP *.DAT/PU

3.1.2 Deleting

To delete all files of a particular kind, the switch /DE is used. Some examples:

(1) Deleting all data files (all versions):

```
MCR>PIP *.DAT;*/DE
```

(2) Deleting all Source, Object and Task files called PROG:

```
MCR>PIP PROG .*;*/DE
```

For those who are reluctant to purge their old files, every two weeks a general disk purge is done of all files but the latest two versions. This is necessary to avoid disk overrun. A packed disk slows down I/O time because of fragmentation. It takes longer to retrieve a file in this way. Also, since the task builder requires a contiguous file, a full disk can make its job more difficult. (See section 1.3).

3.1.3 Renaming

If the bi-weekly disk purge interferes greatly with your file scheme or if you want to rename files for any reason, you can rename the files you don't want purged. This can be done using PIP and the /RE switch. In addition, with the /NV switch, the renamed file can be forced to have a version number which is one greater than the latest version of the previously existing file. Some examples:

(1) Rename PROG.FTN to PROG1.FTN

```
MCR>PIP PROG1.FTN=PROG.FTN/RE
```

(2) Rename PROG.FTN;3 to PROG1.FTN;4

```
MCR>PIP PROG1.FTN/RE/NV=PROG.FTN;3
```

3.1.4 PIP File Transfer

Frequently, it is necessary to transfer files from someone else's UIC to your own or vice versa. Or, you might have files on a magtape that you want to put in your disk space. This is accomplished easily with PIP. The general command consists of: OUTFILE=INFILE (no switches since copy is the PIP default). The "OUTFILE" is the file specifier of the file you want to copy TO and the "INFILE" is the file specifier of the file you want to copy FROM.

Some examples:

(1) Copy all files called FORT from UIC = [111,12] to UIC = [110,10]-

```
MCR>PIP [110,10]=[111,12]FORT.*;*
```

If no version is specified, the latest is assumed.

(2) Copy the latest version of FILE FORT.DAT from UIC = [30,30] to UIC = [40,40] -

```
MCR>PIP [40,40]=[30,30] FORT.DAT
```

(3) If you wanted to copy this to [50,50] changing the name to SAMP.DAT:

```
MCR>PIP [50,50]SAMP.DAT=[40,40] FORT.DAT
```

There are two restrictions. First, no wild cards (asterisks) are permitted in the output file name. Second, if an output file is specified, there may be only one input file.

Copying to or from tapes is basically the same procedure, only the input or output device would be MM0: or MM1: Also, the tape would have to be mounted beforehand. In transferring files from one UIC to another, the owning UIC of the file as well as its protection is preserved. In order to assure that the file(s) transferred into your disk is/are owned by you, the following PIP command is needed following the transfer:

All data files were transferred into your UIC:
[100,150] from UIC:[200,300] -

```
MCR>PIP [100,150]*.DAT;*/PR/FO
```

The /PR switch allows you to change the protection of a file if you wish. The /FO switch allows you to set file ownership to the UIC specified.

There are four categories of file protection:

1. System – Specifies what categories of access system accounts are allowed to the file.

2. Member – Specifies what categories of access the member has allowed himself.

3. Group – Specifies what categories of access other members in the same group have.

4. World – Specifies what categories of access have been given all other accounts not covered.

For each category the user can specify whether that category can Read, Write, Extend or Delete the file. The default access rights are:

RWED for System and Member
RWE for Group
R for World

The format of the command to alter protection of a file is as follows:

```
infile/PR[/SY[:RWED]][/OW[:RWED]]
    [/GR[:RWED]][/WO[:RWED]][/FO]
```

where /SY is the system subswitch
/OW is the number subswitch
/GR is the group subswitch
/WO is the world subswitch

for example, to change the protection of a file so that world has RWED privleges:

```
MCR>PIP PROG.FTN;3/PR/WO:RWED
```

If any of the above subswitches are present and no value is given then no privleges are granted for that category.

For example,

```
MCR>PIP FILE.FTN;10/PR/OW:RWE/GR:RWE/WO
```

sets the protection so owner and group have RWE privleges and world is denied all access.

Another feature of PIP is the /LI switch. This provides a directory listing of a UIC with the following information:

(1) File name, type, version.

(2) Number of blocks used (decimal).

(3) File Code:

(Null) – non contiguous.

C – Contiguous.

L – Locked.

(4) Creation date and time.

(5) Totals line, indicating total # of files and blocks used.

Some examples:

(1) Obtain a directory listing of UIC [150,150] on the line printer –

```
MCR>PIP LP:=[150,150]/LI
```

(2) Specify only FORTRAN files –

```
MCR>PIP LP:=[150,150]*.FTN/LI
```

(3) A /FU switch yields more detailed information on a UIC, e.g.,:

```
MCR>PIP LP:=[150,150]/FU
```

Some additional PIP switches include:

```
/BL:n
```

This subswitch specifies the number of contiguous blocks to be allocated to the output file, where n is an octal or decimal value. This is useful for copying a contiguous file and changing its size.

/CO

This subswitch specifies that the output file be contiguous. When copying contiguous files (.TSK) from magnetic tape, both /CO and /BL:n must be specified because PIP cannot determine the length of the input file when it allocates file space. (Space is allocated before copying begins.)

/BS:n

This switch defines the block size for magnetic tapes. It allows you to write bigger blocks onto magnetic tape. It can appear on the input or output file specifier. If the blocksize specified is smaller than the actual blocksize, an I/O error occurs.

/RW

This rewind switch allows you to rewind a magnetic tape. It can be applied to both input and output specifiers. However, if specified on the output side it erases the tape. When applied to the input specifier, /RW rewinds the tape before opening the input file. This can be used to save search time. If you know a file is behind the tapes current position, /RW rewinds the tape before searching for the file. This saves the time that otherwise would have been taken to search for the file between the current position and the end of the tape.

Presented here are the most frequently used switches to PIP. Additional information may be found in Chapter 2 of the Utilities Procedures Manual. PIP's error messages may be found in the "Error" section of the manual.

3.2 Backup of Personal Files

Due to the occasional problems that have occurred with the disk and the fact that the system is not protected against commands that can wipe out whole libraries, it is suggested strongly that you back up your UIC library on a periodic basis convenient to you. If you do occasional work a weekly basis might be suitable. However, for those who make major changes to their files on a daily basis, a weekly back-up might not be sufficient.

There are two methods you can use to back up your disk files, FLX and PIP. The following are some considerations in determining which method is best for you:

(1) FLX cannot handle large data files. ($\approx$100 blocks).

(2) FLX works well with source files and other files (.OBJ and .TSK) of reasonable size (< 100 blocks).

(3) FLX is faster than PIP and uses less tape.

(4) FLX doesn't copy multiple versions of a file and the one that it does copy may not be the latest version.

(5) FLX only recognizes 6-character data names, not 9.

(6) PIP can handle any files.

(7) PIP copies everything in your library (all versions of all files).

PROCEDURES:

Using FLX: Back-up.

(1) MCR>MOU MMØ:/CHA=[FOR]
(2) MCR>FLX MMØ:/ZE
(3) MCR>FLX MMØ:[Your UIC]/DO=DBØ:[Your UIC]*.*/RS
(4) MCR>FLX LP:=MMØ:[Your UIC]*.*/LI (directory listing)
(5) MCR>DMO MMØ:

Restoring.

(1) MCR>MOU MMØ:/CHA=[FOR]
(2) MCR>FLX DBØ:/RS/UI=MMØ:[Your UIC]*.*/DO
(3) MCR>DMO MMØ:

Using PIP: Back-up.

(1) MCR>INI MMØ:Volume Name (User Provided)
(2) MCR>MOU MMØ:Volume Name
(3) MCR>PIP MMØ:[Your UIC]=DBØ:[Your UIC]*.*;*
(4) MCR>PIP LP:=MMØ:[Your UIC]*.*;*/LI (directory listing)
(5) MCR>DMO MMØ:

Restoring.

(1) MCR>MOU MMØ:Volume Name
(2) MCR>PIP DBØ:[Your UIC]=MMØ:[Your UIC]*.*;*
(3) MCR>DMO MMØ:

Task files (.TSK) backed up on tape using FLX or PIP do not retain their contiguous status when restored to the disk. Therefore, in order to make runable versions of your programs after they have been restored to disk, you must use PIP to create contiguous disk images for each task file. For example, MCR>PIP FILENAME.TSK/CO=FILENAME.TSK. Each file has to be done individually since PIP does not allow wild cards in the destination.

A time-consuming, but space-saving way to transfer contiguous (.TSK) files using PIP from magnetic tape would be to specify both the /CO and /BL:n together on the output file.

For example,

MCR>PIP DBØ:[Your UIC]/CO/BL:n=MMØ:[Your UIC]PROG.TSK;15

Each file must be done individually.

3.3 File Creation and Editor

Creating data files or sources programs on the PDP is done through the use of the Line Text Editor Utility. This is done by entering the following response to the MCR prompt:

```
MCR>EDI FILENAME.TYPE
```

The Editor will respond with:

```
CREATING NEW FILE
INPUT
```

The Editor is now in input mode and the user can begin entering his source code following each line with a (CR). All source files should be "typed" as FTN (FORTRAN) or MAC (MACRO).

The Editor is capable of operating in two control modes:

(1) EDIT MODE (Command Mode)
(2) INPUT MODE (Text Mode)

To go from Input Mode to Edit Mode, the user types a Double (CR). To go from the Edit Mode into the Input Mode, the user types the Insert Command (I), followed by a (CR).

Edit Mode is characterized by an asterisk (*) as a prompt. EDI acts upon control words and data strings to open and close files; to bring in lines of text from an open file; to change, delete or replace information in an open file; or to insert single or multiple lines anywhere in a file. Files that are introduced to the Editor from the card reader should have the program BLANK run on them first to remove the trailing blanks.

Within Edit Mode there are two modes of accessing and manipulating lines of text. These modes are:

(1) Line-by-line Mode.

This allows the user to access lines of text one line at a time. It has the disadvantage that once a line is edited and written to the output file, it can only be accessed again by the user issuing a TOP command which places the pointer at the top of the file. However, it does have the advantage that when a search is being done, the whole file is searched and not just a block (80 lines), as in Block-edit Mode.

(2) Block-edit Mode.

This is the default editing mode. To use Line-by-line Edit Mode, it is necessary to issue the BLOCK OFF command. In this mode, 80 lines are made available for editing. However, EDI commands are executed only with respect to the current block. Lines of text can be referenced forward and backward without issuing a TOP command.

EDIT COMMANDS

COMMAND	FORMAT	DESCRIPTION
ADD & PRINT	AP [STRING]	Append [STRING] to current line and print new line.
BLOCK ON/OFF	BL ON or BL OFF	Switch editing modes.
BOTTOM OR END	BO END	Set current line to last line in file or block buffer. The commands are equivalent.
CHANGE	[n]C /STRING-1/ STRING-2	Replace String-1 with String-2 n times in the current line.

EDIT COMMANDS (Continued)

COMMAND	FORMAT	DESCRIPTION
DELETE	D [n] or D [-n]	Delete current line and next n-1 lines if n is (+); delete n lines preceeding current line if n is (-). (-n) block-edit only.
EXIT	EX	Close files, name output file and EDI exits.
INSERT	I [STRING]	Enter [STRING] following current line or enable input mode if [STRING] not specified.
KILL	KILL	Input and output file are closed.
LOCATE	[n] L [STRING]	Locate n-th occurrence of STRING.
RETYPE	RETYPE STRING	Replace current line with string or delete current line if string is null.
RENEW	REN [n]	Write current block to output file and read new block from input file.
SAVE	SA [n] [filespec]	Saves current line and next n-1 lines in specified file. If file not specified, lines are saved under SAVE.TMP.
TAB	TA ON or TA OFF	Turn on or off automatic tabbing. If TAB ON, all lines moved over 8 spaces.
TOP OF FILE	TOF	Return to top of input file and save all pages previously edited. This command creates a new version of your file.
PRINT	P [n]	Print current line and next n-1 lines.

EDIT COMMANDS (Continued)

COMMAND	FORMAT	DESCRIPTION
OLD PAGE	OL n	Return to TOF and read Page n into block buffer.
PAGE LOCATE	[n]PL (STRING)	Search successive blocks for the n^{th} occurrence of string.
UNSAVE	UNS [filespec]	Insert all lines from specified file following current line. If no file is specified, default file is SAVE .TMP.

In many cases with the EDI commands the user must identify a string of characters to be located and/or changed. To reduce the number of terminal entries, the following special string constructs might be very useful.

Case 1. String1 . . . String2

Any string that starts with string1, continues with any number of intervening characters, and ends with the first occurrence of string2.

Case 2. . . . string

Any string that starts at the beginning of the current line and ends with the first occurrence of string.

Case 3. string . . .

The first string that starts with string and ends at the end of the current line.

Case 4. . . .

The entire current line.

3.4 Preparing a FORTRAN Program for Execution

3.4.1 FORTRAN Compiler

Once a source file has been created, it is necessary to submit it to the compiler and task builder before it becomes a runable version (.TSK). The general procedure for source files is depicted below in Figure 1. Three characters above a box indicate how that operation is evoked via MCR and the three characters below a box are the output file extension type from that operation.

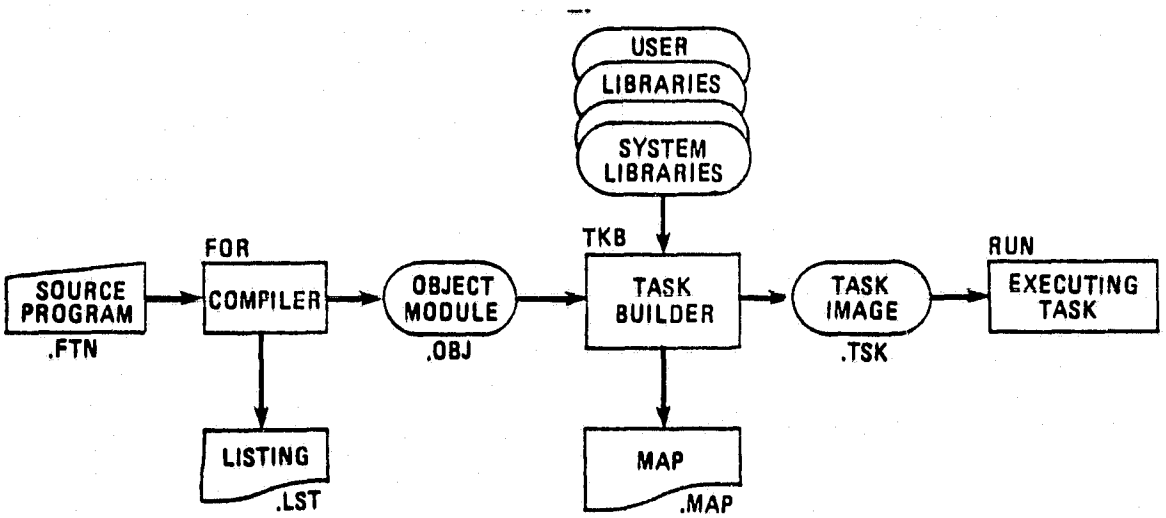


Figure 1

The Compiler produces relocatable object modules from FORTRAN Source programs. The general format of the command line to the FORTRAN Compiler is:

MCR>FOR OUTPUT FILES LIST=INPUT FILES LIST

A maximum of two output files can be specified: The Object Module File and the Listing File. Multiple input files may be specified. The default type for these is FORTRAN (.FTN). The Listing File can be omitted from the command line.

Some examples:

(1) Compile PROG1.FTN with Listing File:

MCR>FOR PROG1,PROG1=PROG1

Output: PROG1.OBJ
PROG1.LST

(2) Compile PROG1.FTN without Listing File, but produce listing on LP:

```
MCR>FOR PROG1,LP:=PROG1
```

(3) Compile PROG1.FTN without listing:

```
MCR>FOR PROG1=PROG1
```

Some useful compiler switches:

SWITCH	DESCRIPTION
/CK	Checks all array references to make sure they are within the array address bounds specified by the program.
/CO:N	Allows a maximum of N continuation lines in the program. Default is 10. Not to exceed 99.
/LI:N	Specifies listing options; 0<N<3 N=0 - Minimal Listing File: diagnostic messages and program section summary. N=1 - Source listing and program section summary. N=2 - Source listing, program section summary and storage map (DEFAULT). N=3 - Source listing, assembly code, program section summary and storage map.
/TR	Controls the amount of extra core included in the compiled output for use by the OTS (Object Time System) during error traceback.
/TR:BLOCKS	- Traceback information is compiled for subroutine and function entries and for selected source statements. The source statements are initial statements in sequences called blocks. (Refer to Page 4-6 of FORTRAN IV-Plus User's Guide for more information.)

3.4.2 Task Builder

The Task Builder is a system program that links relocatable object modules to create a task image. It is the last step before the source program becomes runable. The object modules can come from user specified input files, user libraries or system libraries. References to symbols defined in one module and referenced in other modules are resolved with the Task Builder. Any remaining unresolved symbols are searched for in the system object library:

```
DB0:[1,1]SYSLIB.OLB
```

The format of the command line to the task builder is similar to the compiler. For example:

```
MCR>TKB OUTFILE=INFILE
```

The first output file specifies the Task Image File (.TSK). A second file may be specified if a memory allocation map is desired. Lastly, a third file is the Symbol Definition File. The Memory Allocation File (.MAP) contains information about the size and location of components within the task. The Symbol Definition File (.STB) contains the global symbol definitions in the task and their virtual or relocatable addresses in a format suitable for re-processing by the Task Builder. The input files are combined to form a single executable task image.

Any number of input files may be specified. The Task Builder prompts for input until it receives the terminating sequence, "//". This instructs the task builder to stop accepting input, build the task, and return to the MCR level.

As with the FORTRAN Compiler, there are some switches which might be useful to FORTRAN programmers:

(1) /SH on the MAP file will produce an abbreviated form of the Memory Allocation MAP.

(2) /LB when specified with an input file, specifies that the file is a library of relocatable object modules.

(3) /MP on an input file specifies that the file is an overlay description file. It must be the only input file specified.

(4) /CR on memory allocation file produces a global cross reference.

The task builder has the additional feature of prompting for options after a "/" (Slash) is typed. Options are used to specify the characteristics of the task being built. Some commonly used options are:

(1) ASG: Logical unit numbers assigned to physical devices:

ASG=DEV1:N1:N2:... , DEV2:M1:M2:...

Defaults are:

ASG=DBØ:1:2:3:4, T1:5, LP:6

(2) COMMON: All system global common blocks referenced must be specified:

COMMON=NAME:ACCESS

Those using the Vector General Graphics package must include the following line in their TKB:

COMMON=VGCOM:RW:6

This allows READ/WRITE access to the display list.

(3) LIBR: All shared libraries referenced must have this option:

LIBR=NAME:ACCESS

All users should include the following line to reduce their Task Image File:

LIBR=SYSRES:RO

Since the Task Builder accepts indirect command files, this is the most convenient way to build your program. A file of type .CMD is created using the Editor. This is then submitted to the Task Builder:

MCR>TKB @PROG1

The Sample File PROG1.CMD:

```
PROG1=PROG1
/
LIBR=SYSRES:RO
COMMON=VGCOM:RW:6
ASG=MMØ:1,MM1:2
//
```

3.5 Indirect Files

An indirect file is a file containing a sequence of comman lines that can be interpreted by a single task such as a utility or the Task Builder. They are extremely useful when submitting the same sequence as a very similar sequence of commands to a task or utility.

The command string contained in the indirect file are executed when the indirect file is invoked. For example, to perform a series of PIP commands, an indirect file might look like:

```
DBØ:=MMØ:[11,14]*.FTN
LP:=*.FTN/LI
PROG.FTN;20/DE
```

To invoke such an indirect file, enter the command

```
MCR>PIP @PIPCMDS.CMD
```

PIP is then invoked and accesses the full PIP CMDS.CMD which contain the above sequence of commands. PIP executes the commands and returns control to MCR.

3.6 Executing a Program

```
MCR>RUN [50,50] PROG (esc key)
```

SAMPLE: A FORTRAN program called PROGR.FTN

```
MCR>For PROGR=PROGR          - Creates PROGR.OBJ
     (wait)
MCR>TKB @PROGR               - Creates PROGR.TSK
```

where PROGR.CMD contains:

```
PROGR=PROGR
/
LIBR=SYSRES:RO
ASG=TI:5,LP:6
//

     (wait)
MCR>RUN PROGR  (hit esc key) - Executes PROGR.TSK
```

3.7 MCR Commands, General Information

A. Default LUNs:

DBØ:	1-4	(refers to the system disk)
TI :	5	(Terminals)
LP :	6	(Line printer)

B. File Specifiers:

DEV:	= Physical device on which the volume containing the desired file is mounted.
[UFD]	= User File Directory containing the desired file.
FILENAME	= The name of the file. Up to 9 alphanumeric characters in length.
.TYPE	= File type, e.g. .FTN or .OBJ.
;VERSION	= An octal number used to differentiate analog versions of a file. Version numbers can range from 1 to 377.

Some Useful MCR Commands:

(1) Abort Command:

Function: Allows user to terminate the execution of tasks which have been initiated from that terminal.

Format: MCR>ABO TASKNAME

(2) Active Task List Command:

Function: Enables the terminal user to obtain a list of the tasks active within the system, along with status information on this particular task.

Format: MCR>ACT [TASKNAME] [SWITCH(s)]

Switches: /FU - Full listing. Task name must be specified.
/ALL - All active tasks listed.

(3) BYE Command:

Function: Allows user to log off system.

Format: MCR>BYE

(4) SYS Command.

Function: Allows user to see all the active tasks in the system.

Format: MCR>SYS /ATL

(5) Dismount Volume Command.

Function: Allows user to logically dismount a previously mounted volume.

Format: MCR>DMO DEV:[volume label]

(6) HELLO Command.

Function: Allows user to log onto a terminal.

Format: MCR>HEL [UIC]

(7) Logical Unit Numbers Command.

Function: Lists on the user's terminal the physical device units and corresponding logical unit numbers for an indicated task.

Format: MCR>LUN TASKNAME [,TASKNAME, . . .]

(8) Mount Command.

Function: Allows a user to make a selected volume visible to the system.

Format: MCR>MOU DEV:[volume label] [/Switches]

Switches: /CHA =[FOR]
Not an RSX-11D structured volume – foreign.
/DENS = 800 or 1600
Specifies density of tape.

(9) Password Command.

Function: Allows user to change or create a password for his UFD.

Format: MCR>PWD [UIC]
The following message is printed:

PASSWORD>

The user now types in his password; maximum of 6 characters

(10) QUE Command.

Function: Print queued files on LP.

Format: MCR>QUE FILENAME.TYPE

(11) Resume Command.

Function: Allows user to continue execution of a previously suspended task.

Format: MCR>RES TASKNAME

(12) Run Command.

Function: Allows user to initiate execution of a particular task.
Format: MCR>RUN FILENAME

(13) Terminal Status Command.

Function: Indicates which terminals are in use.
Format: MCR>WHO

3.8 File Dump Utility (DMP)

The File Dump Utility is very useful when an ASCII or octal dump of a file is needed. DMP runs in either one of two modes:

(1) File Mode:

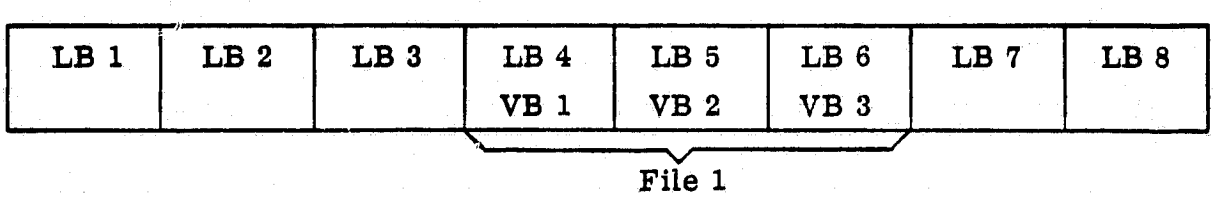
In File Mode, one input file is specified and all, or a specified range of Virtual Blocks of the named file are dumped. Virtual Blocks refers to the blocks of data in a file.

(2) Device Mode:

In Device Mode, only the device is specified and a specified range of Logical Blocks are dumped. Logical blocks refers to the actual 512-byte blocks on the disk.

To clarify the difference between Virtual and Logical blocks on a device, consider the following illustration:

LB = Logical Block VB = Virtual Block



The first block of file 1 begins in LB 4. With respect to File 1, that block is known as VB 1. Subsequent blocks of that file are known as VB 2 and VB 3. However, with respect to the device, those blocks that contain File 1 are known as LB 4, LB 5 and LB 6.

DMP can handle physical records up to 2048 byte in length.

DMP Switches:

SWITCH	DESCRIPTION
/AS	Data dumped in ASCII Mode.
/BL:N:M	Specifies the first (N) through the last (M) logical or Virtual Blocks to be dumped.

Notes:

(1) When the /BL:N:M switch is specified in File Mode, it specifies the range of Virtual blocks to be dumped.

(2) The /BL:N:M switch is required in Device Mode. When specified in this mode it is the Logical Blocks to be dumped.

/BY Specifies that the data should be dumped in byte octal format. The default is Word Mode Octal Format.

/HD Optional parameter to be used in File Mode. It causes the file header to be dumped.

Note: If just the file header portion is desired, the user can specify /HD/BL:0.

/DC Specifies that the data be dumped in decimal word format.

/RC Specifies that the data be dumped a record at a time.

/LB Logical block. This switch gives the user only the starting block number and a contiguous or noncontiguous indication for the file.

3.9 File Compare Utility (CMP)

This utility allows you to compare two ASCII source files. The comparison is done line-by-line to determine whether parallel records are identical. Some of the features are:

(1) Generate a listing showing the differences between the two files.

(2) Generate a listing in the form of one list with differences marked by change bars.

CMP also provides switches that allow you to control compare processing.

The format for specifying the CMP command line is:

outfile[/sw . . .]=infile1[/sw],infile2[/sw . . .]

where outfile represents the file specification for the output file, infile 1 represents the input file specification for the file to be compared to infile 2

and infile 2 represents the file specification for the input file to be compared to infile 1 and /sw represents the switches applied.

Switches

/BL	specifies that blank lines in both files be included in compare processing
/CB	Specifies that CMP list infile 2 with change bars in the form of exclamation marks (!) applied to each line that does not have a corresponding line in infile1.
/CO	Specifies that comments be included in compare processing
/DI	Specifies that CMP print the differences between the two files.
/FF	Specifies that records consisting of a single form-feed character be included in compare processing
/LI:n	Specifies that a number of lines (n) must be identical before CMP recognizes a match. Default is 3.

3.10 Overlays

3.10.1 Introduction

The maximum size program that a user can create on the 11/70 is 32K (words). However, there is an overlay capability to reduce the memory requirements of a task. In utilizing this the user would divide his task into a series of segments consisting of:

(1) A single root segment – always in memory.

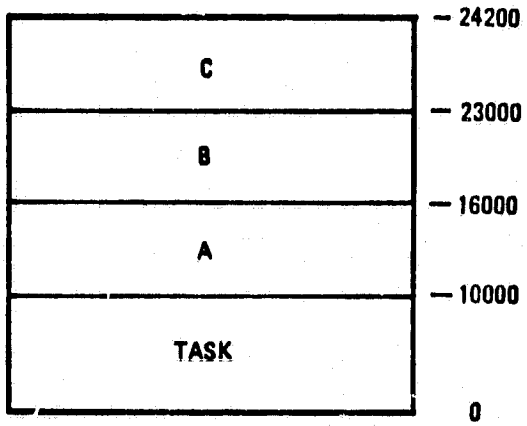
(2) Any number of overlay segments which share memory with one another.

However, care must be exercised in doing this, and not all programs are suitable for overlaying. All segments that overlay each other must be logically independent; i.e., none of the components of one segment can reference any of the components of the other segment. In addition to this consideration, is the general flow of control within the user task. A task that moves sequentially through a set of modules is well suited to the use of overlay structure. However, a program that jumps back and forth between modules and passes data between them would not.

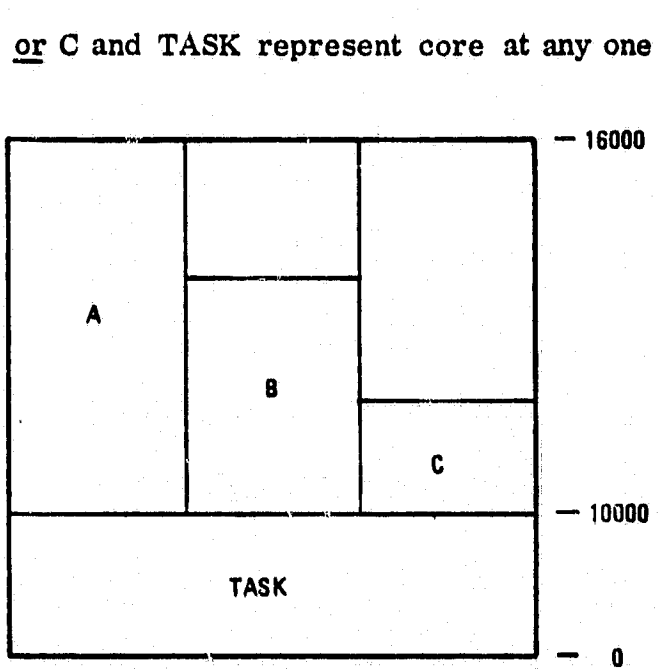
3.10.2 Structure

The amount of storage required for the task is determined by the length of the root segment plus the length of the longest overlay segment. This is illustrated below with a root segment called TASK and 3 modules A, B and C:

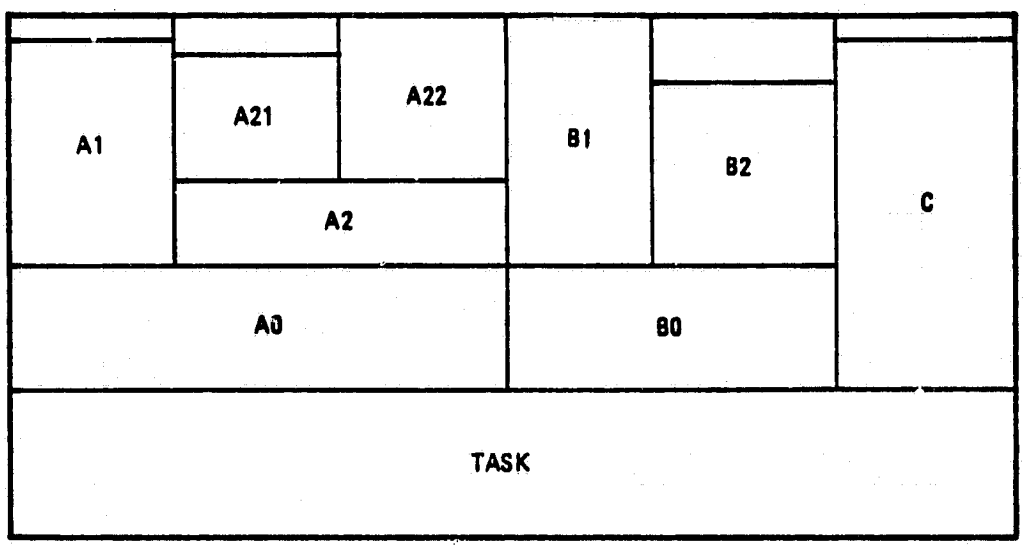
Without Overlays:



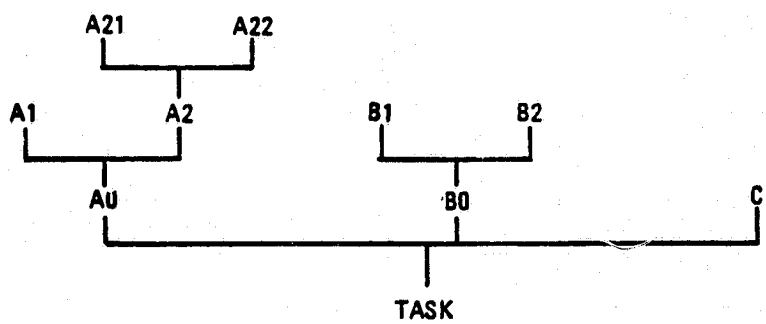
With Overlays. A, B, or C and TASK represent core at any one time:



If "A" could be further divided up into segments, then storage could be reduced even more. This is illustrated below. "A" was broken down into two independent modules, A1 and A2. A2 was further broken down into A21 and A22. "B" was divided into two independent modules, B1 and B2.



This scheme gives rise to the task builder language for representing an overlay. This structure can best be illustrated as a tree, as follows:



The tree has a root, Task, and three main branches, A0, B0 and C. It also has six leaves, A1, A21, A22, B1, B2 and C. The tree has as many paths as it has leaves. One path up may be defined as:

TASK-A0-A2-A22.

Understanding the tree and its paths is important to understanding the overlay loading mechanism and the resolution of global symbols.

Loading Mechanism:

Modules can call other modules that exist on the same path. TASK is common to all modules, so it can call and be called by every module. Module A2 can call A21 and A22, but A2 cannot call A1.

Resolution of Global Symbols in a Multi-segment Task:

Basically, the task builder performs the same activities in resolving global symbols for a multi-segment task as it does for a single-segment task. However, in a multi-segment task, a module can only reference a global symbol that is defined on a path that passes through the segment to which the module belongs. Two global symbols can be defined with the same name as long as the definitions are on separate paths.

Resolution of P-sections in a Multi-segment Task:

A program section, or P-section, is the basic unit of memory for the task. A FORTRAN source language program is translated into an object module consisting of P-sections. The object module produced by compiling a typical FORTRAN program consists of a P-section containing the code generated by the compiler, a P-section for each common block defined in the FORTRAN program, and a set of P-sections required by the object time system. These sections are divided up into local (LCL) or global (GBL).

Local P-sections with the same name can appear in any number of segments. Storage is allocated for each local P-section in the segment in which it is declared. However, when a global P-section is defined in several overlay segments along a common path, the Task Builder allocates all storage for the P-section in the overlay segment closest to the root. FORTRAN common blocks are translated into Global P-sections with the overlay attribute.

3.10.3 Overlay Description Language (ODL)

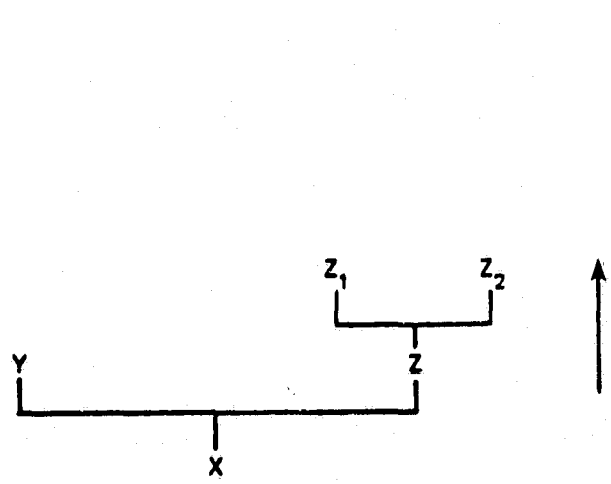
The Task Builder provides a language that allows the user to describe the Overlay Structure or Tree. It contains 5 directives. There must be only one .ROOT Directive and one .END Directive. .ROOT tells the Task Builder where to start building the Tree and the .END tells the Task Builder where the input ends.

The arguments of the .ROOT Directive make use of two operators to express concatenation and overlaying. A pair of parentheses delimits a group of segments that start at the same location in Memory. The maximum number of nested parentheses cannot exceed 32.

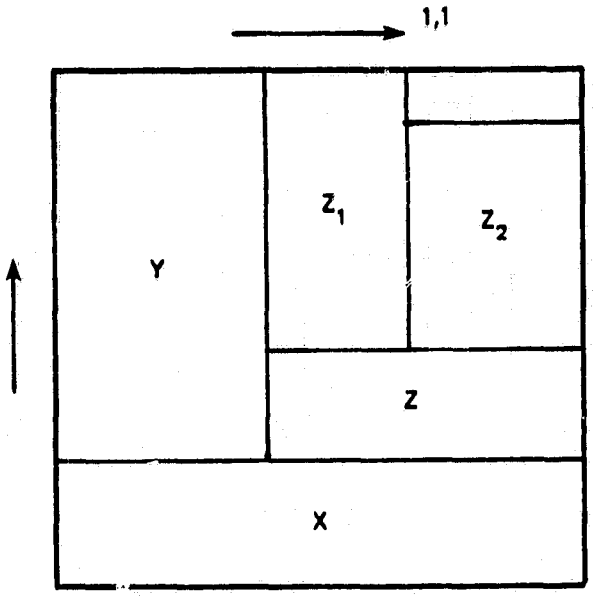
Operators:

(A) The "-" indicates the concatenation of storage or moving vertically in the Block Storage Diagram (see figure below).

(B) The "," appearing within parentheses indicates the overlaying of storage or moving horizontally in the Block Storage Diagram.



TREE



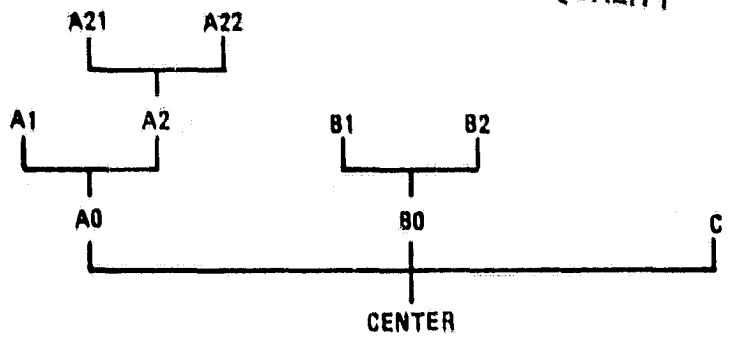
BLOCK STORAGE DIAGRAM

The following .ODL file would describe the above Tree:

```
.ROOT X-(Y,Z-(Z1,Z2))
.END
```

Starting with the inner parentheses, Z_1 and Z_2 appear horizontally (Z_1, Z_2) in the Block Storage Diagram and therefore share storage. Z appears vertically (Z-(Z_1,Z_2) in the Block Storage Diagram with respect to Z_1 and Z_2 and so indicates the concatenation of storage. Y and Z appear horizontally (Y,Z) and therefore share storage. X appears vertically with respect to Y and Z (X-(Y,Z)) and are concatenated.

Some overlay structures are complicated. The .FCTR directive allows the user to build large Trees and represent them systematically. Basically, it allows the user to extend the tree description beyond a single line. The following example illustrates its use:




```
        .ROOT   CENTER-(ACTR,BCTR,C)
ACTR:   .FCTR   A0-(A1,A2-(A21,A22))
BCTR:   .FCTR   B0-(B1,B2)
        .END
```

The decision to use the .FCTR directive is based on considerations of space, style and readability of a complex ODL file.

The .PSECT directive allows the placement of a Global P-section to be specified directly. The name of the P-section and its attributes are given in the .PSECT directive. Then the name can be used explicitly in the definition of the Tree to indicate the segment in which the P-section is to be allocated. The following example shows how the allocation of a global common block is forced into the Root Segment:

```
        .PSECT  DATA5, RW, GBL, REL, OVR
        .ROOT   CENTER-DATA5-(ACTR,BCTR,C)
ACTR:   .FCTR   A0-(A1,A2-(A21,A22))
BCTR:   .FCTR   B0-(B1,B2)
        .END
```

In addition to the overlaying capabilities mentioned above, the Task Builder allows the specification of more than one Tree within the overlay structure. Further information can be obtained by reading the "Task Builder Manual," Pages 5-10.

Finally, to build the task with an overlay structure the user types:

```
MCR>TKB FILENAME=FILENAME/MP
```

The switch MP tells the Task Builder that there is only one input file, FILENAME.ODL, and this file contains an overlay description for the task.

4. Magnetic Tape

4.1 Mounting and Dismounting Tapes

Looking at Figure 1, the empty reel is above the knob upon which your tape is mounted. The tape reel is pushed onto the lower knob, and the knob is turned clockwise to tighten. At this point, the four switches on the control panel on the lower left should be pressed to OFF LINE, STOP, between LOAD and BR REL (this is the BRAKE position) and FWD.

After tightening the tape reel, press BR REL to release the brake. Thread tape clockwise around reel, according to the illustration. It is wound clockwise around the empty reel, enough wound to tighten the tape so that turning the empty reel moves the full reel. The load point must be on the full reel side of the read head. Now press LOAD and wait until the LOAD light is on. Then press START and the tape will move to the load point as indicated by the LD PT light. Press ON LINE and the tape is mounted and ready.

To rewind to the load point or reverse, the ON LINE/OFF button must be at the OFF LINE position. The tape will automatically stop at the load point.

To dismount the tape, put drive OFF LINE and press REV. Then press the START button. When tape is off the top reel, press BR REL and finish winding tape by hand. Put on BRAKE position and turn knob counter-clockwise to loosen the reel. It can now be removed.

TAPE DRIVE

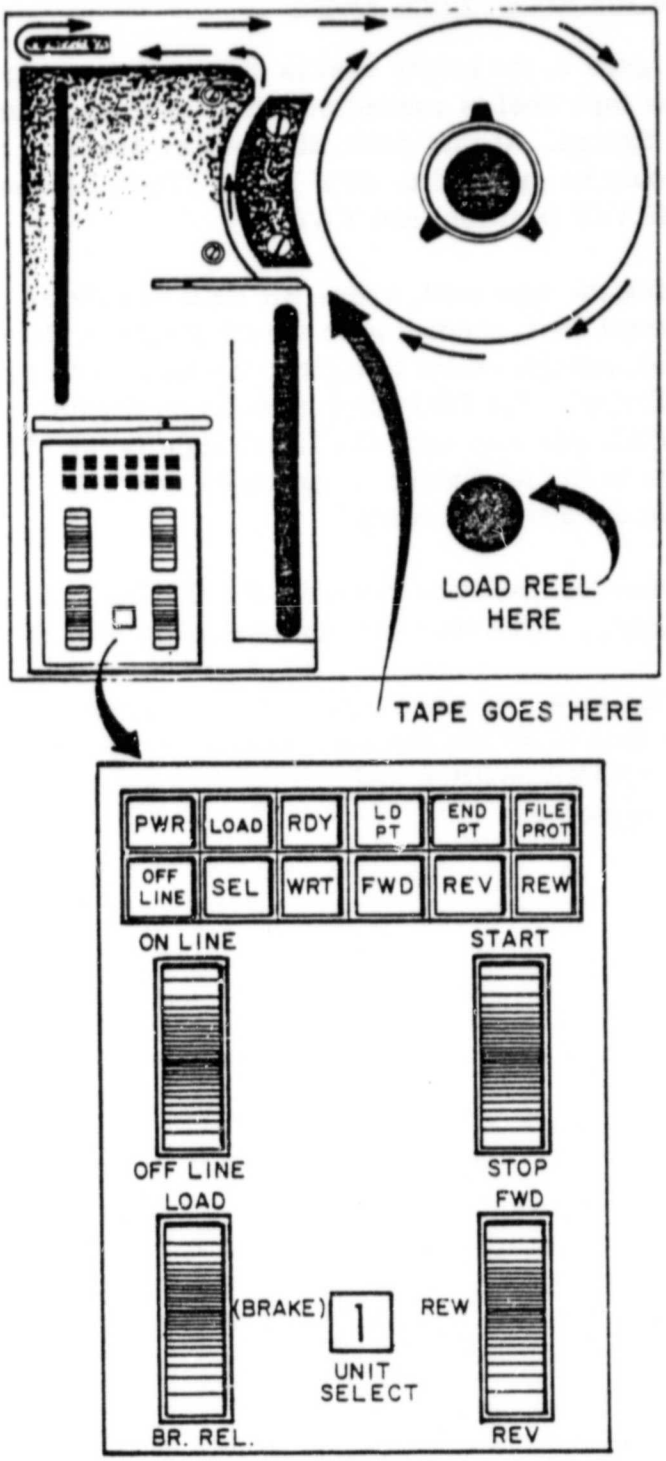


Figure 1

5. Paper Tape

5.1 Loading, Hardware

The paper tape assembly consists of one punch, and one reader. Both may be used simultaneously if desired.

If the paper tape is correctly loaded, tape from the punch comes out folded, with arrows showing direction of the beginning of the tape.

The paper tape reader must be loaded with the arrows pointing right to left, and the feed sprocket firmly in the row of feed holes along the tape. The knob above the light is turned clockwise to raise the foot, which keeps the tape aligned. When tape is loaded, the foot is lowered by turning the knob counter-clockwise. Make sure the control tape is under the plate at the scan light.

5.2 Paper Tape Reader

If it is desired to read data into an array during a FORTRAN program from the paper tape reader, subroutine [2,75]PTREAD must be used.

Call PTREAD(LUN,A,LEN,IOST). "LUN is the logical unit number assigned to the reader, "A" is the array to receive the data, "LEN" is the number of bytes to be read in, and "IOST" is the returned error code.

It is the user's responsibility to make sure "LEN" does not exceed the maximum dimension of "A." Otherwise, a data overrun in core will result.

When positioning the paper tape on the reader, keep in mind that it always reads first and moves second. The first byte to be read should be in line with the light detector. However, the user can do his own checking for null characters to pass through the leader (NULL = 0).

READS A SOURCE TAPE INTO A FILE 'TEMP'

```
LOGICAL*1 A(8000),B(2000)                ! 8000 BLOCKS
CALL PTREAD(3,A,8000,IOST)               ! READ TAPE
K=1
DO 30 I=1,8000                           ! WEED NULLS
IF(A(I).EQ.0)GO TO 30
B(K)=A(I)
K=K+1
```

```
 30       CONTINUE
          LEN=K-1
          CALL ASSIGN(4,'TEMP',4)              ! READ TO 'TEMP.'
          WRITE (4,200)(B(K),K=5,LEN)          ! 1-4 ARE LEAD-IN CHARS.
100       FORMAT(I5)                           ! FROM PUNCH
200       FORMAT'(104A1)
          STOP
          END
```

5.3 Paper Tape Punch

The punch may be called from PIP or via a FORTRAN routine called **PTPNCH.**

FROM PIP: MCR>PIP PP:=FILENAME

FROM FORTRAN: CALL PTPNCH(LUN,FILENAME,LEN)

LUN = logical unit device of punch.
FILENAME = hollerith string or character array. The character array must be LOGICAL*1.
LEN = length of hollerith string or character array.

NOTE: Using PIP to punch a tape, four control characters are punched at the beginning of the tape, preceeding the data.

6. Floppy Disks

6.1 Loading, Hardware

The Floppy disk drives are located to the right of the TU16 Magtape drives. Dust covers have been placed over the slots to keep the read/write heads dust free. To load a floppy in the drive, lift the dust cover, slide the floppy disk into the slot with the small hole and the elongated opening toward the drive. The small square tag should be on the right top as you insert the floppy. See Figure 1.

6.2 Storing Data on the Floppy

Floppy disks are used as auxiliary storage on the PDP 11/70. Their advantage is that they are faster than magtape and provide quick and easy access to data. Floppy disks can only be used as Files-11 devices. Thus to access them they must be initialized as Files-11 (Run [2,75] INITIAL) and then PIP'ed onto or written to with Fortran I/O statements. For example, to prepare and write data into a Floppy:

```
MCR > RUN [2,75] INITIAL$

      (INITIAL completes)

MCR > MOU DXO: volume name (specified in INITIAL)

MCR > PIP DXO [UIC] = [UIC] DATA.DAT,DATA.FTN
```

The maximum capacity of each floppy is 470. blocks.

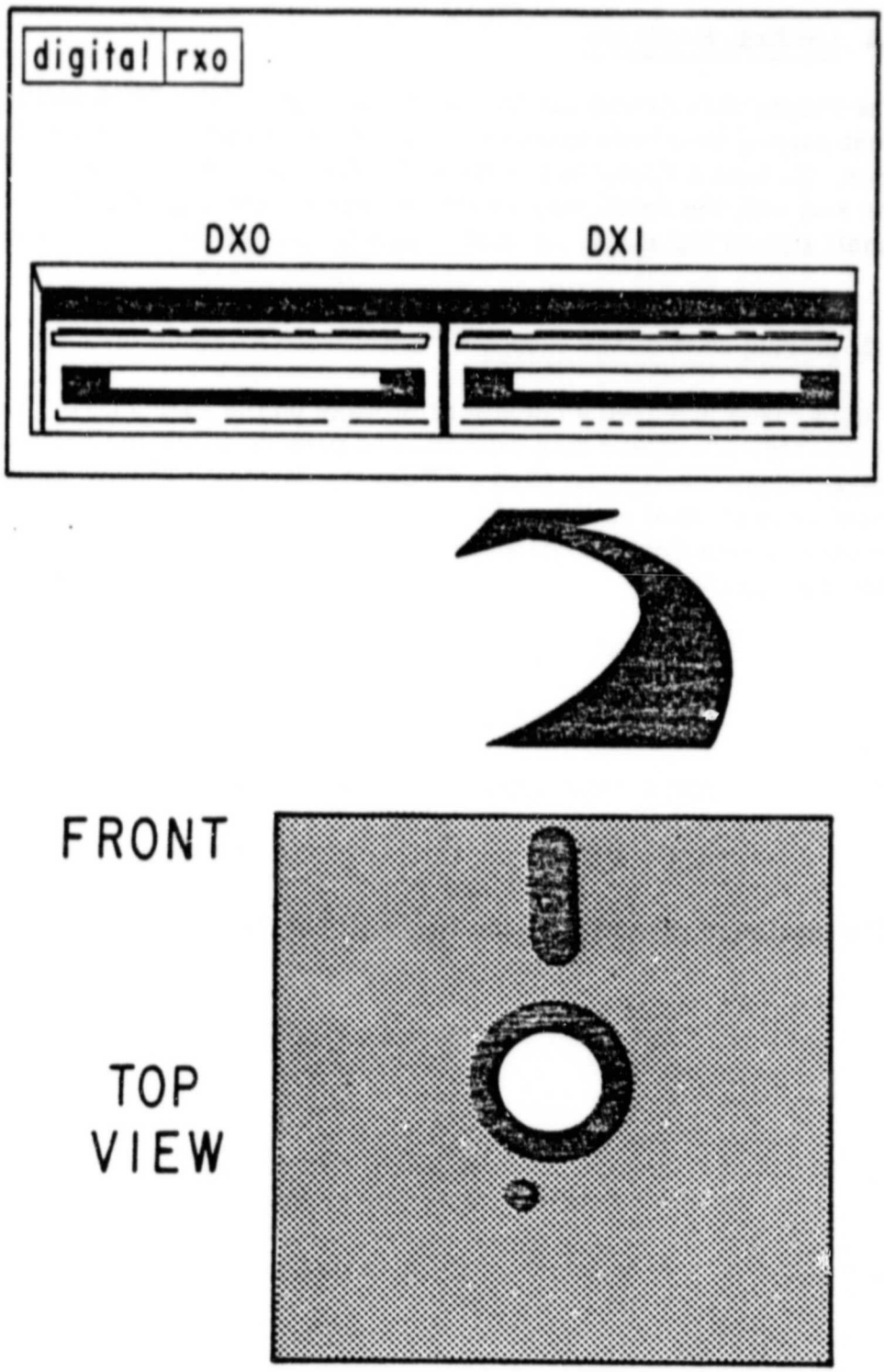


Figure 1. Disk Guide

7. The Vector General Graphics Display Unit

7.1 Hardware

The Vector General consists of a screen, alpha-numeric keyboard, function keyboard, and a light pen. The intensity, focus and scale can be changed by the corresponding knobs on the right-hand side of the VG unit. Do not adjust the threshold knob. Detailed description of the hardware features may be found in Section 1.3.2.

7.2 Programming the Vector General

7.2.1 Introduction

The programming for the Vector General is done by calling special VG subroutines from within a FORTRAN program. These subroutines perform all the graphics work. There are four basic steps to graphics programming:

(1) CALL GINIT – readies the VG screen for use.

(2) Graphics Subroutines – perform the drawings.

(3) CALL GRUN – displays the drawings.

(4) CALL GTERM – ends the VG screen use.

The following pages describe in detail each of the subroutines in the graphics package. Programming ideology is presented first, followed by each subroutine. A sample program and task build sample then follow, which the reader is advised to study. Also included is a list of typical mistakes made by VG programmers which cause programs to run incorrectly or to not run at all, and some hints for greater efficiency.

Interaction with the Vector General is accomplished with three hardware devices as follows:

1. Function keyboard – The routine RQATN is called to suspend the execution of the program and activate certain buttons as defined in the arguments. When one of these buttons is depressed, control is returned to the user's program with information concerning the light penned portion of the display in the RQATN parameter list. (see RQATN)

2. Light pen – The routine RQATN is called to suspend the execution of the program until the light pen is pressed against the screen, pointed at a display section which had been created as a light-pennable element. Control is returned to the user's program with information concerning the light penned portion of the display in the RQATN parameter list. (see RQATN)

3. Alpha-numeric keyboard – In using this device, several steps must be taken in the fortran code.

a. Call to TEXT to create a character string with a key. (see TEXT)
b. Call to ICURS to display blinking cursor. (see ICURS)
c. Call to RQATN to suspend program while user types in characters from keyboard. (see RQATN)
d. Call to RDCHR to read the character string into an array.

Notes and Conventions:

1. "ACTIVE" element – an element whose associated pictures are currently displayed.

2. "INACTIVE" element – an element which has been created but which has been inactivated from display with OMIT or GHALT.

3. The "beam" – the hardware component which draws every display. It remains positioned where the last display of that element was drawn.

4. A "negative" intensity means an intensity value beneath 0, which is the average brightness. Thus an intensity value of -16 is actually a near-zero intensity level.

5. All arguments to the routines which are enclosed in brackets [ ] are optional. If the value is omitted, commas must mark its place if further arguments are to be specified.

6. byte – LOGICAL*1, one byte
   I*2 – INTEGER*2, 2 byte integer
   I*4 – INTEGER*4, 4 byte integer
   R*4 – REAL*4, 4 byte real number
   R*8 – REAL*8, 8 byte real number

7. "Display list" – The list of all Vector General display instructions in the running program. It consists of integer numbers arranged in order of element number which the Vector General handler decodes to generate all displays. It is stored in the 4K common global area called VGCOM, as specified in the task build.

Each picture to be displayed consists of a set of logically connected primitives called elements. Each element is identified by a unique number ranging from 1 to 249 each of which causes the reloading of all relevant hardware registers (e.g., intensity, X, Y, delta X, delta Y, light pen sensitivity, picture scale and coordinate scale. Elements are created by calling INIT, SINIT or COPY. These are then updated by referencing the element number in the other subroutine calls.

The Vector General Graphics Subroutine package consists of a set of FORTRAN-callable subroutines which build display instructions and pass control information to the device handler. The device handler builds the display file and controls the refresh rate. It also services interrupts and passes information back to the user. Communication between the user package and the device handler is accomplished through the use of a 4K global common area named VGCOM and global event Flags 41 and 42.

The user package places in common area VGCOM control information followed by display instructions. When this is completed, Event Flag 55 is set which causes the handler to process the stored information. The user package then waits on Event Flag 56 which the handler sets on completion.

7.2.2 Programming Ideology

The programming ideology operates with the concept of "elements." An "element" is a list of characteristics such as the intensity level, or whether or not to blink. It is an intangible set of parameters by which an entire display will be drawn.

Each element is created by INIT, SINIT, or COPY routines with an element number, and the consequent display takes on all the characteristics as defined by the element. Once an element has been defined by INIT, SINIT, or COPY, any number of displays associated with the element may be created. Nothing may be drawn without previously defining the element number. A maximum of 249 such parameter sets may be defined, Element Numbers 1 to 249.

7.2.3 VG Internal Ideology

The Vector General Graphics Display Unit consists of a Vector General 2D CRT equipped with an alphanumeric keyboard, 32 key program function keyboard and a light sensitive pen. The CRT screen is 12 inches by 12 inches addressable from -2048 to +2047 in both the X and Y axes.

The display file is built in a 12K shared global common area named VGCOM. The first word is used for communications with the hardcopy task. VGCOMM is a short MACRO-11 routine that brings the shared global common into core when the handler is loaded and removes it when the handler terminates. Additionally, VGCOMM acts as a monitor for running hardcopy routines when called by the handler. The active display file contains contiguous display instructions. When an instruction or instructions are added to an element, all elements after it are relocated the appropriate number of words and the new instructions are added to the end of the element. All updates to the display file are performed between refresh cycles. No refreshing is allowed until the updates are completed.

7.2.4 The Vector General Graphics Routines

A. Initiating and Terminating Routines

(1) CALL GINIT:

This routine initializes the display areas and resets all tables. Processing is stopped and HALT mode is entered. This routine must be the first graphics routine called. It may be called to create other displays by reinitializing, but only after GTERM has been called.

(2) CALL GTERM:

This routine terminates the use of the CRT and all elements associated with it. It should be called when use of the CRT is no longer needed. If it is not called, the user program will terminate, yet the active display will continue to appear on the screen. Depress the "SPEC" and "FS" keys simultaneously to clear the display if a task ends without calling GTERM.

(3) CALL GRUN:

This routine causes the CRT to display the current active elements. It must be called to start the display since GINIT puts the display into a HALT mode. GRUN need be called only once after GINIT or GHALT has been called. CAUTION: There must be at least one active element before GRUN may be called.

(4) CALL GHALT:

This routine puts the display into a HALT state and the CRT will no longer display the active elements. It is useful when major changes are being made to the display since the display must be halted in order to perform updates.

The following arguments are used commonly by many of the subroutines:

ELEM - The name of the element referred to is in the subroutine call. I*2 Range is 1 to 249.

LP - Light pen sensitivity option. 0= not light pen sensitive, 1= light pen sensitive. I*2.

INTENS - The value of the intensity register for a particular element (range is -16 to +15). I*2. Default is 0, which is medium intensity.

BLINK - If value is a 1, the entire element will blink. If value is a 0, the element will not blink. I*2. Default is to not blink.

PS - Picture scale value for the element ranging from -2048 to 2047. I*2. Default is 2047. See section 7.2.5, note #6.

CS - Coordinate scale value for the element ranging from -2048 to 2047. I*2. Default is 2047. See section 7.2.5, note #6.

IDISPL - I*2 variable returned. This is the displacement into the display list of instructions for the element. If a light penable element is used, RQATN subroutine can indicate which element is light penned. See section 7.2.5, note #3 for formulae for each subroutine containing IDISPL.

B. Element-creating Routines

(1) Call INIT(ELEM,[LP],[INTENS],[BLINK],[PS],
[CS],[X],[Y],[DX],[DY])

The INIT subroutine causes the creation of an element and sets all the characteristics of that element:

X - I*2 initial horizontal coordinate of the beam.
Y - I*2 initial vertical coordinate of the beam.
DX - I*2, initial increment value. See Note 2.
DY - I*2, initial increment value. See Note 2.

(2) Call SINIT(ELEM,[LP],[INTENS],[BLINK],
MODE,X1,Y1,X2,Y2,[LX],[LY],[UX],[UY])

The SINIT routine is very similar to the INIT routine except that it uses the defaults for PS, CS, X, Y, DX, DY (which can be changed by use of CHNGE

routine). The SINIT routine defines the size and position of a particular element with respect to the boundaries on the screen.

The screen is the total space which may be addressed. SINIT allows the user to create a rectangular area which is a subset of the total screen.

SINIT also allows the user to define the data type of the X and Y positions and to scale them to appear on the screen.

MODE: Data type of all X and Y coordinates for this particular element –

1 = byte
2 = I*2
3 = I*4
4 = Real *4
5 = Real *8

X1, Y1 constants or variables of the type indicated by the mode for the X and Y coordinates of the lower left corner of the element. These coordinate values must be less than the coordinate values used to represent the upper right corner of the element. They may be any number within proper bounds of their mode. If mode 1, 3, or 5 was specified, then a variable of proper type must be used.

X2, Y2 constants or variables of the type indicated by the mode for the X and Y coordinate of the upper right corner of the element. These coordinate values must be greater than the coordinate values used to represent the lower left corner of the element. They may be any number within proper bounds of their mode. If mode 1, 3, or 5 was specified, then a variable of proper type must be used.

LX, LY I*2 constants or variables representing the X and Y coordinates that correspond to the lower left corner of the screen. Default -2048, -2048.

UX, UY are I*2 constants or variables representing the X and Y coordinates that correspond to the upper right corner of the screen (must be greater than the lower left coordinate). Default 2047, 2047.

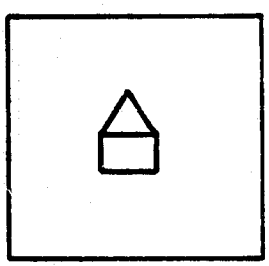
(3) CALL COPY(ELEM 2,ELEM 1,[LP],[INTENS],
[BLINK],[PS],[CS],[X],[Y],[DX],[DY])

COPY is used to create a new element by duplicating an existing one. The arguments are similar to the INIT subroutine. If an argument is specified, it's value replaces that of the original element.

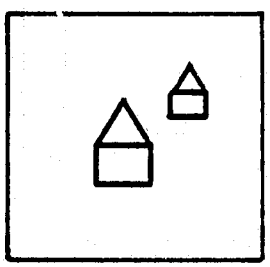
ELEM 2 = I*2, the element # being created.
ELEM 1 = I*2, element # being copied.

EXAMPLE:

Suppose Element 1 is active and it creates:



The CALL COPY (2,1,,,,,1024,,,1000,1000) is made, resulting in:



NOTE: COPY also copies all scaling information. When changing the new element, make sure proper arguments are used.

C. Graphics Drawing Routines

Index: Below is a table of routines possible to achieve the desired display or effect. Groups in ( ) must be used together

Display or Effect	Possible Routines to Call
1 or more characters	TEXT
1 or more line segments	PLINE, PLOT, (SETVM, VECT, VECTT)
tracking cross	PENTRK
read from screen	(TEXT,ICURS)
Move an image on screen	CHNGE, changing DX and/or DY (RESET, then draw at new coords)
Remove image from screen	OMIT, DELMT, RESET
position the beam	POSN, INIT with X and Y, CHNGE with X and Y
to light pen a display or piece of a display	(INIT, SINIT, CHNGE or COPY so that element is light-pennable, RQATN with Attention Code 34)

(1) CALL CHNGE(ELEM,[LP],[INTENS],[BLINK],
[PS],[CS],[X],[Y],[DX],[DY])

Subroutine CHNGE is used to modify the attributes of an element without affecting the contents of the element. Arguments are like INIT.

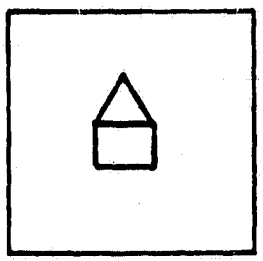
If an argument is omitted, its value is not changed.

EXAMPLE:

Suppose an element is created by the instruction

CALL INIT (1).

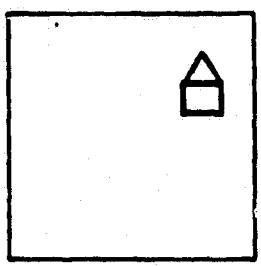
Then, other calls are used to create an image in Element 1 so that there is a picture on the screen, centered at (0,0), scale defaults to 2048.



Then the instruction is executed:

CALL CHNGE (1,,,,, 1024,,,1000,1000).

This changes the CS to 1024 (half scale) and the DX, DY to 1000,1000. The resulting picture is:



It's 1/2 the size and centered at (1000,1000) rather than (0,0). Thus, the picture created full-scale and centered, was moved and shrunk in any manner. Subsequent calls to change DX and DY actually move the picture to the new coordinates.

(2) CALL DELMT(ELEM)

This routine erases an element and all images generated by it from the screen. The element # is no longer in use, and may be recreated by the use of INIT, SINIT or COPY.

An element which has an active cursor on it (by use of the ICURS command) cannot be deleted until the cursor has been removed (RCURS).

(3) CALL ICURS(Key 1 [,Key 2] [,Key 3)] ...)

The ICURS routine is used to insert a CURSOR into a text image on the screen. The cursor is a symbol represented on the screen as a blinking dash beneath a character. It marks the position on the screen at which the next character typed on the keyboard will appear. The FS key will space the cursor to the next character in the text string, and the BS key will

move the cursor to the previous character in the text string. The FS key has no effect when the cursor is under the last character in the text string and similarly for the BS key when at the start of the text string. The → key is used to move from one text string to another.

Note: Only one cursor may appear on the screen at any given time.

The → key is used to move from one text string to another. Calling Arguments:

Key # – I*2 variable or constant. This is the number which was associsted with a text string in a previous call to text (see Call TEXT Description).

EXAMPLE:

```
LOGICAL*1 TEST(5)
DATA TEST/'T,' 'E,' 'S,' 'T',Ø/
CALL INIT(1)
CALL TEXT(1,TEST,-1000,0,2,5)
CALL TEXT(2, 'THIS',1000,-100,2,7)
CALL ICURS(5,7)
```

The first TEXT command causes a key of 5 to be associated with the text array TEST. The second command causes a key of 7 to be associated with the text string THIS.

A cursor now appears under the first T of 'TEST'. If the user presses the → key, the cursor will jump to the T of THIS, and if → is pressed again it will jump back to the first T of TEST. The RDCHR subroutine is used to read the text from the screen.

(4) CALL INCLD(ELEM)

Activates an element which had been inactivated by a call to OMIT.

(5) CALL OMIT(ELEM)

Removes the element and its associated images from the screen. However, the element is not lost nor is it changed. It is inactive. The INCLD subroutine will allow the element to be active again.

(6) CALL PENTRK(ELEM,XRET,YRET,[X,Y])

The PENTRK subroutine is used to display a light pen tracking cross which can be moved around the screen by use of the light pen. When the PENTRK routine is invoked, a tracking cross will appear on the screen and function keys 0 and 31 will be lit. Function Key 0 is used to change the pen tracking speed from fast mode to slow mode and back again. Function Key 31 is used to indicate the end of light pen tracking and the position of the tracking cross will be returned to the user.

ELEM - Element number to get scaling formulae from. If no scaling is desired, set ELEM to zero

XRET, YRET - Variables must be the same type as the element mode. It is returned to the user the X, Y coordinate of the tracking cross when Function Key 31 was pressed.

X,Y - Variables or constants of same type as element mode or Integer*2 if elem is set to zero, the initial position of the tracking cross.

NOTE: PENTRK sets the PS and CS to full scale. If the element that is being used for scaling does not have the PS and CS also set to full scale, there will be a discrepancy between the X,Y returned from PENTRK and the actual X,Y of that element.

(7) CALL PLINE(ELEM,TYPE,X,Y,[IDISPL])

Subroutine PLINE creates a visual vector on the screen from the current X, Y position of the beam to the X, Y value specified in the calling argument.

TYPE - see reference to Subroutine PLOT.

X, Y - coordinates of the end points of the vectors in the same mode as the element.

(8) CALL PLOT(ELEM,TYPE,XARRAY,YARRAY,COUNT,[IDISPL])

PLOT creates a graphic image that displays one or more connected line segments.

TYPE - Type of line (I*2)
0 = solid
1 = dashed
2 = dotted
3 = end points
4 = dash-dot-dash

XARRAY, YARRAY - Array name in proper mode for this element. These X and Y arrays contain the coordinates of the points that will be connected to form the image.

COUNT - I*2 constant or variable which is the number of line segments produced by this call plus one. This is less than or equal to the dimension of XARRAY and YARRAY.

(9) CALL POSN(ELEM,X,Y,[IDISPL])

This subroutine POSN is used to position the beam to a particular location on the screen. It is used primarily with the PLINE routine to generate line segments from one point to another.

X,Y - X,Y coordinates in proper mode for the element. The position on the screen the beam will be moved to. This move will not generate any visual image on the screen.

(10) CALL RCURS

This subroutine removes the cursor from the screen.

(11) CALL RDCHR(KEY,TEXTARRAY,NUM)

The RDCHR subroutine causes the transfer of a text string from the display image buffer into an array. The data is placed in the array at 2 characters per word.

The system functions ENCODE and DECODE are used profusely with this routine. See Section 6.2.5 for detail.

KEY - I*2. This is the key of the text string to read.

TEXT ARRAY - Name of an array whose dimension is greater than or equal to the number of characters (bytes) to be read. The text string is returned in this array.

NUM - I*2 variable. The number of characters in the text string is returned in this variable.

NOTE: The PDP-11 FORTRAN has two functions, ENCODE and DECODE, to change data to and from alphanumeric formats. See additional notes at end of chapter for detailed examples.

(12) CALL RESET(ELEM,[IDISPL])

The RESET subroutine removes displays created with an element. If reset is called with only 1 argument, the element will contain no IDISPL displayable images; that is, it will be identical to a call INIT or SINIT. IF IDISPL was used to create a part of the display, that display part and all displays in that element created after it will be deleted. See Section 6.2.5, Note #3 for discussion of IDISPL.

(13) CALL RKEY(KEY 1,[KEY 2]. . .)

This subroutine is used to disassociate a text string from a key. The reason for this routine is to help conserve keys, since the user has a limit of 20 text string keys that may be used at any one time. The key numbers in the argument list need not be in order, nor may the number of keys exceed 20.

KEY 1, KEY 2, etc. - I*2 variables or constants which were associated with text strings in TEXT commands.

NOTE: A key cannot be removed if it has been called in an ICURS routine and an RCURS has not been issued. In this case, the key is considered active and will remain associated with the text string.

(14) CALL RQATN(ICODE,IARRAY,ATTNCODE,[ATTNCODE. . .])

The RQATN subroutine allows the user to wait for a response from the graphics operator. When the RQATN routine is called, the subroutine will wait until one of the specified attention devices is activated and will return the device number.

ICODE - I*2 variable which is returned to the user to indicate the device number which caused the wait to be terminated.

IARRAY - I*2 array name dimensioned to 20. Upon return from the RQATN subroutine, this array will contain the following information:

IARRAY SUBSCRIPT	CONTENTS
1	Element number of the light pen hit.
2	Offset of instruction from start of element when light pen interrupted, used with IDISPL.
3	X location in screen coordinates of light pen hit.
4	Y location in screen coordinates of light pen hit.
5	PS of element light pen hit was detected on.
6	CS of element light pen hit was detected on.
7	DX of element at light pen hit
8	DY of element at light pen hit.
9	Program Interrupt Register at time of light pen hit.
10	Mode Control Register at time of light pen hit.
11	Last character entered on keyboard.
12	Which of Function Keys 0-15 was depressed at time of return. Bit 15 (most significant) is set if Function Key 0 was depressed, bit 14 is set if Function Key 1 was depressed, etc.

IARRAY SUBSCRIPT	CONTENTS
13	Which of Function Keys 16-31 was depressed at time of return. But 15 (most significant) is set if Function Key was depressed, bit 14 is set if Function Key 17 was depressed, etc.
14	Character which the light pen was pointed at when light pen interrupted.
15	Reserved.
16	Reserved.
17, 18	X Coordinate of light pen hit in scaled mode of the element. If Real*4, Real*8, or I*4, both 17 and 18 are used. If I*2, only 18 is used.
19, 20	Y coordinate of light pen hit in scaled mode of the element. If Real*4, Real*8, or I*4, both 19 and 20 are used. If I*2, only 20 is used.

NOTE: See Additional Notes, Section 7.2.5 #3 for detail on retrieving scaled coordinates and using IDISPL.

ATTENCODE - I*2 variable or constant. Code of attention device to wait for.

1 = Function Key 0

2 = Function Key 1

.

.

.

32 = Function Key 31

33 = End of sequence character (usually carriage return)

34 = Light Pen

35 = Clock

When ATTENCODE is 35, the next argument is the number of 8.3 millisecond clock ticks to wait. This next argument is mandatory.

EXAMPLE 1:

To call a halt in the program until function key 5 is pressed:

```
CALL RQATN(ICODE,IARRAY,6)
```

EXAMPLE 2:

To halt until either a display is light-penned or function key 8 is pressed, whichever occurs first:

```
CALL RQATN(ICODE,IARRAY,9,34)
```

EXAMPLE 3:

To halt for 200 clock ticks or until function key 23 is pressed whichever occurs first:

```
CALL RQATN(ICODE,IARRAY,24,35,200)
```

EXAMPLE 4:

To specify the range of codes 5, 6, 7, 8, 9 and 10, use the arguments 5, -10. For example, to wait for Function Keys 1, 3, 5, 6, 7, 8, 9, 10, 17, 18, 19, 20, and 1200 clock ticks use the following call:

```
CALL RQATN(ICDDEA,IRRAY,2,4,6,-11,18,-21,35,1200)
```

(15) CALL SETVM(ELEM,TYPE,VECT MODE,[X],[Y],[INCR],[M])

This subroutine is used to set the vector type for the VECT routine. Detailed description of VECT MODE is in additional notes in Section 6.2.5 #4.

TYPE - Reference subroutine PLOT.

VECT MODE - I*2 vector mode type:
- 0 = relative
- 1 = relative auto X
- 2 = relative auto Y
- 4 = absolute
- 5 = absolute auto X
- 6 = absolute auto Y
- 8 = relative compact
- 9 = relative compact auto X
- 10 = relative compact auto Y

X, Y - Starting position (for relative types) of the image. Must be in proper mode for this element.

INCR - Increment for auto X or auto Y. This is the increment that the appropriate coordinate will be stepped by. Must be in the proper mode for the element.

M - Used for relative compact mode. If M = 0, no scale mode; if 1, scaled mode.

NOTE: See Additional Notes, Section 6.2.5 #4 for application.

(16) CALL STEOS(ICHAR)

This routine is used to set the end of order sequence character (the character which when pressed will cause the RQATN routine to return with a code of 33). The default is the carriage return character (CR).

ICHAR - I*2 number or variable set to the ASCII code of the character which is to be the end of order sequence (EOS) character. If all characters are to be EOS characters, set ICHAR to a -1.

EXAMPLE: To change the EOS character to an ASCII code of 101 octal ('A'):

```
Call STEOS(65)
```

(17) CALL TEXT(ELEM#,TEXT,X,Y,[SIZE],[KEY],[IDISPL])

The TEXT subroutine creates text images in the element. The X and Y values determine where on the screen the text will be started. The first character is centered at (X,Y). Characters produced may be of 4 sizes and of two orientations, vertical and horizontal.

The system functions ENCODE and DECODE are very often used with this routine. See Section 6.2.5 #1 for detail.

TEXT - is a variable, array name or hollerith string of the characters to be displayed. The characters in the string text must be packed 1 character per byte. The text is terminated when a byte of zero is encountered or a byte of Octal 024. See Section 6.2.5 #5 for a chart of possible characters to draw.

X, Y - are variables or constants in the proper mode for this element representing the X and Y coordinates of the center of the first character.

SIZE - I*2. Is a variable defining the size and orientation of the text. Values:

1 = smallest character, horizontal orientation.
-1 = smallest character, vertical orientation.
2 = 1.5* smallest character, horizontal.
-2 = 1.5* smallest character, vertical.
3 = 2* smallest character, horizontal.
-3 = 2* smallest character, vertical.
4 = 4* smallest character, horizontal.
-4 = 4* smallest character, vertical.

Default is size 3.

CHAR SIZE	COL/ LIN	LINES/ PAGE	COL/ CHAR	LINES/ CHAR
1	120	60	34	68
2	81	41	50	100
3	60	30	68	136
4	32	16	128	256

KEY - I*2 variable or constant representing a number associated with this string of text for cursor control. Key must range from 1 to 250 and must be unique (no 2 texts can have the same key). See Section 6.2.5 #3 for discussion of IDISPL.

(18) CALL VECT(V1,[V2],[OPERATION])

V1, V2 - V1, V2 take on different meanings, depending on the vector mode specified in SETVM.

If the mode was 0, 4 or 8 (relative, absolute, relative compact), V1 and V2 are the X and Y coordinates in the appropriate scaling. In the relative compact form V1, V2 are converted to screen coordinates and treated as described in the discussion.

If the mode was Auto X (1, 5 or 9), V1 is the Y coordinate value in the proper mode for element. V2 is optional and if used will cause the increment to be changed. It also must be in the proper mode for the element specified in the SETVM call.

If the mode was Auto Y (2, 6 or 10), V1 is the X coordinate and V2 changes the increment. Identical to above.

If mode is 9 or 10, V1 is DX, V2 is DY, both are needed. If omitted they will default to zero.

OPERATION - I*2, if OPERATION is a 1 (default) a draw command will be issued to draw a vector from the previous X, Y to the current X, Y. If operation is a 2, the beam will be moved to the X, Y value, but no draw will be performed.

NOTE: See Additional Notes in Section 7.2.5, #4 for application.

(19) CALL VECTT(IDISPL)

This subroutine is used to terminate the SETVM instruction and place the created image in the appropriate element.

NOTE: See Additional Notes in Section 7.2.5, #4 for application.

7.2.5 SAMPLE PROGRAM ON THE VECTOR GENERAL

```
DIMENSION IR(20),IX(5),IY(5)          !ARRAYS FOR RQATN,PLOT
DATA IX/-100,-100,100,100,-100/
DATA IY/-100,100,100,-100,-100/
CALL GINIT                            ! INITIALIZE VG SCREEN
CALL INIT(1)                          ! CREATE ELEMENT #1.
C
C    I WILL WRITE "EXAMPLE" AT THE
C    MIDDLE LEFT OF THE SCREEN OF CHARACTER SIZE 2.
C
     CALL TEXT(1,'EXAMPLE',-500,0,2)
     CALL GRUN
C    I WILL DRAW A SQUARE IN THE CENTER OF THE
C    SCREEN. THE COORDINATES OF THE VERTICES ARE:
C    (-100,-100) (-100,100) (100,100) (100,-100)
C    THE HORIZONTAL COORDINATES ARE HELD IN IX.
C    THE VERTICAL COORDINATES ARE HELD IN IY.
C    THE DIMENSIONS ARE 5 IN ORDER TO CLOSE THE
C    SQUARE. NOTE THAT THE FIRST AND LAST COORDINATES
C    ARE THE SAME.
C
     CALL PLOT(1,0,IX,IY,5)

C    I WANT TO LOOK AT THE PICTURE UNTIL I
C    PRESS KEY 4. SO HALT THE PROGRAM:
```

```
CALL RQATN(IC,IR,4)
CALL GTERM                          !END USE OF VG
STOP
END
```

7.2.6 Task building

The task building commands must include the 4K common area VGCOM. Thus, a typical task build is:

```
TKB
NAME= NAME
/
LIBR=SYRES:RO
COMMON=VGCOM:RW:6
ASG= TI:5
//
```

7.2.7 Hints for Programming the VG

The following is a list of commonly occurring software errors and several suggestions to increase efficiency.

1. Call GRUN properly.

   If GRUN is called before a display is set up, or if it is not called at all, the program will not run but will inhibit the use of the VG, even if the program is aborted. GRUN may be called any number of times, but must appear after every GINIT or GHALT.

2. Call GTERM at end of program.

   If GTERM is not called, the program stops, leaving the picture on the screen.

3. Call GINIT at the beginning.

   If GINIT is not called before any other graphics routine, the program will automatically abort with no error messages appearing. Also, if GINIT is called again within the same program, GTERM must precede it.

4. Don't call RQATN until after GRUN has been called.

   Otherwise, no image will appear and no function key will light. The VG will be inhibited, even if the program is aborted.

5. If a curve plot of some sort is desired where either axis is incremented at a constant rate, use the SETVM, VECT, and VECTT routines to conserve buffer area.

6. If a "moving" picture effect is desired, a CHNGE call within a DO loop may be made, changing DX and DY each time. The displays in the element follow the DX,DY changes.

7. If an array of data are to be plotted, one may use IDISPL with the light pen to distinguish points rather than returning coordinates in RQATN. This is almost essential if roundoff of coordinate values allows two points the same value.

7.2.8 Additional Notes

(1) ENCODE, DECODE:

ENCODE (c,f,b[ERR=5])[LIST]

DECODE (c,f,b[ERR=5])[LIST]

(See Volume 3 of RSX-11D Fortran Language Reference Manual, Page 5-29, for explanation of these requirements.)

The formats for ENCODE and DECODE are exactly like a WRITE and READ format, respectively. Format such as (1X,I2) is valid as long as the total number of spaces is added to c.

EXAMPLES:

(a) Want NUM=214 to be displayed on the screen.

```
Code:  LOGICAL*1  A(4)
       DATA A/4*0/
       NUM=214
       ENCODE(3,100,A)NUM
       CALL INIT(1)
       CALL TEXT(1,A,0,0,1)
100    FORMAT(I3)
```

NOTE: The fourth array subscript = 0 so TEXT will end the display string. See TEXT.

(b) You have read a number 3.2 from the screen which had been displayed with TEXT, and had KEY=1 associated with it, into a LOGICAL*1 array A(3) with RDCHR. Now you want to decode it.

```
      LOGICAL*1  A(4)
      DECODE(3,100,A)RNUM
100   FORMAT(F3.1)
```

NOTE: A(4) = 0 for TEXT to end.

(c) Use other than LOGICAL*1 arrays to ENCODE and DECODE into another array.

```
      DIMENSION  I(12),J(3)
      DATA J/316,1,9260/
      ENCODE(3,100,I)J(3)
      DECODE(3,100,I)J(3)
100   FORMAT(I4)
      RESULT: I(1)='9',I(2)='2',I(3)='6',I(4)='0'
```

(d) Use DECODE to pick characters from within an array to form numbers.

```
      LOGICAL*1  TEST(10)
      DATA  TEST/'1','2','3','4','5','6','7','8','9','0'/
      DECODE(1,100,TEST)NUM1
      DECODE(3,200,TEST)NUM2
      DECODE(4,300,TEST)NUM3
      DECODE(7,400,TEST)NUM4
      DECODE(10,500,TEST)NUM5
100   FORMAT(I1)
200   FORMAT(I3)
300   FORMAT(2X,I2)
400   FORMAT(4X,I3)
500   FORMAT(9X,I1)
```

RESULT: NUM1=1, NUM2=123, NUM3=34, NUM4=567, NUM5=0.

(e) Want to ENCODE a real number,

```
      LOGICAL*1  A(7)
      RNUM=16.1237
      ENCODE(7,100,A)RNUM
100   FORMAT(F7.4)
      RESULT: A(1)='1',A(2)='6',A(3)='.', ETC.
```

(f) Use ENCODE to write a number.

```
      LOGICAL*1  TEST(6)
      DATA  TEST/'X','=',4*' '/
      NUM=2142
      ENCODE(5,100,TEST)NUM
100   FORMAT(2X,I4)
```

RESULT: If printed in character format, TEST would be X = 2142. Notice how 2X erased the 'X' and '=' data in TEST.

(2) RQATN:

Formats for retrieving coordinates in scaled mode:

INT*2,LOG*1 — X coord=IRRAY(18), Y coord=IRRAY(20)

INT*4

```
INTEGER*4 IX,IY
DIMENSION IRRAY(20)
EQUIVALENCE (IX,IRRAY(17)),(IY,IRRAY(19))
Then X coord=IX, Y coord=IY
```

REAL*4

```
DIMENSION IRRAY(20)
EQUIVALENCE (X,IRRAY(17)),(Y,IRRAY(19))
Then X coord=X, Y coord=Y
```

REAL*8

```
REAL*8 X,Y
DIMENSION IRRAY(20)
EQUIVALENCE (X,IRRAY(17)),(Y,IRRAY(19))
Then X coord=X, Y coord=Y
```

(3) Using IDISPL:

When an element is created, it occupies a storage area of its own in the display list. Each subsequent call using the element is stored together, thus each display has its own position. The variable IDISPL records that offset from the first instruction, and if the element is light penable, RQATN may be used to differentiate displays.

Given the RQATN array name as IR, dimensioned at 20, then IR(2) returns a number used with IDISPL to calculate the particular image light penned. However IR(2)-IDISPL varies depending on which command was used to create the display light penned.

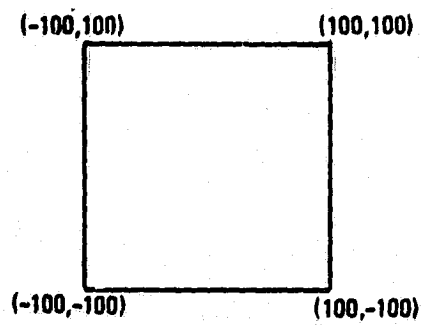
ROUTINE	IR(2) - IDISPL(words)	EACH FURTHER CALL(words)
PLINE	2	3
PLOT	4	-
POSN	-	3
TEXT	4	5
VECTT	6 in compact modes 7 in non-compact modes	-

IDISPL EX. 1

A square is drawn with element 1, by calls to POSN then PLINE. The coordinates of the vertices are: (-100,100), (-100,-100), (100,-100), and (100,100). A solid line is drawn.

```
CALL INIT(1,1)
CALL POSN(1,-100,100,IDISPL1)
CALL PLINE(1,0,-100,-100,IDISPL2)
CALL PLINE(1,0,100,-100,IDISPL3)
CALL PLINE(1,0,100,100,IDISPL4)
CALL PLINE(1,0,-100,100,IDISPL5)
CALL GRUN
```

RESULT:



VALUES:

IDISPL1	15 (INIT took 1st 14)
IDISPL2	18
IDISPL3	21
IDISPL4	24
IDISPL5	27

If RQATN was called next and IR(20) is RQATN array, the value of IR(2) is as follows:

IDISPL of Vector that was hit	IR(2)
IDISP1	no image
IDISP2	20
IDISP3	23
IDISP4	26
IDISP5	29

Notice that IR(2)-IDISPL = 2 as in the chart.

IDISPL example 2

Points will be displayed using IX(10), IY(10) to hold the coordinates. A call to TEXT with IX, IY will place an 'X' in the appropriate position. For example, if IX(2) = 1000 and IY(2) = -500, an 'X' will be drawn at (1000,-500). The initial displacement of the first TEXT command is recorded in IDISPL. Then using the light pen, the point number hit by the light pen will be calculated.

```
    CALL INIT(2,1)
    IDISPL=0
    DO 10 I=1,10
    CALL TEXT(2,'X',IX(I),IY(I),1,,,ISPL)
    IF(IEQ.1) IDISPL=SPL
10  CONTINUE
```

Result: 10 'X' 's at various places on the screen.

The first 'X' has IDISPL=15 (INIT takes 1st 14). If RQATN is now called with RQATN array IR(20), IR(2) returns the displacement +4 of the character light penned. Since each text instruction takes 5 words, a formula to find the point number would be (IR(2)-IDISPL+1)/5. Thus: The first character if light penned is:

```
IR(2)=19   IDISPL=15
     (19-15+1)/5 = 1
```

The second character if light penned is:

```
IR(2) = 24    IDISPL=15
      (24-15+1)/5=2
```

Note: IR(2) starts at IDISPL+4. Since each command generates 5 words, IR(2) increases by 5 with each point light penned.

Since PLOT and VECTT cause multiple drawings with one command, a formula for computing the point number drawn within each call is:

PLOT:	(IR(2)-IDISPL-2)/2	
VECTT:	(IR(2)-IDISPL-5)/2	non-compact types
	(IR(2)-IDISPL-5)	compact types

where IDISPL is returned in PLOT and VECTT.

(4) SETVM, VECT, VECTT

SETVM, VECT, VECTT subroutines are used together to create vectors or end point plots. They produce the same thing as the PLOT routine, but allow full usage of all plotting capabilities offered by the Vector General controller hardware.

The following paragraphs explain the features offered by these three subroutines. The reader may want to reference the VG Manual #VG 101056 "Graphics Display Unit" for more information.

There are 9 Vector modes offered by the Vector General Hardware:

(1) Absolute: The actual coordinate value of each X and Y is used. It draws a line from the previous X,Y position to the specified X,Y position. (VECT MODE = 4).

(2) Absolute Auto X: The actual coordinate value for each Y is used, but it steps the X value by a specified increment before each draw or move. Notice that display space usage is decreased since only the Y coordinate is needed. (VECT MODE = 5).

(3) Absolute Auto Y: This is the same as absolute auto X, only the X and Y coordinates are reversed. (VECT MODE = 6).

(4) Relative: Each X,Y coordinate is assumed to be ΔX and ΔY from the last coordinates, so a line is drawn from the last position – say, X',Y' to X'+X, Y'+Y, where X, Y are the input X and Y. Hence, each vector is relative to the previous one. (VECT MODE = 0).

(5) Relative Auto X: The Y coordinate is used as a ΔY adding it to the last Y position, but the X is incremented by a specified increment before each draw or move. (VECT MODE = 1).

(6) Relative Auto Y: This mode is similar to relative auto X, but the Y is auto-incremented and the X is relative. (VECT MODE = 2).

(7) Relative Compact Mode: This mode is used exactly like the relative mode described in (4) above; however, the data words are compacted to reduce space and increase performance. There are two types of compact modes: no scaling and scaling. In the no scaling mode, the coordinates must be in the range -64 to 63. In the scaling mode, the coordinates must be in the range -2048 to 2047; however, only the high order 7 bits of the number are used. That is to say, suppose the number was 1271, which is 010011110111 in binary, the lower 5 binary digits are 10111. Hence, this number is the same as 1248. In fact, any numbers between 1248 and 1279 are the same as 1248. In this mode space and performance are gained, but precision or large stepping is lost. (VECT MODE = 7).

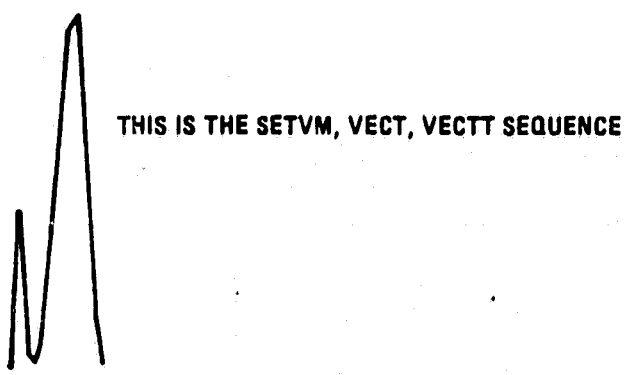
(8) Relative Compact Mode, Auto X: In this mode the X value is stepped by a constant increment (-2048 to 2047), but the Y value is in the compact mode form as explained before. (VECT MODE = 8).

(9) Relative Compact Mode, Auto Y: This mode is similar to the relative compact auto X mode, but the Y is incremented and the X is in compact form. (VECT MODE = 9).

```
C
C   THIS IS AN EXAMPLE OF USE OF THE SETVM,VECT,VECTT
C   ROUTINES. NOTE THAT NO GRAPHICS COMMAND MAY BE
C   CALLED AFTER SETVM EXCEPT VECT AND VECTT. ONCE VECTT
C   IS CALLED, GRAPHICS COMMANDS MAY BE RESUMED.
C
    DIMENSION IR(20)
    CALL GINIT                      !INITIALIZE VG
    CALL INIT(1)                    !CREATE ELEM #1
    CALL INIT(2)
    CALL TEXT(2,'  THIS IS THE SETVM,VECT,VECTT SEQUENCE',
   *  -200,200,1)                   !WRITE ON VG
    CALL GRUN                       !DISPLAY THE TEXT
C
C   SETVM IS CALLED WITH MODE 5 AND INCREMENTS OF 50.
C   THE INITIAL POSITION IS -1500,-1500 AND
C   SOLID LINES WILL BE DRAWN.
C
```

```
      CALL SETVM(1,0,5,-1500,-1500,50)
      DO 10 I=1500,3000,100              !LOOP TO DRAW SEGMENTS
      R=FLOATI(I)
      ICOORD=1500*COS(R)                 !CALCULATE COORD
      CALL VECT(ICOORD)
10    CONTINUE
      CALL VECTT(IDISPL)                 !TERMINATE SEQUENCE
      CALL CHNGE(1,,,,1000,1000)         !CHANGE SCALE TO FIT ON PAGE
      CALL RQATN(IC,IR,2)                !OTHER GRAPHICS COMMANDS
C                                        !MAY BE CALLED
      CALL GTERM
      STOP
      END
```

RESULT:



NOTE: The following is a sample task builder file for a graphics program. The common statement must be included.

```
VECT=VECT
/
COMMON=VGCOM:RW:6
LIBR=SYSRES:RO
//
```

(5) This table lists the possible characters one can draw on the Vector General. They may be drawn as follows:

```
CALL INIT(1)
ICHAR="354                    (draws a λ," means octal)
CALL TEXT(1,ICHAR,0,0,1)
```

(6) The picture scale and coordinate scale values (PC,CS respectively) are factors used in computing the size and orientation of an element's displays. They differ from each other only by their position in the formulae:

$$X'=PS/2047*(CS/2047*X+DX)$$
$$Y'=PS/2047*(CS/2047*Y+DY)$$

X',Y' - new coordinates
X,Y - old coordinates

DX,DY - increments of position of the center of the programmable area of elements

Thus, the numbers DX and DY are added to the CS scaled coordinates before the picture is scaled with PS. Note that if DX and DY are zero (default values), PS and CS are identical.

The following program exemplifies the use of PS and CS.

PURPOSE:

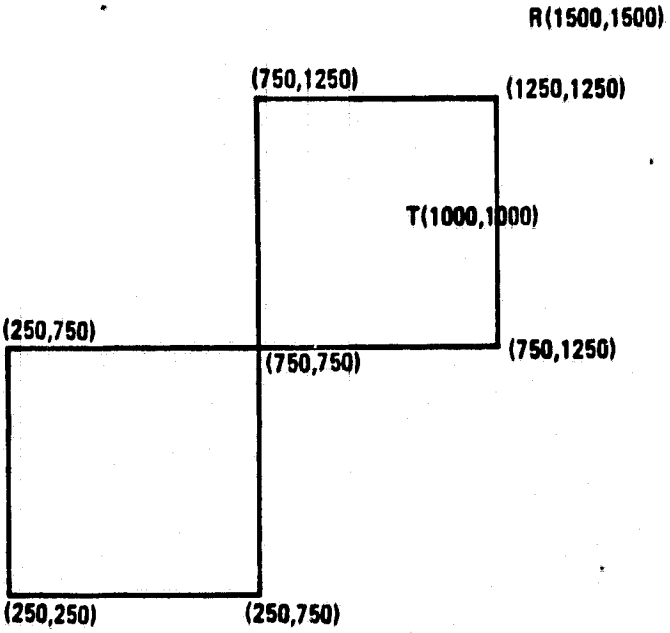
Draw 2 boxes:

Box 1: CS = 1024, PS = 2048, DX = DY = 1000
Box 2: CS = 2048, PS = 1024, DX = DY = 1000

CODE:

```
DIMENSION IR(20), IX(5), IY(5)
DATA IX, IY/-500,-500,500,500,-500,-500,500,500,-500,-500/
CALL GINIT
CALL INIT(1)
CALL INIT(2,,,1)
CALL PLOT(1, 0, IX, IY, 5)
CALL GRUN
CALL RQATN(IC, IR, 3)
CALL CHNGE(1, 0, 0,, 2047, 1024,,, 1000, 1000)
CALL TEXT(1, 'R', 1000, 1000, 1)
CALL RQATN(IC, IR, 2)
CALL PLOT(2, 0, IX, IY, 5)
CALL CHNGE(2,0,0,, 1024,2047,,, 1000,1000)
CALL TEXT(2,'T',1000,1000,1)
CALL RQATN(IC,IR,4)
CALL GTERM
STOP
END
```

RESULT:



No change formula is 1*(1*COORD+DX)

The 1st square's coords are changed via:
1*(.5*COORD+1000)

The 2nd square's coords are changed via:
.5*(1*COORD+1000)

The net effective difference is that the DX or DY is scaled when PS is changed, but is not scaled when CS is changed. Thus the second square was moved only 500 (half scale) from (0,0) whereas the first square was moved 1000 from (0,0).

Note 2:

As can be seen in the above note, DX and DY are used to compute the center of the element with respect to the screen's center. They are scaled with the display only by PS. DX and DY are increments of a change to the center of the element. Thus, if DX=100 and PS is changed to half scale (1024), the element is centered 50 away from (0,0). However if CS had seen changed to half scale, it would be centered at 100 away from (0,0).

ASCII CHARACTER CODES

Hex	Octal Left	Octal Right	Character Generator Symbol	Alpha-Numeric Keyboard Keys
00	000000	000	NUL(ignored)	@ ctrl
01	000400	001	SO "	A ctrl
02	001000	002	STX "	B ctrl
03	001400	003	ETX "	C ctrl
04	002000	004	EOT "	D ctrl
05	002400	005	ENQ "	E ctrl
06	003000	006	ACK "	F ctrl
07	003400	007	BEL "	G ctrl
08	004000	010	BS	BS
09	004400	011	HT(LF, conv)	I ctrl
0A	005000	012	LF	LF
0B	005400	013	VT(up, conv)	K ctrl
0C	006000	014	FF(up, left)	L ctrl
0D	006400	015	NL/CR, LF)	CR
0E	007000	016	SE (ignored)	N ctrl
0F	007400	017	SI (ignored)	O ctrl
10	010000	020	DLE (ignored)	P ctrl
11	010400	021	DC1 (=LF)	Q ctrl
12	011000	022	DC2 (=SE)	R ctrl
13	011400	023	DC3 (=SE)	S ctrl
14	012000	024	DC4 (term)	T ctrl
15	012400	025	NAK (ignored)	U ctrl
16	013000	026	SYN (ignored)	V ctrl
17	013400	027	ETB (ignored)	W ctrl
18	014000	030	CAN (ignored)	X ctrl
19	014400	031	EM (ignored)	Y ctrl
1A	015000	032	SUB (ignored)	Z ctrl
1B	015400	033	ESC (ignored)	[ctrl
1C	016000	034	FS (ignored)	ctrl
1D	016400	035	GS (ignored)	] ctrl
1E	017000	036	RS (ignored)	ctrl
1F	017400	037	US (ignored)	—
20	020000	040	Space	Sp bar
21	020400	041	!	1 shift
22	021000	042	"	2 shift
23	021400	043	#	3 shift
24	022000	044	$	4 shift
25	022400	045	%	5 shift
26	023000	046	&	6 shift
27	023400	047	'	7 shift
28	024000	050	(	8 shift
29	024400	051	)	9 shift
2A	025000	052	*	: shift
2B	025400	053	+	; shift
2C	026000	054	,	,
2D	026400	055	-	-
2E	027000	056	.	.
2F	027400	057	/	/
30	030000	060	0	0
31	030400	061	1	1
32	031000	062	2	2
33	031400	063	3	3
34	032000	064	4	4
35	032400	065	5	5
36	033000	066	6	6
37	033400	067	7	7
38	034000	070	8	8
39	034400	071	9	9
3A	035000	072	:	:
3B	035400	073	;	;
3C	036000	074	<	, shift
3D	036400	075	=	- shift
3E	037000	076	>	. shift
3F	037400	077	?	/ shift
40	040000	100	@	@
41	040400	101	A	A shift
42	041000	102	B	B shift
43	041400	103	C	C shift
44	042000	104	D	D shift
45	042400	105	E	E shift
46	043000	106	F	F shift
47	043400	107	G	G shift
48	044000	110	H	H shift
49	044400	111	I	I shift
4A	045000	112	J	J shift
4B	045400	113	K	K shift
4C	046000	114	L	L shift
4D	046400	115	M	M shift
4E	047000	116	N	N shift
4F	047400	117	O	O shift
50	050000	120	P	P shift
51	050400	121	Q	Q shift
52	051000	122	R	R shift
53	051400	123	S	S shift
54	052000	124	T	T shift
55	052400	125	U	U shift
56	053000	126	V	V shift
57	053400	127	W	W shift
58	054000	130	X	X shift
59	054400	131	Y	Y shift
5A	055000	132	Z	Z shift
5B	055400	133	[	[
5C	056000	134	\	\
5D	056400	135	]	]
5E	057000	136	^	^
5F	057400	137	(superscript) (subscript)	_ shift
60	060000	140	`	@ shift
61	060400	141	a	A
62	061000	142	b	B
63	061400	143	c	C
64	062000	144	d	D
65	062400	145	e	E
66	063000	146	f	F
67	063400	147	g	G
68	064000	150	h	H
69	064400	151	i	I
6A	065000	152	j	J
6B	065400	153	k	K
6C	066000	154	l	L
6D	066400	155	m	M
6E	067000	156	n	N
6F	067400	157	o	O
70	070000	160	p	P
71	070400	161	q	Q
72	071000	162	r	R
73	071400	163	s	S
74	072000	164	t	T
75	072400	165	u	U
76	073000	166	v	V
77	073400	167	w	W
78	074000	170	x	X
79	074400	171	y	Y
7A	075000	172	z	Z
7B	075400	173	{	[shift
7C	076000	174	|	\ shift
7D	076400	175	}	] shift
7E	077000	176	~	^ shift
7F	077400	177	del	DEL
80-9F	100000-	200-	(*Note)	(**Note)
	117400	237		
A0	120000	240	□	space spec
A1	120400	241	↓	1 shift spec
A2	121000	242	↑	2 shift spec
A3	121400	243	○	3 shift spec
A4	122000	244	▷	4 shift spec
A5	122400	245	◁	5 shift spec
			(centered)	
A6	123000	246	∫	6 shift spec
A7	123400	247	√	7 shift spec
A8	124000	250	⊂	8 shift spec
A9	124400	251	⊃	9 shift spec
AA	125000	252	10(subscript)	*
AB	125400	253	+	!
AC	126000	254	≤	, spec
AD	126400	255	≠	- spec
AE	127000	256	≥	. spec
AF	127400	257	¬	/ spec
B0	130000	260	°	0 spec
B1	130400	261	↑	1 spec
B2	131000	262	½	2 spec
B3	131400	263	¾	3 spec
B4	132000	264	↓	4 spec
B5	132400	265	∧	5 spec
			(centered)	
B6	133000	266	∂	6 spec
B7	133400	267	∠	7 spec
B8	134000	270	∪	8 spec
B9	134400	271	∩	9 spec
BA	135000	272	• (center dot)	: spec
BB	135400	273	×	; spec
BC	136000	274	←	, shift spec
BD	136400	275	≠	= shift spec
BE	137000	276	→	. shift spec
BF	137400	277	■	/ shift spec
C0	140000	300	**	@ spec
			(superscript)	
C1	140400	301	∀	A shift spec
C2	141000	302	—	B shift spec
C3	141400	303	⊥	C shift spec
C4	142000	304	△	D shift spec
C5	142400	305	∃	E shift spec
C6	143000	306	∈	F shift spec
C7	143400	307	∘	G shift spec
C8	144000	310	⊤	H shift spec
C9	144400	311	ψ	I shift spec
CA	145000	312	∘	J shift spec
CB	145400	313	"	K shift spec
CC	146000	314	△	L shift spec
CD	146400	315	⊞	M shift spec
CE	147000	316	⊓	N shift spec
CF	147400	317	⊡	O shift spec
D0	150000	320	∏	P shift spec
D1	150400	321	∩	Q shift spec
D2	151000	322	⊥	R shift spec
D3	151400	323	⊆	S shift spec
D4	152000	324	⊕	T shift spec
D5	152400	325	∪	U shift spec
D6	153000	326	∫	V shift spec
D7	153400	327	∧	W shift spec
D8	154000	330	∫	X shift spec
D9	154400	331	⌐	Y shift spec
DA	155000	332	⌐	Z shift spec
DB	155400	333	L	[spec
DC	156000	334	⊕	\ spec
DD	156400	335	⊢	] spec
DE	157000	336	⁻ (super)	^ spec
DF	157400	337	° (super)	_ spec
E0	160000	340	‗	@ shift spec
			(blinking)	
E1	160400	341	A	A spec
E2	161000	342	B	B spec
E3	161400	343	C	C spec
E4	162000	344	D	D spec
E5	162400	345	E	E spec
E6	163000	346	F	F spec
E7	163400	347	G	G spec
E8	164000	350	H	H spec
E9	164400	351	I	I spec
EA	165000	352	J	J spec
EB	165400	353	K (super)	K spec
EC	166000	354	L	L spec
ED	166400	355	M	M spec
EE	167000	356	N	N spec
EF	167400	357	O	O spec
F0	170000	360	P	P spec
F1	170400	361	Q	Q spec
F2	171000	362	R	R spec
F3	171400	363	S	S spec
F4	172000	364	T	T spec
F5	172400	365	U	U spec
F6	173000	366	V	V spec
F7	173400	367	W	W spec
F8	174000	370	X	X spec
F9	174400	371	Y	Y spec
FA	175000	372	Z	Z spec
FB	175400	373	[	[spec shift
FC	176000	374	|	\ spec shift
FD	176400	375	⊣	] spec shift
FE	177000	376	⁻ (super)	^ spec shift
FF	177400	377	■	DEL shift
=0	100000	200	(* note)	↑
=1	100400	201	(* note)	↓
=2	101000	202	(* note)	HOME
=3	101400	203	(* note)	→
=4	102000	204	(* note)	←

Note: * optional special characters
** ctrl and spec and ↑ A-Z [\ ] ^ _

7.2.9 Hardcopy Procedure

The entire hardcopy procedure consists of 4 basic steps:

(1) User depresses appropriate keys.
(2) Creation of PARAM.BIN and VECT1.BIN files.
(3) User initiates CPY.
(4) CPY generates hardcopy from above files.

Step 1 is accomplished by depressing SPEC and BS on the Vector General keyboard simultaneously while the desired picture is on the screen. A message, "HARDCP--STOP," will be printed on the decwriter. As soon as this is printed, the user may request another hardcopy using the SPEC/BS keys. Steps 3-4 are accomplished by typing CPY and pressing esc key on the decwriter. The output time depends on the complexity of the picture; CPY will eventually produce the final hardcopy on the Versatec.

8. Error Messages and Procedures

8.1 VG Errors

Errors may occur which relate only to calls made to the Vector General. These errors will be reported in the following form:

VECTOR GENERAL ERROR NUMBER ____

The program name where the error occurred, an error number 73 and trace-backs then follow. The error numbers reported are outlined below.

ERRORS IN GRAPHICS

ERROR NUMBER	EXPLANATION
6	No more room exists in the display Buffer to receive this operation or any others. Room must be created by deleting or resetting elements or parts of elements.
8	COMMON=VGCOM:RW:6 was not specified in the task build file. Non-existent error numbers may follow this error.
10	Tried to delete a non-existent element with DELMT.
12	Tried to activate an element that was already active with INCLD.
14	Tried to INCLD a non-existent element.
16	Tried to OMIT a non-existent element
17	Called GRUN before any elements were active or existent.
18	Tried to OMIT an element that is already inactive.
20	Tried to CHNGE a non-existent element.
22	Either a COPY of a non-existent element or a COPY to create an existent element was made.
24	Tried to PLOT or TEXT with a non-existent element.
28	Tried to ICURS an invalid key number.
29	Tried to RESET a non-existent element.

30	Tried to ICURS a non-existent key
32	Tried to RKEY a non-existent key
33	Tried to RESET an element which has an active cursor within it.
34	Tried to RKEY a key which has been activated by ICURS. RECURS must first be called.
38	Tried to INIT an existent element.
39	Tried to DELMT an element which has an active cursor within it. RECURS must first be called.
42	Tried to create a key with 20 others in existance. The maximum number of keys is 20. Use RKEY to remove any unneeded key.
43	Tried to create a key which already exists.
47	Tried to RKEY an invalid or non-existent key.
50	RQATN has no arguments or invalid arguments.
60	Hardware error in function keys. If this occurs excessively contact Code 664 personnel.
62	PS and CS are too small causing a display which would burn the screen. Thus, the call is rejected.
64	Argument problem in POSN, INIT, COPY, or CHNGE.
66	Argument problem in PLINE.
67	Argument problem in COPY.
72	Someone else is using the Vector General, GTERM has not been called previous to the call to GINIT, VGI was not installed, or VG is off.
73	COMMON=VGCOM:RW:6 was not included in task build.
74	VECTT was called before either SETVM or VECT.

78	Argument problem in STEOS.
79	Argument problem in PENTRK.
80	Argument problem in ICURS.
82	Argument problem in RKEY.
84	Argument problem in RDCHR.
85	Argument problem in SETVM.
86	Argument problem in PLOT.
87	VECT was called before SETVM.
88	Argument problem in RESET.
92	Argument problem in TEXT.
94	Argument problem in RQATN.
99	Argument problem in PENTRK or SINIT.
100	Element called is non-existent. Graphics package may be destroyed and VGI may need to be restarted. This error is serious.
113	A function key is stuck. The key number is reported. Gently lift up on it. The key may also have been pressed down too long.
120	Tried to create an element number greater than 249.
160	Key called for in RDCHR could not be found.

8.2 Other Errors

8.2.1 Utility Errors

Most errors encountered by the RSX-11D operating system utilities are reported to the user via a standard format. This consists of three characters identifying the utility, followed by the error message on the same line.

A. PIP Error Messages

Presented here are some of the most frequently encountered PIP messages. For more detail or additional messages consult pages 2-29 through 2-36 of the Utility Procedures Manual.

1. PIP -- BAD USE OF WILD CARDS IN DESTINATION FILE NAME

PIP has strict rules concerning the use of wild cards "*" in the destination. The user has used it illegally. Check Chapter 3 of this manual for the proper use of wild cards.

2. PIP -- OPEN FAILURE ON OUTPUT FILE

This usually means the file being accessed is locked. Issue the PIP command with the /UN switch. For example:

```
MCR>PIP PROG.FTN;25/UN
```

If this doesn't rectify the problem then there is the possibility of a privilege violation or parity error. Also, the file being accessed may not exist under the users UIC.

3. PIP -- OUTPUT FILE ALREADY EXISTS - NOT SUPERSEDED

An output file of the same name, type and version as the file already exists. This usually occurs when the user attempts to transfer files to tape or another UIC which already has those files on it. By using the /SU switch, the user can replace the old files on the output device with the new input files of the same name, type and version.

B. FLX Error Messages

Presented here are some of the most frequently encountered FLX messages. For more detail or additional messages consult page 3-18 through 3-25 of the Utility Procedures Manual.

1. FLX -- FILE ALREADY EXISTS

A file of the same name and type already exists on the output device. Since there is no supersede switch with FLX, the user must specify a new or corrected file-name, or use PIP instead.

2. FLX -- * IN VERSION NUMBER NOT ALLOWED

Wild cards are not allowed in the version number field of a file specifier. All version numbers must be explicitly specified.

C. DMP Error Messages

DMP Error Messages may be found on pages 4-4 through 4-6 of the Utility Procedures Manual.

D. Editor ERROR Messages

Presented here are some of the most frequently encountered EDITOR messages from some of the four classes of errors. These four classes are:

(a) Command Level informational and error messages (Pages 5-50 to 5-54 of the Utility Procedures Manual)

(b) File access warning messages (Pages 5-54 to 5-55 of the above manual)

(c) Error Messages that result in Restarting the editing session (Pages 5-55 to 5-57 of the above manual)

(d) Fatal error messages that result in EDI closing all files and terminating its execution. (Pages 5-57 to 5-60 of the above manual)

All messages from class (a) are designed to be helpful to the user. They also indicate errors with the previous command and are followed by a prompt for a new command.

For example, if a delete command encounters an end-of-buffer in block made the following message is issued:

```
[EDI -- *EOB*]
*
```

(Notice the prompt for a new command, '*'

Frequently encountered messages in this class include:

```
[EDI -- BUFFER CAPACITY EXCEEDED BY]
offending line
[LINE DELETED]
```

Files not created by the EDITOR cause this message to be printed.

To rectify the situation, the following five step procedure is suggested:

1. Start the editing session by specifying a filename that does not correspond to any file in the current directory. This causes the EDITOR to open a new file and prompt for input.

2. Enter edit mode (type CR)

3. Using the size command, reduce the number of lines read into each buffer.

   For example:

   *SIZE 50

4. Use the kill command to terminate the creation of the file.

5. Now enter the name of the desired file when the EDITOR prompts for a new file specification.

[EDI -- CREATING NEW FILE]

The input file specified does not exist so EDI has created a new file with the name specified.

[EDI -- NO MATCH]

A change command has specified a string to be changed that is not the current line.

[EDI -- *EOF*]

The user has read the end-of-file on the input file.

[EDI -- *EOB*]

The end-of-buffer has been reached. The current line pointer now points to the end of the buffer. If new lines are added at this point they will be inserted after the last line of the buffer.

Messages in class (b) indicate that the user is attempting to access directories, files or devices that are not present in the system. After each message EDI

returns to the command mode and waits for input. However, some errors in this class should not occur. If they do, consult the system manager. These messages are:

[EDI -- DEVICE NOT READY]
[EDI -- FILE ALREADY OPEN]
[EDI -- RENAME NAME ALREADY IN USE]
[EDI -- WRITE ATTEMPT TO LOCKED UNIT]
[EDI -- BAD FILE NAME]

The third class of errors, (c), are caused by conditions encountered by EDI that make it impossible to continue the current editing session. EDI closes all open files (with the exception of the secondary input file), reinitializes, and then prompts for the next file to be edited.

Frequently encountered messages in this class are:

[EDI -- FILE IS ACCESSED FOR WRITE]

The input file is currently being written by another task. Wait for the file to be written and then retry the command.

[EDI -- PRIVILEGE VIOLATION]

This usually means that the UIC under which EDI is running does not possess the necessary privileges to access the specified directory.

In this class there also are messages which indicate failure in the EDITOR and should not occur. Consult the system manager if you encounter any of the following

[EDI -- BAD DIRECTORY SYNTAX]
[EDI -- DUPLICATE ENTRY IN DIRECTORY]
[EDI -- ILLEGAL RECORD ACCESS BITS - FILE NOT USABLE]
[EDI -- ILLEGAL RECORD NUMBER - FILE NOT USABLE]

The last class of error messages (d) are considered fatal and result in the EDITOR closing all files and terminating its execution. They represent system end ware error conditions which make it impossible for EDI to continue after last message is followed by the exit message.

For example:

[EDI -- FILE HEADER CHECKSUM ERROR]
[EDI -- EXIT]

Some other messages in this class are:

[EDI -- FILE PROCESSOR DEVICE READ ERROR]
[EDI -- PARITY ERROR ON DEVICE]

Please contact the system manager if any messages in this class occur as they may indicate serious hardware malfunctions. Additionally, if the message:

TASK . . . EDI TERMINATED

appears please notify the system manager.

E. CMP Error Messages

Error messages associated with the Compare Utility may be found on pages 12-6 and 12-7 of the Utilities Procedures Manual.

F. CREF Error Messages

Messages associated with the cross-reference processor may be found on pages D-5 through D-8 of the Utilities Procedures Manual.

G. Task Builder Error Messages

Presented here are some of the most frequently uncountered error messages of the Task Builder. Additional messages may be found on pages A-1 through A-10 of the Task Builder Reference Manual.

MODULE module-name MULTIPLY DEFINES SYMBOL symbol-name

A symbol within the user program has been defined more than once.

ALLOCATION FAILURE ON FILE file-name.

Not enough contiguous space on the disk is available to build the task. Please check with the system manager if this occurs.

number UNDEFINED SYMBOLS SEGMENT seg-name

The user has an undefined symbol in this program that the task builder couldn't resolve after checking all system tables.

H. MCR Error Messages

Error messages from MCR commands may be found in alphabetical order (of MCR command) on pages A-11 through A-56 of the User's Guide.

I. System Standard Error Messages

All negative errors can be looked up on pages N-1 to N-3 of the Users Guide. These are standard errors occurring on any device or command.

J. Fortran IV Plus Compiler Errors

Fortran compiler errors may be found on pages C-1 through C-11 of the Fortran IV Plus User's Guide. Of these, the compiler distinguishes three classes of source program errors.

1. F - Fatal Errors which must be corrected before the program can be compiled. (no .OBJ file produced)

2. E - Errors which should be corrected. (.OBJ file produced)

3. W - Warning messages are issued for statements using non-standard, though accepted, syntax, and for statements corrected by the compiler. However, these statements may not have the intended result before execution. (These are produced only when the /WR switch is used)

K. Fortran (Run-time) Error Messages

Fortran error messages found on pages C-15 through C-22 of the Fortran IV Plus, User's Guide are non-recoverable and cause your task to exit.

9. PDP 11/70 Hardware Failures

Due to the fact that the PDP 11/70 does not have an operator present, it becomes necessary for users to be informed on what to do in the case of hardware failures. In general, this is just a matter of recording information so that appropriate decisions can be made at a later time based on this information.

When peripheral devices fail (such as the Hazeltines, card reader or tape drives) there is not much that can be done until normal working hours. It is best to inform others of a malfunctioning piece of equipment by leaving a note. Also, a note describing the circumstances under which the failure occurred must be left for the system manager.

Total failures of the PDP Central Processing Unit is termed a "crash". A crash always accompanied by a message on the decwriter such as:

Crash -- Cont with scratch on MMO:

There are several information gathering steps to be taken at this point that will help in determining why that crash occurred. Without changing any console switches, mount a tape on drive MMO:. Next, depress the CONT switch on the console. This will initiate a system utility, Core Dump Analyzer, to dump the contents of core at the time of the crash to tape. The Core Dump Analyzer is complete when the tape remains stationary and the console lights are fixed. When this has occurred, hit the HALT switch. Rewind and dismount the tape. Leave it on the system manager's desk.

The next step is to examine the following registers by toggling in their address, hitting the LOAD ADRS switch, then the EXAM switch. When the EXAM switch is pressed, the contents of that register should appear in the console lights. Record the octal number reflected in the console lights for each register examined. This information should be given to the system manager.

Registers	Type
17776714	RP04
17776740	RP04
17776742	RP04
17777744	Memory
17777740	Lower Address (mem)
17777742	High Address (mem)
17777766	CPU
17776710	RP04

Occassionally, a system freeze-up will occur. This is not accompanied by a message on the decwriter and is recognized only by the "frozen" console lights and no system response. This is not considered a traceable hardware error and the only action taken should be a reboot. However, if this situation occurs frequently, the system manager should be informed immediately.

Note that all unusual system activity or responses should be reported immediately to the system manager. This aids in trouble shooting hardware problems before they cause serious damage to the computer.

10. IBM 360/PDP 11/70 Tape Compatibility

10.1 SOURCE Programs

Source tapes generated on IBM machines use the Extended Binary Coded Decimal Interchange Code (EBCDIC) for the representation of characters. DEC machines, however, use American Standard Code for Information Interchange (ASCII) for characters codes. Three utility programs are available for easy interchange of information of these 8 bit codes on the 11/70 through the use of mag tapes.

[2.75] TUTILS (Tape Utilities) is used to print an EBCDIC source tape on the line printer.to create an ASCII tape from an EBCDIC source tape, and to accomplish the reverse. Since source tapes, no matter what code they are written in, contain characters in sequential order, no swapping of bytes is performed. For more detailed information on these programs, please refer to Section 13.

10.2 Transfer of Data Files

10.2.1 Introduction

There has developed a need for algorithms for conferting data on an IBM 360-generated mag tape to recognizable PDP-11 format and algorithms for generating IBM-360 mag tapes on the PDP-11. This need originates from the difference in byte addressing between the two computers. The problem applies to any INTEGER*2, INTEGER*4, REAL*4, REAL*8 or COMPLEX*8 variable. For a detailed description of the differences, refer to GSFC Technical Note #75-001, "Magnetic Tape Formats and Information Exchange Considerations." The examples in this section use tape processing subroutines which are explained in Section 11.1.

10.2.2 PDP11-IBM 360 Conversion Routines

(A) TPDPFS (To PDP From Single) – converts an IBM single-precision floating-point quantity to a PDP single-precision floating-point quantity. TPDPFS requires one or two arguments.

CALL TPDPFS(INQ[,OUTQ])

(1) INQ - specifies the variable to be converted.

(2) [,OUTQ] - specifies the destination of the variable. If omitted, the quantity is returned as a function value.

(B) TPDPFD (To PDP From Double) - converts an IBM double-precision floating-point quantity to a PDP double-precision floating-point quantity. TPDPFD requires one or two arguments:

CALL TPDPFD(INQ[,OUTQ])

(1) INQ - specifies the variable to be converted.

(2) [,OUTQ] - specifies the destination of the converted variable. If omitted, the quantity is returned as a function value.

(C) TIBMFS (To IBM From Single) - converts a PDP single-precision floating-point quantity to an IBM single-precision floating-point quantity. TIBMFS requires two arguments:

CALL TIBMFS(INQ,OUTQ)

(1) INQ - specifies the variable to be converted.

(2) OUTQ - specifies the destination of the converted variable.

(D) TIBMFD (To IBM from Double) - converts a PDP double-precision floating-point quantity to an IBM double-precision floating-point quantity. TIBMFD requires two arguments.

CALL TIBMFD(INQ,OUTQ)

(1) INQ - specifies the variable to be converted.

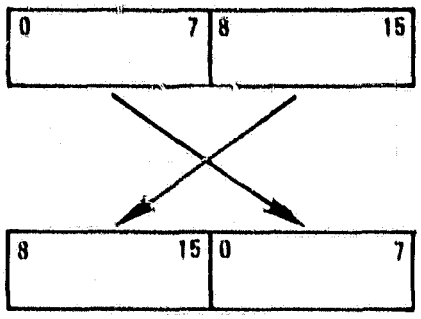
(2) OUTQ - specifies the desgination of the converted variable.

Some of the variables used in the examples which follow are:

(A) BUFF - Address of Data Area
(B) LEN - Length of Block to be Read from Tape
(C) TDAT - Halfword for INTEGER*4 Value
(D) SDAT - REAL*4 Parts of C8DAT

10.2.3 IBM 360 Tape to PDP Format

(A) INTEGER*2



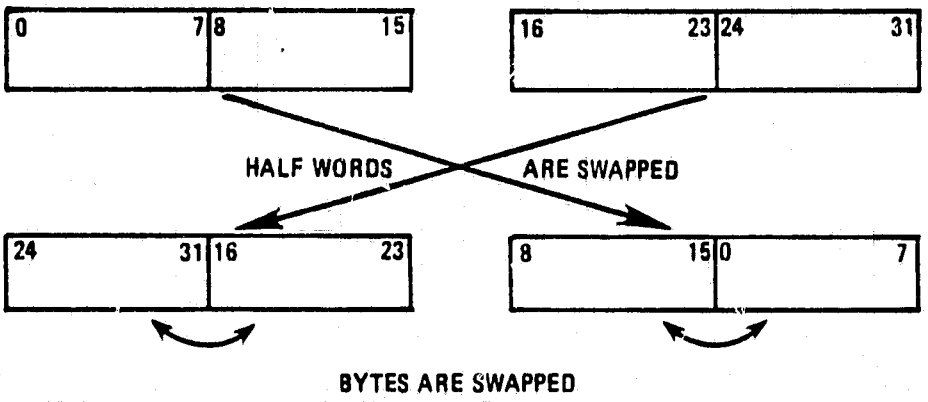
THE BYTES ARE SWAPPED

To retrieve the correct INTEGER*2 data value from a 360-generated mag tape, SWABI is called.

EXAMPLE 1: Assume the first two bytes of BUFF are an IBM 360 INTEGER*2 variable. We wish to convert these bytes to a PDP recognizable INTEGER*2 variable. The result appears in the variable I2DAT as follows:

```
LOGICAL*1 BUFF(100), IVSN(6)
INTEGER*2 I2DAT
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
EQUIVALENCE (BUFF(1), I2DAT)
CALL MOUNT (5, IVSN, 1, 'NL', 1600)
CALL DCB(BUFF, 5, 100, 100, 'FB')
CALL FREAD(BUFF, 5, LR, IOST)
CALL SWABI (I2DAT, 2)
          .
          .
          .
          .
[CONTINUE PROGRAM]
          .
          .
          .
          .
CALL DISMNT(5)
STOP
END
```

(B) INTEGER*4

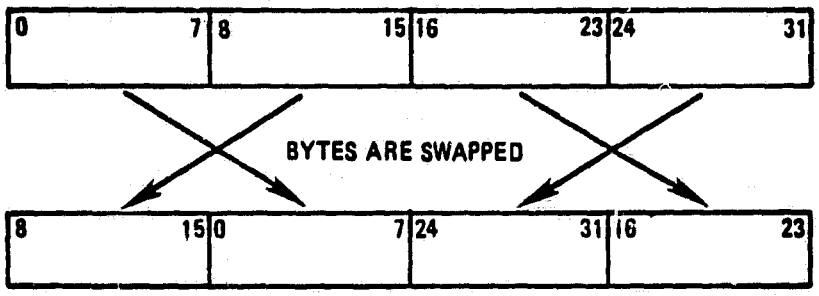


To retrieve the correct INTEGER*4 data value from a 360-generated mag tape, halfwords must be swapped following the call to SWABI.

EXAMPLE 2: Assume the first four bytes of BUFF are an IBM 360 INTEGER*4 variable. We wish to convert these bytes to a PDP recognizable INTEGER*4 variable. The result appears in the variable I4DAT as follows:

```
LOGICAL*1 BUFF(100), IVSN(6)
INTEGER*2 TDAT(2),K
INTEGER*4 I4DAT
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
EQUIVALENCE (BUFF(1),I4DAT), (TDAT(1), I4DAT)
CALL MOUNT(4, IVSN, 1, 'NL', 1600)
CALL DCB(BUFF, 4, 100, 100, 'FB')
CALL FREAD(BUFF, 4, LR, IOST)
K=TDAT(1)
TDAT(1)=TDAT(2)
TDAT(2)=K
CALL SWABI(I4DAT, 4)
          :
          :
          :
[CONTINUE PROGRAM]
          :
          :
          :
CALL DISMNT(4)
STOP
END
```

(C) REAL*4



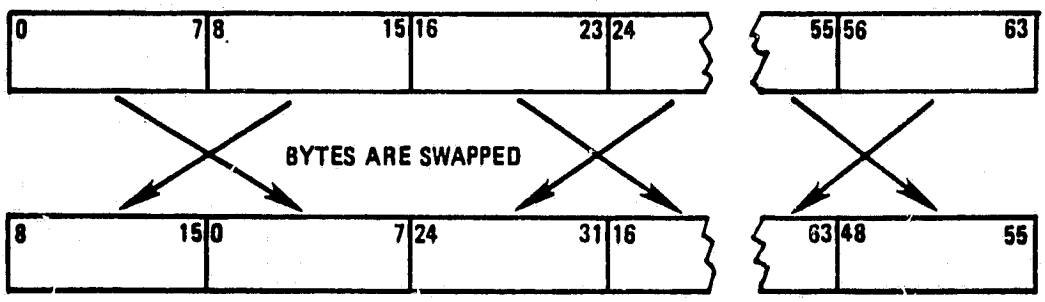
TPDPFS
PERFORMS FLOATING POINT CONVERSION

To retrieve the correct REAL*4 data value from a 360-generated mag tape, SWABI is first called, followed by the calling of TPDPFS.

EXAMPLE 3: Assume the first four bytes of BUFF are an IBM 360 REAL*4 variable. We wish to convert these bytes to a PDP recognizable REAL*4 variable. The result appears in the variable R4DAT in the following program:

```
LOGICAL*1 BUFF(100),IVSN(6)
REAL*4 R4DAT
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
EQUIVALENCE(BUFF(1),R4DAT)
CALL MOUNT(3,IVSN, 1, 'NL', 1600)
CALL DCB(BUFF, 3, 100, 100, 'FB')
CALL FREAD(BUFF, 3, LR, IOST)
CALL SWABI(R4DAT, 4)
CALL TPDPFS(R4DAT, R4DAT)
          :
          :
          :
[CONTINUE PROGRAM]
          :
          :
          :
CALL DISMNT(3)
STOP
END
```

(D) REAL*8



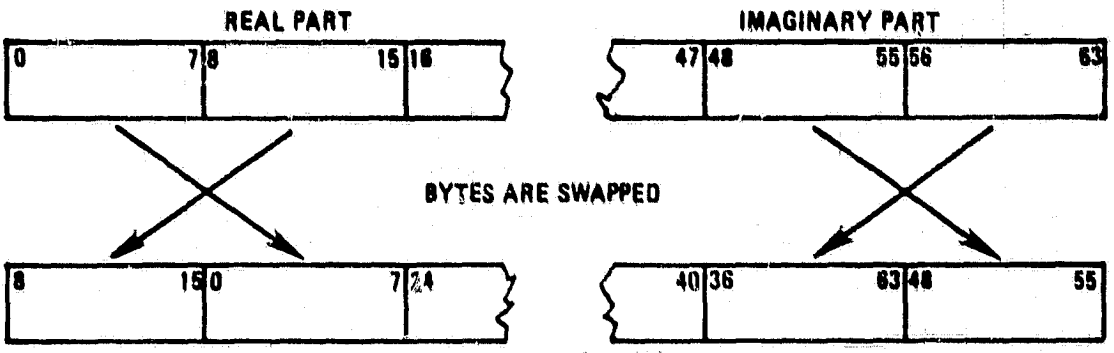
TPDPFD
PERFORMS FLOATING POINT CONVERSION

To retrieve the correct REAL*8 data value from a 360-generated mag tape, SWABI is first called, followed by the calling of TPDPFD.

EXAMPLE 4: Assume the first eight bytes of BUFF are an IBM 360 REAL*8 variable. We wish to convert these bytes to a PDP recognizable REAL*8 variable. The result appears in the variable R8DAT in the following program:

```
LOGICAL*1 BUFF(100), IVSN(6)
REAL*8 R8DAT
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
EQUIVALENCE(BUFF(1), R8DAT)
CALL MOUNT(2, IVSN, 1, 'NL', IDEN)
CALL DCB(BUFF, 2, 100, 100, 'FB')
CALL FREAD (BUFF, 2, LR, IOST)
CALL SWABI(R8DAT,8)
CALL TPDPFD(R8DAT, R8DAT)
          :
          :
          :
[CONTINUE PROGRAM]
          :
          :
          :
CALL DISMNT(2)
STOP
END
```

(E) COMPLEX*8



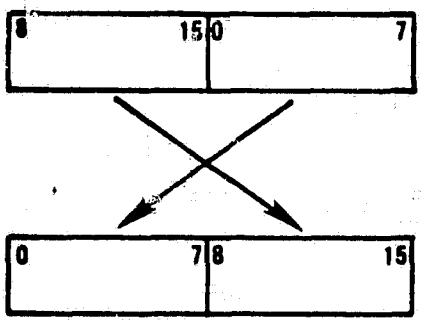
To retrieve the correct COMPLEX*8 data value from a 360-generated mag tape, the whole value is treated as two (2) REAL*4 values. Again, SWABI is first called, followed by the calling of TPDPFS.

EXAMPLE 5: Assume the first eight bytes of BUFF are an IBM 360 COMPLEX*8 variable. We wish to convert these bytes to a PDP recognizable COMPLEX*8 variable. The result appears in the variable C8DAT in the following program:

```
LOGICAL*1 BUFF(100), IVSN(6)
REAL*4 SDAT(2)
COMPLEX*8 C8DAT
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
EQUIVALENCE(BUFF(1), C8DAT), (BUFF(1), SDAT(1) )
CALL MOUNT(1, IVSN, 1, 'NL', IDEN)
CALL DCB(BUFF, 1, 100, 100, 'FB')
CALL FREAD(BUFF, 1, LR, IOST)
CALL SWABI (SDAT, 8)
CALL TPDPFS(SDAT(1), SDAT(1))
CALL TPDPFS(SDAT(2), SDAT(2))
C
C THE COMPEX*8 VALUE IS NOW CONVERTED AND
C CAN BE REFERRED TO AS C8DAT
          :
          :
          :
[CONTINUE PROGRAM]
          :
          :
          :
CALL DISMNT(1)
STOP
END
```

10.2.4 PDP 11 Tape to IBM 360 Tape Format

(A) INTEGER*2



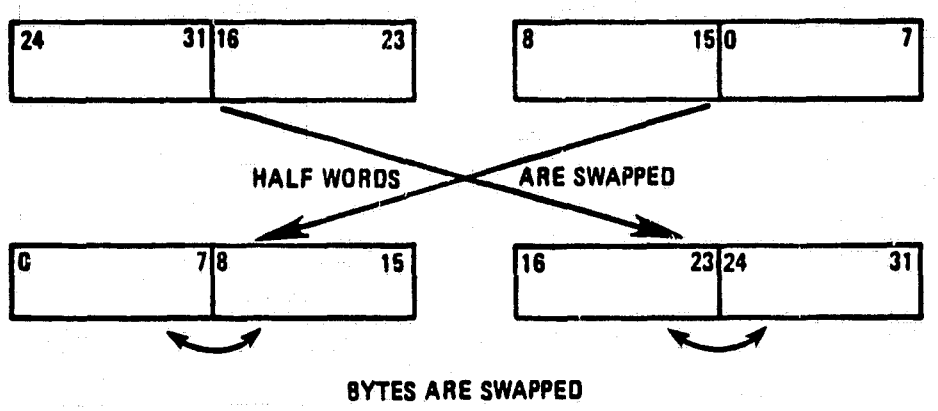
THE BYTES ARE SWAPPED

To generate the correct INTEGER*2 value onto a 360 mag tape, SWABI is called.

EXAMPLE 1: Assume the variable I2DAT is a PDP-11 INTEGER*2 variable which is to be converted to an IBM recognizable INTEGER*2 variable. After converting, the result appears in the variable I2DAT and is then written to a mag tape.

```
LOGICAL*1 BUFF(100), IVSN(6)
INTEGER*2 I2DAT
DATA IVSN/'I', 'N'. 'P', 'U', 'T', ' '/
EQUIVALENCE(BUFF(1), I2DAT))
CALL MOUNT(5, IVSN, 1, 'NL', 1600)
CALL DCB(BUFF, 5, 100, 100, 'FB')
          :
          .
          :
[CONTINUE PROGRAM]
          :
          .
          :
CALL SWABI(I2DAT, 2)
CALL FWRITE(BUFF, 5, LEN, IOST)
CALL DISMNT(5)
STOP
END
```

(B) INTEGER*4

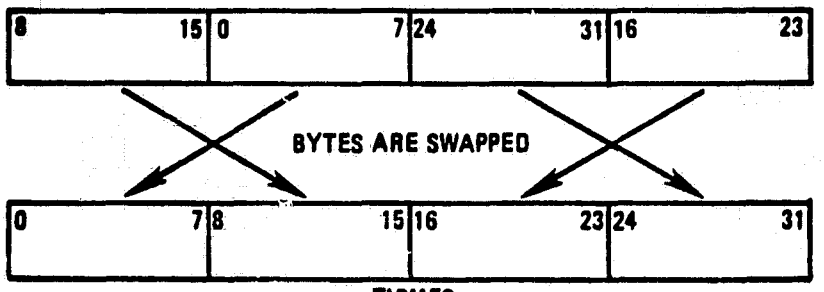


To generate the correct INTEGER*4 value onto a 360 mag tape, halfwords must be swapped following the calling of SWABI.

EXAMPLE 2: Assume the variable I4DAT is a PDP-11 INTEGER*4 variable which is to be converted to an IBM recognizable INTEGER*4 variable. After converting, the result appears in the variable I4DAT and is written to a mag tape.

```
LOGICAL*1 BUFF(100), IVSN(6)
INTEGER*2 TDAT(2),K
INTEGER*4 I4DAT
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
EQUIVALENCE (BUFF(1),I4DAT),(TDAT(1),I4DAT)
CALL MOUNT(3, IVSN, 1, 'NL', 1600)
CALL DCB(BUFF, 3, 100, 100, 'FB')
          :
          .
          :
[CONTINUE PROGRAM]
          :
          .
          :
K=TDAT(1)
TDAT(1)=TDAT(2)
TDAT(2)=K
CALL SWABI(I4DAT,4)
CALL FWRITE(BUFF,3,LEN,IOST)
CALL DISMNT(3)
STOP
END
```

(C) REAL*4



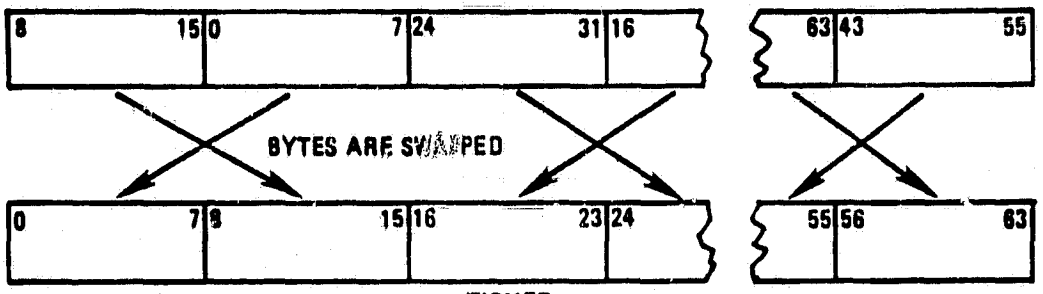
TIBMFS
PERFORMS FLOATING POINT CONVERSION

To generate the correct REAL*4 value onto a 360 mag tape, TIBMFS is first called, followed by a call to SWABI.

EXAMPLE 3: Assume the variable R4DAT is a PDP-11 REAL*4 variable which is to be converted to an IBM recognizable REAL*4 variable. After converting, the result appears in the variable R4DAT and is written to a mag tape.

```
LOGICAL*1 BUFF(100), IVSN(6)
REAL*4 R4DAT
EQUIVALENCE (BUFF(1),R4DAT)
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
CALL MOUNT(2,IVSN,1,'NL',1600)
CALL DCB(BUFF,2,100,100,'FB')
          :
          :
          :
[CONTINUE PROGRAM]
          :
          :
          :
CALL SWABI(R4DAT,4)
CALL TIBMFS(R4DAT,R4DAT)
CALL FWRITE(BUFF,N,LEN,IOST)
CALL DISMNT(2)
STOP
END
```

(D) REAL*8



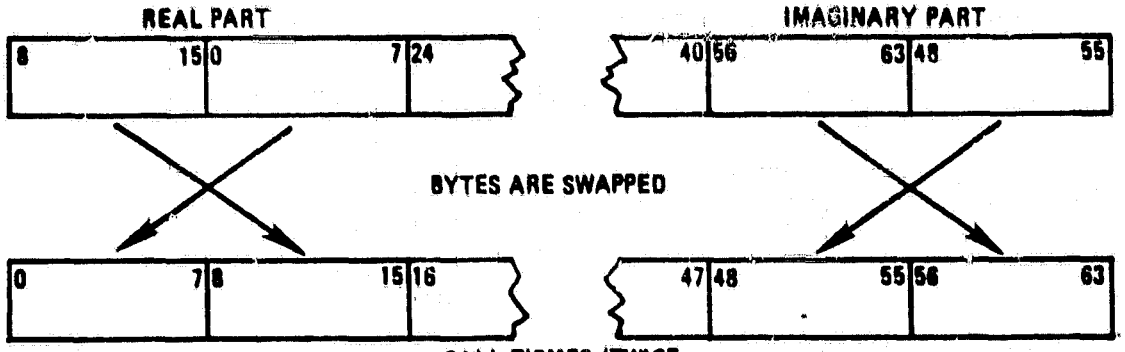
TIBMFD
PERFORMS FLOATING POINT CONVERSION

To generate the correct REAL*8 value onto a 360 mag tape, TIBMFD is first called, followed by a call to SWABI.

EXAMPLE 4: Assume the variable R8DAT is a PDP-11 REAL*8 variable which is to be converted to an IBM recognizable REAL*8 variable. After converting, the result appears in the variable R8DAT and is written to a mag tape.

```
LOGICAL*1 BUFF(100), IVSN(6)
REAL*8 R8DAT
DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
EQUIVALENCE (BUFF(1),R8DAT)
CALL MOUNT(3,IVSN,1,'NL',1600)
CALL DCB(BUFF,3,100,100,'FB')
          .
          .
          .
[CONTINUE PROGRAM]
          .
          .
          .
CALL SWABI(R8DAT,8)
CALL TIBMFD(R8DAT,R8DAT)
CALL DISMNT(3)
STOP
END
```

(E) COMPLEX*8



To generate the correct COMPLEX*8 value onto a 360 mag tape, the whole value is treated as two (2) REAL*4 values. Again, TIBMFS is first called followed by a call to SWABI.

EXAMPLE 5: Assume the variable C8DAT is a PDP-11 COMPLEX*8 variable which is to be converted to an IBM recognizable COMPLEX*8 variable. After converting, the result appears in the variable C8DAT and is written to a mag tape.

```
      LOGICAL*1 BUFF(100), IVSN(6)
      REAL*4 SDAT(2)
      COMPLEX*8 C8DAT
      DATA IVSN/'I', 'N', 'P', 'U', 'T', ' '/
      EQUIVALENCE (BUFF(1),C8DAT),(BUFF(1)SDAT(1)
      CALL MOUNT(4,IVSN,1,'NL',1600)
      CALL DCB (BUFF,4, 100, 100,'FB')
                  :
                  :
                  :
      [CONTINUE PROGRAM]
                  :
                  :
                  :
      CALL SWABI(SDAT,8)
      CALL TIBMFS(SDAT(1),SDAT(1))
      CALL TIBMFS(SDAT(2),SDAT(2))
      CALL FWRITE(BUFF,N,LEN,IOST)
C
C     THE CORRECT COMPLEX*8 VALUE IS
C     NOW WRITTEN ONTO TAPE AS C8DAT
C
      CALL DISMNT(4)
      STOP
      END
```

10.2.5 Card Reader to Disk

Files can be transferred to the disk via the card reader.

To create a disk file from a deck of cards:

1. Place deck in card reader with an E-O-F card behind the deck.
2. Press the RESET button on the card reader to get the blower started.
3. MCR> PIP filename.typ = CR:
4. RUN [2,75] UNBLNK
   to eliminate the trailing blanks at the end of each 80 byte record.

To list a deck of cards on the LP:

Perform 1. & 2 of the previous example.

3. MCR > PIP LP: = filename.typ .

11. Scientific Subroutines Package

11.1 Introduction

This Scientific Subroutines Package (SSP) is DEC's RT-11 SSP installed under RSX-11D operating system. The only changes made in implementing this SSP under RSX-11D were in the sample programs provided by DEC.

11.2 User Interface

All routines have been compiled and are available in .OBJ form. Any subroutine in the SSP can be linked to your calling routine by adding the SSP routine name to the TKB input file list as:

```
MCR>TKB
TKB>Yourname=Yourname,[2,105]SSPname1,SSPname2,...
```

For example, to link in the SSP analysis of variance routine,'ANOVA':

```
MCR>TKB
TKB>Myprog=Myprog,[2,105]ANOVA
```

11.3 Documentation Available

1. Several copies of the DEC manual "RT-11 FORTRAN SSP REFERENCE MANUAL" are available for loan and a copy is on file in the computer room.

2. The file [2,105]SSP.DOC contains the introductory comment block of all the SSP routines. This file contains the calling sequence, method used, etc. for each routine in the SSP and can be printed via:

```
MCR>PIP LP:=[2,105]SSP.DOC
```

or

```
MCR>QUE [2,105]SSP.DOC
```

Note that the file is 740 blocks long and takes 15 minutes to print.

3. Source listings of any routines may be printed as follows.

```
MCR>PIP LP:=[2,105]SSPname.FOR
```

For example, to print the source to 'ANOVA',

```
MCR >PIP LP:=[2,105]ANOVA.FOR
```

Section 12 deleted

13. Magnetic Tape Utilities

13.1 Introduction

A set of Fortran I/O routines have been developed for the PDP 11/70 similar in nature to those available in the IBM 360 FTIO package. This new package operates under the RSX-11D version 6.2 operating system with any number of dual density 9-track tape drives. Its main purpose is to read IBM unformatted binary tapes (NL or SL) in both the variable block and fixed block format. Unblocking of records is provided for in either format. It is the users responsibility to perform the necessary byte adjustments to make a 360 generated tape compatible with the 11/70.

The routines are additionally capable of reading and writing 11/70 generated tapes which are 360 compatible. These tapes can only be written in the fixed block format.

An extensive error message capability is also provided.

All tapes handled by this package must be classified as foreign volumes. All non-foreign volumes are assumed to have the ANSI standard format. Link into [2,75] FTIO2 at task build time to use this package.

13.2 Routines

A.	DCB	Sets up data control block--must immediately follow the call to mount.
B.	FREAD	Reads logical blocks.
C.	FWRITE	Writes Logical blocks.
D.	FPOSN	Positions to a specified file or writes EOF mark.
E.	MOUNT	Mount a specified volume.
F.	DISMNT	Dismount a specified volume.
G.	SWABI	Swap a specified number of bytes

13.3 Parameter Description to all Routines

Note that all references to FB or VB refer to the IBM 360 'RECFM' parameter in the JCL.

FB = Fixed Block Records
VB = Variable Block Records

Note: All arguments are I*2 except those that refer to addresses.

N = Logical unit number (User specified)

IVSN = Address of volume serial number (six byte ASCII code left-justified blank filled)

NF = Parameter which contains the file number to be processed next on the specified unit.

LEN = Number of bytes to write. This must be an even number > or equal to 14 bytes. The maximum record size is 65535.

ADR = Address of data area where bytes are to be swapped.

NUMB = Number of bytes to be swapped. Even numbers only.

LABEL = Describes the type of volume to process

LABEL = 'NL' → NL Tape
LABEL = 'SL' → SL Tape

IO = Describes the next type of I/O operation on the specified unit

IO = 'R' → Read operation
IO = 'W' → Write operation
(The 'W' operation is only used for writing EOF marks)

MODE = Type of tape you are processing. Choices are 'FB' and 'VB'.

A = Address of data area. This parameter is specified in the call to FREAD and is the data area into which records or blocks are read. Data area A should be as large as your largest unblocked record. The size of this region is specified in the 'LRECL' parameter in the call to DCB.

LR = Parameter returned to user which indicates the number of bytes read in the last read operation.

IOST = Status of read or write operation. Positive numbers indicate success. Negative numbers indicate either an EOF or error.

IOST = -10 is an EOF

All other errors are listed in Appendix I of the I/O operations references manual.

BUFF = Address of data area. This parameter is specified in the call to DCB. This data area must be large enough to contain your largest block.

When unblocking occurs, the first call to FREAD reads the data block from the tape into data area BUFF and unblocks the first record into data area A. Subsequent calls to FREAD unblocks the rest of the data, record by record, into area A until all data is unblocked. Now an additional call to FREAD will read in a new block into BUFF and again unblock the first record into area A.

When unblocking is not desired, data area BUFF should be the same area as data area A.

When 'VB' is specified the size of this region should include space for the block descriptor word and the record descriptor words which are nested in all IBM 360 variable block tapes.

The size of this region is specified in the IBM 360 job control language parameter blksize, and inthe blksize parameter in the call to DCB.

BLKSIZE = Size of your largest block.

LRECL = Length of your largest record. This number should equal the blksize if you are not unblocking your tape.

IDEN = Density of your tape. Choices are 800 and 1600.

13.4 Call Descriptions

A. CALL Mount (N,IVSN,NF,LABEL,IDEN)

This routine determines if any of the tape drives are available. If one is available the routine logically connects the tape drive to the user task and then suspends the calling task until the user continues his task with the command:

'MCR > CON TASKNAME'.

Appropriate error messages are given for such cases as no tape drives being available.

For each call to this routine there should be a final call to DISMNT on the same unit.

B. CALL DCB(BUFF,N,BLKSIZE,LRECL,IDEN,MODE)

This routine sets up the data control information for all FTIO calls on a particular unit. If you are doing simultaneous processing on two different units you would need two calls to DCB. This routine must follow the call to MOUNT. In general, the order should be a call to MOUNT followed by a call to DCB, then all subsequent FTIO calls.

NOTE: When you are doing a write operation (Call FWRITE) the only valid value for the parameter mode if FB. Additionally, the BLKSIZE and LRECL arguments should be equal. See Example 4.

C. CALL FPOSN(IO,N,NF)

This routine positions the tape on unit N to the beginning of the file specified by NF.

When processing an NL tape the following apply:

1. When an End-of-File is detected during a read operation on an NL tape and the next sequential file is to be read, it is not necessary to call FPOSN to position to the next file; just continue calls to FREAD.

2. To close out a file following calls to FWIRTE (or FPOSN), Call FPOSN with the write option in the IO parameter. This writes an end-of-file and positions the tape such that subsequent calls to FWRITE generate the file NF.

For example to write an End-of-File after File 3 on an NL tape:

```
CALL FPOSN('W',3,4)
```

3. To write an End-of-Volume, the user would CALL FPSON with the write option twice. For example, to write an End-of-Volume after two files have been written:

```
CALL FPOSN('W',2,3)
CALL FPOSN('W',2,4)
```

When processing an SL tape the following apply:

1. When an SL tape is being read, and an End-of-File is detected, it is necessary to call FPOSN to insure that the tape is at the beginning of the next file to be read.

2. Presently, PDP 11/70 FTIO does not process the SL volume, header or trailer labels.

D. CALL FREAD(A,N,LR,IOST)

This routines reads 11/70 FTIO generated tapes as well as FB and VB tapes generated on the IBM 360. This routine is called each time a new block or record is desired.

E. CALL FWRITE(BUFF,N,LEN,IOST)

This routine writes the number of bytes in data area A onto the Unit N.

F. CALL DISMNT(N)

This routine rewinds the specified volume to the load point and logically disconnects the tape drive from the user task. The calling task is then suspended until the user continues his task via the:

```
MCR > CON TASKNAME COMMAND.
```

G. Call SWABI(ADR,NUMB)

This routine swaps the number of bytes (NUMB) located in the data region adr.

13.5 Error Messages

NOTE: All errors abort the task

Error Number	Reason
1	Incorrect number of arguments in call to MOUNT
2	** Time out on attempt to read device directory
3	All drives are in use
4	** Drive not located in table
5	** Mount-insufficient pool nodes (STUFF)
6	** Mount-partition too small (STUFF)
12	Logical unit number specified is inconsistent-mount
13	Incorrect number of arguments in call to DISMNT
14	Logical unit number specified is inconsistent-DISMNT
15	** Insufficient pool nodes (STUFF)-DISMNT
16	** Partition too small (STUFF)-DISMNT
17	Incorrect number of arguments in call to FPOSN
20	Invalid "NF" specification in call to FPOSN
21	Inconsistent logical unit number in call to FPOSN
22	Invalid "IO" specification in call to FPOSN
23	Illegal to specify a "W" in the "IO" parameter when tape is SL

Error Number	Reason
24	Incorrect number of arguments in call to FREAD
25	Invalid logical unit number specified in call to FREAD
26	Data overrun - FREAD. Buffer too small for amount of data read. Some data is lost.
27	Number of bytes read does not agree with blocksize specified-FREAD
30	An odd number of bytes has been specified. Only an even number is acceptable - FREAD
31	** Number of bytes read does not agree with BDW on IBM TAPE-FREAD
32	The record just read is larger than the MAXSIZE specified in LRECL - FREAD
33	Incorrect number of arguments in call to DCB
34	Inconsistent logical unit number specified in CALL to DCB
35	Invalid density specification in CALL to DCB
36	Invalid mode specification in call to DCB
37	BLKSIZE must be a multiple of LRECL in FB tapes - DCB
40	Incorrect number of arguments in call to rewind
41	Inconsistent logical unit number in call to rewind
42	Incorrect number of arguments in call to FWRITE
43	Invalid logical unit number in call to FWRITE
44	Cannot write SL tapes
45	Incorrect number of arguments in call to SKIPREC
50	** Drive has become allocated

Error Number	Reason
51	Invalid label specification in call to MOUNT
54	Incorrect number of arguments in call to SKIP
55	Incorrect number of arguments in call to SWABI

NOTE: Asteriks before an error indicate possible hardware errors. Please contact system manager if any of these errors occur.

13.6 Examples

The following programs are examples of how to use the FTIO package.

1. The first sample program reads the first file of an IBM 360 variable block tape and outputs the records.

   Each block is 382 words or 764 bytes and is composed of 15 records per block which vary in size from a minimum of 20 bytes to a maximum of 80 bytes.

```
C------>  Declare variables

          IMPLICIT INTEGER(A-Z)
          LOGICAL*1 IVSN(6)
          INTEGER*2 LR, RKRD(40), BLK(382)

C
C------>  Set up data for tape label
C
          DATA IVSN/'T','A','P','0','6',' '/

C
C------>  Now call MOUNT.
C------>  The logical unit number chosen is 4
C------>  and the tape is NL.

          CALL MOUNT(4,IVSN,1,'NL',1600)

C
```

```
C------> Now call DCB to set up the internal data area
C------> for FREAD.
C------> Note that the array blk is the data area into which
C------> the block will be read before unblocking
C------> occurs.  Also note that 80 is the size
C------> of the largest record.
C
         CALL DCB(BLK,4,764,80,'VB')
C
C------> Now call FREAD to get the individual records
C------> If the number of FREAD calls exceeds the
C------> number of records within that block, a new
C------> block will be read in automatically provided
C------> another block exists on the tape.
C------> The third paremeter, LR, will tell you how many
C------> bytes were read in the last call to FREAD.
C
10       Continue
         Call FREAD(RKRD,4,LR,IOST)
         IF(IOST .EQ. -10) Go to 200
         WRITE(6,70) LR
70       FORMAT(1H0,'LR=',I5)
C
C------> SWAP the bytes in the data region RKRD
C
C
         CALL SWABI(RKRD,LR)
C------> Get the number of words.
C
         L=LR/2
C
C------> Now write out the record just READ
C
         WRITE(6,20)  (RKRD(I),I=1,L)
20       FORMAT(4(10I7/ /))
         WRITE (6,30)
30       FORMAT('0', 'END OF RECORD')
         Go to 10
C
C------> Dismount the tape on logical unit 4
C
200      Continue
         CALL DISMNT(4)
         STOP
         END
```

2. **The second program reads the first file of an IBM 360 fixed block tape with 800 byte blocks and unblocks it at a record length of 80 bytes.**

```
C------> Declare variables
C
         IMPLICIT INTEGER(A-Z)
         LOGICAL*1 IVSN(6)
         INTEGER*2 LR, REC(40), DAT(400)
C
C-------> Set up data for tape label
C
         DATA IVSN/'I','N','P','U','T',' '/
C
C------> Call MOUNT with logical unit number 5
C------> The tape is SL
C
         CALL MOUNT(5,IVSN,1,'SL',1600)
C
C------> Call DCB to set up the internal data
C------> For FREAD.
C------> DAT is the data area into which
C------> The block is read before unblocking
C------> into records occurs.
C
         CALL DCB(DAT,5,800,80,'FB')
C
C------> Now call FREAD to get the records of 80 bytes.
C
10       Continue
         CALL FREAD(REC,5,LR,IOST)
         IF(IOST .EQ. -10) go to 700
         WRITE(6,100) LR
100      FORMAT(1H0,'LR=',I5)
C
C------> SWAP bytes
C
         CALL SWABI(REC,LR)
C
C
C------> Get the number of words.
C
         L=LR/2
C------> Write out the record
```

```
C
          WRITE(6,200) (REC(I),I=1,L)
200       FORMAT(4(10I7/ /))
          WRITE(6,300)
300       FORMAT('0','END OF RECORD')
          Go to 10
C
C------>  Dismount the tape on logical unit 5
C
700       Continue
          CALL DISMNT(5)
          STOP
          END
```

3. This program reads the second file of an IBM 360 fixed block tape and outputs to the user the whole block without unblocking.

   The BLOCKSIZE is 800 bytes.

```
C------>  Declare Variables
C
          IMPLICIT INTEGER(A-Z)
          LOGICAL*1 IVSN(6)
          INTEGER*2 LR, BLOCK(400)
C
C------>  Set up data for tape label
C
          DATA IVSN/'D','A','T','A','1','5'/
C
C------>  Call mount with logical unit number 5
C------>  Tape is NL
C
          CALL MOUNT(5,IVSN,2,'NL',1600)
C
C------>  Call DCB to set up internal data
C------>  For FREAD
C------>  Note that the third and fourth parameters
C------>  are identical, meaning that the blocksize
C------>  and record size are the same.  This indicates
C------>  to FTIO that no unblocking is to occur.
C
          CALL DCB(BLOCK,5,800,800,'FB')
C
```

```
C------>  Now call FREAD to read the block.
C------>  It is not necessary to set up an additional
C------>  Data area when unblocking is not desired.
C
10        Continue
          CALL FREAD(BLOCK,5,LR,IOST)
          IF(IOST .EQ. -10) go to 800
          WRITE(6,80) LR
80        FORMAT(1H0,'LR=',I5)
C
C------>  Swap the bytes in the data region
C
          CALL SWABI(BLOCK,LR)
C
C
C------>  Get the number of words.
C
          L=LR/2
C------>  Write out the block
C
          WRITE(6,55)  (BLOCK(I),I=1,L)
55        FORMAT(4(10I7/ /))
          WRITE(6,78)
78        FORMAT('0', 'END OF RECORD')
          Go to 10
C
C------>  Dismount the tape on logical unit 5
C
800       Continue
          CALL DISMNT(5)
          STOP
          END
```

4. This program writes a PDP tape with a blocksize of 800 bytes. Two files, each with only one record, are written and an end-of-volume is placed after the last file.
   Tape density is 1600 BPI.

```
C------>  Declare variables
C
          IMPLICIT INTEGER(A-Z)
          LOGICAL*1 IVSN(6)
          INTEGER*2 INDAT(400)
```

```
C
C------> Set up data for tape label
C
      DATA IVSN/'O','U','T','P','U','T'/
C
C------> Call MOUNT with logical unit number 3 to file 1
C------> Tape is NL
C
      CALL MOUNT(3,IVSN,1,'NL',1600)
C
C------> Call DCB to set up data block information
C
      CALL DCB(INDAT,3,800,800,'FB')
C
C------> Set up data block to be written
C
      DO 100 I=1,400
      INDAT(I)= 5
100   Continue
C
C------> Now write the data out to tape
C
      CALL FWRITE(INDAT,3,800,IOST)
C
C------> Write an End-of-File Mark
C
      CALL FPOSN('W',3,2)
C
C------> Generate some more data

      DO 200 I=1,400
      INDAT(I)=6
200   Continue
C
C------> Write this data into file 2

      CALL FWRITE(INDAT,3,800,IOST)
C
C------> Now write an End-of-Volume mark

      CALL FPOSN('W',3,3)
      CALL FPOSN('W',3,4)
C
```

```
C------> Dismount the tape
C
         CALL DISMNT(3)
C
         STOP
         END
```

5. This program operates on a PDP 11/70 IO FTIO generated tape which has three files on it. Each block is 800 bytes in length. Unblocking of records is not desired.

   The program first positions to file 3 via the call to MOUNT. It then reads the first record of file 3 and then positions to the beginning of file 2. Finally, it reads the first record of file 2 and terminates.

```
C------> Declare variables
C
         IMPLICIT INTEGER (A-Z)
         LOGICAL*1 IVSN(6)
         INTEGER*2  DATA(400)
C
C------> Set up data for tape label
C
         DATA IVSN/'O','U','T','P','U','T'/
C
C------> Call MOUNT with logical unit number 5 to file 3.
C
         CALL MOUNT(5,IVSN,3,'NL',1600)
C
C------> Set up DCB
C
         CALL DCB(DATA,5,800,800,'FB')
C
C------> Read data
C
         CALL FREAD(DATA,5,LR,IOST)
C
C------> Now position to the second file
C
         CALL FPOSN('R',5,2)
C
C------> Now read this file
C
         CALL FREAD(DATA,5,LR,IOST)
```

```
C------> Dismount the unit
C
         CALL DISMNT(5)
         STOP
         END
```

13.7 TUTILS - General Purpose Tape Utilities

13.7.1 Capabilities of TUTILS

TUTILS is a collection of subroutines which performs various tasks using magnetic tape. It copies from one tape to another, converting from EBCDIC to ASCII or vice versa. TUTILS dumps any tape to the line printer, whether it is ASCII or EBCDIC source, or data which can be dumped in octal or hexadecimal. A very useful function is that which reads a tape and reports on record sizes, file sizes, and the number of files on a tape. This function, contained in TUTILS, requires only the label and density of the tape. TUTILS also can write an end of volume after any file on a tape, and can label a tape with either a standard label or a "no label" label, both of which are required by the IBM 360 on a new tape.

13.7.2 Sample Run

Note that the $ is printed when the (ESC) key is used on the 11/70.

The other functions available but not used in this example are designed to be self-explanatory at run time.

Run [2,76] TUTILS$

This multipurpose tape utility will perform the following routines:

1. Copy a tape to another tape with option to convert from ASCII or EBCDIC. Can be used to merge tapes.

2. Analyze a tape, revealing record sizes, number of records, and number of files.

3. Create a listing of an ASCII or EBCDIC source tape.

4. Dump a tape in octal or hexadecimal.

5. Label a tape "standard" or "No Label".

6. Write an end of volume on a tape.

7. Exit TUTILS.

Enter the number of the desired routine.

7

TUTILS -- STOP

14. General Purpose Utilities and Subroutines

This chapter contains utilities written by Code 664 personnel. All of these programs are under the UIC of [2,75].

14.1 IOPACK - Input/Output Package

IOPACK is a comprehensive input/output utility package designed to facilitate file transfer capabilities for any feasible combination of I/O devices. In addition, blocking capabilities, as well as mode translation (ASCII or EBCDIC) are provided.

The following matrix representation depicts the various devices supported for data transfer by IOPACK.*

TO: FROM	ASCII tape	EBCDIC tape	Disk	TI	LP	PTP
ASCII TAPE	X	X	X	X	X	X
EBCDIC tape	X	X	X	X	X	X
DISK	X	X	X	X	X	X
CR	X	X	X	X	X	X
TI	X	X	X	X	X	X
PTR	X	X	X	X	X	X

The program writes all operating instructions to the terminal, whereby the user makes his/her requests interactively. Any read/write device errors or character mode translation errors are displayed on the terminal.

Once the user has specified an output device, as well as an output record length, he/she may transfer as many files from as many different input devices as he/she desires.

*TI is the user's terminal, LP is the line printer, PTP is the paper tape punch, CR is the card reader, and PTR is the paper tape reader.

The instructions displayed on the user's terminal are designed to be self-explanatory. Additional information concerning IOPACK may be found under the following data set name: [2,75]IOPACK.DOC.

The following page depicts a successful run of IOPACK. Problem Definition:

Copy a disk source file to paper tape. An echo of the input file is to be directed to the line printer.

14.1.1 Sample IOPACK Run

```
*** File Copy Utilities ***

Enter responses in free format, with multiple parameters separated by commas.

Any I/O errors will be outputted to the terminal

*** Input device specifications ***

Enter device code:
(Device codes are: TAPE=1 or 2,CR=3,DB0=4 or 8,TI=5,LP=6,
Paper tape=Reader:7 or Punch:9)
4

Enter source data set name: 4
WRITE.FTN

Unit 4 is assigned to data set WRITE.FTN

Enter physical record length
80

*** Output Device Specifications ***

Enter Device Code:
(Device codes are: Tape=1 or 2,CR=3,DBO=4 or 8,TI=5,LP=6,
Paper tape=Reader:7 or Punch:9)
9

Enter physical length of output records (must be a divisor of the input physical
record length)
80
```

Enter 1 if echo print of input to LP is desired.
Enter 0 if echo is not desired.
1

Input file from LUN 4 copied to output file on LUN 9

Do you wish to copy more files to the same output device with the same output record length?
(1=yes,0=no)
0
IOPACK -- STOP

14.2 UNBLNK - Eliminates Trailing Blanks

When data are transferred from the card reader to the disk the whole 80 byte record is transferred regardless of the trailing blanks. Use UNBLNK to eliminate all unnecessary trailing blanks, thereby reducing your file size on disk. A sample run follows:

```
MCR > RUN [2,75]UNBLNK$
INPUT FILE?
TEST.FTN
END OF FILE. 3 LINES CONVERTED.
UNBLNK -- STOP
```

14.3 STUFF - Executes MCR Commands from Fortran

STUFF is a Fortran callable subroutine that provides an interface to the MCR task. It provides the capability of executing an MCR command from within a user program.

Calling Sequence:

```
CALL STUFF (ILUN, IBUF, [ISIZE], [IEFN], [IPRI], [ISTAT])
```

ILUN — is an integer specifying the logical unit number previously assigned to device SM: via TKB or the call assign subroutine.

IBUF — is an array containing or character string specifying the MCR command to be executed.

Any valid MCR command may be specified. The programmer must bear in mind that the MCR command tasks will run under the same UIC as that of the calling task and that they will be subject to the privilege status of the associated (TI) terminal.

ISIZE — is an integer specifying the length in bytes of the MCR command in IBUF; this cannot exceed 80. If omitted, the string in IBUF is assumed to be terminated by a null (octal 0) byte. (If IBUF is a character string, Fortran provides this null byte.)

IEFN — is an integer specifying the event flag to be set when this STUFF request completes. If omitted, no event flag will be set.

IPRI — is an integer specifying QIO request priority. If omitted, the priority of the task itself is used.

ISTAT — is a 2-word integer array to receive status from SM. If omitted, no status is returned. There are two categories of error codes:

Category One is identified by a zero in the first status word. This indicates an error in processing the MCR command. The following are possible codes in the second word:
-1 or -6 - invalid MCR command specification
-2 - MCR command task not installed

Category Two is identified by a non-zero in the first status word. This indicates successful completion or an error in requesting the task. The possible codes are:
+1 -- successful request completion (MCR completed)
0 -- successful request completion (MCR initiated)
-1 -- insufficient pool nodes available (6 required)
-3 -- partition too small for task

In general, if the second status word is non-negative, the STUFF succeeded.

NOTE: SM must be assigned a LUN via TKB or the CALL ASSIGN Statement. Also in the TKB, STUFF must be referenced. Sample TKB command file:

```
PROG=PROG, [11,2] STUFF
/
```

```
ASG=SM:1,TI:5,LP:6
UBR=SYSRES:RO
//
```

14.4 AECON -- ASCII-EBCDIC Conversion Routine

AECON is a Fortran callable subroutine that converts character arrays from ASCII to EBCDIC or EBCDIC to ASCII.

Calling Sequence:

```
CALL AECON(MODE,A,LEN,IERR)
```

MODE	is a positive or negative integer <0 is EBCDIC to ASCII >0 is ASCII to EBCDIC
A	address of array to be converted
LEN	integer specifying length (in bytes) or array
IERR	number of conversion errors (returned to user)

Subroutine AECON is located in UIC [2,75].

14.5 INITIAL - Initializes Floppy Disks or Magnetic Tapes

The INITIAL program is used to prepare floppy disks or magnetic tapes for use. It creates a Files-11 device that can subsequently be used under PIP.

```
MCR > RUN [2,75]INITIAL$
```

This program will prepare a magnetic tape or floppy disc for use with PIP.
CTRL Z will exit program.

```
Type tape for tape, floppy for floppy  >Floppy
Type 0 for DX0: or 1 for DX1:,UIC for floppy, a zero to six character volume
for floppy
Example:: 0,[2,104],BOB
0,[300,222],TOM
Put floppy in drive now and hit return
MOUNT-**VOLUME INFORMATION**
```

```
    DEVICE      =DX0
    CLASS       =FILE 11
    LABEL       =TOM
    UIC         =[1,1]
    ACCESS      =[RWED,RWED,RWED,RWED]
    CHARAC      =[ ]
F11ACP -- DX0: ** DISMOUNT COMPLETE **

***** FLOPPY NOW READY FOR USE *****

INITIAL -- STOP
```

14.6 SRD - Search Directory Utility

SRD is an RSX-11D utility that allows a User File Directory to be sorted in alphabetic order, to be selected according to version, date, file type, or characters within the file name. Files may also be selectively deleted, sorted and then written back in the order specified. With the last option, a PIP listing would then reveal the new ordered directory.

To invoke the SRD utility on the PDP 11/70 type:

```
MCR> SRD <CR>
```

SRD will then prompt for input:

```
SRD>
```

The format of the command line is:

```
[OUTFILE=][UIC][/SW]
```

All fields are optional. Responding to the SRD prompt with a null line causes the current UFD to be outputted in alphabetical order by type first, to the listing device.

The default order on the files is to sort on the type field first, then by name. The latest version always appears first.

SWITCH OPTIONS:

1. NAME: /NA

   With this switch the directory from the specified UIC is listed in alphabetical order.

   e.g. SRD > LP:= [100,*]/NA

2. SELECT VERSION: /SV

   e.g. SRD > LP:= [300,77]/SV

   The listing of file names in directory [300,77] is restricted only to the highest version.

3. DATE: /DA:DD-MMM-YY

   This switch allows selection of files only created on the specified day. If no date is specified, the current date is used.

   Subswitches for date selection are:

   A. BEFORE: /BE:DD-MMM-YY
   B. AFTER: /AF:DD-MMM-YY

   When one of these options are used, it causes the listing to include files created before, after or on the specified date. For example, to list all files in alphabetical order that were created on or before March 1, 1978:

   SRD > LP:= [11,111]/NA/BE:01-MAR-78

4. SELECT: /SE:NAME.TYP

   This switch allows file names to be selected based on a sub-set string match. For example, to select all files with a 'M' as the first character in the name regardless of the rest of the name:

   SRD > LP:= [1,36]/SE:M

   With this switch, it is also possible to select files with only certain characters in the file name field. For example, the following option would select any file with a name starting with 'R' having any characters in the 2nd and 3rd position and a 7 in the 4th position.

```
SRD>LP:=[2,44]/SE:R???.*
```

A subswitch for /SE is NE which causes the files selected to be those that do not match. For example,

```
SRD>LP:=[2,44]/SE:S/NE
```

selects all files that do not start with 'S'.

5. HIGHER OR SAME VERSION: /HV:N

This switch can be used to cause SRD to list only files that have a version higher or equal to a specified value. For example,

```
SRD>LP:=[2,44]/HV:10
```

will create a listing of files with version 10(octal) or higher only.

6. SELECTIVE DELETE /SD

This option causes SRD to list the selected files to the listing device. The user can then enter'Y' if the file is to be deleted. Any other response causes SRD to proceed without deleting the file. For example,

```
SRD>/DA/SD:RTK
```

causes SRD to select all the files created on the current date with the first three characters in the name 'RTK' to be listed for selective deletion.

The subswitch /DE may be applied to /SD to cause all those files selected to be deleted. For example,

```
SRD>/BE:1-JAN-78/SD/DE
```

will delete all files created on or before Jan 1, 1978. Note that entering CTRL/Z will terminate the selection and return to the SRD prompt.

7. FULL LISTING: /FU :N

This is the same as the PIP switch and can be used with all of the above options. It creates a full listing.

8. WRITE BACK: /WB

This option causes SRD to write the directory back to the disk in the order specified. This not only orders the directory, but compresses it. It reduces search time by the File Control Processor if the directory has had a lot of files deleted. For example,

SRD>[2,55]/-LI/WB/NA

causes SRD to read the directory, sort it by name, and write it back without generating a listing. Write access to the directory is required.

NOTE: Since /WB rewrites your directory, be sure that you have adequate backups in case a failure occurs. Also, note that the SRD utility only reorders and listings directories for quick access. No actual movement of files takes place.

14.7 SELECT - Moves files selected with the SRD utility to UIC [222,222]

This program was developed as one of the steps in a multi-step process to aid users of the PDP 11/70 in selecting and removing certain files from the main disk on the computer. Basically, the process consists of three steps:

1. Use the SRD utility to create a file DIRECTORY.LST (default filename under utility SRD) that contains all the selected filenames to be rolled off the disk.

2. Run the SELECT program to transfer all the files in DIRECTORY.LST to UIC=[222,222]. **Note that UIC [222,222] must be empty before starting.

3. Use PIP or FLX to output those files in UIC [222,222] to auxiliary storage media (tape, floppies, etc..)

For example, to roll off all files created before or on date June 15, 1978:

1. first create DIRECTORY.LST using SRD

MCR>SRD <CR>

SRD>[300,333]= [300,333]/BE:15-JUN-78

SRD>CNTRL Z

2. now run SELECT to move those files to UIC [222,222]

```
MCR>RUN SELECT$
      IS UIC [222,222] CLEAR?  (Y or N)
      Y <CR>

       SELECT-STOP
```

3. finally run INITIAL to initialize a tape or floppy and then PIP them onto the media.

```
MCR>HEL[2,75]

MCR>RUN INITIAL$
            .
            .
            .
INITIAL - STOP

MCR>MOU MMx:labelname

MCR>PIP MMx:=[222,222]*.*;*
```

INDEX