REPORT NO. DOT-TSC-OST-72-14

DISK OPERATING SYSTEM USER'S GUIDE

JAN P. CARLSON
TRANSPORTATION SYSTEMS CENTER
55 BROADWAY
CAMBRIDGE, MA. 02142

MAY 1972
USERS' MANUAL

Prepared for:
DEPARTMENT OF TRANSPORTATION
OFFICE OF THE SECRETARY
WASHINGTON D.C. 20590

TRANSPORTATION SYSTEMS CENTER
CAMBRIGE, MA 02142

The contents of this report reflect the views of the Transportation Systems Center which is responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policy of the Department of Transportation. This report does not constitute a standard, specification or regulation.

ACKNOWLEDGEMENTS

I would like to express appreciation to Mr. R. Hinckley for his assistance and motivation in preparing this document as part of the overall documentation for the DDP-516 Graphics System.

I would also like to express appreciation to the following people for their overall contributions to this document:

Mr. M. Form
Miss R. Fuller
Mrs. J. Gertler
Mr. R. Hinkley
Mr. C. Kalinowski
Mr. D. Kipping
Mr. L. Liebson
Dr. J. Poduska
Mr. S. Rotman
Mr. M. Scott
Mr. J. Steinberg
Mr. D. Udin
Mr. A. Zellweger

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

I. INTRODUCTION

This document serves the purpose of bringing together in one place most of the information a user needs to use the DDP-516 Disk Operating System, (DOS). DOS is a core resident, one user, console-oriented operating system which allows the user to completely control the computer and execute programs with little use of paper tape. DOS consists of:

1. an interactive command language, which controls execution of file system commands, utility functions, system programs, I/O routines, and user programs;

2. a file system and disk management system which simplifies the creation, deletion, and updating of source, object, and data files;

3. a collection of system programs including a FORTRAN compiler, a DAP assembler, loaders, source editors, on-line debug packages, and a binary editor, which are fully integrated with the file system and command language;

4. a comprehensive graphics library to fully interact with displays and light pens.

One of the more salient features of DOS is its file system. The file system gives the user the ability to organize collections of data, source, and object files under user-specified names. He has the ability to save and delete files by merely supplying the correct command and a name. All files are stored in a user file directory (UFD) which is made available to the user when he attaches to that directory. Each user has his own directory and all he has to do is insert and delete his files. All other data management and clerical functions are provided for the user by DOS.

The user interacts with DOS by typing commands at the KSR-35 console. When the command is received, the supervisor either performs the desired action immediately or initiates the appropriate system program. When the action has been performed or if an error is detected, the user is notified and the supervisor waits for a new command. Typical commands (with simplified parameter list) are:

```
ATTACH ufd
      gains access to another UFD
DELETE name
      deletes file name from current UFD
```

SAVE name sa ea pc
save core memory from sa (starting address) thru ea (ending address) and the pc (program counter) as a file named name

RESUME name
loads and executes the program SAVEd as file name

FIN name
compile the FORTRAN source file name

The Time Shared Disk Operating System is a time-shared seven user system which is compatible with DOS and uses the same disk. Under TSDOS the user sees a system nearly identical to DOS. All commands have the same name and format; the file structure is identical and the user invokes the same subroutines to communicate with the teletype, the disk, the display and light pen. Other I-O devices are not available to the user under TSDOS, but he may prepare a program which uses these devices, save it on the disk, then later run the program under DOS. The user actually has more memory available for programming under TSDOS because the user sees an empty 32K of memory whereas under DOS, he must share this memory with the operating system.

The Honeywell DDP-516 is an integrated circuit 16 bit word two's complement machine. It has 32,768 words of memory with a memory access time of 1 microsecond. Both indexing and indirect addressing are provided. Options this machine has include a high speed arithmetic unit, a real time clock, a 16 channel priority interrupt system and a direct multiplexed control channel allowing I/O without CPU intervention at a four memory cycle per word transfer rate.

Peripheral equipment includes refreshing monochrome and color displays with light pens, a Sylvania data tablet, a Calcomp plotter, six teletypes, a high speed paper tape reader and punch, a magnetic tape unit and three moving head disk files.

Chapter II gives an overview of the capabilities of the system. Chapter III lists all the hardware and peripheral equipment attached to the 516. Chapter IV runs through DOS from a beginners point of view. Chapter V gives a more detailed description of the system and explains all the commands and subsystems (computers, assemblers,etc.) available to the user. The next three chapters explain the Time Shared Disk Operating System, the H632-DDP516 Coupler, and the graphical features of the DDP-516. Chapters IX through XVII explain how to use each of the more complex subsystems such as the FORTRAN compiler Chapter XVIII describes the overlay feature. The next section explains how to read cards in, get listings out, and use paper tape. Chapter XX explains setup, shutdown

and backup procedures to be repeated daily. Due to these actions, users cannot lose more than one day of work from a hardware or software crash. Chapter XXI is a memory refresher section on how to do all the activities described in Chapters V and IX through XVII having to do with editing, compiling, loading and running programs. Chapter XXII gives a list of error messages a user can get with the DOS system, the compiler, assembler and loader.

The following special symbols are used in the text:

␣	indicates a space
.NL.	new line character
.CR.	carriage return character
.LF.	line return character
(Control X-off)	result of hitting control key and S key simultaneously. Under TSDOS, this character will cause the supervisor to take you into command wait status.

II. SYSTEM CAPABILITIES

(To be supplied at a later date)

III. HARDWARE DESCRIPTION

A. List of Equipment

1. The Computer

The CPU is a Honeywell DDP-516, an integrated circuit 16-bit word machine. It has 32K of core memory with a memory access time of 0.96 microseconds. Paging modifications have increased the basic memory access time to 1.080 microseconds. The machine organization is parallel and the arithmetic unit is two's complement. Both indexing and multilevel indirect addressing are provided. The instruction set includes 72 instructions and a single instruction may directly address any one of 1024 words. If intelligent programming is done however, a single instruction will usually only address a maximum of 512 words directly.

The following standard Honeywell options are available on our machine.

- High Speed Arithmetic Unit allowing:
  5.28 microsecond maximum multiply
  10.56 microsecond maximum divide
- Real Time Clock
- Restricted Mode Option
- Extended Addressing Control for 32K memories
- 16 Channel Priority Interrupt System
- 16 Channel Rate Multiplexed Control (DMC)
  allowing I/O without CPU intervention
  at a four memory cycle per word transfer
  rate.

For a complete description of the central processor and its options see DDP-516 Programmers Reference Manual.

The following options built in-house are available for use.

Paging Hardware

In addition to the above option, CPU modifications and special logic circuits are used to allow paging. This system includes a four member associative store and a base relocation register. The paging mechanisms can be bypassed if so desired.

Bit Banger Multiplexer and Clock

A simple teletype multiplexer and a 330 cps clock used for time sharing.

2. Peripheral Hardware

The following standard peripherals are available.

- ASR Console Teletype
  10 characters per second
- Moving Head Disk File
  3 - controlled 3.6 million word moving head disks. Access time is between 20 and 160 milliseconds with a word transfer rate of about 12 microseconds per word.
- High Speed Paper Tape Reader
  300 characters per second
- High Speed Paper Tape Punch
  110 characters per second
- Single-Line Controller
  This device is used to send and receive data over a 2400 bps dataphone.
- Low Capacity Multiline Controller Special Option
  Allows up to 32 teletype communication lines to be serviced by the 516 as a single device.
- Magnetic Tape Controller
  Controls one 36 inches-per-second 7 channel tape drive. Maximum transfer rate is 9,600 words per second.
- 516-832 Direct Coupled System
  Allows high speed interchange of information between the two computers (DMC speed). 150,000 words/second.
- High-Level Analog Input Subsystem
  Scans, multiplexes and encodes high voltage level inputs. Was to be used for speech input.

3. Non-Honeywell Peripherals

- Displays
  1-Information Displays, Inc. display generator.
  3-IDI high resolution, monochromatic equipped with light pen displays.
  1-International Telephone and Telegraph three color, lower resolution display.
- ARDS storage tube display with mouse.
- Drum
  Vermont Research Corp. magnetic drum memory with about 2 million word capacity with maximum access time of 17 milliseconds and transfer rate of 1 word every 12 microseconds.
- Sylvania Data Tablet
  allows high resolution (1024 x 1024) data input at a maximum rate of 1 point every 5 milliseconds.

- Calcomp Plotter
  Hard copy graphic output.

B. How to use Equipment

1. High Speed Arithmetic Unit

   a. Assembly Language Instructions
   see Programmers Reference Manual
   b. Program packages
   used automatically by FORTRAN library.

2. Real Time Clock

   a. Assembly Language Instructions
   see Programmers Reference Manual. This option is wired to priority interrupt 11 (Memory location 76_8).
   b. Program packages
   see Section VIII.E. of this manual

3. Restricted Mode Option

   a. Assembly Language Instructions
   See Programmers Reference Manual. This option is wired to priority interrupt 12 (Memory location 77_8).
   b. Program Packages
   This option is only used for time sharing. See III. 2.1 page 4 of TSDOS Reference Manual.

4. Extended Addressing Control for 32K Memories

   a. Assembly Language Instructions
   See Programmers Reference Manual.
   A modification has been implemented in the Extended mode option. When in the extended mode, indexing will occur before indirecting if the effective address of an instruction is in sector zero and is less than 32. This modification allows such instructions as LDA* 0,1 to work the same way whether in the extended mode or not.

   b. Programming Packages
   none

5. 16 Channel Priority Interrupt System

   a. Assembly Language Instructions
   See Programmers Reference Manual. The interrupt structure has been altered to allow better protection

of certain instructions. The JST instruction protects itself and the following instruction from interrupt. Thus, a re-entrant subroutine may be constructed with an initial instruction of INH or LDX (if the DAC is never re-used). The ENB instruction protects the following instructions from interrupt even when already enabled. There is a problem in that the sequence

```
ENB
NOP
INH
```

will not allow a standard interrupt to occur. This sequence appears in the 46T1 disk diagnostic and consequently the standard 46T1 will not run properly. An additional peculiar feature of the SI interrupt is that an interrupt may occur immediately after an INH if the INH is not otherwise protected by a JST or ENB instruction.

b. Program Package
none

6. 16 Channel Data Multiplexed Control

a. Assembly Language Instruction
See Programmer Reference Guide.

b. Program Packages
none

7. Paging Hardware

a. Assembly Language Instructions.
Bit 10 of the operation register indicates that the machine is in the paged mode. The paged mode is enabled by an SMK '1320. Before executing the SMK instruction, the A-register must be set up for loading the Page Map Register (PMAR), according to the following scheme:

(FOR 256 WORD PAGES)

BITS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	PMAR									PAGE NUMBER						
	9	10	11	12	13	14	15	16								

Corresponding bits of A-Register transferred to PMAR on an SMK '1320

E.G., To start the Page Map at 10000_8 (Bit 4), the A-Register should contain 40_8 (Bit 11 set)

After the execution of the SMK '1320, paging will be enabled with the PMAR set. However, mapping will not occur until after the excution of the first JMP instruction. This permits paging to be enabled and followed by a non-paged jump to a location where page mapping will begin. Master clear disables paging and clears the PMAR and the associative store.

b. Program Packages
The paging hardware is used only by the time sharing system.
See section V. of TSDOS Reference Manual.

8. Bit Banger Multiplex and Clock

a. Assembly Language Instructions
See VI. 2.2 of TSDOS Reference Manual.

b. Program Packages
The bit banger multiplexor is used only with time sharing.
See VI. 2.3 of TSDOS Reference Manual.

9. ASR Console Teletype

a. Assembly Language Instruction
See Programmers Reference Manual.

b. Program Packages
Normal FORTRAN drivers and alternate teletype I.O.
See X.B

10. Moving Head Disk File

a. Assembly Language Instruction
See Programmer Reference Manual.

b. Program Packages
The DOS system user routine described in IV. 2 of TSDOS Reference Manual. The user will not use these routines but will read and write information on the disk through calls to subroutines in the FORTRAN library. See section X. B of this manual. Furthermore the disk is available as a FORTRAN I.O device.

11. High Speed Paper Tape Reader

a. Assembly Language Instructions
See Programmers Reference Manual.

A simple change has been made to the paper tape reader to cause it to work in a 'single-character' rather than 'continuous mode'. When the paper tape reader reads a character, its motor will be turned off until the computer reads in the character. Normally, the INA for reading occur within microseconds of the read and the motor never slows. However, if there should be a delay in reading, the paper tape reader will no longer run away.
This modification is especially useful for interrupt driven I/O or time-shared operation of the reader.

b. Program Packages
Users are discouraged to use the paper tape reader. Encouraged is getting paper tapes onto the disk then having program read the disk.
See section XIX of this manual for this procedure.

12. High Speed Paper Tape Punch

a. Assembly Language Instructions
See Programmers Reference Manual

b. Program Package
Users are discouraged to use the punch. Disk files may be punched onto paper tape by a procedure explained in Chapter XIX.

13. Single Line Controller

a. Assembly Language Instructions
See Single Line Controller Option Manual.

b. Program Packages
See System Specifications for DDP-516- Universal 1004.
Communications Link.

14. Low Capacity Multiline Controller

a. Assembly Language Instructions
See Low Capacity Multiline Controller Option Manual.

b. Program Package
The multiline controller is used with TSDOS.
See VI. 6 of TSDOS Reference Manual.

15. Magnetic Tape Controller

a. Assembly Language Instructions

See Programmer Reference Manual.

b. Program Packages
Magnetic tape is available as a standard FORTRAN I/O device.
See Chapter XIX to read magnetic tape into a disk file or write from a disk file to magnetic tape.

16. Direct Coupled System

a. Assembly Language Instructions
See System 32/16 Coupler Option manual.

b. Program Packages
See section VII of this manual

17. High Level Analog Input System

a. Assembly Language Instructions.
See High Level Analog Input Subsystems Special Options Manual.

b. Program Packages
none

18. Displays

a. Assembly Language Instructions
See VI 2.7 of TSDOS Reference Manual.

b. Program Packages
See Graphic Reference Manual and section IV. C. 3 of this manual. The display is available as a FORTRAN I/O device.

19. ARDS Storage Display

a. Assembly Language Instructions
See ADRS Manual.

b. Program Packages
See Chapter VIII. B of this Manual

20. Drum

a. Assembly Language Instructions

OCP	'0042	Reset Drum Controller
SKS	'0242	Skip if ready for command transfer
SKS	'0342	Skip if ready for data transfer
SKS	'0442	Skip if no data transmission error
SKS	'0542	Skip if no reading error
INA	'0142	Input status word
INA	'0342	Input data word
OTA	'0242	Output command
OTA	'0342	Output data

b. Program Packages
none

21. Sylvania Data Tablet

a. Assembly Language Instructions

INA	'240	read y and z and reject information
INA	'140	read x and z and reject information
format	bit 1	if on point is reject
	bits 2-4	height info 000 on surface
		001 near
		011 up
		111 way up
	bits 5-14	x or y information 0, 0 is at lower left

b. Program Packages
See section VIII. B of this Manual

IV. TYPICAL CONSOLE SESSION WITH DOS

A. Introduction

The reader is assumed to know nothing about the DOS system although he is expected to know FORTRAN and the fundamentals of computer programming. The reader will be led through the process of which a FORTRAN program is inputed, compiled, loaded, and executed under the DOS system. No attempt will be made to explain in detail the external commands used, but references to other chapters will be used. The chapter assumes only one disk drive is being used.

B. Brief View of DOS

The Disk Operating System (DOS) for the DDP-516 is a core resident, one user, console-oriented operating system which allows the user to completely control the computer and execute programs with little use of paper tape. All files for source binary and core images are known by name to the user, and DOS handles all problems of allocating the resources of the disk.

C. A Console Session with DOS

Let us start from the point a user enters the computer room with the computer shut down. First turn on the machine, load the DOS master disk pack on disk drive zero and turn on the disk drive. Users should not do the above unless they have authorization.

Load a ten foot paper tape labeled DOS BOOTLOADER in the paper tape reader. Turning to the console, master clear the computer, check that all sense switches are in the reset position and start the computer with the program counter set to 1. The BOOTLOADER will read into the machine and begin execution. This program reads DOS from the disk into core then jumps to DOS. DOS responds on the console teletype "OK,". From this point on, all user action is initiated from the console teletype.

```
user:       ATTACH TSDOSA
response:   OK;
```

Before we can begin work, we must "attach" to one of 60 possible user file directories (ufd). Each ufd contains access to up to 62 files on the disk. These files may contain FORTRAN source programs, binary data, core images or other data. User file directories are themselves files of the master-file directory (MFD). The MFD is also stored as a file on the disk.

"ATTACH" is one of many internal commands of DOS. "TSDOSA" is the name of one of the user file directories of DOS. The command line "ATTACH TSDOSA" causes us to have access to any file whose names are in the TSDOSA user file directory.

```
user:       AAA?EDXX
response:   EDXX    NOT FOUND
```

The "?" character kills all input on the line up to the "?". "EDXX" is a word not recognized by DOS as a legitimate command. The response is a DOS error message.

```
user:       ED
response:   GO
            INPUT
```

ED is the name of an external command. When a user types a command to DOS, DOS looks the word up in a list of internal command words. If the name is not in the list, DOS looks the name up in a special external command directory CMDHCO. If the name is found, DOS reads in a core image file and starts executing the program with initial program counter, A register, B register, index register and keys as specified in a special part of the file. An external command, in other words, is a command which causes a core image file to be read into core and begin execution. When execution is ready to begin, DOS types "GO" and transfers control to the newly loaded program. After this point, the teletype as well as other devices are controlled by the new program and not by DOS. The program types "INPUT" and is now waiting for teletype input.

ED is a text editor. This program allows one to input a FORTRAN source program as well as any other text through the teletype. We may then edit the text by a series of commands to ED. Other ED commands cause ED to read and write files of text on the disk. ED does this indirectly by calling on special DOS subroutines to do the actual reading and writing. ED commands should not be confused with DOS commands.

At this point in time, ED has just typed INPUT informing

us that we are in high speed input mode. In this mode, typed lines are entered directly into ED's text buffer

```
user:       C/IDIOT FORTRN"AN PROGRAM
response:   none
```

The first line is a FORTRAN comment. The backslash character "\" is the logical tab convention for ED. The logical tabs may be reset by the editor command TABSET. Initial tabs are set at columns 6, 12, and 20. Spaces are entered in the text buffer to the next tab position. No spaces are typed in response to the input.

The character " is the erase character. Typing this character will cause the preceding letter to be deleted from the line. Successive use of the character will cause successive preceding characters to be removed. As in DOS, a "?" will "kill" the entire input line up to the "?". The erase and kill characters are effective whether in the INPUT or EDIT mode. The editor makes no responses so long as in INPUT mode.

```
user:       \␣INTEGER JIT
user:       \␣JIT=1
user:       \␣WRIT(1,10)JIT
user:        10\␣FORMT C5X,15)
user:       \␣CALL EXIT
user:       \␣END
user:        $0
user:
response:   EDIT
```

Two carriage returns in succession cause the editor to switch modes. Since we were in INPUT mode, we have switched to EDIT mode. While in EDIT mode, typed lines are interpreted as commands to the editor.

EXIT is a special subroutine, which when called, returns program control to DOS which types "OK".

```
user:       TOP
response:   none
```

TOP is a command to the editor which moves a pointer which points to lines of text in the text buffer. This pointer is moved to point to a "null" line at the beginning of the text buffer.

```
user:       FIND 11
response:   BOTTOM
```

The FIND command moves the pointer forward from the line following the current line to the first line beginning with the string "11". The pointer now points to a "null" line after the last line of text in the buffer. What I meant to do was type FIND 10, but mistyped it.

```
user:       TOP
response:   none

user:       FIND   10, PRINT
response:   10   FORMT(5X,15)
```

The sequence of commands above means go back to the top of the buffer, look for a line beginning with "10", then print that line. The second user input shows stacking of commands. Two commands have been typed on the same line separated by a comma. The commands are not executed until the entire line has been typed. The commands are then executed left to right

```
user:       RETYPE 10  FORMAT(5X,15)
response:   none
user:       PRINT
response:   10   FORMAT(5X,15)
```

RETYPE replaces the current line by the string that is the argument of the retype command.

```
user:       FILE IDIOT
response:   OK;
```

FILE is a command to write a file which contains the text buffer of the editor as data. IDIOT is an alphanumeric name chosen by the programmer. The file name must start with a letter and may be 1-6 characters in length. The FILE command returns control to DOS at its completion. "OK" is then typed by DOS.

Chapter XV describes in detail all the commands of the editor. This chapter should be used along with this one in case the user wishes to manipulate the text buffer in a way not described here.

```
user:       FTN IDIOT
response:   GO
            WRIT(1,10)JIT
            ****CH
            CC, OK;
```

FTN is an external DOS command. The FTN core image file

of the directory CMDHCO is read into core and execution begun. FTN compiles the source file IDIOT. FTN generates a name which it uses for writing binary data by concatenating "B←" onto the beginning of the filename and truncating the resulting name to six characters. In this case, binary data is written on file B←IDIO.

When compilation errors are detected, the line causing the error is typed followed by the error type. When FTN reads the $0 line of the file IDIOT, FTN types "CC," for compilation complete than returns control to DOS. DOS types "OK".

Our FORTRAN program contains one error. By examining the erroneous lines we see that WRITE was misspelled. We must correct the source file and recompile it. See chapter IX for a complete description of how to use FTN.

```
user:       DELETE B←IDIOT
response:   OK;
```

One should always "erase" the old copy of the binary file generated by a bad compilation before recompiling. This is done with the DOS internal command DELETE. DELETE returns the disk storage used by the files whose name is the argument of DELETE. The filename is also removed from the file directory.

```
user:       ED IDIOT
response:   GO
            EDIT
```

We are preparing the file IDIOT to be modified.

```
user:       LOCATE WRIT, PRINT
response:         WRIT(1,10)JIT
```

LOCATE repositions the current line pointer to the next line in the text buffer containing an occurrence of the string that is its argument.

```
user:       CHANGE   /WRIT/WRITE/, PRINT
response:         WRITE(1,10)JIT
```

The change command is used to replace one string of characters in the current line by another. "/" is used in this case as the delimiter of the strings, but almost any character will do.

```
user:       FILE
response:   OK;
```

The FILE command with no argument will write the text out on the same filename as the text was read in on. FILE returns us to DOS command level.

```
user:       FTN IDIOT
response:   GO
            CC, OK;
```

Recompile IDIOT with binary output B←IDIO. This time there are no errors.

```
user:       CLRCOR
response:   GO
            OK;
```

CRLCOR is an external DOS command that puts zero in locations 20-64777 octal. This is the area of core in which programs are normally loaded. This step should always be done before loading

```
user:       LDR  B←IDIOT
            GO
            MR, OK;
```

LDR is another DOS external command. LDR is a program that loads binary files into core. LDR loads the file B←IDIO from the user's file directory. DOS automatically truncates the name to six letters. When the loader detects a piece of binary data equivalent to the $0 line of the FORTRAN source program, the LDR types MR, meaning "more" or LC meaning "loading complete", then returns to command level DOS. The loader automatically loads programs beginning at location 1000 octal with "links" beginning at 100 octal. Programs are loaded in the extended addressing mode. See Chapter XIV for a complete description of how to use the loader

```
user:       ATTACH U
response:   OK;
```

We must load a binary data file FTNLIB to complete our load. Since this file is used by all users, it is accessed through the U directory, a directory which is used by all users and belongs to no one user. We have "attached" to the U directory in preparation for loading this file.

```
user:       START FTNLIB
response:   GO
            LC, OK;
```

The DOS internal command START transfers control to the

location given as its first numerical argument. If no numerical argument is given, control is transferred to the location plus one of the places from which DOS was last entered. The location in this particular case, is the place in the loader to continue a normal load. The loader starts reading the FTNLIB file but only loads those routines necessary. "LC" indicates we now have a complete load.

```
user:       CLOSE ALL
response:   OK;
```

The loader leaves the file it has just read from in an active state. CLOSE ALL closes out the file.

```
user:       ATTACH TSDOSA
response:   OK;
```

Return to our own directory.

```
user:       START 63002
response:   complete load map is typed.
```

Location 63002 octal is the place to begin execution of the loader to generate a load map. The loader returns to DOS command level when done.

```
user:       SAVE *IDIOT 100 5000 1000
response:   OK;
```

Generate a core image file named *IDIOT containing locations 100 to 5000 as data. Also save a vector of machine initial conditions for restoration when the file is resumed. The machine conditions are given beginning as the fourth parameter of the SAVE Command. These are in order program counter, A-register, B-register, index register, and keys. In this particular case, the program counter initial condition is set to 1000 octal; all other machine registers are left to the value they were the last time control was returned to DOS. In particular, the keys are set to indicate extended mode, as the loader runs in extended mode and was the last program to return to DOS. In this way, *IDIOT is saved to run in the extended addressing mode, the mode it was loaded in. The user may specify all machine registers explicitly if he wishes.

The lower limit for the SAVE is always 100. The upper limit is derived by looking at the number associated with the word HIGH on the load map. This number is the highest location in octal used by the programs that have been loaded. The initial program counter is 1000 octal because the main

program was loaded beginning at location 1000.

```
user:       RESUME *IDIOT
response:   GO
                        10K;
```

RESUME is a DOS internal command that reads a core image file into core, sets the machine registers from a part of the file and begins execution.

The FORTRAN program now executes. It types out spaces followed by a "1" then returns to DOS command level. DOS types "OK".

```
user:       LISTF
response:   UFD=TSDOSA
            IDIOT
            B-IDIO
            *IDIOT
            OK;
```

LISTF is an internal DOS command which lists the contents of the file directory one is currently attached to.

```
user:       CLOSE ALL
response:   OK;

user:       ATTACH
response:           NOT FOUND
```

The above two commands are given before a user leaves the console when his time is up. CLOSE ALL insures all files are closed and ATTACH "deattaches" from your directory leaving you attached to no directory. This prevents the next user from accidently using your directory.

V. DOS SYSTEM

A. DOS File System

1. File Organization

The file system provides a mechanism whereby the user may store all the information required before, during and after operation with the system. The information required is maintained at the user's fingertips, and protected from interference whether intentional or otherwise.

Information on each disk is recorded in files, and these files are composed of fixed length records which are recorded chained together by pointers on the disk.

A *file* is simply an ordered linear array of words known by a name. A file is given a name, say "alpha,; and one can usefully speak of the ith word of alpha as "alph(i)." The name is from 1 to 6 alphanumeric characters. The file system has no notion of the contents of the file: It matters not whether the file "alpha" contains symbolic, binary, or procedural information. In this sense, the file system is a purely *external* file system.

The collection of files belonging to a user is organized into a User-File-Directory (UFD). Each UFD is a file and contains a four-word entry per user file (presently 62 files max.), 3 words for name (6 characters), and 1 word for the starting record address.

The collection of all UFD's on one disk is organized into a file directory called the Master-File-Directory (MFD). The MFD is structured in precisely the same way as is the UFD. The MFD's are the only files known absolutely to the system; all other files are known relatively by pointers through file directories.

The files on a disk are recorded as chained strings of fixed length records as shown schematically in Figure III.1. Each record of the file has three words at the beginning which function as follows: word one points to the successor record (zero, if none), word two points to the predecessor record (zero, if none), and word three is the count of the data words contained in this record. The forward and backward

pointers are especially important because they allow easy traversal of the file forward and backward while at the same time providing a large measure of protection against snow-balling disk errors.

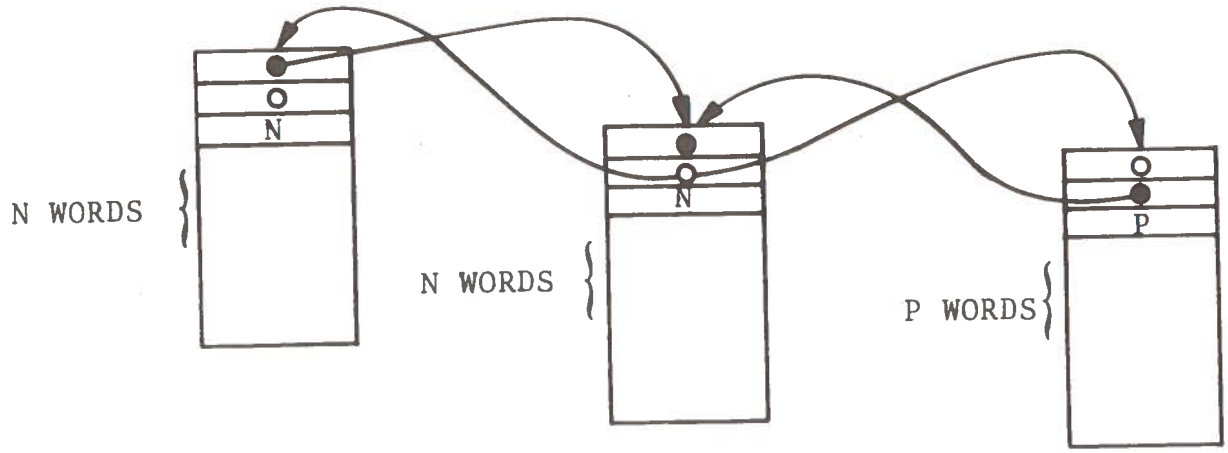


Figure III.1

2. Structure and Allocation of Disk Records

Information on the disk is recorded in 432 word records. Each disk pack (IBM-1316) has 203 cylinders of 10 tracks each, and 4 records of 432 words are formatted per track. (Actually, 435 words could have been recorded with a gap of 363 bits; but with 432 words, the intersector gap can be 460 bits allowing for more software setup time.) The record address recorded with each record is an encoding of sector, head, and cylinder as follows:

RECADR = SECTOR + 4* (HEAD+10*CYL).

Since files are composed of chained records, the selection of records can be left entirely to the file system. The

file system must select free records when a file is to be written or extended, and the free records of a deleted file must be returned for later re-use. For this purpose a Disk-Record-Availability-Table (DSKRAT) is maintained with one bit for each record on a disk; a "1" means the record is available and a "0" means the record is in use. Since there are 203*10*4=8120 records per disk, 8120/16=507.5 words are required for the DSKRAT. The DSKRAT is recorded as a file on the disk of length 512 words and its first record is on cylinder zero, track zero, sector zero. All user file directories are allocated between records 0 and 117 octal. Records 120 to 1177 octal are reserved for TSDOS core memory storage. Records 1200 and up are allocated to regular DOS files.

3. Internal File Structure

An ordinary file is a linear array of words whose content is independent of the file system. However, the rest of DOS and the user may imply a certain format for a particular type of file. There are in fact four types of files in wide usage on DOS as follows:

1. card image
2. card image compressed
3. relocatable binary
4. saved memory image

The *Line Image* is an array of 60 word blocks containing 119 ASCII characters representing a *line*. The *Line Image compressed* file is similar except that blanks are compressed by the relative-horizontal-tab (221) convention, and lines are terminated by a new-line character (212). The relative-horizontal-tab convention replaces consecutive blanks with a half-word "221" and a half-word binary counter of the total number of blanks. The *Relocatable Binary* file consists of blocks of object code preceded by a block word count (a count of zero serves as an end-of-file mark). The *Saved-Memory-Image* format consists of a nine-word header block followed by a direct transcription of a memory image. The header block contains starting address (SA), ending address (EA), program counter (PC), A-registers (AR), B-register (BR), index-register (XR), status key word (KEYS), checksum of file (CKSUM), and a spare word.

4. Multiple Disk Organization

DOS can handle up to four disk drives. Each disk pack contains its own Master-File-Directory and DSKRAT. All files in a user file directory reside on that disk and all

user files directories in the MFD reside on that disk.

After loading disk packs on the drives, the user should have the paper tape DOS BOOTLOADER boot the operating system from the master disk pack. The operating system is restored from physical drives 0-3 depending on sense switches one and two as follows

sense switch 1	sense switch 2	disk
down	down	0
up	down	1
down	up	2
up	up	3

After boot loading the system, a user tells the system how many disk drives are in use and specifies a logical-to-physical ordering by means of the STARTUP command. Thus the command

```
STARTUP 2 0 1
```

associates physical drive 2 with logical 0 (master), drive 0 with logical 1 (first alternative) and drive 1 with logical 2 (second alternative). This association determines the order DOS searches MFD's in attempting to ATTACH to a directory (e.g. ATTACH UTIL will attach to UTIL on logical device 0, here physical 2. If there were no UTIL on that disk, it would look to drive 0, then drive 1). To override the search order in the STARTUP command, supply the logical drive desired as a numerical argument. Thus

```
ATTACH UTIL 2
```

will attach to UTIL on physical drive 1 in the above circumstances. If UTIL is not on that drive, no other drives are searched and an error message is given.

External commands such as ED, LDR, and FTN which are core image files, are resumed from the CMDHCO file directory of logical drive 0. Thus only one copy of the system commands must be stored on the set of disk packs, on the master pack.

If only the master disk pack is in use on physical drive 0, the appropriate startup command (STARTUP 0) need not be given when the user boot loads the system.

A user may have more packs than drives, in which case he may wish to unmount one or more packs and mount others. All packs may be unmounted. The sequences of commands is

CLOSE ALL

followed by the appropriate STARTUP command for the new configuration of packs given before the disk pack or packs are unmounted.

5. File System I/O

Although the file system can contain an indefinite number of files, only eight may be active at any one time. A file becomes active when you connect it to a UNIT represented by the integers 1-8 which functions as a port to the system. One file at a time can be assigned to each unit. At the same time, the mode for processing the file is specified (read, write, read and write). Files may be active on several disks at once. The transfer of data to and from the file through a unit is accomplished by READ and WRITE routines. An existing file may be selectively overwritten or extended but cannot be shortened except to delete the file from a ufd.

Many DOS commands refer implicitly to specific units, e.g., in DAP assembly the command "INPUT" opens unit 1 for the input file, "BINARY" opens unit 3 for the output of the compiler (i.e., the object file), and if "LISTING" is requested the listing file uses unit 2. Other commands, notably "OPEN", allow the user to assign a file to the unit of his choice. A user should not use unit 8, which is used extensively by DOS for reading and writing of system files such as DSKRAT or user file directories.

User programs may call on the file system to manipulate files. These routines are described in detail under section X.B.3 on DOS Interface Routines of Fortran Library Extensions.

B. DOS Core Usage

DOS is a core resident operating system. The DOS BOOTLOADER paper tape restores the core image file *HDOS from the DOS16 file directory into locations 67000 to 77776 octal. Locations 60200 to 66777 are used for eight DOS buffers, allocated downward from 67777. One buffer is allocated for each active unit. Normally, only one or two buffers are used.

When a user is loading programs, the loader must be resident. The loader occupies locations 57000 to 63777 during this time, leaving locations 1000 to 57000 for user programs and common. If room is short, a special loader HLDR will allow common to overlay the loader.

The machine language debuggers TRACE and DEBUG occupy 6400-64777 when they are invoked. If many files are open, these programs may be wiped out by DOS buffers.

Nearly all other external commands use core starting at location 100.

C. DOS Command Structure

There are two levels of communications between a console user and DOS. The user can either communicate with the DOS supervisor or with a program currently being run. In the supervisor mode, all teletype input is interpreted as supervisor commands. In the latter case, the teletype input is read as data by the program and can be interpreted by the program as desired. Under TSDOS, the one exception to this is the CONTROL S or X - OFF character which is always interpreted by the supervisor as a QUIT character. This character will immediately terminate the processing of any user program, type QUIT on the user console, and place the user back into supervisor mode. Under DOS if a user program gets in an infinite loop or halts the machine, the user may restart DOS by starting execution from the push-button console at location 70,000 octal (See DDP 516 Users Guide for details on how to operate the console). Upon completion of a user program or an external command (e.g., FTN, ED, etc. which invokes a stored system program) the user is always returned to supervisor. At this level the user communicates directly with DOS.

The DOS command language is simple in format. The user types the command and any arguments on a single line. The command and each argument is separated by at least one space, and the final character for the command line is the carriage return. When a user hits the carriage return, DOS analyzes and executes the command. If there are any errors, DOS will return an error message indicating the problem. Before hitting the carriage return, the user can delete an entire line by typing a question mark ("?"). The syntax for a DOS command is:

command-name (name - 1) (name - 2) (par - 1) ... (par - 9)

The names are alpha-numeric strings beginning alpha but only the first six characters are used. The command name is required but the other two names may be absent, in which case they are taken to be six spaces. The parameters are octal and if absent are taken as zero. The last six digits are used for each parameter. Blank lines are ignored.

Following are the DOS commands. Elements in CAPITAL

LETTERS are command names. Elements in lower case immediately following are arguments. Elements in square brackets [ ] are optional; elements within braces { } are alternatives of which one must be chosen; and an elipsis ("...") indicates that the preceding element may be repeated. In addition, the letters which compose the abbreviation for each command are underlined. A* indicates this is a TSDOS command only.

D. Internal Commands

ATTACH ufd password disk

ATTACH attaches the specified ufd if the passwords match; ufd is the name of the user file directory. If disk is blank, ufd is searched in order on logical drive 0, then 1, 2, and 3 depending on the number of drives assigned by the STARTUP command. If disk is a number 0-3, the search order is overridden and only logical disk disk is searched for ufd. When a user is not attached to a directory, the user cannot use the file system. For system user use, the universal password may be used.

LISTF

LISTF types out the names of all the files which are listed in the current directory, i.e., the user file directory to which the user is currently attached.

OPEN name unit status

OPEN opens the specified unit (integer 1-8) and associates it with the specified file name. It also sets the status of the unit to indicate reading (status of 1), writing (status of 2), or both (status of 3).

CLOSE ... [name] [$unit_1$][$unit_2$] ...[$unit_9$]

CLOSE closes the named files and specified units. CLOSE ALL closes all files and units.

INPUT name

INPUT opens unit 1 to read a source program from the file name. Is same as OPEN name 1 1.

LISTING name

LISTING opens unit 2 to write a listing file named name. This file is entered in the current ufd as a file named

name. Is same as OPEN name 2 2. This command is used to receive a listing file from DAP assemblies.

BINARY name

BINARY opens unit 3 to write an object file named name. This file is written in the current ufd. Is same as OPEN name 3 2. Command is used for DAP assemblies.

SAVE name sa ea [pc] [a] [b] [x] [keys]

SAVE save core memory from sa (starting address) through ea (ending address) as a file named name in the current directory. Also saved with the same file are if specified pc (program counter), a (A-register), b (B-register), x (X-register), and the keys. These values are used to initialize the register settings when this file is RESTORE'd or RESUME'd. If any of the optional parameters are not specified, the register values saved are those previously set in the DOS vector RVEC. RVEC is set by the commands SAVE, RESTOR, RESUME, START and when programs exit to DOS. RVEC can be examined by the PM command.

RESTORE name

RESTORE restores a disk-resident file named name by reading it into primary storage of the users segment from sa to ea values saved with the file. It also stores the saved values of the registers into the DOS vector RVEC to be ready for a START command.

START [pc] [a] [b] [x] [keys]

START loads the specified register values from the command line if specified or from the DOS vector RVEC and starts execution at the current location of the pc (program counter).

RESUME name [pc] [a] [d] [x] [keys]

RESUME is the equivalent of a RESTORE name and START. The file name is read into the user segment of primary storage and the registers are reset to the saved register values or to the parameter values if typed.

DELETE name

DELETE frees the disk storage space used by the file

name and removes the specified name from the current directory.

PM

PM types out the contents of the DOS RVEC vector, namely the starting address, ending address program counter, A-reg., B-reg., X-reg., and KEYS on the user console. PM is a way to find the save parameters of a file just restored.

*ASSIGN device console

ASSIGN assigns a device (i.e., DISPLAY, only device at present time available for assignment) to a user console if the device is currently unassigned (e.g., not being used by another user). The console parameter need be specified only if the command is given by the supervisor console.

*SWITCH device

SWITCH logically sets program variables necessary to turn ON or OFF a physical device (i.e., display).

COMINPUT {filename} {TTY} {CONTINUE}

COMINPUT allows users to prepare a list of commands with the Editor, file it on the disk, and have DOS read commands from this file rather than from the teletype. The command COMINPUT "filename" causes DOS to take subsequent commands from "filename." The last command in file "filename" should be COMINPUT TTY, which tells DOS to take subsequent commands from the teletype. Example: using the Editor, a user creates a file PMLIST which consists of the lines

```
PM
LISTF
COMINPUT  TTY
```

When the user types

```
COMINP  PMLIST
```

DOS types back

```
OK; PM
P=num A=num B=num X=num K=num
OK; LISTF
```

```
list of user files typed out
OK; COMINP  TTY
OK;
```

To have DOS type the results of the commands PM and LISTF, the user can type one command to DOS instead of two.

DOS reads commands from "filename" by opening unit six, reading then executing one line at a time. When the command COMINP TTY is encountered, DOS closes unit six and takes subsequent commands from the teletype.

Several other actions besides COMINPUT TTY can cause command input to be switched to the teletype. Any DOS error message will cause this as well as CNTRL X-OFF, the TSDOS "quit" character. In these two cases however, the command input file is left open. A user may retype the command that caused the error message then continue reading from the command input file by typing

```
COMINPUT  CONTINUE
```

Do not use the command CLOSE ALL in a command input file. This will close the command input unit and cause the message "file not open for reading."

COMINPUT affects only commands to DOS, not all teletype input. Teletype input to subsystems such as the Editor or the Binary Editor is not affected.

This command is useful for updating large programs which consist of many files, use several library files or require special loading procedures. For example, suppose a user has a program consisting of three FORTRAN source files MAIN, SUB1, and SUB2. This program requires two libraries, GRALIB and FTNLIB. A user makes up the following command input file DPROG

```
FTN  MAIN
FTN  SUB1
FTN  SUB2
CLRCOR
LDR  B←MAIN
START B←SUB1
START B←SUB2
ATTACH U
START GRALIB
START FTNLIB
ATTACH ufd
```

```
START 63002
CLOSE 1
COMINPUT  TTY
```

A user may then make editing changes to his programs, then COMINPUT DPROG causes his programs to be compiled, loaded and load map to be printed out. The user can now SAVE his program. The file DROG serves as documentation of the source files that make up the program and loading procedure for the program.

UPDATE

UPDATE writes onto the disk the current directory and the physical device record availability table (RAT) if there have been any changes to the files. It is not necessary under ordinary circumstances to use this command as DOS automatically does this when the directory or RAT is changed.

STARTUP diska [diskb] [diskc] [diskd]

The STARTUP command has been discussed in detail under section A4 of the chapter, Multiple Disk Organization. STARTUP initializes the configuration of disk drives DOS sees. STARTUP should be used sparingly, only when loading the DOS system at the beginning of the day or when changing disk packs.

The parameters must be integers 0-3. STARTUP connects physical disk diska as logical disk 0, physical disk diskb as logical 1, physical disk diskc as logical 2 and physical disk diskd as logical 3. The number of parameters given indicate to DOS the number of logical drives assigned to the system.

The STARTUP command need not be given if the user is only using one disk on physical drive 0.

If DOS has been used after initial loading, the STARTUP command must be preceeded by CLOSE ALL.

E. External Commands

External commands are actually system programs. When a user types an external command the program by that name is resumed from the CMDHCO file directory. External commands have been grouped into General, Utility and I/O commands.

1. General Commands

BASIC

BASIC invokes the Honeywell BASIC interpreter. See the Honeywell BASIC manual for details.

CRIBBAGE

CRIBBAGE is a program that plays Cribbage with the user.

DAP [param1] [param2]

DAP invokes the DAP assembler to assemble a source program. DAP expects that input and output files have already been opened with INPUT and BINARY commands, respectively. DAP also leaves these files open when it returns to DOS. For more details, see Chapter XIII.

DEBUG

DEBUG invokes the interactive debug package. DEBUG is useful as a program debugging aid. DEBUG lets the user dump core, make on-line in core modifications, take break points, and utilize other debugging features. For a detailed description of DEBUG see Chapter XVII, Section A.

ED [name]

EDIT involves the interactive text editor. For a description of editor commands and use of the editor, see Chapter XV.

EDB[name1] [name2]

EDB invokes the interactive binary editor. EDB works with input file name1 and optional output file name2. For a description of EDB see Chapter XVI.

FILED [name]

FILED invokes the interactive file-to-file text editor. See Chapter XV for a description.

FTN name [param1] [param2]

FTN invokes the FORTRAN compiler to compile a source program. FTN compiles name with binary output files B←name (truncated to six letters) and types lines with

errors along with error messages on the teletype. Files are automatically opened and closed. For more detail including options see Chapter IX.

HAMBUG

HAMBUG invokes an interactive debug package. HAMBUG enables the user to utilize debugging features similar to debug. For a detailed description of HAMBUG see Chapter XVII, Section B.

HLDR

HLDR invokes a version of interactive loader to load one or more object files. This loader loads COMMON higher in memory than the normal interactive loader LDR and must be used with caution. For a description of the loader, see chapter XIV.

LDR name

LDR invokes the interactive loader to load one or more object programs. For a description of the loader see Chapter XIV.

TRACE

TRACE invokes an interactive debug package. Like DEBUG and HAMBUG, TRACE enables the user to utilize a powerful set of debugging features. For a detailed description of TRACE see Chapter XVII, Section C.

XREF

XREF generates a cross reference index to symbols in a DAP source file. This index is useful as an addendum to DAP listing files. To use, type

```
user:       INPUT "DAP source file"
response:   OK,
user:       LISTING name
response:   OK
user:       XREF
response:   GO
            OK,
user:       CLOSE ALL
response:   OK
```

2. Utility Commands

ALOOK name

ALOOK displays source file name on the ARDS storage display. Hitting any character on the ARDS keyboard except Q will display the next page of text. When the last page has been displayed, two characters will close the file and return control to DOS. The character Q will cause this action at the end of any page.

BOOT

This command will load a new, initialized copy of the DOS system. This command should be typed only after the CLOSE ALL command has been typed. This command is equivalent to loading the BOOTLOADER paper tape in the reader and starting at location 1. Sense switches 1 and 2 control which disk DOS is loaded from as follows

sense switch 1	sense switch 2	disk
down	down	0
up	down	1
down	up	2
up	up	3

CLRCOR

CLRCOR zeros locations 20 to 64777 octal. CLRCOR takes up locations 65000-65400 octal CLRCOR is used to zero core memory before loading.

COPY afile bfile

The command

COPY afile bfile

copies afile to bfile. bfile does not have to exist at the time of the copy; a new file will be generated if necessary. To copy a file from one directory to another, use the following procedure

```
user:      COPY afile bfile 1000
response:  GO
           OK
user:      ATTACH ufd password
response:  OK
user:      START
response:  OK
```

This command is quite useful in moving files from one disk to another, as the directories may be on different disks.

CREATE ufd disk

This command adds user file directory ufd to the Master File Directory on logical drive 0-3 if disk is blank, ONE, TWO, or THREE respectively. The new ufd contains no files and a blank password.

DLISTF

This command will display your file directory in alphabetical order on the IDI display. Files may be deleted by pointing the light pen at the names of the files. The names are removed from the main list and put in the files-to-be-deleted list. Typing YES on the console will delete those files in that list. To return to DOS, toggle sense switch 1.

LOOK name [csize]

LOOK invokes a program that will display the source file name on the display. If csize is omitted the file will be displayed using medium size characters, optionally, S, L, and X use small, large, and X-large characters, respectfully. Hitting any character except Q will display the next display page full of text until the entire file has been displayed. Two additional characters will close the file and return the user to the supervisor level. Q will close the file and return control to DOS.

PAUSE

PAUSE is a command useful for giving directions to users from within a command input file. PAUSE simply waits until a character is typed on the teletype, then returns to DOS. Example:

Command input file contains

```
PAUSE RUN/LEADER/ON/PUNCH
PALAP 0 100 2000
COMINP  TTY
```

PASSWORD [name]

PASSWORD changes the password of the current user file directory to name. PASSWORD with no argument will change

the password to blank.

RENAME afile bfile

RENAME will change the name of afile to name bfile.

3. I/O Commands

CARDIN name

Cards may be read into the DDP-516 by means of the H632 card reader. A user should be familiar with the H632. Place the cards in the card reader and ready it. Using the DOS32 operating system on the H632, type the command CARDIN. On the DDP516, type CARDIN name. Cards should be read into file name on the 516. When done, the 516 must be restarted at 70000 octal and the CLOSE ALL command given.

DUPE

DUPE will duplicate a paper tape. To use:

1. Put the paper tape in the reader

2. "DUPE" Response- GO
   LC (if loading is complete)

3. Place the tape in the reader again, put sense switch one up and push the start button. The response is VC if verify is complete. If the response is not VC, redo this step or go back to DOS and start from the beginning with step 2.

4. Put sense switch one down and push start to punch a tape.

   Response- PC

5. Put the new tape in reader, put sense switch one up, and push the start button to verify the new tape. The response is VC if the verify is complete.

6. To punch another tape, put sense switch one down and push the start button. The response should be PC.

LPOUT name

Listings may be generated on the H632 computer line printer. A user first types the command LPOUT to the

DOS32 operating system of the H632. Then he types LPOUT name on the 516. File name is sent to the line printer. At the end, EOF READING message is given and file unit 1 is left open.

PALAP 24000 sa ea [pc] [ounit] [bootstrap]

PALAP provides a user with the ability to punch in invisible format, self-loading paper tapes of a segment of memory sa through ea,. For details see Chapter XIX, Section E.

RDIBM

RDIBM reads the next file from magnetic tape unit 0 onto the file open or unit 2 for writing RDIBM asks the user if he wants to read or ship the next file and whether or not the user wants to include sequence numbers in his file.

RDTAP

RDTAP does the same thing as RDIBM except reads in Honeywell rather than IBM format for magnetic tapes.

SKIP

SKIP is a command to skip any number of files forward or backward on a magnetic tape. This command is used in conjunction with RDTAP or RDIBM or WRTAP or WRIBM. Example:

```
user:      SKIP
response:  SKIP X FILES
user:      5,
response:  tape skips 5 files on the magnetic tape
           OK
user:      SKIP
response:  SKIP X FILES
user:      -5,
response:  tape backs up five file marks
           OK
```

If you have just read a file you have just read a file mark. To back up to the beginning of the file you must skip backwards 2 file marks.

WEOF

Write one end-of-file mark on unit 0.

WRIBM name

File name is opened on unit 1 and written in IBM format on magnetic tape unit 0. Another file may be written once WRIBM is in core by typing START filename.

WRTAP name

WRTAP does the same thing as WRIBM except in Honeywell rather than IBM magnetic tape format.

VI. TSDOS SYSTEM

A. Introduction

The Time Shared Disk Operating System is an operating system very similar to DOS which allows up to five users to run simultaneously. TSDOS is run from 2 p.m. to 5 p.m. daily, if there is demand for it.

Under TSDOS, all commands have the same name and format. The file structure is identical and the user invokes the same subroutines to communicate with the teletype, the disk, the display and light pen. Other I-O devices are not available to the user under TSDOS, but he may prepare a program which uses these devices, save it on the disk, then later run the program under DOS. The user actually has more memory available for programming under TSDOS because the user sees an empty 32K of memory whereas under DOS, he must share this memory with the operating system.

For a complete description of TSDOS, see DDP-516 TSDOS REFERENCE MANUAL by M. Laurence Liebron.

B. User Operation Under TSDOS

To use this system a user signs up as usual on the sign up sheet. Up to five people are allowed to sign up for the same block of time. To run under TSDOS, a user goes to the teletype room, which is the room in back of the air conditioner, sits in front of one of the five teletypes and follows the directions taped on the right side of the teletype. These directions tell the user how to connect the teletype to the TSDOS system. Do not use the 516 console teletype: it is not available under TSDOS.

Teletypes under TSDOS operate in the full duplex mode. That is, every character you type is read then echoed by TSDOS. If you type a character and get no response, it means your teletype input buffer is full and TSDOS or your program is not ready to accept any more data. The character you typed did not get read by TSDOS. After a while, try typing the character again. If still no response, it means your program is in a loop or TSDOS has crashed. To get out of an infinite program loop type CNTRL X-OFF.

If a user program tries to execute a halt instruction, TSDOS will return control to the supervisor and the message "PROGRAM HALT AT LOCATION" will be typed out. If a user program tries to execute an instruction that does not exist or an I/O instruction to a device not implemented under TSDOS, the message "ILLEGAL INSTRUCTION AT LOCATION" is typed.

To stop a runaway program or long LISTF, or long load map type CNTRL X-OFF, that is, push the CNTRL and S key simultaneously. Response is "QUIT" on teletype. You are now talking to TSDOS. The command PM (for post mortum) will type the contents of the program counter, A-reg., B-reg., X-reg., and KEYS on the user console. A PM after CNTRL X-OFF is equivalent to stopping your program by turning to the control panel and pushing the run switch to SI and examining the registers.

Two users cannot be attached to the same directory at once or have the same file open at once. So get in and out of the U directory as fast as possible. If someone is using the U directory when you want to use it, you will get "UFD IN USE: error message. Wait a minute or two and try again.

A LISTF stops other users from running while names are being typed out, so use LISTF sparingly.

To use the display and light pen under TSDOS, keep in mind:

A. The display buffer is half the size under TSDOS as under DOS. Extra information is simply lost and does not cause DSP ERR 3.

B. Only one TSDOS user may use the display at once. When done, type the command SWITCH DISPLAY OFF.

C. User may use both IDI displays at once and the color display under TSDOS.

D. Use TSGLIB, not GRALIB. TSGLIB takes up more space than GRALIB, so some programs may have difficulty in loading. If you load your program with TSGLIB, it will run under TSDOS or DOS equally well.

E. Under TSDOS, DSPBUF does not contain the users display buffer. The buffer is in a special supervisor area which the user cannot access.

All commands involving the ARDS, paper tape, magnetic tape or the H632 are not available under TSDOS. In addition, commands CREATE, DLISTF, PASSWORD, RENAME and XREF are not available. The overlay feature and editor display feature are not available. It is not possible to generate FTN listing files under TSDOS.

VII. DDP516-H632 COUPLER

A. Introduction

The COUPLER is a Honeywell option that allows high-speed transfer of data between the DDP516 memory and the H632 memory. The H632 is a powerful Honeywell computer located in the same room as the GOTS DDP516. See Honeywell reference manuals for programming information and DOS32 Users Guide for information on how to run programs. The COUPLER transmits information about 100,000 16 bit words per second. The COUPLER is used to send or receive information which must use the H632 line printer or card reader. See Chapter XIX for a description of how to do this. The COUPLER is also used to allow two computer programs, one in each machine, to communicate, and if requested run in parallel. This procedure allows programs too large or too slow for one machine to be run using both machines.

B. Functional Description

The COUPLER operates in a half-duplex mode. It is connected to the DDP516 I/O Bus or a DMC sub-channel. The other end is connected to the IOP of the H632.

The COUPLER has a 16-bit data buffer register for storing a 16 bit data word when data is being interchanged between the two computers. The data may be packed (or unpacked) into (from) the H632 memory, either one-half word to a memory location or two-half words to a memory location.

Either computer can establish a data path and initiate the transfer of data in either direction. The COUPLER is a symmetrical device (slave/slave) and it works in a first-come first-served basis. It can also be operated with the use of software to work in a master/slave relationship to their respective computers.

The data transfer is in asynchronous mode, on a request response basis at a rate determined by the slower data channel. Data transfers will continue until one of the computers issues a STOP command.

A complete description of instructions for the coupler can be found in Coupler Option Manual.

C. 516-832 COUPLER PACKAGE

A number of programs have been developed for transmission of binary and symbolic files[1] over the GOTS-832 and TAG-832 couplers and for conversion of binary files from 516 to 832 format and vice-versa. Symbolic files are automatically converted by the transmission program. (Note: Line lengths for symbolic files are currently limited to 80 characters.) Subroutines for file transmission can be used for communication between user programs on the 516 and the 832.

1. FILE TRANSMISSION PROGRAMS

a. TAG SYSTEM

516 Program (binary)[2] - FTRANS
832 Program (load module)[3] - TAGFIL

TAG 516 operation - enter:
EX FTRANS
the program then asks for the following parameters:

- File Type - enter "S" for symbolic file; enter "B" for binary file.
- Transmission Type - enter "S" to send file to the 832; enter "R" to receive a file from the 832.
- File Name - enter the name of the 516 file. (If a binary file is to be sent to the 832, the first record address must be entered instead of the file name.

  messages: "NORMAL END OF TRANSFER"
  "LINE OVER 80 CHARACTERS"

- If a line in a symbolic file exceeds 80 characters, the program is aborted.

---

[1] All files in the 516 and 832 are "binary files". "Symbolic files" are a special subclass of files made up of lines of characters. Typically, symbolic files are used by the editors and are given as input to the compilers and assemblers. On the H-832 and the GOTS 516 symbolic files are read and written in "line mode" (see DOS manuals).

[2] In directory 1224.

[3] This program resides in COMDIR.

H-832 operation - enter:
  TAGFIL
The program asks for the following parameters:

1. File Type - enter "S" or "B".

2. File Name

3. Transmission Type - enter "S" or "R".

Note: If "R" is specified and a file by the given name already exists, the user is given the option of terminating the program.

messages: "NORMAL END OF TRANSMISSION"

Note: A file was transmitted successfully only if both computers indicate normal end of transmission.

b. GOTS SYSTEM

516 Programs[4] - FSEND (516 to 832 transmission)
FRECV (832 to 516 transmission)
832 Program[5] - GOTSFIL

516 operation - enter:
  FSEND file name
  or
  FRECV file name

The programs will ask for the file type, that is "S" for symbolic file and "B" for binary file. Be sure to follow this parameter with a carriage return.

messages: "NORMAL END OF TRANSFER"

832 operation - for both sending and receiving, enter:

GOTSFIL

---

[4] These load modules reside in the command directory.

[5] This load module resides in COMDIR.

The program asks for the following parameters:

1. File Type - enter "S" or "B"

2. File Name

3. Transmission Type - enter "S" to send a file to the 516; enter "R" to receive a file from the 516. If "R" is specified and a file by the given name already exists, the user is given the option of terminating the program.

messages: "NORMAL END OF TRANSMISSION"

Note: A file was transmitted successfully only if both computers indicate normal end of transmission.

2. DATA TRANSMISSION SUBROUTINES

Subroutines can be called by 516 and by 832 programs to transmit data over the coupler. The calls must be coordinated so that when one computer sends data the other will receive data. A program that calls a coupler routine on one computer will wait until the program on the other computer calls a coupler subroutine. After a successful transmission, both subroutines will return to their calling program. Note that if the H-832 sends n 32-bit words, the 516 will receive 2n 16-bit words. Similarly if n 16-bit words are sent from the 516 the 832 will receive n/2 32-bit words.

The user is responsible for any formating of words needed by either machine. See section D for a discussion of data conversion.

a. 832 SUBROUTINES - The FORTRAN library contains the coupler subroutines CPXSN1 and CPXRC1 (for TAG 516) and CPXSN2 and CPXRC2 (for GOTS 516).

- Sending to the 516
  CALL CPXSN1(N,ARRAY, IRET)
  N - data length (number of 32 bit words)
  ARRAY - data
  IRET - return code (see section E for a discussion of codes).

  Note: Use "CPXSN2" for GOTS 516.

- Receiving from the 832
  CALL CPXRC1(N,ARRAY,LIM,IRET)
  N - on return N is the data length in 32 bit words.
  ARRAY - array that will contain the transmitted data.
  LIM - maximum number of words to be transmitted. Transmission will be stopped after LIM words are received.
  IRET - see above.

  Note: Use "CPXRC2" for GOTS 516.

b. TAG 516 SUBROUTINES - The file "S:CPLR" in directory 1224 contains the symbolic code for transmission. The assembled version requires 176 (OCTAL) locations.

- To send data to 832 -
  Entry Point - "CPSX"
  A register - address of a message
  B register - number of 16 bit words in the message (must be even).

- To receive data from the 832
  Entry Point - "CPRX"
  A register (on entry) - buffer address
  (on return) - message length (i.e. number 16 bit words).

  Note: No limit is set on the message length.

NOTE: If SS1 is on, transmission will be via the I/O Bus rather than the DMC.

c. GOTS 516 SUBROUTINES - The 'COULIB' in the U directory contains the coupler subroutines "CPRCV" and "CPSEND".

CALL CPSEND(ARRAY,N)

ARRAY - an array containing the N words to be transmitted.
N - number of 16-bit words to be transmitted (N must be even).

CALL CPRCV(ARRAY,N)

ARRAY - On return, this contains the data transmitted.

N - On return, this contains a count of the number of 16-bit words that were transmitted.

Note: If SS2 is on, transmission will be via the I/O Bus rather than the DMC.

d. DATA CONVERSION - FORTRAN integers on the H-832 end on bit 30, while 516 integers end on bit 15. Thus a user must shift the 516 integers 1 bit left before use on the 832 and he must shift the 832 integer 1 bit right before use on the 516. Furthermore, an integer on the 832 takes up 32 bits while it only takes up 16 bits on the 516. If the 516 word is negative, it must be preceded by a word of all 1's to make an 832 word; if the 516 word is positive it must be preceded by a word of all 0's to make an 832 word.

Characters in the DDP-516 have bit 0 of each character on whereas in the H-832 the bit is off.

Floating point number take up 32 bits in both machines, but their representations are quite different. Two H-832 subroutines "FCONV" and "FCONX" will convert from 516 to 832 representation and from 832 to 516 representation respectively. These subroutines are in the FORTRAN library on the H-832. They are used as follows:

- CALL FCONV (A,B)
  A - DDP-516 flt. pt. number
  B - On return, the number converted to H-832 representation.

- CALL FCONX (A,B,IRET)
  A - H-832 flt. pt. number
  B - On return the number converted to DDP-516 representation.

  IRET = 0 normal conversion = 1 H-832 number was too large
  If the H-832 number is too small, a value of zero is returned in B.

NOTE: Some accuracy is lost in this conversion process.

e. COUPLER SUBROUTINE RETURN CODES

On every call, the coupler transmits two messages

(M1 and M2). M1 is 32 bits long and contains the length of the data message M2. One 32 bit status word is constructed from the status of the two transmissions.

- Bits 0-7 status of M2 transmission
- Bits 16-23 status of M1 transmission
- Bits 8-11 and 24-27 contain a code that indicates in what portion of the channel program an error was detected.

Codes:
0 - normal termination of channel program.
1 - incorrect initialization interrupt from 516 (16 sending).
2 - incorrect signal for termination of transmission from 516. (16 sending).
3 - same as 1 (32 sending).
4 - same as 2 (32 sending).

- All other bits are zero.

Refer to the attached excerpt from the coupler manual to interpert bits 0-7 and 16-23 of the status word.

Series 32 Status Bits

	Status Bits			
	0	1	2	3
No Power on 16 CPU	0	X	X	X
Normal (Output Mode)	1	0	X	X
Normal (Input Mode)	1	1	X	X
16 Requesting Service	1	X	1	X
16 Acknowledge 32 Request	1	X	X	1
		4	5	
Error		0	0	
16 Ended XFER (OCP Disable)		1	0	
Normal XFER END (IOT Stop)		1	1	
		6	7	
Last Word (I/O) (Bits 0-15)		0	1	
Last Word (I/O) (Bits 16-31)		1	1	

Examples: (The following are 32 bit hex codes that were actually generated by the coupler).

CF00CF00 - normal termination of 32 to 16 transmission
8F008F00 - normal termination of 16 to 32

transmission.

8E008F00 - 832 terminated transmission (from 516) because range specified in call to coupler subroutine was exceeded. All data was transmitted correctly. (Note: this bit pattern is not "legal" according to coupler manual).

C940CF00 - 516 terminated transmission from 832 because range specified in call to 516 coupler subroutine was exceeded. All data was transmitted correctly. Although this indicates that lost word sent was bits 0-15 of an 832 word, last bits received by 516 were 16-31 of an 832 word.

8B208F00 - transmission from 516 to 832 was probably normal. This status means that the signal generated by the 516 upon completion of transmission (OCP DISABLE) reached the 832 before the 832 issued an IOT STOP.

3. OPERATOR INSTRUCTIONS FOR 516-832 COUPLER USERS

a. Before using the coupler programs, make sure that both computers are powered-up.

b. It does not matter which computer begins execution of the coupler routing first.

c. If a coupler program does not terminate normally, you must push 'SYSTEM', 'CP', 'START' (in that order) on the H-832 and 'MASTER CLEAR' the DDP-516. It does not matter in what order these two corrective operations are performed.

d. If the coupler does not seem to work properly, try the corrective procedure outlined in 3. above and restart the coupler programs.

NOTE: Both DDP-516 couplers use channel 6 of IOP 1.

4. BINARY FILE CONVERSION PROGRAMS

"CNV32" and "CNV16" are two H-832 Programs which convert a binary file from 832 to 516 format and from 516 to 832 format respectively. These programs require, as input, two files called "pattern files". One describes the format of the input data file and the other describes the format of the output file (i.e. the file to be created by the program).

Thus, "ICF" and "ICSF" are equivalent. Similarly "ISF" and "IWF" are equivalent.

c. Output Codes

I - integer
F - floating point number
C - character
N - number (one byte)
S - insert "fill" character (see note below)
W - insert a word of zeros
B - insert a blank (one byte)
Z - insert a zero (one byte)
T - end of pattern (mandatory)

Note: The "fill" character is a blank if the last byte-command was "C" or "B" and a zero if it was "N" or "Z". Initially, the "fill" character is a blank.

d. Writing Patterns

Integers and parentheses may be used to specify repetition of single letter codes or sub-patterns. For example: "13FB2(N2(XI))T" and "IFFFBNZIZINZIZIT" are equivalent.

e. Rules

Patterns must end with a "T".

Pattern files must have eight characters (counting digits and parentheses) per line. The last line may have fewer than eight.

Input and output codes (other than skip characters and fill characters - S,W,B,Z) must match. The includes terminal characters "T".

Examples:

input pattern	output pattern	legal?
IFCBNT	WZIBFCNWT	Yes
INFBT	IZFBT	no

The patterns may be considered to define a block. An input file may consist of many such blocks, but no partial blocks at the end of a file are permitted.

a. PROGRAM OPERATION

To start the file conversion program, type:

CNV16

or CNV32

at the command level under DOS-32.

The Program will ask the following parameters:

- INPUT PATTERN FILE NAME
- OUTPUT PATTERN FILE NAME
- INPUT FILE NAME
- OUTPUT FILE NAME

b. PATTERN SPECIFICATION

1. Terminology

- Data files are made up of words. An 832 data file has 32-bit words, a 516 data file has 16-bit words. Both, of course, are stored on the 832 disk, with two 516 words to one 32-bit word on the disk.

- Words contain four bytes (832) or two bytes (516)

- There are four data types: characters, one-byte numbers, integers, and floating point numbers.

  characters and one byte numbers take up one byte in a file.

  integers take up one word in a file (i.e. 16 bits in a 516 file and 32 bits in an 832 file).

  floating point numbers take up one word in an 832 file and two words in a 516 file. (Note that a 516 file could contain 2 characters, a floating pt. number and an integer - in that order. This would be stored as two 32-bit words on the disk).

b. Input Codes

I - integer
F - floating point number
C - character
N - number (one byte)
S - skip next byte
W - skip next word
T - end of pattern (mandatory)

Note: When I, F or W are specified, the next full word will be picked up.

e. An expanded pattern (i.e. after removal of integers and parentheses) may not exceed 2000 characters.

5. MESSAGES

Self-explanatory messages are generated if any of the rules (section IV B.5) are violated. If a rule is violated, the program will be aborted. The message "NORMAL TERMINATION" is printed on the teletype after a successful run.

USE OF TAG 516-H832 COUPLER AND THE VECTOR GENERAL DISPLAYS

Due to certain hardware problems, the 516-832 coupler subroutines 'CPSX' and 'CPRX' will not work correctly while the VECTOR GENERAL displays are running. This memo describes the program modifications required to the DDP-516 coupler subroutines so they can be used in conjunction with the graphics language AGL. Basically, the coupler subroutines must ensure the following order of events:

1. Call to coupler subroutine on 516.
2. Request 832 service, receive 832 enable.
3. Scopes start, run through entire display file and then stop.
4. Disable Vector General interrupts.
5. Start coupler and run to completion.
6. Disable coupler.
7. Enable Vector General interrupts and start scopes up again.

PROGRAM MODIFICATION

The subroutines 'VGWT' and 'VG1' must be added. Certain constants like 'DRUN' in sector 20 refer to the program AGL.

SUBROUTINE 'VGWT'

```
DRUN EQU   '20777
DDD  DAC   DRUN  1 IF RUN
**
**
**WAIT FOR SCOPE TO START AND STOP
```

```
**
VGWT DAC     **
     INA     '1451
     JMP     *-1
     ALR     3      GET BIT 4 IN SIGN POSITION
     SPL
     JMP     VGW2   JUMP OUT IF BIT IS SET
     LDA*    DDD
     SNZ             1 IF RUNNING
     JMP     *-2
     LDA*    DDD
     SZE             0 WHEN HALTED
     JMP     *-2
```

SUBROUTING 'VGWT' (CONTINUED)

```
     JMP*  VGWT
**   IF SWITCH 4 IS SET THEN START COUPLER WHEN SCOPES START
VGW2 LDA*  DDD
     SZE          I IF RUNNING
     JMP   *-2
     LDA*  DDD    0 IF HALTED
     SNZ
     JMP   *-2
     JMP*  VGWT
```

SUBROUTINE VG1

```
IMSK EQU   '20776
MSKE DAC   IMSK
MCR  EQU   '22775
MC   DAC   MCR    MODE CONTROL WORD
     OCT   40005  LOAD MODE CONTROL WORD
HLTD OCT   1 MCR ITSELF
     OCT   130000  HALT DISPLAY
HLT1 DAC   HLTD-1
HLT2 DAC   HLTD+1
**
**
VG1  DAC   **
     STA   HLTD  SAVE THE MCR
     INA   '1451
     JMP   *-1
     ALR   3      GET SWITCH 4
     SPL
     JMP*  VG1 IGNORE HALTING PROCEDURE IF COUPLER AND SCOPES
**           START TOGETHER
     LDA   ='1000  CLEAR AND HALT DISPLAY
     OTA   '0153
     JMP   *-1
```

```
      LDA   HLT1   SET UP DMC
      STA   '26
      LDA   HLT2
      STA   '27
      LDA   ='176400   START THE DISPLAY
      OTA   '0153
      JMP   *-1
      LDX   =-10
      IRS   0
      JMP   *-1  WAIT FOR DISPLAY TO FINISH
      LDA   ='1000  VG IS AGAIN TURNED OFF, TO ENSURE
      OTA   '0153
      JMP   *-1
      JMP*  VG1   RETURN
**
** WE SET THE MCR TO EITHER DISALLOW ALL INTERRUPTS OR ALLOW
   THEM
**  AGAIN AFTER TRANSMISSION
**
```

The following changes must be made to the CPSN subroutine:

1. Replace the instruction:

```
    CPS2  OCP  '1175   SET COUPLER FOR INPUT TO 32 with:
    JST   VGWT         WAIT FOR VG TO START AND STOP.
    LDA   =1
    JST   VGL          DISABLE VG INTERRUPTS
    OCP   '1175        SET COUPLER FOR INPUT TO 32
```

and replace:

```
CPS9      OCP  '1775   DISABLE COUPLER
```

with:

```
CPS9      OCP  '1775   DISABLE COUPLER
          LDA* MC
          JST  VG1     RESTART SCOPES
          LDA* MSKI    RESET MASK
          SMK  '0020
```

The corresponding changes must be made in the subroutine 'CPRC' to the instructions labelled 'CPR2' and 'CPR9'.

VIII. GRAPHICAL CAPABILITIES

A. Refreshing Displays With Light Pen

The display hardware consists of a display generator and four separate display consoles. Three display consoles (supplied by Information Displays, Inc.) are high resolution and monochromatic. They are equipped with a light pen, and can be driven at full speed by the display generator. One display console (supplied by International Telephone and Telegraph) is of lower resolution, has three primary colors (red, green, and blue) but no character capability.

The software to run this equipment is described in detail in Graphics Reference Manual.

To run graphics programs, load GRALIB (TSGLIB under TSDOS) before loading FTNLIB.

B. Tablet

A Sylvania Data Tablet is available for graphical input. The following routines are available for the tablet. They are part of GRALIB. A tracking cross on any/or all of the displays which will follow the pen on the data tablet is displayed by

A. Call TCMODE (I1, I2, I3)

Where: I1, I2, I3 = 2 will put the cross on console 1, 2, 3 respectively

I1, I2, I3 = 1 will remove the cross from console 1, 2, 3 respectively

I1, I2, I3 = 0 will leave the condition on console 1, 2, 3 unchanged

E.G., to put the cross on console 1 & 3 and not on 2 - call TCMODE (2, 1, 2)

To now take it off console 3 and put it on 2 - call TCMODE (0, 2, 1)

TCMODE will put commands in the display buffer, and

calling CLRBUF will remove the cross. It is necessary to call TCMODE again after calling CLRBUF if you want the cross back.

B. Call TCREAD (IX, IY, IZ)

Reads the present data tablet pen position and returns it to the user.

IX, IY will be the X, Y location of the pen on the tablet (0-1023) IZ will be the height of the pen above the surface;

1 = on surface point pushed down
2 = just resting on surface
3 = somewhat above surface
4 = way above surface.

C. Call TABRD (IX, IY, IZ)

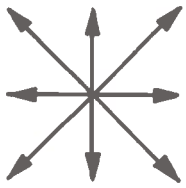
Arguments are the same as for TCREAD. This routine reads the data tablet pen position on a demand basis. The values returned will be those of the pen when called. The values returned from TCREAD come the last time the system interrogated the tablet, which occurs at the end of every display buffer refresh cycle.

C. Calcomp Plotter

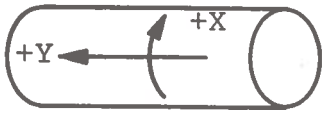
1. Hardware Description

The plotter is a 36 1/2 inch drum, incremental calcomp plotter, model 1136. Three pens are available.

In addition to pen up, pen down; plotting is down by use of the eight possible incremental vectors shown below;



The assumed direction are as diagrammed below;



The plotter is assigned device number '047, and commands are output to it by means of at OTA '047.

2. Software Description

To run Calcomp programs, load CCPLIB before FTNLIB. The subroutines available are described below.

1. PLTINT (IPEN) IPEN = 1, 2, 3 for pen to start with

This routine initializes the plotting system and establishes the origin at the present pen position. PLTINT should be called before any other plotter routines.

2. PLOT (X, Y, ±IPEN)

X, Y are the coordinates of the terminal position to which the pen is moved, in inches from the current reference point (origin). An origin may be established anywhere (on or off) the plotting surface as explained below for negative IPEN values.

IPEN = 2; the pen is down during movement (visible line)

IPEN = 3; the pen is up during movement

IPEN = -2, -3; a new origin is defined at the terminal position after the movement is completed as if IPEN were positive. The logical X, Y coordinates of the new pen position are set equal to zero, so that that position is the reference point for succeeding pen movements.

IPEN = ±12, ±13; X and Y are scaled and offset according to the values set up in a call to offset (explained below) by the equations.

$$X' = (X-Xmin)/XFAC$$
$$Y' = (Y-Ymin)/YFAC$$

The movement is then made as if IPEN were ±2 or ±3.

3. FACTOR (FACT)

The factor subroutine enables the user to enlarge or reduce the size of the entire plot by changing

the effective number of plotter steps per inch of page coordinates.

FACT is the ratio of the new plot size to the normal plot size. For example, if FACT = 2.0, all pen movements will be twice their normal size. When fact returns to 1.0 all plotting returns to normal size.

4. WHERE (X, Y, FACT)

This subroutine returns the current pen-position coordinates and the scaling factor.

X, Y are locations that will be filled with the current pen position coordinates resulting from the last call to plot (which may have been called by one of the system routines).

Fact is filled with the current plot scaling factor (the value supplied by a call to factor, or a 1.0).

5. OFFSET (XMIN, XFAC, YMIN, YFAC)

This routine supplies the system with values to be used with calls to plot where IPEN is 12 or 13 as explained under plot. Essentially, this performs user defined scaling and translation.

6. NEWPEN (N)

Cause PEN N (where N = 1 to 3) to become the current pen.

7. SYMBOL (X, Y, HGHT, IBCD, THTA, NS)

This subroutine produces plot annotation at any angle and practically any size. There are two symbol call formats:

(1) The standard call to draw text;
(2) The special call to draw special centered (and some not centered) symbols

Refer to Table 1 for legal characters and the special characters.

TABLE 1

A. LEGAL CHARACTERS & THEIR 8-BIT OCTAL CODE

CHAR.	CODE	CHAR.	CODE
@	300	SPACE	240
A	301	!	241
B	302	"	242
C	303	#	243
D	304	$	244
E	305	%	245
F	306	&	246
G	307	'	247
H	310	(	250
I	311	)	251
J	312	*	252
K	313	+	253
L	314	,	254
M	315	-	255
N	316	.	256
O	317	/	257
P	320	0	260
Q	321	1	261
R	322	2	262
S	323	3	263
T	324	4	264
U	325	5	265
V	326	6	266
W	327	7	267
X	330	8	270
Y	331	9	271
Z	332	:	272
[	333	;	273
\	334	<	274
]	335	=	275
↑	336	>	276
←	337	?	277

B. SPECIAL SYMBOLS AND THEIR DECIMAL EQUIVALENT

CENTERED

	0
	1
	2
+	3
X	4
	5
	6
	7
Z	8
Y	99
	10
*	11
	12
|	13
	14
-	15

NOT CENTERED

|	16
↓	17
≡	18
→	19
≥	20
∧	21
≠	22
±	23
≤	24

a. STANDARD CALL - SYMBOL (X, Y, HGHT, IBCD, THTA, NS)

X, Y coordinates in inches of the lower left-hand corner (before rotation) of the first character to be drawn. The pen is up during movement to this point. If X and/or Y is zero the last value calculated will be used. To use an actual zero for coordinates, use 0.001.

HGHT is the height in inches of the characters to be plotted. For best results it should be a multiple of 0.07, but other values are acceptable. The width of a character, including spacing, is normally the same as the height.

IBCD is the array (or single variable) containing the characters to be plotted, two per word, set up in a Hollerith statement or by A-format. If NS is 1, the single character must be left justified. If one character, right justified, is to be drawn, NS = 0 will do it.

THTA is the angle in degrees, at which the annotation is to be plotted (0 = parallel to the X-axis).

NS is the number of characters to be plotted.

b. SPECIAL CALL - SYMBOL (X, Y, HGHT, INT, THTA, NS)

X, Y, HGHT, THTA are the same as in the standard call. If the symbol is one that is centered, (INT. LE.15) then X, Y are the geometric center of the symbol. INT is the number (0-24) of the symbol to be plotted (see table). For INT between 0 and 15 the symbol is centered.

NS = -1 to have the pen up during movement to X, Y
= -2 to have the pen down during movement to X, Y

8. NUMBER (X, Y, HGHT, FPN, THTA, NDEC)

This routine is a pre-processor to the symbol routine. It converts a real variable to the appropriate string of characters so that it may be plotted by symbol in a FORTRAN F-type format.

X, Y, HGHT, THTA are as explained under the description of symbol.

FPN is the location of the number to be plotted.

NDEC is the number of decimal digits to be drawn (maximum of 7). A (-1) value will suppress the decimal point.

9. SCALE (X, S, N, K)

This routine examines the data values in an array to determine a starting value, either minimum or maximum, and a scaling factor, positive or negative, such that:

(1) The scale annotation drawn by axis at each division will properly represent the range of data values.

(2) The data points when plotted by line will fit in a given plotting area.

These two values will be stored at the end of the array.

The scaling factor that is computed represents the number of data units per inch of axis, but adjusted so that it is always an interval of 1, 2, 4, 5, or 8 x 10^N (where N is an exponent consistent with the original unadjusted scaling factor).

The starting value (which will appear as the first annotation on the axis) is computed as some multiple of the scaling factor which is equal to or outside the limits of the data in the array.

X is the array of the data points
S is the length of the axis to which the data is to be scaled

N is the number of data points. The array must be dimensioned at least two greater than N
K is an integer representing the repeat cycle of a mixed array (normally 1)

The minimum value and scale factor will be stored in X(N*K+1) and X(N*K+K+1) respectively.

10. LINE (X, Y, N, K, J, L)

This routine produces a line plot of the pairs of data values in the two arrays X, Y. The data points may be represented by centered symbols and/or connecting lines between points.

The minimum values and scaling factors must be supplied in the two arrays as described in scale.

X is the array of ordinate values
Y is the array of abcissa values
N is the number of points in the array
K is the repeat cycle (normally 1)
J is a number giving the following options

J = 0 line plot
J = 1 line plot with symbol at every point
J = 2 line plot with symbol at every other point
J = -1 symbol at every point, no line
J = -2 symbol at every other point, no line

L is the number of the symbol to be used (0-24) as given under the special characters in Table 1.

11. AXIS (X, Y, IBCD, N, SIZE, THTA, XMIN, DX)

This routine draws an axis for a graph, with a variable starting point, labeling either side, variable size, variable angle of inclination and variable scaling. When both an X and Y axis are required, this routine must be called twice.

X, Y are the coordinates of the starting point of the axis line. The axis should be at least one-half inch from any side to allow

space for scale annotation and title.

IBCD is the alpha-numeric title to be put with axis.

N is the number of characters in the title. Positive N puts the title on the counter-clockwise side of the axis. Negative N puts the title on the clockwise side of the axis.

SIZE is the length of the axis to be drawn. It should be a multiple of 10.0/DV.

THTA is the angle of the axis measure counter-clockwise from the axis.

XMIN is the value of the variable at the first point of the axis (normally defined by scale)

DX is the difference between the second and first values of the variable along the axis (normally defined by scale)

12. PCIRCL (X, Y, R)

A circle is drawn with radius R and center X, Y.

13. PBRIT (I)

This subroutine conditions drawing. I = 0 sets all plotting to invisible. I = 1 sets all plotting to visible. PLTINT sets plotting to visible.

14. PSTRCT (I)

This subroutine conditions drawing of lines. I = 1 indicates solid; I = 2 indicates dotted; I = 3 indicates dashed and I = 4 indicates dash-dot. PLTINT sets drawing to solid.

15. PCGLOB (A, B, C)

This subroutine specifies letter conditions. A specifies aspect ratio, B specifies spacing ratio, and C specifies angle of italilization. PLTINT sets A, B, and C to 1, 1, and 0 respectively.

16. ESYMB (X, Y, HGHT, IBCD, THETA, NS)

Has same action as SYMBOL, described above, except that character codes are interpreted as the IDI display character set. See Graphics Reference Manual for these characters and their codes.

D. ARDS Storage Display

A Computer Displays Advanced Remote Display Station model 100A is available for graphical output. This storage display may write characters at the rate of 100 characters per second. Lines are drawn in a special graphics mode by sending groups of characters.

The software to run the ARDS is described in The Basic Software System for ARDS and The Graph Plotting System for ARDS.

To run ARDS programs, load NARDLIB before FTNLIB.

E. Real Time Clock

CALL SETCLK (TIME)

TIME is a real variable representing the time in hours to which the user would like to set the clock. Normally, this would be called with TIME = 0.0 at the point where the user wants to start keeping time. It should be called at least once before calling CLOCK, since the clock routine may have random numbers in it to begin.

CALL CLOCK (TIME)

TIME is a real variable representing the time in hours, as updated by the real time clock. Again, to insure a clean reference point, SETCLK(T) should be called at least once before calling CLOCK. Calling SETCLK succeeding times will reset the clock to the argument and thus wipe out the previous reference point.

IX. FORTRAN

A. Introduction

A new version of the FORTRAN compiler has been installed in the DOS system. This memo tells the user how to run the compiler, what new features have been added, extensions to the FORTRAN library, and what bugs the compiler still has. For information on how to use FORTRAN programs see the FORTRAN manual for DDP computers.

B. How to Compile a Program

1. Normal Compilation

Assume you have a FORTRAN source program on file TEST in your directory. To compile the program make sure all units are closed, then type:

```
user:      FTN TEST
response:  GO
           CC, OK
```

FTN will read source statements from file TEST and write binary object onto file B←TEST. FTN generates the name it will use for writing binary data by concatenating "B←" to the beginning of the filename and truncating the resulting name to six characters. When FTN detects errors, the line causing the error is typed followed by the error type. When FTN reads the $0 line of the file TEST, FTN types "CC," for compilation complete then returns control to DOS. If the file TEST contains many FORTRAN subroutines, each subroutine should have a "$1" line after the "END" line with a "$0" line following the last subroutine. FTN will type "CC", following the compilation of each subroutine.

2. Options

a. In addition to compiling, generate a listing file with error messages; no errors on teletype. This option is useful if a user has a large number of errors in his program. The user can quickly scan the listing file on the display using the LOOK command. FTN will write the listing file onto a name generated by concatenating the letters "L←" onto the beginning of the

filename and truncating the name to six characters. To invoke this option type:

```
user:      FTN filename 1000 777
response:  GO
           CC, OK
```

b. In addition to compiling, generate a listing file (with errors) which also contains the symbolic assembly language compilation of each statement. This option is useful for debugging programs at the machine language level. To invoke this option, type:

```
user:      FTN filename 1000 40777
response:  GO
           CC, OK
```

C. New Features

1. Improved FORTRAN Input-Output

The following devices are available through FORTRAN READ and WRITE statements:

	device #
typewriter	1
paper tape	2
IDI display	3
disk	31 to 38
magnetic tape	5

Use of paper tape and magnetic tape is strongly discouraged, as these devices are not available under the time sharing system.

FORTRAN teletype input has been improved. If a user makes an error, the input line may be modified by the kill character "?" or the erase character ". The "?" character will erase the entire input line up to the "?". Typing the " character will cause the preceding letter to be deleted from the line.

Formatted teletype output has been improved. An imbedded 1H$ in a format statement will print a $; if 1H$ is at the end of the format statement, it will suppress a new line before the next write statement. "1H1" at the beginning of a write statement only prints a 1 in column 1 on the teletype. Simarilarly for 1H+, 1H␣, and 1H0.

Formatted input data from any device may be separated by a comma, with a comma also following the last item on a particular line. An input of 0 may be implied by two successive commas.

Example

```
          READ(1,10)I,J,K,L
10        FORMAT (4I5)
A teletype line of
          1,2,3,,
```

read by the above read statement will put 1 into I, 2 into J, 3 into K and 0 into L.

2. FORTRAN Input-Output to Disk

A user may prepare a data file to be read by the FORTRAN READ or WRITE statement. The user can do this by typing the data in using the editor (see Chapter II for an example) or by preparing cards or paper tape as a user would do when preparing a program for the DDP-516.

To read this data with a FORTRAN program, a user must call subroutine SEARCH to open the file for reading on a unit of the file system. The user then may read the data with a formatted READ statement using FORTRAN device numbers 31-38. When the user is done reading, he should call subroutine SEARCH to close the unit. Device numbers 31-38 correspond to units 1-8.

Example

```
          CALL SEARCH(1,6HTESTFL,1)
          DO 200 I=1,10
          READ(31,10)I,J,K,L
          WRITE(1,10)I,J,K,L
200       CONTINUE
          CALL SEARCH(4,0,1)
10        FORMAT(4I5)
```

This program fragment would read ten lines from file TESTFL typing each line on the teletype. The file of course, is expected to contain 10 lines of 4 integers each.

To write data on the disk, the user must call subroutine SEARCH to open the file for writing on a unit of the file system. The user may then write data with a formatted WRITE statement using fortran device number 32. When the user is done writing, he should call subroutine SEARCH to close the

unit. Example:

```
          CALL SEARCH(2,6HOUTPUT,2)
          I=1
          J=2
          K=3
          L=4
          WRITE(32,10)I,J,K,L
          I=10
          J=20
          K=30
          L=40
          WRITE(32,10)I,J,K,L
          CALL SEARCH(4,0,2)
10        FORMAT(4I5)
```

The above program fragment will write two lines on file OUTPUT. They will be

```
          1    2    3    4

          10   20   30   40
```

This file may be examined using the editor or examined on the display using the LOOK command. For more information on subroutine SEARCH see X.B.3. Lines in either input or output mode are a maximum of 119 characters.

The FORTRAN language does not specify what action occurs if an end-of-file is encountered during reading. If the user expects to encounter an end-of-file on disk, he should call either SETEOF(0) or SETEOF(1). CALL SETEOF(0) sets the processing of disk file end-of-file to normal. If end-of-file is encountered, EOF READING is typed followed by return of control to DOS. CALL SETEOF(1) sets processing of disk file end-of-file to ignore mode. In this mode, if end-of-file is encountered, a blank line is returned as the line read. The user must check after each READ statement if end-of-file was encountered by function IEOF. I=IEOF(DEVICE) returns value 1 if DEVICE (range 37-37) has just encountered end-of-file, otherwise IEOF returns 0.

Formatted file output has been improved as for teletype. An imbedded 1H$ in a format statement will print $; if 1H$ is at the end of the format statement, it will suppress a new line before the next write statement.

3. FORTRAN Output to the Display

Flexible output to the display is available through formatted input-output so that subscripted expressions are possible. First, directions for normal use will be given.

The user may treat the IDI display as if it were a teletype, with a few exceptions. The user must first call subroutine CLRBUF to clear the display. The position of the first line to be displayed must be set by a call to subroutine SETPT. The user may then use WRITE statements for FORTRAN device 3 and output lines will appear on the display one under the other. When the display becomes filled up, say about 50 lines, the user should program a delay so he can view the output. The user must then clear the display and position the initial line for the next page. Example:

```
          CALL CLRBUF(1)
          CALL SETPT(0,950)
          I=1
          J=2
          DO 10 K=1,50

          WRITE(3,20)I,J
10        CONTINUE
20        FORMAT(2I5)
          CALL T1IN(ICHAR)
          CALL CLRBUF(1)
          CALL SETPT(0,950)
          I=10
          J=20
          DO 30 K=1,20
          WRITE(3,20)I,J
30        CONTINUE
          CALL T1IN(CHAR)
```

The above program puts 50 lines on the display, each line composed of the integers 1 and 2. The program then calls subroutine T1IN which waits until a character is typed. The character is put in ICHAR and twenty lines are displayed, each line composed of the integers 10 and 20. The program then waits for another character to be typed.

The following are instructions for specialized output.

The user should follow the instructions given for normal usage. The user has the option of including special commands in his format statement.

1) The occurrence of a 1H$ at the end of a format statement will suppress the outputing of a newline.

2) The format buffer is emptied one character at a time. The presence of the character ! will cause all the following characters to be interpreted as control characters for the display, until the occurrence of the next ! at which point the format buffer is treated in the normal manner.

3) To put say three control characters in the buffer, the format statement should look like:

```
FORMAT(---LIST---,5H!ABC!,---REST--)
```

4) The user may set the default character size and escape character by a

```
CALL SETOAD(CHARSIZE,ESCAPECHAR),
```

where if either is a zero the normal default condition will prevail. These are: character size = 2, escape character = !.

5) The legal control characters are (illegal are ignored):

```
A->BACK SPACE IN THIS CHAR. SIZE
B->GIVE A NEWLINE IN THIS CHAR. SIZE
C->CLEAR THE DISPLAY ON PUT BEAM IN UPPER LEFT CORNER
D->GIVE ONLY A CARRIAGE RETURN
E->SET BRIGHTNESS TO HIGHEST (4)
F->SET BRIGHTNESS NORMAL (3)
G->HALF LINE FORWARD(TOWARD BOTTOM) IN THIS CHAR. SIZE
H->HALF LINE BACKWARD
I->HOR. TAB (5 SPACES TO RIGHT)
J->VERT. TAB (5 LINES TOWARD BOTTOM)
K->SET CHAR. SIZE TO 1
L->SET CHAR. SIZE TO 2
M->SET CHAR. SIZE TO 3
N->SET CHAR. SIZE TO 4
```

When loading programs which use display output, the Library GRALIB in the U directory must be loaded before the FORTRAN library is loaded.

4. INSERT Statement

A new statement has been added to the FORTRAN for the 516. The statement "INSERT filename" when read by the compiler during compilation will cause the text of file filename to be compiled in place of that statement. The INSERT feature is useful when one has a large number of subroutines each of which requires an identical common area. Rather than having the COMMON as part of each subroutine, the statement INSERT COMMON, for example, is put in the place where common should go and the file COMMON contains the common statements for each subroutine. Besides saving space on the disk and making program listings shorter, this feature allows easy modification of common, as it must be modified in only one place. Insert files may not contain insert statements.

5. Assigned GO to Statement

The syntax of an assigned GO to statement was

GO TO i,(K_1,K_2..K_n)

where i is an integer variable reference set by an ASSIGN statement to one of the K_n statement numbers. This list is now optional and serves no useful purpose, as the compiler does not check the k_n statement numbers. Below is an example of how the GO to statement may be used to transfer to a statement in another subroutine.

```
          ASSIGN 100 to I
          CALL B(I)
100       CALL EXIT
          END
$1
          SUBROUTINE B(IX)
          I=IX
          GO TO I
          RETURN
          END
$0
```

EXIT is a subroutine that returns control to DOS. The RETURN statement in subroutine B is never executed but is necessary for successful compilation.

6. Global Variable Mode

It is often convenient to extend the Implied Integer rule of all variables starting with I, J, K, L, M, or N, are Integer Mode unless specified otherwise' to say 'All variables are Integer Mode unless otherwise specified. This can now be done by a Fortran statement INTEGER followed by no variables. All other modes may be specified as normal in the same way.

Example 1: All variables except A, B, C1, C3, and X1 are Integer

```
INTEGER
REAL A,B,X1
COMPLEX C1,C2
```

Example 2: All variables are REAL except I1, I2, J, K, L1, L2

```
INTEGER I1,I2,J,K
LOGICAL L1,L2
REAL
```

Example 3: All variables are INTEGER mode

```
INTEGER
```

7. Octal constants

Example: I = 2Ø77

will set I to 77 octal or 65 decimal. The 2 specifies the number of octal digits and the Ø (not zero) indicates octal. Any digit can be used in place of the 2, as the compiler recognizes a digit followed by Ø as the beginning of an octal constant.

8. Intrinsic Functions

The following functions described in Chapter X B are compiled into in-line code rather than calls to functions
XOR(A,B)
AND(A,B)
NOT(I)
IABS(I)

The following functions are compiled into in-line code if the second argument is a constant or calls to the function if the second argument is a variable.

RS(I,J)
LS(I,J)
RT(I,J)
LT(I,J)

9. TRACE Debug Feature

The FORTRAN TRACE feature has been modified so logical if statements are processed. Complex and double precision variables can no longer be processed.

If a user loads a special library, TRCLIB before loading the FORTRAN library, an expanded TRACE feature rather than the regular TRACE package is loaded. This feature allows trace output to the teletype, display, ARDS storage display or disk. During execution, the trace routine asks the user which output device to use. Trace output is suspended while sense switch four is up. If the display is chosen, the screen will be filled with lines of output, type a "bell", then wait while the user examines the screen. A newline or carriage return causes the picture to "move up" one line. A vertical tab pushes the picture up six lines, whereas any other character causes the picture to move up 30 lines. In each cause, new output fills the remaining part of the screen.

If the ARDS is chosen, the screen fills up and the program waits while the user examines the screen. To continue, type any character on the ARDS keyboard.

D. FORTRAN Bugs

a. The compiler does not always print the error message immediately below the line with the error. If the previous statement does not seem to have errors, look several statements above.

b. If the user has one compilation error in his program, other error messages may appear below the initial error that are spurious error messages. If you can't find the error in the following lines, remove the first error and recompile.

c. A single common area labeled or not, or a dimensioned variable cannot be larger than 8192. No error message results but incorrect code is produced.

d. the program

```
I=0
GO TO (10, 20)I
```

When executed will go into an infinite loop. To allow generality for the GO TO statement, error checking has been removed. The user must check the range of the index on the computed GO TO statement.

e. To increase efficiency of integer multiply and divide, no error checking takes place. A multiply overflow looses the most significant bit and divide by zero is treated as a no-operation instruction.

f. The use of a comma to separate input data items will not work for either logical or A format. Also, beware of this feature! If you specify I5 for integer input then type 5 digits followed by a comma, the comma is seen as specifying a zero input for the following variable, not as a separator between the current and following variable. To solve this problem, specify 16 in your format statement. This rule follows for other format types also.

g. The octal and quoted hollerith string features do not work in all cases.

h. The global variable mode when used in a subroutine or function does not affect the arguments declared for that subroutine or function .

i. An expression which contains two function calls to the same routine may be optimized by the compiler. Example:

```
I= FUNCT(A) + FUNCT(A)
```

will be optimized to the machine language equivalent of

```
I= 2*FUNCT(A)
```

This optimization will cause a program bug if A is a return argument which depends on previous calls to FUNCT, as allowed by ASA FORTRAN.

j. BLOCK DATA subprograms, when loaded cause the error message MO unless BLOCK DATA is for blank COMMON only. BLOCK DATA subprograms with named COMMON therefore do not work.

X. FORTRAN LIBRARY

A. Introduction

The FORTRAN library as supplied by Honeywell has been altered and expanded to implement the new features of FORTRAN described in the last chapter. Section B of this chapter describes routines which do list processing, bit manipulation, and teletype I/O without FORMAT statements. Routines to interface to the disk operating system are also described. These subroutines have also been added to the FORTRAN library. Section C gives an alphabetical list and description of all subroutines of the FORTRAN library.

B. FORTRAN Library Extension

1. List Processing

The following features provided the basic building blocks for creating and manipulating and referencing list structures.

```
Z=ILOC(X)
```

Z is filled with the address of variable X.

LIST--The inverse operation of ILOC is performed by defining an array in block common to start at location 1. The present commands LDR and HLDR have been modified so that block common names and subroutine names are equivalent. Furthermore, the block common name LIST has been permanently put in the initial table of subroutine names with a load address of 1. A user wishing to use the LIST features must put a common area LIST in every subroutine using it. For example:

```
COMMON/LIST/LIST(2)
```

LIST may be used on the left or the right of an equals sign. For example:

```
LIST(29) = I
```

This will set location 29 to the current value of I.

```
ZA=Y
ZB=LIST(LOC(Y))
```

ZC=LOC(LIST(Y))

In the above, ZA=ZB=ZC.

The user must be careful not to specify the subscript of LIST as larger than 32,767, nor as negative.

2. Bit Manipulation

The following FORTRAN integer functions allow bit manipulation of integer variables (one DDP-516 machine word).

Y=RS(X,N)

Y is set equal to X right shifted by N bits. If N is greater than 16 or less than zero, Y becomes 0. RS is an integer function and must be declared INTEGER by the FORTRAN program which uses it.

Y=LS(X,N)

Y is set equal to X left shifted by N bits. If N is greater than 16 or less than zero, Y becomes 0.

Y=RT(X,N)

Y is cleared and then set equal to the rightmost N bits of X. If N is greater than 16 or less than zero, Y equals X unchanged. If N=0, Y=1000000. RT is an integer function and must be declared INTEGER by the FORTRAN program which uses it.

Y=LT(X,N)

Y is cleared and set equal to the leftmost N bits of X. These bits will not be right justified, i.e., Y=LT(123456,3) = 120000. If N is greater than 16 or negative, Y equals X unchanged.

Y=AND(X,Y)

Each bit in X is Boolean ANDed with the corresponding bit in Y and the result put in Z. This function (AND) must be declared INTEGER in the FORTRAN program which uses it. Furthermore, only sixteen (16) bits of X and sixteen bits of Y are dealt with yielding 16 bits of Z. Therefore, X,Y, and Z should be declared INTEGER, or else the user should realize that he is working with only the first word of a real two word quantity or of a double precision triple word quantity.

Z=OR(X,Y)

Each bit of X is Boolean ORed with the corresponding bit in Y and the result placed in Z. This function (OR) must be declared INTEGER in the FORTRAN program which uses it. Furthermore, only 16 bits of X and 16 bits of Y are dealt with yielding 16 bits of Z. Therefore X, Y, and Z should be declared INTEGER or else the user should realize that he is working with only the first word of a real two word quantity or of a double precision triple word quantity.

Z=XOR(X,Y)

Each bit of X is Boolean XORed with the corresponding bit in Y and the result placed in Z. This function (XOR) must be declared INTEGER in the FORTRAN program which uses it. Furthermore, only 16 bits of X and 16 bits of Y are dealt with yielding 16 bits of Z. Therefore, X, Y and Z should be declared INTEGER.

3. DOS Interface Routines

These subroutines allow FORTRAN programs to read and write files on the disk, pick up the DOS command line for examination and return control to DOS at the end of a program.

SEARCH

The calling sequence is CALL SEARCH (KEY,NAME,UNIT,ALTRIN). The SEARCH routine is used in conjunction with READ and WRITE to read and write information on the disk through program control. These routines work equally well whether a program is running under the DOS or the TSDOS operating system. The SEARCH routine coordinates all activities with file directories. The KEY specifies the activity and has the following meaning: 1 - open for reading only, 2 - open for writing only, 3 - open for reading and writing, 4 - close file, 5 - delete file, 7 - rewind file. NAME is a 3 word or 6 character array. If the name is less than 6 characters, the name should be left justified with trailing blanks. The UNIT is the unit number associated with the file and must be between 1 and 7. ALTRIN is an alternate return taken in case of uncorrectable errors (e.g. attempting to open a file already open). SEARCH allows a user to do under program control the operations of commands INPUT, LISTING, BINARY, OPEN, CLOSE and DELETE.

READ

The calling sequence is CALL READ (UNIT,ARRAY,NWRDS, ALTRIN). The READ subroutine reads NWRDS from unit UNIT into ARRAY. The return is to ALTRIN (if specified) if the end-of-file is encountered reading or if the unit is not open. Note that the accumulator, obtained on End-of-File-Return, contains the number of words not read in the final call.

GETA

The calling sequence is CALL GETA(A). GETA puts the accumulator into A. This routine is useful with subroutine READ, above.

WRITE

The calling sequence is CALL WRITE(UNIT,ARRAY,NWRDS, ALTRIN). The WRITE subroutine writes out NWRDS from ARRAY to unit UNIT. ALTRIN (if specified) is taken in case of uncorrectable errors.

SAVE

The calling sequence is CALL SAVE(VECT,NAME). A user sets up a nine word vector VECT(9) before calling SAVE. VECT(1) should be set to an integer which is the lst location of core to be saved and VECT(2) should be set to the last location to be saved. The rest of the vector may be set at the programmer's option. VECT(3) represents the saved program counter, VECT(4) saved A register, VECT(5) saved B register, VECT(6) saved X register, VECT(7) saved keys register. VECT(8) is reserved for a checksum which is calculated and inserted in this location by SAVE. VECT(9) is a spare location. VECT(7) should always be set to 20000 octal to indicate extend mode. SAVE will write out the nine word vector VECT followed by the core image starting at VECT(1) and ending at VECT(2) on file NAME. The SAVE subroutine has the same effect under program control as the DOS SAVE command.

RESTOR

The calling sequence is CALL RESTOR(VECT,NAME). Subroutine RESTOR performs the inverse operation. A file previously written with SAVE is read. The first 9 words of the file are entered into VECT(9). RESTOR then reads the rest of the file into sequential locations

starting at VECT(1) and ending at VECT(2). The user may examine VECT(3) - VECT(9) which contain saved machine active registers. RESTOR subroutine has the same effect under program control as the DOS RESTOR command.

RESUME

The calling sequence is CALL RESUME(NAME). File NAME is "resumed" as if the user had typed the command RESUME NAME.

ATTACH

The calling sequence is CALL ATTACH(UFD,DISK,PASSWORD, TSET,ALTRIN). ATTACH attaches to UFD (a six character name) if PASSWORD (another name) matches the stored password. UFD is searched on logical disk drive DISK (range 0-3) or all disks assigned to the system in logical disk order if DISK is 100000 octal. Return is to ALTRIN (if specified) if UFD is not found. If TSET is true, HOME-UFD, a DOS vector is set to this new directory. If false, no action is taken.

HOMUFD

The calling sequence is CALL HOMUFD. CALL HOMUFD will attach to the directory whose name is in HOME-UFD, a DOS vector. See ATTACH, above.

CMREAD

The calling sequence is CALL CMREAD(ARRAY). CMREAD reads 18 words which represent the last command line typed into ARRAY as follows:

ARRAY(1)	COMMAND	(or spaces)
ARRAY(2)		
ARRAY(3)		
(4)	NAME1	(or spaces)
(5)		
(6)		
(7)	NAME2	(or spaces)
(8)		
(9)	PAR1	(or zero)
(10)	PAR2	(or zero)
.		
.		
.		
(17)	PAR9	(or zero)

The command line may then be accessed directly.

EXIT

The calling sequence is CALL EXIT. EXIT returns to DOS or TSDOS which will type out "OK". A user may open or close files or switch directories and restart his program at the next FORTRAN statement by typing START.

O$NU

The calling sequence is CALL O$NU(BUF,N,ALTRIN). O$NU takes a line of up to 119 characters stored two to a word in array BUF, and writes it on the disk on unit N (under DOS or TSDOS) in compressed format. Null characters and trailing blanks are deleted from the line before writing. Multiple blanks are replaced by a relative horizontal tab character (221) followed by a number indicating the number of blanks. Newline characters (212) should not be put in the output buffer. O$NU writes a newline character on the disk after the last word in the buffer thus using the character to indicate the break between lines. If non-zero, ALTRIN is an address to which control is returned in case of uncorrectable errors. If ALTRIN is 0, control is returned to the supervisor and an error message is typed out. N must be in the range 1-7 and BUF must be an array of 60 words.

I$NU

The calling sequence is CALL I$NU (BUF,N,ALTRIN). I$NU does the inverse operation of I$NU. O$NU reads a line of characters from the disk on unit 1 (under DOS or TSDOS) into a 60 word buffer BUF, stored two characters per word. I$NU uncompresses the data, and replaces the end-of-line character newline (212) with a blank and fills the remainder of the buffer with blanks. ALTRIN is used as in O$NU. Control will also pass to ALTRIN if there are no more lines of data in the file. Compressed format files are also read and written by the commands ED, FTN, DAP, and LOOK. N must be in the 1 7. To insure ALTRIN is taken, the user should call SETEOF (0) at the beginning of his program.

4. Alternate Teletype Input-OUTPUT

Some users may wish to resort to using this teletype package and avoid formatted FORTRAN input-output completely if they have difficulty fitting their programs in the machine. This package handles only integers and characters. The normal user can skip this section.

PUTC

The calling sequence is: CALL PUTC(BUF,CHAR). Initially array BUF(3) should be set to 6 spaces. PUTC when called successively will insert character CHAR into array BUF so as to compose a six character name. If PUTC is called more than six times, no action is taken.

T1IN

The calling sequence is CALL T1IN(CHAR). T1IN reads a character from the teletype into CHAR, skipping nulls and echoing carriage returns for linefeeds and vice versa. (In either case, CHAR, is set to new line).

T1OU

The calling sequence is CALL T1OU(CHAR). T1OU types out CHAR, inserting a carriage return if CHAR is a new line.

TNOU

The calling sequence is CALL TNOU(ARRAY,NCHARS). TNOU types out NCHAR characters from ARRAY and adds a new line.

TNOUA

The calling sequence is CALL TNOUA(ARRAY,NCHARS). TNOUA types out NCHARS characters from ARRAY, but does not add a new line.

TOOCT

The calling sequence is CALL TOOCT(NUMBER). TOOCT types out NUMBER in the ASCII representation of the number converted to octal.

TONL

The calling sequence is CALL TONL. This types out a new line and is useful after such routines as TOOCT and TODEC which do not put out new lines.

TIDEC

The calling sequence is CALL TIDEC(N10). This routine is called to input an unsigned or a negative decimal integer on the teletype. If the user makes an error

in typing, " is acceptable as an erase character and ? as a line kill. If an integer larger than 32767 or less than -32768 is typed, the rightmost digits are saved, and there is no error indication. If a character is not recognized, a " is echoed on the teletype, and the typed character is not stored and need not be erased. N.B. A plus (+) is not recognized.

TIOCT

The calling sequence is CALL TIOCT(N8). This routine is called to input an octal number of the teletype. If the user makes an error in typing, " is acceptable as an erase character and ? as a line kill. If a character is not recognized, a " is echoed on the teletype and the typed character is not stored and need not be erased. Among unrecognized characters are alphabetic characters, special characters and the digits 8 and 9 (i.e., only digits 0-7, ?, ", and line feed are legitimate.) The routine accepts 100000 to 077777, i.e., the maximum first digit is 1, and the maximum 2nd through 6th digit is 7. If a number too large were typed, only the rightmost bits would be saved. (i.e. 277777 would be saved as 077777).

TODEC

The calling sequence is CALL TODEC(N8). This routine is used to output a decimal integer on the teletype. One may convert decimal integers to their octal equivalent by coding:

```
CALL TIOCT(N8)
CALL TODEC(N8)
```

TODEC does not convert the octal number 100000 correctly. (It should be -32768, not -0 as the printout gives.)

RDCOM

The calling sequence is CALL RDCOM(BUF). RDCOM reads up to 80 characters from the teletype into an 80 word buffer BUF one character per word. Teletype input is modified by the kill character (?) and the erase character ("). A newline character (linefeed) terminates input and is put in the buffer. The loading of this subroutine causes a named common area "X" to be put in the user's load map. RDCOM also causes the only variable PEDL (integer) in this common area to be set to 1.

GETWRD

The calling sequence is CALL GETWRD(BUF,NAME). Successive calls to GETWRD pick up successive "English" words from array BUF(80) where characters are stored one per word. GETWRD puts these names in array NAME(3) packed 2 characters per computer word and left justified with trailing blanks. If a word in BUF is longer than six characters, it is truncated to six characters. The "English" words may contain any character or number except space, newline, comma, or semicolon. This subroutine should be used in tandem with RDCOM. The loading of this subroutine causes a named common area "X" to be put in the user's load map. PEDL(integer), the only variable in the common area X, is used by GETWRD and reset by RDCOM to indicate the current character being processed.

5. Miscellaneous

F$AT

The calling sequence is CALL F$AT

```
        OCT   N
ARG1    DAC   **
ARG2    DAC   **
        ........
ARGN    DAC   **
```

N is the number of arguments to be transferred; ARG1, ARG2,...,ARGN are the argument names in the subroutine. F$AT is called whenever a FORTRAN subroutine has calling arguments to be passed. Calls to F$AT are not to be coded by the user in a FORTRAN program, but F$AT is useful for programming DAP subroutines to be called by FORTRAN. For example, below is coding to call a DAP routine ASEMRT and pass the arguments X and Y with F$AT.

```
FORTRAN main program
CALL ASEMRT(X,Y)
DAP subroutine

    SUBR   ASEMRT,A
A   DAC    **          entry point
    CALL   F$AT
    OCT    2
X   DAC    **
Y   DAC    **
```

VALUE=NAMEQV(NAM1,NAM2)

VALUE and NAMEQV are logical variables and must be so declared. VALUE is set to true if arrays NAM1(3) and NAM2(3) are equivalent; false otherwise. This function is useful for comparing 6 character names with each other.

VALUE2=COMEQV(COM1,COM2)

VALUE2 and COMEQV are logical variables and must be so declared. VALUE2 is set to true if the leftmost characters (up to six) of the array COM2(3) are equivalent to the leftmost characters of array COM1(3). There is the possibility of comparing a blank array COM2(3) with a six character name in COM1(3) and they will set VALUE2 to true. (i.e., the program considers the COM1 and COM2 equivalent).

C. List of FORTRAN Library Routines

This section describes all the routines in the FORTRAN library. There are several conventions used in describing the functions of the library:

1. For function names:
   - A = add
   - C = convert
   - D = divide
   - E = exponentiation
   - F = Fortran utility
   - H = store (hold)
   - I = input
   - L = load
   - M = multiply
   - N = negate
   - O = output
   - S = subtract
   - Z = clear (zero)

2. For arguments:
   - 1 = integer
   - 2 = real
   - 5 = complex
   - 6 = double precision
   - 8 = exponent

Any routine listed twice, one with an "X" like (A$66 / A$66X) means there are two versions in the library. The "X" version indicates it is written to use the high-speed arithmetic option, which our machine has.

Each routine is listed under the library where it is to be found. The column "filed under" tells under which routine listing the routine in question will be described more fully. "Description" is a brief explanation as to what the routine does. An asterisk (*) indicates that this is either a new routine (unavailable from Honeywell) or contains changes that we have made from Honeywell version.

(a) FINLIB

Routine	Filed Under	Description
ABS	ABS	$R_2 = \|R_1\|$
AC1-AC5	AC1	The pseudo-accumulator used by the machine
AIMAG	AIMAG	if $C = R_1 + iR_2$ then R_2 = AIMAG(C)
AINT	AINT	truncates real to integer but expresses it as real
(ALOG / ALOGX)	ALOGX	$R_1 = LOG_e\ (R_2)$
ALOG10	ALOGX	$R_1 = Log_{10}\ (R_2)$
AMAXO	MAXO	chooses largest argument (of list of integers) (must have more than 1) and expresses it as Real
AMAX1	MAX1	chooses largest argument (of list of reals) (# of arguments >1) = R
AMINO	MINO	same as AMAXO except minimum of list is chosen
AMIN1	MIN1	same as AMAX1 except minimum of list is chosen
AMOD	AMOD	$R_3 = R_1 (\text{mod } R_2)$
*AND	RS	R_3 = logical AND bits of $R_1 + R_2$
ARG$	ARG$	converts indirect address of argument to its direct address
ATAN	ATAN	$R_2 = \arctan\ (R_1)$

Routine	Filed Under	Description
ATAN2	ATAN	R_3 = arctan (R_1/R_2)
*ATTACH	FILPAK	attach to user file directory
A$22	A$22	$R_3 = R_1 + R_2$
(A$52)	A$52	$C_2 = C_1 + R_1$
A$55	A$55	$C_3 = C_1 + C_2$
(A$62)	A$62	$D_2 = D_1 + R_1$
(A$66 A$66X)	A$66XRA	$D_3 = D_1 + D_2$
A$81	A$81	Add integer to characteristic of D, i.e., multiply by a power of 2
CABS	CABS	if $C = R_1 + iR_2$ then cabs(c) = $R_3 = \sqrt{R_1^2 + R_2^2}$
CCOS	CCOS	C_2 = cos(C_1) (in radians)
CEXP	CEXP	$C_2 = e^{(C_1)}$
CHAIN	CHAIN	reads in next segment of a chained program
CLOG	CLOG	$C_2 = \log_e(C_1)$
CMPLX	CMPLX	C_1 = CMPLX(R_1,R_2) uses R_1 as real part, R_2 as imaginary part to form complex number
*CMREAD	Read	read DOS command line
*COMEQV	NAMEQV	command equivalence test (up to 6 characters compared to 6 characters)
*CONCOM	RS	constant common - constants available to all
CONJ	CONJG	C_2 = complex conjugate of C_1
COS	Sin, COS	R_2 = cosine (R_1)(in radians)
CSIN	CSIN	C_2 = sine (C_1) (in radians)

Routine	Filed Under	Description
CSQRT	CSQRT	$C_2 = \sqrt{C_1}$
*C$KBUF	FRW	part of 0$31-0$38
*C$12	C$12	converts integer to real
C$16	C$16	converts integer to double precision
*C$21	C$21	converts real to integer
C$25	C$25	converts real to complex format
C$26	C$26	converts real to double precision
C$61	C$61	converts double precision to integer
C$62	C$62	converts double precision to real
C$81	C$81	converts double precision exponent to integer
DABS	DABS	$D_2 = \lvert D_1 \rvert$
DATAN	DATAN	D_2 = arctan (D_1)
DATAN2	DATAN2	D_2 = arctan (D_1/D_2)
DBLE	DBLE	$D_1 = R_1$ - converts mode of argument
DCOS	DCOS	D_2 = cosine (D_1) (in radians)
DEXP	DEXP	$D_2 = e^{D1}$
DIM	DIM	$R_3 = R_1 - R_2$ if $(R_1 - R_2) > 0$ $R_3 = 0$ if $(R_1 - R_2) < 0$
DINT	DINT	I_1 = DINT(D_1) truncate to integer
DLOG	DLOG	$D_2 = \text{Log}_e(D_1)$
DLOG2	DLOG	$D_2 = \text{Log}_2(D_1)$
DLOG10	DLOG10	$D_2 = \text{Log}_{10}(D_1)$
DMAX1	DMAX1	chooses largest argument of list (length 1) of double precision arguments = D

Routine	Filed Under	Description
DMIN1	DMIN1	same as DMAX1 except chooses smallest
DMOD	DMOD	$D_3 = D_1 \pmod{D_2}$
DSIGN	DSIGN	magnitude of 1st argument, sign of 2nd argument (2 double precision arguments)
DSIN	DSIN	$D_2 = \text{sine}(D_1)$ (in radians)
DSQRT	DSQRT	$D_2 = \sqrt{D_1}$
*D$11	M$11	$I_3 = I_2/I_1$
(D$22 D$22X)	M$22	$R_3 = R_2/R_1$
(D$52)	D$52	$C_2 = C_1/R_1$
D$55	D$55	$C_3 = C_2/C_1$
D$62	D$62	$D_2 = D_1/R_1$
(D$66 D$66X)	A$66XRA	$D_3 = D_2/D_1$
*EXIT	READ	return to DOS
EXP	EXP	$R_2 = e^{R1}$
(E$11 E$11X)	E$11	$I_3 = I_2**I_1$
E$21	E$21	$R_2 = R_1**I_1$
E$22	E$22	$R_3 = R_2**R_1$
(E$26)	E$26	$D_2 = R_1**D_1$
E$51	E$51	$C_2 = C_1**I_1$
E#61	E$61	$D_2 = D_1**I_1$
E$62	E$62	$D_2 = D_1**R_1$
E$66	E$66	$D_3 = D_1**D_2$
FLOAT	FLOAT	$R_1 = I_1$ converts argument mode

Routine	Filed Under	Description
F$AR	F$IO	Fortran IV I/O system handles control & conversion
*F$AT	F$AT	transfers arguments
F$CB	F$IO	same as F$AR
F$D2	F$D5-9	write stop code for end-of-file on paper tape
F$D5-9	F$D5-9	Mag. tape end-of-file driver
F$DN	F$D5-9	variable driver
*F$ER	F$HT	outputs 2 character error message
F$F5-9	F$F5-9	Mag. tape backspace driver
F$FN	F$F5-9	Mag. tape variable backspace driver
*F$HT	F$HT	When "stop" is encountered in Fortran
*F$IO	F$IO	same as F$AR
*F$L1-6	F$L1-6	logical ifs
*F$RN	F$RN	variable input driver selection
*F$R1	FR1	Input BCD from typewriter
F$R2	F$R2	Input BCD from paper tape reader
F$R3	F$R3	Input BCD from card reader
*F$R4	F$R4	Input BCD from DOS file unit
F$R5-9	F$R5-9	Magnetic tape input driver
*F$R31-38	FR31-38	input from disk file units 1-8.
*F$TR	DTRCE	Fortran trace
*F$TRCE	FTRCE	Fortran trace
*F$WN	F$WN	Variable output driver selection
*F$W1	FW1	Output BCD on typewriter

Routine	Filed Under	Description
F$W2	F$W2	Output BCD on paper tape
*F$W4	F$W4	Output BCD on DOS file unit #2
F$W5-9	F$W5-9	Magnetic tape output driver
*F$W31-38	FW31-38	output to disk file units 1-8
*GETA	FILPAK	pick up accumulator
*Getwrd	Getwrd	See section B
HOMUFD	FILPAK	Attach to HOME-UFD
H$22	H$22	Single precision real hold
H$55	H$55	Complex hold
H$66	H$66	Double precision hold
IABS	IABS	Integer absolute value
IDIM	IDIM	Integer positive difference
IDINT	IFIX	Truncate double precision to integer
IFIX	IFIX	Converts argument mode I = R
INT	IFIX	Truncates real to integer
*ILOC	SFTPAK	Z = ILOC(X) puts address of X in Z
ISIGN	ISIGN	(2 integers) to create value with (1) magnitude of 1st argument (2) sign of 2nd argument
*I$NU	FRW	expand input buffer
*I$001	FREAD	to input a 40-word buffer from teletype
*I$031-038	FREAD	input buffer from disk
*LS	SFTPAK	left shift
*LT	SFTPAK	left part of
L$22	L$22	Single-precision load
L$33	L$33	Form inclusive or

Routine	Filed Under	Description
L$55	L$55	Complex load
L$66	L$66	Double-precision load
*MOD	M$11	$I_3 = I_2 \bmod I_1$ (integer remainder)
MAXO	MAXO	Picks largest integer of group of integers, returns integer
MAX1	MAX1	Picks largest real of group of reals, returns integer
MINO	MINO	same as MAXO for minimum
MIN1	MIN1	same as MAX1 for minimum
*M$11	M$11	$I_3 = I_2 * I_1$
(M$22 M$22X)	M$22	$R_3 = R_2 * R_1$
(M$52)	M$52	$C_2 = C_1 * R_1$
M$55	M$55	$C_3 = C_2 * C_1$
(M$62)	M$62	$D_2 = D_1 * R_1$
(M$66 M$66X)	A$66XRA	$D_3 = D_2 * D_1$
*NAMEQV	NAMEQV	name equivalence test, (six characters to six characters)
N$22	N$22	negate real ($R_2 = -R_1$)
N$33	N$33	(only in logical "if" starts) negate logical L_2 = NOT (L_1)
N$55	N$55	negate complex $C_2 = -(C_1)$
N$66	N$66	negate double precision $D_2 = -D_1$
*OR	RS	bit wise or of two integers
OVERFL	OVERFL	indicates if error flag is necessary
*OVERLA	OVERLA	DOS overlay feature

Routine	Filed Under	Description
O$AC	O$AP	carriage control - types line, returns carriage, advances to new line
O$AC	O$AP	
O$AP	O$AP	
O$LF	O$LP	advance to new page, advance to new line print new line
O$LO	O$LP	
O$LP	O$LP	
O$PC	O$PP	punch a line, punch carriage return, advance to new line
O$PF	O$PP	
O$PP	O$PP	
*O$001	FREAD	output to teletype
*O$031-38	FREAD	output to disk units 1-8
*PUTC	PUTC	store character in array
*RESTOR	RESTOR	restores core image
*RESUME	OVERLA	resumes core image
*RETURN	OVERLA	overlay feature
*RDCOM	GETWRD	read teletype line
*RDLIN	RDLIN	read line from disk unit one
*READ	FILPAK	read from DOS file
*RS	SFTPAK	right shift
*RT	SFTPAK	right part of
*SAVE	RESTOR	saves core image
*Search	Read	change status of DOS unit
Sign	Sign	(2 reals) magnitude of 1st, sign of 2nd argument
SIN	SIN, COS	$R_2 = SIN(R_1)$
SLITE	SLITE	sense switch operations
SLITE	SLITE	
SSWTCH	SLITE	

Routine	Filed Under	Description
SNGL	C$62	converts argument mode $R_1 = D_1$
(SQRT / SQRTX)	SQRTX	$R_2 = R_1$
S$22	A$22	$R_3 = R_2 - R_1$
(S$52)	S$52	$C_2 = C_1 - R_1$
S$55	S$55	$C_3 = C_2 - C_1$
(S$62)	S$62	$D_2 = D_1 - R_1$
(S$66 / S$66X)	A$66XRA	$D_3 = D_2 - D_1$
TANH	TANH	R_2 = hyperbolic tangent (R_1)
*TIDEC	TODEC	input decimal number, teletype
*TIOCT	TODEC	input octal number, teletype
*TODEC	TODEC	output decimal number, teletype
*TNOU	PutC	output string of characters
*TNOUA	PutC	same as TNOU except no newline at end
*TONL	PutC	output newline
*TOOCT	PutC	output octal number
*TllN	PutC	input character
*TlOU	PutC	output character
*WRBN	WRBN	WRITES BINARY ON DISK
*WRLIN	Rdlin	write line to disk unit 2
*WRITE	Read	write to DOS file
*XOR	RS	bitwise XOR of two integers
Z$80	Z$80	clear (zero) double precision exponent

XI BASIC

A BASIC interpreter is available. See the Honeywell BASIC manual for how to program in BASIC. BASIC is a self-contained system. A user does not create a BASIC program using the editor, but instead enters his program by typing it under the input section of the BASIC system.

To invoke BASIC, type BASIC. Then follow the instructions given starting at page 5-4 of the BASIC manual.

XII GASP SIMULATION SYSTEM

GASP is a group of FORTRAN subroutines designed to be used in programming simulation models. These subroutines include input processing, statistical distributions, data collection routines, and report generation. The programmer must write a calling program and as many event routines as his model requires. The purpose of this memo is to describe how to use GASP on the DDP-516. (For details on the GASP routines and their functions, the reader is referred to SIMULATION WITH GASP II by Pritsker and Kiviat.)

A GASP LIBRARY

The GASP subroutines and functions are stored on the U directory in file GASPLIB. These routines are described below. The order in which they are listed is the order in which they are stored on the file. The routines marked * usually will not be called directly by the programmer. Routines marked "F" are functions.

GASP (NSET) - GASP is the master control routine.

*DATAN (NSET) - Initializes all GASP COMMON variables, reads in initial events and initial file entries.

*SET (JQ, NSET) - Initializes filing array NSET, updates pointer system that establishes the relationships between entries of file JQ, and maintains statistics on the number of entries in file JQ.

FILEM (JQ, NSET) - Files an entry in File JQ of the array NSET.

RMOVE (KCOL, JQ, NSET) - Removes an entry from File JQ of array NSET. Any entry whose column number KCOL is known can be removed from any file in array NSET.

FIND (XVAL, MCODE, JQ, JATT, KCOL, NSET) - Locates a column KCOL in Files JQ of the array NSET. KCOL contains a value bearing a relationship designated by MCODE to some value, XVAL. JATT is the row of NSET containing the attribute value to be used in finding KCOL.

*PRNTQ (JQ, NSET) - Computes and prints the time-integrated average and standard deviation of the number of entries in a particular file (JQ) and the maximum entries since the file JQ are also printed out.

*SUMRY (NSET) - Processes data collected in COLCT, TMST, and HISTO and prints out a data summary.

ERROR (J, NSET) - Called when an error is detected in any GASP routine except PRNTQ, SUMRY, and MONTR. The error message printed provides a code that indicates the subroutine in which the error occurred. In addition, the file status and summary statistics are printed and the simulation is terminated.

COLCT (X, N, NSET) - Collects sample data on the value of variable X. N is the code given to variable X by the programmer.

TMST (X, T, N, NSET) - Collects sample data on observations of a variable X over a period of time, N is the code assigned to variable X, and T is the current time.

MONTR (NSET) - Provides capability to monitor events by printing either the entire filing array or only the next event to be removed from the event file.

DRAND (ISEED, RNUM) - The GASP pseudo-random number generator. ISEED is the intial random integer, and RNUM is the random number generated.

F UNFRM (A, B) - Generates a deviate from a uniform distribution on the interval (A, B).

F AMIN (ARG1, ARG2) - Finds the minimum of two floating-point variables.

F AMAX (ARG1, ARG2) - Finds the maximum of two floating-point variables.

F XMAX (IARG1, IARG2) - Finds the maximum of two fixed-point variables.

HISTO (X1, A, W, N) - Tabulates the number of times X1 is within the cell limits of histogram N. A is the lower of the second cell of the histogram and W is the width of each cell.

F ERLING (J) - Generates a deviate from an ERLING distribution. J corresponds to the row of array PARAM containing the parameter values for use with this function.

NPOSN (J) - Generates a deviate, NPSSN from a POISSON distribution. J corresponds to the row in array PARAM containing the parameters for this distribution.

F RNORM (J) - Generates a deviate from a normal distribution. The parameters for RNORM are stored in row J of PARAM.

F SUMQ (JATT, JQ, NSET) - Computes the sum of all attribute values stored in row JATT of file JQ.

F PRODQ (JATT, JQ, NSET) - Computes the product of all attributes in row JATT of file JQ.

In addition, a subroutine EVNTS must be written to schedule the events of the system. The length of the simulation run may be controlled through a scheduled end of simulation event subroutine (ENDSM) called by ENDSV or the termination time may be specified in the input data. (See section 2.0) A subroutine OTPUT may be written for specially tailored output. OTPUT is always called by GASP so a dummy subroutine consisting of just a RETURN command should be included in the program if no special output is required.

B INPUT FORMAT

There are eight different data card types. The sample Input Data Worksheet presents the layout of these cards. An explanation of the variables initialized on the data cards is presented below. When typing data on the teletype, free-formatted input (using commas) may be used.

CARD 1

GASP Variables	Definition
NAME	Programmer's name.
NPROJ	Project number.
MON	Month number.
NDAY	Day number.
NYR	Year number.
NRUNS	Number of simulation runs to be made.

CARD 2

NPRMS	Number of sets of parameters to be used in simulation.
NHIST	Number of histograms required for this simulation.
NCLCT	Number of variables for which statistics are collected in subroutine COLCT.
NSTAT	Number of variables for which statistics are collected in subroutine TMST.
ID	Number of columns (entries) in the filing array NSET.
IM	Maximum number of attributes associated with any entry of the filing array.
NOQ	Number of files contained in the filing array.
MXC	Largest number of cells to be used in any histogram.
SCALE	Parameter used to scale attribute values to avoid truncation errors due to the use of a fixed-point array for the filing array.

CARD 3 (optional)

Defines number of cells in each histogram if histograms are used.

CARD 4

Values of vector KRANK are initialized. [KRANK(J) = is attribute on which File J is ranked.]

CARD 5

Values for vector INN. INN(J) specifies whether file J is low-value-first (LVF) or high-value-first (HVF). If INN(J) = 1, file J is LVF and if INN(J) = 2, file J is HVF.

CARD 6 (optional)

Used if parameters are used. (See Section 5.)

CARD 7

GASP Variables		Definition
MSTOP	0	End-of-simulation event has been furnished by the programmer. The ENDSM routine should set MSTOP to -1.
	+	The simulation is to end when TNOW ≥ TFIN.
	-	Programmer has indicated the simulation run is completed and final reports should be given, if requested.
JCLR	0	Do not clear statistical storage areas.
	1	Clear statistical storage areas.
NORPT	0	Final GASP summary reports to be printed (SUMRY + OTPUT will be called.)
	1	No final summary reporting required.
NEP		A control variable which determines the data card type at which subroutine DATAN will begin to process for the next simulation run.
TBEG		Beginning time of the simulation run, i.e., initial value of TNOW.

TFIN		Final or ending time of the simulation if MSTOP is positive. If +, SUMRY is automatically called.
JSEED	0	Random number seed is not to be changed, and TNOW is not to be set to TBEG.
	-,+	Random number seed ISEED is set to JSEED.

CARD 8

To initialize filing array and to insert initial entries into it.

C ERROR CODES

Subroutine ERROR prints error codes when an error occurs during execution of a GASP model. The standard codes and their error sources are listed below. Any unassigned numbers may be used as error codes for programmer written routines.

Error Code	Subprogram in Which Error Occurred	Error Code	Subprogram in Which Error Occurred
84	PRODQ	93	GASP
85	SUMQ	94	PRNTQ
87	FILEM	95	DATAN
88	SET	96	not used
89	FIND	97	RMOVE
90	COLCT	98	SUMRY
91	TMST	99	MONTR

Note: Codes 94, 98, and 99 are printed out by the detecting subroutine and not subroutine ERROR.

D DISTRIBUTION FUNCTIONS

Parameters are used to store the arguments of the distribution functions that are used. The following table summarizes the parameter assignments that must be used for the different functions:

	Parameter Number			
Function	1	2	3	4
RLOGN(J)	u_x	min X	max X	$\sqrt{X}$
ERLNG(J)	$1/u_x$	min X	max X	K
NPOSN(N, NPSSN)	$x-n_0$	n_0	max X	-not used-
RNORM(J)	u	minX	max X	$\sqrt{X}$

E GASP FILES

A GASP model may contain up to four files. File 1 must always be the event file. Any other files may be assigned as required. Up to four attributes may be associated with each file.

The attributes for File 1 are assigned to the following:

Attribute 1: Scheduled time of event
Attribute 2: Event Code (if more than 1 event type is used)
Attribute 3: Optional
Attribute 4: Optional

Every GASP model must contain at least one file--the event file.

F GASP LOADING

In loading a large program under DOS, the ordering of the subroutines in the files is important. In loading a GASP model, the file listing textbook are extremely helpful. (A copy of Appendix D is attached to this memo.)

The following procedure illustrates the loading of a GASP model:

(file TEST1 contains user written routines)

```
USER:       CLRCOR
RESPONSE:   GO
            OK

USER:       LDR B←TEST1
RESPONSE:   GO
            MR, OK

USER:       ATTACH U
RESPONSE:   OK

USER:       START GASPLIB
RESPONSE:   GO
            MR, OK

USER:       ATTACH USER DIRECTORY
RESPONSE:   OK

USER:       START B←TEST1
RESPONSE:   GO
            MR, OK

USER:       ATTACH U
RESPONSE:   OK

USER:       START GASPLIB
RESPONSE:   GO
            MR, OK

USER:       START FTNLIB
RESPONSE:   GO
            MR, OK

USER:       START 63002
RESPONSE:   GO load map is typed
            LC, OK

USER:       ATTACH USER DIRECTORY
RESPONSE:   OK

USER:       SAVE *TEST 100 *PBRK 1000
```

If program is to be run:

```
USER:       R  *TEST
RESPONSE:   GO
            type data.
```

G DEBUGGING AIDS

Subroutine MONTR provides the capability to selectively monitor events. It gives the option of a printout of all files or only the next event to be removed from the event file.

Monitor events can be inserted at the start of a simulation run by inserting an event into the event file with an event code of 100 or 101. Code 101 causes the entire filing array to be printed. Code 100 causes each event to be printed until a second event with a code of 100 occurs. The programmer may want to modify MONTR from the standard GASP II version to obtain additional information about a particular simulation.

Diagnostics are limited to the error codes printed by subroutine ERROR and the FORTRAN compiler. Programmer written subroutines such as the event routines should make use of the model.

H ADDITIONAL INFORMATION

A careful reading of the GASP textbook is recommended for anyone attempting to use GASP. The routines have been tested on the DDP-516, however, problems may still be encountered. A version of GASP for the IBM 1130 was converted for use on the DDP-516. Any difficulties with GASP should be reported to Judy Gertler (Rm. 11-29, Ext. 2231).

APPENDIX D

SUBPROGRAMS USED BY GASP II SUBPROGRAMS

GASP Subprogram	Subprograms Used	Users of This Subprogram
GASP	DATAN MONTR RMOVE EVNTS SUMRY OTPUT ERROR	MAIN
DATAN	ERROR EXIT DRAND SET FILEM	GASP
SET	XMAX ERROR EXIT	DATAN FILEM RMOVE
ERROR-1	EXIT	GASP, DATAN SET, FILEM RMOVE, COLCT, TMST
ERROR-2	MONTR PRNTQ SUMRY EXIT	Same as ERROR-1
FILEM	ERROR SET	DATAN
RMOVE	ERROR SET	GASP
COLCT	ERROR AMIN AMAX	NONE

TMST	ERROR	NONE
SUMRY	EXIT PRNTQ	GASP ERROR-2
PRNTQ	EXIT	ERROR-2 SUMRY
MONTR	EXIT	GASP ERROR-2
FIND	ERROR	NONE
HISTO	EXIT	NONE
NPOSN	DRAND RNORM EXP	NONE
RNORM	DRAND ALOG COS	NPOSN
DRAND	RANDU	DATAN, NPOSN, RNORM
ERLNG	EXIT DRAND ALOG	NONE
RLOGN	RNORM EXP	NONE
UNFRM	DRAND	NONE
SUMQ	ERROR	NONE
PRODQ	ERROR	NONE
AMIN	NONE	COLCT,TMST
XMAX	NONE	SET
AMAX	NONE	COLCT, TMST

XIII DAP

A. Introduction

This chapter tells the user how to run the DAP assembler. See the Honeywell DAP manual for how to program in DAP.

B. Normal Assembly

Assume you have a DAP source program on file TEST in your directory. To assemble the program you must do the following steps.

```
user:       INPUT TEST
response:   OK
```

Prepare file TEST to be read on logical unit 1. DAP expects to read source text from unit 1, and write binary data on unit 3.

```
user:       BINARY B←TEST
response:   OK
```

Prepare a file with name B←TEST to be written on logical unit 3. If file B←TEST already exists, prepare file B←TEST to be overwritten. If B←TEST does not presently exist, generate a file initially containing no data but with name B←TEST.

```
user:       DAP
response:   GO
            NO ERRORS IN ABOVE ASSEMBLY
            AC,
            OK
```

The DAP assembler reads the source file TEST, "rewinds" the file when the END card is detected and reads the file again for the second pass. During the second pass, the assembler generates binary output on B←TEST. If any errors occur, they will be typed on the teletype along with the line containing the error. When the second pass is complete, DAP tells you if any errors have occurred, types "AC" for assembly complete, then returns to DOS which types "OK".

```
user:       CLOSE ALL
response:   OK
```

Release logical units from any files they are associated with. In this case files TEST and B←TEST are released from logical units 1 and 3 respectively. These logical units are not free to be associated with different files.

C. Options

It is not possible to assemble more than one DAP program or subroutine at once. This means each DAP subroutine must be in a separate file and assembled by a separate sequence of commands.

It is possible to generate a DAP listing file with a format as described in the DAP manual. In this case, errors are not typed but appear at the appropriate place in the listing file. The following is an example of this option.

```
user:       INPUT TEST
response:   OK

user:       LISTING B←TEST
response:   OK
```

The listing command means open unit 2 for writing. The DAP assembler expects to write the listing on unit 2.

```
user:       BINARY B←TEST
response:   OK

user:       DAP 400 100555
response:   GO
            NO ERRORS IN ABOVE ASSEMBLY
            AC,
            OK

user:       CLOSE ALL
response:   OK
```

The user may examine the listing file on the display using the LOOK command or he may examine it using the test editor.

D. Calling FORTRAN Programs from DAP Programs

See Chapter X, Section B, part 5 for this procedure.

E. XREF

XREF generates a cross reference index to symbols in a DAP source file. This index is useful as an addendum to DAP listing files. To use, type

```
user:       INPUT "DAP source file"
response:   OK

user:       LISTING name
response:   OK

user:       XREF
response:   GO
            OK,

user:       CLOSE ALL
response:   OK
```

XIV LOADER

A. Introduction

This chapter describes how to load programs from relocatable binary files which are produced by the FORTRAN compiler or the DAP assembler.

B. Simple Load Procedure

If a user has a program which is contained in a single DOS file, the loading procedure is easy. Example:

```
user:       CLRCOR
response:   GO
            OK
```

CLRCOR is a DOS command that puts zero in all locations available for loading user programs. This step should always be done before loading.

```
user:       LDR B←TEST
response:   GO
            MR, OK
```

The loader (command name LDR) loads the binary file B←TEST from the users file directory. When the loader detects a piece of binary data equivalent to the $0 line of a FORTRAN source program or the END line of an assembly source program, the loader types MR, meaning "more" or LC, meaning loading complete, then returns to command level DOS. The loader automatically loads programs beginning at location 1000 octal with "links" b beginning at 100 octal. Programs are loaded in the extended addressing mode.

In many cases, the loading of the FORTRAN library is also required. The following is an example of this.

```
user:           ATTACH U
response:       OK
```

We must load a binary data file, FTNLIB, to complete our load. Since this file is used by all users, it is accessed through the U directory, a directory which is used by all users

it is accessed through the U directory, a directory which is used by all users and belongs to no one user. We have "attached" to the U directory in preparation for loading this file.

```
user:           START FTNLIB
response:       GO
                LC, OK
```

The DOS internal command START transfers to the location given as its first numerical argument. If no numerical argument is given, control is transferred to the location plus one from which DOS was last entered. The location in this case is the place in the loader to continue a load. The loader starts reading the FTNLIB file but only loads those routines necessary. "LC" indicates a complete load.

```
user:           CLOSE ALL
response:       OK
```

The loader leaves the last file read in an active state. CLOSE ALL closes out the file.

```
user:           START 63002
response:       GO

                complete load map is typed.
```

The load map consists of names paired with octal numbers. The first seven names have a special meaning. In the following, higher locations in memory extend towards location 77777.

1. *START - Location where first program load was begun.

2. *HIGH - Highest location loaded, plus one.

3. *NAMES - The loader stores the names of the loadmap as a list running downward from the bottom of the loader. Programs cannot be loaded over these locations.

4. *COMN - The bottom location of loaded common areas. Common starts at 57777. Programs cannot overlay common.

5. *PBRK - The location where the next program will be loaded by the loader.

6. *BASE - The next location the loader will load links. This number may not exceed 1000.

7. *LIST - A special variable defined with address of location 1. See the list processing section of the chapter on FORTRAN for details of its use (IX.B.1)

Other names in the load map are subroutine name paired with the location they are loaded. If a load map is typed when all subroutines are not loaded (MR), those names followed by * indicate subroutines needed which have not been loaded. The number following tells the last place these routines were called from.

If the *HIGH number was 5000, the user should type:

```
user:       SAVE *TEST 100 5000 1000
response:   OK
```

Generate a core image file named *TEST containing location 100 to 5000 as data. Also save a vector of machine initial conditions for restoration when the file is resumed. The machine conditions are given beginning as the fourth parameter of the SAVE command. They are in order program counter, A-register, B-register, index register and keys. In this particular case the program counter initial conditions is set to 1000 octal because the main program was loaded beginning at location 1000. All other machine registers are left to the value they were the last time control was returned to DOS. In particular, the keys are set to indicate extended mode, as the loader runs in extended mode and was the last program to return to DOS. In this way, *TEST is saved to run in the extended addressing mode, the mode it was loaded in. The user may specify all machine registers explicitly if he wishes.

The above step should always be taken by the user before running his program so that if his program fails, he may rerun it for debug purposes without reloading.

C. How to Find Program Errors by Examining the Load Map

Often, a user will make an error that is not caught by the compiler but can be detected on a load map.

1. Failure to get a "load complete" Message

This can happen if subroutine or function names have been misspelled either in a call statement or the subroutine statement. Also, if a subscripted variable has not been dimensioned, FORTRAN will assume the variable is actually a FORTRAN function and the name will appear in the load map.

2. Library Routine Examination

FORTRAN will automatically convert real-to-integer and integer-to-real. This conversion is done by subroutines C$12 and C$21. Often, a user will be working strictly with integers or real numbers. In this case, the presence of C$12 or C$21 indicates the user has not declared all his variables to be of the correct mode. The user should examine his program carefully and correct these errors.

D. Loading Two or More Files

Let us call these binary files B←T1, B←T2, and B←T3. The loading procedure would be:

```
user:       CLRCOR
response:   GO
            OK

user:       LDR B←T1
response:   GO
            MR, OK

user:       START B←T2
response:   GO
            MR, OK
```

```
user:       START B←T3
response:   GO
            MR, OK
```

The user may now load the FTNLIB as normal if he wishes.

E. Loading the FORTRAN Library

The format analyzer subroutine F$IO must be loaded below 40000 octal. A user should check his load map to make sure this is the case. If formatted I/O is not used, then there is no problem. In a large program, a user can load F$IO in lower core by loading a few subroutines, one of which contains format statements, loading the FORTRAN library, the rest of his program, and then the FORTRAN library again. The FORTRAN library, and other libraries in the U directory contained at the beginning a library mode block which causes only those routines necessary to be loaded from the library. This flag is added to the library by the Binary Editor. See Chapter XVI.

F. Loading Large Programs

1. Problem

If a user loads a large program, he may get a memory overflow message (MO) from the loader. He should first determine what caused the memory overflow message, as many problems may cause this message. Some of these problems can be solved by an alternate loading procedure as explained in part 2.

The loader will abort a load after filling up about 23,000 memory locations. The DDP-516 has a 32,768 bank of memory. When a user is loading a program, part of this memory is used by the operating system and the loader. The following is a map of the memory with locations given in octal.

0-100
: Reserved for hardware bootstraps, interrupts, etc.

100-777
: Reserved for user program links(indirect references between sectors).

1000-57000(approx.)
: Available for user programs. The loader normally uses this space.

57000 (approx.) - 63777
The loader program.

64000-64777
This space is reserved for DEBUG, a machine language debug package often used to debug user programs.

65000-66117
Empty.

66120-77777
Disk Operating System.

After a MO error message is encountered, the user should get a load map and examine the first two lines of the map. The MO error is encountered for one of four reasons.

1) The program's code has overlapped the program's common area. Common is allocated downward starting at 57777 octal. That is COMMON normally overlaps the loader. The user can detect this problem by comparing *PBRK and *COMN to see if they match. If so the user should use the alternate loading procedure.

2) The programs code has overlapped the loader names table. This is the problem if *NAMES is equal to *PBRK. The user should use the alternate loading procedure.

3) The program has filled up the link table. This is true if *BASE is 1000. See part 3 for solutions.

4) None of the above conditions can be found by examining the load map. The message can also occur if the user has preset common variables in his FORTRAN program. The comiler will not catch this bug. The user should examine his program for this problem.

2. Alternate Loading Procedure

Normally, Common is allocated downward from 57777 octal, because if a user has all 8 DOS disk file units open at once, the area from 60000 to 67777 is used for DOS file buffers. In HLDR, a version of the loader, Common is allocated downward from 63777 allowing 4000 octal more memory for user programs. Only two files may be open simultaneously during program execution if this loader is used.

To use HLDR simply type HLDR in place of LDR when loading programs.

If the user's program still will not fit, overlays must be used. See chapter XVIII on Overlays.

3. How to Minimize Links

A recurring problem faced by users with large programs on the DDP-516 is running out of room in sector zero for links. See Library memo #36 for the procedure to minimize links.

XV EDITOR

A. Introduction

The EDITOR is an on-line program within the DOS software system which allows the user to create files and edit them by context. This section serves as a USER'S GUIDE to the EDITOR.

The EDITOR is entered from the supervisor by means of the command

ED 'filename'

This command will cause a search of the user's file directory. If 'filename' is found it will be read in the EDIT will be printed (indicating the user is in EDIT MODE). If 'filename is not found, or no filename is specified, INPUT will be printed and the user will be placed in high-speed input mode. All filenames must be alpha-numeric beginning alpha, i.e., no '_', '#', etc.

B. Error Restart

An error that takes program control out of the editor can be recovered by starting at 1000 octal. The editor starts at 1001 octal and a new file may be edited by START 'filename' 1001.

C. File Size Restriction

The editor brings the entire file into core for editing. If the file is too large (longer than 2000 lines of DAP code), the editor will type BUFFER OVERFLOW LOADING. The user should at this point use an alternate editor. To use this editor type FILED 'filename'. FILED is a file-to-file editor which uses two auxiliary files while in use, EDT1 and EDT2. Unlimited length files can be handled FILED cannot "back up" one or more lines in the text as ED can, but can go to the beginning of the text repeatedly by means of the TOP command.

D. High-Speed Input Mode

When a user wishes to create a file or make additions to an old file he enters this mode. High-Speed input mode allows

the user to type at any speed he wishes because no response is typed for input lines. When the user types two successive .NL.'s, the EDITOR will place the user's console in EDIT MODE.

E. Edit Mode

When the user enters the EDIT MODE, EDIT is typed out on the user's console. From this point until the user types two .NL.'s the user's console will remain in this mode. All EDIT changes take place immediately, so that the user may make recursive modifications to his file.

The user's file is made up of groups of characters, delimited by .NL. characters. The EDITOR makes use of these .NL.'s to edit files on a line-oriented basis. The user may imagine a pointer that can move down (or up) a "listing" of his file. The user is, at any point in time, looking at only a single l line of his file - the one at which the pointer is stationed. The pointer is not affected by transitions between EDIT and INPUT modes and can only be moved by specific command while in the EDIT mode, or by typing lines in the INPUT mode.

If the user enters EDIT mode from INPUT mode the pointer is positioned at the last statement typed by the user while in INPUT mode. If however, the user begins in EDIT mode the pointer is positioned at the first line of the file. If the end of file is reached by an EDIT request the comment:

BOTTOM

is typed out. The pointer will be positioned at the last line of the file.

Any command which is not a legitimate EDITOR request will cause a "?" to be printed. Requests may be upper or lower case or any mixture.

The arguments of a request must be separated by at least one space. Most requests may be abbreviated, and their descriptions (and abbreviations) appear in the EDIT REQUESTS section. No spaces are needed between requests and integers. Several requests may appear on a line if they are separated by commas.

F. Display Feature

The command VERIFY if followed by (DISPLAY will cause all

subsequent output to appear on the IDI display. Output can be switched to the teletype by giving the BRIEF command. Lines of text output appear until the screen is full, at which time the bell is rung and the editor waits for you to examine the screen. The next character typed following the bell is not interpreted as an editor command but as a control to the display. A new-line or carriage return causes the display to "move up" one line. A "vertical tab" causes the picture to be pushed up six lines, whereas any other character causes the picture to be pushed up by 30 lines.

G. Tabulation

To facilitate ease of input, the EDITOR makes use of tab stops. These stops are originally set at 6 and 12 characters from the left, but by using the TABSET request (see EDIT REQUESTS) up to 8 tabs can be set anywhere along the line. To use tab feature, the user simply types a backslash (\). This character is interpreted as meaning fill in this line with blanks until the next tab stop. Hence, if one wishes to input the following DAP code: (Periods represent spaces in line).

```
A. . . .DAC. . . .STOP
```

he needs simply type:

```
A\DAC\STOP
```

H. Erase and Kill Characters

Lines typed to the editor may be changed if an error is made. The character " is the erase character. Typing this character will cause the preceding letter to be deleted from the line. Successive use of this character will cause successive characters to be removed. A "?" will "kill" the entire input line up to the ?. The erase and kill characters are effective whether in the INPUT or EDIT mode. The erase or kill character may be changed by means of the ERASE or KILL edit request.

I. Special Characters

The character '#' will match any number of spaces in a LOCATE or FIND command.

The character "!" is a wild card character and will match any character. It cannot be used in the CHANGE command.

The character ^ CARAT is the logical escape with the following conventions:

^000 octal character

^L move to lower case

^U move to upper case

J. Edit REQUESTS

In each of the following requests, if the request causes the pointer to reach the top of the file, TOP is typed and the request is terminated. If the request causes the pointer to reach the bottom of the file, BOTTOM is typed and the request is terminated. There is a dummy "null" line at both the top and the bottom of the file which is for editing convenience and does not appear in the file when it is on the disk rather than in the editor buffer in core. The letters which compose the abbreviation for each command are underlined.

PRINT 'n'

If 'n' is positive, the PRINT request prints 'n' lines starting with the line at which the printer is currently positioned. If n=0 or left unspecified, one line will be printed. If 'n' is negative, the PRINT request moves the pointer up 'n' lines and prints that line. Upon completion, the pointer will be left pointing to the last line printed. If the end of file is reached, BOTTOM is typed and the pointer is positioned at the last line of the file, a dummy "null" line. If the print request causes the pointer to reach the top of the file, TOP is typed and the pointer is stationed there, at a dummy "null" line. A print request when the pointer is either at the top or bottom of the file will type .NULL. indicating a dummy blank line. If 'n' is negative when using FILED, the command is ignored.

NEXT 'n'

The NEXT request moves the pointer forward if n positive, or backward if n is negative 'n' lines from the current line. If n=0 is not specified, it is assumed to be 1 and the pointer will be moved to the next line in the file. If 'n' is negative when using FILED, the command is ignored.

TOP

The TOP request causes the pointer to be positioned at the first line in the file, a dummy "null" line.

BOTTOM

The BOTTOM request moves the pointer to the last line of the file, a dummy "null" line.

FIND 'string'

The FIND request moves the pointer forward from the line following the current line to the first line beginning with 'string'. This facilitates location of lines by statement labels. If an end of file is reached, the pointer is positioned at the last line of the file.

LOCATE 'string'

The LOCATE request is used to move the pointer forward from the line following the current line to the first line containing 'string'. The full line is scanned.

CHANGE % 'string1' % 'string2' % 'n' G

The CHANGE request examines 'n' lines starting at the line at which the pointer is currently positioned. If 'n'=0 or left unspecified, it is assumed to be 1 and only the current line will be examined. '%' is the delimitor and can be any character except the break, erase, or kill characters. 'String1" and 'string2" are arbitrary character strings and may be of different lengths. The action of this request is to search the next 'n' lines for occurrences of 'string1'. If it is found, it is replaced by 'string2'. If 'G' is present (GLOBAL) every occurrence of 'string1' in a line will be replaced by 'string2'. Otherwise, only the first occurrence of 'string1' will be replaced in each examined line.

RETYPE 'line'

The RETYPE request causes the line pointed to by the pointer to be replaced by 'line'. The Pointer is not moved by this request.

INSERT 'line'

The INSERT request inserts a single line 'line' after the line currently pointed to without changing to high-speed input mode. The pointer is repositioned at the inserted line.

DELETE 'n'
DELETE TO 'string'

The DELETE request deletes 'n' lines from the file starting with the line at which the pointer is currently positioned. (If 'n' is negative, the command is ignored.) If n=0 or is not specified, it is assumed to be 1 and the current line is deleted. The pointer is left at the line following the last line deleted. If the end of the file is reached, all the lines up to that point will be deleted and BOTTOM will be typed. The DELETE TO 'string' option deletes all lines including the present one up to a line containing 'string' as would be found by the LOCATE command.

VERIFY
VERIFY (DISPLAY

The VERIFY request makes the EDITOR 'talkative'. After a verify request, the CHANGE, NEXT, TOP, FIND, LOCATE, RETYPE, and LOAD requests will all print out the current line after the changes (if any) have been made. The command VERIFY is followed by (DISPLAY will cause all subsequent output to appear on the IDI display. Output can be switched to the teletype by giving the BRIEF command. Lines of text output appear until the screen is full, at which time the bell is rung and the editor waits for you to examine the screen. The next character typed following the bell is not interpreted as an editor command but as a control to the display. A newline or carriage return causes the display to "move up" one line. A "vertical tab" causes the picture to be pushed up six lines, whereas any other character causes the picture to be pushed up by 30 lines.

BRIEF

The BRIEF request makes the EDITOR quicker because response is kept to a minimum. The PRINT request is the only one which will print any lines of text. This is the initial mode of the EDITOR.

LOAD 'filename'

The LOAD request loads file 'filename' after the current line in the EDITOR file. After the file is loaded, the pointer points to a "null" point following the loaded file. The load request is an easy way to concatenate files into one large file. If the filename is not found in the user's ufd, the editor will return to DOS which will type "FILE NOT FOUND". To restart the editor type "START 1000".

FILE 'filename'

The FILE request writes the EDITOR file out on a file called 'filename'. If 'filename' is not given the EDITOR assumes it is to write the EDITOR file out on the original file specified in the EDIT or LOAD commands. If no filename has yet been used with the EDITOR or two or more filenames have been used, the EDITOR will type NAME = ? as an error message. The user must reissue the request with a filename.

QUIT

This request returns control to DOS without writing the editor file.

INPUT
INPUT (PTR)

The INPUT request places the user's console in high-speed INPUT mode, after typing the message 'INPUT'. Paper tape can be read on input with INPUT (PTR) but no erase, kill, or escape characters are interpreted, and logical tabs are interpreted. A blank line returns to EDIT mode.

PUNCH 'n'

The PUNCH command will punch N lines on the paper tape punch followed by a .LF., .CR., and a 20 inch trailer. The eighth punch is never punched except for the .CR. character.

ERASE '%'

The ERASE request changes the current erase character to the character '%' (any ASCII printable character). If '%' is any break character, however, the request is ignored. The erase character is initially set to

'"'. Note: The break characters are the current erase and kill characters plus ',',';','+','-', and .NL.

KILL '%'

The KILL request changes the current kill character to the character '%' (any ASCII printable character). If '%' is any break character, however, the request is ignored. The kill character is initially set to '?'.

TABSET 'n1' 'n2' ...'n8'

The TABSET request sets the tabs to the spaces numbered x_1 (up to 8 tabs stops may be set). The x_i must be in numberically increasing order and separated by spaces and must be less than 72.

PTABSET 'n1' 'n2' ...'n8'

Physical tabs are implemented on output and are preset to 0, i.e., no tabs. The console teletype does not have tabs. The time sharing ASR-35 terminals do have tabs and the time sharing editor has the tabs preset to 6 11 21 31 41.

K. String Buffers

NOTE: String buffering is mainly for use under a Q.E.D.-type of editor. A description of it is given here for general information but its value in the present version of the EDITOR is marginal.

The EDITOR has three one line buffers into which character strings can be stored and acted on. The commands which work on string buffers are MOVE and XEQ. (See EDIT REQUESTS). The MOVE command moves one line of code from one specified buffer to another. The buffers are STRA, STRB, STRC (the three string buffers), EDLIN (the buffer containing the latest command line typed) and INLIN (the buffer containing the current line being edited). The XEQ command causes the specified buffer to be taken as a command line and executed.

The value of string buffers comes when a certain command, or sequence of commands must be done over and over again in a file. For instance, in a FORTRAN program one might want to delete all occurrences of the line X=3. One could do this by the following sequence of commands:

```
TOP
L X=3
D 1
L X=3
D 1
.
.
.
```

If however, one can use a string buffer the commands are:

```
TOP
I L X=3, D1, XEQ STRA
MOV INLIN STRA
XEQ STRA
TOP
D 1
```

This is much easier to use if there are a large number of changes to be made, but it is still a rather awkward way of doing things. This is because in order to load a string buffer it is necessary to load INLIN and then move the character line to one of the string buffers. This, however, has the unfortunate effect of leaving an extraneous line in the file. Therefore, it is necessary to go back and delete it later.

For another example, suppose that before each labelled COMMON area you want to remove the existing comment card and replace it with another comment. Then one could use STRB to hold the comment to be inserted and do the following:

```
TOP
I C COMMON AREA
MOVE INLIN STRB
```

```
L COMMON, N-1, MOVE STRB INLIN, N2, XEQ EDLIN


D 1
```

Note that the MOVE command writes over what was previously in the buffer it is moving a string into. Therefore in the above example the MOVE command replaces the input line in INLIN with STRB. By using XEQ EDLIN we continually execute the current command line.

MOVE 'buffer' 'buffer'

The MOVE request moves one line of code from one specified buffer to another. The buffers are STRA, STRB, STRC (the three string buffers), EDLIN (the buffer containing the latest command typed), and INLIN (the buffer containing the current line being edited). See the section on string buffers for examples of use of the request.

XEQ 'buffer'

The XEQ request causes the specified buffer to be taken as a command line and executed. The possible buffers are STRA, STRB, STRC, EDLIN and INLIN. See the string buffer section for examples of use.

* This is a request meaning "execute (XEQ request) this command line again". It is short for XEQ EDLIN. Example to delete everything in a file from the current line to the top of the file, type

```
DELETE, NEXT -1,*
```

TOP is typed when done.
To type out all DO statements in your program type

```
TOP

LOCATE DO, PRINT, *
```

XVI BINARY EDITOR

A. Introduction

EDB, a binary editor useful for operation on DDP-516 loader-compatible object text blocks as generated by the FORTRAN Compiler and DAP Assembler programs, is available for creating and updating library subroutine files on disk or paper tape. EDBIN is operational under both JULYDOS and TSDOS (note, however, that the high-speed paper tape reader/punch does not operate under TSDOS).

B. Features

(1) Input may be from disk or paper tape; output may be to disk or paper tape.

(2) Multiple input files may be open concurrently.

(3) A large command set for creating and updating binary files.

(4) Explicit error messages.

C. Initialization Procedures

EDBIN is loaded and initialized (at location 1000_8) by inputting, through the ASR keyboard, a command line beginning with 'EDB'. In general, the command line for initialization is as follows:

EDB (INPUT FILENAME) (OUTPUT FILENAME)

If either the input or output file is on paper tape, the appropriate filename is '(PTR)'. Output is optional, and, as a result, an output file need not be specified. Note that, when an output filename to disk is specified, disk storage is allocated to a file by that name.

When properly initialized, EDBIN types 'ENTER' and then loops for user command input.

D. User Commands

Commands to EDBIN consist of the following (parenthesis means optional):

VERIFY	Places the editor into 'verify' mode. All subroutine names and entry points, as they are encountered by EDBIN, will be output to the teletype. Unless otherwise specified by a TERSE or BRIEF instruction, EDBIN is assumed to be in the 'verify' mode.
TERSE	Places the editor into 'terse' mode. The first name only of each subroutine name block as encountered by EDBIN will be output to the teletype.
BRIEF	Places the editor into 'brief' mode. This inhibits the printout of all subroutine names and entry points as they are encountered by EDBIN.
TOP or T	Moves the binary location pointer to the top of the binary input file. Useful only when the input file is on disk.
OPEN (NAME)	Opens a file for writing only (output file). NAME must be specified for opening a file on disk. Close out previously open output file if any.
NEWINF (NAME) or N (NAME)	Closes the current binary input file and opens a new file for reading only (input file). The binary location pointer will be placed at the top of the new file. NAME must be specified for opening a new file on disk.
FIND NAME or F NAME	Moves the binary location pointer to a position on the input file (disk or Paper tape) corresponding to the beginning of a subroutine which includes NAME as an entry point. If NAME is not encountered on the input file, '.BOTTOM.' will be output to the teletype.

COPY NAME or C NAME	Copies all main programs and subroutines (omitting special action blocks; i.e., end of tape blocks, library mode flag blocks, force load flag blocks, etc.) from the current position of the binary location pointer to the beginning of a subroutine called NAME or to the beginning of a subroutine which includes NAME as an entry point. If NAME is not encountered, copying will proceed to the bottom of the input file and '.BOTTOM.' will be output to the teletype. This command moves the binary location pointer.
INSERT NAME or I NAME	Opens a second file for reading only and copies it entirely (omitting all special action blocks) onto the output file. Upon completion, the second input file is once again closed. During an INSERT, the binary location pointer is unmoved and remains positioned on the original input file. An INSERT command operates only when the second input file and the original output file are both on disk (however, the original input file may be paper tape).
GENET (G) or G (G)	Copies the subroutine to which the binary location pointer is currently positioned and follows it with an end-of-tape mark (203_8, 223_8 on paper tape; zero word on disk). If a global copy is requested, all subroutines from the current position of the binary location pointer are copied, each followed by an end-to-tape mark. When the bottom of the input file is encountered, '.BOTTOM.' is output to the teletype. This command moves the binary location pointer.
OMITET (G) or O (G)	Copies the subroutine to which the binary location pointer is currently positioned. If a

	global copy is requested all subroutines from the current position of the binary location pointer are copied, omitting all special action blocks. When the bottom of the input file is encountered, '.BOTTOM.' is output to the teletype. This command moves the binary location poi pointer.
QUIT or Q	Closes all files and exits to the operating system. (When paper tape is the output file, and end-of-tape mark is punched before closing).

Thus far, all commands with the exception of GENET completely disregard (i.e., omit) special action blocks. The following instructions to EDBIN are included specifically for generating special action object blocks on the output file.

RFL	Generates a reset-force-load-flag (library mode) block on the output file. This block is used to initialize a true library file and enables the selective loading of subroutines within the file until a set-force-load-flag (force load mode) block is encountered. This command operates only when output is to disk.
SFL	Generates a set-force-load-flag block on the output file. This block is used to reenable the DDP-516 Loader Program in force load mode. In this mode all subroutines, whether or not required, will be loaded (assumption is that the user knows what he is doing). A true library file should be terminated by an SFL block followed with an end-of-tape mark. This command operates only when output is to disk.
ET	Generates an end-of-tape mark on the output file (203_8, 223_8 on paper tape; zero word on disk).

EDBIN opens all lines with 'ENTER' and marks most errors with '?'. Other error messages include

FILE NAME DOES NOT EXIST OR ALREADY OPEN

USER MUST SPECIFY INPUT FILE

YOUR INPUT FILE LOOKS LIKE SOURCE CODE

CHECKSUM ERROR-IRRECOVERABLE

BLOCK ERROR-IRRECOVERABLE

E. Examples

Several examples are given below, each with an example and step-by-step explanations. In each description UPPER CASE indicates machine response whereas lower case is user action. Letters to the left are references to notes which follow.

1. To delete routines from a library.

EX. In user's drectory he has a library, LIBE, containing 6 files. He wishes to create a new library, without 2 of the routines in it.

contents of LIBE

```
ROUT 1
TEST 1 <------- } to be removed
TEST 2 <------- }
ROUT 2
More
Again
```

```
Attach user's
OK
```

a) edb libe file
GO

b) ENTER, brief

c) ENTER, copy testl
BOTTOM

d) ENTER, find rout2

e) ENTER, copy all
BOTTOM

f) { ENTER, et
{ ENTER, q
OK

Notes

a) This command puts you into the binary editor. The first name following edb is the name of the original file. The second name is the name of the created file. If you intend to edit the file, you must include the 2nd name--different from the original one. If you do not, the machine will balk at first "copy" instruction it receives--it does not know where to copy on to.

b) When you first enter edb, after ENTER, if you put "brief" the system will not type out everything it does. The default option is "verify".

c) The "copy" command will copy (into the created file) all the files up to but not including the file name you give it. Here routl (but not testl) is copied into the file FILE.

d) The "find" command sets the pointer at file name given (Rout2).

e) "copy all" will copy everything from ROUT2 (where the pointer is) to the end of LIBE into FILE.

f) These two commands generate an end-of-tape mark and put you back into D.O.S.

NOW

contents of LIBE	contents of FILE
ROUTl	ROUTl
TESTl	ROUT2
TEST2	MORE
ROUT2	AGAIN
MORE	
AGAIN	

If you wish to retain the name "LIBE" you must just "delete libe" to get rid of old version then "name file libe" to change the name "file" to "libe".

2. To put subfiles of one library into separate files.

Ex. In user 1 directory FILIN exists. It contains 3 different subroutines FILE1, FILE2, FILE3. You wish to put these in 3 separate files to be handled individually.

contents of FILIN

FILE1
FILE2
FILE3

attach user

a) edb filin file1
GO
ENTER, brief

b) ENTER, copy file2
BOTTOM

c) ENTER, et

d) ENTER, open file 2

e) ENTER, copy file 3
ENTER, et

f) ENTER, open file 3
ENTER, copy all
ENTER, et
ENTER, q
OK

Notes

a) You go into the binary editor with original file name FILIN and indicate you wish to create a file named FILE1.

b) "copy" copies into file1 from filin all things up to, but not including file2.

c) Open file 2 will close file 1 and open file 3 for writing.

d) Indicate how much to copy (in this case you indicate that copying is done from where the pointer is up to, but not including file3).

e) "copy all" will instruct that everything from where the pointer (at file3) is to be copied.

NOW

contents of FILIN

FILE1
FILE2
FILE3

Aside from FILIN, your directory also contains three more files --(the binary versions of) file1, file2 and file3. If you wish, you may delete FILIN now.

3. To combine files under one name.

Ex. USER1 now has FILE1, FILE2 and FILE3 in his directory. He wishes to combine them under 1 name.

User1's directory

file1
file2 remember---these are all binary versions of the
file3 programs.

```
     attach user 1

a)   edb file1 filout
     GO
     ENTER, brief
     ENTER, copy. all
     BOTTOM

b)   ENTER, insert file2
     BOTTOM
     ENTER, insert file3
     BOTTOM
     ENTER, et
     ENTER, q
     OK
```

Notes

a) The file to be put in lst is file1 so it is the lst name after edb. FILOUT is indicated as the file to be created.

b) To add more files (after file1) all you need do is "insert" them in order. No need to go back to D.O.S. as long as they are the binary versions of the program.

NOW

user1's directory

file1
file2
file3
filout

XVII INTERACTIVE DEBUGGING AIDS

A. DEBUG

DEBUG, a programming aid for on-line debugging, is loaded (into location 777_8 and 64000 through 64777_8) and initiated in the extended mode under both JULYDOS and TSDOS by inputting, through the ASR, the command word 'DEBUG'. When initiated (at 64000_8), DEBUG types a '$' and then loops for user input.

Commands to DEBUG are as follows (parenthesis mean optional):

$A	Dump, in octal, the contents of the A-Register
$A 'OCTAL VALUE'	Set the contents of the A-Register to the specified octal value
$B	Clear all breakpoints
$B L1	Set breakpoint at L1 (4 maximum)
$B L1 L2 (L3)	Set breakpoint at L1, dump L2-L3 on break
$C (N1)	Continue execution from breakpoint N1 times
$D L1 (L2)	Dump, in octal, L1-L2, open L1 for new value
$.LF.	Dump next location, open it for new value
$'OCTAL VALUE"	Replace current open cell with specified octal value, then dump next
$M L1 L2 M1 M2	Dump every cell in L1-L2 for which AND(M2, XOR(LIST(L), M1)) .EQ.O
$S (L1)	Start execution at L1. When no L1 is specified, execution begins at location specified in the last '$S L1' operation (under JULYDOS,

when L1 has never been specified, execution will proceed to supervisor)

$Z L1 (L2)	Zero all cells in L1-L2
?	Ignore all characters from last '$' to current '?'

DEBUG opens all lines with '$' and marks all errors with '?'.

B. HAMBUG

Hambug is a run time debugging aid written by Jim Hamilton. It occupied locations 20000-22777 vital and 777 octal with a starting address of 20000 octal.

The program expects input from the typewriter in the form of command lines. Any command line may be modified by the kill character (?) or the erase character (#). The terminating character (carriage return) will cause Hambug to execute the command line.

The main advantage of HAMBUG over debug is the ability to dump locations in instruction formats. Other major advantages are the ability to dump or change locations in character or decimal mode.

The following is a description of the commands. Actual commands are underlined.

I. Mode Changing Commands

a. I - set mode to instruction
b. D - set mode to decimal
c. O - set mode to octal
d. H - set mode to character.

Mode is initially set to octal.

II. Dump Core Locations

a. 1000 - dump location 1000 octal in the mode previously set

If location 1000 contains 120240 octal, output will be:

```
1000     121240     octal mode
1000     -20319     decimal mode
1000                character mode

      (space space)

1000    JST*1240    instruction mode
```

See Section VIII for a detailed description of the modes.

b. 1000,1003 dump locations 1000 to 1003 inclusive

If 1000-1003 contain and mode is octal, output is:

```
1000     000001
1001     000001
1002     000001
1003     000001
```

c. 1000/3 - dump 3 locations
starting at 1000

1000+5-1/3-1 - dump 2 locations
starting at 1004

III. Change Core Locations

a. Octal Mode

To put 3 location 200 the command is:

U 200 3 - Update location 200 will 3

To put 77 in location 200 and 277 in location 201 and 177 in location 202:

U 200 77 277 177

To put 101 in locations 5000 to 5005:

U 5000,5005 101 or
U 5000/6 101

b. Decimal Mode

Update location 1005 octal with 97 decimal

U 1005 97

Example of other update commands in decimal mode are:

U 10/100 297

U 103, 104 9971

c. Character Mode

Update 1000 with characters CH
1001 with characters AR
1002 with characters AC
1003 with characters TE
1004 with characters R
is done with the command below

U 1000/CHARACTER/

The / character in this case acts as a delimitor although any character except ?, #, can be used.

The same result can be achieved by

U 1000 $CHARACTERS$

d. Instruction Mode

Updating while in instruction mode is illegal.

IV. Start Command

Hambug stores within itself saved values of the active machine registers. These values are initially set to zero. These registers may easily be accessed and changed using HAMBUG.

S 1000 - - Move saved A, B, X, K registers to the
- active registers and jump to 1000
A - - dump saved A register
Q - - dump saved B register
X - - dump saved X register
K - - dump saved K register
U A 1000 - Update saved A register with 1000
S - This command is meaningless and will not work as it does in DEBUG.

V. Breakpoint commands

B 1000

Put breakpoint at 1000. When user program is started by use of the S command, whenever control passes to location 1000, control will trap to HAMBUG. Active registers are stored away within HAMBUG and "BREAK", is typed out.

P

Proceed from breakpoint. Move saved active machine registers to A,B,X,K and start at the location of the breakpoint +1.

B 1010

Remove breakpoint from 1000, insert one in 1010.

B

Remove breakpoint.

VI. The Special Character (.)

Period (.) always has the value of the 1st location dumped or updated on the most recent dump/updat command. (.) can only be used with dump commands.

Example of use
Octal mode

1000/1

output is

1000 000000

./1 (Immediately after last command)

output is

1000 000000

.-1/1

output is

777 000000

2000/10 followed by ./1 will give

2000 000000

VII. Carriage Return

Carriage return typed alone is the same as the command to typed the next location.

VIII. Format of Modes on Dumping

a. Instruction Mode

All generic instructions are printed with no address.
All input-output instructions are printed with 4 octal digits following the instruction.
All memory reference instructions are printed along with 5 octal digits which represent the actual address referenced by the instruction (indirect references not traced out).
All shift instructions are followed by the decoded shift count.
If the indirect bit is set, a * is printed adjacent to the instruction.
If the index bit is set, a ,1 is printed after the address part of the instruction (with the exception of STX and LDX instructions).
If the word resembles no instruction, Hambug will type out the letters *** followed by 6 octal digits.

Examples

location	octal	instruction
15000	101040	SNZ
15001	130001	INA 0001
15002	103010	JMP* 15010
15003	142010	JMP* 00010,1
15004	141077	LLL 01

b. Octal Mode

All locations are printed as six octal digits.

c. Decimal Mode

All locations are printed as a variable number of

digits and as a negative number if the leftmost bit is set.

Examples

octal	decimal
000000	0
000014	12
177777	-1
77777	32767

d. Character Mode

All locations are printed as 2 characters

Example

octal	character
146706	MF
142240	D␣

IX. Miscellaneous

Multiple commands may be typed on the same line.

Example

```
0 1000 D 1000
```

If 1000 contains 12 decimal response is
1000 000014
1000 12

List of all commands

A	refers to saved A register
B	insert and remove breakpoint
D	set mode to decimal
H	set mode to character
I	set mode to instruction
O	set mode to octal
P	proceed from breakpoint

Q	refers to saved B register
S	start command
U	update command
X	refers to saved index register
(carriage return)	dump next location
(.)	refers to current location
to dump any location, type its address in octal	
K	refers to saved keys
?	kill character
#	erase character

C. TRACE

Introduction

A new debugging routine X16DEBUG has arrived from Honeywell and promises to be a great improvement over the current DEBUG. This memo contains a description of its availability and use.

Availability

X16DEBUG is currently available on DOS by typing TRACE. It is one sector long and starts at 64001.

Command Structure

Each command consists of a single-letter function code; followed by a colon and one or more octal values. Values are separated by commas, and the last value used must be followed by a carriage-return.

Values are right-justified octal integers, if no digit follows a comma, the value is made zero.

First value is a starting address.

Second value, if used, is usually the end address of the block started by the first value (function C,D,F,H,L,S,V), or an initial content for register 'A' (functions E,J, M,R,T).

Third value, if used, is either the start address of another block (functions C,V) or an initial content for register 'B'(functions E,J,M,R,T), or an object pattern (functions F,S).

Other values are interpreted variously (see specific functions).

If values V4 and/or V5 are unspecified, they are set to all one's (177777).

ERRORS

A slash may be used to abort any input and return to start.

Function code is defined by last character before colon. If wrong character is keyed, follow it with Proper letter code for desired function.

Octal value fields ignore all characters except 0,1,2,3,4, 5,6,7,/, *,comma,carriage-return. To cancel an incorrect octal value, key in "*".

Note - If more than 5 digits are keyed, last 16 bits are used. Any attempt by the user program to change extended, nonextended mode during interpretive execution is trapped and the message CA is typed followed by a dump of the instruction warning; the GOTS DDP-516 has been modified so that, in the extended addressing mode indexing will occur before indirection if the effective address is in sector 0 and is less than 32. The interpretive execution part of TRACE has been modified to work correctly for our machine. Except for routines which have been written in DAP by the user, this should be no problem.

USE

Access Memory Words

A:V1(CR)

-Access word(s) in memory (starting) at location V1. The 'DEBUG' program types out the address, V1, and its content, then waits for keyboard input.

-To change the content, key in the new octal value, followed by a 'CARRIAGE-RETURN'. The program then types out the next higher address and its content.

-To progress to the next higher address without changing the content of the current location, key in a 'COMMA'. (Characters keyed in before this 'COMMA' are ignored.)

-The look/change cycle continues until the operator keys in a '/'.

Breakpoint Set

B:V1(CR)

-Insert breakpoint link in object program at location V1.

-If object program is later executed, and if control reaches location V1, an indirect jump through location '00060 returns control to the 'DEBUG' program, which prints the register contents, then awaits further commands.

-Print format is given under functions 'E' and 'R'.

-Only one breakpoint can be inserted in a program, only at the execution time, and is removed after each use. However, the breakpoint address is retained for re-use, and requires user action only to change it.

-If object program does not reach breakpoint, stop computer manually, and re-start in 'DEBUG' at XX007.

-To remove breakpoint completely, key in B:1(CR).

Copy Memory Block in Memory

C:V1,V2,V3(CR)

-Copy memory block at locations V1 through V2 into block at locations V3 through V3-V2-V1. If V2 does not exceed V1, only the word at location V1 will be copies (into V3). If V3 lies between V1 and V2, the block between V1 and V3-1 will be repeated cyclically until location V3-V2-V1 has been written into.

Dump Memory Block to Typewritter

D:V1,V2(CR)

-Dump memory block at locations V1 through V2 to ASR typewriter. The basic typing format is 8 octal words per line, preceded by the octal address of the first word printed on the line. (if this address does not

end with a zero digit, appropriate spaces are inserted to maintain column integrity.) Repetitious words are suppressed as follows:

1) If the remainder of the current line is identical to word last printed, the line is terminated.

2) If one or more subsequent lines are identical to word last printed, typewriter skips one line.

Execute Subroutine

E:V1,V2,V3(CR)

-Execute subroutine by performang 'JST' to location V1.

-Prior to subroutine entry, V2 is loaded into register A, and (except on the DDP-416) V3 is loaded into register B.

-The subroutine return should be via indirect jump through its entry point, incremented by 0, 1, or 2.

-Upon return from the subroutine, the 'DEBUG' program prints the register contents as noted under function 'R', except that one or two meaningless words may precede the specified format to indicate that the subroutine has incremented its return link by 1 or 2.

High Speed Punch in Self Wading Format

H:V1,V2,(CR)

-High-speed punch, PAL2 format, of memory block at locations V1 through V2.

Jump Trace Object Program (Dynamic)

J:V1,V2,V3(CR)

-Dynamically trace object program starting at location V1, with registers A and B initially set to V2 and V3, respectively. A diagnostic printout is produced prior to the interpretive execution of any object JMP or JST or HLT. (See function 'T' for further details and restrictions.)

Low Speed Punch in Self Loading Format

L:V1,V2(CR)

-Low-speed punch, PAL2 format, of memory block at location V1 through V2. Operator should turn on ASR punch approximately two seconds after keying 'CR'.

Monitor Object Program for Effective Address (Dynamic)

M:V1,V2,V3,V4(CR)

-Dynamically monitor object program starting at location V1, with registers A and B initially set to V2 and V3, respectively. A diagnostic printout is produced prior to the interpretive execution of any object memory-reference instruction whose effective address equals V4. (See function 'T' for further details and restrictions.)

Patch Object Program

P:V1,V2(CR)

-Insert patch to V1 in object program at location V2, by replacing instruction at V2 with jump to location V1. Storing the displaced instruction at V1, and entering function 'A' with value V1.

-Operator must key in desired patch, with suitable return.

-Note...V1 and V2 must lie in the same sector.

Run Object Program

R:V1,V2,V3(CR)

-Run object program by performing 'JMP' to location V1. Prior to program entry, V2 is loaded into register A, and V3 is loaded to register B.

-Control does not return to the 'DEBUG' program unless a breakpoint is encountered, or the operator takes over manual control via the computer console.

-If a breakpoint is encountered, the print format is:

DDP-516: INSTR (A) (B) (X) (KEYS)

WHERE KEYS REPRESENT VALUES OF C, DP, PMI, AND SC.

Search Memory Block Under Mask

S:V1,V2,V3,V4(CR)

-Search memory block at locations V1 through V2 for words equal to V3 under the mask V4. (If no mask is specified, the entire word is tested.) When a match is found, the address and its content are typed out, and the search continues until location V2 has been tested.

Trace Object Program (Dynamic)

T:V1,V2,V3(CR)

-Dynamically trace object program starting at location V1, with registers A and B initially set to V2 and V3, respectively. A diagnostic printout is produced prior to the interpretive execution of each object instruction.

-Printout is formatted as eight octal words, representing:

DDP-516: (P) INSTRA EA EA (A) (B) (X) (KEYS)

NOTE...FOR NON-MEMORY-REFERENCE INSTRUCTIONS, THE THIRD WORD = '077777 AND THE FOURTH REPEATS THE INSTRUCTION WORD.

T:V1,V2,V3,V4(CR)

-Same as above, but printout produced only when P=V4.

T:V1,V2,V3,777777,V5(CR)

-Same as above, but printout produced every V5 instructions.
Note. . .If V5 is negative, its absolute value is used. If V5 is zero, it is treated as 65536.

T:V1,V2,V3,V4,0(CR)

-Same as above, but printout produced the first time P=V4, and every instruction thereafter.

-Restrictions on all 'T' functions (including 'J' and 'M')

1. HLT instructions always cause printout, followed by a STOP. Tracing resumes when the 'RUN' button is actuated.

2. Interrupts are executed in real time, not in interpretive mode. Tracing is resumed when the interrupt routine exists.

3. Tracing of input-output routines is only possible if no timing constraints are violated by the fact that internal speeds are reduced by a factor of about 60 (AVG) to 80 MAX) when no printout is involved.

Verify Memory Block Against Copy in Memory

V:V1,V2,V3(CR)

-Verify memory block at locations V1 through V2 against a copy in locations V3 through V3-V2-V1. The program types the address and content of each location in the V1 block which does not match corresponding word in the V3 block.

XVIII. OVERLAYS

A. Introduction

Many users have programs which either overrun memory or overflow the link table.

Two system subroutines OVERLA and RETURN have been written that allow a user to overlay one core image by another, then return to continue execution of the original core image if the user desires. Communication between the two core image programs is through COMMON, which is not overlaid. Any number of overlay segments may be used and to any depth.

To make use of the overlay feature in an efficient manner, the user must break his program up into two or more semi-independent parts. Overlays will not be useful if the segment is called into core thousands of times during the execution of your program. Overlay segments take on the order of 1 second to load into the machine.

B. Simple Example of Overlay

This example consists of a main program and one overlay segment.

PROG 1

```
      COMMON NUM
      CALL TIDEC (NUM)
      CALL OVERLA (6H*SEG␣␣,0,0,0)
      END
$0
```

SEG

```
      COMMON NUM
      CALL TODEC (NUM)
      CALL EXIT
      END
$0
```

Each program is compiled and loaded separately as if they were two independent programs. Each program is loaded with the FORTRAN library and any other routines until a load complete

message is given. The core images saved do not include COMMON. In this case each core image will occupy from 100 to 2000 octal, approximately.

To start the program, the user types R *PROG1. The program

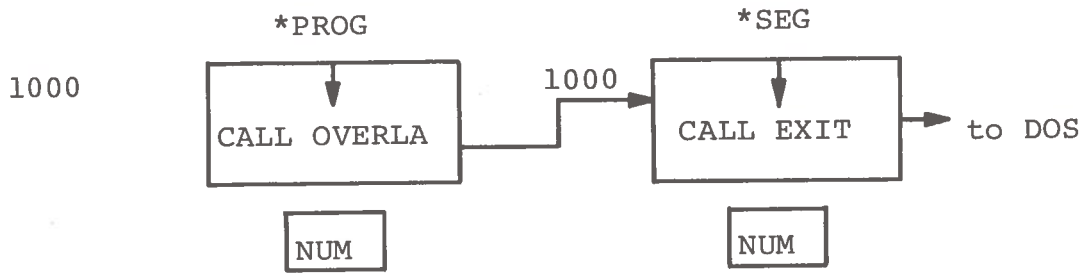
1. waits for the user to type a number

2. calls subroutine OVERLA with the first argument a six character hollerith string equal to the name of the saved overlay segment *SEG and starts execution at 1000 octal in this case.

3. subroutine OVERLA resumes the file *SEG as if the user had typed RESUME *SEG on the teletype. This action reads in the core image *SEG and starts execution at 1000 octal in this case.

4. the program *SEG types out NUM on the teletype then returns to DOS through CALL EXIT. The common variable NUM was not changed by the resuming of the program *SEG.

Pictorially, this is how the flow of control went



C. Overlay with Control Returning to Main Program

PROG1

```
            COMMON NUM
C           I=1
            CALL TIDEC(NUM)
            CALL OVERLA(6H*SEG␣␣,6H*PROGX,64,1024)
            CALL TODEC(I)
            CALL EXIT
            END
$0
```

```
                COMMON NUM
C
                CALL TODEC(NUM)
                CALL TONL
                CALL RETURN(6H*PROGX,0,0,0)
                END
$0
```

The above two programs are compiled and loaded separately as before. To start the program type R *PROG1. The program

1. sets I to 1. It was 0 as are all unset variables at the beginning of execution.

2. waits for user to type a number

3. calls subroutine OVERLAY with the same first argument as before. The second argument is the name of the file on which to write the present core image. The third and fourth arguments are the beginning and ending locations of the present core image.

4. subroutine OVERLAY writes out the present core image with name *PROGX with limits 64 and 1024. The saved program counter is set so that when *PROGX is resumed, execution will start at the statement after CALL OVERLA. Subroutine OVERLA now resumes *SEG as in the first example.

5. the program *SEG types out NUM on the teletype, followed by a new line

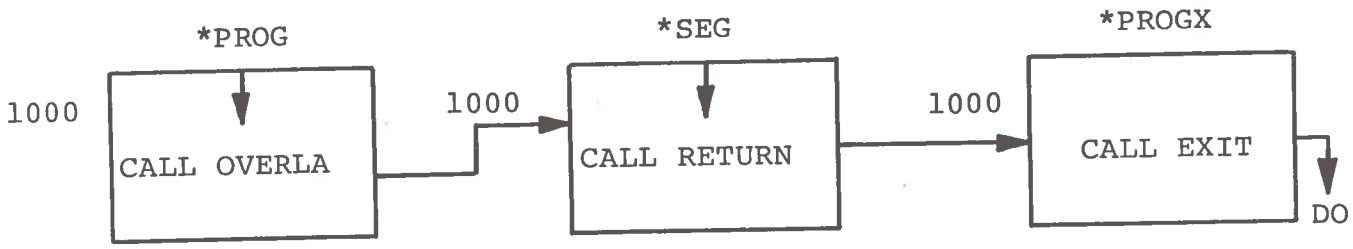
6. *SEG calls subroutine RETURN with the first argument equal to the saved file to resume execution

7. subroutine RETURN resumes the file *PROGX. Execution starts at the statement following CALL OVERLA in that saved file

8. *PROGX (which is the same program as *PROG except in a different stage of execution) types out I which has previously been set to 1

9. control is returned to DOS.

Pictorially, this is how the flow of control went.



D. Description of Subroutines OVERLA and RETURN

```
CALL OVERLA(infile, outfil, sa, ea)
CALL RETURN(infile, outfil, sa, ea)
```

Subroutine OVERLA checks the second argument to see if it is a name. If so, it saves a core image from location sa to location ea as name outfil. Outfil must be a six character hollerith string or a three word array containing such. A program counter is saved with the core image such that when outfil is resumed by a CALL RETURN, control passes to the statement following the CALL OVERLA. Subroutine OVERLA, after saving away the file outfil if that parameter is not zero, then resumes the file with the six character hollerith string infile, just as if the user had typed RESUME INFILE.

Subroutine RETURN is identical to subroutine OVERLA. Only the name is different to help the programmer keep straight what he is doing.

E. Several Overlay Segments, Each Returning to Main Program

Example:

PROG1

```
      COMMON NUM
      CALL TIDEC(NUM)
      CALL OVERLA(6H*SEG1␣,6H*PROGX,64,1024)
      CALL TIDEC(NUM)
      CALL OVERLA(6H*SEG2␣,6H*PROGX,64,1024)
      CALL TIDEC(NUM)
      CALL OVERLA(6H*SEG3␣,6H*PROGX,64,1024)
      CALL TODEC(NUM)
      CALL EXIT
      END
$0
```

SEG1

```
      COMMON NUM
      CALL TODEC(NUM)
      CALL RETURN(6H*PROGX,0,0,0)
      END
$0
```

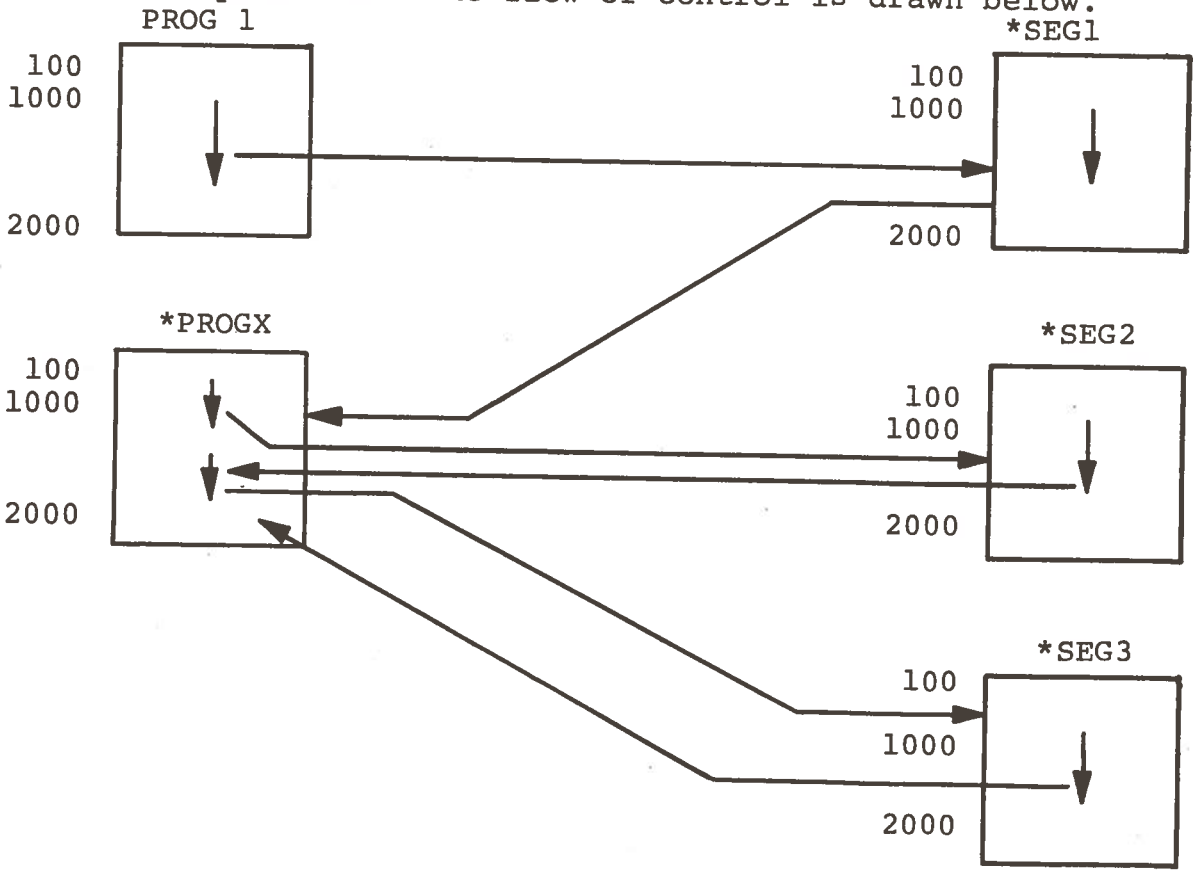
SEG2

```
      COMMON NUM
      CALL TODEC(NUM-10)
      CALL RETURN(6H*PROGX,0,0,0)
      END
$0
```

SEG3

```
      COMMON NUM
      CALL TODEC(NUM+10)
      CALL RETURN(6H*PROGX,0,0,0)
```

The reader is urged to figure out the output of the program and the steps taken. The flow of control is drawn below.



Notice that when each segment is brought in, the same file name is used to store the current main program core image. The name should never be the same name as the file the user resumes. Notice that no segment must be saved by the overlay procedure because no segment is ever entered twice.

F. Nested Overlay Segments

This section contains an example of overlay segments that call in other overlay segments then return to the main program. Overlay segments may be nested to any depth.

Example:

PROG1

```
COMMON NUM
CALL TIDEL(NUM)
CALL OVERLA(6H*SEG1 ,6HPROGX,64,1024)
CALL TODEC(NUM)
CALL EXIT
END
$0
```

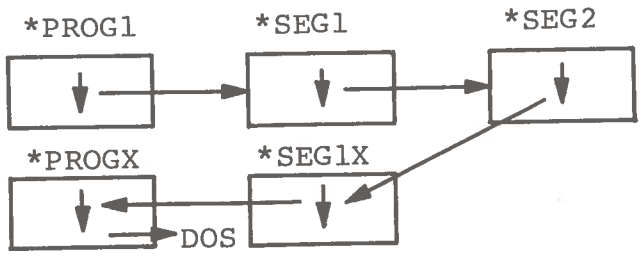
SEG1

```
COMMON NUM
CALL TODEC(NUM)
CALL TONL
CALL OVERLA(6H*SEG2 ,6H*SEG1X,64,1024)
CALL TODEC(NUM-10)
CALL RETURN(6H*PROGX,0,0,0)
END
$0
```

SEG2

```
COMMON NUM
CALL TODEC(NUM+10)
CALL TONL
CALL RETURN(*SEG1X,0,0,0)
END
$0
```

Flow of control:



G. Problems in Using Overlays

1. A user must be very careful in using overlays with programs that use interrupts. A user must either make sure his interrupt handling routine and any routines it uses are never overlaid or turn off the interrupts, overlay, then turn the interrupts back on. In particular, graphics users should turn the display off, overlay, then turn the display back on through calls to CLKON and CLKOFF.

2. Debugging requires care. Case 1 is a user who suspects a bug in his main program. Using DEBUG, he should insert a breakpoint as normal. Case 2 is a user who suspects a bug in his overlay program. The user should restore the program, insert the breakpoint, then save the overlay program away as if he had just loaded the program. When the breakpoint is encountered in the overlay program the expected action takes place.

3. The word GO is typed every time a program is loaded by OVERLA or RETURN. This tells the user he has entered a new core image. If this output is annoying or delays a program give the command KILGO before running your program. KILGO will prevent GO from being typed.

XIX. MOVING LARGE PROGRAMS AND DATA IN AND OUT OF THE DDP-516

TCC personnel will get large programs in and out of the DDP-516 if users request - ask Jan Carlson room 11-27. If users wish however, they may do the work themselves.

A. Input from Cards

The user or operator must use the H832 and the DDP-516. The user should have prior experience with the H832 before using it. The card deck should be punched on an 0.29 card punch and end with two cards containing $EOF starting in column 1.

Step 1 - Turn on the H832.

Turn the key of the main processor to the on position. Load the disk pack labeled DOS MASTER on the disk drive zero and turn the drive on. Load your cards in the card reader. Turn the power on, the transport on and push the START button.

Step 2 - Boot the Disk Operating System.

Turn to the console: All switches on the console should be in the middle position except three. The abs-of switch should be in the abs position; the device select switches should indicate 13 in binary where "on" is down; and the IOP 1 switch should be down. Push the system and start buttons. The disk should click for a few seconds and the following lin should be typed on the console teletype - "DATE"

Step 3 - Initiate Card Reading

The user should now type the following

```
user:      type date
response:  TIME=
user:      type time
response:  OK
user:      CARDIN
```

Turn to the DDP-516 teletype and type CARDIN name, where name is a filename in which the cards will be stored. Cards should be read by the card reader. When done, the 516 must be restarted at 70000 octal and the CLOSE ALL command given.

B. Large Listings

Source files longer than five pages are usually printed on the line printer when the user wants an up-to-date listing.

Step 1 - Turn on the H832
See step 1 of part A for details

Step 2 - Boot the Disk Operating System
See step 2 of part A for details

Step 3 - Initiate Line Printer Output

The user should now type the following

```
user:       type data
response:   TIME=
user:       type time
response:   OK
user:       LPOUT
```

Turn to the DDP-516 teletype and type LPOUT name, where name is the file to be sent to the line printer. Other files may be sent by repeating the command to the 516 with new file names. When done, the CLOSE ALL command should be given.

C. Input from Paper Tape

Users will rarely use paper tape for input. Paper tape input is used to put new or revised software delivered by Honeywell or some other 516 users. The paper tape can be in one of three forms - source code, object code, or core image.

To Load a source code tape into the DOS file system; put the tape in the paper tape recorder, then

```
user:       ED
response:   INPUT
user:
response:   EDIT
user:       INPUT(PTR)
response:   paper tape is read in; EDIT (if EDIT is not typed,
            restart machine at 1000 octal and type CR.)
user:       FILE 'filename'
response:   OK
```

to load object code, put the tape in the paper tape reader, then

```
user:       EDB (PTR) 'filename'
response:   ENTER

user:       BRIEF
response:   ENTER

user:       COPY ALL
response:   tape is read into file 'filename'
            ENTER

user:       ET
response:   ENTER

user:       QUIT
response:   OK (back in DOS)
```

To load a core image tape, check the label to make sure it will not load over the operating system (locations 66120 to 77777 octal). Put the tape in the reader, then

```
user:       START 1
response:   tape is read in, halts the machine when done or
            returns to DOS.

user:       restart the machine at 70000
response:   OK

user:       SAVE 'filename' 'sa' 'ea' 'pc'
response:   OK
```

Saves the core image from location 'sa' to 'ea' with saved program counter 'pc' for use with RESUME.

D. Output to Paper Tape

The user will generate paper tape output only when he wishes to give a paper tape program to another DDP-516 installation.

To punch source code:

```
user:       ED 'filename'
response:   EDIT

user:       PUNCH 10000
response:   file is punched out
            BOTTOM

user:       QUIT
response:   OK (back in DOS)
```

To punch object code:

```
user:       EDB 'filename' (PTR)
response:   ENTER

user:       BRIEF
response:   ENTER

user:       COPY ALL
response:   file is punched out
            ENTER,

user:       QUIT
response:   OK (back in DOS)
```

To punch core image code:

```
user:       RESTOR 'filename'
response:   OK

user:       PALAP 24000 'sa' 'ea' 'pc'
response:   tape is punched
            OK (back in DOS)
```

For a more complete description of the PALAP command see the following section.

E. PAL-AP

Introduction

PAL-AP provides a user with the ability to punch, in 'invisible format', self-loading paper tapes of any desired segment of memory. PAL-AP is presently useful for operation with DOS.

Operating Procedures

PAL-AP is loaded (into locations 24000_8 through 25374_8) and initiated in the extended mode by inputting, through the AST keyboard, a command line beginning with 'PALAP'. When PAL-AP is initiated, a self-loading paper tape will be punched, followed by a return of control to the supervisor (signalled by the typeout of 'OK' on the ASR).

The command line for initiating PAL-AP is as follows:

PALAP 24000 (SA) (EA) (PC) (KEYS) (OUNIT) (BOOTSTRAP)

SA — Starting address of segment

EA — Ending address of segment

PC — Value of program counter. The PC will be set to this value after loading (using the self-loading bootstrap) the punched segment. This enables load and go without user intervention. A PC value of zero will cause a program halt after self-loading, and requires the user to restart.

KEYS — Value of program control keys (C-bit, DP mode, EXTMD, shift count). The keys will be set to this value by the self-loading bootstrap prior to execution of the punched segment.

OUNIT — Output unit. Output (punched tape) may be produced from the ASR-33, ASR-35, or the high-speed punch.

'0' - high-speed punch
'1' - ASR-35
'2' - ASR-33

BOOTSTRAP — Loading address of bootstrap (PAL-AP actually modifies this value to the LOGICAL AND of the specified address with 37000_8, plus 566_8. This enables the bootstrap to be loaded from location 566_8 through location 777_8 of any sector in the lower memory bank). If the specified address is a zero, the default address for loading the bootstrap is 24566_8).

Once a self-loading tape has been produced, the procedure for load and go is as follows:

1) Place the leader portion of the program in the paper tape reader.

2) Type, on the ASR, KEYIN (the command line for KEYIN may include an A-register setting for load-time relocation of the bootstrap. The setting, for proper execution of the bootstrap, must be of the form $XX566_8$).

3) If the PC value in the PALAP command line was set to zero, the program will halt prior to execution. This requires that the user restart with the appropriate

value in the program counter.

Punched Tape Structure

Any PAL-AP punched tape has the following overall structure:

1) No frame on the tape will contain what resembles a rubout character (377_8). This insures compatibility within the framework of TSDOS.

2) PAL-AP will first punch out its own loader (bootstrap) section in '8-8 format', followed by twelve inches of leader.

3) Next, the desired program is punched in 'invisible format' which is recognized by the bootstrap.

The bootstrap is capable of loading itself starting in the location specified by the PALAP command line, loading the user program, and initializing the user program for execution.

The user program is punched in blocks of 50 words each. Salient characteristics of the block structure area are as follows:

1) A start of message character (201_8) is punched at the beginning of each block.

2) Following this, the address of the first memory location is punched in 'invisible format' (4-6-6 code).

3) Each block is ended by a checksum, followed by an end of message character (223_8).

4) Nine frames of leader are punched between blocks and twelve inches of leader are punched at the beginning and end of a program.

The format of the punched words is as follows:

1) Non-zero words are punched in 'invisible format'. Each 16-bit word is written as a four-bit and two six-bit characters on tape. The four-bit character represents the high-order four bits of the word. Each six-bit character has the high-order bit in channel eight and the five low-order bits in channels five through one. Ordinarily, nothing is written in channels eight through five of the four-bit character or in channel six and seven of the six-bit characters.

2) Eight characters cause special action by the ASR. These are 023_8 and 223_8 (x-off), 021_8 and 221_8 (x-on), 012_8 and 212_8 (line feed), 005_8 and 205_8 (wru). These are translated into 173_8 and 373_8, 176_8 and 376_8, 175_8 and 375_8, 174_8 and 374_8, respectively. In the case of each of these characters, channels six and seven are punched.

3) When one or more consecutive zero words are encountered in memory, they are represented by one punched word. This word consists of the two's complement of the number of consecutive zero count words encountered. In order to distinguish these zero count words, channel eight of the high-order (four-bit) character is punched.

Procedures for Punching and Loading Multiple Segments

The structure of PAL-AP is such that multiple segments of memory (contiguous or non-contiguous) may be punched and later loaded with a minimum of user responsibility. As an example, consider the following:

A user types the command line -

```
PALAP 24000 100 777 1
```

PAL-AP will now punch locations 100_8 through 777_8 to the high-speed punch and return control to the supervisor.

The user now types a second command line -

```
PALAP 24000 30000 37777 1
```

PALAP will now punch locations 30000_8 through 37777_8 to the high-speed punch and return control to the supervisor.

The user now types a third command line -

```
PALAP 24000 60000 65777 70000 020000
```

PAL-AP will now punch locations 60000_8 through 65777_8 to the high-speed punch and return control to the supervisor.

The punched tape now contains images of all three segments of memory and may be loaded by typing the following command line -

```
START 1
```

This command will load all three segments and cause program initialization at 70000_8 in the extended mode. This is possible because:

1) The PC values for the first two command lines were both set to one (the location of the key-in loader).

2) The PC value for the third command line was set to the desired starting address of the user program (in this case 70000_8).

3) The KEYS for the first two command lines were both set to zero (non-extended mode).

4) The KEYS for the third command line were set to the extended mode (for user program execution in that mode).

XX. DOS SYSTEM MAINTENANCE

This section gives the procedures that should be followed by the person or persons responsible for setting up the DOS system.

To be done daily.

1. Turn on machine

2. Put the DOS MASTER disk pack on disk drive 0 and turn on the disk drive.

3. Boot load DOS by putting the paper tape labeled DOS BOOTLOADER in the reader and starting the computer at location 1. All sense switches should be in the reset position. Response should be OK on teletype.

4. Run FIXRAT, a software maintenance program by typing:

```
user:       ATTACH DIAG
response:   OK
user:       R FIXRAT
```

If any error messages appear see Jan Carlson.

5. Copy the DOS MASTER disk into the daily backup disk by putting the disk pack on drive 1, turning it on and typing:

```
user:       ATTACH DIAG
response:   OK
user:       R COPY
            GO
response:   disk will be copied
            OK
```

6. If the day is Monday or Thursday, copy the Monday or Thursday backup disk by using the procedure of step 5

7. Time Sharing may be run on a regular basis. If so, it is the duty of the system maintenance person to set up Time Sharing and take it down. Starting with the DOS system running, to set up Time Sharing for 7 users:

   put the paging disk on drive 1 and turn it on.

```
user:       ATTACH JDOS
response:   OK

user:       R XXDOS
response:   OK

user:       ATTACH DIAG
response:   OK

user:       RR *TSDOS
response:   GO
            QUIT

user:       ? STARTUP 10 2 1 1 2
response:   LOGIN PLEASE

user:       RESTOR BBFIX
response:   OK
```

To set up Time Sharing for 3 users, don't put the paging disk on and replace the STARTUP command in the above procedure with

STARTUP 10 1 0 1

8. To take down time sharing, type SHUTDN on the console teletype. Response should be TSDOS NOT IN OPERATION. If one wishes to use DOS, it must be boot loaded from paper tape at this point. Turn off paging disk and put it away.

9. To take down DOS, simply turn the machine off.

FIXRAT

A. Introduction

FIXRAT is a JULYDOS software maintenance program. FIXRAT should be run only by authorized system maintenance operators, not general users. FIXRAT checks out a JULDOS disk pack to verify that all files on the disk can be read and have a legitimate format. If there are problems, FIXRAT will repair or delete the broken file and print an error message on the teletype. FIXRAT should be run daily.

B. Review of File System

To explain in detail the operation of FIXRAT, it is necessary to briefly review the file structure of JULYDOS. Information on the disk is recorded in files, and these files

are composed of fixed length records, 432 words each, which are recorded chained together by pointers on the disk. A file is simply an ordered linear array of words known by a single six-character name. The file system has no notion of the contents of the file.

The collection of files belonging to a user is organized into a User-File-Directory (UFD). Each UFD is a file and contains a four-word entry per user file (presently 62 files maximum), 3 words of name (6 characters) and one word for starting record address.

The collection of all UFD files is organized into a Master-File-Directory (MFD) which is itself a file much like the UFD's. (The MFD is also used in TSDOS to hold the names of commonly used disk commands, FORTRAN, NAP, etc.) The MFD begins at record one on the disk.

The files on the disk are recorded as chained strings of fixed length records as shown schematically in Figure 1.

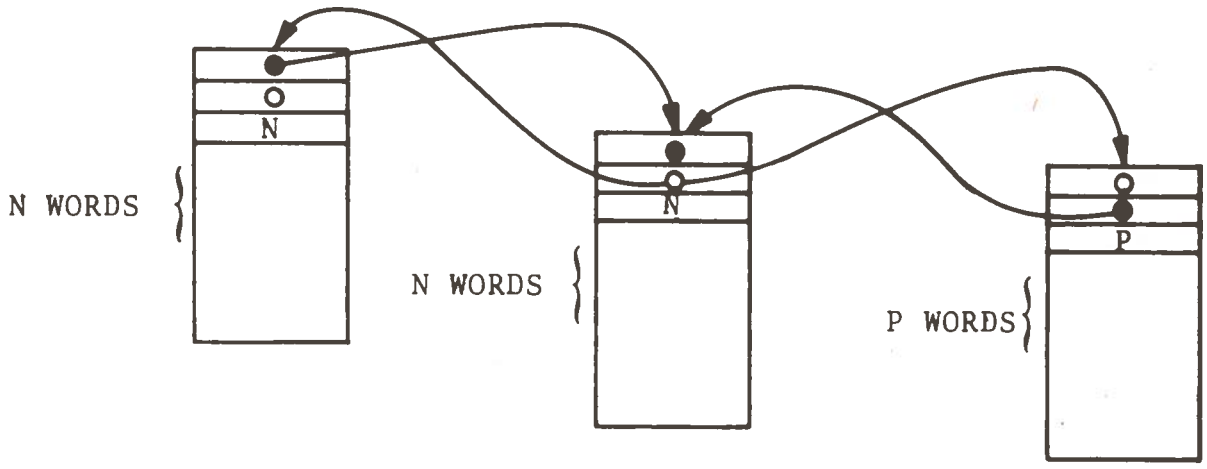


Figure 1. DOS File Structure

Each record of the file has three words at the beginning which function as follows: word one points to successor record (zero if none), word two points to the predecessor record (zero if none), and word three is count of the data words contained in the record. The forward and backward pointers are especially important because they allow easy traversal of

the file forward and backward while at the same time providing a large measure of protection against snow-balling disk errors.

Since files are composed of chained records, the selection of records can be left entirely to the file system. The file system must select free records when a file is to be written or extended, and the free records of a deleted file must be returned for later re-use. For this purpose a Disk-Record-Availability-Table (DSKRAT) is maintained with one-bit for each record on the disk; a "1" means the record is available and a "0" means the record is in use. Since there are 8120 records on the disk, 8120/16 = 507.5 words are required for DSKRAT. DSKRAT is allocated 512 words for simplicity. The DSKRAT is recorded as a file on the disk and its 1st record is on record #0, its 2nd is on record #2.

C. Operation of FIXRAT

Before describing FIXRAT in detail, a word of warning must be given. FIXRAT is intimately connected with the JULYDOS system. If JULYDOS is ever reloaded or modified, FIXRAT may have to be changed. In particular, FIXRAT uses JULYDOS subroutines RREC, WREC, ATTACH, and UPDATE. It also uses the JULYDOS common areas LSMAN and FILDIR. Any redesign of JULYDOS may mean FIXRAT must be changed also. R FIXRAT DISK will check out logical disk drive 0-3 if DISK is blank, ONE, TWO or three, respectively.

Step 1 - FIXRAT causes JULYDOS to attach to the MFD.

Step 2 - FIXRAT picks up the beginning record address of the first file in the directory and calls subroutine REDFIL.

Step 3 - Subroutine REDFIL reads the first three words of each record of the file. These words contain the forward pointer, backward pointer and word count respectively. For each record, REDFIL verifies has correct format for a file, a "1" bit is entered in a "local" DSKRAT called TSTRAT. A counter of all records used is bumped by 1 for each record read.

Step 4 - Step 3 is repeated for each beginning record address of each file in the directory.

Step 5 - FIXRAT causes JULYDOS to attach to the first UFD in the MFD, skipping over the MFD as a special case.

Step 6 - Steps 2 and 3 are repeated for each file in the UFD.

That is, FIXRAT reads every record of every file in the UFD, checks it for proper format, enters the proper bit in the TSTRAT, and counts the record.

Step 7 - Step 6 is repeated for all UFD's in the MFD. In this way every file on the disk is processed.

Step 8 - The TSTRAT table is inverted. Each "0" bit in the table is changed to a "1" bit. Each "1" bit is changed to a "0" bit.

Step 9 - Records 120 to 1177 are set to used. These records are reserved for core memory storage for TSDOS. FIXRAT compares the local TSTRAT table against the JULYDOS DSKRAT table. If they match, a message is printed out to that effect. If they do not match, FIXRAT replaces the JULYDOS DSKRAT with the newly calculated TSTRAT which represents the records the combined directories say are in use at the present time. The program changes a JULYDOS variable CHRAT to TRUE and calls the JULYDOS subroutine UPDATE. This procedure writes the newly replaced DSKRAT on the disk. A mismatch message is typed.

Step 10 - FIXRAT types out the number of records used and unused in octal then returns to JULYDOS command level.

Under no error conditions, FIXRAT types the name of each user directory processed followed by the number of records allocated to files in that directory. Errors induce additional typeout.

The above description should give an overall view of FIXRAT. Step 3 in the description assumes a record is in the correct format for a file. Six types of incorrect formats cuase FIXRAT action. In every case, a message followed by the directory name, file name, and offending record number is typed. In each case the file directory entry is removed from the directory.

Error type 1: The beginning record address (BRA) of a file is 0.

Message: FILE OF NO RECORDS DELETED.

Special note: DSKRAT has a BRA of 100000 octal. The leftmost "1" is ignored by the JULYDOS system but is recognized by FIXRAT to indicate a special file with effective BRA of 0.

Error type 2: Zero or negative word count.

Message: ZERO OR NEGATIVE WORD COUNT.

Error type 3: The backward pointer of the successor to the current record does not point to the current record.

Message: POINTER MISMATCH

Error type 4: The backward pointer on the first record of a file is not zero.

Message: ERROR IN FIRST RECORD.

Error type 5: Two files point to the same record. That is, the current file being processed apparently uses a record which has already been used by a file FIXRAT has previously processed during the current run of FIXRAT.

Message: TWO FILES POINT TO SAME RECORD

Error type 6: File in illegal area. The record has correct format but is in a bad place on the disk. The restrictions are: records 0 to 117 octal are used for all UFD's, the MFD and the DSKRAT. Records 120 to 1177 are reserved for storing user memory during TSDOS operation. Records 1200 and up are available for regular user files.

Message: FILE IN ILLEGAL AREA.

FIXRAT recognizes UFD's from regular files by the beginning-record address of file directory entries. All files with a BRA less than 120 are considered UFD's, with the exception of DSKRAT. All files with BRA's greater than 120 octal are considered TSDOS commands. Therefore, if a new UFD is created, it must have a BRA less than 120 octal. UFD's are created by command CREATE.

Known bugs: The counter of records left is off slightly too low. When the disk is very full, this counter will register a very large octal number.

D. Operator Instructions for FIXRAT

To run FIXRAT, attach to DIAG and resume FIXRAT. The name of each directory is typed prior to processing of that

directory. FIXRAT will return to JULYDOS when done. If the message "100 DISK ERRORS" is typed, a particular record on the disk cannot be read. This is caused by either a bad or worn out disk pack or a misaligned disk drive. If any error messages appear, FIXRAT must be rerun until none appear.

XXI. QUICK REFERENCE GUIDE

A. Connect with System

TSDOS - go to one of the ASR-35 teletypes in the teletype room. Push the 'orig' button - you should get a dial tone. Dial 2866 or if busy 2870. When answer signal is heard, push "full duplex" button. You are ready to type commands.

DOS - if console teletype does not respond to commands, boot load the system. Load a ten foot paper tape labeled BOOTSTRAP in the paper tape reader. Turning to the console, check that all sense switches are in the reset position and start the computer with the program counter set to 1. The BOOTSTRAP should read into the machine and the response should be OK. You are ready to type commands.

B. Sign On

user: ATTACH ufd
response: OK

C. List Directory

user: LISTF
response: list of files is typed

D. EDIT File

user: ED file
response: GO

EDIT

(editing)

user: FILE file
response: OK

E. Compile and Check

user: FTN file
response: GO (errors if any typed along with line causing error)
CC, OK

F. Compile with Errors only in Listing File

```
user:       FTN file 1000 777
response:   GO
            CC, OK
            (listing file will be L←file)
```

G. Compile with Symbolic Assembly Listing

```
user:       FTN file 1000 40777
response:   GO
            CC, OK
            (listing file will be L←file)
```

H. Assemble

```
user:       INPUT file
response:   OK

user:       BINARY B←file
response:   OK

user:       DAP
response:   GO
            (errors typed here if any)
            no errors in above assembly
            AC, OK

user:       CLOSE ALL
response:   OK
```

I. Assemble with Listing

```
user:       INPUT file
response:   OK

user:       LISTING L←file
response:   OK

user:       BINARY B←file
response:   OK

user:       DAP 400 100555
response:   GO
            no errors in above assembly
            AC, OK
```

J. Load and Get Map

```
user:       CLRCOR
response:   GO
            OK
```

```
user:       LDR B←fill
response:   GO
            MR, OK

user:       START B←fil2
response:   GO
            MR, OK

user:       ATTACH U
response:   OK

user:       START FTNLIB
response:   LC, OK

user:       ATTACH ufd
response:   OK

user:       START   63002
response:   load map is typed
            OK
```

K. Save Core Image and Go

```
user:       SAVE *file 100 *pbrk 1000
response:   OK
            (note *pbrk is number following pbrk on load map)

user:       R *file
response:   GO
            program is running
```

L. Kill Runaway Program

under TSDOS

```
user:       (CONTROL X-OFF)
response:   QUIT

user:       PM
response:   program counter, A, B, index, and keys register is
            typed.  The purpose of this step is to find where
            your program went wild.
```

under DOS

Turn to control push button console. Put the MA-SS-RUN switch in SS position. You may now read from the lights where your program went wild. Restart DOS by setting the program counter to 70000 octal, putting the MA-SS-RUN switch in RUN position and pushing the start button. OK should be typed.

M. Sign Off

```
user:       CLOSE ALL
response:   OK

user:       ATTACH
response:   NOT FOUND
```

N. Special Characters

```
"     Erase one character
?     Kill line
(CONTROL X-OFF) Quit, back to TSDOS
      Tab
```

O. Summary of DOS Commands

The following is a list of all internal commands available to the user. The letters which compose the abbreviations for each command are underlined. A * indicates this is a TSDOS command only. Abbreviation may be used only under TSDOS with the exception of the RESUME command.

ASSIGN device [console]

ATTACH ufd password [disk]

BINARY name

CLOSE $\left\{\begin{matrix}\text{name}\\ \\ \text{unit}\end{matrix}\right\}$ $[\text{unit}_2]$ ... $[\text{unit}_9]$

COMINPUT $\left\{\begin{matrix}\text{name}\\ \text{TTY}\\ \text{CONTINUE}\end{matrix}\right\}$

DELETE name

INPUT name

LISTING name

LISTF

OPEN name unit status

PM

RESTOR name

RESUME name [pc] [a] [b] [x] [keys]

SAVE name sa ea [pc] [a] [b] [x] [keys]

START [pc] [a] [b] [x] [keys]

SWITCH device {ON / OFF}

STARTUP disk [disk] [disk] [disk]

The following is a list of all DOS external commands which are simply saved system programs that are loaded into the user's segment and executed. These commands can not be abbreviated. A % indicates this is a DOS command only.

% ALOOK name

BASIC

% BOOT

% CARDIN name

CLRCOR

COPY name1 name2

% CREATE ufd [disk]

CRIBBAGE

DAP [400] [param]

DEBUG

% DLISTF

% DUPE

ED [name]

EDB name1 [name2]

FILED [name]

FTN name [1000] [param1]

HAMBUG

% LPOUT name

LOOK name [csize]

% PALAP 24000 sa ea [pc] [ounit] [bootstrap]

% PASSWORD [name]

PAUSE

% RDIBM

% RDTAP

% RENAME name1 name2

% SKIP

% WEOF

% WRIBM name

% WRTAP name

% XREF

P. Summary of Editor Requests

BOTTOM

BRIEF

CHANGE␣% 'string1' % 'string2' %␣'n'␣G

DELETE 'n'

DELETE TO 'string'

ERASE 'character'

FILE ['filename']

FIND 'string'

INPUT

INPUT (PTR)

INSERT 'line'

KILL 'character'

```
LOAD 'filename'
LOCATE 'string'
MOVE 'buffer' 'buffer'
NEXT 'n'
PRINT 'n'
PTABSET 'n1' ['n2'] ... ['n8']
PUNCH 'n'
QUIT
TABSET 'n1' ['n2'] ... ['n8']
TOP
VERIFY [DISPLA ]
XEQ 'buffer'
*
```

XXII. ERROR MESSAGES

A. Disk Operating System

The following error messages indicate that either the DOS system has been partially overwritten by a user program or system files on the disk have been overwritten. The user should boot load the system and try again. If these error messages are still produced, he should immediately notify Jan Carlson or Michael Form of the DOS system failure.

BAD POINTER READING

BAD POINTER WRITING

BUFFER SPACE EXHAUSTED

CHKSUM ERROR IN ATTACH

CHKSUM ERROR IN MFD

FILE I/O DEVICE NOT ATTACHED

IMPROPER CALL TO RTNBUF

INITIAL BACK POINTER NOT ZERO

NO DEVICE TO UPDATE

RAT CHKSUM ERROR

SYSTEM UNIT ACTIVE

UNIT 8 ACTIVE

The following errors indicate user errors and are recoverable

AVAILABLE RECORDS EXHAUSTED

> the disk storage available to users has been used up. User must either delete some of his files or tell system people.

BAD CALL TO DSPRES

BAD CALL TO DSPSAV

BAD CALL TO SAVE

BAD CALL TO SEARCH

BAD CALL TO RESTOR

BAD command - necessary command argument missing

BAD PARAMETER

BAD PASSWORD

BAD POINTER DELETING - a broken file has been successfully deleted

command NOT FOUND

DELETE FILE IS OPEN

DISPLAY IN USE

DISPLAY NOT ATTACHED

EOF READING

ERROR RESTORING filename

FILE NOT OPEN FOR READING

FILE NOT OPEN FOR WRITING

filename ALREADY OPEN

filename NOT FOUND

ILLEGAL CALL TO READ

ILLEGAL CALL TO WRITE

ILLEGAL INSTRUCTION AT location

NO UFD ATTACHED

PASSWORD INCORRECT

PROGRAM HALT AT location

PUNCH NOT ATTACHED

READER NOT ATTACHED

REWIND FILE NOT OPEN

UFD OVERFLOW

UNIT IN USE

UNIT NUMBER OUT OF LIMITS - TSDOS message only

B. FORTRAN

AD ADDRESS OFSET TOO LARGE (-8192.LE.AF.LT.8192).

AE ARITHMETIC STATEMENT FUNCTION HAS OVER 10 ARGUMENTS.

AG SUBROUTINE OR ARRAY NAME NOT IN AN ARGUMENT

AR ITEM NOT AN ARRAY NAME

BD CODE GENERATED WITHIN A BLOCK DATA SUBPROGRAM

BL BLOCK DATA NOT FIRST STATEMENT

CE CONSTANT'S EXPONENT EXCEEDS 8 BITS (OVER 255)

CG COMPILER OR COMPUTER ERROR CAUSED A JUMP TO 00000

CH IMPROPER TERMINATING CHARACTER (PUNCTUATION)

CM COMMA OUTSIDE PARENTHESIS, NOT IN A DO STATEMENT

CN IMPROPER CONSTANT (DATA INITIALIZATION)

CR ILLEGAL COMMON REFERENCE

DA ILLEGAL USE OF A DUMMY ARGUMENT

DD DUMMY ITEM APPEARS IN AN EQUIVALENCE OR DATA LIST

DM DATA AND DATA NAME MODE DO NOT AGREE

DT IMPROPER DO TERMINATION

EC EQUIVALENCE GROUP NOT FOLLOWED BY COMMA OR CR (CARRIAGE RETURN)

EQ EXPRESSION TO LEFT OF EQUALS, OR MULTIPLE EQUALS

EX SPECIFICATION STATEMENT APPEARS AFTER CLEANUP

FA	FUNCTION HAS NO ARGUMENTS
FD	FUNCTION NAME NOT DEFINED BY AN ARITHMETIC STATEMENT
FR	FORMAT STATEMENT ERROR
FS	FUNCTION/SUBROUTINE NOT THE FIRST STATEMENT
HF	HOLLERITH CHARACTER COUNT EQUALS ZERO
HS	HOLLERITH DATA STRING EXTENDS PAST END OF STATEMENT
IC	IMPOSSIBLE COMMON EQUIVALENCING
ID	UNRECOGNIZABLE STATEMENT
IE	IMPOSSIBLE EQUIVALENCE GROUPING
IF	ILLEGAL IF STATEMENT TYPE
IN	INTEGER REQUIRED AT THIS POSITION
IT	ITEM NOT AN INTEGER
MM	MODE MIXING ERROR
MO	DATA POOL OVERFLOW
MS	MULTIPLY DEFINED STATEMENT NUMBER
NC	CONSTANT MUST BE PRESENT
ND	WRONG NUMBER OF DIMENSIONS
NE	NO END CARD PRIOR TO CONTROL CARD
NF	NO REFERENCE TO FORMAT STATEMENT
NI	INSERT STATEMENT WITHIN AN INSERT FILE
NR	ITEM NOT A RELATIVE VARIABLE
NS	SUBPROGRAM NAME NOT ALLOWED
NT	LOGICAL NOT, NOT AN UNARY OPERATOR
NU	NAME ALREADY BEING USED
NZ	NON-ZERO STRING TEST FAILED

Error Messages Generated by the Library Subroutines		Generating Subroutine
Error Message	Condition	
DA	Arithmetic Overflow	A$66/S$66 (Double Add/Sub)
DL	Negative or Zero Argument	DLOG/DLOG10 (Double Logarithm)
DM	Arithmetic Overflow or Zero Divisor	M$66/D$66 (Double Mult/Div)
EQ	Arithmetic Overflow	A$81 (Add Integer to Double Exponent)
EX	Exponent Overflow	EXP (Real Exponential)
FE	Format Error	F$IO (Format Scanner and Conversions)
II	Result Greater (2**15)-1	E$11 (Integer Raised to Integer)
IN	Input Error	F$IO (Format Scanner and Conversions)
RI	Exponent Greater than 15	C$21 (Convert Real to Integer)
SA	Arithmetic Overflow	A$22/S$22 (Real Add/Sub)
SD	Arithmetic Overflow or Zero Divisor	D$22 (Real Div)
SM	Arithmetic Overflow	M$22 (Real Multiply)
SQ	Negative Argument	SQRT (Real Square Root)

OP	MORE THAN ONE OPERATOR IN A ROW
PA	OPERATION MUST BE WITHIN PARENTHESIS
PH	NO PATH LEADING TO THIS STATEMENT
PR	PARENTHESIS MISSING IN A DO STATEMENT
PW	* PRECEDED BY ANOTHER OPERATOR OTHER THAN ANOTHER *
RL	MORE THAN 1 RELATIONAL OPERATOR IN A RELATIONAL EXAMPLE
RN	REFERENCE TO A SPECIFICATION STATEMENT'S NUMBER
RT	RETURN NOT ALLOWED IN MAIN PROGRAM
SC	STATEMENT NUMBER ON A CONTINUATION CARD
SP	STATEMENT NAME MISSPELLED
SR	BAD ARGUMENT TO INTRINSIC SUBROUTINE.
ST	ILLEGAL STATEMENT NUMBER FORMAT
SU	SUBSCRIPT INCREMENTER NOT A CONSTANT
TF	"TYPE" NOT FOLLOWED BY "FUNCTION" OR LIST
TO	ASSIGN STATEMENT HAS WORD TO MISSING
UO	MULTIPLE + OR-- SIGNS, NOT AS UNARY OPERATORS
US	UNDEFINED STATEMENT NUMBER
VD	SYMBOLIC SUBSCRIPT NOT DUMMY IN DUMMY ARRAY, OR SYMBOLIC SUBSCRIPT APPEARS ON A NON-DUMMY ARRAY
VN	VARIABLE NAME REQUIRED AT THIS POSITION

C. FORTRAN LIBRARY

LG	negative or zero arg for real
PE	parity error reading mag. tape
FE MS	mag. tape end of bit
D	DAP
A	Address Field missing where normally required; error in address format
C	Erroneous conversion of a constant; Address Field of data-defining pseudo-operation in improper format
E	Executable code generated before EXT pseudo-operation; external name modified by addition; external name used in Address Field of something other than a memory reference instruction*
F	Major formatting error
L	Label (Location Field) missing where normally required; error in label symbol*
M	Multiply defined symbol
O	Operation Field blank or not recognized; Operation Field not legal for object configuration
R	Relocation assignment error*
S	Address of Variable Field expression not in sector being processed or sector zero (applicable only in LOAD mode)
T	Improper use of Index Subfield; error in Index Subfield
U	Undefined symbol
V	Unclassified error in Address Field of multiple-subfield pseudo-operation
Z	Conditional assembly error; ELSE used outside of conditional assembly; END reached before all IFs matched by ENDCs*

---

*DAP-16 Mod 2 only.

E LOADER

Message	Meaning	Recovery Action
LC	Loading complete	none
MR	more subroutines required	continue loading
CK	Checksum error in last block read	restart loading
BL	block too large or improperly formatted	restart loading verify you are not attempting to load a source file
MO	memory overflow	restart loading

REFERENCES

1. A General Description of the Paging Modifications to be Incorporated into the DDP-516; KC-T-028; Dean Vanderbilt; July 14, 1967.

2. Advanced Remote Display Station (ARDS 100A) Reference Manual. Computer Displays Incorporated.

3. DDP-516 Assembler Manual; Doc. No. 70130072442A; Honeywell, August 1970.

4. DDP-516 Fortran IV Manual; Doc. No. 130071364A; Honeywell, April 1967.

5. DDP-516 Programmer Reference Manual; Doc. No. 70130072156E; Honeywell; November 1970.

6. DDP-516 Operator's Guide; Doc. No. 70130072165A; Honeywell; May 1969.

7. DDP-516 Voice I/O; Stephen R. Rotman, Service Technology Corporation job #D7031B; September 1969.

8. DOS, the DDP-516 Disk Operating System, J. Poduska, D. Udin, J. Carlson, J. Green; February 1969.

9. Graphics Reference Manual, D. Kipping, August 1969.

10. High Level Analog Input Subsystem Model 840 Special Option Instructions Manual; Doc. No. 7013007220B; Honeywell; August 1969.

11. H632 Programming System Manual; Doc. No. 130072009A; Honeywell; December 1968.

12. Low Capacity Multiline Controller Special Option Instruction Manual; Doc. No. 70130072154A; Honeywell; February 1969.

13. Models 316 and 516 BASIC Language, Honeywell; May, 1971.

14. System Specifications for DDP-516 - Univas 1004 Communications Link; D. Yourshaw; Service Technology Corporation. job #S0 430A; August 1969.

15. System 32/16 Coupler Option Manual; Doc. No. A253203; Honeywell; May 1969.

16. TSDOS Reference Manual; L. Liebson, Service Technology Corporation; November 1970.

REFERENCES

1. A General Description of the Paging Modifications to be Incorporated into the DDP-516; KC-T-028; Dean Vanderbilt; July 14, 1967.

2. Advanced Remote Display Station (ARDS 100A) Reference Manual. Computer Displays Incorporated.

3. DDP-516 Assembler Manual; Doc. No. 70130072442A; Honeywell, August 1970.

4. DDP-516 Fortran IV Manual; Doc. No. 130071364A; Honeywell, April 1967.

5. DDP-516 Programmer Reference Manual; Doc. No. 70130072156E; Honeywell; November 1970.

6. DDP-516 Operator's Guide; Doc. No. 70130072165A; Honeywell; May 1969.

7. DDP-516 Voice I/O; Stephen R. Rotman, Service Technology Corporation job #D7031B; September 1969.

8. DOS, the DDP-516 Disk Operating System, J. Poduska, D. Udin, J. Carlson, J. Green; February 1969.

9. Graphics Reference Manual, D. Kipping, August 1969.

10. High Level Analog Input Subsystem Model 840 Special Option Instructions Manual; Doc. No. 7013007220B; Honeywell; August 1969.

11. H632 Programming System Manual; Doc. No. 130072009A; Honeywell; December 1968.

12. Low Capacity Multiline Controller Special Option Instruction Manual; Doc. No. 70130072154A; Honeywell; February 1969.

13. Models 316 and 516 BASIC Language, Honeywell; May, 1971.

14. System Specifications for DDP-516 - Univas 1004 Communications Link; D. Yourshaw; Service Technology Corporation. job #S0 430A; August 1969.

15. System 32/16 Coupler Option Manual; Doc. No. A253203; Honeywell; May 1969.

16. TSDOS Reference Manual; L. Liebson, Service Technology Corporation; November 1970.