Next, you can specify the horizontal character spacing. Horizontal spacing is often expressed as a pitch of so many characters per inch, for example, Elite 12 pitch font. However, the number given to HTYPE must be the actual character spacing in 1/120 of an inch increments. Thus for a 12 pitch font, you specify 120/12 = 10 as follows:

Horizontal character spacing (1/120 of an inch)[12]: **10**

For a 10 pitch PICA font, use 120/10 = 12 (this is the default). Larger numbers mean greater spacing of characters. Other values may be used in order to squeeze or expand the text horizontally.

Next, you can specify the vertical spacing. As for horizontal spacing, the number specified is the actual space between lines. In this case, it is expressed in 1/48 of an inch increments. A 10 pitch font usually has 6 lines per inch for single spaced copy. The number you would specify to HTYPE is 48/6 = 8. 8 lines per inch is standard for a 12 pitch print wheel. For this, you would specify 48/8 = 6 as follows:

Vertical line spacing (1/48 of an inch)[8]: **6**

Other spacings might be used for special purposes.
Next, you may enable the multi-column feature. When this feature is enabled, lines of text may be caused to appear in multiple columns. Text to be printed in this fashion should be prepared with an initial full width section which might contain page headers etc. followed by a reduced width section which will be printed in multiple columns followed by a final full width section. There are four parameters which control the multi-column feature. First is the column spacing which determines the distance (in characters) from the left edge of one column to the left edge of the next. The default value for this is 33 characters. Next is the first line on which multiple columns should begin. This is one greater than the number of lines which are printed full width. Next is the number of lines per column. If two columns are to be printed, then the text being printed should contain twice this number of narrow lines. Next is specified the number of columns. The default for this is two. Finally, the rejustification margin is specified for the multiple column region. This number serves the same purpose as the rejustification margin above except that it applies to the multiple column region of the file. After the final column is

finished, the left margin reverts to the original left side of the paper.
The remainder of HTYPE's operation is similar to that of ZTYPE. Note that the special graphics control characters are different for the 1700 than for the Bedford terminal. The appropriate version of the "SUBSUPER" file should be used.

The commands which may be typed at page pauses are summarized below. Note that echoes are turned off as these commands are typed to avoid the necessity of inserting a piece of scrap paper.

Pnnn<sp> Print page nnn next. The set of pages remaining to be printed is not altered and the page which would have been printed next will be printed after the page specified by the command.

X Exit to the exec.

B Jumps back to the beginning of the parameter setting dialog. Thus certain parameters may be altered without re-specifying the rest. The parameter setting dialog is not conducted invisibly.

Any other character indicates that the paper is ready and printing begins.

Two special characters are recognized while HTYPE is printing. Control-X aborts printing the current page and enters a page pause. Use this character if the page has been garbled somehow and you wish to start again. Note, that the next page printed will normally be the next one specified at the initial dialog. If the same page is to be printed, use the **Pnnn<sp>** command. Control-P may be typed to force a page pause at the end of the current page. This is useful when operating in no pause mode and you want to re-adjust the paper or reprint a garbled page.

GRIPE
#
GRIPE allows you to send complaints or suggestions about subsystems to system personnel.
#
When it asks you "Griping on subject of ", type the name of the subsystem. If it doesn't accept that name, try "general" instead of the subsystem name.
#
When it asks you for "Message", type your message terminated by control-Z. For details on entering the message, see "Message" under SNDMSG in this manual.

IDDT

1. Introduction

IDDT is a debugger for TENEX programs. It has many of the same commands as the standard DDT10X (SDDT and UDDT) and ordinarily may be used without regard to the fact that it is a different debugger. The user is directed to the DDT section of the DECsystem-10 ASSEMBLY LANGUAGE HANDBOOK for information regarding the basic features and use of DDT.

The primary feature of IDDT is that it operates on user programs which run in an inferior fork under IDDT. Thus, an errant user program cannot destroy the debugger or its symbol table because the debugger is in a totally different address space. This relation between the program being debugged and IDDT is much the same as the relation between current user programs (including IDDT) and the EXEC. Because of this, IDDT must simulate many of the services ordinarily provided by the EXEC, such as @GET, @LOADER, @RUN, etc.

The following describes the new features in IDDT and how they may be used for debugging. Some of the features are bound to change, and others will be added.

2. Using IDDT

IDDT may be called into service either before or after programs have been loaded into memory. This is done by telling the EXEC

@IDDT

This command causes the EXEC to splice a fork containing IDDT in between itself and the program to be debugged. This operation is done in a way that preserves the state of the user's program including its fork structure. It is possible to ^C out of a running program and get IDDT. If this is done, a $P (Proceed) command will resume running the user program.

The EXEC command "NO IDDT" will unsplice the fork containing IDDT in the event the user wishes to continue his program without having an IDDT above it.

A fairly common practice is to get IDDT first and use it to load the program to be debugged. One of three IDDT commands may be used to load the object program: $L (run the LOADER in the user fork), ;L (Loadgo of named file), or ;Y Yank the named file). The first of these is essentially the same as the EXEC command, @LOADER. The second is comparable to @RUN, while the last is similar to @GET.

3. Symbol Table Considerations

When initially started, and after successful execution of a ;L or ;Y command, IDDT will obtain a new symbol table if it exists. It does this by copying (and sometimes sharing) pages of the user fork. Thus, those user programs which need access to their own symbols will behave the same, and IDDT will have its own copy of the symbol table which is protected from the user.

The $L command causes IDDT to run the LOADER in the user's address space. Upon completion, the LOADER returns control to IDDT. At this point IDDT will have the LOADER's symbol table. In order to switch to the symbols of the program which was loaded, the ;S command should be typed. ;S tells IDDT to look for a standard symbol table pointer in location 116 (.JBSYM). If the user has merged in a file which contains its own symbols, he may switch to that table by typing a;S where "a" is the address of the location containing a pointer to the new table.

;? with no argument causes IDDT to type the error string
associated with the most recent error in the user program fork.
With an argument (600121;? or IOX5;?), the corresponding error
string will be typed.

;O Obtains a symbol file directly into IDDT without modifying the user's memory. The old symbol table is replaced, and a new entry vector is taken only if there was no old one. This makes it possible to debug one file with symbols obtained from a different file.

Symbols may be written out on a specified file by using the ;W command. This saves the symbols in a way that they may be obtained later with the ;O command. Along with the main symbol table, the undefined symbol table is saved in symbol files. One should be careful that symbol files and core image (;U) files are kept paired if any undefined symbols exist. Executing a GET JSYS on a symbol file will get both tables. The default file extension for symbol files written by IDDT is .SYMBOLS.

Note: Symbols added to or deleted from IDDT's symbol table by commands to IDDT will not be seen by the user program.

4. EXEC-like Features

For convenience, the EXEC has several commands which provide the same services as some EXEC commands. These are:

;A	TYPES THE USER'S ADDRESS SPACE. (MEMSTAT)
;F	DOES A FORKSTAT ON THE USER FORK. (THIS FEATURE WILL BE OPERATIONAL AS SOON AS JSYS 166 IS IMPLEMENTED.)
;Y	@GET
;M	@MERGE
;L	@RUN
$$G	@START
$$1G	@REENTER
$$nG	@START AT n-TH ENTRY VECTOR LOCATION
$P	@CONTINUE
# $L	@LOADER
;U	@SSAV Ø 777 (I.E. "UNGET")
;O	OBTAIN SYMBOL FILE -- NO EXEC EQUIVALENT
;W	WRITE SYMBOL FILE -- NO EXEC EQUIVALENT
;H	@QUIT (HALT, RETURN TO EXEC)

;W, ;M, ;Y, ;O, ;L, and ;U ask for a file name from the user. The default extension will be .SAV or .SYMBOLS as required.

5. Access Control

An EXEC-like feature has been included in IDDT which has no analogy in the current EXEC. This is the $U command (UNPROTECT), which allows the user to manually change the protection on various pages of his fork. This command has several forms:

a<b$nU	CHANGE PROTECTION ON PAGES a THROUGH b INCLUSIVE
a$nU	CHANGE protection of page a
$nU	CHANGE PROTECTION OF THE CURRENT PAGE

(WHERE POINT "." IS)

N is always a three-bit number (Ø-7). The 4-bit means allow read access, the 2-bit should be on to allow write access, and the 1-bit for execute access. If N is not specified at all, it will be taken as 7. Thus, the command $U frees up the current page, giving it read, write and execute access.

$U changes the protection on pages of the user's fork. It does not affect the protection of a file page which might be mapped into that fork. Because it is sometimes convenient to change the contents of fork pages which have write-protected files mapped into them, $U commands which ask for write access will either get it, or will get write-copy access.

While IDDT is running, it temporarily changes the access of each page that it maps to have read, write, and execute access. The user's access is reset when the page is mapped out. This allows IDDT to insert breakpoints, retrieve trapping instructions, etc. The $U command allows the user to protect pages from his own program by effecting a permanent change in the page access.

6. Rubout

IDDT arms the RUBOUT button as an interrupt character. If a user program has been started under IDDT, pressing RUBOUT will gracefully suspend that process and give control to IDDT which then types a message of the form

```
XXX:FOO+5/   MOVE A,DAT+21
```

The interrupt is understood to have occurred immediately before this instruction, and that if a $P (proceed) command is typed, this instruction will be the next one executed by the user.

RUBOUT's typed while in IDDT behave much the same as they do in normal DDT's. That is, the current command is aborted. This is particularly convenient for stopping long searches ($W, $N and $E COMMANDS), with IDDT because it is an interrupt and does not have to be read by a PBIN to initiate action as it does with old-style DDT's. RUBOUT's typed while IDDT is in control cause the terminal output buffer to be cleared.

Since some programs such as TECO use RUBOUT as a command,
the IDDT ;E command can be used to change the IDDT interrupt
character from RUBOUT to any other control character.

7. "GO" Commands

IDDT has several variations of the standard $G command available. GO commands with two ALTMODES, such as $$G, FOO$$G, and $$2G, cause the user's pseudo interrupt system to be cleared before they take effect. If there is a number between the ALTMODE(s) and the G, this number is taken as an index into the entry vector of the user's fork, and the program is restarted as indicated by the corresponding entry vector element. Thus, $$ØG is the same as the EXEC command "START", while $$1G is equivalent to "REENTER". The command $$G is an abbreviation for $$ØG.

Ordinary GO commands still exist. They look like FOO$G and BEGIN$$G. The user's program counter is stored in the "GO" register, which is named $G. This can be examined by commands such as $G/ .

8. Control-T

Frequently the user would like to know whether his program
is making progress. To facilitate this, the EXEC arms T
(control-T) as an interrupt character, which types the program
state, the load average, amount of CPU and CONSOLE time used (in
seconds), and the ratio of console to CPU time (the "activity
ratio"). Note that the user's program is not stopped while his
information is being typed and if his program is typing out, ^T
will result in a garbled typescript.

9. Interface with the EXEC

The EXEC command "FORK n" may be used to switch the EXEC's attention between the fork containing IDDT and the one containing the user's program. This may be done for the purpose of doing a "MEMSTAT" or ^T. The EXEC examine and deposit commands (/ and \) also pertain to the currently selected fork.

Regardless of which fork has been selected, a "CONTINUE" will always resume a ^C . If the user has returned to the EXEC by typing ;H to IDDT, IDDT may be resumed by a "CONTINUE". A HALTF in the user's program will return to IDDT. It may be continued by a $P to IDDT.

10. Zero-ing core

THE $$Z COMMAND behaves the same as it does with old DDT except that if it is used to zero whole pages, they are PMAP-ed out of existence, rather than being actually cleared. If such a page is brought into existence again by a reference, it will be cleared by TENEX when created.

If a $$Z command is used to clear any word(s) between 700000 and 712777, compatibility code for the user is dismissed. Ordinary register operations like slash can be used to examine or modify the compatibility code (PA1050) as usual.

The Zero command has been generalized so that it can fill
core with a specific decimal number. To fill locations 100
through 177 with the number 12 (decimal) the user would type
100<177$$12Z.

11. Internal Registers

IDDT maintains several "internal registers" which may be manipulated as if they were in the user's address space. These are listed below, and will be described in detail in subsequent sections.

$G	CONTAINS FLAGS,,PC FOR THE USER PROGRAM
$M	MASK FOR SEARCHES
$X	LOCATION FOR SPECIAL EXECUTE
$W	PAGER TRAP STATUS WORD AT MEMORY VIOLATION
$W+1	PAGER WRITE DATA AT MEMORY VIOLATION
$I	INTERRUPT CHANNELS WITH BREAKS WAITING
$I+1	INTERRUPT CHANNELS ASSIGNED FOR USER
$I+2	BREAKS IN PROGRESS WORD
$I+3	0 IF USER'S INTERRUPT SYSTEM IS OFF. NON-0 OTHERWISE.
$I+4	IDDT'S FORK HANDLE ON USER
$I+5	SIXBIT OF SAVED USER SUBSYSTEM NAME (This may be made inaccessible in the future!)
$nB+k	BREAKPOINT REGISTERS. n is between 1 and 8 inclusive (i.e., IDDT has eight breakpoints), k is between 0 and 6. Thus there are seven registers of information associated with each of the breakpoints.

As an example of an internal register reference, consider looking at the proceed count of breakpoint 3:

```
$3B+2/ 105 3
```

The user changed the proceed count from 105 to 3.

IDDT's current location may be internal to IDDT. This allows the user to use linefeed and up-arrow to look at internal registers. IDDT has special address printing routines that print things like $I+3 instead of this address in terms of user defined symbols.

Attempts to define address tags when "point" is at an IDDT internal register will be given IDDT's ubiquitous "?" error. Also, IDDT will not allow expressions with more than one mention of an internal symbol name. Thus, $M+3 is allowed, but $I+$M is not.

12. The User Program PC

The internal register $G contains the user's PC and FLAGS. This is defined to always point at the next executable instruction. The proceed command ($P) simply starts the user at the address in $G. Illegal instruction traps back up the user's PC so that it points at the offending instruction, in hopes that he will repair it and proceed. In such a case, the repaired instruction will be executed first.

$G is setup from the entry vector after every ;Y, ;L, and ;S command. Thus, the user can ;Y (yank) a file and immediately start it with a $P.

Bit 5 of the "GO" word $G is the user-mode bit which will normally be on if $G is examined. It may be off due to an interrupt out of a JSYS or after an illegal instruction. Because this bit is essential to the restarting of the user's fork, it is not left entirely under his control. In particular, the user-mode flag may be turned on by changing the contents of $G, but it may not be turned off. This means that if a JSYS (such as GTJFN) has been interrupted, the usermode flag turned on, and $P typed, that the interrupted JSYS will be re-executed, rather than resumed.

13. Saving a Core Image

The ;U command asks for a file name and then does an SSAVE from page 0 through page 777 on this file. The entry vector will be copied if it exists. If no entry vector has been declared for the fork, IDDT will set a length one entry vector at "." . A message is typed to this effect.

14. Single Instruction Executes

When the user types an instruction followed by $X, IDDT pushes down several words of state information, plants the instruction in the user's address space followed by three breakpoints, and restarts the user at this special location. When the instruction completes, IDDT types the proper number of $-signs to indicate how many times the instruction skipped, and pops back the saved state information. The state information currently includes the program counter ($G), and which breakpoint (if any) the user was stopped at. This makes it possible to hit a breakpoint, execute an instruction (which might be a PUSHJ to a subroutine), and then, upon completion of the $X, do a $P to proceed the breakpoint.

IDDT's $X/ register points in the user's address space to the four words which will be used for $X commands. $X initially contains 777774 so that the top four words are used. The user is free to change the contents of $X.

If a RUBOUT has interrupted the program being debugged while it was in the middle of a JSYS -- usually a "long" JSYS like SOUT or PBIN -- and then an instruction executed with the $X command, a $P will not resume the original sequence back in the middle of the interrupted JSYS. Flag bit 5 will be off if the interrupt came out of a JSYS. A proceed ($P) immediately after a RUBOUT, with no intermediate $X will always resume exactly at the interrupt point however.

15. Breakpoints

Associated with each breakpoint are seven internal registers. The first four of these are the same as those in older DDT's, while the last three have been added.

```
#       $nB/    TRACEVALUE,,LOCATION
#
#       $nB+1/  Ø  OR CONDITIONAL BREAK SKIP
#
#       $nB+2/  PROCEED COUNT (>Ø for normal, <Ø for auto, Ø for none)
#
#       $nB+3/  Ø OR STRING POINTER (fed to IDDT when this BPT breaks)
#               ***Not implemented yet***
#
#       $nB+4/  SAVED INSTRUCTION WHILE USER IS RUNNING
#
#       $nB+5/  Ø OR ELSE -1 FOR AUTOPROCEED MODE
#
#       $nB+6/  ASCII NAME OF THIS BREAKPOINT, USUALLY "$nB"
```

Usually these values are changed only by setting and clearing breakpoints with the $B command. if he wishes, the user may change these quantities. For instance, if he wants hits on breakpoint three to print as

```
HELP>> FOO+23
```

he would type the following:

```
$3B+6! "/HELP/
```

This stores the ASCII string for the new name in the print name cell of breakpoint three.

Proceeding after a breakpoint hit happens much in the same way as a single instruction execute command ($X). Again four words of memory are written into. However, in this case the four words are the instruction at the break location and three JRST's to the three locations following the break location. The JRST's account for possible skips by the break instruction.

If a breakpoint is hit, and the user changes the contents of $G, and then proceeds (with $P), the break instruction is not executed. Control simply resumes at the new location given by $G. Old DDT's execute the instruction under the breakpoint, and then transfer control to the new place.

16. JSYS Typeout Format

When IDDT attempts to print an opcode 104 instruction symbolically, it first looks for an exact match in the user's symbols. If one is found, the corresponding user-supplied name is printed. Otherwise, IDDT checks its own internal JSYS symbol table (hopefully, the same as JSYS's defined in <SYSTEM>STENEX.MAC) for an exact match. If none is found in either place, the instruction will print as JSYS 501, i.e., "JSYS" and address.

17. Other Features

The search commands ($W, $N, and $N) have been generalized to take an argument which specifies the maximum number of "finds" that shall occur before the search will terminate. An example is:

```
FOO<BAR>QQZZ$5E
```

This command will stop after typing five instructions lying

between locations "FOO" and "BAR" which have an effective address of "QQZZ".

Internal register $I+4 contains the fork handle that IDDT uses to reference the user. This register is writeable.

$Q has the value of the last quantity typed, as always. $$Q has this value with halves swapped. Thus, ($$Q)= will type the same value as $Q= will.

$V is the value of the left half of the last quantity typed. $$V is the same with the sign extended. Thus, assuming the last value typed to be -3,,FOO , $V= would yield Ø,,-3 whereas, $$V= would type -3.

;? prints the most recent error encountered in the program.
#
;<space> prints the contents of $Q in the current typeout
mode.

IMGPTP

Copies image-mode binary files (one byte per word), such as those written by PALX, to the paper tape punch.

IMGPTP is designed to operate in quasi-command style (TENEX Executive Manual, p. 13). To use the program in this manner:

```
@IMGPTP (INPUT FILE) filename.ext [CONFIRM]%
@
```

Filename recognition operates for entry of the input file name, with the extension .BIN supplied automatically (if it exists). If the filename typed is in error, a ? is typed and the name entry must be begun again. If the PTP: is unavailable the subsystem so informs the user and terminates.

If there are no errors, IMGPTP first punches some blank leader at the PTP:, then punches a visible (human-readable) identifying leader, then copies the contents of the input file to the punch, and finally punches blank trailer before returning control to the EXEC. The identifying leader consists of the full name and date-last-written of the input file, e.g.,

```
<DIREC>NAME.BIN;2  10-JUN-73.
```

LBLOCK

LBLOCK is a small subsystem to perform "lineblocking" of text files. Its function is to read a text file, and write another text file, after inserting NULL characters where necessary to satisfy the condition that no line of text may be split across a record boundary (as defined in the DEC terminology).

The input file may be any TENEX text file, with the usual name recognition and defaults. The output file is specified in the shorter DEC form of "device:name.ext" where DSK is assumed for device and "name" may be up to 6 characters long, and "ext" may be up to 3 characters long.

Sample dialogue:

```
@LBLOCK
INPUT FROM FILE UNBLOCKED.DATA;3
OUTPUT TO FILE BLOKED.DAI
DONE.
@
```

LINK1Ø

```
# LINK-1Ø, the DECsystem-1Ø Linking Loader is a utility program which
# can merge independently-translated modules of a person's program into
# a single module.  It prepares and links this input with other
# modules required by the user into a form that can be executed by
# the operating system.
#
# The general command format is
# */Switches File spec /Switches , /Switches File spec /Switches -
# , /Switches File spec /Switches /GO
#
# Lines may be continued by use of hyphen (minus sign) at end of line.
#
# A File spec is some of DEVICE:FILE.EXT[directory]
# Output file specs may be separated from input ones by = ,
# in which case they must be first on the line.
#
# Switches take optional arguments, these are :-
# Keyword       a symbolic keyword
# Symbol        a sixbit symbol names (in ascii)
# Value         in decimal (octal if preceded by # )
# Value         in octal
# Core size     in K (2ØØØ words) or P (1ØØØ words)
#               as in nK or 1K+hK
# Version       standard version number
#
# Switches and keywords preceded by * are unique to  one character.
# Switches enclosed in parentheses are switches known to SCAN but
# not used by LINK-1Ø.
#
# Switches enclosed in angle brackets are only available with
# non-standard assembly options.
#
# Switches are :-
# BACKSPACE     decimal value
# (BEFORE)
# COMMON         symbol:decimal value
# CORE          core size
# CONTENTS      keyword
#               Default, All, None, Global, Noglobal, Local, Nolocal,
#               Entry, Noentry, Relocatable, Norelocatable, Absolute,
#               Noabsolute, Common, Nocommon, Zero, Nozero
# COUNTER
# DATA
# *DEBUG         keyword
#               Macro, Ddt, Fortran, Mantis, *Cobol, Cobddt
# DEFAULT        keyword
#               Input, Output
# DEFINE         symbol:decimal value
# (DENSITY)
```

```
# ENTRY
# (ERNONE)
# (ERPROTECTION)
# ERRORLEVEL    decimal value (0-30)
# ESTIMATE      decimal value
# EXCLUDE
# *EXECUTE
# FOROTS
# FORSE
# FRECOR         decimal value
# *GO
# HASHSIZE      decimal value
# *HELP
# INCLUDE        symbol
# *LOCALS
# <KLUDGE>
# LOG
# LOGLEVEL      decimal value (0-30)
# *MAP          keyword
#               End, Now, Error
# MAXCOR         core size
# MPSORT         keyword
#               Alphabetical, Numerical
# MTAPE          keyword
#               Mtwat, Mtrew, Mteof, wmtskr, wMskt, Mtbsr, Mteot
#               Mtunl, Mtblk, Mtskf, Mtbsf, Mtdec, Mtind
# NEWPAGE        keyword
#               Low, High
# NOENTRY        symbol
# NOINITIAL
# <NOKLUDGE>
# *NOLOCAL
# (NOOPTION)
# (NOPHYSICAL)
# NOREQUEST     symbol
# NOSEARCH
# NOSTART
# (NOSTRS)
# NOSYMBOL
# NOSYSLIB      keyword
#               Default,F40,Cobol,Algol,Neliac,Fortran
# (OKNONE)
# (OKPROTECTION)
# (OPTION)
# OTSEGMENT     keyword
#               Default, Low, High
# (PARITY)
# PATCHSIZE     decimal value
# (PHYSICAL)
# PROTECTION    octal value
# REQUEST
```

```
# REQUIRE       symbol
# REWIND
# RUN           file spec
# RUNAME        symbol
# RUNCOR        core size
# (RUNOFFSET)
# SAVE          core size
# *SEARCH
# SEGMENT       keyword
#               Default, Low, High
# SEVERITY      decimal value (0-31)
# SET           symbol:symbol or  octal value
# (SINCE)
# SKIP          decimal value
# SSAVE         core size
# START         symbol or  octal value
# (STRS)
# SYMBOL        keyword
#               radix50, triplet
# SYMSEG        keyword
#               Default, Low, High
# SYSLIB        keyword
#               Default,F40,Cobol,Algol,Neliac,Fortran
# SYSORT        keyword
#               Alphabetical, Numerical
# TEST          keyword
#               Macro, Ddt, Fortran, Mantis, *Cobol, Cobddt
# *UNDEFINED
# UNLOAD
# VERBOSITY     keyword
#               Short, Medium, Long
# VALUE         symbol
# VERSION       version
# XPN
# ZERO
```

LOADER

The LOADER is essentially the most recent PDP-10 LOADER release with a few modifications for use on TENEX. It is capable of loading REL files produced by all language processors including FAIL, SAIL and ALGOL.

1) The /Y switch causes the high segment load point to be moved to the next 1000 word page boundary. /Y does the same for the low segment.

2) /nH where n is a big number, behaves for the high segment the way /nO does for the low segment.

3) Multiple listings of undefined and multiply defined globals is suppressed.

4) A special block type is implemented to handle the ASSIGN pseudo-op. See MACRO writeup for more details.

5) /S and /B switches are defaulted to be "ON".

6) Just before the LOADER finishes, it sets the current subsystem name to "(PRIV)". This name is saved and restored by the EXEC and IDDT when the loaded program is started.

For further information refer to DECsystem10 Assembly Language handbook.

LOGOS (The LOGO Language)

LOGO is an interpreted, procedure-based language with strings as its fundamental data type. It was designed at BBN to be used by students across a wide range of educational situations. LOGO has been used by students at all levels from the second grade through college to write programs ranging from a couple of lines to complex procedure structures embodying several hundred lines in all. Such use is enhanced by LOGO's very clean syntax and good error evaluation.

LOGO permits recursion in user-defined procedures, in fact, this is the basic form for repetition. Recursion, together with a primitive which evaluates its input as a LOGO command string (an EVAL) and with primitives which access procedure lines make possible fairly complex program structures.

A complete reference manual describing LOGO has been written. The introduction is appended here to give a "sense" of the language.

LOGO REFERENCE MANUAL

1. A Look at LOGO

We introduce LOGO by writing several small procedures. The following examples serve to show what LOGO "looks like". Several features are used without definition or even explanation, where we think their meanings are clear from context. All of LOGO is comprehensively described in later sections.

LOGO, as an interpretive language, can execute single commands directly. Thus,

PRINT SUM OF 2 AND 2 (The user's typing is underlined) 4

But, the most important feature of LOGO is that such commands can be incorporated in user-written procedures. The definition of any procedure results in an object which is treated just like any primitive. Thus, in a very real sense, as the user writes his own procedures, he is gradually extending the basic language to more exactly fill his needs.

A very simple (although by no means simplest) procedure, for example, prints the double of its input.

```
TO DOUBLE :N:
10 PRINT SUM OF :N: and :N:
END
```

```
# This procedure, DOUBLE, is now "part" of LOGO.
#
#  DOUBLE 123
# 246
#  DOUBLE WORD OF 1 AND 1
# 22
#
# If the concatenation of DOUBLE with other procedures is desired,
# DOUBLE should OUTPUT rather than PRINT its results; OUTPUT
# meaning that the result is given to the calling procedure. The
# modified procedure is:
#
# TO DOUBLE :N:
# 10 OUTPUT SUM OF :N: and :N:
# END
#
# This new version of DOUBLE can be used in direct commands,
#
#  PRINT DOUBLE DOUBLE DOUBLE 3
# 24
#
# or can be used as the basis for other procedures,
#
# TO QUADRUPLE :N:
# 10 OUTPUT DOUBLE OF DOUBLE OF :N:
# END
#
# and so on. This very natural use of functions in LOGO is
# particularly valuable, since to program a problem a user can keep
# on breaking it up until he sees subproblems which he feels will
# be easy to program. This heuristic is used by sophisticated
# problem-solvers generally, whether or not computer programming is
# involved.
#
# An extension of this LOGO facility for using procedures in
# defining other procedures is its ability to handle recursive
# procedure definitions. The recursive can be a linear one, which
# is equivalent to iteration, as in the following procedure which
# calculates the factorial function:
#
#                n! = n . (n-1) ... 2.1.
#
# TO FACTORIAL :NUMBER:
# 10 TEST IS :NUMBER: 1
# 20 IF TRUE OUTPUT 1                    (1! = 1)
# 30 OUTPUT PRODUCT OF :NUMBER: AND
#       (FACTORIAL OF DIFFERENCE OF  (n! = n . (n-1)!)
#       :NUMBER: AND 1)
# END
```

```
#  PRINT FACTORIAL OF 100
# 93326215443944152681699238856266700490715968264381621468592963895 21
# 75999932299156089414639761565182862536979208272237582511852109168 64
# 00000000000000000000000
#
# LOGO makes few distinctions between numbers and more general types  of
# string.   Thus,  a  procedure to reverse a string looks very much like
# FACTORIAL.
#
# TO REVERSE :STRING:
# 10 TEST IS :STRING: :EMPTY:          (The reverse of the empty string
# 20 IF TRUE OUTPUT :EMPTY:            is the empty string.)
# 30 OUTPUT WORD OF LAST OF :STRING:   (The reverse of the nonempty
#       AND (REVERSE OF BUTLAST OF     string is the string formed by
#       :STRING:)                      following the last character by
# END                                  the reverse of the rest of the
#                                      string.)
#
# ("AND", "OF", and parentheses  are  optional  "noise  words"  for
# convenience in writing expressions.)
#
#  PRINT REVERSE OF "JABBERWOCKY"
# YKCOWREBBAJ
#
```

LPTPLOT

Permits plotting graphical data on the line printer.

```
@RUN LPTPLOT.SAVE;1

WOULD YOU LIKE INSTRUCTIONS? ANSWER Y OR N:Y

THIS PROGRAM PERMITS A FAIRLY COARSE X-Y PLOT OF DATA
CONTAINED IN ANY ASCII DISK FILE TO BE PREPARED FOR LISTING ON
THE LINE PRINTER.  THE PLOTTING GRID CONTAINS 61 POINTS
(6 INCHES) HORIZONTALLY AND 49 POINTS (8 INCHES) VERTICALLY.
THE PROGRAM REQUESTS XMIN AND YMIN COORDINATES (WHICH WILL
FALL IN THE LOWER LEFT CORNER OF THE PLOT) AND SCALE FACTORS
XINT AND YINT FOR EACH ONE-INCH INTERVAL HORIZONTALLY AND
VERTICALLY.  A ONE-LINE TITLE MAY BE ENTERED.  THEN YOU CAN
CYCLE THROUGH THE PROGRAM AN UNLIMITED NUMBER OF TIMES,
DRAWING LINES (BY SPECIFYING THE COORDINATES OF THE END
POINTS), DRAWING CIRCLES (BY SPECIFYING THE ORIGIN
COORDINATES AND THE RADIUS), OR PLOTTING X-Y DATA FROM AN
ASCII DISK FILE.  AT EACH STEP, ARBITRARY PLOTTING SYMBOLS
(*,@,X,ETC) CAN BE SPECIFIED, AND THE PLOTTED DATA WILL
OVERLAY PREVIOUSLY PLOTTED DATA.  DISK FILE NAMES ARE LIMITED
TO 5 CHARACTERS PLUS A 3 CHARACTER EXTENSION.  AN ARBITRARY
NUMBER OF FILE LINES CAN BE SKIPPED TO AVOID TITLE
INFORMATION, AND AN ARBITRARY FORTRAN FORMAT (UP TO 25
CHARACTERS) CAN BE SPECIFIED FOR READING DATA.  AN EXAMPLE:
(45X,2F7.Ø).  THE PARENTHESES MUST BE PRESENT.  NORMALLY, THE
FIRST NUMBER READ WILL BE CONSIDERED THE X-VARIABLE AND
THE SECOND NUMBER THE Y-VARIABLE.  BUT IF A "Y" ANSWER IS
GIVEN TO "REVERSE X AND Y?", THE SECOND NUMBER WILL BE
CONSIDERED THE X-VARIABLE.  WHEN ALL PLOTTING IS DONE, ANSWER
"N" TO "MORE?".  THE PROGRAM WILL THEN PRODUCE A DISK FILE
NAMED PLOT.DAT, WHICH CAN BE LISTED.

XMIN AND INTERVAL (THERE ARE 6 INTERVALS):-3. 1.

YMIN AND INTERVAL (THERE ARE 8 INTERVALS):-4. 1.

TITLE:
THIS IS A SAMPLE PLOT OF A CIRCLE, A SINE WAVE, AND A LINE.

DRAW LINE?N

DRAW CIRCLE?Y

XØ,YØ=Ø. Ø.
```

```
RADIUS=3.

PLOTTING SYMBOL=@

READ INPUT DATA?Y

FILE NAME(5 CHARS)=SINE

FILE EXT (3 CHARS)=TEL

SKIP FIRST N LINES. N=Ø

INPUT FORMAT FOR X,Y=(2F6.Ø)

REVERSE X AND Y?N

PLOTTING SYMBOL=*

MORE?Y

DRAW LINE?Y

X1,Y1=-3. 4.

X2,Y2=3. -4.

PLOTTING SYMBOL=Ø

DRAW CIRCLE?N

READ INPUT DATA?N

MORE?N

CPU TIME: 7.47 ELAPSED TIME: 5:20.ØØ
NO EXECUTION ERRORS DETECTED

EXIT.
```

MACRO

The TENEX MACRO is a slightly modified version of the most recent DEC MACRO-10.

The modifications are:

1) The /D switch causes tabs to be included in MACRO arguments. This is the DEC standard and may be necessary for assembling certain programs such as SNOBOL.

2) ASSIGN SYM1, SYM2, N pseudo-op causes the global SYM1 to have the same value as SYM2 and then increments the value of SYM2 by N. If N is not specified, it is assumed to be one. Note: a special REL file block type and appropriate LOADER modification were created to complement ASSIGN.

3) JSYS and UMOVEx are included in the OP-code table.

4) ORG ADDR pseudo-op behaves like either a LOC or RELOC depending on the mode of ADDR. ADDR may be an expression.

5) The version of MACRO on the system has all of the STENEX (JSYS and error mnemonics) definitions already included by use of the UNIVERSAL pseudo-op. This means that any program previously assembled with SYS:STENEX can now be assembled by including the statement SEARCH STENEX source file at the beginning. That is, if you add the search statement to the beginning of the program (immediately after the TITLE is a good place), you don't have to include SYS:STENEX in the command input. This results in a considerable saving of time, particularly significant in the case of small programs.

The only difference in this procedure is that MACRO will include in the output symbol table (for DDT) only those JSYS and error mnemonic definitions actually used in the program. This means that DDT* will not know about the other JSYS definitions which you may have occasion to use while debugging. If you want DDT to know all JSYS and error definitions, you can include SYS:STENEX.REL with the loading of your program. This has all of the JSYS and error mnemonic definitions defined as global symbols.

---

*IDDT does have the JSYS definitions built in, but it does not have the error mnemonics. For further information refer to DECsystem10 Assembly Language handbook.

MAILER

```
# MAILER is the subsystem that delivers queued mail (i.e. mail
# which has been queued by SNDMSG). It runs in the background
# automatically, so it is ordinarily unnecessary for a user to run
# it. The following description is in two parts: the first gives
# general information on the handling of mail and is relevant to
# all users; the second is only relevant to a person who runs
# MAILER.
#
# 1. Handling of mail
#
# Mail that has been queued (by SNDMSG) is placed in the file
# [--UNSENT-MAIL--].address in the directory of the user who sent
# it. (If a single message was sent to more than one address,
# there is a separate queued file for each.) MAILER, running
# automatically at frequent intervals (several times an hour)
# attempts to deliver each such file to the specified address.
#
# If it succeeds, it deletes the file.
#
# If it fails for some transient reason (such as the host being
# unavailable), it leaves the file alone and tries it again next
# time it runs.
#
# If it decides the file cannot be delivered (for example, there is
# no such address) it renames it to be /UNDELIVERABLE-MAIL/.address
# (thus MAILER does not try again to send it, but it is not
# destroyed). MAILER also sends you (i.e. to your mailbox) a
# message (called a negative acknowledgement) telling you that the
# message was undeliverable and why. You may delete the
# undeliverable file, requeue it with a corrected address, etc.
# using MAILSTAT. If for some reason the negative acknowledgement
# cannot be delivered (for example, your mailbox is in use),
# it too is queued (it is placed in the file
# ]--UNSENT-NEGATIVE-ACKNOWLEDGEMENT--[.address in your directory,
# where "address" is still the address of the original,
# undeliverable message) and delivered later.
#
# 2. Running MAILER
#
# Here is a sample run, which was done while logged in as and
# connected to SUSSMAN.
#
# @MAILER
#
# No acknowledgments for SUSSMAN
# Queued mail from SUSSMAN
# BURCHFIEL@, sent ok, deleted.
# CLEMENTS@, sent ok, deleted.
# @
```

```
# MAILER processes queued mail and queued negative acknowledgements
# in the logged in directory and the connected directory (if
# different from the logged in directory).  Before processing a
# message, it types its address.  After processing the message, it
# types the outcome, which is one of:
#
#
#   1) deleted - the file was deleted (after successful
#      transmission)
#   2) requeued - the file was left alone - it couldn't be
#      delivered now, but will be processed again later
#   3) renames - the message was undeliverable, and the file was
#      renamed as described above
#
# Between the address and the outcome, messages may tell you why
# the file was undeliverable, that it was sent OK, etc.
```

```
#                              MAILSTAT
#
#
# MAILSTAT lists all queued and undeliverable mail in the connected
# directory.   It   also   accepts   commands   to   manipulate   the
# undeliverable messages - they can be deleted or put back  on  the
# queue to be mailed (with a different address if desired).
#
# See Examples below of MAILSTAT features upon usage.
#
#
#
# EXAMPLES ----     (user typing is underlined)
# @mailstat
# MAILSTAT 1B(11)
# Type ? for help
# *?
# MAILSTAT lists existing queued and undeliverable mail in the
# connected directory.  It also permits manipulation of
# undeliverable mail.
#
# Commands are:
#
# Q - list queued mail
# U - list or manipulate undeliverable mail
# H - halt - exits program
# ? - types this message
# carriage return - lists both queued and undeliverable mail
#
#
# *Queued mail:
#
# AIGHES     27-JAN-75 12:04:53
# AMSDEN     27-JAN-75 12:04:54
# CWILLIAMS     27-JAN-75 12:04:57
#
# No undeliverable mail.
#
# *u
# type ? for help
# **?
# Commands are:
#
# L - list undeliverable mail
# carriage return - same as L
# H - halt - exit to higher level (back to *)
# ? - types this message
# M - manipulate undeliverable mail -
#     Same as L, but after each item is listed, waits for you to choose
#     one of the following options.  Except as noted below, the
#     option must be confirmed with a carriage return, or may be
```

```
#      aborted with rubout (del).
#
#    S - save - does nothing - status of mail is unchanged
#    carriage return - same as S, but no confirmation required
#    D - delete message
#    Q - requeue message
#    A - address change - allows you to specify a different address for the
#        mail and requeues the mail for the modified addressee.
```

MIDAS

MIDAS is the assembler normally used at M.I.T. PROJECT MAC on their PDP1Ø's, and under their ITS timesharing system. A version of MIDAS has been modified to run under TENEX. It does not use 1Ø/5Ø UUO's or ITS UUO's. It uses TENEX JSYS's.

Documentation on the MIDAS language will be found in PROJECT MAC AI memo number 9Ø, "MIDAS", by Peter Samson".

The following are the only changes to MIDAS for the TENEX version.

1. The default input file extension is "MID" instead of ">".

2. MIDAS's initial symbol table contains the TENEX JSYS definitions (but not the error codes).

It should be noted that the binary output of MIDAS is not compatible with LOADER. PROJECT MAC's loader has not yet been TENEX-ized. A loader for the SBLK format produced by MIDAS is being worked on.

MAILBOX

The "mailbox finder" is a TENEX subsystem used to determine the appropriate user name and site name to which an individual's mail should be sent. It may be run directly from a terminal or called as an inferior procedure. This latter mode may allow mail forwarding by FTP, SNDMSG and MAILER. For instance,

The mailbox data is a text file stored at each TENEX site supporting the mailbox finder. Individual sites may wish to maintain mailbox information particular to their needs. An example is the forwarding of mail to either the BBN-TENEX system or the BBN-TENEXA system, depending on where the addressee maintains his mailbox, and independent of which system initially receives the mail.

The mailbox finder returns a mailbox specification as "user@site". The input to the mailbox finder is also a user@site pair, where the site may be omitted. (The default site is the local site.) When a person wishes to receive mail under any of several synonyms, he has the text file updated to include all such synonyms. Whenever a person or program asks the mailbox finder about such a synonym (including the mailbox itself), the mailbox user@site pair will be returned. Thus, for example, the text file entry for user JONES with mailbox on BBNA could look like

```
\                              ;backslash delimits mailboxes
JONES@BBN-TENEXA               first entry is mailbox
JONES@BBN-TENEX                ;synonym for mail forwarding
JJJ@(NIC ID)                   ;network information center identification
\
```

A person at a terminal could produce the typescript

```
@MAILBOX JONES@BBN
JONES@BBN-TENEXA
*
@
```

An additional feature of the mailbox finder is the ability to limit the search to the specified site. This is accomplished by a "V" and an "S" on the second argument line, which is shown blank in the example above. The V says begin matching only to the specific site given, and the S says suppress hunting. If hunting is allowed (which is the default case), the mailbox finder would find all matches to JONES@BBN, and then continue to find any matches to JONES at any other site, and finally any NIC IDs which are "JONES".

Other possible arguments are "N", which says begin matching only NIC IDs, and "A", whih says begin matching all actual sites (but not NIC IDs). The default, as shown in the example above, is "A", begin matching all sites, then hunt to NIC IDs.

The text file format includes a mechanism for flagging a mailbox as belonging not to a person, but rather to a group. In this case a file name is returned. The group's secretary may update the file so named as the group's constituency changes. Systems programs such as FTP may use this feature to automatically distribute copies of group mail. When run from a terminal, the mailbox finder types an error comment when the user name supplied is a group name.

As suggested by the example and discussion above, NIC IDs are treated as users at a site named "(NIC ID)". Matching to them is under separate control, however, as specified by the N argument. For example, either

```
     @MAILBOX JJJ
     NS
or
     @MAILBOX JJJ@(NIC ID)
     VS
would produce the single reply
     JONES@BBN-TENEXA
     *
     @
and would not match, for instance, a mailbox
     \
     JJJ@BBN-TENEXA ;Johnson, J.J.
     \
On the other hand,
     @MAILBOX JJJ
     AS
would match only Johnson
     JJJ@BBN-TENEXA
     *
     @
To match both, a user could type
     @MAILBOX JJJ
     JJJ@BBN-TENEXA
     JONES@BBN-TENEXA
     *
     @
```