MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Artificial Intelligence
Memo No. 147A (Revised 147 by T. Knight) September 1971

DDT REFERENCE MANUAL

Eric Osman

ABSTRACT

This memo describes the version of DDT used as the command level of the A.I. Laboratory Time Sharing System (ITS). Besides the usual program control, examination, and modification features, this DDT provides many special utility commands. It also has the capabilitiy to control several programs for a user and to a single instruction continue mode and interrupt on read or write reference to a given memory location. This memo was prepared with the assistance of Donald E. Eastlake and many others.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0002.

Table of Contents

I. Introduction 01

II. When DDT is loaded 02

III. System Monitor commands 03

IV. Monitor commands doing DDT type functions 18

V. DDT Expressions are composed of: 20

VI. "comma" and "space", the field separaters ',' ' ' . . . 22

VII. DDT Operators 24

VIII. Priority of DDT Operators 27

IX. Arguments for "DDT" Format commands 28

X. DDT expression type-out 29

XI. Examining and Modifiing Memory with DDT 32

XII. Symbol Management with DDT 38

XIII. Program control with DDT 41

XIV. Job commands that have no "monitor" command equivalent . 45

XV. DDT output control commands 50

XVI. Typing Error Correcting 52

XVII. Errors reported by DDT 53

XVIII. Error messages typed because DDT received an interrupt from an inferior 57

XIX. Some more-advanced DDT features 58

XX. Special DDT locations 61

XXI. Cross Reference Index of DDT Format Commands 64

XXII. Index of Monitor Commands 74

XXIII. Sample Console Session 76

I. Introduction

A. DDT, standing for Dynamic Debugging Technique, was originally developed for the PDP computers made by Digital Equipment Corporation. In ITS, DDT has been expanded to assume the role of a monitor, interpreting commands for logging in and out, loading programs, sending messages to other users, and many other monitor-type operations. The top-level DDT loaded automatically is known as a "HACTRN". This is easy to remember because HACK means job, and "TRN" suggests "transfer"; HACTRN transfers between hacks. Although this mnemonic exists for the word "HACTRN", it is not the origin of it. The correct pronunciation is "hack-tran".

B. Main uses of DDT

1.) Creating, transferring between, and killing jobs

2.) Loading and running programs

3.) Controlling programs

(a.) Memory location examination and modification
(b.) Controlled execution and error condition reports

C. Two types of commands

1.) "Monitor" format commands

(a.) always begin with a colon
(b.) end with space, altmode, or carriage return
(c.) usually have a synonomous "DDT" Format command
(d.) are allowed to consist of only a file name

2.) "DDT" format commands

(a.) a single character preceded by 0, 1, or 2 altmodes
(b.) may have a prefix set of arguments (before command)
(c.) may have an infix argument (between altmode and command)
(d.) may have suffix arguments (after itself)

II. When DDT is loaded

A. user types control-Z on a free console

B. ITS loads a copy of DDT for the user, known as the user's HACTRN

C. DDT begins by executing some initial operations

1.) types ITS and DDT version numbers currently in use

2.) types how many users are on the system and other useful information (e.g. parity error reports)

3.) types out the message of the day which is the contents of a file named SYSTEM MAIL on the SYS device

4.) The information typed by DDT when it is loaded may be obtained again at any time by use of the :SSTATU and :VERSIO commands.

D. DDT will be automatically reloaded if a fatal error is encountered in its execution.

III. System Monitor commands

Some monitor commands never take arguments. Commands of this type will be executed by DDT as soon as you type the command followed by a space or a carriage return. Another type of monitor command always takes an argument. This type must be separated from its argument by a space or carriage return. After the argument or arguments have been typed, you signal to DDT that you are through typing by typing a terminating carriage return, except that a space will suffice for termination if the argument list for the particular command is always exactly one word. The third type of monitor command is one which may optionally take an argument or arguments. This type has the same input format as the last type when issued with arguments. However when giving this type with no arguments to DDT you must end it with a carriage return. If the command takes file name information as arguments and you want to issue the command with no arguments, you must type two carriage returns, one to separate the command from its arguments and another to tell DDT that no arguments will follow. If DDT is given an unrecognizable command, it tries to find a system program with the name given as the command. If found, DDT loads and starts it running. Actually if you type :COMMND and :COMMND is not a recognized command, DDT acts as though you typed COMMND|K. Therefore if you want to load and start the system program called TECO, you can do it by typing :TECO.

A. :? and :??

1.) :? causes DDT to type a list giving a brief description of most ':' type commands.

2.) :? ? causes DDT to list every monitor command, the internal address within DDT that is the beginning of that command's internal code, and usually a description of the command.

B. :LOGIN

1.) synonomous to the $U command

2.) takes one argument consisting of 1 to 6 sixbit characters

3.) DDT sets its UNAME to the given argument

4.) system attempts to log user in with given argument, and if successful:

(a.) sets UNAME (user name) to the argument
(b.) sets SNAME (system name) to the argument
(c.) The UNAME may not be changed by the user except by logging out and logging in again
(d.) The SNAME may be changed with various commands during the console session
(e.) types 'INIT.' if a file .DDT. (INIT) exists on the disk in "uname"'s file directory
(f.) types contents of the file "uname" MAIL from the COM device if it exists
(g.) deletes file "uname" OMAIL from the COM device if it exists
(h.) renames "uname" MAIL to "uname" OMAIL on the COM device
(i.) executes contents of .DDT. (INIT) as DDT commands if it exists

5.) will fail for any of the following reasons:

(a.) someone else is already logged in with specified argument
(b.) :LOGIN command given at a console that is already logged in
(c.) illegal character in argument

6.) examples of logging in: :LOGIN FOO or FOO$U, the first requiring a carriage-return or space terminating it to have effect

7.) the :LOGIN command may be used on an already logged in console to examine the core image of another user's job

(a.) suppose you are logged in as anything except FOO
(b.) suppose someone else is logged in as FOO and owns a job

named JOB1

(c.) you type FOO$U

(d.) although you are not logged in as FOO, by now typing JOB1$J, you will be able to examine the core image of JOB1, because 'FOO$U' caused DDT to set its internal UNAME to FOO.

C. :LOGOUT

1.) synonomous to the $$U command

2.) causes system to attempt to log out the user

3.) takes no arguments

4.) the file "uname" OMAIL on the COM device is deleted if it exists

5.) entire procedure tree is expunged from system, thus logging out the user

6.) The console is now "free", and a message to that effect is typed thereon by the system

7.) examples of logging out: :LOGOUT followed by a space, or $$U

8.) The :LOGOUT command may also be given at a console which has a HACTRN associated with it, but is not logged in

9.) It will fail if the running DDT is not top level, in which case the inferior DDT responds with 'NOT TOP LEVEL, CAN'T LOG OUT ?'

D. :JOB

1.) synonomous to the $J command

2.) This command is the basic DDT job control command. DDT will allow up to 8 jobs to be in existence simultaneously. Only one is the current job at any one time. The :JOB command controls which job is the current job, and creates new jobs. The current job may belong to another user, in which case you may only examine it.

3.) When typed with no argument

(a.) if no jobs have been created, no action is taken by DDT

(b.) if only one job exists, DDT types the name of that job followed by $J, and it remains the current job

(c.) if more than one job exists, DDT causes a job other than the current job to become the current job

(d.) in either of the last two condition, if the new present job is not really a job, but only a read-only core image of one, DDT types '#'

(e.) created jobs are remembered by DDT such that repetitive $J commands with no argument will cause every job to become the current one once before repeating

4.) with one argument of 1-6 SQUOZE characters

(a.) if a job exists with the given argument, that job becomes the current job

(b.) if no job exists with the given argument, DDT tries to create one with the given argument as its name. If DDT's "uname" has been changed with the $U command and a job with the given argument is the name of a job owned by the user logged in as that "uname", DDT types '#' after the '!' and the created job allows location examination of that other user's job. If the "uname" of DDT had been changed, but the given argument is not the name of any job under the "uname", DDT changes its "uname" back to that of what you logged in as. Then DDT types "uname"$U! and tries again as though the "uname" in DDT was never changed. newly created jobs have 2000(8) words of zeroed core.

(c.) If a job with given argument as its name was disowned, it is reowned

5.) examples of job selections: :JOB FOO or FOO$J with a space after the first example

6.) if a new job is requested, but too many jobs already exist, the command has no effect, and DDT types TMJ? signifiing 8 jobs already exist.

7.) the job name HACTRN is special and will allow examination of one's own HACTRN but modification of locations therein is not permitted

8.) the special job name SYS allows examination of any absolute location in the system.

9.) the job name PDP6 or PDP10 allows access to all of the core in the PDP-6

E. :LISTJ

1.) synonomous to the $$V command

2.) causes DDT to type a list of all the jobs it knows about

3.) every element of the list is typed on one line and has the

following information in the given order:

(a.) the JNAME of the job
(b.) status code for the job

(1.) "minus sign" for a job just created
(2.) 'R' for a running job
(3.) 'P' for a job which may be proceeded from some interrupt such as control-Z or execution of a .VALUE
(4.) n'B' for a job that has been interrupted at the nth breakpoint

(c.) the job index which is a unique number assigned to the job by the system at creation time

4.) an asterisk is printed before the current job entry in the job list

5.) If any one of the jobs is only a read-only core image of another job, the "uname" to whom it belongs will appear in the list before the "jname"

6.) example: The command :LISTJ could yield:

```
   T P 25
 * J 1B 27
```

F. :LOAD

1.) synonomous to the $L command

2.) used to load a binary file into the current job with its symbols

(a.) first core is zeroed
(b.) the old symbols are erased
(c.) any breakpoints will remain

3.) uses three arguments to determine what to load from where, however any of the arguments may be ommitted while typing the command

(a.) filename

(1.) one or two words separated by a space designating file to be loaded
(2.) If the current device is DSK, one of the file names may be the '<' or '>' signs in which case it is not interpreted literally, but rather it stands for a numerical name that is the greatest or least in a group. For example, if the files FOO 21, FOO 16, and FOO 34 exist in a directory, typing :LOAD FOO > would cause FOO 34 to be loaded, and

:LOAD FOO < would mean load FOO 16.

(3.) If the two word file name is ommitted from the command, the same file as in the last :LOAD or :DUMP is used. If no command has been given to load or dump a file this terminal session yet, the assumed name is @ BIN. If only one of the two file names is supplied, DDT replaces the first of the two assumed names with the given one.

(4.) If more than two file names are supplied, the third is interpreted as a device and the fourth is interpreted as a system name.

(b.) device code

(1.) designates from what device the file is to be loaded (DSK, PTR, UTn, etc.)

(2.) If the device code is ommitted, the same device as was used in the last :LOAD or :DUMP is used, or DSK is used if none has been previously specified.

(3.) The device code is typed followed by a colon, unless the $L format is used, in which case the device may also be typed as a prefix argument. For instance, DSK$L, $L DSK:, and :LOAD DSK: all say to load a file from the disk. A number may also precede the command meaning the microtape drive with that number, for instance '3$L' means to load a file from microtape drive #3.

(c.) system name, known as "sname"

(1.) This argument is initially set to the name the user logged in as. It can be changed by many commands including :LOAD. When used by the system to load a file from a directory device such as the disk, the "sname" designates from who's directory the file is to be sought.

(2.) This argument must be followed immediately by a semi-colon.

(3.) Specifiing the "sname" in a :LOAD or $L command universally changes the "sname". That is, subsequent commands for directory listings, file deletions, file dumps and loads, etc. may easily be affecting another user's files than your own, so caution must be exercised.

4.) examples of :LOAD commands:

(a.) :LOAD DSK:HUR;TICTAC BIN
(b.) SYS$L TS TECO
(c.) :LOAD TS TECO DSK:SYS; (this is the same as the last example)
(d.) $L CHESS >
(e.) DSK$L SYS; HACTRN

5.) after typing :LOAD followed by a space or $L, DDT will type a space automatically to separate the command from any arguments that might follow

6.) typing a load command from an unlogged-in console will have no effect and produce the response from DDT, 'LOGIN?'

7.) If DDT fails to find the file it is looking for, it will type: 'FILE NOT FOUND nnnn, dev?' where nnnn is an address in DDT within the load routine, and "dev" is the device that DDT tried to load the file from

8.) If the :LOAD command is issued with no current job, DDT types out 'JOB? ' and no action is taken

9.) If the current job is only looking at another job's core image, DDT will type 'JOB NOT INFERIOR?', if a :LOAD command is given.

G. :SYMLOD or :SL

1.) take arguments like :LOAD

2.) causes only the symbols for the specified file to be loaded into the current job after any previously existing symbols are deleted.

H. :DUMP and :PDUMP

1.) :DUMP is synonomous to the $Y command

2.) used to save a core image of the current job

3.) same format as :LOAD command

4.) saves symbols

5.) if a core image is :PDUMPed instead of :DUMPed, any read-only pages will be recorded such that the next time the image is loaded, the same pages will be read-only. Also, if more than one copy of a :PDUMPed file is loaded, all read-only pages will be shared.

I. :KILL

1.) synonomous to the $|X. command.

2.) Takes no arguments

3.) Causes the current job to be destroyed, and all system references to it erased

4.) If another job still exists after DDT kills the current one, DDT will execute a :JOB command with no argument to minimize the time in which there is no current job

5.) example:

(a.) suppose you have two jobs A and B, with A the current job
(b.) now you type :KILL followed by a space, and DDT will type B$J and B is now the current job, with job A having been killed
(c.) you could have also typed $|X. and produced the same result

6.) typing the command :KILL or $|X. when no jobs exist causes the response 'JOB? ' and no action is taken

7.) The dot "." is required after '$|X' to lessen the chances of you unintentionally issuing the "job destroy" command.

J. :LISTF

1.) synonomous to the control-F command

2.) used to obtain a file directory listing from the system

3.) Takes an argument list like :LOAD, but only the device and "sname" are used

4.) the control-F form of the command may have no argument before it, a microtape drive number, or a device name. Drive number 0 implies the disk.

5.) either form of the command given without an argument will produce a directory of the current device and system name which was previously set by another command or are DSK and "uname" respectively, if they have never been changed.

6.) If the system does not recognize the device or system name supplied, DDT responds with '? '

7.) if the device is a non-directory device such as PTR or LPT, it's directory is the string 'NON-DIRECTORY DEVICE'

8.) examples of directory requests: :LISTF DSK:, UT4|F, :LISTF

PG;, 3|F

K. :PRINT

1.) causes DDT to list the contents of the specified file on the console

2.) Takes arguments in the same format as :LOAD. DDT remembers the file last :PRINTed and prints that one if no argument is supplied to the :PRINT. Initially, the :PRINT device is COM

3.) If the system fails to find the desired file, it types the same error message as it would for a :LOAD command.

4.) examples of listing commands: :PRINT COM:TTY MEMO, :PRINT, :PRINT RWG;HACK >

L. :DELETE

1.) synonomous to the control-O command

2.) used to delete files

3.) takes arguments like :LOAD, and the control-O form may be preceded by a device just as $L may.

4.) examples of file deletion commands

(a.) :DELETE PRIMES < RG;
(b.) UT3|O TEST 46
(c.) |O

5.) Typing |O or :DELETE with no arguments will cause DDT to attempt to delete the last file :PRINTed or :DELETEd, if no other intervening file commands have been issued.

M. :LINK

1.) synonomous to the $|F command

2.) takes two arguments, each a file specification as with :LOAD

3.) causes DDT to link the first fle specified to the second

4.) If a first file name is linked to a second name, any future reference to read the first name is interpreted by the system as a reference to the second.

5.) example of a link: Suppose you normally are logged in as FOO and frequently reference a program on the SYS device called TS MUSCOM. By typing :LINK FOO;U BIN SYS:TS MUSCOM, any future references for reading a file on your directory called U BIN will actually be interpreted as a reference to SYS:TS MUSCOM.

N. :FLAP

1.) synonomous to the $$|F command

2.) used to rewind and unload a microtape from the system

3.) takes an argument designating which drive should be unloaded

(a.) $$|F takes a prefix number
(b.) :FLAP takes a suffix argument

4.) examples:

(a.) :FLAP 3
(b.) 3$$|F

O. :PRGM

1.) used to get the name of the selected relocatable subprogram

2.) takes no arguments

3.) causes DDT to type out the selected program name or '"GLOBAL"' if the program is not relocatable or if a particular subprogram has not yet been selected

P. :LISTP

1.) used to get a list of relocatable subprograms that have been loaded into the current job

2.) every element of the list contains two elements itself:

(a.) the program name which is 1-6 SIXBIT characters

(b.) an octal full-word number whose right half is the starting address of the program and whose left half is the highest address in that program

Q. :MAIL

1.) used to prefix a user's mail file (user MAIL) with a message. This file is the one typed out on a user's console upon logging in.

2.) takes one argument, the "uname" to whom you want to mail the message, followed by the message which must be terminated by a control-C.

3.) What is actually added to the mail file is the following in the given order:

(a.) the sender's "uname"
(b.) the date
(c.) the time on a 24 hour basis
(d.) the message

4.) example:

(a.) suppose you are logged in as HUR
(b.) imagine today is June 15, 1971 and is half-past nine in the morning
(c.) suppose you type :MAIL PG HI, PHIL! followed by a control-C
(d.) the file PG MAIL on the COM device will now be prefixed with:

HUR 06/15/71 09:30:44 HI, PHIL!

R. :SEND

1.) used to send a message directly to a console

2.) if the above example were executed with :SEND instead of :MAIL, what would have been prefixed to the mail file is sent directly to the console on which the person logged in with the given argument is. If that person is not logged in anywhere, then immediately following the ':SEND PG ', DDT types ' (MAIL)' and the message follows the same course as :MAIL would have given it

3.) example of sending a message:

(a.) suppose you type ':SEND R ARE YOU WINNING?|C'
(b.) assume you are logged in as HUR
(c.) if someone is logged in as R and depending on the date and time, the following could appear on R's console: 'MESSAGE FROM HUR HACTRN 10:13:05 ARE YOU WINNING?'

S. :GAG

1.) used to control receipt of messages sent with :SEND

2.) takes a numerical argument which is bit decoded as follows with the low order bit #1.

(a.) #1 refers to other user's inferiors
(b.) #2 refers to other user's HACTRNs
(c.) #3 refers to your own inferiors
(d.) #4 refers to your own HACTRN

3.) If a bit is a one, messages are allowed to come through from the sender that bit corresponds to, and if it is a zero, messages are barred from being received

4.) The initial mode for :GAG is 17

5.) examples of gagging:

(a.) :GAG 17 will allow messages to be received from anywhere
(b.) :GAG 14 will not allow reception from anyone except yourself

T. :DISOWN

1.) synonomous to the $$|K command

2.) used to disown the current job, letting it sit in core unattached to any HACTRN

3.) a disowned job may be reowned by the :JOB command

4.) DDT simulates the $J command with no arguments following the execution of a :DISOWN

5.) this command never takes any arguments because the job being disowned is always the current one

U. :XFILE

1.) takes an argument list like :LOAD and executes the specified

file as DDT commands

2.) DDT stops reading the file when it encounters a control-C or control-@ (ASCII values 3 and 0 respectively)

3.) example: :XFILE DDT DOTHIS

V. :BUG

1.) used to report bugs in ITS or any other system program

2.) takes no argument, only a space followed by the message, followed by a control-C.

3.) :BUG ". . . message . . ."|C is synonomous to :MAIL SYS ". . . message . . ."|C

4.) example of a gripe: ':BUG IN TECO, ONE OUGHT TO BE ABLE TO TYPE A QUESTION MARK AFTER TECO FAILS TO FIND A FILE. TECO SHOULD THEN SAY WHAT FILE IT WAS LOOKING FOR AND COULDN'T FIND.|C'

W. :SLEEP

1.) causes DDT to sleep for awhile

2.) takes a numerical argument designating the number of thirtieths of a second DDT should sleep

3.) During the time DDT is asleep, no commands except control-G will be recognized. Control-G will cause DDT to wake up prematurely.

X. :SSTATU

1.). causes DDT to type out the following information about the system:

(a.) 'ITS BEING DEBUGGED!', if this be the case.
(b.) 'ITS GOING DOWN IN ' followed by hours, minutes, and seconds until system death, if ITS is set to go down.
(c.) Also, if ITS is set to go down, DDT prints on the TTY the contents of the file called DOWN MAIL from the SYS device, if it exists.
(d.) '# USERS = ' followed by the decimal number of users on the system.

(e.) 'MEM ERRS' n'/HOUR' if there are any.

2.) takes no arguments

Y. :ERR

1.) takes no arguments

2.) This command causes DDT to use $Q as an I/O channel .STATUS word and type out the contents of the ERR device.

3.) Example: Suppose the current job just interrupted because of an I/O channel error on channel 3. To find why the I/O error occured, type the following: '.IOS+3/'. DDT will type the .STATUS of channel 3. Now type :ERR (followed by a space), and DDT will open the :ERR device and probably tell you why the error occured.

Z. :VERSIO

1.) causes DDT to type version numbers of ITS and DDT currently running

2.) takes no arguments

3.) This command is useful to help associate particular bugs in ITS and DDT with particular versions.

AA. :WALLP

1.) causes DDT to begin outputting to another device as well as the teletype.

2.) takes an argument list in the same format as :LOAD

3.) The initial current data for the :WALLP command is such that if the :WALLP is issued without an argument list, it is synonomous to :WALLP LPT:WALL PAPER

4.) typing control-E closes the file onto which DDT is writing because of a :WALLP command

5.) Remember that until you type control-E following a :WALLP to

the line printer, no other user may use the line printer

BB. :$ colon-altmode

1.) This command is used to enter comments

2.) DDT will ignore everything typed between the ':$' and the next altmode.

3.) Example of a comment:

':$THIS IS A COMMENT$'

IV. Monitor commands doing DDT type functions

A. :V

1.) synonomous to the control-V command

2.) turns DDT typeout back on, if it was turned off with control-W

3.) takes no arguments

B. :VK

1.) same as :V except it also causes DDT to type a carriage return and linefeed to indicate it is alive.

2.) The K stands for "kerchink"

C. :START or :GO

1.) may take one argument

2.) causes DDT to start the current job at the specified location

3.) with no argument, causes DDT to start at the assembled or dumped starting address.

D. :GZP This is the same as :GO or :START except that the teletype is not given back to the job, allowing subsequent DDT commands during job execution. This allows for running several jobs "simultaneously".

E. :CONTIN

1.) takes no arguments

2.) synonomous to the '$P' command (without an argument)

3.) causes DDT to resume execution of the current job after an interrupt (such as you having typed control-Z)

F. :PROCED

1.) synonomous to the control-P command

2.) same as :CONTIN except the teletype is not given back to the running job

G. :VP is the same as :V followed by :CONTIN

H. :SLIST

1.) takes no arguments

2.) causes DDT to list the symbols of the current job.

V. DDT Expressions are composed of:

A. Numbers

1.) Any continuous string of characters including:

(a.) digits
(b.) an optional decimal point implying base ten
(c.) An 'E' for standard exponent format input

2.) are interpreted as integers unless a decimal point is followed by digits, in which case the number is taken to be floating decimal

3.) Examples of DDT numbers:

(a.) 3
(b.) 0.4
(c.) -7.9
(d.) 5.6E7

B. Symbols

1.) a string of characters including only letters, digits, period, percent sign , dollar sign, or an altmode.

(a.) If a period is in the symbol, some other constituents must exist, and at least one of them must be a non-digit
(b.) A symbol may not be all digits

2.) must be defined in order to be used in an expression, one of the following:

(a.) opcode such as JRST or CIRC
(b.) user-defined symbol such as FOO or LUP
(c.) DDT special symbol such as $Q
(d.) '', which causes DDT to inclusively "or" a 20 into the left half of the expression being typed in.

3.) Examples of symbols:

(a.) B7
(b.) PRICON
(c.) AB%

C. Defined operators which are:

1.) one or two characters usually taking a prefix and suffix argument

2.) not a command

3.) examples of some operators:

(a.) * (integer multiply)
(b.) # (exclusive OR)
(c.) $! (floating divide)

D. One of three kinds of character strings:

1.) ASCII string, which is followed by an altmode and preceded by '$n"' where '$' is altmode and 'n' is a number and '"' is a double quote. To enter control letters within the string that normally cause special DDT functions, type an uparrow '|' followed by the character to be "controlled". To enter an altmode or uparrow in the ASCII string, precede it with a control-Q. Example of an ASCII string that could be typed within a DDT expression: '$69"FOOEY$'

2.) SIXBIT string, which is entered in the same format as an ASCII string, except that ''' is used in place of '"'. Example of a SIXBIT string: '$2'E4$'

3.) SQUOZE string, which consists of 6 or less SQUOZE characters preceded by '$n&' and terminated with the first non-SQUOZE character following the '$n&'. This non-SQUOZE character is not taken as part of the SQUOZE input format, but acts as itself. To illustrate all of this, suppose we want DDT to tell us the numerical value of the SQUOZE string 'BARF'. We type to DDT: '$0&BARF=' and DDT types out 11301666000. Note that although the '=' ended the SQUOZE string, it also served the normal function of the '=' command. The 'n' in '$n&' is left-shifted 32 bits to the left and ANDed with 740000, and becomes the contents of the 4 code bits in the SQUOZE word.

VI. "comma" and "space", the field separaters ',' ' '

A. Let a field denote any DDT expression not containing any spaces or commas, and let F stand for any field. In DDT, a string of fields separated by spaces or/and commas are interpreted in a similar way that MIDAS interprets them. More specifically, a few standard typein formats exist in DDT:

1.) F,,

(a.) causes F to be shifted 18 bits to the left and added to whatever comes after the ',,'. Any fields occuring after the ',,' will be "anded" with 777777 and added into the expression.
(b.) most often used for half-word typein, such as the following: 3,,4 evaluates to 3000004
(c.) other examples of F,, format:

(1.) 3,,6,,1 evaluates to 3000007

(2.) 3,,6,1 is synonomous to the last example

2.) F F,

(a.) causes the second field to be leftshifted 27 bits, "anded" with 17, and added to the first field. Any subsequent fields in the expression will be "anded" with 777777 and added on.
(b.) most often used in standard format instruction typein. PUSHJ 3,4 evaluates to 260140000004
(c.) other examples:

(1.) 4 2,7 evaluates to 100000013

(2.) 1 2,3,,4 5 evaluates to 100000015

3.) F,

(a.) causes the field to be "anded" with 17 and put in the AC "field" of the word, except when other fields follow in the same expression in which case it is not a standard form
(b.) mainly used to enter a number in the AC "field" of a word, for instance 3, evaluates to 140000000

B. Any other formats are dealt with in a common way. That is, a string of fields separated by spaces, and commas are evaluated in the same way. The first field is taken as a 36 bit quantity and each of the rest are added in after having been "anded" with 777777, with carrying

suppressed after the half-word point. The most common unstandard format is F F such as JRST 3 which evaluates to 254000000003. Other examples:

1.) 3,4 which equals 7

2.) 4,5,,6 which equals 17

3.) AOS,JRST which equals AOS

4.) 2,3 5,,1 which equals 13

VII. DDT Operators

A. plus sign '+'

1.) infix operator

2.) takes two arguments

(a.) any legal DDT expressions
(b.) treats its arguments as signed 36-bit integers

3.) adds the two arguments

4.) ommitting the first argument implies 0.

B. minus sign '-'

1.) infix operator taking arguments like the plus sign

2.) subtracts second argument from the first

3.) ommitting the first argument causes that argument to be 0, allowing for the unary minus function

C. asterisk '*'

1.) infix operator like plus and minus

2.) multiplys its arguments as integers

3.) omitting the first argument implies 1.

D. exclamation point '!'

1.) divides its first signed integer argument by the second, returning an integer quotient, and the remainder is discarded.

2.) Omitting the first argument implies 1.

E. a single altmode followed by either:plus, minus, asterisk, exclamation $+ $- $* $!

1.) The alt-mode in front of either of the above four operators causes the operator to treat its left and right operands as floating point numbers. They all return a floating point result

2.) examples:

(a.) 1.0$!3.0 evaluates to 0.33333333
(b.) 1234.5679$!.45 evaluates to 555.55555
(c.) 1.34$+7.9 evaluates to 9.2400000

3.) Omitting the first operand for '$+' and '$-' implies 0.

4.) Omitting the first operand for '$*' and '$!' implies 1.0. This allows for using '$!' with only a right operand to get the reciprocal. Example: '$!3.0' is 0.333333 ...

F. sharp sign '#'

1.) infix operator

2.) exclusive ors its two operands as 36 bit quantities (add without carry)

3.) example: 3#5 evaluates to 6

4.) Omitting the first operand implies -1, thus complimenting the second argument.

G. ampersand '&'

1.) infix operator

2.) logically ands its two 36 bit arguments

3.) example: 3&5 evaluates to 1

4.) Omitting the first argument implies -1, causing the unary '&' function to be a no-op; that is it does nothing.

H. parenthesis '(' and ')'

1.) take an infix expression as an argument

2.) The expression within the parenthesis is swapped. Then, after DDT receives the entire expression, the value gotten from swapping what

was in the parenthesis is added on, unless an arithmetic operator immediately preceded the open parenthesis in which case instead of adding at the end, the specified operation is performed.

3.) mainly used to enter quantities in the index field of an instruction

4.) examples:

(a.) 3(4) evaluates to 4000003
(b.) ((4)) evaluates to 4
(c.) JRST(5) evaluates to 254005000000
(d.) 3*((5)) evaluates to 17

I. altmode backarrow '$_'

1.) infix operator

2.) left shift operator

3.) returns the value of its left argument logically shifted the number of places specified by its right argument

4.) examples:

(a.) 1$_2 evaluates to 4
(b.) 16$_-2 evaluates to 3

5.) Omitting the first argument implies -1.

J. altmode altmode backarrow '$$_'

1.) infix operator

2.) floating scale operator

3.) treats its left argument as a decimal floating number and multiplies it by 2|N where N is the integer argument on the right

4.) examples:

(a.) 6.0$$_3 evaluates to 48.0
(b.) 3.4$$_7 evaluates to 435.2
(c.) 435.2$$_-7 evaluates to 3.4

5.) Omitting the first argument implies -1.0.

VIII. Priority of Operators

A. There is a priority order for operations in DDT. For '*' to have a higher priority than '+' for example, means that '*' has a higher binding power than '+'. That is, whether you type 3*4+2 or 2+3*4, DDT still does the multiplication first and gets the answer 16. The priority order from highest to lowest is as follows:

1.) '#' and '&', '$_' and '$$_'

2.) '*', '$*', '!', and '$!'

3.) '+', '$+', '-', and '$-'

B. Changing the priority

1.) In DDT, the mathematical grouping symbols are '<' and '>'

2.) When surrounding an expression they cause the included expression to be evaluated before DDT even looks at what is outside the angle brackets

3.) example: 3+4*5 evaluates to 27 but <3+4>*5 evaluates to 43

C. A string of operators of the same priority level in DDT causes them to be evaluated from left to right. Example: 3&4#5 is synonomous to <3&4>#5

IX. Arguments for "DDT" Format commands

A. Prefix arguments are used from left to right, so excesses on the right will be ignored.

B. Infix arguments will be ignored if unnecessarily supplied. For example, typing $$30 causes the permanent radix for type-out to be octal, and the 3 is ignored. Infix arguments more complex than single numbers must be enclosed in angle brackets. For instance, to type the 4 in $4R by expressing 4 as 1+1+1+1, you must type $<1+1+1+1>R. More specifically said, an altmode followed by an expression in angle brackets is interpreted by DDT to be an altmode followed by the numerical value of the expression. For example $<JRST> typed to DDT is like typing $254000000000.

C. altmode comma and altmode space '$,' '$ '

1.) '$,' and '$ ' separate arguments in multiple argument type-in.

2.) example: Typing 3$_2$,2$_1 causes DDT to evaluate the two expressions '3$_2' and '2$_1' as two prefix arguments.

D. Most commands take expressional arguments. The exception to this rule is commands that take an undefined symbol as an argument (probably one to be defined).

X. DDT expression type-out

A. Modes in DDT for type-out of expressions

1.) symbolic mode

(a.) absolute: DDT types out symbolic opcodes only eg. TLNE 3,@5(16)

(b.) relative: DDT types out symbols when convenient eg. TLNE C,@POINTR(INDEX)

2.) constant mode: DDT types out a single integer in the current radix with a decimal point if the current radix is ten.

3.) floating point mode: DDT types out floating decimal numbers preceded by a '#' if they are not normalized. Examples: #1.0 3.14E+26

4.) half-word: DDT types out two 18 bit quantities separated by ',,'. Relative and absolute apply in this mode, too. examples: 100,,236 or GO,,VARCOM+5

5.) byte mode: DDT types out the expression in bytes of any desired size

6.) ASCII mode: DDT types $1"string$ where "string" is the ASCII representation of the expression

7.) SIXBIT mode: DDT types $1'string$ where "string" is the SIXBIT representation of the expression

8.) SQUOZE mode: DDT types $n&string$ where "string" is the SQUOZE for the expression and "n" is the number represented by the 4 most significant bits in the word

B. Expression typeout commands

0.) '_' causes the last expression typed by you or DDT to be retyped symbolically

1.) '=' causes last expression typed by you or DDT to be typed out

in the current radix

2.) '$=' causes the last expression typed by you or DDT to be retyped in floating point mode

3.) '$$=' takes up to three expressions as arguments and causes them to be typed out in the mode they were typed in. Example: 3$,4$,5$$= produces:

```
<3>
<4>
<5>
```

4.) ''' (single quote) causes the last expression to be typed out in SIXBIT preceded by '$1'' and followed by a dollar sign.

5.) '"' (double quote) causes the last expression to be typed out in ASCII preceded by '$1"' and followed by a dollar sign

6.) ';' causes the last expression typed by you or DDT to be typed out in semi-colon mode which is initially floating point

7.) altmode semi-colon '$;' causes the last expression typed by you or DDT to be typed out in semi-colon mode and also causes the current mode to be changed to semi-colon mode

8.) '$$;' causes the same thing as '$;' but the permanent mode is also changed to semi-colon mode

C. The commands for setting the type-out mode are all one character followed preceded by 1 or two altmodes:

1.) one altmode set the current (temporary) and semi-colon mode

2.) two altmodes before the character does the same thing as with one altmode but also set the permanent mode and causes DDT to type a tab

3.) The meaningful mode command characters are S A R C H T F " ' &

(a.) $S and $$S change the mode to symbolic
(b.) $A and $$A change it to absolute addressing mode
(c.) $R and $$R change it to relative addressing mode
(d.) $nR and $$nR change the radix to n
(e.) $C and $$C change the mode to constant typeout

(f.) $H and $$H change the mode to half-word
(g.) $nT and $$nT set the mode in which expressions are typed out in bytes of size n
(h.) $F and $$F set typeout to floating point mode
(i.) $" and $$" set typeout to ASCII mode
(j.) $' and $$' set typeout to SIXBIT mode
(k.) $& and $$& set typeout to SQUOZE mode

4.) Besides these standard mode setting commands, the following radix setting commands are available:

(a.) $D and $$D sets the radix to ten
(b.) $O and $$O sets the radix to eight

XI. Examining and Modifiling Memory with DDT

A. $$Z command

1.) mainly used to zero core

2.) takes up to three prefix arguments:

(a.) first argument tells lowest location to be zeroed
(b.) second tells highest location to be zeroed
(c.) third argument, if supplied, means set core to that expression rather than zero
(d.) only one argument means zero core from that argument up
(e.) no arguments means zero all core, except that pure and absolute pages are ignored

3.) Example: LIST$,LIST+69.$,-1$$Z means set all memory between LIST and LIST+69. to -1, inclusive

4.) DDT types carriage return and linefeed upon completion of executing $$Z

B. The usual way to cause DDT to modify the contents of a location is to first give DDT one of many commands telling it to open a register. Only one location may be open at a time, but DDT remembers the locations opened in a point ring buffer, which may hold up to 8 register names. Locations in the point ring buffer may be conveniently referred to and reopened by several DDT commands, without having to retype the name of the location. The currently open location is the only one available for modification.

C. Special symbols that frequently change their value in DDT:

1.) Point '.' has the value of the most recently opened register that DDT rmembers. So .-3 represents the number 75 if the last location open was 100.

2.) altmode-Q '$Q' has the value of the last expression typed by
- for example, if DDT just typed out the contents of locatior

100, $Q is equal to that contents.

3.) '$nQ' where 'n' has octal value -1<n<11 stands for the nth previous value of $Q. $0Q is the same as $Q, $1Q is the second to last thing DDT typed, $2Q is the third to last, etc. $$nQ may also be typed, in which case the value of the corresponding $nQ replaces the value of $Q.

4.) altmode-greaterthan '$>' preceded by an argument has the value of $Q with the non-zero fields given in the argument replacing the corresponding fields in $Q. For instance if $Q is CAIE 4,5 , then 6,$> is evaluated to CAIE 6,5. This is mainly used to change only part of an instruction without retyping the instruction. The fields involved are any of the following:

(a.) opcode field, masked by 777000,,
(b.) accumulator field, masked by 17,
(c.) indirect bit, masked by 20,,
(d.) index field, masked by (17)
(e.) effective address field, masked by 0,,-1

5.) altmode-point '$.' has the value of the program counter of the current job.

D. To close the currently open location means to make it no longer available for modification. There are many DDT commands for closing the currently open location. Every one closes the location in exactly the same way, with the differences between the commands being in what they do after closing the current location. The contents of the current location opened may be changed only by you typing one of the register closing commands immediately preceded by an expression. In other words just giving a closing command not preceded by an expression does not affect the contents of the location being closed. DDT remembers, in a ring buffer, the last eight locations opened by all register openin

[illegible] which have a different effect o

the . ring buffer as explained are several register opening commands that easily cause locations in the . ring buffer to be reopened.

E. Commands for memory examination and modification

1.) slash '/'

(a.) DDT opens the register named by the right 18 bits of the last expression typed by DDT or the user
(b.) types out the contents of the location being opened
(c.) Turns register contents typeout back on if it was turned off by '$$!'.

2.) altmode-slash '$/' is the same as slash except that the left 18 bits are addressed instead of the right 18 bits.

3.) open bracket '|' is the same as slash but types out the contents of the register as a constant regardless of the current type-out mode.

4.) altmode open-bracket '$|' is like '|' except the left 18 bits are used instead of the right 18 bits.

5.) close bracket '|' is the same as slash except it causes the contents to be typed out symbolically regardless of the current type-out mode.

6.) altmode close-bracket '$|' is like '|' except it uses the left 18 bits of the last expression typed for its address.

7.) carriage return

(a.) DDT closes the currently open location, if one exists.
(b.) sets current type-out mode to the permenant mode
(c.) Turns contents typeout back on if it was turned off by '$$!'.

8.) altmode-carriage return closes any open register and opens the last one in . ring buffer

9.) linefeed

(a.) DDT closes any currently open register
(b.) opens .+1
(c.) types out its contents

[illegible] the last location opened by any command

except linefeed or uparrow. Every issuance of the linefeed or uparrow command causes the location name being opened to replace the remembered address in the . ring buffer.

10.) altmode-linefeed closes any open register and opens the one after altmode-return would have opened

11.) uparrow '|'

(a.) DDT closes any currently open location
(b.) opens .-1
(c.) types out its contents
(d.) See the explanation of linefeed above for the special . ring buffer effect of uparrow and linefeed.

12.) altmode-uparrow '$|' closes any open register and opens the one before altmode-return would have opened

13.) tab (control-I)

(a.) DDT closes any currently open register
(b.) goes to the next line and does the same thing as a slash would have caused

14.) altmode-tab '$||' same as tab but uses left 18 bits of the expression instead of right 18 bits

15.) altmode-altmode tab '$$||'

(a.) This command is the same as tab, except that DDT actually does an effective address calculation of $Q and opens the effective address. DDT will only look down twelve levels of indirect addressing.
(b.) Example: Suppose location 3 contains 3 and location 6 contains 7. You type '14/' and DDT types '3(3)'. Now if you type '$$||', DDT will go to the next line and type '6/ 7'.

16.) backslash '¬'

(a.) like tab except DDT doesn't change .
(b.) example: You type '3/'. DDT types out '55' which is the contents of register 3. Now you type '¬' and DDT opens register 55. You type linefeed and DDT closes 55 and opens register 4 because . was still 3.

17.) altmode backslash '$¬' is like '¬' except it uses the left 18 bits of the last expression typed instead of the right 18 bits.

18.) altmode altmode exclamation point '$$!' is the same as '/', except that DDT suppresses the typeout of the contents of the location

... '$$!'. DDT will continue supppressing the

typeout of location contents until you issue either the '/' or carriage return command.

F. '$N' command

1.) used to examine all locations in an area of core whose contents are NOT a certain value (usually zero) in bits masked by the contents of $M.

2.) The mask may be set by foo$M where "foo" is the mask. Initially, $M's contents is -1 (all bits turned on)

3.) takes up to three prefix arguments:

(a.) One argument causes DDT to type out the names and contents of all locations in the current job's core image whose contents are not equal to the argument in bits masked by the contents of $M. Thus 0$N causes DDT to type all non-zero core, if $M is -1.
(b.) For two arguments, only locations greater than or equal to the first argument are typed out, if their contents is not equal to the second argument in the masked bits.
(c.) With three arguments, DDT types all locations between the first and second arguments inclusive, whose contents are not equal to the third argument in the masked bits.

4.) works for noncontinuous core images

5.) Example: BOARD$,BOARD+77$,0$N will cause DDT to type all non-zero location names and their contents between BOARD and BOARD+77, providing the mask is all ones.

G. '$W' command

1.) used to examine all locations whose contents are equal to an argument in bits masked by the contents of '$M'

2.) takes up to three arguments with the arguments playing the same role as in the '$N' command except that the last argument is the word to be searched for. DDT types all locations and their contents if the contents matches the last argument in the bits masked by the contents of $M.

3.) Example: Typing 777000,,$M followed by JRST$W will cause a type-out of all locations containing a JRST instruction of any type.

H. '$E' command

1.) used to print out all locations whose effective address is the given argument. If more than one argument is supplied, the nonlast arguments are used the same way as in the $W and $N commands.

2.) Example: 0$,17$,LOC$E will cause the type-out of any accumulator and its contents whose effective address is LOC.

3.) When the '$E' command is being used to examine the system, DDT actually does an '$w' with $M containing 0,,-1. This is because it is pointless to do an address calculation with the system accumulators which are constantly changing.

I. All of the commands '$N', '$W', or '$E' leave the locations listed in the . ring buffer, and . is set to the last location listed. Typing any character except ^B, ^E, ^V, ^W will stop a listing prematurely.

XII. Symbol Management in DDT

A. DDT uses two words per symbol in its symbol table. One word holds the numerical value for the symbol. The other word holds the SQUOZE for the name of the symbol. The code bits in the SQUOZE are used to tell DDT what type of a symbol it is. The different types are as follows:

1.) full symbol; may be typed in an expression and DDT will type it out during symbolic type-out

2.) half-killed symbol; will never be seen in type-out, but may be typed in

3.) killed symbol; no longer available for type-out or type-in

4.) program name

B. How to define a symbol in DDT with a colon ':'

1.) Typing sym: defines the full symbol "sym" equal to .

2.) expr$,sym: defines the full symbol "sym" equal to the expression "expr"

3.) sym$: sets the relocatable subprogram name to "sym"

C. Killing and half-killing symbols with '$$K' and '$K'

1.) sym$K half-kills "sym"

2.) sym$$K completely kills "sym"

3.) $$K kills the entire symbol table

D. altmode altmode control-C '$$|C'

1.) DDT half-kills the last symbol it typed

2.) If another full symbol exists with the same value, DDT types

that symbol.

E. control-Y '|Y'

1.) Takes one prefix argument

2.) DDT treats the argument as an AOBJN pointer to the current job's core and augments the current job's symbol table with the core pointed to. The core being used should itself be a symbol table in the format described above.

3.) Example: Suppose location 100 contains the SQUOZE value for the symbol DOG, and location 101 contains the value 3. Typing -2,,100|Y causes the symbol DOG with and the value 3 to be added to DDT's symbol table for the current job.

F. altmode control-Y '$|Y'

1.) takes prefix argument like control-Y

2.) This is like control-Y except the entire symbol table is replaced with the one specified in user core rather than just being augmented with it.

G. altmode altmode control-Y '$$|Y'

1.) takes one prefix argument which is a location address in user core

2.) Specified location should contain an AOBJN pointer pointing to some sort of symbol table already existing in the current job's core

3.) DDT augments this table with its own symbol table, and modifies the specified pointer in the user's core.

4.) Example: Suppose DDT's symbol table for the current job contain's three symbols, ONE, TWO, and THREE with values 1, 2, and 3 respectively. Assume location 6 contain's the pointer 0,,100. By typing 6$$|Y, you will cause DDT to put the SQUOZE values for the symbols ONE, TWO, and THREE in locations 100, 102, and 104 respectively,

and put the values 1, 2, and 3 in locations 101, 103 and 105 respectively. Finally, location 6 will be modified to contain the pointer -6,,100 designating that the old list of 0 elements starting at location 100 is now a 6 element list starting at the same place.

XIII. Program control with DDT

A. Starting Execution with altmode-G '$G'

1.) '$G' causes DDT to start the current job at the address specified by the contents of a location named '$9B' (altmode-9-B).

2.) loc$G causes the job to be started at "loc"

3.) loc$$G is the same as loc$G except that "loc" is put in $9B, making "loc" the standard starting address.

B. Breakpoints

1.) DDT may be made to interrupt programs at arbitrary locations, at which point the user may conveniently resume execution, after using any DDT commands for debugging his program.

2.) '$$B' removes all breakpoints from the current job

3.) 'n$B' causes a breakpoint to be put at the location specified by the right half of"n". If the left half of "n" is non-zero, then whenever the breakpoint specified by the right half of "n" is hit, DDT types out the contents of the register specified by the left half of "n". For instance, typing 200(3)$B causes location 200 to be a breakpoint. When 200 is hit, DDT will break in the normal way, and type out the contents of register 3.

4.) When DDT breaks at location "n", it types out '$mB>>n ' where "m" designates which breakpoint has been hit, and "n" is the PC. Only eight breakpoints may be set. Typing n$mB causes the "mth" breakpoint to be set at location "n". To remove the "mth" breakpoint, type 0$mB. Location 0 may not be a breakpoint.

5.) To continue execution after a breakpoint (or many other types of interrupts), type $P (altmode-P). n$P will continue "n"-1 times

through that breakpoint before breaking. $$P will cause DDT to automatically procede every time after that particular breakpoint. Typing any character will stop it.

6.) For every breakpoint, there are four locations within DDT that are available to the user, that are useful. They are named $nB (altmode-n-B) through $nB+3, where "n" is the breakpoint number.

(a.) $nB holds in the right half, the location this breakpoint corresponds to. The left holds the location that should be automatically examined upon breaking.
(b.) $nB+1 may hold an instruction typed in by the user. This allows for conditional breakpoints. If this instruction is non-zero, then when the user types m$P, DDT will execute the instruction every time it hits the breakpoint. If the instruction causes a skip before "m"-1 times through, DDT will break; otherwise, it will break after "m"-1 passes by the breakpoint.
(c.) $nB+2 holds the counter "m", initialized by the user typing m$P.
(d.) $nB+3 holds the instruction that is in the user's program at the breakpoint.

C. The MAR interrupt (memory address register)

1.) The computer may be set to interrupt automatically upon reading, writing, or fetching an instruction, from any particular location.

2.) Typing A$nI causes a MAR interrupt when the condition specified by "n" is imposed on location "A". Useful values of "n" are:

(a.) 0: never break
(b.) 1: break on an instruction fetch from location "A"
(c.) 2: break before writing into location "A"
(d.) 3: break on any reference to location "A"

3.) If "n" is ommitted, 3 is assumed unless "A" is also ommitted in which case the MAR interrupt is removed if it existed.

4.) When the MAR interrupt happens, DDT types 'MAR.loc>>inst ' where "loc" is the PC and "inst" is the instruction that caused the interrupt.

5.) The interrupt may be dismissed by $P or IP.

D. control-P '|P'

1.) This and $P both continue a program after an interrupt but |P doesn't give the TTY back to the program.

2.) This allows for starting more than one program at once.

3.) If DDT receives an error interrupt from an inferior, it types a bell to signal you that a job needs attention.

E. control-X '|X'

1.) This command interrupts the current job if it was proceeded with '|P'.

2.) DDT types out the PC and its contents

F. single instruction execution, altmode-X '$X'

1.) Typing inst$X causes DDT to execute the expression "inst"

2.) DDT types two carriage-return linefeed pairs if the instruction causes a skip, and one pair if it doesn't

3.) The last instruction $Xed is left in location 34 of the current job. The symbol $X (dollarsign-X) has the value 34. DDT also uses locations 35 and 36 for special purposes, so it is not a good idea to use any of these 3 locations for program memory.

G. The one-proceed feature, control-N '|N' and altmode control-N '$|N'

1.) control-N causes the current job to procede but with the one-proceed interrupt enabled causing only one instruction to be executed.

2.) When the program is interrupted, DDT types 'loc>>inst' where "inst" is the next instruction to be executed and "loc" is the PC

3.) "p"|N causes DDT to let the program execute "p" instructions before stopping it and printing the PC and its contents.

4.) altmode control-N '$|N' is the same as '|N', but the teletype is not given back to the job.

H. The .VALUE instruction

1.) If the current job executes a .VALUE instruction, it stops and interrupts DDT.

2.) If the effective address of the .VALUE is 0, DDT types 'nnn>>.VALUE 0 ' where 'nnn' is the location of the .VALUE.

3.) If the effective address of the .VALUE is non-zero, and the contents of '$9B+1' (altmode-9-B plus 1) is negative, DDT interprets the contents of the effective address of the .VALUE as an ASCII command string. DDT reads and executes characters in the effective address and consecutive locations until a zero character '|@' or a control-C is read.

4.) Example: Suppose you wanted a program to kill itself: Put the instruction .VALUE 200 in location 100. Put the ASCII string ':KILL' in location 200 and the ASCII code for "space" left-justified in location 201. IF DDT executes location 100, the .VALUE will cause DDT to execute the command :KILL.

5.) The contents of '$9B+1' is initially -1.

XIV. Job commands that have no "monitor" command equivalent

A. altmode altmode J '$$J'

1.) may take one SIXBIT argument

2.) name$$J causes DDT to try to rename the current job as "name"

3.) $$J with no argument causes DDT to type information about the current job:

(a.) name
(b.) index
(c.) status (running, interrupted, just loaded etc.)

4.) Information printed by '$$J' with no argument i the entry for the current job in the list that '$$V' would print.

B. altmode altmode L '$$L'

1.) similar to '$L'

2.) core is not zeroed prior to loading

3.) I/O channels are not closed either

4.) new symbols are appended to the already existing symbol table.

5.) allows for loading one binary file "on top of" another

C. altmode Y '$Y'

1.) takes arguments like '$L' except the device name may not precede the '$Y', but two expressions may be typed as prefix arguments

2.) causes the current job's core image to be saved on the specified file with symbols.

3.) With arguments preceding it:

(a.) one argument causes only core from that location up to be saved

(b.) With two arguments core between ARG1 and ARG2 is saved inclusive

4.) examples of $Y:

(a.) MAIN$,MAIN+30$Y DUMP BIN UT1: will cause 31 locations to be saved on microtape

(b.) $Y DSK: SAVE THIS will save the entire core image on the disk

5.) If a file with the specified name exists prior to the time the command is given, it is deleted before the dumping is done. Thus, if FOO 1 and FOO 3 both exist, typing :DUMP FOO > will cause a dump on FOO 2, and FOO 3 will be gone.

D. altmode-altmode-y '$$Y' is the same as '$Y' except that the file being dumped on is not deleted prior to dumping. Hence if the above example were executed with '$$Y', the image would be saved on FOO 4, and nothing would be deleted.

E. control-S with a prefix argument

1.) '|S' preceded by a name sets the current job's "sname" to that name

2.) altmode control-S '$|S' preceded by a name does the same thing as '|S' but also sets DDT's "sname" to that name. By typing FOO$|S followed by DSK|F, you will causes DDT to type FOO's disk directory.

3.) altmode altmode control-S '$$|S' does the same thing as '$|S' but also sets the master SNAME to the given argument. This makes so you can constantly be referencing another user's directory as though it were your own. Every time a new job is created, DDT sets the SNAME of that job to the master SNAME.

F. altmode altmode control-X point '$$|X.'

1.) causes all inferiors to the running DDT to be killed

2.) equivalent to repetitive issuances of $|X. (which kills the

current job and does an '$J' to get another job as the current one)

3.) The dot "." is required to lessen the likelihood of you accidently issuing this command.

G. control-T and altmode control-T '|T' and '$|T'

1.) used to make an entry in the translation table

2.) $|T makes a universal entry whereas |T just makes an entry in the current job.

3.) takes two :LOAD type argument lists as suffix arguments

(a.) first specified file is entered as a translation to second
(b.) An asterisk '*' as part of the first argument list matches any file or device name.
(c.) An asterisk as part of the second argument list stands for the same corresponding name in the first list.
(d.) Only device and file names should be given. System names are never translated.

4.) May be preceded by:

(a.) 'A' meaning atomic entry, no more levels of translation will be done.
(b.) 'I' meaning input only, only input files will be translated
(c.) 'O' meaning output only, only output files will be translated

5.) Ommitting the prefix letters implies 'IO'

6.) Example of an entry in the translation table: $|T SYS ATSIGN DSK DDT BIN.

K. control-U and altmode control-U

1.) take arguments like control-T

2.) used to remove entries from the translation table

I. control-K '|K'

1.) NAME|K is equivalent to :NAME, assuming NAME is not a "monitor" command

2.) NAME|K causes the system program NAME to be loaded into a job called NAME, and execution to be started

3.) More accurately, NAME|K is equivalent to typing the following:

(a.) NAME$J name a new job (DDT types '!')
(b.) SYS$L TS NAME load the program
(c.) $$K kill the symbols
(d.) $G begin execution

4.) DDT types '!' to signify creation of a new job. If '!' is not typed by DDT, it means that a job named NAME already existed, and DDT reloads the program into that job and makes it the current job.

5.) Examples:

(a.) :TECO followed by a space
(b.) MIDAS|K

6.) SYS|K, HACTRN|K, and PDP6|K are synonomous to these names followd by '$J' (eg. SYS$J)

J. altmode control-K '$|K'

1.) If preceded by a name, does the same thing as '|K' except that symbols are loaded also

2.) If the argument is ommitted, it causes symbols to be loaded into the current job from the last file that was |Ked.

K. altmode U point '$U.'

1.) takes no arguments

2.) causes all procedures to be flushed and a new top-level DDT to be loaded and started

3.) you are still logged in.

4.) The period "." terminating the command is required to lessen the possibility of you issuing this command accidently.

L. control-G

1.) causes DDT to quit whatever it is doing and go back to listening for commands

2.) Only should be used in an emergency (such as an infinite loop) because crucial variables may be left uninitialized

3.) Not to be used to stop type-out. control-S should be used for that purpose.

4.) If typeout was turned off, it is turned on again.

XV. DDT output control commands

A. control-B '|B'

1.) turns on DDT lineprinter output

2.) synonomous to :WALLP LPT:

B. control-C '|C' causes DDT to type a carriage return and linefeed

C. control-E '|E' closes any file that :WALLP or |B opened

D. control-L '|L' clears terminal screen

E. control-S '|S' with no argument

1.) empties the DDT typeout buffer

2.) This is the key to hit when DDT is longwinded

F. control-V '|V'

1.) turns typeout on if it was off

2.) This is the initial state

G. control-W '|W'

1.) turns off DDT typeout, but not :WALLP output

2.) error messages will still be typed, however

I. space ' '

If DDT knows it is outputting to a display terminal, it will pause if there is more to type at one time than will fit on the screen, and type '--MORE--'. At this point You may type a space and DDT will clear

the screen and continue the output. If you type any character other than a space after '--MORE--', DDT will type 'FLUSHED' and no more output will be given.

XVI. Typing Error Correcting

A. control-D '|D'

1.) Optionally preceded by altmodes, control-D causes everything you typed since to the last time DDT did to be deleted

2.) DDT types 'XXX?' to signify deletion.

B. rubout

1.) If a rubout is typed in the middle of a syllable, the last character typed is echoed and deleted.

2.) If characters are rubbed out beyond the beginning of the current syllable, whole syllables are echoed and deleted.

3.) "rubbing out" beyond the first character typed since DDT last did causes DDT to type '?? ' and no action is taken.

XVII. Errors reported by DDT

A. '?U? '

1.) You have typed an undefined symbol

2.) The undefined symbol and the operator immediately after it is ignored, the rest of the input string is remembered, and you may type a defined symbol to take the place of the undefined one.

3.) example: You type 'A+B*C-' and DDT types '?U?' because 'C' is undefined. If you now type 'D-E=', and D and E are defined, DDT sees 'A+B*D-E=' and acts accordingly.

B. 'OP? '

1.) You have typed in an undefined operator

2.) DDT ignores the undefined operator and waits

3.) example: You type '3.0$$-' and DDT types 'OP? ' because '$$-' is not a legal operator. Then you type '-1.4=' and DDT sees '3.0-1.4=' and responds with '1.6'.

C. 'NOS? '

1.) You have typed in two expressions not separated by an operator

2.) Within an entire string, every time DDT sees this situation (looking from left to right), it types 'NOS? ' and replaces the second value with a plus sign

3.) example: You type '<1><2><3><4>=' and DDT responds with 'NOS? NOS? 4'. DDT actually evaluated the string '1+3+='. The '<2>' and '<4>' were replaced with '+' signs.

D. '?' or '??'

1.) something is wrong

2.) DDT ignores what it didn't like

E. 'PUR? '

1.) You tried to modify a location within a pure page of memory

2.) DDT ignores the attempt.

F. 'DSN? ' DDT failed at disowning the current job

G. 'CFT? ' DDT couldn't flush the teletype (give it back to the current job)

H. 'BIN? '

1.) DDT or ITS did not like the format of the file you requested to be loaded.

2.) The most common cause is you telling DDT to load a source file instead of its binary code.

I. 'CKS? ' checksum error during loading

J. 'IOC? '

1.) I/O channel error interrupted DDT

2.) This message is preceded by the error code obtained by .STATUS, the channel number, and the PC

K. 'INT? '

1.) DDT received some random interrupt

2.) message is preceded by interrupt word and the PC

L. 'COR? '

1.) not enough core (probably during a load)

2.) very rare message to get now that swapping exists

M. 'NML? ' "not my loser"; DDT received an unrecognized interrupt from an inferior

N. 'UNF? ' DDT couldn't flap a microtape

O. '??? ' DDT couldn't do a .GETSYS for SYS$J

P. 'TMJ? ' You tried FOO$J after 8 jobs already existed

Q. 'LOGIN? ' You are not logged in and you should be

R. 'JOB? ' There is no current job

S. 'BRF? ' "barf" .GETSYS for initial symbols failed

T. 'NO CORE FOR SYMBOLS'

1.) program has been successfully loaded, but there wasn't enough free core for the symbols

2.) This is a rare message now that we have swapping

U. 'JOB ALREADY EXISTS' You typed FOO$$J and a job already exists with the name FOO.

V. 'ARG? ' You gave ridiculous arguments to a command.

W. 'JOB NOT INFERIOR? ' You tried to do something to the current job that is impossible because the current job is not really an inferior procedure to DDT.

XVIII. Error messages typed because DDT received an interrupt from inferior

A. Each of these error messages is followed by 'PC>>Inst'

B. 'ILOPR.' illegal instruction

C. 'MPV.' memory protection violation

D. 'IOC.' I/O channel error. Probably an .IOT was given before the channel was .OPENed.

E. 'DPY.' Display list memory protection violation

F. 'BADPI.' bad location 42

G. 'MAR.' MAR interrupt

H. 'ILUAD.' memory protection violation on reference to an inferior

XIX. Some more-advanced DDT features

A. DDT Patch feature

1.) Often while debugging some PDP-10 Assembly code, you want to insert some instructions "between" two others. Generally, the way this is done is the following:

(a.) The instruction after which the insert is desired is replaced with a transfer instruction to the beginning of a patch area.
(b.) The replaced instruction is put at the beginning of the patch area.
(c.) Then the insert is put following that first instruction.
(d.) And finally a transfer back to the instruction after where the insert should be.

2.) control-backslash '|¬', :PATCH, and :PAT cause DDT to start a patch. A location must be open when one of these commands is issued or else DDT types '?? ' and the command is ignored. Assume a location named HERE is open and you type |¬. DDT does the following: If a location named PAT or PATCH exists, DDT opens it, types out its contents (probably 0) and simulates you typing into it the contents of HERE. DDT waits at the end of this instruction as though you had just typed it but hadn't closed the register yet. If no symbol PAT or PATCH exists in the symbol for the current job, DDT uses location 50 in the current job's core image, and names it PATCH.

3.) Assuming you have started a patch as described, you now repeatedly type linefeed followed by an instruction until you have completed the desired insert.

4.) Now, by typing control-closed bracket '|' or :ENDPAT, DDT will end the patch in the following way:

(a.) First, the location after the last one in which you put one of the inserted instructions is opened and the instruction 'JUMPA 1,HERE+1' is deposited.
(b.) 'JUMPA 1,HERE+2' is put in the location after the JUMPA 1,HERE+1'. This allows for "skip" returns.

(c.) The contents of HERE is changed to JUMPA 3,PATCH (or PAT)

5.) Example of making a patch:

(a.) You type 'HOP+2/' and DDT types 'PUSHJ P,GETNIT '
(b.) You type '|¬' and DDT types 'PAT/ 0 PUSHJ P,GETNIT'
(c.) Now you type in the patch
(d.) Let's say you entered the last instruction at 'PAT+2' and have just typed a linefeed. DDT types 'PAT+3/ 0 '
(e.) Now you type '|', and DDT types the following:

```
'PAT+3/ 0 JUMPA 1,HOP+3
PAT+4/ 0 JUMPA 1,HOP+4
HOP+2/ PUSHJ P,GETNIT JUMPA 3,PAT
'
```

and the patch is done.
(f.) DDT remembers the last location used as a patch and defines or redefines the symbol PATCH to the location after this one. So sometime after this example, if you again typed :PATCH, DDT would start the new patch at 'PAT+5'.

6.) altmode control-close bracket '$||' finishes a patch like '||' except that immediately preceding the two JUMPAs, DDT puts the contents of the location you patched from.

7.) control-uparrow '||' or :UNPAT are used to remove a patch. Specifically what happens is DDT replaces the contents of the currently open location with the contents of the location addressed by the right half of the currently open location. For insatnce, suppose location BUG contains JRST PAT, and PAT contains BLT 1,3644. If location BUG is open and you type '||', DDT types BLT 1,3644, and that becomes the new contents of BUG.

B. Multiple-word ASCII and SIXBIT input

1.) '$$n"' and '$$n'' (two altmodes followed by a number, followed by a double or single quote) cause DDT to enter ASCII and SIXBIT input respectively.

2.) ASCII input ends with an altmode, SIXBIT with any non-SIXBIT character.

3.) These two modes are the same as the related commands with only

a single altmode except that when you type more characters than can fit in a word (5 or 6 depending on the mode) before termination, DDT does the following:

(a.) types a '$'
(b.) closes the currently opened register, depositing the ASCII or SIXBIT for the characters typed so far
(c.) opens the next register, and types '$$1' followed by the appropriate quote followed by the last character you typed on the line before, which is really being deposited as the first character of this new line.
(d.) waits for more characters or/and terminator character

4.) In ASCII mode, to enter control characters that might normally cause undesireable action by DDT, type an uparrow '|' followed by the character to be "controlled".

5.) To enter an altmode or uparrow in ASCII input mode, precede it with a control-Q (NOT uparrow followed by a Q)

6.) example of SIXBIT input:

Suppose you have opened location 200 and you type '$$1'THIS IS AN EXAMPLE$'. What you would see on your console would be the following:

```
$$1'THIS IS$
201/ 0 $$1'S AN EX$
202/ 0 $$1'XAMPLE.$
203/ 0 $$1'.$
```

XX. Special DDT locations

Many of the system variables associated with the current job may be examined and modified with DDT. To examine or modify a variable, all you have to do is open a special location with DDT and look at, and maybe modify, its contents. The following is a list of special locations and a description of the significance of their contents. Each location name begins with a period '.' , and the period will be assumed prefixing each symbol in the list.

A. UPC contains the program counter (PC) for the current job.

B. VAL holds the effective adddress of the last .VALUE instruction executed by the current job.

C. TTY holds the .STATUS bits for the console TTY attached to you.

D. FLS holds the "flush instruction" for the current job. If this instruction is 0 or causes a skip when executed and the USTP for the job is 0, the procedure is runnable.

E. UNAME holds the name you logged in as stored in SIXBIT.

F. JNAME holds the name of the current job in SIXBIT.

G. MASK holds the first of two interrupt words for the current job.

H. USTP contains the current job's stop word. A zero in this location implies the job is running or hung on its flush instruction.

I. PIRQC holds the first interrupt request word for the current job.

J. INTB holds the bits used to interrupt this procedure's immediate superior, probably HACTRN.

K. MEMT holds the lowest address that is "out of bounds" for this job.

L. SV40 holds the last system UUO executed.

M. IPIRQ holds the same as PIRQC, but on deposit the information is IORed into it.

N. APIRQ is the same as IPIRQ but with the bits ANCAMed in.

O. SNAM holds the current system name.

P. PICLR holds -1 if interrupts are enabled.

Q. MARA holds the control bits for the MAR interrupt.

R. MARPC contains the PC at the last MAR interrupt.

S. UUOH holds the address one after the one where the last UUO trapping to the system was executed.

T. UIND holds the index assigned to this job by the system.

U. RUNT holds the runtime for this job in 4.069 microsecond units.

V. MSK2 holds the second interrupt word.

W. IFPIR contains the second interrupt request word.

X. APRC holds the disowned job and processor CONO status.

Y. SV60 contains the saved absolute location 60.

Z. IIFPI and AIFPI are the second interrupt request word and on deposit are IORed and ANDCAMed into, repectively.

AA. IMASK and AMASK are the interrupt first mask word and on deposit are IORed and ANDCAMed into, respectively.

BB. IMSK2 and AMSK2 are the second interrupt mask and on deposit are IORed and ANDCAMed into, respectively.

CC. JPC holds the PC at the last jump type of instruction.

DD. OPC contains the old PC stored after certain interrupts.

EE. RTMR holds the runtime interrupt, or -1 if the interrupt is not enabled.

FF. 60H holds the last instruction executed in the current job that trapped to location 60.

GG. IOC+n holds the I/O channel word and on deposit for channel 'n'.

HH. IOS+n holds the .STATUS bits for I/O channel 'n'.

11. IOP+n holds the contents of the IOPDL which has four pairs of alternating channel and status words.

XXI. Cross Reference Index of DDT Format Commands

A. This index lists every DDT format command, a brief description of its purpose, and where in this manual a detailed explanation of the command may be found. An uparrow preceding a command character means that letter is a control letter, ie '|A' means the character whose ASCII value is 001. When no description or cross reference follows a character, the character is not a DDT recognized command.

B. Ccommand characters taking no preceding altmodes:

|@

|A

|B :WALLP to lineprinter. XV.A.

|C causes DDT to type a carriagereturn and linefeed. XV.B.

|D causes type-in deletion. XVI.A.

|E closes :WALLP output. XV.C.

|F gives file directory listings. III.J.

|G emergency full-quit, only in matters of life or death. XIV.L.

|H

|I closes the currently open location and opens address location. XI.E.13.

|J or linefeed closes the current location and opens the next one. XI.E.9

|K NAME|K loads the system program NAME and starts it. XIV.I.

|L clears screens. XV.D.

|M or carriagereturn closes the currently open location. XI.E.7

|N one instruction proceed. XIII.G.

|O deletes files. III.L.

|P continue execution, but leave TTY with DDT. XIII.D.

|Q same as '|' (uparrow)

|R

|S stops typeout. XV.E., FOO|S sets the SNAME of the current job to FOO XIV.E.

|T enters information in the current job's translation table. XIV.G.

|U removes information from the current job's translation table. XIV.K.

|V turns on DDT typeout. XV.F.

|W turns off DDT typeout. XV.G.

|X interrupts the current job, if it was |Ped. XIII.E.

|Y. used to augment DDT's symbol table with one from the current job's core image. XII.E.

|Z Has no affect on DDT, but interrupts an inferior that has the teletype, and returns the teletype to DDT.

$ (altmode) used for many DDT command format commands.

|¬ starts a patch. XIX.A.2.

|| ends a patch. XIX.A.4.

|| unpatches. XIX.A.7.

|_

<space> field separater. VI., continues output after DDT types '--MORE--'. XV.I.

! integer divide. VII.D.

" causes ASCII typeout. X.B.5.

exclusive Or. VII.F.

$ (dollar sign) symbol constituent. V.B.

% symbol constituent. V.B.

& logical AND. VII.G.

' causes SIXBIT typeout. X.B.4.

(and) swaps left and right half of infix argument. VII.H.

* integer multiply. VII.C.

+ integer add. VII.A.

, field separater. VI.

- integer subtraction. VII.B.

. used in numbers. V.A.1.B., symbol constituent. V.B., current location value. XI.C.1.

/ causes DDT to open location address specified by right 18 bits of last word typed. XI.E.1.

<digits> used to form numbers. V.A.

: used to define symbols. XII.B.

; causes "semi-colon" mode typeout, initially floating point. X.B.6.

< and > mathematical grouping symbols. VIII.B.

= causes numerical type-out. X.B.1.

?
indirect bit, IORed into expression. V.B.2.d.

<letters> symbol constituent. V.B.
like '/', but types out numerically. XI.E.3.

| like ||, but doesn't change current location. XI.E.16.
like '/', but types out symbolically. XI.E.5.

| causes current location to be closed and previous one to be opened. XI.E.11.

_ causes symbolic typout. X.B.0.

<rubout> for character or syllable typein deletion. XVI.B.

C. Command characters taking one preceding altmode.

$|

$|a

$|b

$|C

$|D typein deletion. XVI.A.

$|E

$|F for file links. III.M.

$|G

$|H

$|I opens address pointed to by left 18 bits of last expression DDT typed. XI.E.14.

$|J or altmode linefeed closes open register and opens the one after $|M would have opened. XI.E.10.

$|K load system program and/or its symbols. XIV.J.

$|L

$|M or altmode carriagereturn closes currently open register and opens the last one in the . ring buffer. XI.E.8.

$|N one proceed, but teletype is not given to job. XIII.G.4.

$|O

$|P

$|Q

$|R

$|S sets DDT's "sname". XIV.E.2.

$|T enter information in DDT's translation table. XIV.G.2.

$|U remove information from DDT's translation table. XIV.K.

$|V

$|W

$|X kill the current job, requires a dot ".". III.I.

$|Y replaces DDT's symbol table with one from the current job's core. XII.F.

$|Z The |Z will not be seen by DDT.

$$ (altmode altmode) part of many commands.

$|¬

$|| end a patch. XIX.A.6.

$||

$|_

$<space> argument separater. IX.D.

$| floating divide. VII.E.

$" changes current typeout mode to ASCII.X.C.3.i., with an infix argument allows ASCII string input. V.D.1.

$#

$$ (altmode dollar-sign)

$%

$& changes current typeout mode to SQUOZE. X.C.3.k., with an infix argument, allows SQUOZE input. V.D.3.

$' changes current typeout mode to SIXBIT. X.C.3.j., with infix argument, allows SIXBIT string input. V.D.3.

$(and $)

$* floating multiply. VII.E.

$+ floating add. VII.E.

$, argument separater. IX.D.

$- floating subtract. VII.E.

$. PC of current job. XI.C.5.

$/ like '/' but uses left 18 bits for address to be opened. XI.E.2.

$<digits> The digits will be part of a DDT format command infix argument.

$: for naming relocatable prorams. XII.B.3.

$; like ;, but changes typeout mode to "semi-colon" mode. X.B.7.

$< for special infix grouping. IX.B.

$= causes floating typeout. X.B.2.

n$> has value of $Q with non-zero fields of n replacing fiels in $Q. XI.C.4.

$?

$@

$A changes current typeout mode to absolute addressing. X.C.3.b.

$B breakpoint type commands. XIII.B.

$C changes current typeout mode to constants. X.C.3.e.

$D sets current radix to ten. X.C.4.a.

$E effective address search. XI.H.

$F changes current typeout mode to floating point. X.C.3.1.

$G start job execution. XIII.A.

$H change current typeout mode to halfword. X.C.3.f.

$I MAR interrupt controlling. XIII.C.2.

$J job selection. III.D.

$K half kills symbols. XII.C.1.

$L loading binary files. III.F.

$M holds the search mask for $W. XI.G.2.

$N not-word search. XI.F.

$O changes current radix to octal. X.C.4.b.

$P continues job execution. III.E.

$Q has value of last thing DDT typed. XI.C.2., with infix argument, has value of nth last thing typed. XI.C.3.

$R changes current typeout mode to relative addressing. X.C.3.c., with infix argument changes current radix to 'n'. X.C.3.d.

$S changes current typeout mode to symblic. X.C.3.a.

$nT changes current typeout mode to bytes of size 'n'. X.C.3.g.

$U for logging in. III.B., or for reloading DDT, requires dot ".". XIV.k.

$V

$W word search. XI.G.

$X single instruction execution. XIII.F.

$Y dump core. XIV.C.

$Z

$| like '|', but uses left 18 bits for address. XI.E.4.

$¬ like $||, but DDT does not change "point". XI.E.17.

$| like '|' but uses left 18 bits for adress. XI.E.6.

$| closes open register and opens one before $|M would open. XI.E.12.

$_ left shift. VII.I.

$<rubout> The rubout will delete the altmode.

D. Commands taking two altmodes

$$|@

$$|A

$$|B

$$|C

$$|D typein deletion. XVI.A.

$$|E

$$|F flaps microtape. III.N.

$$|G

$$|H

$$|I Does effective address calculation on $Q and opens that register. XI.E.15.

$$|J

$$|K disowns jobs. III.T.

$$|L

$$|M

$$|N

$$|O

$$|P

$$|Q

$$|R

$$|S sets master "sname". XIV.E.3.

$$|T

$$|U

$$|V

$$|W

$$|X kills all jobs, requires a ".". XIV.F.

$$|Y augments a symbol table in the current job's core image with DDT's. XII.G.

$$|Z The |Z will not be seen by DDT.

$$$ (altmode altmode altmode) The same as two altmodes.

$$|¬

$$||

$$||

$$|_

$$<space>

$$! opens register, but suppresses typeout of contents. XI.E.18.

$$" sets permenant typeout mode to ASCII. X.C.3.i., with an infix argument sets permanent ASCII typein. XIX.B.

$$#

$$$

$$%

$$& sets permanent typeout to SQUUZE. X.C.3.k.

$$' sets permanent typeout to SIXBIT. X.C.3.j., with an infix argument sets permanent SIXBIT typein. XIX.B.

$$(and $$)

$$*

$$+

$$,

$$-

$$.

$$/

$$<digits> The digits will be part of an infix argument.

$$:

$$; same as $;, but permanent mode is changed to "semi-colon" mode. X.B.8.

$$< special infix grouping. IX.B.

$$= types out up to three given arguments in the mode they were typed in. X.B.3.

$$>

$$?

$$@

$$A changes permanent typeout mode to absolute addressing. X.C.3.b.

$$B removes all breakpoints. XIII.B.2.

$$C changes permanent typeout mode to constants. X.C.3.e.

$$D sets permanent radix to ten. X.C.4.a.

$$E

$$F sets permanent typeout mode to floating point. X.C.3.h.

$$G starts execution and sets default starting address. XIII.A.3.

$$H sets permanent typeout mode to half-words. X.C.3.f.

$$I

$$J renames jobs and tells job status. XIV.A.

$$K kills symbols. XII.C.

$$L loads file on top of existing core. XIV.B.

$$M

$$N

$$O sets permanent radix to octal. X.C.4.b.

$$P indefinite continue after breakpoint. XIII.B.5.

$$Q takes infix argument and replaces $Q with $nQ. XI.C.3.

$$R changes permanent typeout mode to relative addressing. X.C.3.c., with infix argument sets permanent radix to argument. X.C.3.d.

$$S changes permanent typeout mode to symbolic. X.C.3.a.

$$T takes infix argument and sets permanent typeout mode to bytes of size 'n'. X.C.3.g.

$$U logs you out. III.C.

$$V list jobs. III.E.

$$W

$$X

$$Y dumps core onto specified file, but doesn't delete existing file first. XIV.D.

$$Z zeros core. XI.A.

$$|

$$¬

$$|

$$|

$$_ floating scale. VII.J.

$$<rubout> The rubout will delete the second altmode.

XXII. Index of Monitor Commands

This index lists most monitor command, a short description of the command, and a cross reference to the detailed description within the manual.

:BUG document bug. III.V.

:CONTIN continue giving job tty. IV.E.

:DELETE delete file. III.L.

:DISOWN disown current job. III.T.

:DUMP dump from job. III.H.

:ENDPAT end patch. XIX.A.4.

:ERR ioc error status. III.Y.

:FLAP flap dectape. III.N.

:GAG control receipt of messages. III.S.

:GO start inferior. IV.C.

:GZP alt g, ctl z, ctl p. IV.D.

:JOB create or select job. III.D.

:KILL kill current job. III.I.

:LINK create link. III.M.

:LISTF list files. III.J.

:LISTJ list jobs. III.E.

:LISTP list relocatable programs. III.P.

:LOAD load from following file. III.F.

:LOGIN login as following name. III.B.

:LOGOUT auto-expunge. III.C.

:MAIL add to user's mail file. III.Q.

:PAT the same as :PATCH. XIX.A.2.

:PATCH patch. XIX.A.2.

:PDUMP pure dump. III.H.

:PRGM print current program name. III.O.

:PRINT print file. III.K.

:PROCED procede job, leave tty with ddt. IV.F.

:SEND send message. III.R.

:SL the same as :SYMLOD. III.G.

:SLEEP sleep n 30'ths of a second. III.W.

:SLIST list symbols. IV.H.

:SSTATU type system status. III.X.

:START start inferior. IV.C.

:SYMLOD load, symbols only. III.G.

:UNPAT unpatch. XIX.A.7.

:V turn typeout on. IV.A.

:VERSIO type version #'s. III.Z.

:VK do a :V and type a carriage return and linefeed. IV.B.

:VP do a :V and then a :CONTIN. IV.G.

:WALLP |B to specified file. III.AA.

:XFILE execute file as ddt commands. III.V.

:? list most : commands. III.A.

:?? list all : commands. III.A.

:$ (colon-altmode) used to enter comments. III.BB.

XXIII. Sample Console Session

```
ITS.698. DDT.316.
# USERS = 7.
WORLD:  PLEASE TYPE COM↑F AND DELETE ANYFILES
YOU PUT THERE AND KNOW NEEDN'T BE SAVED.   --HUR
UPDATED .INFO.; LISPIO NOTE -- RMS(9/5/71).
:PRINT DSK:.INFO.;LISPIO NOTE

CHANGES TO LISP I-O:

CONTENTS:
        FILES, AND HOW TO OPEN THEM
        SELECTION FORINPUT
        ENABLING FOR↑S

↑BDEVICE NOT AVAILABLE
 10163, LPT?   :JOB DEMO1LOGIN?   DEMO$U

HUR09/10/71 15:50:28 SAMPLE MAIL FOR DEMONSTRATION.

:TECO !
TECO.206
E←TPL:$DSK:.INFO.;LISPIO NOTE$$
TPLDEMO:
FILE NOT FOUND

E←DSK:.INFO.;LISPIO NOTE$TPL:$$
↑Z

2230)   .IOT 1,16   DEMOJ$J!
:LOAD  TICTAC BIN
NON-EXISTENT USERNAME
 10546, DSK?   :LOAD  HUR;

$G

NEED HELP?N
YOUR MOVE?  ↑Z

GETMOV+1)   .IOT 0   $$J   DEMOJ P 27
$$V
  TECO P 30
* DEMOJ P 27

:SEND PRINT STAVROS, WHAT DAY ARE YOU
QUITTING WORK TO GO BACK TO
SCHOOL.
```

```
↑C


MESSAGE FROM PRINT HACTRN
15:54:53 TODAY

$J TECO$J
$↑X. DEMOJ$J
B=?U?B←?U?69.$,B:    $Q=40000000000
B=105    $C$D;69.
B=105    B$$K    B=?U?
1↑F
MFT
FREE FILES 13    FREE BLOCKS 160
  PRIME   NUMBER2
  LATEST MANUAL 152
  CHECKR 1       2
  CHECKR2       7
  DDTMEM 32      99
  ERIC    WINNER 3
  TICT@C BIN     12
  TICTAC >       23
  /L             19
  REEF    TECO    80

:FLAP 1
:LISTJ
* DEMOJ P 27

PRICON$,OOBY$,0$N
PRICON+1/    SETOM CMOVE
PRICON+2/    SETZB FOO,FOO2
PRICON+3/    HRRZ PT,LENGTH
PRICON+4/    MOVNS PT
PRICON+5/    HRLZS PT
HOP/    MOVE ROW,MLIST(PT)
HOP+1/    SKIPE BOARD(ROW)
HOP+2/    JRST ISZC
HOP+3/    MOVEI PLANE,PLANE
HOP+4/    IMULI PLANE,0(ROW)
HOP+5/    MOVE PRR,3255
HOP+6/    LDB L1PR,PRR
HOP+7/    MOVNSL1PR
HOP+10/    HRLZS L1PR
OOBY/    SETZM FOO

PRICON$B    :CONTIN
111.    MY MOVE:  I MOVE TO 332.

YOURMOVE?  ↑Z

GETMOV+1)    .IOT 0    $1B/    PRICON    $Q+1
$P
222.    MYMOVE:
$1B>>PRICON+1    ↑N
```

```
PRICON+2>>SETZB F00,F002    ↑N

PRICON+3>>HRRZPT,LENGTH    LENGTH$2I
$P
 I MOVE TO 333.

YOUR MOVE? 334.    MY MOVE:
MAR.ONWARD+4>>MOVEM A,LENGTH    $I
:CONTIN

$IB>>PRICON+1    ONWARD/    PRYNT 3247
ONWARD+1/    MOVSI A,-100
ONWARD+2/    MOVEM A,MLIST(A)    ↑\

PAT/    0    MOVEM A,MLIST(A)
PAT+1/    0    AOS F002↑]
PAT+2/    0    JUMPA A,ONWARD+3
PAT+3/    0    JUMPA A,ONWARD+4
ONWARD+2/    MOVEM A,MLIST(A)    JUMPA C,PAT
$Y DSK:HUR;PATED 3DTTT

$J DEMOJ$J
$J DEMOJ$J
:JOB ROR!
:LOAD  ROR BIN
FILE NOT FOUND
 10546, DSK?    :LOAD  HUR;
FILE NOTFOUND
 10546, DSK?    MIDAS↑K!
MIDAS.75
HUR;ROR BIN←HUR;ROR >

THE FAMOUS ROR PROBLEM!!!
THE FAMOUS ROR PROBLEM!!!
CONSTANTS AREA INCLUSIVE
FROM    TO
20607    20644


:KILL  ROR$J
$L

$G
HOW MANY ELEMENTS? 5

↑Z

NOROT+6)    SETZM RECORD    ↑P

$$V
* ROR R 4
  DEMOJ 1B 27
```

```
$J DEMOJ$J
$↑X. ROR$J
$$J   ROR R4
↑X

BACKUP+13)   JRST NOROT   ;PROCED

$$↑X
;LISTJ

ROR$J!
↑X

172)   JRST 161   $↑X.
;SSTATU
# USERS = 3.
;VERSIO
ITS.698. DDT.316.
J$J!
100/   0   .OPEN 200
200/   0   $1'  !TTY$$
101/   0   .VALUE
102/   0   HRRI 1,300
300/   0   $$1"THIS I$
301/   0   $$1"IS A D$
302/   0   $$1"DEMONS$
303/   0   $$1"STRATI$
304/   0   $$1"ION PR$
305/   0   $$1"ROGRAM$
306/   0   $$1"M.$
$
103/   0   HRLI 1,440700
104/   0   ILDB 2,1
105/   0   .IOT ↑DXXX?
JUMPE 2,400
400/   0   .VALUE 310
310/   0   $1":GO 1$
311/   0   $1"00$$
401/   0   $
106/   0   ↑
105/   0   ↑
104/   ILDB 2,1
105/   0   JUMPE 2,400
106/   0   .IOT 2
107/   0   JRST 103
;START 102
```

```
IOC.106>>.IOT 2    .IOS/   PTP 0    :ERR CHNL NOT OPEN
   0$N
1/   AOS 16,300
2/   124
$X/   JRST 102
35/   .BREAK 16,0
36/   .BREAK 17,0
40/   .IOT 2
100/   .OPEN 200
101/   .VALUE 0
102/   HRRI 1,300
103/   HRLI 1,440700
104/   ILDB 2,1
105/   JUMPE 2,400
106/   .IOT 2
107/   JRST 103
200/   646471(1)
300/   HLLOM 4,151500(11)
301/   EQVM 11,@40500(4)
302/   ANDCMM 2,@547634(11)
303/   HRLZM 12,240650(12)
304/   EQVM 7,@620240(11)
305/   HLLZM 7,@751202(10)
306/   ORCMI 7,0
31↑S
$$"   200$,0$N
200/   $1"↑@↑@↑NM↑\$
300/   $1"THIS $
301/   $1"IS A $
302/   $1"DEMON$
303/   $1"STRAT$
304/   $1"ION P$
305/   $1"ROGRA$
306/   $1"M.↑@↑@↑@$
310/   $1":GO 1$
311/   $1"00↑@↑@↑@$
400/   $1"↑HH↑@↑@D$

200/   $1"↑@↑@↑NM↑\$    '$1'   !TTY$
$$S   ./    646471(1)    :GO 100
TTTT
```

```
103>>HRLI 1,440700   107701 ?? 104/   ILDB 2,1   107/   JRST103
./   JRST 103   $Q+1
./   JRST 104   $P

$1B>>104   1/   ANDCB 16,300   $$B   $P
THIS IS A DEMONSTRATION PROGRAM.

:GO 100 =?  ??
400/   .VALUE 310
310/   AOSE 3,@720142(11)   "$1":GO 1$
311/   CAIL 10,0   "$1"00↑@↑@↑@$   $1"00 $
./   CAIL 10,0(4)   "$1"00 ↑@↑@$
:GO 100
THIS IS A DEMONSTRATION PROGRAM.

:GO 100 $
400/   .VALUE 310   $
1/   FDV 16,306   $
107/   JRST 104
310/   AOSE 3,@720142(11)   "$1":GO 1$
311/   CAIL 10,0(4)   "$1"00 ↑@↑@$   $1"00$
./   CAIL 10,505000(1)   "$1"00↑M↑J↑@$   $1"00$
./   CAIL 10,500000(1)   "$1"00↑M↑@↑@$   :GO 100
THIS IS A DEMONSTRATION PROGRAM.

:GO 100

THIS IS A DEMONSTRATION PROGRAM.

:GO 100

THIS IS A DEMONSTRATION PROGR@M.

:GO 100

THIS IS A DEMONSTRATION PROGRAM.

:GO 100

THIS IS A DEMONSTRATION PROGRAM.

:GO 100



100)   .OPEN 200   400/   .VALUE 310
310/   AOSE 3,@720142(11)   $$1":LOGOU$
311/   CAIL 10,500000(1)   $$1"UT $
100$G
THIS IS A DEMONSTRATION PROGRAM.

:LOGOUT
ITS 698  CONSOLE 0 FREE. 16:30:17
```

MIT Libraries
Document Services

Room 14-0551
77 Massachusetts Avenue
Cambridge, MA 02139
Ph: 617.253.5668 Fax: 617.253.1690
Email: docs@mit.edu
http://libraries.mit.edu/docs

DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Due to the poor quality of the original document, there is some spotting or background shading in this document.