DDTG: FUNCTIONAL SPECIFICATION FOR THE RTPP MONITOR/DEBUGGER

NCI/IP Technical Report #2

February 5, 1976

Peter Lemkin

National Cancer Program

U.S. Department of Health, Education, and Welfare / National Institutes of Health / National Cancer Institute

NCI/IP-76/02

DDTG:
FUNCTIONAL SPECIFICATION FOR THE RTPP MONITOR/DEBUGGER

NCI/IP Technical Report #2

February 5, 1976

Peter Lemkin

Image Processing
Division of Cancer Biology and Diagnosis
National Cancer Institute
National Institutes of Health
Bethesda, Maryland 20014

"We here highly resolve . . ."

T A B L E O F C O N T E N T S

NCI/IP-76/02

Functional Specification for the RTPP monitor - debugger

DDTG

NCI/IP Technical Report #2

Peter Lemkin

Image Processing Unit
National Cancer Institute, DCBD
National Institutes of Health
Bethesda, Md 20014

February 5, 1976

Abstract

DDTG, a monitor/debugger is constructed for user and/or computer control of the Real Time Picture Processor (RTPP). The latter, a multiprocessor image acquisition/analysis system functions under DDTG in either stand alone mode (direct user control) or is driven by one of several complex interpreter/model structures existing on a remote time shared PDP10 computer. In the latter case the overall system, i.e. DDTG running on the RTPP, and driven by PDP10 structures, constitutes the CELMOD system.

SECTION 1

DDTG operations

DDTG, a monitor/debugger is constructed for user and/or computer control of the Real Time Picture Processor (RTPP) ([Carm74], [Lem74], [Lem76b]) (Figure 1). The latter, a multiprocessor image acquisition/analysis system functions under DDTG in either stand alone mode (direct user control) or is driven by one of several complex interpreter/model structures existing on a remote time shared PDP10 computer. In the latter case the overall system, i.e. DDTG running on the RTPP, and driven by PDP10 structures, constitutes the CELMOD system. This is illustrated in a block diagram in Figure 2.

Functionally, DDTG is more than a simple combination of monitor and debugging facilities. As the RTPP operating system, it is required to interpret direct (i.e. via control console) user commands, or a string of commands generated by user-PDP10 interaction. In addition, it is required to provide access and control (at machine language word level) of major memory structures (PDP8e core, General Picture Processor (GPP) program memory [Lem76b], GPP general register memory (GR) [Lem76b] and buffer memories [Lem76b]). It provides full control for a variety of image acquisition and low level analytic peripheral devices (e.g. Axiomat microscope stage, focus, etc. the Quantimet 720 and a variety of its plug in modules, a special mirror scanner, a sonic tablet, and a rapid scan spectrometer). Display control functions of DDTG also include the Dicomed and Quantimet displays.

DDTG has the ability to store, retrieve and execute (from PDP8e disks) PDP8e and/or GPP programs (user or PDP10 generated) (cf. Figure 2). Since in stand alone mode, DDTG is to be used as much by biologists as by computer scientists, the user interface allows many high level and seeming English command constructs. This in turn permits easy construction of understandable control programs for stand alone use, exploration and debugging at a very high level.

DDTG, written in standard PDP8e Fortran-Sabr consists of 4 major parts: an interpreter, parser, symbol table, and a large set of worker routines. The last include such features as loaders for the various component computers of the RTPP, as well as extensive and flexible disk I/O routines, a wide

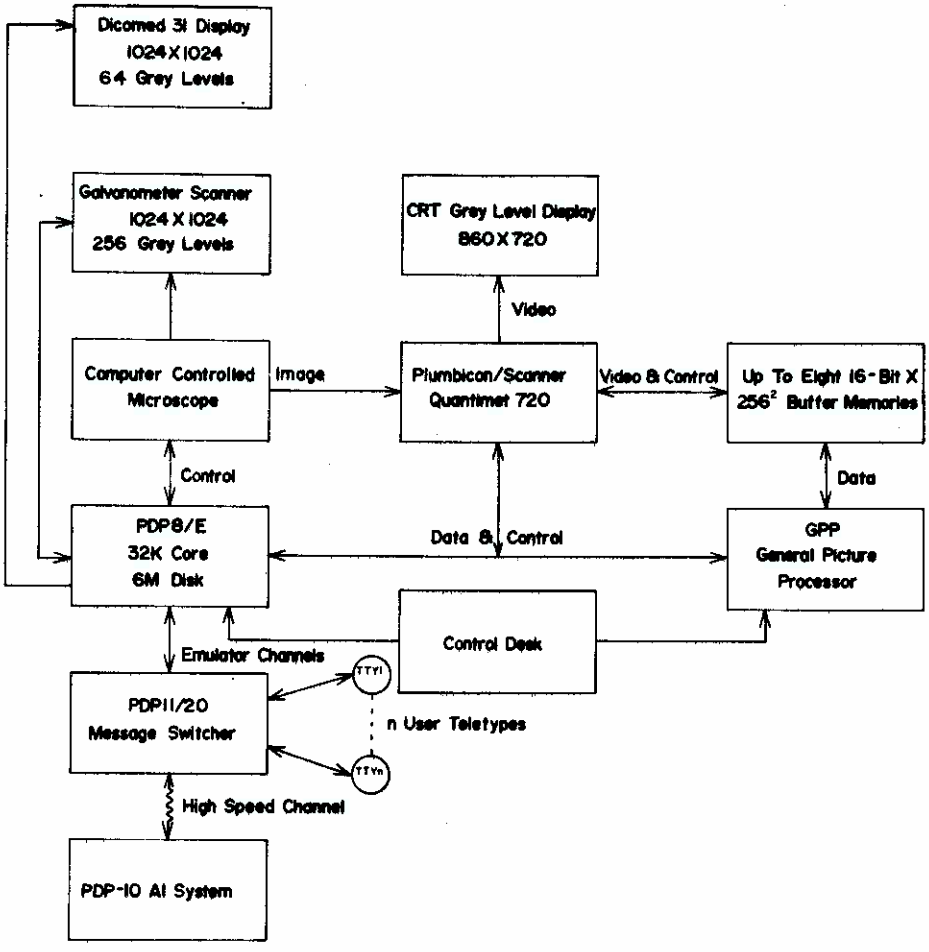
variety of stylized data structures.

DDTG has capabilities which allow it to load and run programs in the RTPP and to monitor their activity. An extensive set of commands are implemented to facilitate image data acquisition and display, and the running of small picture operation programs called "special segments". The General Picture Processor (GPP) portion of the RTPP will, in performing picture operations, require special segment support from DDTG. In addition, mechanisms are available for RTPP program interaction with OS/8 in order to facilitate the implementation of image processing programs (exclusive of DDTG) to process DDTG produced data.

Section 1 describes the operation of DDTG while sections 2 through 3 describe loader and data acquisition file formats respectively. Section 4 discusses the implementation of DDTG. Section 5 discusses the creation of RTPP special segment program functions. Section 6 lists references used in this paper. The Appendices list various tables (DDTG error messages, PDP8e IOT's used in the RTPP, alphabetic list of DDTG commands) in Appendices A, B and C respectively. Appendix D discusses the direct memory access (DMA) channel operation in the RTPP.

To run DDTG, type ".R DDTG" at the OS/8 monitor level. DDTG responds with a "*". Commands are entered at DDTG monitor "*" level, generally followed by a "carriage return". To exit DDTG, type Control/C. This saves the state of DDTG and the so-called Special Segment (cf. 1.1.3). Typing ".R DDTG" will reenter DDTG with the previous state of the system (at the time the Control/C was typed) intact.

At the DDTG monitor ("*") level, a basic clock polling loop is active. This loop checks the control desk stepping motor and threshold control keys and moves the corresponding devices accordingly while also checking for DDTG command to execute.



BLOCK DIAGRAM OF RTPP

Figure 1. Real Time Picture Processor

A block diagram of the real time picture processor is illustrated. The PDP8e computer directs the microscope stage to positions determined either manually by the operator or automatically by the PDP10 system through DDTG. Images may be acquired by the buffer memories for processing by the GPP. Raw images as well as processed images may be displayed on the Quantimet 720 display. Precision scanning and display are implemented by the galvanometer mirror scanner and Dicomed display. The user may interact with the system either through a control desk or a teletype.

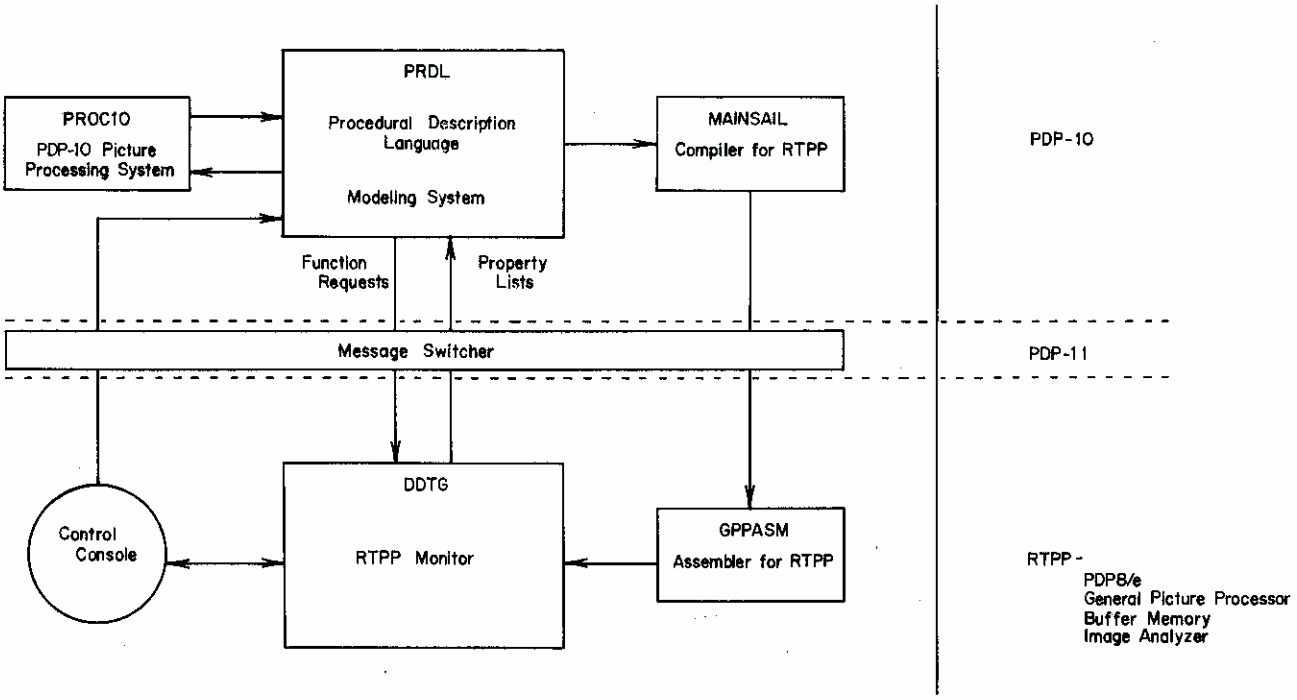


Figure 2. CELMOD system

Software structures are on the left while hardare is to the right. The relationship of DDTG to the other software components of the overall cell modeling analysis system is shown. DDTG resides on the RTPP hardware system. It may recieve direct console input, but in full system operation it communicates with PRDL especially sending property lists of analyzed images and receiving command strings of various functions. It also recieves assembled code from GPPASM loading for loading into the RTPP. The PDP11 and its resident message switches aare essentially user and computer transparent.

1.1 Data spaces

A data space is a contiguous region of memory in a particular hardware processor. This section discusses the accessing and modification of data spaces for the purpose of debugging and monitoring. There are seven data spaces in the RTPP. The program memory PM, the general registers GR, the three triple line buffers I1, I2, I3, the buffer memories BM, and the PDP8e special segment. The data space mode may be set by any of the following:

```
$DSPACE_$PDP8E
$DSPACE_$PM
$DSPACE_$GR
$DSPACE_$I1
$DSPACE_$I2
$DSPACE_$I3
$DSPACE_$BM
```

Typing

```
$DSPACE=
```

($ is dollar sign - not escape character) will give the current data space mode. The PM and GR address 000000:177777 octal (65K). The 8e addresses 00000:77777 (32K) while I1, I2, and I3 address 0:377 (y-1), 400:777 (y) and 1000:1377 (y+1). The BM addresses 0000000:1777777 (8 buffer memories) (8 x 65K).

1.1.1 Opening and modifying data spaces

DDTG may open, modify, and close all GPP memory and registers (including the special registers) which are normally accessible in the GPP address space. These include the PM, GR, special registers, I1, I2, and I3, BM memory, and 4K locations of PDP8e memory (field 7) called the "8e special segment". The concept of "opening", "closing" etc. memory spaces is explained in [DEC73] and [DEC74] under sections on the operation of DDT and ODT (debugging languages for the PDP10 and the PDP8e respectively).

DDT (ODT) is a symbolic (octal) debugger program for debugging assembly language core image programs. ODT and to a lesser extent DDT are "invisible" in that they may reside in the system memory but take up almost no space in and thus

interfere minimally with the program under test. DDTG is designed with similar goals in mind, that is to allow complete interrogation of the underlying hardware/software machine with minimum constraints imposed by the debugger/monitor.

The PM, GR and BM memories are "opened" and "closed" via Direct Memory Access (DMA) from the PDP8e. In the case of the three triple line buffers I1, I2, and I3, a program is swapped into the PM from the PDP8e to copy the line buffer line to the PDP8e via GR DMA. After modification, the GPP PM and GR are restored. In general, memory, modulo the current data space, may be opened in the current data space by typing (using the OS/8 ODT conventions)

nnnn/ XXXX yyyy (carriage return)

where the XXXX was typed in response by the system and the lower case characters by the user. The user may respond by typing an expression yyyy (carriage return) to replace the contents of nnnn (i.e. XXXX) with yyyy. By responding with just a carriage return, the contents remains unchanged.

For example:

```
400 / CLA        tad+.+1
401 / JMP 403    disp1
```

To replace the contents of a location with 6-bit Ascii string data, the Ascii string should be enclosed in the delimiter ! (e.g. !ABCD!). Up to four characters may be coded this way since the maximum variable size is 24-bits. Six-bit conversion is done right justified (left filled with zeros). For example:

```
200/nnnn    !AB!     (load 6-bit "AB" into loc 200)
```

Ascii string data occurs in I/O statements and in string data used in interpreters such as DDTG.

1.1.2 Temporary data space specification

A temporary access to a data space mode different from the current mode may be made by using the construct "&<data space>" following the address but before the "/". For example,

```
nnnn&PM/
nnnn&GR/
nnn&I1/
nnn&I2/
nnn&I3/
mnnnnnn&BM/          (m is memory nnnnnn is yxaddr)
mnnnn&PDP8E/         (m is field, nnnn is address. m is
                     ignored when in $PROTECT mode
                     and field 7 special segment is assumed.)
```

1.1.3 PDP8e special segment

In general, in the RTPP, the PDP8e plays the role of a sequencer for the GPP. In order to cotrol specific GPP programs, 4K of PDP8e memory is allocated. This memory space is called the PDP8e special segment. It is possible to load PDP8e programs (specifically absolute binary files) into the special segment (see sections 2 and 8) using the LOAD command (see section 1.14).

As the user is able to modify the PDP8e memory space it is possible to destroy DDTG. Two commands to DDTG allow the user to protect the PDP8e DDTG memory outside of the special segment, and to unprotect it.

$PROTECT (the 4K field 7 data space is the special segment and all 8e addresses map into field 7.)

$UNPROTECT (the 32K absolute 8e address needs to be specified in this mode.)

The default on entering DDTG is PROTECT mode.

1.2 DDTG command input

Input is taken from the teletype (or from a disk file--see $EXECUTE command) and is accepted until a break character (line feed, carriage return, /, <, >, =) is typed. Up to that point, various editing characters may be used to facilitate command input. These are as follows:

Edit commands

"Rubout" will erase the last DDTG teletype input character.

Control/R will print out the cleaned up teletype input entered so far (cf. line feed in OS/8 monitor level).

Control/C will save the state of DDTG on the SYS: and exit to OS/8.

Control/U will erase the current DDTG teletype line.

Control/T will print out the status of DDTG (Cf. DDTCMN.FT for discussion on DDTGSTATUS).

Control/E is only operative during the reediting of a DDTG function (cf. 1.10.1) and is used to terminate additional input text.

Control/Z is only operative during the reediting of a DDTG function (cf. 1.10.1) and is used to terminate text input.

Break characters

"/" opens the specified (or current if not specified) location in specified (or current if not specified) data space.

">" or line feed will open .+1 location in the current data space. Line feed is added for symmetry with ODT. It does not close or change the current location.

"<" will open .-1 location in the current data space. It does not close or change the current location.

Carriage return will either store the value of an expression, close the current opened location if it was open or execute the given DDTG command. If on closing a location, no expression is given, then the contents of the location remains unchanged. Note that during the reediting of a function (cf. 1.10.1) only the carriage return break character is operative.

"=" will cause the printing of the value of the specified expression.

Modes

Various modes of operation are available in DDTG. Typing $MODE or $MODES will type out the current state of the DDTG mode switches.

$ECHO will cause the input command stream to be echoed on the TTY:.

$NOECHO will cause the input command stream to not be echoed on the TTY:.

$SPOOL causes the teletype output to be written onto MTA1: in addition to being typed on the teletype. On entry to DDTG, the magtape is rewound and on exit, the spooled file is closed. This is one aspect of the automatic "note taking" capability.

$NOSPOOL turns off the spooling but does not close the spooler file. This allows selected output to be spooled. On entry, DDTG is set to NOSPOOL.

$PROTECT (the 4K data space is the special segment) (cf. Section 1.1.3). (The default option.)

$UNPROTECT (the 32K address needs to be specified in this mode.) (cf. Section 1.1.3).

$OCTAL sets the TTY I/O to octal. (The default option).

$DECIMAL sets the TTY I/O to decimal.

$BCD sets the internal number conversion I/O to BCD and TTY I/O to decimal. This mode is useful for looking at Qunatimet data which is usually in BCD. It effectively performs BCD conversion on all register I/O transfers.

$SIXBIT toggles (alternates the on/off value of) a printout on/off switch which will cause the contents of locations which are opened to be printed in 6-bit Ascii. It overides the effect of the $OCTAL, $DECIMAL, $BCD switches and erases the effect of the $SYMBOLIC switch. This switch is similar to "nT" printout in PDP10 DDT.

$SYMBOLIC toggles a printout on/off switch which will cause the contents of locations which are opened to be printed in as op-codes associated with that address

space. It overides the effect of the $OCTAL, $DECIMAL, $BCD switches and erases the effect of the $SIXBIT switch. This switch is similar to nT printout in PDP10 DDT. An alternate form of the $SYMBOLIC command is $S.

$COMMENT toggles on/off a printout switch which will cause a request for comment to be issued when a data file is created and the switch is on. If a data file is being read by DDTG and the comment switch is on, then the comment is printed on the teletype before the operation proceeds.

$STDDET assigns the control desk switches thresholds 1 and 2 to control the Quantimet Standard detector thresholds B and C. $STDDET is the default mode.

$DIGDET assigns the control desk switches thresholds 1 and 2 to control the Quantimet Digitizer detector thresholds A and B.

$MANUAL enables the manual control of the mechanical state of the RTPP from the control desk keys. This includes the control of the Axiomat x and y stage, Axiomat focus and zoom, the light source variable wavelength and neutral density rotary filters and the Quantimet thresholds 1 and 2.

$NOMANUAL disables the manual control of the state described above.

$SETQMT resets the Quantimet program words (QPROGn), status register, Frame and scale positions and sizes, clears the Mask register to the standard configuration.

1.3 Breakpoints

Breakpoints may be inserted in the GPP PM or PDP8e special segment. If PDP8e is in $UNPROTECT mode then breakpoints may be put in DDTG itself! The concept and use of breakpoints is explained in detail in [DEC73] and [DEC74]. Eight breakpoints are available in DDTG.

GPP PM breakpoints are implemented by swapping a PM "HLT" instruction and comparing the GPP PC on detecting a status (data address bus trap) register "run" off. The DABTRP (data address bus trap) register could also be used for detecting particular data patterns occuring on the DAB. The PCTRP (program counter trap) register is more useful in console debugging than in DDTG. Details on the trap registers is given in [Lem76b].

PDP8e special segment breakpoints are implemented by breakpoint processing code to be inserted into locations [70024:70027] which are therefore not to be used by the programmer.

The format is:

nnnn$BREAK m - m is 1 to 7 and is the breakpoint number in the current data space (PDP8e or PM) and nnnn is the address to put the break.

or

nnnn$BREAK m&PM - force the break to be in the PM data space.

or

nnnn$BREAK m&PDP8E - force the break to be in the PDP8e data space.

$BREAK m - remove breakpoint number m wherever it is.

$BREAK m= - prints the address nnnn of breakpoint m.

$BREAK= - prints the addresses of all breakpoints

1.3.1 Continuing from breakpoints

A command is available to continue from a breakpoint "n" times. This is implemented by DDTG successively testing a

counter each time the breakpoint is encountered until the counter counts down to 0. If the counter is not 0, then the breakpoint is automatically continued. The format is:

nnnn$CONT m - continue from breakpoint m, nnnn times

or

$CONT m - continue from breakpoint m once.

1.4 Starting the GPP and PDP8e special segment

The GPP or PDP8e special segment may be started at any location by either of two methods. The $START command (discussed in section 1.14) and the $GO command. The $GO command has the following syntax and semantics.

nnnn$GO - start the current GPP or PDP8e special segment data space at nnnn.

or

nnnn$GO&PM - to start the PM while in another data space.

or

nnnn$GO&PDP8E - to start the PDP8E while in another data space.

The special segment (GPP) is started if the current data space is the PDP8e (PM).

1.5 Data space memory searches

A data space memory search is a sequential checking of the contents of sucessive data space locations for a match against a search value nnnn. The format is

nnnn$SEARCH - search in the current data space.

or

nnnn$SEARCH&PM - search in the PM data space.

or

<value>$SEARCH - where <value> is either a PDP8e IOT or a variable in which case the value of the variable is

used.

CLA $SEARCH	- search for CLA in the PDP8e
VAR1$SEARCH	- search for the contents of the variable VAR1.

The addresses of all locations in the specified data space which match the search variable are printed. Typing control/o during a search will terminate the search.

1.6 DDTG symbols

DDTG recognizes symbolic equivalents for GPP operators, special registers, GR "Ri", PDP8e operators etc. DDTG has the facility to add GEEPER compiled program symbols as well as DDTG user generated symbols to its symbolic memory. These symbolic equivalents may be used instead of numbers wherever desired. The exact interpretation of these symbols depend on their usage in the current data space.

e.g. 1234/ ADD,'I1[5],#6,I3[5]

would be used to enter "I3[5]<=='I1[5] ADD #6".

Where ' denotes an indirect address specification and # denotes an immediate address specification.

A+1=

will print the value of the contents of variable A plus 1.

1.6.1 Adding new symbols

New symbols are entered by using the backslash (\ not /) using the syntax

<ival[1]>,<ival[2]>,<symbol type>\<symbol name>

where the three fields ival[1], ival[2] and itype are discussed in more detail in section 4.4.

For example:

```
        0,1234,PM\NEWPMSYM    - (new symbol at address 1234 in the PM)
or
        0000,1600,PDPMRI\TADI - (define 8e memory ref. instruction)
or
        0000,0146,GPPOPR\XOR  - (define GPP operator)
```

1.6.2 Deleting symbols

Symbols may be deleted by the $DELETE command which deletes the list of symbols which preceed it. Only the symbolic equivalent of a symbol is deleted, not its usage (i.e. value) in its related data space, e.g.

```
        <list of symbols>$DELETE
```

1.6.3 Listing the symbol table

The symbol table may be listed selectively by data space by typing the following (with control/O to stop the listing):

```
        <list of data spaces>$LISTSYM
or
        <list of examples of other data types>$LISTSYM
```

The examples include numbers, variables, function names, file names and extensions, I/O devices, PDP8e IOT's etc.

For example, to look at PDP8e load IOT's use an example of one such as:

```
        MQL$LISTSYM
e.g.
        PM$LISTSYM    (to list all PM symbols)

        A_-4
        A$LISTSYM     (to list all DDTG variables since A
                       is a variable)
```

Specific information on the value and type fields of

one or more symbols may be obtained by typing:

```
<list of symbols to print>$SYMPRINT
```

1.6.4 Dollar '$' operators

DDTG multi-character operators and PDP8e Input Output Transfer instructions (IOT's are discussed in [DEC74]) may be used with or without a "$" prefix. The "$" prefix is useful as a delimiter but a space or comma may be used as well. For example,

```
$READSTATE
```

and

```
READSTATE
```

are equivalent.

1.6.5 Symbolic expressions for addresses and computations

Arithmetic expressions using +, -, *, % (integer add, subtract, multiply and divide) may be used in all symbolic address specifications. No parentheses are allowed. Operator precedence ((*,%) over (+,-)) is governed by a left to right scan. The data space mode used is that of the last symbol scanned before the "/" or the current mode if arithmetic without symbols is performed.

e.g.

```
GO+6/
```

or

```
LIST+OFFSET-5/
```

or

```
5+6*4/
```

or

```
.+6/
```

Expressions may be evaluated on the terminal by typing an expression followed by an equal sign "=".

```
GO+6=
```

1.6.6 Expression assignment and evaluation statements

The value of an expression may be stored in the value part of a variable symbol using an assignment statement. The assignment operator is the backarrow "_".

```
A_B+C*D
```

or

```
A_$HPR   (reads the BCD frame and scale position and stores
          it in A.)
```

The value of the variable may then be used in an expression. For example

```
A+1=
```

1.6.7 Subscripts

Single index subscripts are allowed on data spaces and address symbols as a means of accessing elements of the data space when the data space is treated as a 1-dimensional array with subscript value 1 corresponding to the address of the array name variable. The expression inside of "[...]" must evaluate to a number in the legal range of the data space or address.

A line buffer current neighborhood is addressed by omitting the square brackets as in I12 (which is just an address looked up in the symbol table). That is, I<line buffer name 1, 2 or 3><current neighborhood pixel name>. Note: the line buffers consist of 3 lines: the first (y-1) ranges from 0:377, the y line from 400:777, and the (y+1) line from 1000:1377.

For example, to access element "12345" of the buffer memory BM2.

```
A_BM2[12345]
```

To access the 3rd point in line buffer I1 in line Y+1 first define the line offset YPLUS1.
Then

```
YPLUS1_1000
```

```
B_I1[3+YPLUS1]
```

For example, to access an array in the PDP8e such as the polling loop read IOT's array POLLR.

```
POLLR[4]/
```

The write IOT would be similarly used:

```
POLLW[4]/
```

1.7 PDP8e Input Output Transfer Instruction - IOT execution

A set of PDP8e Input Output Transfer instructions (IOTs) are used to control the GPP, the Quantimet and control desk. The RTPP specification [Lem76b] for more details on these IOT's. Appendix B lists the PDP3e IOT's used in DDTG. This list is broken up into GPP (General Picture Processor) related IOT's and Quantimet control desk related IOT's.

The IOTs may be evaluated at DDTG control teletype level. The contents of the PDP8e accumulator (AC) at the time of the IOT evaluation may be specified by an optional left argument. If no number is specified, c(AC)=0000 before execution of the instruction. For those instructions which read the AC, the value of the AC is printed after their execution if the read IOT is given by itself.

In addition, the IOT may be used in arithmetic expressions taking its argument (if a LOAD IOT) or returning a value (if a READ IOT) from/to the expression evaluation. For example:

```
$BCD
$STQMT
$QMSKP
QDAT1*1000+$QDAT2=
```

will run the QMT and then compute the double precision BCD QMT data.

```
$BCD
HORPOSVAR_$HPR+10
HORPOSVAR$HPL
```

will read the frame and scale horizontal postion, add 10 to it and move the frame right to this new position by loading the value into the horizontal frame and scale position register. Registers which use BCD input and output should be used only in BCD mode and vice versa as the BCD conversion is done during the actual register I/O.

If the symbol is used as the contents in the response for a PDP8e opened location, it is not evaluated and its corresponding 12-bit "value" will be used.

e.g. 200&PDP8E/ 0000 $STQMT

will insert the opcode for STQMT in special 8e segment location 200.

1.7.1 Polling loop execution of read/write IOT pairs

At the leve of user interaction, DDTG is in a polling loop. This loop polls the clock, manual state control desk switches, and teletype at 200 Hz. In addtion, it evaluates up to 16 ordered [0:15] read/write IOT pairs. This facilitates the building of simple minded monitors for debugging the hardware and other types of control functions. The syntax is as follows (where "^" is an uparrow character and index is a number in the range [0:15]):

<write IOT> ^ <read IOT>,<index>

For example, a polling loop monitor could be built to load QPROG3, DETC and DISP1 from FBW5, FBW7 and RKYPDL control desk switch IOTs respectively.

```
$QPROG3 ^ $FBW5, 1
$DETC ^ $FBW7, 2
$DISP1 ^ $RKYPDL, 3
```

This would have the effect of loading part of the Quantimet program word from switches on the control desk, loading the Quantimet threshold C detector level from control desk switches and loading one of the displays on the control desk from the control desk keypad.

Initially, the read/write polling loop is cleared. It may easily be setup by using a set of polling loop triples in a

$EXECUTE function. As the polling loop is part of COMMON it will be restored whenever DDTG is exited and reloaded. The polling loop is initially null when a system is built (.R INISYM.SV). It may also be cleared by entering (0^0,i) to clear entry i. The $ZEROSTATE command will also clear all entries.

1.8 Microstate stage control

The state of the auxillary microscope stage (x,y,focus), light source intensity and wavelength, and Quantimet detector threshold values is controlled directly by the command

$MOVESTATE(Th-2,Th-1,x,y,focus,zoom,intens,wavelth,<switch>)

This command uses the MOTORS/MVMTR subroutines modified by Gerson Grosfeld from the NCI GRain Counter 1.1 [LipL74] to move the selected motors. The 8 arguments (Th-2,Th-1,x,y,focus,zoom,freq,dens) specify the change in the position of the respective state while <switch> is the symbol REL or ABS. If the motion is absolute, the symbol P is used to denote no change in the corresponding microstate motor position. For example:

$MOVESTATE(0,0,+3,0,0,0,0,0,REL)

or

$MOVESTATE(P,P,100,-34,+2,P,P,P,ABS)

will move the stage in the first case +3 RELative in x. In the second case (x,y,focus) will change to the ABSolute state position of (P,P,100,-34,+2,P,P,P,P) where P means the previous position. Variables can also be used as well as numbers to specify a change in state. The command

$READSTATE

prints the 8 state vectors.

If 8 variable arguments are specified for $READSTATE (like in $MOVESTATE) then the values are saved in the variables instead of being printed.

The microstate is set to a zero vector when the DDTG symbol table is compiled. It is possible to zero it at any time using the $ZEROSTATE command. The manual state control default is normally disabled. It is toggled on and off by the switches $MANUAL and $NOMANUAL respectively.

1.8.1 Control desk keys - detector assignment

The control desk keys Threshold-1 and Threshold-2 are assignable by DDTG to either of the two externally programmable

QMT detector modules. This key correspondance assignment is performed by giving one of the following commands.

$STDDET - assign the Standard Detector with a range of [0:4095] (0 is white and 4095 black).

$DIGDET - assign the Digitizer Detector with a range of [0:63] (0 is white and 63 black).

The threshold keys are not activated unless the system is put into MANUAL mode by the $MANUAL command.

1.8.2 Sampling the 16 channel analogue/digital converter

A 16 channel multiplexing analogue to digital converter is avalable on the PDP8e with channels 0 and 1 assigned to the galvanometer scanner and control desk knobs[0:7] assigned to channels [8:15]. The A/D (DEC AM8e/AD8e) has a 10-bit resolution on -1 to +1 volt input. The $SAMPLE(channel number) function is available which returns the value of the sampled channel. $SAMPLE may be used in any arithmetic expression that numbers or variables may be used in. For example:

```
(3)$SAMPLE=
```

will sample A/D channel 3

or

```
VALUE_100+(FBW4)$SAMPLE
```

sample channel selected by control desk register 4.

1.8.3 Standard Quantimet configuration

As the Quantimet program words (described in [Lem76b]), status register, frame and scale, and other related control registers constitute a rather large number of conditions to be concerned with, it was felt that a standard state condition would be helpful.

Because of the great complexity and the large number of possible Quantimet configurations, we have specified a standard configuration for the Quantimet subsystem of the RTPP. As as result of some simple hardware it is possible to reconfigure the Quantimet configuration under program control (Program control is effected by changing various hardware registers called QPROGi which are listed in [Lem76b]. This is, of course, without the necessity of the user changing front panel controls. The standard Quantimet configuration does assume certain front panel switch settings such as all programable modules being in the AUTO position.

One would start from this state modifying it as required for the particular experiment being performed through incremental changes to the QPROGn words. The $STDQMT command will set up the standard state as follows:

1. Set the Frame and Scale to a 256x256 frame at (384,384) the standard central frame for doing galvanometer scans.

2. Set the QSTAT status register to 4000 to enable the Frame and Scale remote controls.

3. Set the MS3 computer to area, the Digitizer/Detector to off, the Amender to unmodifier, the Function computer # 1 to area, # 2 to Volume.

4. Clear the mask register.

5. Zero the remote chord sizer registers SIZEA, SIZEC, SIZEM, SIZES.

The $STDQMT command also has the effect of moving the galvanometer scanner to (0,0) physical position.

1.9 GET/PUT data file space - data acquisition

In addition to serving as a debugger and minitor for the RTPP, DDTG also has some image acquisition and display I/O facilities. These facilities allow the use of parts of the initial RTPP system to be used for image processing experiments. The Philosophy is that all I/O done in this mode will be between a file structured device such as a disk and the actual image acquisition or display device. Thus one 'gets' data from a scanner or Grafpen and 'puts' data into a display or mask register. The commands to implement this idea are $GET and $PUT.

Before going into the details of $GET/$PUT operation, some terms will be defined. A picture or image is a square array of gray (darkness) values. In the RTPP images may be either 256x256 or 1024x1024. The density values range from 0 (white) to 255 (black) for the galvanometer scanner (to be discussed). The Quantimet video system has 64 gray values ranging from 0 to 63. A mask is a binary array or of the same size as pictures. It is used to denote whether one will use gray scale data in an associated image. Maks may be stored in several ways: outlines of delimited regions (such as in the MASK register) or as full pixel representation (as in buffer memory masks). Each data file generated by DDTG and read by DDTG uses a file header which contains information about the type of data file being used as well as historical information.

Data may be acquired from the RTPP and stored in a PDP8e file using a $GET construction. Similarly, data may be loaded into the RTPP from PDP8e disk files using the $PUT constuction. These are given as follows:

$GET

$GET <file spec.>,<data type>,<optional switches>

or

<file spec.>_<data type>,<switches>

$PUT

$PUT <data type>,<file spec.>,<optional switches>

or

<data type>_<file spec>,<optional switches>

where <file spec> is a complete OS/8 file specification such as "DSK:F.DA".

The file specification includes a device name such as SYS:, DSK:, DSKG:, DTA1: etc. A file name is a 6 character alphemeric file name followed by an optional "." (period) and 2 character alphemeric file extension.

If the special symbol "FILEGEN" is used for a filename in the specification then a unique filename is generated. The generated name then consists of the first two letters "FN" followed by a sequentially derived four digit octal number in the range of [0:7777] which is also the value of FILEGEN if used as a variable. The file extension is that specified by the user.

If the $COMMENT mode is active, the a comment will be requested on each $GET and the comment associated with the file printed on each $PUT. If the USECLASS switch is used on a $GET, the system waits for the user to press a class key on the control desk before proceeding with the $GET. The command lights are lighted before the key is pushed and go out when the key is pressed as visual feedback. The class number is then stored in the file header as explained in section 3.

1.9.1 DDTG data file spaces

There are seven types of DDTG data file spaces available. These data file types correspond to physical devices. The physical characteristics of these I/O devices are discussed in [Lem76b]. Data sub-types are specified using the switches to be discussed. The data types are:

BM0 through BM7 (buffer memories)
MASK (mask register)
QMT (Quantimet function computer and ACP data)
GRAFPEN (Graf-Pen)
GALSCAN (galvanometer scanner)
DICMED (Dicomed gray scale display).
STATE (8-tuple RTPP mechanical stepping motor state)

1.9.2 DDTG data file switches

Switches are used to modify the basic data file space types listed above. Their effect is to create data file space submodes.

```
ALLBM     - transfer both high and low byte of BM
FILL      - fill in a line drawing boundry image
                with constant gray value
FULLRASTER - perform 1024x1024 image transfer
HGHBYT    - use the high byte image in the BM transfer
NOVECTOR  - don't generate interpolated line drawings
USECLASS  - wait for Class key to be entered on $GET
USEDIC    - use the dicomed to display during transfer
USEFILEXY - use the (x,y) window position in the file
                to specify the window instead of the
                Frame and Scale on $PUT.
USEFRM    - accept line data inside of FRAME AND SCALE
USEMSK    - use the mask register to mask GALSCAN
            or GRAFPEN input.
```

The above data file types and switches may be represented in the following table which specifies legal combinations with a Y and illegal combinations with a "-".

DATA FILE TYPE	$GET	$PUT	NOVECTOR	FULLRASTER	USEFILEX	FILL	USEMASK	USEDICB	ALLBMY	HGHBYT	USEFRM	USECLAS
BM0-BM7	Y	Y	-	-	-	-	-	Y	Y	Y	-	Y
MASK	Y	Y	-	Y	Y	Y	-	Y	-	-	Y	Y
QMT	Y	Y	-	-	-	-	-	-	-	-	-	Y
GRAFPEN	Y	-	Y	-	-	-	-	Y	-	-	Y	Y
GALSCAN	Y	-	-	Y	-	-	Y	Y	-	-	Y	Y
DICMED	-	Y	-	Y	Y	Y	-	-	-	-	Y	-
STATE	Y	Y	-	-	-	-	-	-	-	-	-	Y

Table 1. Switch applicability table for $GET and $PUT.

The $GET and $PUT operations are legal only where appropriate (where the data types are compatible). For example, the Dicomed can display data files generated on the buffer memories, galvanometer scanner, Graf-Pen, and Mask register. The mask register may be loaded with a file generated by the Graf-Pen or thresholded BM or galvanometer scanner images. It is impossible to get an image from the Dicomed - so that $GET DICMED would be meaningless.

The $PUT command is very useful for manipulating BM windows where more than 8 images are involved. Notice that no <data type> specification is needed as this information is carried along with the file.

BM0-BM7

Data in the eight picture buffer memories (called BM0, through BM7) is 16-bit data with a high and a low 8-bit byte. Normally, only the low order byte is transfered unless the HGHBYTE or ALLBM switches are used. A BM (non-ALLBM switch) window is equivalent to a 256x256 pixel galvanometer scanner window and has the same format.
3-bytes/2-words.

MASK

A convex non-reentrant perimeter mask may be used (and generated from the Quantimet detected video. A mask may be acquired by doing the sequence (GETMSK, STQMT). The hardware stores the first detected line intercept and the next undetected detected line intercept as entrance and exit x-coordinates. The mask register is then available to be read by the PDP8e (MASK $GET) or loaded by the PDP8e (MASK $PUT operations). The mask register may then be used to supply a detection frame for the Quantimet in a manner similar to that of the frame and scale. This is discussed in more detail in [Lem76b].

MASK register data is 1024 pairs of 10-bit words consisting of line intercept pairs of x-coordinates (entrance, exit) pixels starting line 0 and going to line 719. Lines 720 to 1023 only may sense when loaded from the PDP8e (such as from the GRAFPEN etc.) and will contain random values after a GETMSK operation.

Another way of loading the mask register is via the GRAFPEN when doing a $GET with the USEMSK switch. This is discussed below.

QMT

The Quantimet (QMT) interface is capable of acquiring specific features for all detected objects (up to a maximum of 1024 objects) in a single Quantimet scan. This is done by setting the MS3 computer module in the Quantimet to "pattern recognition" mode. The assumption is made that the program for the Quantimet and thresholds have been previously selected and that a STQMT command has been issued. The control of the Quantimet is discussed in more detail in [Lem76b].

The RTPP provides special purpose hardware for on line data acquisition and storage from individual Quantimet scans.

Such data concerning as many as 1024 separate objects may be stored in this special shift register hardware. In addition to the x-y coordinates of each object (Anti-cooincidence point), the 5-tuple may contain feature data such as area, perimeter, density, and various types of projections. The nature of the data stored is of course dependent on the state of the Quantimet as determined by the set of QPROGi.

GRAFPEN

The Graf-Pen is used only with a $GET as it does not make sense to load the pen with a $PUT. Pen data acquisition starts when the pen is put down and stops (with the file being closed) when the pen is positioned in the region at (2000,2000) on the tablet. The QMT cursor is constantly loaded with the current pen position even when pen data is not being saved. Data is acquired for use by DDTG when the Graf-Pen microswitch at the tip of the pen is pressed.

The Graf-pen data acquisition presents a further difficulty. It samples at a maximum rate of 200 x-y pairs/second. Consequently if the pen is moved too fast then data will not necessarily be taken from adjacent pixels (which would be necessary for a contiguous boundary to be acquired). Therefore, an automatic point generation mode (default VECTOR) is used to fill in missing points by doing a linear (Y=MX+B) approximation between sample points. The vector generator part of the system was written by Bruce Shapiro using the PDP8e floating point processor hardware. If the line length is greater than 30 pixels, then a new line vector is started. This allows the painting and editing of different unconnected line segments.

This latter mode of operation is necessary in order that Graf-pen data be usable by the mask register. Vector mode may be bypassed by specifying the NOVECTOR switch during Graf-pen data acquisition. E.g.,

```
SYS:A.DA_GRAFPEN,NOVECTOR
```

In either case, all Graf-pen data taken is checked to see if it is within an equivalence (currently being used) region so as to prevent excessive data from being taken when the pen is stopped. The smallest equivalence region is the pixel which was last acquired.

A Frame and Scale window less than or equal to 256x256 pixel size may be used to mask acceptable Graf-Pen data by

use of the USEFRM switch. Data outside of this window is ignored. As the window requires 8-bits of information, (x,y) data is mapped to [0:255] within the window region. The actual window horizontal and vertical positions are stored in the file header so no actual information is lost.

Note that without USEFRM, it is possible to enter (x,y) data with a range of [0:1999]. The representation of using this data on the Dicomed or Mask register is to have wraparound. Normally, the cursor provides visual feedback to the user via the QMT display so that the wraparound would be detected and avoided if desired. This 11-bit data is permitted so that Graf-pen applications where more than 10-bit resolution is required could be performed.

Graph-pen data may be constantly loaded into the mask register as it is acquired by using the USEMSK switch. This mode of operation lets one create and edit masks using the Graph-pen with the intention of ignoring the resultant GRAFPEN data file. Moving the pen downward defines the entrance perimeter points while moving it upward defines the exit perimeter points on the mask. To clear the mask inside of the frame at any time while using the USEMSK switch, press the red "erase" control desk command key.

Graf-pen data may be displayed during acquisition by use of the USEDIC switch. This causes the (x,y) data to be display at gray level 63 on the Dicomed display.

GALSCAN

The galvanometer scanner (available with a $GET) is an 8-bit gray level 1024x1024 pixel mirror-photomultiplier scanner mounted on the microscope. It may operate in two modes. The normal mode is that of a 256x256 raster scanner.

A full 1024x1024 raster scan window will also become available and is invoked by the use of the FULLRASTER switch. Alternatively, the scan may be directed to scan a set of points inside of the mask register (USEMSK switch) and/or the Frame and Scale (USEFRM switch). That is, g(x,y) outside of the specified mask will be set to 0 otherwise it will be the actual gray value at that point. The result of running the scanner in any case is a raster file of 8-bit gray scale data.

The position of the 256x256 scan window is specified by the value of the (HPR,VPR) frame and scale upper left-hand corner coordinate pair.

Currently, in order to be congruent with the Quantimet (to within 2 pixels) only the central 256x256 raster window is usable. Eventually, a dual resolution scanner drive will be added so that full raster windows will be available. The central raster is located at (384,384). The $SETQMT command (amoung other methods) will reset the Frame and Scale to the standard galvanometer scan windw (384,384) and 256x256.

To display the scan while is is being acquired, the USEDIC swith is used. The galvanometer scanner function is documented in [Lem76b].

If a non-zero number or variable is included in the specification, then each point sampled in the scan will be averaged over that number of samples. The maximum number of samples in each average is 16.

DICMED

The Dicomed 31 storage display (available with a $PUT) is a 1024x1024 6-bit gray scale display which may operate in three modes. The first is 256x256 or 1024x1024 raster mode where the input data file contains 8-bit gray scale data. The top 6-bits of the 8-bit byte is used as data. The 2nd mode is point mode where (x,y) pairs are specified as from Graf-pen data. The third mode generates filled single gray scale images from mask register (run-end type data) or raster images. The gray scale is black if it is not specified as a number in the $PUT command or is specified from the file if a number was specified with the file when it was generated with a $PUT.

The system makes use of the file header information and determines what type of source device generated the data and displays the appropriate image accordingly.

The position of the 256x256 window is specified by the value of (HPR,VPR) frame and scale upper left-hand corner coordinate pair. The Dicomed display may be controlled to some extent by DDTG. If the USEFILEXY switch is specified (x,y) are gotton from the window position associated with the file rather than the current (HPR,VPR).

The $ERASE command erases the Dicomed and waits 20 seconds for the erase to finish. The $VIEW and $NOVIEW commands turn the picture viewing light on and off respectively.

The FILL switch is used with the display of MASK register data to generate a filled in area image. Otherwise

just the perimeter is displayed.

Data may be displayed while a $GET operation is in progress by using the USEDIC switch where appropriate. This is valid for GRAFPEN and GALSCAN modes.

STATE

As was discussed in Section 1.8 the STATE of the RTPP electromechanical hardware may be controled through DDTG. This control may be performed either through the use of keys on the control desk (while in MANUAL mode) or via the $MOVESTATE command. In addition, the state may be associated with data files and may be saved and restored as such.

The 8-tuple RTPP state consists of the current positions of the mechanical stepping motors to control the microscope and the two threshold keys. The 8-tuple is described in microstate stage control. The state may be changed using the $MOVESTATE command. It may be read with $READSTATE and cleared with $ZEROSTATE. The use of the STATE data file space with $GET/$PUT allows entire states to be saved and restored easily by name. As the actual state information is stored in the header of the file, any other data file space file may be used to restore the state with a $PUT command.

1.9.3 The Logical Coordinate System - LCS

The Logical Coordinate System (LCS) used is that which is common to most of the physical devices Dicomed, (QMT, BM, frame and scale, mask register, x-y-ACP's, Cursor). LCS has (0,0) as the upper left-hand coordinate and (1023,1023) as the lower right-hand coordinate. Positive X is to the right and positive Y is down. There are no negative coordinates. In order for the LCS concept to be operational, all devices must be aligned and calibrated to be congruent.

Several of the physical I/O devices are not in the LCS and must be mapped by DDTG into the LCS. This mapping is done automatically by DDTG. The physical galvanometer scanner ranges over (+512 to -512) by (+512 to -512) in the corresponding mapping to the LCS. The physical Graf-pen ranges over the positive quadrant Cartesian coordinates with (0,0) at the lower left-hand corner and (4095,4095) at the upper right-hand corner. Both the galvanometer scanner and the Graf-pen may be used in the LCS with no user concern what-so-ever as the LCS coordinate values are mapped to/from these physical devices internally by DDTG.

A standard window of 256x256 pixels is used throughout the RTPP and DDTG system. To load the LCS 256x256 window position one loads the frame and scale position registers (HP and VP). For example:

```
$BCD
100$HPL
100$VPL
```

sets the window at (100,100) upper left-hand coordinate. The window size is set by:

```
256$HSL
256$VSL
```

The $SETQMT command will set up a standard galvanometer scann window at (384,384) and 256x256.

The USEFILEXY switch with $PUT commands substitutes the (x,y) window position associated with the input file for the normally used values of (HP,VP).

One should note also that within a 256 pixel line (such as the triple line buffers I1, I2 and I3 one addresses a pixel

by an index in the range of [0:255] rather than the range [1:256].

1.9.4 Examples of the use of GET/PUT on various data types

Some examples of the GET/PUT syntax are given below where NOVECTOR, FULLRASTER, USEFILEXY, FILL, USEMSK, USEDIC, and ALLBM are switches.

(a)
```
ERASE                  (erase Dicomed)
SYS:A.DA_MASK          (256X256 mask image at HPR,VPR)
DICMED_SYS:A.DA,FILL   (display area mask data
                        at HPR,VPR)
DICMED_SYS:A.DA        (display perimeter mask data)
```

(b)
```
SYS:A.DA_MASK,FULLRASTER  (1024X1024 mask image)
DICMED_SYS:A.DA (display 1024X1024 perimeter)
```

(c)
```
SYS:A.DA_GRAFPEN (default generate vector line points)
DICMED_SYS:A.DA (display Graph-pen boundary)
MASK_SYS:A.DA,FILL (put the boundary into the mask
                    and display in area mode)
```

(d)
```
SYS:A.DA_GRAFPEN,USEDIC,NOVECTOR (display data on
                  Dicomed and don'te generate the vector
                  line segments between points.)
```

(e)
```
SYS:A.DA_GRAFPEN,USEFRM  (accept only those points
                  inside of the 256x256 frame and scale
                  window.)
```

(f)
```
SYS:A.DA_GRAFPEN,USEMSK (load the mask register with
                  pen data as it is qacquired using
                  the convention that down is entrance
                  and up is exit line segments.)
```

(g)
```
SYS:A.DA_QMT   (acquire function computer/ACP data)
QMT_SYS:A.DA     (load only the X,Y ACP coords)
```

```
(h)     SYS:A.DA_GALSCAN (scan 256X 256 at HPR,VPR)
        DICMED_SYS:A.DA (display at HPR,VPR)
        MASK_SYS:A.DA,200    (load thresholded image into the
                                 mask for G(X,Y) > 200)
        10$HPL          (change Frame and Scale position)
        10$VPL
        DICMED_SYS:A.DA,USEFILEXY (display file at its window
                       rather than current Frame&Scale)

(i)     SYS:A.DA_GALSCAN,FULLRASTER (scan 1024X1024)
        DICMED_SYS:A.DA    (display 1024X1024)

(j)     SYS:A.DA_GALSCAN,USEMSK (scan 256X256 using the
                                 mask register)

(k)     DICMED_SYS:A.DA (display on the Dicomed)
        MASK_SYS:A.DA (load the mask register using derived mask)

(l)     SYS:A.DA_GALSCAN,USEMSK,FULLRASTER (scan 1024X1024
                                 using the mask register)
        DICMED_SYS:A.DA (display on the Dicomed)
        MASK_SYS:A.DA (load the mask register using derived mask)

(m)     SYS:A.DA_BM1 (get 8 bit low byte packeded data from BM 1)
        DICMED_SYS:A.DA (display BM data on the Dicomed)

(n)     SYS:A.DA_BM1,ALLBM   (get 16 bit packed data from
                                 buffer memory 1)

(o)     BM2_SYS:A.DA,HGHBYT   (put 8 bit ,high byte data into
                                 buffer memory 2).

(p)     SYS:A.DA_STATE            (save the current RTPP state)
        STATE_SYS:A.DA            (restore the RTPP state)

(q)     SYS:A.DA_GRAPPEN,USECLASS  (request Class key before
                       continuing with  $GET)

(r)     SAMPLES_10       (set to sample 10 times/pixel)
        SYS:A.DA_GALSCAN,SAMPLES
```

1.10 Indirect DDTG command execution $EXECUTE

Commands may be composed into programs and stored in files. A command file may be throught of as a indirectly executed command. Executing a command file changes the DDTG command input device from the teletype to that of the input file.

$EXECUTE <opt. device name>:<file><opt. .DA extension>

or

$EXECUTE <function name which is really SYS:<name>.DA>

will open the file on the device specified and start reading commands from <file>.DA. Thus "plans" consisting of strings of DDTG commands may be saved for DDTG itself from the PDP10 or locally. The default device is "SYS:" and default extension is ".DA". The abreviated form of the command is "$EX". Note that the line feed and carrage return commands are ignored for $EXECUTE files.

Note that currently, special segments which require file input and use Fortran II will clobber the $EXECUTE input pointer. Therefore, it is recommended that IO.FT be used for file input where possible in those special segments which are to be embedded in DDTG functions.

1.10.1 DDTG functions definition and editing

DDTG functions of more than 1 line may be defined and edited while in DDTG. These function names are 6 character (or less) symbols. They must be defined or exist on a device as files with a ".DA" extension. They are defined by typing

$DEFINE Fi

at which point

NEW-INPUT:

is typed. the function definition is terminated by entering a control/Z character. A "#" is typed at the beginning of each input line, with control/U, control/R, rubout edit characters operative.

To edit an existing function Fi type

```
$EDIT Fi
```

at which point

```
REDEF-INPUT:
```

is printed.

On "REDF-INPUT", the old lines are typed one at a time. After each old line is typed, "OK?" is typed. If the user types "Y", the old line is preserved. Otherwise a "N" must be typed or the "OK?" question is repeated until this happens. If the line is not to be preserved than a "#" is given requesting the user to enter replacement lines (terminated with a control/E). At this point, editing of the old text continues until the control/Z of the source file (or one entered by the user) is reached. To delete a line simply type control/E when replacing a line.

To evaluate a function type:

```
$EXECUTE Fi
```

or

```
$EX Fi
```

To print a function, type:

```
$PRINT Fi
```

Often one wishes to add a function to DDTG which has been defined with a OS/8 text editor such as TECO or EDIT. To let DDTG know about the function you must DDTG EDIT the function thus instantiating the function in DDTG.

1.10.2 DDTG function control statements

Two control statements, the IF and the GOTO are available when $EXECUTEing a function.

```
IF <expression 1> <condition> <expression 2>
THEN <next DDTG statement>
```

where: <expression i> is a DDTG variable or constant,
<condition> is GT, LT, EQ, LE, GE,

<next DDTG statement> is evalueated if the condition is true.

NOTe: IF statements may NOT be concatinated as follows:

```
IF A LE B
THEN IF C LE D
```

is illegal.

```
GOTO n
```

where: control passes to the n'th DDTG command in the function.

As the GOTO is implemented by reopening the file and reading past n-1 commands it is fairly slow and so should be used with care. However, for such things as the creation of successive data sets using FILEGEN, this should be no problem. For example:

```
ERASE
DSKG:FILEGEN.DA_GRAFPEN,USECLASS,USEDIC,USEMSK
GOTO 1
```

used as a function will keep generating Graf-pen segment files until the user Control/O's or Control/C's out.

1.11 Chaining to/from OS/8 programs

Other programs may be chained to from DDTG. Since DDTG saves the status of DDTG in a SVDDTG.DA file, restoring the DDTG state is easily acheived.

```
$CHAIN  <.SV file name><optional files and variables>
```

On restarting DDTG the SVDDTG.DA DDTG state file is always loaded before standard initialization takes place. This mechanism allows intermediate data reduction to take place using stand alone software (which in turn might chain back to DDTG).

DDTG has a separate entry point (under the file RDDTG.SV) to facilitate chaining from DDTG. Upon entering RDDTG, it restores COMMON, the special segment and other information from the SVDDTG.DA file. It then tests whether the contents of location 17600 (OS8 command decoder location) is

Ascii dollar sign ($). If it is not, then control passes to the "*" DDTG command loop. If it is, then the 8-bit Ascii command in locations [17600:17642] is evaluated.

An alternative way to use RDDTG is to embed $CHAIN commands within $EXECUTE functions. Control returns to the line after the $CHAIN command from DDTG if the program chained TO exits by doing a chain back to RDDTG. This is the preferred way to use DDTG in auxiliary operating systems. Arguments including file specifications (must be conplete specification), numbers and variables may be passed to the chained programs via an argument stack in COMMON explained in Section 5.2.

1.12 Messages

Messages or comments may be printed on the teletype by enclosing them in sets of double quote marks as "....". This is useful when printing messages in the middle of executing a file. Messages are not interpreted by DDTG. For example:

```
"This is a message
and this is the rest of the message."
```

1.13 Construction of mini-RTPP monitors

Procedures to control the RTPP through the control desk, Graf-Pen, etc. are usually run through the DDTG system as pairs of [GPP,special segment] programs which may be prepared via the GEEPER assembler/compiler using the PAL switch. (Initially, mini-monitors will be prepared only for the PDP8e special segment through the use of PAL8, PAL10, or FLAP which generate absolute binary ".BN" files or FORTRAN II loaded into field 7 to generate ".SV" files.) Then they may be invoked via DDTG $RUN ($LOAD, $START) commands.

It is possible to use Fortran II programs for the mini-monitors. This is discussed in more detail in section 6.

1.13.1 DDTG user service routines - DUSR's

To facilitate the construction of mini-monitors, several commonly used but complex functions have been made part of DDTG and are callable from the special segment. These are called DDTG user service routines or DUSR's through a special calling sequence to DDTG.

The DUSR call to DDTG is stored by DDTG in field 7 of the PDP8e page 0 locations 20-23 and whenever GPPLDR loads the special segment. This overlay consists of the following code:

```
FIELD 7
*20
DUSR,   0       /entry from special segment
        CIF <to DDTG DUSR.FT subroutine>
```

```
        JMP I .+1
        <DDTG'S entry for the "DUSR">
```

To call a DUSR, one JMS's to location 20 in field 7 with the DUSR number as the first argument after the JMS. For example, to write out two Fortran integers I and J on the teletype one might do the following call.

```
        JMS DUSR
          7             /DUSR for writing on the TTY:
          2             /number of arguments
          FORMPTR       /pointer to the format statement
          I             /pointer to 1st variable
          J             /pointer to 2nd variable
        NORMAL RETURN

/The format statement is set up with the text command in PAL.
        JMP SKIPFORMAT
FORMPTR,        TEXT /('I=',I5,', J=',I5)/
SKIPFORMAT,     NOP /continue ...
```

The list of DUSR's is given in the following table:

DUSR #	Function	number of args passed to DUSR
1	CTROTST	none - if TTY: CTRL/O then goto DDTG else return
2	MOTORS	none - done through COMMON
3	MVMTR	none - done through COMMON
4	MANUAL	0
5	CLOCK	0
6	IO	(IDEVICE,FILE,EXT,IOPR); value(IO)==>c(AC)
7	WRITE(1,fmt)list	(size of list,fmt ptr, list)
8	READ(1,fmt)list	(size of list,fmt ptr, list)
9	SPOOL OUTPUT	c(AC)==>spooler
10	EXECUTE DDTG cmd string	pointer to string (6-bit)
11	SUBMIT a CCL cmd string to CCL.	pointer to string (6-bit)
12	RETURN TO OS/8	0
13	RETURN TO DDTG	0
14	CONVERT BCD to integer	c(AC)<==BCD-to-integer(c(AC))
15	CONVERT integer to BCD	c(AC)<==integer-to-BCD(c(AC))
16	CONVERT D.P. / float	(integer-D.P., float var)
17	CONVERT float to D.P.	(integer-D.P., float var)
18	CALL DOAUX(IOPR)	IOPR<==c(AC), arguments in COMMON
19	CHAIN to SYS:FILE	file-name pointer

1.14 The RTPP loader - GPPLDR

A GPP loader subroutine GPPLDR is used in DDTG to run GPP programs. DDTG can then be used to access and modify them using a loader input symbol table. This is similar in concept to the use of the MIT "ITS" (Incompatible Time Sharing System) [East69] in loading and running programs in a debugging environment. Assume that a loader file is on the PDP8e disk file system (generated on either the 8e (PAL8, FLAP, FORTRAN II) or by the PDP10 assembler (GEEPER) for the RTPP).

Until GEEPER is built, only ".BN" and ".SV" files will be used. Therefore, the PAL switch is the default. PDP8e ".SV" files may be run in the special segment if they were prevously loaded and saved in the field 7 special segment area. This is discussed in more detail in section 6.

(a) $START (default GPP start address from Load file)
or
$START nnnnnn&PM, nnnn&PDP8E

switches ;NO8E (don't start 8E special segment)
;NOPM (don't start GPP special segment)
;PAL (start the 8e special segment)

(b) $LOAD DSK: A.<GP> (optional extension)

switches ;NEW <defines new starting address>
;MERGE (symbol table with rest of DDTG)
;CLEAR (symbol table before adding)
;NO8E (don't load the 8e special segment)
;RELOCATE <relocation addr. for GPP PM code>
(for merging PM macros,
will also relocate symbols,
cf. section 1.14.1)
;PAL (load ".BN" or ".SV" file
into the special segment))

Note that the A.GP type of file is a special load file to be discussed subsequently.

(c) $SAVE DSK:A.<GP>;<PM limits>;<GR limits>;<8e limits>;
;PAL (save ".BN" file <==special segment)
;SAVSYM (symbol table to be saved)
;PM,<start address - PM>
;PDP8e,<start address - PDP8e>

where: if there are no limit switches save all.

```
<PM limits> = &PM, n1-m1,n2-m2,,nk-mk
<GR limits> = &GR, p1-q1,p2-q2,...,pk-qk
        note: a segment is an address range "n-m" or "p-q".
<8e limits> = &PDP8e, s1-t1,s2-t2,...,sk-tk. (in [0:7777])
```

Note: if the PAL switch is used and a ".BN" extension is specified then it will be save in a absolute binary formated file. If the PAL switch is set and the "."SV" extension is specified then it will be saved in OS8 SAVE file format.

(d) $RUN DSK:A.<GP>

is equivalent to:

```
$LOAD DSK:A.<GP>
$START
```

The GPPLDR subroutine could be used in making other special systems for the RTPP where programs are to be run on the GPP from GPP disk loader files.

1.14.1 Relocatable code

Once only GPP PM code can be used to set up indirect "PUSHJ" procedure table in the GR. This can be used to effect relocatable GPP code. A relocatable segment is assembled starting at location 0000. Then, if there are n procedures, the once only code sets up a transfer vector table in the GR as follows.

```
"OFFSET<==PC SUB #1;
RPROC1<==PROC ADD OFFSET;
RPROC2<==PROC2 ADD OFFSET;
                .
                .
                .
RPROCn<==PROCn ADD OFFSET;
```

To call a relocatable procedure is done by indirectly doing an indirect push jump to general register location RPROCj.

```
"PUSHJ 'RPROCj"
```

to call procedure PROCj. Note that there must not be GR use conflicts which implies the use of a data stack to pass arguments. This method is compatible with the planned GEEPER compiler generated code.

SECTION 2

Loader File Format

Programs for the RTPP are written and compiled on the PDP10 using the GPP assembler [GrosG76]. The GPP assembler produced loader file is discussed here.

The 12-bit binary file residing on the PDP8e disk consists of 3 parts: a header, a data section, and a symbol table area. The header is accessed by all users of the loader file to ascertain the size and location of data and symbols.

The symbol table section is used only with DDTG. It contains symbols which address the corresponding PM, GR, and PDP8e data segments. Note that a PDP8e <file> block is 256 (decimal) 12-bit words while a GPP segment may be any number of words (60-bit PM or 16-bit GR words) less than 4096. The maximum size of the PM and GR memory spaces is 65K each.

The PDP8e "special segment" is to be used to set up the Quantimet and other hardware as well as to act as a mini-monitor for the GPP to post images, acquire images, communicate with the PDP10, etc. This enables running the GPP without a full blown PDP8e monitor for executing GPP procedures. This code may be optionally started (at special segment address 200) when the GPP is started. The PDP8e special segment resides in the 4K field 7 of the PDP8e.

2.1 Load file header

The Header is 256 PDP8e words or 1 PDP8e block.

Header	Data to be loaded (PM)-(GR)-(8e)	Symbol table
1 block	n blocks	m blocks

(a) 0000 - denotes a loader file data file mode
(b) OS8 6-bit Ascii Load filename (A6.A2) - 4 words
(c) OS8 packed date word - 1 word

(d) File length in blocks - 1 word
(e) GPP starting address in PM address space - 2 words
(f) PDP8e starting address in special segment - 1 words
(g) Number of PM segments - 1 word
(h) Number of GR segments - 1 word
(i) Size of 8e special "special code" segment
(j) Relative block where PM segment starts
(k) Relative block where GR segment starts
(l) Relative block where 8e code segment starts - 1 word
(m) Relative block where symbol table starts, 0 if none
    - 1 word
(n) Number of PM/GR/8e symbols - 1 word.
(o) Ascii device name (2-words) and ".SV" file name
    (3 words) and ".SV" extension (1 word) of "DEV:FILE."SV
    which if non-zero indicates the name of a ".SV"
    file to be loaded into the special segment area
    instead of the ".BN" segment of the GEEPER file.
(p) [PM segment 1 start address
    PM segment 1 end address] - 2 words
.
.
.
(q) [GR segment 1 start address
    GR segment 1 end address] - 2 words
.
.
(r) PM segments data - 5 PDP8e words to 1 PM word
(s) GR segments data - 4 PDP8e words to 3 GR words (packed OS8)
(t) 8e segments data PAL10 .BIN format (OS8 packed)
(u) Symbol triples (NAME[1:3],IVAL[1:2],ITYPE[1])
    - 6 words/datum.

2.2 Loader data segments: PM/GR/PDP8e

There are three distinct data segment areas on the data portion of the loader file. These are discussed below. Data is written in a continuous stream of groups of 12 bit binary words (compatible with OS/8 device handlers). There is no checksum used.

(a) PM segment data is written in five 12-bit-8e words/PM word. Then a PM segment with starting address N1, end address M1 has 5(M1-N1+1) 8e words of data. Data is packed MSB to LSB.

(b) GR segment data is written in four 12-bit binary 8e words/ three GR words. Then a GR segment with starting address P1, end address Q1 has (4/3)(Q1-P1+1) 8e words of data.

(c) PDP8e special segment code is 4096 words long for the segment which is optionally included. This mini-monitor code is executed while the GPP program is running. As this code is produced by the PAL10 program on the PDP10, it is in 8e binary loader format thus facilitating overlays etc. DDTG modifies its idle loop before starting it so as to bring it under DDTG's control. If control does get lost, the PDP8e may be restarted by the PDP10 using the remote hardware bootstrap.

SECTION 3

GET/PUT data file space format

Data space files may be generated or used with the $GET and PUT commands. These data files are of a variable format with the exact format described in the first (header block) of the file. The format is a function of the data type. There are five basic types of data:

(a) (x,y) coordinate pairs such as graph-pen data. These may be stored as either packed 8-bit datat relative to the window used to generate it or as 10-bit right justified data in 12-bit PDP8e words.

(b) (x,y,grayscale) triples (not used to date).

(c) Grayscale g(x,y) raster for 256x256 or 1024x1024 image. The largest (FULLRASTER) window is 1024x1024. The default window is the 256x256 size. Its upper left-hand coordinate is defined as the position of the frame and scale window (HPR,VPR).

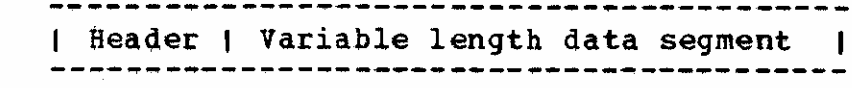
(d) (entrance(y),exit(y)) coordinate pairs of line intercepts from the Mask register.

(e) A non-image data file space type is also used. This is for Quantimet data and is a list of 5-tuples. The 5-tuple is:

[funct. comp. 1, funct. comp. 2, X-acp, Y-acp, det. video]

3.1 Data file space header

A data file consists of a file header followed by data where the data format is described in the header.



The header contains the following information:

12-bit word number	field and function
1	(a) data file space mode (7 to 15 decimal) - 1 word BM=7, MASK=10, QMT=11, GRAFPEN=12, GALSCAN=13, STATE=15.
2	(b) file length in blocks (0 means variable) - 1 word
3:4	(c) number of data (0 means variable) - 2 words
5	(d) number of words/datum (a fraction expressed by the numerator in the left 6-bit byte, and the denominator in the right 6-bit byte - 1 word.
6	(e) data file submode number (see DDTG.DOC table 2) - 1 word
7	(f) data length - 1 word (0=8-bit, 1=10-bit, 2=12 bit data, 3=16-bit, 4=1-bit)
8	(g) information type - 1 word (0=z data, 1=xy data, 2=xyz data (not used), 3=(x(e),x(x)) MASK register data, 4=QMT function computer data, 5=binary mask)
9	(h) data packing mode - 1 word (0 means packed 8-bit, 1 means unpacked in 12-bit words).
10	(i) OS/8 date word when file created - 1 word month - bits 0:3 day - bits 4:8 year (0 to 7) - bits 9:11.
11	(j) horizontal window coordinate before scan - 1 word
12	(k) vertical window coordinate before scan - 1 word
13:16	(l) file name and extension (6-bit Ascii) - 4 words
17:52	(m) 72 character comment (6-bit Ascii) with zero last character - 36 words
53:76	(n) 12-tuple double precision CURRENT position state vector, see MDPDATA[7:8,1:12] in COMMON - 24 words
77	(o) gray value used for filling masks (8-bits lsb default is 255) - 1 word
78	(p) Class key (0 if not used), 1 to 12 if used.
79:80	(q) Time of day in two words packed 4A6
81	(r) Size of image for tsubmode 15 as (2**n)-1

Table 2. Data space file header information

The table below specifies (a)-(h) where appropriate for the 12 data file modes.

Data file mode	(a)	(b)	(c)	(d)	(e)
1 BMi	7	171	65K	2/3	1 (8-BIT PACKED)
2 BMi	7	342	65K	4/3	2 (16-BIT PACKED)
3 MASK	10	8	1024	2	3
4 QMT	11	8 MAX	1018MAX	6	4 (VAR)
5 GRAF-PEN	12	VAR.	VAR.	2	5 (10-BIT VECTOR)
6 GRAF-PEN	12	VAR.	VAR.	2	6 (10-BIT NOVECTOR)
7 GRAF-PEN	12	VAR.	VAR.	4/3	7 (8-BIT VECTOR)
8 GRAF-PEN	12	VAR.	VAR.	4/3	8 (8-BIT NOVECTOR)
9 GALV-SCAN	13	171	65K.	2/3	9 (256 RASTER)
10 GALV-SCAN	13	1736	1048K	2/3	10 (1024 RASTER)
11 GALV-SCAN	13	171	65K.	2/3	11 (256 RAST-MASK)
12 GALV-SCAN	13	1736	1048K	2/3	12 (1024 RAST-MASK)
13 STATE	15	1	12	2	13 (see header)
14 BMASK	16	VAR	VAR	12	14 (in PROC10)
15 IMAGE	17	VAR	VAR	2/3	15 (in PROC10)

Data file mode	(f)	(g)	(h)	
1 BMi	0	0	0	(MEMORY i=[0:7])
2 BMi	3	0	0	(MEMORY i=[0:7])
3 MASK	1	3	1	
4 QMT	1	4	1	
5 GRAF-PEN	1	1	1	(DOUBLE 0 IS EOF)
6 GRAF-PEN	1	1	1	(DOUBLE 0 IS EOF)
7 GRAF-PEN	0	1	0	(DOUBLE 0 IS EOF)
8 GRAF-PEN	0	1	0	(DOUBLE 0 IS EOF)
9 GALV-SCAN	0	0	0	
10 GALV-SCAN	0	0	0	
11 GALV-SCAN	0	0	0	
12 GALV-SCAN	0	0	0	
13 STATE	1	4	1	
14 STATE	4	5	0	
15 STATE	0	0	0	

SECTION 4

DDTG implementation

DDTG is implemented in OS/8 Fortran II using interspersed SABR type coding which allows easy access to hardware registers. The system consists of the DDTG.FT main program and subroutines called by it in a hierarchical tree structure. This section goes into this structure in more detail.

DDTG.FT is the line scanner/finite state parser for DDTG, the RTPP debugger. It processes the TTY: or DSK: character input stream by parsing it into a stack (IPSTK) of symbol table indices to be interpreted. It then interprets this stack. The parser section uses a 128 character jump table as part of its finite state machine (FSM). Lower case letters are mapped to upper case before interpreting. The use of the symbol table is explained in more detail in Section 4.4.

The symbol definitions and interpreter process numbers are defined for the system on a SYS: file "INISYM.DA" which is compiled by INISYM.SV into the 64 block file SYS:CMPSYM.DA. The latter file simulates a 16K PDP8e memory for paging the DDTG symbol table.

The DDTG program works as follows. A command is input from the logical teletype (or $EXECUTE file). Commands are compiled and executed one line at a time into "IPSTK" with pointer "IPTOP". Operator precedence [(*,%) over (+,-) etc.] is performed using a temporary operator stack "IOPSTK" with pointer "IOPTOP".

Characters are first loaded into a line buffer "LINE" with pointer "ITTYP" using internal subroutine "GETLINE". "GETLINE" allows local editing with (rubout, Ctrl/U, Ctrl/C, Ctrl/T, Ctrl/R). Break characters are (line feed, carriage return, =, /, <, >, (and Ctrl/Z and Ctrl/E in DEFINE function editing mode)).

The parser (internal DDTG.FT subroutine "PARSE") then compiles the buffer "LINE(ITTYP)" into a reverse Polish stack "IPSTK(IPTOP)". Entries in the stack are all the indices of symbols in the symbol table. All symbols, numbers, operators, and $operators (and temporaries created during interpretation)

are stored as symbols (See section 4.4 for more details on their definition and use). Characters are parsed by a set of finite state machines (FSM) using a jump table "TABLE" consisting of a FSM to service one or more input characters. The character to be parsed is in variable "ICHAR". (Note: it is an onto mapping (in the algebraic sense) since many characters have the same FSM, e.g. all upper case letters have FSM "letter" etc.). Further testing is performed within each FSM when required to take the state of DDTG into account in the parse.

Furthermore, undefined symbols and numbers are also pushed by the parser onto a garbage collection stack (ITMPSTK) which is used to clean up the symbol table at the end of the interpreter phase.

The interpreter

The interpreter (external subroutine DINTRP.FT) interprets the Polish stack which was generated during the parse by "PARSE" in DDTG.FT. Specific DDTG features are implemented at this point. "DINTRP.FT" scans "IPSTK(IP)" from IP=IPTOP to 0 looking for ITYPE(index) values which are not type I or II process commands or PDP8e IOT's.

PDP8e IOT's are treated as operators in the "non-open" state and as operands in the "opened" state. Type I and II processes, and PDP8e IOT's are pushed onto OPSTK while the search for an operand continues. Note that operator precedence was performed in "PARSE" and already exists here. IP is then decremented until an operand is found at which time the top operator in the IOPSTK is evaluated. Operator processes are responsible for popping the IPSTK.

Errors are noted by an error typeout (residing in
Type I and PDP8e IOT operator processes are located in DINTRP.FT while type II processes are located in IDTYPEII.FT. Note that all external I/O obeys the OCTAL or DECIMAL (BCD is the same as DECIMAL) switch mode MODEN. All internal arithmetic, however, is performed in decimal.

DDTG.FT) consisting of an error number with the rest of the command being ignored (or the entire command if no backup is required). There are two types of errors: fatal (denoted by negative internal error numbers where control goes to OS/8) and non-fatal (denoted by positive internal error numbers where control goes to DDTG's get next command (in the listen loop).) The error numbers are listed in section 5.

Processes

The actual processes used to implement the actions of DDTG are described in five groups of subroutines:

(a) ODTSIM which simulates the ODT like debugging envirnment through data space opening and closing, search, breakpoint insertion and service and continue;

(b) GPPLDR and DUSR which loads, saves and starts special segments and services special segment requests respectively;

(c) GETPUT, DOAUX, and DICMED which implements the $GET and $PUT operations;

(d) MANUAL and MOVESTATE which control the stepping motor envirnment.

(e) Low level primitive functions used throughout the system include IO.FT, IBCD.FT, DPCVRT.FT, OCT.FT and SYMTAB.FT.

.1 Arguments for Special Segments

Whenever any file specification is analyzed for any file related operations, it is put onto stacks in COMMON. These stacks (SDEVICE, ISDEVICE, SFILE, SEXT with pointer IFILTOP) and (ISVARLST with pointer ISVTOP) are available to special segments. Note that when one RUNs a special segment the file and starting address (0 for default file starting address) are in the COMMON variables (DEVICE, IDEVICE, FILE, EXT). Therefore, the IFILTOP and ISVTOP pointers are 1 more than the number of arguments actally desired to be passed to the special segment (i.e. fetch args for I=2:IFIL(ISV)TOP).

4.1 BNF grammar for DDTG

The following BNF grammar is given for DDTG. The semantics of the grammar is given in the rest of DDTG.DOC.

```
<Command>::= <expr><break char>

<expr>::= <arith expr> | <assignment state> | <dspace assign> |
                <get cmd> | <put cmd> | <loader cmd> |
                <control cmd> | <mode>
                <list> | <opcode> | <address> |
                <new symbol definition> | <odt command> |
                <dicomed cmd> | <microstate cmd> |
                <if statement> | <then statement> |
                <goto statement> | <polling cmd> | <eval cmd>

<break char>::= / | < | > | = | <line feed> | <carriage return>

<arith expr>::= <term> | <term>+<term> |<term>-<term> | -<term>
<term>::= <factor> | <term>*<factor> | <term>_%<factor>
<factor>::=  <variable> | <number> | <8eIOT> | <null>

<assignment state>::= <variable>_<expr>
<new symbol definition>::= <new sym triple>\<new symbol>
<dspace assign>::= DSPACE_<data space>

<variable>::=<symbol>

<symbol>::= <symbol><letter> | <symbol><number> | <letter>
<number>::= <number><digit> | <digit>
<statement number>::=<number>

<list>::= <list><delim><expr> | (<list>) | <expr>
<delim>::= ; | , | <space character>

<new sym triple>::=<ival[ 1]><delim><ival[ 2]><delim><itype>
<ival[ 1]>::=<expr>
<ival[ 2]>::=<expr>
<itype>::=<expr> | <data space>
<new symbol>::=<symbol>

<mode>::= BCD | DECIMAL | OCTAL | ECHO | NOECHO | SPOOL |
                NOSPOOL | PROTECT | UNPROTECT | SIXBIT |
                SYMBOLIC | MODES | MODE | DSPACE | LISTSYM |
                SYMPRINT | DELETE | VERSION | STDDET |
                DIGDET | COMMENT | MANUAL | NOMANUAL | SETQMT
```

```
<data space>::= PM | GR | I1 | I2 | I3 | BM | PDP8E | PDPMRI

<data f space>::= BM<octal digit> | MASK | QMT | GRFPEN |
                  GALSCAN | DICMED | STATE

<if statement>::= <if statement><carriage return><then statement> |
                  IF<expr><condition><expr>
<condition>::= GT | LT | EQ | GE | LE
<then statement>::=  THEN<expr>

<goto statement>::= GOTO<statement number>

<dicomed cmd>::= $ERASE | $VIEW | $NOVIEW

<microstate cmd>::= $MOVESTATE (<8 state args>,<m mode>) |
                  $READSTATE (<8 state args>) |
                  $READSTATE | $ZEROSTATE | $SAMPLE (<expr>)
<m mode>::= ABS | REL
<8 state args>::= <t2>,<t1>,<x>,<y>,<focus>,<zoom>,<nd>,<wavel>
<t2>::=<arg>
<t1>::=<arg>
<x>::=<arg>
<y>::=<arg>
<focus>::=<arg>
<zoom>::=<arg>
<nd>::=<arg>
<wavel>::= <arg>
<arg>::= <number> | <variable> | P

<odt command>::= <address expr><odt sym><breakpoint expr>
<odt sym>::= BREAK | CONT | SEARCH | GO
<breakpoint expr>::= <arith expr>
<address expr>::= <arith expr>
<switches>::= <delim><switch types><switches>
<switch types>::= <GET/PUT switches> | <loader switches>
<GET/PUT switches>::= NOVECTOR | FILL | FULLRASTER | USEMSK |
                  USEDIC | ALLBM | USEFILEXY | HGHBYT | USEFRM | USECLASS
<loader switches>::=  NO8E | NOPM | NEW | MERGE | CLEAR |
                  RELOCATE | SAVSYM | PAL

<get cmd>::= <file spec>_<data f space><switches> |
                  GET <file spec><delim><data f space><delim><switches>

<put cmd>::= <data f space>_<file spec><switches> |
                  PUT <file spec><delim><data f space><delim><switches>

<loader cmd>::= <loader file cmd><delim><file spec><switches>
```

```
<control cmd>::= <ctrl file cmd><delim><file spec>


<ctrl file cmds>::= $EXECUTE | $EX | $CHAIN | $DEFINE |
                    $PRINT | $EDIT
<loader file cmds>::= $RUN | $SAVE | $LOAD | $START

<file spec>::= <OS/8 device>:<file name> | <file name>.DA
<OS/8 device>::= SYS | DSK | DSK<letter> | DTA<digit> |
                 MTA0 | MTA1 | LPT | TTY | HSP |
                 <user assigned devices-see OS8 handbook>
<file name>::= <symbol> | <symbol><extension> | FILEGEN<extension>
<extension>::=.<symbol> | null

<polling cmd>::= <load/signal IOT>^<read IOT>,<number>

<eval cmd>::= <expr>=

<8eIOT>::= <load/signal IOT> | <read IOT> | <skip IOT>
<load/signal IOT>::= CLKACK | STQMT | MSTAG | ...
                 MQL | QSTAT | HPL | VPL | LQDT1 | ...
<read IOT>::= MQA | RQSTAT | HPR | VPR | QDAT1 | ...
<skip IOT>::= CLKSKP | QMSKP | IZSKP | ...

<opcode>::= <GPP opcode> | <8e opcode>
<address>::= <addr mod><expr> |
                 <addr mod><expr>&<data space>
<addr mod>::= ' | # | <Null>

<8e opcode>::= <8e IOT/OPR> | <8e mri>+<arith expr>
<8e mri>::= AND8 | TAD | ISZ | DCA | JMS | JMP |
                ANDI8 | TADI | ISZI | DCAI | JMSI | JMPI
                ANDZ8 | TADZ | ISZZ | DCAZ | JMSZ | JMPZ
                ANDIZ8 | TADIZ | ISZIZ | DCAIZ | JMSIZ | JMPIZ
<8e IOT/OPR>::= CLA | MQL | STQMT | QMSKP | QSTAT | ...
<GPP opcode>::= ADD | SUB | MUL | ...
```

4.2 Logical structure of DDTG

The logical control structure of the DDTG parser is outlined below:

1. Initialize symbol tables (once only INISYM.FT).
2. TTY command from user or PDP10 and polling loop of which the TTY is part.
   2.1 GETLINE or file ==>line buffer LINE[1:ITTYP].
   2.2 Parse LINE into Polish stack==>IPSTK(IPTOP).
      2.2.1 Define new symbols using SYMTAB.FT.
   2.3 Interpret stack <==IPSTK.
      2.3.1 Process DDTG functions by calling DINTRP.FT.

Logical structure of DINTRP

The logical control structure of DINTRP is outlined below:

[1] Initialization of the current IPSTK and IOPSTK pointers.
[2] Decrement the current IPSTK pointer IP from IPTOP to 0;
    [2.1] If IPSTK is null then done else goto [4.1];
[3] Look for type I, II processes, or PDP8e IOTs as scan stack.
    [3.1] If one is found, then push it into IOPSTK and continue scan.
[4] If the top of the IPSTK is an operand then do [4.1] else do [2];
    [4.1] Dispatch a type I or II process on top of IOPSTK.
[5] Goto [2] .

4.3 Literal structure of DDTG.FT

The DDTG ain is broken into 5 parts listed below in the order in which they appear in the source code.

(1) [1] Initialization.
(2) [2] Read-parse-eval loop.
(3) Internal worker routines: PARSE, GETLINE, INCHAR, OUT, PUSHP, PUSHOP, OPMOVP.
(4) Internal finite state machines for the parser (cf. 4.5).
(5) Parser character jump table "TABLE".

(6) Error service routine "ERROR".

Literal structure of DINTRP

Fortran labels are allocated as follows:

1. Main interpreter: 1 to 499, and 2000 to 2048

[1] Initialization of the current IPSTK and IOPSTK pointers.

[2] Decrement the current IPSTK pointer from IPTOP to 0 testing when done. (Label 2000)

[2.1] If IPSTK is null then done else goto [4.1];

[2.1.1] Terminate interpretation (label 2010)

[3] Look for type I or II processes as scan stack. If one is found, then push it into IOPSTK and continue scan.

[4] If the top of the IPSTK is an operand then do [4.1] else do [2];

[4.1] Dispatch a type I or II process on top of IOPSTK.

[5] Goto [2].

2. Type I processes: 500 to 999

[I.01] to [I.09] 510 to 599
[I.10] to [I.19] 610 to 699
[I.20] to [I.29] 700 to 799
[I.30] to [I.39] 800 to 899

3. Type II processes: 1000 to 1599 (in IDTYPEII.FT)

[II.01] to [II.09] 1010 to 1099
[II.10] to [II.19] 1110 to 1199
[II.20] to [II.29] 1200 to 1299
[II.30] to [II.39] 1300 to 1399
[II.40] to [II.49] 1400 to 1499
[II.50] to [II.59] 1500 to 1599
[II.60] to [II.69] 1600 to 1699

4. PDP8e IOT execution 1700 to 1799.

5. Internal subroutines: 1800 to 1999.

4.4 Use of symbols in DDTG

All operators and data are encoded internally as symbols and the "indices" of these symbols are manipulated.

subroutine "SYMTAB" allows the creation, accessing, and modification of symbols. A symbol in the symbol table is a triple (name, value, type). A symbol (NAME[1:3]) is 6 characters or less (left justified, right filled with 0's) in length (6-bit Ascii). It has two associated fields: a value field (IVAL[1:2]) and a type field (ITYPE[1]). These are specified below. The symbol table is initially compiled from the Ascii SYS:INISYM.DA file. DDTG common area (DDTCMN.FT) is also initialized during the compilation of the symbol table and is saved in SVDDTG.DA being loaded on DDTG entry and saved on DDTG exit.

The symbol table in DDTG can hold up to 2039 (a prime number) decimal symbols of up to 6 characters each. A folded hashing scheme is used on a prime number hash table which is searched modulo 2039 to handle clashes as I<==(I+HASH(X))MOD 2039. Six PDP8e words are used to store the symbol table entry.

4.4.1 The symbol table ITYPE field

The "ITYPE" field of all symbols on stack "IPSTK" are typed either by subroutine INISYM or the parser "parse" as:

```
10:14 for file structured data types
                10= mask register
                11 = QMT data (func.comp., xy-acp's,det)
                12 = Graf-pen data
                13 = galvanometer scanner
                14 = Dicomed display
                15 = State of RTPP stepping motors and thrs.
 9 for PDP8e memory reference instructions
                PDPMRI=9
 2:8 for symbols of specific data types
                PM=2
                GR=3
                I1=4
                I2=5
                I3=6
                BM=7
                PDP8E=8
 1 for GPP opr device code
                GPPOPR=1
 0 for special DDTG symbols
-1 for DDTG operators
```

```
-2 for DDTG numbers and switches
-3 for DDTG $operators
-4 for DDTG variables
-5 for DDTG functions.
-6 **not used***
-7 for DDTG OS/8 device names (4 char max)
-8 for DDTG for OS/8 file names (6 characters)
-9 for DDTG OS/8 file extensions (2 chars)
```

4.4.2 The symbol table IVAL[1:2] field

The "IVAL" field is used differently for the different types of symbols.

```
ITYPE           IVAL[1:2]
-----           ---------
ITYPE = [10:15], IVAL[1:2] is not used.
ITYPE = 9, IVAL[2] is the memory reference instruction
ITYPE = [2:8], IVAL[1:2] is an address
ITYPE = 1, IVAL[2] is the GPP operator code
ITYPE = 0, IVAL is information for special symbols.
ITYPE = -1, IVAL[1] is the operator precedence
        0 is highest, +n is lowest;
        IVAL[2] bits [6:11] is a type I process pointer
        IVAL[2] bits [0:5] is a condition number
        where applicable.
ITYPE = -2, IVAL[1] is MSW, IVAL[2] is LSW of 2's
              complement number. (Note: NAME[1:3]
              stores ("!!"&IVAL[1:2]).
ITYPE = -3, is used for three types of special
              operators. The IVAL[1] field IS USED
              to specify which subtype is being used.
            IVAL[1]      IVAL[2]
            -------      -------
               0            type II process pointer.
               1            PDP8e "load or pulse IOT"
               2            PDP8e "read IOT"
               3            PDP8e "SKP IOT"
               4            DDTG switches
ITYPE = -4, IVAL[1:2] is same as ITYPE = -2.
ITYPE = -5, IVAL[2] is the active(1)/unactive(0)
              switch.
ITYPE = -6, **not used***
ITYPE = -7, IVAL[1] is the OS/8 device number
ITYPE = -8, IVAL[1:2] is not used.
ITYPE = -9, IVAL[1:2] is not used.
```

4.5 Internal subroutines

The following subsections (4.5.-) list the internal and external subroutines embedded in the Fortran source subroutines. Externally callable (Fortran II/SABR) subroutines embedded in these sources are marked with "*EX*".

4.5.1 Internal DDTG subroutines

```
1. PARSE          - Parse input line into Polish stack "IPSTK"
2. GETLINE        - *EX* Get next input line intop line buffer,
                    Break for (carriage return, line feed,
                          =, /, >, <, Ctrl/C)
                    Edit with (rubout, Ctrl/U, Ctrl/T, Ctrl/R,
                    Ctrl/E, Ctrl/Z).
  2.1 AGETLINE    - alternate entry to GETLINE without (CRLF*) on entry.
  2.2 CTLC        - Control/C service, save state and exit.
3. INCHAR         - *EX* Get next input character from input stream.
4. OUT            - *EX*  Print  next character in the output stream.
5. PUSHP          - Push index(CURSYM,IVAL,ITYPE)==>IPSTK[IPTOP].
6. PUSHOP         - Push index(CURSYM,IVAL,ITYPE)==>IOPSTK[IOPTOP].
7. OPMOVP         - Empty IOPSTK==>IPSTK (copy indices).
######  S t a r t     of  finite state machines #########
8. SYMBOL         - FSM: assemble symbol in "CURSYM".
9. LOWER          - FSM: convert lower case to upper.
10. NUMBER        - FSM: assemble number into "CURSYM".
11. COMMA         - FSM: push argument symbol and test ignore THEN.
12. OPERATOR      - FSM: push opr ==>IPSTK (except +,-,%,*).
13. ARITHOP       - FSM: implement operator precedence
                         using IOPSTK and IPSTK.
14. COMMENT       - FSM: ignore input ICHARi's until next ".
15. DOLLAR        - FSM: set special $operator mode.
16. PERIOD        - FSM: to process current ptr's or file.ext syntax.
17. COLON         - FSM: to "inquire" from the USR (OS/8) the
                    device name of CURSYM and push device number.
18. LFEED         - FSM: to push the symbol "$LFEED" for line feed.
19. CRTRN         - FSM: to push the symbol "$CRTRN" for
                    carriage return.
20. SIXBIT        - FSM: create 6 -bit numbers using "!" as delimiters.
21. LEFTARROW     - FSM: detect GET/PUT use of "_".
```

4.5.2 Internal DINTRP subroutines

1. CTROTST - *EX* test for control/o, terminate interp.
2. BUMPL - test IP > 1, get symbol at The top of IPSTK, decr. IP.
3. CVIVAL - *EX* convert IVAL[1:2] to F.P. # fc based on "MODEN" switch.
4. BINARG - *EX* test if binary arguments exist, then get 2 top args IVAL[1:2] (of IPSTK) into IA[1:2] and IB[1:2], and convert IA to FA, IB to FB.
5. FCSTORE - *EX* convert FC to IVAL[1:2] and store "number" symbol into IPSTK(IP).
6. FILESPEC - *EX* get the device name, device number, file name and extension into COMMON.
7. DOSYMTAB - call SYMTAB(CURSYM,IVAL,ITYPE,INDEX,IDOSYMTAB)

4.5.3 Internal IO subroutines

1. ISETDEV - fetches the input device handler
2. OSETDEV - fetches the output device handler
3. R - read 1 block into IBUF buffer
4. W - write 1 block from JBUF buffer
5. GETC - get next character from IBUF
6. PUTC - put next character into JBUF
7. INNC - *EX* external version of GETC
8. OUTC - *EX* external version of PUTC, error in AC

4.5.4 Internal ODTSIM subroutines

1. ODTOPEN - opens the address "MODED!KURPTR" by putting its value in "KODTM".
2. ODTCLOSE - closes the address "MODED!KURPTR" by depositing the value contained in "KODTM".

4.5.5 Internal GETPUT subroutines

1. DMPBUF - push the ac into the IBUF1[1:512], write
2. LODBUF - load the ac from IBUF4[1:256], read the buffer when empty.
3. GETXYZ - get the next (Z, or XY) argument using using GETC or LODBUF according to KSUBTYPE.
4. DICPNT - test if USEDIC then display (IX,IY,IZ) or 256
5. XYOUT - output either 8-bit (USEFRM on) or 10-bit (off) raster.

4.5.6 Internal SYMTAB subroutines

1. SETADR - set the symbol node ptr from I to NODE.
2. WBLK - write the symbol table page back onto the disk.
3. CBLKOFF - compute (using EARE) IRELBLK, IRELOFF from index I.

4.5.7 Internal IDTYPE subroutines

1. BUMPL - test IP > 1, get symbol at the top of IPSTK, decr. IP.
2. GOCHAIN - *EX* execute the CHAIN sequence with the filename FILE in COMMON

4.5.8 Internal DICMED subroutines

1. DICSND - send (<x or y coord in acc>, z coord in MQ) to the Dicomed.
2. DICOUT - actual Dicomed I/O routine.

4.5.9 Internal DOAUX subroutines

1. GRFPIN - get the Graf-Pen data into (IX,IY).
2. GALPNT - position galvanometer scanner at (IX,IY)

then sample A/D channel 0 ==>IZ.

3. CHKFRM - If USEFRM=1 and NOT((IX,IY) in F&S) then return -1 else 0;
4. SETQMT - initial the QMT, F&S, QPROG, Mask Reg, etc.
5. SAMPLE - sample the A/D channel in ICHAN and put the result in IZ.
6. SETVCTR - given two points (IX1,IY1) and (IX2,IY2), set the starting vector at (IX1,IX2).
7. VECTOR - given two points (IX1,IY1) and (IX2,IY2), specified by "SETVCTR" call, successively return (IX,IY) pairs on a linear approximation between points 1 and 2. Set (IX,IY)=(-1,-1) when done.
8. RQMT - read the QMT srg into IBUF1[1:768] using six words/entry for 128 entries/call. An entry is (FC1H,FC1L,FC2H,FC2L,XACP,YACP). Do a "RSRGI, ZSRGI" before the initial call to RQMT. Keep track of valid data yourself; generally, a STQMT was done previously to doing RQMT calls.
9. WQMT - write QMT srg data into (xacp,yacp) from IBUF1[1:768] using the same 6 word format as for RQMT. Do a max of 128 writes/call do not write if the (x,y) arg =(0,0) or (1777,1777)
10. SETWIN - reads the frame and scale (HP,HS,VP,VS), then converts them to decimal from BCD and returns (IXPOSITION,IHSIZE,IYPOSITION,IYSIZE) IF USEFILEXY=1 then use file header (KX,KY) instead of (HP,VP).
11. WRTHEADER - compute and write out the header block.
12. SWANALYSIS - analyze the switches in the IPSTK.

4.5.10 Internal DUSR subroutines

1. CLOCK - *EX* external clock subroutine
2. GPTR - get the next pointer from the special segment.

4.5.11 Internal MANUAL subroutines

1. SETUP - calls MOTORS to change desire and set active bit
2. MOTORS - *EX* change desired state

3. MVMTR - *EX* change current state to desired state
4. GETBIT - put + bit for selected motor into LINK register
5. REPLACE - restores motor pattern in COMMON at end of MVMTR

4.5.12 Internal MOVESTATE subroutines

1. GETSTK - pop IPSTK(IP) into CURSYM,IVAL, ITYPE.

4.5.13 Internal GPPLDR subroutines

1. GETPAGE - get the next input page for loading .SV files.

4.6 External FORTRAN subroutine files in DDTG

The following list of subroutines are used in DDTG.

1. RDDTG - re-entrant DDTG address for use with chaining
2. IO.FT - General 8 bit Ascii I/O and bloc I/O.
3. (D)SYMTAB.FT - Make, fetch, delete (name,value,type) symbol triples and indices. (D) is for disk simulation version.
4. DPCVRT.FT - D.P. Integer to/from F.P.
5. OCT.FT - D.P. Octal to/from D.P. Integer
6. DINTRP.FT - DDTG interpreter called after parse is done.
7. IDTYPE.FT - function to implement type II interpreter processes
8. ODTSIM.FT - implements ODT type commands
9. GPPLDR.FT - GPP loader
10. GETPUT.FT - implements GET/PUT commands
11. CLOCK.FT - 200 Hz clock.
12. MANUAL.FT - joystick, threshold control service
13. MOTORS.FT - MOVESTATE legal state checker
14. MOVEST.FT - move state dispatcher
15. MVMTR.FT - actual motor state service.
16. DOAUX.FT - auxillary subroutines For GETPUT
17. DUSR.FT - DDTG user service routine

Subroutine (D)INISYM.FT is used for Symbol table initialization and returns to OS/8 when it is done. The (D) version is for the disk simulation version.

4.7 DDTG I/O

Various types of I/O (Ascii and binary byte, and block) methods are required by DDTG. The $EXECUTE <file> command takes input characters from the specified .DA file using Fortran II general (device 4) input and uses them as input to the DDTG interpreter instead of getting input from the TTY:. The output spooler sends character I/O to an open ended output file. The FORTRAN general (device 4) output channel is used for the output spooling. A "/H" switch is specified during DDTG.SV loading so that the logical MTA1: OS/8 device can be used for spooling. Various other types of 8-bit and 12-bit I/O are also done by DDTG and its subroutines. These may require the use of 2 or more I/O channels to be open at the same time.

Data acquisition and data display require a more advanced I/O facility than that offered by Fortran II. For this reason, subroutine IO.FT was written which facilitates mixed 8-bit byte and block I/O as well as random accessing block structured random access devices.

Program (D)INISYM.FT uses Fortran input to read the SYS:INISYM.DA file while subroutine (D)SYMTAB.FT random accesses the SYS:CMPSYM.DA file using the OS/8 system device handler.

4.8 Compiling DDTG

DDTG consists of a set of subroutines which may be compiles separately under OS/8 as follows. The set of compile statements is included in the batch program DDTGCP.BI running on the PDP8e.

```
.R FORT
*DDTG.RL,DDTG.LS_DDTCMN.FT,DDTG.FT

.R FORT
*DINTRP.RL,DINTRP.LS_DDTCMN.FT,DINTRP.F2

.R FORT
*IDTYPE.RL,IDTYPE.LS_DDTCMN.FT,IDTYPE.F2

.R FORT
*IO.RL,IO.LS_IO.FT

.R FORT
*GPPLDR.RL,GPPLDR.LS_DDTCMN.FT,GPPLDR.F2

.R FORT
*GETPUT.RL,GETPUT.LS_DDTCMN.FT,GETPUT.F2

.R FORT
*DOAUX.RL,DOAUX.LS_DDTCMN.FT,DOAUX.F2

.R FORT
*MOVEST.RL,MOVEST.LS_DDTCMN.FT,MOVEST.FT

.R FORT
*ODTSIM.RL,ODTSIM.LS_DDTCMN.FT,ODTSIM.F2
```

```
.R FORT
*DSYMTAB.RL,DSYMTAB.LS_DSYMTAB.FT

.R FORT
*DINISYM.RL,DINISYM.LS_DDTCMN.FT,DINISYM.FT

.R FORT
*DUSR.RL,DUSR.LS_DDTCMN.FT,DUSR.FT

.R FORT
*DICMED.RL,DICMED.LS_DDTCMN.FT,DICMED.FT

.R FORT
*DPCVRT.RL,DPCVRT.LS_DPCVRT.FT

.R FORT
*OCT.RL,OCT.LS_OCT.FT

.R FORT
*IBCD.RL,IBCD.LS_IBCD.FT

.R FORT
*MANUAL.RL,MANUAL.LS_DDTCMN.FT,MANUAL.FT

.R SABR
*MP.RL,MP.LS_MP.SB
```

4.8.1 Building a DDTG.SV core image

DDTG.SV may be built on a PDP8e using the following loader sequence. The OS/8 batch file DDTGBL.BI contains this sequence.

```
.R LOADER
*IO/O/I/H
*DOAUX,IDTYPE,DINTRP,GPPLDR,DSYMTAB,GETPUT,ODTSIM,DICMED,DUSR
*LIB8/L
*MOVEST/7
*MANUAL/7
*DDTG,DSYMTAB,DPCVRT,OCT,IBCD
*LIB8/L
*DDTG.MP/M<
*/M$
.SAVE SYS:DDTG
```

The symbol table compiler program is loaded as follows:

```
.R LOADER
*IO/I/O/H
*DSYMTAB/2
*DINISYM,OCT/I/H
*INISYM.MP/M<$
.SAVE SYS:INISYM
```

To compile the symbol table (before running DDTG) type:

```
.R INISYM
```

To which INISYM will respond with two messages, one at the start of compilation and the other at the end with the number of symbols compiled being printed.

SECTION 5

Creating OS/8 Fortran II special segments

Special segments may be created using the Fortran II language in OS/8 which can access all of the DDTG.SV Fortran subroutines. This is possible because of the fact that the OS/8 LOADER program assigns entry point numbers to external subroutine calls by their occurance during loading. If the first reference to an external subroutine is the same for both DDTG.SV and the special segment then the special segment can be loaded separatedly but can call the same subroutines. The actual linkage is made at runtime by the Fortran runtime system.

To insure that the loading sequence will be the same as for DDTG, a dummy SABR file MP.SB may be loaded with the special segment. MP.SB contains calls to all of the external subroutines which are used in DDTG and must therefore be the first file loaded in the loader sequence for DDTG.SV. That is, replace

```
*IO/I/O/H
```

with

```
*MP.RL/I/O/H/7
*IO/O
```

In the special segment loading sequence do the following:

```
.R LOADER
*MP.RL/I/O/H/7
*MOVEST/7
*MANUAL/7
*SPECIALSEGMENT.RL/7
*/M$     (use this to get the <starting address>)
.SAVE SYS:SPECIAL.SV 70000-77577
```

Then run DDTG and RUN (using the DDTG RUN command) the special segment.

```
.R DDTG
*RUN SYS:SPECIAL.SV
```

or

```
*RUN SYS:SPECIAL.SV
```

5.1 Creating internal subroutines for special segments

As the Fortran II permits up to 64 external subroutines and DDTG.SV uses 64 externanal subroutines, there are no free slots for addtional special external subroutines. This means that additional special segment subroutines should be made internal in the Fortran special segment. The following example shows how this may easily be done.

(1) Put all internal subroutines just before the END statement with a CALL EXIT before the first one.

(2) Call the subroutine with a JMS.

(3) Begin the actual code for the subroutine with the following:

```
S       CPAGE 3 /KEEP THE RETURN PTR AND THE JMP I TOGETHER
S RNAME1,       JMP I NAME1 /RETURN
S NAME1,        0 /ENTRY
                .
                .
                .
S               JMP RNAME1 /EXIT TO RETURN
```

Note that Fortran II will reset the data field to the current field on the return from an internal subroutine call.

5.2 Fetching arguments from the DDTG stack

Since special segments are generally started via a $RUN <filespec> command, it is easy to pass additional arguments to the special segment through the working stack (IPSTK) which has working pointer IP and top pointer IPTOP. The convention is that the first file specification and all switches in the stack are used by the DDTG GPPLDR routine. The indices removed from the IPSTK can be analyzed as to type and value using a (IOPR=3) SYMTAB call. This then is an effective way to pass file and variable names to special segments.

An more effective way of accessing up to 4 additional

file spcifications is via file specification stacks in COMMON. The file specifications are already decoded so that SDEVICE(1:IFILTOP-1) contains the ASCII 6-bit device names, ISDEVICE(1:IFILTOP-1) contains its OS8 device number, SFILE(1:IFILTOP-1) the 6-bt file name, and SEXT(1:IFILTOP-1) its 6-bit extension. The list of variables and numbers is given by its symbol table index so that the user must look it up with a call to SYMTAB with IOPR=3. This list is contained in ISVARLST(2:ISVTOP). ISVARLST(1) is the starting address (if nonzero) of the special segment and so is not used to pass arguments.

SECTION 6

References

Carm74. Carman G, Lemkin P, Lipkin L, Shapiro B, Schultz M, Kaiser P:A real time picture processor for use in biological cell identification - II hardware implementation. J. Hist. Cyto. Vol 22, 1974, 732:740.

DEC73. Digital Equipment Corporation:DEC System 10 - Assembly Language Handbook. Maynard, Mass. (cf. 863:914 on DDT), 1973.

DEC74. Digital Equipment Corporation:OS/8 Handbook. Maynard, Mass. (cf. 1-113:1-122 on ODT), 1974.

East69. Eastlake, D, Greenblatt R, Holloway J, Knight T, Nelson S:ITS 1.5 Reference Manual. MAC AIM 161, July 1969.

GrosG76.Grosfeld G, Lemkin P, Shapiro B:GPP assembler for the RTPP. NCI/IP Technical Report #16. In prep.

Lem74. Lemkin P, Carman G, Lipkin L, Shapiro B, Schultz M, Kaiser P:A real time picture processor for use in biological cell identification - I systems design. J. Hist. Cyto. Vol 22, 1974, 725:731.

Lem76a. Lemkin P, Shapiro, B:PRDL - PRocedural Description Language. NCI/IP Technical Report #15. In prep.

Lem76b. Lemkin P, Carman G, Lipkin L, Shapiro B, Schultz M:The real time picture processor - description and specification. NCI/IP Technical Report #7. In prep.

Lem76c. Lemkin P F:Description of the NCI Biological Image Modelling System - CELMOD. NCI/IP Technical Report #14. In prep.

LipL74. Lipkin L E, Lemkin P F, Carman G:Automated Autoradiographic Grain Counting in Human Determined Context. J. Hist. Cyto. Vol 22, 1974, 755:765.

ShapB76a. Shaprio B:PDP11 message switcher. NCI/IP Technical Report #17. In prep.

ShapB76b. Shapiro B:SLR(1) parser generator. NCI/IP Technical Report #9.

SECTION A

List of error numbers

When an error occurs in any procedure in DDTG, an internal error number is generated and is passed backwards from the error condition to internal subroutine ERROR in DDTG.FT. The error number is passed through the COMMON variable IERRNUM.

ERROR searches file "SYS:DDTGER.DA" for the error number and prints it and its associated error message. DDTG then clears various DDTG switches and restarts at the "*" command level.

ERROR CODE ALLOCATION

```
000:099 - DDTG PARSER
200:299 - ODTSIM ERRORS
300:399 - GPPLDR ERRORS
400:499 - MOVESTATE ERRORS
500-599 - GETPUT ERRORS
600:699 - DOAUX ERRORS
700:799 - DICMED ERRORS
800:899 - DUSR ERRORS
900:999 - MANUAL, MOTORS, MVMTR ERRORS
```

ERROR LIST

```
  0 !!! ILLEGAL ERROR MESSAGE NUMBER !!!
  1 <DIGITS><LETTERS> IS ILLEGAL SYMBOL
  2 UNTERMINATED QUOTE (")
  3 DOLLAR ($) IS NOT FIRST CHAR OF SYMBOL WHERE IT APPEARS.
  4 "FATAL" SPOOLER OUTPUT ERROR.
  5 ILLEGAL OS/8 DEVICE NAME.
  6 ILLEGAL PARSE CHARACTER.
  7 ILLEGAL <FILE> SPECIFICATION.
  8 ILLEGAL UNARY OPERATOR SEQUENCE
  9 TOO MANY DIGITS IN <NUMBER>
 10 ** NOT USED**
 11 SIXBIT CONVERSION ERROR
 12 FATAL: NO SVDDTG.DA FILE ON SYS:
 13 FATAL: NO CMPSYM.DA FILE ON SYS:
 14 PARENTHESIS MIS-MATCH ERROR

101 BINARY OPR DOES NOT HAVE 2 ARGS
```

```
102 FATAL: SYMTAB OVERFLOW DURING INTERPRETATION
103 UNARY OPERATOR ARG ERROR
104 ILLEGAL ASSIGNMENT SYNTAB.
105 ILLEGAL FILE NAME SPECIFICATION SYNTAX
106 FILE LOOPUP FAILED (FILE DOES NOT EXIST)
107 UNDEF SYMBOL - NO OPERATION PERFORMED WITH COMMAND LINE.
108 ILLEGAL DATA SPACE SPECIFICATION
109 EVAL ARG ERROR FOR "=".
110 ILLEGAL "BREAK" SYNTAX
111 ILLEGAL "CONTINUE" SYNTAX
112 ILLEGAL "SEARCH" SYNTAX
113 ILLEGAL DATA SPACE ASSIGNMENT
114 ILLEGAL "GO" RETURN FOR ODTSIM
115 ILLEGAL "TYPE" RETURN FOR ODTSIM
116 ILLEGAL "FUNCTION" RETURN FOR ODTSIM
117 ILLEGAL POLLING LOOP R/W IOT SPECIFICATION
118 ILLEGAL "IF CONDITION" ARGUMENT
119 ILLEGAL "GOTO" ARGUMENT
120 ILLEGAL LEFT SYMBOL IN ASSIGNMENT STATEMENT.

201 ILLEGAL ODTSIM FUNCTION REQUEST
202 ILLEGAL ODTSIM DATA SPACE MODE
203 ILLEGAL BREAK POINT NUMBER
204 BREAKPOINT IN USE
205 ILLEGAL BREAKPOINT DATA SPACE
206 NO BREAKPOINT PRESENT
207 ILLEGAL "SEARCH" DATA SPACE MODE
208 ILLEGAL "GO" DATA SPACE MODE
209 ILLEGAL "CONTINUE" DATA SPACE

301 ILLEGAL GPPLDR FUNCTION REQUEST
302 GPPLDR ERROR ATTEMPTING TO OPEN INPUT FILE
303 GPPLDR ERROR ATTEMPTING TO OPEN OUTPUT FILE
304 GPPLDR GPP LOAD NOT IMPLEMENTED YET
305 GPPLDR GPP START NOT IMPLEMENTED YET
306 GPPLDR GPP RUN NOT IMPLEMENTED YET
307 GPPLDR GPP SAVE NOT IMPLEMENTED YET
308 GPPLDR INTERNAL "OPEN" ERROR
309 GPPLDR PREMATURE END-OF-FILE ON INPUT
310 GPPLDR INTERNAL "START" ERROR
311 GPPLDR SAVE COMMAND "CLOSE" I/O ERROR
312 GPPLDR "READ BLOCK" ERROR
313 GPPLDR "WRITE BLOCK" ERROR
314 GPPLDR ILLEGAL ".SV" FILE

401 ILLEGAL MOVEST FUNCTION REQUEST

501 ILLEGAL GETPUT FUNCTION REQUEST
```

```
502 <NOT USED>
502 ILLEGAL GETPUT SWITCH
503 GETPUT ENTER, WRITE, CLOSE ERROR
504 GETPUT LOOKUP, READ ERROR
505 ILLEGAL READ ON WRITE ONLY DEVICE - $GET IGNORED
506 ILLEGAL WRITE ON READ ONLY DEVICE - $PUT IGNORED
507 DATA FILE SPACE TRANSFER MODE ON $PUT IGNORED
508 DATA FILE SPACE TRANSFER MODE ON $GET IGNORED

601 ILLEGAL DOAUX FUNCTION REQUEST
602 DOAUX ILLEGAL GETPUT/GPPLDR SWITCH
603 DOAUX ILLEGAL VECTOR GENERATED
604 DOAUX HEADER OUTPUT I/O ERROR
605 DOAUX ILLEGAL DATA FILE MODE "MODGP" SWITCH VALUE

701 ILLEGAL DICOMED FUNCTION REQUEST

801 ILLEGAL DUSR FUNCTION REQUEST

901 MANUAL TOGGLE SWITCH NOT HELD LONG ENOUGH
921 MOTORS, ILLEGAL MOTOR #: < 1 OR > 12
922 MOTORS, ILLEGAL OPERATION
923 MOTORS, DESIRE > UPPER LIMIT
924 MOTORS, DESIRE < LOWER LIMIT
```

SECTION B

Table of PDP8e IOT's used in DDTG

PDP8e instructions may be used in various DDTG operations to control the RTPP. The following two tables list RTPP specific input/output transfer instructions (IOT's). The concept of IOT is explained in detail in the DEC "Small Computer Handbook" [DEC74]. The first table lists GPP (General Picture Processor) related IOT's while the second lists IOT's which control the Quantimet and control desk functions.

B.1 List of PDP8e IOTs for the GPP

The following list of GPP operators are discussed in detail in [Lem76b].

$DABTDS	disable DAB trap register
nnnn$DABTRP	set the DAB trap addr/enable
nnnn$DMACA	load the DMA CA register
$DMACLR	clear all DMA channels
nnnn$DMAGO	load the DMA cmd register, start DMA
DMASKP	wait DMA to be done
nnnn$DMAWC	load the DMA WC register
nnnn$EXDMA1	load the DMA peripheral device address high
nnnn$EXDMA2	load the DMA peripheral device address low
nnnn$GETBIN	enable BMs to access binary images
nnnn$GETPIX	enable BMs to access gray scale images
$GPPCLR	clear the GPP
$GPPCONT	continue the GPP
$GPPHLT	halt the GPP
nnnn$GPPLAD	load GPP PC from EXDMA1,2
nnnn$LBMX	load BM XP register
nnnn$LBMY	load BM YP register
$LBM0	(XP,YP)==>BM0
$LBM1	(XP,YP)==>BM1
$LBM2	(XP,YP)==>BM2
$LBM3	(XP,YP)==>BM3
$LBM4	(XP,YP)==>BM4
$LBM5	(XP,YP)==>BM5
$LBM6	(XP,YP)==>BM6

```
$LBM7           (XP,YP)==>BM7
nnnn$LPMPCH     load the GPP high PC<==8e ACC.
nnnn$LPMPCL     load the GPP low PC<==8e ACC.
$PCTDS          disable PC trap register
nnnn$PCTRPH     set the PC high trap addr/enable
nnnn$PCTRPL     set the PC low trap addr/enable
nnnn$POSTPIX    post selected gray scale BM windows
nnnn$POSTBIN    post selected binary BM windows
$RGPPCH         read the GPP high PC==>8e ACC.
$RGPPCL         read the GPP low PC==>8e ACC.
$STATG1         read high 4 bits of GPP STATUS reg.
$STATG2         read low 12 bits of GPP STATUS reg.
nnnn$X8ECTL     load the X8E control word
nnnn$X8ECA      load the X8E current address word
```

B.2 List of PDP8e IOTs for the QMT/control desk RQC

A similar set of commands are implemented for the QMT imterface. Command sequences which would normally use a "skip-done" type of 8e sequence to test when done will automatically invoke such a sequence internal to DDTG after issue of the "GO" type IOT. These IOT's are discussed in more detail in [Lem76b].

```
$ADVSR          advance the SRG one word
$CSRGI          clear the SRGI
nnnn$DETB       load standard detector B
nnnn$DETC       load standard detector C
nnnn$DET1       load 1-D detector threshold 1
nnnn$DET2       load 1-D detector threshold 2
nnnn$DETDIG     load densitometer thresholds
nnnn$DISP1      load control desk display 1
nnnn$DISP2      load control desk display 2
nnnn$EXADR      load external interface channel
                 address (0 to 7)
$EXIN           read external interface channel
nnnn$EXOUT      load external interface channel
$FBW1           read  z,y,x  joystick/focus (bits 3 4 5
                        speed bits 6-7=z, 8-9=y,
                        10-11=x)
$FBW2           read the 12 command keys
$FBW3           read the 12 momentary class keys
$FBW4           read the 12 on/off toggle switches
$FBW5           read digiswitch octal digits 0 1 2 3
$FBW6           read digiswitch octal digits 4 5 6 7
```

```
$FBW7           read digiswitch octal digits 8 9 10 11
$FBW10          read  3 5 position switches TH1, TH2
                      Zoom.   Same  format  as FBW1
$FBW11          read 3 5 position switches freq, intens,
                      spare 1.  Same format as FBW1
$FBW12          read  5 position  switch spare2
                      (speed  bit  3,  motion bits 6
                      7), and momentary execute bit 0,
                      grap-pen tip switch bit 11.
$GETMSK enable   acquiring   a  mask  from  QMT
                DET-video
GRFRI           get next x or y grafpen datum
GRFSKP          skip on grafpen ready
nnnn$HPL        load F&S BCD horizontal position
$HPR            read F&S BCD horizontal position
nnnn$HSL        load F&S BCD horizontal size
$HSR            read F&S BCD horizontal size
nnnn$LDXP       load QMT cursor X register
nnnn$LDYP       load QMT cursor Y register
NNNN$LFBW2      load the 12 command key lights
nnnn$LMASKE     load mask entrance(MSKADR)
nnnn$LMASKX     load mask exit(MSKADR)
nnnn$LQDT1      load QMT right display reg lsb
nnnn$LQDT2      load QMT right display reg middle
nnnn$LQDT3      load QMT right display reg msb
nnnn$LSRGB      load the SRGB input register
nnnn$LGALX      load galvanometer scanner x D/A
nnnn$LGALY      load galvanometer scanner y D/A
nnnn$MSKADR     load the mask line address register
nnnn$MSTAG      load stepping motor word
$QDAT1          read low QMT BCD data==>8e ACC
$QDAT2          read middle QMT BCD data==>8e ACC
$QDAT3          read hi QMT BCD data==>8e ACC
$QMSKP          wait for QMT ready.
nnnn$QPROG1     load QMT program word 1 (see [Lem76b])
nnnn$QPROG2 "   load QMT program word 2 "
nnnn$QPROG3 "   load QMT program word 3 "
nnnn$QPROG4 "   load QMT program word 4 "
nnnn$QPROG5 "   load QMT program word 5 "
nnnn$QPROG6 "   load QMT program word 6 "
nnnn$QPROG7 "   load QMT program word 7 "
nnnn$QPROG8 "   load QMT program word 8 "
nnnn$QSTAT      load QMT status register "
$RFC1H          read the FC1 msb
$RFC1L          read the FC1 lsb
$RFC2H          read the FC2 msb
$RFC2L          read the FC2 lsb
$RMASKE         read mask entrance(MSKADR)
```

$RMASKX	read mask exit(MSKADR)
$RQSTAT	read QMT status register
$RSRGI	read the SRG index register
$RSRGX	read the SRG XACP
$RSRGY	read the SRG YACP
nnnn$SIZEA	load BCD Amender sizer register
nnnn$SIZEC	load BCD Classifier collector limit reg.
nnnn$SIZED	load BCD Standard Comp. sizer register
nnnn$SIZEM	load BCD MS3 Computer sizer register
$SMACP	simulate an ACP
$SMCLK	simulate the QMT clock
$SMHLD	simulate the QMT hold
$SMSYN	simulate the QMT sync
$SMVTG	simulate the QMT vtrig
$STEP	step MSTAG stepping motor word
$STQMT	start the QMT and automatically
nnnn$VPL	load F&S BCD vertical position
$VPR	read F&S BCD vertical position
nnnn$VSL	load F&S BCD vertical size
$VSR	read F&S BCD vertical size
$RPENX	read light pen X register
$RPENY	read light pen Y register
$ZSRGI	zero the SRGI and do IZSKP

SECTION C

Alphabetic list of DDTG commands

In order to facilitate the use of DDTG, several alphabetic lists of commands are given below.

C.1 List of $operator commands

Command	Special forms
$BCD	set teletype I/O to BCD number conversion
$BREAK	nnnn $BREAK m (make breakpoint in current data space) nnnn $BREAK m &<PDP8E or PM> $BREAK m (delete breakpoint wherever it is) $BREAK m= (type out breakpoint) $BREAK= (type all breakpoints)
$CHAIN	$CHAIN <.SV file name>
$COMMENT	toggle the comment mode switch
$CONT	nnnn $CONT m $CONT m
$DECIMAL	set teletype I/O to decimal number conversion
$DEFINE	$DEFINE <.DA function file name>
$DELETE	<list of symbols> $DELETE
$DIGDET	use the control desk thresholds to control Digitizer
$DSPACE	$DSPACE_<data space> $DSPACE=
$ECHO	turn on the teletype echo on command input.
$EDIT	$EDIT <.DA function file name>
$ERASE	erase the dicomed display
$EXECUTE	$EXECUTE <.DA file specification> $EX <.DA file specifiation> (alternate form)
$FILEGEN	
$GET	$GET filespec, data type, switches filespec._data type, switches
$GO	$GO <starting address in current data space> $GO <starting address> &<PDP8E or PM>
$GOTO	$GOTO <command number>
$IF	$IF <expr1><conditional><expr2> THEN <do next command> else skip the next command.
$LISTSYM	<list of data spaces> $LISTSYM

```
                <list of examples of other symbols> $LISTSYM
$LOAD           $LOAD filespec, switches
$MANUAL         enable control desk control of stepping motors
$MODE           $MODES (print DDTG modes)
$MOVESTATE      $MOVESTATE(t2,t1,x,y,f,zm,nd,wv,<cmd>)
                   where: <ai>::=<number> | <variable> | P
                          <cmd>::= REL | ABS
$NOECHO         don't echo teletype command input
$NOMANUAL       disenable control desk control of stepping motors
$NOSPOOL        turn off the teletype spooling
$NOVIEW         turn off the Dicomed view light
$OCTAL          set teletype I/O to octal number conversion
$PRINT          $PRINT <.DA function file name>
$PROTECT        allow DDTG to access only special segment part of PDP8e
$PUT            $PUT data type, filespec, switches
                data type_filespec, switches
$RUN            $RUN filespec, switches
$READSTATE      print the microstate of the RTPP motors and thresholds
                $READSTATE(t2,t1,x,y,f,zm,nd,wv)
$SAMPLE         $SAMPLE <channel number>, returns value
$SAVE           $SAVE filespec, switches
$SEARCH         nnnn $SEARCH
                nnnn&<data space> $SEARCH
$SETQMT         reset QMT etc to standard state
$SIXBIT         toggle switch to show opened loc. data as 6-bit
$SPOOL          turn on the teletype spooler
$START          $START <starting address in current data space>
$STDDET         use the control desk threshold keys to control Std. Det.
$SYMBOLIC       $S (special form)
$SYMPRINT       print the value and type of specified symbols
$UNPROTECT      allow DDTG to access all 32K PDP8e core
$VIEW           turn on the Dicomed view light
$ZEROSTATE      zero the motor and threshold microstate vector
```

<write IOT>^<read IOT>,<index>

C.2 List of unary operators

The unary operators used in DDTG are listed below. Characters not listed are trapped by DDTG when used.

Character	function
+	add operator
-	subtract operator
*	multiply operator
%	divide operator
\	define new symbol
=	eval and print expression
&	temporarily change data space
_	(back arrow) assignment
space	separate symbols
,	separate symbols
;	separate symbols
(	start list
)	end list
[	start subscript
]	end subscript
"	comment terminators
^	(upparrow) polling specification
$	(dollar sign) define $operator
.	evaluates to current location or file extension delim
'	indirect bit for GPP address
#	immediate bit for GPP address
>	open next location
<	open previous location
/	open current location
LF	(line feed) open next location
CR	(carriage return) eval expression and close opened location

Editing characters

The following characters are used in editing command inputs to DDTG.

ctrl/C	save state of DDTG and exit to OS8
ctrl/E	stop adding text during $EDIT
ctrl/O	stop processing and return to DDTG "*" level

ctrl/R	print the cleaned up teletype input line
ctrl/T	toggle DDTG status word debug bit and print status
ctrl/U	erase line currently input from teletype
ctrl/Z	stop adding text when $DEFINE $EDIT
rubout	erase last symbol typed.

C.3 List of data spaces - file data spaces and switches

The four lists which follow sumarize the various command switches and data (file) modes.

Data spaces (used with DSPACE and &)

```
BM
GR
I1
I2
I3
PDP8E
PDPMRI
PM
```

Data file spaces (use with $GET and $PUT)

```
BM0
 .
 .
 .
BM7
DICMED
GALSCAN
GRAFPEN
MASK
QMT
STATE
```

Switches (used with $GET and $PUT)

```
ALLBM
FILL
FULLRASTER
HGHBYT
NOVECTOR
USECLASS
USEDIC
USEFILEXY
USEFRM
USEMSK
```

Switches (used with loader LOAD, RUN, START, SAVE)

```
CLEAR
MERGE
```

```
NEW
NOPM
NO8E
PAL
RELOCATE
SAVSYM
```

SECTION D

DMA data transfers between the PDP8e and GPP/BM

Various data transfers must be set up and carried out between the 8e and the GPP and the PDP8e and BM. The following examples of 8e code illustrate how this may be implemented.

DMA from the PDP8e is distributed from one general DMA card which plugs into the "Master" (RTPP) PDP8e. Several commands are implemented for the DMA channel control which enable the selection and specification of DMA activity. As the DMA channel selected by the PDP8e DMAGO[9:11] only one channel can be active at a time! This section describes the DMA channel.

DMAGO : 6070 DMA instruction to start the specified DMA channel. A command word in the AC is loaded by the DMAGO.

bit	function
0	Output or read = 0. Input or write = 1.
1	Wait for GPP I/O instruction = 0. Direct I/O (forced by the PDP8e) = 1.
3:4	packing mode for BM data 00 8e-words/1 16-bit packed EAE format 01 4 8e-words/6 low 8-bit BM bytes (OS8 packed) 10 4 8e-words/6 high 8-bit BM bytes (OS8 packed) 11 4 8e-words/3 16-bit BM bytes (OS8 packed)
6:8	PDP8e extended current address.
9:11	DMA channel select. 000 BM I/O. 001 X8e I/O. 010 GPP general (GR) I/O. 011 GPP program memory (PM) I/O. 100 GPP microprogram memory (MPM) I/O is allocated (even if not eventually used). 101 spare

110 spare
111 spare

DMASKP : 6071 Skip on DMA channel done.

DMAWC : 6072 Load the DMA PDP8e word count from the PDP8e AC, i.e. binary number of PDP8e words to be transfered. NOTE: DMAWC = 0000 will transfer 4096 words.

DMACA : 6073 Load the DMA PDP8e current address from the PDP8e AC, i.e. address of first transfer.

DMACLR : 6074 Clear the DMA channels.

Each DMA peripheral device (BM, GR, etc.) requires an additional address at which to perform the DMA in the peripheral device address space.

EXDMA1 : High 12 bits of I/O device address.
EXDMA2 : Low 12 bits of I/O device address.