a reference manual for

D D T

"Dunion's Debugging Tool"

a screen-oriented debugging program for
use with the OSS MAC/65 Macro-Assembler
on computers built by Atari, Inc.

Please contact Mr. Dunion or OSS, Inc., at
1173-D Saratoga Sunnyvale Rd.
San Jose, California, 95129
Telephone (408) 446-3099

Rev 1.0

PREFACE

DDT is the original design and product of Mr. James J. Dunion. Versions of DDT have been produced for disk based systems (sold through the Atari Program Exchange), but this version marks the first time DDT has been integrated with an assembler.

We at OSS like to think that it is especially appropriate that Mr. Dunion chose to allow us to link the fastest macro assembler for Atari computers with the most exciting concept in debugging tools. We hope you enjoy this powerful package as much as we have enjoyed preparing it for you.

TRADEMARKS

The following trademarked names may be used in various places within this manual, and credit is hereby given:

DOS XL, BASIC XL, MAC/65, and C/65 are trademarks of
Optimized Systems Software, Inc.

Atari, Atari 400, Atari 800, Atari Home Computers, and
Atari 850 Interface Module are trademarks of
Atari, Inc., Sunnyvale, CA.

Table of Contents

Section 1: An Introduction to DDT

1.1 What DDT Is

The name "DDT" (a software analog to the biological bug killer of the same name?) has been used for many other debug programs on other systems (where it usually stands for "Dynamic Debugging Tool"). We at OSS are proud to offer the best and most "authentic" DDT, "Dunion's Debugging Tool", by Jim Dunion.

DDT has become one of the most popular debugging tools ever invented for use with Atari computers. In this version, OSS and Mr. Dunion have attempted to keep the spirit and flavor of DDT while scaling its size down enough to fit in an OSS SuperCartridge with MAC/65. This combination of MAC/65 and DDT is truly an all-in-one development system for assembly language programmers.

The heart of DDT is its ability to show what is happening inside the computer on a special display screen. This special screen is kept completely separate from your program's screen, whether you are using sophisticated graphics or simply Atari standard character I/O.

In effect, then, DDT tries to be as invisible as possible to your Atari computer's operating system, screen display handlers, keyboard handler. More importantly, though, DDT attempts to perform its tasks without interfering with your program.

This extraordinary separation of debugger and user program is coupled with the ability to easily change and monitor the internal state of "your" machine's environment, so that you can get a much clearer picture of exactly what's going on inside your system and program at any instant.

As with any software-based debugger, there are limitations on speed, instruction and memory tracing, and interrupt processing. All in all, though, DDT comes close to providing you with the best possible debugging environment, probably matched only by hardware logic analyzers costing hundreds of times more.

1.2 An Overview of the Workings of DDT

DDT is separated into four major functional parts: a display generator, a breakpoint handler, an instruction interpreter, and a user command processor.

Generally, when you enter DDT from MAC/65 (via the "DDT" command, of course), you are presented with an arbitrary display of a portion of memory with the values of the 6502 registers (at the time DDT was entered). Naturally, if you intend to debug your own program, you must first tell DDT where it is. You do this via the command processor (but we won't discuss exactly how at this point).

If you are reasonably cautious, you will probably wish to step through your program a line at a time. You can do this thanks to DDT's instruction interpreter.

Once you have a subroutine or set of routines reasonably debugged through the use of single stepping, you will probably wish to execute them without full trace. Or you may wish to allow your program to run up to a certain point before you examine registers, memory locations, etc. DDT's breakpoint handler accomplishes both these tasks.

With a few exceptions, you accomplish all these tasks by using the command processor of DDT. By simple, easy to remember commands, you can ask DDT to interpret your program, show you the contents of memory in either instruction or memory dump formats, change memory or register values, and (in general) control the flow and environment of the program you are debugging.

1.3 An Example of Using DDT and MAC/65

We will present here a short and simple program, written in MAC/65, which we ask you to type in to the MAC/65 editor. We will then assemble and debug this program using DDT. We will not perform the more complex operations of DDT, but we hope that we will give you at least a feel for using DDT and its flexible commands.

NOTE: We assume here that you have read the MAC/65 manual and can use the MAC/65 editor and its commands. For this example, though, we will call out every keystroke to be used with DDT, including [RETURN] keys, unless we note otherwise

To begin, then, boot your DOS (if you are using a disk) and enter the MAC/65 cartridge. To the "EDIT" prompt, type "NUM" and enter the following program:

```
10 ; EQUATES -- FROM 'AMPPING THE ATARI'
20 HPOSP0 =   $D000     ;Hort. POSn, Player 0
30 PCOLR0 =   $02C0     ;Player COLoR 0
40 CHSET  =   $E000     ;addr of std char. set
50 PMBASE =   $D407     ;Player/Missile BASE addr
60 SDMCTL =   $022F     ;Set DMa ConTroL
70 GRACTL =   $D01D     ;GRAphics ConTroL
80 ;
90      *= $3800        ;an arbitrary address
0100 ;  SET UP FOR PM GRAPHICS
0110 ;
0120 SETUP
0130    LDA  # >CHSET   ;we use the char. set
0140    STA  PMBASE     ;...as data for player
0150    LDA  #4*16+4    ;color: hue 4, intensity 4
0160    STA  PCOLR0     ;for our player
0170    LDA  #$2A       ;std playfield,DMA,players
0180    STA  SDMCTL     ;...are all enabled
0190    LDA  #2         ;the bit for players
0200    STA  GRACTL     ;...is turned on
0210 ;
0220    LDX  #100       ;init our hort. pos'n
0230 ;
0240 LOOP
0250    STX  HPOSP0     ;where we want the player
0260    LDY  #10
0270 DELAY
0280    DEY             ;just wait for awhile
0290    BNE  DELAY
0300 ;
0310    INX             ;to next position
0320    JMP  LOOP
0330 ;
0340    .END
```

When you are satisfied that you have entered the program correctly, you might save it to disk or cassette and then assemble it. We used

```
ASM ,#P:
```

to get the listing which appears below. Of course, using the '#P:' requires that you have a printer hooked up to your computer, so you may wish to modify this command to suit your system's set-up (and see your MAC/65 manual for details on how to do so).

Verify that your listing is essentially identical (we have omitted the symbol table listing here).

```
                10 ; EQUATES -- FROM 'MAPPING THE ATARI'
    =D000       20 HPOSP0 =  $D000     ;Hort. POSn, Player 0
    =02C0       30 PCOLR0 =  $02C0     ;Player COLoR 0
    =E000       40 CHSET  =  $E000     ;addr of std char. set
    =D407       50 PMBASE =  $D407     ;Player/Missile BASE addr
    =022F       60 SDMCTL =  $022F     ;Set DMa ConTrol
    =D01D       70 GRACTL =  $D01D     ;GRAphics ConTroL
                80 ;
0000            90       *=  $3800     ;an arbitrary address
               0100 ; SET UP FOR PM GRAPHICS
               0110 ;
3800           0120 SETUP
3800 A9E0      0130    LDA  # >CHSET ;we use the char. set
3802 8D07D4    0140    STA  PMBASE   ;...as data for player
3805 A944      0150    LDA  #4*16+4  ;color: hue 4, intensity 4
3807 8DC002    0160    STA  PCOLR0   ;for our player
380A A92A      0170    LDA  #$2A     ;std playfield,DMA,players
380C 8D2F02    0180    STA  SDMCTL   ;...are all enabled
380F A902      0190    LDA  #2       ;the bit for players
3811 8D1DD0    0200    STA  GRACTL   ;...is turned on
               0210 ;
3814 A264      0220    LDX  #100     ;init our hort. pos'n
               0230 ;
3816           0240 LOOP
3816 8E00D0    0250    STX  HPOSP0   ;where we want the player
3819 A00A      0260    LDY  #10
381B           0270 DELAY
381B 88        0280    DEY           ;just wait for awhile
381C D0FD      0290    BNE  DELAY
               0300 ;
381E E8        0310    INX           ;to next position
381F 4C1638    0320    JMP  LOOP
               0330 ;
3822           0340    .END
```


Presuming that you have typed in and assembled this program correctly, it is time to lead you through the debugging process.

So give MAC/65 the "DDT" command, and you will be presented with a display similar to the one given below (though the screen version will be easier to read than our printed copy, thanks to inverse video, etc.).

```
+--------------------------------+
|    LOC. VAL INSTRUCTION        |
+--------------------------------+
|  DDT (c) 1984 JAMES J. DUNION  |
+--------------------------------+
|   >BED4  A9 LDA # $04          |
|    BED5  04                    |
|    BED6  48 PHA                |
|    BED7  20 JSR  $A50E         |
|    BED8  0E                    |
|    BED9  A5                    |
|    BEDA  68 PLA                |
|    BEDB  38 SEC                |
|    BEDC  E9 SBC # $01          |
|    BEDD  01                    |
+--------------------------------+
|BKP1 BKP2 BKP3 BKP4 NV BDIZC    |
+--------------------------------+
|0000 0000 0000 0000 10110000    |
+--------------------------------+
| PC  A  X  Y  S  ENTER COMMAND  |
+--------------------------------+
|BED4 8D FF 00 FF                |
+--------------------------------+
```

For now, let's not worry about what all that means. Suffice to say, DDT thinks that your program's PC is at location $BED4 and is showing you the code that it finds at that location.

But our assembly placed our main code at location $3800, so let's tell DDT to change what it is displaying. We do that by entering a command (which will be shown under the words "ENTER COMMAND") as follows:

```
* 3800[RETURN]
```

NOTE that we do NOT type in the space between the '*' and the '3'. DDT does that for us.

Now look at the main display window. The '>' symbol should be pointing to location 3800. Do you see your code listed there? If you typed in the program exactly as we specified, and if you started from a "cold" (power-on) machine, you will probably find nothing but a series of 'BRK' instructions being displayed.

What went wrong? Actually, nothing. At this time, you should go back to MAC/65 by typing the DDT command 'Q' (just push the Q key, nothing else). Now type the following:

```
1    .OPT  OBJ
```

This line is necessary if you wish MAC/65 to assemble code and place the resultant object directly in memory. So, once again, you need to assemble your program. You may do so by simply typing

```
ASM
```

as a command to MAC/65. And, when the assembly is finished, you can go to DDT with the DDT command.

This time, after giving the

```
* 3800[RETURN]
```

command, you should see the beginning of your program displayed in DDT's main display window. Compare what you see to either your printed listing or the listing of Figure 1.2 to be sure that all is okay.

At last we are ready to try debugging our little program.

The first thing we will do is single step through the first part of our program. At this time, push the [OPTION] key one time. What happened? Presumably, the '>' is now pointing to location 3802. Also, the value of the PC (displayed under the letters 'PC') should be 3802. Notice expecially that the A-register now contains E0. In other words, we just executed the instruction 'LDA #$E0' which was at location 3800, and DDT is telling us what the new state of the CPU is.

Now push [OPTION] four more times, observing changes to the PC, display window, and A register. If you have done everything the same way we did, the A-register should contain 2A and the PC should be set at 380C. IF NOT, CHECK TO BE SURE YOUR PROGRAM MATCHES OURS!

Now comes the fun part. Push [OPTION] one more time. Did your display change dramatically? Remember, in section 1.1 we said there were a few limitations on display processing, etc.? We have just run into one of these limitations.

With this instruction (a STore A-register into SDMCTL, the system DMA control), we altered the width of the Atari's "playfield". DDT normally uses a narrow display. We requested a "normal" display. DDT accepts our choice and allows the change in display formats.

Surprisingly, DDT continues to function! And, if you are willing to ignore some of the junk on the screen, you can even read and understand most of the display. (Simply ignore the last 8 charactr positions on each line.)

We could "fix" the display (by pushing the [SELECT] button twice), but let us NOT do so at this time. (If we did, we wouldn't be able to see what happens next.)

Push the [OPTION] key four more times. Presto, an Atari "player" stripe full of character shapes appears. Since this is a demo of DDT, not an explanation of the Atari hardware characteristics, we don't want to spend too much time here explaining what has happened, but a very brief explanation will probably help you if now if you are not experienced with Atari hardware. The explanation which follows is given by address(es) from our little program.

3800-3804	By using the built-in character set as player 'data' we eliminate the to make player shapes for this demo.
3805-3809	This is the same as BASIC XL's PMCOLOR 0,4,0 and similar to SETCOLOR 0,4,0
380A-380E	We enable players and use a "standard" width playfield (character display)
380F-3813	This is a "must", to enable the player data registers. Actually, at this time the player is turned on and active. It's simply too far left of the screen to see.
3814-3818	Move the player stripe to horizontal position 100, which is a little left of the middle of the screen.

Now simply hold down the [OPTION] key. Watch the display of the registers. In particular, watch the values for the X and Y registers (displayed under the letters 'X' and 'Y'). Y seems to be decreasing at about one count per second. When it gets to zero, X is incremented and the player is moved right a little bit. Why? Because we stored X in the horizontal position register for our player.

If you continue to hold down [OPTION], the process will continue, albeit very slowly, and the player will move right across the screen. When you are tired of watching this, release the [OPTION] key.

Let's try something new. Push the 'I' key. What happened? Actually, what you are seeing is the same thing you saw when you held down the [OPTION] key, it's just happening much faster. You get to watch the registers changing, the instruction being executed moving (apparently up and down in the DEY loop, but that's an illusion), and the resultant movement of the player. Again, when you are tired of this, push the [BREAK] key.

So now we have seen two different speeds of instruction interpretation. But there is yet a third. First, though, push the [SELECT] key twice to restore DDT's normal display.

Again, enter the command sequence:

```
* 3800[RETURN]
```

And the PC and '›' displays should both again refer to location 3800. Push the [SELECT] key. The MAC/65 screen should reappear, just as you left it. CAUTION: you are NOT back in MAC/65! This simply demonstrates the independent screen display of DDT. Cute, yes?

Now, very carefully, push just the 'I' key. Once again, the player should appear and start moving across the screen. But now it is much, much faster. Why? Simply because DDT knows that it does not need to continually update its display of the registers, instructions, etc. Yet STILL your program is being interpreted!

When you are ready, press [BREAK] and DDT will regain control. For our last experiment, let's enter the DDT command sequence:

```
G 3800[RETURN]
```

Again, remember that DDT puts the space in for you. Do NOT type it in.

What happened? Presumably you have a very messy, smeared player moving impossibly fast across your display. This demonstrates the true speed of assembly language: the TV screen is not fast enough to keep up!

Push [CTRL][ESC] (hold down the [CTRL] key while pushing [ESC]). You should be back in DDT.

One final experiment: use the DDT command sequence:

```
E 381A[RETURN]
```

to move the display pointer '›' to location 381A. Then enter the sequence:

```
D 00[RETURN]
```

which alters the contents of 381A. Finally, again use the command:

```
G 3800[RETURN]
```

And observe the player, in more visible form, moving rapidly across the screen. Believe it or not, this is the slowest we can move the player if we use a simple single register delay loop (the code from 381B to 381D).

And now we are done with our demonstration. You may use [CTRL][ESC] to get back to DDT. Use 'Q' to return to MAC/65. Or simply reboot your system if you are done using DDT at this time.

Section 2: THE DDT SCREEN DISPLAY

The DDT Screen Display shows a user the internal state of the machine. The display screen is divided into several display areas which show different aspects of what is going on inside the computer.

Please refer to Figure 1.1 in the previous section for a rough picture of a typical display. Remember, to view the DDT display simply type the command 'DDT' from the editor of MAC/65.

The display areas are called :

- REGISTER DISPLAY -- Shows the current contents of the 6502 registers
- DISPLAY WINDOW -- A window into memory
- BREAKPOINT TABLE -- Shows the settings of DDT's breakpoint registers
- COMMAND WINDOW -- Where you enter DDT commands from the keyboard

The following sections describe each of these display areas in more detail. However, for a full understanding of the capabilities of these deceptively simple displays, you must read this entire manual. And, of course, you should try using DDT. Only then will you understand how these displays can be used to their best advantage.

2.1 Register Display

The left side of the lowest part of the display screen is used to display the current contents of the 6502 processor registers. Excepting that the status flag register is shown on the right side of the lines next to the bottom, on the same line as the breakpoints.

Whenever DDT is entered, the contents of the processor registers are copied into register shadows which are then displayed. These shadows are used to restore the 6502 registers before control is released back to the program being tested.

In the next to last line of the DDT display, the names of the 6502 registers are displayed. The current user-program values (contents) of these registers are shown (in hexadecimal notation) in the Register Display area directly beneath their names:

PC = Program counter
A = Accumulator
X = X index register
Y = Y index register
S = Stack pointer

Excepting for the PC, the values (contents) shown for these registers are all single byte values, thus displaying two hexadecimal digits. This is, of course, because all registers on the 6502 CPU chip are a single byte in size. The sole exception is the Program Counter (PC), which is 16 bits (two bytes) in size and is displayed with four hexadecimal digits.

Not shown in the basic Register Display area is the processor status register. In order to allow you to more easily view and understand the value of the status register, it is shown in binary form. That is, each bit of the status register's contents is displayed in a special area of the DDT screen.

The legend "NV BDIZC" on the screen indicates that the bit values shown directly under the legend correspond to the various CPU status bits. In particular, the letters stand for (and the bit values are to be interpreted as):

N = Negative flag
V = Overflow flag
B = BRK instruction flag
D = Decimal mode flag
I = Interrupt disable flag
Z = Zero flag
C = Carry bit

The blank in the legend (and the corresponding bit under it) is an unused bit in the 6502 status register and should be ignored.

2.2 Display Window

The display window forms a window into the system memory address space. This window is located in the top portion of the display screen, and occupies most of the screen. The window is set to an arbitrary address upon entry to DDT, but the initial address shown in the window may be changed by several commands (as described in later sections).

This display window may be thought of as having one of two possible filters in front of it.

The Disassembly Filter

The first filter, which is set upon initial entry to DDT is a disassembly filter. A GREATER THAN sign (>) points to what is called the current position.

In the disassembly display, each line from the current position down is shown in a similar format: the hexadecimal address of a location, its contents and then a disassembly readout. Standard 6502 mnemonics are used, with conventional address mode indications.

Note that the NCR 65C02 additional instructions and address modes are supported.

Several features have been added to aid debugging. If a mnemonic is shown in inverse video, it indicates that a breakpoint has been set at that location. In fact, if you look at the actual contents of that location, it will be a 0.

If the mnemonic in inverse video is a BRK instruction, that particular BRK instruction was not placed there by DDT. This would occur, for instance, in looking at memory that contains all zeros.

Secondly, if the instruction is one of the branch instructions, the computed target branch address is shown. An arrow (↑ or ↓) is used to indicate the direction of the conditional branch.

The Hexadecimal Filter

The second filter is a hexadecimal filter. This filter causes the display window to show the hexadecimal value and ATASCII representation of up to 40 memory locations. Again, the > sign indicates the current position.

If the hexadecimal filter is in place, each line after the current position line will start on an even 4 byte boundary.

This means the current position line can have 1 to 4 values on it. The current position line values will always be left justified.

2.3 Breakpoint Table

The Breakpoint table is located just above the register display.

There are four user definable breakpoints (labeled 'BKP1', BKP2', 'BKP3' and 'BKP4' in the display), each of which will be shown with its current setting.

If a register is clear (i.e., not set), then the value shown will be 0000.

If a breakpoint register is set, the value in that register will be the location (address) in memory where DDT has placed a BRK instruction.

2.4 Command Window

The extreme right hand part of the bottom of the screen is devoted to the command window. This is the area that shows the command that a user is typing in.

Often, a DDT command will consist of simply a single keystroke. Since DDT executes commands very quickly, you may never see the key appear in the command window. Be assured, however, that every key you type (other than the [OPTION], [SELECT], and [START] buttons) is echoed in this window.

Note that DDT commands requiring a following value, etc., automatically display a space after the first keystroke you type. This is for ease of understanding only. You do NOT type the space.

Section 3: An Overview of the DDT Commands

The command interpreter allows a user to issue keyboard commands to DDT. You may recall from Section 2 that the command window is shown in the lower right hand portion of the display screen.

Each DDT command requires only a single keystroke. If the key typed is not a valid DDT command, it will be ignored. If a key is a valid command and requires no additional arguments, the command which the key represents is executed immediately. Again, recall from Section 2 that most DDT commands execute so quickly that you may never see the command key echoed in the command window; but it really does go there, however briefly.

Some DDT commands, though, require one or more additional arguments. If you request a DDT command which needs one or more parameters, DDT will wait for you to enter the arguments it needs before proceeding.

SPECIAL NOTE: DDT always puts a space after the command key when it echoes the key in the command window. You do NOT type the space key. DDT places it there automatically.

COMMENT: In addition to the keyboard commands, DDT understands three "pushbutton commands", which are described in Section 5.

3.1 A Summary of the Keyboard Commands

The DDT Keyboard Commands are :

```
B <1,2,3,4>,<addr>..[1]  Breakpoint 1-4 set to given addr
D <hstring>.........[2]  Deposit hex string
E <addr>............[3]  Examine address addr
G <addr>............[4]  Go at address addr
I ..................[5]  Interpretive mode
M <addr><addr><len>.[6]  Move memory
N ..................[7]  Next instruction
Q ..................[8]  Quit, return to MAC XL
R <P,A,X,Y,S>,<val>.[9]  Register selected receives val
S <hstring>.........[10] Search for hex string
W ..................[11] Window filter toggle
↓ ..................[12] Move display window down/higher
↑ ..................[13] Move display window up/lower
* <addr>............[14] Set Program counter
```

In the list above, the numbers in square brackets (e.g., [3]) indicate the subsection number in chapter 4 where a full description of the command may be found.

The abbreviations enclosed in <angle brackets> are described in the LEGEND (in section 3.2), starting on the next page.

3.2 Legend

In the summary of section 3.1, certain abbreviations were enclosed in angle brackets (e.g., <addr>). In this section we explain the meanings and legal range of values for the data these abbreviations represent.

Also, these same abbreviations are used in Section 4, where each DDT command is described in detail.

You may recall that the abbreviations were as follows:
<1,2,3,4> <addr> <hstring> <val> <len> <P,A,X,Y,S>

We explain these abbreviations in two groups and then follow with some comments about delimiters.

3.2.1 Specific Selections: <1,2,3,4> and <P,A,X,Y,S>

When the commands 'B' or 'R' are used, each expects to be followed immediately by a single character. The characters between the angle brackets are the ONLY characters which will be accepted by DDT in each of these cases.

That is, if you type a 'B' as a DDT command, you MUST follow it with a '1', a '2', a '3', or a '4'. Any other characters are illegal.

If you type in the wrong character (e.g., you type 'B4' when you meant to type 'B3'), you may push the delete (back space) key. DDT will back up and delete the offending entry, and you may re-enter it.

See the descriptions of the 'B' and 'R' commands in Section 4 for more details.

3.2.2 Hexadecimal Values: <addr>,<val>,<hstring>,<len>

First, we must note that the abbreviations <addr>, <byte>, <hstring>, and <len> all represent hexadecimal values which you, the user, must type in. When DDT is expecting a hexadecimal value, it ONLY recognizes the characters 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, and F (the traditional hexadecimal 'numeric' characters).

Specifically, DDT expects a certain number of hex digits, as noted in the following list:

<addr>	=	address value, 1 to 4 hexadecimal digits (i.e., 2 bytes)
<byte>	=	a single byte value, 1 or 2 hexadecimal digits
<hstring>	=	a hex string up to 12 digits long (i.e., 6 bytes)
<len>	=	a two byte length specification, must be either 3 or 4 hex digits

Generally, although DDT will accept fewer than the maximum number of digits, it will NOT accept MORE hex digits than it expects. Thus, if the legend <addr> appears in the summary of a DDT command, you will usually find that you will be unable to enter more than 4 characters (each of which must, of course, be a hex digit).

You can however, delete characters, and then enter new characters. Deleting back past the starting point of the value field will result in the previous item in the command being erased.

There are a couple of special cases in the above rules about field sizes, but they will be clearly described in Section 4, where individual commands are detailed.

3.2.3 Delimiters

There are two usages for delimiters.

First, the commands 'B' and 'R' require both a specific selection and a hexadecimal entry. You MUST separate the selection from the hex entry.

You may use either a [SPACE], a [COMMA], or a [RETURN] as a delimiter (separator). However, whichever you choose, DDT always DISPLAYS a comma as the delimiter.

Second, every hexadecimal value must terminated by a delimiter (except see Section 3.3 for the special cases of '*' and '>'). If DDT did not wait for such a delimiter, you would not be able to correct mistakes.

Again, you may use a [SPACE], a [COMMA], or a [RETURN]. Since a hexadecimal value is always the last item in the command entry, your delimiter is NOT displayed in the command window. Instead, the command is immediately executed.

Once a command has been executed, the command window is cleared to make room for your next command.

3.3 Special Characters: '*' and '>'

For input convenience, there are two special characters, '*' and '>'. These are used as shorthand ways of entering addresses (i.e., where the summary above calls for an <addr>).

'*' means the current value of the PC (as you might expect if you are familiar with 6502 assembly language). Generally, when an <addr> is called for, you may type just a single asterisk (*), and DDT will supply the current value of the PC (as displayed in the register display) for you.

Similarly, '>' means the current position of the Examine window pointer (the '>' symbol on the screen). Anytime an address is expected, you may type just a single greater than sign (>), and DDT will supply the address which the Examine window pointer (>) is pointing to.

In the command descriptions in Section 4, special note will be made if either or both of these characters are not legal for a given command.

SPECIAL NOTE: When either of these special characters is used as shorthand for an address, the command is immediately executed. DDT does NOT expect nor wait for a delimiter in this case.

CAUTION: Note that '*' is itself a legitimate DDT command. Do NOT confuse its usage as an address marker with its usage as a command.

Section 4: Command Descriptions

In this section, we present a more detailed description of each of the DDT keyboard commands. For the meaning and legal values of items enclosed in angle brackets (e.g., <addr>), please refer to Section 3.2. For usage of delimiters (shown in this section as commas), see Section 3.3.

The commands are presented in alphabetical order, as presented in the summary table in section 3.1.

4.1 B -- Set or Reset a Breakpoint

Format: B <1,2,3,4>,<addr>

Examples:
```
B 1,4000
B 1,,
B 2,*
B 4,>
```

You use the Breakpoint command to set (or reset) one of DDT's four breakpoint registers to a memory location (presumably an instruction byte) of your choice.

Note that two values (the breakpoint register number, and the breakpoint location) are required for this command. Both fields must be terminated with a delimiter.

To enter the command given in the first example, above, you could type 'B' then '1' then SPACE then '4000' then RETURN. (Remember, though, that all delimiters--SPACE, COMMA and RETURN--are treated identically. Remember, also, that DDT automatically supplies the space following the B. You do not type it in.)

If a value other than a 1,2,3, or 4 is entered for the breakpoint register, it will usually be ignored. If, however, you type in some other valid hexadecimal digit, the command will be terminated when you enter the following delimiter.

When a breakpoint is set, the location you specified shows up in the breakpoint register display under the breakpoint register number you specified.

If an Examine command is issued to look at a location in memory where a breakpoint has been set, a '00' data (instruction code) value will be seen, even though the proper mnemonic is shown in the disassembly.

Also, if a breakpoint is set at an examined location, the mnemonic will be shown in inverse video. This is a special feature of DDT, to make it easier for you to graphically see where a breakpoint is set and how.

If a breakpoint register is already in use when a new breakpoint is requested, the instruction at the old breakpoint location is first restored to its original value.

To clear a breakpoint register and restore the source code, type any delimiter after selecting the desired breakpoint register (e.g. typing 'B' then '1' then COMMA then COMMA will clear breakpoint 1 and restore the source code).

Trying to clear a breakpoint that is not set will not harm anything. Note, however, that trying to set a breakpoint in ROM, in hardware registers, or in non-existant RAM will have unpredictable (and possibly disastrous) results.

SPECIAL NOTE: Remember, you may use '*' and '>' as shorthand notations for the current value of your PC and the display window pointer. Thus you might examine memory until you find a location where you want a breakpoint. Then simply enter the command

B 2,> [RETURN]

(as an example only) to set breakpoint number two at the displayed location.

COMMENTARY: Physically, a '00' value (a BRK instruction) is stored in memory at the requested location. When DDT performs a disassembly and encounters a BRK instruction, it searches its breakpoint table to see if it had set that particular BRK. If so, it recovers the instruction for the disassembly but displays the mnemonic in inverse video.

4.2 D -- Deposit value(s) in memory

Format: D <hstring>

Examples: D 0
D 313233343536
D 1234

The Deposit command is used to place one through six bytes in memory.

A string of hexadecimal values (up to 12 characters, 6 hex bytes) may be entered. The values entered will be placed in successive locations starting at the current position indicated in the display window (i.e., the address pointed to by the '>'), replacing whatever was there.

The input string is decoded two characters per hex byte at a time. If there is an odd character left at the end, it will be interpreted as the low order nibble of a hex value.

For example, entering a string of 01AAB0 will result in three bytes (01, AA, and B0) being placed in memory. However, entering 01AAB will result in 01, AA, and 0B being deposited.

Note that depositing a byte or a series of bytes will NOT move the display window. This must be done with the examine or the move window up or down commands.

SPECIAL FEATURE !!

DDT is able to switch screens by saving 13 locations the operating system uses in managing the system graphics. Thus, before each value is deposited, it is examined to see if it should be deposited to these graphics locations. If so, the value is placed instead in an internal save table. Thus, for example, you can deposit values directly to the color shadow registers and affect the color of the user screen and not the DDT screen.

See Section 8.5 for a list of the locations saved in this fashion.

4.3 E -- Examine memory

Format: E <addr>

Examples: E 5000
E *
E 0

The Examine command is used to set the display window to view an area of memory. The extreme left hand edge of the display window has a GREATER THAN sign (>) in the 3rd row. This points to what we refer to as the "current position" in the display window.

Unless you have used the '↓' or '↑' commands, the current position will be the address entered via the last 'E' command.

Note that the 'E' command does NOT change the state of the display window filter, nor will it affect which instruction will next be executed by a single step command.

Since you may specify any arbitrary address as the location to be Examined, and (if you are using the disassembly filter) since you may accidentally disassemble a nonsense intruction byte, we recommend one or more of the following:

1. Examine only locations known to contain valid instruction bytes. Refer to a printer listing to be sure you are doing so.

2. After using 'E', move the display window up (lower in memory) a few bytes and then back down (via the '↑' and '↓' commands), to ensure that you are displaying instructions which are on true instruction boundaries.

3. Examine a few bytes ahead of where you really want to be. Then move down (via the '↓' command) to the proper position.

(See also the SPECIAL NOTE in Section 4.2.)

4.4 G -- Go to a Program at a Given Address

Format: G <addr>

Examples: G 5000
G *
G >

The Go command is used to begin execution of your program at a specific location in memory.

Before control is transferred to this location, several actions take place:

1. All registers are updated based upon the current contents of the displayed registers.

2. The 13 locations saved for the graphics display (see Section 4.2, above, and Section 8.5) are restored, thus restoring your display and removing DDT's display from the screen.

3. Vertical Blank Interrupts and Display List Interrupts are BOTH enabled.

Obviously, since Going to your program can be dangerous (e.g., your program may wipe out all of memory, attempt to illegal I/O, or other miscellaneous nasties). We therefore urge caution on your part (including, at the least, saving your latest version of your program to disk or cassette) before using this command.

For all intents and purposes, once you issue a Go command your program has complete control of the Atari computer. There are two methods of returning to DDT: (1) If your program executes a BRK instruction (a zero instruction byte), DDT is entered at its breakpoint entry (see Section 7.3). (2) If you push [CTRL][ESC] (hold down the [CTRL] key while hitting [ESC]), DDT is entered at its "flash" entry point (see Section 7.2).

Method 1 is the most common method and is commonly used when debugging. Method 2 is an emergency method, reserved for when your program starts looping and nothing else will get you out.

Breakpoints are discussed in some detail in Section 6. The "Flash" entry point to DDT is discussed in Section 7.2.

NOTE: The special command sequence 'G *' is exactly equivalent to pushing the [START] button. See Section 5.1 for usage of the [START] button.

4.5 I -- Interpretive Mode

Format: I

Example: I

The Interpretive Mode command is used to place DDT in an automatic single step mode.

Interpretive mode will run with either the user screen or the DDT screen being shown, but you pay a severe time penalty for selecting the DDT screen. After each instruction is interpreted, the screen display is updated if the DDT screen is turned on. The display window is automatically placed in the disassembly mode, and all registers are displayed along with the updated disassembly.

Interpretive mode runs much faster if the user screen is selected, because DDT does not have to update it's screen if it is not active. See Section 5.2 for information on how to enable and disable your display screen when using DDT.

Pressing the BREAK key halts the interpretive mode. Encountering and attempting to execute a BRK instruction halts the interpretive mode.

COMMENTARY: When in interpretive mode, DDT attempts to execute your program as true to form as possible. To this end, DDT moves the instruction pointed to by your PC to a special working area and executes it at that location. Although, if the instruction is one which transfers control (e.g., JMP, JSR, BEQ, etc.), DDT truly "interprets" it.

Also, before DDT executes each instruction, it restores all your registers to the values shown in the register display. After executing (or interpreting) the instruction, DDT restores the proper register values in the register display.

SPECIAL NOTE

Because of the way interpretive mode works, you MAY interpret through ROM-based code. You should NOT, however, attempt to interpret any real-time I/O code (whether in ROM or not), including disk and other serial I/O.

4.6 M -- Move memory

Format: M <addr><addr><len>

Examples: M E000600000400
M 6000600010040

The Move memory command simply does what its name implies: it moves one or more bytes of memory from one location to another.

This command requires a somewhat special format for its values. Specifically, all three values (both <addr>'s and the <len>) MUST be given, but you are NOT allowed to put ANY delimiter(s) (including spaces) between the values.

Both the <addr> values MUST be specified with EXACTLY four hexadecimal digits (using leading zeroes if needed).

The <len> may be any number from 0001 to FFFF (though disastrous results will obviously occur if you try to move all--or even major significant portions--of memory), but even <len> must be specified with three or four hexadecimal digits.

The first <addr> given is assumed to be the source or "from" address. The second <addr> is thus the destination or "to" address. And, of course, the <len> specifies the number of bytes to move.

Thus, the first example shown above will move $0400 (1024 decimal) bytes from memory location $E000 (through $E3FF--the main character set area of ROM) to memory location $6000 (through $63FF).

DDT does NOT check for possibility of overlapping "from" and "to" memory areas before it does the move, so an attempt to use a Move as in the second example above may or may not work the way you expect it to.

4.7 N -- Next

Format: N

Example: N

The Next command is really a shorthand method of program tracing which combines some of the best features of breakpoints with the ease of interpretive mode.

Using the Next command is equivalent to visually examining the disassembly display, determining the address of the next instruction (after the one the '*' is pointing to), setting a breakpoint at that address, and (finally) executing a 'G *' command (or [START] pushbutton command--see Section 5.1).

Most of the time, then, using N is equivalent to interpreting a single instruction (as may be done via the [OPTION] button--see Section 5.3). However, there are several important differences:

1. The Next command uses its own internal breakpoint and places it after the next instruction to be executed. This internal breakpoint is never displayed.

2. The user's screen is restored (as with the Go command, Section 4.4, above) while the instruction is being executed.

3. The instruction is truly executed, not interpreted, so you may not use 'N' when your PC points to ROM code.

4. If the instruction being pointed to by your PC (the '*') is a JSR, then the entire subroutine will be executed before DDT regains control! This allows you to execute ROM code or real-time I/O code at full processor speed and yet view the results immediately after the called routine finishes.

   CAUTION: If your subroutine performs an error exit and does not "properly" return (presumably via an RTS instruction) to the calling program, the breakpoint set by 'N' may never be executed.

5. If the instruction being pointed to by your PC is a JMP or branch instruction, you should usually NOT use the 'N' command, since the program may never reach the point where the internal breakpoint has been set.

4.8 Q -- Quit DDT, Reenter MAC/65

Format: Q

Example: Q

There is nothing fancy about this command. It is simply a means of exiting from DDT back to MAC/65.

Before transferring control to MAC/65, DDT restores MAC/65's zero page locations and its critical page 4 locations (as described in Section 8.1).

DDT also removes all its own breakpoints from user code before Quitting and "unhooks" its Flash entry point from the system keyboard routine (see Section 7.2 and 8.2).

Upon re-entry to DDT (via MAC/65's "DDT" command), the user should restore any critical breakpoints by hand.

4.9 R -- Register Modify Command

Format: R <P,A,X,Y,S>,<val>

Examples: R A,00
R X,FF
R P,01

The Register command is used to modify the contents of any of the 6502's registers except the PC.

After typing 'R', only a 'P','A','X','Y', or 'S' will be allowed. Any other character will be ignored. No other character other than [DELETE] will be allowed until a delimiter is typed.

'P' indicates the processor status register (which is displayed in binary form under the "NV BDIZC"). 'A', 'X', and 'Y' are the normal 6502 registers of the same names. 'S' represents the value of the stack pointer.

After entering the register designator , only two hex digits (i.e. one byte) will be accepted. Note that this command requires two separate values and two separate delimiters.

WARNING! Indiscriminate use of this command to change the stack value (the 'S' register) could make it impossible for DDT to continue to function without being reset.

4.10 S -- Search for a String of Bytes

Format: S <hstring>

Examples: S 31
S 5F5F
S 8D0003

The Search command is used to locate a specific sequence of bytes in memory.

You may enter a hex string of up to 12 characters which will be interpreted as up to 6 bytes. DDT will search for the string you specify, starting from the current position (as indicated by the '>' in the display window) upwards (increasing addresses) through memory.

If the search is successful (the sequence of bytes is found), the display window will be repositioned (and the '>' will point to the first byte of the found sequence). If it is unsuccessful, the command window will simply be cleared for the next command, and the display window will not move.

If no value is entered after the 'S' (i.e. just a delimiter is typed), the previous search string will be used. This allows for easily finding multiple occurences of the search string.

The three examples given above might be interpreted as follows:

```
S 31  -- find a '1' character
S 5F5F   -- find a pair of question marks ('??')
S 8D0003 -- find a 'STA $0300' instruction
```

4.11 W -- Window Change Command

Format: W

Example: W

The Window command is used to change the "filter" over the display window.

You will recall from Section 2.% that there are two different "filters" available to you: a disassembly filter and a hexadecimal filter.

The 'W' command simply toggles between the two.

Note that certain commands will automatically change the filter to their "desired" state. You may use the 'W' command to change the filter back to the one you wanted if your choice does not correspond to DDT's.

4.12 ↓ -- Move Display Window Down (Higher in Memory)

Format: ↓

Example: ↓

The Move Window Down command is used to change the memory being displayed in the display window.

Specifically, the '>' pointer will be changed to point to a location higher in memory. How far the window and pointer are moved depend on which filter (hexadecimal or disassembly) is in place at the time the key is pushed.

If the hexadecimal filter is in place, pushing the '↓' key will move the window down (higher in memory) by one byte.

If the disassembly filter is in place, pushing the '↓' key will move the window down (higher in memory) by one full instruction (which may be one, two, or three bytes).

SPECIAL NOTE: You should NOT hold down the CTRL (control) key when using this command. DDT recognizes '=' as the 'down arrow key' even without CTRL pressed.

ALSO NOTE: Auto Repeat on the keyboard IS active, so that continuing to press the '↓' key will continue to move the window down.

4.13 ↑ -- Move Display Window Up (Lower in Memory)

Format: ↑

Example: ↑

The Move Window Up command is used to change the memory being displayed in the display window.

Specifically, the '>' pointer will be changed to point to a location one byte lower in memory.

Since an instruction could be 1,2 or 3 bytes long, you must be careful to watch and ensure that you remain on instruction boundaries if the disassembly filter is in place.

SPECIAL NOTE: As with the '↓' key (section 4.12, above), you should NOT use the CTRL key to select '↑' and auto repeat IS active for '↑'.

4.14 * -- Set Program Counter

Format: * <addr>

Examples:
```
* 5000
* >
*
```

The command is used to set the program counter.

After you enter the '*' command, DDT expects you to enter an address which will become the new PC contents.

After changing the PC, you may use the 'I' or 'N' commands or the [OPTION] or [START] buttons to begin or continue program execution (or interpretation) at the new location shown in the PC portion of the register display.

DDT always selects the disassembly filter after executing the '*' command and always sets the display window pointer (>) to the same address as the PC.

Note that you may type '*>' as a shorthand notation to set the PC to the address currently being shown in the display window (as indicated by the '>' pointer).

Note also that you may simply type '*' followed by [RETURN] to force the display filter to hexadecimal and force the display window pointer equal to the PC. This can be thought of as a shorthand notation for 'E*' (see Section 4.3) possibly followed by 'W' (see Section 4.11).

Section 5: Push Button Controls

The three ATARI console push buttons are used by DDT for useful and special operations. In many ways, you may think of these butons as extensions to the commands given in Section 4.

Each console button has a unique use, which is described below.

5.1 The START Button

A press of the START button is usually indicated in this manual by the notation [START].

[START] is used to continue code execution at the location indicated by the PC register.

Your screen display is restored and all 6502 registers are updated with the current displayed contents before control is transferred.

Pushing [START] is functionally equivalent to executing the command sequence 'G*', and we suggest reading Section 4.4 for more information on the Go command.

5.2 The SELECT Button

A press of the SELECT button is usually indicated in this manual by the notation [SELECT].

[SELECT] is used to toggle back and forth between the DDT screen and whatever screen dynamics were active before DDT was called and/or reentered (e.g., via a breakpoint).

An attempt has been made to allow for most alternative display features such as mixed Display lists, VBLANK routines, alternative character sets, display list interrupts, playfield size changes, and player-missiles. Thus, whenever DDT is entered or reentered, the locations necessary to restore these features are "remembered" by DDT before DDT puts its own display on the screen. When you execute your program (via the 'G' command, the [START] button, or the 'N' command), DDT restores your screen display as well as it can (and it usually does pretty well).

Generally, then, [SELECT] has only two primary purposes:

1. When you simply wish to look at your display screen momentarily.

2. When you wish to interpret your program (via the 'I' command or the [OPTION] button) while keeping your display active instead of DDT's.

5.3 The OPTION Button

A press of the OPTION button is usually indicated in this manual by the notation [OPTION].

[OPTION] is used to "single step" the processor through your program.

This causes the disassembly filter to be turned on, but will not automatically toggle the display screen. Holding down the OPTION button will continue to single step, at the rate of approximately two instructions per second.

Excepting for the fact that only a single instruction is executed before a pause is made, the [OPTION] button "command" works identically with the 'I' (Interpretive Mode) capability. Therefore, see Section 4.5 for more details on interpreting code via DDT.

Note that you may NOT interpret a BRK instruction. The interpreter will, for all intents and purposes, halt when it encounters a BRK.

Section 6: Breakpoints and Breakpoint Processing

One of the most common debugging techniques is to make use of a breakpoint.

This manual contains much additional information on breakpoints, so we refer you also to Sections 7.3, 4.1, 4.7, and 8.4. This Section will attempt to provide an overview on breakpoints as well as suggested uses for them.

The fundamental mechanism of a breakpoint is fairly simple:

1. When a running program encounters a 'BRK' instruction (a zero byte), the 6502 CPU simulates an interrupt (an IRQ, not an NMI, except that the 'SEI' instruction can NOT disable 'BRK' interrupts).

2. The only real difference between a true IRQ and a BRK-simulated interrupt is that a BRK causes the 'B' bit (bit 4, $10) to be set upon entry to the interrupt handler.

3. When the BRK-simulated interrupt occurs, Atari's OS uses the 'B' bit to recognize the fact and transfers control, via a RAM vector, to DDT.

4. DDT's breakpoint entry simply saves all the user's registers (A,X,Y,Processor status, Stack, and the Program Counter). It then sets the display window pointer (>) equal to the user's PC, selects the disassembly filter, saves the usual graphics information (see Section 4.2 and 8.5), and presents you with the typical DDT screen display.

After a breakpoint has been encountered, and control has been transferred to DDT, there are several ways to leave DDT. The 'N' command (Section 4.7) will set a breakpoint at the next location and then continue code execution, [START] (section 5.1) simply continues code execution. 'G' (section 4.4) can be used to transfer control to another location.

There are three ways to set a BRK instruction and thereby allow a breakpoint to happen.

1. You can use the 'B' command (as described in Section 4.1) to set up to four special DDT breakpoints. There are two advantages to this method: (a) DDT remembers the instruction which was at the location before you set the breakpoint, so when you reset or remove the breakpoint DDT can automatically restore the instruction for you. (b) The disassembly display shows your original instruction in inverse video, as a convenient reminder.

2. You can actually store a zero byte (a BRK instruction) in your code. You can do this either with the 'D' (Deposit) command or by actually assembling a BRK in you source code.

3. You can use the 'N' command (Section 4.7), which automatically sets a BRK instruction in the byte which follows the current instruction. Again, as with the 'B' command, DDT remembers your original instruction and restores it without effort on your part. Note that you will never see the BRK placed by 'N', as it is automatically removed as soon as DDT recovers control.

The best use of multiple breakpoints is to set one at every path in your program where you do NOT expect to or want to go (execute). That way, if your program takes a wrong turn, DDT will alert you by saying, "Hey! How'd we get to this breakpoint?"

Also, of course, you will normally step through your code a little at a time, setting a breakpoint a little farther ahead each time. For this use, we recommend reserving a single breakpoint register (usually number 1). Use the other registers (2 through 4) for the "side" or unexpected paths mentioned in the previous paragraph.

When one of the breakpoints is encountered in intrepretive mode, it will halt the intrepretive mode at that point.

Section 7: DDT Entry Points

There are four ways of entering or reentering DDT:

MAIN ENTRY
FLASH ENTRY
BREAKPOINT ENTRY
RESET ENTRY

Each is described separately below.

Sometimes, it will seem that the computer has locked up and none of the Entry methods described below will work. Generally, this is because something has gone wrong in the program you are debugging, and it has modified certain critical memory locations.

Disabling interrupts (executing an 'SEI' instruction) and/or modifying the interrupt vectors of Atari's OS are particularly insidious ways of destroying DDT's access to the system. And accidentally using the Move command incorrectly can obviously wipe out wholesale hunks of memory.

These are obviously only some of the ways to effectively disable DDT, but we would hope the most users will not encounter any of them. It is usually only the more sophisticated and complicated programs which will be altering locations which DDT is sensitive to.

7.1 Main Entry to DDT

When you give MAC/65's editor the "DDT" command, DDT is entered at what we call its Main Entry point.

Section 8.1 describes in some detail the process DDT goes through when it is entered. In particular, DDT saves the state of MAC/65 so that you do not lose your source code.

See also Section 8.1.

7.2 "Flash" Entry to DDT

This entry point is provided to allow immediate reentry to DDT regardless of most other circumstances.

When DDT is called, the operating system code that looks at the keyboard is modified so that it looks for a special character first, before handling normal keyboard input.

The special character looked for is one which is unused by normal Atari operations: [CTRL] [ESC]

In other words, to reenter DDT when your program is running, simply press both the Control and the Escape keys at the same time.

When DDT's modified keyboard handler finds the [CTRL][ESC] character, DDT is entered immediately through the FLASH ENTRY point (which is essentially equivalent to encountering a breakpoint).

Using the 'N' command or pressing START will return control to whereever the processor was at when the DDT special character was typed.

For more information on the Flash entry mechanism, and some warnings about how you may inadvertantly make it inoperative, see section 8.2.

7.3 Breakpoint Entry to DDT

Breakpoint entries are the most common way of entering DDT.

Once DDT has been entered via the Main entry, DDT's breakpoint handler is set up. Thereafter, anytime your program (or, for that matter, any program) attempts to execute a BRK instruction (a zero byte), DDT is entered at its Breakpoint Entry.

For more information on the use and characteristics of breakpoints, see Sections 6 and 4.1.

7.4 RESET Entry to DDT

If DDT was active before you executed your program (e.g., via the 'G' or 'N' commands or the [START] button), then pushing the [RESET] button should return control to DDT.

Obviously, if your program has scrambled enough locations which are vital to DDT and/or the Atari OS, then the RESET handling may never have a chance to occur.

Section 8: Technical Details of DDT

8.1 Interaction with MAC/65

DDT is designed to be compatible with MAC/65 so that you can easily go from editing to assembling to debugging and so forth.

Specifically, you enter DDT via the 'DDT' command from the MAC/65 editor. At that point, DDT saves certain memory areas which are critical to MAC/65's functioning in memory reserved by MAC/65 (and pointed to by MAC's 'lomem' pointer--the first value given in the response to a 'SIZE' command in MAC).

When you use the 'Q' command to exit DDT and reenter MAC/65, DDT restores the critical memory areas. If, during the course of your debugging session with DDT, you have not changed any of the memory bounded by the low and high values given in response to MAC's 'SIZE' command, you will find your source code (if any) intact and ready to edit and/or (re)assemble.

MAC/65 and DDT cooperate in another way: when you push the [RESET] button on the Atari keyboard (hopefully, only as a last gasp desparate measure), MAC/65 attempts to determine whether DDT or MAC had control when the button was pushed. If DDT had control, MAC automatically and immediately reenters DDT at a special RESET entry point.

8.2 Keyboard Scanner

During DDT initialization the system keyboard vector is redirected to a preprocessor which checks for the DDT FLASH ENTRY special character ([CTRL] [ESC]). If this character is seen, control transfers to the FLASH ENTRY point, otherwise control passes to the normal keyboard processing routine.

Note that keyboard interrupts must be enabled. If your program alters or disables the keyboard interrupt (or its vector), DDT will not be able to regain control. You may or may not be able to push [RESET] to reenter DDT.

Not that this implies that the 'SEI' instruction will also disable the DDT keyboard scanner. This is somewhat important, but since 'SEI' disables all keyboard activity we would hope it is an instruction you will use with care.

8.3 DDT's use of System Resources (RAM and ROM)

DDT itself occupies a portion of the MAC/65 cartridge space. When it is called, the upper half of page zero and certain locations in page four ($400-$4FF, but not all of this range) is saved for later use by MAC/65 (see 8.1, above).

While DDT is running, then, locations $80 through $AF are used by DDT and should be avoided by user programs. Otherwise, the locations in the upper part of page zero may be used.

Also, DDT takes the RAM area from $3FD through $57F for its display screen, breakpoint registers, etc. Since the cassette buffer occupies $3FD to $47F, this implies that you can NOT do cassette I/O from within DDT (though you may load a tape from MAC/65 before entering DDT or save to a tape after exiting).

Remember, also, that MAC/65 is NOT capable of assembling directly into page six ($600 through $6FF). You may, however, assemble into other (higher and safer) memory with an offset (see the MAC/65 manual, Section 8.88) and then use DDT's Move command to place the resultant code in page six.

8.4 Things to Watch Out For

There are a few areas where you have to be careful in using the DDT cartridge. In general, these occur when you are single stepping or running interpretively.

If the code being interpreted alters the display list or does direct access to ANTIC or CTIA/GTIA, then you might end up with a scrambled screen. Usually this is non fatal, just distracting. (See our example program and debug session in Section 1 for an instance of just this occurrence.)

To restore the normal DDT screen, press the BREAK key to halt the interpretive mode, then press SELECT twice (though doing so may turn off any players, etc., which you had made active).

Trying to do I/O from disk or any other real time activity in either interpretive mode or single step mode will almost certainly not work.

You should set up breakpoints around the real-time code so that this type of I/O is done in real time. For example, try using the "N" command anytime your code does a JSR to CIO or SIO.

8.5 Graphics Location Saved by DDT

Whenever DDT is entered (see section 7), it saves certain memory locations pertaining to user graphics before presenting its own display (also see Section 4.2). The locations saved are all "shadow registers" or vectors in page two. The following are a list of the locations saved, by label, hex address, number of bytes saved, and very brief description. The labels given are those used in "Mapping the Atari" (from Compute! Books) and in the Atari OS listings (part of the Atari Technical Manual set), and we refer you to those publications for more information.

Label	Address	# bytes	Description
VVBLKI	$0222	2	VBI immediate vector address
VVBLKD	$0224	2	VBI deferred vector address
SDLSTL	$0230	2	Start address of display list
SDMCTL	$022F	1	DMA control register
GPRIOR	$026F	1	Priority selection register
COLOR1	$02C5	1	Color register 1
COLOR2	$02C6	1	Color register 2
COLOR4	$02C8	1	Color register 4
CHACT	$02F3	1	Character mode register
CHBAS	$02F4	1	Character set base address

If your program uses other system memory locations which are altered by DDT, or if your program changes graphics characteristics by direct changes to the Atari hardware registers, DDT will NOT be able to completely restore the screen display your program was exhibiting when DDT was entered.

8.6 Using MAC/65 as a Mini-Assembler for DDT

Those of you who have used other debugging tools may note that, while DDT has a fairly sophisticated disassembler, it lacks a built-in mini-assembler. Thanks to the integrated nature of MAC/65 and DDT, though, you may never even notice this omission.

Let us suppose, for the moment, that you have just assembled a small to medium sized program from source code in memory, placed the object code in memory, and have entered DDT. When you discover an error in your code, you can simply pop back to MAC/65, change the offending code, re-assemble, and be back in DDT in a matter of a very few seconds.

But what if the code you want to patch is NOT in memory or is not directly related to the source currently in memory. What can you do then? We suggest the following steps:

```
Exit DDT via the Q (Quit) command.
If there is a source program in MAC/65's edit
      buffer, type in  "RENUM 1000,1"
Type NUM 10,10.
Enter your patch, using "*=" to control where
      the patch goes and " .OPT OBJ" to
      ensure the patch really ends up in memory
Assemble via the "ASM" command.
Go back to DDT (via "DDT", of course), and your
      patch is in place.
NOTE: to get rid of your patch code without
      affecting your main program, you may
      type "DEL 1,999" to MAC/65.
```

There ARE some things to watch out for if you use this method. Primarily, you want to ensure that MAC/65 doesn't wipe out the program you are trying to debug. There are two possible ways this could happen.

First, remember that MAC/65 destroys the lower half of page 6 ($600 through $67F). If you are using page six, then, you should use the Move command to move page six to someplace "safe" before going to MAC/65 (then move it back when you return to DDT).

Second, the very process of editing (writing) even a small program will overwrite some of memory. However, we direct your attention to the MAC/65 LOMEM and SIZE commands. You can use LOMEM to set the bottom of the memory that MAC/65 will use. You can use SIZE to determine what memory MAC/65 is, indeed, using. We suggest, if you intend to use the method we have outlined, that you use LOMEM when you first enter MAC/65.

The other major problem you can encounter relates to the way DDT saves the state of MAC/65 when it is entered. Since the two programs (and they really do operate as almost totally separated programs) share some of the same memory space, DDT saves part of page zero and part of page four ($80-$FF, $480-$4FF) when it is entered. It saves these locations at the start of MAC/65's "buffer" space.

You can determine where the buffer space is by using SIZE: the first number displayed in response to SIZE is the hexadecimal address of this buffer. You can change where this buffer is by using LOMEM.