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.