

# **IBM Personal Computer Seminar Proceedings**

**The Publication for Independent Developers  
of Products  
for IBM Personal Computers**

**Published by International Business Machines Corporation  
Entry Systems Division**



Changes are made periodically to the information herein; any such changes will be reported in subsequent Proceedings.

It is possible that this material may contain reference to, or information about IBM products (machines and programs), programming or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM believes the statements contained herein are accurate as of the date of publication of this document. However, IBM makes no warranty of any kind with respect to the accuracy or adequacy of the contents hereof.

This publication could contain technical inaccuracies or typographical errors. Also, illustrations contained herein may show prototype equipment. Your system configuration may differ slightly. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

All specifications are subject to change without notice.

Copyright ©  
International  
Business  
Machines  
Corporation  
08/84

Printed in the  
United States  
of America

All Rights  
Reserved



# Contents

<b>Introduction and Welcome</b>	<b>1</b>
Purpose	1
Topics	1
<b>IBM Personal Computer Resident Debug Tool</b>	<b>2</b>
Overview	2
Organization	3
What You Need	3
How To Load The Resident Debug Tool	4
Resident Debug Tool Display Screen	6
Resident Debug Tool Command Line Input	7
Command Line Control Keys	7
Command Processing	8
Command Parsing	8
Expression Evaluation	8
Resident Debug Tool Windowing	10
Memory Windowing	10
Disassemble Windowing	12
Trace Windowing	14
Math Co-Processor Windowing	15
Resident Debug Tool Commands	16
Register Commands	16
General Register Commands	16
Specific Register Commands	16
Segment Register Commands	16
Flag Register Commands	16
Execution Commands	17
Variable Commands	17
Breakpoint Variable Commands	17
Scratchpad Variable Commands	17
Windowing Commands	18
Memory Window Commands	18
Disassemble Window Commands	18
Trace Window Commands	18
Math Co-Processor Window Command	18
Memory Commands	19
Memory Block Commands	19
Find Commands	19
Input/Output Commands	19
Disk and Diskette Read/Write Commands	19
I/O Port Commands	19
Debug Environment Commands	19
Screen Control Commands	20
Move Display Commands	20
Reset Commands	20
Appendix A	21
Sample RDT Display Screens	21
Appendix A, Screen 1	22
Sample RDT Logo Screen	22
Appendix A, Screen 2	23
RDT Display Screen Layout	23
Appendix A, Screen 3	24
Sample RDT Initial Entry Screen	24
Appendix A, Screen 4	25
Sample RDT Memory Windowing Screen	25
Appendix A, Screen 5	26

Sample RDT Disassemble Windowing Screen .....	26
Appendix A, Screen 6 .....	27
Sample RDT Partial Trace Windowing Screen .....	27
Appendix A, Screen 7 .....	28
Sample RDT Full Trace Windowing Screen .....	28
Appendix A, Screen 8 .....	29
Sample RDT Math Co-Processor Windowing Screen .....	29
<b>Questionnaire .....</b>	<b>30</b>

# Introduction and Welcome

These are the Proceedings of the IBM Personal Computer Seminar, designed for independent developers of products for IBM Personal Computers. The purpose of these Proceedings is to aid you in your development efforts by providing relevant information about new product announcements and enhancements to existing products. This issue is prepared in conjunction with this seminar. The Proceedings of future seminars for the IBM Personal Computers also will be published and will cover topics presented at those seminars.

Throughout these Proceedings, the term Personal Computer and the term family of IBM Personal Computers address the IBM Personal Computer, the IBM Personal Computer XT, the IBM PCjr, the IBM Portable Personal Computer, and the recently announced IBM Personal Computer AT.

## Purpose

What is our purpose in issuing a publication such as this? It is quite simple.

The IBM Personal Computer family is a resounding success. We've had a lot of help in achieving this success, and much of it came from the independent developers.

As you proceed with your development, do you at times wish for some bit of information or direction which would make the job easier? Information which IBM can provide? This is the type of information we want to make available to you.

Since we want to be assured of giving you the information you need, we ask you to complete the

questionnaire which appears at the end of these Proceedings. Your response to this questionnaire will be taken into account in preparing the content of future issues, as well as the content of seminars we will present at microcomputer industry trade shows.

## Topics

The following list gives a general indication of the topics we plan to cover in future seminars and include in the IBM Personal Computer Seminar Proceedings:

- Information exchange forum — letters to the editor format
- Development tools — languages, database offerings
- Compatibility issues
- New devices — capacities and speeds
- System capacities — disk and memory
- Enhancements in maintenance releases
- Tips and techniques
- New system software
- Hardware design parameters
- Tips on organizing and writing documents for clear and easy reading
- Changes to terms and conditions

# IBM Personal Computer Resident Debug Tool

## Overview

This Proceeding describes the IBM Personal Computer Resident Debug Tool. The Resident Debug Tool (RDT) is a functional programming tool.

RDT executes on the IBM Personal Computer processor using the Personal Computer keyboard/display to interface with the programmer. The Resident Debug Tool can be used to aid in software development and maintenance in an IBM Personal Computer Disk Operating System (DOS) environment.

A full-screen display and simple command structure provide a comprehensive but comprehensible view and control of the processor and its memory. A wide range of functions are supported by RDT. The basic functions include:

- Stop/Start program execution
- Single/Multiple step program execution
- Trace program instructions to a buffer while stepping
- Display/Alter processor registers, memory or I/O space
- Address stops (breakpoints)
- Search memory contents to find an ASCII/EBCDIC/hexadecimal string
- Copy an area of memory from one location to another location
- Compare one area of memory with another area of memory

- Instruction disassembly
- Hexadecimal expression evaluation, with scratchpad space for evaluated results
- Stop at program request
- Disk/diskette display, alter, find, verify
- Print memory, instruction trace buffer, or disassembled instructions
- Print the user program screen or debug tool screen
- Display math co-processor (8087) state (if installed)

A very useful user interface has been built around these basic functions to extend their capabilities. The RDT full screen display format normally displays many items of interest so no action need be taken to view these items. The wide variety of RDT commands are all two characters in length to minimize keystrokes and to assist users in remembering them. The full screen display is, to a large extent, self prompting; many of the valid commands and valid terms for a RDT expression are a part of the normal display.

The Resident Debug Tool operates as an extension of the Disk Operating System (DOS) and acts as an interrupt handler to control user program execution. When one of the interrupts being handled by RDT occurs, the debug tool is invoked. A programmer may then enter RDT commands to perform a desired function. Eventually, the programmer may resume the program's execution by issuing the appropriate RDT command. While a user's program is executing, RDT remains resident waiting to be invoked by an RDT-handled interrupt.

## Organization

This Proceeding is intended to provide an understanding of the capabilities of the IBM Personal Computer Resident Debug Tool. This document will:

- List the equipment and software you will need to use Resident Debug Tool
- Describe what to do to start using the RDT
- Explain how to load RDT and what options are available to you
- Discuss the RDT display screen format and explain Resident Debug Tool command line input
- Describe the Resident Debug Tool windowing modes, memory, disassemble, trace, and math co-processor windowing modes
- Provide an overview of the commands available in the Resident Debug Tool

## What You Need

You need the following hardware and software to operate the Resident Debug Tool:

- An IBM Personal Computer with a minimum of 128K of memory.
- At least one diskette drive.
- An 80-column display (with appropriate IBM Monochrome or IBM Color/Graphics Display Adapter).
- IBM Disk Operating System (DOS).

The Resident Debug Tool also supports the following:

- Both an IBM Monochrome Display and 80-column Color monitor (with appropriate display adapters).
- IBM Matrix Printer or any compatible printer.
- Math Co-processor option.

# How To Load The Resident Debug Tool

After loading the IBM Disk Operating System (DOS) on an IBM Personal Computer, Personal Computer XT or the IBM Portable Personal Computer, insert the diskette containing the Resident Debug Tool into a diskette drive and enter the following command:

```
<drive:> RDT <options>
```

where:

'drive:' is the diskette drive in which the diskette containing the Resident Debug Tool has been placed

'RDT' is the name of the Resident Debug Tool  
'options' may be any of the following RDT options:

'a' specifies that you want the RDT program to allocate 4K within the debug tool memory space for the saving of a full monochrome display buffer or first 4K of the color display buffer when the user and the debug tool are sharing the same display. This enables any alphanumeric display page to be properly saved and restored while debugging. ('a' and 'g' are mutually exclusive options.)

'c' specifies that you want the RDT program to initially use the color display for the RDT display screen. ('c' and 'm' are mutually exclusive options.)

'g' specifies that you want the RDT program to allocate 16K within the debug tool memory space for the saving of a full monochrome display buffer or full color display buffer when the user and the debug tool are sharing the same display. This enables any alphanumeric or graphic display to be properly saved and restored while debugging. ('a' and 'g' are mutually exclusive options.)

'k' specifies that you want the RDT program to gain control whenever an Interrupt 5 (print-screen interrupt) is issued. RDT will save the current Interrupt 5 vector and replace it with one which causes entry into the debug tool. This is normally the 'print-screen' interrupt and is issued from the BIOS keyboard handling routine. This option will allow the user to pass control to the debug tool when using keyboard I/O which makes use of the BIOS code. The user presses 'Shift-PrScr' in this case to have the debug tool take control. When RDT is handling this interrupt, it makes the necessary adjustments to the state of the machine

so that the user state appears just as it did before going off to service the interrupt. The Interrupt 5 should only be issued from BIOS when making use of this option! When using this option, never place an 'INT 5' instruction in your program, as this will have unpredictable results.

'm' specifies that you want the RDT program to initially use the monochrome display for the RDT display screen. ('c' and 'm' are mutually exclusive options.)

'n' specifies that you want the RDT program to handle soft Interrupt 2's (non-maskable interrupts), such as those caused by an Interrupt 2 instruction or a math co-processor exception condition.

'p' specifies that you want the black and white attribute set to be the default color attributes when the Resident Debug Tool is using the color/graphics adapter to display the RDT screen. This makes the RDT normal color attribute to be white on black, RDT highlight color attribute to be intense white on black, and the RDT reverse color attribute to be black on white. This option is primarily intended for those who use a two-color monitor with the color/graphics adapter, such as the IBM Portable Personal Computer.

't size' specifies the size of the RDT instruction trace buffer to be included in the debug tool where the size is the number of instructions in hexadecimal. For DOS 1.1, the minimum size which can be specified is hexadecimal '20' (32 decimal) and the maximum size which can be specified is hexadecimal 'D0' (208 decimal). For DOS 2.0 and above, the minimum size which can be specified is hexadecimal '20' (32 decimal) and the maximum size which can be specified is hexadecimal '924' (2340 decimal).

'x' specifies that the user wishes to not have to press any key to continue when the IBM logo screen is displayed. This enables the invocation of RDT to be contained in a batch EXEC along with invocation of the program being debugged without having any human intervention required.

If neither the 'm' nor 'c' options are specified when the RDT program is loaded, the current display screen is used for the RDT display screen. The Resident Debug Tool uses the 80x25 display mode for either the monochrome or color display. If RDT and the user's program being debugged use the same display monitor, RDT contains code to support 'swapping' the display states as necessary while debugging user programs. In addition, if RDT and the user's program are on the same display, an 'a' or 'g' option specified, and the user's program in an appropriate mode (alphanumeric for



'a', alphanumeric or graphics for 'g'), the user's program display screen is saved by RDT upon entry to the tool and restored upon resuming execution of the user's program.

If no trace buffer size option is specified, a default buffer size for 32 instructions (hex '20') is created within RDT. Twenty-eight bytes are required for each instruction within the RDT trace buffer. Therefore, a trace buffer capable of holding 32 instruction will require 896 bytes. Likewise, the maximum DOS 2.0 trace buffer of 2340 (hex '924') instructions requires 65,520 bytes.

When the Resident Debug Tool is loaded and all options specified are valid, an IBM logo screen will appear on the selected (or default) debug tool display (see Appendix A, Screen 1 for an example of the RDT logo screen). The user will be asked to press any key to continue. After having pressed a key (or after a few seconds if the 'x&s sq. option was specified), the IBM Personal Computer Resident Debug Tool is made a resident extension of DOS. From this point on, the Resident Debug Tool will reside in low memory just above DOS. The user's programs can then be loaded, just as they normally are. DOS will load these programs above the Resident Debug Tool.

The amount of space occupied by RDT is determined by the options specified when the tool is loaded. If no options are specified, the size of the debug tool is approximately 42K. The 'c', 'k', 'm', 'n', and 'x' options have no effect on this base 42K amount. The 'm' and 'g' options are mutually exclusive. No more than one of these should be specified. The 'm' option requires an additional 4K within RDT. The 'g' option requires an additional 16K within RDT. The 't size' option requires from 0K-63K additional depending on the size requested. When all of these options are considered, the amount of memory occupied by the Resident Debug Tool will range from approximately 42K to 121K.

When RDT is made a resident extension of DOS, it saves the user's Interrupt 2 vector and then modifies the Interrupt 3 (breakpoint interrupt) vector and the Interrupt 2 (non-maskable interrupt) vector (and optionally the Pr-Scr interrupt vector if the 'k' option was specified) to point to RDT interrupt handlers. RDT will then act as an interrupt handler to control user program execution. These vectors return control to the debug tool via an interrupt. An Interrupt 3 is triggered by either a user requested breakpoint set by the debug tool or by an INT3 (hex 'CC') instruction within code being executed by the IBM Personal Computer. An Interrupt 2 can be classified as either hard or soft. A hard Interrupt 2 is issued by the system when a memory parity error or I/O channel check occurs, or by the user pressing the NMI button on the card which can be

used with RDT and releasing after one to two seconds. A soft Interrupt 2 is triggered by either an INT 2 (hex 'CD02') instruction within code being executed by the IBM Personal Computer or by a math co-processor exception condition.

This description of RDT Interrupt 2 handling is primarily intended for those may wish to have their own Interrupt 2 routines handle math co-processor exception conditions. The Resident Debug Tool always handles hard Interrupt 2's. RDT will, however, only handle soft Interrupt 2's if the user requests that it do so, either by an 'n' load-time option or with the 'SN' command. When an Interrupt 2 occurs, RDT determines whether or not to handle it. If RDT is not to handle soft Interrupt 2's, it must determine whether or not a soft Interrupt 2 has in fact occurred. If the Interrupt 2 is soft, RDT will jump to the user's Interrupt 2 routine so that it may handle the interrupt condition. The address of this interrupt routine is determined from the user's Interrupt 2 vector, which was saved when RDT was first made resident and may have been since modified. When the user resumes program execution (via the 'EX' command) a check is made to determine whether or not the Interrupt 2 vector still points to the RDT Interrupt 2 handler. If it does not, this vector is saved as the user's new Interrupt 2 vector by RDT and the Interrupt 2 vector in low memory is modified to point to the RDT Interrupt 2 handler. If the user's program modifies the Interrupt 2 vector, the user should return control to RDT by one of the other Interrupts RDT handles (such as INT3) so that RDT may save the new user Interrupt 2 vector and replace it with a vector pointing to the RDT interrupt handler when user program execution resumes.

A good practice for users writing programs which might be debugged using the Resident Debug Tool is to place an INT3 instruction at the beginning of the program. This will cause control to be returned to the debug tool immediately after DOS has loaded the program. The user may then set breakpoints in their program or use any of the other facilities of the debug tool. In normal operation, when no debug tool is present, DOS ignores the INT3 instructions so there is no need to worry about removing them from the code. code (unless the one byte used for an INT3 is of concern).

The following are three examples of commands that might be entered to load the Resident Debug Tool, and what result the command will have:

'RDT' - will cause the Resident Debug Tool to be loaded from the current default drive; RDT will initially use the current display for the debug tool display; RDT will retain control of the Interrupt 3 and Interrupt 2 vectors, and will handle all Interrupt

3's and hard Interrupt 2's; the trace buffer size will be 32 (hex '20') instructions; no RDT user display buffer area will be allocated; when RDT is loaded, an IBM logo screen will appear, and wait for any key to be pressed before continuing.

'B:RDT N A C' - will cause the Resident Debug Tool to be loaded from the 'B' drive; RDT will initially use the color display for the debug tool display; RDT will retain control of the Interrupt 2 (NMI) and Interrupt 3 vectors and will handle all Interrupt 2's and Interrupt 3's; the trace buffer size will be 32 (hex '20') instructions; a 4K RDT user display buffer area will be allocated; when RDT is loaded, an IBM logo screen will appear, and wait for any key to be pressed before continuing.

'RDT N G M K T 100 X' - will cause the Resident Debug Tool to be loaded from the current default drive; RDT will initially use the monochrome display for the debug tool display; RDT will retain control of the Interrupt 2 (NMI), Interrupt 3, and Interrupt 5 (Print Screen) vectors and will handle all Interrupt 2's, Interrupt 3's, and interrupt 5's; the trace buffer size will be 256 (hex '100') instructions; a 16K RDT user display buffer area will be allocated; when RDT is loaded, an IBM logo screen will appear for a few seconds before continuing.

## Resident Debug Tool Display Screen

The Resident Debug Tool full screen display can be viewed as being five separate areas (see Appendix A, Screen 2 for an example of the RDT display screen layout). The first line displays the release and version number and the release date of the Resident Debug Tool. Lines 2-9 display the address stops, variables, and processor registers. Line 10 is the command line. Line 11 is the message line. Lines 12-25 are the windowing area of the RDT display.

The cursor may be placed anywhere in the input area of the command line, or it may be moved to selected parts of the windowing area. The command line is used for entering all of the RDT commands, while the windowing area is used to display/alter memory, display/alter disassembled instructions, display the trace buffer, or display the math co-processor state (if installed).

This section will concentrate on the first four areas of the RDT display screen, since these remain the same no matter which of the windowing modes you are in. The following two sections (4 and 5) explain Resident Debug Tool command line input and the different Resident Debug Tool windowing modes and their operation.

Appendix A, Screen 3 illustrates an example of what the RDT display may look like when first returning control to the debug tool by way of an 'INT3' instruction encountered while executing a user's program. As Screen 3 demonstrates, all of the processor registers are displayed in the top portion of the screen along with the address stops (Sn) and variables (Vn). The 8088 processor flag register is displayed in both hexadecimal and binary forms. The 8088 segment registers are displayed in 20-bit 'real-address' representations. A segment register is only 16 bits long, which corresponds to the first 4 characters of the segment register's displayed value. But since most calculations using the segment registers involve their real 20-bit values, a '0' is appended to the 16-bit segment register content. The center of the screen contains the command line, followed by the status (or message) line. Initially, RDT is in memory windowing mode. In the memory windowing mode, the bottom portion of the screen contains the memory window area; line numbers, addresses, and hexadecimal representations appear on the left side while character (EBCDIC or ASCII) representations appear on the right side.

When it is the initial entry into the Resident Debug Tool, none of the breakpoints or variables have been set, so they appear simply as dots. Since no windowing activity has taken place, the first line of the memory windowing area contains the hexadecimal address '00000'. Initially there is only one 'memory window'; it is 14 lines long, each line displaying 16 bytes of the PC's memory. The values displayed for the 8088 processor general purpose registers are those which were present when the user program issued the INT3 instruction. The EX field is a pseudo 20 bit program counter which is maintained by RDT; it contains the sum of the CS and IP registers, and is followed by a one-to-seven byte hexadecimal memory display of the next instruction to be executed. Above the hexadecimal display is a disassembled representation of the instruction. If the instruction has an Operand which references memory, the 20-bit memory address is calculated (segment register + displacement + base/index register(s)) and displayed as the OP: variable. If the instruction has an operand which references a particular byte or word in memory, this byte or word is displayed under the OPerand location. The undisplayed variable IL (Instruction Length) contains the length of the instruction. The Flag Register (FL) has several bits that are architecturally undefined, but the fact that the OF, DF, TF, SF, AF, and CF bits are zero and the IF, ZF, and PF bits are one is displayed both in binary form and in 16 bit (F246) form. The default step count (CT) is set to one.

The 'D1' in the upper right part of the display indicates that this is RDT Display Screen number 1. The Resident Debug Tool supports 9 debug display screens. Some variables are common to all 9 displays and some variables are unique to each display. For example, the Processor registers and memory contents are common for all display screens while the displayed variables (V1-V9, S1-S9, L1-L9, M1-M5) are unique to each of the 9 displays. A useful debug technique is to use a different display for each of the user code sections which is being debugged.



## Resident Debug Tool Command Line Input

This section describes cursor motion and general capabilities of the command line. The command line is used for entering all of the commands to the Resident Debug Tool.


### Command Line Control Keys

When the Resident Debug Tool is first loaded, all command line alphabetic input is upper case, regardless of the shift key position. numeric and special character keys always have dual case capability. The F3 key may be used to change from single case alphabetic input to dual case alphabetic input and back again; the cursor must be on the command line for the F3 key to cause this change.

Outboard keys and inboard keys other than data keys function are as follows:

- Enter key (  ) : moves the cursor to the beginning of the command line and executes the commands on the command line. This key performs this function regardless of the cursor position on the screen.
- CAPS LOCK key : moves the cursor down into the window area of the screen (if possible for the current window mode)
- Up arrow : same as CAPS LOCK
- Down arrow : same as CAPS LOCK
- Right arrow : moves the cursor right one position unless the cursor is at the end of the command line.
- Left arrow : moves the cursor left one position unless the cursor is at the beginning of the command line.
- Tab key (  ) : moves the cursor to the next command on the command line. The tab key actually moves the cursor to the position

following the next semi-colon; if there are no semi-colons to the right of the cursor, the cursor is moved to the beginning of the command line.

- INSert Key (or F4) : changes between the insert/overstrike input keying modes. A blinking asterisk at the beginning of the command line indicates that insert mode is active.
- Delete key (DEL) : deletes the character over the cursor and shifts the remainder of the command line left one character.
- Backspace key (  ) : moves the cursor left one position, removes the character over the new cursor position, and, if insert mode is active, shifts the remainder of the command line left one position.
- HOME key : moves the cursor to the beginning of the command line. This key performs this function regardless of the cursor position on the screen.
- END key : moves the cursor to the position following the last non-blank character on the command line. If the command line is blank, the cursor is placed on the first input position of the command line.
- Shifted Print-Screen Key (PrtSc): causes the native tool screen to be printed.
- Control key (CTRL) : ignored for command line input.
- F1 : same as HOME key.
- F2 : does a Find ASCII, or Find EBCDIC, depending on the setting of the 'DISPLAY' variable, using the command line as the Find command input parameters; what is passed to the Find command is a string which begins at the beginning of the command line and ends one character before the cursor position. If the cursor is at the beginning of the command line, searches memory for the previous search argument used in a Find command.
- F3 : changes between the single/dual case alphabetic input keying modes.
- F4 : same as INSert key.
- F5 : erases the entire command line and places the cursor at the beginning of the command line.
- F6 : erases the command line beginning at the cursor position continuing to the end of the command line.

- F7 : restore the user command line which has been saved with the 'F7' command to the command line. If no command has been saved this key has no effect.
- F8 : restore the user command line which has been saved with the 'F8' command to the command line. If no command has been saved this key has no effect.
- F9 : restore the user command line which has been saved with the 'F9' command to the command line. If no command has been saved this key has no effect.
- F10 : retrieves the previously 'Entered' commands to the command line. First depression of the F10 key will restore the most previously 'Entered' command. Second depression of the key will restore the next most previously 'Entered' command, and so on. The command retrieve area is a 10-deep circular queue. Up to the 10 most previously 'Entered' commands will be saved.

## Command Processing

When the ENTER key is pressed, the command line is processed. Command processing begins with the first (leftmost) command on the command line and continues with the remaining commands. Commands are separated by semi-colons. Command line processing stops when a command does not return with a zero return code to the command line processor (normally when the command encounters an error).

When command line processing finishes, the cursor moves to the beginning of the command line. The command line is cleared unless the first character of the command line is a slash (/) or unless an error was encountered. The sole function of the slash at the beginning of the command line is to preserve the command line after command line processing is finished.

As many commands may be placed on the command line as will physically fit. Commands may be preceded and/or followed by as many or as few blanks as desired. There is only one restriction concerning commands which may/may not be used in a multiple command sequence. If the 'F7', 'F8' and 'F9' (Store User Command) commands appear in a command line, they will be the last command executed on that line (as the rest of the line will be saved as the stored user command).

## Command Parsing

The nominal command line format is as follows:

```
/ cmd = opnd ; cmd = opnd ; cmd = opnd ; . . .
```

The slash at the beginning of the command line is optional. If it is present, the command line will not be erased when command line processing is finished (after the ENTER key is pressed).

Commands are always two characters in length. The last command may be followed by a semi-colon. This trailing semi-colon does not affect command line processing, but it may make it easier to use the TAB key to position the cursor after the last command in the event the user wishes to add another command.

The blanks as well as the '=' sign between the 'cmd' and the 'opnd' are optional. The syntax is not pretty, but many users save unnecessary keystrokes by using the format:

```
cmdopnd;cmdopnd;cmdopnd; . . .
```

The syntax of the Resident Debug Tool defines the operand as beginning with the first non-blank character (other than '=') following the command and continuing to the last non-blank character preceding the semi-colon or the end of the command line. If there are no non-blank characters following the command, the operand is considered to be null.

## Expression Evaluation

The operand of most commands is an expression. An expression is defined here as one or more terms which are added together. A term may be self-defining hexadecimal, one of the processor registers, or one of the Resident Debug Tool variables.

Terms are always added together. The unary signs '+' and '-' may be used to precede terms. The unary '+' sign performs no useful function, but can improve operand readability. The unary '-' sign negates the term before the addition is performed. Multiple unary signs may precede a term if desired.

Blanks may be used freely between terms in an expression. Blanks may also appear between a unary sign and its term.

Terms which are processor registers or RDT variables are always 2 characters in length. Self-defining hexadecimal terms are variable in length. A blank, unary sign, or end of operand must follow a hexadecimal term to delimit the end of the term; there is no such requirement for register or variable terms.

The following is intended to exemplify expression syntax:

```
AX = V1 - BX + 4A + AX
AX=V1-BX+4A+AX
AX V1-BX 4A AX
AXV1-BX4A AX
```

The above commands are equivalent. The RDT variable V1, the negative value of processor register BX, hexadecimal 4A, and the processor register AX are added together with the result being placed into the AX processor register. The first example illustrates optimum readability. The second example removes the optional blanks following the command and between terms. The third example removes the optional '=' sign and '+' unary signs. The fourth example removes optional blanks. The fourth example is not very readable, but many users prefer it because of the savings in keystrokes.

The following are the valid processor register and Resident Debug Tool variable terms which may be used in an expression:

AH	CL	DL	LC	S1-S9
AL	CO	DS	L1-L9	V1-V9
AX	CS	DX	M1-M9	W1-W9
BH	CT	ES	OP	X1-X9
BL	CX	EX	SI	Y1-Y9
BP	DH	FX	SP	Z1-Z9
BX	DI	IL	SS	
CH				

The variables OP and IL may be used as command operands. For example, the command L1=OP may be used to display the contents of memory at the instruction operand address. The command /IP=IP+IL may be used to disassemble a program line-by-line without actually executing the program. (Since the program is not actually executing, the processor registers are not being updated and the OP variable is probably not useful.)

The value of the variables L1-L9 and M1-M5 depend upon which of the RDT windowing modes you are in. While in disassemble windowing mode, these variables take on the sum of their respective disassemble window line's CS and IP values. You will find that this is an especially nice feature when setting breakpoints. While in all other RDT windowing modes, the L1-L9 and M1-M5 variables take on the values they assume in the memory windowing mode. For more on windowing modes, refer to Resident Debug Tool Windowing.

## Resident Debug Tool Windowing

One of the most powerful features of the IBM Personal Computer Resident Debug Tool is its extensive windowing capabilities. The user has the choice of using the windowing area of the RDT screen to display and alter of memory windows, display of windows of disassembled instructions and alter of their hex representations, display of RDT's instruction trace buffer, which contains instructions which have been saved in the trace buffer while stepping according to user specified trace options, or display the state of the math co-processor if one is installed.

The 'WINDOW' variable on Line 4 of the Resident Debug Tool shows the current windowing mode. This variable could be 'MEMORY' for memory windowing mode, 'DISASM' for disassemble windowing mode, 'TRACEP' for partial trace windowing mode, 'TRACEF' for full trace windowing mode, or 'COPROC' for math co-processor windowing mode. The 'MW' (Memory Windowing), 'DW' (Disassemble Windowing), 'TW' (Trace Windowing), and 'CW' (Co-processor Windowing) commands are used for switching from one windowing mode to another. The facilities which each of these windowing modes offer will now be discussed.

### Memory Windowing

Memory windowing enables the user to perform memory display (hex and EBCDIC or ASCII), alter, and scrolling functions.

If the user is not already in memory windowing mode, the 'MW' (Memory Windowing) command may be used to place RDT in memory windowing mode. RDT being in memory windowing mode is reflected by the setting of the 'WINDOW' variable on line 4 of the screen to 'MEMORY' (see Appendix A, Screen 4 for an example of the RDT display screen while in memory windowing mode).

While in memory windowing mode, the Down Arrow or Up Arrow keys may be used to move the cursor off the command line to the memory window area of the display. With the cursor in the memory window area of the display screen, the user may perform the memory display, alter and scrolling functions.

Each memory window line consists of a line number, address, and a hexadecimal representation on the left side and a character (EBCDIC or ASCII) representations on the right side. Each memory display line displays 16 bytes of memory. The EBCDIC or ASCII formats may be either filtered or

unfiltered. When the EBCDIC or ASCII display is filtered, only the alphanumerics (letters and numbers) are displayed; all other hexadecimal codes are displayed as dots. When the EBCDIC or ASCII display is unfiltered, all hexadecimal codes are displayed. The F3 key (with the cursor in the memory window display area) is used to change between the filtered and unfiltered display modes.

A memory display window is defined as one or more memory display lines that generally display a contiguous block of memory. A memory display window is as small as one line or as large as the entire memory edit display (14 lines).




A memory window is created by entering a new address for a given line; the new address may be entered by overtyping the old address or by one of the L1-M5 variable commands while in memory windowing mode. When an address is keyed on any given memory display line, RDT defines that line as being the first line of a memory display window. An asterisk is displayed at the left of the memory address to indicate that the line begins a new display window. The length of the window is determined by the next line which defines the beginning of another window. The Resident Debug Tool will create a new memory display window when a new address is entered. A memory window is destroyed (combined with the previous window) by pressing the DElete key with the cursor anywhere in the window or by one of the L1-M5 variable commands while in memory windowing with a null operand.

Memory may be altered with hexadecimal, EBCDIC, or ASCII input. Altering is performed by moving the cursor to the appropriate byte (hex, EBCDIC, or ASCII) and overtyping the data that is displayed. Each keystroke results in a modification of memory; a nibble within a byte is modified for each hexadecimal digit entered, and one full byte is modified for each EBCDIC or ASCII character entered. To prevent unintended memory patching, RDT will not move the cursor from the memory address display area to the hexadecimal memory display area when a character key is pressed. One of the cursor motion keys (or space bar) must be used to move the cursor into the hexadecimal, EBCDIC, or ASCII display area.

A previous and current display of memory may be achieved by displaying the same area of information on two different (normally adjacent) lines. As the keystrokes are entered on one line, memory is updated and that display line reflects the updated memory. The second memory display line is not updated at that time, however, and therefore may be used to see what memory was before it was patched.

A concept known as indefinite windows has been implemented for memory patching. When the last byte of a memory display segment has been modified, RDT automatically scrolls the memory display window right one byte to display the next memory location and make it available for modification. The user may therefore patch a string of bytes (hex, EBCDIC, or ASCII) of indefinite length without entering a new address. The space bar and backspace key also cause automatic scrolling of the memory display window. If the user wishes to move the cursor without causing automatic scrolling, he must use one of the 4 cursor motion keys or the tab or carrier return (CAPS LOCK) keys.

The following keys may be used to control cursor motion and scrolling when the cursor is in the memory window area:

- Enter key (  ) : returns the cursor to the beginning of the command line and executes the command line.
- CAPS LOCK key : moves the cursor to the beginning of the next line down. When the cursor reaches the bottom line, the cursor is moved next to the command line.
- Up Arrow : moves the cursor up one line. When the cursor reaches the top of the memory window display area, it wraps around to the bottom line.
- Down Arrow : moves the cursor down one line. When the cursor reaches the bottom of the memory window display area, it wraps around to the top line.
- Right Arrow : moves the cursor right to the next available position into which keystrokes may be entered. When the cursor reaches the end of a line, it wraps to the beginning of the next line.
- Left Arrow : moves the cursor left to a position into which keystrokes may be entered. When the cursor reaches the beginning of a line, it wraps to the end of the previous line.
- Tab Key (  ) : moves the cursor right to the next memory display column. When the cursor is in the rightmost column (the EBCDIC/ASCII display area), the Tab key moves the cursor to the beginning of the next line.
- Backspace Key (  ) : functions similarly to the Left Arrow, except that it causes automatic scrolling when the beginning of a memory display window is reached.
- Shifted Print Screen Key (PrtSc): causes the RDT display screen to be printed.
- Space Bar : functions similarly to the Right Arrow, except that the space bar is a valid data key when entering EBCDIC or ASCII data and will cause automatic scrolling when the end of a hexadecimal memory display window is reached.
- HOME key (or F1) : returns the cursor to the beginning of the command line.
- F3 : changes between filtered and unfiltered display modes when displaying EBCDIC or ASCII translations.
- F7 : restore the user command line which has been saved with the 'F7' command to the command line. If no command has been saved this key has no effect.
- F8 : restore the user command line which has been saved with the 'F8' command to the command line. If no command has been saved this key has no effect.
- F9 : restore the user command line which has been saved with the 'F9' command to the command line. If no command has been saved this key has no effect.
- F10 : retrieves the previously 'Entered' commands to the command line. First depression of the F10 key will restore the most previously 'Entered' command. Second depression of the key will restore the next most previously 'Entered' command, and so on. The command retrieve area is a 10-deep circular queue. Up to the 10 most previously 'Entered' commands will be saved.
- Up Arrow (CTRL 'ed): Scrolls the memory display window up one line. (The cursor may be located anywhere within the memory display window which is to be scrolled.)
- Down Arrow (CTRL 'ed): Scrolls the memory display window down one line.
- Right Arrow (CTRL 'ed): Scrolls the memory display window right one byte.
- Left Arrow (CTRL 'ed): Scrolls the memory display window left one byte.
- Tab Key (CTRL 'ed): Moves the cursor to the beginning of the next EBCDIC or ASCII display area.
- Page Up (PgUp): Scrolls the memory display window up by the size of the window.
- Page Down (PgDn): Scrolls the memory display window down by the size of the window.

## Disassemble Windowing

Disassemble windowing enables the display of disassembled instructions in the windowing area of RDT display screen and altering of the instructions' hex representations or the bytes or words referenced in memory by the instructions.

If the user is not already in disassemble windowing mode, the 'DW' (Disassemble Windowing) command may be used to place RDT in disassemble windowing mode. RDT being in disassemble windowing mode is reflected by the setting of the 'WINDOW' variable on line 4 of the screen to 'DISASM' (see Appendix A, Screen 5 for an example of the RDT display screen while in disassemble windowing mode).

While in disassemble windowing mode, the windowing area of the RDT display screen contains one or more 'disassemble windows'. A disassemble window is defined as one or more lines of program instructions disassembled from a contiguous block of memory. A disassemble window may be as small as one instruction or as large as the entire windowing area (14 instructions). Each disassemble window line consists of a label (L1-M5), a CS (code segment) value, an IP (instruction pointer) value, disassembled hexadecimal and character representations of the disassembled instruction pointed to by the combination of the CS and IP values, and may contain an operand or jump location, and a byte or word value if the instruction disassembly indicates that they exist for the referenced instruction.

The first line of the windowing display area always defines the beginning of a disassemble window while in disassemble windowing mode. A new disassemble window is created by moving the cursor down to any given window area line and overtyping the CS or IP value, or by entering one of the label (L1-M5) commands specifying and new CS and IP pair for a given window area line. An asterisk is displayed at the left of the CS and IP values to indicate that the line begins a new disassemble window. The size of the disassemble window is determined by the next line which defines the beginning of another window (or the bottom of the display screen window area). A disassemble window is displayed by taking the CS and IP values for the window area line which begins a window, and disassembling instructions from this start location until another window begins or the bottom of the window area is reached. The address of each successive instruction in a window is taken as the address of the previous instruction address plus the length of the previous instruction (returned

from the disassembler routine). A disassemble window is destroyed (combined with the previous window) by pressing the 'Delete' key with the cursor anywhere in the window or by entering one of the label (L1-M5) commands with a null operand while in disassemble windowing mode.

Disassemble windowing allows the programmer the powerful capability of interactively modifying instructions in a disassemble window. This is accomplished by moving the cursor off of the command line by using the Down Arrow or Up Arrow keys, to the hexadecimal representation of the instruction desired to be changed, and overtyping the hexadecimal nibbles. As the new hexadecimal nibble is typed, the entire window in which the instruction appears is immediately updated to reflect the modified instruction. Also, instructions which reference bytes or words in memory may have these operand values modified by moving the cursor to the byte or word within the disassemble window instruction and overtyping the hexadecimal nibble with the new value.

One of the most powerful features of disassemble windowing is the ability to scroll the disassemble windows down, left, or right. This is accomplished by pressing the 'Ctrl' key in combination with the down, left, or right arrow keys, with the cursor anywhere in the window to be scrolled. Scrolling down causes the current IP value at the top of the disassemble window to be replaced with the value the value of what would be the next instruction in the window, and the entire window is updated to reflect the scroll operation. Similarly, scrolling left or right causes 1 to be subtracted from or added to, respectively, the window beginning IP value, and the window updated.

Using the 'Page Down' key, and disassemble window may be scrolled down by entire size of the window. The last instruction in the window is found, its IP value retrieved, its length determined, this value added to the retrieved IP value, and the result made the IP value of the first instruction in the disassemble window.




Another helpful feature built into the disassemble windowing technique is the idea of the 'current' disassemble window. When in disassemble window mode, the first disassemble window (beginning with the first line in the window area) is always updated, after entry into RDT from user program execution, to display the next user instruction(s) to be executed.

There exists an alternative way of displaying a disassembled instruction's memory location in a disassemble window line. If the user has set up an code origin (CO) value using the 'CO' command, and the combination of the CS value and IP value



for a particular disassemble window line are not more than 64K greater than the CO value, instead of displaying the CS and IP values for the line, the CO and a pseudo-location counter (LC) value are displayed. This allows a disassemble window to appear just like a program assembler listing with the pseudo-LC value matching that which appears on the listing. If the disassemble line is being displayed in this format, a dollar sign ('\$') is displayed to the left of the line. The rules for updating the instruction's hexadecimal representation and operands remain the same. However, for a line in this format, the user cannot type over the CS and IP values for the disassemble window line. He or she instead can update the pseudo-LC value displayed for the line by typing over the displayed value. A new disassemble window is created by moving the cursor to any given window area line and overtyping the LC value, or by entering one of the label (L1-M5) commands specifying and new LC for a given window area line. The effect this new value has on the disassemble window line's CS and IP values is exactly the same as the effect the 'LC' command from the command line has on the real CS and IP values.

The following keys may be used to control cursor motion and scrolling when the cursor is in the disassemble window area:

- Enter key (  ) : returns the cursor to the beginning of the command line and executes the command line.
- CAPS LOCK key : moves the cursor to the beginning of the first valid input field for the next disassemble line down. When the cursor reaches the bottom line, the cursor is moved next to the command line.
- Up Arrow : moves the cursor up one line. If the new location of the cursor is no a valid input position for a particular disassemble window line, the cursor continues to move up. When the cursor reaches the top of the disassemble window display area, it wraps around to the bottom line.
- Down Arrow : moves the cursor down one line. If the new location of the cursor is no a valid data input position for a particular disassemble window line, the cursor continues to move down. When the cursor reaches the bottom of the disassemble window display area, it wraps around to the top line.
- Right Arrow : moves the cursor right to the next available position into which keystrokes may be entered. When the cursor reaches the last valid input position of a given disassemble window, it wraps to the beginning of the first valid input field on the next disassemble window line.
- Left Arrow : moves the cursor left to a position into which keystrokes may be entered. When the cursor reaches the first valid input position of a given disassemble window, it wraps to the end of the last valid input field on the previous disassemble window line.
- Tab Key (  ) : moves the cursor right to the next valid disassemble window display input field. When the cursor is in the rightmost valid input field for a disassemble window line, area, the Tab key moves the cursor to the first valid input field in the next disassemble window line.
- Backspace Key (  ) : functions same as the Left Arrow.
- Shifted Print Screen Key (PrtSc): causes the RDT display screen to be printed.
- Space Bar : functions same as the Right Arrow.
- HOME key (or F1) : returns the cursor to the beginning of the command line.
- F7 : restore the user command line which has been saved with the 'F7' command to the command line. If no command has been saved this key has no effect.
- F8 : restore the user command line which has been saved with the 'F8' command to the command line. If no command has been saved this key has no effect.
- F9 : restore the user command line which has been saved with the 'F9' command to the command line. If no command has been saved this key has no effect.
- F10 : retrieves the previously 'Entered' commands to the command line. First depression of the F10 key will restore the most previously 'Entered' command. Second depression of the key will restore the next most previously 'Entered' command, and so on. The command retrieve area is a 10-deep circular queue. Up to the 10 most previously 'Entered' commands will be saved.
- Down Arrow (CTRL 'ed): Scrolls the disassemble display window down one instruction (or line). (The cursor may be located anywhere within the disassemble display window which is to be scrolled.)

- Right Arrow (CTRL 'ed): Scrolls the disassemble display window right one byte.
- Left Arrow (CTRL 'ed): Scrolls the disassemble display window left one byte.
- Page Down (PgDn): Scrolls the disassemble display window down by the number of instructions in the window.

## Trace Windowing

Trace windowing enables the display, in the windowing area of the RDT display screen, of instructions which have been placed in the RDT trace buffer while stepping, according to user specified trace options using the 'TR' command.

If the user is not already in trace windowing mode, the 'TW' (Trace Windowing) command may be used to place RDT in either partial or full trace windowing mode. RDT being in a trace windowing mode is reflected by the setting of the 'WINDOW' variable on line 4 of the screen to either 'TRACEP', for partial trace windowing, or 'TRACEF', for full trace windowing.

While in partial trace windowing mode (see Appendix A, Screen 6 for an example of the RDT display screen while in partial trace windowing mode), the windowing area of the RDT display screen contains 14 trace buffer entries, one to a line, each of which, if not empty, show both the hexadecimal and disassembled representation of an instruction which has been placed in the RDT trace buffer, preceded by the CS and IP register value which, when combined, determine the address of the instruction in memory (when it was placed into the trace buffer).

While in full trace windowing mode (see Appendix A, Screen 7 for an example of the RDT display screen while in full trace windowing mode), the windowing area of the RDT display screen contains 3 trace buffer entries, each 3 lines in length, and each of which, if not empty, show a line identical to that which would be shown if partial trace windowing. In addition, each entry consists of two lines which show all of the 8088 processor registers at the time the instruction was stepped.


While in either trace windowing mode, the 'TB' variable for each trace buffer entry shows the relative position of the instruction in the RDT trace buffer. The value of the 'TB' variable is the hex offset from the end of the RDT trace buffer for that trace entry. For example, an entry with the 'TB' variable of hex '0000' is the last instruction which

was placed in the trace buffer, and an entry with the 'TB' variable of hex '0001' is the second to last instruction which was placed in the trace buffer, and so on. While in a trace windowing mode, the TB variables increase in value moving from the bottom of the windowing area to the top. This enables instructions which have been placed in the trace buffer to be viewed in a natural top to bottom fashion, with the latest instruction in the trace window appearing at the bottom of the screen.

The user has the ability to change the view of the trace window area to the limits of the size of the RDT trace buffer. This is accomplished by moving the cursor off of the command line, using the Down Arrow or Up Arrow keys. This action moves the cursor to the bottom 'TB' variable of the trace windowing area. The user may then change the window view of the trace buffer in either of two ways. The first method is by scrolling the trace buffer window. This is accomplished by holding the 'Control' key down in combination with either the 'Up' or 'Down' arrow keys or by pressing the 'Page Up' or 'Page Down' keys (which scroll by the size of the window instead of by only one line). This causes the window to be scrolled in the respective direction, to limits of the trace buffer. An alternative method is by overtyping the 'TB' variable directly with the cursor on the nibble to be changed. RDT will only change this bottom 'TB' variable if the new value will permit all of the 'TB' variables in the windowing area to assume a value within the valid RDT trace buffer size range.

An asterisk next to a trace buffer entry line in the trace window display area indicates that one or more instructions may be missing from the trace buffer. Each of these instructions is a MOV or POP into a segment register. Hardware limitations on certain Intel 8088 processors cause the 'Trap Flag' to not be recognized immediately after a MOV or POP into a segment register. This will cause one or more instructions to appear to be skipped when single stepping through a MOV or POP into a segment register. What really happens though is the instructions are executed but control does not return to RDT until two instructions past the MOV or POP instruction (if one instruction past the MOV or POP is not itself a MOV or POP into a segment register). RDT places an asterisk on the trace buffer line to indicate the possible missing instruction(s) immediately following the asterisked instruction.

The following keys may be used to control cursor motion and scrolling when the cursor is in the disassemble window area:

- Enter key (  ) : returns the cursor to the beginning of the command line and executes the command line.

- Right Arrow : moves the cursor right to the next available position into which keystrokes may be entered. When the cursor reaches the last valid input position of the 'TB' input field, it moves no further.
- Left Arrow : moves the cursor left to a position into which keystrokes may be entered. When the cursor reaches the first valid input position of the 'TB' input field, it moves no further.
- Tab Key (→|) : moves the cursor to the beginning of the 'TB' input field, which in the last trace buffer line displayed in the RDT trace windowing area.
- Backspace Key (←) : functions same as the Left Arrow.
- Shifted Print Screen Key (PrtSc): causes the RDT display screen to be printed.
- Space Bar : functions same as the Right Arrow.
- HOME key (or F1) : returns the cursor to the beginning of the command line.
- F7 : restore the user command line which has been saved with the 'F7' command to the command line. If no command has been saved this key has no effect.
- F8 : restore the user command line which has been saved with the 'F8' command to the command line. If no command has been saved this key has no effect.
- F9 : restore the user command line which has been saved with the 'F9' command to the command line. If no command has been saved this key has no effect.
- F10 : retrieves the previously 'Entered' commands to the command line. First depression of the F10 key will restore the most previously 'Entered' command. Second depression of the key will restore the next most previously 'Entered' command, and so on. The command retrieve area is a 10-deep circular queue. Up to the 10 most previously 'Entered' commands will be saved.
- Up Arrow (CTRL 'ed): Scrolls the RDT trace buffer display window up one trace buffer entry, to the limits of the RDT trace buffer.
- Down Arrow (CTRL 'ed): Scrolls the RDT trace buffer display window down one trace buffer entry, to the limits of the RDT trace buffer.
- Page Up (PgUp): Scrolls the RDT trace buffer display window up by the number of trace buffer entries displayed in the window to the limits of the RDT trace buffer.
- Page Down (PgDn): Scrolls the RDT trace buffer display window down by the number of trace buffer entries displayed in the window to the limits of the RDT trace buffer.

### Math Co-Processor Windowing

Math co-processor windowing enables the display, in the windowing area of the RDT display screen, of the state of the math co-processor if one is installed.

If the user is not already in math co-processor windowing mode, the 'CW' (math Co-processor Windowing) command may be used to place RDT in math co-processor windowing mode. RDT being in math co-processor windowing mode is reflected by the setting of the 'WINDOW' variable on line 4 of the screen to 'COPROC' (see Appendix A, Screen 8 for an example of the RDT display screen while in math co-processor windowing mode),

While in math co-processor windowing mode, the windowing area of the RDT display screen contains the current state of the math co-processor. This state includes a display of the math co-processor's control word, status word, exception pointers, and register stack. The control word and status word are displayed in their hexadecimal representations, as well as having the individually defined bits for the two words displayed in a binary format. For more information on the content of the math co-processor control word, status word, exception pointers, or register stack, the user should refer to their IBM Technical Reference manual.

One note on the Resident Debug Tool's handling of math co-processor exceptions. When an exception occurs within the math co-processor, if interrupts are enabled for the math co-processor, and if the exception condition is not masked off within the math co-processor, an Interrupt 2 (NMI interrupt) occurs on the main processor (8088). If the user has directed RDT to handle soft NMI interrupts, either by way of the 'n' load-time option or the 'SN' (Set NMI) command, RDT contains the logic required to handle the math co-processor exception. RDT can distinguish a software or math co-processor NMI interrupt from other types of NMI interrupts, such as an NMI switch depression, memory parity error, or I/O channel check. If RDT is handling NMI interrupts, a math co-processor is installed, and a software or math co-processor exception NMI interrupt occurs, RDT is immediately placed in math co-processor windowing mode. In this case, when the RDT display appears, there will

be a message on the message line indicating the type of NMI interrupt and the math co-processor state will be displayed in the windowing area of the RDT display screen. This will enable users to immediately see the math co-processor exception which occurred, if any, by checking the exception bits of the status word. When handling the interrupt, RDT will also clear the exception condition, so that after the initial display of the math co-processor state, any subsequent display of the math co-processor state will show that the exception has been cleared.

If the user is debugging a math co-processor exception handler, the user should not instruct RDT to handle NMI interrupts, so that the user's exception handler will be invoked when an NMI occurs.

### Resident Debug Tool Commands

This section presents an overview of the commands available within the Resident Debug Tool. The commands are divided into seven functional groups. This section is not meant to be a user's guide, but rather to give a flavor of the wide variety of commands available.

### Register Commands

These register commands alter the contents of each of the 8088 processor general registers, specific registers, segment registers and flag register. The command line is used to enter commands to set the individual registers to a specific hex value or the value of a single or multiple term expression.

### General Register Commands

Each 8088 general register appears on the Resident Debug Tool display and has associated RDT change commands:

AX	BX	CX	DX
AL	BL	CL	DL
AH	BH	CH	DH

The register commands (AX, BX, CX, DX) alter the registers in their entirety while the remaining commands (AL, BL, CL, DL, AH, BH, CH, DH) alter the high or low byte in the respective general purpose register.

### Specific Register Commands

Each 8088 general register appears on the Resident Debug Tool display and has an associated RDT change command:

SI	Source Index register command
DI	Destination Index register command
SP	Stack Pointer register command
BP	Base Pointer register command
IP	Instruction Pointer register command

These registers are altered in the same manner as the AX, BX, CX, and DX registers.

### Segment Register Commands

Each 8088 segment register appears on the Resident Debug Tool display and has an associated RDT change command:

CS	Code Segment register command
DS	Data Segment register command
SS	Stack Segment register command
ES	Extra Segment register command

These registers are altered using 20-bit values which represent the 16 bits to actually be placed into the segment register appended with 4 zero bits; this is consistent with the way RDT displays segment registers as 20-bit quantities which represent their true 8088 memory addressing value.

### Flag Register Commands

The 8088 Flag register appears on the Resident Debug Tool display in both its 16-bit value and in the one-bit values of the individual flag bits. The Flag register can be altered bit-by-bit, in its entirety and zeroed using these commands:

FL	Flag register command
AF	Auxiliary Flag command
CF	Carry Flag command
DF	Direction Flag command

<b>IF</b>	Interrupt Flag command
<b>OF</b>	Overflow Flag command
<b>PF</b>	Parity Flag command
<b>SF</b>	Sign Flag command
<b>TF</b>	Trap Flag command
<b>ZF</b>	Zero Flag command

The FL command alters the contents of the entire flag register. The other flag commands set the respective flag bit to 1 or 0.

### Execution Commands

These commands are used to resume user program execution:

<b>EX</b>	EXecute command
<b>ST</b>	STep command
<b>CT</b>	step CounT command

The EX command passes control to the user program. Execution begins at the instruction determined by the sum of the CS and IP registers. An EX command parameter may be used to set the IP value before execution begins.

The ST command executes a specified number of user program instructions. The number of instructions executed is determined by the ST command parameter or, if none is specified, by the step count (CT).

The CT command is used to modify the default step count.

### Variable Commands

Within the Resident Debug Tool, variable commands are used to set and clear variables which specify breakpoints or scratchpad results. In the top portion of the Resident Debug Tool display appears the variables V1-V9 and S1-S9. These variables are used to display breakpoints or scratchpad results.

The S1-S9 variables are initially used as breakpoint variables. They are reserved for breakpoint values but may be deactivated and used as scratchpad variables.

The V1-V9 variables are initially used as scratchpad variables. They are reserved for scratchpad values but may be activated and used as breakpoint variables. In addition, there are a large number of non-displayed variables which may be used to store scratchpad results.

### Breakpoint Variable Commands

The variables V1-V9 and S1-S9 in the top portion of the Resident Debug Tool display are used to display breakpoints or scratchpad results.

The following vector commands may be used to set or clear breakpoints in the V1-V9 and S1-S9 variables:

<b>S0</b>	to set or clear all S1-S9 variables
<b>S1-S9</b>	breakpoint (or scratchpad)
<b>V0</b>	set or clear all V1-V9 variables
<b>V1-V9</b>	breakpoint (or scratchpad)

### Scratchpad Variable Commands

The following vector commands may be used to set or clear scratchpad results in the variables which may be used as scratchpad variables:

<b>S0</b>	to set or clear all S1-S9 variables
<b>S1-S9</b>	scratchpad (or breakpoint)
<b>V0</b>	to set or clear all V1-V9 variables
<b>V1-V9</b>	scratchpad (or breakpoint)
<b>W0</b>	to set or clear all W1-W9 variables
<b>W1-W9</b>	scratchpad variable commands
<b>X0</b>	to set or clear all X1-X9 variables
<b>X1-X9</b>	scratchpad variable commands
<b>Y0</b>	to set or clear all Y1-Y9 variables
<b>Y1-Y9</b>	scratchpad variable commands
<b>Z0</b>	to set or clear all Z1-Z9 variables
<b>Z1-Z9</b>	scratchpad variable commands

The W1-W9, X1-X9, Y1-Y9, and Z1-Z9 variables are not displayed on the Resident Debug Tool display. They are used to store away scratchpad results. They may be used to save displayed V1-V9 or S1-S9 values and later restore these values to the displayed variables.

### Windowing Commands

The Resident Debug Tool's extensive windowing capabilities are among its most powerful features. Commands which may be used to take advantage of these capabilities are separated into four groups according to the four types of windowing: memory, disassemble, trace, and math co-processor.

### Memory Window Commands

Memory windowing allows the display and alter of memory in the windowing area of the Resident Debug Tool display. The following vector commands are used to control memory windowing:

**MW** Memory Window command

**AS** AScii format command

**EB** EBcdic format command

**L0** to set or clear the L1-L9 variables

**L1-L9** windowing variable command

**M0** to set or clear the M1-M5 variables

**M1-M5** windowing variable command

The MW command places the Resident Debug Tool in memory windowing mode.

The AS and EB commands determine whether the character representations of memory are displayed in ASCII or EBCDIC.

The L1-L9 and M1-M5 commands set and clear memory windows.

### Disassemble Window Commands

Disassemble windowing allows the display of disassembled instructions in the windowing area of the Resident Debug Tool display and alter of the instructions' hex representations. The following commands are used to control disassemble windowing:

**DW** Disassemble Window command

**L1-L9** windowing variable command

**M1-M5** windowing variable command

**PD** Print Disassemble command

The DW command places the Resident Debug Tool in disassemble windowing mode.

The L1-L9 and M1-M5 commands set and clear disassemble windows.

The PD command is used to print instructions disassembled from a contiguous block of memory.

### Trace Window Commands

Trace windowing allows the display of instructions which have been placed in the Resident Debug Tool trace buffer while STEpping a user program. The following commands are used to control trace windowing:

**TW** Trace Window command

**TR** TRace options command

**TC** Trace Clear command

**PT** Print Trace command

The Trace command (TR) is used to select the desired trace options. These options determine what instructions are to be put in the trace buffer while a user steps (ST) through a program. The trace buffer is displayed while in trace windowing mode (TW). At any time, the trace buffer may be printed (PT) or cleared (TC).

### Math Co-Processor Window Command

Math co-processor windowing allows the display of the state of the 8087 math co-processor in the windowing area of the Resident Debug Tool display, if a math co-processor is installed. The following command is used to place the Resident Debug Tool in math co-processor windowing mode:

**CW** Math Co-processor Window command

## Memory Commands

In addition to the memory display and alter capability provided while in memory windowing mode, there are several commands which enhance the user's ability to access memory.

### Memory Block Commands

The following commands perform operations on blocks of memory:

**CP** CoPy Memory Command

**CM** Compare Memory Command

**PM** Print Memory Command

The CP command copies a block of memory from one location to another. The size of the copy can be as small as one byte or as large as 64K-1 bytes.

The CM command compares two blocks of memory, and displays differences which exist.

The PM command prints the contents of a specified size block of memory.

### Find Commands

Find commands allow for the search of memory for a specified ASCII, EBCDIC or hexadecimal search string. The following are the commands which perform the find function:

**FA** Find Ascii command

**FE** Find EbcDic command

**FX** Find heXadecimal command

### Input/Output Commands

The input/output commands are used to access disks, diskettes, and I/O ports.

## Disk and Diskette Read/Write Commands

The following commands allow reading from and writing to disks and diskettes:

**RD** Read Disk or Diskette Command

**WD** Write Disk or Diskette Command

The RD command reads a specified number of sectors from a disk or diskette and places the results at a specified location in memory.

The WD command writes a specified number of sectors to a disk or diskette using the data which begins at a specified location in memory.

### I/O Port Commands

The following commands are used to input and output bytes and words from I/O ports:

**IB** Input Byte Command

**IW** Input Word Command

**OB** Output Byte Command

**OW** Output Word Command

A command parameter entered determines the I/O port to be input or output. A value read with an input command (IB, IW) is placed in the displayed V1 variable. The value in V1 is used to write out for an output command (OB,OW).

### Debug Environment Commands

There are several commands which are used to further control the Resident Debug Tool Environment. These commands are divided into screen control, move display, and reset commands.

## Screen Control Commands

These commands alter the appearance of the Resident Debug Tool display screen:

- CD** Color Display command
- MD** Monochrome Display command
- CA** Color Attributes command
- PR** Print user screen command
- SC** toggle user SScreen command

The CD and MD commands allow the debug tool to be dynamically switched from one display to another when a user has both types of displays attached.

The CA command allows the user to modify the foreground and background colors used for the debug screen when on a color display.

The PR command may be used to print the users screen.

The SC command toggles between the user program display screen and the debug tool display screen when both are on the same display.

## Move Display Commands

The Resident Debug Tool supports nine display screens. These commands are used to switch from one display screen to another:

### D1-D9 display screen command

Each display screen has its own unique set of breakpoints, displayed scratchpad variables, and window variables. Portions of a user's program may be monitored on separate screens with separate breakpoints and variables.

## Reset Commands

These commands are used to alter the initial Resident Debug Tool load options:

- RK** Reset 'K' load option command
- SK** Set 'K' load option command
- RN** Reset 'N' load option command
- SN** Set 'N' load option command
- SR** System Reset command

The SK and RK commands set and reset the keyboard interrupt option.

The SN and RN commands set and reset the non-maskable interrupt option.

The SR command performs a system reset by exiting the Resident Debug Tool and transferring control to the BIOS entry point to reboot the system.

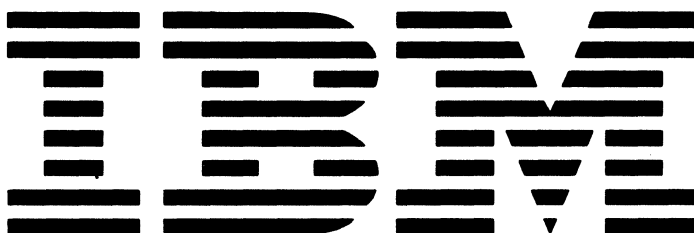


## **Appendix A**

### **Sample RDT Display Screens**

This appendix contains sample RDT display screens.

**IBM** *Personal Computer*



**Personal Computer**

**Resident Debug Tool  
Version 1.00**

**(C) Copyright IBM Corp 1983, 1984  
Written by  
Anthony D. Hooten,  
John M. Van Buren and Gordon W. Arbeitman**

**Press any key to continue**

**IBM *Personal Computer*****Display Line**

<b>1</b>	<b>Release Number , Title , Release Date</b>
<b>2-9</b>	<b>Breakpoints and Scratchpad Variables Processor Registers and Current Instruction Disassembly</b>
<b>10</b>	<b>Command Line</b>
<b>11</b>	<b>Message Line</b>
<b>12-25</b>	<b>Windowing Area Memory, Disassemble, Trace Partial, Trace Full or Math Coprocessor</b>

# IBM Personal Computer

```

Rel 1.00          IBM PERSONAL COMPUTER RESIDENT DEBUG TOOL          D1          07/01/84
V1:..... V2:..... V3:..... V4:..... V5:..... V6:..... V7:..... V8:..... V9:.....
S1:..... S2:..... S3:..... S4:..... S5:..... S6:..... S7:..... S8:..... S9:.....
                                DISPLAY: ASCII          WINDOW: MEMORY

AX: 0000 BX: 0000 CX: 00FF DX: 111E          TR: 00 .....-.....
SP: 0200 BP: 0000 SI: 0000 DI: 0100          FL: F246 OF: 0 DF: 0 IF: 1 TF: 0
CS: 112E0 DS: 111E0 SS: 112F0 ES: 111E0          SF: 0 ZF: 1 AF: 0 PF: 1 CF: 0
          LC:.....          INT          3          OP: .....
          IP: 0000          EX: 112E0          CC          STEP CT: 0001 CO:.....

==>

L1  * 00000 4331E300 3F017000 71040E06 C3040E06 *C1...p.q.....*
L2  00010 3F017000 CC040E06 23FF00F0 23FF00F0 *..p.....*
L3  00020 A5FE00F0 96070E06 23FF00F0 23FF00F0 *.....*
L4  00030 23FF00F0 600700C8 57EF00F0 3F017000 *.....W....p..*
L5  00040 65F000F0 4DF800F0 41F800F0 560200C8 *e...M...A...V...*
L6  00050 39E700F0 59F800F0 2EE800F0 D2EF00F0 *9...Y.....*
L7  00060 000000F6 860100C8 6EFE00F0 38017000 *.....n...8.p..*
L8  00070 4BFF00F0 A4F000F0 22050000 000000F0 *K.....*
L9  00080 FB08E300 80014205 8C024205 99024205 *.....B...B...B..*
M1  00090 E2044205 D414E300 2115E300 E727E300 *...B.....*
M2  000A0 070CE300 26017000 00000000 00000000 *.....p.....*
M3  000B0 00000000 00000000 6D034205 00000000 *.....m.B.....*
M4  000C0 EA080CE3 00000000 00000000 00000000 *.....*
M5  000D0 00000000 00000000 00000000 00000000 *.....*

```

# IBM Personal Computer

```

Rel 1.00          IBM PERSONAL COMPUTER RESIDENT DEBUG TOOL          D1          07/01/84
V1:..... V2:..... V3:..... V4:..... V5:..... V6:..... V7:..... V8:..... V9:.....
S1:..... S2:..... S3:..... S4:..... S5:..... S6:..... S7:..... S8:..... S9:.....
                                DISPLAY: ASCII          WINDOW: MEMORY

AX: 0000 BX: 0000 CX: 00FF DX: 111E          TR: 00 .....-.....
SP: 0200 BP: 0000 SI: 0000 DI: 0100          FL: F246 OF: 0 DF: 0 IF: 1 TF: 0
CS: 112E0 DS: 111E0 SS: 112F0 E8: 111E0          SF: 0 ZF: 1 AF: 0 PF: 1 CF: 0
LC:.....          INT 3          OP: .....
IP: 0000 EX: 112E0 CC          STEP CT: 0001 CO:.....

==>

L1 *00000 4331E300 3F017000 71040E08 C3040E06 *C1 .?p. q t.....*
L2 00010 3F017000 CC040E08 23FF00F0 23FF00F0 *?p. t ..... ≡# . .
L3 00020 A5FE00F0 96070E08 23FF00F0 23FF00F0 . = . .
L4 00030 23FF00F0 600700C8 57EF00F0 3F017000 . .....*
L5 *05040 00000000 00000000 00000000 00000000 . .....*
L6 05050 00000201 01010002 70000C00 63010205 *€ | € 0 ..... | f 3 *
L7 05060 00BD0370 00FD007D 42E30000 00000000 * ≡ # 5 6 F Y ) . .
L8 05070 00000000 00000000 00000000 00000000 . .....*
L9 05080 00000000 00000000 00000000 00000000 . .....*
M1 *05390 FF3696EE 9A030059 068D66F4 5DCA0800 . .....*
M2 053A0 558BEC83 C50681EC 0800C606 C6F320C6 *C1π ? p. q t.....*
M3 053B0 06C7F320 C606C8F3 20C606C9 F320C606 *?p. t ..... ≡# . .
M4 053C0 CAF320C6 06CBF330 C746F806 00837E00 *N ≡.....Wn ≡? p.. .
M5 053D0 00743583 7EF8017C 2FB90A00 8B460033 * # / _ ~ | < U +.*

```

**IBM Personal Computer**

```

Rel 1.00                IBM PERSONAL COMPUTER RESIDENT DEBUG TOOL        D1          07/01/84
V1:..... V2:..... V3:..... V4:..... V5:..... V6:..... V7:..... V8:..... V9:.....
S1:..... S2:..... S3:..... S4:..... S5:..... S6:..... S7:..... S8:..... S9:.....

                                DISPLAY: ASCII        WINDOW: DISASM

AX:  0100  BX:  0000  CX:  0007  DX:  007F          TR: 01 00000 - 00000
SP:  0BB8  BP:  0000  SI:  004C  DI:  01AA          FL: F246  OF: 0  DF: 0  IF: 1  TF: 0
CS:  00E30  DS:  00E30  SS:  00E30  ES:  00E30          SF: 0  ZF: 1  AF: 0  PF: 1  CF: 0
LC:  .....          CALL  047FA                      OP: 047FA
IP: 3988      EX: 047B8  E83F00          STEP CT: 0001 CO: .....

==>

L1:  *00E30:3988  E83F00  CALL  047FA  047FA
L2:  00E30:398B  72F7    JB    047B4  047B4
L3:  00E30:398D  268A05  MOV   AL,ES:(DI)  00FDA=61
L4:  00E30:3990  3CFF    CMP   AL,FF
L5:  00E30:3992  74F0    JZ    047B4  047B4
L6:  *00E30:2994  4A      DEC   DX
L7:  00E30:2995  4A      DEC   DX
L8:  00E30:2996  D3E2    SHL   DX,CL
L9:  00E30:2998  0AD3    OR    DL,BL
M1:  00E30:299A  26C3560B  ADD   DX,ES:(BP+0B)  00E3B=6F20
M2:  00E30:299E  59      POP   CX
M3:  00E30:299F  C3      RET
M4:  *00E30:39A0  1201    ADC   AL,DS:(BX+DI)  00FDA=61
M5:  00E30:39A2  33F6    XOR   SI,SI

```

# IBM Personal Computer

```

Rel 1.00          IBM PERSONAL COMPUTER RESIDENT DEBUG TOOL          D1          07/01/84
V1:..... V2:..... V3:..... V4:..... V5:..... V6:..... V7:..... V8:..... V9:.....
S1:..... S2:..... S3:..... S4:..... S5:..... S6:..... S7:..... S8:..... S9:.....
                                DISPLAY: ASCII          WINDOW: TRACEP
AX: 0100 BX: 0000 CX: 0007 DX: 007F          TR: 01 00000 - 00000
SP: 0BB8 BP: 0000 SI: 004C DI: 01AA          FL: F246 OF: 0 DF: 0 IF: 1 TF: 0
CS: 00E30 DS: 00E30 SS: 00E30 ES: 00E30          SF: 0 ZF: 1 AF: 0 PF: 1 CF: 0
          LC:....          CALL 047FA          OP: 047FA
          IP: 3988          EX: 047B8          E83F00          STEP CT: 0001 CO:.....

==>

TB: 000D 00E30:3094 CD28 INT 28
TB: 000C 00E30:0C07 CF IRET
TB: 000B 00E30:3096 9D POPF
TB: 000A 00E30:3097 C3 RET
TB: 0009 00E30:357F 74FB JZ 043AC 043AC
TB: 0008 00E30:357C E819FB CALL 03EC8 03EC8
TB: 0007 00E30:3098 53 PUSH BX
TB: 0006 00E30:3099 33DB XOR BX,BX
TB: 0005 00E30:309B E87CEF CALL 02E4A 02E4A
TB: 0004 00E30:201A 16 PUSH SS
TB: *0003 00E30:201B 1F POP DS
TB: 0002 00E30:201D 57 PUSH DI
TB: 0001 00E30:201E E86619 CALL 047B7 047B7
TB: 0000 00E30:3987 50 PUSH AX

```

# IBM Personal Computer

```

Rel 1.00          IBM PERSONAL COMPUTER RESIDENT DEBUG TOOL      D1      07/01/84
V1:..... V2:..... V3:..... V4:..... V5:..... V6:..... V7:..... V8:..... V9:.....
S1:..... S2:..... S3:..... S4:..... S5:..... S6:..... S7:..... S8:..... S9:.....
                                DISPLAY: ASCII      WINDOW: TRACEF
AX: 0100  BX: 0000  CX: 0007  DX: 007F      TR: 01 00000 - 00000
SP: 0BB8  BP: 0000  SI: 004C  DI: 01AA      FL: F246  OF: 0  DF: 0  IF: 1  TF: 0
CS: 00E30 DS: 00E30 SS: 00E30 ES: 00E30      SF: 0  ZF: 1  AF: 0  PF: 1  CF: 0
      LC:....      CALL 047FA      OP: 047FA
      IP: 3988      EX: 047B8  E83F00      STEP CT: 0001 CO:.....

==>

CS: 00E30 DS: 00E30 ES: 00E30 SS: 00E30 AX: 0100 BX: 0000 CX: 0007 DX: 007F
IP: 201D SI: 004C DI: 01AA SP: 0BBE BP: 0000 FL: F246      EX: 02E4D
TB: 0002 00E30: 201D 57      PUSH DI

CS: 00E30 DS: 00E30 ES: 00E30 SS: 00E30 AX: 0100 BX: 0000 CX: 0007 DX: 007F
IP: 201E SI: 004C DI: 01AA SP: 0BBC BP: 0000 FL: F246      EX: 02E4E
TB: 0001 00E30: 201E E86619      CALL 047B7      047B7

CS: 00E30 DS: 00E30 ES: 00E30 SS: 00E30 AX: 0100 BX: 0000 CX: 0007 DX: 007F
IP: 3987 SI: 004C DI: 01AA SP: 0BBA BP: 0000 FL: F246      EX: 047B7
TB: 0000 00E30: 3987 50      PUSH AX

```



**IBM Personal Computer**

Rel 1.00 IBM PERSONAL COMPUTER RESIDENT DEBUG TOOL D1 07/01/84

V1:..... V2:..... V3:..... V4:..... V5:..... V6:..... V7:..... V8:..... V9:.....  
S1:..... S2:..... S3:..... S4:..... S5:..... S6:..... S7:..... S8:..... S9:.....

DISPLAY: ASCII WINDOW: COPROC

AX: 0000 BX: 0000 CX: 00FF DX: 11BE TR: 00 .....-.....

SP: FFFE BP: 0000 SI: 0100 DI: 0100 FL: F246 OF: 0 DF: 0 IF: 1 TF: 0

CS: 11BE0 DS: 11BE0 SS: 11BE0 ES: 11BE0 SF: 0 ZF: 1 AF: 0 PF: 1 CF: 0

LC: .... WAIT OP: .....

IP: 0108 EX: 11CE8 9B STEP CT: 0001 CO: .....

==&gt;

## MATH COPROCESSOR STATE

CONTROL WORD: 03FF IC: 0 RC: 00 PC: 11 IEM: 1 PM: 1 UM: 1 OM: 1 ZM: 1 DM: 1 IM: 1

STATUS WORD: 4100 B: 0 CC: 1001 ST: 000 IR: 0 PE: 0 UE: 0 OE: 0 ZE: 0 DE: 0 IE: 0

		SIGN	EXPONENT	SIGNIFICAND				TAG
	ST	0	0000	0000	0000	0000	0000	11
EXCEPTION POINTERS	ST (1)	0	0000	0000	0000	0000	0000	11
	ST (2)	0	0000	0000	0000	0000	0000	11
INSTRUCTION ADDRESS : 11D02	ST (3)	0	0000	0000	0000	0000	0000	11
INSTRUCTION OPCODE : D9EE	ST (4)	0	0000	0000	0000	0000	0000	11
OPERAND ADDRESS : 00000	ST (5)	0	0000	0000	0000	0000	0000	11
	ST (6)	0	0000	0000	0000	0000	0000	11
	ST (7)	0	0000	0000	0000	0000	0000	11

# Notes

IBM Corporation  
Editor, IBM Personal Computer Seminar Proceedings  
4629  
Post Office Box 1328  
Boca Raton FL 33432



