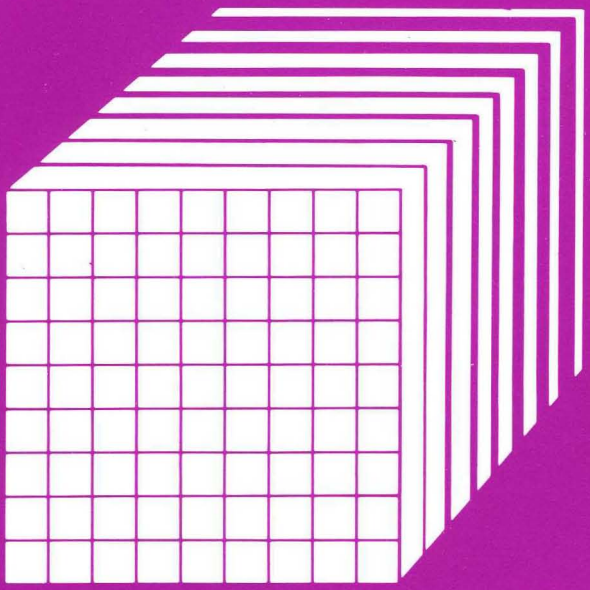
IBM Virtual Machine/ Personal Computer Service Aids

Productivity Family



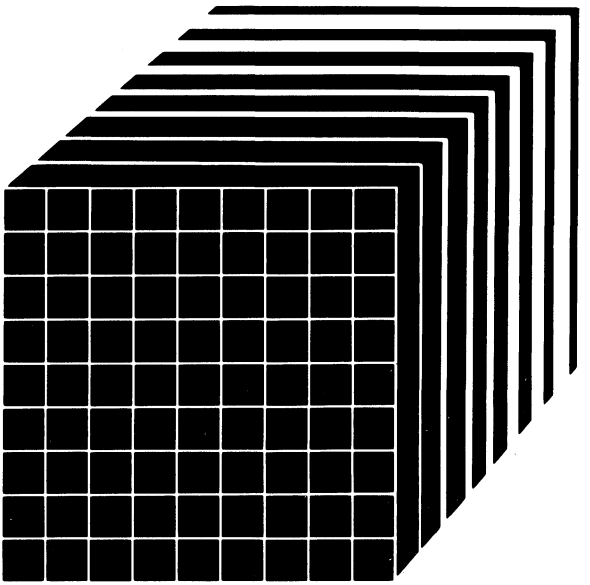
IBM

Personal
Computer
Software

6137796

IBM Virtual Machine/ Personal Computer Service Aids

Productivity Family



IBM

Personal Computer Software

First Edition (December 1984)

International Business Machines Corporation provides this manual "as is" without warranty of any kind, either express or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this manual at any time and without notice.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Products are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM Marketing Representative.

Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication.

A Product Comment Form is provided at the back of this publication. If this form has been removed, address comments to IBM Corporation, Department G60, P. O. Box 6, Endicott, New York 13760. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligations whatever.

Preface

This book is for users who need to debug VM/PC software. Our goal is to explain the available debug tools, and the type of problems they may help solve. Toward this end, the book is divided into four parts:

Part 1. Introduction - describes the tools that you can use to debug VM/PC software, and when each may be useful.

Part 2. Using VMPCDEB - describes VMPCDEB, a debugging tool for the CPIO portion of VM/PC. This section gives some general information about how VMPCDEB works, and then describes all the available commands in detail.

Part 3. Using CPIO DUMPER - describes how to use the dump subroutine of CPIO, and how to reformat the dump into files that can be read by VMPCDEB.

Part 4. Using the 370 Processor Control Session - describes how to use the VM/PC 370 Processor Control Session to debug the 370 code portion of VM/PC.

Related Publications

When using this book you may have to refer to other books, because we assume you have a basic understanding of the hardware and of DOS. The list below contains titles of books you may find useful.

IBM DOS (Disk Operating System)

IBM Guide to Operations - Personal Computer XT

IBM Guide to Operations - Personal Computer XT/370

IBM Guide to Operations - Personal Computer AT

IBM Guide to Operations - Personal Computer AT/370

IBM Virtual Machine/Personal Computer User's Guide

IBM Virtual Machine/System Product CMS Command and Macro Reference, ST00-1357

Contents

Part 1. Introduction

Chapter 1. Debugging Tools

What Debugging Tools Are Available?

VM/PC has several debugging tools you can use. They are:

1. **VMPCDEB**
   VMPCDEB is a debug facility designed to load, control, and provide debug facilities for the VM/PC CPIO program. The majority of this book is devoted to describing VMPCDEB. For more information about VMPCDEB, see "Part 2. Using VMPCDEB."

2. **CPIO Dump Routine and the VM/PC Reformatter**
   The CPIO Dump Routine (also known as DUMPER) is a subroutine of CPIO that lets you create memory dumps of your system. The VM/PC Reformatter (also known as VMPCREF) takes the memory dump created by DUMPER and reformats it for VMPCDEB. Both of these tools are described in this book starting on Page 13-1.

3. **370 Processor Control Session**
   The 370 Processor Control Session is a general purpose debug facility similar to those you would find on other IBM 370 processors. It lets you to start and stop the processor, step through instructions, compare addresses, and display and alter memory and registers. The 370 Processor Control Session is explained in more detail starting on Page 15-1 of this book.

4. **CMS DEBUG**
   CMS DEBUG is a CMS command that puts you into the CMS debug environment. Once you're in this environment, you can use subcommands to set breakpoints, display and alter memory locations and registers, define symbols, and create memory dumps. CMS DEBUG is not described in this book. For more information about CMS DEBUG see the *VM/PC User's Guide* or the *VM/SP CMS Command and Macro Reference.*

5. **DOS DEBUG**
   DOS DEBUG is a special program designed to load, control, and provide debug facilities for DOS programs. DOS DEBUG is not described in this book. For more information about DOS DEBUG see the *DOS* manual.

Which Debugging Tool Should I Use?

Kind of program	Debugging Tool
DOS	DOS DEBUG
CPIO	VMPCDEB
VM/370 code (CP and CMS)	370 Processor Control Session
VM Application	CMS DEBUG

Debugging With A Memory Dump

If you cannot resolve your problem with the tools listed in the table, you can create a memory dump using a subroutine of CPIO. You can use this subroutine (called "DUMPER") when you enter a special option with the VMPC command. DUMPER is described in Chapter 13, "Starting DUMPER" starting on Page 13-1.

The dump subroutine was written using BIOS routines to keep the amount of critical memory down to a minimum. The three critical areas of memory for a successful dump are the:

- CPIO keyboard interrupt handler
- CPIO dump subroutine
- Keyboard interrupt vector (4 bytes in low memory)

If these three areas of memory (totalling about 2K bytes) have not been overwritten, and the keyboard interrupt level has not been masked, you can create a dump. This dump will be all of real memory and will be dumped onto diskettes.

If you want to reload this dump under VMPCDEB, there is a special reformatting program (VMPCREF) which will reformat the dump into files which can be used by VMPCDEB.

Installing the Debugging Tools

Many people who order VM/PC will not be interested in using the debugging tools. For this reason, installing these tools is not part of the normal VM/PC installation process. If you want to use these tools, you can install them at any point after you've installed VM/PC. You only have to install the debugging tools once, because after you install them, they'll reside on your fixed disk.

There are eight diskettes at the back of this book. The first seven contain the VM/PC code. The eighth diskette contains the debugging tools. It's labeled "VM/PC Service Aids."

To install the tools, you must be in DOS. If you're not already using DOS, do so now. Then insert the Service Aids diskette into diskette drive A and enter the following command:

```
C>copy a:*.* c:
```

When you see the DOS prompt, the installation is complete. You can take the diskette out and start debugging.

Three files have been copied onto your fixed disk:

1. **VMPCDEB.COM**
   Contains the VMPCDEB debugging program. This program is designed to load, control, and provide debug facilities for the VM/PC CPIO program. A description of the VMPCDEB program starts on Page 2-1 and continues through Page 12-5.

2. **IWSUNPIK.DEB**
   Contains dis-assembly code used by VMPCDEB.

3. **VMPCREF.EXE**
   Contains the VMPCREF program. This program reformats memory dumps taken by the VM/PC CPIO dump routine. A description of the CPIO dump routine starts on Page 13-1 and a description of the VMPCREF program starts on Page 14-1.

Part 2. Using VMPCDEB

Chapter 2. An Introduction to VMPCDEB

VMPCDEB is a special program designed to load, control, and provide debug facilities for the VM/PC CPIO program. In many ways, it is similar to the DOS DEBUG program. In fact, VMPCDEB and DEBUG are so much alike that running both at the same time may cause problems. Both VMPCDEB and DEBUG use the interrupt locations for "single step," "break-in," "program terminate," and "INT3" (the trap instruction). Also, both place INT3 (break-point) instructions in the program you are testing.

VMPCDEB lets you:

- Load programs
- Start programs
- Run programs in "step mode"
- Insert break-points
- Alter memory locations and registers
- Print memory locations and registers
- Produce an annotated listing of the VM/PC common area
- Take memory dumps
- Dis-assemble code areas.

Once the program under test is running, VMPCDEB will regain control only if one of the following events occurs:

- Program terminate (INT 20)
- Break-in
- Non-maskable interrupt
- Error trap (INT 60) within CPIO
- Break-point instruction
- After the specified number of instructions when in step mode.

These events are described in more detail under Chapter 7, "Halting a Program" on page 7-1.

Special Note for PC XT/370 users:

If you have:

3277 Emulator card	you'll be debugging a 3277 session
3278 Emulator card	you'll be debugging a 3278 session
both	you'll be debugging a 3278 session. The 3278 Emulator card takes precedence over the 3277 Emulator card. If you want to debug a 3277 session, you must remove the 3278 card before you start VMPCDEB.

Chapter 3. Starting VMPCDEB

VMPCDEB runs under DOS, ***not*** VM/PC. To start the VMPCDEB program, you enter the VMPCDEB command after the DOS prompt. Its format is:

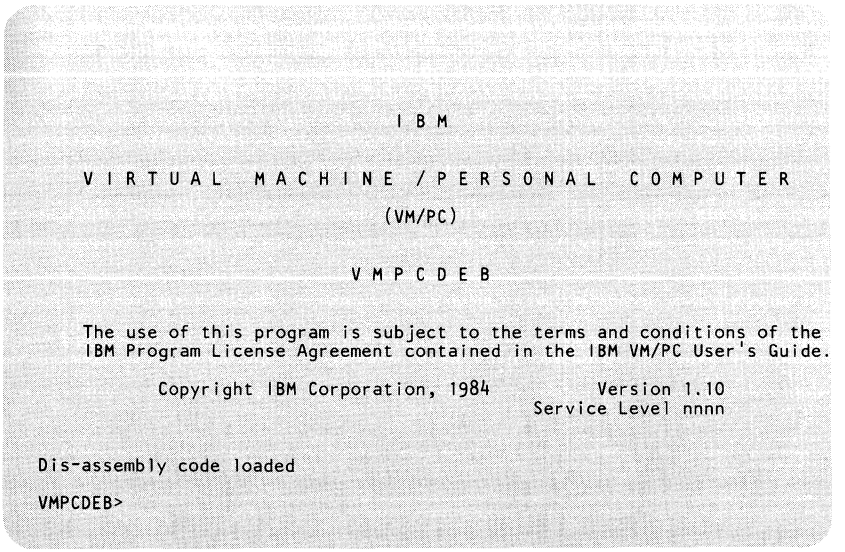
```
[d:]VMPCDEB
```

Where:

`d:`

is the disk id where the VMPCDEB program resides. If VMPCDEB resides on the default disk drive, you can omit this field.

Once you enter this command, you'll see:



This is the VMPCDEB Logo Screen. The `VMPCDEB>` at the bottom of the screen indicates you are in *command input mode*. This means you can now enter any valid

VMPCDEB command. There is a summary of all VMPCDEB commands on Page 12-1 and "Command Reference" on Page 11-7 lists each of these commands in alphabetical order and describes them in detail.

When you start VMPCDEB, it:

- saves the value of the IMR (Interrupt Mask Register) for later use.
- makes a copy of the current DOS interrupt vector to use while VMPCDEB is running. VMPCDEB makes the following changes in this copied interrupt vector:
  - Uninitialized entries (entries which equal zero) point to an INTRET instruction within BIOS (at hexadecimal F000:FF53).
  - Interrupts '0A'X and '0B'X (interrupt locations for the device emulation adapter controller) also point to the INTRET instruction mentioned above
  - Interrupt '0F'X takes on the value of interrupt '08'X (the timer interrupt location).
- loads the dis-assembly code, if the file IWSUNPIK.DEB (containing the dis-assembly code) has been installed.

Chapter 4. Stopping VMPCDEB

To stop the VMPCDEB program, use one of these two commands:

END

QUit

Both commands work in exactly the same way. They stop VMPCDEB from running and return you to DOS.

When you enter an END or a QUIT command, VMPCDEB:

- resets the device emulation adapter and issues a "reset disk" request to BIOS, if bit 0 and bit 6 at location '250'X equal one.
- restores the system memory size to its original value (or to the value set by a MEMSIZE command, if the two values are different).

Chapter 5. Loading a Program

There are three commands you can use to load a program:

Command	Description
LOAD Page 11-98	Loads a program you want debugged. There are only two file extensions you can use: COM (the default) or DUM. If you use a file extension other than COM or DUM (or you omit the field), VMPCDEB loads the program as if it were a COM file. When you enter this command, the system loads the file, but you have to enter a GO command (Page 11-81) to start execution.
RUN Page 11-126	Loads a program you want debugged and immediately starts it for you. There are only two file extensions you can use: COM (the default) or DUM. If you use a file extension other than COM or DUM (or you omit the field), VMPCDEB loads the program as if it were a COM file.
L370 Page 11-99	Loads 370 processor memory dumps. The only file extension you can use is "370." When you enter this command, the system loads the file and immediately starts it for you.

Loading a COM File

Note: The following section also applies to any file with an extension other than COM or DUM. The system treats these files as if they were COM files.

A file with an extension of COM is a standard DOS COM file. This means the file is a straight copy of memory (from location '100'X on) which does not contain any control information.

When you load a COM file, using either a LOAD or a RUN command, VMPCDEB:

- will load a file that is larger than 64K bytes. But VMPCDEB will not load a file that won't fit into available memory. It checks to see that the file won't overwrite the transient portion of the command process (the top 16K bytes of memory).
- restores the system memory size to its original value (or to the value set by a MEMSIZE command, if the two values are different).
- sets up a program segment prefix at the next segment boundary (some multiple of 16 bytes) above the top of VMPCDEB. Then VMPCDEB loads your program into the segment as a straight copy from disk into memory, starting at location '100'X within the segment.

- sets up the registers as follows:

AX	'FFFF'X
BP	'0000'X
BX	'0000'X
CS	segment address of the program segment prefix
CX	'0000'X
DI	'0000'X
DS	segment address of the program segment prefix
DX	'0000'X
ES	segment address of the program segment prefix
FLAGS	'0200'X
IP	'0100'X
SI	'0000'X
SP	'00FE'X
SS	segment address of the program segment prefix

- looks for "filename.SYM" (a symbol table). If it finds one, VMPCDEB loads the table into its own memory. VMPCDEB stops loading this table if it reads a record which *does not* start with a percent sign (%).

  Once this symbol table is loaded, you can use symbol names in commands which require addresses. Whenever VMPCDEB displays an address, it will be of the form:

  ```
  symbol:offset
  ```

- cancels all break-points.

- clears all interrupt traps.
- turns step mode off.
- resets the device emulation adapter and issues a "reset disk" request to BIOS, if bit 0 and bit 6 at location '250'X equal one.
- If the device emulation adapter interrupt handler is active, VMPCDEB uses its own timer interrupt handler (instead of the BIOS timer interrupt handler). The VMPCDEB timer interrupt handler does not re-enable the diskette drive DMA (Direct Memory Access) when it turns off the diskette drive motors, as the BIOS timer does. The BIOS timer checks the diskette drive motor count. Every time it expires, BIOS turns off the diskette drive motors which makes the diskette drive controller enable its DMA and interrupts. In VMPCDEB, this would interfere with the operation of the device emulation adapter, so VMPCDEB uses its own timer interrupt handler.
- resets the IMR (Interrupt Mask Register) to its original value, except that the device emulation adapter interrupts (interrupt levels 2 and 3) are disabled and diskette interrupts are enabled.
- maintains two copies of interrupt vector entries 0-127. One of these is used when VMPCDEB is running, and the other is used when your program is running. When you start and stop your program, VMPCDEB swaps these two interrupt vector entries.
- activates a keyboard interrupt handler which checks for break-ins. Anything that is not a break-in is passed on to the address specified in the previous contents of the keyboard interrupt location (normally BIOS). The keyboard interrupt handler also checks to see if your program altered the keyboard interrupt address (to intercept keyboard interrupts).

- displays the base address and size of your program when it finishes loading. Both the address and the size are displayed as absolute 20-bit hexadecimal addresses. The size is always rounded up to the next complete paragraph.

Loading a DUM File

A file with an extension of DUM is a memory dump file. This file consists of a 768-byte header followed by the memory dump. If you used the VMPCDEB DUMP command (Page 11-66) to create this file, it will be in the correct format. If you used the VM/PC Dump facility to create this file, you must reformat the dump (using VMPCREF) before loading it. See Page 13-1 for more information about the VM/PC Dump facility.

When you load a DUM file, using either a LOAD or a RUN command, VMPCDEB:

- will not load a file that won't fit into available memory. VMPCDEB checks to see that the file won't overwrite the transient portion of the command process (the top 16K bytes of memory). If the amount of memory above VMPCDEB is less than the size of the DUM file, the command will fail.

  If the amount of memory above VMPCDEB is greater than the size of the DUM file, VMPCDEB adjusts the size of the last memory block so that the size of the DUM file equals the size of memory above VMPCDEB.

- restores the system memory size to its original value (or to the value set by a MEMSIZE command, if the two values are different).

- loads the dump into memory immediately above VMPCDEB.

- enables the address relocation facility (described on Page 5-10), if the loaded address of the dump is different than the original address of the program (before the dump was taken).

In general, these two addresses are different. When they are different, you cannot run the dump after it's been reloaded, because VMPCDEB can't find all the segment addresses which need to be adjusted to allow for program relocation.

- cancels all break-points.
- clears all interrupt traps.
- turns step mode off.
- resets the device emulation adapter and issues a "reset disk" request to BIOS, if bit 0 and bit 6 at location '250'X equal one.
- If the device emulation adapter interrupt handler is active, VMPCDEB uses its own timer interrupt handler (instead of the BIOS timer interrupt handler). The VMPCDEB timer interrupt handler does not re-enable the diskette drive DMA (Direct Memory Access) when it turns off the diskette drive motors, as the BIOS timer does. The BIOS timer checks the diskette drive motor count. Every time it expires, BIOS turns off the diskette drive motors which makes the diskette drive controller enable its DMA and interrupts. In VMPCDEB, this would interfere with the operation of the device emulation adapter, so VMPCDEB uses its own timer interrupt handler.
- resets the IMR (Interrupt Mask Register) to its original value, except that the device emulation adapter interrupts (interrupt levels 2 and 3) are disabled and diskette interrupts are enabled.
- maintains two copies of interrupt vector entries 0-127. One of these is used when VMPCDEB is running, and the other is used when your program is running. When you start and stop your program, VMPCDEB swaps these two interrupt vector entries.

- activates a keyboard interrupt handler which checks for break-ins. Anything that is not a break-in is passed on to the address specified in the previous contents of the keyboard interrupt location (normally BIOS). The keyboard interrupt handler also checks to see if your program altered the keyboard interrupt address (to intercept keyboard interrupts).

- if bit 2 of byte 22 in the dump file header equals one, VMPCDEB sets the registers equal to bytes 31-58 of the header. VMPCDEB adjusts the contents of the four segment registers (CS, DS, ES, and SS) to allow for program relocation.

  If this bit equals zero, VMPCDEB sets the registers as follows:

AX	'FFFF'X
BP	'0000'X
BX	'0000'X
CS	segment address of the program segment prefix
CX	'0000'X
DI	'0000'X
DS	segment address of the program segment prefix
DX	'0000'X
ES	segment address of the program segment prefix
FLAGS	'0200'X
IP	'0100'X
SI	'0000'X

SP	'00FE'X
SS	segment address of the program segment prefix

- if bit 3 of byte 22 in the dump file header equals one, VMPCDEB sets the interrupt vectors equal to bytes 256-767 of the header. VMPCDEB adjusts segment addresses to allow for program relocation.

  If this bit equals zero, VMPCDEB sets interrupt vectors as if dump was a COM program.

- adjusts the addresses in the memory control blocks to allow for program relocation.

- checks to see if the memory control blocks in the dump file are corrupt. If they are, VMPCDEB displays a warning message, places the reloaded dump program in a single memory block occupying all the available memory, and may or may not adjust the addresses in the memory control blocks of the loaded dump file to allow for program relocation.

- displays the base address and size of your program when it finishes loading. Both the address and the size are displayed as absolute 20-bit hexadecimal addresses. The size is always rounded up to the next complete paragraph.

Address Relocation

Note: Address (or "program") relocation applies only to reloaded memory dumps (files with an extension of DUM). It has no effect on COM files, or any other type of program.

In most cases, you will probably reload a memory dump to an address different from that which it was dumped. When this happens, the addresses in the dump must be "relocated" to reflect this difference. VMPCDEB has an address relocation facility which will add (or subtract) the relocation value for you. This facility makes it appear as though the reloaded memory dump is located at the same base address as the original program.

If you reload the memory dump to a different address, then the system automatically enables the address relocation facility. When this facility is enabled, VMPCDEB adds (or subtracts) the relocation value to any address you specify which includes a numeric segment portion. If the address includes a segment portion which is an expression involving a register value, the relocation value *is not* added (or subtracted). Address relocation does not affect offsets relative to segment addresses.

You can enable or disable address relocation yourself by using the RELOC command (Page 11-122).

Note: When the address relocation facility is enabled:

1. if you reload a program and the relocation value is `R`, then you specify address `X:Y`, VMPCDEB automatically relocates the address to address `X+R:Y` in real memory. If you want to refer to *real* address `X:Y`, you must disable address relocation first (using the RELOC command).

2. when you display or dis-assemble areas of memory, VMPCDEB subtracts the relocation value from the segment portion of the address.

3. when you print interrupt vector entries, if the segment address points to within the reloaded dump, then VMPCDEB subtracts the relocation value from the segment portion of the address which is displayed.

4. when you print the CPIO system common header, VMPCDEB subtracts the relocation value from the segment part of the 32-bit pointers.

5. when you print CPIO system common SFVs, DQEs, RQEs, FCBs, PCIBs, the timer blocks, the patch area, and the status area, VMPCDEB subtracts the relocation value from the segment portion of the printed addresses, except for the "return address" in a PCIB.

Chapter 6. Starting a Program

Once you've loaded your program, you can start execution by entering:

```
Go
```

The system starts with the instruction pointed to by CS:IP and runs until one of the events listed in Chapter 7, "Halting a Program" on page 7-1 occurs. When any of these events occurs, VMPCDEB returns to *command input mode*. When you're in command input mode, you'll see:

```
VMPCDEB>
```

At this point, you can enter any valid VMPCDEB command (one of which is the GO command).

Note: There is another way to start your program: entering a decimal integer. If you enter an integer, instead of a GO command, VMPCDEB executes that number of instructions and then returns to command input mode.

For example, if you enter:

```
27
```

VMPCDEB runs for 27 instructions. The program may stop before 27 instructions if, for example, a break-in occurs. This does not affect the current setting of step mode.

Chapter 7. Halting a Program

When a program is halted, VMPCDEB regains control and you are put into *command input mode*. When you're in command input mode, you'll see the VMPCDEB prompt:

```
VMPCDEB>
```

When you see this prompt, you can enter any valid VMPCDEB command. (These commands are listed alphabetically starting on Page 11-8.) You can restart execution any time you want using the VMPCDEB GO command.

Whenever a program is halted, VMPCDEB displays a message indicating the kind of program interruption that occurred. VMPCDEB also displays the values in the registers. These register values are saved and will be restored when you restart the program. If dis-assembly code was loaded, VMPCDEB also displays the program's next instruction in dis-assembled form.

There are several ways to halt a program:

- Normal program termination
- Break-ins
- Non-maskable interrupts
- Error traps
- Break-points
- Step mode
- VMPCDEB function entry.

Each of these methods is described in more detail in the following pages.

Normal Program Termination

The normal way to terminate (or "exit") a program is to insert an interrupt hex 20 (INT 20) as the last instruction in your program. When VMPCDEB encounters this instruction, the program terminates and you're put back into command input mode.

Break-ins

You can "break-in" on the execution of a program by pressing one of the following key combinations:

- **Ctrl-Break** - When you break-in on a program using the Ctrl-Break keys, VMPCDEB performs an address check. This is to insure that the break-in did not occur within DOS or BIOS. If you try to break-in while in DOS or BIOS, VMPCDEB will ignore the break-in, because problems can arise in these situations. Always try to use Ctrl-Break to break-in on a program. If Ctrl-Break is unsuccessful and you really must break-in, you can use Alt-NumLock instead.

  Important Notes:

  1. If you press the Ctrl-Break keys while VMPCDEB is printing, the printer will stop and you'll return to command input mode. When you restart the program, the printer starts where it left off.

  2. If you break into VMPCDEB from VM/PC, you may lose interrupts. This can cause the system to fail or to lock up. If this happens, you'll have to turn your machine off and turn it back on again before continuing.

- **Alt-NumLock** - When you break-in on a program using the Alt-NumLock keys, no address check is performed. This means that you can use Alt-NumLock in situations where Ctrl-Break is ineffective. But, remember that you are circumventing the safeguard (the address check) and problems may arise. Always try Ctrl-Break first.

You can only use break-ins when your program is running. If you try to break-in while VMPCDEB has control, you'll see the following message:

```
Break-in - use the command END to return to DOS
```

If you were trying to terminate VMPCDEB, use an END or QUIT command. These two commands will return you to DOS. A break-in will not.

Non-Maskable Interrupts

Although it is very rare, it is possible that the program you're testing may go into an infinite loop with interrupts disabled. When this happens, it is impossible to break into the program using Ctrl-Break or Alt-NumLock. If you add a special hardware switch (a key) to your keyboard, you can generate non-maskable interrupts. Unless a special interrupt handler is provided, these interrupts cause the system to halt in BIOS with the message:

```
PARITY CHECK 2
```

VMPCDEB has code which traps these interrupts, and they can be used to break-in on the program you're testing. If the system receives one of these interrupts while DOS or BIOS has control, it is ignored.

Error Traps

You can place error traps in your program using an INT 60 instruction. When VMPCDEB executes an INT 60 instruction, it prints a message. This messages gives you the module number (stored in the AX register) and the error number (stored in the BX register).

Break-Points

You can insert break-points in the program you're testing using the AT command (Page 11-19). With one AT command, you can set up to 24 break-points. You can also use the AT command to list the current break-points.

You can cancel break-points using the OFF command (Page 11-104). With the OFF command, you can cancel one break-point or all break-points.

Note: If you use a DELETE, LOAD, MEMSIZE, or RUN command, VMPCDEB will automatically cancel all break-points.

When you set a break-point, VMPCDEB goes to the address you've specified and replaces that command with an INT3 instruction (a break-point). These INT3 instructions only appear while you're program is running. When control returns to VMPCDEB, all INT3 instructions are removed and the instructions they replaced are re-inserted. This way, if you decide to list your program, you see the "correct" version without any break-points. When you start running your program, the break-points are re-inserted.

When the system reaches an INT3 instruction, the instruction pointer (IP) register is decremented by one, the INT3 instruction is replaced by your program's original instruction, and you are put into command input mode. This way, when you restart your program, the first instruction executed is the one that was replaced by the break-point.

When you restart the program, if the first instruction is a break-point, VMPCDEB skips that instruction and single steps the program for one instruction. After executing that instruction, VMPCDEB puts the break-point back in and starts the program normally. This lets you put break-points in loops (even single instruction loops).

Note: If you're using the DOS DEBUG facility and come across this situation, DEBUG *does not* skip the INT3 instruction.

Step Mode

You can halt your program's execution after a specified number of instructions by using "step mode." Running in step mode is not the default. If you want to use step mode, you must turn it on using a STEP command (Page 11-134). Once step mode is on, VMPCDEB executes that number of ***your program's instructions*** before putting you in command input mode. If your program happens to execute some instructions within DOS or BIOS, it does not affect the step mode counter.

You can also use the STEP command to turn step mode off or to display the current setting of step mode. If you use a DELETE, LOAD, MEMSIZE, or RUN command, the system will automatically turn step mode off.

VMPCDEB Function Entry

You can perform certain functions by using an INT 6F instruction in your program. At this time, the only function available is a request to read an additional symbol table file. Before using this function, you should set the following registers:

Register	Contents
AX	'01'X
BX	the offset you want added to the addresses in the symbol table file
ES:DI	an address which points to an 8 byte area containing the filename of the symbol table file

If these registers are set and VMPCDEB encounters an INT 6F instruction, VMPCDEB will:

- halt your program
- save the current register values
- display a message that it is reading the symbol table file
- read the file "filename.SYM"
- restart the program.

This is assuming no errors occur. If an error occurs, VMPCDEB will print the error message, the program registers, and so forth, as it would for a normal program interruption. Then VMPCDEB will put you in command input mode.

Chapter 8. Symbolic Addresses

When you load your program, if it is a COM file with a symbol table, VMPCDEB will read the symbol table into memory. Then when VMPCDEB displays addresses, they will appear in two forms: hexadecimal and symbol:offset. You can also use symbol names in addresses for any command.

Your program, after you load it, occupies a certain number of DOS memory blocks. For each of these memory blocks (not including the first one), VMPCDEB generates a label of the form:

```
MEMxxxx
```

where xxxx is the hexadecimal paragraph address of the start of the memory block. VMPCDEB uses these labels printing addresses. You can also use these labels in commands which require address fields.

If you loaded your program with a symbol table, you can use the following commands:

Command	Page	Purpose
SYMBOL	11-135	displays an address in symbolic form (symbol name plus offset)
ADDSYM	11-10	reads in a supplementary symbol table file
HIDESYM	11-84	"hides" a symbol name so VMPCDEB will not use it when printing addresses

Chapter 9. Registers

VMPCDEB has fourteen 16-bit registers:

Register	Full Name	Page
AX	Accumulator	11-20
BP	Base Pointer	11-27
BX	Base	11-29
CS	Code Segment	11-43
CX	Count	11-45
DI	Destination Index	11-50
DS	Data Segment	11-64
DX	Data	11-69
ES	Extra Segment	11-75
FLAGS		11-78
IP[1]	Instruction Pointer	11-89
SI	Source Index	11-128
SP	Stack Pointer	11-130
SS	Stack Segment	11-132

The items in column one are also commands. You can display or change the contents of an individual register using what's in the first column. For more information about these commands, go to the page listed in column three.

---

1 You can also refer to the IP register as the PC (Program Counter) register. See Page 11-108 for more details.

If you want to display the contents of all 14 registers at the same time, use one of the following commands:

Command	Page
Regs	11-120
Display REGS	11-54
List REGS	11-91
Print REGS	11-110

These commands list the contents of all 14 registers in hexadecimal form. If you loaded your program with a symbol table, the current code address CS:IP is also displayed as a symbol name plus an offset. Also, if address relocation is in force (See "Address Relocation" on page 5-10), the relocation value is subtracted from the four segment registers (CS, DS, ES, and SS).

Four of the 16-bit registers can also be divided into 8-bit registers:

16-bit Register	8-bit Registers	Page
AX	AH AL	11-11 11-13
BX	BH BL	11-23 11-25
CX	CH CL	11-36 11-39
DX	DH DL	11-48 11-61

These 8-bit registers are also commands. You can display or change 8-bits of a 16-bit register by using what's in column two. For more information about

these commands, go to the page listed in the third column.

Besides the fourteen VMPCDEB registers, you can also display the registers for a CPIO manager using one of the following commands:

Command	Page
Display MAN	11-54
List MAN	11-91
Print MAN	11-110

Note: If the specified manager is the current active manager, these commands will display the same registers as a REGS command.

Chapter 10. Displaying Memory

There are three commands you can use to display the contents of various locations of memory or a formatted listing of parts (or all) of the VM/PC System Common:

Command	Page
Display	11-54
List	11-91
Print	11-110

Address relocation affects the action or output of some forms of the PRINT command (see "Address Relocation" on page 5-10).

Displaying in ASCII and EBCDIC

When you use one of the three commands above, the system displays memory in ASCII format by default. But, you can also display memory as a translation from ASCII to EBCDIC. There are two commands which control how the system displays memory:

Command	Page
ASCII	11-18
EBCDIC	11-72

These two commands are *permanent*. When you use one of these two commands to change the way the system displays memory, the system will keep displaying memory that way until you use the other command.

Or, you can change the way the system displays memory *temporarily* (for one command). To do this, insert (A) or (E) immediately after the DISPLAY, LIST, or PRINT commands.

Chapter 11. Using Commands

This chapter describes what you need to know about VMPCDEB commands:

- Where and how you enter commands
- What's the standard command format
- What are the commands and what do they do.

Entering Commands

There are two ways to enter commands in VMPCDEB:

1. ***From your keyboard***

   You can enter any valid VMPCDEB command from your keyboard when you're in *command input mode*. Whenever you see the VMPCDEB prompt:

   ```
   VMPCDEB>
   ```

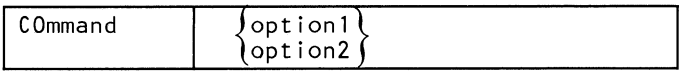
   you're in command input mode. The commands are described (in alphabetical order) starting on Page 11-8, or there's a summary of commands on Page 12-1.

2. ***From a file***

   Using the READ command (Page 11-118), you can have VMPCDEB execute commands from a file. When VMPCDEB finishes executing these commands, control reverts back the keyboard.

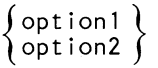
Command Format

Starting on Page 11-8 is an alphabetical listing of all the valid VMPCDEB commands you can use. Each command has a specific format which looks something like:



`COmmand` is the name of the command. Some commands have two forms: a long form and a short form. When the command is shown in uppercase and lowercase letters, you can use either the long or the short form. In the example above, you could enter `command` (the long form), or `co` (the short form). When the command is shown in all uppercase, there is no short form and you must enter the whole command.

Note: Even though the command names are shown in uppercase and lowercase, you don't have to type them that way. You can enter commands in uppercase, lowercase, or any combination of the two.



are the options that this command supports. When you see options surrounded by braces (as they are here), then you must choose one option from the list. The command won't work if you don't choose an option, nor will it work if you choose more than one option.

Optional Fields

Many commands have optional fields which are surrounded by square brackets. For example, if you see something like:

```
option1[option2]
```

Then you must type `option1`, but because `option2` is in brackets, you can omit it.

Addresses

For some commands, you'll need to supply an address. You can use any of the following for an address:

- **A hexadecimal number**

  If you enter a single hexadecimal number, VMPCDEB assumes it is an offset and adds the offset to a default segment address. Depending on which command you're using, the default segment address will be the contents of the CS register, the DS register, the segment address of the base of CPIO System Common.

- **A decimal number (preceded by #)**

  If you enter a single decimal number (preceded by a #), VMPCDEB assumes it is an offset and adds the offset to a default segment address. Depending on which command you're using, the default segment address will be the contents of the CS register, the DS register, the segment address of the base of CPIO System Common.

- **A hexadecimal number, a colon, another hexadecimal number**

  If you enter a hexadecimal number, followed by a colon, followed by another hexadecimal number, VMPCDEB assumes it is an absolute address within real memory. For example, if you were to enter:

  ```
  49C:11A
  ```

  VMPCDEB would use 11A as the offset and 49C0 as the segment address.

- **A segment register**

  If you enter a segment register, VMPCDEB uses the contents of the register as the address. The segment registers you can use are: CS, DS, ES, and SS.

- **A segment register, a colon, a hexadecimal number**

  If you enter a segment register, followed by a colon, followed by a hexadecimal number, VMPCDEB uses the contents of the register as the segment portion of the address and the hexadecimal number as the offset. The segment registers you can use are: CS, DS, ES, and SS.

- **A symbol name**

  If you have a symbol table, you can use symbol names to represent addresses. You must enclose the symbol name in "chevrons" like this:

  ```
  <symbol name>
  ```

  Note: If you enter the chevrons without a symbol name in between, VMPCDEB will use the last symbol name you entered.

- **A symbol name followed by a number**

  If you have a symbol table, you can use a symbol name and a hexadecimal number to represent an address and an offset. (You can also use a decimal number as long as it is preceded by a #.) Again, you must enclose the symbol name in "chevrons."

Note: VMPCDEB generates symbolic names for each memory block of your program, except for the first memory block. You can use these symbolic names in your commands. They look like:

```
MEMxxxx
```

where xxxx is the hexadecimal paragraph address of the base of the memory block.

If your program is a reloaded memory dump, be aware that VMPCDEB has an address relocation facility which automatically adjusts addresses within the dump. For more information, see "Address Relocation" on page 5-10.

Synonyms

When you first start VMPCDEB, synonym translation is on by default. This means you can use synonyms in commands whether you enter them from the keyboard or from a file (with a READ command). The SYNONYM command (Page 11-136) lets you:

- define a synonym
- delete a synonym
- display a synonym
- turn synonym translation on or off.

Once you define one or more synonyms, VMPCDEB scans every command looking for a synonym. When VMPCDEB encounters a synonym, it makes the substitution and scans the command again. VMPCDEB will

keep scanning a command until no synonyms are found. So, if you have more than one synonym in a command, VMPCDEB translates all of them, not just the first one.

If you turn synonym translation off, VMPCDEB will not scan any commands. But turning synonym translation off has no affect on the current synonym definitions, nor does it stop you from defining new synonyms or deleting current synonyms.

Command Reference

This section is an alphabetical listing of all the valid VMPCDEB commands. Each command has:

- a description of what it does
- a box showing its format
- a description of the parameters (if any)
- notes on how to use it (when applicable).

There is a summary on Page 12-1 which lists all the commands alphabetically and gives a brief description of each.

A

Use the A command to perform simple integer arithmetic (add, subtract, multiply, and divide). The system displays the answer in both decimal and hexadecimal form. The format of the A command is:

A	expression

Where:

`expression`

is one of the following:

a hexadecimal integer
a decimal integer (preceded by a #)
a register name
integer operator integer
expression operator integer

Usage Notes:

1. The operators you can use are:

Add	+
Subtract	-
Multiply	*
Divide	/

2. All answers are in integer format. If you divide two numbers, any decimal places (or remainder) are truncated. For example, if you divide three into five, the answer will be one. If you divide five into three, the answer will be zero.

3. The expression is evaluated left to right. Parentheses are not allowed and there is no operator preference.

4. There is no unary minus.

5. Only 16-bit unsigned arithmetic is performed.

6. You cannot divide a number by zero.

Examples:

Command	Hex Answer	Decimal Answer
A 16 + 10	0026	38
A #16 + #10	001A	26
A #16 + 10	0020	32
A 16 + #10	0020	32

Note: The H command (Page 11-82) works exactly like the A command.

ADDSYM

Use the ADDSYM command to read in a supplementary symbol table file. This file is then added to the symbol table currently held by VMPCDEB. The format of the ADDSYM command is:

ADDSYM	address filespec

Where:

`address`

is a paragraph address of the origin to be added to the symbol addresses in the symbol table file.

`filespec`

is the name of the symbol table file you want added. The format of a filespec is:

```
drive id:filename.extension
```

If the symbol table file is on the default disk drive, you can omit the `drive id:` field. The default extension field is SYM. You can omit this field unless your file extension is something other than SYM.

Usage Notes:

1. The records in this file may have trailing spaces, which are ignored.

2. If the program you're testing is a reloaded dump program, with address relocation, the address you use in the ADDSYM command should be the **real** address of the relocated program, not the original address of the program from before you created the dump. This provides symbolic debugging for the dump file.

AH

Use the AH command to list or alter the contents of the AH register. The AH register is the high-order (bit 8 through bit 15) byte of the Accumulator (AX) register. The format of the AH command is:

AH	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	AH 5A	'5A'X
String	AH ''Z''	'5A'X
Decimal	AH #90	'5A'X

AL

Use the AL command to list or alter the contents of the AL register. The AL register is the low-order (bit 0 through bit 7) byte of the Accumulator (AX) register. The format of the AL command is:

AL	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

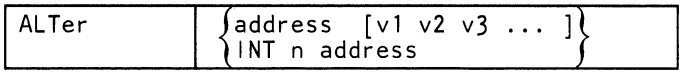
4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	AL 5A	'5A'X
String	AL "Z"	'5A'X
Decimal	AL #90	'5A'X

ALTER

Use the ALTER command to change the contents of memory locations or an interrupt vector location. The format of the ALTER command is:



Where:

`address`

is the location (in hex) where you want to change memory.

`v1 v2 v3 ...`

are the new values you want in memory. These values can be either numbers (one or two digit) or character strings (enclosed in quotes). You cannot use negative decimal numbers.

`n`

is the number of the interrupt vector location you want changed. When you enter this number, VMPCDEB assumes it is in hexadecimal form. If you want to enter a decimal number, you must precede the number by a pound sign (#). This number must be between 0 and 127 (decimal), inclusive.

`address`

is the 32-bit address you want placed in interrupt vector `n`.

Usage Notes for `ALTer address [v1 v2 v3 ... ]`

1. You cannot use this command if bit 7 at location '250'X of VMPCDEB is zero. If you enter an ALTER command and bit 7 is zero, the command

terminates without changing any memory locations. By default, this bit is zero. If you need to use the ALTER command, use the BYTE250 command (described on Page 11-31) to change the bit from zero to one.

2. If you supply values for "v1 v2 v3 ...," the system takes these values and places them in successive bytes of memory, starting at the address you've supplied.

   Note: If you try to place more than 8 bits of data into a byte, VMPCDEB stops execution *at that point* and displays the following message:

   ```
   Value has more than 8 bits
   ```

   For example, if you were trying to alter 4 bytes of memory and the third value you entered was too large, VMPCDEB would alter the first two bytes and then stop. Although the fourth byte may have been valid, once VMPCDEB spotted the error in the third byte, the command terminated.

3. If you enter the ALTER command with just an address, the system displays the contents of successive bytes until you tell it to quit. The system displays the contents of a byte, then it waits for input from you. Once you supply the input, it will display the next byte and wait for input (unless you tell it to quit). The input can be one of the following:

   - a one or two digit number to be placed in the byte
   - an 8-bit register containing the value to be placed in the byte
   - a character string (enclosed in quotes). The system places the characters in successive bytes. No nulls strings are allowed.
   - **U** (leave the contents "Unchanged")
   - **Q** ("Quit" this command and return to command input mode).

4. If you only supply an offset (not a segment address), the system uses the contents of the DS register as the segment address.

5. If address relocation is in force, it affects the address of the location(s) you're changing.

6. The system checks to see that the address is not below the start of the program segment prefix. If it is below this point, you'll get a warning message.

Example:

The command:

```
ALTer DS:12A 47 5E "The cat" 0
```

will change 10 bytes of memory starting at the contents of the DS register (plus an offset of '012A'X). These bytes will contain: 47, 5E, 54, 68, 65, 20, 63, 61, 74, and 00, respectively.

Usage Note for `ALTer INT n address`

You can only temporarily change the contents of an interrupt vector location. Once you stop testing the program, the system restores the interrupt vector location with the "normal" DOS value.

ASCII

Use the ASCII command to set the print mode flag (bit 5 of location '250'X of VMPCDEB) to one. When this bit equals one and you are printing areas of memory as text, the output will be printed in ASCII format. The format of the ASCII command is:

ASCii	

Usage Notes:

1. When you first start testing a program, the default is to print as ASCII (i.e., the bit 5 in location '250'X of VMPCDEB equals one).

2. Use the EBCDIC command (Page 11-72) to change the bit to zero.

3. If the bit equals one, you must use the EBCDIC command to change it to zero. The bit will remain zero until you issue another ASCII command.

4. You can turn this bit on or off temporarily for one command (DISPLAY, LIST, or PRINT) by using the letters A or E within the command. See the DISPLAY (Page 11-54), the LIST (Page 11-91), and the PRINT (Page 11-110) commands for more details.

AT

Use the AT command to set or display break-points. The format of the AT command is:

AT	[address]

Where:

`address`

is where you want the break-point placed. If you specify only an offset (instead of a segment address), the contents of the CS register will be used as the segment address.

Usage Notes:

1. You can set up to 24 break-points at the same time.

2. If you enter the AT command without an address, the system lists the current break-point addresses.

3. When you set a break-point, VMPCDEB checks to see if the address you specified is greater than or equal to the address of the start of the program. In other words, VMPCDEB checks to see if you're setting a break-point in DOS or VMPCDEB code. If bit 7 at location '250'X of VMPCDEB equals zero (the default), you see a warning message and that break-point won't be set. If this bit equals one, you'll see the warning message, but the break-point will be set. Be very careful when you're setting break-points in DOS or VMPCDEB! They are liable to cause problems, such as crashing the system, and should be used with extreme caution.

Note: To cancel a break-point (or break-points), use the OFF command (Page 11-104).

AX

Use the AX command to list or alter the contents of the Accumulator (AX) register. The AX register is a 16-bit register that is divided into two 8-bit registers: the AH and AL registers. The format of the AX command is:

AX	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

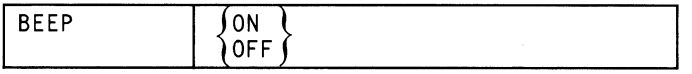
4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	AX 4849	'4849'X
String	AX "HI"	'4849'X
Decimal	AX #18505	'4849'X

BEEP

Use the BEEP command to turn your machine's speaker (or "beeper") on or off. The format of the BEEP command is:



Where:

ON

turns the speaker on.

OFF

turns the speaker off.

The sound frequency (pitch) is not set or altered by this command.

BH

Use the BH command to list or alter the contents of the BH register. The BH register is the high-order (bit 8 through bit 15) byte of the Base (BX) register. The format of the BH command is:

BH	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`BH 5A`	'5A'X
String	`BH "Z"`	'5A'X
Decimal	`BH #90`	'5A'X

BL

Use the BL command to list or alter the contents of the BL register. The BL register is the low-order (bit 0 through bit 7) byte of the Base (BX) register. The format of the BL command is:

BL	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`BL 5A`	'5A'X
String	`BL "Z"`	'5A'X
Decimal	`BL #90`	'5A'X

BP

Use the BP command to list or alter the contents of the Base Pointer (BP) register. The format of the BP command is:

BP	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`BP 4849`	'4849'X
String	`BP "HI"`	'4849'X
Decimal	`BP #18505`	'4849'X

BX

Use the BX command to list or alter the contents of the Base (BX) register. The BX register is a 16-bit register that is divided into two 8-bit registers: the BH and BL registers. The format of the BX command is:

BX	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`BX 4849`	'4849'X
String	`BX "HI"`	'4849'X
Decimal	`BX #18505`	'4849'X

BYTE250

Use the BYTE250 command to change the value of the byte at location '250'X of VMPCDEB. You can set certain options and defaults within VMPCDEB by changing the value of this byte. The format of the BYTE250 command is:

BYTE250	[n]

Where:

n

is a hexadecimal number from 0 to FF (inclusive) or a decimal number from 0 to 255 (inclusive).

Usage Notes:

1. ***Use this command with caution!***

2. If you enter this command without supplying a value, the system displays the contents of the byte and prompts you for a new value. At this point, if you just press the ENTER key, the contents of the byte remain unchanged.

3. This byte is bit significant (see the table which follows) with bit 0 being the left-most bit. When you first start testing a program, the value of this byte is 10111010 (binary) or BA (hexadecimal).

Bit	Meaning
0	**If this bit equals one:** the device emulation adapter interrupt addresses point to the BIOS interrupt address when VMPCDEB is loaded. **If this bit equals zero:** device emulation adapter resets are never performed and no attempt is made to see if the timer interrupt handler within VMPCDEB should be used in place of the BIOS timer routine. This is the same as saying there is no device emulation adapter.
1	If this bit equals one, memory allocation/deallocation request debug prints occur. This is included as a facility for debugging VMPCDEB.
2	not used
3	not used
4	(page mode flag) If this bit equals one, page mode operation is on by default.
5	(ASCII/EBCDIC flag) If this bit equals one, character codes are converted from EBCDIC to ASCII before printing memory areas (as characters).
6	If this bit equals one, device emulation adapter resets are performed when a program is loaded or when VMPCDEB terminates. These resets are performed regardless of the state of the device emulation adapter/controller as held in VM/PC CPIO system common. *This facility is disabled if bit 0 equals zero.*
7	If this bit equals one, you can alter memory (using the ALTER command) and set break-points (using the AT command) below the start of the program you're testing. In both cases, a warning message will be displayed. Also, if you use a MEM command, the system lists the memory blocks occupied by VMPCDEB as well as those occupied by the program under test.

BYTE251

Use the BYTE251 command to change the value of the byte at location '251'X of VMPCDEB. You can set certain options and defaults within VMPCDEB by changing the value of this byte. The format of the BYTE251 command is:

BYTE251	[n]

Where:

n

is a hexadecimal number from 0 to FF (inclusive) or a decimal number from 0 to 255 (inclusive).

Usage Notes:

1. ***Use this command with caution!***

2. If you enter this command without supplying a value, the system displays the contents of the byte and prompts you for a new value. At this point, if you just press the ENTER key, the contents of the byte remain unchanged.

3. This byte is bit significant (see the table which follows) with bit 0 being the left-most bit. When you first start testing a program, the value of this byte is 11000000 (binary) or C0 (hexadecimal).

Bit	Meaning
0	If this bit equals one, VMPCDEB (when it's first loaded) tries to include the dis-assembly code, by attaching the file IWSUNPIK.DEB to the end of VMPCDEB's code area.
1	If this bit equals one, synonym translation is turned on by default.
2	If this bit equals one, immediately after a program load, the memory block size for the program under test is set to the size calculated from the program file.
3	not used
4	not used
5	not used
6	not used
7	not used

CD

Use the CD command to display or change the current directory path. The format of the CD command is:

CD	[d:][path]

Where:

d:
is the drive id that the directory resides on. If the directory resides on the default drive, you can omit this field.

path
is the new directory path.

Usage Notes:

1. If you enter this command without any parameters, the system displays the current directory path of the default drive.

2. If you enter this command and you specify a drive id, but not a path, the system displays the current directory path of that drive.

3. If you enter this command and you specify a new path, but not a drive id, the system changes the directory on the default drive to the path you've indicated.

4. If you enter this command and you specify a drive id and a new path, the system changes the directory on the drive you've indicated to the path you've indicated.

Note: The CHDIR command (Page 11-38) works exactly like the CD command. If you need to know more about the VMPCDEB CD command, see the DOS CHDIR command.

CH

Use the CH command to list or alter the contents of the CH register. The CH register is the high-order (bit 8 through bit 15) byte of the Count (CX) register. The format of the CH command is:

CH	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`CH 5A`	'5A'X
String	`CH "Z"`	'5A'X
Decimal	`CH #90`	'5A'X

CHDIR

Use the CHDIR command to display or change the current directory path. The format of the CHDIR command is:

CHDIR	[d:][path]

Where:

`d:`
is the drive id that the directory resides on. If the directory resides on the default drive, you can omit this field.

`path`
is the new directory path.

Usage Notes:

1. If you enter this command without any parameters, the system displays the current directory path of the default drive.

2. If you enter this command and you specify a drive id, but not a path, the system displays the current directory path of that drive.

3. If you enter this command and you specify a new path, but not a drive id, the system changes the directory on the default drive to the path you've indicated.

4. If you enter this command and you specify a drive id and a new path, the system changes the directory on the drive you've indicated to the path you've indicated.

Note: The CD command (Page 11-35) works exactly like the CHDIR command. If you need to know more about the VMPCDEB CHDIR command, see the DOS CHDIR command.

CL

Use the CL command to list or alter the contents of the CL register. The CL register is the low-order (bit 0 through bit 7) byte of the Count (CX) register. The format of the CL command is:

CL	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`CL 5A`	'5A'X
String	`CL "Z"`	'5A'X
Decimal	`CL #90`	'5A'X

CLEAR

Use the CLEAR command to clear the current VMPCDEB screen. The format of the CLEAR command is:

CLEAR	

Note: The CLS command (Page 11-42) works exactly like the CLEAR command. If you need to know more about the VMPCDEB CLEAR command, see the DOS CLS command.

CLS

Use the CLS command to clear the current VMPCDEB screen. The format of the CLS command is:

CLS	

Note: The CLEAR command (Page 11-41) works exactly like the CLS command. If you need to know more about the VMPCDEB CLS command, see the DOS CLS command.

CS

Use the CS command to list or alter the contents of the Code Segment (CS) register. The format of the CS command is:

CS	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`CS 4849`	'4849'X
String	`CS "HI"`	'4849'X
Decimal	`CS #18505`	'4849'X

CX

Use the CX command to list or alter the contents of the Count (CX) register. The CX register is a 16-bit register that is divided into two 8-bit registers: the CH and CL registers. The format of the CX command is:

CX	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`CX 4849`	'4849'X
String	`CX "HI"`	'4849'X
Decimal	`CX #18505`	'4849'X

DELETE

Use the DELETE command to delete the program you're testing. The format of the DELETE command is:

DELete	

Usage Notes:

1. This command also:

   - restores the system memory size to its original value (or to the value set by a MEMSIZE command, if the two values are different)
   - sets up the program registers as if a COM file had been loaded
   - clears the symbol table
   - clears all break-points
   - clears any interrupt traps you set with the TRAP command
   - resets the device emulation adapter (if bit 0 and bit 6 at location '250'X equal one)
   - issues a "reset disk" request to BIOS
   - resets all asynch cards
   - resets the PC timer interrupt frequency
   - resets the IMR to its original value, except that the device emulation adapter interrupts (interrupt levels 2 and 3) are disabled and diskette interrupts are enabled.

DH

Use the DH command to list or alter the contents of the DH register. The DH register is the high-order (bit 8 through bit 15) byte of the Data (DX) register. The format of the DH command is:

DH	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`DH 5A`	'5A'X
String	`DH "Z"`	'5A'X
Decimal	`DH #90`	'5A'X

DI

Use the DI command to list or alter the contents of the Destination Index (DI) register. The format of the DI command is:

DI	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`DI 4849`	'4849'X
String	`DI "HI"`	'4849'X
Decimal	`DI #18505`	'4849'X

DIR

Use the DIR command to list the contents of a directory or of specified files. The format of the DIR command is:

DIR	[d:][path][filename.ext]

Where:

`d:`

is the letter of the fixed disk where the directory resides. If the directory resides on the default drive, you can omit this field.

`path`

is the name of a specific directory you want listed.

`filename.ext`

is the name and extension of a specific file you want listed.

Usage Note:

This command lists the name, extension, time, date, and size (in decimal form) for each file. It also displays the volume identification and the amount of free space left. The VMPCDEB DIR command is the same as the DOS DIR command except for the following differences:

1. If you specify a filename, you must also specify an extension. To see a list of all files with the same filename, enter:

   `filename.*`

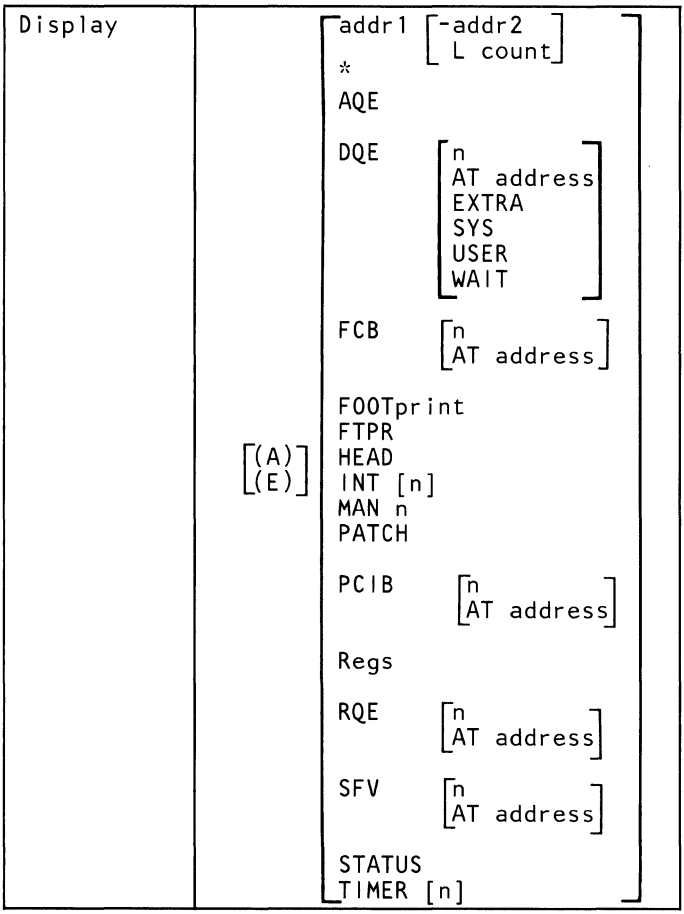
2. For some files, if certain bits are set in the attribute byte, one (or more) letters are displayed. The letters and their meanings are as follows:

A	the archive bit is set.
D	the file is a directory.
H	the file is "hidden."
R	the file is read only.
S	the file is a system file.
V	the directory entry is a disk volume label.

If you need to know more about the VMPCDEB DIR command, see the DOS DIR command.

DISPLAY

Use the DISPLAY command to display the contents of various locations of memory, or to create a formatted listing of parts (or all) of the VM/PC System Common. The format of the DISPLAY command is:



Where:

```
addr1 [-addr2  ]
      [L count]
```

is the address of the area you want displayed.

If you use `addr1` with nothing after it, VMPCDEB will display 16 bytes starting at that address.

If you use `addr1-addr2`, these are the beginning and ending addresses of the area you want displayed. These two addresses must be separated by a dash (-).

If you use `addr1 L count`, VMPCDEB displays the area starting at `addr1` for the number of bytes you specified in `count`. (The `L` is short for "Length.")

Note: If you don't specify a segment address (only an offset), the system will use the contents of the DS register as the segment address.

`*`

displays the same locations as the last DISPLAY command.

`AQE`

displays the active AQE and its associated RQE chain pointer.

`DQE n`

displays DQE "n" and its associated RQE chain pointer. If you omit "n," the system displays all DQEs and their associated RQE chain pointers.

`DQE AT address`

displays the DQE at the address you specify and its associated RQE chain pointer.

`DQE EXTRA`

displays all EXTRA DQEs and their associated RQE chain pointers.

`DQE SYS`

displays all SYS DQEs and their associated RQE chain pointers.

`DQE USER`

displays all USER DQEs and their associated RQE chain pointers.

`DQE WAIT`

displays all WAIT DQEs and their associated RQE chain pointers.

`FCB n`

displays a formatted listing of FCB "n" in hexadecimal and character format. If you omit "n," the system displays all FCBs.

`FCB AT address`

displays the FCB at the address you specify.

`FOOTprint`

displays the dispatcher footprints table. The system starts with the entry pointed to by the table index within system common and wraps around from the bottom of the table back up to the top. This is the same as the DISPLAY FTPR command.

`FTPR`

displays the dispatcher footprints table. The system starts with the entry pointed to by the table index within system common and wraps around from the bottom of the table back up to the top. This is the same as the DISPLAY FOOTPRINT command.

HEAD

displays a formatted listing of the CPIO System Common Header.

INT n

displays interrupt vector entry n. If you omit n, the system displays the first 128 interrupt addresses. These addresses are those used by the program you're testing, which may or may not be the same as those used when VMPCDEB is running.

MAN n

displays the SFV entry, the DQE, and the current registers for the CPIO manager whose SFV index is n. If the manager you specify is the current active manager, the system displays the current registers for the program you're testing (as in a REGS or DISPLAY REGS command). Otherwise, the system displays the register values that were saved on the specified manager's stack.

PATCH

displays the system common patch area in hexadecimal and character format.

PCIB n

displays a formatted listing of PCIB n. If you omit n, the system displays all PCIBs.

PCIB AT address

displays the PCIB at the address you specify.

displays the contents of the following registers (in hex):

AX	(Accumulator)
BP	(Base Pointer)
BX	(Base)
CS	(Code Segment)
CX	(Count)
DI	(Destination Index)
DS	(Data Segment)
DX	(Data)
ES	(Extra Segment)
FLAGS	
IP	(Instruction Pointer)
SI	(Source Index)
SP	(Stack Pointer)
SS	(Stack Segment)

The four data registers (AX, BX, CX, and DX) are 16-bit registers that can also be divided into 8-bit registers (AH, AL, BH, BL, CH, CL, DH, and DL, respectively).

Also, after the hex value of the FLAGS register, the system lists the two letter mnemonic associated with each flag. For example, if each flag was clear, you'd see something like:

```
FLAGS 0000 ( NV UP DI NT PL NZ NA PO NC )
```

If the program was loaded with a symbol table, the current code address CS:IP is also displayed as a symbol name plus an offset.

If address relocation is active, the relocation value is subtracted from the four segment registers (CS, DS, ES, and SS).

Note: You can also use the REGS command (Page 11-120) to list the contents of all registers.

Entering DISPLAY REGS gives you the same information as entering REGS.

`RQE n`

displays a formatted listing of RQE `n`. If you omit `n`, the system displays all RQEs.

`RQE AT address`

displays the RQE at the address you specify.

`SFV n`

displays a formatted listing of SFV `n`. If you omit `n`, the system displays all SFVs.

`SFV AT address`

displays the SFV at the address you specify.

`STATUS`

displays the system common session status and device status areas in hexadecimal and character format.

`TIMER n`

displays a formatted listing of timer entry `n`. If you omit `n`, the system displays all timer entries.

Usage Notes:

1. If you enter the DISPLAY command without any parameters, the system displays the same number of bytes as the last DISPLAY command starting where it left off. If you haven't used a DISPLAY command yet, the system displays 16 bytes starting at CS:IP.

2. When you're displaying areas of memory as text, the default is to display these areas as ASCII. You can choose to display as a translation from

EBCDIC to ASCII. The ASCII (Page 11-18) and EBCDIC (Page 11-72) commands change the way the system displays memory.

You can change the way the system displays memory *temporarily* by using a special parameter on the DISPLAY command. To do this, you insert (A) or (E) immediately after the DISPLAY command. For example, if you enter:

```
asc
d cs:100
d (e) cs:100
d cs:100
```

The first command tells the system to display in ASCII. The second command tells the system to display 16 bytes starting at location CS:100. The third command tells the system to display the same 16 bytes, but now translate them to EBCDIC. The last command tells the system to display the same 16 bytes with ASCII translation.

3. The LIST (Page 11-91) and PRINT (Page 11-110) commands work exactly like the DISPLAY command.

DL

Use the DL command to list or alter the contents of the DL register. The DL register is the low-order (bit 0 through bit 7) byte of the Data (DX) register. The format of the DL command is:

DL	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`DL 5A`	'5A'X
String	`DL "Z"`	'5A'X
Decimal	`DL #90`	'5A'X

DRIVE

Use the DRIVE command to display or set the default disk drive. The format of the DRIVE command is:

DRIVE	[d]

Where:

d

is the letter of the new default disk drive.

Note: You only need to enter a single letter. In DOS, you would need to enter a letter followed by a colon (:).

Usage Notes:

1. If you enter this command without supplying a letter, the system will display the letter of the current default disk drive.

2. If you enter this command with a letter (A, C, D, and so forth), the system will change the current default disk drive to that letter.

3. If you enter an invalid letter (a letter for a drive id you don't have), VMPCDEB will not change the default disk drive.

DS

Use the DS command to list or alter the contents of the Data Segment (DS) register. The format of the DS command is:

DS	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`DS 4849`	'4849'X
String	`DS "HI"`	'4849'X
Decimal	`DS #18505`	'4849'X

DUMP

Use the DUMP command to create a memory dump of the program you're testing. The format of the DUMP command is:

DUMP	filespec

Where:

`filespec`

is the name of the dump file you want created. The filespec must be in the following format:

```
drive id:filename.extension
```

If the file is on the default drive, you can omit the `drive id:` field. If you omit the `extension` field, the system will use an extension of DUM. If you are planning to reload this file, the extension must be DUM.

Usage Notes:

1. If the file already exists, this command will not work. The system displays an error message and will not create a new dump.

2. If the file does not exist, the system creates the dump and writes it to the new file. This new file will be a dump of all memory from the end of VMPCDEB to the end of memory. The end of memory is calculated using the current value of location 40:13 (the location used by BIOS to hold the memory size of the system).

3. The first 16 bytes of the dump contain the memory control block from immediately below the program segment prefix of your program.

4. When you use this command, the system creates a file which consists of a 768-byte header followed by the memory dump. This header gives the following information:

Byte	Meaning
0-21	the character string "VMPC Debug version n."
22	a bit field, with bit 0 as the left-most bit. If the bit equals one, it has the following meaning: bit 0 it's a VMPCDEB dump. bit 1 time and date fields (bytes 24-30) are meaningful. bit 2 register values (bytes 31-58) are meaningful. bit 3 interrupt vectors (bytes 256-767) are meaningful. bit 4 entry type (byte 23) is meaningful. bit 5 it's a DOS dump and it includes the DOS memory control blocks.
23	a number indicating the type of the last entry to VMPCDEB from the program. The numbers are: 0 immediately after program load 1 program halted (INT 20) 2 operator break-in 3 error report (INT 60) 4 break-point trap 5 instruction count expired (in step mode) 6 NMI break-in 7 function call (INT 6F) 255 unknown (VMPCDEB didn't create the dump)
24	hour

Byte	Meaning
25	minutes
26	seconds
27	month
28	day of month
29-30	year
31-32	paragraph address of the start of the memory area dumped. When you reload the dump, VMPCDEB uses this address to calculate the address relocation value (it is unlikely that you'll reload to the same address). If it's a DOS dump, this is the address of the memory control block immediately before the program.
33-34	size of dumped memory area, in paragraphs.
35-62	program register contents, in the following order: AX, BX, CX, DX, SP, BP, SI, DI, IP, CS, DS, SS, ES, FLAGS.
63-255	unused
256-767	contents of interrupt locations 0-127 when the program is running.

DX

Use the DX command to list or alter the contents of the Data (DX) register. The DX register is a 16-bit register that is divided into two 8-bit registers: the DH and DL registers. The format of the DX command is:

DX	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	DX 4849	'4849'X
String	DX "HI"	'4849'X
Decimal	DX #18505	'4849'X

D370

Use the D370 command to create a dump of the 370 processor. The format of the D370 command is:

D370	filespec

Where:

`filespec`

is the name of the dump file you want created. The filespec must be in the following format:

```
drive id:filename.370
```

If the file is on the default drive, you can omit the `drive id:` field. If you omit the file extension (370), the system will use 370 by default. Whether or not you supply the file extension, it *must* be 370.

Usage Notes:

1. If the file already exists, this command will not work. The system will display an error message and will not create the dump.

2. If the file does not exist, the system creates the dump and writes it to the new file. The system takes 320K bytes of memory (starting at address 256K) and writes it to the new file as a simple copy of memory. Then, the system creates another new file: "filename.371," and writes 192K bytes of memory (starting at address 576K) to that file.

3. Use the L370 command (Page 11-99) to load these memory dumps for future analysis.

EBCDIC

Use the EBCDIC command to set the print mode flag (bit 5 of location '250'X of VMPCDEB) to zero. When this bit equals zero and you are printing areas of memory as text, the output will be printed as a translation from EBCDIC to ASCII. The format of the EBCDIC command is:

EBCdic	

Usage Notes:

1. When you first start testing a program, the default is to print as ASCII (i.e., the bit 5 in location '250'X of VMPCDEB equals one).

2. Use the ASCII command (Page 11-18) to change the bit to one.

3. If the bit equals zero, you must use the ASCII command to change it to one. The bit will remain one until you issue another EBCDIC command.

4. You can turn this bit on or off temporarily for one command (DISPLAY, LIST, or PRINT) by using the letters A or E within the command. See the DISPLAY (Page 11-54), the LIST (Page 11-91), and the PRINT (Page 11-110) commands for more details.

END

se the END command to terminate VMPCDEB. The format of the END command is:

END	

Note: The QUIT command (Page 11-117) works exactly like the END command.

ERASE

Use the ERASE command to erase a file from disk. The format of the ERASE command is:

ERASE	filespec

Where:

`filespec`

is the name of the file you want deleted. The format of a filespec is:

`drive id:filename.extension`

If the file you want deleted is on the default drive, you can omit the `drive id:` field.

If you need to know more about the VMPCDEB ERASE command, see the DOS ERASE command. Both commands work the same way, *except* VMPCDEB will not let you use an asterisk (*) in the `extension` field. When you erase a file, you must give the full name.

ES

Use the ES command to list or alter the contents of the Extra Segment (ES) register. The format of the ES command is:

ES	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

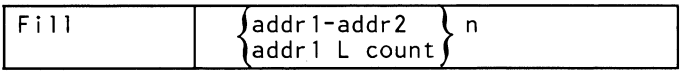
4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`ES 4849`	'4849'X
String	`ES "HI"`	'4849'X
Decimal	`ES #18505`	'4849'X

FILL

Use the FILL command to replace the contents of all bytes in the address range with a specified value. The format of the FILL command is:



Where:

`addr1-addr2`
`addr1 L count`

is the address of the area you want displayed.

If you use `addr1-addr2`, these are the beginning and ending addresses of the area you want displayed. These two addresses must be separated by a dash (-).

If you use `addr1 L count`, VMPCDEB displays the area starting at `addr1` for the number of bytes you specified in `count`. (The `L` is short for "Length.")

Note: If you don't specify a segment address (only an offset), the system will use the contents of the DS register as the segment address.

`n`

is an 8-bit hexadecimal number you want put in the address range.

Usage Note:

This command will only accept an 8-bit hex number. No provision is made for character strings or register names.

FLAGS

Use the FLAGS command to list or alter the contents of the FLAGS register. The format of the FLAGS command is:

FLAGS	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and the state (set or clear) of each flag. (These states are shown in Note 5 below.) Then the system waits for a new value (or values). If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value (or values), the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

5. When you display the contents of this register, you'll see the states of each flag. The flags and the states are listed in a table which follows. Bit 15 is the left-most (or most significant) bit.

bit #	flag name	set	clear
15	not used	-	-
14	not used	-	-
13	not used	-	-
12	not used	-	-
11	Overflow Flag (yes/no)	OV	NV
10	Direction Flag (decrement/increment)	DN	UP
9	Interrupt Enable Flag (enable/disable)	EI	DI
8	Trap Flag (yes/no)	TR	NT
7	Sign Flag (negative/positive)	NG	PL
6	Zero Flag (yes/no)	ZR	NZ
5	not used	-	-
4	Auxiliary Carry Flag (yes/no)	AC	NA
3	not used	-	-
2	Parity Flag (even/odd)	PE	PO
1	not used	-	-

bit #	flag name	set	clear
0	Carry Flag (yes/no)	CY	NC

6. You can set or clear individual flags by using the two-letter acronym from the table above, instead of specifying a hex value or a character string. For example, the command:

   ```
   FLAGS OV CY NA
   ```

   sets the overflow (bit 11) and the carry (bit 0) flags on, clears the auxiliary carry flag (bit 4), and leaves the rest of the flags unchanged.

GO

Use the GO command to start testing a program that has previously been loaded. The format of the GO command is:

Go	

Usage Note:

When you enter this command, the program starts with the instruction specified by the current setting of the CS and IP registers. The program will run until it is "halted" by one of the following:

- Program termination
- Break-ins
- Non-maskable interrupts
- Error traps
- Break-points
- Step mode
- VMPCDEB function entry.

(For more detail about these "events," see Chapter 7, "Halting a Program" on page 7-1.) When one of these events occurs, VMPCDEB returns to command input mode.

H

Use the H command to perform simple integer arithmetic (add, subtract, multiply, and divide). The system displays the answer in both decimal and hexadecimal form. The format of the H command is:

H	expression

Where:

`expression`

is one of the following:

a hexadecimal integer
a decimal integer (preceded by a #)
a register name
integer operator integer
expression operator integer

Usage Notes:

1. The operators you can use are:

Add	+
Subtract	-
Multiply	*
Divide	/

2. All answers are in integer format. If you divide two numbers, any decimal places (or remainder) are truncated. For example, if you divide three into five, the answer will be one. If you divide five into three, the answer will be zero.

3. The expression is evaluated left to right. Parentheses are not allowed and there is no operator preference.

4. There is no unary minus.

5. Only 16-bit unsigned arithmetic is performed.

6. You cannot divide a number by zero.

Examples:

Command	Hex Answer	Decimal Answer
H 16 + 10	0026	38
H #16 + #10	01A	26
H #16 + 10	0020	32
H 16 + #10	0020	32

Note: The A command (Page 11-8) works exactly like the H command.

HIDESYM

Use the HIDESYM command to "hide" a symbol name so that VMPCDEB will not use it when printing addresses. The format of the HIDESYM command is:

HIDESYM	symbol name

Where:

`symbol name`

is the name of the symbol you want "hidden."

Usage Notes:

1. Once you enter this command, the symbol name is "marked" and VMPCDEB will never use it when printing addresses.

2. You will still be able to use the symbol when you input addresses.

IMR

Use the IMR command to list or change the contents of the Interrupt Mask Register (IMR). The format of the IMR command is:

IMR	[n]

Where:

n

is the new value you want in the IMR. This value must be one of the following:

A hexadecimal number
A decimal number (preceded by a #)
A register name.

Usage Notes:

1. The IMR is the register in the 8259 interrupt controller which enables or disables individual hardware interrupt levels. While you're testing a program, VMPCDEB saves a copy of the value of the IMR. When the program halts or is interrupted, VMPCDEB uses this copy to restore the register before you restart the program.

2. If you enter this command without supplying a new value for the register, the system displays the current value and prompts you for a new value. This new value will be used the next time the program is restarted. If you just press the ENTER key, the value will not change.

3. If you enter this command with a new value for the register, the system will change the value without displaying the previous value.

4. If you're using a decimal number (preceded by a #), it must be between 0 and 255, inclusive.

INB

Use the INB command to input a byte from a port address and display the byte that was input. The format of the INB command is:

INB	n

Where:

n

is the port address of the byte.

INT

Use the INT command to simulate an interrupt. The format of the INT command is:

Where:

n

is the interrupt number. It must be a hexadecimal number between 1 and F, inclusive.

address

is the address where you want the interrupt to happen.

Usage Notes:

Once you've entered this command, the following occurs:

1. The system simulates an interrupt at the interrupt number or the address you've specified.

2. The CS, FLAGS, and IP registers are pushed onto the program's stack.

3. The CS and IP registers are set to the address associated with the interrupt number or the address you've specified.

4. The Interrupt Enable Flag (bit 9 in the FLAGS register) and the Trap Flag (bit 8 in the FLAGS register) are cleared, as if a normal interrupt had occurred.

When you issue a GO command, the program runs as if an interrupt had occurred.

INW

Use the INW command to input a word from a port address and display the word that was input. The format of the INW command is:

INW	n

Where:

n

is the port address of the word.

IP

Use the IP command to list or alter the contents of the Instruction Pointer (IP) register. This register is also referred to as the Program Counter (PC) register. The format of the IP command is:

IP	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

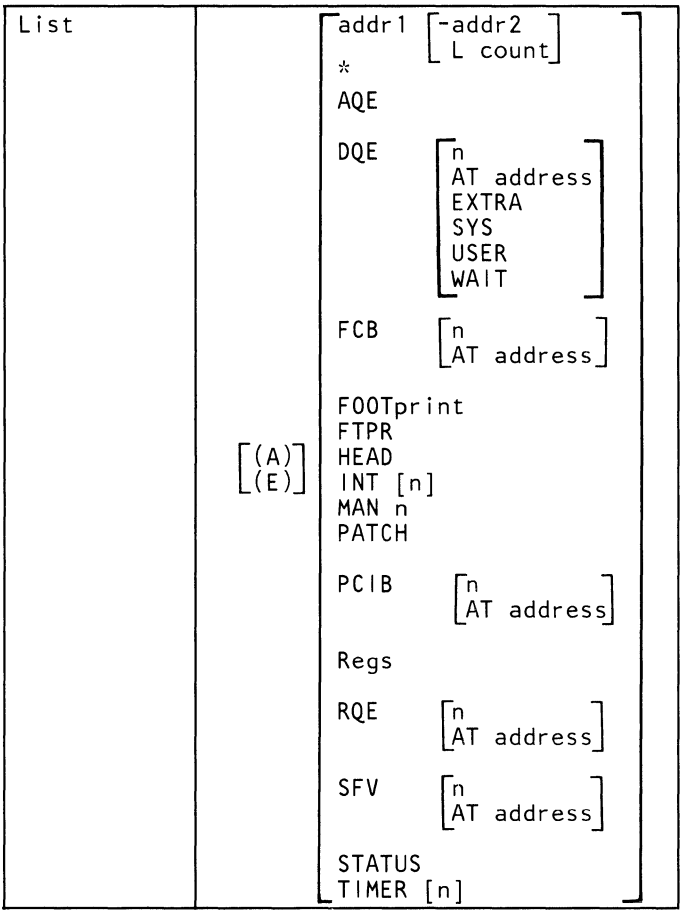
4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	IP 4849	'4849'X
String	IP "HI"	'4849'X
Decimal	IP #18505	'4849'X

LIST

Use the LIST command to list the contents of various locations of memory, or to create a formatted listing of parts (or all) of the VM/PC System Common. The format of the LIST command is:



Where:

```
addr1 [-addr2  ]
      [L count ]
```

is the address of the area you want displayed.

If you use `addr1` with nothing after it, VMPCDEB will display 16 bytes starting at that address.

If you use `addr1-addr2`, these are the beginning and ending addresses of the area you want displayed. These two addresses must be separated by a dash (-).

If you use `addr1 L count`, VMPCDEB displays the area starting at `addr1` for the number of bytes you specified in `count`. (The `L` is short for "Length.")

Note: If you don't specify a segment address (only an offset), the system will use the contents of the DS register as the segment address.

`*`

lists the same locations as the last LIST command.

`AQE`

lists the active AQE and its associated RQE chain pointer.

`DQE n`

lists DQE `n` and its associated RQE chain pointer. If you omit `n`, the system lists all DQEs and their associated RQE chain pointers.

`DQE AT address`

lists the DQE at the address you specify and its associated RQE chain pointer.

`DQE EXTRA`

lists all EXTRA DQEs and their associated RQE chain pointers.

`DQE SYS`

lists all SYS DQEs and their associated RQE chain pointers.

`DQE USER`

lists all USER DQEs and their associated RQE chain pointers.

`DQE WAIT`

lists all WAIT DQEs and their associated RQE chain pointers.

`FCB n`

lists a formatted listing of FCB `n` in hexadecimal and character format. If you omit `n`, the system lists all FCBs.

`FCB AT address`

lists the FCB at the address you specify.

`FOOTprint`

lists the dispatcher footprints table. The system starts with the entry pointed to by the table index within system common and wraps around from the bottom of the table back up to the top. This is the same as the LIST FTPR command.

`FTPR`

lists the dispatcher footprints table. The system starts with the entry pointed to by the table index within system common and wraps around from the bottom of the table back up to the top. This is the same as the LIST FOOTPRINT command.

HEAD

lists a formatted listing of the CPIO System Common Header.

INT n

lists interrupt vector entry n. If you omit n, the system lists the first 128 interrupt addresses. These addresses are those used by the program you're testing, which may or may not be the same as those used when VMPCDEB is running.

MAN n

lists the SFV entry, the DQE, and the current registers for the CPIO manager whose SFV index is n. If the manager you specify is the current active manager, the system lists the current registers for the program you're testing (as in a REGS or LIST REGS command). Otherwise, the system lists the register values that were saved on the specified manager's stack.

PATCH

lists the system common patch area in hexadecimal and character format.

PCIB n

lists a formatted listing of PCIB n. If you omit n, the system lists all PCIBs.

PCIB AT address

lists the PCIB at the address you specify.

lists the contents of the following registers (in hex):

AX	(Accumulator)
BP	(Base Pointer)
BX	(Base)
CS	(Code Segment)
CX	(Count)
DI	(Destination Index)
DS	(Data Segment)
DX	(Data)
ES	(Extra Segment)
FLAGS	
IP	(Instruction Pointer)
SI	(Source Index)
SP	(Stack Pointer)
SS	(Stack Segment).

The four data registers (AX, BX, CX, and DX) are 16-bit registers that can also be divided into 8-bit registers (AH, AL, BH, BL, CH, CL, DH, and DL, respectively).

Also, after the hex value of the FLAGS register, the system lists the two letter mnemonic associated with each flag. For example, if each flag was clear, you'd see something like:

```
FLAGS 0000 ( NV UP DI NT PL NZ NA PO NC )
```

If the program was loaded with a symbol table, the current code address CS:IP is also listed as a symbol name plus an offset.

If address relocation is in force, the relocation value is subtracted from the four segment registers (CS, DS, ES, and SS).

Note: You can also use the REGS command (Page 11-120) to list the contents of all registers.

Entering LIST REGS gives you the same information as entering REGS.

`RQE n`

lists a formatted listing of RQE `n`. If you omit `n`, the system lists all RQEs.

`RQE AT address`

lists the RQE at the address you specify.

`SFV n`

lists a formatted listing of SFV `n`. If you omit `n`, the system lists all SFVs.

`SFV AT address`

lists the SFV at the address you specify.

`STATUS`

lists the system common session status and device status areas in hexadecimal and character format.

`TIMER n`

lists a formatted listing of timer entry `n`. If you omit `n`, the system lists all timer entries.

Usage Notes:

1. If you enter the LIST command without any parameters, the system lists the same number of bytes as the last LIST command starting where it left off. If you haven't used a LIST command yet, the system lists 16 bytes starting at CS:IP.

2. When you're listing areas of memory as text, the default is to list these areas as ASCII. You can choose to list as a translation from EBCDIC to ASCII. The ASCII (Page 11-18) and EBCDIC

(Page 11-72) commands change the way the system lists memory.

You can change the way the system lists memory *temporarily* by using a special parameter on the LIST command. To do this, you insert (A) or (E) immediately after the LIST command. For example, if you enter:

```
asc
L cs:100
L (e) cs:100
L cs:100
```

The first command tells the system to list in ASCII. The second command tells the system to list 16 bytes starting at location CS:100. The third command tells the system to list the same 16 bytes, but now translate them to EBCDIC. The last command tells the system to list the same 16 bytes with ASCII translation.

3. The DISPLAY (Page 11-54) and PRINT (Page 11-110) commands work exactly like the LIST command.

LOAD

Use the LOAD command to load a program to be debugged. The format of the LOAD command is:

LOad	filespec

Where:

`filespec`

is the name of the program (file) you want debugged. The format of a filespec is:

```
drive id:filename.extension
```

If the file is on the default disk drive, you can omit the `drive id:` field. And, if you omit the `extension` field, the system will use COM by default.

Usage Notes:

1. There are two special file extensions you can use when loading your file: COM and DUM. If you load your file with an extension other than COM or DUM, VMPCDEB will treat your file like a COM file.

2. After you've loaded the file, you have to enter a GO command (Page 11-81) to start execution.

3. See Chapter 5, "Loading a Program" on page 5-1 for more information about loading programs.

L370

Use the L370 command to load a 370 processor memory dump. The format of the L370 command is:

L370	filespec

Where:

`filespec`

is the name of the dump file you want loaded. The filespec must be in the following format:

```
drive id:filename.370
```

If the file is on the default drive, you can omit the `drive id:` field. If you omit the file extension (370), the system will use 370 by default. Whether or not you supply the file extension, it *must* be 370.

Usage Notes:

1. Once you enter this command, the system takes the file you've specified and loads it into memory, starting at address 256K. This file must be 320K bytes in size. Then the system looks for a file with the same filename and an extension of "371" which must be 192K bytes in size. This file is also loaded into memory, starting at address 576K.

2. When loading these files, the system checks to see that your machine has 640K bytes of memory. These two files overwrite the previous contents of memory.

3. This command also resets the device emulation adapter and issues a "reset disk" request to BIOS, if bit 0 and bit 6 at location '250'X equal one.

4. The system does nothing to maintain the integrity of the DOS memory control blocks, nor does it patch the memory size to 256K bytes.

5. Use the D370 command (Page 11-71) to create these dumps.

MEM

Use the MEM command to list the DOS memory blocks allocated to the program you're debugging. The format of the MEM command is:

MEM	

Usage Notes:

1. This command also prints the system memory size which is stored at location 40:13.

2. If bit 7 at location '250'X equals one, this command will also list the VMPCDEB memory blocks.

MEMSIZE

Use the MEMSIZE command to change the system memory size. The format of the MEMSIZE command is:

MEMSIZE	n

Where:

n

is a decimal number from 128 to 640 (inclusive).

Usage Notes:

1. This command is ***permanent***. When you terminate VMPCDEB, the following are set to nK:

   - system memory size (stored at location 40:13 hex)
   - memory control blocks
   - program segment prefix of subsequently loaded programs

2. After you enter this command, the system executes an *implied* DELETE command (Page 11-47).

NAME

Use the NAME command before using the WRITE command (Page 11-146), if you want the name of the program being written to be different from the name of the loaded program. The format of the NAME command is:

Name	filespec

Where:

`filespec`

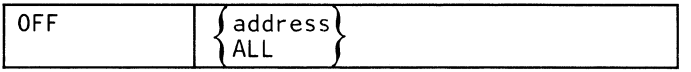
is the name of the file that you want the loaded program written to. The format of a filespec is:

```
drive id:filename.extension
```

If you want the program written on the default disk drive, omit the `drive id:` field. If you omit the `.extension` field, the system will use COM by default.

OFF

Use the OFF command to cancel one or more break-points. The format of the OFF command is:



Where:

`address`

is the address of a specific break-point you want cancelled. If you supply only an offset (instead of a segment address), the contents of the CS register will be used as the segment address.

`ALL`

cancels all break-points.

Usage Notes:

1. For a list of the current break-points, use the AT command (Page 11-19) without any parameters.

2. The DELETE, LOAD, MEMSIZE, and RUN commands also cancel all break-points.

OUTB

Use the OUTB command to output a byte to a port address. The format of the OUTB command is:

OUTB	n m

Where:

n

is the port address of the byte.

m

is the value you want put in the byte.

OUTW

Use the OUTW command to output a word to a port address. The format of the OUTW command is:

OUTW	n m

Where:

n

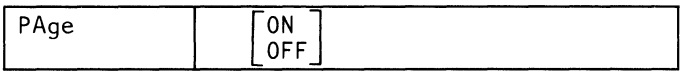
is the port address of the word.

m

is the value you want put in the word.

PAGE

Use the PAGE command to turn the page mode facility on or off. The format of the PAGE command is:



Where:

ON

turns the page mode facility on. When this facility is on and you enter a command that displays more than screen of information, VMPCDEB fills up a screen and then waits for you to press a key before displaying more. You can press any key except Alt, Ctrl, Esc, or Shift to continue. The Alt, Ctrl, and Shift keys have no affect. Pressing the Esc key terminates the command and returns you to command input mode.

OFF

turns the page mode facility off. When this facility is off and you enter a command that displays more than screen of information, VMPCDEB displays all the output without stopping.

Usage Notes:

1. If you enter this command without a parameter, the system reverses the setting of the page mode facility. If page mode was on, it changes to off. If it was off, it changes to on.

2. If you enter this command with the ON parameter, the system sets the page mode flag to one.

3. If you enter this command with the OFF parameter, the system sets the page mode flag to zero.

PC

Use the PC command to list or alter the contents of the Program Counter (PC) register. This register is also referred to as the Instruction Pointer (IP) register. The format of the PC command is:

PC	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

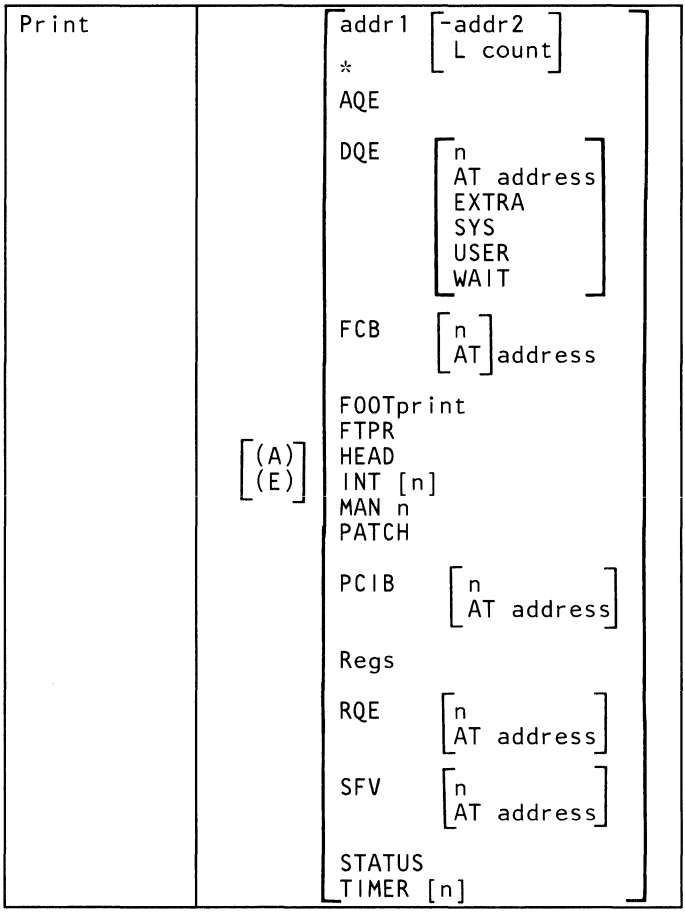
4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`PC 4849`	'4849'X
String	`PC "HI"`	'4849'X
Decimal	`PC #18505`	'4849'X

PRINT

Use the PRINT command to print the contents of various locations of memory, or to create a formatted listing of parts (or all) of the VM/PC System Common. The format of the PRINT command is:



Where:

```
addr1 [-addr2  ]
      [L count ]
```

is the address of the area you want displayed.

If you use `addr1` with nothing after it, VMPCDEB will display 16 bytes starting at that address.

If you use `addr1-addr2`, these are the beginning and ending addresses of the area you want displayed. These two addresses must be separated by a dash (-).

If you use `addr1 L count`, VMPCDEB displays the area starting at `addr1` for the number of bytes you specified in `count`. (The `L` is short for "Length.")

Note: If you don't specify a segment address (only an offset), the system will use the contents of the DS register as the segment address.

`*`

prints the same locations as the last PRINT command.

`AQE`

prints the active AQE and its associated RQE chain pointer.

`DQE n`

prints DQE `n` and its associated RQE chain pointer. If you omit `n`, the system prints all DQEs and their associated RQE chain pointers.

`DQE AT address`

prints the DQE at the address you specify and its associated RQE chain pointer.

`DQE EXTRA`

prints all EXTRA DQEs and their associated RQE chain pointers.

`DQE SYS`

prints all SYS DQEs and their associated RQE chain pointers.

`DQE USER`

prints all USER DQEs and their associated RQE chain pointers.

`DQE WAIT`

prints all WAIT DQEs and their associated RQE chain pointers.

`FCB n`

prints a formatted listing of FCB `n` in hexadecimal and character format. If you omit `n`, the system prints all FCBs.

`FCB AT address`

prints the FCB at the address you specify.

`FOOTprint`

prints the dispatcher footprints table. The system starts with the entry pointed to by the table index within system common and wraps around from the bottom of the table back up to the top. This is the same as the PRINT FTPR command.

`FTPR`

prints the dispatcher footprints table. The system starts with the entry pointed to by the table index within system common and wraps around from the bottom of the table back up to the top. This is the same as the PRINT FOOTPRINT command.

HEAD

prints a formatted listing of the CPIO System Common Header.

INT n

prints interrupt vector entry n. If you omit n, the system prints the first 128 interrupt addresses. These addresses are those used by the program you're testing, which may or may not be the same as those used when VMPCDEB is running.

MAN n

prints the SFV entry, the DQE, and the current registers for the CPIO manager whose SFV index is n. If the manager you specify is the current active manager, the system prints the current registers for the program you're testing (as in a REGS or PRINT REGS command). Otherwise, the system prints the register values that were saved on the specified manager's stack.

PATCH

prints the system common patch area in hexadecimal and character format.

PCIB n

prints a formatted listing of PCIB n. If you omit n, the system prints all PCIBs.

PCIB AT address

prints the PCIB at the address you specify.

prints the contents of the following registers (in hex):

AX	(Accumulator)
BP	(Base Pointer)
BX	(Base)
CS	(Code Segment)
CX	(Count)
DI	(Destination Index)
DS	(Data Segment)
DX	(Data)
ES	(Extra Segment)
FLAGS	
IP	(Instruction Pointer)
SI	(Source Index)
SP	(Stack Pointer)
SS	(Stack Segment)

The four data registers (AX, BX, CX, and DX) are 16-bit registers that can also be divided into 8-bit registers (AH, AL, BH, BL, CH, CL, DH, and DL, respectively).

Also, after the hex value of the FLAGS register, the system lists the two letter mnemonic associated with each flag. For example, if each flag was clear, you'd see something like:

```
FLAGS 0000 ( NV UP DI NT PL NZ NA PO NC )
```

If the program was loaded with a symbol table, the current code address CS:IP is also printed as a symbol name plus an offset.

If address relocation is in force, the relocation value is subtracted from the four segment registers (CS, DS, ES, and SS).

Note: You can also use the REGS command (Page 11-120) to list the contents of all registers.

Entering PRINT REGS gives you the same information as entering REGS.

`RQE n`

prints a formatted listing of RQE `n`. If you omit `n`, the system prints all RQEs.

`RQE AT address`

prints the RQE at the address you specify.

`SFV n`

prints a formatted listing of SFV `n`. If you omit `n`, the system prints all SFVs.

`SFV AT address`

prints the SFV at the address you specify.

`STATUS`

prints the system common session status and device status areas in hexadecimal and character format.

`TIMER n`

prints a formatted listing of timer entry `n`. If you omit `n`, the system prints all timer entries.

Usage Notes:

1. If you enter the PRINT command without any parameters, the system prints the same number of bytes as the last PRINT command starting where it left off. If you haven't used a PRINT command yet, the system prints 16 bytes starting at CS:IP.

2. When you're printing areas of memory as text, the default is to print these areas as ASCII. You can choose to print as a translation from EBCDIC to ASCII. The ASCII (Page 11-18) and EBCDIC

(Page 11-72) commands change the way the system prints memory.

You can change the way the system prints memory *temporarily* by using a special parameter on the PRINT command. To do this, you insert (A) or (E) immediately after the PRINT command. For example, if you enter:

```
asc
p cs:100
p (e) cs:100
p cs:100
```

The first command tells the system to print in ASCII. The second command tells the system to print 16 bytes starting at location CS:100. The third command tells the system to print the same 16 bytes, but now translate them to EBCDIC. The last command tells the system to print the same 16 bytes with ASCII translation.

3. The DISPLAY (Page 11-54) and LIST (Page 11-91) commands work exactly like the PRINT command.

QUIT

Use the QUIT command to terminate VMPCDEB. The format of the QUIT command is:

QUit	

Note: The END command (Page 11-73) works exactly like the QUIT command.

READ

Use the READ command to read the contents of a file as VMPCDEB commands. The format of the READ command is:

READ	filespec

Where:

`filespec`

is the name of the file you want read. The format of a filespec is:

```
drive id:filename.extension
```

If the file is on the default disk drive, you can omit the `drive id:` field. And, the default extension field is DEB. You can omit this field unless your file extension is something other than DEB.

Usage Notes:

1. This command causes the specified file to be read as VMPCDEB commands. VMPCDEB ignores blank lines, null commands, and comments (any line with an asterisk (*) in column one). When all the commands in the file have been executed, control returns to the keyboard.

2. The commands in the file cannot be longer than 80 characters.

3. If there are trailing spaces after the command, they will be removed *before* synonym substitution occurs.

4. You cannot have nested READ commands. If there's a READ command within a READ file, the first file is closed and the rest of the commands are read from the new file.

5. If there's an error reading the file, execution terminates and a message is displayed.

REGS

Use the REGS command to list the contents of all registers. The format of the REGS command is:

Regs	

Usage Notes:

1. This command will display the contents of the following registers (in hex):

AX	(Accumulator)
BP	(Base Pointer)
BX	(Base)
CS	(Code Segment)
CX	(Count)
DI	(Destination Index)
DS	(Data Segment)
DX	(Data)
ES	(Extra Segment)
FLAGS	
IP	(Instruction Pointer)
SI	(Source Index)
SP	(Stack Pointer)
SS	(Stack Segment)

   The four data registers (AX, BX, CX, and DX) are 16-bit registers that can also be divided into 8-bit registers (AH, AL, BH, BL, CH, CL, DH, and DL, respectively).

   Also, after the hex value of the FLAGS register, the system lists the two letter mnemonic associated with each flag. For example, if each flag was clear, you'd see something like:

```
FLAGS 0000 ( NV UP DI NT PL NZ NA PO NC )
```

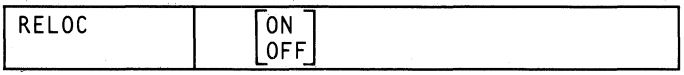
2. If the program was loaded with a symbol table, the current code address CS:IP is also displayed as a symbol name plus an offset.

3. If address relocation is in force, the relocation value is subtracted from the four segment registers (CS, DS, ES, and SS).

Note: You can also use the PRINT command (Page 11-110) to list the contents of all registers. Entering PRINT REGS gives you the same information as entering REGS. Also, you can use PRINT MAN to list the contents of all the registers for a CPIO manager. If the specified manager is the current active manager, entering PRINT MAN gives you the same information as entering REGS.

RELOC

Use the RELOC command to turn the address relocation facility on or off. (When you first start VMPCDEB, the address relocation facility is off.) The format of the RELOC command is:



Where:

`ON`

turns the address relocation facility on.

`OFF`

turns the address relocation facility off.

Usage Note:

If you enter this command without a parameter, the system reverses the status of the address relocation facility. If address relocation is on, it will be turned off. If it is off, it will be turned on.

RENAME

Use the RENAME command to change the name of a file. The format of the RENAME command is:

REName	[d1:][path1]filespec1 [d2:][path2]filespec2

Where:

d1:

is the drive id of the current file. If the file resides on the default disk drive, you can omit this field.

path1

is the name of the subdirectory of the current file. If the file is in the default directory, you can omit this field.

filespec1

is the current name of the file. The format of filespec1 is:

filename.extension

You must specify both the filename and the extension.

d2:

is the drive id of the renamed file. If the drive id of the renamed file is the same as the drive id of the current file, you can omit this field.

path2

is the name of the subdirectory of the renamed file. If you want the renamed file on the default directory, you can omit this field.

`filespec2`

is the new name of the file. The format of filespec2 is:

```
filename.extension
```

You must specify both the filename and the extension.

RESET

Use the RESET command to:

- reset the timer frequency
- reset all asynch cards
- issue a "reset disk" request to BIOS
- reset the device emulation adapter (if bit 0 and bit 6 at location '250'X equal one).

The format of the RESET command is:

RESET	

RUN

Use the RUN command to load and start a program to be debugged. The format of the RUN command is:

RUN	[filespec]

Where:

`filespec`

is the name of the program (file) you want debugged. The format of a filespec is:

```
drive id:filename.extension
```

If the file is on the default disk drive, you can omit the `drive id:` field. And, if you omit the `extension` field, the system will use COM by default.

Usage Notes:

1. There are two special file extensions you can use when loading your file: COM and DUM. If you load your file with an extension other than COM or DUM, VMPCDEB will treat your file like a COM file.

2. See Chapter 5, "Loading a Program" on page 5-1 for more information about running programs.

SHOW

Use the SHOW command to display the current saved screen. Using this command does not corrupt the contents of the screen when the program you're testing is restarted and the screen is restored. The format of the SHOW command is:

SHOW	

SI

Use the SI command to list or alter the contents of the Source Index (SI) register. The format of the SI command is:

SI	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`SI 4849`	'4849'X
String	`SI "HI"`	'4849'X
Decimal	`SI #18505`	'4849'X

SP

Use the SP command to list or alter the contents of the Stack Pointer (SP) register. The format of the SP command is:

SP	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`SP 4849`	'4849'X
String	`SP "HI"`	'4849'X
Decimal	`SP #18505`	'4849'X

SS

Use the SS command to list or alter the contents of the Stack Segment (SS) register. The format of the SS command is:

SS	[n]

Where:

n

is the new value you want in the register. This number can be a:

hexadecimal number
decimal number (preceded by a #)
character string (enclosed in quotes)
null value

Usage Notes:

1. If you enter this command *without* supplying a new value, the system displays the contents of the register (in hex) and waits for a new value. If you enter a null value (i.e., you press the ENTER key), the contents of the register do not change.

2. If you enter this command *with* a new value, the system changes the contents of the register without displaying the old value.

3. You cannot use negative decimal numbers.

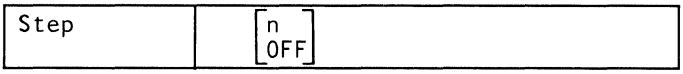
4. No automatic allowance is made for program relocation. When you change (or display) the contents of the register, the system *does not* add (or subtract) the relocation value.

Examples:

	Command	Register Value
Hex	`SS 4849`	'4849'X
String	`SS "HI"`	'4849'X
Decimal	`SS #18505`	'4849'X

STEP

Use the STEP command to turn step mode operation on or off. The format of the STEP command is:



Where:

n

is the number of instructions you want executed each time the program is started. VMPCDEB assumes this is a decimal number, so you don't have to preface it with a pound sign (#). You can use any number between 0 and 9999, inclusive.

OFF

cancels step mode operation.

Usage Notes:

1. If you enter this command without a parameter, VMPCDEB displays the current setting of step mode.

2. Step mode operation is turned off by default. When you enter a STEP command, it is turned on.

3. When step mode operation is on, only the program's instructions are counted. If one of these instructions causes the execution of a DOS or BIOS instruction, the count is not affected.

4. Inputting an integer (instead of a GO command) to start your program, does not affect step mode operation.

5. The DELETE, LOAD, MEMSIZE, and RUN commands cancel step mode operation.

SYMBOL

Use the SYMBOL command to translate an address to symbolic form (symbol name plus offset). The format of the SYMBOL command is:

SYmbol	address

Where:

`address`

is the address you want translated to symbolic form.

Usage Note:

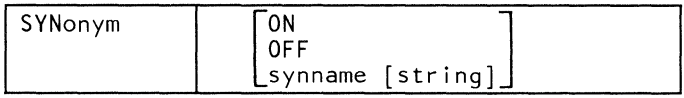
This command only works if you loaded a symbol table.

SYNONYM

Use the SYNONYM command to:

- define a synonym
- delete a synonym
- display a synonym
- turn synonym translation on or off.

The format of the SYNONYM command is:



Where:

`ON`

turns synonym translation on (the default). Synonym translation is in force when bit 1 at location '251'X of VMPCDEB equals one.

`OFF`

turns synonym translation off. Synonym translation is not in force when bit 1 at location '251'X of VMPCDEB equals zero.

`synname`

is a one to four character synonym name. This name can be any combination of letters and numbers, but no special characters are allowed.

Warning: If you specify a numeric "synname," you may run into problems if you try to use that number in another command.

`string`

is whatever value you want assigned to "synname." This string can be any combination of letters, numbers, special characters, and blanks. The system looks for a blank (or blanks) to separate "synname" from "string." When the system finds this blank (or blanks), it assigns everything from the first non-blank character to the end of the record to "string."

Usage Notes:

1. When you turn synonym translation off (using the "OFF" keyword), the system will not make any synonym substitutions. You can still define, delete, and display synonyms, but they won't be translated until you turn synonym translation back on.

2. If you enter this command without any parameters, the system will list the contents of the current synonym table.

3. If you enter this command with a "synname" but no "string," the system deletes that synonym name from the synonym table. If synonym translation is in force, you may run into a slight problem. The system will scan the line, make the substitution, and your command will fail. You need to turn off synonym translation first. You can do this in one of two ways: using the SYNONYM OFF command or prefixing the command with an exclamation point (see Note 8 below).

4. If you enter this command with a "synname" and a "string," the system adds that synonym name to the synonym table. You must separate `synname` and `string` by one or more blanks.

5. You can use synonyms in any VMPCDEB command. It doesn't matter whether you enter the command from your keyboard or from a file (using a READ command).

6. When synonym translation is in force and you have one or more synonyms defined, the system scans all commands for synonyms. If the system makes a substitution on the first scan, it will scan the command again. The system will keep scanning the same command until no synonym translation occurs. This means you can have "nested" synonyms.

7. When the system substitutes a value for a synonym, it checks to see that the resulting command does not exceed 80 bytes. If the command exceeds 80 bytes, the system rejects it and you'll see:

```
Record too long
```

   Because of this, you cannot have recursive synonyms.

8. You can turn off synonym translation for one command by inserting an exclamation point (!) before the command. For example, when you want to delete a command, you'll need to precede the SYNONYM command with an exclamation point or VMPCDEB will translate the synonym before executing the command.

9. Synonyms are at least five bytes long. Each synonym has a different length depending on how many bytes the string takes up. The synonym table cannot be more than 256 bytes total.

TRAP

Use the TRAP command to set, unset, or display the trapping of hardware interrupts. The format of the TRAP command is:

TRAP	[n[OFF]]

Where:

n

is the number of the interrupt vector location. It must be a hexadecimal number from 8 to F, inclusive.

Note: Use Trap 8 and Trap 9 with care! They are used to trap timer and keyboard interrupts, respectively. If you're not careful, your keyboard may lock up.

OFF

(when used with n) cancels one trapping interrupt.

Usage Notes:

1. If you enter this command without any parameters, the system displays the numbers of the interrupts that are being trapped.

2. If you enter this command with a number between 8 and F, the system will trap any interrupts which occur at that interrupt vector location.

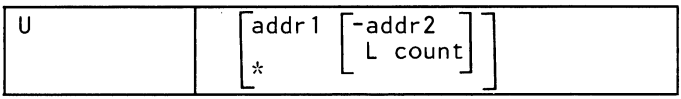
3. If you enter this command with a number and the keyword "OFF," the system cancels that interrupt trapping.

4. When an interrupt occurs, the system makes a note of the interrupt number and executes an INTRET instruction. You are not notified immediately of the interrupt. The system waits for some kind of stopping point, such as a break in, before telling you about any interrupts. When a stopping point arises, the system prints out the interrupt numbers for those locations which have had one or more interrupts. The system prints the locations which have had interrupts since VMPCDEB last regained control or since the last such message was printed (whichever comes first). The system does not keep track of how many times one location was interrupted. The location may have been interrupted only once or it may have been interrupted fifty times.

5. The DELETE, LOAD, MEMSIZE, and RUN commands cancel all interrupt trapping.

U

Use the U command to dis-assemble (or "Unassemble") a specified area of memory. The format of the U command is:



Where:

`addr1 [-addr2 ]`
`[L count]`

is the address of the area you want displayed.

If you use `addr1` with nothing after it, VMPCDEB will display 16 bytes starting at that address.

If you use `addr1-addr2`, these are the beginning and ending addresses of the area you want displayed. These two addresses must be separated by a dash (-).

If you use `addr1 L count`, VMPCDEB displays the area starting at `addr1` for the number of bytes you specified in `count`. (The `L` is short for "Length.")

Note: If you don't specify a segment address (only an offset), the system will use the contents of the DS register as the segment address.

*

tells the system to dis-assemble the same locations of memory as the last U command.

Usage Notes:

1. When you first start VMPCDEB, bit 0 at location '251'X of VMPCDEB equals one. This causes VMPCDEB to load a separate code module (held in IWSUNPIK.DEB) as an "add-on" to itself. This module allows you to print areas of memory as dis-assembled instructions. It takes up about 5.25K bytes of memory.

2. When you're testing a program and control returns to VMPCDEB, VMPCDEB displays the program's next instruction in dis-assembled form (if IWSUNPIK.DEB was loaded). If this next instruction accesses memory, the contents of memory are also displayed. If, at this point, you enter a REGS (or PRINT REGS) command, VMPCDEB will display the program's next instruction in dis-assembled form and the contents of memory (if applicable).

3. If you enter this command without any parameters, the system will dis-assemble the same number of bytes as the last U command, starting where it left off.

4. The system always dis-assembles whole instructions. If the range you specify is too short, the system will go beyond that range to finish dis-assembling the instruction.

   Example:

   If you enter:

   ```
   u cs:ip
   ```

   the system will dis-assemble 16 bytes of instructions starting at the address in CS:IP. If, after 16 bytes, there is still part of an instruction left to dis-assemble, the system will continue dis-assembling until the entire instruction is disassembled. Let's say the instruction fills up two more

bytes. That means the system would dis-assemble 18 bytes starting at the address in CS:IP. Now, if you enter:

```
u
```

the system will dis-assemble 16 bytes starting at location CS:IP + 18. Even though the system dis-assembled 18 bytes in the last command, the range is still 16 bytes, because that's the default unless you specify otherwise. The system starts dis-assembling at CS:IP + 18, because that's where it left off.

VERSION

Use the VERSION command to display the VMPCDEB version and service level numbers. The format of the VERSION command is:

VERSION	

When you enter this command, you'll see something like:

```
VMPC Debug version 1.10  service level nnnn
```

VMPCDEB

Use the VMPCDEB command to start the VMPCDEB program. The format of the VMPCDEB command is:

```
[d:]VMPCDEB
```

Where:

`d:`

is the disk id where the VMPCDEB program resides. If VMPCDEB resides on the default disk drive, you can omit this field.

WRITE

Use the WRITE command to write the program you're testing to disk. The format of the WRITE command is:

Write	[n]

Where:

n

is the number of "paragraphs" (number of bytes divided by 16) to be written to disk. Only specify this parameter if the size of the program has changed since it was loaded.

Usage Notes:

1. This command takes the program you're testing and writes it back out to disk. Unless you've used the NAME command (Page 11-103), the program written to disk will have the same name as the loaded program.

2. The system writes the program as a straight copy of memory. It starts at address '0100'X within the program you're testing. If you omit the parameter in the WRITE command, the system writes the same number of bytes out to disk as there were bytes read in from disk when the program was loaded.

3. It is not a good idea to write a program out to disk after it has been running for a while.

4. To patch a program, the sequence of commands is:

```
LOAD
ALTER (if necessary)
NAME (to avoid writing over the original file)
WRITE
```

Note: Unlike the DOS DEBUG facility, you can use VMPCDEB in this manner to patch COM programs that are larger than 64K bytes.

Chapter 12. Command Summary

The table below lists all of the VMPCDEB commands and their associated functions.

Command	What it does
!	turns off synonym translation for one command.
A	performs simple integer arithmetic (add, subtract, multiply, and divide).
ADDSYM	reads in a supplementary symbol table file.
AH	lists or alters the contents of the AH register.
AL	lists or alters the contents of the AL register.
ALTer	changes memory contents or alters an interrupt vector.
ASCii	displays output of PRINT command in ASCII format (default).
AT	sets or displays break-points.
AX	lists or alters the contents of the Accumulator (AX) register.
BEEP	turns the speaker on or off.
BH	lists or alters the contents of the BH register.
BL	lists or alters the contents of the BL register.
BP	lists or alters the contents of the Base Pointer (BP) register.
BX	lists or alters the contents of the Base (BX) register.

Command	What it does
BYTE250	displays or changes the value of the byte at location '250'X.
BYTE251	displays or changes the value of the byte at location '251'X.
CD	displays or changes the current directory path.
CH	lists or alters the contents of the CH register.
CHDIR	displays or changes the current directory path.
CL	lists or alters the contents of the CL register.
CLEAR	clears the current VMPCDEB screen.
CLS	clears the current VMPCDEB screen.
CS	lists or alters the contents of the Code Segment (CS) register.
CX	lists or alters the contents of the Count (CX) register.
DELete	deletes the program currently under test.
DH	lists or alters the contents of the DH register.
DI	lists or alters the contents of the Destination Index (DI) register.
DIR	lists the contents of a directory.
Display	displays the contents of various locations of memory or creates a formatted listing of parts (or all) of the VM/PC System Common.
DL	lists or alters the contents of the DL register.
DRIVE	displays or alters the default disk drive.
DS	lists or alters the contents of the Data Segment (DS) register.
DUMP	creates a memory dump.
DX	lists or alters the contents of the Data (DX) register.

Command	What it does
D370	creates a 370 processor memory dump.
EBCdic	displays output of PRINT command in EBCDIC format.
END	terminates VMPCDEB.
ERASE	erases a DOS file.
ES	lists or alters the contents of the Extra Segment (ES) register.
Fill	replaces the contents of all bytes in the address range with a specified value.
FLAGS	lists or alters the contents of the FLAGS register.
Go	starts a program that has beendloaded.
H	performs simple integer arithmetic (add, subtract, multiply, and divide).
HIDESYM	"hides" a symbol name so that VMPCDEB will not use it when printing addresses.
IMR	lists or alters the contents of the Interrupt Mask Register (IMR).
INB	inputs a byte from a port address.
INT	simulates an interrupt.
INW	inputs a word from a port address.
IP	lists or alters the contents of the Instruction Pointer (IP) register (also known as the Program Counter or PC register).
List	displays the contents of various locations of memory or creates a formatted listing of parts (or all) of the VM/PC System Common.
LOad	loads a program to be debugged.

Command	What it does
L370	loads a 370 processor memory dump.
MEM	list the DOS memory blocks allocated to the program being tested.
MEMSIZE	change the memory size of the system (as seen by DOS).
Name	names a program to be written to disk using the WRITE command.
OFF	cancels one or more break-points.
OUTB	outputs a byte to a port address.
OUTW	outputs a word to a port address.
PAge	turns the page output facility on or off.
PC	lists or alters the contents of the Program Counter (PC) register (also known as the Instruction Pointer or IP register).
Print	displays the contents of various locations of memory or creates a formatted listing of parts (or all) of the VM/PC System Common.
QUit	terminates VMPCDEB.
READ	reads the contents of a file as VMPCDEB commands.
Regs	lists the contents of all registers.
RELOC	turns the address relocation facility on or off.
RENname	changes the name of a file.
RESET	resets timer frequency and all asynch cards.
RUN	loads and starts a program.
SCReen	turns screen swap mode on or turns screen mode facility on or off.
SHOW	displays the current saved screen.

Command	What it does
SI	lists or alters the contents of the Source Index (SI) register.
SP	lists or alters the contents of the Stack Pointer (SP) register.
SS	lists or alters the contents of the Stack Segment (SS) register.
Step	turns step mode operation on or off.
SYmbol	translates an address to symbolic form.
SYNonym	defines a synonym, displays current synonyms, or turns synonym translation on or off.
TRAP	sets, unsets, or displays trapping of hardware interrupts.
Unassemble	dis-assembles a specified area of memory.
VERSION	displays the VMPCDEB version and service level numbers.
VMPCDEB	starts VMPCDEB.
Write	writes the program being tested to disk.

Part 3. Using CPIO DUMPER

Chapter 13. Starting DUMPER

There is an internal subroutine of CPIO which writes all of real memory onto diskettes. Throughout this section, we'll be referring to this subroutine as "DUMPER."

To use DUMPER, you must enter a special option when you enter the VMPC command. When you start VM/PC, enter:

```
C>vmpc /d
```

When you use the `/d` option, you won't notice any differences in VM/PC. This option sets it up so that you can create your dump whenever you want, but it doesn't interfere with the normal operations of VM/PC.

When you want DUMPER to start creating your dump, press the system reset keys:

```
Ctrl-Alt-Del
```

Because you used the `/d` option, pressing the system reset keys causes DUMPER to go to work instead of causing a system reset.

Once DUMPER takes control, it prompts you for what it needs. The following is an example of what you would see if you were writing 640K of memory onto two diskettes.

When you press the system reset keys, the screen clears and you'll see:

```
Starting VMPC Memory Dump Routine.
The only way to terminate this routine is to power down.

Insert new formatted diskette in drive A.
Strike any key when ready.
```

At this point, the only way you can stop DUMPER is to turn off all your equipment. To continue, insert the first diskette in drive A[1] and press any key.

Then you'll see:

```
Dumping memory.
```

After a while, you'll see:

```
Diskette # 1 is full.
Insert new formatted diskette in drive A.
Strike any key when ready.
```

DUMPER has finished with the first diskette, and is ready for the second one. Take out the first diskette and insert the second one. Then press any key.

You'll see:

```
Continuing ...
Dumping memory.
```

When DUMPER is done, you'll see:

```
Memory dump complete.
The processor has been halted.
```

Take out the second diskette and ***turn off all your equipment!*** You cannot return to where you were and continue working. You must start VM/PC again and go from there.

---

1 DUMPER always writes onto diskettes in drive A. Changing the default disk drive before starting DUMPER has no effect.

Warning: Pressing system reset (Ctrl-Alt-Del) while DUMPER is working will cause unpredictable results.

Design

DUMPER does not rely on any DOS functions. It uses BIOS routines for reading memory, writing to diskettes, and for all of the screen and keyboard I/O. These routines reside in ROM, which keeps the amount of critical memory (memory never overwritten) needed to take a dump down to a minimum.

To take a dump, you need three critical areas of memory:

1. **CPIO keyboard interrupt handler**

2. **CPIO dump subroutine (DUMPER)**

3. **keyboard interrupt vector** - stored in 4 bytes in low memory. Other interrupt vectors can be overwritten since all other vectors needed by DUMPER are restored from ROM before the dump begins. The keyboard interrupt vector must be valid because it is the vector that points to the CPIO keyboard interrupt handler, which in turn traps the Ctrl-Alt-Del key sequence and branches to DUMPER.

If these three areas of memory (totalling about 2K) were not overwritten and keyboard interrupts were not masked, then you can create a dump.

Contents of the Dump Diskette

The diskettes that DUMPER creates ***are not in DOS format!*** DUMPER places its output onto the diskette in the format described in the next section. Each diskette contains a 512 byte piece of the memory dump. Each of these pieces of memory is stored in one good sector of the diskette. Any bad sectors are skipped.

DUMPER requires that you use formatted diskettes, but DUMPER destroys the File Allocation Table, the DOS directory, and any files which may have been on the diskette. DUMPER reserves the first track on each side of the diskette for a table. That table coordinates a real memory address with the track, head, and sector of the diskette that it resides on. The table also contains a checksum for each sector and a unique label for identification.

Format of the Dump Diskettes

All diskettes created by DUMPER have the following format:

Head 0
Sector 1 — Not used.

Head 0
Sector 2 — VMPCREF uses the following bytes when reformatting:

Bytes	Meaning
0	Status (number of sectors in Track 0 which could not be written)
1-20	the ASCII characters 'IBM VMPC MEMORY DUMP'
21	Diskette Number
22-26	Identifying Number for diskettes belonging to the same dump (the number of counts since power on)
27-28	CPIO segment (the start of CPIO)
29-30	CPIO displacement (always zero)
31-32	Highest Segment (lowest segment which is greater than any memory written on this diskette)

Bytes	Meaning
33-34	Lowest Segment (segment which is the lowest memory location written onto this diskette)
35-62	Contents of registers AX, BX, CX, DX, SP, BP, SI, DI, IP, CS, DS, SS, ES, and FLAGS (in that order) when you started DUMPER
63-158	Interrupt Vectors 8 to 1F (absolute memory addresses 20 to 7F)
159-164	Bad Segments (segment addresses for three sectors which are known to have invalid checksums)
165-166	Number of paragraphs of PC storage (Segment address of one byte beyond valid PC address)

Head 0 Sector 3 to Head 1 Sector 2

The address/sector/checksum table which is used as a directory for the diskette. The table is almost nine sectors long and has one entry for each sector of the diskette (except Track 0). The first entry in the table is for Track 1 Head 0 Sector 1. The sectors are ordered first by track, then by head and lastly by sector.

Each entry is six bytes with the following format:

Byte	Meaning
1	Status - This byte will be: '77'X if a block of memory was written to the sector (bytes 2 to 6 are only used when the status byte is '77'X, the rest of the time they are zero) 'FF'X if the sector could not be written to '00'X if the sector was never used
2	Checksum High Byte
3	Checksum Middle Byte
4	Checksum Low Byte - three byte arithmetic sum of the 512 bytes in that sector
5	Segment Address Low Byte
6	Segment Address High Byte - segment portion of the address of the first byte of memory written to this sector (the displacement will always be zero)

Head 1 Sector 3 to Head 1 Sector 9 — Filled with zeros.

TRACK 1 - 39

Each sector of these tracks will either contain the contents of a 512 byte block of memory, or will not be used by this routine. The status byte contained in the table will indicate what is actually in each sector.

Notes About Dumper

1. After pressing the Ctrl-Alt-Del keys to invoke DUMPER, be sure to follow the directions exactly. If you don't, your results will be unpredictable and may affect the value of the dump.

2. When you're creating a dump, DUMPER asks you to insert a formatted diskette. Be careful not to insert a diskette containing information you want to keep. DUMPER does not check to see if the diskette is blank and will overwrite the files on the diskette.

3. There will always be bad checksum values in the address table (in Track 0) for the sector:

   - containing the BIOS return code
   - where the DUMPER stack segment begins
   - where the DUMPER stack segment ends

   These areas are changed by BIOS after the write has been requested. There is no way of predicting the contents, so the checksums will never be correct. The header record of each diskette notes the sectors with bad checksums, and VMPCREF will ignore these checksums. If, by chance, the stack begins and ends in the same 512 byte sector,

there will only be two bad checksums instead of three.

4. Since 370 private storage is dumped, a copy of the 370 PSW and registers (control, floating point and general), are available in the dump. **This copy will be correct only when the 370 processor was stopped before the dump**. To stop the 370 processor, switch to the 370 processor control session and toggle F2 (the stop switch). This causes the 370 processor to stop and execute instructions to update private store with the current registers and PSW.

Chapter 14. Reformatting DUMPER Diskettes

Because DUMPER diskettes are not in DOS format, you must reformat them before you can reload the dump. There is a special program, VMPCREF, which reformats DUMPER diskettes into four files (in DOS format) containing all of real memory. These files can then be used by VMPCDEB.

1. `VMPC1.DUM` - (DOS memory)

   Contains real memory from location '00'X to CPIO's starting address. CPIO's starting address is variable, therefore the length of this file will vary. VMPCREF appends a header record to this file.

2. `VMPC2.DUM` - (CPIO memory)

   Contains real memory from CPIO's starting address '3F000'X. CPIO's starting address is variable, therefore the length of this file will vary. VMPCREF appends a header record to this file.

   Note: VMPC1.DUM and VMPC2.DUM hold the low 256K of real memory.

3. `VMPC370.370` - (first part of 370 memory)

   Contains real memory starting with address '3F001'X. There is no header record for this file.

4. VMPC370.371 - (second part of 370 memory)

   Contains real memory up to the last address (which could vary). There is no header record for this file.

   Note: VMPC370.370 and VMPC370.371 contain all of 370 memory: main memory and private memory. VMPCREF puts 370 memory into two files in case you want these files loaded onto diskettes. The size of each file is not significant, because VMPCDEB loads both files together. If you add more memory, VMPC370.371 grows in size.

The reformat program also has an option to display any real address of memory from the CPIO dump diskettes. This is useful if you just want to check a few areas of memory.

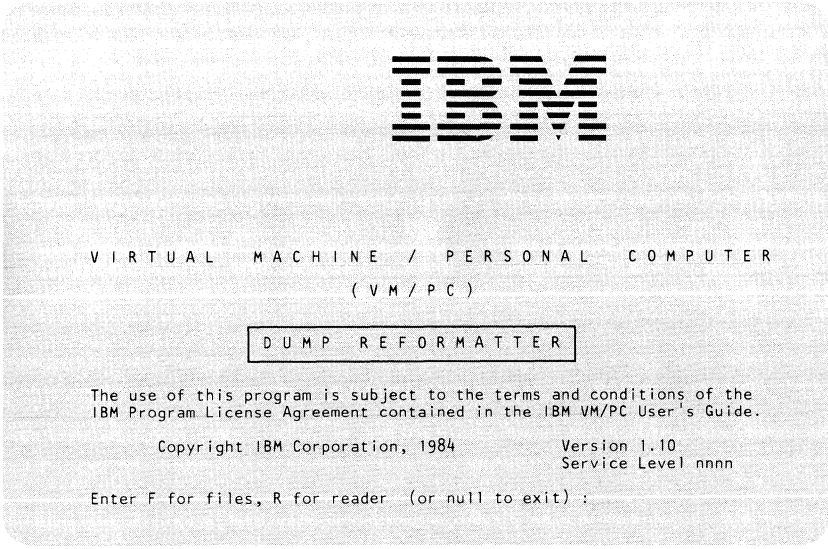
Starting VMPCREF

To start reformatting diskettes, enter:

```
C>vmpcref
```

VMPCREF prompts you for input.

The following is an example of what you would see if you were reformatting two diskettes onto your fixed disk (the most common example). Once you enter the VMPCREF command, you'll see:



The last line is asking you whether you want to reformat the diskettes into the four VMPCREF files or just read the diskettes and display a sector on the screen. In this example, we'll be creating the four files, so enter:

```
f
```

(If you wanted to display memory, you would have entered an R. Then you would be prompted for the real hexadecimal address you wanted displayed.)

Next, you'll see:

```
Insert diskette in Drive A
Strike any key when ready
```

Insert the first diskette and press any key. VMPCREF checks the diskette to make sure it is a VM/PC dump diskette. Then it checks to make sure it's the first diskette in the series (if the dump is on more than one diskette). Next, you'll see:

```
Enter drive to write files to, either B, C, or D
a null line will default to the C drive :
```

Since you're reformatting the dump onto your fixed disk, all you have to do is press the ENTER key. After a while you'll see:

```
Finished with this dump diskette
Get dump diskette #2 ready

Insert diskette in Drive A
Strike any key when ready
```

Take out the first diskette and insert the second one. Then press any key.

VMPCREF checks to make sure the diskette you just inserted is from the same dump as the first diskette, and it checks to make sure this is the next diskette in the series. Then VMPCREF reads each sector and verifies the appropriate checksums. (If there are any problems, you'll see error messages.)

After a while you'll see:

```
The reformatting of the dump has been completed
All checksums for this dump were valid
```

Take out the second diskette. You should be in DOS. You can now start up VMPCDEB and load the dump.

Notes About Reformatting

1. You can load the reformatted dump files in VMPCDEB, and look at everything, but you cannot restart a program from the dump files using VMPCDEB.

2. If the files are being written to a hard disk (C or D), the program checks the hard disk that you selected and only continues if there is enough space available for all four files. If the files are being written to the diskette drive (B), the program requests a diskette and checks for enough space each time a file is output.

3. When an address is requested for display, the address must be input in hex, it must be a real addresses not a segment and offset type (i.e. use 1100 not 100:100).

4. DUMPER modified interrupt vectors 8-1F ('20'X through '7F'X in the first file) to insure that the BIOS subroutines would run. The original values are put into the header record of the first and second file, and that means that VMPCDEB will display the interrupt vectors that were present when the system was dumped. But remember that the first file has the corrected set of interrupt vectors, not the original (although they are probably identical unless you were writing in low store).

Part 4. Using the 370 Processor Control Session

Chapter 15. The 370 Processor Control Session

The 370 Processor Control Session is a general purpose debug facility similar to the debug facilities found on the operator console of other IBM 370 processors. It runs on the IBM PC XT/370 and the IBM PC AT/370 as one of several concurrent sessions. It can be entered from any other session and conversely, can exit to any other session.

The 370 Processor Control Session allows you to debug code by providing the following functions to the 370 processor:

- Stopping and starting the processor
- Instruction stepping the processor
- Stopping the processor by instruction address compare
- Reset and Clear Reset of the 370 processor
- Generation of an External Interrupt to the 370 processor
- Full screen editing of the following:
  - the 370 General Purpose Registers
  - the 370 Floating Point Registers
  - the 370 Control Registers
  - the 370 Program Status Word Register
  - the 370 Storage, both Real and Virtual
  - the 370 Page Address Table

Screen Format

The 370 Processor Control Session is a full screen utility designed to increase the productivity of debugging sessions through the use of such features as full screen register and storage editing. There are no commands to remember. All actions are communicated to the 370 Processor Control utility through the use of function keys. The description of these keys is on the screen in case you should forget them (there are very few). The 370 Processor Control Session was also designed to be keystroke efficient. That is, actions you request require a minimum amount of keystrokes.

The 370 Processor Control screen is divided into four areas as indicated in Figure 15-1. The documentation that follows references these areas. Figure 15-2 on page 15-3 shows the actual layout of the screen.

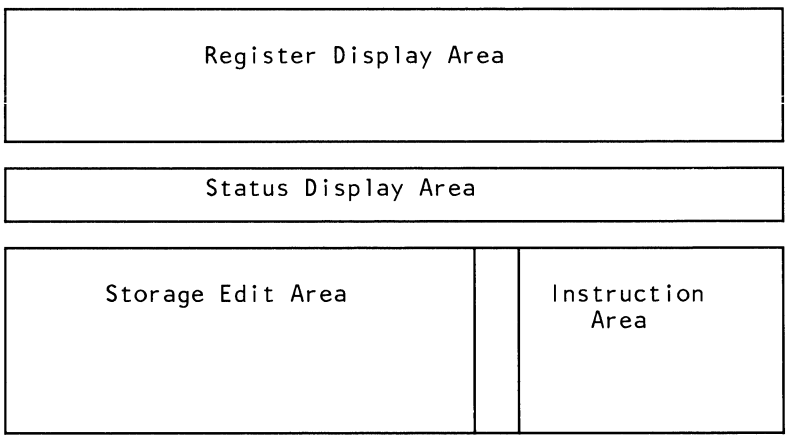


Figure 15-1. Four Areas of 370 Processor Control Screen

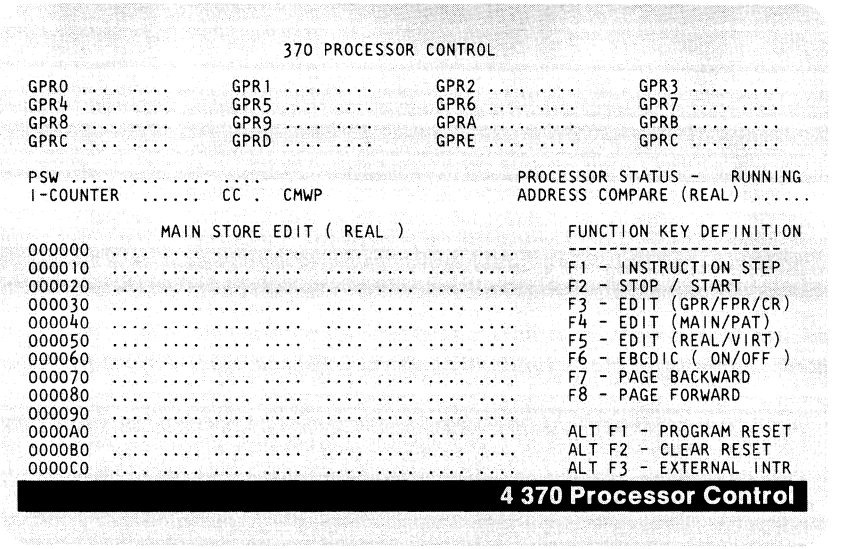


Figure 15-2. Actual Layout of 370 Processor Control Screen

F2 Key - Start/Stop

You use the F2 key to start and stop the 370 processor. Normally, when you enter the 370 Processor Control session, the processor will be in the RUNNING state. (See Processor State below.) To stop the processor, press the F2 key. The processor status field on the screen indicates that the processor is STOPPED. To start the processor, press the F2 key again. Note that you use the F2 key to both stop and start the processor. If the processor is stopped, pressing F2 starts it. If the processor is started, pressing F2 stops it. Many of the other function keys have this same reverse action.

When you stop the processor, the register and storage areas on the screen are unlocked. You can then move the cursor around the screen and change register and storage contents. Conversely, when you start the 370 processor, these areas are locked.

Processor Status Field

The 370 processor can be in one of 5 states. The processor status indicator field, located in the status area of the screen, identifies the processor state. This field is the only one on the screen that blinks.

The five possible processor states are:

1. STOPPED - indicates the processor is not executing instructions, so you are able to alter storage and register contents.

2. RUNNING - indicates the processor is executing instructions or could be in the 370 wait state. You cannot alter storage and register contents in this state.

3. ADDR-COMPARE - indicates that an address compare match has occurred. This state will appear on the screen automatically. While you are in this state, the processor is stopped. To exit this state, use the F1 (Instruction Step) or F2 (Start) keys.

4. INSTR-STEP - indicates that the processor is executing one 370 instruction at a time. Each time you press the F1 key, the processor will execute one 370 instruction. While you are in this state, the processor is stopped. To exit this state, use the F2 (Start) key.

5. ISTEP-WAIT - indicates that the processor is in the "Instruction Step - Wait" state. You get to this state by pressing the F1 key while the processor is in the wait state. The 370 instruction counter does not advance in this state because the processor will not execute instructions in the wait state. While you are in this state, the processor is stopped. To exit this state, use the F2 (Start) key.

F1 Key - Instruction Step

You press the F1 key if you want to have the processor execute one instruction at a time. If the processor was in the "RUNNING" state, pressing F1 has the same effect as stopping the 370 processor, and the status area indicates the "STOPPED" state. Each time you press the F1 key, the 370 processor will execute one 370 instruction. The status area will indicate "INSTR-STEP." You would normally exit from this state by pressing the F2 (Start) key.

Editing 370 Registers

The registers you would most commonly edit are the 370 General Purpose Registers (GPRs) and the 370 Program Status Word Register (PSW). Both are displayed on the screen.

The 370 Program Status Register is always displayed on the screen and you may edit it when the processor is in the "STOPPED" state. If the processor is not stopped, the field is locked.

Note that the 370 Instruction Counter, the 370 Condition Code, and the CMWP fields are taken from the PSW and displayed immediately below the 370 Program Status Word Register. You may not change these fields. If you want to change them, you change the PSW.

F3 Key - Edit (GPR/FPR/CR)

You can edit the 370 GENERAL PURPOSE REGISTERS if they are displayed on the screen and the fields are unlocked. Again, the fields are unlocked if the processor is stopped.

If you want to edit the 370 FLOATING POINT REGISTERS or the CONTROL REGISTERS, they must be displayed on the screen. Press the F3 key to display them.

When you press the F3 key the register display area of the screen is overlayed with a different set of registers. Initially, the 370 General Purpose Registers are displayed in this area. If you press F3, the Floating Point registers are displayed. Press F3 again and the Control Registers are displayed. Press F3 again and the General Purpose registers are again displayed. You can move through displays of these 3 sets of registers by pressing the F3 key.

Editing Storage

There are three kinds of storage you can edit using the 370 Processor Control Session:

- 370 real main storage
- PC storage
- private storage

When you first see the 370 Processor Control Screen, you'll see:

```
MAIN STORE EDIT ( REAL )
```

on the first line of the Storage Edit Area (see Figure 15-1 on page 15-2). This says you're editing 370 real main storage.

To edit PC storage, enter a question mark (?) in the first storage address field, like so:

```
?00000 .... .... .... ....
000010 .... .... .... ....
```

You can enter this question mark when you're editing either main storage or private storage. Either way, the first line of the Storage Edit Area changes to:

```
IBM PC EDIT
```

When you want to get back to main storage, enter another question mark.

To edit private storage (370 microcode), enter an exclamation point (!) in the first storage address field, like so:

```
!00000 .... .... .... ....
000010 .... .... .... ....
```

You can enter this exclamation point when you're editing either main storage or PC storage. Either way, the first line of the Storage Edit Area changes to:

```
PRIVATE STORE EDIT
```

When you want to get back to main storage, enter another exclamation point.

No matter which kind of storage you're editing, the next thirteen lines display sixteen bytes of storage. In the first column of each of these thirteen lines, there is a six digit address. The other eight columns are the contents of storage. If the processor is stopped, these fields are unlocked and you may edit them.

You can change the starting address of the storage display by typing an address over the first address (000000). This is the only address that isn't highlighted, and the cursor is generally placed at the beginning of this field. You don't have to type leading zeros for the address, but can type any non-valid hexadecimal character after the last entered digit. If you enter an

invalid address, such as too large an address, the 370 Processor Control utility forces the address to zero.

The storage address field is unlocked, even if the processor is running. You may display storage at all times but can only alter it if the processor is stopped. You must press the enter key to refresh the storage data on the screen. In other words, the storage data on the screen is not constantly updated as a running program is changing it. You must request periodic updates, if required, by pressing the ENTER key. You cannot update displayed registers this way. The contents of displayed registers are only updated when the processor is stopped.

If you enter an invalid character in any of the input fields (except the storage edit address and address compare stop fields), the alarm will sound, the input field will be displayed in reverse video format, and the cursor will be placed under the invalid character prompting you for a correction.

F7 and F8 Keys - Page Backward and Forward

Press the F7 key to have the next (higher address) block of storage displayed on the screen. Press the F8 key to have the previous (lower address) block of storage displayed.

F6 Key - EBCDIC (On/Off)

Press the F6 key to display the EBCDIC representation of the data. The EBCDIC representation overlays the instruction area of the screen. Press the F6 key again to have the instructions displayed again. Since the instruction display is generally more important than the EBCDIC display, if you press almost any of the attention generating keys (F1, F2, F3, etc.) the instructions reappear on the screen. Only the F7, F8, and ENTER keys don't cause this to happen.

F5 Key - Edit (Real/Virtual)

Press the F5 key to display VIRTUAL STORAGE. The virtual storage display overlays the real 370 storage display. Press F5 again to display the real 370 storage again. By pressing F5 you can switch between displays of real and virtual storage in the storage display area of the screen. If the data for a virtual address is not currently in storage, the storage contents will be displayed as "X"s and the fields will be locked.

F4 Key - Edit PAT/Edit Main

Press F4 to edit the PAT (Page Address Table). This is the hardware table that maps virtual addresses to real addresses. The PAT edit screen data overlays the 370 storage edit data. Pressing the F4 key again causes the storage edit data to be displayed again.

The PAT edit format has a column of virtual addresses on the far left of the screen, followed by columns of the corresponding real addresses, the valid bits, the reference bits, and the change bits respectively. For further information on the PAT, see the *IBM PC XT/370 Technical Manual*, and the section on Dynamic Address Translation in the *370 Principles of Operations*.

You should only alter the PAT data if you absolutely know what you're doing. Virtual storage management is a function of the 370 operating system. Altering this data could cause unpredictable results.

Address Compare Stop Function

The address compare feature automatically stops the 370 processor when it detects an instruction execution at an address you've entered. To use this function, enter the address in the Address Compare Field located in the status area on the screen. You need not supply leading zeros of the address, but may type any non-valid hexadecimal character after the last entered digit. If you enter an invalid address, such as too large, the 370 Processor Control utility makes address compare inactive. You can cancel an address compare simply by typing a non-hexadecimal character in the first digit location.

This function has some limitations:

- It is only an instruction compare stop and not a data compare stop.
- The address entered must be the address of the start of the instruction. Entering an address in the middle of an instruction could cause the program to have unpredictable results.
- It only works with addresses in real 370 storage (as opposed to virtual addresses).

ALT F1 Keys - Program Reset

If you simultaneously press the Alt key and the F1 key, you cause a Program Reset to occur to the 370 processor (see Program Reset in *370 Principles of Operations*). The processor is in the "STOPPED" state at the end of this operation.

ALT F2 Keys - Clear Reset

If you simultaneously press the Alt key and the F2 key, you cause a Clear Reset to occur to the 370 processor (see Clear Reset in *370 Principles of Operations*). This clears main storage. The processor is in the "STOPPED" state at the end of this operation.

ALT F3 Keys - External Interrupt

If you simultaneously press the Alt key and the F3 key, you cause an External Interrupt to occur to the 370 processor (see External Interrupt in *370 Principles of Operations*).

Appendix A. Messages

A

`A synonym has been deleted`

You used the SYNONYM command (Page 11-136) to change the value assigned to an existing synonym name. For example:

Original Command	`SYN A B`
Command to change value	`!SYN A C`

`Address is outside program`
`Alterations completed`

You used the ALTER command (Page 11-15) with an address that was below the start of the program segment prefix. VMPCDEB has accepted your command and has made the changes.

`Address is outside program`
`Changes not made`

You used the ALTER command (Page 11-15) with an address that was below the start of the program segment prefix. VMPCDEB ***did not*** accept your command. Nothing was changed.

B

`Bad drive input, try again`

VMPCREF asked you for the letter of the drive you wanted the dump file written to. The drive letter that you supplied does not exist. To continue, enter a new drive letter.

`BEWARE - There were invalid checksums on this dump.`

You'll see this message from VMPCREF when the dump is invalid. If you still want to create a dump, you must start all over again.

`Break-in at 'nnnn:nnnn'`

This is an informational message telling you the address where your program stopped. You'll see this message when you press Ctrl-Break while your program is running. At this time, VMPCDEB has control.

`Break-in - use the command END to return to DOS`

You pressed Ctrl-Break while VMPCDEB had control. You can use Ctrl-Break to break-in while your program is running, but you cannot use it to break-in on VMPCDEB. If you want to stop VMPCDEB, enter an END or a QUIT command.

`Break-point trap at 'nnnn:nnnn'`

You used the AT command (Page 11-19) to set a break-point, and your program stopped execution at the address displayed in this message.

`Byte '250'X = 'nn'`

This is an informational message telling you the contents of the byte at location '250'X of VMPCDEB. See the BYTE250 command (Page 11-31) for information about what each individual bit represents.

`Byte '251'X = 'nn'`

This is an informational message telling you the contents of the byte at location '251'X of VMPCDEB. See the BYTE251 command (Page 11-33) for information about what each individual bit represents.

C

`Command format error`

You entered a VMPCDEB command with an option that the system didn't recognize. The commands are listed in alphabetical order starting on Page 11-8. Look up the appropriate command and check the command format.

`Command not recognized`

You entered a command that VMPCDEB does not recognize. Check to make sure you spelled the command correctly. The commands are listed in alphabetical order starting on Page 11-8.

`COMMON address is not set up`

The program you loaded has not run far enough to set up the System Common area.

`Current directory = 'd':/'sub'`

This is an informational message telling you the current directory path of the default drive (or any drive you specify). VMPCDEB replaces **d** with the letter of the default drive and **sub** with the name of the subdirectory (if there is one). You'll see this message when you use the CHDIR (or CD) command.

D

`Default disk drive = 'd'`

You used the DRIVE command (Page 11-63) which causes VMPCDEB to display the default disk drive letter. This is the same as the DOS default disk drive.

`Dis-assembly code loaded`

VMPCDEB loaded IWSUNPIK.DEB (the file containing the dis-assembler code).

Dis-assembly code NOT loaded
VMPCDEB could not find IWSUNPIK.DEB (the file containing the dis-assembler code).

Dump file is corrupt
You're trying to load a dump file which is not the correct length, and so must have been corrupted.

E

End of read file
VMPCDEB reached the end of the command file it was processing.

ERROR - Address was not found in the table.
VMPCREF displays this message when you try to display the contents of a memory address that is not in the dump. To continue, enter a new address.

ERROR - Unable to write to diskette in drive A.
There's a problem with the diskette you inserted in the diskette drive. To fix this problem, take out the diskette and insert it again. Make sure the diskette was not inserted upside down and that the diskette door is closed. Then press any key to continue. If you see the message again, you have a bad diskette. Get another diskette and insert it. Then press any key to continue.

Error reading symbol table
VMPCDEB could not read the symbol table file (CPIO.SYM).

Error writing table
Your diskette has gone "bad" while you were using the CPIO dump routine to create a memory dump. After CPIO writes the contents of memory on the diskette, it tries to write a sector of the address/checksum table in Track 0. When you get this message, CPIO was unable to write to this sector. CPIO will continue to write to any remaining sectors of the table, but since the table is

incomplete, you won't be able to reformat this dump. There is no way to correct this problem. If you need this dump, you'll have to restart the whole process (with a different diskette).

F

`Faulty string`
VMPCDEB tried to enter a character string of incorrect format into memory.

`File already exists`
When you use the DUMP (or D370) command to create a memory dump, VMPCDEB checks to see if there is a file named "filename.DUM" (or "filename.370" or "filename.371"). If VMPCDEB finds one of these files, it displays the message above and terminates the command. If you still want to create a dump, use the ERASE command (Page 11-74) to erase the existing file.

`File record faulty or corrupt`
The file you're trying to load contains a record with a non-hexadecimal character or in some other way violates the requirements for loading files under VMPCDEB.

I

`Index out of range`
VMPCDEB displays this message when you enter a command to alter and interrupt, print an element, or trap an interrupt and the interrupt or element number does not exist.

`Input file record too long`
A record in a VMPCDEB read file exceeds the maximum record length of 80 bytes.

`Insufficient memory to reload this dump`

VMPCDEB displays this message when the size of the dump you're trying to reload is larger than the memory size of your system. To reload this dump, you'll have to find a bigger system.

`Insufficient room in synonym table`

You used the SYNONYM command (Page 11-136) to set up another synonym, but there's no more room in the synonym table. VMPCDEB lets you set up a maximum of 40 synonyms and you've exceeded the limit. If you need to add another synonym, you must first use the SYNONYM command to delete one of your existing synonyms.

`Invalid address`

You supplied an invalid hexadecimal address while using VMPCREF to display memory. Enter a new address to continue.

`Invalid address range`

You supplied an address range in your last command where the ending address was less than the starting address. Enter the command again, but this time make sure the ending address is greater than the starting address.

`Invalid file name/extension`

VMPCDEB displays this message if you enter a command with a filename (or a file extension) that is not in the proper DOS format. You will also see this message if you enter the D370 command (11-71) with a file extension other than "370."

`IWS CP/88 error report at 'nnnn:nnnn'`

Your program encountered an error and terminated at the address shown in the message. VMPCDEB now has control.

M

`more . .`

VMPCDEB displays this message in the bottom left corner of your screen when page mode is on and page overflow occurs. To see the next page, press the ENTER key. To terminate the command, press Ctrl-Break.

N

`NMI break-in at 'nnnn:nnnn'`

This is an informational VMPCDEB message telling you the address where your program was halted by a Non-Maskable Interrupt. For more information about NMIs, see "Non-Maskable Interrupts" on page 7-3.

`No current symbol address`

You can refer to symbolic addresses in commands using:

```
<symbol name> nnnn
```

After referring to a symbolic address in this manner, you can omit the symbol name in future commands like this:

```
<> nnnn
```

VMPCDEB will use the last symbol name you supplied. If you see the message above, you haven't supplied a symbol name in any previous commands and VMPCDEB doesn't know what symbol name to use. Enter the command again, but this time spell out the symbol name.

`No program loaded`

The command you entered will only work on a program you're testing, but you haven't loaded any programs yet. Load the program you want tested before entering this command again.

`No symbol table`

VMPCDEB does not recognize the symbolic name you used in your command because you haven't loaded a symbol table yet.

`No synonyms defined`

You used the SYNONYM command (Page 11-136) without any parameters to see a list of your synonyms. This message tells you that you haven't assigned any synonyms yet.

`Not a valid 370 memory dump file`

You used the L370 command (Page 11-99) to load a 370 memory dump, but the sizes of the files are incorrect. `Filename.370` should be 320K bytes long and `filename.371` should be 192K bytes long for a total of 512K bytes.

`Original load address = 'nnnn:nnnn'`

VMPCDEB has relocated a program for you and this message tells you the address where the program was originally loaded.

`Patch area at 'nnnn:nnnn'`

This is an informational message from VMPCDEB telling you the address of the beginning of the patch area of the program you're testing.

`Path not found`

You used the CD command (Page 11-35) or the CHDIR command (Page 11-38) to change the current directory path. VMPCDEB cannot find the directory path you named in the command, and so did not change the current directory path.

`Program halted at 'nnnn:nnnn'`

The program you're testing terminated (because of an INT 20 instruction) at the address displayed in the message above. VMPCDEB now has control.

R

`Read file closed`

The file you asked VMPCDEB to read contained another read command. When this happens, VMPCDEB closes the first read file and issues the message above.

`Record too long`

VMPCDEB translated a synonym in your command which caused the command to exceed 80 bytes in length. When this happens, VMPCDEB rejects the command.

`Resetting memory size to 'nnn'K`

This is an informational VMPCDEB message telling you the current size of PC memory. When CPIO is running, the memory size is 256K. When CPIO terminates, VMPCDEB sets the memory size back to its original value and displays the message above.

S

`Step at 'nnnn:nnnn'`

Step mode is on, and VMPCDEB stopped your program at the address shown in this message.

`Step length = 'n'`
This is an informational VMPCDEB message to tell you the current setting of step mode.

`Symbol table full`
You used the ADDSYM command (Page 11-10) to load an additional symbol table, but this caused the total number of symbols to exceed 260. VMPCDEB loaded as many symbols as it could before reaching the 260 limit. If there were more symbols in the symbol table, they were not loaded.

`Symbol table loaded`
VMPCDEB loaded a symbol table for you, either through a LOAD command or an ADDSYM command.

`Synonym not found`
You used the SYNONYM command (page 11-136) to display a synonym name that does not exist.

`Synonym value cannot be defined as itself`
You used the SYNONYM command (Page 11-136) to create a new synonym, but the string you assigned to the new synonym contained the new synonym name. VMPCDEB did not add this synonym to the table. Try the command again, but this time don't use the new synonym name in the definition string.

`The checksum for this sector failed.`
You'll see this message from VMPCREF when the dump is invalid. If you still want to create a dump, you must start all over again.

`The following interrupts are trapped:`
When you use the TRAP command (Page 11-139) without any parameters, VMPCDEB displays the message above. Then VMPCDEB displays one of

two things: the interrupt number(s) or the word "none" (if no interrupts are being trapped).

`The requested drive is unavailable, try again`
You tried to reformat the dump to a drive that doesn't exist. Enter a new drive letter to continue.

`There are break-points at:`
This is an informational VMPCDEB message telling you the addresses of all the break-points you set for your program. VMPCDEB displays both the absolute and symbolic addresses, if you've loaded a symbol table.

`There are no break-points defined`
You used the AT command (Page 11-19) without any parameters to display a list of the current break-point settings. You haven't set any break-points.

`There is a previous break-point at this address`
You used the AT command (Page 11-19) to assign a break-point to an address that you have already used for a break-point.

`There is no break-point at this address`
You used to OFF command (Page 11-104) to remove a break-point that does not exist.

`There is not enough space on this disk(ette)`
You tried to reformat the dump to a fixed disk or a diskette which does not have enough room to hold the data. To continue, try another fixed disk or diskette.

`This diskette is not from the same memory dump`
You're trying to reformat diskettes from two different dumps. If you see this message, you've processed at least one diskette correctly. Take the diskette out and insert the next diskette from the proper dump.

`This is not a VMPC Memory Dump diskette`
The diskette you're trying to reformat was not created by the CPIO DUMP routine. Take the

diskette out and insert one of the diskettes containing the dump.

`This is not the first diskette of the dump`
The diskette you're trying to reformat is not the first one in the series. Take the diskette out and insert the first dump diskette.

`This is not the next dump diskette in sequence`
You're trying to reformat the dump diskettes in the wrong order. If you see this message, you've processed at least one diskette correctly. Take the diskette out and insert another dump diskette.

`This name is not in the symbol table`
You used a symbol name in your last command that VMPCDEB did not recognize.

`Too many break-points`
You used the AT command (Page 11-19) to set a new break-point, but you've already set the maximum number of break-points. VMPCDEB only lets you set 24 break-points. If you want to set another break-point, you must first use the OFF command (Page 11-104) to turn off one or more break-points.

U

`Unrecognized file extension - will be loaded as a COM file`
When loading a file, if you don't specify a file extension, or you use a file extension other than COM or DUM, VMPCDEB automatically loads the file as a COM file.

V

`Value has more than 8 bits`

You used the ALTER command (Page 11-15) to change the contents of a memory location (or locations), but one of the replacement values you specified would not fit in one byte. If you were only altering one byte of memory, your command failed and nothing was changed. If you were altering more than one byte, VMPCDEB altered those bytes that came *before* it encountered the error. Any bytes you were trying to change *after* the error have not been altered.

`Value out of range`

You used the MEMSIZE command (Page 11-102) to change the system memory size. The value you entered was either less than 128 or greater than 640. VMPCDEB won't except your value. Enter the MEMSIZE command again, this time using a value greater than (or equal to) 128, but less than (or equal to) 640.

`VMPCDEB>`

This is the VMPCDEB prompt. It tells you that VMPCDEB is waiting for input from you. When you see this prompt, you can enter any valid VMPCDEB command.

W

`Write command abandoned`

VMPCDEB encountered a write error while executing your WRITE command, and has terminated your command.

`Writing 'nnnn' bytes`

You used the WRITE command (Page 11-146) to write the program you're testing to disk. This is an informational message telling you how many bytes VMPCDEB is writing.

`You cannot divide by zero`

You tried to divide by zero in an A or an H command. VMPCDEB will not let you do this.

`You have had hardware interrupt 'nnn'`

When there is a break-in on the program you're testing, VMPCDEB displays the interrupt numbers for those locations which have had one or more interrupts. See the TRAP command (Page 11-139) for more information about interrupts.

Index

Special Characters

A

B

C

D

E

F

G

H

I

U

V

W

Numerics

The Personal Computer
Productivity Series

Reader's Comment Form

VM/PC Service Aids 6137796

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold here

Please do not staple

Tape

Tape

The Personal Computer
Productivity Series

Reader's Comment Form

VM/PC Service Aids 6137796

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold here

Please do not staple

Tape

Tape

The Personal Computer
Productivity Series

Reader's Comment Form

VM/PC Service Aids 6137796

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

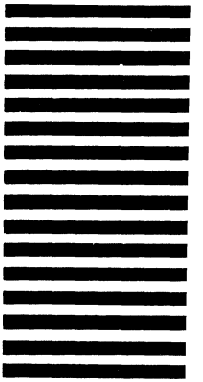
Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE



International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold here

Please do not staple

Tape

Tape

International Business
Machines Corporation
P.O. Box 1328-S
Boca Raton, Florida 33432

Printed in the
United States of America

6137796