APPENDIX C: MICROSOFT CODEVIEW

ISBN 0-13-660390-4

C.1 INTRODUCTION

Suggested Level of Programming Proficiency. I recommend that you at least read through Chapter 3 before reading this guide to CodeView. If possible, compile and debug your programs on the computer's hard drive.

Preparing Programs for Debugging. Before a program may be debugged, it must be assembled and linked with the debugging options turned on. They are **/zi** (for MASM) and **/co** (for LINK). Assuming that no errors were found, run CodeView and tell it to load TEST.EXE. You will have begun what programmers call a *debugging session*. The three commands are

```
masm/zi  test;
link/co  test;
cv  test
```

If you are linking to the irvine.lib link library (introduced in Chapter 4), change the link command to

```
link/co test,,,irvine;
```

There are two types of windows in CodeView: dialog windows (containing commands) and display windows (containing source code).

C.2 EXPRESSIONS

CodeView has a built-in expression evaluator that has unique properties for each of the following source languages: C, C++, FORTRAN, BASIC, or Pascal. By default, CodeView uses the C expression evaluator:

C-Expression Operators (in order of precedence)

```
(), [], ->, .
!,  ~,  unary -,  (type),  ++, unary *,  &,  sizeof
*,  /,  %,   :
+,  -
```

```
<,  >,  <=,  >=
==,  !=
&&
||
=  +=  -=  *=  /=  %=
BY  WO  DW
```

An *expression* may be a label, variable, symbol or constant, or some arithmetic combination of these. For example:

`var1`	The address of a variable
`percent * 100`	A symbol multiplied by 100
`es:var2`	Segment-offset memory reference

When debugging an assembly language program, the C expression evaluator changes in the following ways: Numbers are assumed to be hexadecimal unless specified otherwise, the register window is on by default, and case sensitivity is off by default.

Memory Operators. The CodeView expression evaluator does not accept assembly expressions with brackets, such as [bx] or list[si]; instead, one must use a *memory operator*. The latter returns the contents of memory, using a specific size attribute. Examples:

`BY bx`	Byte pointed to by BX (same as byte ptr [bx])
`WO si`	Word value pointed to by SI (same as word ptr [si])
`DW di+4`	Doubleword (same as dword ptr [di+4])

Format Specifiers. Use a format specifier to force an operand to be displayed in a particular format. The default format is hexadecimal. The following options are available and are case-insensitive:

d signed decimal integer (or i)
u unsigned decimal integer
x hexadecimal integer
f floating-point decimal
c single character
s string, terminated by null (zero) byte

Range. A *range* refers to a group of contiguous memory locations. It may be used to control program execution (when tracing a program, for example), or it may be used when displaying the contents of variables. Two formats are available:

```
Format 1:   startaddress  endaddress
Format 2:   startaddress L count
```

Format 1 is a pair of memory addresses that specify the beginning and ending of a block of memory. Format 2 uses *count* to specify the number of objects to be displayed. Examples:

`db var1`	Dump 128 bytes starting at a variable called **var1**.
`dw buffer L 5`	The first 5 words of an array called **buffer**.
`array array+10`	The first 16 bytes of array.
`u label1`	Disassemble instructions beginning at **label1**.

More Examples of Expressions:

`? var1,x`	Display **var1** in hexadecimal.
`? var1,i`	Display **var1** in signed decimal.
`? var1,u`	Display **var1** in unsigned decimal.
`da msg`	Display ASCII string **msg**.
`db 100`	Dump memory bytes at offset 100.
`dw var1`	Dump the contents of **var1**.
`dw es:200`	Dump memory words beginning at offset 200 in the segment pointed to by ES.
`db array L 20`	Dump 20 bytes beginning at location **array**.
`db 200 300`	Dump memory bytes, offsets 200 to 300.
`dw ds:100`	Enter new word value into memory at DS:0100.
`eb count`	Enter new hexadecimal byte value for **count**.
`ei signed`	Enter new signed decimal value for **signed**.
`r cx=20`	Set the CX register to 0020h.
`rf zr`	Set the Zero flag.
`? count=5`	Change the value of **count** to 5.

C.3 KEYBOARD COMMANDS

The cursor arrow keys, PgUp, and PgDn keys move the cursor only in the current (highlighted) window. The following special keys are active throughout your CodeView session:

Function Keys:

F1	Display help.
F2	Toggle the register window.
F3	Switch between source, mixed, and assembly modes.
F4	Switch to program output screen.
F5	Execute to next breakpoint or the end of the program.
F6	Move cursor between the dialog and display windows.
F7	Execute program up to the current line.
F8	Execute a single statement (trace into Calls).
F9	Set/clear breakpoint on the cursor line.
F10	Execute a single statement (step over Calls).

Other Keys:

^G	Make current window grow in size.
^T	Make current window smaller.
Home	Top of file or command buffer.
End	Bottom of file or command buffer.
^Break	Halt program execution (unless /D was used on the CodeView command line when the program was started).
Alt-	Pull down a menu choice beginning with the same letter. For example: **Alt-V** pulls down the VIEW menu choice.

C.4 CONTROLLING PROGRAM EXECUTION

The following general commands are used for running programs. You can single-step through instructions with **T** and **P**, and you can run the program with **G** and **E**:

T Trace single step.

P Procedure trace.

G Go – run program at full speed.

E Execute – run program in slow motion.

L Restart program at beginning.

Q Quit (return to DOS).

Setting Breakpoints

Breakpoints are locations in your program at which you can choose to pause execution. Once CodeView stops at a breakpoint, you can execute a command and resume execution when ready. Breakpoints are a powerful tool when debugging large programs, because you usually don't want to trace through all instructions one at a time. The easiest way to set a breakpoint is to move the cursor to the desired source line and press F7. To set a permanent breakpoint, move the cursor to the desired line and press F9. To remove it, press F9 again. The following breakpoint commands may be typed in the command window. You may set as many as 20 breakpoints:

BC *	Clear all breakpoints.
BC 0	Clear breakpoint 0 (the first one that was set).
BL	List all breakpoints.
BP	Set breakpoint at current line.
BP sub1	Set a breakpoint at label **sub1**.
BD 1	Disable breakpoint #1.
BE 1	Enable breakpoint #1.

All breakpoints are automatically enabled when you restart a program from the RUN menu (or use the L dialog command).

C.5 EXAMINING AND MODIFYING DATA

C.5.1 Examining and Modifying Data and Expressions

?	Display an expression or identifier.
D	Dump variable or memory contents.
U	Disassemble machine instructions.
V *.n*	View source code starting at line *n*.
A	Assemble new instructions.
E	Enter values into memory.
R	Display/modify register.

Dump Memory (D). If you follow D with a type specifier, the memory bytes will be formatted as one of the following:

B	Hexadecimal byte
A	ASCII string
I	Signed integer
U	Unsigned integer
W	Hexadecimal word
D	Doubleword

Examples.

DB list	Dump **list** as hexadecimal bytes.
DA string	Dump ASCII characters.
DI val1	Dump **val1** as a singed decimal integer.
DU val1	Dump **val1** as an unsigned decimal integer.
DW val1	Dump **val1** in hexadecimal.
DD longVal	Dump **longVal** as a 32-bit hexadecimal doubleword.

C.5.2 Watching Variables During Program Execution

The WATCH command opens a window at the top of the screen called a *watch window,* where the contents of variables may be displayed. As you trace and execute a program, the variables in the watch window are continually updated. There are four basic formats for this command:

```
W?  expression
W?  expression,format
W   range
W   type range
```

Examples.

`W? var1`	Watch **var1** in hexadecimal.
`W? percent * 100`	Watch **percent** multiplied by 100.
`W? es:var2`	Watch **var1**, located in segment pointed to by ES.

C.5.3 Direct Dialog Output to File/Printer

The **T>** and **T>>** commands allow you to echo debugging output to a file or printer. The changing contents of *watch* and *register* windows will not be written, but you can use the **T**race

and **D**ump commands to direct output from the command window at the bottom of the screen to the file or printer. Here are some examples:

`T> PRN`	Echo CodeView output on the printer.
`T> outfile`	Echo CodeView output to a file named outfile.
`T>> outfile`	Append CodeView output to a file named outfile.

When tracing a program, press **T** rather than **F8**; the latter will not output to a file.

C.6 HANDS-ON TUTORIAL

Assemble and link the following program and load it into CodeView:

```
Title CodeView Tutorial Example     (cview1.asm)
.model  small
.stack  100h
.data
byte1   db   1
byte2   db   0
word1   dw   1234h
word2   dw   0
string db   'This is a string',0

.code
   main proc
   mov   ax,@data
   mov   ds,ax

   mov   ax,0          ; AX=0000
   mov   al,byte1      ; AX=0001
   mov   byte2,al
   mov   cx,word1      ; CX=1234
   mov   word2,cx

  mov  ax,4C00h
  int  21h
main endp
end main
```

The Debugging Screen. After running CodeView, you should see the source code for the CVIEW.ASM program, the registers at the right side, and a small window at the bottom for

commands. If the source code does not appear, check the command-line options you used for compiling and linking and try again.

Change the View Options. With the program on the screen in CodeView, select VIEW/ASSEMBLY from the menu. You should be able to see the segment:offset address of each statement at the left side of the screen. This is followed by the machine code for each statement. Notice that variable names have been converted to numbers (as is done in Debug).

Choose VIEW/MIXED from the menu. Now you see alternating lines of source code and machine instructions. Because this is somewhat hard to read, choose VIEW/SOURCE to return to the starting display. This is usually adequate for program debugging, unless we need to see actual machine addresses.

Using the Mouse. Debugging is easier in CodeView if you have a mouse. If you have one, move it to the horizontal bar under the source code window and hold the left button down. Drag the bar up and down to change the size of the window. You can also use the mouse to choose menu options and to scroll windows horizontally and vertically.

Examining Data. Scroll the code window up so the program variables appear. (If you have no mouse, use the arrow keys. If nothing happens, press F6 until the cursor moves to the code window and try again.)

Now move the cursor to the command window at the bottom of the screen by pressing F6. Type the following commands to display the values of variables:

```
? byte1
? word1
? word1,i
? string
da string
```

Notice that the *i* option (the third command) displays the decimal value of **word1**. Also, the first display command for **string** displays 0x0054, the ASCII code of the string's first byte. The **DA** command correctly displays the string in ASCII format.

Trace the Program. Now we're ready to trace the program one instruction at a time. Press F8 to begin tracing the program. A highlight bar appears over each instruction about to be executed. As you trace the program, keep an eye on the register window to see the changes to AX, CX, DS, and IP. Continue until the *Program Terminated Normally* message appears.

Restart the Program. Choose RUN/RESTART from the menu so CodeView can reload the program and reset IP to the beginning. This time we will create *watch variables*, which allow us to monitor changes to variables as the program is traced.

Create a Watch Window. Choose WATCH/ADD WATCH from the menu. A dialog box appears, prompting for a *watch expression*. Type **byte2** and press Enter. Notice that a window appears at the top of the screen, showing the current value of **byte2**. Do the same for **word2**. Now trace the program by repeatedly pressing F8, and notice how the watch window shows

the current values of the two variables. (This is immensely useful for checking on programs that accidentally overwrite variables.)

Displaying String Variables. This program has a variable called **string**, whose value we would like to display. We can use either **D**ump or **W**atch:

```
da  string
w   string L 10
wa string L 10
```

The first command, **da**, displays **string** at the bottom of the screen. The second one adds **string** to the watch window as an array of hexadecimal bytes. The third command adds **string** to the watch window as an ASCII string. The **L 10** used here specifies a range of 16 (10h) bytes.

Final Note. There is much more to CodeView than we have time to show here. Microsoft includes a tutorial with sample files on the CodeView distribution disk. The assembler also includes an extensive CodeView manual.