3. Familiarity with MASM, Codeview, Addressing Modes

Part I: Background

The Microsoft Assembler package, MASM, is a programming environment that contains two major tools: the assembler/linker and the CodeView debugger. The assembler/linker translates x86 instructions to machine code and produces a ".exe" file that can be executed under DOS. The CodeView tool is an enhanced version of DEBUG with a graphical interface that also handles 32 bit instructions. A help program called 'qh' is a DOS-based utility that provides documentation on MASM and CodeView. Appendix A of this lab has some tips concerning MASM installation on your PC.

Objectives:

Learn to:

A. Use the MASM program to assemble and link a program.

B. Use CodeView to debug and execute an assembler language program.

C. Explore some of the addressing modes available in the x86 instruction set.

Pre-Lab

Read Chapter 3 and Appendix C in the Irvine Textbook. Chapter 3 gives a good introduction to the Microsoft assembler, basic arithmetic instructions (add, subtract, increment, decrement), and basic addressing modes.

1. Answer Question 41 in the Irvine Textbook.
2. Answer Question 43 in the Irvine Textbook.
3. Explain what *direct* addressing is and give an example.
4. Explain what *indirect* addressing is and give an example.

Lab

A.1 The Assembly Language Process Using the Command line

The following section explains how to assemble and link a file using the command line from a DOS window. The steps are:

1. Create or edit the source code *(.asm* file) using any ASCII text editor. Warning -- the file must be saved in an ASCII format - some editors like 'winword', or 'word' store the file by default in a binary format. To save as an ASCII format in some of the microsoft editors, select output type as *.TXT but specify the full file name as *myfile.asm* (the .asm extension should be used for assembly language files). A program called 'PFE32' (Programmer's File Editor) is installed on the PCs in the Micro I lab and is a good choice for a text editor.
2. Invoke the *masm* program to assemble the file and produce a *.obj* file and optionally, a *.lst* file.
3. Invoke the *link* program to produce a *.exe* program (or a *.com* program via a command line argument).

Assume we have an assembly language file called test.asm that has been saved in ASCII format. Open a DOS window. To assemble the file, change to the directory where the file is via the 'cd' command, and type:

```
C:\>  masm test
```

If assembly is successful, this will produce a file called *test.obj*. If errors are present, you will be given the line numbers where the syntax errors ocurred. You can also produce a listing file *(.lst*) which shows opcodes for all instructions via:

```
C:\> masm test,test, test
```

It is a good idea to **always** create a *.lst* file output.

A *.exe* file must be created from the *.obj* file via the link program. Type:

```
C:\> link test
```

You will be prompted for file names for the Run file, List file, libraries, and Definitions file. Just hitting <enter> for each choice will use the defaults. This will produce a *test.exe* file which can then be executed. You can also produce the .exe file with no prompting from the link program via:

```
C:\> link test,,,,,
```

Use 5 commas after filename (test) to provide defaults for all other choices.

Using the command line for masm/link is probably the easiest thing to do if you are only assembling/linking one source file. Most of your labs will only consist of one source file.

Section B discusses how to use a debugger called '*codeview*'. In order to view the source code of your program within the 'codeview' debugger, you need to use some command line switches with the masm and link programs in order to include debugging information. The switches are "/zi' for masm, and "/co' for link as shown below:

```
C:\> masm /zi  test,test, test

C:\> link /co  test,,,,,
```

You can get help on many topics (MASM, codeview, the x86 instruction set, etc) by typing 'qh' (Quick Help) in a command window - this brings up the help files that included with the MASM installation. You should try this at least once to see what is available.

A.2 A Sample Program

Use a text editor and type in the following program (and yes, you have to type it in instead of copying it from somewhere because you will probably make mistakes in typing it, and this will force you to deal with MASM syntax errors).

Example 4.1

```
.model small
.586
.stack 100h

.data
byte1 db 1
byte2 db 0
word1 dw 1234h
word2  dw 0
string db "This is a string", "$"
;---- this is a comment
.code
MAIN            PROC
                Mov ax, @data
                Mov ds, ax

                Mov ax,0
                Mov AL, byte1
                Mov byte2,AL
                Mov cx, word1
                Mov word2, cx
                Mov ax, 4c00h
                Int 21h
Main            endp
End main
```

Look at Example 1 (The *Hello World* program) in Irvine, Chapter 3 for an explanation of what the various assembler directives mean. There are some differences between this program and the *Hello World* program:

1. The ".586" assembler directive allows us to use Pentium instructions in this program. By default, only instructions that were included in the original 8086 can be used. Later on, this will prove to be useful so go ahead and use it.
2. This program does not use a "Title" assembler directive -- this is optional.
3. The "dw" assembler directive specifies that a word of storage be allocated (1 word = 2 bytes = 16 bits).

Assemble and link this program using MASM. Make sure that you produce a listing (.lst) file.

Lab Question 1: Open the listing (.lst) file with a text editor.

A. Give the machine code for the instruction "mov ax,0" and the instruction "mov ax, 4c00h". From these two sets of machine codes, what is the OPCODE and what is the DATA for the instruction.

B. Look at the machine code for the instruction "mov ax, @data". You will see "----" in the machine code. This is because the symbol "@data" represents the segment address of the data segment for this program. This value is NOT KNOWN until the program is loaded into memory. Why are the first two statements in this program needed? (mov ax, @data and mov dx,ax) What do they accomplish? How could our program malfunction if these were not included?

B. Debugging Programs Using Code View

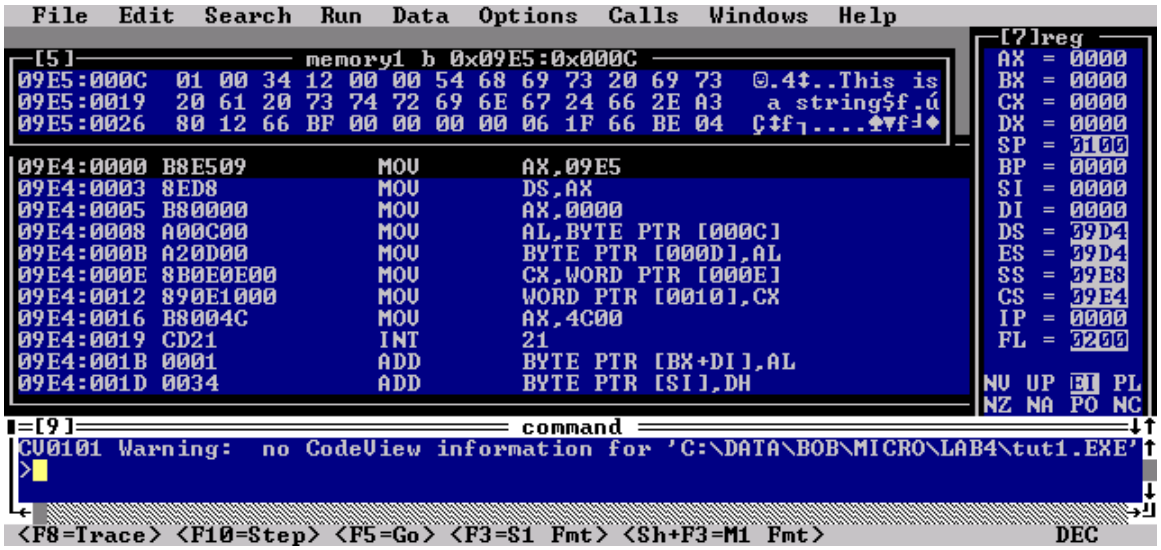
Codeview (cv.exe) is an external debugger that offers many more features than the 'debug.exe' program. You can debug programs simply by using debug.exe, but Codeview allows you easily track both memory and register changes. It is recommended that you use Codeview for debugging your programs..

Codeview (cv.exe) can be executed from the DOS command line as shown below:

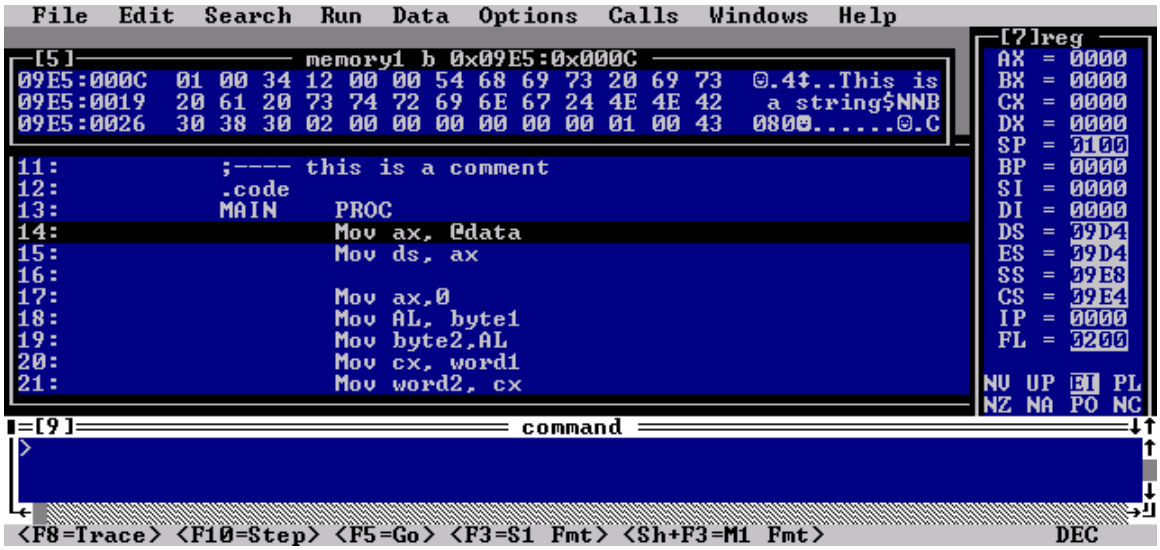
C:\> cv myfile.exe

This will bring up codeview for the file *myfile.exe*.

Run Codeview on the program that you just assembled and linked. The initial window that Codeview pops up for you may differ somewhat from the one shown below:



The Codeview window above shows a command window, code window, a memory1 window, and a register window. This is the minimum set of windows that you probably want displayed at anytime. If you do not see all of the windows above, use the WINDOWS menu choice and open the window types seen above. The window above was for an .exe file that was produced without the /zi masm option and the /co link option. The window below shows the difference when these options are used:



Notice that the 'code window' displays the source code lines but does not have the machine code displayed. You can turn on the machine code display by using the "View→Source Window' option and enable both source code and machine code views. To answer questions in this lab, you will need to be able to see the machine code of the instructions.

From the first codeview window screen, you can easily correspond between the instructions shown and the instructions that are in your source code.

For example, the displayed instruction "mov al, byte pointer [000C]" corresponds to the instruction "mov al, byte1" in your source code. This means that when the program was loaded into memory, the location "byte1" was assigned the address "DS:000C", where DS is the segment address of the data segment. As previously stated, the symbol "@data" is the segment address of the data segment, and from the first codeview window it can be seen this value is 09E5. This means that the complete logic address of the value "byte1" is "09E5:000C".

You ALWAYS want your memory window to be displaying the contents of the data segment that your program is using so that you can track changes to your data. After you open a memory window, you should use the mouse and edit the starting SEGMENT:OFFSET value in the memory window such that you can see the data used by your program. You can do this by clicking on the first address with the mouse and just entering a new YYYY:XXXX value. The memory window shown in these codeview screens has been changed to display the data used by this program.

Use the F10 key to step through the program. As you step through the program, Codeview will highlight Registers and Memory locations that are changed by the instruction that is executed.

Lab Question 2: Step through the program using F10 until you get to the "INT 21H" instruction and answer the following questions:

A. What is the logical address of the byte2 memory location?

B. What is the logical address of the word2 memory location?

C. What is the final value of the CX register?

D. What is the final value of the word2 memory locations?

E. You must include the LISTING FILE of this program in your lab report.

C. Addressing Modes

An addressing mode is how an instruction gets a value from memory. The previous example used the DIRECT addressing mode (mov cx, word1) to load/store values from memory. The following example will illustrate another addressing mode known as indirect addressing.

Create a new file that has the following program in it.

```
.model small
                .586
                .stack 100h

                .data
                byte1 db 1
                byte2 db 0
                word1 dw 1234h
                word2  dw 0
                string db "Hello", 0dh, 0ah,"$"
                ;---- this is a comment
                .code
                MAIN            PROC
                                Mov ax, @data
                                Mov ds, ax

                                Mov dx, offset string
                                Mov ah, 9
                                Int 21h
                                Mov bx, offset string
                                Mov al, [bx]
                                Mov ah, [bx+1]
                                Mov [bx], ah
                                Mov [bx+1],al
                                Mov ah,9
                                Mov dx, offset string
                                Int 21h
                                Mov ax, 4c00h
                                Int 21h
                Main            endp
                End main
```

Assemble this program and execute it outside of codeview -- note what gets printed to the screen. Now, look at the program again and answer the following questions.

Lab Question 3:

A. The instruction "move bx, offset string" moves the offset portion of the logical address for string into register 'bx'. Use codeview to determine the full logical address of 'string' (Segment:offset).

B. Comment each line of the program and explain how it contributes to the end result that you see on the screen when the program is executed. You must include the LISTING FILE of this program in your lab report.

C. The instructions that use [bx] to address memory are using an addressing mode called 'indirect addressing'. Instead of using register 'bx' for indirect addressing, try using register 'dx'. What happens and why? Give some other register that could be used instead of 'bx'.

D. A Final Task

Lab Question 4: Create a new version of the program in PART C such that the second string that is printed is "olleH" ("Hello" backwards). Do this by adding/changing the 'mov' instructions that are between the first INT 21H and the second INT 21h. Include the full LISTING file of this program in your lab report.

Lab Report

A. Describing What You Learned

Include all of the "Lab Question" answers in your lab report. Include the LISTING files (.lst) of parts A, C, D in your lab report.

B. Applying What You Learned

You MUST DEMO the program that you write in PART D for the TA before the end of lab or at the beginning of the next lab period.

Appendix A : MASM Installation tips

The CDROM in the back cover of the Irvine text has the MASM program distribution contained in the 'masm' subdirectory. Execute the 'setup.exe' program in the *masm* subdirectory and a DOS command window will appear to guide you through the installation process via a series of questions. The default answers are ok for everything - however, when it asks you what operating system to use choose "DOS/Windows & NT" (if you don't choose this option, the Codeview program gets left out for some obscure reason). The default installation directory is C:\MASM611 (or c:\MASM613 if you have this version) and the rest of these notes assume you use this directory. The MASM program works ok with all varieties of Windows (95/98/NT/2000/XP).

For correct operation after MASM is installed, certain environment variables must be modified. The BAT file "c:\masm611\binr\new-vars.bat' will do these modifications if executed after you start a DOS command window. One easy way method of getting the correct environment settings is to create a shortcut to a DOS command prompt and modify the shortcut properties.

1. Under Win95/98, you can specify a starting BAT file on the short cut -- you should specify the *new-vars.bat* file above (right click on the shortcut and open the Properties window, specify the full pathname). When you open a DOS command window via this shortcut, the *new-vars.bat* file will be executed and all of your environment variables will be set correctly.
2. Under WinNT/2000/XP, you will need to modify the 'target' line and add a command switch to specify the starting BAT file. After the 'cmd.exe' part of the target line, add " /k c:\masm611\binr\new-vars.bat" to execute *new-vars.bat* on startup.

To see if your environment variables are correct, open your shortcut and type 'masm' on the command line. If the program is found, then the modifications have been made correctly. You can also try 'qh' which execute the MASM/Codeview help program - if 'qh' can locate the help files then your installation has been successful.

WINDOWS 2000 MOUSE PROBLEM

Under Windows 2000, the default options for a command window disables DOS mouse operation within a command window (ie., if you run CodeView, mouse operation of the menus may not work). If you have this problem, right click on the command window title bar and open the Properties window. Turn off the QuickEdit mode by right clicking on this check box (the check should disappear). After this, the mouse should operate correctly in the command window.

MASM Installation From Your Browser

Newer versions of the Irvine book allow installation from your browser:

1. Place the CD from the Irvine text in your CDROM drive. In your browser (such as Internet Explorer), use the url address of "**D:**" (or whatever drive letter your CDROM is at).

2. Click on the web page "index.html"

3. Click on the Link "**Install Microsft MASM 6.13**" to install the assembler onto your PC. You can use all of the default directories. Use the choice of running the program from its location to install from the CD drive.

4. Click on the link "**Install the Sample Programs and Irvine Link Libraries**" to install the examples from the Irvine book.

5. Click on "**View Tips on Setting Up Your System Environment Variables**" to find out what paths need to be added to your autoexec.bat file.

6. Click on "**View Tips on Running The Microsoft Assembler**" to show how to assemble and link programs using the ML command.

7. Change your autoexec.bat file by adding the commands listed in the "View Tips on setting up Your system environment Variables" page using an text editor without formatting. For Win 2000

users, you will need to access your system environment variables from the Control Panel → System entry. Or, instead of modifying your environment variables you can execute the 'new-vars.bat' file each time you start a command window (see the previous section for how to do this automatically when the command window is opened).

You should now be able to run MASM and Codeview from your PC.

Note: You can also use the "RUN" option from the START button to install each of these programs.