Lab 1: Assembly Language Tools and Data Representation

Contents

1.1 Introduction to Assembly Language Tools

Software tools are used for editing, assembling, linking, and debugging assembly language programming. You will need an assembler, a linker, a debugger, and an editor. These tools are briefly explained below.

1.1.1 Assembler

An **assembler** is a program that converts **source-code** programs written in **assembly language** into **object files** in machine language. Popular assemblers have emerged over the years for the Intel family of processors. These include MASM (Macro Assembler from Microsoft), TASM (Turbo Assembler from Borland), NASM (Netwide Assembler for both Windows and Linux), and GNU assembler distributed by the free software foundation. We will use MASM 6.15.

1.1.2 Linker

A **linker** is a program that combines your program's **object file** created by the assembler with other object files and **link libraries**, and produces a single **executable program**. You need a linker utility to produce executable files. Two linkers: **LINK.EXE** and **LINK32.EXE** are provided with the MASM 6.15 distribution to link **16-bit real-address mode** and **32-bit protected-address mode** programs respectively.

We will also use a link library for basic input-output. Two versions of the link library exist that were originally developed by Kip Irvine. The 32-bit version is called **Irvine32.lib** and works in Win32 console mode under MS-Windows, while the 16-bit version is called **Irvine16.lib** and works under MS-DOS.

1.1.3 Debugger

A **debugger** is a program that allows you to trace the execution of a program and examine the content of registers and memory.

For 16-bit programs, MASM supplies a 16-bit debugger named CodeView. CodeView can be used to debug only 16-bit programs and is already provided with the MASM 6.15 distribution.

For 32-bit protected-mode programs, you need a 32-bit debugger. The latest version of the 32-bit Windows debugger is available for download for free from Microsoft.

1.1.4 Editor

You need a text editor to create assembly language source files. You can use NotePad , or any other editor that produces plain ASCII text files. You can also use the ConTEXT editor, which is distributed as a freeware at http://www.context.cx. ConTEXT is a powerful editor

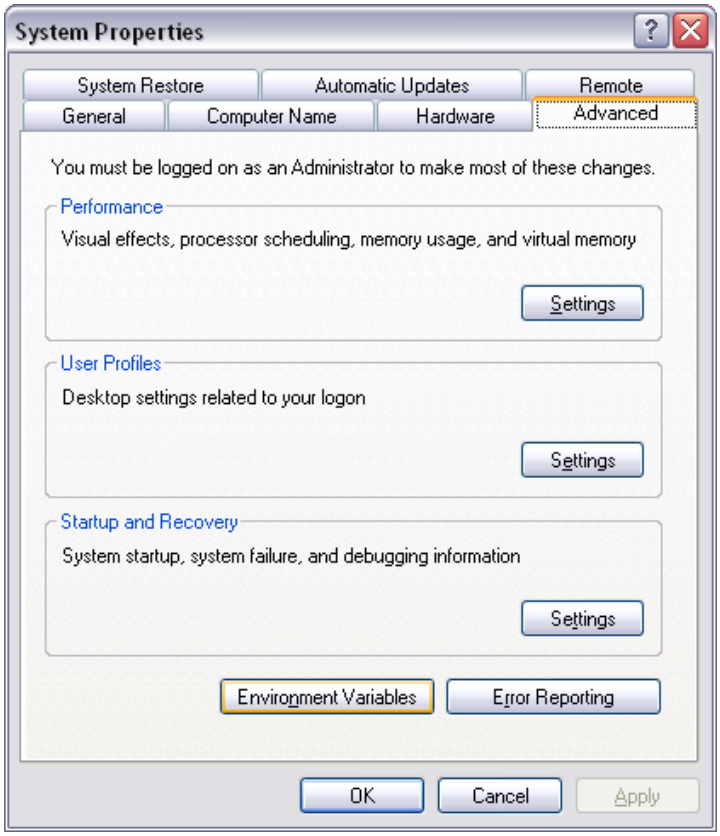
that can be easily customized and can be used as a programming environment to program in assembly language. It has built-in syntax highlighting feature.

1.2 Lab Work: Installing MASM 6.15

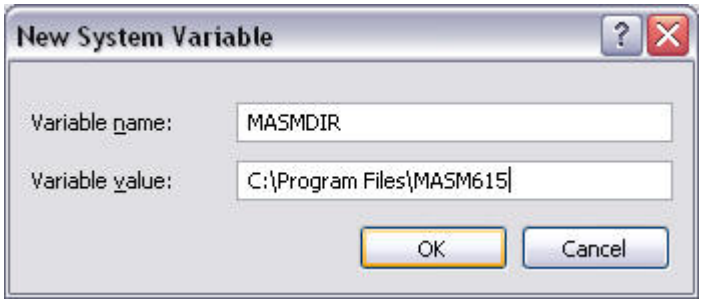
Step 1: Download MASM615.exe, a self-extract executable file, from http://www.ccse.kfupm.edu.sa/~mudawar/coe205/lab/index.htm .

Step 2: Double click on **MASM615.exe** to extract the files. Specify the installation directory. We recommend using **C:\Program Files\MASM615** as the destination directory, but any other directory will do.

Step 3: Define an environment variable **MASMDIR** for the installation directory. Under **Control Panel**, double-click on **System** to obtain the **System Properties** dialog box. Under **System Properties**, click on the **Advanced** tab. Click on the **Environment Variables** button.



Under **Environment Variables**, Click on the **New** button to add a **New System Variable**. Add **MASMDIR** as the variable name and the **C:\Program Files\MASM615** as the variable value and press OK. The **MASMDIR** variable and its value should now appear under System variables. If a different installation directory is chosen for MASM 6.15 then specify it here.



Step 4: Edit the **Path** system variable by inserting **%MASMDIR%;** (don't forget the semicolon) at the beginning of the variable value.

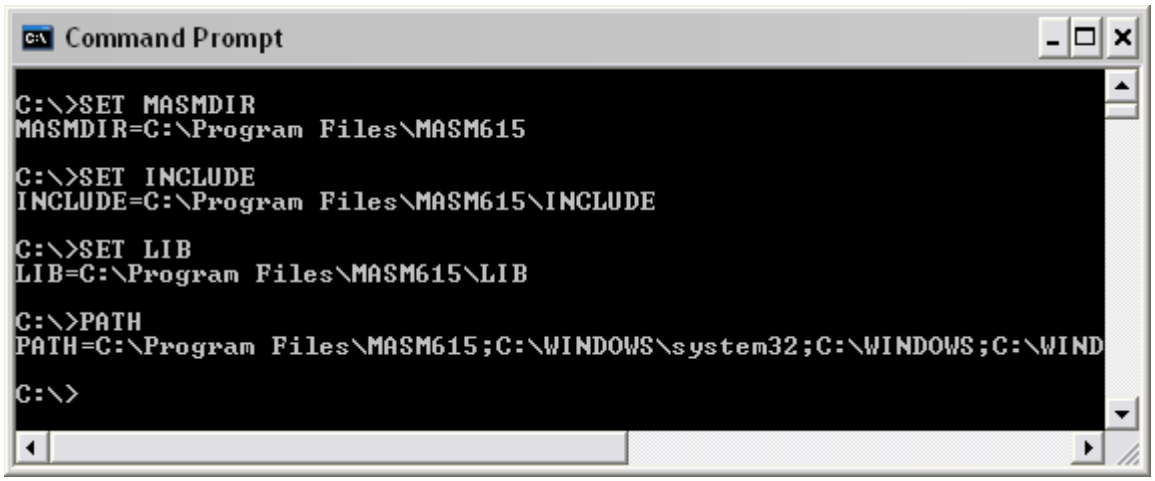
Step 5: Define a new system variable called **INCLUDE** with value **%MASMDIR%\INCLUDE** as show below and press OK. This variable specifies the directory that contains the include (**.inc**) files.

Step 6: Define a new system variable called **LIB** with value **%MASMDIR%\LIB** as show below and press OK. This variable specifies the directory that contains the link library (**.lib**) files.

Step 7: Check the environment variables. Open a Command Prompt and type:

- **SET MASMDIR**
- **SET INCLUDE**
- **SET LIB**
- **PATH**

These commands should display the **MASMDIR**, **INCLUDE**, **LIB**, and **PATH** environment variables as shown below. If the installation steps are done properly, you can start using the MASM commands.



1.3 Displaying a Welcome Statement

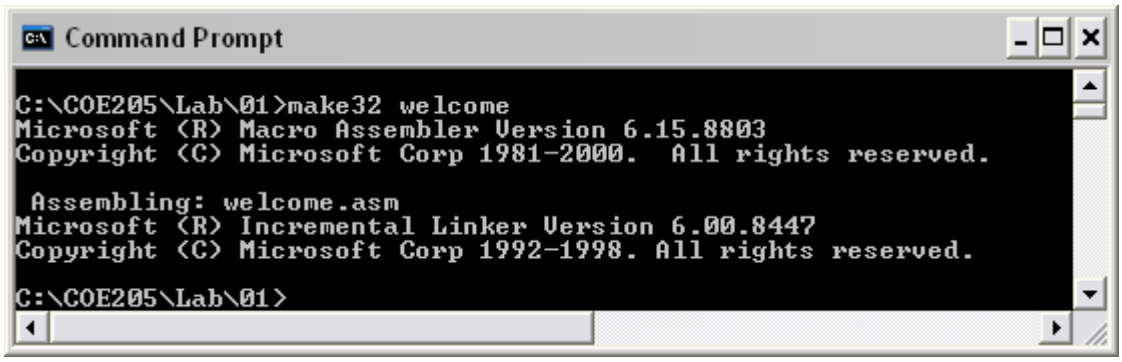
The first assembly-language program that you will assemble, link, and run is *welcome.asm*. This program displays a welcome statement on the screen and terminates. You can open this program using any text editor. We will not go over the details of this program in this first lab. You will understand these details in future labs.

```
TITLE Displaying a Welcoming Message  (welcome.asm)

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data

CR  EQU     0Dh         ; carriage return
LF  EQU     0Ah         ; line feed

welcome BYTE    "Welcome to COE 205",CR,LF
        BYTE    "Computer Organization and Assembly Language",CR,LF
        BYTE    "Enjoy this course and its lab",CR,LF,0

.code
main PROC
; Clear the screen
    call Clrscr         ; Call procedure Clrscr

; Write a null-terminated string to standard output
    lea  edx, welcome   ; load effective address of welcome into edx
    call WriteString    ; write string whose address is in edx
    exit
main ENDP
END main
```

1.3.1 Lab Work: Assembling and Linking a Program

Open a Command Prompt and type the following command. This command will assemble and link the *welcome.asm* program.

make32 welcome



1.3.2 Lab Work: Running a Program

The **make32** command will generate is the *welcome.exe* executable file. You can now run the *welcome.exe* program by simply typing **welcome** at the command prompt. Watch the output of this program and write it down in the following box:

Console Output

1.4 Lab Work: Installing the 32-bit Windows Debugger

The latest version of the 32-bit Windows debugger is available for download from Microsoft at http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx. Alternatively, you can download version 6.5.3.7 (released on June 2005) from http://www.ccse.kfupm.edu.sa/~mudawar/coe205/lab/index.htm.

Step 1: Download the 32-bit debugger installer.

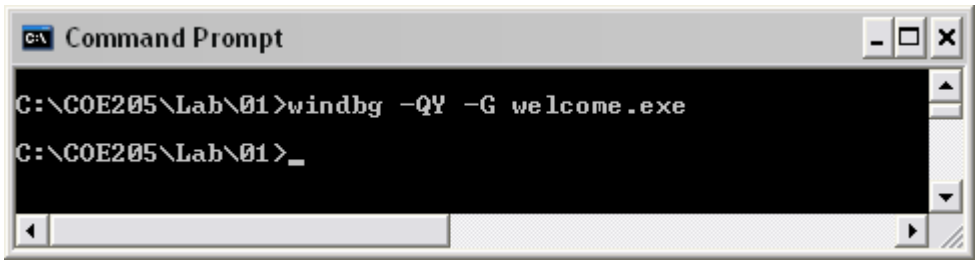
Step 2: Double click on the installer executable file and follow the on-screen instructions.

Step 3: Edit the **Path** system variable by appending the installation directory of the windows debugger **C:\Program Files\Debugging Tools for Windows** at the end of the **Path** value. Don't forget to use the semicolon as a separator between various directories in the **Path** system variable.

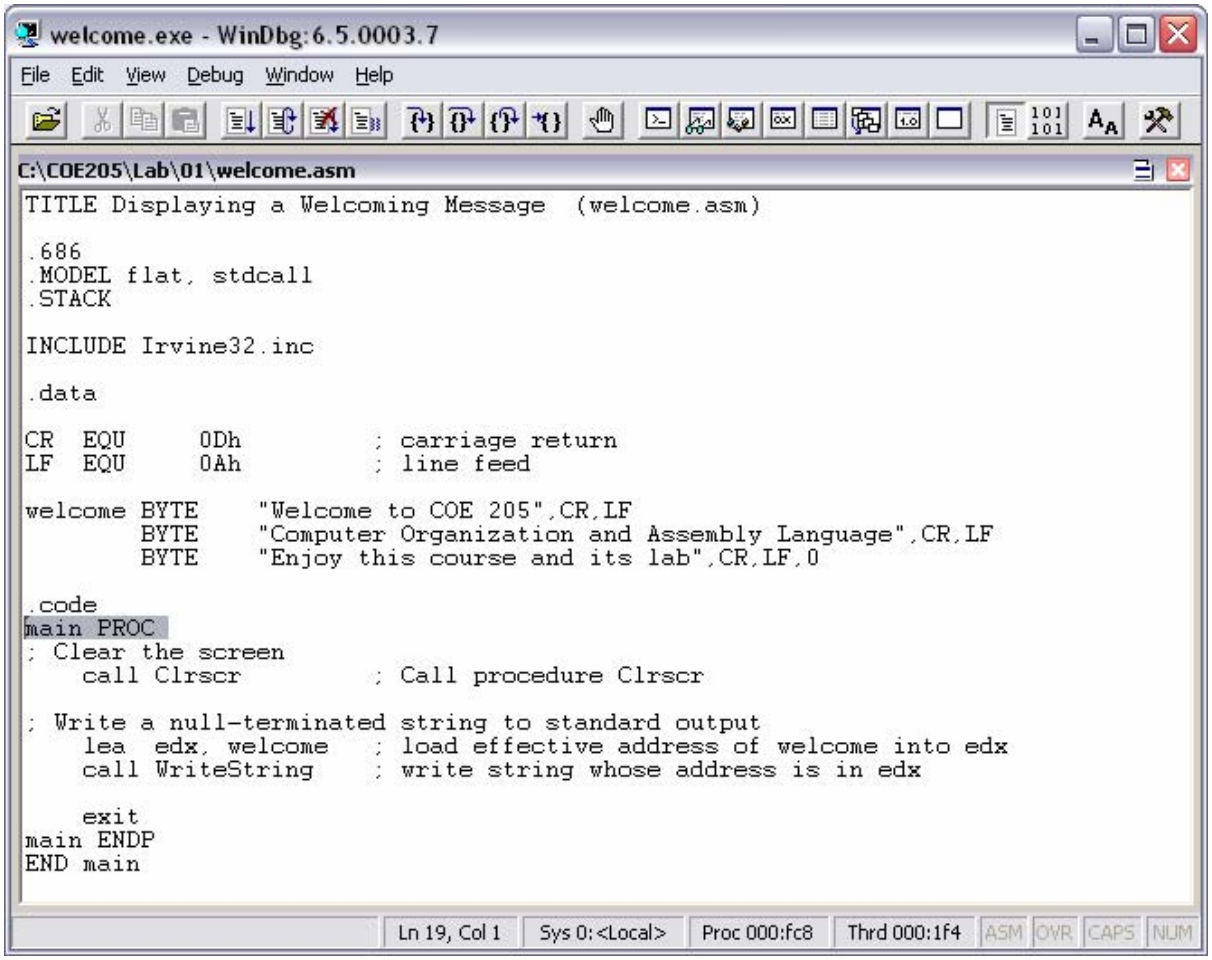
Step 4: Open a Command Prompt and type: **path**. This command should display the value of the path variable. Make sure to have the installation directory of the debugger **C:\Program Files\Debugging Tools for Windows** as part of the **PATH** variable.

1.5 Lab Work: Using the 32-bit Windows Debugger

We will use the Windows debugger extensively throughout this semester to trace and debug programs. At the Command Prompt, type: **windbg –QY –G welcome.exe** to run the Windows Debugger.



Open the source file *welcome.asm* from the **File** menu (or click on the button). Place the cursor at the beginning of the main procedure and press **F7** (or click the button) to start the execution of the main procedure. Press **F10** (or click the button) to step through the execution of the main procedure. Observe the console output as you step through the execution.



1.6 Practice: Data Representation

Before going into the details of assembly language programming, it is important that you master skills and develop fluency about data representation. Computer data can be represented in a variety of ways. We will examine the content of memory and registers at the machine level. We will use the binary, decimal, and hexadecimal number systems. Here are some practice exercises:

1.6.1 Write each of the following Decimal Numbers in Binary:

a) 2 =
b) 7 =
c) 9 =
d) 13 =
e) 27 =
f) 62 =

1.6.2 Write each of the following Binary Numbers in Decimal:

a) 00001010 =
b) 00001111 =
c) 00101000 =
d) 01010111 =
e) 10000000 =
f) 11000111 =

1.6.3 Write each of the following Binary Numbers in Hexadecimal:

a) 00001010 =
b) 00001111 =
c) 00101000 =
d) 01010111 =
e) 10000000 =
f) 11000111 =

1.6.4 Write each of the following Hexadecimal Numbers in Binary:

a) 0B =
b) 4C =
c) AF =
d) 3D15 =
e) 6E70 =
f) 8A9B =

1.6.5 Write each of the following Hexadecimal Numbers in Decimal:

a) 0B =
b) 4C =
c) AF =
d) 3D15 =
e) 6E70 =
f) 8A9B =

1.6.6 Signed Integers: 2's Complement Notation

In mathematics, the negative of a number *n* is the value when added to *n* produces zero. For example: 3 + (-3) = 0. Programs often include both subtraction and addition operations. However, the CPU only performs addition internally. When subtracting A – B, the CPU performs A + (-B). For example, to subtract 6 – 4, the CPU does 6 + (-4).

When working with binary numbers, how does the CPU compute the negative of a number? The answer is that the CPU computes the 2's complement. The 2's complement is the negative of a number. For example, consider the following 8-bit binary number: **00010110**, which is equal to **22** in decimal. The 2's complement is obtained by reversing each bit of a binary number (called the 1's complement) and then adding 1. For example,

2's complement of 00010110 = 11101001 (1's complement) + 1 = 11101010

```
  00010110      Since 00010110 in binary = 22 in decimal, then
+ 11101010      11101010 in binary = -22 in decimal
----------
 (1)00000000    The carry is 1, but it is ignored, since we are representing the number in 8 bits
```

1.6.7 Write each of the following Integers in 8-bit 2's Complement Notation:

a) -1 =

b) -17 =

c) -19 =

d) -62 =

e) +127 =

f) -128 =

1.6.8 Write each of the following 8-bit Signed Binary Integers in Decimal:

a) 01011100 =

b) 11011100 =

c) 10001111 =

d) 01111110 =

e) 10010001 =

f) 10000000 =

1.6.9 Indicate the sign for each of the following 16-bit signed hex integers:

a) 7FB9

b) D000

d) 8123

d) 6FFF

1.6.10 Write each of the following signed integers as 16-bit hexadecimal value:

a) -1 =

b) -127 =

c) -256 =

d) -8193 =

1.6.11 Largest and Smallest

a) What is the largest positive 8-bit value in binary, hexadecimal, and decimal?

b) What is the smallest negative 8-bit value in binary, hexadecimal, and decimal?

c) What is the largest positive 16-bit value in binary, hexadecimal, and decimal?

d) What is the smallest negative 16-bit value in binary, hexadecimal, and decimal?

Review Questions

1. Name four software tools used for assembly language programming:

2. What is an assembler?

3. What is a linker?

4. What is a debugger?

Programming Exercises

1. Modify the *welcome.asm* program to display a message of your choice.

