# Experiment N$^{o}$ 1

## Introduction to Assembly Language Programming

### Introduction:

The aim of this experiment is to introduce the student to assembly language programming, and the use of the tools that he will need throughout the lab experiments. This first experiment let the student use the DOS Debugger and the Microsoft Macro Assembler (MASM). MASM related tools are introduced; these include the Programmer's WorkBench (PWB) and CodeView (CV). Such tools are interactive means for writing linking, and debugging assembly language programs.

### Objectives:

1- Use of the Dos **Debugger** program
2- Introduction to the Microsoft Macro Assembler (**MASM**)
3- Use of the PWB, and Code View (CV).
4- General structure of an assembly language program
5- Introducing Data representation in assembly

### Overview:

In general, programming of a microprocessor usually takes several iterations before the right sequence of machine code instructions is written. The process, however, is facilitated using a special program called an "Assembler". The Assembler allows the user to write alphanumeric instructions, or mnemonics, called Assembly Language instructions. The Assembler, in turn, generates the desired machine instructions from the Assembly Language instructions.

Assembly Language programming consists of the following steps:

| | STEP | PRODUCES |
|---|---|---|
| 1 | Editing | Source File |
| 2 | Assembling | Object File |
| 3 | Linking | Executable File |
| 4 | Executing | Results |

Table 1.1: Assembly Language Programming Phases

**Assembling the program:**

The assembler is used to convert the assembly language instructions to machine code. It is used immediately after writing the Assembly Language program. The assembler starts by checking the syntax, or validity of the structure, of each instruction in the source file. If any errors are found, the assembler displays a report on these errors along with a brief explanation of their nature. However, if the program does not contain any errors, the assembler produces an object file that has the same name as the original file but with the "obj" extension.

**Linking the program**:

The linker is used to convert the object file to an executable file. The executable file is the final set of machine code instructions that can directly be executed by the microprocessor. It is different than the object file in the sense that it is self-contained and re-locatable. An object file may represent one segment of a long program. This segment can not operate by itself, and must be integrated with other object files representing the rest of the program, in order to produce the final self-contained executable file.

In addition to the executable file, the linker can also generate a special file called the "map" file. This file contains information about the start, end, and the length of the stack, code, and data segments. It also lists the entry point of the program.

**Executing the program**

The executable file contains the machine language code. It can be loaded in the RAM and be executed by the microprocessor simply by typing, from the DOS prompt, the name of the file followed by the Carriage Return Key (Enter Key). If the program produces an output on the screen, or a sequence of control signals to control a piece of hardware, the effect should be noticed almost immediately. However, if the program manipulates data in memory, nothing would seem to have happened as a result of executing the program.

**Debugging the program**

The Debugger can also be used to find logical errors in the program. Even if a program does not contain syntax errors it may not produce the desired results after execution. Logical errors may be found by tracing the action of the program. Once found, the source file should be reedited to fix the problem, then re-assembled and re-linked. A special program called the "Debugger" is designed for that purpose.

The debugger allows the user to *trace* the action of the program, by *single stepping* through the program or executing the program up to a desired point, called *break point*. It also allows the user to inspect or change the contents of the microprocessor internal registers or the contents of any memory location.

**The DOS-Debugger:**

The DOS "Debug" program is an example of a simple debugger that comes with MS-DOS. Hence, it is available on any PC. It was initially designed to give the user the capability to trace logical errors in executable files. It allows the user to take an existing executable file and unassemble it, i.e. convert it to assembly language. Also, it allows the user to write assembly language instructions directly, and then convert them to machine language. The program is simple and easy to use, but offers limited capabilities, which make it unsuitable for serious Assembly language programming.

Below, are summarized the basic DOS-Debugger commands.

| COMMAND | SYNTAX |
|---|---|
| Assemble | A [address] |
| Compare | C range address |
| Dump | D [range] |
| Enter | E address [list] |
| Fill | F range list |
| Go | G [=address] [addresses] |
| Hex | H value1 value2 |
| Input | I port |
| Load | L [address] [drive] [first sector] [number] |
| Move | M range address |
| Name | N [pathname] [argument list] |
| Output | O port byte |
| Proceed | P [=address] [number] |
| Quit | Q |
| Register | R [register] |
| Search | S range list |
| Trace | T [=address] [value] |
| Unassemble | U [range] |
| Write | W [address] [drive] [first sector] [number] |

**Table 1.2**: Common DOS-Debug commands

**MS-MASM**:

Microsoft's Macro Assembler (MASM) is an integrated software package written by Microsoft Corporation for professional software developers. It consists of an editor, an assembler, a linker and a debugger (CodeView). The programmer's workbench combines these four parts into a user-friendly programming environment with built in on line help.

The following are the steps used if you are to run MASM from DOS.

| | COMMAND | FILE NAME |
|---|---|---|
| **1** | **Edit**, any editor will do | Name.asm |
| **2** | **Masm** Filename | Name.obj |
| **3** | **Link** Filename | Name.exe |
| **4** | Filename | |
| | Note: Steps 2 and 3 may be done in one single command: **ML filename.asm** | Name.asm and Name.obj |

Table 1.3: Assembly Language Programming Phases

**MS-PWB:**

The PWB allows the user to define a project that may contain one or more files. Then, the user may select and save all the necessary assembling, linking, and debugging options for that project. Once these options are set, the user need not set them again for that project. The PWB allows the user to edit, assemble, run, or debug his program without leaving the PWB environment. It also allows the user to get help on any keyword by pointing to the keyword and pressing the F1 key.

**Notes on the use of MS-MASM:**

MASM may be run under DOS environment, or through PWB.

**Running MASM and CodeView Debugger from the MSDOS prompt:**

If you don't like the integrated PWB, you might run just the necessary programs from the MSDOS prompt. Here are the steps, assuming that your program is called "proj":

1. Open a DOS window.
2. Set the PATH so that the MASM programs are available
   The MASM programs are on the E drive; set the path so that DOS can find them. This only needs to be done once each time you open a MSDOS prompt.
   ```
   set PATH=%PATH%;E:\masm611\bin; E:\masm611\binr
   ```
3. Use a Text Editor to Edit the .ASM File
   Create your file using one of the following programs:
   ```
   notepad proj.asm
   wordpad proj.asm
   edit proj.asm
   ```
   Make sure it has a .ASM ending. Also, be sure you are doing your work on the A: drive, or on the D:\WORKAREA, and not in the `C:\WINDOWS` directory, or other drives.
4. Run the Compiler/Linker to generate an .EXE file from the .ASM file
   ```
   ml /Fl /Zi proj.asm
   ```

Make sure you use the two switches: `/Fl` creates a listing (.LST) file (the letter to the right of the F is a lower-case L); `/Zi` makes the finished product compatible with the CodeView debugger.
You'll be notified of any errors with your program, and error statements should be placed in your .LST file (`proj.lst` in this case).

5. Run the Program
   Your final program (if there were no errors in the previous step) will have a .EXE ending. To just run it, type:
   ```
   proj
   ```
   If you want to use the CodeView debugger to examine the instructions, registers, etc.:
   ```
   cv proj
   ```
   This brings up the regular full-screen version of CodeView.

**Running MASM and CodeView** PWB**:**

The basic steps for creating, building, and debugging an assembly language program are given below. The examples given in this first session are very simple; their main purpose is to show how o uses the PWB environment.
A project is a complete set of files needed to define and build an assembly language program. The source language files for your programs are, of course, part of this project, but also there are listing files, and symbol table files for debugging, and files giving directions for building your program.
It is best to create a separate folder to contain the project files for each programming assignment. You should copy this folder on a floppy disk for later use.

1. Create a folder for your new project. For the first assignment, the folder name could be lab1.
2. Copy any files that are needed from your floppy disk into your folder.
3. Run PWB.EXE by double clicking on the file or its shortcut in the C:\MASM611\BIN folder. The shortcut may be copied and the copy moved to the desktop to make it easier to find the PWB.
4. Once the PWB is running (and the DOS window appears), you may create a new project using the project menu. Type the correct path to the new project folder with project name and press the **set project template** button. Select Runtime support: **None**, Project Templates: **DOS:EXE**, and press **OK.**
5. You don't have any program files to add to the project yet, so click **save list** when the add file window appears.
6. Under the Options menu, select **build options**. Select **Use Debug Options.** Select **OK**.
7. Using the **Options menu:language options:MASM,** select **Set Debug Options.** Check the boxes that will create a listing file with source, machine language, and symbol table: **Generate Listing File, List Generated Instructions, Include all Source Lines, CodeView Debugger. Press OK.**
8. Using the Options **menu:link** options, select **CodeView debugger, debug options**. Press the additional debug options "button" and select **Full** map output. This will give you a link map that tells where each program component will be loaded. Select **OK.**

9. Select **new** under the file menu to get an editor window for your program. Enter the program. Save your program naming it with a .ASM suffix, e.g. : P11.ASM
10. Select **edit project** from the project menu, and add your new assembly program into the project list. **Save the list**.
11. Start the debugger, running your program, by selecting **debug** from the Run menu. It will ask you to build your project. That is a good idea, select **Rebuild All**. The Assembler will assemble your code, the linker will be called, and if there are no errors, the debugger window will appear. You can then start debugging.
12. If there were any errors, select **View Results**, then click **next error** under the project menu. It will show you the line containing the error in your source program, and you can correct it at this point.
13. Continue asking to see the next error until you run out of errors.
14. Select **debug** from the project window to have another try.
15. Repeat this procedure until you finally get the debugger.
16. Now you can run your program. You may step through your program one line at a time by using **F8** and **F10**, or you may select go (**F5**) and run it without interruption.
17. In order to watch the contents of memory change, you need to select the memory window options in the options menu (of the debugger). Check the box which requests that the memory window be continually re-evaluated. If you don't check this box, the memory window will continue to reflect memory contents just before your program started running.
18. Breakpoints may be inserted at any line of code using the **set breakpoint** item in the Data menu, or press **F9**. If **go** (or **F5**) is selected, the program will stop on any line encountered with a breakpoint that is set.
19. Selecting source window options from the window menu allows you to have mixed source lines and object code. This will allow you to compare the instructions that are actually running with the lines of source code that you provided.
20. **F10** will single step through your code, but will not single step through any called subroutines (It will "step-over" the subroutine call.) The subroutine will be called correctly; it will simply not be debugged. This is handy when you are using a subroutine that has already been debugged, or when a system routine is called.
21. **F8** will single "step-into" a subroutine that is called, allowing you to debug the subroutine itself.
22. You may execute down to a given instruction by right clicking on the source line, moving the cursor to the source line, and then pressing the **F7** key.
23. Taking the time to add comments to your code while first entering it will save much time in the long run. It is very hard to figure out what an undocumented assembly language program is doing after even a brief intermission. Comments help you to quickly locate desired sections of code.
24. **Note:** On the machines at the lab, it is frequently necessary to make sure the path is set correctly for the PWB to run. Right mouse-click on the PWB shortcut icon. Select **Properties**, and then select the **Program** tab. Under **Batch file**, type **C:\MASM611\BIN\NEW-VARS**. In the lab, only the network administrator can do this procedure.

**Pre Lab Work** :

In this experiment you will practice editing, assembling, and linking an Assembly language program. First you will type a short program, using DOS Edit, then assemble and link the program using the MASM Assembler and Linker. Second you will practice the DOS-Debugger, and compare it to MASM's CodeView. Finally, you will use MS-PWB to develop a project around your short program and practice some of the advantages of PWB.

1- Study the material given in part I of this manual.
2- Review the material related to data representation.
3- Write the attached programs and bring them on a floppy disk to the lab. Use the DOS editor or the Windows notepad. If you use a word processor, make sure that you chose the option *Save As Text* while saving

**Note:**

The Exit function:
The following instructions terminate the program and exit to DOS.

```
MOV AH, 4CH
INT 21H
```

**Lab Work:**

1- Assemble, Link and Run program 1.
2- Use the Debugger to run your program.
3- Notice the values given by the assembler to the numbers you used in your program. Draw a table and write each value with its corresponding representation. What do you conclude?
4- Repeat the same procedure using PWB and CodeView. What do you conclude?
6- Dress a table and write the values assigned by the assembler to the values you wrote in the editor. Explain that.
5- Assemble, Link and Run program 2.
6- What value do you find in DX register?
7- Repeat the same procedure using PWB and CodeView.
8- Change the line:

```
        MULT1 EQU 25
```

   By:

```
        MULT1 EQU 25H,
```

   and run the program again. What do you notice?
9- Can you explain what the program does?

The first two lines, starting with a semi colon (;) are just comments, they may be omitted, they are however very useful.

; COE 205: Lab Exp. # 1 Program # 1
; Student Name: Student ID: Section:

```
TITLE "A simple program"
.MODEL SMALL
.STACK 32
.CODE
       MOV AX, 2000
       MOV BX, 2000H
       MOV CX, 10100110B
       MOV DX, -4567
       MOV AL, 'A'
       MOV AH, 'a'

       MOV AX, 4C00H
       INT 21H
END
```

---

; COE 205: Lab Exp. # 1 Program # 2
; Student Name: Student ID: Section:

```
TITLE "Our second program"
.MODEL SMALL
.STACK 32
.DATA
      MULT1     EQU  25
      MULT2     DW    5
.CODE
      MOV AX,@DATA
      MOV DS,AX

      MOV AX,00
      MOV BX,MULT1
      MOV CX,MULT2
MULT: ADD AX,BX
      DEC CX
      JNZ MULT
      MOV DX,AX

      MOV AX,4C00H
      INT 21H
END
```