LAB 02

Computer Organization & Assembly Language (COAL) | EL 2003

Student Name: _____

Roll No.: _____

Section: _____

Marks Awarded _____

Prepared by: Talha Shahid

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Lab Session 02:

ASSEMBLY LANGUAGE FUNDAMENTALS

Objectives:

- Debugging of Programs
- Basic Elements of Assembly Language

Steps Involved in Creating and Running a Program

Assembler

It converts the assembly language to machine language (Object Code) may contain unresolved references (i.e., file contains some or all of complete program).

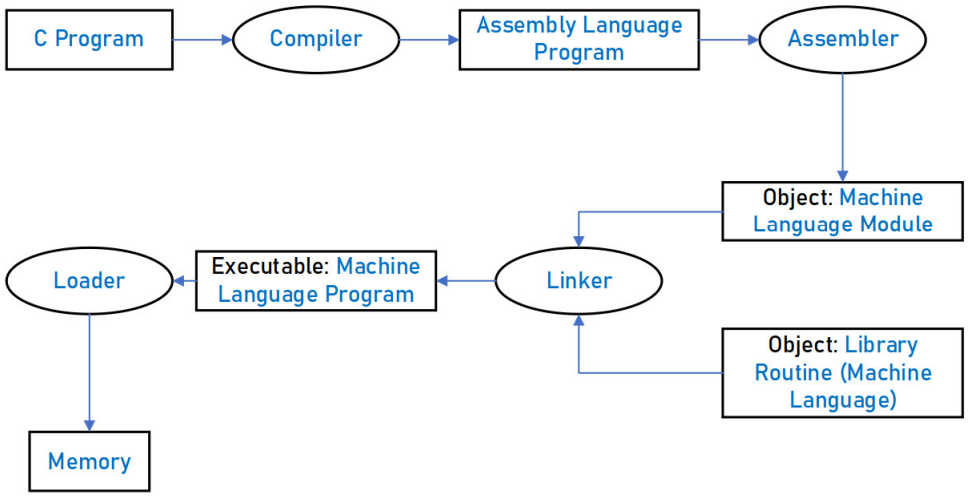
Linker

A program that combines object files to create a single "executable" file. Major functional difference is that all references are resolved. (i.e., Program contains all parts needed to run) A program that loads executable files into memory, and may initialize some registers (e.g., IP) and starts it going.

Debugger:

A program that loads but controls the execution of the program. To start/stop execution, to view and modify state variables.

Steps in Creating & Running Code



Section 01 | Debugging of Programs

We have seen how to configure Visual Studio 2022 Community Edition for Assembly Language and tested it with a sample program. The output of our sample program was displayed using a console window but it is usually more desirable to watch the step-by-step execution of our program with each line of code using breakpoints.

Let us briefly define the keywords relevant to debugging in Visual Studio and then we will cover an example for understanding.

Debugger

The (Visual Studio) debugger helps us observe the run-time behavior of our program and find problems. With the debugger, we can break execution of our program to examine our code, examine and edit variables, view registers, see the instructions created from our source code, and view the memory space used by our application.

Breakpoint

A breakpoint is a signal that tells the debugger to temporarily suspend execution of your program at a certain point. When execution is suspended at a breakpoint, your program is said to be in break mode.

Code Stepping

One of the most common debugging procedures is stepping: executing code one line at a time. The Debug menu provides three commands for stepping through code:

- Step Into (By pressing F11)
- Step Over (By pressing F10)
- Step Out (Shift+F11)

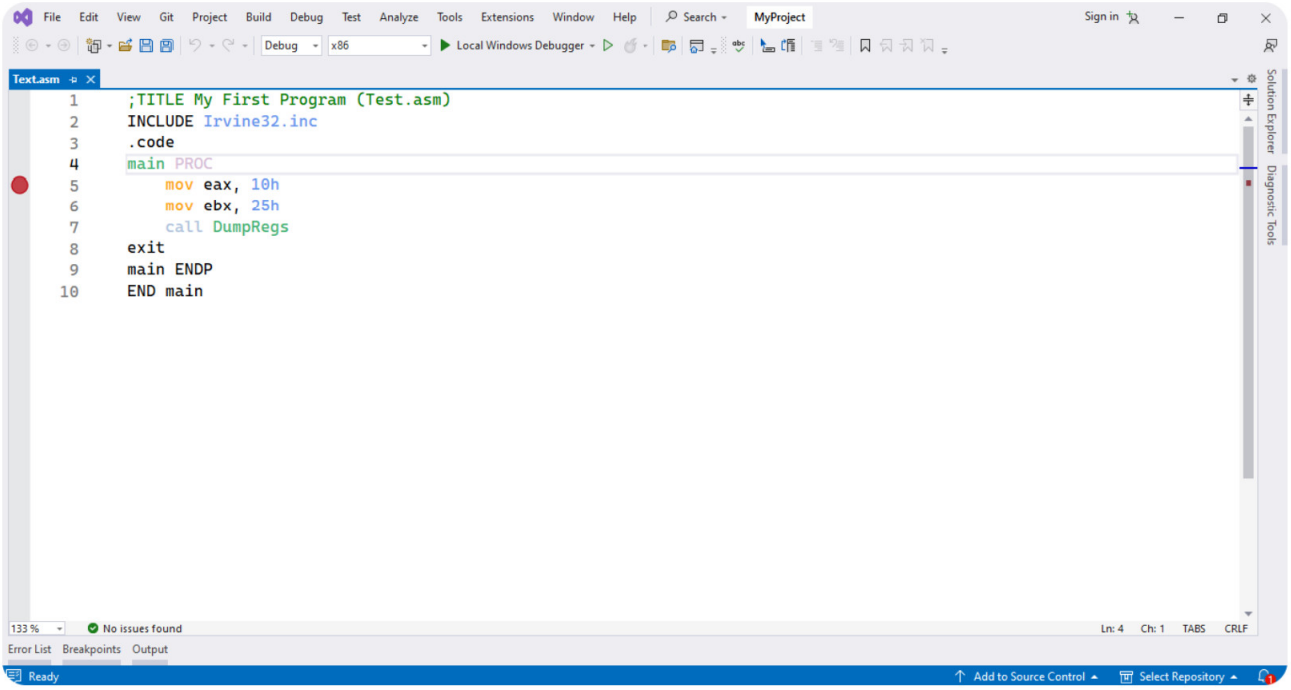
Single Stepping

To see the values of internal registers and memory variables during execution, let us use an example. Copy the following code onto your Text.asm file.

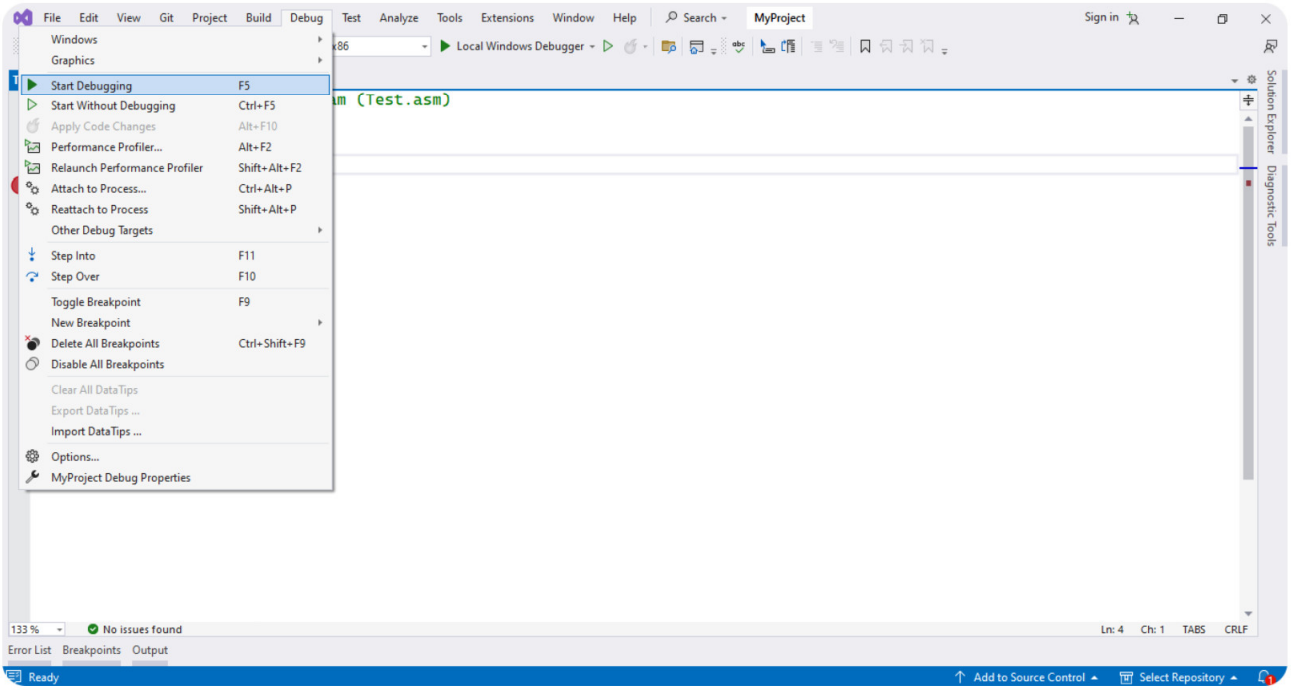
Code:

```
;TITLE My First Program (Text.asm)
INCLUDE Irvine32.inc
.code
main PROC
     mov eax, 10h
     mov ebx, 25h
     call DumpRegs
exit
main ENDP
END main
```

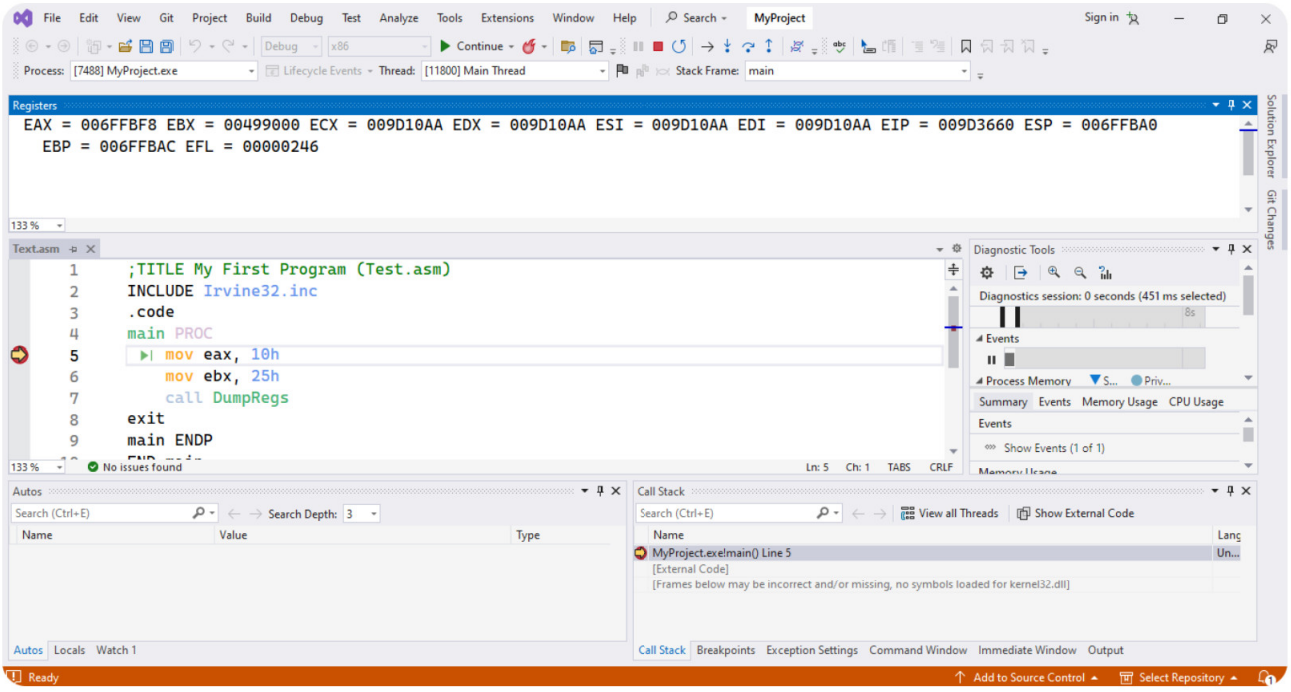
1. Right-Click on line 5 to create a breakpoint OR hover over the cursor at line and insert the breakpoint.



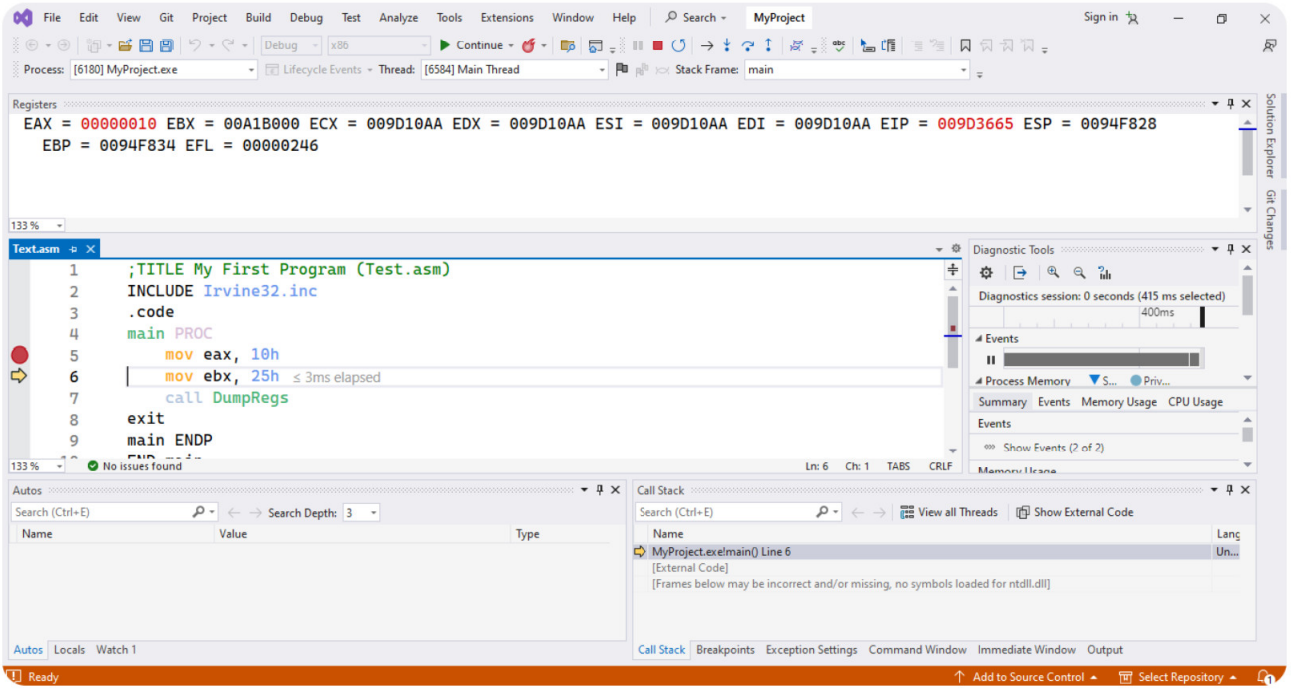
2. Click on Debug tab from the toolbar, select Start Debugging (F5).

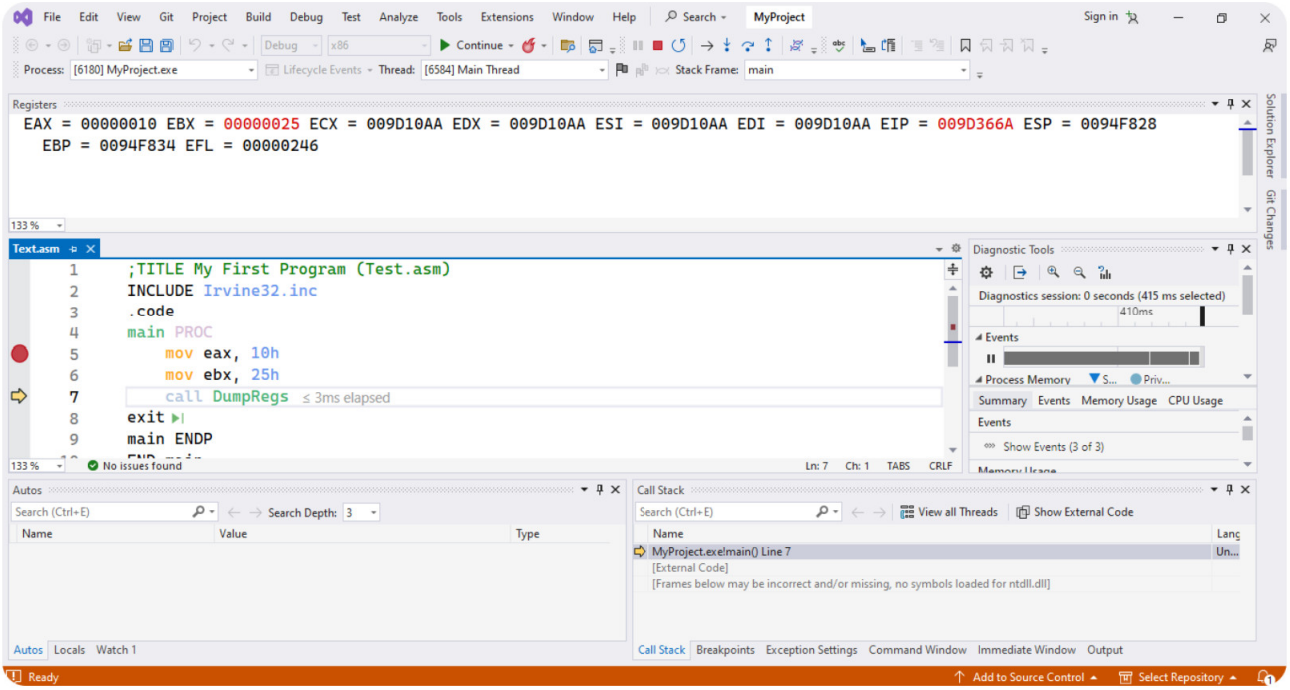


3. Click on “Debug” tab, select “Windows” and then select “Registers” option OR press “CTRL + ALT + G”.



4. Press F11 to step into each instruction. You can see the reflected stepping for each instruction in “Editor” and in the “Registers” windows.





5. The stepping will terminate when it reaches the line with a call to “exit”.

Section 02 | Basic Elements of Assembly Language

Integer Constants

Integer constants are made up of an optional leading sign, one or more digits and an optional suffix character.

Format:

`[ {+ | -}]` digits radix

Examples:

- `26` for decimal
- `26d` for decimal
- `10111110b` for binary
- `42o` for octal
- `1Ah` for Hexadecimal
- `0A3h` for Hexadecimal

Character Constants

Character constants are made up of a single character enclosed in either single or double quotes.

Example:

```
'A' "d"
```

String Constants

A string of characters enclosed in either single or double quotes.

Example:

```
"Hello World"
```

Identifiers

An identifier is a programmer-defined name of a variable, procedure or code label.

Format:

They may contain between `1` and `247` characters. They are not case sensitive. The first character must be a letter (`A..Z, a..z`), underscore `_`, `@`, `?`, or `$`. Subsequent characters may also be digits.

An identifier cannot be the same as an assembler reserved word. For example: reserved words are instruction mnemonics, directives, attributes, operators, predefined symbols.

Examples:

```
myVar
_abc
hello2
```

Directives

A directive is a command embedded in the source code that is recognized and acted upon by the assembler. Directives do not execute at runtime. They can assign names to memory segments. In MASM, directives are case insensitive. For example, it recognizes `.data`, `.DATA` and `.Data` as equivalent.

Let us see what different directives we can use to define segments of our program.

The `.data` directive identifies the area of a program containing variables:

Syntax:

```
.data
```

The `.code` directive identifies the area of a program containing executable instructions:

Syntax:

```
.code
```

Program Template

```
; Program Template (Template.axm)
; Program Description
; Author
; Creatin Date
: Revisions
; Date

INCLUDE Irvine32.inc
.data
     ; insert variables here
.code
     main PROC
          ; insert executable instruction here
     exit
     main ENDP
     ; insert additional procedure here
END main
```

Comments

Comments are an important way for the writer of a program to communicate information about the program's design to a person reading the source code.

Comments can be specified in two ways:

Single-line comments: beginning with a semicolon character (;). All characters following the semicolon on the same line are ignored by the assembler.

Block comments: beginning with the COMMENT directive and a user-specified symbol. For example:

```
COMMENT !

This line is a comment

This line is also a comment

!
```