Assembly Language LAB

Islamic University – Gaza
Engineering Faculty
Department of Computer Engineering
2013
ECOM 2125: Assembly Language LAB
Eng. Ahmed M. Ayash

Lab # 1

Introduction to Assembly Language

February 11, 2013

Objective:

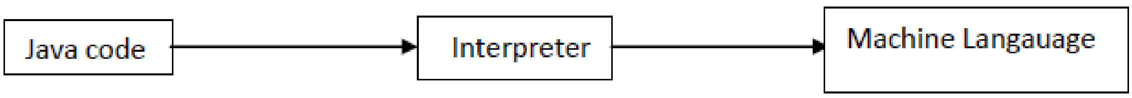
To be familiar with Assembly Language.

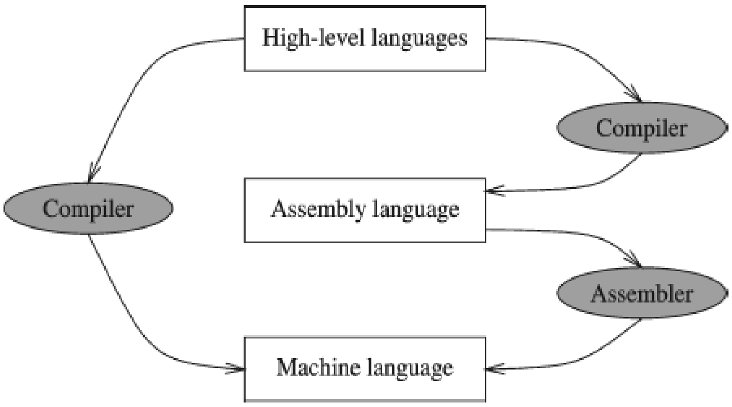
1. Introduction:

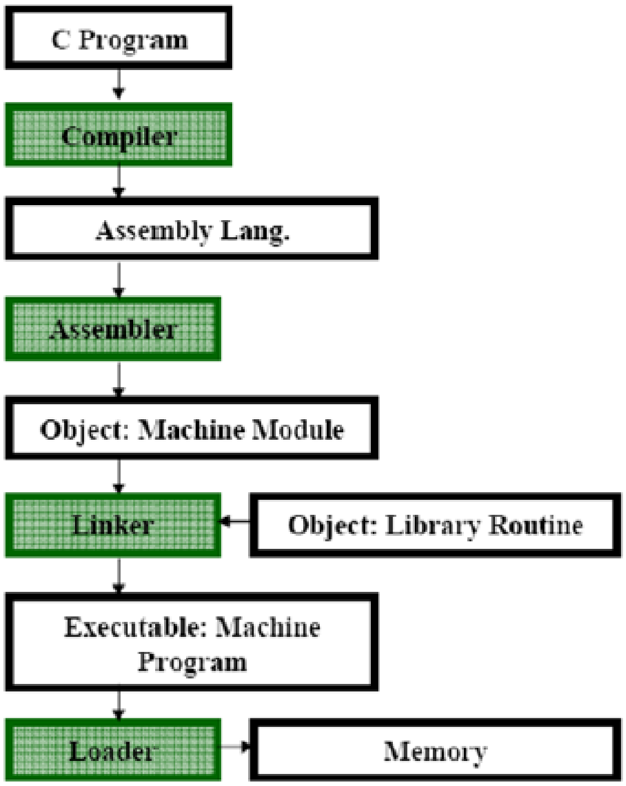
Machine language (computer's native language**)** is a system of impartible instructions executed directly by a computer's central processing unit (CPU).

- Instructions consist of binary code: 1s and 0s

Machine language can be made directly from java code using interpreter.





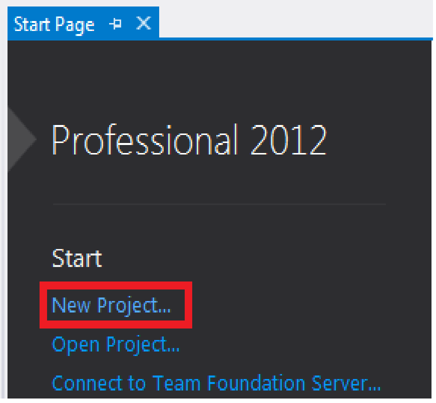


The difference between compiling and interpreting is as follows. Compiling translates the high-level code into a target language code as a single unit. Interpreting translates the individual steps in a high-level program one at a time rather than the whole program as a single unit. Each step is executed immediately after it is translated.

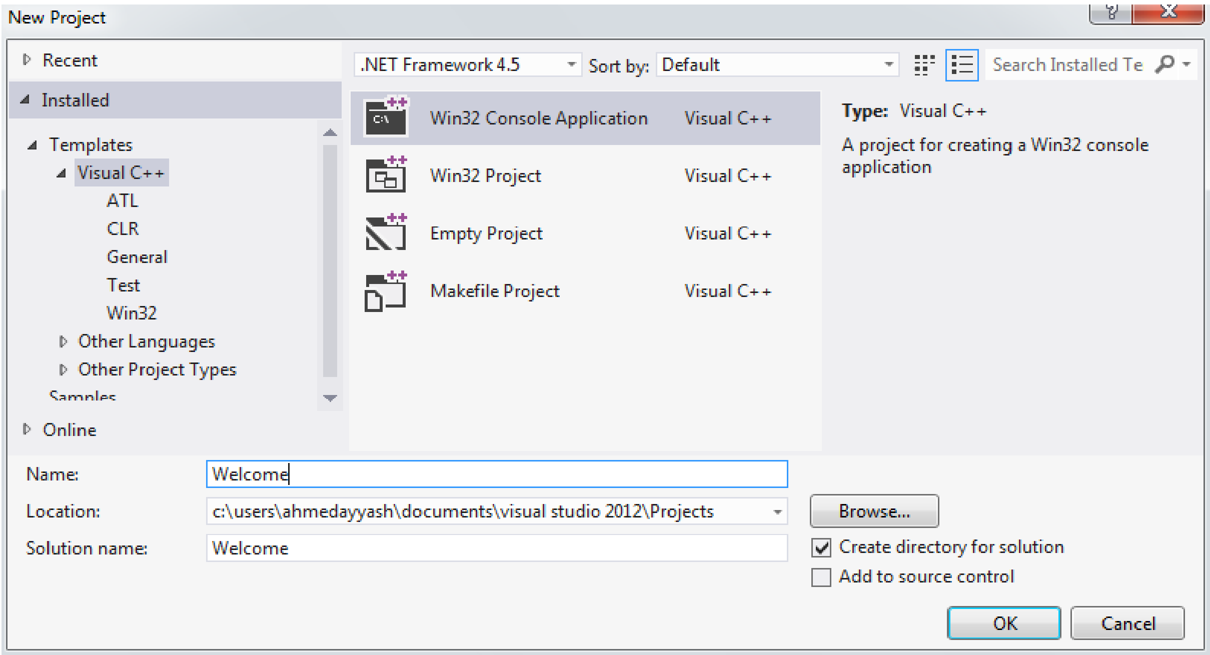
C, C++ code is executed faster than Java code, because they transferred to assembly language before machine language.

Using Visual Studio 2012 to convert C++ program to assembly language:

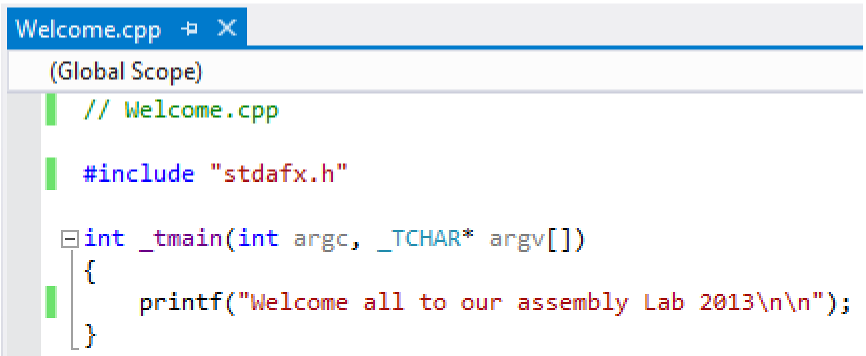
- From **File** menu >> choose **new** >> then choose **project**.
  Or from the **start** page choose **new project.**



- Then the new project window will appear,
  - choose visual C++ and win32 console application
  - The project name is welcome:



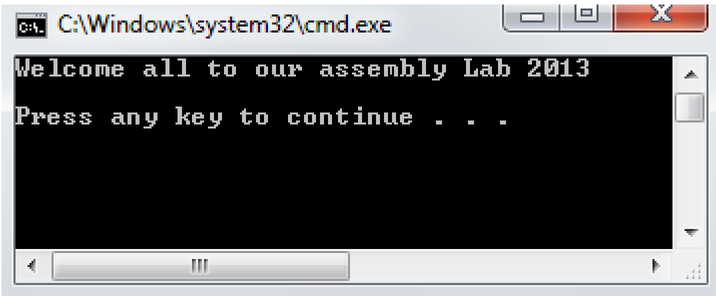
This is a C++ Program that print "Welcome all to our assembly Lab 2013"



To run the project, do the following two steps in order:

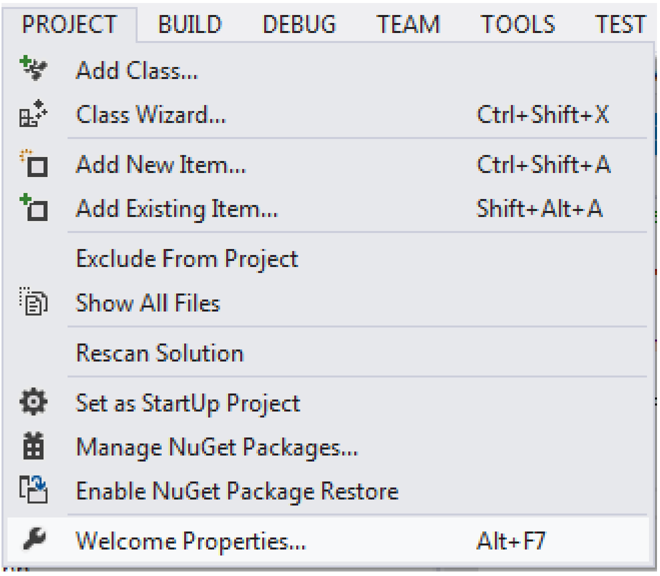
1. From **build** menu choose **build Welcome**.
2. From **debug** menu choose **start without debugging**.

The output is

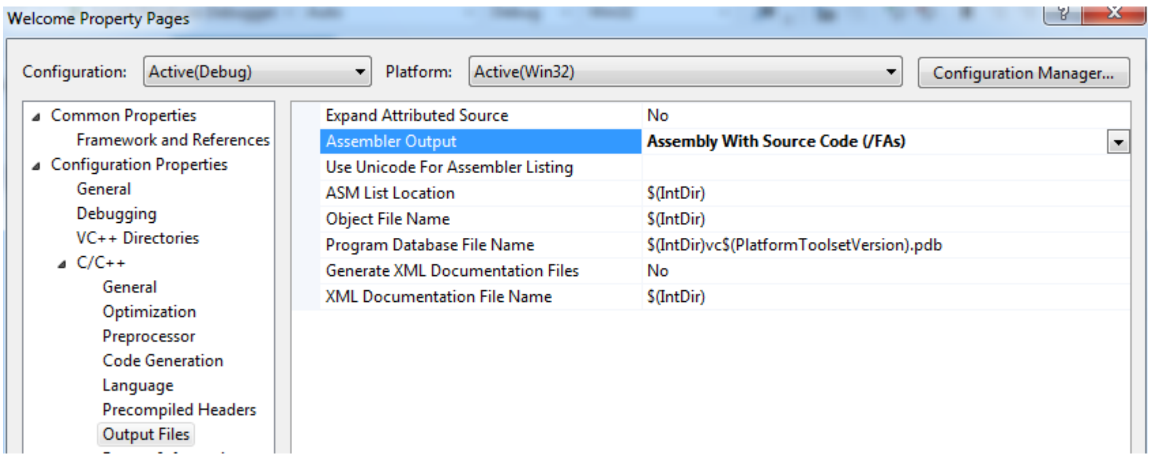


To convert C++ code to Assembly code we follow these steps:

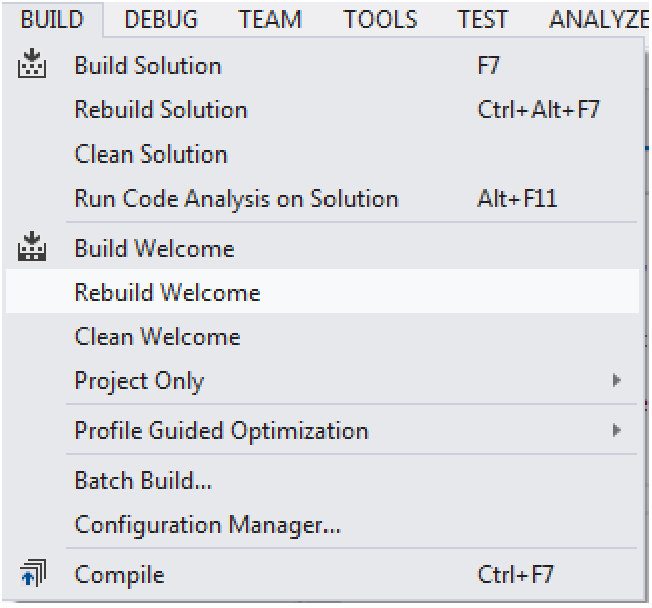
1)



2)



3)



We will find the Assembly code on the project folder we save in (*Visual Studio 2012\Projects\Welcome\Welcome\Debug*), named as **Welcome.asm**

Part of the code:

```
; 7    :  printf("Welcome all to our assembly Lab 2013\n\n");

     mov  esi, esp
     push OFFSET ??_C@_0CH@DDJPFAA@Welcome?5all?5to?5our?5assembly?5Lab?5@
     call DWORD PTR __imp__printf
     add  esp, 4
     cmp  esi, esp
     call __RTC_CheckEsp
```

2. Assembly Language

Assembly Language is a programming language that is very similar to machine language, but uses symbols instead of binary numbers. It is converted by the assembler (e.g. **Tasm** and **Masm**) into executable machine-language programs.

Assembly language is machine-dependent; an assembly program can only be executed on a particular machine.

2.1 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.

2.1.1 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 and TASM.

- Masm.exe creates an .obj file from an .asm file.

2.1.2 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.

- Link.exe creates an .exe file from an .obj file.
- Use make16.bat to assemble and link a 16-bit format assembly program.
- Use make32.bat to assemble and link a 32-bit format assembly program.

2.1.3 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.

2.1.4 Editor

You need a text editor to create assembly language source files. MASM6.15 has its own editor or you can use for example Notepad++.

To make programs in assembly language, you must know some information about the 8086 microprocessor. The 8086 contains 14 registers. Each register is 16 bits long. See Figure (1)

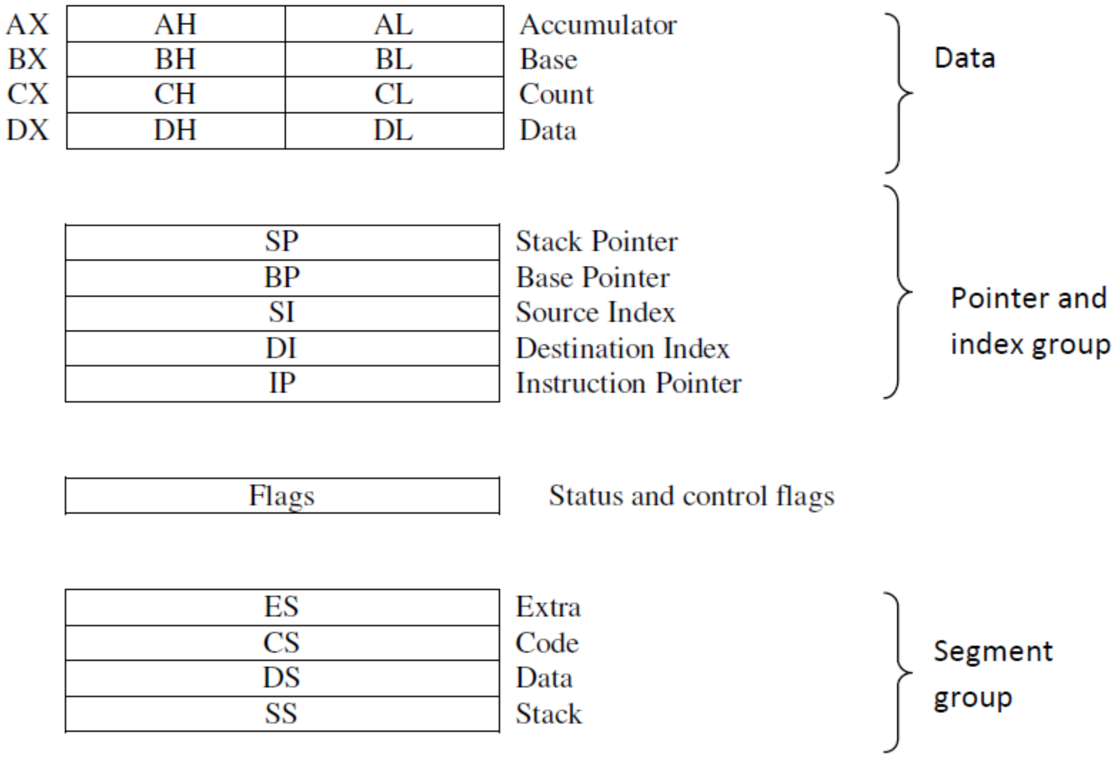
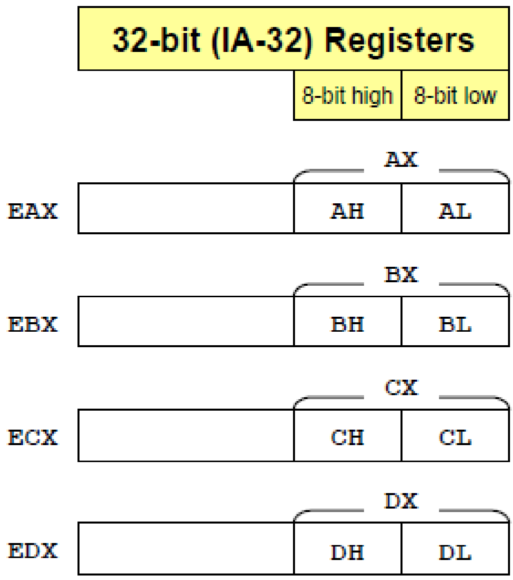
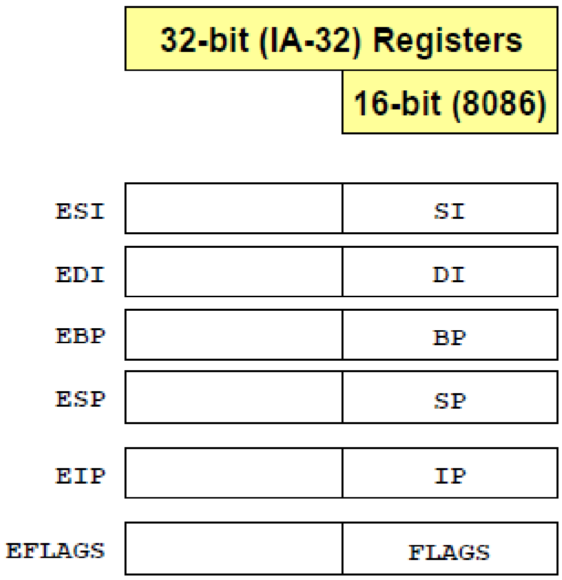


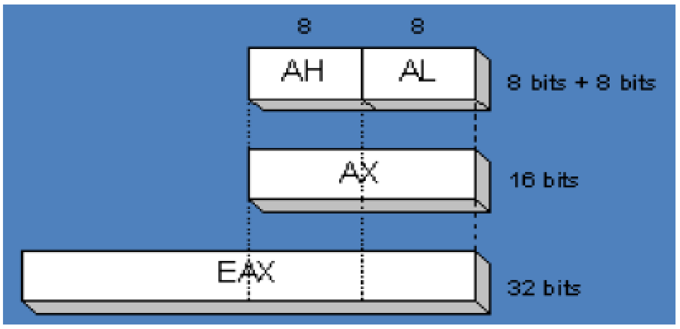
Figure (1): Registers of 8086 microprocessor





General-Purpose Registers:

Named storage locations inside the CPU, optimized for speed.
To Access Parts of Registers we can Use 8-bit name, 16-bit name, or 32-bit name; this is applied to EAX, EBX, ECX, and EDX.



Each register has different usage as shown in Table (1) below. The general purpose registers can be "split". AH contains the high byte of AX and AL contains the low byte. You also have: BH, BL, CH, CL, DL, DH.
So if for example, DX contains the value 1234h DH would be 12h and DL would be 34h.

Segment Registers		
CS	Code Segment	16-bit number that points to the active code-segment
DS	Data Segment	16-bit number that points to the active data-segment
SS	Stack Segment	16-bit number that points to the active stack-segment
ES	Extra Segment	16-bit number that points to the active extra-segment
Pointer Registers		
IP	Instruction Pointer	16-bit number that points to the offset of the next instruction
SP	Stack Pointer	16-bit number that points to the offset that the stack is using
BP	Base Pointer	used to pass data to and from the stack
General-Purpose Registers		
AX	Accumulator Register	mostly used for calculations and for input/output
BX	Base Register	Only register that can be used as an index
CX	Count Register	register used for the loop instruction
DX	Data Register	input/output and used by multiply and divide
Index Registers		
SI	Source Index	used by string operations as source
DI	Destination Index	used by string operations as destination

Table(1): Registers of 8086 microprocessor and their purposes

And a 16-bit FLAG Register. The FLAGS Register consists of 9 status bits. These bits are also called flags, because they can either be **SET** (**1**) or **NOT SET** (**0**). All these flags have a name and purpose.

Control Flags: Control flags control the CPU's operation. For example, they can cause the CPU to break after every instruction executes, interrupt when arithmetic overflow is detected. Programs can set individual bits in the EFLAGS register to control the CPU's operation. Examples are the **Direction, Trap** and **Interrupt** flags.

Status Flags: The Status flags reflect the outcomes of arithmetic and logical operations performed by the CPU. They are the **Overflow**, **Sign**, **Zero**, **Auxiliary Carry**, **Parity**, and **Carry** flags.

Abr.	Name	Description
OF	Overflow Flag	if set ,an instruction generates an invalid signed result
DF	Direction Flag	used for string operations to check direction
IF	Interrupt Flag	if *set*, interrupt are enabled, else disabled
TF	Trap Flag	if *set*, CPU can work in single step mode
SF	Sign Flag	if *set*, resulting number of calculation is negative
ZF	Zero Flag	if *set*, resulting number of calculation is zero
AF	Auxiliary Carry	is set when an operation produces a carryout from bit 3 to bit 4
PF	Parity Flag	is set when an instruction generates an even number of 1 bits in the low byte of the destination operand.
CF	Carry Flag	is set when the result of an unsigned arithmetic operation is too large to fit into the destination.

Table(2): FLAGS Register

Instruction Forms:

Assembly instructions are made up of an operation code (op-code) and set operands. The *op-code* identifies the action to be taken. The operands identify the source and destination of the data. The operands identify CPU registers, memory locations, or I/O ports. The complete form of an instruction is:

```
op-code destination operand, source operand
```

For example:

```
INC AX            ; one operand (add 1 to register AX)
MOV AX, 100       ; two operands (store 100 in register AX)
```

Segments:

Code, Data, Stack and Extra; within the 1 MB of memory space the 8086 defines four 64 Kbyte memory blocks called the code segment, data segment, stack segment, and the extra segment.

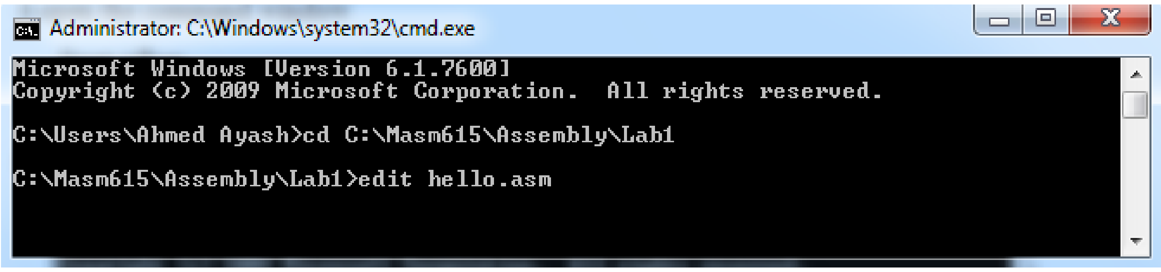
3. Hello Program on MASM assembler

Installing MASM:

Just extract the MASM6.15.zip on C:

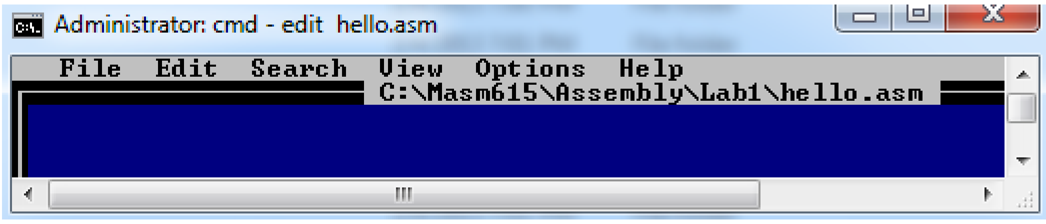
Assembling, Linking, Running a .asm File on MASM:

1-Make a folder for your assembly file
2-Extract the file *MASM Files*
3-Copy all the files in *MASM Files* to your Folder.
4-Open the command window
    Start->Run
5-Write **cmd** then enter
6-Change Directory to your Folder using cd command

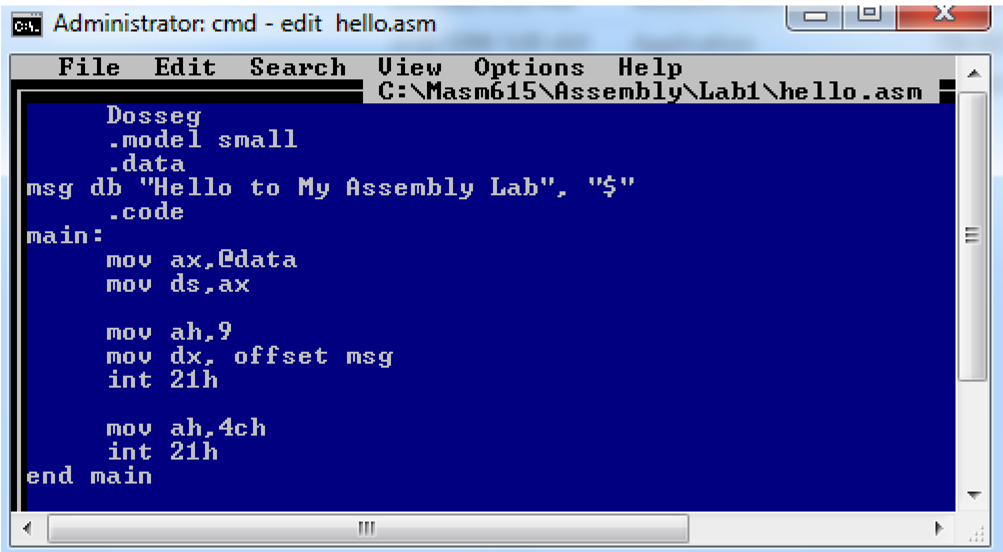


Or, instead of the previous steps, you can just open the cmd shortcut, which is in the *MASM Files* folder, and then the cmd will run with the location you are in.

7-On cmd write **edit hello.asm** then, the following screen will appear.

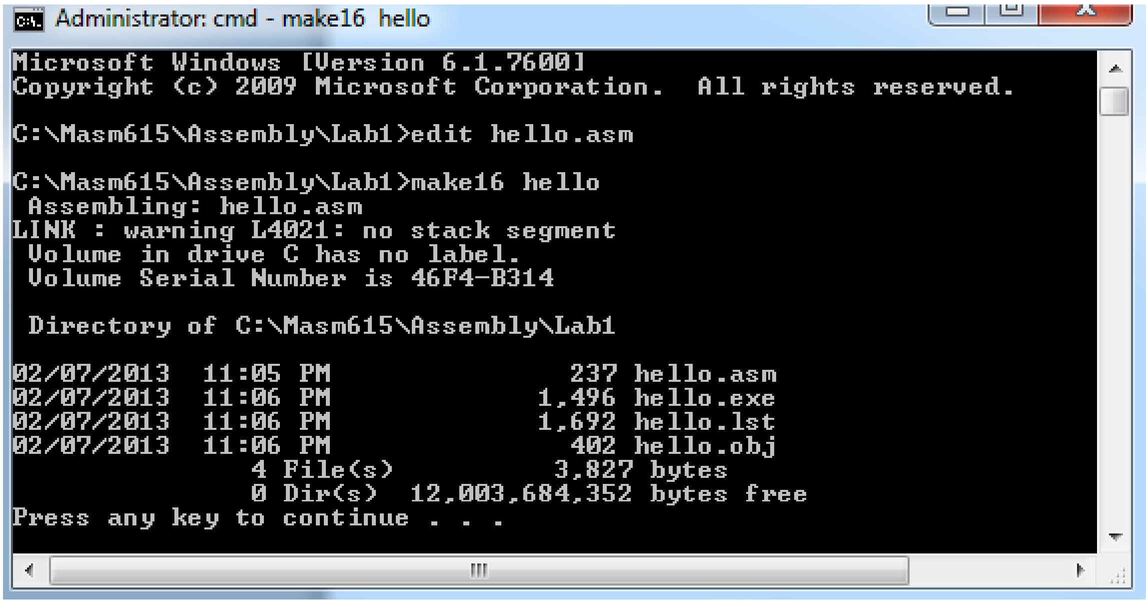


8-Write your Assembly code as the following:



9-Save your work then exit the editor

10-Type **make16 hello** then enter



Once a program is written, it can be assembled and linked using the **make16**.

Note:
If you use a 32 bit registers you will assemble and link using make32 instead of make16 command.

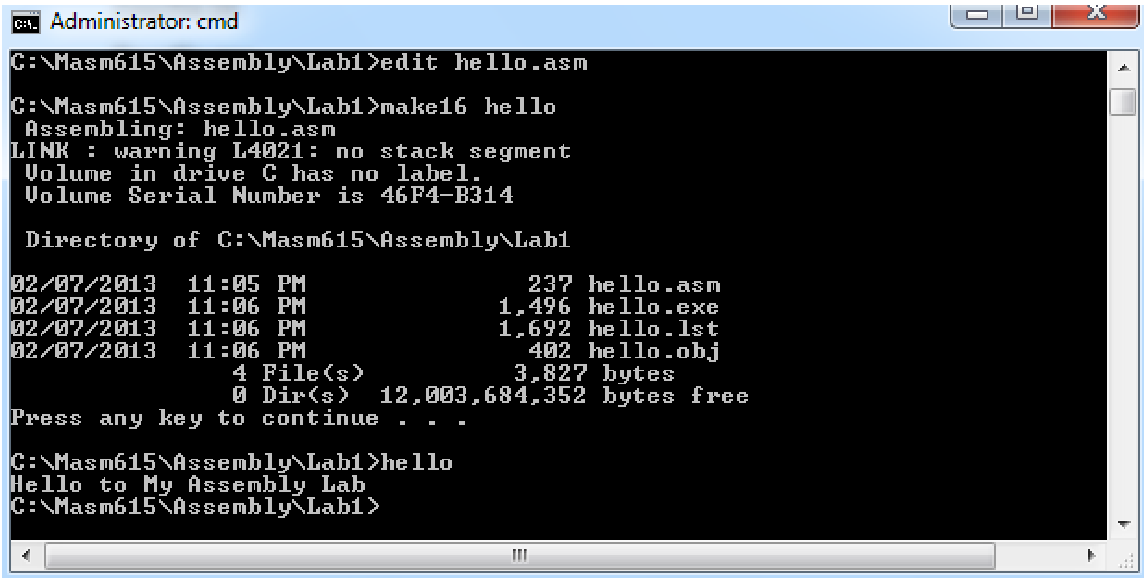
The following files will be created

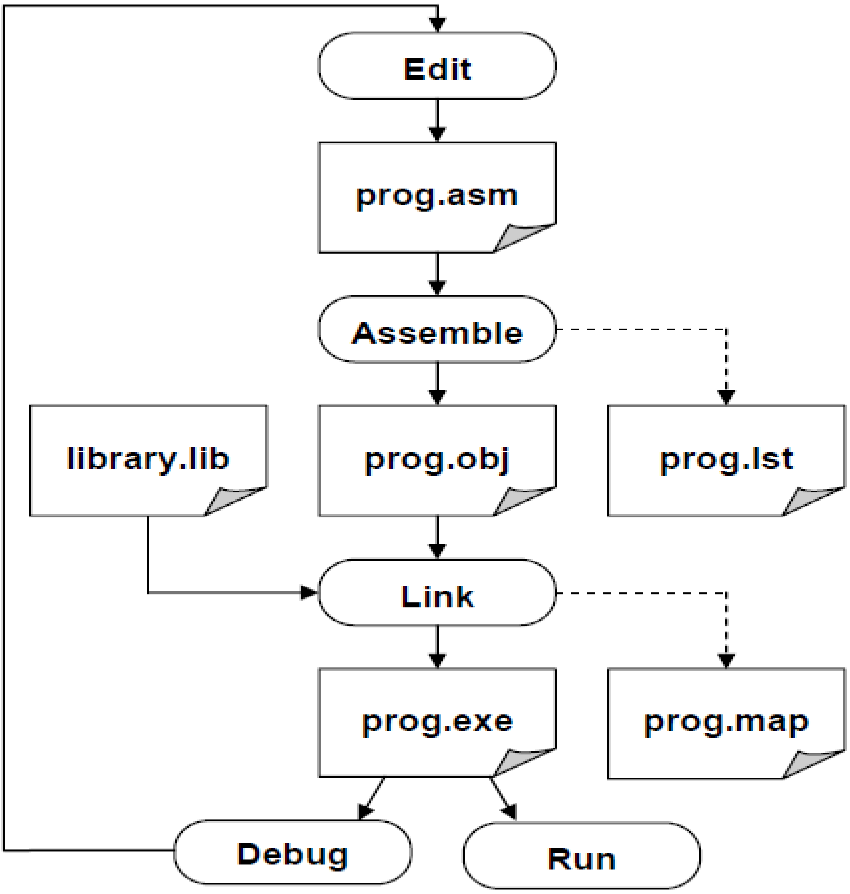
Yourfile.obj

Yourfile.lst

Yourfile.exe

11-Run your program using *hello*





Analysis of the Hello program:

Instruction	*Description*
DOSSEG	is a directive to tell the assembler to arrange data segment, code segment and stack segment as DOS arrangement.
MODEL SMALL	is a directive to tell the assembler to use one data segment and one code segment. All data fits in one 64K segment, all code fits in one 64K segment. Maximum program size is 128K.
.DATA	is a directive to put in data segment.
.CODE	is a directive to put in code segment.
@Data	is a default address of data segment to put it in ax register.
mov ax, @data **mov ds, ax**	As note we can't put data in ds register directly. So we use intermediate register (ax) as in the **mov ds, ax**
mov ah, 9	Put the service 9 in ah.
int 21h	(Interrupt 21 hexa), it has many services like 9,8,2, and each one has special work.
mov ah,4ch **int 21h**	The two statements to terminate the execution of the program.
END	is a directive to indicate the end of the file

Hello Program on TASM assembler

Assembling, Linking, Running a .asm File on TASM:

1-Extract the file *TASM Files*

2- Click **Start** ➔ **Run** then write **cmd** and click **OK.**

3- Go to directory **C:\Tasm\Bin**

4- Type the command **C:\Tasm\Bin\edit hello.asm**

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Ahmed Ayash>cd C:\TASM\BIN

C:\TASM\BIN>edit hello.asm
```

5- Write your code then save and exit.

```
Administrator: cmd - edit  hello.asm
  File  Edit  Search  View  Options  Help
                            C:\TASM\BIN\hello.asm
     Dosseg
     .model small
     .data
msg db "Hello to My Assembly Lab", "$"
     .code
main:
     mov ax,@data
     mov ds,ax
     mov ah,9
     mov dx, offset msg
     int 21h
     mov ah,4ch
     int 21h
end main
```

6- Write **C:\Tasm\Bin\tasm hello.asm** to create the file **hello.obj.** This file is the machine language for the program.

7- Write **C:\Tasm\Bin\ tlink hello.obj** to create the file **hello.exe.** This file is executable program.

8- Finally, write **C:\Tasm\Bin\hello.exe.** You will show the message on DOS screen.

```
Administrator: C:\Windows\system32\cmd.exe - tlink  hello
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Ahmed Ayash>cd C:\TASM\BIN

C:\TASM\BIN>edit hello.asm

C:\TASM\BIN>tasm hello
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996 Borland International

Assembling file:   hello.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  418k


C:\TASM\BIN>tlink hello
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
Warning: DOSSEG directive ignored in module HELLO.ASM
Warning: No stack

C:\TASM\BIN>hello
Hello to My Assembly Lab
C:\TASM\BIN>
```

Notes:

1. In TASM your code must be written in the following path ***C:\TASM\BIN***.
2. We use the following code to print **msg** on the screen.

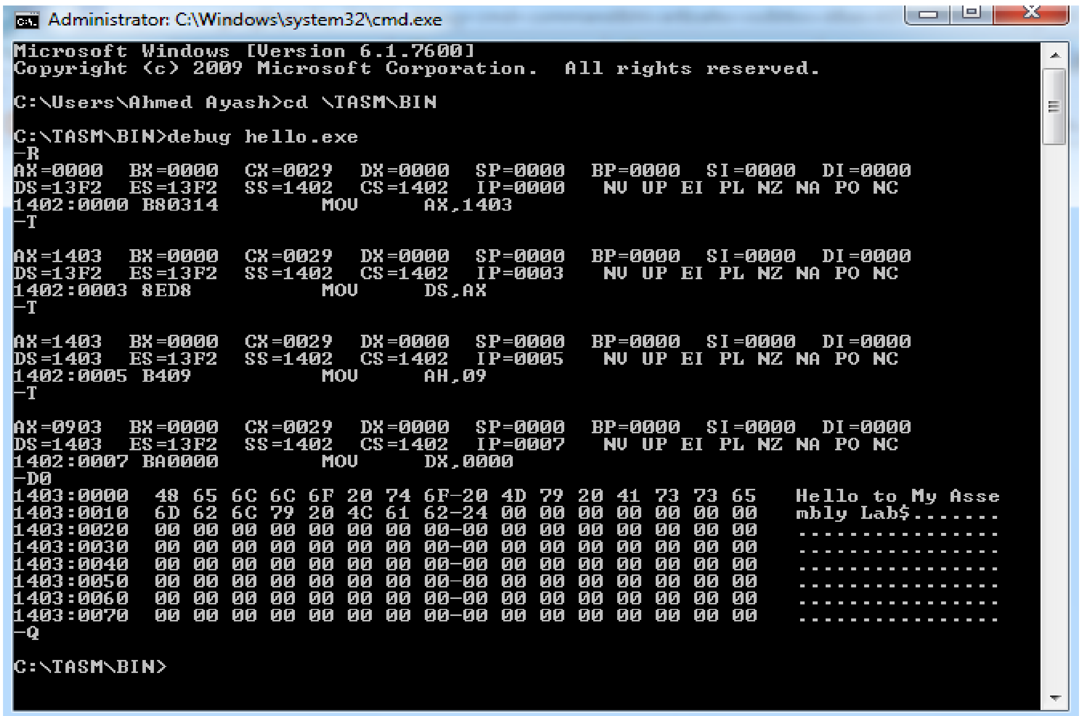
```
mov ax,@data
mov ds,ax
mov ah,9
mov dx, offset msg
int 21h
```

Debugging Assembly Language programs:

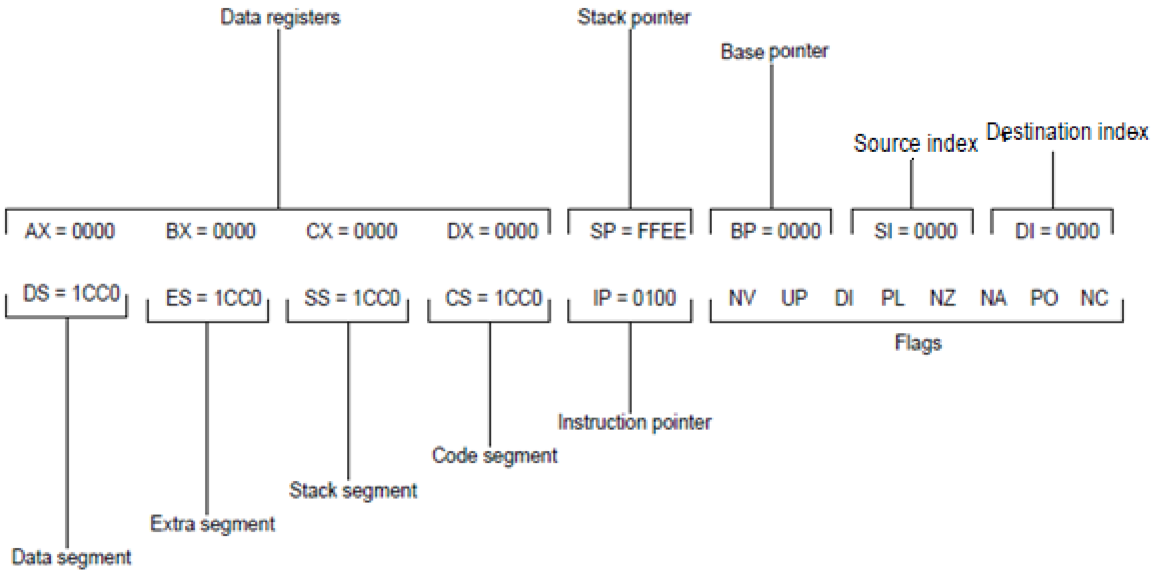
For Debugging you can use:

Debug hello.exe

COMMAND	SYNTAX	FUNCTION	EXAMPLE
Register	**R** [Register Name]	Examine or modify the contents of an internal register of the CPU	-**R** AX (AX reg.) -**RF** ZR (zero flag)
Dump	**D** [Start Addr] [End Addr]	Display the contents of memory locations specified by Address	-**D** DS:100 200 -**D** start-add end-add
Enter	**E** [Address] [Data]	Enter or modify the contents of the specified memory locations	-**E** DS:100 22 33 -**E** address data data
Fill	**F** [Start Addr] [End Addr] [Data]	Fill a block of memory with data	-**F** DS:100 120 22
Assemble	**A** [Starting address]	Convert assembly lang. instructions into machine code and store in memory	-**A** CS:100 -**A** start-address
Un-assemble	**U** [Starting Address]	Display the assembly instructions and its equivalent machine codes	-**U** CS:100 105 -**U** start-add end-add
Trace	**T** [Address][Number]	Line by line execution of specific number of assembly lang. instructions	-**T**=CS:100 -**T**=starting-address
Go	**G** [Starting Address] [Breakpoint Add.]	Execution of assembly language instructions until Breakpoint address	-**G**=CS:100 117 -**G**=start-add end-add



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Ahmed Ayash>cd \TASM\BIN

C:\TASM\BIN>debug hello.exe
-R
AX=0000  BX=0000  CX=0029  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1402  CS=1402  IP=0000   NV UP EI PL NZ NA PO NC
1402:0000 B80314        MOV     AX,1403
-T

AX=1403  BX=0000  CX=0029  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1402  CS=1402  IP=0003   NV UP EI PL NZ NA PO NC
1402:0003 8ED8          MOV     DS,AX
-T

AX=1403  BX=0000  CX=0029  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1403  ES=13F2  SS=1402  CS=1402  IP=0005   NV UP EI PL NZ NA PO NC
1402:0005 B409          MOV     AH,09
-T

AX=0903  BX=0000  CX=0029  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1403  ES=13F2  SS=1402  CS=1402  IP=0007   NV UP EI PL NZ NA PO NC
1402:0007 BA0000        MOV     DX,0000
-D0
1403:0000  48 65 6C 6C 6F 20 74 6F-20 4D 79 20 41 73 73 65   Hello to My Asse
1403:0010  6D 62 6C 79 20 4C 61 62-24 00 00 00 00 00 00 00   mbly Lab$.......
1403:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1403:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1403:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1403:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1403:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1403:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-Q

C:\TASM\BIN>
```



The complete set of possible flag mnemonics in Debug (ordered from left to right) are as follows:

Set	*Clear*
OV = Overflow	NV = No Overflow
DN = Direction Down	UP = Direction Up
EI = Interrupts Enabled	DI = Interrupts Disabled
NG = Sign Flag negative	PL = Sign Flag positive
ZR = Zero	NZ = Not Zero
AC = Auxiliary Carry	NA = No Auxiliary Carry
PO = Odd Parity	PE = Even Parity
CY = Carry	NC = No Carry

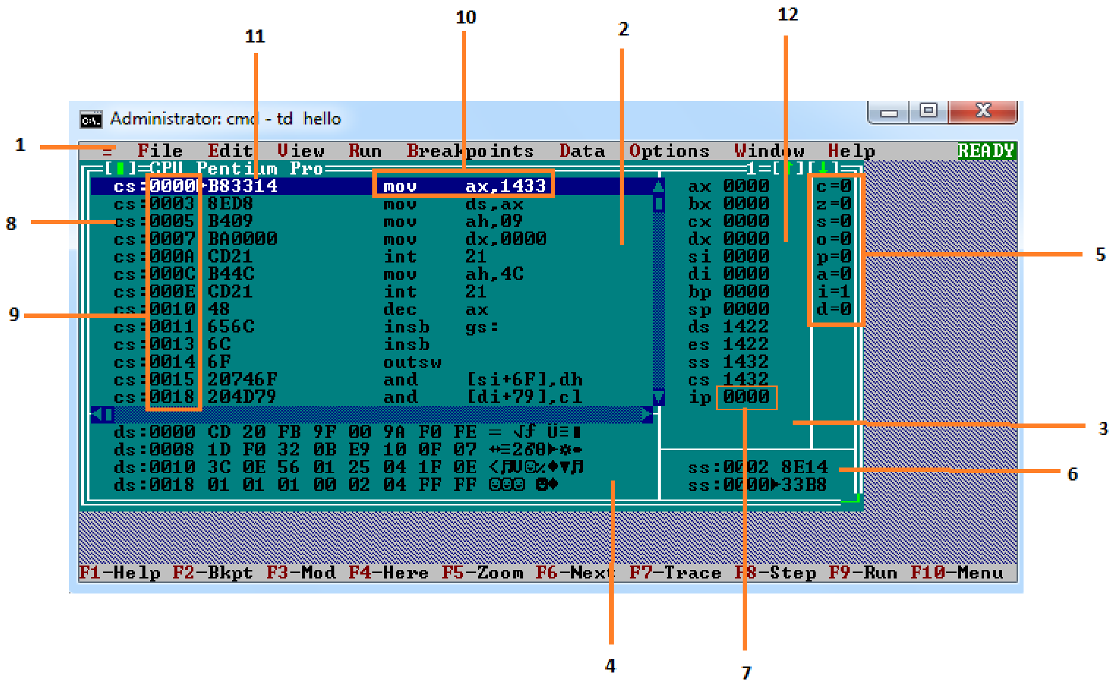
Command	Description	Default Segment
A	Assemble	CS
D	Dump	DS
E	Enter	DS
F	Fill	DS
G	Go (execute)	CS
L	Load	CS
M	Move	DS
P	Procedure trace	CS
S	Search	DS
T	Trace	CS
U	Unassemble	CS
W	Write	CS

Default Segments for Debug Commands.

Turbo Debugger (TASM Debugger):

The Turbo Debugger is a program that allows you to single-step your program (that means run it line-by-line while you watch what happens). You can observe the registers, the memory dump, individual variables, flags, and the code as you trace through your program. Also it is used to debug errors that have to be made by logic reasons. After you write your program you can use assembly turbo debugger by follow the following:

C:\Tasm\Bin\td Hello



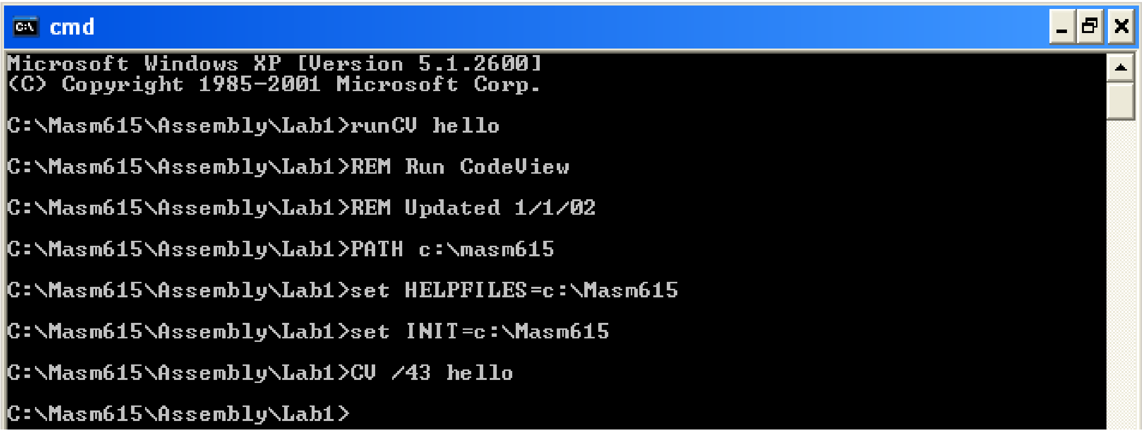
Number	Description
1	Indicate to the menu bar of turbo debugger.
2	Indicate to the region contain Code pane.
3	Indicate to the region contain Register pane.
4	Indicate to the region contain Data pane.
5	Indicate to the region contain Flag pane.
6	Indicate to the region contain Stack pane.
7	Indicate to the instruction pointer (IP) it contains the offset address of the instruction will be execute.
8	Indicate to Code register that have value of (1432) and we get it from register pane.
9	The offset address of each instruction.
10	This statement tells the assembler to put (@data) default offset address in AX and this value from figure equal to (1433).
11	Indicate to the machine language of statement and from figure it is equal to (B83314).
12	This column is the values of Registers.

CodeView Debugger (MASM Debugger):

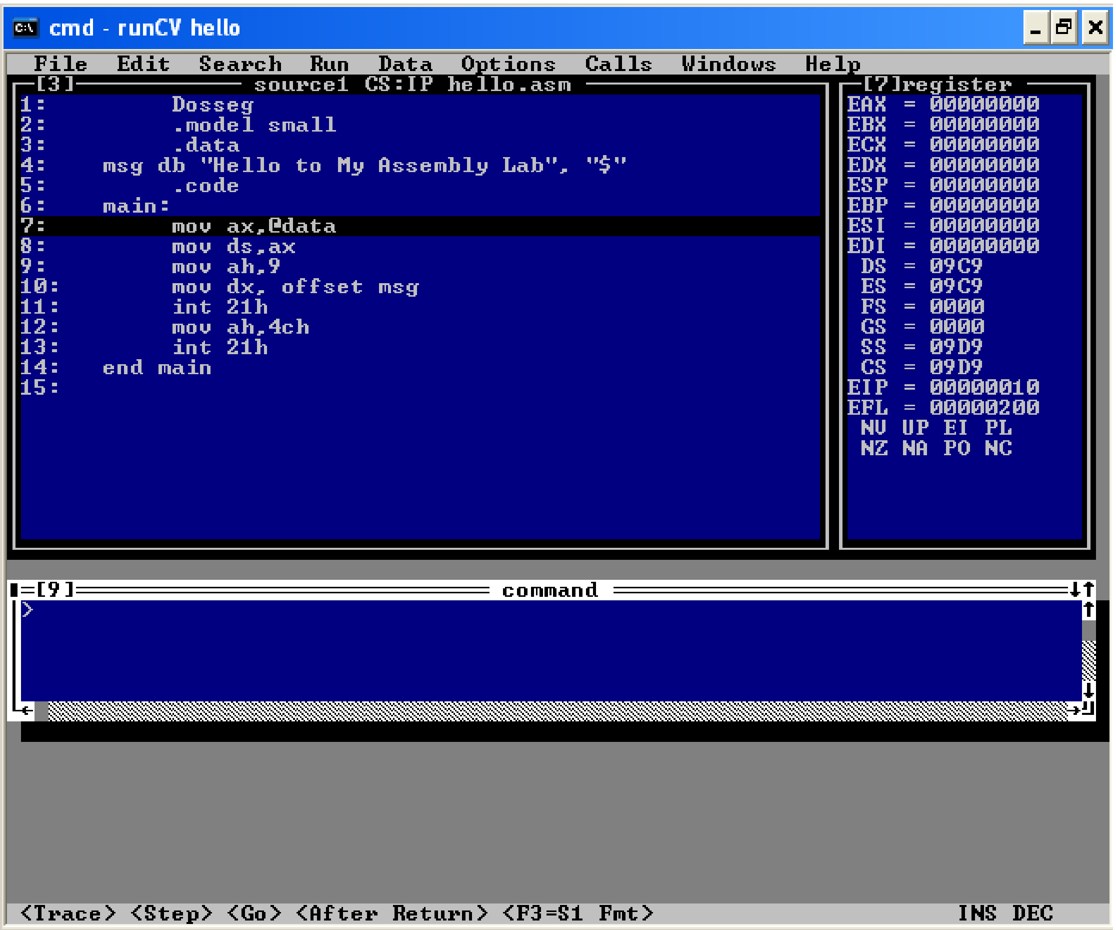
- For 16-bit programs, MASM supplies a 16-bit debugger named CodeView. CodeView can be used to debug only 16-bit programs and is already provided with the MASM 6.15 distribution.
- For 32-bit protected-mode programs, you need a 32-bit debugger. The latest version of the 32-bit Windows debugger is available for download for free from Microsoft.

To debug your code do as follows:

runCV Hello

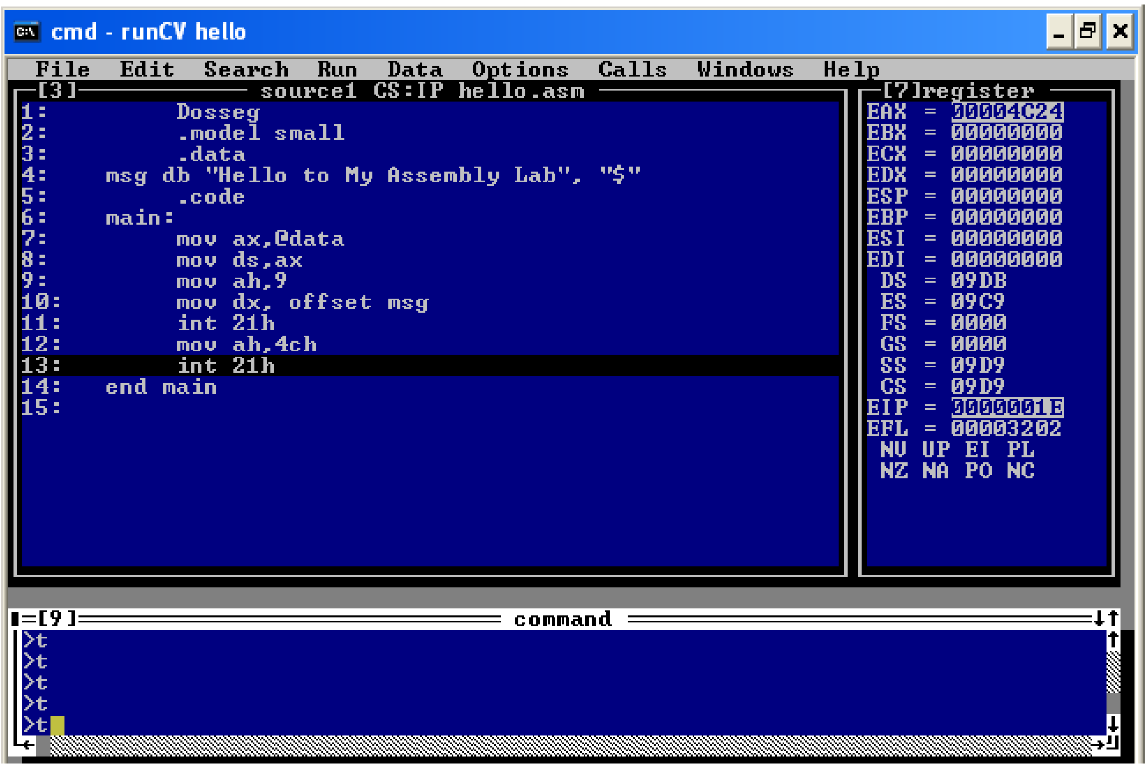


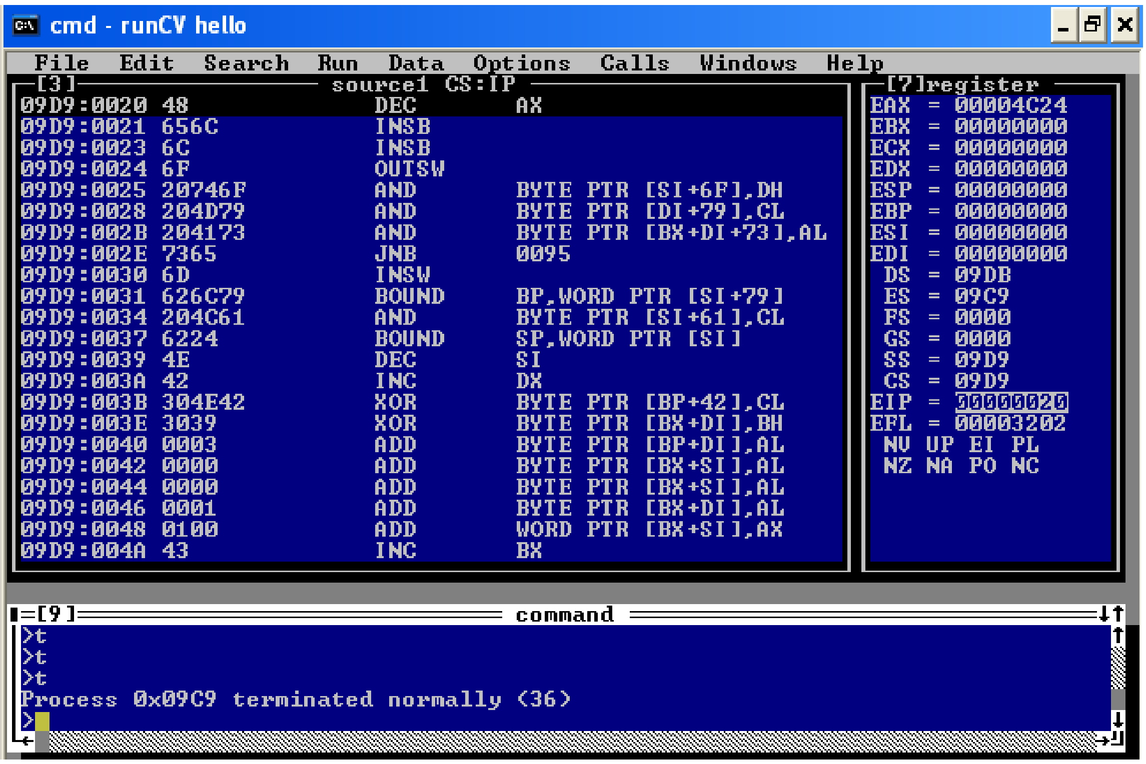
The highlight bar in the source window is automatically placed on your program's first executable instruction:



Begin Tracing the Program:

Press the F8 function key to begin tracing the program. Watch the highlight bar in the source window move downward, each time showing the next instruction about to be executed. Watch the registers change values.





Restart the Program:

Any time while debugging, you can restart a program by selecting *Restart* from the *Run* menu. Try it now, and again trace the program using the F8 key. After finishing the debugging, select Exit from the File menu.

Note:

1. RunCV does not work on windows 7, but it works fine on windows xp.
2. For the students who don't have windows xp, you can install xp on a virtual box such as VMware.

4. Lab work:

Write the previous code, "hello.asm" program then use Debugger to single step through this program using the (TRACE) command.

Homework:

Write an assembly language program to print all letters as follows:

AB..........YZ

Note: To print a character on the screen you have to use the **int 21h** with the service **2**, the character to be printed have to be in **dl**. For Example, the following code print **A** on the screen.

```
mov ah, 2
mov dl, 41h
int 21h
```

Note: Don't use loop.