Assembly Language Lab # 3
Assembly Language FundamentalsII

Assembly Language FundamentalsII

Objective:

To be familiar with Assembly Language

Assemble-Link Execute Cycle:

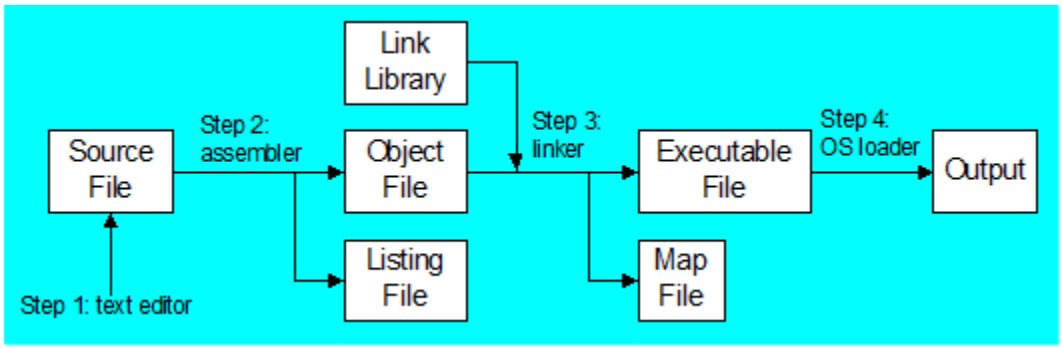
Following is a detailed description of editing, assembling, linking, and executing assembly language programs:

Step 1: A programmer uses a text editor to create an ASCII text file named the source file.

Step 2: The assembler reads the source file and produce s an object file , a machine-language translation of the program. Optionally, it produces a listing file. If any errors occur, the programmer must return to Step 1 and fix the program.

Step 3: The linker reads the object file and checks to see if the program contains any calls to procedures in a link library. The linker copies any required procedures from the link library, combines them with the object file, and produces the executable file. Optionally, the linker can produce a map file.

Step 4: The operating system loader utility reads the executable file into memory, branches the CPU to the program's starting address and the program begins to execute.
The following diagram describes the steps from creating a source program through executing the compiled program:



Notes:

If the source code is modified, Steps 2 through 4 must be repeated

.386:

- Enables assembly of nonprivileged instructions for the 80386 processor; disables assembly of instructions introduced with later processors.

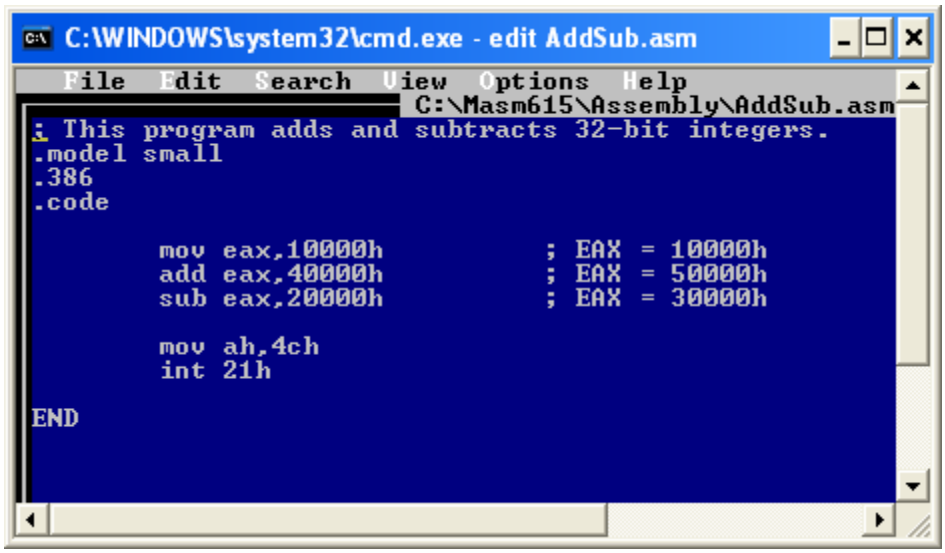
- The .386 directive identifies the minimum CPU required for the program (Intel386).

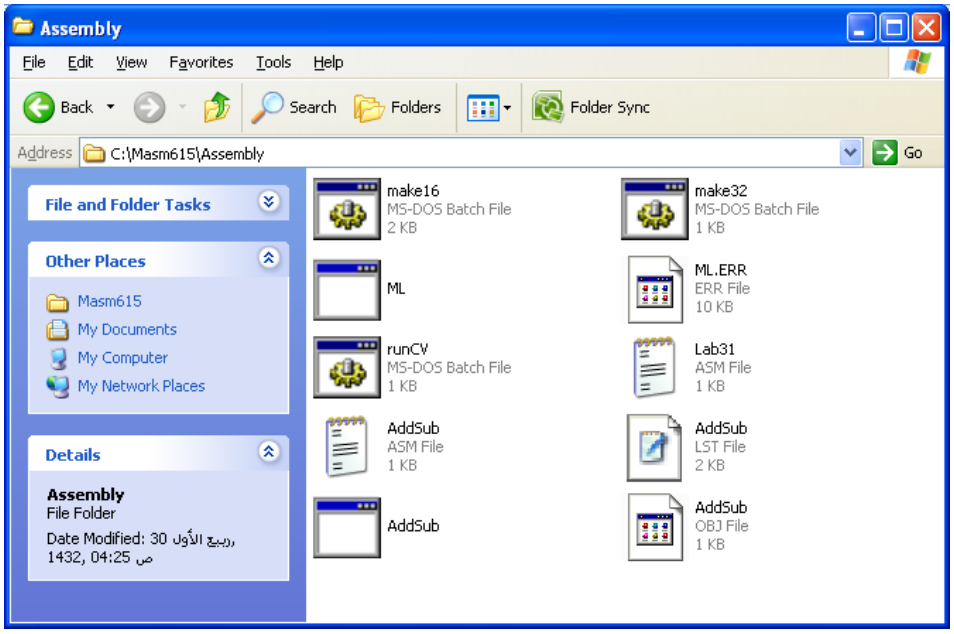
- Used after the .model directive in our small model. In other models it is used before .model directive.

Lab work:

Exercise1:

Example: AddSub.asm(The source file)





Listing File:

Use it to see how your program is compiled

Contains : source code , addresses , object code (machine language) , symbols (variables, procedures)

Example: addSub.lst

AddSub - Notepad

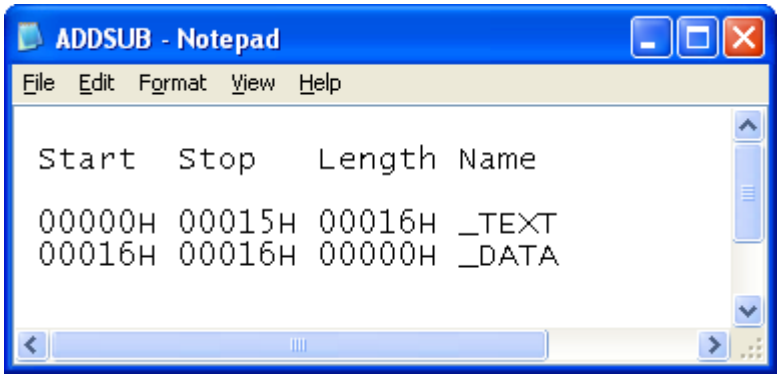
File Edit Format View Help

```
Microsoft (R) Macro Assembler Version 6.15.8803             03/05/11 04:25:19
AddSub.asm                                                   Page 1 - 1


                                ; This program adds and subtracts 32-bit integers.
                                .model small
                                .386
 0000                           .code

 0000   66| B8 00010000                 mov eax,10000h          ; EAX = 10000h
 0006   66| 05 00040000                 add eax,40000h          ; EAX = 50000h
 000C   66| 2D 00020000                 sub eax,20000h          ; EAX = 30000h

 0012   B4 4C                           mov ah,4ch
 0014   CD 21                           int 21h

                                END
Microsoft (R) Macro Assembler Version 6.15.8803             03/05/11 04:25:19
AddSub.asm                                                   Symbols 2 - 1



Segments and Groups:

                N a m e                 Size     Length   Align   Combine Class

DGROUP . . . . . . . . . . . . . .      GROUP
_DATA  . . . . . . . . . . . . . .      16 Bit   0000     Word    Public  'DATA'
_TEXT  . . . . . . . . . . . . . .      16 Bit   0016     Word    Public  'CODE'


Symbols:

                N a m e                 Type     Value    Attr

@CodeSize  . . . . . . . . . . . .      Number   0000h
@DataSize  . . . . . . . . . . . .      Number   0000h
@Interface . . . . . . . . . . . .      Number   0000h
@Model . . . . . . . . . . . . . .      Number   0002h
@code  . . . . . . . . . . . . . .      Text     _TEXT
@data  . . . . . . . . . . . . . .      Text     DGROUP
@fardata?  . . . . . . . . . . . .      Text     FAR_BSS
@fardata . . . . . . . . . . . . .      Text     FAR_DATA
@stack . . . . . . . . . . . . . .      Text     DGROUP

            0 Warnings
            0 Errors
```

Map File:

Contain Information about each program segment: Starting address, ending address, size

Example: addSub.map



Note:

1. Note that in our all previous programs, we have a warning that there is no stack segment, so from now we will define the stack segment in the next codes.

2. To define stack segments do as follows:

```
.STACK [[size]]
```

3. The optional size specifies the number of bytes for the stack (default 1,024)

Exercise2:

What is the output of the following Assembly language program?

1- Using msg1

2- Using msg2

Administrator: cmd - edit ex2.asm

File Edit Search View Options Help

C:\Masm615\Assembly\Lab2\ex2.asm

```
  .model small
  .386
  .stack 100h
.data
msg1 db "Letters from A to E ",0Ah,0Dh
     db "EEEEE",0Ah,0Dh
     db "DDDD",0Ah,0Dh
     db "CCC",0Ah,0Dh
     db "BB",0Ah,0Dh
     db "A",0Ah,0Dh,'$'

msg2 db "Letters from A to E ",0Ah,0dh
     db 5 dup("E"),0Ah,0Dh
     db 4 dup("D"),0Ah,0Dh
     db 3 dup("C"),0Ah,0Dh
     db 2 dup("B"),0Ah,0Dh
     db 1 dup("A"),0Ah,0Dh,'$'

.code
main:
mov ax,@data
mov ds,ax

mov ah,09h
mov dx,offset msg2
int 21h


mov ah,4ch
int 21h
end main
```

Homework:

1. Write an assembly code that writes the String “This String won't be displayed” on the console, then clear the Console and Write instead the following String “Hello displayed”.

2. Write an assembly language program that moves in ebx value 12344321 and moves in dx value 4334 using the variables String1 and String2 stored in memory.

String1 dd 12343412h
String2 dd 43214321h