LAB 4: Introduction to MASM

---

1. Introduction
    - An assembler converts the assembly instructions to machine-understandable code. We will use MASM which is the Microsoft Macro Assembler, an x86 assembler that uses Intel Syntax for MS-DOS and Microsoft Windows. After assembling the code, we get a .obj file.
    - After assembling the code, the linker combines the various modules and libraries to create a .exe file.
    - Finally, we run the generated .exe file.

2. Getting started
    - Mount the drive with the command mount c c:/masm11/bin (for CC lab systems).

    - Once you are in the bin directory you can try writing the below hello world program in a text editor of your choice (preferably notepad++ or visual studio code) and try compiling it using the below written commands

3. Hello world program

ASSEMBLER DIRECTIVES:

Assembler directives are the directions to the assembler which indicate how an operand or section of the program is to be processed. These are also called pseudo-operations which are not executable by the microprocessor. The various MASM directives are explained below.

.CODE	Indicates the start of the code segment (models only)
.DATA	Indicates the start of the data segment (models only)
.EXIT	Exits to DOS (models only)
.MODEL	Selects the programming model
.STACK	Selects the start of the stack segment (models only)
.STARTUP	Indicates the starting instruction in a program (models only)
ALIGN n	Align to boundary n (n = 2 for words, n = 4 for doublewords)
ASSUME	Informs the assembler to name each segment (full segments only)
BYTE	Indicates byte-sized as in BYTE PTR
DB	Defines byte(s) (8 bits)
DD	Defines doubleword(s) (32 bits)
DQ	Defines quadwords(s) (64 bits)
DT	Defines ten byte(s) (80 bits)
DUP	Generates duplicates
DW	Define word(s) (16 bits)
DWORD	Indicates doubleword-sized, as in DWORD PTR
END	Ends a program file
ENDM	Ends a MACRO sequence
ENDP	Ends a procedure
ENDS	Ends a segment or data structure
EQU	Equates data or a label to a label
FAR	Defines a far pointer, as in FAR PTR
MACRO	Designates the start of a MACRO sequence
NEAR	Defines a near pointer, as in NEAR PTR
OFFSET	Specifies an offset address
ORG	Sets the origin within a segment
OWORD	Indicates octalwords, as in OWORD PTR
PROC	Starts a procedure
PTR	Designates a pointer
QWORD	Indicates quadwords, as in QWORD PTR
SEGMENT	Starts a segment for full segments
STACK	Starts a stack segment for full segments
STRUC	Defines the start of a data structure
USES	Automatically pushes and pops registers
USE16	Uses 16-bit instruction mode
USE32	Uses 32-bit instruction mode
WORD	Indicates word-sized, as in WORD PTR

MASM programs have 2 formats for developing software: Full segment and models. We will look at both the approaches below.

The full-segment definitions are common to most assemblers, including the Intel assembler, and are often used for software development. The models are easier to use for simple tasks. The full-segment definitions offer better control over the assembly language task and are recommended for complex programs.

There are many models available to the MASM assembler, ranging from tiny to huge. To designate a model, use the. MODEL statement followed by the size of the memory system. The TINY model requires that all software and data fit into one 64K-byte memory segment; it is useful for many small programs. The SMALL model requires that only one data segment be used with one code segment for a total of 128K bytes of memory. Other models are available, up to the HUGE model.

You can use any method to write your programs, although using models is recommended for the lab.

1. Models approach:

```
1   .model small
2   .data
    1 reference
3   msg db "Hello World!$"
4   .code
5   .startup
    1056 references
6   mov ax,@data
    1056 references
7   mov ds,ax
    96 references
8   lea dx,msg
9
    1056 references
10  mov ah,9h
    332 references
11  int 21h
12  .exit
    47 references
13  end
14
```

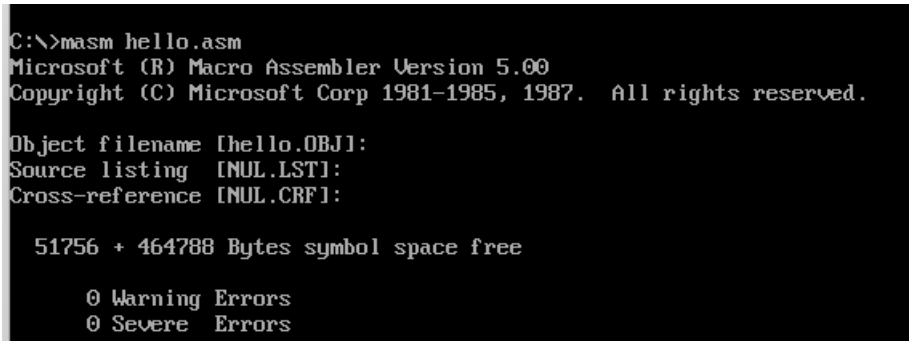
- In the code given below, INT 21H represents an interrupt to the operating system, if no key is typed it returns equal and MOV AX, 4C00H/4CH is used to terminate an application normally. LEA here stands for load effective address which calculates the effective address and stores it in the specified register. 09H displays a string of characters whose offset is specified by DX.
  So essentially, we are printing a string using these instructions.

- After writing the program, save it. (File -> save). To exit the text editor, click on File -> Exit

- To run the program, follow the instructions:

1. Compile Command– masm hello.asm

Press Enter 4 times to go with default settings.



2. Linking Command– link hello.obj

Press Enter 4 times to go with default settings

you can also avoid this warning by allocating some stack space in your code or by using model tiny instead of model small

3. Run– hello.exe

Another way of compiling

```
C:\MASM611\MASM611\BIN>ml hello.asm
Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993.  All rights reserved.

 Assembling: hello.asm

Microsoft (R) Segmented Executable Linker  Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992.  All rights reserved.

Object Modules [.obj]: hello.obj
Run File [hello.exe]: "hello.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

C:\MASM611\MASM611\BIN>hello.exe
Hello World!
```

if you don’t want to print the output you can open the debugger after the ml command to check your msg

```
C:\MASM611\MASM611\BIN>ml hello.asm
Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993.  All rights reserved.

 Assembling: hello.asm

Microsoft (R) Segmented Executable Linker  Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992.  All rights reserved.

Object Modules [.obj]: hello.obj/t
Run File [hello.com]: "hello.com"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:

C:\MASM611\MASM611\BIN>debugx hello.com
-_
```

```
-u 100
0865:0100 8D160C01        LEA     DX,[010C]
0865:0104 B409            MOV     AH,09
0865:0106 CD21            INT     21
0865:0108 B44C            MOV     AH,4C
0865:010A CD21            INT     21
0865:010C 48              DEC     AX
0865:010D 65              SEG     GS (unused)
0865:010E 6C              INSB
0865:010F 6C              INSB
0865:0110 6F              OUTSW
0865:0111 20576F          AND     [BX+6F],DL
0865:0114 726C            JB      0182
0865:0116 642124          AND     FS:[SI],SP
0865:0119 B8F909          MOV     AX,09F9
0865:011C BA2E15          MOV     DX,152E
0865:011F 52              PUSH    DX
-g =100 108
Hello World!AX=0924 BX=0000 CX=0019 DX=010C SP=FFFE BP=0000 SI=0000 DI=0000
DS=0865 ES=0865 SS=0865 CS=0865 IP=0108 NV UP EI PL ZR NA PE NC
0865:0108 B44C            MOV     AH,4C
```

Question 1: Convert Roman to Integer
Sample Test Case: Input: “MCMIV”
Output: “1904”

Instructions: You can store the input string by allocating a memory location in the data section. No need to use interrupts for this lab. You can use a special character such as ‘$’ or ‘#’ at the end of input. This will let you know when you arrive at the end of the sentence. For example:

```
.MODEL TINY
.DATA
    input_buffer    db  'MCMIV$'

.CODE
.STARTUP
```

Run till AH, 4C instruction location in the assembler. Your output should show in DX register in this case(hex of 1904 as output is fine). Conversions for reference:

SYMBOL	VALUE
I	1
IV	4
V	5
IX	9
X	10
XL	40
L	50
XC	90
C	100
CD	400
D	500
CM	900
M	1000

Question 2: Reverse words in a sentence.
Sample Test Case: Input: “This is a sample sentence”
Output: “sentence sample a is This”

Instructions: You can store the input string by allocating a memory location in the data section. No need to use interrupts for this lab. You can use a special character such as ‘$’ or ‘#’ at the end of input. This will let you know when you arrive at the end of the sentence. For example:

```
.MODEL TINY
.DATA
    inputString DB 'This is a sample sentence$'
    buffer DB 255 DUP(?)
.CODE
.STARTUP
```

Run till AH, 4C instruction location in the assembler. Your output should show in memory location of ‘buffer’ in this case(see the above example).