Assembly Programming

Lesson 15

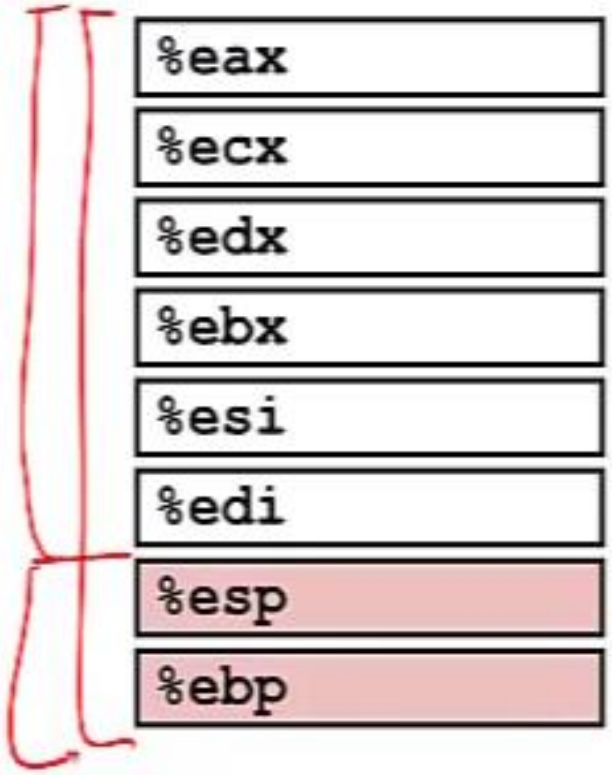
Exercise

1. Write an assembly program to display a text on the screen
2. Write an assembly program to adds two integers
   - Work in your groups, and submit, and each group will make a presentation of your work
   - Presentations will be during Tuesday class session
   - Each group will have 10 minutes of presentation
   - The work will be marked out of 10 (5 points for each question)

Quick Review

Moving Data: IA32

- Moving Data
  - `movx` *Source, Dest*
  - `x` is one of {`b`, `w`, `l`}
  - `movl` *Source, Dest*:
    Move 4-byte "long word"
  - `movw` *Source, Dest*:
    Move 2-byte "word"
  - `movb` *Source, Dest*:
    Move 1-byte "byte"



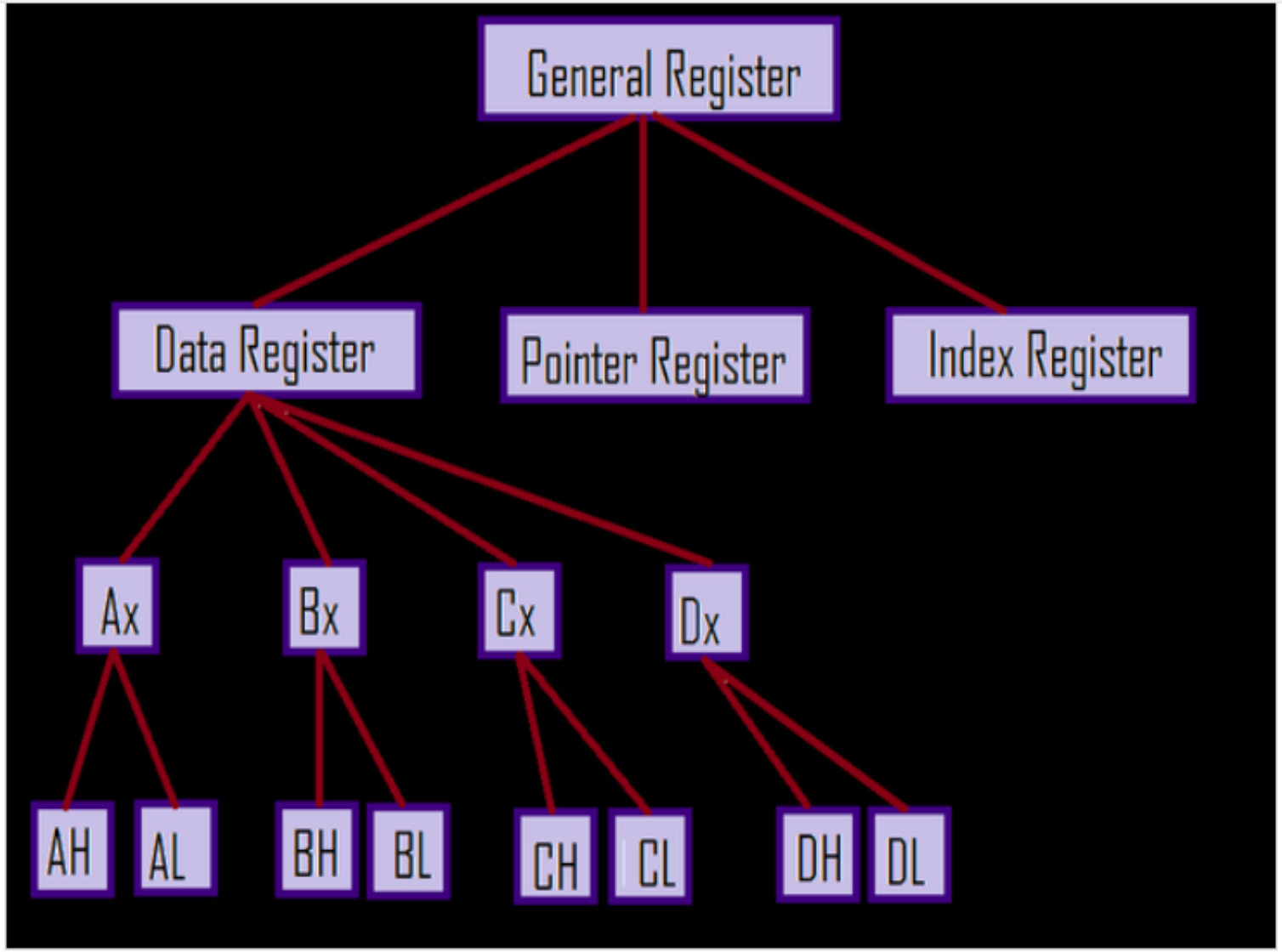
x86 Registers

32 bits

16 bits

EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL
ESI			
EDI			
ESP	STACK POINTER		
EBP	BASE POINTER		

Register:


General Register
Data Register
Pointer Register
Index Register
Ax
Bx
Cx
Dx
AH
AL
BH
BL
CH
CL
DH
DL

General Registers:

Ax:

- Ax is an accumulator register that is used by the processor to store result obtained by performing Arithmetic and logical operation.
- CPU uses this memory to store the result and later retrieve this information for the further process.
- The accumulator is very important, Otherwise, the processor would have to store the result in the primary memory (RAM), which would take a lot of time which will eventually lead to inefficiency.

Bx:

- Bx register is a base register.
- This register is used when addressing memory addresses.
- It is mainly used for the indexing of the addresses.
- Base register along with the offset can point to a memory address.

Cx:

- Cx register is a counter register. The counter register stores the loop count in the process of iteration.
- This counter register also acts as a counter in string manipulation and other shift/rotate instructions.

Dx:

- Dx register is a Data register.
- This register act as a temporary memory holder.
- Whenever there is a need to transfer data from one location to another, the information is stored in the data register.
- Data registers contain the whole information.

Download and install MASM32 software

Assembly Basic Syntax

An assembly program can be divided into three sections:

- The **data section**
- The **bss section**
- The **text section**

1. The data Section:

- **The data section is used for declaring initialized data or constants.**
- **This data does not change at runtime.**
- **You can declare various constant values, file names or buffer size etc. in this section.**

- **The syntax for declaring data section is:**

  section .data

2. The bss Section:

- **The bss section is used for declaring variables.**
- **The syntax for declaring bss section is:**

  section .bss

3. **The text section**:

- The text section is used for keeping the actual code.
- This section must begin with the declaration global main, which tells the kernel where the program execution begins.
- The syntax for declaring text section is:

```
section .text
    global main
main:
```

4. Comments

- Assembly language comment begins with a semicolon (;).
- It may contain any printable character including blank.
- It can appear on a line by itself, like:

```
; This program displays a message on screen
```

or, on the same line along with an instruction, like:

```
add eax ,ebx     ; adds ebx to eax
```

A Typical Nasm file layout

```
section .data
        ;this is a comment
        hello: db 'this is an example string',10
        HelloLength:  equ 26
        ;db = define byte, dw = define word, dd = define word

section .bss
        ;here you declare modifiable variables
        Variable1: resb 255
        Variable2: resb 1
        Variable3: resw 1

section .text
        ;this is were your program code goes
        global _start     ;this allows external programs to see the start of your program
        _start:           ;after this label the program actually begins
              ;start coding here
```

Assembly Language Statements

Assembly language programs consist of three types of statements:

1. Executable instructions or instructions
2. Assembler directives or pseudo-ops
3. Macros

- The **executable instructions** or simply instructions tell the processor what to do.
- Each instruction consists of an operation code (opcode). Each executable instruction generates one machine language instruction.
- The **assembler directives** or pseudo-ops tell the assembler about the various aspects of the assembly process.
- These are non-executable and do not generate machine language instructions.
- **Macros** are basically a text substitution mechanism.

Syntax of Assembly Language Statements

Assembly language statements are entered one statement per line. Each statement follows the following format:

```
[label]    mnemonic    [operands]    [;comment]
```

The fields in the square brackets are optional.

A basic instruction has two parts:

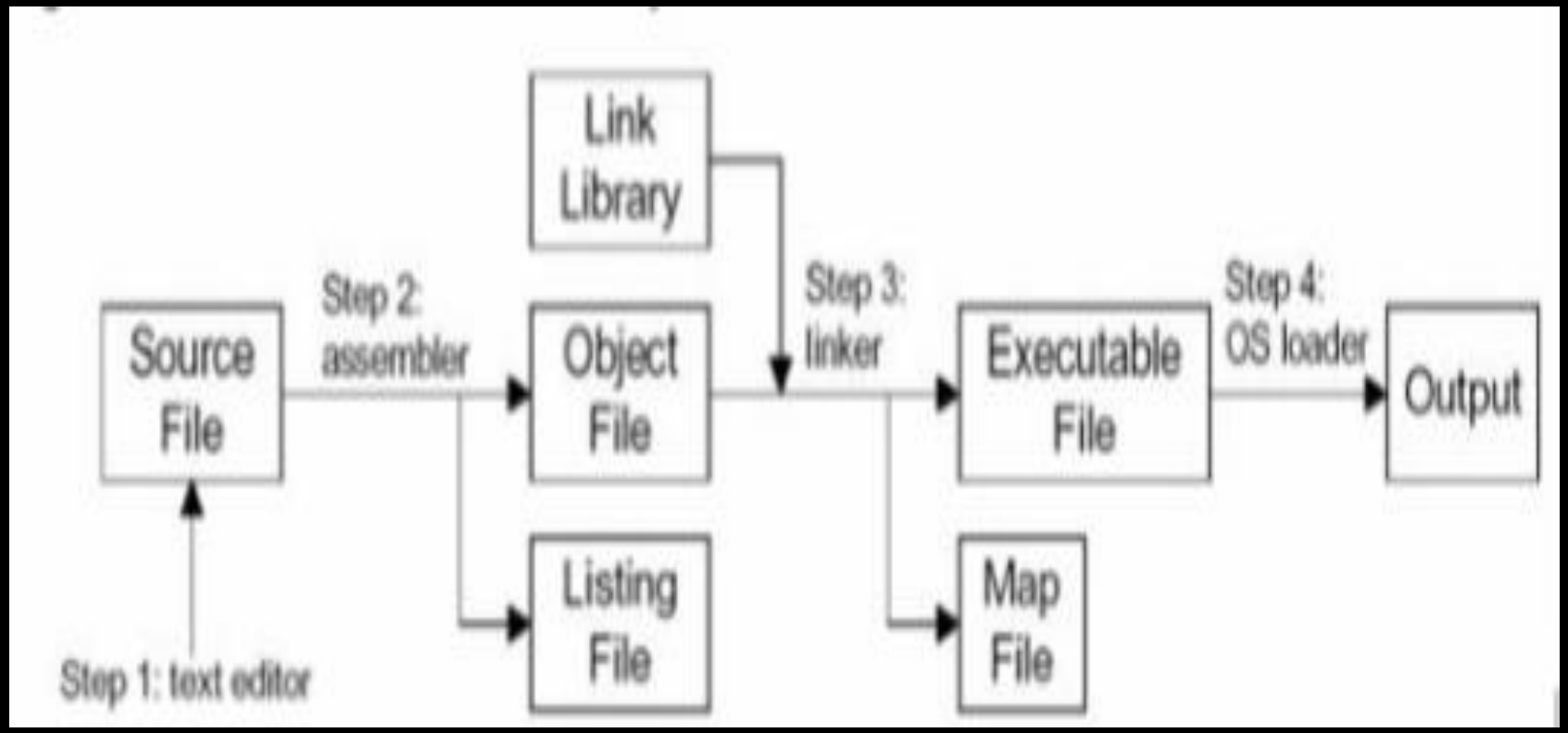
- The first one is the name of the instruction (or the mnemonic) which is to be executed, and the
- second are the operands or the parameters of the command.

- INC COUNT ; Increment the memory variable COUNT
- MOV TOTAL, 48 ; Transfer the value 48 in the
  ; memory variable TOTAL
- ADD AH, BH ; Add the content of the
  ; BH register into the AH register
- AND MASK1, 128 ; Perform AND operation on the
  ; variable MASK1 and 128
- ADD MARKS, 10 ; Add 10 to the variable MARKS
- MOV AL, 10 ; Transfer the value 10 to the AL register

ASSEMBLE, LINK, AND RUN A PROGRAM

- MASM & LINK are the assembler & linker programs.
  - Many editors or word processors can be used to create and/or edit the program, and produce an ASCII file.
  - The steps to create an executable Assembly language program are as follows:

Step	Input	Program	Output
1. Edit the program	keyboard	editor	myfile.asm
2. Assemble the program	myfile.asm	MASM or TASM	myfile.obj
3. Link the program	myfile.obj	LINK or TLINK	myfile.exe


Link Library
Source File
Step 2: assembler
Object File
Step 3: linker
Executable File
Step 4: OS loader
Output
Listing File
Map File
Step 1: text editor

- https://www.youtube.com/watch?v=lCjbwLeLNfs

```
.386                                        ; Use the 386 instruction set

.model flat, stdcall                        ; Memory model - flat for Windows programs

option casemap :none                        ; Labels are case sensitive
include \masm32\include\windows.inc         ; Include files are required and add
include \masm32\include\kernel32.inc        ; functionality to your program
include \masm32\include\masm32.inc          ;
includelib \masm32\lib\kernel32.lib         ;
includelib \masm32\lib\masm32.lib           ;

.data                                       ; Initialized data follows this directive
 msg1 db "Assembly Message: ", 0            ; Variables go here
 msg2 db "Your first Assembly program!", 0

.code                                       ; Starting point for your main code
start:                                      ; Code execution begins now
 invoke StdOut, addr msg1                   ; Calls the StdOut, passing addr of msg1
 invoke StdOut, addr msg2                   ; Calls the StdOut, passing addr of msg2
 invoke ExitProcess, 0                      ; Successful return code
end start                                   ; Code ends now
```

```
; #2 Save: this code as Test1.asm
; #3 Open: command prompt
; #4 To navigate to your file, type: cd PathTo\Test1.asm
; #5 To assemble your program, type: \masm32\bin\ml /c /Zd /coff Test1.asm
;
;
; OUTPUT
; Microsoft (R) Macro Assembler Version 6.14.8444
; Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

; Assembling: Test1.asm

; ***********
; ASCII build
; ***********
;
;
; #To link your program, type: \masm32\bin\Link /SUBSYSTEM:CONSOLE Test1.obj
```