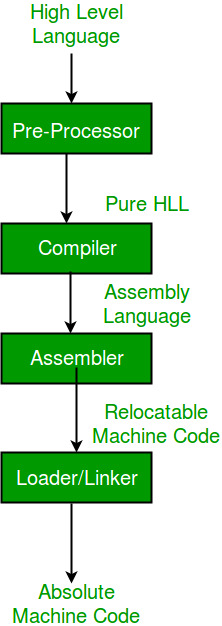
**Lab 01: Introduction to Assembly Language**

**OBJECTIVE**

To learn the basic commands, CPU registers and assembly language program structure.

**Introduction**

* Assemble language focuses on programming Intel microprocessors.
* Machine language is a Boolean language consisting of numbers that is specifically understood by a computer’s processor (the CPU).
* Of all languages, Assembly has the closest resemblance to the native machine language of the computer.
* For this reason, it is fast/speedy and it provides you direct access to the computer hardware.
* The only disadvantage is that assembly language programs are large and they would take too much time to write and maintain.
* An assembler is a program that converts source-code programs from assembly language into machine language.
* One Assembly language instruction corresponds to one machine language instruction.



**Theory**

**Central Processing Unit (CPU)**

The Central Processing Unit (CPU) is the physical device that executes instructions. The instructions that CPUs perform are generally very simple. Instructions may require the data they act on to be in special storage locations in the CPU itself called registers. The CPU can access data in registers much faster than data in memory. However, the number of registers in a CPU is limited, so the programmer must take care to keep only currently used data in registers.

**CPU Registers**

Registers are special work areas inside the CPU designed to be accessed at high speed. The registers are 16-bit long but you have the option of accessing the upper or lower halves of the four registers:

* Data Registers 16-bit: AX, BX, CX, and DX

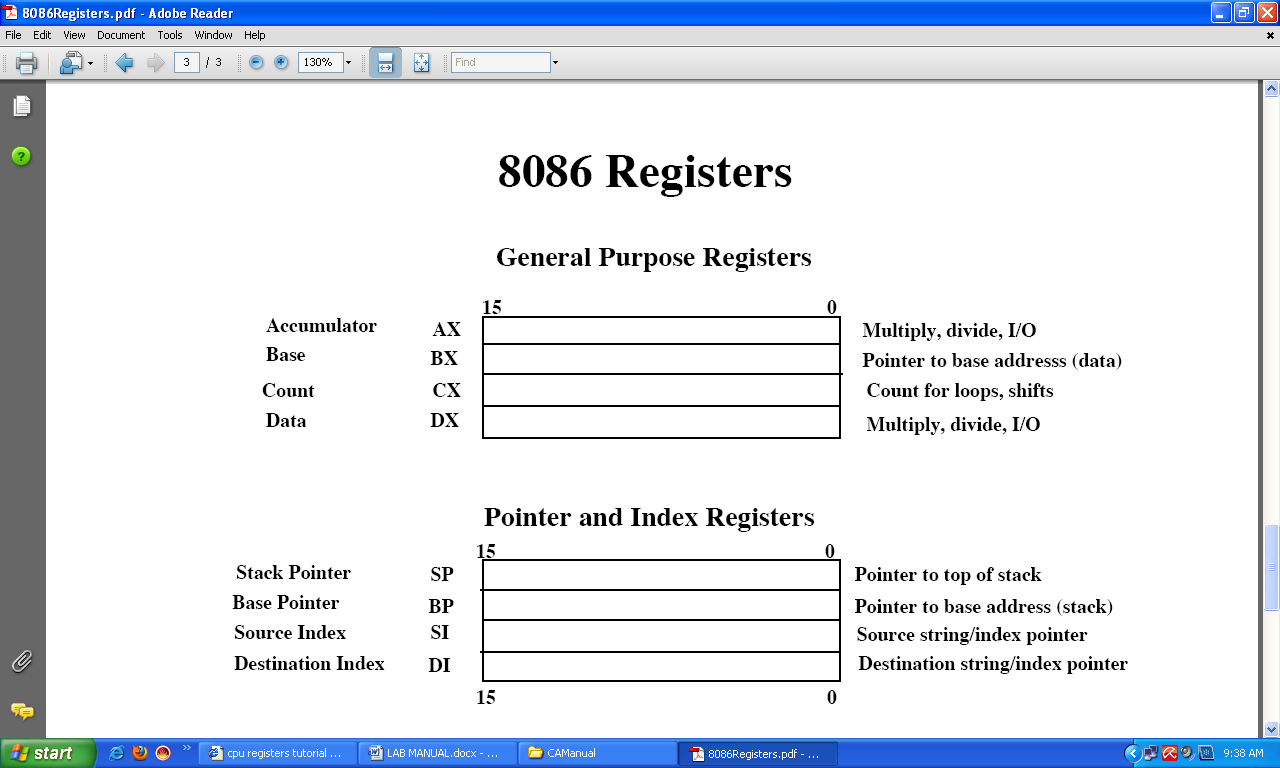
8-bit: AH, AL, BH, BL, CH, CL, DH, DL

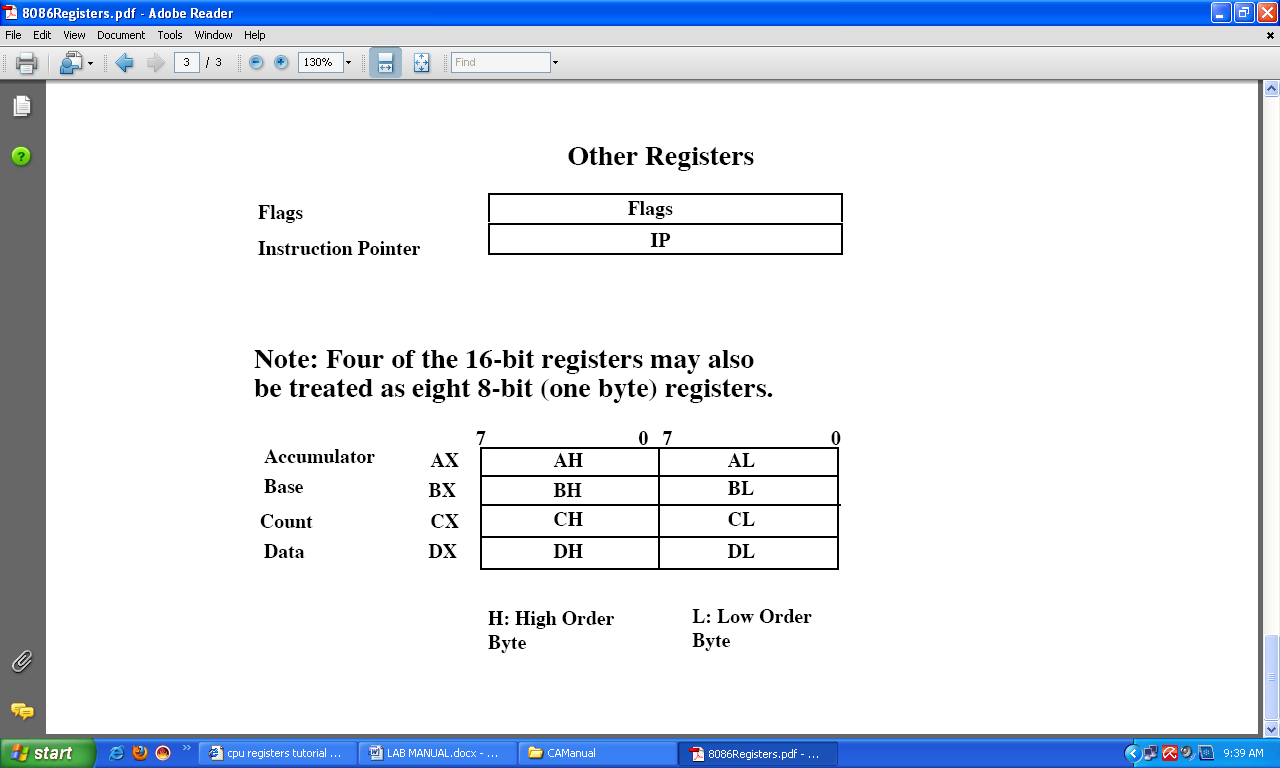
* Segment Registers CS, DS, SS, ES (Code, Data, Stack, Extra)
* Index Registers SI, DI, BP (Source, Destination, Base Pointer)
* Special Registers IP, SP (Instruction Pointer, Stack Pointer)
* Flag Registers Overflow, Direction, Interrupt, Trap, Sign, Zero, Auxiliary,

Carry, Parity

**Data Registers**

Four Registers named data registers or general purpose registers are used for arithmetic and data movement. Each register may be either addressed as 16-bit or 8-bit value. Bit positions are always numbered from right to left, starting with 0:





Each general purpose register has special attributes:

**AX** (Accumulator Register): AX is called the accumulator register because it is favored by the CPU for arithmetic operations.

**BX** (Base Register): BX register can also perform arithmetic and data movement, and it has special addressing abilities. It can hold a memory address that points to another variable.

**CX** (Counter Register): It acts as a counter for repeating or looping instructions. These instructions automatically repeat and decrement CX and quit when it is equal to 0.

**DX** (Data Register): DX is used for output purpose/display. It has a special role in multiply and divide operations. When multiplying for example DX holds the high 16-bits of the product.

**How to Start an Assembly Program**

Essential code instruction for all programs start with the following commands:

.model small

.stack 100h

.data

.code

MY FUNCTION

Mov ah,2 ; display function

Mov dl,al ; character to display

Int 21h ; interrupt

Mov ah,4ch ; exit to DOS

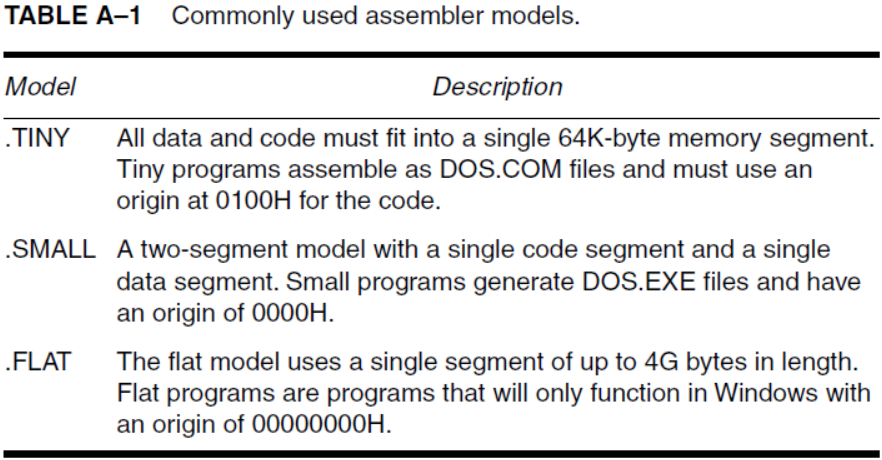
main proc

(your function is written inside this)

main endp

end main

**.model** determines the size of code and data a program can have e.g. **small**.

****

**.stack 100h** stores current address of ax, bx registers into memory block.

**.data** corresponds to data segment, and it initializes global variables or strings. It is used for all the variable definitions.

**.code** corresponds to code segment and all instructions come in this.

**main proc** is the ‘procedure statement’ and all code is written below this.

**main endp** is the ‘end procedure’ statement.

**end main** is the last line of code.

**NOTE:** Any function can be defined between **main proc** and **main endp**.

The most basic instruction in assembly is the ‘mov’ instruction. It moves data from one location to another. It takes two operands: **mov** destination, source

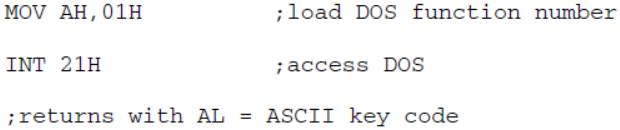
The data specified by **source is copied to destination**. Both operands may or may not be memory locations. Another important point to be kept in mind is that the source and destination operands must be same size.

* **IN/OUT Library**

**func # 1**  is for ‘input’ and **func # 2**  is for ‘output’.

Example: ah = func # (e.g. 1 or 2)

**INPUT EXAMPLE:** (function # 1 for single character input) [Read Operation]



{

e.g. mov ah,1

int 21h

**int** is for ‘interrupt’. ‘Int21h’ has the highest priority and it can invoke large no. of MS-DOS function calls.

**al** is used for single character input by the user (Default register byte).

**NOTE:** Before we call interrupt we assign ‘ah’ some value to read from.

**OUTPUT EXAMPLE:** (function # 2 for single character output/display)

mov ah,2

mov dl,'A'

int 21h

mov ah,2

mov dl,8h ;backspace

int 21h

mov ah,2

mov dl,'Z'

int 21h

e.g mov ah,2

{

mov dl, ‘A’

int 21h

**NOTE:** Default output byte is ‘dl’.

**;** is used to write comments to a line.

Example: mov ah,1; single character read in from ‘al’.

* **To Generate ENTER/New Line** following two commands are used:

0dh : (zero) carriage return (backspace).

🡨 Backspace

\_\_ New line

0ah : new row (line generation)

**4ch** returns control to DOS, is (used to exit). Example: mov ah,4ch.