# Microprocessor 8086
# Assembly Language Programming

Satish Chandra

## Assembly language programming

- Learning assembly language programming will help understanding the operations of the microprocessor
- To learn:
  - Need to know the functions of various registers
  - Need to know how external memory is organized and how it is addressed to obtain instructions and data (different addressing modes)
  - Need to know what operations (or the ***instruction set***) are supported by the CPU. For example, powerful CPUs support floating-point operations but simple CPUs only support integer operations

2

## How to learn programming

- *C* –Concept
- *L* – Logic thinking
- *P* – Practice
- Concept – we must learn the basic *syntax*, such as how a program *statement* is written
- Logic thinking – programming is problem solving so we must think logically in order to derive a solution
- Practice – write programs

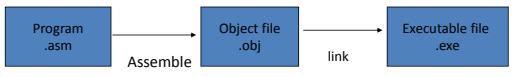
3

## Assembly Program

- The native language is machine language (using 0,1 to represent the operation)
- A single machine instruction can take up one or more bytes of code
- Assembly language is used to write the program using alphanumeric symbols (or mnemonic), eg ADD, MOV, PUSH etc.
- The program will then be assembled (similar to compiled) and linked into an executable program.
- The executable program could be .com, .exe, or .bin files

4

## Flow of program development




5

## Example

- Machine code for MOV AL, 00H
- B4 00 (2 bytes)
- After assembled, the value B400 will be stored in the memory
- When the program is executed, then the value B400 is read from memory, decoded and carry out the task

6

## Assembly Program

- Each instruction is represented by one assembly language statement
- The statement must specify which operation (opcode) is to be performed and the operands
- Eg ADD AX, BX
- ADD is the operation
- AX is called the destination operand
- BX is called the source operand
- The result is AX = AX + BX
- When writing assembly language program, you need to think in the instruction level

7

## Example

- In FORTRAN, you can do A = (B+C)*100
- In assembly language, only one instruction per statement

```
A = B        ; only one instruction - MOVE
A = A+C      ; only one instruction - ADD
A = A*100    ; only one instruction - Multiply
```

8

## Format of Assembly language

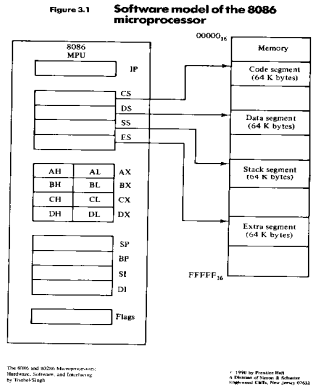
- General format for an assembly language statement
- Label Instruction Comment
- Start: MOV AX, BX ; copy BX into AX

Start is a user defined name and you only put in a ***label*** in your statement when necessary!!!!

The symbol **:** is used to indicate that it is a label

9

## 8086 Software Model




10

## Software model

- In 8086, memory is divided into segments
- Only 4 64K-byte segments are active and these are: *code, stack, data, and extra*
- *When you write your assembly language program for an 8086, theoretically you should define the different segments!!!*
- To access the active segments, it is via the segment register: CS (code), SS (stack), DS (data), ES (extra)
- So when writing assembly language program, you must make use of the proper segment register or index register when you want to access the memory

11

## Registers

- In assembly programming, you cannot operate on two memory locations in the same instruction
- So you usually need to store (move) value of one location into a register and then perform your operation
- After the operation, you then put the result back to the memory location
- Therefore, one form of operation that you will use very frequent is the store (move) operation!!!
- And using registers!!!!!

12

## Example

- In FORTRAN, A = B+C ; A, B, C are variables
- In assembly language A,B, C representing memory locations so you cannot do A = B+C
  - MOV AL, B ; move value of B into AL register
  - ADD, AL, C ; do the add AL = AL +C
  - MOV A, AL ; put the result to A

13

## Data registers

- AX, BX, CX,and DX – these are the general purpose registers but each of the registers also has special function. Example
  - AX is called the accumulator – to store result in arithmetic operations
- Registers are 16-bit but can be used as 2 8-bit storage
- Each of the 4 data registers can be used as the source or destination of an operand during an arithmetic, logic, shift, or rotate operation.

14

## Data register

- In based addressing mode,
- Base register BX is used as a pointer to an operand in the current data segment.
- Code register, CX is used as a counter in some instructions.
- Data register, DX is used in all multiplication and division.

15

## Pointer and index registers

- Stack – is used as a temporary storage
- Data can be stored by the PUSH instruction and extracted by the POP instruction
- Stack is accessed via the SP (Stack Pointer) and BP (Base Pointer)
- The BP contains an offset address in the current stack segment.

16

## Pointer and Index Register

- Source index register (SI) and Destination index register (DI) are used to hold offset addresses for use in indexed addressing of operands in memory
- When indexed type of addressing is used, then SI refers to the current data segment and DI refers to the current extra segment
- The index registers can also be used as source or destination registers in arithmetic and logical operations. But must be used in 16-bit mode

17

END