Session 3

Organization of the IBM Personal Computers

OBJECTIVES:

- Students will have a brief know about the Intel 8086 family of micro-processors.
- They will acquire knowledge about the organization of 8086 micro-processors and the PC.

Intel 8086 Family

- 8086 and 8088 Microprocessor

  8086 is the first 16-bit microprocessor. It has 16-bit data bus where 8088 has 8-bit data bus. 8086 has faster clock rate than 8088.

- 80286 Microprocessor

  It is 16-bit microprocessor. It is faster than 8086. It can operate in real address mode and protected virtual address mode. In protected mode, it can address 16MB of physical memory. It can treat external storage as physical memory.

- 80386 Microprocessor

  80386 is the first 32-bit microprocessor. 8086 has 32-bit data bus. It has high clock rate. It can execute instructions in fewer clock cycles. It can operate in real and protected mode. In protected mode it can address 4GB of physical memory and 64TB of virtual memory.

- 80486 Microprocessor

  80486 is 32-bit microprocessor. It is the fastest and more powerful processor of the family. It has numeric processor, cache memory and more advanced design. It is three times faster than 80386 running at same clock speed.

Organization of the 8086/8088 Microprocessors

- Registers
  Information inside the microprocessor is stored in registers. There are fourteen 16-bit registers in 8086.

    - Data Registers
      Data registers hold data for an operation. 8086 has four general data registers:

1. Accumulator Register (AX): Used in arithmetic, logic and data transfer instructions. In multiplication and division, one of the numbers involved must be in AX or AL. Input and output operations also required AX and AL.
2. Base Register (BX): Serves as an address register
3. Count Register (CX): CX serves as a loop counter. CL is used as a count in instructions that shift and rotate bits.
4. Data Register (DX): Used in multiplication and division. Also used in I/O operation.

- Address Registers

  Address registers hold the address of an instruction or data. Address registers divided in three types:

  1. Segment Registers

     Segment registers have a very special purpose, pointing at accessible blocks of memory. Segment registers work together with general purpose register to access any memory value.

     - Code Segment (CS): Points at the segment containing the current program.
     - Data Segment (DS): Points at segment where variables are defined. By default, BX, SI and DI registers work with DS segment register.
     - Stack Segment (SS): Points at the segment containing the stack. BP and SP work with SS segment register.
     - Extra Segment (ES): It is up to a coder to define its usage.

  2. Pointer Registers
     - Stack Pointer (SP): Used with SS for accessing the stack segment.
     - Base Pointer (BP): Used primarily to access data on the stack
     - Instruction Pointer (IP): It is used to access instructions. CS contains the segment number of the next instruction and IP contains the offset. IP is updated each time an instruction is executed. An instruction cannot contain IP as an operand.

  3. Index Registers

     - Source Index (SI): Point to memory locations in the data segment addressed by DS. Incrementation of SI give easy access to consecutive memory locations

- Destination Index (DI): Same function as SI. String operations use DI to access memory locations addressed by ES.

- Status Register

  Status register keeps the current status of the processor. In 8086 status register is called FLAGS register. There are two types of Flags:

  1. Status Flags: Reflect the result of an instruction executed by the processor. For example, if an arithmetic operation produces a 0 value as a result then the zero flag (ZF) is set to 1.
  2. Control Flags: Enable or disable certain operations of the processor. For example, if interrupt flag (IF) is set to 0, inputs from the keyboard are ignored by the processor

Memory Segment

8086 is a 16-bit processor using 20-bit address. Addresses are too big to fit in a 16-bit register or memory word. 8086 gets around this problem by partitioning its memory into segments.

- Segment Number

Memory segment is a block of 2^{16} = 64K consecutive memory bytes identified by a segment number starting with 0. Segment number is 16-bit so the highest segment number is FFFFh.

- Offset

Within a segment a memory location is specified by giving an offset. This is the number of bytes from the beginning of segment. With 64-bit segment the offset can be given as a 16-bit number. The first byte in a segment is offset 0 and the last offset in a segment is FFFFh.

Physical Address

A memory location may be specified by providing a segment number and an offset written in the form segment: offset. This form is known as logical address.

A4FB:4872h means offset 4872h within segment A4FBh.

For obtaining 20-bit physical address, 8086 first shifts the segment address four bits to the left and then adds the offset. The physical address for A4FB:4872 is A9822h.

Organization of the PC

- Operating System
- Memory Organization
- I/O Port Addresses
- Start-up Operation

Sample Math

A memory location has physical address 80FD2h. In what segment does it have offset BFD2h.

Answer:

Physical address = Segment* 10h + Offset

Segment * 10h = Physical address – offset

Segment * 10h = 80FD2h – BFD2h

Segment * 10h = 75000h

Segment = 7500h

So, the segment address is 7500h.

Session 4

Introduction to IBM PC Assembly Language

OBJECTIVES:

- Students will come to know the syntax of Assembly language.
- They will acquire knowledge on program data, variables and named constants.
- They will get to know about a few basic instructions of assembly language.
- Students will be able to Translate high-level language to assembly language.
- They will come to know about program structure of assembly language.
- Students will learn the input and output instructions.

Assembly Language Syntax

Statements are of two types: Instruction and Assembler Directive.

Statements can be written in the form: Name Operation_Field Operand(s)_Field Comment

Example: Instruction - START: MOV CX,5; initialize counter

Assembler directive- MAIN PROC

Name Field

Used for instruction labels, procedure names and variable names. Assembler translates name into memory address. Names can be 1 to 31 characters long and may consist of letters, digits or special characters. If period is used, it must be first character. Embedded blanks are not allowed. May not begin with a digit. Not case sensitive.

Operation Field

Contains symbolic (Operation code). Assembler translates op code translated into machine language op code. Examples: ADD, MOV, SUB.
In an assembler directive, the operation field represents pseudo-op code. Pseudo-op code is not translated into machine code; it only tells assembler to do something. Example: PROC pseudo-op is used to create a procedure.

Operand Field

Specifies the data that are to be acted on by the operation. An instruction may have zero, one or more operands. In two-operand instruction, first operand is destination, second operand

is source. For an assembler directive, operand field represents more information about the directive. Examples: NOP, INC AX, ADD WORD1, 2.

Comment Field

Say something about what the statement does. Marked by semicolon in the beginning. Assembler ignores anything after semicolon. It is optional but a good practice.

Program Data

Processor operates only on binary data. Data can be given as a number or a character.

- Numbers
    - Binary
    - Decimal
    - Hexadecimal
- Characters

Named Constants

- Use symbolic name for a constant quantity
- Syntax:
    - name EQU constant

Variables

- Each variable has a data type and is assigned a memory address by the program.
- Possible Values:
    - 8-bit Number Range: Signed (-128 to 127), Unsigned (0 to 255)
    - 16-bit Number Range: Signed (-32,678 to 32767), Unsigned (0 to 65,535)
    - ? - To leave variable uninitialized
- Mainly three types
    - Byte Variables
    - Word Variables
    - Arrays

Data Defining Pseudo-Ops

Pseudo - ops	Description	Bytes	Examples
DB	Define Byte	1	var1 DB 'A' array1 DB 10, 20,30,40
DW	Define Word	2	var2 DW 'AB' array2 DW 1000, 2000
DD	Define Double Word	4	Var3 DD -214743648
DQ	Define Quad Word	8	Var DQ ?
DT	Define Ten Bytes	10	Var DT ?

A Few Basic Instructions

- MOV

  Transfer data between registers, between register and a memory location or move a number directly to a register or a memory location.
  Syntax: MOV destination, source
  Example: MOV AX, WORD1

- XCHG

  Exchange the contents of two registers or register and a memory location.
  Syntax: XCHG destination, source
  Example: XCHG AH, BL

- ADD

  To add contents of two registers, a register and a memory location, a number to a register or a number to a memory location.
  Syntax: ADD destination, source
  Example: ADD WORD1, AX

- SUB

  To subtract the contents of two registers, a register and a memory location, a number from a register or a number from a memory location.
  Syntax: SUB destination, source
  Example: SUB AX, DX

- INC

  To increment one to the contents of a register or memory location.
  Syntax: INC destination
  Example: INC WORD1

- DEC

  To decrement one from the contents of a register or memory location.
  Syntax: DEC destination
  Example: DEC BYTE1

- NEG

  To negate the contents of destination.
  Syntax: NEG destination
  Example: NEG BX

Translation of High-level Language to Assembly Language

- Consider instructions: MOV, ADD, SUB, INC, DEC, NEG
- A and B are two-word variables
- Translate statements into assembly language

Statement	**Translation**
B = A	MOV AX, A MOV B, AX

A = 5 – A	MOV AX, 5
	SUB AX, A
	MOV A, AX
	OR
	NEG A
	ADD A, 5

A = B – 2 x A	MOV AX, B
	SUB AX, A
	SUB AX, A
	MOV A, AX

Program Structure

- Machine programs consists of
    - Code: Contains a program's instructions.
        - Syntax: .CODE name
    - Stack: A block of memory to store stack
        - Syntax: .STACK size
    - Data: Contains all variable definitions
        - Syntax: .DATA

Memory Models

- Determines the size of data and code a program can have.
- Syntax:
    - .MODEL memory_model

Model	Description
SMALL	Code in one segment, data in one segment
MEDIUM	Code in more than one segment, data in one segment
COMPACT	Code in one segment, data in more than one segment
LARGE	Both code and data in more than one segments. No array larger than 64KB
HUGE	Both code and data in more than one segments. Array may be larger than 64KB

The Format of a Code

```
.MODEL SMALL
.STACK 100h
.DATA
;data definition go here
.CODE
MAIN PROC
;instructions go here
MAIN ENDP
;other procedures go here
END MAIN
```

Input and Output Instructions

Function Number	Routine	Function	Code
1	Single Key Input	Input: AH=1 Output: AL=ASCII Code if character is pressed AL=0 if non-character key is pressed	MOV AH,1; input key function INT 21H; ASCII code in AL

2	Single Character Output	Input: AH=2
		DL=ASCII code of the display character or control character
		Output: AL=ASCII code of the display character or control character
			MOV AH,2; display character function
			MOV DL,'?'; character is?
			INT 21H; display character

9	Character String Output	Input: DX=offset address of string. The string must end with a '$' character	MSG DB 'HELLO$'

LEA Instruction

It means Load Effective Address. To get the offset address of the character string in DX we use LEA instruction. Destination is a general register and source is a memory location

- Syntax
    - LEA destination, source
- Example
    - LEA DX, MSG

Sample Question:

Write an assembly code to input an uppercase letter and output the letter in lowercase form.

Sample Input:

```
Enter an uppercase letter: A
```

Sample Output:

```
The lowercase letter is: a
```