Experiment Number : (2)

The Inside of 8086 Microprocessor & Move Instruction
Part (1)

Object:

To study the 8086 microprocessor and to learn how to use their registers.

Theory:

1. 8086 Assembly Language

Assembly language is a low level programming language. You need to get some knowledge about computer structure in order to understand anything. The simple computer model is:

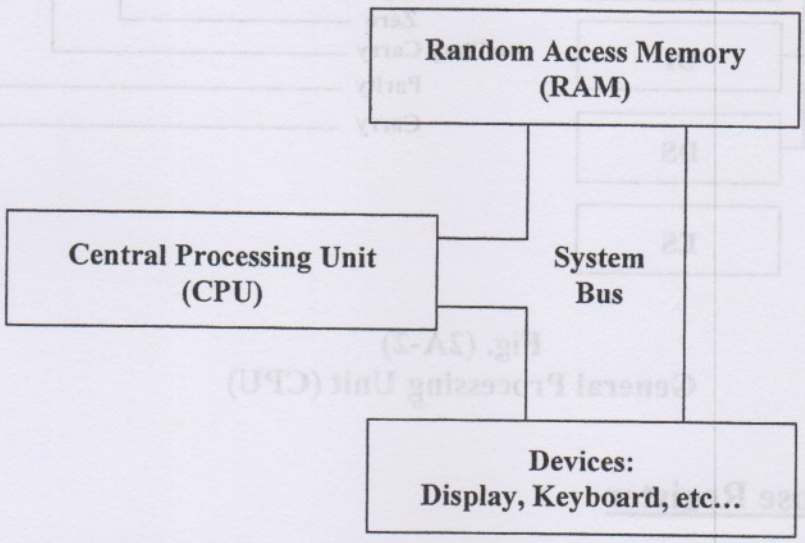

Fig. (2A-1)
Simple Computer Model

The system bus connects the various components of a computer.

The CPU is the heart of the computer, most of computation occur inside the CPU (Central Process Unit).

RAM is a place to where the programs are loaded in order to be executed.

2. Inside The CPU:

Fig. (2-2) shows the general processing unit (or CPU).

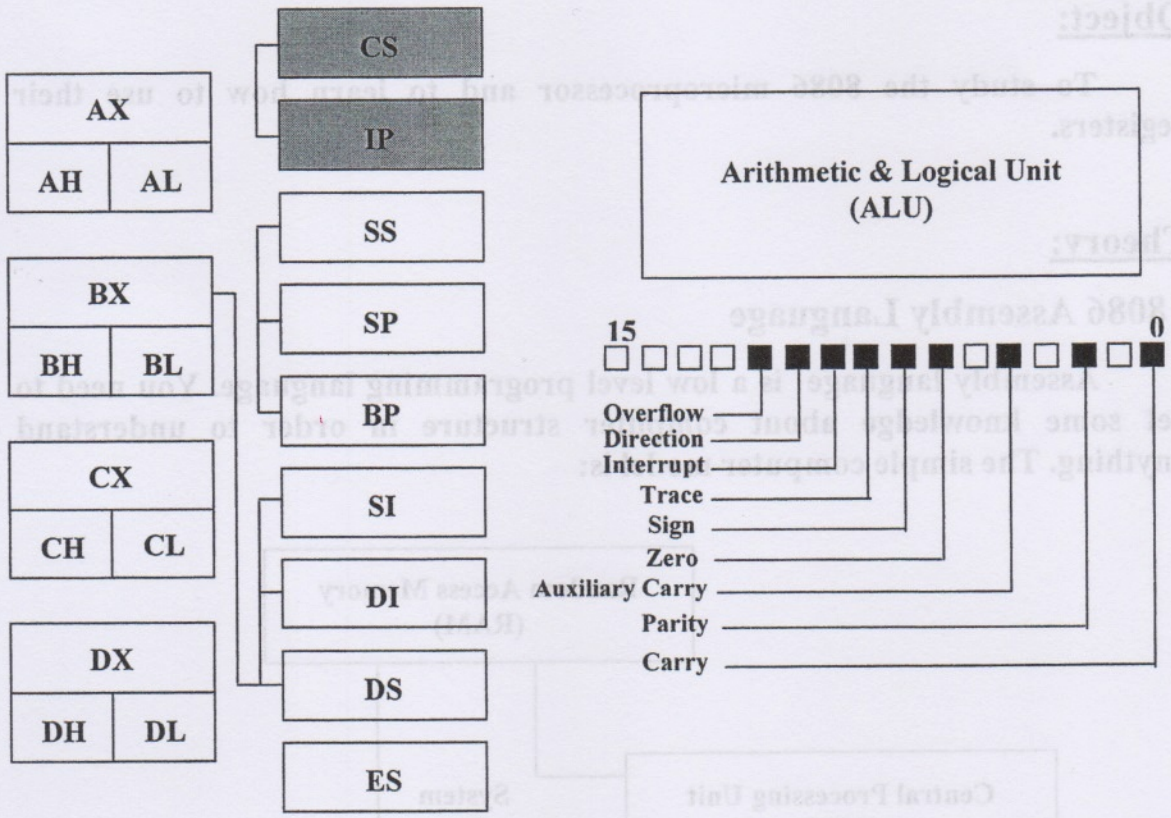

Fig. (2A-2)
General Processing Unit (CPU)

General Purpose Register

8086 CPU has 8 general-purpose registers, each register has its own name:

AX: The accumulator register (divided into AH / AL).
BX: The base address register (divided into BH / BL).
CX: The count register (divided into CH / CL).
DX: The data register (divided into DH / DL).
SI: Source index register.
DI: Destination index register.
BP: Base pointer register.
SP: Stack pointer register.

Despite the name of a register, it's the programmer who determines the usage for each general purpose register. The main purpose of a register is to keep a number (variable). The size of the above register is a 16 bit, its something like: "0011000001110010b" (in binary form), or "12345" in decimal (human) form.

Four general purpose registers (AX, BX, CX, DX) are made of two separate 8 bit registers, for example if AX = 0011000001110010b, then AH = 00110000b and AL = 01110010b. Therefore, when you modify any of the 8 bit registers 16-bit register is also updated, and vice-versa. The same is for other 3 registers, "H" is for high and "L" is for low part.

Because registers are located inside the CPU, they are much faster than memory. Accessing a memory location requires the use of a system bus, so it takes much longer. Accessing data in a register usually takes no time. Therefore you should try to keep variables in the registers. Register sets are very small and most registers have a special purpose which limit their use as a variables, but they are still an excellent place to store temporary data of calculations.

Segment Registers

CS: Code Segment points at the segment containing the current program.
DS: Data Segment generally points at segment where variables are defined.
ES: Extra segment register, it's up to a coder to define its usage.
SS: Stack segment points at the segment containing the stack.

Although it is possible to store any data in the segment registers, this is never a good idea. The segment registers have very special purpose-pointing at accessible blocks of memory.

Segment registers work together with general purpose register to access any memory value.

Special Purpose Register

IP: The instruction pointer.
Flag Register: Determine the current state of the processor.
IP register always work together with CS segment register and it's points to currently executing instruction.
Flag Register is modified automatically by CPU after mathematical operations, this allows to determine the type of the results, and to determine conditions to transfer control to other part of the program.
Generally you cannot access these registers directly.

Therefore we can see all the above registers is 16 bits registers and we can classified the buss through the size as:

1. **Data Bus: This bus is to transfer the data from and to processor and have 16 bits size.**
2. **Address Bus: This bus is used to access the memory and have 20 bits size.**
3. **Control Bus: This bus is used to control of all operations of the microprocessor.**

Procedure:

1. **Write down the following program in the 8086 trainer:**

```
MOV AL, 44H
MOV AH, AL
MOV DX, 0100H
MOV BL, BH
MOV CH, 47H
MOV CL, 'A'
MOV DL, 67H
MOV DH, DL
MOV DS, AX
MOV SS,BX
MOV ES, CX
INT A5
```

2. **Execute the program and write down all the output registers.**

Home Work

1. **What are the results of the following program:**

```
MOV  AL, 0AH
MOV  BL, O6H
ADD  AL, BL
MOV  CL, AL
RET
```

2. **What are the results of the following program:**

```
MOV  AL, 44H
MOV  AH, AL
MOV  DX, 256
MOV  BL, BH
MOV  CH, 01000111B
MOV  CL, 'A'
MOV  DL, 174 O
MOV  DH, DL
MOV  DS, AX
MOV  SS,BX
MOV  ES, CX
INT A5
```

3. **What are the data representation types used in the program above and what their letter indication.**
4. **Why is the better to use the registers to transform the data from and to the CPU over use the memory?**

Experiment Number : (2)

The Inside of 8086 Microprocessor & Move Instruction Part (2)

Object:

To study the 8086 microprocessor and to learn how to use their registers.

Theory:

Memory Access

To access memory we can use these four register: (BX, SI, DI, BP). Combining these registers inside [ ] symbols, we can get different memory locations.

Let we want to store the data in AX in the memory location (11000H) we must set the data segment and the offset of the above location as:

Offset = 1000H

$$\text{Segment} = DS = \frac{11000 - 1000}{10} = 1000H$$

Always the offset is put either in the register SI or DI or BX only, you know the DS cannot access directly therefore we must put the value of the DS at first in the one of the other three general purpose registers (I.E. AX, CX, DX) and then transfer it to the DS register, let the data in the AX registers is (FFAAH) the AH=FFH and the AL=AAH, after we execute the program we see the AL is in the memory location (11000H) and AH in (11001H)

See Fig. (2B-1)

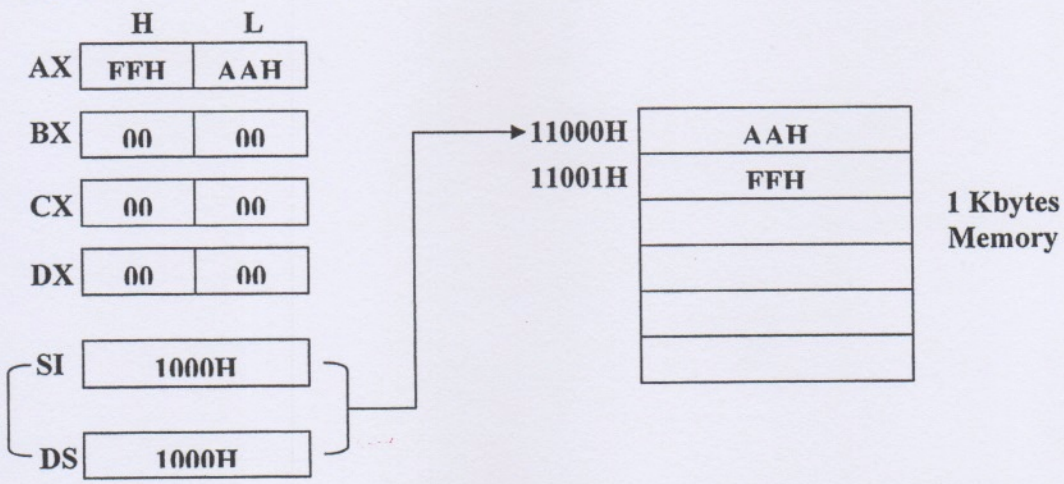

Fig. (2B-1)
Memory Access

Example if we would like to access memory at the physical address 12345h (hexadecimal), we should:
12345h the first four number from right to left side is an offset and put it into SI register, therefore SI = 2345h and the data segment register DS must be 1000h because $DS = \frac{(\text{phiscal address} - \text{offset})\text{h}}{10} = \frac{(PA - SI)\text{h}}{10}$

$DS = \frac{(12345 - 2345)\text{h}}{10} = 1000\,\text{h}$. This is good, since this way we can access much more memory than with a single register that is limited to 16 bit values.
CPU makes a calculation of physical address by multiplying the segment register by 10h and adding general purpose register (SI) to it.
PA = (DS*10h) + SI, therefore PA = (1000h*10h)+2345h = 12345h.

The address formed with 2 registers is called an effective address. By default BX, SI, and DI registers work with DS segment register; BP, and SP work with SS segment register.
Other general purpose registers cannot form an effective address. Also, although BX can form an effective address, BH, and BL cannot!

Move Instruction

Copies the second operand (Source) to the first operand (destination), the source operand can be immediate vale, general purpose register or memory location.
The destination register can be a general purpose register or memory location. Both operand must be the same size, which can be a byte or a word.

- These types of operand are supported:

MOV REG, MEMORY
MOV MEMORY, REG
MOV REG, REG
MOV MEMORY, IMMEDIATE
MOV REG, IMMEDIATE
REG: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL, DI, SI, BP, SP.
Immediate: 5, -24, 3FH, 1000101B, etc...

- For segment register only these type of MOV are supported:

MOV SREG, MEMORY
MOV MEMORY, SREG
MOV REG, SREG
MOV SREG, REG
SREG: DS, ES, SS, and only the second operand: CS.

REG: AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL, DI, SI, BP, SP.
MEMORY: [BX], [SI]+BX, [DI]+BX, etc...

Note: The move instruction cannot be used to set the value of the CS and IP.

Procedure:

1. **Write down the following program:**

```
MOV  AX, 0B800H
MOV  DS, AX
MOV  CL, 'A'
MOV  CH, 01011111B
MOV  BX, 15EH
MOV  [BX], CX
INT A5
```

2. **Calculate the offset (BX), segment (DS), and the physical address of the above program.**
3. **Execute program and write down the results of the output register and the memory.**

Home Work:

1. **Write a program to store (10101010B) to the memory location 47511H.**
2. **What are the results of the following program:**

```
MOV  AX, 3000H
MOV  DS, AX
MOV  AX, 4000H
MOV  SS, AX
MOV  BX, 0600H
MOV  BP, 0700H
MOV  DX, A6E2H
MOV  SI, 0040H
MOV  DI, 00E0H
MOV  [BX]+SI, AH
MOV  [BX]+DI, DH
MOV  [BP]+DI, DL
MOV  [BX]-20H, DH
MOV  [DI]+4H, DL
MOV  [BP]+100H, AL
INT A5
```

3. **Give the physical address of all access memory of the above program.**
4. **You have DS=4000H, AH=7FH, AL=44H, SI=1000, what are the physical address of the instruction**

```
MOV  [SI]+10H,AX
```

And how the data store in the memory?

Experiment Number : (3)

Defined Data

Object:

To learn how to define data and use this data as single data or a matrix (one or two dimensions).

Theory:

There are two types of the variables, there are:

1. **Numeric variable: this is divided into two parts:**
   - **A. Constant.**
   - **B. Variable.**
2. **string variables.**

The structure of define variable in Assembly language is:

```
.DATA
Var1    EQU 05H                  To define constant.
Var2    (DB or DW) 05H or 0005H  To define numeric variable.
Var3    DB 05, 06, 07, ----      To define numeric array of the variables.
Var4    DW 0005, 000A, 0ABC, --- To define numeric array of the word variables.
Var5    DB 'A'                   To define the string.
Var6    DB 'Ahmed'               To define array of the string.
```

These definitions are put before the starting code, (I.E. before .CODE).

@data and Offset Instructions:

To set the data segment as a default value we must use a (@ data) instruction such as example below:

```
.DATA
DT1 DB 10H
.CODE
MOV AX,@DATA
MOV DS,AX
```

Also to set the offset as a default value (choose by CPU) we must use the offset instruction, such as:

MOV SI,OFFSET DT1

DUP Instruction:

This instruction is used to put the numeric array of variable to one values.

Example;

VAR1 DB 10 DUP(0)

This is means the 10 values of the array (VAR1) is equal (0).

VAR2 DB 5 DUP(1,2)

This is means the 5 values of the array (VAR2) is equal 1 & 2 respectively.

How To Defined Variable In 8086 Trainer:-

Example:

Write 8086 program to copy the value of the DT1 to the DT2 if the value of the DT1 = 10H.

Solution:

The program of this example is:

```
.DATA
DT1 DB 10H
DT2 DB 0
.CODE
MOV AX,@DATA
MOV DS,AX
MOV SI,OFFSET DT1
MOV DI,OFFSET DT2
MOV AL,[SI]
MOV [DI],AL
RET
```

1. **In 8086 trainer the (.data, @data, offset, dup) instruction doesn't works.**
2. **The section of memory to store data is beginning at physical address 10000H.**
3. **First we must access the memory (DS=1000, Offset=0000)H and put the value of DT1 in this location.**

4. Write above program such as:

```
MOV AX,1000
MOV DS,AX
MOV SI,0000        DT1 Location (Source)
MOV DI,0050        DT2 Location (Destination)
MOV AL,[SI]
MOV [DI],AL
INT A5
```

5. Execute the program and access the memory again to find the result of the DT2 variable in the location (1000:0050).

Example:

Write 8086 program to find the area of the square which the length is stored in the location DT1 store the results in the location DT2 and DT1= 2, 4, 8.

```
.DATA
DT1 DB 2,4,8
DT2 DB 3 DUP(0)
.CODE
MOV AX,@DATA
MOV DS,AX
MOV SI,OFFSET DT1
MOV DI,OFFSET DT2
MOV CX,0003
MOV BX,0
L1: MOV AX,0
MOV AL,[SI+BX]
MUL AL
MOV [DI+BX],AL
INC BX
LOOP L1
RET
```

In 8086 trainer we must access the memory at the location 1000:0000 and put the values of dt1 such as:

```
1000:0000 2
1000:0001 4
1000:0002 8
```

And write above program such as:

```
MOV AX,1000
MOV DS,AX
MOV SI,0000
MOV DI,0050
MOV CX,0003
MOV BX,0
L1: MOV AL,[SI+BX]
MUL AL
MOV [DI+BX],AL
INC BX
LOOP L1
INT A5
```

Procedure:

1. **Fill 10 bytes of the memory (DT1) by the number (1,2,3,…,10).**
2. **Write 8086 program to add above 10 bytes by 20H and store the results into DT2 location.**
3. **Execute the program and write down the results.**
4. **Modify the above program to store the results in the same location DT1.**

Home Work:

1. **Write 8086 program to find the area of the rectangle which the length and width is stored in the location DT1 respectively, store the results into location DT2.**
   DT1=4, 2, 8, 3, 5, 7, 1, 10
2. **Write the 8086 program to calculate the expression (X=A*10) where A is store in the location DT1 and DT1= 1,2,3,…,10. Store the results into DT2.**
3. **Write 8086 program to store DT1 in the Stack where DT1=10,20,30,40,50 D**
4. **Write 8086 program to divided above location (in step 3) by 2 and store to the location DT2.**